

# VBA

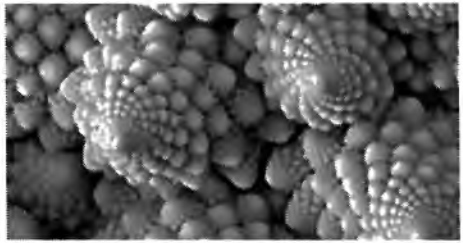
В.Г. Кузьменко

## Эффективное использование

- Запись и выполнение макросов в приложениях Microsoft Office
- Описание языка программирования Visual Basic for Applications
- Использование встроенных и написание собственных процедур и функций
- Введение в объектно-ориентированное программирование
- Основы программирования Windows-интерфейса пользователя



**БИНОМ**



**VBA**

Эффективное  
использование



**В. Г. Кузьменко**

# **VBA**



Москва  
Издательство **БИНОМ**  
2012

УДК 004.432  
ББК 32.973.26-018.1  
К89

**Кузьменко В. Г.**

VBA. — М.: ООО «Бином-Пресс», 2012 г. — 624 с.: ил.

В книге содержится краткий курс по языку программирования Visual Basic for Applications (VBA), используемого в качестве языка программирования в приложениях Microsoft Office (2000/2/3 и 2007). Книга предназначена для тех, кто в своей профессиональной деятельности часто использует приложения Microsoft Office и кому необходимо повысить эффективность работы с этими приложениями посредством автоматизации многих повторяющихся операций.

Материала книги достаточно для изучения основ языка Visual Basic и создания как простых макросов, предназначенных для автоматизации рутинной повторяющейся работы с документами, электронными таблицами, диаграммами, презентациями и т.д., так и для разработки довольно сложных приложений обработки данных с использованием диалоговых окон, обеспечивающих пользователей самыми современными интерфейсными средствами для работы с Windows-приложениями.

ISBN 978-5-9518-0444-0

© Кузьменко В. Г.  
© Издательство Бином

Научно-техническое издание

**Кузьменко В. Г.**

**VBA**

Оформление обложки *И. Ю. Буровой*

Подписано в печать 12.12.2011. Формат 70×100/16. Усл. печ. л. 50,7.

Гарнитура «Школьная». Бумага газетная. Печать офсетная.

Тираж 1000 экз. Заказ № 1012

Издательство «Бином-Пресс», 2012 г.

141077, Королев, Московская обл., ул. 50 лет ВЛКСМ, 4-Г

# Содержание

<b>Введение . . . . .</b>	<b>13</b>
<b>Глава 1. Введение в макросы . . . . .</b>	<b>15</b>
Из истории VBA . . . . .	16
Для чего нужен язык программирования VBA . . . . .	19
Создание макросов . . . . .	20
Создание макроса в Word . . . . .	21
Запись новых макросов . . . . .	23
Запись макроса в Word . . . . .	26
Запись макроса в Excel . . . . .	32
Запись действий . . . . .	34
Код макроса . . . . .	35
Выполнение макросов . . . . .	37
Сохранение документа с записанным макросом . . . . .	38
<b>Глава 2. Редактирование макросов . . . . .</b>	<b>41</b>
Модули . . . . .	41
Модули в Word . . . . .	41
Редактор Visual Basic . . . . .	42
Панели инструментов Редактора VB . . . . .	54
Редактирование макросов . . . . .	58
Составные части записанного макроса . . . . .	62
Написание новых макросов и процедур . . . . .	71
<b>Глава 3. Типы данных, переменные, константы и выражения . . . . .</b>	<b>83</b>
Экспоненциальное представление . . . . .	85
Тип Date . . . . .	85
Числа . . . . .	87
Текстовые строки . . . . .	89
Логические значения . . . . .	89
Тип данных Variant . . . . .	90
Переменные . . . . .	90
Что такое переменная? . . . . .	90
Выбор имен для переменных . . . . .	91
Создание переменных . . . . .	93
Задание типа данных переменной . . . . .	96
Требование явного объявления переменных . . . . .	103
Константы . . . . .	105
Создание именованных констант . . . . .	106
Область действия констант . . . . .	106
Написание литеральных констант . . . . .	107
Поиск имеющихся внутренних констант с помощью Object Browser . . . . .	111
Получение данных от пользователя . . . . .	112
Выражения в Visual Basic . . . . .	114
Совместимость типов данных . . . . .	116
Преобразования численных типов . . . . .	118
Арифметические операции . . . . .	121
Сравнение строк . . . . .	128

Сравнение объектов . . . . .	132
Логические операторы . . . . .	133
Конкатенация строк . . . . .	138
Приоритеты выполнения операций при вычислении сложных выражений . . . . .	141
<b>Глава 4. Функции в Visual Basic. . . . .</b>	<b>143</b>
Использование функций в выражениях . . . . .	144
Аргументы и возвращаемое значение функции . . . . .	145
Игнорирование результата функции . . . . .	145
Использование именованных аргументов функции . . . . .	147
Использование других функций VBA . . . . .	151
Математические функции . . . . .	152
Функции преобразования данных . . . . .	153
Функции даты и времени . . . . .	155
Строковые функции . . . . .	156
Использование функций для манипулирования строками . . . . .	158
Удаление ненужных символов . . . . .	159
Определение длины строки . . . . .	160
Сравнение и поиск строк . . . . .	161
Разбиение строки на меньшие части . . . . .	163
Форматирование значений данных . . . . .	167
Использование функций host-приложений . . . . .	177
Создание функций и функций-процедур . . . . .	179
Использование функций-процедур в VBA . . . . .	185
Использование функций пользователя в рабочих листах Excel . . . . .	188
Создание функций для Excel . . . . .	190
<b>Глава 5. Изменение порядка выполнения операторов в VBA . . . . .</b>	<b>195</b>
Простой выбор . . . . .	196
Использование необязательных аргументов . . . . .	200
Выбор ветви с помощью If...Then...Else . . . . .	201
Сложный выбор . . . . .	203
Использование If...Then...ElseIf . . . . .	205
Безусловный переход . . . . .	211
Использование MsgBox для обеспечения возможности выбора . . . . .	213
Дополнительные свойства процедур и функций . . . . .	216
Раннее окончание процедур, функций и целых программ . . . . .	216
Использование оператора Exit . . . . .	216
Использование оператора End . . . . .	219
Дополнительные свойства необязательных аргументов . . . . .	222
Управление передачей аргументов . . . . .	223
Передача аргументов по ссылке и по значению . . . . .	223
Определение способа передачи аргумента . . . . .	224
Рекурсия . . . . .	226
Примеры рекурсивных функций . . . . .	226
Как избежать случайной рекурсии и других проблем . . . . .	228
<b>Глава 6. Введение в объекты и коллекции . . . . .</b>	<b>231</b>
Объекты . . . . .	231
Свойства объекта . . . . .	232
Методы объекта . . . . .	233
Классы объекта . . . . .	233
Использование объектов . . . . .	234
Объявление объектных переменных . . . . .	248
Объекты в выражениях . . . . .	249
Ссылка на объекты с помощью With...End With . . . . .	253



Работа с коллекциями объектов и контейнерами объектов . . . . .	254
Добавление объектов к коллекциям . . . . .	257
Ссылка на конкретные объекты в коллекции или контейнере . . . . .	258
Фигуры в слое векторной графики . . . . .	259
Добавление к коллекции Shapes «автофигур» . . . . .	259
Добавление к коллекции Shapes специальных фигур . . . . .	262
Использование Object Browser для работы с объектами, методами и свойствами . . . . .	271
<b>Глава 7. Повторение действий в Visual Basic: циклы и массивы . . . . .</b>	<b>275</b>
Команды организации циклов . . . . .	275
Повторение цикла фиксированное число раз: циклы For . . . . .	277
Циклы Do . . . . .	286
Как прервать выполнение макроса или процедуры . . . . .	288
Использование циклов, тестирующих условия до выполнения тела цикла . . . . .	289
Использование циклов, тестирующих условия после выполнения тела цикла . . . . .	294
Вложенные циклы . . . . .	297
Вложение циклов For . . . . .	297
Вложенные циклы Do . . . . .	299
Пример использования коллекции в Word . . . . .	301
Массивы . . . . .	302
Размерность массива . . . . .	302
Статические и динамические массивы . . . . .	304
Оператор Option Base . . . . .	304
Объявление массивов . . . . .	305
Использование массивов . . . . .	306
Использование ReDim с динамическими массивами . . . . .	309
Функции LBound и UBound . . . . .	315
Использование Erase для очистки или удаления массивов . . . . .	316
Использование массивов в качестве аргументов процедур и функций . . . . .	318
<b>Глава 8. Управление файлами с помощью VBA . . . . .</b>	<b>321</b>
Управление файлами . . . . .	321
Что такое управление файлами . . . . .	321
Возможности VBA по управлению файлами . . . . .	322
Атрибуты файла . . . . .	323
Получение атрибутов файла . . . . .	325
Изменение атрибутов файла . . . . .	328
Как находить файлы . . . . .	329
Использование функции Dir для нахождения файлов . . . . .	329
Использование встроенных диалоговых окон Excel для получения имен файлов . . . . .	334
Использование метода GetOpenFilename . . . . .	334
Использование метода GetSaveAsFilename . . . . .	338
Использование встроенных диалоговых окон Word для получения имен файлов . . . . .	341
Использование Word-диалогового окна Open . . . . .	342
Использование Word-диалогового окна Сохранение документа . . . . .	344
Работа с дисками и папками . . . . .	346
Получение пути текущей папки и буквенной метки диска . . . . .	346
Изменение текущей папки . . . . .	347
Создание дисковых папок . . . . .	349

Копирование и удаление файлов . . . . .	351
Копирование файлов . . . . .	351
Удаление файла . . . . .	352
Переименование или перемещение файлов . . . . .	353
Получение информации о файлах . . . . .	355
Получение времени и даты создания/модификации файла . . . . .	355
Получение длины файла . . . . .	356
<b>Глава 9. Элементы диалоговых окон . . . . .</b>	<b>359</b>
Формы пользователя . . . . .	359
Свойства объекта UserForm . . . . .	361
Методы объекта UserForm . . . . .	362
События и событийные процедуры . . . . .	363
Примеры программ модуля класса формы . . . . .	365
Элементы управления . . . . .	370
Использование Toolbox (панели элементов) . . . . .	377
Добавление к форме элементов управления . . . . .	378
Редактирование элементов управления на форме . . . . .	387
Копирование, вставка и удаление элементов управления . . . . .	388
Редактирование или форматирование заголовков элементов управления . . . . .	388
Управление последовательностью перехода . . . . .	389
Задание свойств формы и элементов управления в режиме разработки . . . . .	390
Использование дополнительных элементов управления . . . . .	392
<b>Глава 10. Управление host-приложениями VBA . . . . .</b>	<b>397</b>
Работа с Excel . . . . .	397
Работа с объектами Worksheet . . . . .	406
Методы, возвращающие объекты Range . . . . .	411
Использование метода Cells . . . . .	413
Работа с Cells и Ranges . . . . .	418
Работа с объектами Word . . . . .	423
Работа с объектами Document . . . . .	424
Работа с объектами Template . . . . .	432
Компоненты объекта Document . . . . .	435
Как задать диапазон . . . . .	436
Работа с объектом Selection . . . . .	441
Как переместить или «свернуть» объекты Selection и Range . . . . .	441
Добавление текста . . . . .	444
Как вырезать, скопировать, вставить и удалить текст . . . . .	447
<b>Глава 11. Отладка VB-кода. Поиск и устранение ошибок . . . . .</b>	<b>451</b>
Типы ошибок . . . . .	451
Средства отладки . . . . .	454
Режимы отладки . . . . .	455
Режим останова . . . . .	456
Переход в режим останова из окна сообщения об ошибке . . . . .	456
Точки останова . . . . .	457
Использование оператора Stop . . . . .	458
Переход в режим останова прерыванием исполнения кода . . . . .	459
Выход из режима останова . . . . .	459
Использование команды Step Into . . . . .	460
Наблюдение за значениями выражений . . . . .	461
Редактирование наблюдаемого выражения . . . . .	463
Удаление наблюдаемого выражения . . . . .	464

Использование команды Step Over . . . . .	464
Использование окна Locals . . . . .	465
Трассировка вызовов процедур . . . . .	465
Использование окна Immediate . . . . .	467
Оператор Debug.Print . . . . .	468
Оператор Debug.Assert . . . . .	468
Обработчик ошибок . . . . .	469
Выход из блока обработки прерывания по ошибке . . . . .	471
Программная обработка ошибок при помощи оператора On Error . . . . .	475
<b>Приложение А. Операторы в VBA . . . . .</b>	<b>477</b>
AppActivate . . . . .	477
Beep . . . . .	477
Call . . . . .	478
ChDir . . . . .	479
ChDrive . . . . .	479
Close . . . . .	479
Синтаксис . . . . .	479
Const . . . . .	480
Date . . . . .	481
Declare . . . . .	481
Deftype . . . . .	483
DeleteSetting . . . . .	484
Dim . . . . .	485
Do...Loop . . . . .	486
End . . . . .	486
Enum . . . . .	487
Erase . . . . .	488
Error . . . . .	489
Event . . . . .	490
Exit . . . . .	490
FileCopy . . . . .	491
For Each...Next . . . . .	492
For...Next . . . . .	493
Function . . . . .	493
Get . . . . .	496
GoSub...Return . . . . .	498
GoTo . . . . .	499
If...Then...Else . . . . .	499
Implements . . . . .	501
Input # . . . . .	501
Kill . . . . .	502
Let . . . . .	502
Line Input # . . . . .	503
Load . . . . .	504
Lock, Unlock . . . . .	504
LSet . . . . .	505
Mid . . . . .	505
MkDir . . . . .	506
Name . . . . .	506
On Error . . . . .	507
On...GoSub, On...GoTo . . . . .	509
Open . . . . .	510
Option Base . . . . .	511
Option Compare . . . . .	511
Option Explicit . . . . .	511

Option Private . . . . .	512
Print # . . . . .	512
Private . . . . .	513
Property Get . . . . .	514
Property Let. . . . .	517
Синтаксис . . . . .	517
Property Set. . . . .	519
Public. . . . .	521
Put . . . . .	522
RaiseEvent . . . . .	524
Randomize . . . . .	525
ReDim . . . . .	525
Rem . . . . .	526
Reset . . . . .	526
Resume . . . . .	526
Примеры . . . . .	527
Rmdir . . . . .	529
RSet . . . . .	529
SaveSetting . . . . .	529
Seek . . . . .	530
Select Case . . . . .	531
SendKeys . . . . .	532
Set . . . . .	534
SetAttr . . . . .	534
Static. . . . .	535
Stop . . . . .	536
Sub. . . . .	536
Time . . . . .	538
Type . . . . .	538
Unload . . . . .	539
While...Wend . . . . .	539
Width # . . . . .	540
With . . . . .	540
Write #. . . . .	540
<b>Приложение Б. Функции в VBA . . . . .</b>	<b>543</b>
Abs. . . . .	543
Array . . . . .	543
Asc. . . . .	543
Atn. . . . .	544
CallByName . . . . .	544
Функции преобразования типов . . . . .	545
Choose . . . . .	545
Chr. . . . .	546
Cos . . . . .	546
CreateObject. . . . .	546
CurDir . . . . .	547
CVErr. . . . .	547
Date . . . . .	548
DateAdd . . . . .	548
DateDiff . . . . .	548
DatePart . . . . .	549
DateSerial. . . . .	550
DateValue . . . . .	551
Day. . . . .	551
DDB . . . . .	551
Dir . . . . .	552
Environ . . . . .	553



EOF	553
Error	553
Exp.	554
FileAttr	554
FileDateTime	554
FileLen	555
Filter	555
Fix	556
Format	556
FormatCurrency	557
FormatDateTime	558
FormatNumber	559
FormatPercent	559
FreeFile	560
FV	560
GetAllSettings	561
GetAttr	562
GetObject	562
GetSetting	563
Hex	564
Hour	564
IIf	564
Input	565
InputBox	565
InStr	566
InStrRev	567
Int	568
IPmt	568
IRR	569
IsArray	570
IsDate	570
IsEmpty	571
IsError	571
IsMissing	571
IsNull	572
IsNumeric	572
IsObject	572
Join	573
LBound	573
LCase	574
Left	574
Len	574
Loc	575
LOF	575
Log	575
LTrim	576
Mid	576
Minute	576
MIRR	577
Month	577
MonthName	577
MsgBox	578
Now	579
NPer	580
NPV	580
Oct	581
Partition	581

Pmt . . . . .	582
PPmt . . . . .	583
PV . . . . .	584
QBColor. . . . .	585
Rate . . . . .	586
Replace . . . . .	587
RGB . . . . .	588
Right. . . . .	589
Rnd. . . . .	589
Round . . . . .	590
RTrim . . . . .	590
Second . . . . .	591
Seek . . . . .	591
Sgn. . . . .	591
Shell . . . . .	592
Sin . . . . .	593
SLN . . . . .	593
Space . . . . .	593
Spc . . . . .	593
Split . . . . .	594
Sqr . . . . .	595
Str . . . . .	595
StrComp . . . . .	595
StrConv. . . . .	596
StrReverse . . . . .	597
String. . . . .	597
Switch . . . . .	597
SYD . . . . .	598
Tab . . . . .	598
Tan . . . . .	599
Time . . . . .	599
Timer . . . . .	599
TimeSerial. . . . .	599
TimeValue. . . . .	600
Trim . . . . .	600
TypeName. . . . .	600
UBound . . . . .	601
UCase. . . . .	602
Val . . . . .	602
VarType. . . . .	602
Weekday . . . . .	603
WeekdayName. . . . .	604
Year . . . . .	605

<b>Предметный указатель. . . . .</b>	<b>607</b>
--------------------------------------	------------

# Введение

Несмотря на то, что многие задачи Microsoft Office позволяет выполнить и без программирования, одним из наиболее важных и полезных его свойств является возможность автоматизации процессов взаимодействия пользователя и приложений Microsoft Office, которая позволяет решать, практически, все современные бизнес-задачи: от создания простых документов и отчетов до полной автоматизации документооборота с использованием систем управления базами данных. Это обстоятельство связано с тем, что все приложения Microsoft Office поддерживают язык программирования Visual Basic for Applications (VBA), совместимый с Visual Basic 6. Важнейшим достоинством VBA является возможность объединять любые приложения Microsoft Office для выполнения одной или нескольких задач.

Привлекательная особенность VBA в том, что он очень удобен для первого знакомства с программированием в среде Windows. В наиболее используемых приложениях Microsoft Office имеется Редактор VB (Visual Basic Editor) и макрорекордер — средство кодирования действий пользователя для последующего их повторения. Эти инструменты, кроме всего прочего, могут с успехом использоваться для обучения программированию — без утомительного поиска информации по различного рода справочникам вы можете «записать» многие свои действия посредством макрорекордера, а затем рассмотреть результирующий код. Редактор VB — это не только средство редактирования кода, но и замечательное средство отладки кода и изучения языка Visual Basic, так как вы можете в интерактивном режиме проверить действие любой синтаксической конструкции.

В настоящее время Microsoft Office является не набором программных продуктов индивидуального применения, а комплексной интегрированной системой, в которую, кроме уже знакомых многим пользователям средств (Word, Excel и т.д.), входят серверы, службы и приложения для настольных компьютеров, предназначенные для использования при решении широкого спектра бизнес-задач. Несмотря на то, что многие практические задачи с использованием Microsoft Office можно выполнить без программирования, имеет смысл изучения языка VBA для расширения спектра решаемых проблем.

Приложения Microsoft Office обеспечивают доступ ко многим современным системам баз данных. Пользователь может внести результаты сложного SQL-запроса из удаленной базы данных коллективного доступа в любой создаваемый им отчет или проект. Применение для этих целей VBA-кода расширяет такие возможности и улучшает пользовательский интерфейс.

В этой книге содержится вводный курс по использованию языка и системы VBA в основном для Microsoft Word 2007, Microsoft Excel 2007, хотя почти все Word/Excel-программы (процедуры) будут работать и для Word/Excel 2000/2/3. Книга предназначена для начинающих программировать в среде Windows 2000/XP с использованием в качестве базовых объектов Word и Excel, таких как документы, рабочие книги, листы, презентации и так далее, т.е. объекты, которые хорошо известны пользователям этих приложений. Только теперь имеется возможность посмотреть на эти объекты «изнутри», узнать их строение: свойства, методы, взаимосвязи.

Поработав с этой книгой, вы сможете создавать как простые макросы, помогающие автоматизировать рутинную повторяющуюся работу над документами и электронными таблицами, так и довольно сложные приложения, обрабатывающие данные в диалоговых окнах, обеспечивающих пользователя самими современными интерфейсными средствами. Далее вам ничто не мешает начать изучение системы Visual Basic (системы программирования подобной C++Builder или Delphi) и Visual Basic .NET, открывающих перед вами еще большие горизонты и позволяющих также использовать объекты приложений Microsoft Office.

Большую пользу эта книга может принести менеджерам, которые в своей профессиональной деятельности много времени работают за компьютерами и решают задачи, в которых нет места таким системам программирования, как Visual C++, Delphi и так далее. Более того, поскольку менеджерам чаще всего приходится использовать продукты Microsoft Office, именно VBA им может пригодиться больше всего, потому что это — «родной» язык для этих продуктов. К тому же возможности VBA совсем не уступают другим языкам программирования в Windows и постоянно растут.

## Элементы оформления книги

Для улучшения восприятия материала книги в ней используются следующие элементы оформления:

- впервые встречающиеся термины выделены курсивом;
- ключевые слова языка программирования, наименования элементов управления выделены шрифтом Courier New Cyr, **Bold**;
- команды меню, наименования кнопок, окон и других элементов управления диалоговых окон выделены шрифтом Arial, **Bold**;
- имена переменных VB-кода, наименования файлов и просто текст, на который следует обратить внимание, отмечен как **Bold**.



# Введение в макросы

Далеко не все пользователи Word и Excel и других продуктов Microsoft Office знают о том, что эти приложения совершенно не настроены для выполнения тех задач, для которых предназначены. Мало кто из пользователей, активно работающих, например, с Word или Excel, когда-либо вообще читал пособие по работе с этими продуктами — просто установили их (может быть, при помощи кого-нибудь поопытнее, чем сами) и начали создавать полезные (и не очень) документы. Между тем, все приложения Microsoft Office являются такими же конструкторами, как и те компьютеры, которые мы обычно используем с этими приложениями. Для многих типов задач, решаемых при помощи этих приложений, их нужно настраивать. Примером настроек могут служить многочисленные шаблоны документов в Word, о которых, впрочем, тоже знают далеко не все пользователи. Даже среди знающих о существовании шаблонов и стилей (!) многие не знают самых простых дополнительных возможностей этих продуктов, которые реализуются посредством макросов. Конечно, и Word, и Excel предоставляют достаточно широкий набор функций для выполнения самых разнообразных офисных задач, но все же знание и использование макросов VBA дает такие преимущества, которые невозможно переоценить.

Перед тем как вы начнете писать собственные макросы, вам следует хорошо понять, что такое макросы и чем отличается VBA-программируемый макрос от записанного макрорекордером.

Независимо от используемых вами операционной системы и программных приложений вы часто выполняете одни и те же последовательности команд для многих рутинных задач. Вместо повторения последовательности команд каждый раз, когда вам необходимо выполнить какую-либо задачу, вы можете создать *макрос (macro)*, который вместо вас будет выполнять эту последовательность. Макросы позволяют вводить одиночную команду, выполняющую ту же задачу, для реализации которой вам было бы необходимо вводить несколько команд вручную. Например, если вам нужно для многих абзацев большого документа изменить стили на другие (стандартные, ранее вами подготовленные или экспортированные из других документов), вы могли бы записать действия, связанные с поиском и выбором стиля, при помощи макрорекордера с назначением записанному макросу «горячих клавиш». В этом случае для изменения стиля абзаца вам достаточно было бы поместить курсор вставки на нужный абзац и нажать комбинацию клавиш. Чем больше подобных изменений необходимо внести в документ, тем больший эффект можно получить от такого элементарного записанного макроса или нескольких: сколько используемых стилей — столько и макросов с «горячими клавишами».

Записанные макрорекордером последовательности команд первоначально назывались *макрокомандами* (*macro-commands*). Сейчас этот термин сократился до более простого слова *макрос*. (Термин *macro* используется как префикс в нескольких словах в английском языке; он произошел от греческого слова, означающего *расширенный* или *растянутый*). Применительно к информатике и программным приложениям под словом *макрос* всегда подразумевается макрокоманда.

Макросы, кроме удобства, имеют и другие преимущества. Поскольку компьютеры приспособлены для выполнения повторяющихся задач больше, чем люди, запись макрорекордером неоднократно выполняемых команд повышает точность и скорость работы. Другим преимуществом использования макросов является то, что при их выполнении обычно нет необходимости в присутствии человека-оператора. В случае, если макрос очень длинный или выполняет операции, требующие значительного времени (например, поиск в базе данных и сортировка), вы можете оставить работающий компьютер и, например, попить кофе или переключиться на другое приложение, если, конечно, вы свято верите в то, что Windows — многозадачная система.

Макрорекордер (или просто «рекордер») записывает *все* действия пользователя, включая ошибки и неправильные запуски. Когда программа воспроизводит макрос, она выполняет каждую записанную рекордером команду точно в такой последовательности, в которой вы их выполняли во время записи. Первые макрорекордеры имели серьезный недостаток. Если вы записывали длинную серию действий, содержащую небольшую ошибку, единственной возможностью удалить эту ошибку являлась повторная запись макроса. Кроме того, если вам было необходимо внести небольшое изменение в длинный макрос, то также приходилось перезаписывать весь макрос. Перезапись длинного макроса часто приводила к дополнительным ошибкам в новой записи. По этим причинам разработчики программного обеспечения добавили макрорекордерам возможность редактирования макросов, чтобы вы могли легко исправлять небольшие ошибки или вносить другие изменения в макрос без его полной перезаписи.

## Из истории VBA

Несмотря на то что Visual Basic for Applications (VBA) — это относительно новый продукт, предпосылки его появления имеют историю, почти столь же долгую, что и вся компьютерная индустрия. Язык VBA является современным диалектом языка программирования BASIC, который был создан в начале 60-х годов. (BASIC — это сокращение от Beginner's All Purpose Symbolic Instruction Code.)

Хотя по сегодняшним стандартам первоначальный язык BASIC имел значительные ограничения, изучать и понимать его было легко, что способствовало его быстрому распространению. Версии языка BASIC создавались (и все еще создаются) для использования на всех типах компьютеров. Версия Microsoft GWBASIC (GW означает Graphics Workshop) была одним из первых языков программирования, доступных для компьютеров, которые эволюционировали в современные персональные компьютеры. GWBASIC поставлялся с версиями MS-DOS до Version 5.0. Ранние ПК, выпускаемые IBM, имели даже версию BASIC, встроенную в микросхемы ROM (Read Only Memory) компьютера.

С годами первоначальная структура и спецификации BASIC были улучшены. По мере совершенствования и изменения технологии для языков программирования различные создатели программного обеспечения добавляли некоторые возможности к первоначальному BASIC. Современные диалекты BASIC обычно имеют многие или все свойства, существующие в других более поздних языках программирования, таких как Pascal или C.

В конце 80-х годов Microsoft опубликовала значительно улучшенную версию языка BASIC, названную QuickBASIC. QuickBASIC включал почти все возможности современных систем разработки программного обеспечения. Microsoft поставляла версию QuickBASIC с DOS Version 6.0 и более поздними версиями (но не Windows 95).

После нескольких версий QuickBASIC в 1992 году Microsoft представила Visual Basic for Windows. Как и в случае с QuickBASIC, Visual Basic for Windows был дополнен современными возможностями и тесно интегрирован в среду Windows. Visual Basic предоставляет команды для создания и управления необходимыми элементами программы в Windows: диалоговыми окнами, линейками меню, раскрывающимися списками, командными кнопками, флажками, панелями инструментов и так далее. В частности, Visual Basic включает необходимые команды для использования Object Linking and Embedding (OLE) и Dynamic Data Exchange (DDE) для связи или совместного использования данных с другими приложениями Windows. Visual Basic является существенно новым языком программирования для Windows со своими корнями в BASIC.

В то время как BASIC развивался и улучшался, изменялись макрорекордеры, используемые в программных приложениях. С годами макросы приложений постепенно становились все сложнее в ответ на пожелания пользователей сделать макросы более гибкими в действии и более легкими для сопровождения. Многие макроязыки стали включать возможности, подобные тем, которые обычно имеются только в законченных языках программирования.

Макроязыки многих приложений значительно изменяются от продукта к продукту. Это означает, что вам может понадобиться изучить несколько различных макроязыков; в результате эффективность вашей работы может снизиться во время изучения нового макроязыка. Чтобы избежать необходимости изучения нового макроязыка для каждого продукта, Microsoft начала включать элементы языка BASIC в макроязыки своих продуктов. Примером может служить макроязык для Word for Windows фирмы Microsoft (до Word 97), известный как WordBASIC, тогда как язык программирования для Access фирмы Microsoft был известен как Access Basic.

Для унификации макроязыков в своих приложениях Windows и для интеграции приложений на этих макроязыках с DDE и OLE Microsoft создала специальную версию языка Visual Basic, названную Visual Basic for Applications (сокращенно VBA). Excel 5 был первым продуктом, включающим Visual Basic for Applications (VBA). С появлением Microsoft Office 97 VBA реализуется в Word, Excel, Access. В основные приложения Office (Word, Excel, PowerPoint), начиная с версии 2000, включен редактор сценариев — Microsoft Visual Basic Script Editor, который позволяет редактировать Web-страницы (например, в Word) и придать им динамику. Язык VBScript (Visual Basic Scripting Edition) входит в семейство языков Visual Basic и, являясь упрощенной версией языка Visual Basic, позволяет программировать документы, отображаемые браузерами World Wide Web.

Visual Basic for Applications в основном является тем же, что и Visual Basic for Windows, с некоторыми небольшими различиями. Макропрограммы VBA сохраняются в файловом формате, используемом приложением, в котором вы написали макрос Visual Basic for Applications, а не в отдельных текстовых файлах. Например, макропрограммы VBA, созданные в Excel, сохраняются в файле рабочей книги Excel, программы VBA в Word сохраняются в файле документа, а программы VBA в Access — в файле данных Access. Для выполнения макропрограммы VBA вы должны сначала запустить приложение, в котором написали этот макрос. Например, вы не можете запустить макрос Excel VBA из любой другой программы, кроме Excel. Несмотря на то, что основные возможности VBA остаются теми же в приложениях, каждое приложение добавляет специальные команды и объекты (в зависимости от конкретного приложения) в Visual Basic for Applications. Например, VBA в Excel содержит много команд, которые относятся только к рабочим листам и задачам, которые вы можете выполнить с рабочим листом. Аналогично, VBA в Word содержит команды, относящиеся только к операциям над текстом в документе, тогда как VBA в Access содержит команды, относящиеся только к операциям с базой данных, и так далее.

Если до появления MS Office 97 средства программирования в офисных пакетах предназначались в основном для создания макрокоманд, расширяющих возможности конкретных приложений, то с выходом MS Office 97 специалисты получили универсальную систему программирования для прикладных программ на базе Visual Basic. Впервые MS Office был представлен как единая платформа для создания бизнес-приложений, предназначенных для решения специализированные задачи пользователей. Это новое качество MS Office обуславливалось специальным выпуском для разработчиков — Developer Edition. По сведениям Microsoft, в настоящее время MS Office в качестве платформы создания бизнес-приложений используют несколько миллионов профессиональных разработчиков.

Пакет Microsoft Office по мере развития из набора приложений индивидуального применения превратился в комплексную интегрированную систему, в которой ее разработчики большое внимание уделили именно совместной работе пользователей. В Microsoft Office System, кроме привычных пользователям средств (Word, Excel и т.д.), входят серверы, службы и приложения, разработанные для совместного применения при решении широкого круга бизнес-задач. Microsoft Office 2007 позволяет эффективно разрабатывать мощные и удобные приложения и предоставлять пользователям информацию из любых источников в рамках удобного и многофункционального интерфейса. Разработчики могут создавать приложения, использующие средства языка XML и Web-служб, возможности смарт-тегов инструменты Visual Studio .NET и информацию из баз данных таких систем, как Microsoft Access 2007 и Microsoft SQL Server 2000/5.

Поскольку книга адресована в основном тем, кто только начинает программировать в среде Windows, здесь рассматривается VBA-программирование только для двух наиболее используемых приложений — Word и Excel. Основной целью, которую преследует автор книги, является представление основ языка VBA, который, как утверждает Microsoft, начиная с версии MS Office 2000, не отличается от обычного Visual Basic 6. VBA-программирование (разработка макросов) рассматривается для самого распространенного (по крайней



мере, в России) пакета MS Office 2007, в состав которого входят такие приложения, как Microsoft Office Access, Microsoft Office Word, Microsoft Office Excel, Microsoft Office Outlook, Microsoft Office PowerPoint, Microsoft Office InfoPath, Microsoft Office OneNote, Microsoft Office Publisher.

## Для чего нужен язык программирования VBA

Поскольку вы можете использовать макрорекордер в Word или Excel для записи ваших действий в макрос (а в Access для этой цели использовать своего рода конструктор) и затем воспроизводить их, сначала может показаться, что изучать VBA необязательно. Однако одни записанные макросы не могут удовлетворить все ваши потребности. Записанный макрорекордером макрос может только воспроизводить без отклонений каждое действие в той же последовательности, в которой вы первоначально выполняли эти действия. VBA можно использовать для улучшения макросов, записанных макрорекордером, значительно повышая их мощь и возможности.

Записанные макрорекордером макросы лишены гибкости, поэтому они не могут реагировать на изменившиеся или меняющиеся условия. На языке VBA вы можете написать макрос, который проверяет соответствие различным предопределенным условиям и выбирает последовательность действий на основе этих условий. Если вы, например, выполняете какой-либо Excel-макрос, который пытается выбрать рабочий лист с именем «Продажи», в то время, когда в книге нет такого рабочего листа, записанный вами макрос будет выполняться неправильно и Excel выведет на экран сообщение об ошибке. Добавив VBA-команды к этому записанному макросу, вы могли бы с его помощью выполнить проверку наличия этого определенного рабочего листа перед его выбором или даже вставить новый рабочий лист и дать ему соответствующее имя, если нужный рабочий лист не существует.

Что касается повторяющихся действий в самом макросе, *записанные рекордером* макросы имеют значительные ограничения. Если вам необходимо, чтобы записанный макрос повторял какое-либо действие несколько раз, вы должны вручную повторить это действие нужное количество раз, когда записываете макрос. Такой макрос всегда повторяет это действие одинаковое количество раз, всякий раз, когда вы его запускаете, до тех пор, пока вы не отредактируете или не перезапишете его. Напротив, VBA-макрос, *написанный пользователем*, может использовать предопределенные условия или вводимые пользователем данные для повторения действия различное количество раз или для принятия решения о необходимости выполнения этого действия.

В качестве примера вы можете записать рекордером макрос для изменения ширины нескольких смежных столбцов в рабочем листе Excel. Если вам необходимо, чтобы макрос изменял ширину первых трех столбцов в рабочем листе, вы должны вручную повторить операцию изменения размера для каждого из трех столбцов при записи макроса. Записанный макрос будет только (и всегда) изменять ширину первых трех столбцов в рабочем листе. Вы не можете использовать тот же макрос для изменения ширины двух или четырех столбцов. Кроме того, если вы изменили ширину первых трех столбцов, записанный макрос будет выполнять действие только над тремя первыми столбцами. Вы не

сможете использовать тот же макрос для изменения ширины столбцов со второго по четвертый. Добавив команды VBA к этому записанному рекордером макросу, вы можете создать макрос, который запрашивает, размер скольких и каких столбцов необходимо изменить, и даже позволяет задавать новую ширину столбцов.

Эти два примера представляют пару простейших задач, которые вы можете выполнить с командами VBA в макросах. Существует много обстоятельств, при которых вам может понадобиться добавлять команды принятия решений к макросам, записанным макрорекордером. Единственный способ получить такие возможности — это вручную добавить операторы VBA к записанному макросу.

Кроме улучшения определенных макросов, записанных макрорекордером, вы можете использовать VBA для соединения, организации и управления несколькими записанными макросами, с помощью которых вы выполняете сложную общую задачу, состоящую из нескольких меньших задач. Например, вы можете регулярно импортировать данные из базы данных в рабочий лист Excel, форматировать эти данные для вывода на экран, генерировать диаграмму из этих данных и затем распечатывать эту диаграмму и отформатированный отчет.

Для того чтобы объединить эти отдельные задачи для создания единственной задачи, выполняемой одним макросом, вы можете сначала написать отдельный макрос для каждой из отдельных задач: для импортирования данных, форматирования этих данных для последующего отображения, создания диаграммы и печати данных. Далее вы могли бы организовать записанные макросы так, чтобы они выполнялись в нужной последовательности одним макросом, который вы напишете посредством команд VBA.

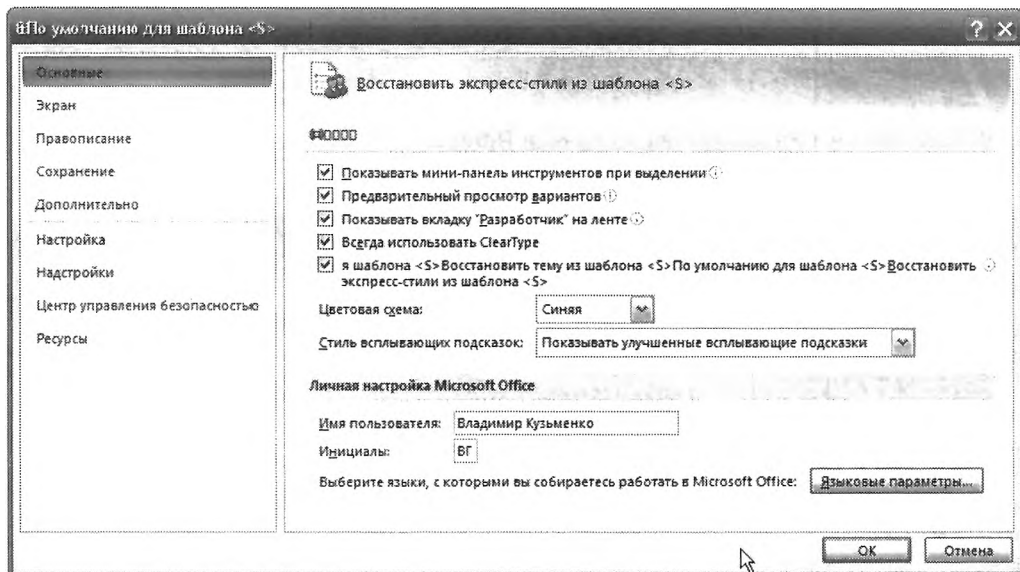
С помощью макросов можно создавать пользовательские меню, диалоговые окна, которые могут до неузнаваемости изменить интерфейс всем известным продуктам Word и Excel. Уместно здесь отметить также и возможность создания разнообразной системы проверки данных, вводимых пользователем в диалоговых окнах. Когда вы научитесь писать программы на языке VBA, вы, скорее всего, уже никогда не станете начинать создание макроса с использования рекордера. С другой стороны, довольно часто будете обращаться к нему за помощью при выполнении задач, которые раньше делать не приходилось: рекордер запишет макрос, а вы узнаете, ранее не используемые вами свойства или методы.

## Создание макросов

Существует мнение, что изучение языка VBA легче всего начать с записи нового макроса при помощи рекордера. Поэтому большая часть литературы по VBA следует схеме, по которой читателю предлагается сначала посмотреть, как макросы создаются самими продуктами, такими как Word и Excel, а затем, осторожно внедряя свои инструкции в работающие макросы, изучить новый для них язык. Мне же представляется более продуктивным другой подход, при котором вы начнете писать макросы (с использованием Visual Basic Editor — Редактора), как обычные программы на новом для вас языке программирования, и каждый свой шаг будете проверять контрольными запусками кода.

## Создание макроса в Word

Запустите Microsoft Office Word 2007, добавьте к панели **Ribbon** вкладку **Разработчик (Developer)**, для чего нужно, щелкнув кнопку **Office**, выбрать **Параметры Word** и в появившемся окне для группы параметров **Основные** установить флажок **Показывать вкладку «Разработчик» на ленте** (рис. 1.1).



**Рис. 1.1.** Чтобы вкладка **Разработчик** появилась на панели **Ribbon**, нужно, щелкнув кнопку **Office**, выбрать **Параметры Word** и в появившемся окне для группы параметров **Основные** установить флажок **Показывать вкладку «Разработчик» на ленте**

На вкладке **Разработчик** (рис. 1.2) расположены команды, предназначенные для работы с VBA, макросами, элементами управления, XML, шаблонами и др.

Для вывода диалогового окна **Microsoft Visual Basic** (Редактора Visual Basic) щелкните на вкладке **Разработчик** кнопку **Visual Basic** (в левом углу вкладки) или нажмите комбинацию клавиш **Alt+F11**.

Далее следует создать модуль, в котором будет размещаться код вашего первого макроса. Щелкните для этого на значке **Project(<имя документа>)** в окне **Project**, а затем — на кнопке с вертикальной стрелкой рядом с кнопкой **Insert Module** (вторая кнопка сверху слева) и в появившемся меню выберите **Module**. (рис. 1.3).

Не вникая в маловажные для первого раза детали, поместите курсор в самое большое окно (**Code**) и введите следующий текст без нумерации строк (далее мы будем разывать это *листингом*):

### Листинг 1.1. Первый макрос в Word

```
1: Sub MyFirstMacro()  
2:   MsgBox ("Макрос в Word!")  
3: End Sub
```

Первая и третья строки этого макроса-процедуры являются заголовком и окончанием процедуры, соответственно. Причем **MyFirstMacro** — это имя макроса, под которым он будет узнаваем системой. Вторая строка макроса содержит функцию вывода диалогового окна с текстом «Макрос в Word!».

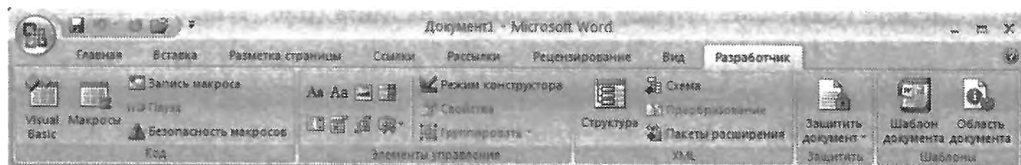


Рис. 1.2. Вкладка **Разработчик** на панели **Ribbon**

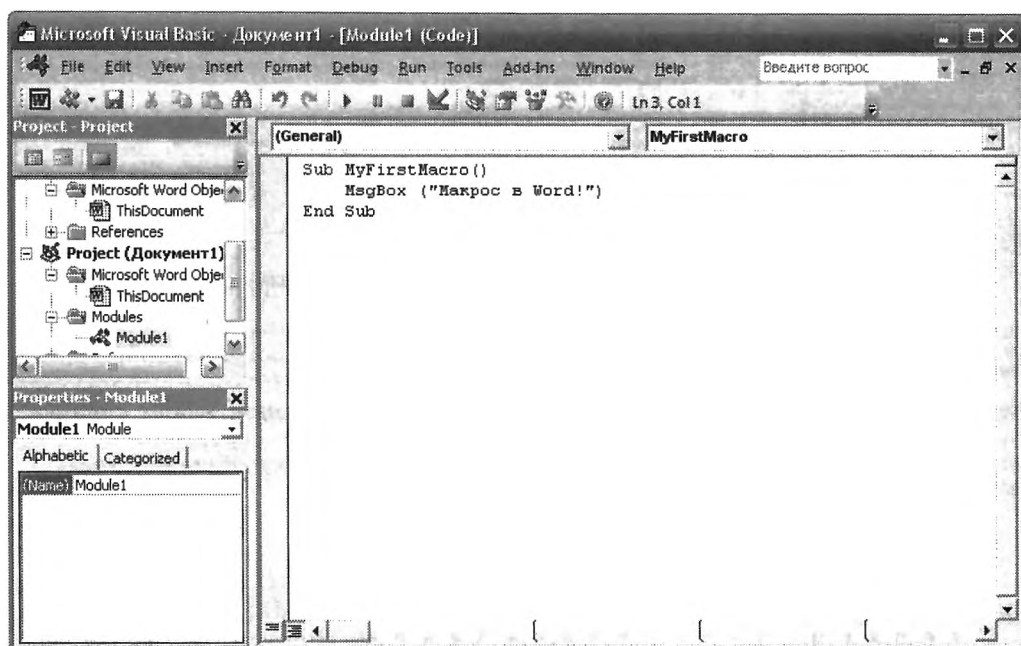


Рис. 1.3. Макрос в окне **Code** Редактора Visual Basic

После ввода инструкций (иногда так называют операторы языка программирования, чтобы не путать их с математическими и другими операторами) в окно **Code** ваш экран должен иметь вид, как на рис. 1.3. Подобным образом можно вывести предварительно вычисленные или введенные (в диалоговом окне) данные. Таким образом, вы — уже на полпути к написанию простейших макросов.

Теперь можно выполнить контрольный запуск макроса, не выходя из Редактора Visual Basic. Выберите **Run | Run Sub/UserForm**. При этом курсор вставки должен находиться на любой строке макроса. Результатом работы макроса будет выдача на экран диалогового окна. Сравните его с рис. 1.4.

**Рис. 1.4**

Результат выполнения вашего первого макроса



Задержитесь немного в Word и запустите созданный макрос непосредственно из Word (все-таки Редактор VB в Word — это не совсем Word). Для этого покиньте Редактор Visual Basic либо при помощи комбинации клавиш **Alt+Q**, либо при помощи меню **File | Close and Return to Microsoft Word**.

После того как вы окажитесь в привычном для всех окне Word, щелкните кнопку **Макросы** на вкладке **Разработчик**. В появившемся диалоговом окне **Макрос (Macros)** выберите (видимо, единственный) уже знакомый вам макрос **MyFirstMacro** и щелкните на кнопке **Выполнить (Run)**.

## Запись новых макросов

В этом разделе рассказывается о том, что необходимо знать для записи какого-либо макроса. Затем вы выполните необходимые шаги для записи определенного макроса как в Word 2007, так и в Excel 2007.

Обычно запись макроса включает четыре основных шага:

1. Задание стартовых условий для макроса. Это означает задание тех условий вашей рабочей среды, которые должны быть соблюдены во время воспроизведения записанного макроса.
2. Запуск макрорекордера и присвоение имени макросу. Одновременно с запуском макрорекордера вы должны дать имя макросу и выбрать место, где он должен быть сохранен. При запуске макрорекордера вы можете выбирать между назначением команды запуска макроса с помощью так называемой «горячей клавиши» или связыванием макроса с каким-либо меню или панелью инструментов (в зависимости от того, записываете ли вы макрос в Excel или в Word).
3. Выполнение действий, которые вам необходимо записать для использования позже. Вы можете записать в макрос любое действие, которое можно выполнить, используя клавиатуру или кнопки мыши, включая выполнение ранее записанных макросов. Конкретные выполняемые вами действия зависят от задачи, которую вам необходимо записать.
4. Остановка макрорекордера. Когда вы останавливаете макрорекордер, ваши действия больше не записываются и не сохраняются в макросе. Как только вы останавливаете рекордер, новый записанный макрос сразу же готов к использованию.

В последующих двух разделах описываются более подробно два первых общих шага при записи макроса. Затем будут представлены конкретные пошаговые инструкции для записи макроса в Word и Excel.

### Задание стартовых условий для макроса

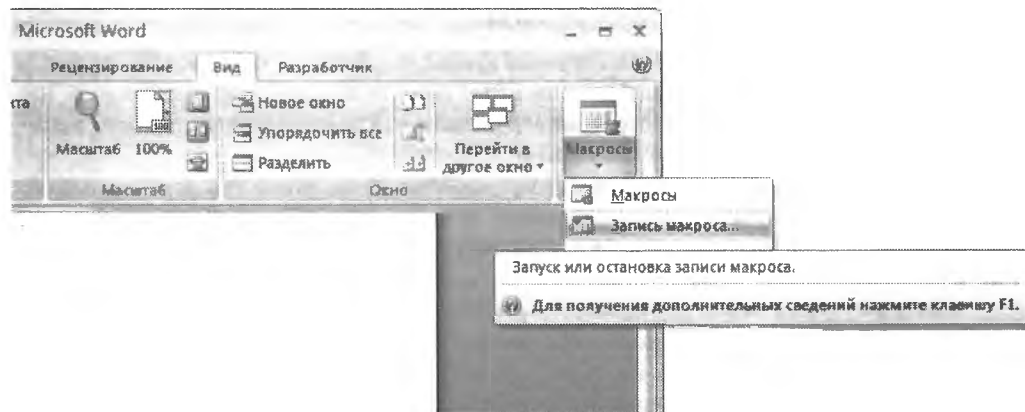
Перед записью какого-либо макроса вы должны задать условия, при которых вы будете запускать его позже. Выполнение (*running* или *executing*) макроса означает воспроизведение записанных в этом макросе инструкций. Предположим, например, что вам необходимо создать макрос, который будет при-

менять определенный шрифт, размер и цвет шрифта в любом выбранном тексте документа в Word. Стартовыми условиями для этого макроса будут открытый документ с выделенным блоком текста. Если вы хотите создать подобный макрос форматирования текста в Excel, стартовыми условиями будут открытая рабочая книга, выбранный рабочий лист и выделенная ячейка или диапазон ячеек.

Вам необходимо задать стартовые условия для макроса перед тем, как запускать макрорекордер, потому что макрорекордер будет записывать *все* действия, которые вы выполняете. Если вы запустите макрорекордер, а затем откроете некоторый документ и выделите текст, эти действия станут частью полученного в результате записи макроса. Ваш законченный макрос будет слишком специфическим: он будет всегда открывать один и тот же документ и форматировать один и тот же блок текста. Для создания общего макроса, который вы можете использовать для форматирования *любого* выбранного текста, вам следует запускать макрорекордер *после* открытия документа и выбора текста.

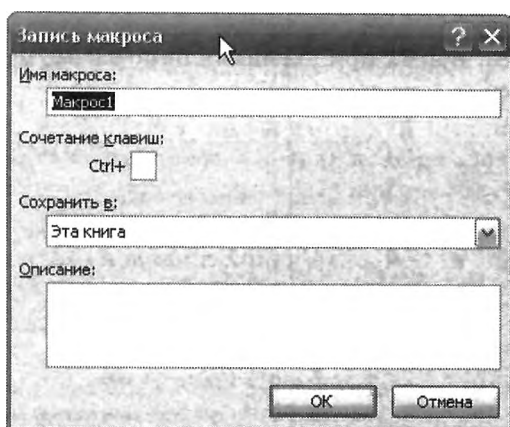
### Запуск макрорекордера и присваивание имени макросу

Независимо от того, работаете ли вы в Word или Excel, запись нового макроса осуществляется одним и тем же способом. На вкладке **Вид** щелкните кнопку **Макросы** и в появившемся меню выберите команду **Запись макроса** (рис. 1.5) или щелкните одноименную кнопку на вкладке **Разработчик** (рис. 1.6). И в Word, и в Excel отображается диалоговое окно **Запись макроса (Record Macro)**, которое используется для того, чтобы вы могли дать имя новому макросу, выбрать место его сохранения и другое.



**Рис. 1.5.** Выберите команду **Запись макроса** на вкладке **Вид**

Диалоговое окно **Запись макроса** может немного отличаться в зависимости от того, записываете ли вы макрос в Word или Excel. Диалоговое окно **Запись макроса** в Word 2007 (рис. 1.7) позволяет по выбору назначать «горячую клавишу» для запуска нового макроса или добавлять новый макрос как кнопку на панели быстрого доступа. В Excel 2007 диалоговое окно **Запись макроса** (рис. 1.8) дает

**Рис. 1.8**Окно **Запись макроса** в Excel

Диалоговые окна **Запись макроса** в Word и в Excel имеют следующие три общих элемента:

1. **Текстовое окно Имя макроса (Macro Name).** По умолчанию VBA выбирает имя макроса, состоящее из слова **Макрос**, за которым следует номер, соответствующий числу макросов, уже созданных в этом сеансе работы. Вам следует давать макросам имена, которые несут определенную информацию о том, что выполняют эти макросы.
2. **Текстовое окно Описание (Description).** Информация в текстовом окне **Описание** макросом непосредственно не используется. Текстовое окно **Описание** предназначено для ввода некоторых замечаний и комментариев о том, что выполняет данный макрос.
3. **Раскрывающийся список Макрос доступен для: (Store macro in:).** Этот раскрывающийся список позволяет выбрать, где следует сохранить записанный макрос. Макросы, записываемые в Word, сохраняются в файле документа (.docm) или в файле шаблона документа **Normal.dotm**. Макросы, записываемые в Excel, сохраняются в файле рабочей книги (.xlsx) или файле **PERSONAL.XLSB**.

## Запись макроса в Word

Предположим, что вы часто форматируете слова или фразы, к которым хотите привлечь внимание в вашей документации, с помощью полужирного шрифта Arial 12-го размера. Поскольку вы форматируете только одиночные слова или короткие фразы, использовать какой-либо именованный стиль невозможно, так как стили Word применимы только к целым параграфам. Поэтому вам приходится применять форматирование шрифта вручную. Выбор шрифта Arial, изменение его размера и применение атрибута полужирного шрифта к тексту включают несколько операций с мышью или клавиатурой. Если вы часто выполняете эту задачу, можете сберечь время и усилия, записав макрорекордером макрос для выполнения нужного форматирования любого выделенного блока текста.

### Задание стартовых условий в Word

Поскольку вам необходимо, чтобы макрос выполнял действие над любым выделенным блоком текста в любом открытом документе, стартовыми условиями для этого макроса являются открытый документ и выделенный блок текста. Для задания стартовых условий в этом упражнении выполните следующие шаги:

1. Запустите Word.
2. Откройте какой-либо документ.
3. Выделите любое слово или фразу в документе. Выделив текст в открытом документе, вы создали необходимые стартовые условия для записи общего макроса с целью применения форматирования символов к любому выбранному тексту.

### Присваивание имени макросу и выбор места для его сохранения

Теперь вы готовы запустить макрорекордер в Word, дать имя макросу, выбрать место для сохранения нового макроса и выбрать дополнительные опции. Выполните следующие шаги:

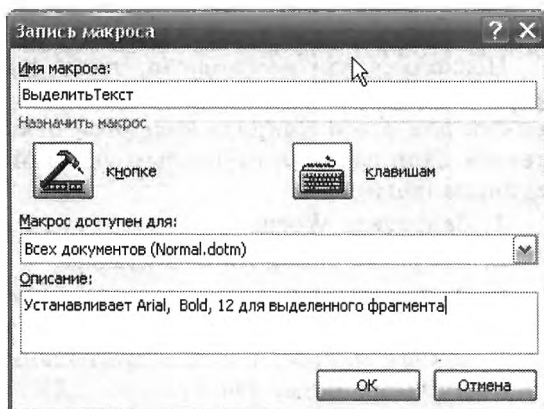
1. На вкладке **Вид** щелкните кнопку **Макросы** и в появившемся меню выберите команду **Запись макроса** или щелкните одноименную кнопку на вкладке **Разработчик** (или, наконец, просто щелкните кнопку **Запись макроса** в строке состояния в нижней части окна приложения). В Word отображается диалоговое окно **Запись макроса**.
2. В текстовом поле **Имя макроса (Macro name)** введите **ВыделитьТекст** в качестве имени макроса. Это имя помогает запомнить, что выполняет макрос.
3. В текстовое поле **Описание (Description)** введите следующий текст: «устанавливает Arial, Bold, 12 для выделенного текста». Этот комментарий поможет вам (и другим) определить назначение макроса.
4. Используйте раскрывающийся список **Макрос доступен для: (Store Macro in:)** для выбора места, в котором будет сохранен записанный макрос. Вы можете сохранить макрос в любом загруженном в данный момент глобальном шаблоне или в текущем документе. Так как вам необходимо, чтобы макрос можно было использовать в любом документе, над которым вы будете работать, выберите **Всех документов Normal.dotm**.
5. Щелкните на кнопке **ОК** для начала записи макроса, если только вы не хотите добавить ваш макрос на панель быстрого доступа или назначить для него «горячую клавишу». Для выбора любой из этих опций выполняйте инструкции, описанные в следующем разделе этой главы. Итак, вы теперь готовы выполнять действия, которые хотите записать.

Конкретные места хранения, доступные в Word в диалоговом окне **Запись макроса**, зависят от того, какие глобальные шаблоны у вас загружены в данный момент. Доступные варианты всегда включают шаблон **Normal.dotm** и текущий активный документ.

На рис. 1.9 показано диалоговое окно **Запись макроса**, заполненное для макроса-примера.



**Рис. 1.9**  
Окно **Запись макроса** в Word,  
заполненное для записи макроса  
**ВыделитьТекст**



### *Использование панели инструментов и клавиатуры для запуска макроса*

Существуют две дополнительные опции, которые вы можете выбрать в диалоговом окне **Запись макроса** в Word: либо назначить «горячую клавишу» для нового макроса, либо связать его с командной кнопкой на **панели быстрого доступа**. Если вы назначаете «горячую клавишу» для макроса, вы можете позже запускать этот макрос нажатием сочетания клавиш. Если вы связываете макрос с командной кнопкой на **панели быстрого доступа**, то можете запускать его, щелкая эту кнопку на **панели быстрого доступа**.

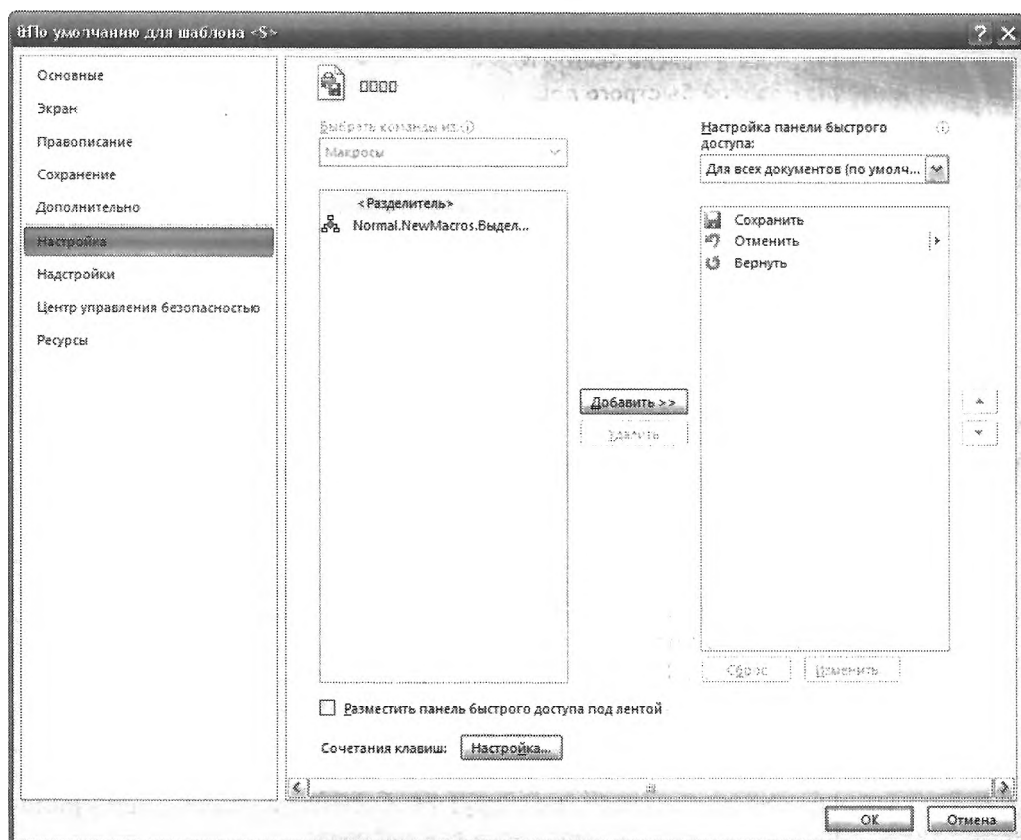
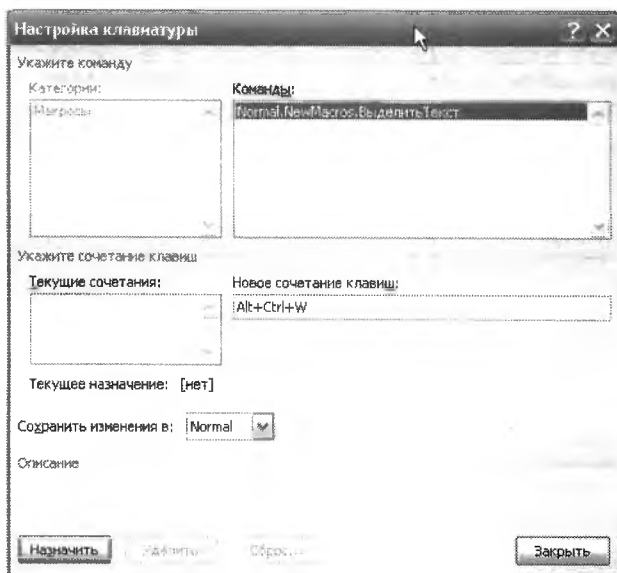
Используйте опции «горячей клавиши» и клавиатуры только в случае, если уверены, что будете очень часто использовать макрос, который собираетесь записывать макрорекордером. Если вы будете назначать «горячие клавиши» всем вашим макросам, у вас скоро не останется неназначенных сочетаний «горячих клавиш».

Щелчок мышью на кнопке **клавишам** для назначения записываемому макросу горячей клавиши приводит к запуску макрорекордера (при выходе из диалога назначения). Поэтому, если вы собираетесь использовать обе опции, выберите **кнопке**, поскольку назначить «горячие клавиши» можно также и из окна, которое вызывается при щелчке на **кнопке**.

Для того чтобы назначить макросу «горячую клавишу», щелкните на кнопке **клавишам** в диалоговом окне **Запись макроса** в Word. Открывается окно **Настройка клавиатуры (Customize Keyboard)**, приведенное на рис. 1.10. Так как вы назначаете «горячую клавишу» новому макросу (который еще не был записан), список **Категории (Categories)** отключен. Имя записываемого вами нового макроса отображается выделенным по умолчанию в списке **Команды (Commands)**.

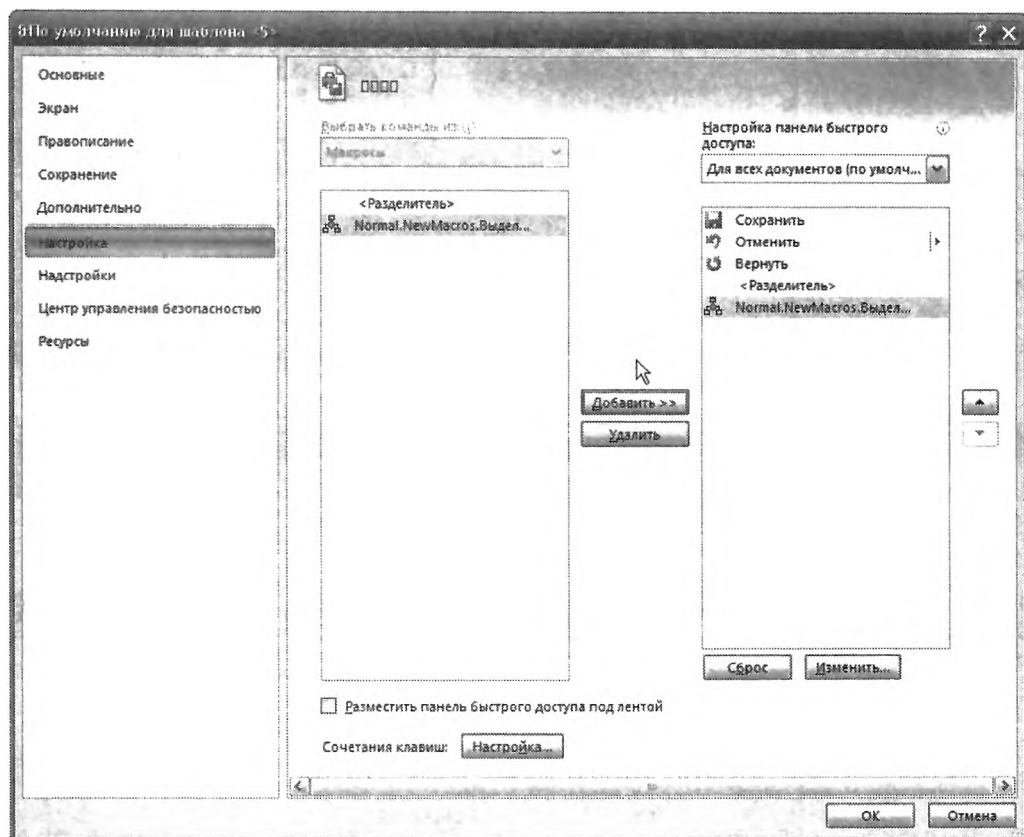
Нажмите сочетание клавиш, которое хотите использовать для запуска этого макроса; клавиши, которые вы нажимаете, появляются в текстовом окне **Новое сочетание клавиш**. Щелкните на кнопке **Назначить**, чтобы назначить сочетание клавиш новому макросу. Щелкните на кнопке **Заккрыть** для выхода из диалогового окна **Настройка клавиатуры** и переходите к записи макроса.

**Рис. 1.10**  
**Окно Настройка клавиатуры**  
 в Word для макроса **ВыделитьТекст**



**Рис. 1.11.** Окно По умолчанию для шаблона в Word для макроса **ВыделитьТекст**

Для того чтобы поместить на панель быстрого доступа кнопку для запуска нового макроса, щелкните в окне **Запись макроса на кнопку**. В окне **По умолчанию для шаблона** (рис. 1.11) поместите наименование (полное) нового макроса в список **Настройка панели быстрого доступа**, используя список **Выбрать команды из** и кнопку **Добавить**, как показано на рис. 1.12 (на рис. 1.12 добавлен также элемент **<Разделитель>**).

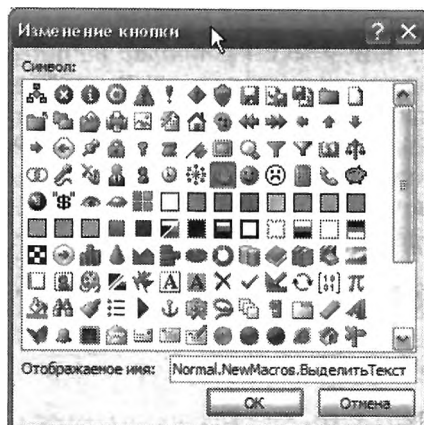


**Рис. 1.12.** Окно **По умолчанию для шаблона** в Word для макроса **ВыделитьТекст**

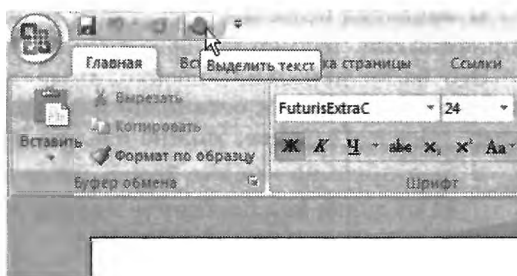
В указанных списках слева от наименования макроса располагается изображение кнопки такое же, какое будет на панели быстрого доступа. Его можно изменить в окне **Изменение кнопки** (рис. 1.13), которое открывается при помощи кнопки **Изменить**. В этом же окне можно задать текст подсказки, появляющейся при наведении курсора на кнопку панели, которая ее содержит (рис. 1.14).

И последнее, в окне **По умолчанию для шаблона** имеется кнопка для вызова окна **Настройка клавиатуры** — **Настройка**. Это окно (рис. 1.15) отличается от того, которое приведено на рис. 1.10, тем, что позволяет задать «горячие клавиши» не только записываемому макросу, но и другим элементам управления приложения (как, впрочем, и окно **По умолчанию для шаблона**).

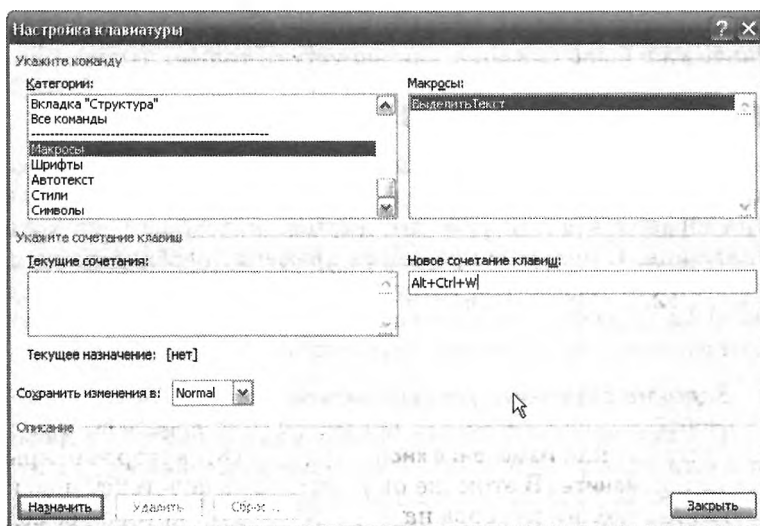
**Рис. 1.13**  
Окно **Изменение кнопки**



**Рис. 1.14**  
Кнопка макроса **ВыделитьТекст** на панели быстрого доступа



**Рис. 1.15**  
Окно **Настройка клавиатуры** в Word



Как только вы закрываете диалоговое окно **Настройка клавиатуры** или **По умолчанию для шаблона**, Word активизирует макрорекордер и начинает запись ваших действий.

### Запись ваших действий

Когда макрорекордер начинает записывать ваши действия, на экране появляется значок магнитофонной кассеты как часть указателя мыши, а в нижней части экрана (*строке состояния*) приложения кнопка **Запись макроса** меняет свой вид и подсказку на **Идет запись макроса**. Щелкните на ней, чтобы остановить запись (рис. 1.16). Макрорекордер сохраняет каждое действие, которое вы выполняете, в новом макросе до тех пор, пока вы не остановите рекордер или не сделаете паузу.



Рис. 1.16. Кнопка Запись макроса

В качестве примера выполните следующие: на вкладке **Главная** выберите шрифт **Arial**, размер шрифта **12** и стиль **Полужирный**.

Это были действия, необходимые для записи этого отдельного макроса, и теперь вы готовы к остановке макрорекордера, что, как и многое в приложениях Microsoft Office, можно сделать разными способами:

- ☐ щелкните на кнопке **Запись макроса** в строке состояния;
- ☐ на вкладке **Вид** щелкните кнопку **Макросы** и в появившемся меню выберите команду **Остановить запись**;
- ☐ щелкните кнопку **Остановить запись** на вкладке **Разработчик**.

Ваш новый Word-макрос является теперь законченным и готовым к запуску. Далее в этой главе вы узнаете, как запускать этот новый макрос, хотя немного ранее уже было сказано, как запустить только что созданный макрос.

### Запись макроса в Excel

Для этого примера предположим, что вам также часто приходится применять полужирный шрифт Arial 12-го размера в качестве стиля форматирования символов ячеек рабочих листов, к которым вы хотите привлечь особое внимание. С целью сокращения времени, необходимого для форматирования текста, вам нужно записать макрос, который выбирает полужирный шрифт Arial 12-го размера и применяет это форматирование к любой ячейке или диапазону ячеек выделенного фрагмента.

#### Задание стартовых условий в Excel

Так как вам необходимо, чтобы макрос работал с любой выделенной ячейкой или диапазоном ячеек, стартовыми условиями для этого макроса являются открытая рабочая книга с выделенным диапазоном ячеек в активном рабочем листе. Чтобы задать стартовые условия для данного конкретного примера в Excel, выполните следующие шаги:

1. Запустите Excel 2007, если он еще не запущен.
2. Откройте какую-либо рабочую книгу.
3. Выберите какой-либо рабочий лист.
4. Выделите любую ячейку в рабочем листе.

Вы только что создали необходимые стартовые условия, чтобы записать общий макрос для форматирования символов любой выделенной ячейки или диапазона ячеек в рабочем листе Excel.

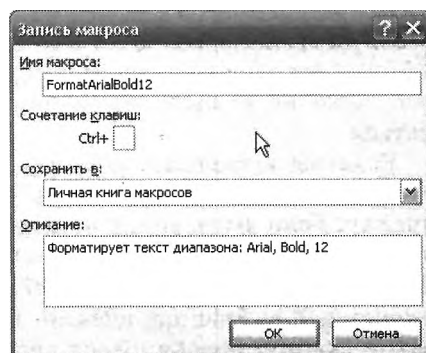
### Назначение имени и выбор места для сохранения макроса

Для запуска макрорекордера в Excel, назначения имени макросу, выбора места для сохранения нового макроса и выбора дополнительных опций выполните следующие шаги:

1. На вкладке **Вид** щелкните кнопку **Макросы** и в появившемся меню выберите команду **Запись макроса** или щелкните одноименную кнопку на вкладке **Разработчик** (или, наконец, просто щелкните кнопку **Запись макроса** в строке состояния в нижней части окна приложения). В Excel отображается диалоговое окно **Запись макроса**.
2. В текстовом окне **Имя макроса (Macro Name)** введите **FormatArialBold12** в качестве имени макроса. Это имя помогает запомнить, что выполняет макрос.
3. В текстовом окне **Описание (Description)** введите следующий текст: «форматирует текст диапазона: Arial, Bold, 12». Этот комментарий поможет вам не забыть назначение данного макроса.
4. Используйте список **Сохранить в (Store macro in)** для выбора места, в котором будет сохранен записанный макрос. Доступными вариантами являются **Личная книга макросов (Personal Macro Workbook)**, **Новая книга (New Workbook)** и **Эта книга (This Workbook)**. Поскольку вам необходимо, чтобы этот макрос был доступен во всех ваших рабочих книгах, выберите **Личная книга макросов**. Выбор **Эта книга** приведет к тому, что Excel сохранит новый макрос в текущей активной рабочей книге. Выбор опции **Новая книга** приведет к созданию в Excel новой рабочей книги, в которой будет сохранен этот макрос, — рабочая книга, которая была активной при запуске вами макрорекордера, остается активной рабочей книгой; любые действия, которые вы записываете, выполняются в этой книге, а не в новой рабочей книге, созданной для сохранения макроса.
5. Если вы уверены в том, что будете часто использовать макрос, который собираетесь записывать, можете назначить для его запуска горячую клавишу. Если — да, введите горячую клавишу в текстовое окно **Сочетание клавиш (Shortcut Key)** окна **Запись макроса**.
6. Щелкните на кнопке **ОК** для начала записи макроса.

Рис. 1.17

Заполненное диалоговое окно **Запись макроса** для макроса-примера **FormatArialBold12**



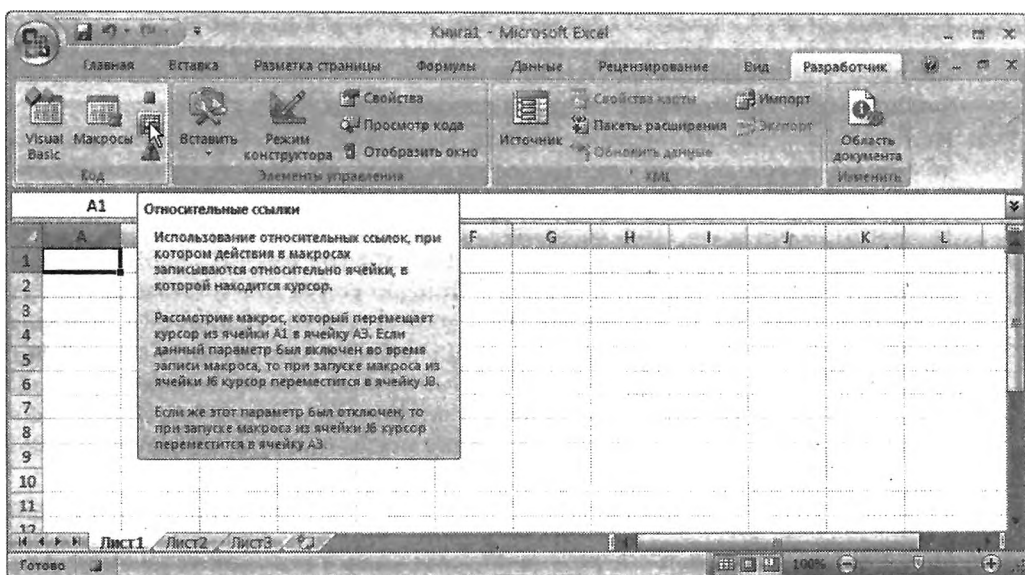
На рис. 1.17 показано диалоговое окно **Запись макроса**, заполненное для макроса-примера **FormatArialBold12**.

## Запись действий

Как только вы щелкните на кнопке **OK** в диалоговом окне **Запись макроса**, Excel запустит макрорекордер и начнет запись ваших действий. Макрорекордер сохранит каждое ваше действие в новом макросе.

Заметьте, что в строке состояния приложения кнопка **Запись макроса** меняет свой вид и подсказку на **Идет запись макроса**. Щелкните, чтобы остановить запись, как напоминание о том, что Excel записывает ваши действия.

При щелчке на кнопке **Макросы** на вкладке **Вид** открывается меню, в котором, кроме опций **Макросы** (о которой будет рассказано далее) и **Запись макроса**, имеется опция **Относительные ссылки (Relative Reference)**. На вкладке **Разработчик** имеется кнопка с таким же именем (рис. 1.18).



**Рис. 1.18.** На вкладке **Разработчик** имеется кнопка **Относительные ссылки**

По умолчанию Excel записывает абсолютные ссылки на ячейки в записываемые вами макросы. Если вы, например, начнете запись с выделенной ячейки **A7**, а затем выделите ячейку справа от **A7**, то есть **B7**, то ваш записанный макрос также будет выделять ячейку **B7**, и так каждый раз, когда он будет запускаться.

Если вы выполните команду **Относительные ссылки**, Excel будет записывать относительную ссылку на ячейку каждый раз, когда вы выделяете какую-либо ячейку. Если выделенной в данный момент является ячейка **A7** и вы выбираете ячейку справа от нее во время записи с относительными ссылками, то Excel записывает, что вы выделили ячейку, находящуюся на 1 столбец и 0 строк правее от текущей выбранной ячейки. Когда вы запустите записанный макрос, он выделит ячейку, находящуюся непосредственно справа от активной ячейки.

Кнопка **Относительные ссылки** на вкладке **Разработчик** является кнопкой-переключателем (toggle). Когда запись с относительными ссылками отключена, кнопка **Относительные ссылки** выглядит плоской; при помещении курсора мыши на кнопку вид кнопки изменяется и она выглядит отжатой. Когда запись с относительной ссылкой включена, кнопка **Относительные ссылки** нажата (находится в «утопленном» положении). Щелкая на кнопке **Относительные ссылки**, можно включать и выключать запись с относительными ссылками во время записи.

Опция **Относительные ссылки** меню кнопки **Макросы** вкладки **Вид** также отображает, включена ли функция записи с относительными ссылками, поскольку слева от наименования опции меню отображается значок, похожий на кнопку **Относительные ссылки** вкладки **Разработчик**.

Для записи макроса **FormatArialBold12** на вкладке **Главная** выберите шрифт **Arial**, размер шрифта **12** и стиль **Полужирный**.

Как и при записи макроса в Word, остановить запись можно следующими способами:

- ☐ щелкните на кнопке **Запись макроса** в строке состояния;
- ☐ на вкладке **Вид** щелкните кнопку **Макросы** и в появившемся меню выберите команду **Остановить запись**;
- ☐ щелкните кнопку **Остановить запись** на вкладке **Разработчик**.

## Код макроса

При записи макроса в Word, Excel (и других приложений) рекордер сохраняет последовательность текстовых инструкций, которые описывают на языке программирования Visual Basic for Applications различные действия, выполняемые вами в то время, когда рекордер включен. Это текстовое описание ваших команд называется *исходным кодом* (source code) для этого макроса. Позже, когда вы запускаете макрос, VBA считывает записанные в исходном коде инструкции и выполняет каждую последовательно, повторяя таким образом все действия, которые вы выполняли, когда записывали макрос.

В листинге 1.2 приведен исходный код, полученный при записи вами макроса **ВыделитьТекст** в Word 2007. Обратите внимание, имя и описание макроса, которые вы ввели в диалоговое окно **Запись макроса**, включены в начало исходного кода записанного макроса.

### Листинг 1.2. Макрос, записанный в Word

---

```
1: Sub ВыделитьТекст()  
2: '  
3: ' ВыделитьТекст Макрос  
4: ' Устанавливает Arial, Bold, 12 для выделенного фрагмента  
5: '  
6:     Selection.Font.Name = "Arial"  
7:     Selection.Font.Size = 12  
8:     Selection.Font.Bold = wdToggle  
9: End Sub
```

---



Подобный код производится и макрорекордером Excel. Код в листинге 1.3 представляет собой исходный код, полученный при записи макроса **FormatArialBold12** в Excel. Заметьте, что имя и описание макроса, которые вы ввели в диалоговое окно **Запись макроса**, также включены в начало исходного кода записанного макроса.

---

**Листинг 1.3.** Макрос, записанный в Excel

---

```
1: Sub FormatArialBold12()  
2: '  
3: ' FormatArialBold12 Макрос  
4: ' Форматирует текст диапазона: Arial, Bold, 12  
5: '  
6: '  
7: '  
8:     With Selection.Font  
9:         .Name = "Arial"  
10:        .Size = 11  
11:        .Strikethrough = False  
12:        .Superscript = False  
13:        .Subscript = False  
14:        .OutlineFont = False  
15:        .Shadow = False  
16:        .Underline = xlUnderlineStyleNone  
17:        .ThemeColor = xlThemeColorLight1  
18:        .TintAndShade = 0  
19:        .ThemeFont = xlThemeFontNone  
20:    End With  
21:    Selection.Font.Bold = True  
22:    With Selection.Font  
23:        .Name = "Arial"  
24:        .Size = 12  
25:        .Strikethrough = False  
26:        .Superscript = False  
27:        .Subscript = False  
28:        .OutlineFont = False  
29:        .Shadow = False  
30:        .Underline = xlUnderlineStyleNone  
31:        .ThemeColor = xlThemeColorLight1  
32:        .TintAndShade = 0  
33:        .ThemeFont = xlThemeFontNone  
34:    End With  
35:End Sub
```

---

На данном этапе работы с книгой еще рано изучать приведенные листинги. Вы можете записывать и выполнять макросы, даже не заглядывая в исходный код или не зная, что данный макрос выполняет. В следующей главе описываются разные составляющие макроса и объясняется, как находить исходный код для определенного макроса и выводить его на экран для редактирования и проверки.

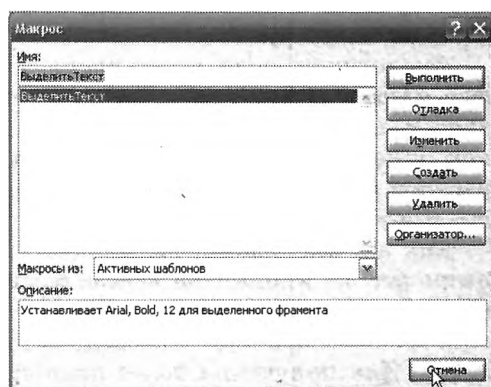
## Выполнение макросов

После того как вы записали рекордером макрос, этот макрос можно выполнить. При выполнении макроса Word, Excel (или другое приложение) следуют всем инструкциям, записанным в макросе.

И в Word, и в Excel макрос запускается выбором кнопки **Макросы** на вкладке **Вид** и опции **Макросы** в выпадающем меню или — кнопки **Макросы** на вкладке **Разработчик**. И в том, и в другом случае на экране отображается диалоговое окно **Макросы** (рис. 1.19) со списком макросов, из которого следует выбрать необходимый макрос и щелкнуть кнопку **Выполнить (Run)**.

**Рис. 1.19**

Выберите макрос **ВыделитьТекст** в списке **Имя** а щелкните кнопку **Выполнить** для запуска макроса



Например, для выполнения макроса **ВыделитьТекст**, который вы записали ранее в этой главе, сначала выделите некоторый текст в документе. Затем щелкните кнопку **Макросы** на вкладке **Вид** и выберите опцию **Макросы** в выпадающем меню. Выберите макрос **ВыделитьТекст** в списке **Имя** (**Macro Name**) и затем щелкните кнопку **Выполнить**, чтобы выполнить этот макрос. Выделенный перед выполнением макроса текст в вашем документе будет отформатирован с использованием полужирного шрифта Arial 12-го размера.

Для выполнения записанного вами в Excel макроса **FormatArialBold12** выберите сначала ячейку в рабочем листе (предпочтительнее ячейку, содержащую некоторый текст, чтобы вы могли видеть изменения). Затем щелкните кнопку **Макросы** на вкладке **Вид** и выберите опцию **Макросы** в выпадающем меню для отображения диалогового окна **Макросы** и затем выберите макрос **PERSONAL.XLSB!FormatArialBold12** в списке **Имя** (**Macro Name**). Наконец, щелкните на кнопке **Выполнить** для запуска макроса **FormatArialBold12**. Текст в любой ячейке, которая была выделена до запуска вами этого макроса, будет теперь иметь формат полужирного шрифта Arial 12-го размера.

В Word диалоговое окно **Макросы** содержит имена макросов, сохраненных в текущем документе или в любых *общих шаблонах* (см. рис. 1.19). Используйте список **Макросы из:** для выбора определенного документа или шаблона, если затрудняетесь найти нужный вам макрос в полном списке доступных макросов.

Диалоговое окно **Макросы** в Excel содержит имена макросов, сохраненных в любых рабочих книгах, открытых в данный момент. Имя рабочей книги, содержащей макрос, помещается перед именем макроса в списке **Имя**, если макрос не находится в текущей рабочей книге (как в случае с макросом PERSONAL.XLSB!FormatArialBold12). Если необходимый вам макрос не находится в списке, откройте рабочую книгу, в которой был сохранен этот макрос, чтобы сделать макрос доступным перед тем, как вы откроете диалоговое окно **Макросы**.

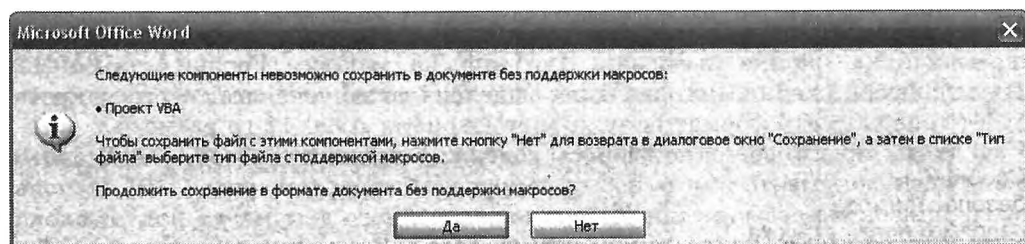
Если вы использовали опции диалогового окна **Запись макроса (Record Macro)** в Word для назначения вашему макросу горячей клавиши или кнопки на **панели быстрого доступа**, вы можете также запускать этот макрос, щелкая на соответствующей кнопке **панели быстрого доступа** или нажимая соответствующее сочетание клавиш.

Аналогично, если вы назначили горячую клавишу в диалоговом окне **Запись макроса** в Excel, вы можете запускать этот макрос, нажимая сочетание назначенных клавиш. Как и в случае с макросами, перечисленными в диалоговом окне **Макросы** в Excel, только макросы, сохраненные в текущей открытой рабочей книге (это не должна быть обязательно активная рабочая книга), могут иметь активизированные горячие клавиши.

Как вы уже знаете, вы можете также запускать макросы в Word и Excel, назначая им кнопки на **панели быстрого доступа** или создавая для их вызова пункты меню. Вы можете также назначить макросу кнопку или графический объект, помещенный непосредственно в рабочий лист Excel или в документ Word. Для получения более полной информации об использовании интерактивных возможностей для назначения макросов кнопкам панели и графическим объектам в документах и рабочих листах обращайтесь к разделам справочной системы Excel и Word.

## Сохранение документа с записанным макросом

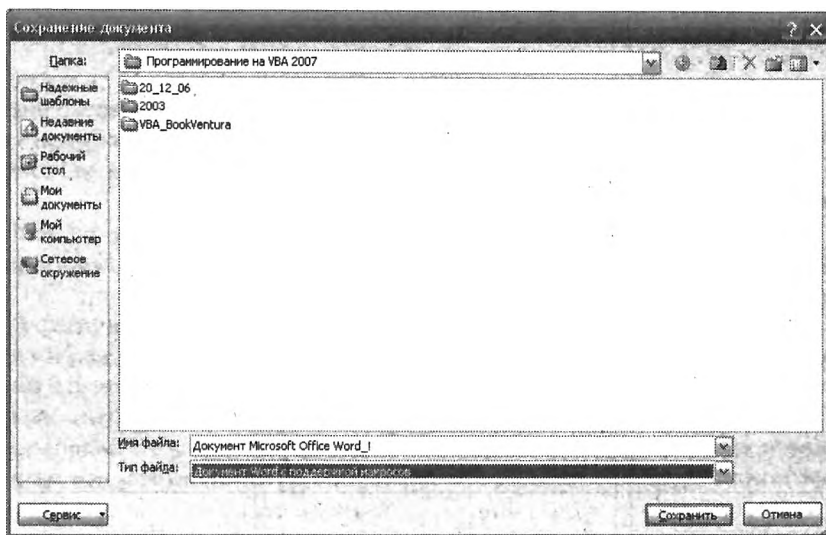
После того как вы записали первый макрос в Word (или Excel), рано или поздно вам придется закрыть соответствующий документ (или книгу). При этом вы можете к своему удивлению увидеть на экране необычное окно с сообщением о том, что ваш документ содержит, кроме обычных данных, макрос — проект VBA, — и не может быть сохранен как документ без поддержки макросов (рис. 20).



**Рис. 1.20.** Окно с сообщением о том, что документ содержит, кроме обычных данных, макрос — проект VBA, — и не может быть сохранен как документ без поддержки макросов

Если записанный макрос вам более не нужен, можете продолжить сохранение, щелкнув кнопку **Да**. В противном случае щелкните кнопку **Нет** и в окне **Сохранение документа** выберите для типа документа **Документ Word с поддержкой макросов** (рис. 1.21). Для Excel в одноименном окне вы должны будете выбрать **Книга Excel с поддержкой макросов**.

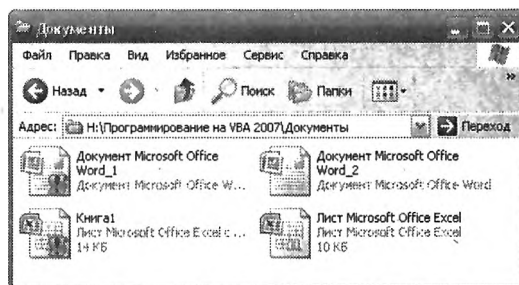
**Рис. 1.21**  
Окно  
**Сохранение**  
**документа**  
в Excel



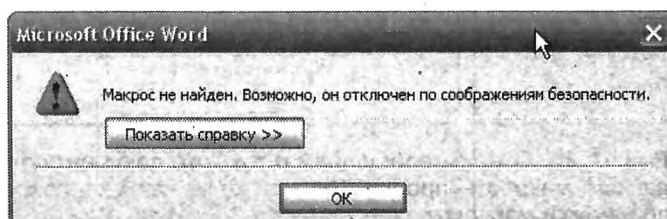
Документы и книги, сохраненные с поддержкой макросов, отличаются от обычных представлением их в **Проводнике**. На рис. 1.22 они расположены в левом столбце.

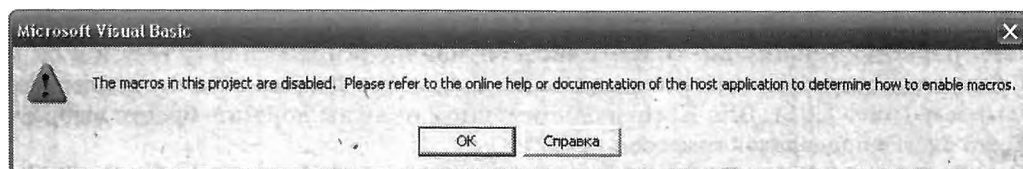
В заключение отметим, что при попытке запустить записанный в предыдущем сеансе текущего документа макрос вы можете получить одно из сообщений от системы безопасности о том, что макрос запустить невозможно (рис. 23, 24).

**Рис. 1.22**  
Документы и книги, сохраненные с поддержкой макросов, отличаются от обычных представлением их в **Проводнике**



**Рис. 1.23**  
Сообщение системы безопасности о невозможности запуска макроса





**Рис. 1.24.** Сообщение системы безопасности о невозможности запуска макроса

Сообщение, приведенное на рис. 23, следует понимать именно так, как оно написано: либо макрос не найден, потому что не принадлежит данной книге, либо действительно для Word отключена возможность выполнения макросов из соображений безопасности.

Если макрос был записан не в текущей книге и не в файле с именем **Normal.dotm**, он не может быть запущен из текущей книги (даже, если книга, в которой он записан, также открыта).

Если для Word отключена возможность выполнения макросов из соображений безопасности, никакой макрос (даже, сохраненный в **Normal.dotm**) не может быть запущен, о чем и сообщает окно, приведенное на рис. 1.24

Чтобы включить возможность выполнения макросов, необходимо в окне **Центр управления безопасностью** (открывается кнопкой **Безопасность макросов** на вкладке **Разработчик**) выбрать переключатель **Включить все макросы**.

# Редактирование макросов

В первой главе вы узнали, как записывать и выполнять макросы в Word и Excel. Перед редактированием и написанием макросов вам следует узнать больше о том, где сохраняются макросы. Необходимо также иметь общее представление о компоненте **VBA Editor**, его командах меню и кнопках панелей инструментов.

## Модули

Вы уже знаете (из первой главы), что макросы Visual Basic for Applications сохраняются как часть файлов, в которых Word, Excel (и, например, Access) обычно содержат свои данные, — макросы сохраняются в файлах документов в Word и в файлах книг в Excel. Макросы сохраняются в специальной части файла данных, называемой *modules (модули)*. Модуль VBA содержит исходный код макроса — текстовое представление инструкций. Каждый документ Word (или шаблон документа) может содержать один или несколько модулей или не содержать никакого модуля; аналогично, каждый файл книги Excel может не содержать модулей или содержать один или несколько модулей. Модули, сохраняемые в одном документе Word или рабочей книге Excel, имеют общее название *Project (проект)*.

Знание того, как Word и Excel именуют создаваемые в них модули, поможет вам находить ваши записанные рекордером макросы, когда вам потребуется просмотреть или редактировать их. В следующих нескольких параграфах описывается, как Word и Excel выбирают имена модулей при записи макроса рекордером.

## Модули в Word

Как и в Word, при записи макроса в Excel вы можете определять только книгу, в которой Excel сохраняет записанный макрос, — текущую книгу, новую книгу или книгу **Personal.xlsb**. Excel выбирает модуль, в котором сохраняется записанный макрос, и при необходимости создает этот модуль. Правила выбора модуля, в котором сохраняется ваш записанный макрос, в Excel являются более сложными, чем в Word.

Когда Excel создает модуль, в котором сохраняется записанный макрос, модулю присваивается имя *ModuleN*, где *N* — это количество модулей, созданных для определенной рабочей книги во время текущего сеанса работы. Например, в первый раз, когда вы сохраняете записанный макрос в **Personal.xlsb** (личной книге макросов), Excel создает модуль с именем *Module1*. Если вы продолжаете записывать макросы в том же сеансе работы и сохранять их в

**Personal.xlsb**, Excel продолжает сохранять записанные макросы в том же модуле (*Module1*) до тех пор, пока вы не выберете другую рабочую книгу. Если позже в том же сеансе работы вы опять захотите сохранить записанные макросы в **Personal.xlsb**, Excel добавляет другой модуль с именем *Module2* в эту книгу.

Если какая-либо рабочая книга уже содержит модуль с тем же именем, что выбран в Excel для нового модуля, Excel увеличивает число в имени модуля до тех пор, пока имя нового модуля не будет отличаться от имен существующих модулей. Например, если вы начнете новый сеанс работы в Excel и затем захотите сохранить записанный рекордером макрос в книге **Personal.xlsb**, Excel сначала выбирает имя *Module1* для нового модуля. Если же книга **Personal.xlsb** уже содержит модули с именами *Module1* и *Module2*, Excel увеличивает число в имени модуля, чтобы оно не совпадало с существующими именами; новый модуль получает имя *Module3*.

## Редактор Visual Basic

Для просмотра модулей, сохраненных в определенном документе, шаблоне или рабочей книге (и исходного кода макроса, который они содержат), вам необходимо использовать компонент Visual Basic Editor (Редактор Visual Basic). Этот компонент предоставляет инструментальные средства, которые используются для создания новых модулей, просмотра содержимого существующих модулей, создания и редактирования исходного кода макроса, создания пользовательских диалоговых окон и выполнения других задач, относящихся к написанию и обслуживанию программ на VBA. Как вы уже могли заметить, в этой книге компонент Редактор Visual Basic называется просто «Редактор VB».

Редактор VB содержит одни и те же возможности в Word, Excel (и, например, Access). Когда вы используете Редактор VB в Word, Excel, вы, фактически, используете тот же Редактор VB, который имеется в вашем распоряжении в Excel. В последующих нескольких параграфах рассказывается о том, как запускать Редактор VB, объясняется назначение различных окон, отображаемых компонентом Редактор VB, и описывается меню Редактора VB и команды панели инструментов.

### Запуск Редактора VB

Как указывалось в предыдущих разделах, чтобы просмотреть модули или исходный код VBA, который они содержат, вам необходимо сначала запустить Редактор VB. Независимо от того, работаете ли вы в Word или в Excel (а также и Access), Редактор VB запускается одним и тем же способом. Используйте для этого один из следующих приемов:

- ☐ Выберите на вкладке **Разработчик** кнопку **Visual Basic**.
- ☐ Нажмите сочетание клавиш **Alt+F11**.

Каким бы приемом вы не воспользовались, Word (Excel) запустит Редактор Visual Basic. На рис. 2.1 показано типичное окно Редактора Visual Basic, отображенное в Word. Окно Редактора VB, отображаемое в Excel (рис. 2.2), в основном подобно окну Редактора VB, показанному на рис. 2.1.

## Окна Редактора VB

В окне Редактора VB имеются три *дочерних* окна, каждое из которых отображает важную информацию о *VBA-проекте*. [*Project (Проект)* — это группа модулей и других объектов, сохраняемых в определенном документе, шаблоне документа, рабочей книге или шаблоне рабочей книги.] Каждое из окон Редактора VB отображается по умолчанию в *прикрепленных (docked)* положениях, показанных на рис. 2.1.

Если необходимо, вы можете переместить любое из дочерних окон Редактора VB в любое место на экране, *строку заголовка (title bar)* этого окна таким же образом, каким бы вы перемещали любое окно на рабочем столе Windows. Перетаскивание одного из дочерних окон из его прикрепленного положения приводит к тому, что оно становится плавающим окном. *Плавающие (floating)* окна всегда остаются видимыми поверх других окон. Вы можете также изменять размер любого из дочерних окон Редактора VB, расширяя или уменьшая рамку окна для увеличения или уменьшения его размера, что подобно изменению размера любого окна на рабочем столе Windows.

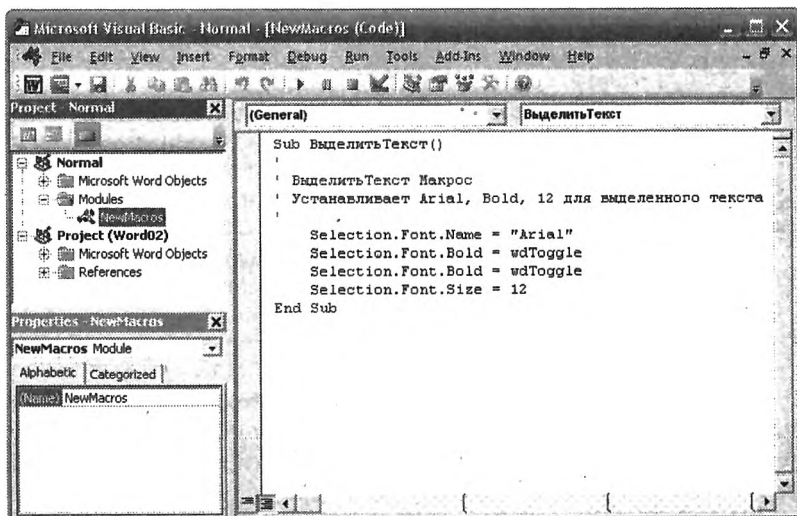
Далее описываются три окна Редактора VB и их назначение.

**Project Explorer** содержит дерево-диаграмму открытых в данный момент файлов (документы, шаблоны или рабочие книги) и объектов, содержащихся в этих файлах (объекты *host-приложения*, модули, ссылки, формы и так далее). **Project Explorer** можно использовать для перехода к различным модулям и другим объектам в проекте VB при помощи кнопок (панели инструментов этого окна) **View Code** (программа), **View Object** (объект) и **Toggle Folders** (папки).

**Properties Window** содержит все свойства объекта текущего выбора. В некоторых случаях, как в окне, показанном на рис. 2.1, свойства объекта состоят только из его имени. (Подробнее об объектах и свойствах объектов вы узнаете из следующих глав.) Вкладка **Alphabetic** (по алфавиту) этого окна предоставляет список свойств выделенного объекта, составленный из имен свойств в алфавитном порядке. Вкладка **Categorized** (по категориям) отображает свойства объекта, отсортированные по категориям.

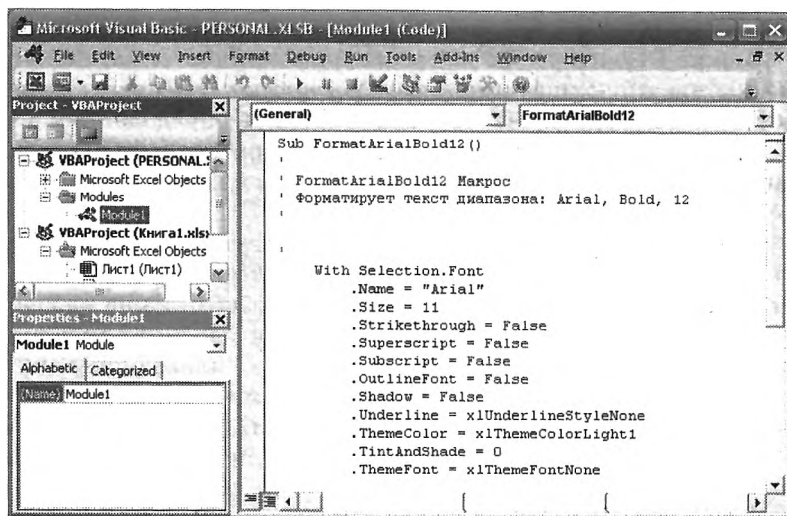
Рис. 2.1

Типичное окно Редактора VB в Microsoft Word

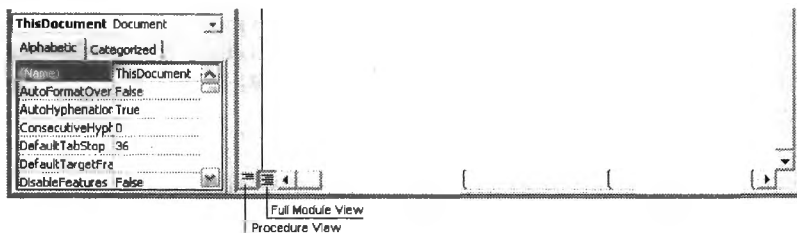




**Рис. 2.2**  
Типичное окно  
Редактора VB  
в Microsoft Excel



**Code Window** — это окно, в котором вы можете просматривать, редактировать или создавать исходный код VBA. В режиме **Full Module View** весь исходный код макроса в модуле отображается в прокручиваемом текстовом окне, а макрос отделяется от другого макроса серой линией. Редактор VB позволяет также просматривать содержимое модуля в режиме **Procedure View** (представление процедуры). Чтобы выбрать режим просмотра, щелкните кнопки в нижнем левом углу **Code Window** (см. рис. 2.3).



**Рис. 2.3.** Для выбора режима просмотра, щелкните кнопки в нижнем левом углу **Code Window**

### Замечание

Если вы одновременно используете несколько документов (проектов), содержащих макросы с одинаковыми именами, вам следует внимательно работать с окном **Project Explorer**, чтобы редактировать именно те макросы, которые необходимо.

Когда **Code Window** находится в режиме **Procedure View**, видимым является исходный код только одного макроса. Используйте раскрывающийся список **Procedure** (процедура) для просмотра другого макроса. В режиме **Full Module View** вы можете также использовать раскрывающийся список **Procedure** для быстрого перехода к отдельному макросу.

Используйте список **Object List** (объект) для выбора объекта, процедуры которого хотите просмотреть или редактировать. В случае стандартных модулей, таких как модули, в которых сохраняются записанные вами макросы, единственным выбором в списке **Object List** является **General** (общая область).

Теперь, ознакомившись с окнами Редактора VB, вы можете перейти к изучению панелей инструментов и меню Редактора VB, которые содержат команды, помогающие управлять окнами **Project Explorer**, **Properties Window**, **Code Window** и их содержанием.

### Меню Редактора VB

В этом и последующих разделах рассказывается о меню и панелях инструментов Редактора VB. Сейчас вам необходимо лишь ознакомиться с имеющимися командами. Можете попробовать использовать некоторые из этих команд, только убедитесь сначала, что вы экспериментируете с документом (или рабочей книгой), не содержащим незаменимых данных.

### Замечание

Далее в тексте значки <, > с находящимся внутри них текстом означают, что в зависимости от ситуации на это место подставляется имя файла, модуля, макроса, проекта и т.д.

### Меню File (Файл)

Меню **File**, как и во всех приложениях Windows, содержит команды, относящиеся к сохранению и открытию файлов. В Редакторе VB меню **File** предоставляет команды, необходимые для сохранения изменений в проекте VBA и вывода на экран исходного кода вашего макроса VBA. В табл. 2.1 приведены команды меню **File**, их горячие клавиши и назначение каждой команды.

Таблица 2.1. Команды меню **File**

Команда	«Горячая» клавиша	Действие
<b>Save</b> <project> (сохранить <проект>)	Ctrl+S	Сохраняет текущий проект (презентацию, рисунок и т.д. в зависимости от приложения, в котором открыт Редактор VB) VBA на диске, включая все модули и формы.
<b>Import File</b> (импорт файла)	Ctrl+M	Добавляет существующий модуль или класс в текущий проект. Вы можете импортировать только модули, формы или классы, сохраненные ранее командой <b>Export File</b> из другого проекта.
<b>Export File</b> (экспорт файла)	Ctrl+E	Сохраняет текущий модуль, форму или класс в формате текстового файла для импортирования в другой проект или в целях архивирования.

Команда	«Горячая» клавиша	Действие
<b>Remove</b> <item> (удалить <...>)		Перманентно удаляет модуль или форму текущего выбора из проекта VBA. Эта команда не доступна, если в <b>Project Explorer</b> не выбран никакой элемент.
<b>Print</b> (печать)	Ctrl+P	Печатает модуль или форму для документирования или с целью архивирования.
<b>Close and Return to ...</b> (закрыть и вернуться в ...)	Alt+Q	Закрывает Редактор VB и возвращает вас в окно host-приложения, из которого был открыт Редактор VB.

### Меню Edit (правка)

Меню **Edit** содержит команды, относящиеся к управлению исходным кодом макроса в Code Window и объектами в формах. В табл. 2.2 приведены имеющиеся команды меню **Edit**, их горячие клавиши и описывается действие, выполняемое каждой командой.

**Таблица. 2.2. Команды меню Edit**

Команда	«Горячая» клавиша	Действие
<b>Undo</b> (отменить)	Ctrl+Z	Отменяет самую последнюю команду. Не все команды могут быть отменены. Меню доступно только в случае, если есть, что отменять.
<b>Redo</b> (вернуть)		Возвращает самую последнюю команду, которую вы отменили.
<b>Cut</b> (вырезать)	Ctrl+X	Вырезает выделенный текст или объект и помещает его в Windows Clipboard. Выделенный текст или объект удаляется из модуля или формы.
<b>Copy</b> (копировать)	Ctrl+C	Копирует выделенный текст или объект и помещает его в Windows Clipboard. Выделенный текст или объект остается неизменным.
<b>Paste</b> (вставить)	Ctrl+V	Вставляет текст или объект из Windows Clipboard в текущий модуль или форму.
<b>Clear</b> (очистить)	Del	Удаляет выделенный текст или объект из модуля или формы.
<b>Select All</b>	Ctrl+A	Выделяет весь текст в модуле или все объекты в форме.
<b>Find</b> (найти)	Ctrl+F	Позволяет находить указанный текст в модуле.

Команда	«Горячая» клавиша	Действие
<b>Find Next</b>	F3	Повторяет последнюю операцию <b>Find</b> .
<b>Replace</b> (заменить)	Ctrl+H	Позволяет находить указанный текст в модуле и заменять его другим текстом.
<b>Indent</b>	Tab	Смещает весь выделенный текст вправо на интервал табуляции.
<b>Outdent</b>	Shift+Tab	Смещает весь выделенный текст влево на интервал табуляции.
<b>List Properties/Methods</b> (список свойств/методов)	Ctrl+J	Открывает список в <b>List Properties/ Methods</b> , отображая свойства и методы объекта, имя которого вы только что ввели. Когда курсор вставки находится на пустом месте в <b>List Properties/ Methods</b> эта команда открывает список глобально доступных свойств и методов.
<b>List Constants</b>	Ctrl+Shift+J	Открывает список в <b>Code Window</b> , отображающий допустимые константы для свойства, которое вы только что ввели с предшествующим знаком «=».
<b>Quick Info</b> (сведения)	Ctrl+I	Открывает всплывающее окно подсказки, отображающее правильный синтаксис для процедуры, функции или метода, который вы только что ввели в <b>Code Window</b> .
<b>Parameter Info</b> (параметры)	Ctrl+Shift+I	Открывает всплывающее окно подсказки, отображающее параметры (называемые также аргументами) процедуры, функции или оператора, который вы только что ввели в <b>Code Window</b> .
<b>Complete Word</b> (завершить слово)	Ctrl+Space	Редактор VB заканчивает слово, которое вы вводите, как только вы введете достаточно символов для того, чтобы VBA распознал ключевое слово.
<b>Bookmarks</b> (закладки)		Открывает подменю с пунктами для помещения, удаления или перехода к закладкам, которые вы ранее поместили в ваш модуль. Закладки Редактора VB не имеют имен.

### Меню View (Вид)

Меню **View** содержит команды, позволяющие выбирать элементы Редактора VB для просмотра и способ просмотра. В табл. 2.3 приведены команды меню **View**, их горячие клавиши и действие, производимое каждой командой.

Таблица 2.3. Команды меню View

Команда	«Горячая» клавиша	Действие
<b>Code</b> (программа)	F7	Активизирует <b>Code Window</b> для отображения исходного кода VBA, ассоциированного с выбранным модулем или формой.
<b>Object</b> (объект)	Shift+F7	Отображает объект текущего выбора в <b>Project Explorer</b> .
<b>Definition</b> (описание)	Shift+F2	Отображает исходный код VBA для процедуры или функции, на которую указывает курсор; отображает <b>Object Browser</b> для объектов в справке VBA.
<b>Last Position</b> (вернуться к последней позиции)	Ctrl+Shift+F2	Переходит в последнюю позицию в модуле после использования команды меню <b>Definition</b> или после редактирования кода.
<b>Object Browser</b> (просмотр объектов)	F2	Открывает <b>Object Browser</b> , позволяющий определять, какие макросы доступны в данный момент.
<b>Immediate Window</b> (окно отладки)	Ctrl+G	Отображает окно VBA-отладчика <b>Immediate Window</b> .
<b>Locals Window</b> (окно локальных переменных)		Отображает окно отладчика <b>Locals Window</b> .
<b>Watch Window</b> (окно контрольного значения)		Отображает окно отладчика <b>Watch Window</b> (контрольные значения).
<b>Call Stack</b> (стек вызова)	Ctrl+L	Отображает список последовательности вызовов для текущей функции или процедуры VBA.
<b>Project Explorer</b> (окно проекта)	Ctrl+R	Отображает <b>Project Explorer</b> .
<b>Properties Window</b> (окно свойств)	F4	Отображает <b>Properties Window</b> .
<b>Toolbox</b> (панель элементов)		Отображает <b>Toolbox</b> . <b>Toolbox</b> используется для добавления элементов управления в пользовательские диалоговые окна.
<b>Toolbars</b> (панели инструментов)		Отображает подменю, позволяющее показывать или скрывать различные панели инструментов Редактора VB или открывать диалоговое окно для настройки одной из панелей инструментов Редактора VB.
<b>&lt;Host application&gt;</b> (host-приложение)	Alt+F11	Возвращает вас в host-приложение, из которого был запущен Редактор VB, но оставляет Редактор VB открытым. Конкретное имя этого пункта меню зависит от того, из какого host-приложения VBA вы запустили Редактор VB.

### Меню Insert (Вставка)

Команды меню **Insert** позволяют добавлять различные объекты, такие как модули и формы, в ваш проект. В меню **Insert** никакие команды не имеют «горячих клавиш». В табл. 2.4 приведены действия, выполняемые каждой командой этого меню.

**Таблица 2.4. Команды меню Insert**

Команда	Действие
<b>Procedure</b> (процедура)	Вставляет новую процедуру ( <b>Sub</b> , <b>Function</b> или <b>Property</b> ) в текущий модуль. ( <i>Процедура</i> — это еще одно название макроса).
<b>Module</b> (модуль)	Добавляет новый модуль в проект. Редактор VB дает этому модулю имя в соответствии с правилами, описанными ранее в этой главе.
<b>Class Module</b>	Добавляет в проект <i>class module</i> (модуль класса). Модули класса используются для создания пользовательских объектов в вашем проекте.
<b>File</b> (файл)	Позволяет вставлять текстовый файл, содержащий исходный код VBA, в модуль.

### Меню Format (Формат)

Команды меню **Format** используются при создании пользовательских диалоговых окон и других форм. Команды меню **Format** позволяют выравнивать объекты в форме по отношению друг к другу, настраивать размер элемента управления в соответствии с его содержимым и выполнять многие другие полезные задачи. Команды меню **Format** представлены здесь для полноты изложения материала, хотя вы не будете их применять до тех пор, пока не начнете создавать собственные пользовательские диалоговые окна. В табл. 2.5 приведены команды меню **Format** и их действия. Заметьте, что эти команды не имеют «горячих клавиш».

**Таблица 2.5. Команды меню Format**

Команда	Действие
<b>Align</b> (выровнять)	Открывает подменю команд, которые позволяют выравнивать выбранные объекты в форме по отношению друг к другу. Здесь можно выравнивать объекты по верхней/нижней, правой/левой границам, по центру или середине создаваемого объекта.
<b>Make Same Size</b> (выровнять размер)	Открывает подменю команд, позволяющих изменять размер выделенных объектов до размера указанного объекта.
<b>Size to Fit</b> (подогнать размер)	Одновременно изменяет ширину и высоту объекта до соответствия размеру его содержимого.

Команда	Действие
<b>Size to Grid</b> (выровнять размер по сетке)	Одновременно изменяет ширину и высоту объекта до ближайших меток сетки. При разработке форм Редактор VB отображает в форме сетку, чтобы было легче располагать и изменять размеры объектов в форме.
<b>Horizontal Spacing</b> (интервал по горизонтали)	Открывает подменю команд, позволяющих устанавливать горизонтальный интервал для выбранных объектов. Здесь можно устанавливать равномерный горизонтальный интервал, уменьшать или увеличивать его или удалять всякий горизонтальный интервал между объектами.
<b>Vertical Spacing</b> (интервал по вертикали)	Открывает подменю команд, позволяющих устанавливать вертикальный интервал для выбранных объектов. Здесь можно устанавливать равномерный вертикальный интервал, уменьшать или увеличивать его или удалять всякий вертикальный интервал между объектами.
<b>Center in Form</b> (разместить по центру в форме)	Открывает подменю команд, позволяющих изменять положение выбранных объектов, чтобы они были центрированы в форме горизонтально или вертикально.
<b>Arrange Buttons</b> (разместить кнопки)	Открывает подменю команд, позволяющих автоматически располагать командные кнопки в форме в ряд с равным интервалом по нижнему или правому краю формы.
<b>Group</b> (группировать)	Связывает несколько выбранных объектов вместе в одну группу, чтобы вы могли перемещать, изменять размер, вырезать или копировать объекты, обращаясь с ними, как с одним целым.
<b>Ungroup</b> (разделить)	Отменяет группировку объектов, которые перед этим были связаны вместе с помощью команды <b>Group</b> .
<b>Order</b> (порядок)	Открывает подменю команд, позволяющих изменять упорядочение сверху вниз (называемое <i>z-order</i> ) перекрывающихся объектов в форме. Используйте команду <b>Order</b> , чтобы обеспечить, например, появление текстового окна всегда поверх графического объекта в форме.

### Меню *Debug* (Отладка)

Команды меню *Debug* используются при выполнении тестирования или отладки макросов. (*Debugging* — так называется процесс нахождения и исправления ошибок в программе.) В табл. 2.6 приведены команды меню *Debug*, их «горячие клавиши» и выполняемые действия.

Таблица 2.6. Команды меню Debug

Команда	«Горячая» клавиша	Действие
<b>Compile &lt;project&gt;</b> (компилировать <проект>)		Компилирует проект, выбранный в данный момент в Project Explorer.
<b>Step Into</b> (шаг с заходом)	F8	Выполняет исходный код вашего макроса по одному оператору каждый раз.
<b>Step Over</b> (шаг с обходом)	Shift+F8	Подобно команде <b>Step Into</b> команда <b>Step Over</b> позволяет выполнять все инструкции в макросе без паузы на каждой отдельной инструкции.
<b>Step Out</b> (шаг с выходом)	Ctrl+Shift+F8	Выполняет все остающиеся операторы в макросе без паузы на каждом отдельном операторе.
<b>Run to Cursor</b> (выполнить до текущей позиции)	Ctrl+F8	Выполняет операторы исходного кода макроса от оператора, выполняющегося в данный момент, до текущей позиции курсора.
<b>Add Watch</b> (добавить контрольное значение)		Позволяет указывать переменные или выражения, значения которых можно наблюдать во время выполнения исходного кода VBA.
<b>Edit Watch</b> (изменить контрольное значение)	Ctrl+W	Позволяет редактировать спецификации для наблюдаемых переменных и выражений, которые были созданы ранее с помощью команды <b>Add Watch</b> .
<b>Quick Watch</b> (контрольное значение)	Shift+F9	Отображает текущее значение выбранного выражения.
<b>Toggle Breakpoint</b> (точка останова)	F9	Отмечает место (или отменяет отметку) в исходном коде VBA, где вы хотите остановить выполнение макроса.
<b>Clear All Breakpoints</b> (снять все точки останова)	Ctrl+Shift+F9	Удаляет все точки останова в модуле.
<b>Set Next Statement</b> (задать следующую инструкцию)	Ctrl+F9	Позволяет менять обычное выполнение кода путем указания вручную следующей строки исходного кода, которая должна выполняться.
<b>Show Next Statement</b> (показать следующую инструкцию)		Приводит к подсветке Редактором VB следующей строки кода, которая будет выполняться.



Команды меню **Debug** позволяют непосредственно контролировать выполнение макроса, останавливать и запускать макрос в заданных точках и отслеживать выполнение макроса по шагам.

### Меню Run (Запуск)

Команды меню **Run** позволяют начать выполнение макроса, прерывать или возобновлять его выполнение или возвращать прерванный макрос в состояние до выполнения.

Таблица 2.7. Команды меню Run

Команда	«Горячая» клавиша	Действие
<b>Run Sub/User Form</b> (запуск подпрограммы/ User Form)	F5	Приводит к тому, что VBA запускает макрос, который редактируется в данный момент, то есть, VBA запускает макрос, на тексте которого находится курсор вставки. Если какая-либо форма активна, VBA запускает эту форму.
<b>Break</b> (прервать)	Ctrl+Break	Прерывает выполнение вашего кода VBA и приводит к тому, что Редактор VB переходит в режим прерывания ( <b>Break mode</b> ). ( <b>Break mode</b> используется при отладке кода VBA.)
<b>Reset &lt;project&gt;</b> (сброс)		Устанавливает все переменные модульного уровня и <b>Call Stack</b> (список последовательности вызовов) в исходное состояние.
<b>Design Mode</b> (конструктор)		Включает и выключает <b>Design mode</b> (режим проектирования или разработки) для проекта. В этом режиме никакой код в вашем проекте не выполняется, и события от элементов управления не обрабатываются.

### Меню Tools (Сервис)

Команды меню **Tools** не только позволяют выбрать макрос для выполнения, но и получить доступ к внешним библиотекам макросов и дополнительным элементам управления форм (кроме встроенных в VBA). Команды меню **Tools** обеспечивают также доступ к диалоговому окну **Options** (параметры) Редактора VB и свойствам проекта VBA текущего выбора в **Project Explorer**. В табл. 2.8 приведены команды меню **Tools** и их действия. Команды меню **Tools** не имеют «горячих клавиш».

Таблица 2.8. Команды меню Tools

Команда	Действие
<b>References</b> (ссылки)	Отображает диалоговое окно <b>References</b> , позволяющее устанавливать ссылки на библиотеки объектов, библиотеки типов или другой проект VBA. После установления ссылки объекты, методы, свойства, процедуры и функции в этой ссылке появляются в диалоговом окне <b>Object Browser</b> .
<b>Additional Controls</b> (дополнительные элементы)	Отображает диалоговое окно <b>Additional Controls</b> , позволяющее настраивать <b>Toolbox</b> (панель элементов) так, чтобы вы могли добавлять элементы управления в формы помимо встроенных в VBA. Диалоговое окно <b>Additional Controls</b> предназначено для добавления к панели элементов кнопок, которые позволяют добавлять к форме объекты.
<b>Macros</b> (макросы)	Отображает диалоговое окно <b>Macros</b> , позволяющее создавать, редактировать, выполнять или удалять макросы.
<b>Options</b> (параметры)	Отображает диалоговое окно <b>Options</b> , позволяющее выбирать различные опции для Редактора VB, такие как число пробелов в интервале табуляции (tab stop), когда VBA проверяет синтаксис ваших операторов, и так далее.
<b>&lt;project&gt; Properties</b> (свойства проекта)	Отображает диалоговое окно <b>Project Properties</b> , позволяющее устанавливать различные свойства вашего проекта VBA, такие как имя проекта, описание и файл контекстной справки. Это диалоговое окно позволяет также защищать проект, чтобы никто не мог его редактировать без указания пароля.
<b>Digital Signature</b> (цифровая подпись)	Отображает диалоговое окно <b>Digital Signature</b> , в котором можно задать для проекта сертификат цифровой подписи.

### Меню Add-Ins

В этом меню находится всего одна команда, **Add-In Manager**, которая приводит к отображению диалогового окна **Add-In Manager**. Это окно позволяет регистрировать, загружать или выгружать и определять поведение дополнений.

### Другие меню

В Редакторе VB имеются два дополнительных меню: **Window** (окно) и **Help** (помощь). Оба этих меню содержат команды, идентичные меню **Window** и **Help**, имеющимся в других приложениях Microsoft Windows. Команды в меню **Window** позволяют выбирать активное окно, разбивать текущее окно, размещать дочерние окна вертикально и горизонтально, организовывать дочерние окна VB в виде каскада или выравнивать значки минимизированных дочерних окон.

Команды меню **Help** также идентичны командам меню **Help** в Word, Excel и других приложениях Microsoft Windows. Меню **Help** Редактора VB позволяет получать контекстно-зависимую подсказку посредством справочной системы Microsoft Office и просматривать файлы справочной системы VBA для

host-приложения, из которого вы запустили Редактор VB. Если у вас имеется модем или доступ к Internet, вы можете использовать Help | MSDN on the Web для соединения с разнообразными страницами Web, содержащими информацию о продуктах Microsoft и VBA.

Последняя команда в меню Help — это команда **About Microsoft Visual Basic**. Она отображает диалоговое окно, содержащее сведения об авторских правах на Microsoft Visual Basic. Диалоговое окно **About Microsoft Visual Basic** содержит также командную кнопку **System Info**, которая отображает информацию о вашей вычислительной системе: какие драйверы видеосистемы, звуковой системы и принтера установлены, какие программы загружены в память в данное время, какие программы зарегистрированы в системном реестре (Windows System Registry) и другую техническую информацию.

## Панели инструментов Редактора VB

Для многих пользователей выбрать командную кнопку с помощью мыши легче, чем выбрать команду в меню. Поэтому Редактор VB предоставляет важнейшие и наиболее часто используемые команды в виде кнопок на панели инструментов. Если вы часто работаете с Редактором VB, можете заметить, что использование командных кнопок на различных панелях Редактора VB ускоряет вашу работу.

По умолчанию Редактор VB отображает только панель инструментов **Standard**, показанную на рис. 2.4.

Как видно из рис. 2.4, в конце панели **Standard**, справа, находится кнопка (со стрелкой) **More Buttons**. Если вы хотите удалить с панели или добавить некоторые кнопки на панель **Standard**, нажмите на эту кнопку, а затем укажите соответствующую кнопку в появляющемся меню (рис. 2.5).

Кроме панели **Standard** Редактор VB имеет еще три панели: **Edit** (правка), **Debug** (отладка) и **UseForm**.

Панель **Edit** Редактора VB содержит несколько командных кнопок, полезных при редактировании текста в **Code Window** (окне кода). Эта панель предлагает даже пару команд, которых нет на панели **Standard**. Панели **Standard** и **Edit** описываются далее в этом разделе.

Вы можете управлять тем, какие панели инструментов отображает Редактор VB, с помощью команды **View | Toolbars** (вид | панели инструментов). Поскольку Редактор VB по умолчанию не отображает панель **Edit**, вам нужно будет выводить ее на экран вручную.



**Рис. 2.4.** Панель **Standard** Редактора VB содержит кнопки для важнейших и наиболее часто используемых команд:

1 – View host-приложение; 2 – Insert UserForm; 3 – Save <Документ>; 4 – Cut; 5 – Copy; 6 – Paste; 7 – Find; 8 – Undo; 9 – Redo; 10 – Run Macro/UserForm; 11 – Breake; 12 – Reset; 13 – Design Mode; 14 – Project Explorer; 15 – Properties Window; 16 – Object Browser; 17 – Toolbox; 18 – Microsoft Visual Basic Help; 19 – More Buttons

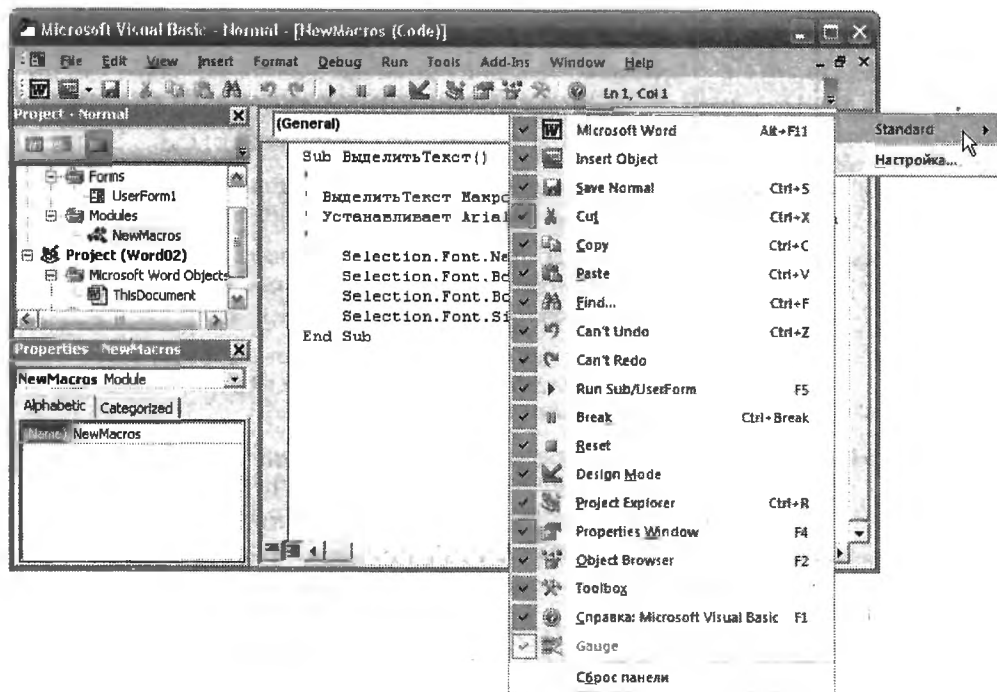


Рис. 2.5. Кнопка **More** панели **Standard** позволяет изменять содержимое этой панели

В следующих нескольких параграфах описываются сначала команды, имеющиеся на панели **Standard**, а затем — дополнительные команды, находящиеся на панели **Edit**.

Панель **Standard**

Итак, наиболее часто используемая разработчиками приложений при работе с Редактором VB панель инструментов **Standard** содержит 19 кнопок. Каждая кнопка является «быстрой» командой меню, действие которой приведено в таблице 2.9. При рассмотрении таблицы 2.9 обращайтесь к рис. 2.4 для сопоставления имени каждой командной кнопки со значком на передней стороне кнопки.

Таблица 2.9. Кнопки панели **Standard**

Наименование кнопки	Назначение
<b>View</b> <host application> (вид <host-приложение>)	Переключает на host-приложение VBA, из которого вы запустили Редактор VB. Значок этой кнопки изменяется в зависимости от конкретного приложения, из которого вы запускали редактор VB.
<b>Insert UserForm</b> (вставить UserForm)	Щелкните на кнопке со стрелкой вниз справа от этой кнопки, чтобы отобразить список объектов, которые вы можете вставить в текущий проект: <b>UserForm</b> , <b>Module</b> , <b>Class Module</b> или <b>Procedure</b> . Это можно выполнить и при использовании одноименных команд меню <b>Insert</b> (вставка).

Наименование кнопки	Назначение
<b>Save</b> (сохранить)	Сохраняет текущий проект так же, как <b>File   Save</b> .
<b>Cut</b> (вырезать)	Вырезает выделенный текст или объект и помещает его в Clipboard; так же, как команда <b>Edit   Cut</b> .
<b>Copy</b> (копировать)	Копирует выделенный текст или объект в Clipboard; так же, как команда <b>Edit   Copy</b> (правка   копировать).
<b>Paste</b> (вставить)	Вставляет текст или объект из Clipboard в Code Window или форму пользователя в позицию курсора так же, как команда <b>Edit   Paste</b> (правка   вставить).
<b>Find</b> (найти)	Открывает диалоговое окно <b>Find</b> для нахождения определенного слова или фразы в модуле; так же, как команда <b>Edit   Find</b> (правка   найти).
<b>Undo</b> (отменить)	Отменяет самую последнюю команду, если это возможно (не все действия могут быть отменены); так же, как команда <b>Edit   Undo</b> (правка   отменить набор).
<b>Redo</b> (вернуть набор)	Возобновляет самую последнюю отмененную команду; так же, как команда <b>Edit   Redo</b> .
<b>Run Macro</b> (запуск подпрограммы/ UserForm)	Запускает текущую процедуру или форму так же, как команда <b>Run   Sub/UserForm</b> .
<b>Break</b> (прервать)	Прерывает выполнение кода VBA; так же, как команда <b>Run   Break</b> (запуск   прервать).
<b>Reset</b> (сброс)	Перезапускает код VBA, как команда <b>Run   Reset</b> (запуск   сброс).
<b>Design Mode</b> (конструктор)	При нажатии этой кнопки Редактор VB переходит в режим Design (конструктора) так же, как командой <b>Run   Design Mode</b> (запуск   конструктор).
<b>Project Explorer</b> (окно проекта)	Отображает Окно проекта (Project Explorer) так же, как команда <b>View   Project Explorer</b> (вид   окно проекта).
<b>Properties Window</b> (окно свойств)	Отображает <b>Properties Window</b> (окно свойств) так же, как команда <b>View   Properties Window</b> (вид   окно свойств).
<b>Object Browser</b> (просмотр объектов)	Отображает диалоговое окно <b>Object Browser</b> (просмотр объектов) так же, как команда <b>View   Object Browser</b> (вид   просмотр объектов).
<b>Toolbox</b> (панель элементов)	Отображает <b>Toolbox</b> (панель элементов) так же, как команда <b>View   Toolbox</b> (вид   панели элементов).
<b>Справка: Microsoft Visual Basic</b>	Отображает окно Справочной системы Microsoft Office для контекстно-зависимой подсказки по текущей задаче.
<b>Cursor position</b> (положение курсора)	Эта область панели инструментов Standard фактически не является командной кнопкой и появляется только тогда, когда курсор находится в Code Window. Эта область показывает, в какой строке модуля, и в каком столбце строки находится курсор вставки.
<b>More Buttons</b>	Позволяет добавить или удалить кнопки панели.

### Панель инструментов *Edit* (правка)

По мере того как вы узнаете о написании кода VBA и все больше работаете в окне **Code Window** (программа), вы можете заметить, что использование панели **Edit** с целью получения «горячих клавиш» для различных команд редактирования экономит ваше время и усилия. Редактор VB не отображает автоматически панель **Edit**, если только вы сначала не вывели эту панель на экран, выполнив два шага, указанные в начале этого раздела. В табл. 2.10 приведены командные кнопки по умолчанию на панели инструментов **Edit** и описывается действие каждой кнопки.

**Таблица 2.10. Команды панели инструментов *Edit***

Наименование кнопки	Назначение
<b>List Properties/Methods</b> (список свойств/методов)	Отображает раскрывающийся список свойств и методов, принадлежащих объекту, который вы только что ввели; так же, как <b>Edit   List Properties/Methods</b> (правка   список свойств/методов).
<b>List Constants</b> (список констант)	Отображает раскрывающийся список predefined констант; так же, как <b>Edit   List Constants</b> (правка   список констант).
<b>Quick Info</b> (сведения)	Отображает всплывающее окно, показывающее правильный синтаксис для объектного метода; так же, как <b>Edit   Quick Info</b> (правка   сведения).
<b>Parameter Info</b> (параметры)	Отображает всплывающее окно, показывающее параметры (называемые также аргументами <i>[arguments]</i> ) функции или оператора VBA; так же, как <b>Edit   Parameter Info</b> (правка   параметры).
<b>Complete Word</b> (завершить слово)	Завершает текущее ключевое слово VBA, как только вы введете достаточно символов, чтобы VBA идентифицировал слово, которое вы вводите; так же, как <b>Edit   Complete Word</b> (правка   завершить слово).
<b>Indent</b> (увеличить отступ)	Сдвигает выделенный текст вправо на одну позицию табуляции; так же, как <b>Edit   Indent</b> (правка   увеличить отступ).
<b>Outdent</b> (уменьшить отступ)	Сдвигает выбранный текст влево на одну позицию табуляции; так же, как <b>Edit   Outdent</b> (правка   уменьшить отступ).
<b>Toggle Breakpoint</b> (точка останова)	Устанавливает или отменяет точку прерывания в исходном коде VBA; так же, как <b>Debug   Toggle Breakpoint</b> (отладка   точка останова).
<b>Comment Block</b> (закомментировать блок)	Превращает выделенный блок текста в <b>Code Window</b> в комментарий, добавляя символ комментария в начало каждой выделенной строки. Эквивалентной команды меню не имеется.
<b>Uncomment Block</b> (раскомментировать блок)	Удаляет символ комментария из блока выделенного текста в <b>Code Window</b> . Эквивалентной команды меню нет.

Наименование кнопки	Назначение
<b>Toggle Bookmark</b> (закладка)	Устанавливает или отменяет закладку в вашем исходном коде VBA; так же, как <b>Edit   Bookmarks   Toggle Bookmark</b> (правка   закладки   закладка).
<b>Next Bookmark</b> (следующая закладка)	Перемещает курсор вставки в <b>Code Window</b> к следующей закладке; так же, как <b>Edit   Bookmarks   Next Bookmark</b> (правка   закладки   следующая закладка).
<b>Previous Bookmark</b> (предыдущая закладка)	Перемещает курсор вставки в <b>Code Window</b> к предыдущей закладке; так же, как <b>Edit   Bookmarks   Previous Bookmark</b> (правка   закладки   предыдущая закладка).
<b>Clear All Bookmarks</b> (снять все закладки)	Удаляет все закладки в модуле; так же, как <b>Edit   Bookmarks   Clear All Bookmarks</b> (правка   закладки   снять все закладки).

## Редактирование макросов

После того как вы немного ознакомились со средой Редактора VB или, по крайней мере, с терминологией, связанной с компонентами редактора, пора рассмотреть более детально те макросы, которые вы должны были записать при помощи макрорекордера, и те, которые были написаны без рекордера и содержали только вывод краткого сообщения.

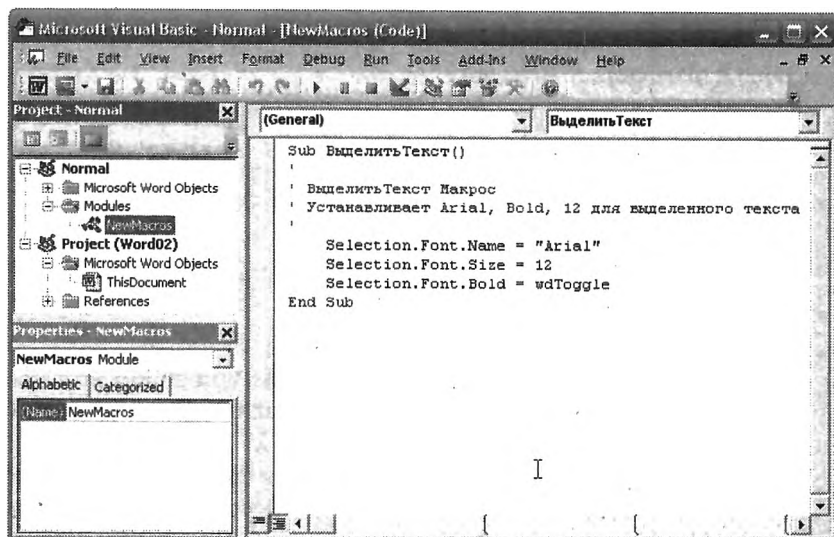
Начнем с редактирования ранее записанных макросов, так как, по мнению многих авторов работ в этой области, такой подход быстро приносит определенную пользу. Перед тем как вы сможете редактировать макрос, необходимо отобразить модуль, содержащий этот макрос. Например, чтобы редактировать макрос **ВыделитьТекст** в Word, который вы могли записать при чтении первой главы, вы должны сначала отобразить модуль в **Normal.dotm** (где было предложено сохранить записываемый макрос), содержащий исходный код макроса **ВыделитьТекст**. Аналогично, для редактирования макроса **FormatArialBold12** в Excel, который вы также записали в главе 1, вы должны сначала отобразить модуль в рабочей книге **Personal.xlsb** (где был сохранен записанный макрос), содержащей исходный код макроса **FormatArialBold12**.

### Отображение модуля

Как найти модуль, содержащий необходимый макрос? Для этого в Word 2007 выполните следующие шаги:

1. Нажмите **Alt+F11**, чтобы активизировать Редактор VB, если он еще не открыт.
2. Выберите команду **View | Project Explorer** (вид | окно проекта), чтобы отобразить **Project Explorer**.
3. Просмотрите дерево-список в **Project Explorer**, чтобы найти модуль, который вам необходимо отобразить. Если опция просмотра папок в **Project Explorer** включена, вы найдете все модули проекта в папке **Modules** вашего проекта (рис. 2.6).
4. Дважды щелкните на модуле, который вам необходим. Редактор VB отображает этот модуль в **Code Window** (окно программы).

**Рис. 2.6**  
Все модули  
проекта можно  
найти в папке  
**Modules**



После того как вы отобразите модуль, можете использовать список **Procedure** (процедура) в **Code Window** для получения определенного макроса в этом модуле. Вы можете также использовать метод, описанный в следующем разделе, для нахождения макроса без предварительного открытия модуля, содержащего этот макрос.

### Как найти записанный макрос

Поскольку Word всегда сохраняет записанные макросы в модуле **NewMacros** документа или шаблонного файла, найти исходный код макроса в Word просто. Найти исходный код макроса в Excel может быть немного сложнее. Когда вы записывали макрос **FormatArialBold12** в Excel 2007 во время изучения первой главы, вы сохранили его в рабочей книге **Personal.xlsb**. Если это был первый макрос, какой вы когда-либо записали и сохранили в **Personal.xlsb**, Excel сохранил макрос **FormatArialBold12** в модульном листе с именем **Module1**. Если бы вы затем записали другой макрос в том же самом сеансе работы, Excel также сохранил бы его в том же самом модуле (**Module1**). Если вы выходили из Excel между временем записи вами макроса **FormatArialBold12** и временем записи другого макроса, Excel сохранил последний макрос в другом модуле (скорее всего, в **Module2**).

### Использование инструмента Object Browser

Вместо поиска по всему тексту во всех модулях определенного макроса вы можете использовать инструмент, называемый **Object Browser**, чтобы найти этот макрос, вывести на экран или редактировать, независимо от того, где он находится — в Word, Excel или других приложениях.

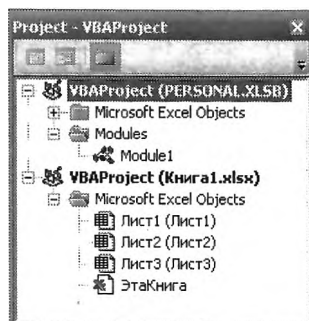
Для использования инструмента **Object Browser** в Excel выполняйте следующие шаги:

1. Нажмите **Alt+F11**, чтобы открыть Редактор VB.
2. Выберите **View | Project Explorer** (вид | окно проекта) для отображения **Project Explorer** (рис. 2.7).



**Рис. 2.7**

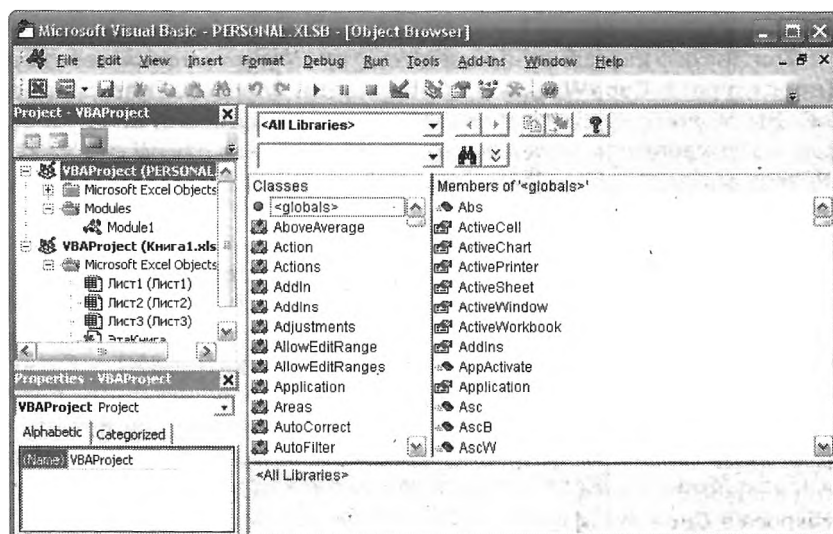
Выделите проект в **Project Explorer** перед поиском макроса в окне **Object Browser**



3. В **Project Explorer** выделите проект VBA, в котором вам необходимо найти макрос. Например, на рис. 2.7 показана книга **Personal.xlsb**, выделенная в **Project Explorer**.
4. Выберите команду **View | Object Browser** (вид | просмотр объектов). Редактор VB отобразит диалоговое окно **Object Browser**, показанное на рис. 2.8.

**Рис. 2.8**

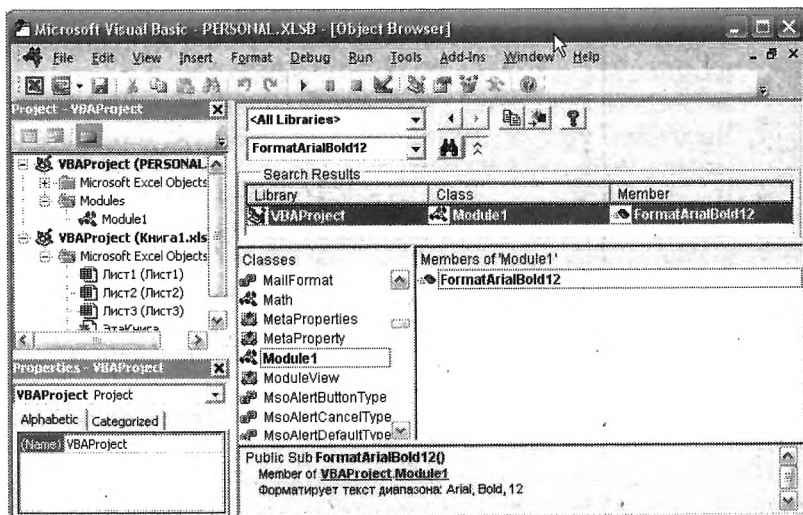
Используйте **Object Browser** для поиска макросов



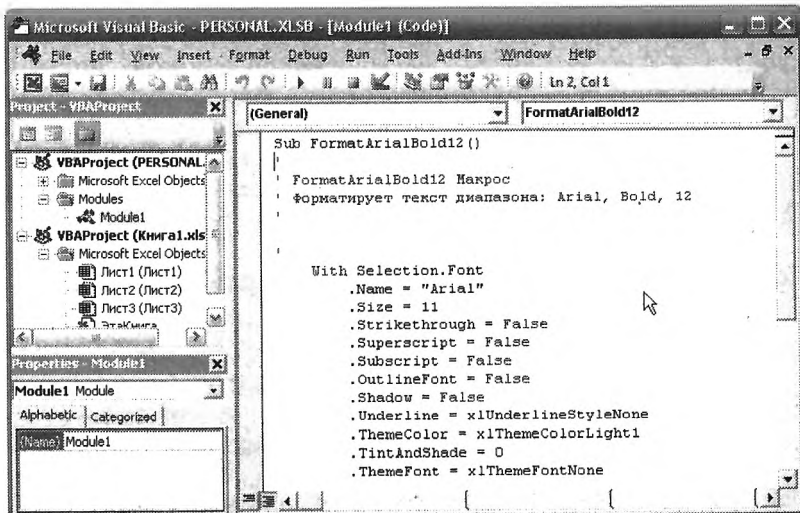
5. Убедитесь, что в раскрывающемся списке **Project/Library** (проект/ библиотека) выделен элемент **<All Libraries>** (<все библиотеки>). Окно **Object Browser** отображает все имеющиеся модули и объекты в списке **Classes** (Классы).
6. Введите имя макроса, который вам необходимо найти, в текстовом окне **Search** (поиск). Например, чтобы найти исходный код для макроса **FormatArialBold12** в Excel, введите **FormatArialBold12** в поле **Search Text** (образец поиска).

7. Щелкните на кнопке **Search** (поиск) в окне **Object Browser** (рис. 2.8). **Object Browser** выполняет поиск текста, который вы ввели в окно **Search Text**, в вашем проекте VBA (а также в других открытых в данный момент библиотеках). После выполнения поиска меняется вид окна **Object Browser**: оно отображает список **Search Results**, показанный на рис. 2.9. Если **Object Browser** обнаружил макросы, совпадающие с текстом в окне **Search Text** (образец поиска), они отображаются в списке **Search Results** (результаты поиска). В противном случае в этом списке отображается сообщение **No items found** (элементы не найдены).
8. Щелкните на макросе в списке **Search Results**, затем — на кнопке **View Definition** (описания) в окне **Object Browser** (рис. 2.9). Редактор VB открывает **Code Window** (программы) для модуля, содержащего этот макрос, и отображает исходный код макроса (рис. 2.10).

**Рис. 2.9**  
Окно **Object Browser** после выполнения поиска макроса



**Рис. 2.10**  
Код найденного макроса в **Code Window**



### Просмотр модулей проекта с помощью Object Browser

Если ваш проект не содержит большого количества различных модулей, то легче находить макрос, выполняя следующую альтернативную процедуру:

1. Нажмите **Alt+F11**, чтобы открыть Редактор VB.
2. Выберите **View | Project Explorer** (вид | окно проекта), чтобы отобразить **Project Explorer**.
3. В **Project Explorer** выделите проект **VBA**, в котором вам необходимо найти макрос.
4. Выберите команду **View | Object Browser** (вид | просмотр объектов). Редактор VB отображает диалоговое окно **Object Browser**.
5. Выделите имя вашего проекта в раскрывающемся списке окна **Object Browser**. Если только вы не изменили имени вашего проекта, он называется **VBAProject в Excel (Project — в Word)**.
6. Выделите модуль в списке **Classes** (Классы). Окно **Object Browser** отображает теперь все макросы в выделенном модуле в списке **Members of <object>** (компонент *<объект>*). (Точный заголовок этого списка изменяется в зависимости от объекта, выделенного в списке **Classes**.)
7. Выделите макрос, исходный код которого вам необходимо увидеть, в списке **Members of <object>**.
8. Щелкните на кнопке **View Definition** (описания) в окне **Object Browser**. Редактор VB открывает **Code Window** для модуля, содержащего этот макрос, и отображает исходный код макроса.

Когда вы выделяете отдельный проект в списке **Project/Library** (проект/библиотека) окна **Object Browser**, список **Classes** (классы) будет содержать только те объекты, которые являются частью вашего проекта, — обычно один или больше модулей, сам документ Word или рабочую книгу Excel и (если ваш проект является рабочей книгой Excel) один или больше рабочих листов.

После того как вы отобразите макрос в **Code Window**, используя любой из методов, описанных в предыдущем разделе, можете просмотреть или отредактировать исходный код этого макроса. На рис. 2.10 показано **Code Window**, отображающее исходный код макроса **FormatArialBold12**, записанного в главе 1. Вы можете использовать горизонтальную и вертикальную линейки прокрутки для изменения позиции макроса в окне. Можно также использовать команду **Edit | Find** (правка | найти) для поиска имени макроса в модульном листе. Использование команды **Edit | Find** для поиска текста в модульном листе сходно с поиском текста в рабочем листе Excel, документе Word и т.д. (Подробнее о команде **Edit | Find** можно узнать из справочной системы Редактора VB.)

### Составные части записанного макроса

Если вы рассмотрите несколько записанных макросов, то заметите, что все они имеют общие элементы. Листинг 2.1 показывает исходный код для макроса **FormatArialBold12** в Excel, который вы записали при изучении главы 1. Листинг 2.2 показывает исходный код макроса **ВыделитьТекст** в Word, также записанный вами во время работы с главой 1.

---

**Замечание**

Исходный код макроса в реальных модулях не включает номер каждой строки; в этой книге во всех листингах приведены номера строк, чтобы было легче обозначать и обсуждать отдельные строки в различных листингах-примерах.

---

---

**Листинг 2.1.** Макрос FormatArialBold12, записанный в Excel

```
1: ' FormatArialBold12 Макрос
2: ' Форматирует текст диапазона: Arial, Bold, 12
3: '
4:
5: '
6:     With Selection.Font
7:         .Name = "Arial"
8:         .Size = 11
9:         .Strikethrough = False
10:        .Superscript = False
11:        .Subscript = False
12:        .OutlineFont = False
13:        .Shadow = False
14:        .Underline = xlUnderlineStyleNone
15:        .ThemeColor = xlThemeColorLight1
16:        .TintAndShade = 0
17:        .ThemeFont = xlThemeFontNone
18:    End With
19:    Selection.Font.Bold = True
20:    With Selection.Font
21:        .Name = "Arial"
22:        .Size = 12
23:        .Strikethrough = False
24:        .Superscript = False
25:        .Subscript = False
26:        .OutlineFont = False
27:        .Shadow = False
28:        .Underline = xlUnderlineStyleNone
29:        .ThemeColor = xlThemeColorLight1
30:        .TintAndShade = 0
31:        .ThemeFont = xlThemeFontNone
32:    End With
33: End Sub
```

---

---

**Листинг 2.2.** Макрос Выделить Текст, записанный в Word

```
1: Sub ВыделитьТекст()
2: '
3: ' ВыделитьТекст Макрос
4: ' Устанавливает Arial, Bold, 12 для выделенного текста
5: '
6:     Selection.Font.Name = "Arial"
7:     Selection.Font.Size = 12
8:     Selection.Font.Bold = wdToggle
9: End Sub
```

---

В листинге 2.1 строка 1 является началом записанного макроса. Каждый макрос VBA начинается с ключевого слова **Sub**, за которым следует имя макроса. Строку, содержащую ключевое слово **Sub** и имя макроса, называют *строкой объявления (declaration) макроса*. За именем макроса всегда следуют пустые круглые скобки.

В листинге 2.1 строки 2–5 являются комментариями. *Комментарий (comment)* — это строка в макросе VBA, которая, действительно, не содержит инструкции, являющиеся частью этого макроса. Следует всегда помещать комментарии в исходный код макроса с информацией о его назначении. Обратите внимание на то, что каждая строка комментария начинается с апострофа (' ). Для VBA любой текст, который следует за апострофом, представляет собой комментарий, начинающийся с апострофа и заканчивающийся в конце этой строки. В языках программирования, в которых допускаются многострочные комментарии, такие комментарии называют *конечными*.

Записанный макрос всегда начинается со строк комментариев, формулирующих имя макроса и содержащих текст, который вы ввели в текстовое окно **Description** (описание) диалогового окна **Record Macro** (запись макроса) в момент записи этого макроса. Конкретное количество и содержимое строк комментариев в записанном макросе зависит от длины вводимого вами описания.

Сразу за объявлением макроса следует *тело (body)* макроса (которое может включать или не включать строки комментариев). В листинге 2.1 строки 6–32 составляют тело макроса **ВыделитьТекст**. Каждая строка в теле макроса состоит из одного или более операторов VBA. *Оператор (statement) VBA* — это последовательность ключевых слов и других символов, которые вместе составляют одну полную инструкцию для VBA. Макрос VBA состоит из одного или нескольких операторов.

Операторы VBA в записанном макросе содержат инструкции, соответствующие действиям, которые вы выполняли, когда записывали этот макрос. Например, строка 21 в листинге 2.1 устанавливает шрифт **Arial**, строка 22 — размер шрифта и так далее. В листинге 2.2 строка 6 устанавливает шрифт **Arial**, а строка 7 устанавливает размер шрифта. Каждый из этих операторов соответствует действию, выполняемому при записи этого макроса.

За телом макроса следует строка, содержащая ключевые слова **End Sub**, которые сообщают VBA о том, что достигнут конец макроса. Макрос в листинге 2.1 заканчивается в строке 33, а в листинге 2.2 — в строке 9.

Когда вы запускаете макрос, VBA начинает выполнение кода с первой строки в теле этого макроса (первая строка после объявления макроса) и выполняет инструкции в этой строке последовательно слева направо. Затем VBA переходит к следующей строке и выполняет инструкции в ней и так далее, пока он не достигнет конца макроса, обозначенного оператором **End Sub**. VBA игнорирует любые строки комментариев, которые могут появиться в теле макроса.

Как вы могли уже заметить, многие строки в обоих записанных макросах имеют отступ от левого края. Каждый уровень отступа помогает отделять одну часть макроса от другой. Заметьте, что все тело каждого макроса смещено вправо между ключевыми словами **Sub...End Sub** для наглядного выделения тела макроса. В других местах операторы расположены с еще большим отступом. Этот больший уровень отступа помогает определить все операторы, заключенные между ключевыми словами **With...End With**.

Макрорекордер при создании макроса автоматически делает отступы в коде, чтобы пользователю было легче читать записанный макрос. Когда вы будете писать собственные макросы, следует всегда делать отступы в коде для выделения различных секций макроса. Сместить строки не обязательно; макросы в листингах 2.1 и 2.2 будут так же выполняться, если все строки будут выровнены по левому краю модуля. Размещение строк с отступом выполняется хорошими программистами лишь для более легкого чтения и сопровождения их программ.

При использовании цветного монитора, когда вы будете просматривать записанный макрос на экране (используя редактор VB), то заметите, что различные части текста макроса отображаются различными цветами. Комментарии отображаются зеленым цветом, тогда как **Sub**, **End Sub** и другие ключевые слова VBA — синим. Остальной текст в макросе отображается черным цветом для указания на то, что он содержит данные и программные операторы, созданные пользователем. VBA выполняет цветовое кодирование текста на экране, чтобы можно было легко определить, какую часть макроса или оператор вы просматриваете.

### Редактирование текста макроса

При редактировании кода макроса в модуле вы можете использовать команды и методы, известные вам как пользователю Windows, Word или Excel. Редактирование текста, отображаемого в **Code Window**, похоже на редактирование текста в Windows Notepad или в WordPad. Вы используете ту же клавиатуру, мышь или команды меню **Edit** (для добавления, удаления, выделения, вырезания, копирования или вставки текста в модуле), которые применяются в Windows Notepad, WordPad.

Для сохранения изменений, которые вы вносите в модуль, используйте команду **File | Save** (файл | сохранить) Редактора VB или щелкните на кнопке **Save** (сохранить) на панели инструментов. Любые изменения, которые вы выполняете в модуле, также сохраняются всякий раз, когда вы сохраняете документ или файл рабочей книги, содержащий этот модуль. (Word автоматически сохраняет любые изменения в шаблоне **Normal.dotm**, когда вы выходите из Word; Excel выдает запрос на сохранение любых изменений, выполненных в **Personal.xlsb**, при выходе.)

### Замечание

---

Вы можете использовать цветовое кодирование для контроля правильности введенных вами операторов: если во введенном операторе (после окончания ввода) ни одно ключевое слово не изменило свой цвет, это явный признак наличия ошибки.

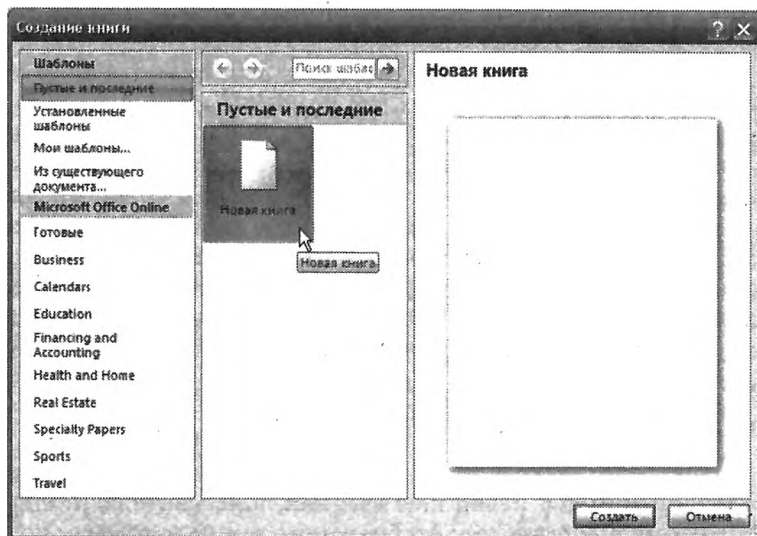
---

В качестве примера редактирования макроса сначала запишите новый макрос в Excel следующим образом:

1. Запустите макрорекордер, устанавливая параметры в диалоговом окне **Record Macro** (запись макроса) так, чтобы макрос сохранялся в книге **Personal**.
2. Дайте макросу имя **NewBook**.

3. Запишите следующие действия: щелкните кнопку **Office**, выберите из меню **Создать** и в окне **Создание книги** щелкните дважды на значке **Новая книга** (рис. 2.11).xls.

**Рис. 2.11**  
**Окно Создание книги**



4. Снова щелкните кнопку **Office**, выберите из меню **Сохранить как** для сохранения этой рабочей книги (используйте имя **NewBook**). Сразу же после сохранения файла щелкните кнопку **Office** и выберите **Заккрыть**, чтобы закрыть файл.
5. Остановите макрорекордер.

Макрос, который будет записан в результате вышеприведенных действий, имеет вид, представленный в листинге 2.3.

### Листинг 2.3. Нередактированный макрос Newbook

```

1: Sub NewBook()
2: '
3: ' NewBook Макрос
4: '
5: '
6:     Workbooks.Add
7:     ActiveWorkbook.SaveAs Filename:=
8:         "Н:\Программирование на VBA 2007\Документы\NewBook.xlsx", _
9:         FileFormat:=
10:         xlOpenXMLWorkbook, CreateBackup:=False
11:     ActiveWorkbook.Close
12:     ActiveWindow.Close
13: End Sub

```

Листинг 2.4 показывает, как может выглядеть макрос **NewBook** после «косметического» редактирования. Исходный код этого макроса был отредактирован так, чтобы он был более читабельным.

**Листинг 2.4. Отредактированный макрос Newbook**

```
1: Sub NewBook()  
2: '  
3: ' NewBook Макрос  
4: '  
5:   Workbooks.Add  
6:   ActiveWorkbook.SaveAs _  
7:     Filename:= _  
8:     "H:\Программирование на VBA 2007\Документы\NewBook.xlsx", _  
9:     FileFormat:=xlOpenXMLWorkbook, _  
10:    CreateBackup:=False  
11:  ActiveWorkbook.Close  
12:  ActiveWindow.Close  
13: End Sub
```

**Символ продолжения строки VBA**

Посмотрите на строки 5–8 в листинге 2.4. Обратите внимание на символ подчеркивания () в конце каждой из этих строк. Заметьте также, что символу подчеркивания предшествует пробел, отделяя его от другого текста в строке. Эта особая комбинация символа пробела, за которым следует символ подчеркивания в конце строки, называется *символом продолжения строки* (*line-continuation character*) и сообщает VBA о том, что следующая строка макроса должна быть присоединена к текущей строке для образования единого оператора. В листинге 2.4 строки 5–8 являются одним оператором, в данном случае — командой, которая сохраняет рабочую книгу и устанавливает несколько опций для нее. Для того чтобы макрос был более читабельным, макрорекордер разделил эту единую *логическую* строку на несколько *физических* строк, используя символ продолжения строки, но это разделение (см. листинг 2.3) не намного улучшило читабельность кода. Поэтому код пришлось еще подправить (см. листинг 2.4).

Заметьте также, что операторы, разделенные (уместнее было бы говорить «объединенные») символом продолжения строки, размещаются с еще большим отступом, чтобы их было легче обнаружить при чтении кода.

VBA проверяет каждую новую или измененную строку в макросе для определения того, является ли эта строка правильной с точки зрения синтаксиса. Если новая или измененная строка имеет правильный синтаксис, VBA выполняет цветовое кодирование частей строки, используя схему цветового кодирования, описанную ранее. Если в строке имеется ошибка синтаксиса, VBA отображает всю строку красным цветом и может вывести одно или несколько возможных сообщений об ошибке синтаксиса. Сообщения о синтаксических ошибках описываются более подробно далее в этой главе.

**Перемещение или копирование макроса из одного модуля в другой**

Для копирования одного макроса используйте Clipboard, выполняя следующие шаги:

1. Отобразите макрос, который вам необходимо копировать.
2. Выделите весь текст исходного кода макроса. Убедитесь, что вы выделили весь макрос, включая строки **Sub**, **End Sub** и все строки между ними.



3. Используйте команду **Edit | Copy** для копирования текста макроса в Clipboard.
4. Отобразите модуль, в который вам необходимо копировать этот макрос.
5. Используйте команду **Edit | Paste** для вставки текста макроса в модуль.

Вы можете пользоваться описанным выше методом для копирования макроса в тот же модуль, в другой модуль одного и того же проекта или в модуль другого проекта. Можно также использовать этот метод для вставки текста макроса в другие приложения Windows, такие как Notepad.

### Сохранение и перенос модулей как текстовых файлов

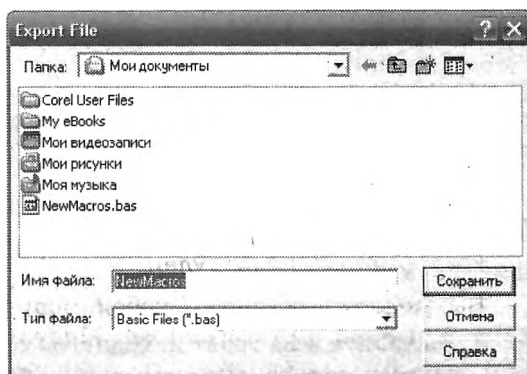
Если вам необходимо сделать резервную копию всего модуля, вы можете сохранить его как текстовый файл. Сохранение целого модуля как текстового файла полезно, если вы хотите импортировать модуль в другой проект VBA, создать архивные копии вашей работы или перенести модули VBA в Visual Basic 6. Такое сохранение модуля называется *экспортированием* модуля. После того как вы экспортируете модуль, вы можете импортировать его в любой проект VBA или Visual Basic.

### Экспортирование модулей

Для экспортирования модуля как текстового файла выполните следующее:

1. Выделите в **Project Explorer** модуль, который хотите экспортировать.
2. Выберите команду **File | Export File**. Редактор VB отображает диалоговое окно **Export File** (экспорт файла), показанное на рис. 2.12. Диалоговое окно **Export File** работает в основном так же, как любое диалоговое окно **File Save** (сохранение документа) в Windows.
3. Убедитесь, что в списке **Тип файла** (Save as type) выделен пункт **Basic Files (\*.bas)**. Расширение **.bas** определяет этот файл как файл исходного кода VBA или Visual Basic. Информация, сохраняемая в этом файле, — это читаемый текст.
4. Введите имя экспортируемого файла в текстовое окно **Имя файла** (File Name). Редактор VB вводит имя модуля по умолчанию.
5. Щелкните на кнопке **Сохранить** (Save), чтобы экспортировать файл. Редактор VB экспортирует выделенный модуль и закрывает окно **Export File**.

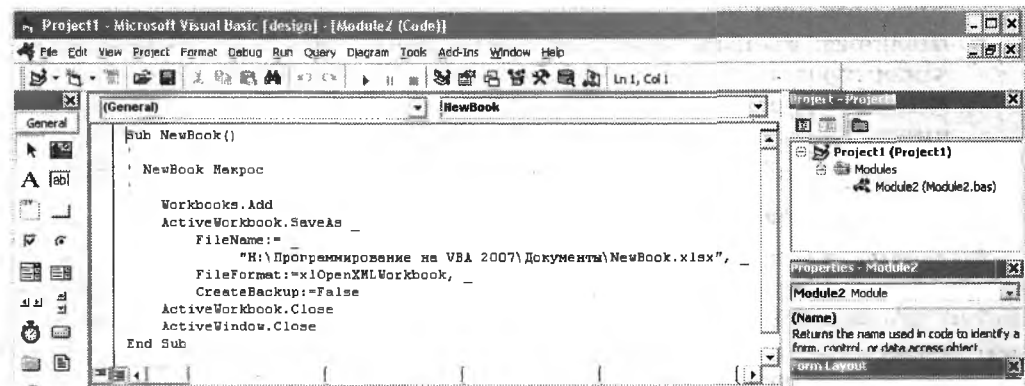
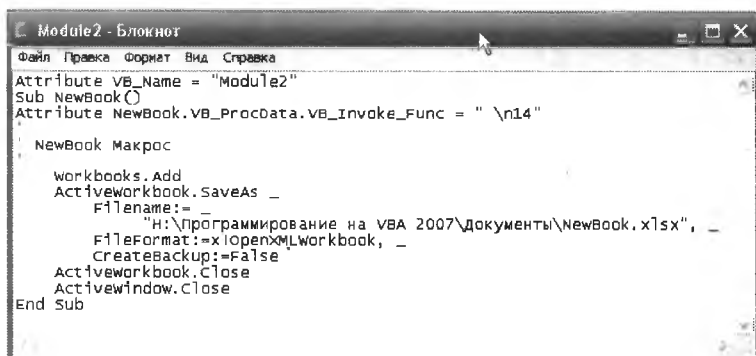
**Рис. 2.12**  
Используйте окно **Export File** для сохранения всего модуля в виде текстового файла



Когда вы экспортируете модуль, Редактор VB создает файл читаемого текста, содержащий все макросы этого модуля. Вы можете просматривать и редактировать .bas-файлы, созданные Редактором VB, с помощью Windows Notepad или Visual Basic. На рис. 2.13 и 2.14 показан экспортированный в файл .bas модуль, содержащий макрос Newbook листинга 2.5, отображенный в Windows Notepad и Visual Basic 6, соответственно.

Обратите внимание на то, что текст в файле .bas, отображенный приложением Notepad (рис. 2.13), содержит строки, начинающиеся со слова **Attribute** и отсутствующие в модуле в Code Window. Редактор VB добавляет эти дополнительные строки к экспортированному файлу .bas, поскольку они содержат информацию, которая потребуется VBA или Visual Basic, если вы в дальнейшем будете импортировать файл .bas в другой проект.

**Рис. 2.13**  
Экспортированный  
модульный файл,  
отображаемый  
в Windows Notepad



**Рис. 2.14**  
Экспортированный модульный файл, отображаемый в Visual Basic 6

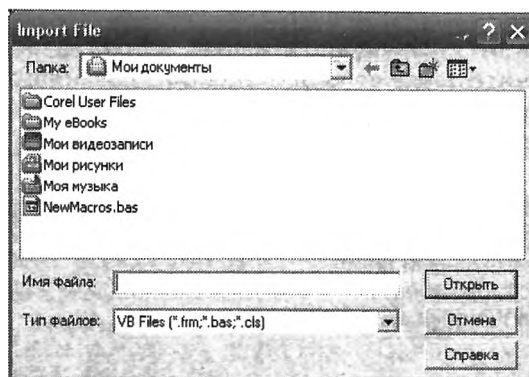
### Импортowanie модулей

Вы можете добавлять любой модуль, экспортированный вами как файл .bas, в любой из проектов VBA. Модуль добавляется в проект импортированием текстового файла .bas. Для того чтобы импортировать .bas-файл, выполните следующее:

1. Выделите проект в **Project Explorer**, в который необходимо импортировать файл **.bas**.
2. Выберите команду **File | Import File**. Редактор VB отображает диалоговое окно **Import File** (импорт файла), отображенное на рис. 2.15. Диалоговое окно **Import File** работает в основном так же, как любое диалоговое окно открытия файла Windows.

**Рис. 2.15**

Окно **Import File** позволяет добавить к проекту модуль, ранее экспортированный из Редактора VB как текстовый файл



3. Используйте элемент **Папка (Look In)** для перемещения на диск и в папку, где содержится файл, который вам необходимо импортировать в ваш проект. Это должен быть файл, который вы создали, используя Редактор VB для экспортирования модуля.
4. Убедитесь, что в списке **Тип файлов (Files Of Type)** выделен пункт **VB Files (\*.frm, \*.bas, \*.cls)**.
5. Щелкните мышью дважды на имени файла, который вам необходимо импортировать. Редактор VB читает файл и добавляет модуль в ваш проект. Если проект уже содержит модуль с тем же именем, что и модуль, импортируемый вами, Редактор VB добавляет число в конец имени модуля (1, 2, 3 и так далее) для создания уникального имени модуля.

### Удаление модулей из проекта

Возможно, вы решите, что вам больше не нужны макросы, содержащиеся в определенном модуле. Вам может понадобиться удалить модуль из проекта, потому что макросы, которые он содержит, были заменены новыми версиями или потому, что этот модуль содержит экспериментальные макросы, ненужные вам более. Модуль можно удалить из проекта VBA, выполнив следующее:

1. Выделите модуль, который хотите удалить из проекта, в **Project Explorer** Редактора VB.
2. Выберите команду **File | Remove <object>** (файл | удалить <объект>). Редактор VB отображает окно сообщения, запрашивающее у вас, хотите ли вы экспортировать модуль перед тем, как удалить его (рис. 2.16).

**Рис. 2.16**

Окно сообщения, запрашивающее у вас, хотите ли вы экспортировать модуль перед тем, как удалить его



3. Щелкните на кнопке **Да**, если хотите экспортировать модуль перед его удалением. (Настоятельно рекомендуется экспортировать модуль как файл **.bas**, если только вы не уверены абсолютно, что вам никогда не понадобятся макросы, содержащиеся в этом модуле). Иначе можно щелкнуть на кнопке **Нет** для удаления модуля без его экспортирования.
4. Если вы выберете экспортирование модуля перед его удалением, Редактор VB отображает диалоговое окно **Export File**, описанное ранее в этом разделе. Заполните диалоговое окно **Export File**, как уже было описано, и щелкните на кнопке **Сохранить**.

В любом случае Редактор VB удаляет модуль из проекта.

## Написание новых макросов и процедур

Чтобы записать макрос без использования макрорекордера, вы можете ввести этот макрос в существующий модуль или создать новый модуль, который будет содержать этот макрос.

До сих пор в этой книге использовался термин *макрос (macro)* как для макросов, которые вы записываете рекордером, так и для макросов, которые пишете сами. Существует другой термин, помогающий различать эти макросы. Строго говоря, термин *макрос* применим только для инструкций, которые вы записываете с помощью макрорекордера (или создаете при помощи, например, Access Macro Builder). Иногда в литературе макросами называют программы, которые не имеют аргументов (вы, наверное, уже поняли, что записываемые рекордером макросы не могут иметь аргументов). Макросы, которые вы пишете заново более точно называются *подпроцедурами (subprocedures)* или просто процедура *процедурами (procedures)*. Далее в этой книге термин *макрос* относится к коду, записываемому с помощью макрорекордера, а термин *процедура* — к коду VBA, который вы пишете сами.

### Вставка и переименование модуля

Если документ (шаблон, рабочая книга, презентация, рисунок) в котором вы хотите сохранить процедуру, еще не содержит модуль, вы должны вставить модуль перед тем, как сможете написать процедуру VBA в этом проекте. Вы можете также вставить новый модуль, если существующие модули заполнены (модуль может содержать не более 4000 строк) или если вы просто хотите создать новую процедуру VBA в ее собственном модуле.

Если вы решили написать процедуру VBA в новом модуле, выполните следующие шаги для добавления модуля в проект:

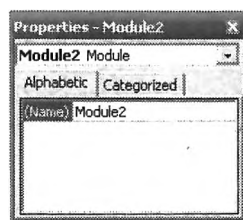
1. Убедитесь, что в host-приложении VBA (Word, Excel, Outlook, PowerPoint, Visio или FrontPage) открыт тот документ, шаблон или рабочая книга, в котором вы хотите сохранить процедуру.
2. Нажмите **Alt+F11** для активизации Редактора VB.
3. В **Project Explorer** выделите проект, в который хотите добавить модуль (скорее всего, в этот момент он там будет один).
4. Выберите команду **Insert | Module** (вставка | модуль) или на панели **Standard** с помощью кнопки **Insert <object>** (вторая кнопка слева) выберите в раскрывающемся меню **Module**. Редактор VB добавляет новый модуль в проект и открывает **Code Window** для нового модуля.

Когда Редактор VB вставляет новый модуль, он дает ему имя в соответствии с правилами, изложенными ранее в этой главе; ваш новый модуль будет иметь имя по умолчанию **Module** с последующим номером. Всякий раз при вставке модуля следует переименовывать его, чтобы этот модуль имел описательное имя. Для переименования модуля выполните следующие шаги:

1. Выделите в Редакторе VB модуль, который будете переименовывать.
2. Если **Properties Window** (окно свойств) еще не выведено на экран, выберите команду **View | Properties Window** (вид | окно свойств) (или щелкните на кнопке **Properties** на панели инструментов), чтобы отобразить его. На рис. 2.17 показано **Properties Window** для модуля; модули имеют только одно свойство — имя.

**Рис. 2.17**

Изменяйте имя модуля, изменяя его свойство **Name**



3. В текстовое поле **Name** (имя) в **Properties Window** введите новое имя для модуля. Как только вы уберете курсор вставки из текстового поля **Name**, Редактор VB переименует модуль.

### Выделение существующего модуля

Для написания новой процедуры в существующем модуле необходимо сначала открыть **Code Window** (окно программы) для модуля, в котором вы хотите написать процедуру. Чтобы открыть **Code Window** для модуля, либо щелкните дважды на этом модуле в **Project Explorer**, либо выделите этот модуль в **Project Explorer** и затем выберите команду **View | Code** (вид | программа).

### Написание текста процедуры

Чтобы написать исходный код для процедуры, независимо от того, добавляете ли вы процедуру в новый или в уже существующий модуль, поместите курсор вставки в **Code Window** в то место в модуле, куда вы хотите ввести новую процедуру.

Вы можете ввести исходный код новой процедуры VBA в любом месте в модуле, следя за тем, чтобы новая процедура начиналась *после*, возможно, имеющегося там оператора **End Sub**, который заканчивает предыдущую процедуру, и *перед* оператором **Sub**, который начинает следующую процедуру в модуле. Конечно, самым простым является добавление новых процедур в конец модуля.

Когда вы пишете новую процедуру, вы должны указать имя процедуры и включить ключевое слово **Sub** в начале процедуры и ключевые слова **End Sub** — в конце. Если вы опустите любой из этих трех элементов, синтаксис процедуры не будет верным и VBA отобразит сообщение об ошибке, когда вы попытаетесь запустить эту процедуру.

Первой классической программой в любом описании языка программирования является программа, отображающая на экране сообщение «*Hello, World*».

В листинге 2.5 приведена такая программа VBA, состоящая из единственного оператора.

### Листинг 2.5. Процедура HelloWorld

---

```
1: Sub HelloWorld()  
2:     MsgBox "Hello, World!"  
3: End Sub
```

---

Для самостоятельного ввода этой программы VBA выполните следующие шаги:

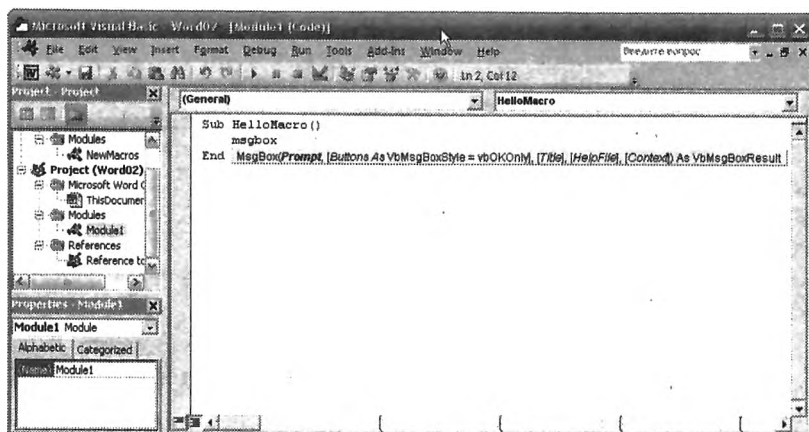
1. Откройте любой документ Word или рабочую книгу Excel (или, наконец, любое приложение Microsoft, которое поддерживает VBA) или создайте новый документ или рабочую книгу.
2. Нажмите **Alt+F11** для активизации Редактора VB.
3. В **Project Explorer** выделите документ или рабочую книгу, в которой хотите сохранить эту программу.
4. Выберите **Insert | Module** (вставка | модуль), чтобы добавить новый модуль к вашему проекту. Редактор VB добавляет новый модуль и открывает для него **Code Window**.
5. Переименуйте новый модуль, дав ему имя, например, **FirstProgram**.
6. Убедитесь, что курсор вставки находится в начале пустой строки в **Code Window**, и введите текст, показанный в листинге 2.5, нажимая на клавишу **Enter** в конце каждой строки для начала новой строки.

Вводите исходный код из листинга 2.5 в модуль точно так, как он выглядит в листинге, но без номеров строк.

Редактор VB содержит несколько возможностей, помогающих в написании процедур. Во-первых, как только вы нажмете на клавишу **Enter** после ввода ключевого слова **Sub** и имени процедуры, Редактор VB автоматически добавляет ключевые слова **End Sub**. Таким образом, вам не нужно беспокоиться о том, что вы можете случайно забыть об этом важном элементе процедуры.

Во-вторых, Редактор VB включает возможность, известную как **Auto Quick Info** (краткие сведения). Как только вы вводите ключевое слово **MsgBox** и нажимаете на клавишу пробела (строка 2 листинга 2.5), точнее, переходите к следующему элементу VB-оператора, появляется всплывающее окно, показывающее полный список аргументов для встроенной процедуры VBA или функции, которую вы только что ввели (**MsgBox** — в данном случае). На рис. 2.18 показано всплывающее окно с информацией об аргументах встроенной VBA-процедуры **MsgBox**. Аргумент, значение которого, как ожидается, вы теперь должны ввести, выделен полужирным шрифтом во всплывающем окне **Auto Quick Info**. [*Аргумент (argument)* — это информация, которая необходима процедуре для выполнения ее задачи; подробнее об аргументах вы узнаете далее]. Всплывающее окно **Auto Quick Info** закрывается, если вы нажимаете на клавишу **Enter** для начала новой строки в **Code Window** или используете клавиши со стрелками или мышь для перемещения курсора вставки с текущей строки на другую. Вы можете также закрыть окно **Auto Quick Info**, используя клавишу **Esc**.

**Рис. 2.18**  
Всплывающее  
окно **Auto Quick**  
**Info** для  
**MsgBox**



Вы можете включать и выключать функцию **Auto Quick Info** с помощью команды **Tools | Options** (сервис | параметры) в Редакторе VB. Более подробно о диалоговом окне **Options** (параметры) можно узнать из справочной системы Редактора VB.

Каждое объявление процедуры должно начинаться с ключевого слова **Sub**, за которым следует пробел и затем — имя процедуры. В листинге 2.5 именем процедуры является **HelloWorld**. Финальная часть объявления процедуры — это пара пустых круглых скобок. Эти скобки обязательны. (Вы узнаете о их назначении в следующих главах). Если вы не включаете эти скобки, VBA добавляет их в объявление процедуры, когда курсор вставки убирается вами из строки объявления.

Вторая строка в листинге 2.5 образует тело процедуры и является единственным оператором, выполняющим «полезную» работу. Тело процедуры может состоять из одного или многих операторов или не состоять ни из одного оператора. Оператор **MsgBox** отображает сообщение в диалоговом окне на экране и описывается более подробно далее в этой главе.

Третья и последняя строка процедуры **HelloWorld** — **End Sub** — завершает процедуру. Эта строка сообщает VBA о том, что это — конец процедуры; VBA прекращает выполнение процедуры при достижении им этой строки. Подобно объявлению процедуры оператор **End Sub** должен быть первыми двумя словами в строке и должен быть один в строке, хотя вы можете добавлять конечный комментарий после него. Как было упомянуто ранее, Редактор VB автоматически добавляет эту строку, после того, как вы введете объявление процедуры.

После того как вы введете исходный код для процедуры **HelloWorld**, выполните ее: поместив курсор вставки в любую строку текста процедуры, щелкните на кнопке **Run** (Запуск).

Когда VBA выполняет процедуру **HelloWorld**, он отображает диалоговое окно, показанное на рис. 2.19. Точный заголовок этого диалогового окна зависит от того, в каком host-приложении (Word, Excel и т.д.) вы создали процедуру. Щелкните на кнопке **OK** для установки диалогового окна в исходное состояние и завершения процедуры.

**Рис. 2.19**  
Результат выполнения макроса **HelloWorld**



Заметьте, что даже в этой короткой процедуре тело процедуры расположено с отступом для отделения его от объявления и конца процедуры. Следует всегда размещать код с отступом, чтобы его было легче читать. Сравните листинг 2.5 со следующими далее строками кода; вы видите, что даже короткие процедуры легче читать, если строки в них расположены с отступом.

```
1: Sub HelloMacro()  
2: MsgBox "Hello, World!"  
3: End Sub
```

### Свойство Auto-Indent (автоотступ)

Текстовый редактор VBA содержит свойство, называемое *автоматический отступ* (*auto indenting*), позволяющее форматировать исходный код с различными уровнями отступа. Если свойство автоматического отступа включено, то всякий раз, когда вы нажимаете на **Enter**, чтобы начать новую строку, курсор вставки на новой строке автоматически перемещается в положение, совпадающее с уровнем отступа строки, расположенной выше. (Нажмите на **Backspace**, чтобы вернуться к предыдущему уровню отступа).

Для включения и выключения автоматического отступа устанавливайте или отключайте флажок **Auto Indent** (автоотступ) на вкладке **Editor** (редактор) диалогового окна **Options** (параметры). Свойство автоматического отступа является включенным по умолчанию.

### Запуск процедуры во время редактирования

Независимо от того, пишете ли вы новую процедуру или редактируете записанный макрос, вам потребуется запускать эту процедуру для проверки результатов ваших усилий. Вы уже знаете, как использовать диалоговое окно **Macros** для запуска макроса или процедуры; вы можете их также запускать непосредственно из модуля во время редактирования. Если курсор вставки в момент выбора элемента меню **Run**, будет находиться на тексте процедуры, которая не может быть выполнена, Редактор VB предложит вам выбрать процедуру из списка возможных процедур.

### Отображение сообщений для пользователя процедуры

Листинг 2.5 включает VBA-оператор **MsgBox**, который можно использовать для отображения процедурами сообщения на экране (рис. 2.19). Сообщения или другая информация, которую процедура отображает на экране, пересылает на принтер или записывает в дисковый файл, называется *выходом* (*output*). Оператор **MsgBox** является простейшей формой экранного выхода, которую вы можете включать в процедуры VBA.

Оператор **MsgBox** подобен встроенной в VBA процедуре. Строка в процедуре **HelloWorld**, содержащая оператор **MsgBox**, сообщает VBA, что необходимо вызвать эту встроенную процедуру. Оператор **MsgBox** из листинга 2.5 приведен ниже еще раз:

```
MsgBox "Hello, World!"
```

Текст, заключенный в кавычки, в этой строке после имени процедуры **MsgBox** — это текст сообщения, которое должно отображаться с помощью **MsgBox**. VBA передает эту информацию процедуре **MsgBox**. Информация, передаваемая процедуре, которую вызывает исходный код, называется *аргументом*.



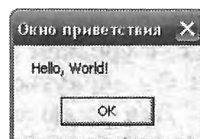
том (*argument*) для этой процедуры. Текст «Hello, World!» является аргументом для процедуры **MsgBox**. Двойные кавычки (") в аргументе «Hello, World!» указывают на то, что текст, заключенный в них, является данными для процедуры, а не инструкциями, которые должен выполнить VBA.

Обратимся еще раз к рисунку 2.19 и заметим, что строка заголовка в диалоговом окне, отображаемом оператором **MsgBox**, содержит слова «Microsoft Word». По умолчанию диалоговое окно, отображаемое оператором **MsgBox**, имеет заголовок, указывающий на *host*-приложение, выполняющее процедуру VBA (в данном случае — Microsoft Word).

Вы можете изменить заголовок диалогового окна, которое отображает **MsgBox**. Листинг 2.6 показывает процедуру **HelloWorld** из листинга 2.5 с оператором **MsgBox**, измененным таким образом, что он отображает диалоговое окно, показанное на рис. 2.20. Сравните рис. 2.19 и 2.20 и обратите внимание, что теперь заголовком диалогового окна **MsgBox** является текст «Окно приветствия».

**Рис. 2.20**

Результат выполнения кода листинга 2.6



---

**Листинг 2.6.** Отображение настраиваемой строки заголовка с помощью **MsgBox**

---

```
1: Sub HelloMacro()  
2:   MsgBox "Hello, World!", , "Окно приветствия"  
3: End Sub
```

---

Оператор **MsgBox** (строка 2 листинга 2.6) теперь отличается от используемого в предыдущем листинге, хотя выполняет ту же самую задачу в процедуре — отображать сообщение для пользователя. В листинге 2.6 оператор **MsgBox** имеет три аргумента (которые следуют после него); каждый аргумент отделяется от других запятой.

Первый аргумент оператора **MsgBox** такой же, как в листинге 2.5, и является текстом, который должен быть отображен посредством **MsgBox** (показанный во всплывающем окне **Quick Info** как аргумент *Prompt*). Поскольку этот вариант оператора **MsgBox** имеет более одного аргумента, за первым аргументом следует запятая; *списки аргументов (argument lists)* в процедурах VBA отделяются запятыми, точно так, как элементы в каком-либо списке.

Как только вы ввели запятую, отделяющую первый аргумент от второго, вы, вероятно, заметили, что всплывающее окно **Quick Info** изменяется: аргумент *Buttons* в окне **Quick Info** выделяется полужирным шрифтом, тогда как аргумент *Prompt* отображается опять обычным шрифтом. В то же время в модуле появляется раскрывающийся список; этот список содержит все возможные значения для аргумента *Buttons*. Появляющийся список является результатом свойства **Auto Data Tips** (подсказка значений данных) Редактора VB. Это свойство работает подобно свойству **Auto Quick Info**, но отображает списки допустимых значений для аргументов процедуры и других элементов в коде VBA.

Второй аргумент оператора **MsgBox** является необязательным; в данном примере необязательный второй аргумент опущен. Поскольку необязательный

аргумент опускается, в списке аргументов имеется один символ пробела. Этот пробел указывает VBA на то, что в этом списке пропущен необязательный аргумент. За символом пробела следует запятая, чтобы отделить его от следующего аргумента в списке. Если вы не введете символ пробела между двумя запятыми, Редактор VB добавляет его (скоро вы заметите, что Редактор VB очень часто добавляет то, что, по его «мнению» нужно добавить, и убирает лишнее, например, пробелы).

Необязательный второй аргумент в операторе **MsgBox** — это аргумент *Buttons*. Он появляется во всплывающем окне **Auto Quick Info**, заключенным в квадратные скобки для указания того, что это — необязательный аргумент. Аргумент *Buttons* определяет, сколько и какого типа командные кнопки появляются в диалоговом окне, отображаемом посредством **MsgBox**. Когда вы опускаете необязательный второй аргумент (как в этом примере), диалоговое окно, которое отображает **MsgBox**, содержит только одну кнопку — кнопку **ОК**. (Подробнее о необязательном аргументе *Buttons* описывается в следующих главах).

Третий и последний аргумент в операторе **MsgBox** определяет заголовок диалогового окна (рис. 2.20). Как только вы вводите запятую, отделяющую второй и третий аргументы в списке аргументов оператора **MsgBox**, всплывающее окно **Auto Quick Info** указывает, что вы теперь должны ввести значение аргумента для аргумента *Title*.

Подобно первому аргументу текст для строки заголовка диалогового окна заключается в кавычки ("). VBA всегда воспринимает заключенный в кавычки текст как данные, а не как текст, содержащий программные инструкции. Если вы опустите кавычки для текста сообщения **MsgBox** или для текста строки заголовка диалогового окна, VBA отобразит сообщение об ошибке. Поскольку третий аргумент является также и последним, никакой запятой после третьего аргумента не требуется.

### Сообщения об ошибках во время написания, редактирования или выполнения процедуры

При написании или редактировании процедуры, вы можете сделать ошибки в операторах. VBA определяет многие из этих ошибок во время написания и редактирования вами исходного кода и может обнаружить некоторые ошибки при выполнении процедуры.

#### Ошибки синтаксиса

*Синтаксисом (syntax)* — называется определенный порядок слов и символов, который образует правильный оператор VBA. Некоторые из наиболее общих ошибок, с которыми вы сталкиваетесь во время написания или редактирования процедур VBA, — это *ошибки синтаксиса (syntax errors)*, например, пропущенные запятые, кавычки, аргументы и так далее.

Всякий раз, когда вы пишете новую строку кода или изменяете существующую, VBA *анализирует (parses)* строку, как только курсор вставки перемещается из новой или измененной строки. [*Синтаксический анализ (parsing)*] — процесс разделения оператора VBA на составляющие части и определение того, какие части строки являются ключевыми словами, переменными или данными; этот процесс подобен анализу предложения в каком-либо языке для определения его составляющих частей — существительных, глаголов, прила-

гательных и так далее]. После выполнения анализа строки кода VBA компилирует эту строку кода. *Компиляция (compiling)* в VBA означает составление исходного кода в форме, которую VBA может непосредственно выполнять без необходимости снова анализировать код.

После того как VBA успешно завершит анализ и компиляцию строки кода в процедуре и не будет обнаружено никаких ошибок, выполнится цветовое кодирование различных частей строки. (Помните, ключевые слова в Редакторе VB отображаются синим цветом, комментарии — зеленым, а данные или другие операторы отображаются в виде черного текста.) Если, однако, VBA обнаруживает ошибку синтаксиса в строке в процессе анализа или компиляции, VBA отображает всю строку красным цветом и выводит на экран диалоговое окно с сообщением об ошибке.

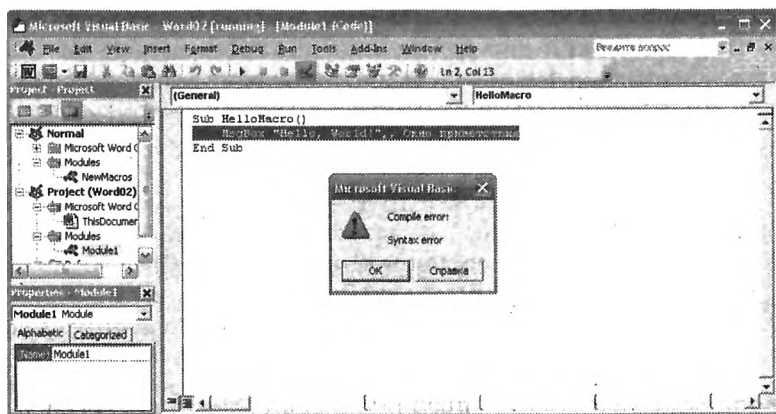
Рассмотрим следующий фрагмент кода, представляющий неправильно написанный оператор с использованием вызова процедуры **MsgBox**:

```
MsgBox "Hello, World!", , Окно приветствия
```

В этом примере кавычки, необходимые вокруг текста, определяющего заголовков диалогового окна, были случайно пропущены. В результате VBA не может определить, что два слова в последнем аргументе являются данными; вместо этого VBA воспринимает слово **Окно** как имя переменной. [*Переменной (variable)* называют место в памяти, используемое для сохранения данных. Переменные описываются в следующей главе.]

Поскольку третий аргумент воспринимается как переменная с именем приветствия, VBA «ожидает» запятую или конец списка аргументов для процедуры **MsgBox**. Вместо этого, VBA находит символ пробела, за которым следует, как это кажется, имя другой переменной. VBA отображает всю строку красным цветом, чтобы указать, что она содержит ошибку, выделяет слово или символ в том месте строки, где, по определению VBA, находится ошибка, и затем отображает диалоговое окно, показанное на рис. 2.21.

**Рис. 2.21**  
Диалоговое  
окно VBA  
с указанием  
ошибки



Если вы получаете подобное сообщение об ошибке синтаксиса или компиляции, щелкните на кнопке **Справка (Help)** для доступа к справочной системе VBA и получения более подробной информации о конкретной встретившейся ошибке синтаксиса, хотя довольно часто справочной информации вам

будет недостаточно для выяснения действительной причины возникновения ошибки.

Для удаления диалогового окна щелкните на кнопке **ОК**. После удаления с экрана диалогового окна с сообщением об ошибке попробуйте устранить ошибку. Вы можете переместить курсор вставки со строки после удаления диалогового окна, но строка остается отображенной красным цветом для указания на то, что в ней содержится ошибка. VBA не анализирует и не компилирует строку, содержащую ошибку синтаксиса снова, пока вы не запустите процедуру или не отредактируете эту строку. VBA анализирует строки в процедуре только тогда, когда вы убираете курсор вставки со строки непосредственно после внесения изменений или когда запускаете процедуру. Если при выполнении процедуры встречается строка с синтаксической ошибкой, выполнение кода прекращается, отображается модуль, содержащий процедуру, в которой имеется ошибка синтаксиса, выделяется конкретная строка в модуле, где обнаружена ошибка, и отображается диалоговое окно с сообщением об ошибке.

Диалоговое окно, которое VBA отображает для ошибки синтаксиса, обнаруженной при выполнении процедуры, обычно содержит гораздо меньше деталей о конкретной ошибке, чем диалоговое окно, которое VBA отображает, когда обнаруживает ошибку синтаксиса после того, как вы напишете или измените строку. По этой причине (и для того, чтобы избежать проблем, вызванных неожиданным прерыванием выполнения процедуры) следует всегда стараться исправлять ошибки синтаксиса, когда VBA их обнаруживает впервые.

VBA обнаруживает многие ошибки синтаксиса, информируя о пропущенных запятых, кавычках и так далее. Однако не всякое сообщение об ошибке такое понятное, как хотелось бы. В некоторых случаях VBA не может определить, что именно неверно в синтаксисе конкретного оператора, а только сообщает об имеющейся ошибке.

### Ошибки времени исполнения

Вы можете создать синтаксически правильный оператор VBA, который все же не выполняется правильно. Ошибки, которые выявляются только при фактическом выполнении процедуры, называются *ошибками времени исполнения* (*runtime errors*) или *runtime-ошибками*. Существуют различные типы таких ошибок. Они обычно вызваны пропуском аргументов процедуры, аргументами неверного типа данных, пропуском ключевых слов, попытками доступа к несуществующим драйверам диска и папкам каталога или ошибками в логике.

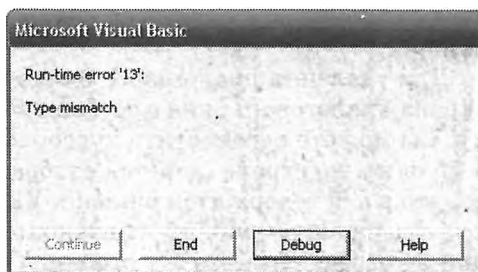
Рассмотрим следующий оператор VBA, который снова содержит неправильный оператор **MsgBox**:

```
MsgBox "Hello, World!", "Окно приветствия"
```

В этом примере VBA не находит ничего неверного в синтаксисе оператора **MsgBox**: текст данных правильно заключен в кавычки и список аргументов правильно разделен запятыми. (Поскольку все аргументы, кроме первого для **MsgBox** являются необязательными, VBA принимает два аргумента для **MsgBox** в качестве правильного синтаксиса.) Однако когда VBA пытается выполнить оператор в этой строке, он отображает диалоговое окно с сообщением об ошибке, показанное на рис. 2.22.

**Рис. 2.22**

Некоторые ошибки VBA обнаруживает во время выполнения макроса



Это диалоговое окно информирует о том, что возникла ошибка при выполнении процедуры, и отображает сообщение, описывающее эту ошибку. В данном случае ошибкой является *ошибка несоответствия типа (type mismatch error)*. Но это не означает, что вы использовали неправильный тип данных, к этому несоответствию могла привести какая-то другая ошибка.

Если вы еще раз посмотрите на вышеприведенный оператор **MsgBox**, то можете заметить, что отсутствует запятая, отмечающая место для необязательного **MsgBox**-аргумента *Buttons*.

Когда VBA анализирует этот оператор, заключенный в кавычки текст «Окно приветствия» компилируется как второй аргумент **MsgBox**, а не как третий аргумент из-за пропущенной запятой. Поскольку аргумент командной кнопки (второй по списку) должен быть числом, а не текстом, VBA сообщает о том, что тип данных, передаваемый процедуре **MsgBox**, не совпадает с типом данных, ожидаемых для этого аргумента. (Более подробно о типах данных рассказывается в следующей главе.)

Диалоговое окно для сообщений об ошибках времени исполнения содержит несколько командных кнопок. Далее описывается каждая из этих кнопок:

- **Continue** (продолжить) — Выбирайте эту командную кнопку для продолжения процедуры. Некоторые ошибки времени исполнения позволяют продолжать выполнение процедуры; однако для большинства таких ошибок эта кнопка отключена.
- **End** (завершить) — Выбирайте эту кнопку, чтобы закончить процедуру.
- **Debug** (отладка) — Выбирайте эту кнопку для перехода в режим прерывания и используйте возможности отладки Редактора VB для отслеживания и решения любой проблемы, вызванной ошибкой времени исполнения.
- **Help** (справка) — Эта команда предоставляет доступ к справочной системе VBA и отображает текст подсказки, описывающий конкретную ошибку времени исполнения, которая возникла. Используйте эту кнопку, чтобы получить больше информации, если вам не понятно, что означает данная ошибка времени исполнения.

Если вы не понимаете, почему использование определенного ключевого слова VBA или процедуры приводит к ошибке времени исполнения, вы можете получить справку по этому конкретному ключевому слову или процедуре, поместив курсор на этом слове и нажав на клавишу **F1**. VBA отображает раздел справочной системы для этого ключевого слова или процедуры, если такой имеется.

### Печать исходного кода

В некоторый момент времени вам может понадобиться распечатать какой-либо исходный код VBA. Может появиться необходимость печати в целях архивирования или документирования, демонстрации или изучения кода. Следует чаще читать и свой код, и код более опытных пользователей. Чтение бумажной копии кода приносит довольно неожиданные результаты — иногда хочется (и следует) просто переписать весь код и надеяться, что старую версию никто, кроме вас, не видел.

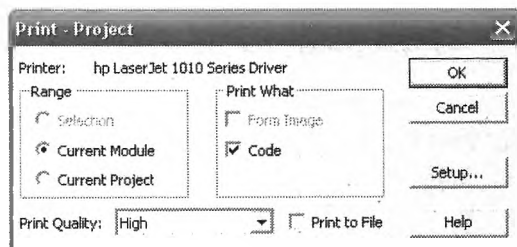
При печати исходного кода VBA можно печатать все модули в проекте или только модуль текущего выбора. Нельзя выделять для печати отдельные процедуры.

Чтобы напечатать исходный код, выполняйте следующие шаги:

1. В **Project Explorer** выделите модуль или проект, который вам необходимо печатать. Если вы хотите печатать только выделенный текст в модуле, выведите модуль на экран и выделите текст, который необходимо печатать.
2. Выберите команду **File | Print** (файл | печать). Редактор VB отображает диалоговое окно **Print** (рис. 2.23).
3. В группе элементов управления **Range** (диапазон печати) укажите, будет ли печататься текст текущего выбора, выделенный модуль или весь проект.
4. В группе элементов управления **Print What** (печатать) должен быть установлен флажок **Code** (программу) (чтобы печатать исходный код); если ваш проект содержит формы, установите флажок **Form Image** для печати рисунка формы.

**Рис. 2.23**

Вы можете задать различные параметры печати в этом окне



5. Установите любые другие параметры печати, какие необходимо, так же, как вы это делаете в любом приложении Microsoft Office.
6. Щелкните на кнопке **OK**, чтобы напечатать исходный код.

Когда вы печатаете проект или модуль, вы не можете предварительно просматривать печатаемый модуль. Отпечатанный листинг из модуля всегда форматируется одним и тем же образом [за исключением ориентации страниц, которую вы можете регулировать, щелкнув на кнопке **Setup** (настройка) в диалоговом окне **Print**].

При разработке проектов, кроме текстов модулей, вы можете выводить на печать графические представления диалоговых окон, похожих на те окна, к которым вы, очевидно, привыкли, работая в среде Windows. Разработке диалоговых окон в VBA будет посвящено несколько глав этой книги.

# Типы данных, переменные, константы и выражения

Из этой главы вы узнаете о типах данных, которыми может управлять VBA, и о том, как в VBA хранятся временные данные. Информация, изложенная в этой главе, применима к VBA в любом host-приложении, реализующем VBA, — Word, Excel, PowerPoint, FrontPage, Visio, Outlook и даже Access. В действительности, информация в этой главе также применима к Visual Basic, и все, что вы узнаете о типах данных, переменных и константах, справедливо в любой из реализаций Visual Basic (до версии .NET, которая, впрочем, совсем не является даже отдаленно напоминающей версию Visual Basic), в которых вы будете программировать.

Перед тем, как изучить категории констант и переменных, необходимо понять, какие виды информации сохраняются в памяти, выделяемой для приложения.

Как и подавляющее большинство систем программирования, Visual Basic разделяет обрабатываемые данные на числа, текст, даты и другие типы. В таблице 3.1 приведены типы данных, используемые в Visual Basic.

*Байт (byte)* — это единица измерения компьютерной и дисковой памяти. Байт состоит из восьми *битов (bits)* или двоичных разрядов. Обычно один алфавитный символ требует для хранения одного байта памяти

Таблица 3.1. Типы данны в VBA

Название типа	Размер в байтах	Описание и диапазон значения
<b>Byte</b>	1	Для хранения положительного числа от 0 до 255.
<b>Boolean</b>	2	Для хранения логических значений; может содержать только значения <b>True</b> или <b>False</b>
<b>Currency</b>	8	От -922337203685477.5808 до 922337203685477.5807.
<b>Date</b>	8	Для хранения комбинации информации о дате и времени. Диапазон дат может быть от 1 января 100 года до 31 декабря 9999 года. Диапазон времени от 00:00:00 до 23:59:59.





## Экспоненциальное представление

В таблице 3.1 вы встретились с представлением данных, называемым *экспоненциальным представлением* (*scientific notation*), которое используется для отображения на внешних устройствах (монитор, принтер и т.д.) очень больших и очень малых чисел в компактном формате. В экспоненциальном представлении значения записываются без начальных и конечных нулей и слева от десятичного знака имеется только одна цифра. Число умножается на 10 в некоторой степени, чтобы показать, где на самом деле находится десятичный знак. Экспоненциальное представление компактно и читабельно по сравнению с обычным представлением числа с 300 нулями после него.

Помните, что отрицательная степень приводит в результате к меньшему числу, а положительная — к большему. В исходном коде невозможно использовать *надстрочные* (*superscript*) символы, поэтому VBA использует вариацию экспоненциального представления, разработанную специально для компьютеров. В экспоненциальном представлении VBA используйте букву *E* с последующей степенью. В следующей таблице приводятся примеры чисел в экспоненциальном и обычном представлении.

-2.43E2	-243
-1.43E-2	-0.0143
5.5E10	55,000,000,000
3E9	3,000,000,000
4.5E-10	0,000000000045
1.6E2	160

Большинство типов данных, приведенных в таблице 3.1, можно преобразовывать в другие типы. Далее в этой главе описывается, как VBA автоматически преобразовывает данные (и как вы сами можете это делать).

### Тип Date

VBA использует тип **Date** для хранения дат и времени. Тип **Date** использует восемь байтов памяти для каждой сохраняемой комбинации дата/время. При работе с данными типа **Date** следует знать, что VBA-типы **Date** не являются одними и теми же, что типы данных, используемые в рабочих листах Excel или базах данных Access, хотя они во многом похожи.

Нет необходимости беспокоиться о том, как VBA хранит данные типа **Date**, — можно просто отображать, сохранять или манипулировать датами; VBA автоматически управляет всеми деталями преобразования последовательного числа в год, месяц, день и время. Когда VBA отображает даты, (например, при применении **MsgBox**), дата представляется в коротком формате, который используется данным компьютером. Аналогично, VBA отображает информацию о времени, сохраняемую с **Date**, используя 12- или 24-часовой временной формат для данного компьютера. (В Windows 98/NT 4.0/2000 вы можете изменять форматы дат и времени, выбрав значок **Язык и стандарты** на **Панели управления**.)

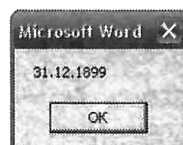
VBA-тип **Date** является типом *последовательных дат* (serial Dates). (Последовательные даты сохраняют дату как число дней от заданной стартовой даты.) Базовой датой для VBA-типа **Date** является 30 декабря 1899. VBA использует отрицательные числа для представления дат ранее 12/30/1899 и положительные — для дат после 12/30/1899. Число 0 представляет саму дату 12/30/1899. По этой схеме 1 января 1900 записывается числом 2 (1/1/1900 — это 2 дня после 12/30/1899), однако число -2 является датой 12/28/1899 (два дня до 12/30/1899). Чтобы убедиться в этом, напишите простой макрос (в VB-редакторе Word или Excel), приведенный в листинге 5.2 (Для запуска макроса используйте меню Run):

### Листинг 3.1. Тестирование типа **Date**

```
Sub DateTest()  
    Dim d As Date  
    d = 1  
    MsgBox d  
End Sub
```

В результате работы этого макроса на экран будет выдана базовая дата, как показано на рис. 3.1.

**Рис. 3.1**  
Базовая дата для VBA-типа **Date**



Некоторые host-приложения VBA, такие как Excel, имеют свои собственные типы данных последовательных дат. Excel, например, также использует числа последовательных дат, но с другой базовой датой: 1 января 1900.

VBA всегда использует одну и ту же базовую дату для последовательных дат типа **Date**, независимо от схемы дат host-приложения — VBA всегда использует дату 12/30/1899 в качестве базовой.

### Замечание

Чтобы проверить, какую базовую дату использует Excel, отформатируйте какую-либо ячейку листа Excel как дату и введите в эту ячейку 1 (единицу). Как только вы перейдете к другой ячейке листа, в предыдущей вместо 1 будет отображена базовая дата.

В значении последовательной даты целая часть (цифры слева от десятичного знака) — это общее число дней от базовой даты. Последовательная дата VBA может иметь цифры справа от десятичного знака; эти цифры обозначают время дня как часть дня. Один час — это 1/24 дня (приблизительно 0,0416. Аналогично, одна минута — это 1/1440 дня, а секунда — 1/86400 дня.

Вы можете вычитать одну дату из другой, добавлять к дате или вычитать числа для изменения ее значения. Например, если вам необходимо определить количество дней между двумя датами, просто вычитайте более раннюю дату из

более поздней. Поскольку это значения типа **Date**, VBA «знает», что целью вычисления является получение разности в днях между двумя этими датами. Аналогично, если вам необходимо определить дату через 60 дней после данной даты, просто прибавьте 60 к этой дате.

В VBA имеется несколько встроенных процедур (описанных в следующих главах) для отдельного извлечения года, месяца, дня, часов, минут и секунд из переменной типа **Date**.

## Числа

VBA имеет шесть различных численных типов данных: **Byte**, **Integer**, **Long**, **Single**, **Double** и **Currency**. Численные типы данных используются для хранения (и манипулирования) чисел в различных форматах, в зависимости от конкретного типа. Численные типы предоставляют компактный и эффективный способ хранения чисел. Численный тип, заполняющий большую часть памяти (имеющий самый большой диапазон возможных значений), занимает не более восьми байтов памяти для хранения чисел, которые могут иметь до 300 цифр.

### Типы данных **Byte**, **Integer** и **Long**

**Integer** — это целое число безо всякой дробной части. Числа 1, 3768 и 12 — это все целые числа; числа 1,5; 3,14; 17,2<sup>1</sup> не являются целыми. (Число 1,0 — не целое, хотя дробная часть равна нулю; целые никогда не содержат десятичного знака, даже если десятичная часть является нулем).

VBA предоставляет три различных целых типа данных: **Byte**, **Integer** и **Long**. Тип **Byte** — это наименьший из трех целых типов данных VBA и предназначен для хранения чисел только от 0 до 255. Тип **Byte** использует только один байт памяти (255 — это самое большое число, какое вы только можете получить с восемью битами). Тип **Byte** не предназначен для хранения отрицательных чисел. Обычно типы **Byte** используются для хранения двоичных данных (графических, звуковых файлов и так далее).

Тип **Integer** требует двух байтов памяти; диапазон чисел, которые вы можете хранить как тип **Integer**, составляет от -32768 до 32767. Поскольку диапазон для типа **Integer** сильно ограничен, в VBA имеется другой, больший тип: **Long** — (длинное) целое. Этот тип для хранения числа использует 4 байта памяти и имеет диапазон значений от -2147483648 до 2147483647.

Типы **Byte**, **Integer** и **Long** имеют пару преимуществ по сравнению с другими численными типами данных: целые требуют меньше памяти для хранения чисел, чем другие численные типы данных VBA, а математические операции и операции сравнения над числами типов **Byte**, **Integer** и **Long** быстрее, чем эти же операции для численных типов данных с плавающей точкой. Типы **Byte**, **Integer** и **Long** имеют различное применение. Наиболее часто они используются для кумулятивных вычислительных операций вследствие их компактного размера и более быстрой скорости выполнения арифметических операций. Используйте типы **Byte**, **Integer** и **Long** для чисел, которые не имеют дробной части.

VBA автоматически преобразует данные типов **Byte**, **Integer** и **Long** в текст, когда вы выводите их на экран с использованием таких процедур, как **MsgBox**.

<sup>1</sup> Здесь и далее для обозначения действительных чисел используется знак запятой (,) поскольку имеется в виду обычная математическая запись числа.

### Числа с плавающей точкой

Числа с плавающей точкой (*floating point numbers*) могут иметь любое число цифр до или после десятичной точки (в пределах границ конкретного типа данных) и получили свое название вследствие того факта, что десятичная точка «плавает» из одной позиции в другую, в зависимости от того, сохраняется в памяти большое или малое значение. Числа 11,0123; -1107,1; 0,0125 и 435,67876 — это числа с плавающей точкой. Числа с плавающей точкой иногда называют также *действительными* (*real*) числами. Используйте типы данных с плавающей точкой всякий раз, когда вам требуется сохранить число, имеющее дробную часть.

VBA имеет два различных типа данных с плавающей точкой: **Single** и **Double**. Для данных типа **Single** требуется 4 байта памяти, и этот тип может использоваться для хранения отрицательных чисел от  $-3,402823 \times 10^{38}$  до  $-1,401298 \times 10^{-45}$  и положительных от  $1,401298 \times 10^{-45}$  до  $3,402823 \times 10^{38}$ . Числа, хранимые с использованием типа **Single**, называются *числами одинарной точности* (*Single-precision numbers*). Для данных типа **Double** требуется 8 байтов памяти, и этот тип может использоваться для хранения отрицательных чисел от  $-1,79769313486232 \times 10^{308}$  до  $-4,94065645841247 \times 10^{-324}$  и положительных от  $4,94065645841247 \times 10^{-324}$  до  $1,79769313486232 \times 10^{308}$ . Числа, хранимые с использованием типа **Double**, называются *числами двойной точности* (*double-precision numbers*).

Хотя числа одинарной и двойной точности имеют большие диапазоны, чем другие численные типы данных, у них имеется два небольших недостатка. Операции, выполняемые над числами с плавающей точкой, немного медленнее подобных операций над другими численными типами данных (впрочем, проблемы быстродействия при выполнении арифметических операций не являются самыми частыми при работе Windows-приложений). Кроме того, числа, хранимые как типы данных с плавающей точкой, могут быть подвержены ошибкам округления. Как и в случае с целыми типами данных, VBA автоматически преобразует значения типа **Single** и **Double** в текст, когда вы отображаете их с помощью процедур, таких как **MsgBox**. Если число с плавающей точкой очень большое или очень малое, VBA отображает его в экспоненциальном представлении.

### Тип данных Currency

VBA-тип **Currency** — это *число с фиксированной точкой* (*fixed-point number*), то есть, десятичная точка всегда находится в одном и том же месте — справа от десятичной точки всегда имеется четыре цифры. Используйте тип **Currency** для хранения чисел, когда точность крайне важна, что бывает при «денежных» вычислениях.

Тип **Currency** требует 8 байтов памяти и может сохранять числа от -922337203685477,5808 до 922337203685477,5807. Математические операции над числами типа **Currency** имеют небольшие или не имеют ошибок округления, поэтому они точнее, чем операции над числами с плавающей точкой. Ошибки округления с числами типа **Currency** обычно возникают только тогда, когда вы умножаете или делите числа типа **Currency** на значения другого численного типа. Подобно обработке всех других численных типов данных VBA автоматически преобразует значения **Currency** в текст, когда вы их выводите на экран.

## Текстовые строки

Любые текстовые данные, сохраняемые в программе VBA, называются *строками (strings)*. Строки в VBA сохраняются с использованием типа данных **String**. Строки получили такое название, потому что текстовые данные обычно рассматриваются как строки символов. Строка может содержать текстовые символы любых типов: буквы алфавита, цифры, знаки пунктуации или различные символы. Строки в коде VBA всегда заключаются в двойные кавычки (""). "**Вася**", "**Петр Первый**", "**3,141**" и "**1 000,00**" — это строки. Существуют две категории строк: строки переменной длины, размер которых растет или уменьшается, и строки фиксированной длины, размер которых всегда остается одним и тем же. Все строки в VBA являются строками переменной длины, если только вы не зададите фиксированную длину (об этом будет рассказано далее в этой главе).

Типы **String** играют важную роль во многих программах VBA (и на любых других языках программирования). Большинство данных ввода пользователей (в диалоговых окнах, ячейках рабочих листов) — это строковые данные. Кроме того, поскольку вы можете отображать на экране только текст, все другие типы данных должны быть преобразованы в строковые данные перед тем, как вы сможете их вывести на экран. Многие встроенные процедуры VBA (подобные **MsgBox**) используют строковые данные во всех или в некоторых своих аргументах. Если вы широко используете Word VBA, то могли заметить, что процедуры манипулируют исключительно строковыми данными.

VBA предоставляет несколько операторов для *конкатенации (concatenate)*, то есть для соединения вместе и для сравнения строк. VBA имеет также несколько встроенных процедур, помогающих извлекать подстроки из более длинных строк, находить символы или слова в строке, изменять регистр букв в строке и так далее.

Очень важно научиться легко обращаться со строковыми данными и использовать функции обработки строк. В следующих главах описываются строковые операторы VBA и процедуры для манипулирования строками VBA.

## Логические значения

Обычно VBA-программа (как и любые другие программы) «принимает» решения, проверяя, являются ли истинными различные условия. Для упрощения тестирования условий и обеспечения сохранения результатов такого тестирования в VBA имеется логический тип данных. Логические значения **True** и **False** называют *булевыми (Boolean)* значениями. (Их название связано с именем математика, разработавшего систему математической логики.) Логический тип данных VBA называют также типом **Boolean**.

**Boolean**-тип VBA требует двух байтов памяти и может иметь одно из двух значений: **True** или **False**. Если вы отображаете тип **Boolean** на экране, VBA автоматически преобразует его в строку, содержащую либо слово **True**, либо **False**. Булевы значения получают как результат операции сравнения. *Операция сравнения (comparison operation)* имеет место при сравнении одного с другим, например, при сравнении двух чисел для определения, которое из них больше, или сравнении двух строк для определения, которая из строк находится ниже в алфавитном порядке.

## Тип данных Variant

Тип данных **Variant** — это особый тип данных, который может сохранять любые типы, приведенные в табл. 4.1, за исключением типа **Object**. VBA использует тип **Variant** для всех переменных, если только вы не объявляете явно тип переменной, как будет описано далее в этой главе.

Данные типа **Variant** принимают характеристики определенного типа, который они сохраняют в данный момент. Например, если данные типа **Variant** содержат строковые данные, **Variant** принимает характеристики типа **String**. Если данные типа **Variant** содержат численные данные, **Variant** принимает характеристики какого-либо численного типа, обычно — **Double**, хотя типы **Variant** могут также иметь характеристики типов **Integer**, **Long**, **Single** или **Currency**.

VBA использует для данных типа **Variant** наиболее компактное представление, возможное для конкретных значений, находящихся в данных. Если VBA сохраняет целое число в переменной типа **Variant**, это число обрабатывается как данные типа **Integer** или **Long**, в зависимости от его размера. Например, для VBA число 15 в типе **Variant** является данными типа **Integer**; VBA обрабатывает число 1 000 000 в **Variant** как данные типа **Long**.

VBA сохраняет большинство чисел с плавающей точкой в данных типа **Variant** как тип **Double** и использует тип **Variant** так, что вам не всегда приходится беспокоиться об указании типов переменных, которые вы используете в коде. Любые переменные, для которых вы не объявляете специально тип, становятся типом **Variant**.

Несмотря на то, что типы **Variant** удобны и избавляют от некоторой части работы при написании процедур, они требуют большего объема памяти, чем любой другой тип данных, за исключением больших строк. Кроме того, математические операции и операции сравнения над данными типа **Variant** выполняются медленнее, чем подобные операции над данными любого другого типа. В общем, следует избегать использования переменных **Variant**: если вы будете полагаться на переменные типа **Variant**, может сформироваться привычка неаккуратного программирования и вам будет трудно находить и устранять ошибки в программах.

## Переменные

Переменные очень важны, поскольку они обеспечивают возможность в компьютерной программе временно сохранять и манипулировать данными. В этом разделе описывается, что такое переменная и как создавать переменные.

### Что такое переменная?

*Переменная (variable)* — это имя, которое разработчик программы может дать области компьютерной памяти, используемой для хранения данных какого-либо типа. Вы можете представлять себе переменную как ящик, в который можно положить любой отдельный элемент данных и хранить его для последующего использования. Имя переменной является идентифицирующей меткой для этого ящика. Содержимое ящика (значение переменной) может ме-

няться, но наименование ящика (имя переменной) остается тем же (пример с ящиком хорошо подходит к описанию переменный типа **Variant**: в ящик можно положить, насыпать, налить все, что угодно). Переменные VBA могут хранить любые типы данных, перечисленные в таблице 3.1.

В некотором смысле переменная подобна именованной ячейке рабочего листа. В Excel вы можете давать имя ячейке в рабочем листе, а затем ссылаться на эту ячейку по ее имени так, что вам не приходится использовать или даже запоминать фактический адрес ячейки из строки и столбца каждый раз, когда необходимо сослаться на данные или формулы в этой ячейке. VBA (как и все системы программирования) справляется со всеми деталями нахождения конкретных мест в памяти компьютера вместо вас. Нет необходимости беспокоиться о конкретных адресах данных какой-либо переменной, просто используйте имя этой переменной для ссылки на сохраняемые данные.

Переменные в операторах VBA используются таким же образом, что и переменные в алгебраических уравнениях. Переменная представляет числа, текстовые данные или другую информацию, которая точно не известна во время написания оператора, но будет в наличии и доступна при выполнении этого оператора.

Всякий раз, когда имя переменной появляется в операторе, VBA вставляет в этот оператор фактическое значение, хранимое в текущий момент в участке памяти, на который ссылается переменная. В следующем примере, если переменная **AnyNum** содержит число **2**, то весь оператор будет иметь в качестве результата **4**:

```
AnyNum + 2
```

В качестве другого, более сложного примера, рассмотрим формулу, вычисляющую, какую часть в процентах одно число составляет от другого:

```
Percent = (Part / Whole) * 100
```

В этом операторе переменные с именем **Part** и **Whole** представляют два различных числа. Формула в этом операторе вычисляет, какую часть составляет значение, хранимое в переменной с именем **Part**, от значения, хранимого в переменной с именем **Whole**, а в переменную с именем **Percent** помещается результат вычисления. При выполнении этого оператора VBA находит все числа, хранимые в памяти, на которые ссылаются **Part** и **Whole**, и вставляет их в оператор перед выполнением любых вычислений. Если, например, до начала выполнения этого оператора значение, хранимое в переменной под именем (можно просто значение переменной) **Part**, было равно **5**, а значение переменной **Whole** — **20**, то результатом, который будет сохранен в переменной **Percent**, станет число **25**.

## Выбор имен для переменных

Можно выбирать любое имя для переменной за небольшими исключениями. В этом разделе сначала объясняются правила и ограничения VBA для имен переменных, а затем описываются некоторые принципы, которыми следует руководствоваться при выборе имен переменных.

## Идентификаторы

*Идентификатор (identifier)* — это имя, которое вы даете элементам в создаваемых вами процедурах и модулях, таким как переменные. Термин идентификатор основывается на том факте, что имена, которые вы создаете, определяют конкретные участки памяти (в случае имени переменной), группы инструкций (в случае имени макроса или процедуры) или другие элементы программы.

При выборе имени переменной необходимо соблюдать следующие правила:

- ☐ Имя переменной должно начинаться с буквы алфавита.
- ☐ После первой буквы имя переменной может состоять из любой комбинации цифр, букв или символов подчеркивания (\_).
- ☐ Имена переменных не могут содержать пробелы, точку (.) или любой другой символ, который VBA использует для обозначения математических операций и операций сравнения (=, +, - и так далее).
- ☐ Имена переменных не могут превышать 255 символов.
- ☐ Имя переменной не может дублировать определенные ключевые слова VBA. Если вы выберете имена переменных, дублирующие ключевые слова, называемые *ограниченными ключевыми словами (restricted keywords)*, VBA отображает одно из нескольких возможных сообщений об ошибке синтаксиса.
- ☐ Имя переменной должно быть уникальным в рамках его *области действия (scope)*. То есть, имя переменной должно быть уникальным в пределах процедуры или модуля, в котором вы объявляете эту переменную.

Некоторые примеры правильных имен переменных:

- ☐ Var12
- ☐ Birth\_Date
- ☐ Ald\_Item
- ☐ Percent
- ☐ Whole
- ☐ Part
- ☐ Line12

Примеры неправильных имен переменных:

- ☐ New Item — Неправильно, потому что содержит символ пробела
- ☐ 5thDimension — Не начинается с буквы
- ☐ Dim — Дублирует ключевое слово VBA
- ☐ Week/Day — Содержит неверный символ: VBA интерпретирует косую черту как оператор деления
- ☐ \_Pay — Содержит недопустимый символ в начале.

Имена переменных не «чувствительны» к состоянию регистра (*not case-sensitive*), то есть, написано ли имя переменной прописными или заглавными буквами не имеет значения. Имена **FirstMyVar** и **firstmyvar** представляют для VBA одно и то же. VBA регулирует написание имен переменных в вашем коде на основе заглавных букв, которые вы использовали в последний раз, когда



вводили имя той или иной переменной. Например, если вы вводите имя **FirstMyVar**, а затем позже — имя переменной **firstmyvar**, VBA заменяет все вхождения имени **FirstMyVar** на **firstmyvar**.

При выборе имен переменных старайтесь делать их по возможности наиболее информативными: выбирайте имена, подобные **AllSum**, а не **x** или **y**. Хорошим именем для переменной, которая хранит значение, представляющее температуру в градусах Цельсия, является, например, **CelsiusTemp** или **DegreesC**. В примере с формулой процентного отношения из предыдущего раздела имена переменных **Part** и **Whole** были выбраны так, чтобы отражать назначение и использование чисел, которые они хранят.

Если вы будете разрабатывать программный продукт в группе разработчиков, выбор наименований переменных (функций и других программных единиц) будет определяться степенью взаимодействия между участниками проекта: чем больше взаимодействие, тем меньше свободы при выборе переменных.

## Создание переменных

Самым простым способом создания переменной является использование ее в операторе VBA. VBA создает переменную и резервирует память для ячейки памяти переменной, когда эта переменная в первый раз появляется в операторе (обычно в операторе, сохраняющем значение данных в переменной).

Сохранение значения данных в переменной называется *присваиванием переменной* (*assigning the variable* или *making an assignment*). Присваивание выполняется с помощью оператора присваивания, представляемого знаком (=). Следующая строка является примером присваивания значения переменной:

```
MyVar = 25
```

Этот оператор сохраняет численное значение **25** в ячейке памяти, заданной именем переменной **MyVar**. Если оператор первым использует данную переменную в процедуре, то VBA создает эту переменную, резервирует ячейку памяти для хранения данных этой переменной и затем сохраняет число **25** в ячейке памяти, заданной именем переменной.

Если переменная **MyVar** уже существует, то VBA сохраняет число **25** в ячейке памяти, на которую ссылается **MyVar**. Новое значение заменяет любое, ранее сохраненное в **MyVar** значение, и предыдущее содержимое **MyVar** терется.

Создание переменной путем ее использования в операторе называется *неявным объявлением переменной* (*implicit variable declaration*). Используя переменную в операторе, вы неявно указываете (объявляете) VBA, что хотите создать эту переменную. Все переменные, которые VBA создает неявным объявлением переменной, имеют тип данных **Variant**. Неявное объявление переменных известно также как объявление переменных «на лету» (*on-the-fly*).

Неявное объявление переменных удобно, но имеет потенциальные проблемы. Например, когда вы уже использовали переменную с именем **MyVar** и делаете позже ошибку в имени, набирая **MVar**. В зависимости от того, где появится в вашем коде неправильное имя переменной, VBA может выдать ошибку времени исполнения или просто создать новую переменную. Если VBA создаст новую переменную, у вас могут возникнуть проблемы, которые очень трудно обнаружить.

Неявное объявление переменных вызывает также проблемы, если вы записываете оператор присваивания, ошибочно полагая, что неявно объявляете новую переменную, тогда как вы на самом деле используете ранее созданную. В этом случае вы нечаянно уничтожаете ранее сохраненное значение. Это обычно не приводит к ошибке времени исполнения, но вызывает другие проблемы, причину которых трудно обнаружить.

Вероятно, по этим или другим причинам VBA предоставляет вам возможность выполнять *явное* (*explicit*) объявление переменных. Явное объявление переменных имеет следующие преимущества:

- Ускоряется выполнение кода (что, конечно, для начинающего программиста не является определяющим). VBA создает все объявленные явно переменные в модуле или процедуре перед выполнением кода процедуры. Скорость выполнения кода увеличивается на то количество времени, которое иначе необходимо для анализа и создания неявно объявляемых переменных.
- Уменьшается количество ошибок в результате неправильного написания имени переменной.
- Код становится более читабельным и понятным. Видя все объявления переменных в начале модуля или процедуры, пользователь может легко определить, какие переменные используются в этом модуле или процедуре.
- С помощью явного объявления переменных можно нормализовать выделение заглавными буквами в имени переменной. Если вы явно объявляете переменную, VBA всегда изменяет заглавные буквы в имени переменной в операторе VBA в соответствии с именем в объявлении переменной. Например, если вы явно объявляете переменную с использованием заглавных букв в имени **MyValue**, а затем позже введете имя этой переменной как **myvalue**, VBA заменяет **myvalue** на **MyValue** для соответствия выделению заглавными буквами в явном объявлении:

MyValue

Для явного объявления переменных используйте VBA-оператор **Dim** со следующим синтаксисом:

### Синтаксис

---

```
Dim name1 [, name2]
```

*nameN* — это любой допустимый идентификатор переменной. Например:

```
Dim PcntProfit
```

```
Dim PcntProfit, Gross_Income, Total_Costs
```

Первый из вышеприведенных операторов указывает VBA создать переменную с именем **PcntProfit**. (Ключевое слово **Dim**, между прочим, является сокращением слова **Dimension**). Все переменные, которые вы создаете с этой формой ключевого слова **Dim**, являются переменными типа **Variant**.

---

Всякий раз, когда VBA создает новую переменную, эта переменная инициализируется: строки инициализируются пустыми строками (не содержат символов), числа — значением 0, переменные **Boolean** — значением **False**, даты — 30 декабря 1899 года.

Если вы, например, выполните код

```
Sub ss()  
    MsgBox tmp  
End Sub
```

то в результате получите пустое окно функции **MsgBox**. Если же вы перед функцией поместите оператор

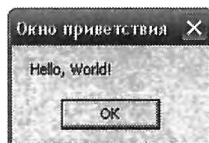
```
Dim tmp as Integer
```

в окне функции **MsgBox** вы увидите, что переменная **tmp** имеет значение **0**. Отметим все же, что опытные программисты не очень полагаются на подобную помощь систем программирования при описании переменных и чаще сами инициализируют используемые ими переменные.

Переменную следует объявлять только один раз в отдельной процедуре или модуле. Так как оператор **Dim** находится перед любыми операторами, фактически использующими переменную, вы можете помещать его в любом месте в процедуре. В практике программирования хорошим правилом является собирать все явные объявления переменных в одном месте в начале процедуры.

В листинге 3.2 приведена ранее рассматриваемая процедура **HelloWorld**, модифицированная для явного объявления переменной с именем **HelloMsg**, которую она использует для хранения текста, отображаемого оператором **MsgBox**. (Процедура **HelloWorld** в листинге 3.2 отображает диалоговое окно, показанное на рис. 3.2).

**Рис. 3.2**  
Окно сообщений, полученное процедурой **HelloWorld**



### Листинг 3.2. Процедура HelloWorld с явным объявлением переменной

```
1: Sub HelloWorld()  
2:   Dim HelloMsg           ' переменная для MsgBox  
3:   HelloMsg = "Hello, World!"  
4:   MsgBox HelloMsg, , "Окно приветствия"  
5: End Sub
```

Оператор **Dim** (в строке 2) листинга 3.1 объявляет переменную **HelloMsg** и резервирует для нее область памяти (в данной подпрограмме **HelloMsg** — это переменная типа **Variant**). Строка 2 включает конечный комментарий, указывающий назначение этой переменной. В строке 3 выполняется присваивание переменной **HelloMsg** строки «Hello, World!». Далее (в строке 4) переменная **HelloMsg** используется как один из аргументов процедуры **MsgBox**. Функция **MsgBox** отображает то же окно сообщения, что и раньше. Хотя **MsgBox** получает теперь свой первый аргумент из переменной, эта переменная содержит ту же строковую информацию, что была ранее записана непосредственно в оператор **MsgBox**.

Заметьте, что в результате мы получили окно сообщений, которое не несет информации о том, в каком приложении работает исходный макрос (ранее эта информация находилась в заголовке окна).

## Задание типа данных переменной

Все переменные в VBA, независимо от того, объявляются ли они неявно или явно, являются переменными типа **Variant**, если только вы не зададите тип переменной в объявляющем ее операторе. Объявление типизированных переменных (переменных, тип данных которых вы задаете) имеет несколько преимуществ:

- Применение типизированных переменных ускоряет выполнение кода программ. Поскольку при объявлении вы указываете VBA тип переменной, выполнение программы ускоряется на то время, которое VBA потратил бы иначе на анализ переменной типа **Variant**, чтобы определить ее конкретный тип.
- Применение типизированных переменных повышает эффективность кода. Переменные типа **Variant** могут занимать гораздо больше памяти, чем переменные определенных типов. Типизированная переменная занимает только память, необходимую для этого определенного типа. Использование типизированных переменных может значительно сократить объем памяти, требуемый для вашей программы VBA; в некоторых случаях использование типизированных переменных может иметь значение с точки зрения того, достаточно ли памяти для работы вашей процедуры или нет.
- Программный код с типизированными переменными легче читать и понимать.
- В программах с типизированными переменными легче обнаруживать некоторые типы ошибок программистов, например, несоответствие типов данных, потому что VBA отображает ошибки времени исполнения, которые не возникли бы, если бы переменные были нетипизированными.

Существуют другие причины, по которым вам может понадобиться объявлять типизированные переменные. Например, несмотря на то, что переменные типа **Variant** могут сохранять даты, переменная типа **Variant** может неправильно сохранить дату из Excel. С переменными, объявленными явно с типом **Date**, такой проблемы не возникает.

Тип переменной объявляется в том же операторе, который используется для объявления самой переменной. Можно объявлять типизированную переменную либо при неявном объявлении, либо при явном — с помощью **Dim**. При объявлении переменных типа **String**, можно также задавать длину строки.

### Использование **Dim** для объявления типизированных переменных

Для объявления переменной и ее типа с помощью оператора **Dim** добавьте ключевое слово **As** перед именем переменной, а затем введите имя типа данных для этой переменной.

### Замечание

---

Как только вы вводите ключевое слово **As** при объявлении переменной, редактор VB предлагает список типов переменных, который зависит от подключенных к проекту библиотек.

---

Вот общий синтаксис для использования оператора **Dim** при объявлении типизированных переменных:

**Синтаксис**

```
Dim varname1 [As type1] [, varname2 [As type2]...] [As type]
```

*VarnameN* представляет любое допустимое имя переменной VBA, а *type(N)* — любое из имен типов данных VBA.

Следующие строки показывают примеры правильного синтаксиса для объявлений типизированных переменных:

```
Dim Счетчик As Single
Dim Цена As Currency
Dim ДатаПлатежа As Date
Dim Сообщение As String
Dim Counter As Integer
Dim PcntProfit As Single, Gross_Sales As Currency, Message As String
Dim NetValue, PcntProfit As Single
```

После объявления типизированной переменной, независимо от того, объявляется ли эта переменная явно или неявно, и как задается тип, эта переменная сохраняет тот же самый тип столько времени, сколько она существует. Нельзя переобъявить переменную или переопределить ее тип.

**Использование символов определения типа для объявления типизированных переменных**

При неявном объявлении можно также задавать тип переменной, добавляя специальный символ, называемый *символом определения типа* (*type-definition character*), в конец имени переменной. В табл. 4.2 приводятся символы определения типа VBA и типы, которые они обозначают.

**Таблица 3.2. Символы определения типа**

Символ определения	Тип
!	Single
@	Currency
#	Double
\$	String
%	Integer
&	Long

Заметьте, что в этой таблице приведены только шесть символов определения типа; нет символов определения типа для **Byte**, **Boolean**, **Date**, **Object** или **Array**. Символы определения типа могут находиться только в конце имени переменной.

Символы определения типа в VBA имеют свою историю. В ранних разновидностях языка BASIC символы определения типа были единственным способом задать тип переменной. Несмотря на то, что следует знать, что такое символы определения типа и как они используются, необходимость в их применении возникает редко — использование оператора `Dim` с ключевым словом `As` намного легче и понятнее. Большинство программистов, работающих на VBA, не используют символы определения типа.

В листинге 3.3 приведен еще один вариант процедуры **HelloWorld**, которая обсуждалась ранее.

### Листинг 3.3. Явное и неявное объявление типа

```
1: Sub HelloWorld()  
2:   Dim HelloMsg As String  
3:   HelloMsg = "Hello, World!"  
4:   Title$ = "Окно приветствия"  
5:   MsgBox HelloMsg, , Title$  
6: End Sub
```

Эта версия процедуры **HelloWorld** работает во многом так же, как и предыдущие. Строка 1 содержит объявление процедуры. В строке 2 оператор `Dim` явно объявляет переменную **HelloMsg**. Поскольку оператор `Dim` включает ключевое слово `As` и имя типа **String**, переменная **HelloMsg** имеет тип **String**. Строка 3 присваивает текст сообщения строковой переменной **HelloMsg**.

Строка 4 неявно объявляет переменную **Title\$** и в то же время присваивает переменной текст заголовка окна сообщения. Поскольку имя переменной **Title\$** имеет на конце символ определения типа для строки, эта переменная также имеет тип **String**. Наконец, строка 5 использует оператор **MsgBox** для отображения окна сообщения; в этом операторе и текст сообщения, и строка заголовка окна являются переменными: **HelloMsg** и **Title\$**, соответственно.

После добавления символа определения типа к переменной необходимо включать символ определения типа каждый раз, когда вы используете имя переменной. Если бы символ определения типа был опущен в имени переменной **Title\$** в строке 5 (где **Title\$** используется первый раз после неявного типизированного объявления), то во время выполнения процедуры произошла бы runtime-ошибка.

#### Использование `Dim` для объявления строковых переменных фиксированной длины

Независимо от того, объявляются ли переменные типа **String** с помощью оператора `Dim` или добавлением символа определения типа `$`, создаваемые вами строковые переменные по умолчанию являются строками переменной длины.

Строковые переменные переменной длины изменяют длину, в зависимости от длины строки, сохраняемой переменной. Иногда может понадобиться использовать строку *фиксированной длины* (*fixed-length*). Строки фиксированной длины всегда имеют одну и ту же длину. Они полезны, если необходимо обеспечить, чтобы текст, сохраненный в строковой переменной, всегда содержал одно и то же число символов.

Можно использовать строки фиксированной длины, например, для выравнивания информации в столбцах при ее отображении или для обеспечения того, чтобы строковые данные, сохраняемые в отдельной переменной, никогда не превышали определенной длины. Существует только один способ объявления строковой переменной фиксированной длины: необходимо использовать оператор **Dim**. Следующая строка показывает общий синтаксис для создания строки фиксированной длины:

### Синтаксис

---

```
Dim varname As String * N
```

*Varname* — это любое допустимое имя переменной, а *N* — это любое число от 1 до максимальной длины строки, равной примерно 2 миллиардам символов.

---

Следующий оператор является примером объявления строки фиксированной длины:

```
Dim LastName As String * 30
```

Символ (\*), за которым следует число после ключевого слова **String** указывает VBA создать строковую переменную как строку фиксированной длины с заданной длиной (в этом случае — 30 символов). Область действия: доступность переменных

Термин *область действия (scope)* относится к области процедуры или модуля VBA, где данная переменная, процедура или другой идентификатор, являются доступными. В этом разделе описываются два базовых уровня области действия: процедурный и модульный. Переменные, процедуры и идентификаторы, которые доступны только в процедуре, имеют область действия процедурного уровня, а те, которые доступны для всех процедур в модуле, имеют область действия модульного уровня.

### Область действия процедурного уровня

Переменная, объявленная в процедуре, является доступной только в этой процедуре. Например, переменная **HelloMsg** из строки 2 листинга 3.3, является доступной только в процедуре **HelloWorld**; никакая другая процедура не имеет доступа к этой переменной. В действительности, переменная **HelloMsg** реально существует только в то время, когда VBA фактически выполняет процедуру **HelloWorld**. Поэтому говорят, что переменная **HelloMsg** имеет *область действия процедурного уровня (procedure-level scope)*.

Может быть, сначала это незаметно, но VBA ограничивает доступ к переменным посредством своих правил области действия для упрощения работы программиста. Поскольку переменная с областью действия процедурного уровня недоступна ни для какой процедуры, за исключением той, в которой объявляется эта переменная, можно не очень беспокоиться о дублировании имен переменных.

Правила выбора имен переменных указывают, что имя переменной должно быть уникальным в ее области действия. Для переменных с областью действия процедурного уровня это означает, что вы не можете объявлять две переменные с одним и тем же именем в одной и той же процедуре. (Очевидно, что ни

VBA, ни пользователь не могут определить, какую переменную вы намереваетесь использовать, если обе они имеют одно и то же имя).

Поскольку область действия процедурного уровня ограничивает доступ к переменной процедурой, в которой объявляется эта переменная, вы можете использовать то же имя переменной в разных процедурах. Листинг 3.4 показывает две законченные процедуры. (**HelloWorld** в этом листинге отображает то же окно сообщения, которое было показано ранее на рис. 3.2.)

---

**Листинг 3.4.** Область действия процедурного уровня

---

```
1: Sub HelloWorld()  
2:   Dim HelloMsg           'сохраняет текст для MsgBox  
3:   HelloMsg = "Hello, World!"  
4:   MsgBox HelloMsg, , "Окно приветствия"  
5: End Sub  
6:  
7: Sub HelloDave()  
8:   Dim HelloMsg           'сохраняет текст для Msgbox  
9:   HelloMsg = "Hello, Dave!"  
10:  MsgBox HelloMsg, , "Другое окно сообщения"  
11: End Sub
```

---

Строки 1–5 содержат ту же процедуру **HelloWorld** из листинга 3.2, которая работает точно так же. В модуль была добавлена процедура **HelloDave**, которая и начинается в строке 7. Процедура **HelloDave** работает так же, как процедура **HelloWorld**, просто она отображает другое текстовое сообщение и заголовок в своем диалоговом окне.

Заметьте, что в строках 2 и 8 обе процедуры используют оператор **Dim** для объявления переменных с именем **HelloMsg**. Это разумно, потому что **HelloMsg** является хорошим описательным именем для содержимого и назначения переменной в обеих процедурах.

Поскольку переменные **HelloMsg** объявляются внутри отдельных процедур и имеют область действия процедурного уровня, не возникает двусмысленности по поводу того, какую переменную будет использовать VBA. В процедуре **HelloWorld** VBA использует переменную **HelloMsg**, объявленную локально (строка 2 в листинге). В процедуре **HelloDave** VBA использует переменную **HelloMsg**, объявленную локально в этой процедуре (строка 8).

### **Область действия модульного уровня**

Иногда необходимо, чтобы несколько процедур имели доступ к одной и той же переменной. Обычно эффективнее вычислить один раз какое-либо значение, сохранить его в переменной, а затем использовать эту переменную в нескольких процедурах, чем вычислять одно и то же значение снова и снова.

VBA позволяет объявлять переменные, доступные для нескольких процедур. Когда какая-либо переменная доступна всем процедурам в модуле, говорят, что эта переменная имеет область действия *модульного уровня* (*module-level*). VBA ограничивает область действия переменной модульного уровня модулем, в котором объявляется эта переменная. (VBA предоставляет способы еще большего расширения области действия переменной; эти методы описываются далее.)



## Замечание

Область в начале модуля перед любыми объявлениями процедур называют областью *объявлений* (*declaration*) модуля, потому что именно туда следует помещать объявления переменных модульного уровня и другие директивы для VBA, влияющие на весь модуль. Вы можете легко переходить к области объявлений любого модуля, выбирая (Declarations) в списке Procedure окна (Code).

Для того чтобы переменная была доступной для всех процедур в определенном модуле, поместите оператор **Dim** для нее в начало модуля перед любыми объявлениями процедур. В листинге 3.5 приведен модуль, содержащий две простые процедуры и единственное объявление переменной модульного уровня. Процедура **HelloWorld** в листинге 3.5 отображает то же окно сообщения, которое показано на рис. 3.2.

### Листинг 3.5. Область действия переменной модульного уровня

```
1: Dim HelloMsg          'используется всеми процедурами в этом модуле
2:
3: Sub HelloWorld()
4:     HelloMsg = "Hello, World!"
5:     MsgBox HelloMsg, , "Окно приветствия"
6: End Sub
7:
8: Sub HelloDave()
9:     HelloMsg = "Hello, Dave!"
10:    MsgBox HelloMsg, , "Другое окно сообщения"
11: End Sub
```

В этом листинге строки 3–6 содержат процедуру **HelloWorld**, а строки 8–11 — процедуру **HelloDave**. Обратите внимание на то, что ни одна из этих процедур не содержит операторов **Dim**. В строке 1 оператор **Dim** используется для объявления переменной с именем **HelloMsg**.

Поскольку строка 1 объявляет переменную **HelloMsg** на уровне модуля, эта переменная доступна для всех процедур в одном и том же модуле. В строках 4 и 5 процедуры **HelloWorld** VBA использует переменную модульного уровня **HelloMsg**. Аналогично, в строках 9 и 10 процедуры **HelloDave** VBA использует ту же переменную модульного уровня **HelloMsg**.

### Использование переменных с одним и тем же именем в различных областях действия

Имя переменной должно быть уникальным в ее области действия. Точно так же, как нельзя объявлять две переменные с одним и тем же именем в одной процедуре, нельзя объявлять две переменные модульного уровня с одним именем в одном и том же модуле (и по тем же причинам).

Однако можно (хотя и не желательно) иметь переменные с одним и тем же именем на *разных* уровнях области действия. Когда переменные имеют одно имя, но разные области действия, VBA использует переменную с наиболее *локальной* (*local*) областью действия. В листинге 3.4 были уже показаны две различные процедуры, каждая из которых объявляет свою переменную, но использует одно и то же имя для переменной. Поскольку эти переменные имеют

область действия процедурного уровня, каждая процедура может использовать только свою *локальную* переменную. В листинге 3.6 приведен модуль, содержащий три процедуры, каждая из которых используют переменные с одним и тем же именем.

---

**Листинг 3.6.** Комбинированная модульная и процедурная область действия

---

```
1: Dim HelloMsg      'используется всеми процедурами в этом модуле,
2:                   'которые не имеют своей переменной HelloMsg
3: Sub HelloWorld()
4:   HelloMsg = "Hello, World!"
5:   MsgBox HelloMsg, , "Greeting Box"
6: End Sub
7:
8: Sub HelloDave()
9:   HelloMsg = "Hello, Dave!"
10:  MsgBox HelloMsg, , "Другое окно сообщения"
11: End Sub
12:
13: Sub AnotherMessage()
14:   Dim HelloMsg      'локальное объявление
15:   HelloMsg = "Еще одно окно сообщения"
16:   MsgBox HelloMsg, , "Еще одно окно сообщения"
17: End Sub
```

---

В листинге 3.6 приведены три различные процедуры. Первые две уже использовались в листинге 3.5. Третья процедура **AnotherMessage** начинается в строке 13 и заканчивается в строке 17. **AnotherMessage** работает так же, как предыдущие процедуры, отображая сообщение в диалоговом окне с использованием оператора **MsgBox**.

Заметьте, что строка 1 этого листинга объявляет переменную модульного уровня **HelloMsg**. Строка 14 в процедуре **AnotherMessage** также объявляет переменную с именем **HelloMsg**. При выполнении процедуры **HelloWorld** VBA использует переменную **HelloMsg**, объявленную в строке 1. **HelloWorld** не имеет переменных, объявленных локально, здесь невозможна двусмысленность, поэтому VBA использует переменную модульного уровня. Аналогично, при выполнении процедуры **HelloDave** VBA также использует переменную **HelloMsg**, объявленную в строке 1.

---

**Замечание**

---

Переменные процедурного уровня часто называют *локальными* (*local*) переменными, потому что их объявления являются локальными для выполняемой в данный момент процедуры.

---

Однако в процедуре **AnotherMessage** имеется свое объявление переменной **HelloMsg** (строка 14). При выполнении строк 15 и 16 VBA устраняет любую двусмысленность по поводу того, какую переменную **HelloMsg** следует использовать, выбирая наиболее локальную переменную: **HelloMsg**, объявленную как переменную процедурного уровня.

Переменная модульного уровня **HelloMsg**, объявленная в строке 1, недоступна для процедуры **AnotherMessage**.

**Персистентность: определение того, как долго переменные удерживают свое значение**

*Персистентность (persistence)* — это термин, используемый для ссылки на продолжительность времени, в течение которого любая данная переменная удерживает значение, присвоенное ей. Значения, присвоенные переменным, сохраняются только, пока переменная является активной в ее области действия.

Когда переменная объявляется в процедуре, она существует только, пока VBA выполняет эту процедуру. При выполнении процедуры VBA резервирует область памяти для всех переменных, объявленных локально в этой процедуре, независимо от того, объявляются переменные явно или неявно. После окончания процедуры VBA освобождает память, использовавшуюся локальными переменными, в общий пул имеющейся в наличии компьютерной памяти и локальные переменные этой процедуры прекращают существование.

Переменные процедурного уровня создаются каждый раз, когда процедура начинает выполняться, и уничтожаются, когда процедура перестает выполняться. Говоря более техническими терминами, локальная переменная является неопределенной или *вне контекста (out of context)* до тех пор, пока процедура, объявляющая ее, не начинает выполняться. Когда процедура, объявляющая переменную, перестает выполняться, переменная опять становится неопределенной.

Значения переменных, объявляемых на уровне модуля, удерживаются в памяти столь долго, сколько VBA выполняет процедуру в этом модуле.

При выполнении процедуры VBA фактически просматривает весь модуль, содержащий эту процедуру, и создает любые переменные модульного уровня. Пока VBA выполняет процедуру в этом модуле, значения переменных модульного уровня сохраняются.

## Требование явного объявления переменных

Хотя неявное объявление переменных (объявление переменных просто их использованием) удобно, с ним связаны некоторые проблемы. Когда переменные объявляются неявно, существует риск нечаянно создать новую переменную, когда на самом деле необходимо использовать уже существующую, или использовать существующую переменную, когда пользователь намеревается создать новую. Обе эти ситуации приводят к ошибкам в коде, которые очень трудно отслеживать. (*Bug* — любая ошибка в коде, которая мешает его правильному выполнению или приводит к ошибочным результатам.)

В этой книге уже рассказывалось, как использовать оператор **Dim** для объявления переменных и уменьшения проблем, связанных с неявным объявлением. Использование только оператора **Dim** для объявления переменных не всегда помогает обнаруживать или предотвращать небольшие ошибки, относящиеся к неявному объявлению переменных.

Чтобы легче было в любое время обнаруживать ошибки, связанные с неявным объявлением переменных, VBA предоставляет команду **Option Explicit**. При использовании **Option Explicit** VBA требует объявления всех переменных (с помощью оператора **Dim**) перед их использованием. Команда **Option Explicit** в основном препятствует неявному объявлению переменных где-либо в модуле, содержащем эту команду.

Чтобы установить режим, при котором VBA требует явного объявления для всех переменных в модуле, добавьте команду **Option Explicit** в область объявлений модуля, то есть в начало модуля перед любыми объявлениями переменных или процедур. В листинге 3.7 приведен тот же модуль, что и в листинге 3.5, но с добавленной командой **Option Explicit**.

### Листинг 3.7. Модульная команда Option Explicit

---

```
1: Option Explicit 'требуется явное объявление переменных модуля
2: Dim HelloMsg    'используется всеми процедурами в этом модуле
3:
4: Sub HelloWorld()
5:     HelloMsg = "Hello, World!"
6:     MsgBox HelloMsg, , "Окно приветствия"
7: End Sub
8:
9: Sub HelloDave()
10:    HelloMsg = "Hello, Dave!"
11:    MsgBox HelloMsg, , "Другое окно сообщения"
12: End Sub
```

---

Кроме строки 1, все в этом модуле работает точно так же, как в листинге 3.5. Строка 1 модуля в листинге 3.7 содержит команду **Option Explicit**. Из-за этой команды все переменные в модуле должны объявляться оператором **Dim**. Если добавить неявное объявление переменных в этот модуль, VBA отображает сообщение об ошибке времени исполнения, утверждая, что эта переменная является необъявленной.

Такие команды, как **Option Explicit**, называются *директивами компилятора (compiler directives)*. Директивы компилятора не вызывают никаких действий VBA. Компилятор VBA является частью VBA, которая читает исходный код и *компилирует (compiles)* его в машинные инструкции, необходимые компьютеру для выполнения поставленной задачи. Директива компилятора просто инструктирует VBA о специфических правилах, которым должен следовать VBA при компиляции вашего исходного кода.

Команда **Option Explicit** действует только на модуль, в котором она появляется. Если проект, содержащий этот модуль, содержит также другие модули, на них не действует команда **Option Explicit**. Необходимо включать команду **Option Explicit** в каждый модуль, для которого требуются явные объявления переменных.

Поскольку включение **Option Explicit** во все модули является очень полезным, Редактор VB предоставляет способ автоматически включать эту команду в каждый новый модуль при его создании. Для того чтобы Редактор VB добавлял команду **Option Explicit** в каждый новый модуль, выберите опцию **Require Variable Declaration** (явное описание переменных) на вкладке **Editor** (редактор) диалогового окна **Options** (параметры) Редактора VBA, выполняя следующие шаги:

1. Выберите команду **Tools | Options** (Сервис | Параметры). Редактор VBA отображает диалоговое окно **Options**.
2. Щелкните на вкладке **Editor** (Редактор) для отображения параметров редактирования, если необходимо.

3. Выберите флажок **Require Variable Declaration** (явное описание переменных).

4. Выберите **ОК**. Редактор VB закрывает диалоговое окно **Options**.

Теперь каждый раз, когда вы (или макрорекордер) вставляете новый модуль в проект, Редактор VB автоматически добавляет команду **Option Explicit** в начало модуля.

Выбор опции **Require Variable Declaration** в диалоговом окне **Options** влияет только на новые модули; если вы хотите запросить явное объявление переменных в существующем модуле, вы должны добавить **Option Explicit**, редактируя модуль самостоятельно. Требование явного объявления переменных действует на все приложения, использующие Редактор VB; при запуске Редактора VB из Excel и установке опции **Require Variable Declaration** новые модули в Word будут также требовать явного объявления переменных.

## Константы

*Константа (constant)* — это значение в программе VBA, которое не меняется. Примеры процедур, уже приведенные в этой книге, используют строковые константы типа «**Hello, World!**» и «**Другое окно сообщения**». Константы, подобные им, называют *литеральными константами (literal constants)*, потому что литеральное значение записывается непосредственно в код.

В коде VBA можно также писать литеральные численные константы и даты; примеры численных литеральных констант включают числа **36**, **3**, **14** и **212**. Примеры литеральных констант-дат включают даты **#12/31/96#** или **#октябрь 28, 1997#**. Если рассматривать большинство записанных макросов, то можно найти другие примеры литеральных констант. Константы можно изменять только редактированием исходного кода VBA.

Константы, подобные строковым константам в процедуре **HelloWorld**, используются для предоставления данных, которые не изменяются (в отличие от переменных, которые используются для предоставления изменяемых данных). Можно использовать константы как аргументы для процедур и в математических или операциях сравнения.

Константы не должны быть обязательно литеральными. VBA позволяет создавать *именованные константы (named constants)*. Именованная константа, подобно переменной, имеет заданное ей имя; это имя представляет конкретное неизменяемое значение. Так же, как и в случае с переменной, VBA подставляет конкретное значение, на которое ссылается имя константы, в оператор в то место, где VBA встречает именованную константу. Однако в отличие от переменной значение именованной константы никогда не изменяется; как и в случае с литеральной константой, единственным способом изменить значение, связанное с именованной константой, является редактирование исходного кода VBA.

Используйте именованные константы для повышения читабельности процедур. Например, процедура, выполняющая геометрические вычисления, легче читается и более понятна, если использовать именованную константу **Pi** ( $\pi$ ) вместо литеральной константы **3,14**. К тому же, в тексте программы значение одной и той же константы может встретиться несколько раз. Если это не такая фундаментальная константа, как  $\pi$ , и ее значение при разных запусках кода может быть различным, то, конечно, правильно будет объявить

константу один раз и использовать ее имя в коде столько раз, сколько необходимо.

Именованные константы можно также использовать для более простого обновления и сопровождения процедур и программ. Например, в программе VBA, вычисляющей налогообложение фирмы, возможно изменение ставки налога когда-нибудь в будущем. Если поместить ставку налога в эту программу как литеральную константу, может оказаться затруднительным изменить программу, чтобы использовать новую ставку налога, — вам придется искать и изменять каждое вхождение значения ставки налога по всей программе. Если же вместо этого для ставки налога использовать именованную константу, то придется изменить значение ставки налога только в одном месте: в операторе, объявляющем именованную константу.

## Создание именованных констант

При выборе имени константы соблюдайте те же правила и рекомендации, которым вы следуете при выборе имени переменной (см. раздел «Создание переменных» в начале этой главы). Как и в случае с переменной, необходимо объявлять именованную константу перед ее использованием. Однако в отличие от переменной необходимо всегда явно объявлять именованные константы, используя ключевое слово **Const**. Следующая строка показывает общий синтаксис для объявления именованных констант:

### Синтаксис

---

```
Const name1 = value1 [operator name2...] [, name3 = value3  
[operator name4] ... ]
```

Здесь *nameN* представляет любой допустимый идентификатор, *valueN* — любое значение данных: численное, строковое или дата, а *operator* — арифметическая или операция сравнения между двумя именами ранее описанных констант.

---

Следующие строки показывают несколько объявлений именованных констант:

```
Const Boilingpoint = 212  
Const SalesTax = 8.25  
Const Greeting = "Hello, Pete!"  
Const PointTwo = 212, SalesTax2 = 8.25*2, Greeting2 = "Hello, "  
Const DangerZone = BoilingPoint + 50
```

## Область действия констант

Как и в случае с переменными, можно объявлять именованные константы в процедурах или в области объявлений в начале модуля. Константа, объявляемая в процедуре, имеет область действия процедурного уровня, а константа, объявляемая в области объявлений модуля, — область действия модульного уровня. Для именованных констант выполняются те же правила области действия, что и для переменных.

Поскольку одной из главных целей использования именованной константы является предотвращение повторения или дублирования значений литеральных констант в процедурах, как правило, бывает необходимо, чтобы имено-

ванные константы были доступны всем процедурам в модуле. Поэтому следует помещать объявления констант на модульном уровне, чтобы у них была наибольшая область действия.

Всякий раз, когда VBA встречает именованную константу в операторе, значение, связанное с константой, вставляется в этот оператор. Рассмотрим листинг 3.8, в котором показан законченный модуль. Эта процедура вычисляет площадь круга и сохраняет значение в переменной.

---

**Листинг 3.8.** Использование констант: вычисление площади круга

---

```
1: Const Pi = 3.14
2: Dim CircleArea As Single
3:
4: Sub Calc_CircleArea()
5:     Dim Radius As Single
6:     Radius = 5
7:     CircleArea = Pi * (Radius * Radius)
8:     MsgBox CircleArea, , "Area of Circle"
9: End Sub
```

---

Строка 1 листинга 3.8 объявляет константу модульного уровня **Pi**, которая представляет приближение числа  $\pi$ . Строка 7 листинга содержит оператор, вычисляющий площадь круга; при выполнении этого оператора VBA вставляет значение **3.14** в место, занимаемое именем константы **Pi**. В процедуре **Calc\_CircleArea** оператор **MsgBox** (в строке 8) отображает число **78.5**. Обратите внимание на то, что переменная **CircleArea** объявляется на уровне модуля для того, чтобы значение, вычисленное в процедуре **Calc\_CircleArea**, было доступным для других процедур в том же модуле.

## Написание литеральных констант

Даже если вы никогда не использовали литеральные константы в коде VBA (что невероятно), вам все же следует писать литеральные константы при объявлении именованных констант. Существует несколько правил, которые необходимо соблюдать при написании литеральных констант; далее описываются правила написания констант типа **String**, **Integer**, **Date** и **Boolean**.

### Константы *String*

При написании литеральных строковых констант в коде VBA выполняйте следующие правила:

- Строковые константы должны быть заключены в двойные кавычки ("). Следующий пример не является допустимым, потому что не имеет кавычек:  
This is not a valid string constant.
- Пустая строковая константа (называемая *нулевой строкой* — *null string* или *empty string*) обозначается двумя двойными кавычками, между которыми ничего нет (").

- Строковая константа должна вся находиться на одной и той же строке. Нельзя использовать символ продолжения строки для продолжения литеральной строковой константы на другой строке. Ни один из следующих примеров не является правильным, потому что строковая константа находится более, чем на одной строке:

"Это пример  
неправильной строковой константы"  
"Это снова не\_  
правильная строковая константа"

Если любой из этих примеров будет включен в процедуру, VBA выдаст ошибку времени исполнения и прекратит выполнение процедуры. Далее следует пример правильной строковой константы:

"Пример правильной строковой константы"

Если же вам необходимо написать константу на двух строках, используйте оператор конкатенации (см. далее эту главу) и символ перехода на следующую строку, как показано ниже:

"Теперь имеем " & \_  
"правильную строковую константу"

### Численные константы

Данные правила применимы к литеральным численным константам; численные константы могут содержать любой из численных типов VBA.

- Численные константы должны состоять только из числовых символов от 0 до 9.
- Численная константа может начинаться со знака (-) и может содержать десятичную точку.
- Можно использовать экспоненциальное представление для численных констант.
- Никакие другие символы или знаки не допускаются в численных константах. Далее следуют примеры правильных численных констант:

12  
-14.3  
6.6E2

### Константы Date

VBA распознает константы даты в любом из нескольких различных форматов; необходимо помещать все константы даты между знаками фунта (#). Следующие строки показывают некоторые из форматов констант дат, которые распознает VBA:

#2-5-97 21:17#  
#February 5, 1997 9:17pm#  
#Mar-31-97#  
#15 April 1997#

Независимо от того, в каком из указанных форматов записывается литеральная константа типа **Date**, VBA переформатирует эту константу (когда курсор вставки будет убран со строки после записи константы) для соответст-



вия одному из двух следующих форматов, в зависимости от того, содержит ли константа **Date** информацию о времени:

```
#2/5/97 9:17:00 PM#  
#2/5/97#
```

Если пропустить знак (#) при записи литеральной константы даты, VBA не сможет правильно интерпретировать константу даты как дату. Вместо этого VBA «пытается» оценить информацию о дате как имена переменных, численные константы и математические операторы. Например, VBA пытается интерпретировать следующую дату (в которой пропущены знаки фунта) как математическое выражение, включающее деление:

```
3/12/94
```

Не заключайте литеральные константы типа **Date** в кавычки, потому что VBA будет интерпретировать дату как строковую константу. Например, VBA интерпретирует следующую строку как строковую константу, а не как дату:

```
"3/11/99"
```

### Константы *Boolean*

Существует только две правильные константы типа **Boolean**: **True** и **False**. В операторах программ VBA для констант типа **Boolean** вы всегда должны использовать ключевые слова **True** и **False**. Когда вы записываете ключевые слова в макросы, пишите слова полностью и не используйте кавычки.

### Задание типа константы

Когда вы объявляете именованную константу или используете литеральную константу, VBA «считает», что значение, представленное этой константой, имеет тип данных, наиболее согласующийся с выражением, присвоенным константе. Например, VBA работает с константой, содержащей строку, как с константой типа **String** при определении того, правильно ли константа сочетается с другими значениями данных.

В VBA можно задать тип константы, и если вы точно знаете тип константы, то лучше объявлять тип, поскольку объявление конкретного типа данных для константы может повысить точность вычислений. При объявлении константы типа **Double**, например, VBA вычисляет результат математических операций, включающих эту константу, используя больший диапазон типа данных **Double**. Можно также задать тип константы как **Integer**, **Long**, **Currency** или другой, чтобы результаты математических операций, использующих эту константу, имели определенный тип.

Для констант можно использовать типы данных **Byte**, **Boolean**, **Integer**, **Long**, **Single**, **Double**, **Currency**, **Date** или **String** (но не **Object** или **Array**). Объявление типа для константы подобно объявлению типа для переменной, за исключением того, что объявление начинается с ключевого слова **Const**.

Общий синтаксис для объявления *типизированной константы* следующий:

## Синтаксис

---

```
Const name As type = value [, name As type = value...]
```

Здесь *name* — это любое допустимое имя константы, *type* — имя любого из типов данных VBA и *value* — значение, которое вы присваиваете константе. Следующая строка иллюстрирует правильное объявление константы с определенным типом:

```
Const Pi As Double = 3.14
```

В этом примере константа **Pi** объявляется как тип **Double**. Можно также использовать символы определения типа для задания типа константы, как показано в следующем примере (действие которого такое же, как в предыдущем примере):

```
Const Pi# = 3.14
```

---

## Внутренние константы

VBA предоставляет несколько *внутренних констант* (*intrinsic constants*), называемых также *предопределенными константами* (*predefined constants*). Внутренняя константа — это именованная константа, которая была определена разработчиками VBA. В дополнение к константам, которые предопределяет VBA, host-приложение VBA также предопределяет несколько констант для использования с этим host-приложением. Например, Excel содержит несколько внутренних констант для использования с рабочими книгами Excel, таблицами и так далее. Аналогично, Word содержит несколько внутренних констант для работы с документами и шаблонами Word, тогда как Access содержит внутренние константы, имеющие отношение к различным операциям с базами данных и свойствам Access-объектов.

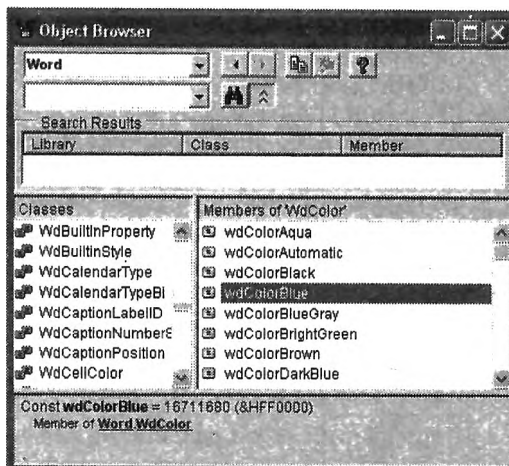
Внутренние константы, определяемые VBA, все начинаются с букв **vb** для указания того, что они определяются языком Visual Basic for Applications (или Visual Basic). Например, константы **vbOKOnly**, **vbOKCancel** и **vbAbort-RetryIgnore** определяются VBA. Внутренние константы Excel начинаются с букв **xl**, чтобы было понятно, что они определяются Excel; некоторые из внутренних констант Excel — это **xlChart**, **xlCountrySetting** и **xlWorksheet**. Внутренние константы Word начинаются с букв **wd**, чтобы было понятно, что они определяются Word; вот некоторые внутренние константы Word: **wdBackward**, **wdForward** и **wdToggle**.

Благодаря внутренним константам, легче использовать некоторые встроенные процедуры VBA, такие как оператор **MsgBox**, о котором вы уже знаете, и оператор **InputBox**, о котором рассказывается далее в этой главе. Хотя ни в одном из показанных уже примеров с оператором **MsgBox** не использовался его необязательный аргумент, вы можете вспомнить из предыдущих обсуждений, что **MsgBox** имеет необязательный аргумент, указывающий число кнопок в диалоговом окне; обычно для аргумента кнопок в **MsgBox** используются внутренние константы VBA.

Внутренние константы, определяемые Excel, Word или другим host-приложением VBA, упрощают использование различных свойств и методов, принадлежащих этому host-приложению. (Подробнее о свойствах и методах вы узнаете из следующих глав.)

Рис. 3.3

Используйте **Object Browser** для просмотра списков констант, определенных в VBA и host-приложениях



## Поиск имеющихся внутренних констант с помощью Object Browser

Для того чтобы увидеть полный список имеющихся в наличии внутренних констант, определяемых VBA или host-приложением, используйте **Object Browser** и выполняйте следующие шаги:

1. Нажмите **Alt+F11** для активизации Редактора VB.
2. Выберите команду **View | Object Browser** (Вид | Просмотр объектов) (или щелкните на кнопке **Object Browser** на панели инструментов). Редактор VB отображает диалоговое окно **Object Browser**. На рис. 3.3 показано диалоговое окно **Object Browser** со списком Word-констант **WdColor**.

После того, как вы откроете диалоговое окно **Object Browser**, выполните следующие дополнительные шаги, чтобы увидеть внутренние константы VBA:

1. Выберите **VBA** в раскрывающемся списке диалогового окна **Object Browser**.
2. Выберите **Constants** в списке **Classes** (Классы).
3. Чтобы получить более подробную информацию об отдельной внутренней константе, выберите эту константу в списке **Members Of 'Constants'**.

После того как вы выберете константу в списке **Members Of 'Constants'**, ее имя появится в нижней части диалогового окна **Object Browser**. Чтобы получить более подробную информацию о выбранной константе, щелкните на кнопке **Help** окна **Object Browser** (на кнопке со знаком вопроса).

Внутренние константы VBA — одни и те же во всех host-приложениях VBA. Внутренние константы для некоторых приложений, например для Excel, доступны только тогда, когда вы работаете с VBA в этом приложении. Например, вы не сможете найти константу **xlWorksheet** Excel в окне **Object Browser** в Word, так как константа **xlWorksheet** доступна только тогда, когда вы используете VBA в Excel.

Если для элемента, выбранного в списке **Classes**, имеется раздел справочной системы, кнопка **Help** окна **Object Browser** доступна, как показано на рис. 3.3. Щелкните на этой кнопке для доступа к справочному разделу для выбранного

элемента. Не все объекты в окне **Object Browser** имеют дополнительный справочный раздел.

Для просмотра списка внутренних констант, определяемых host-приложением VBA, выполняйте те же шаги, что описаны выше, но вместо VBA выберите библиотеку host-приложения в раскрывающемся списке **Project/Library** (Проект/Библиотека). Например, чтобы просмотреть внутренние константы Excel, выберите **Excel** в раскрывающемся списке **Project/Library**; чтобы просмотреть внутренние константы Word, выберите **Word** в списке **Project/Library**. Можно просматривать внутренние константы только для host-приложения VBA, из которого был запущен Редактор VB, если только вы не создаете ссылку на библиотеку другого host-приложения.

## Получение данных от пользователя

Информация этого раздела не связана с основной темой главы, но необходима для оставшегося материала этой и следующих глав.

Получение данных от пользователя, сохранение их в переменной и отображение результатов действий, выполненных над ними, являются основными элементами, необходимыми для написания интерактивных процедур. [Интерактивная (*interactive*) процедура — это процедура, обменивающаяся информацией с пользователем, то есть процедура взаимодействует с пользователем, отображая сообщения и получая ввод]. Интерактивные процедуры часто бывают более полезными, чем макросы, создаваемые исключительно путем записи. Получая информацию от пользователя, интерактивная процедура может выполнять одни и те же операции, используя различные данные.

Данные, вводимые пользователем, называются *входными данными* (*input*). Чтобы получить входные данные от пользователя процедуры, используйте функцию **InputBox**. [Функция (*function*) — это особый тип процедуры VBA, возвращающей значение]. Функция **InputBox** отображает диалоговое окно, содержащее текст, который запрашивает пользователя ввести некоторое значение, и текстовое окно для ввода этого значения. Диалоговое окно, отображаемое **InputBox**, содержит также командные кнопки **OK** и **Cancel**.

Функция **InputBox** имеет следующий синтаксис:

### Синтаксис

---

```
stringvar = InputBox (Prompt [, Title])
```

Здесь *stringvar* представляет любую переменную, которая может сохранять строку (либо переменную типа **String**, либо — **Variant**). Аргумент *Prompt* представляет любое строковое значение (литерал, константу или переменную). **InputBox** отображает эту строку как запрос в диалоговом окне; необходимо всегда предоставлять аргумент *Prompt*, поскольку это — *обязательный аргумент* (*required argument*).

Аргумент *Title* является необязательным вторым аргументом для **InputBox**. *Title* представляет любое строковое значение (литерал, константу или переменную). **InputBox** отображает текст этой строки в строке заголовка диалогового окна. Если опустить аргумент *Title*, VBA отображает в строке заголовка диалогового окна **InputBox** слово «Input».

---

В листинге 3.9 приведен код модуля с одной процедурой. В коде имеется объявление константы и переменной модульного уровня. Эта процедура, подобно процедуре в листинге 3.8, вычисляет площадь круга, но радиус круга получает от пользователя процедуры.

**Листинг 3.9.** Получение входных данных при помощи оператора `InputBox`

```
1: Const Pi As Single = 3.14 'аппроксимация значения pi
2: Dim CircleArea As Single 'сохраняет вычисленную площадь круга
3:
4: Sub Calc_CircleArea()
5:     Const BoxTitle = "Площадь круга"
6:     Dim Radius As Single, Temp As String
7:     Temp = InputBox("Введите радиус круга", BoxTitle)
8:     Radius = CSng(Temp)
9:     CircleArea = Pi * (Radius * Radius)
10:    MsgBox CircleArea, , BoxTitle
11: End Sub
```

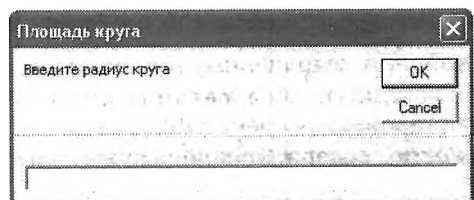
Строка 1 листинга 3.9 объявляет константу модульного уровня **Pi**. Строка 2 объявляет переменную модульного уровня **CircleArea**. Эти элементы объявляются на уровне модуля, чтобы они были доступны во всем модуле; другим процедурам в том же модуле может понадобиться доступ к значению **Pi**, а еще какой-то процедуре может понадобиться использовать значение для площади круга, после того, как оно будет вычислено.

Строка 4 содержит фактическое объявление процедуры; как и раньше, эта процедура имеет имя **Calc\_CircleArea**. Строка 5 объявляет константу процедурного уровня **BoxTitle**; эта константа имеет только локальный доступ в процедуре **Calc\_CircleArea**. **BoxTitle** была объявлена локально в этой процедуре, потому что маловероятно использование этой константы в какой-либо другой процедуре; ее назначение — предоставлять фиксированный текст для строки заголовка всех диалоговых окон, отображаемых **Calc\_CircleArea**.

Обратите внимание на строку 7 листинга 3.9. Этот оператор вызывает функцию **InputBox**. Она отображает свой первый аргумент как текст в диалоговом окне, запрашивая у пользователя ввести значения какого-либо типа. В этом операторе **InputBox** отображает текст «Введите радиус круга», чтобы сообщить пользователям процедуры, какое значение они должны ввести. **InputBox** использует второй аргумент, представленный константой **BoxTitle**, в качестве заголовка диалогового окна. При выполнении оператора **InputBox** отображается диалоговое окно, показанное на рис. 3.4. Пользователь вводит число в текстовое окно и выбирает командную кнопку **OK** или **Cancel**, чтобы закрыть диалоговое окно, как и любое другое окно Windows.

**Рис. 3.4**

Диалоговое окно ввода, отображаемое функцией **InputBox** (в строке 7 листинга 3.9)



Всякий раз, когда вы вызываете какую-либо функцию, необходимо как-то использовать значение, возвращаемое функцией. [Значение, возвращаемое функцией, называется *результатом функции (function result)*]. Часто функции используют результат, присваивая его какой-либо переменной, как показано в строке 7 в случае с функцией **InputBox**, результат которой присваивается переменной **Temp**.

Результат функции **InputBox** всегда является строкой (вот почему переменная **Temp** была объявлена как **String**). Посмотрите еще раз на рис. 3.4. Если пользователь выбирает командную кнопку **OK**, функция **InputBox** возвращает то, что пользователь ввел в текстовое окно, в качестве результата. Если пользователь выбирает командную кнопку **Cancel** (или нажимает клавишу Esc, или использует кнопку **Close** диалогового окна), **InputBox** в качестве результата возвращает пустую строку.

Поскольку переменная **Temp** была объявлена явно как **String**, значение строки должно быть преобразовано в численное значение перед тем, как можно будет его использовать в математических вычислениях. Строка 8 листинга 3.8 выполняет именно это, используя встроенную VBA-функцию **CSng** для преобразования пользовательских данных ввода в число с типом **Single**. **CSng** — это одна из нескольких функций преобразования данных, более подробно описанных далее в этой главе.

## Выражения в Visual Basic

Теперь вы знаете, как записывать и редактировать макросы, писать собственные простые процедуры и создавать и использовать константы и переменные. В этом разделе описывается, как объединять переменные и константы для создания новых значений.

*Выражение (expression)* — это значение или группа значений, выражающая отдельное значение. Каждое выражение *вычисляется до* (или имеет результатом) отдельного значения. Выражение  $2+5$ , например, имеет результатом 7. Выражения состоят из одной или более следующих частей:

- ☐ Константы (литеральные или именованные)
- ☐ Переменные (любого типа данных)
- ☐ Знаки операций
- ☐ Массивы
- ☐ Элементы массива
- ☐ Функции

Все выражения имеют результатом одно значение определенного типа данных, о которых вы узнали ранее в этой главе. Выражения могут также иметь результатом одно из специальных значений **Empty** или **Null**. VBA предоставляет ключевые слова **Empty** и **Null** для указания особых значений в переменной или выражении. Значение **Empty** представляет неинициализированную переменную типа **Variant** или результат выражения, содержащий неинициализированную переменную типа **Variant**. Значение **Null** представляет выражение, содержащее неверные данные.

Вот некоторые примеры выражений:

Выражение	Описание
7	Содержит одно значение и имеет результатом число 7.
«25»	Содержит одно значение и имеет результатом символьную строку «25».
«To» & «day»	Содержит два строковых значения и знак операции; имеет результатом одну строку «Today».
b / (p*10)	Возвращает число, полученное делением содержимого переменной b на значение, полученное после умножения содержимого переменной p на константу 10.
Value <= 17	Вычисляется до логического значения; в этом случае указывая, является ли содержимое переменной Value меньшим или равным константному численному значению 17.

Знаки (обозначения) операций используются для объединения, сравнения или других действий над определенными значениями в выражении. Операциям присвоены имена, и они могут иметь обозначающий их знак, например, операция сложения имеет знак «+». Когда вы используете какой-либо знак в выражении, элементы данных (переменные или константы), над которыми производится действие, называются *операндами (operands)*; большинству операций требуется два операнда. В выражении 3+1, например, числа 3 и 1 являются операндами операции сложения (+). Выражение может содержать один, несколько знаков операции или ни одного.

Выражения используются для выполнения вычислений и сравнения значений, для предоставления переменных в качестве аргументов различным функциям и процедурам VBA. Все выражения Visual Basic вычисляются до значения, имеющего один из типов данных Visual Basic. Далее следует перечень различных типов выражений VBA:

- *Выражение типа даты (date expression)* — это любое выражение, которое вычисляется до значения типа **Date**. Выражения **Date** могут включать константы типа **Date**, переменные, содержащие даты или числа, численные константы, даты, возвращаемые функциями, и знаки арифметических операций.
- *Численное выражение (numeric expression)* — это любое выражение, которое вычисляется до числа любого типа — **Byte**, **Integer**, **Long**, **Single**, **Double** или **Currency**. Численные выражения могут включать переменные, хранящие числа, численные константы, функции, возвращающие числа, и знаки арифметических операций. Численные выражения могут также включать строковые выражения, которые VBA преобразует в число. (Если все символы в строке являются цифрами, то VBA может преобразовать эту строку в численное значение; например, строку «436» в число 436.)
- *Строковое выражение (string expression)* — это любое выражение, которое имеет результатом значение типа **String**. Строковые выражения могут включать переменные, которые хранят строки, строковые константы, функции, возвращающие строки, или знаки (обозначения) операций конкатенации строк. Строковые выражения могут также включать численные выражения, которые VBA может преобразовать в строку.

- *Логическое выражение (logical expression)* — это любое выражение, которое вычисляется до значения типа **Boolean**: **True** или **False**. Логические выражения могут состоять из переменных, содержащих значения **Boolean**, констант **Boolean**, функций, возвращающих значения типа **Boolean**, знаков операций сравнения или логических операций.
- *Объектное выражение (object expression)* — это любое выражение, которое имеет результатом ссылку на объект.

## Совместимость типов данных

Не все типы данных совместимы друг с другом, и нельзя использовать несовместимые типы данных в одном и том же выражении. Например, не имеет смысла арифметическое сложение строки с числом, так как такое выражение не является значащим и VBA не может его оценить.

Многие типы данных совместимы друг с другом. Например, вы можете объединять различные численные типы данных в одном и том же выражении; VBA автоматически выполняет необходимые преобразования типа различных численных типов. VBA может также иногда автоматически преобразовывать другие типы данных так, чтобы все типы в выражении были совместимы, хотя это не всегда возможно. Например, типы **String** и **Date** совместимы с численными типами только при особых условиях, которые будут описаны далее.

Очень важно контролировать и знать тип выражения, потому что если выражения содержат несовместимые типы, VBA выдает ошибку времени исполнения — ошибку *несовпадения типов (type-mismatch)*. Иногда, впрочем, такая ошибка является результатом неправильной интерпретации операторов, например, имеющих несколько аргументов.

При обработке выражения, содержащего различные типы данных VBA сначала «пытается» устранить любое различие типов, преобразуя значения в выражения в совместимые типы данных. Если устранить какие-либо различия преобразованием типов не удается, отображается ошибка времени исполнения и процедура прекращает выполняться.

Если вы присваиваете результат какого-либо выражения переменной, а тип переменной несовместим с типом результата этого выражения, VBA также отображает ошибку несовпадения типов. Аналогично, использование такого результата выражения в качестве аргумента для функции или процедуры также приводит к ошибке несовпадения типов, если тип данных аргументов и тип результата выражения являются несовместимыми.

VBA предоставляет различные функции для преобразования информации одного типа в другой, например, строк в числа, чисел в строки и так далее (ранее уже использовалась функция **CSng**, используемая для преобразования типа **String** в численный **Single**). Функции преобразования данных VBA описываются более подробно в следующей главе.

### Автоматическое преобразование данных Visual Basic

VBA использует различные правила для автоматического преобразования данных в совместимые типы и выбирает правила преобразования данных применительно к определенному выражению на основе конкретных типов и операторов, используемых в выражении.

VBA выполняет автоматическое преобразование типов быстрее, если выражение содержит переменные типа **Variant**, потому что тип данных **Variant**



не является фиксированным. Если выражение содержит литеральные константы, типизированные переменные или типизированные константы, VBA применяет более определенные правила преобразования типов, потому что тип данных этих элементов является фиксированным. (Помните, типизированная переменная или типизированная константа — это переменная или константа, для которой вы явно объявили тип данных либо с помощью ключевого слова **As**, либо символа определения типа.)

Следующие выражения иллюстрируют, как от типов данных и оператора зависят преобразования, выполняемые VBA:

Выражение	Описание и тип результата
Num + Str	Создает ошибку несовпадения типов, когда Num объявляется как численный тип, а Str — как тип <b>String</b> . Выполняется строковая конкатенация, когда Num является <b>Variant</b> , а Str объявляется как тип <b>String</b> ; результат является типом <b>String</b> .
Num + Str	Выполняется арифметическое сложение, когда Num объявляется как численный тип, а Str — как <b>Variant</b> , или когда и Num, и Str являются типом <b>Variant</b> ; результат — численный тип.
Num + «3»	Вызывает ошибку несовпадения типов, если Num объявлен численным типом. Выполняется строковая конкатенация, если Num является типом <b>Variant</b> ; результат имеет тип <b>String</b> .
3 + Str	Вызывает ошибку несовпадения типов, если Str объявлен как тип <b>String</b> . Выполняется арифметическое сложение, если Str является типом <b>Variant</b> ; результат является численным типом. (Заметьте, что это выражение никогда не имеет результатом строковую конкатенацию.)
Num & Str	Всегда выполняет строковую конкатенацию, независимо от типов переменных; результатом является тип <b>String</b> . Это выражение никогда не вызывает ошибки несовпадения типов.

Обратите особое внимание на последнее выражение. В нем используется оператор строковой конкатенации (&), который соединяет («склеивает») вместе две строки. Поскольку можно использовать этот оператор *только* со строками, VBA преобразует типы данных в этом выражении в строки, независимо от их оригинального типа данных и независимо от того, имеют ли какие-либо переменные в выражении определенный тип. По мере описания отдельных операторов в дальнейших разделах этой главы описываются также особые правила преобразования данных, которые VBA применяет для каждого отдельного оператора.

Вы, наверное, уже заметили, что большинство из указанных выше выражений вызывают ошибку несовпадения типов только тогда, когда оба операнда в выражении имеют определенные и разные типы. Вы можете также заметить, что выражения не вызывают ошибку несовпадения типов, когда один из операндов является переменной типа **Variant**. Как было упомянуто выше, VBA легче всего выполняет автоматические преобразования типов с данными типа **Variant**. Не следует, однако, использовать типы **Variant** только для того, чтобы избежать ошибок несовпадения типов, поскольку игнорирование возможности использования встроенных типов может привести к другим ошибкам.

## Преобразования численных типов

VBA обычно преобразует все численные типы данных в выражении в тип наибольшей точности, а затем дает этот тип результату выражения. Например, если выражение содержит численные значения с типами **Integer** и **Single**, результат выражения является типом **Single** — тип наибольшей точности в этом выражении.

Если вы присваиваете результат численного выражения переменной с наименьшей точностью, чем фактический тип результата выражения, VBA округляет результат выражения до тех пор, пока его точность не совпадет с ожидаемым типом. Например, если вы присваиваете численное выражение, имеющее результатом число типа **Double**, переменной типа **Integer**, VBA округляет число двойной точности до типа **Integer**.

### Преобразования строк и чисел

При преобразовании числа в строку VBA создает строку, содержащую все цифры этого числа и десятичный знак (если число имеет его). Число 3413.72<sup>1</sup>, например, преобразуется в строку «3413,72». Если число очень большое или очень маленькое, VBA может создать строковое представление числа в экспоненциальной записи; например, число 0.0000000004927 преобразуется в строку «4,927E-11».

VBA может преобразовывать строку в число, если только эта строка содержит символьное представление числа в десятичном формате или экспоненциальном. Строки «988,6», «812», «-186,7», «1,3E10» представляют числа, и VBA может преобразовать их в числа. Строки «1.045», «\$74.550» и «С добрым утром!» не могут быть преобразованы в числа.

### Преобразования значений типа Boolean

Когда VBA преобразует значения типа **Boolean** в числа, значение **True** преобразуется в 1, а значение **False** — в 0. Когда VBA преобразует число в тип **Boolean**, нуль преобразуется в **False**, а любое другое значение преобразуется в **True**. Когда VBA преобразует значения типа **Boolean** в строки, VBA использует строку «True» для **True** и «False» — для **False**. VBA не может преобразовывать строковые выражения в тип **Boolean**, но может преобразовывать численные выражения в значения этого типа. Когда вы используете значения типа **Boolean** в арифметических выражениях, VBA всегда преобразует их в числа, даже если все другие части этого выражения также являются переменными или константами типа **Boolean**.

### Преобразования значений типа Date

Когда VBA преобразует тип данных **Date** в число, результатом является численное значение — число типа **Double**, которое содержит количество дней от 30 декабря 1899 (отрицательное число представляет дату, более раннюю, чем 12/30/1899). Десятичная часть числа (если имеется) выражает время дня как часть дня; 0 — это полночь, а 0.5 — это полдень. В VBA преобразование численных типов данных в типы **Date** является просто обратным преобразованием типа **Date** в число.

<sup>1</sup> Точка используется для записи числа в коде.

Если выражение имеет результатом значение, находящееся вне диапазона типа данных для этого выражения, VBA отображает сообщение об ошибке переполнения. Если выражение содержит литеральную константу, иногда можно решить эту проблему, используя символ определения типа с литеральной константой, чтобы выражение имело результатом численный тип с большим диапазоном значений.

### Оператор присваивания

Ранее уже использовался *оператор (statement) присваивания*. Этот оператор используется для присваивания результата выражения переменной. Оператор присваивания сохраняет любое значение, которое представлено выражением справа от оператора присваивания (=), в ячейке памяти, на которую ссылается переменная слева от оператора.

Оператор присваивания имеет две синтаксические формы, обе из которых являются одинаково приемлемыми и выполняют одну и ту же задачу. Первая форма оператора присваивания использует ключевое слово **Let** и имеет следующий общий синтаксис:

#### Синтаксис

---

`Let varname = expression`

Переменная *varname* представляет любую переменную VBA, а *expression* — любое выражение VBA. Этот синтаксис является оригинальной формой оператора присваивания, использовавшейся в ранних версиях языка программирования Basic. Следующий оператор VBA является примером оператора присваивания **Let**:

`Let x = y` 'присваивает переменной x значение, представленное  
'переменной y

Вторая форма оператора присваивания более употребительна в программировании на VBA и широко используется в этой книге. Общий синтаксис этой формы оператора присваивания следующий:

`varname = expression`

Переменная *varname* — любая переменная, а *expression* — любое выражение. Далее следуют примеры этого более простого, более употребительного оператора присваивания (первая строка присваивает содержимое Y переменной X; вторая строка присваивает переменной SecondVar результат сложения 102 с содержимым FirstVar):

`X = Y`  
`SecondVar = 102 + FirstVar`

---

При выполнении оператора присваивания VBA сначала вычисляет выражение справа от оператора присваивания (=), а затем сохраняет результат выражения в переменной, имя которой находится слева от оператора присваивания. Новичкам в вопросах программирования бывают не сразу понятны операторы, подобные следующим:

`Номер = Номер + 1`  
`ОбщаяСумма = ОбщаяСумма + ПодСумма`

Особенность этих операторов в том, что и слева, и справа от знака оператора появляется один и тот же идентификатор переменной. Чтобы понять, каким должен быть результат подобных инструкций, следует еще раз внимательно прочитать синтаксис оператора присваивания. Тогда станет ясно, что сначала в некоторую промежуточную ячейку памяти помещается результат вычисления (оценивания) выражения справа от оператора «=» (для первого примера это — `Номер + 1`, для второго — `ОбщаяСумма + ПодСумма`), а затем этот результат помещается в ячейку памяти, соответствующую переменной, идентификатор которой находится слева от оператора «=».

Более короткая форма оператора присваивания используется гораздо чаще, чем длинная, в основном потому, что короткая быстрее вводится с клавиатуры. Длинная форма оператора присваивания с ключевым словом **Let** часто рекомендуется для начинающих программистов, поскольку помогает отличать оператор присваивания от операции сравнения (равенства).

Хотя эти две конструкции используют один и тот же символ, не путайте оператор присваивания со знаком равенства (=), используемым для тестирования на равенство. Следующие два выражения приводят к совершенно разным результатам: первый — является оператором и присваивает значение 5 переменной **FirstVal**; второй — является логическим выражением, которое имеет результатом **True** или **False**, в зависимости от того, содержит ли уже **FirstVal** число 5:

```
FirstVal = 5      'присваивает число 5 переменной FirstVal  
(FirstVal = 5)    'сравнивает содержимое FirstVal с числом 5
```

Заметьте, что вторая строка, фактически, не является законченным оператором VBA (хотя является законченным выражением), поскольку результат логического выражения никаким образом не используется; подобная строка в ваших процедурах может вызвать ошибку синтаксиса. Чтобы получить синтаксически верный оператор VBA, необходимо каким-то образом использовать результат этого выражения: присвоить его переменной, передать его в качестве аргумента функции или процедуре.

Когда вы присваиваете результат выражения переменной с определенным типом данных, этот результат может иметь тип данных, совместимый с типом переменной, получающей новое значение. Во многих случаях VBA может преобразовывать тип данных результата выражения в тип, совместимый с типом переменной, принимающей новое значение, если результат выражения и переменная еще не имеют совместимых типов. Переменным типа **Variant** может быть присвоен любой тип данных.

Запомните следующее:

- Можно присваивать любую численную переменную или выражение любой другой переменной численного типа или переменной типа **Variant**. Если присваивается численное выражение типизированной переменной с меньшей точностью или диапазоном (как при присваивании типа **Double** типу **Long**), VBA округляет значение выражения для совпадения с точностью переменной, принимающей новое значение.
- Если переменной типа **String** присваивается переменная типа **Variant**, содержащая число, VBA автоматически преобразует это число в строку. (При присваивании типизированных численных переменных или численных констант переменной типа **String** происходит ошибка несовпадения типов.)

Арифметические операции

VBA может выполнять все обычные арифметические операции (реализуемые посредством арифметических выражений): сложение, вычитание, умножение и деление, а также возведение чисел в указанную степень и предоставляет дополнительные особые математические операции для целочисленного деления и деления по модулю. В табл. 3.3 приводятся знаки операций, которые можно использовать при написании арифметических VBA-выражений. (В этой таблице Ni — это любое допустимое численное выражение VBA.)

Таблица 3.3. Знаки операций (обозначения), используемые в арифметических VBA-выражениях

Знак	Синтаксис	Имя/Описание
+	N1 + N2	Сложение. Прибавляет N1 к N2.
-	N1 - N2	Вычитание. Вычитает N2 из N1.
*	N1 * N2	Умножение. Умножает N1 на N2.
/	N1/ N2	Деление. Делит N1 на N2.
\	N1\ N2	Целочисленное деление. Делит N1 на N2, отбрасывая любую дробную часть так, чтобы результат был целым числом.
Mod	N1 Mod N2	Деление по модулю. Делит N1 на N2, возвращая только остаток операции деления.
^	N1 ^ N2	Возведение в степень. Возводит N1 в степень N2.

Далее описывается подробно использование каждого знака.

Сложение (+)

Знак операции сложения (+) используется для выполнения операции простого сложения. Оба операнда должны быть численными выражениями или строками, которые VBA может преобразовать в число. Знак операции сложения можно также использовать для построения арифметических выражений с данными типа Date.

Для большего понимания следующего материала войдите в Редактор VB (в Word или Excel выберите Разработчик | Visual Basic). Далее щелкните на кнопке View Code в Project Window. В появившемся Code Window набирайте предлагаемые далее небольшие программы и, если вам этого будет не достаточно, можете проводить свои собственные эксперименты с кодом.

Введите в Code Window код листинга 3.10.

Листинг 3.10. Пример использования знака операции «+»

```
1: Sub list5_01()  
2:   'Примеры арифметических выражений со знаком "+"  
3:   Dim N1, N2, N3 As Integer  
4:  
5:   N1 = 1           'присваивание значения переменной N1  
6:   N2 = 2           'присваивание значения переменной N2  
7:   N3 = N1 + N2     'сложение; результат - в N3
```

```
8: MsgBox (N3) 'Вывод N3 в диалоговом окне
9:
10: End Sub
```

---

Понятно, что при запуске этого кода (**Run | Run Sub/UserForm**) на экране появится диалоговое окно с числом 3.

Тип данных результата выражения сложения обычно тот же, что и наиболее точный тип в этом выражении. Например, если выражение содержит оба типа **Integer** и **Long**, результатом такого выражения будет тип **Long**. Однако существуют некоторые исключения, в частности, если выражение включает переменные типа **Variant**. Далее перечисляются эти исключения:

1. Результатом сложения типа **Single** и **Long** является **Double**.
2. Если складывать тип **Date** с любым другим типом данных, результатом выражения всегда будет тип **Date**.
3. Если результат выражения сложения присваивается переменной **Variant**, имеющей в данный момент тип **Integer**, и если результат выражения больше, чем (переполняет) диапазон значений для типа **Integer**, то VBA преобразует результат в **Long**. После присваивания переменная **Variant** также имеет тип **Long**.
4. Если результат выражения сложения присваивается переменной **Variant**, имеющей в данный момент тип **Long**, **Single** или **Date**, и если результат выражения переполняет диапазон численного типа, VBA преобразует результат в **Double**. После присваивания переменная типа **Variant** также имеет тип данных **Double**.
5. Если любой операнд в выражении сложения является равным **Null** или вычисляется до **Null**, то результатом выражения сложения также будет **Null**. (**Null** — это особое значение, которое можно присваивать только переменным типа **Variant** для обозначения того, что они не содержат действительных данных.)

### Замечание

---

Порядок точности для численных типов данных VBA от наименее точного до наиболее точного следующий: **Byte**, **Integer**, **Long**, **Single**, **Double**, **Currency**.

---

В листинге 3.11 содержится код, демонстрирующий правила преобразования выражений со знаком «+», приведенные в последнем перечне.

### Листинг 3.11. Преобразования типов в выражениях со знаком «+»

---

```
Sub list5_02()
'Примеры арифметических выражений со знаком "+"

Dim S1 As Single
Dim L1 As Long
Dim I1 As Integer
Dim D1 As Date
Dim V1, V2 As Variant

'инициализация переменных
```

```
S1 = 3.333
L1 = 3.333
I1 = 1111
D1 = Now      'получить значение текущей даты и времени

MsgBox TypeName(S1 + L1), vbOKOnly, "Single + Long"
MsgBox TypeName(D1 + L1), vbOKOnly, "Date + Long"

V1 = I1        'переменная типа Variant становится типа Integer
MsgBox TypeName(V1), vbOKOnly, "V1=I1"

V1 = V1 + 99999 'переменная типа Variant становится типа Long
MsgBox TypeName(V1), vbOKOnly, "V1 + 99999"

V1 = D1 'переменная типа Variant становится Date
V1 = V1 + 99999999 'переменная типа Variant становится Double
MsgBox TypeName(V1), vbOKOnly, "V1 = D1; V1 = V1 + 99999999"

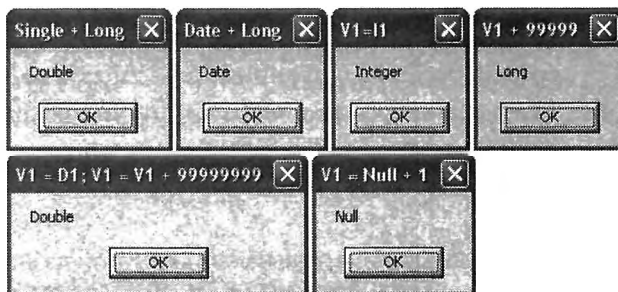
V1 = Null + 1   'переменная типа Variant становится Null
MsgBox TypeName(V1), vbOKOnly, "V1 = Null + 1"
```

End Sub

На рис. 3.5 приведен ряд диалоговых окон, которые отображаются на экране во время выполнения кода листинга 3.11.

**Рис. 3.5**

Диалоговые окна, отображаемые при выполнении кода листинга 3.11



Заметим, что довольно часто начинающие программировать не уделяют этим вопросам должного внимания; это иногда приводит к непонятным для них (и не только) результатам.

### Вычитание (-)

Знак операции вычитания (-) выполняет две задачи: он используется либо для построения арифметических выражений вычитания, либо для обозначения унарного минуса.

*Унарный минус (unary minus)* — это знак минус, который помещают перед числом для указания того, что это — отрицательное число. Знак минус можно также помещать перед переменной или другим выражением для обозначения того, что значение переменной или выражения должно быть отрицательным. Поместить унарный минус перед переменной или выражением означает то же, что умножить это число на -1. Следующие выражения содержат унарный минус:

-10	литеральная отрицательная константа
-MyVar	делает отрицательным любое число, сохраненное в MyVar
-(MyVar + 10)	делает отрицательным результат сложения 10 и MyVar

Следующие несколько выражений показывают примеры использования знака вычитания для построения арифметических выражений:

Now - 60	результат выражения имеет тип Date
40 - MyVar	разность между 40 и MyVar
AnyVar - SomeVar	разность между AnyVar и SomeVar

Оба операнда в выражении вычитания должны быть численными переменными или выражениями, или строковым выражением, которое VBA может преобразовать в число. Знак операции вычитания можно также использовать для построения арифметических выражений с данными типа **Date**. Тип данных результата выражения вычитания обычно является тем же, что и наиболее точный тип данных в этом выражении.

VBA следует тем же правилам для определения типа данных результата выражения вычитания, что и для выражений, использующих знак операции сложения, но имеются следующие дополнительные правила:

- Если один из операндов в выражении вычитания является типом **Date**, то результат выражения имеет тип **Date**.
- Если оба операнда в выражении являются типом **Date**, то результат выражения имеет тип **Double**.

В листинге 3.12 приведен код, демонстрирующий эти дополнительные правила.

**Листинг 3.12.** Дополнительные правила преобразования типов в выражениях со знаком операции «-»

```

1: Sub list5_03()
2:     'Примеры арифметических выражений со знаком "-"
3:     Dim D1, D2 As Date
4:
5:     D1 = Now      'получить значение текущей даты и времени
6:     D2 = Now - 5
7:
8:     gg = MsgBox(TypeName(D2), vbOKOnly, "Now - 5")
9:     gg = MsgBox(TypeName(D1 - D2), vbOKOnly, "D1 - D2")
10:
11: End Sub

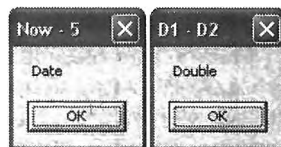
```

На рис. 3.6 приведены диалоговые окна, которые отображаются на экране во время выполнения кода листинга 3.12.



**Рис. 3.6**

Диалоговые окна, отображаемые при выполнении кода листинга 3.12



### Замечание

Несмотря на то, что можно конкатенировать (объединять) две (и более) строки вместе с помощью знака (+), нельзя использовать знак (-), чтобы отделить какую-либо строку. Вместо этого для отделения строк можно использовать VBA-функции **Mid**, **Left** или **Right**.

### Умножение (\*)

Знак операции умножения (\*) используется для построения арифметических выражений с умножением; результатом выражения умножения является произведение двух операндов. Оба операнда в выражении умножения должны быть численными выражениями или строками, которые VBA может преобразовать в числа. Следующие примеры иллюстрируют правильный синтаксис для арифметических выражений с умножением:

```
4 * 10           'умножение 4 на 10; результат равен 40
NumVar * 2       'результат — произведение NumVar и 2
NumVar * OtherVar 'результат — произведение NumVar и OtherVar
```

Тип данных результата выражения умножения обычно тот же, что и наиболее точный тип в этом выражении. VBA следует тем же правилам для определения типа данных результата выражения умножения, что и для выражений, использующих сложение. В выражениях умножения все переменные **Variant**, которые содержат значения типа **Date**, преобразуются в численные значения.

### Деление (/)

Знак операции деления (/) иногда называют знаком деления чисел с плавающей точкой или действительных чисел, чтобы отличать от знака целочисленного деления, который описан в следующем разделе. В выражениях деления первый операнд делится на второй операнд; результатом деления является частное. Оба операнда в выражении деления с плавающей точкой должны быть численными выражениями или строками, которые VBA может преобразовать в числа. Следующие выражения иллюстрируют использование знака операции деления:

```
27 / 3           'деление 27 на 3
MyNumVar / 20     'деление значения в переменной MyNumVar на 20
MyNumVar / MyOtherVar 'делит значение в переменной MyNumVar
                     'на значение в переменной MyOtherVar
```

Если любой операнд в выражении деления имеет значение **Null**, то результатом выражения также является **Null**. Тип данных выражения со знаком деления с плавающей точкой — обычно **Double**, но имеется следующее исключение:

- Если оба операнда в выражении деления имеют тип **Integer** или **Single**, то результат выражения деления с плавающей точкой имеет тип **Single**, если только результат выражения не переполняет диапазон значений для типа **Single**. Если результат переполняет диапазон для типа **Single**, то VBA преобразует результат в тип **Double**.

### Целочисленное деление (\)

Целочисленное деление подобно делению с плавающей точкой, кроме того, что выражения, использующие оператор целочисленного деления (\), всегда имеют результатом целое число без дробной части. Оба операнда в выражении целочисленного деления должны быть численными выражениями или строками, которые VBA может преобразовать в числа. Следующие примеры выражений иллюстрируют правильный синтаксис для целочисленного деления:

4 \ 3	'деление 4 на 3; возвращает значение 1
9 \ NumVar	'деление 9 на содержимое NumVal
MyVar \ NumVar	'деление MyVar на NumVar

В целочисленном делении VBA округляет каждый операнд (если необходимо) до числа типа **Integer** или **Long** перед выполнением операции деления.

VBA отбрасывает любой дробный остаток результата выражения целочисленного деления. Например, выражение деления с плавающей точкой 19 / 5 имеет результатом 3.8, в то время как выражение целочисленного деления 18 \ 5 имеет результатом 3. Заметьте, что VBA не округляет частное целочисленного деления, результат просто укорачивается до целого числа.

Тип данных результата выражения целочисленного деления — либо **Integer**, либо **Long**; VBA использует наименьший тип данных, который соответствует результату выражения. Если какой-либо операнд в выражении целочисленного деления является равным **Null**, то результатом выражения также будет **Null**.

### Деление по модулю (Mod)

Деление по модулю дополняет целочисленное деление. При делении по модулю выражение возвращает только остаток операции деления как целое. Оба операнда в выражении деления по модулю должны быть численными выражениями или строками, которые можно преобразовать в число. Следующие примеры выражений иллюстрируют деление по модулю:

8 Mod 2	'возвращает значение 0
5.1 Mod 3	'возвращает значение 2
6 Mod NumVar	'если NumVar содержит 3, возвращает 0

При делении по модулю VBA округляет каждый операнд (если необходимо) до числа типа **Integer** или **Long** перед выполнением деления так же, как при целочисленном делении. Тип данных результата выражения деления по модулю — это **Integer** или **Long**. VBA использует наименьший тип, который подходит для результата выражения. Если какой-либо операнд в выражении деления по модулю является **Null**, то результатом выражения также будет **Null**.

### Возведение в степень (^)

Знак операции возведения в степень (^) используется в арифметических выражениях для возведения в степень числа или выражения. Показатель степени указывает, сколько раз число (или выражение) должно быть умножено на самое себя:  $3^3$  — это то же, что и  $3 \times 3 \times 3$ . Чтобы записать  $3^3$  как арифметическое выражение VBA, используйте следующее:

3 ^ 3

Оба операнда в выражении возведения в степень должны быть численными выражениями или строками, которые VBA может преобразовать в числа. Опе-

ранд слева от знака возведения в степень может быть отрицательным числом, только если операнд справа является целым. Если какой-либо операнд является равным `Null`, то результатом выражения возведения в степень также будет `Null`; иначе результат выражения будет иметь тип `Double`.

Операции сравнения

Операции сравнения иногда также называют *операциями отношения* (*relational operators*). Чаще всего операции сравнения используются для того, чтобы задать критерии для принятия решения или сформулировать описание условий, при которых группа команд должна быть повторена (организация циклов).

Результатом любой операции сравнения является значение типа **Boolean**: **True** или **False**. Операции сравнения используются для сравнения литеральных, константных или переменных значений любого сходного типа.

В таблице 3.4 перечислены имеющиеся в VBA знаки операций сравнения и описано их назначение (E в этой таблице представляет любое действительное выражение VBA).

Таблица 3.4. Знаки (обозначения) операций сравнения

Операция/ Оператор	Синтаксис	Наименование/описание
=	E1 = E2	Равенство. <b>True</b> , если E1 равно E2, иначе — <b>False</b> .
<	E1 < E2	Меньше, чем. <b>True</b> , если E1 меньше, чем E2, иначе — <b>False</b> .
<=	<=	Меньше, чем или равно. <b>True</b> , если E1 меньше или равно E2, иначе — <b>False</b> .
>	>	Больше, чем. <b>True</b> , если E1 больше, чем E2, иначе — <b>False</b> .
>=	>=	Больше, чем или равно. <b>True</b> , если E1 больше или равно E2, иначе — <b>False</b> .
<>	<>	Не равно. <b>True</b> , если E1 не равно E2, иначе — <b>False</b> .
Is	E1 Is E2	Оба операнда должны быть значениями типа <b>Object</b> . <b>True</b> , если E1 ссылается на тот же самый объект, что и E2, иначе — <b>False</b> .
Like	E1 Like E2	Подобие. Оба операнда должны быть значениями типа <b>String</b> . <b>True</b> , если E1 совпадает с образцом, содержащимся в E2, иначе — <b>False</b> .

Если оба операнда в выражении сравнения имеют один и тот же тип данных, VBA выполняет простое сравнение для этого типа. Если, например, оба операнда являются строками, VBA выполняет сравнение строк; если оба операнда являются датами, то VBA выполняет сравнение дат и так далее.

Следующие выражения иллюстрируют использование различных операторов сравнения:

MyVal = 55	результат <b>True</b> , если MyVal содержит 55
7 <> 5	результат <b>True</b> : 7 не равно 5
NumVar < 57	<b>True</b> , если NumVar меньше, чем 57
Now > #5/30/1997	<b>True</b> , если текущая дата (результат функции Now) больше, чем 30 мая 1997
«Joe» < «Sam»	<b>True</b> ; «Joe» меньше, чем «Sam» (в соответствии с алфавитом)
«Pat» < «Patricia»	<b>True</b> ; «Pat» короче, чем «Patricia»

Помните, что значения дат VBA могут включать информацию о времени. Когда сравниваются значения дат, VBA, фактически, сравнивает информацию о дате и времени. Поэтому, даже если месяц, день и год двух дат являются одними и теми же, они могут не быть равны друг другу, если временные части значения даты различны. Например, дата #1/10/97# меньше, чем дата #1/10/97 9:00:00AM#. В первой дате нет никакого значения времени, и VBA «полагает», что время составляет 00:00:00. Несмотря на то что год, месяц и день во второй дате те же, что и в первой, она имеет более позднее время и, следовательно, является большей, чем первая дата.

Если оба операнда в выражении сравнения имеют определенные типы (либо — константы, либо — типизированные переменные) и эти типы не являются совместимыми, VBA выводит на экран сообщение об ошибке несовпадения типов.

Если один или оба операнда в выражении сравнения являются переменными типа **Variant**, VBA «пытается» преобразовать тип **Variant** в какой-либо совместимый тип, если необходимо. Если преобразование данных необходимо, а VBA не может преобразовать **Variant** в совместимый тип, отображается сообщение об ошибке времени исполнения.

Поскольку не всегда легко определить, будет ли VBA выполнять численное сравнение или строковое, когда в выражении сравнения смешаны численные и строковые значения, следует всегда использовать одну из VBA-функций преобразования типов (например, **CStr**) для явного преобразования значений в числа или строки, чтобы оба операнда в выражении сравнения имели один и тот же тип данных.

## Сравнение строк

Сравнение строк немного сложнее, чем сравнение чисел. В VBA строка равна другой строке, только когда обе строки содержат точно такие же символы в точно таком же порядке и обе строки имеют одну и ту же длину.

Рассмотрим следующие выражения:

«abc» = «abc»	True; обе строки одинаковые
«abc» = « abc »	False; строки не одинаковые

В первом выражении строки равны одна другой: обе содержат те же символы и в том же порядке и имеют одну и ту же длину. Во втором выражении строки не равны одна другой, строка в правой части оператора имеет по одно-

му символу пробела в начале и в конце строки. VBA не игнорирует начальные или конечные символы пробела при сравнении строк. При сравнении строк операторами отношения, VBA сравнивает каждую строку слева направо по-символьно.

Следующие несколько примеров выражений иллюстрируют сравнение строк:

«abb» < «abc»	True
«aab» < «abb»	True
«abc» < «abc »	True
«abcd» > «abc»	True

Будьте внимательны при сравнении строк фиксированной и переменной длины. Помните, что строки фиксированной длины всегда содержат одно и то же количество символов и что размер строки переменной длины увеличивается и уменьшается в зависимости от размера строки, сохраняемой в ней.

**Двоичное и текстовое сравнение строк**

До сих пор в этой главе выражения сравнения строк включали только строки, состоящие из символов нижнего регистра. VBA предоставляет два различных способа сравнения символов различных регистров. Первый метод, который VBA использует для сравнения строк, называется *бинарным (binary)* или двоичным сравнением и является методом сравнения по умолчанию.

Чтобы понять, как работает двоичное сравнение, необходимо помнить, что вся информация в компьютере сохраняется в виде чисел. Для сохранения текста компьютер использует схему, в которой каждый отображаемый символ имеет уникальный номер. Для представления в компьютере все буквы некоторого алфавита имеют уникальные последовательные номера. Обычно буквы верхнего регистра (например, A–Z) имеют меньшие номера, чем буквы нижнего регистра (a–z). Число, соответствующее определенной букве или другому символу, называется *кодом символа (character code)*.

При выполнении двоичного сравнения строковой информации, VBA использует фактический бинарный эквивалент числа для каждого символа, так как при этом сравнивается каждая пара символов. Поскольку буквы верхнего регистра имеют меньшие двоичные номера, буквы верхнего регистра располагаются в алфавитном порядке перед буквами нижнего регистра. Другими словами, для VBA при двоичном сравнении строк, строка «AAA» меньше, чем строка «aaa».

Второй тип сравнения, предлагаемый VBA, называется *текстовым (text)* сравнением. В текстовом сравнении VBA не использует двоичный эквивалент для каждого символа и «рассматривает» буквы верхнего регистра как эквивалентные буквам нижнего регистра. В текстовом сравнении строка «ABC» равна строке «abc».

**Выбор метода сравнения строк**

Для выбора метода сравнения строк, имеющегося в VBA (двоичного или текстового), используйте директиву **Option Compare**:

```
Option Compare [Text|Binary]
```

Чтобы задать двоичное сравнение строк, используйте ключевое слово **Binary**:

Option Compare Binary

Директиву **Option Compare** можно использовать только на модульном уровне, и она влияет только на сравнения, выполняемые процедурами в этом отдельном модуле. Обычно директиву **Option Compare** помещают в область объявлений модуля перед объявлениями переменных или процедур. Если команды **Option Compare** в коде нет, VBA использует двоичные сравнения.

### Сравнение строки с шаблоном

Оператор **Like** дает возможность выполнять особый тип операции сравнения строк и его можно использовать только со строками.

**Like** тестирует строку для определения того, совпадает ли она с заданным шаблоном (образцом). Оператор **Like** можно использовать для выполнения поиска в текстовой информации для нахождения всех слов или фраз, совпадающих с определенным шаблоном. Поиск такого типа часто называют *нечетким поиском* (*fuzzy search*).

Общий синтаксис для оператора **Like** следующий:

### Синтаксис

StrExpr1 Like StrExpr2

*StrExpr1* — любое строковое выражение VBA. *StrExpr2* — строковое выражение, специально созданное для задания шаблона, который оператор **Like** сравнивает с *StrExpr1*. Результатом выражения **Like** является **True**, если первый операнд (*StrExpr1*) совпадает с шаблоном во втором операнде (*StrExpr2*); иначе результатом выражения будет **False**. Оба операнда в этом выражении должны быть строковыми выражениями, или VBA отобразит сообщение об ошибке несовпадения типов.

Шаблон, с которым должна сравниваться строка, задается с использованием различных специальных символов. В таблице 3.5 содержатся методы и символы для создания образцов совпадения для оператора **Like**.

**Таблица 3.5. Символы совпадения с образцом для оператора Like**

Символ образца	Соответствие
#	Любая одиночная цифра от 0 до 9.
*	Любое количество символов в любой комбинации или отсутствие символов.
?	Любой одиночный символ.
[list]	list — это список определенных символов. Совпадение с любым одиночным символом в списке.
[!list]	list — это список определенных символов. Совпадение с любым одиночным символом, не имеющимся в списке.

Следующее выражение является равным **True**, если **AnyStr** содержит строки такие, как «Sadie», «Salone», «Sophie», «Steve» и так далее:

```
AnyStr Like "S*e"
```

Следующее выражение является равным **True**, если **AnyStr** содержит строки такие, как «Penny», «Percy», «Patty» и так далее:

```
AnyStr Like "P????y"
```

Используйте две последние спецификации символов совпадения из табл. 5.3 для указания отдельных символов, совпадение или несовпадение с которыми вам необходимо. Следующие выражения показывают, как использовать квадратные скобки с символьным списком.

Данное выражение является равным **True**, если **AnyStr** содержит «big» или «bid», или — **False**, если оно содержит «bit» или «bin»:

```
AnyStr Like "bi[dg]"
```

Следующее выражение является равным **True**, если **AnyStr** содержит «big», «bid» или «bin», или — **False**, если оно содержит «bit»:

```
AnyStr Like "bi[!t]"
```

Еще одно выражение является равным **True**, если **AnyStr** содержит «bin» или «bit», или — **False**, если **AnyStr** содержит «bid» или «big»:

```
AnyStr Like "bi[!dg]"
```

Можно также использовать квадратные скобки для указания диапазона символов, совпадение или несовпадение с которыми необходимо:

```
AnyStr Like "ci[a-f]"
```

```
AnyStr Like "ci[!a-f]"
```

Результат первого из приведенных выше примеров является равным **True**, если **AnyStr** содержит «cia», «cib», «cic», «cid», «cie» или «cif» и **False** — иначе. Второе выражение обратно первому и является равным **False**, если **AnyStr** содержит одну из строк «cia», «cib», «cic», «cid», «cie», «cif». Второе выражение является равным **True** для любой строки, содержащей первые две буквы ci и не заканчивается на буквы от a до f.

Когда вы указываете диапазон символов, необходимо указывать диапазон от наименьшего до наибольшего символа. Например, [a-f] — правильный диапазон, но [f-a] — нет. VBA игнорирует пары квадратных скобок, в которых ничего не заключено ([]).

Поскольку левая квадратная скобка ([), знак вопроса (?), символ номера (#) и символ (\*) имеют особое значение в строке шаблона, необходимо заключать их в квадратные скобки, если вы хотите, чтобы они были частью шаблона. Например, если необходимо узнать, заканчивается ли строка вопросительным знаком, используйте следующее выражение (которое является равным **True**, если **AnyStr** содержит любое число или комбинацию символов, заканчивающуюся знаком вопроса):

```
AnyStr Like "*[?]"
```

Правая скобка (]) и восклицательный знак (!) также имеют особое значение в строке шаблона; для достижения совпадения с этими символами необходимо помещать их вне квадратных скобок списка символов. Например, чтобы опре-

делить, заканчивается ли строка восклицательным знаком, используйте следующее выражение (которое является равным **True**, если **AnyStr** содержит любое число или комбинацию символов, заканчивающуюся восклицательным знаком):

```
AnyStr Like "!"
```

Для совпадения со знаком дефиса в строке шаблона поместите дефис в начало или в конец списка символов внутри квадратных скобок. Помещение дефиса в любое другое место задает диапазон символов. Следующее выражение показывает, как сопоставлять с символом дефиса (это выражение является **True**, если **AnyStr** содержит «big-headed», «pig-headed», «plug-ugly», «tag-along» и так далее):

```
AnyStr Like "*g[-]*"      'True
```

На результат сравнения строк, которое использует оператор **Like**, оказывает также влияние инструкция **Option Compare**. Если задано двоичное сравнение строк, то оператор **Like** различает буквы верхнего и нижнего регистра. Если выбрана установка текстового сравнения, оператор **Like** не чувствителен к состоянию регистра.

Использует ли VBA в данный момент двоичное или текстовое сравнение, также влияет на то, какие символы оператор **Like** включает в различные диапазоны, задаваемые пользователем. Если, например, был задан диапазон [e-i], а опция сравнения установлена на двоичное сравнение, то только символы e, f, g, h и i совпадают с указанным диапазоном. Если опция сравнения установлена на текстовое сравнение, то диапазон [e-i] включает еще несколько символов — (Е, е, И, и, Й, й, К, к, Ё, ё, F, f, G, g, H, h, I) и I.

### Замечание

---

Если необходимо только узнать, является ли строка частью другой строки, используйте VBA-функцию **Instr** вместо оператора сравнения **Like**. Например, если необходимо только определить, является ли строка «big» частью строки «bigger», используйте функцию **Instr**.

---

### Замечание

---

Оператор сравнения **Like** полезен еще и тем, что он используется в синтаксисе языка запросов SQL. И поскольку вам скорее всего придется заниматься программированием баз данных — самыми распространенными VBA-приложениями — следует с особым терпением разобраться со всеми тонкостями использования этого оператора.

---

## Сравнение объектов

VBA имеет следующий оператор сравнения — **Is**. Этот оператор можно использовать только для сравнения выражений типа **Object**. Выражения, включающие оператор **Is**, всегда имеют результатом значение типа **Boolean**.

Результатом оператора **Is** является **True**, если оба выражения типа **Object** ссылаются на один и тот же объект; в противном случае результатом является **False**. VBA предоставляет оператор **Is**, потому что никакой другой из опера-



торов сравнения не является значащим, когда он используется с выражениями типа **Object**. Выражения типа **Object**, на самом деле, являются адресами в памяти, ссылающимися на определенный объект в host-приложении (Excel, Word, Access или каком-либо другом приложении, реализующем VBA).

## Логические операторы

Чаще всего логические операторы VBA используются для объединения результатов отдельных выражений сравнения, чтобы создать сложные критерии для принятия решений в процедуре, или для создания условий, при которых группа операторов должна повторяться.

Можно использовать любое действительное выражение, имеющее результат типа **Boolean**, в качестве операнда для логического оператора или число, которое VBA может преобразовать в значение типа **Boolean**. Для VBA 0 является эквивалентным **Boolean**-значению **False**, а любое другое численное значение — значению **True**.

Обычно результатом логической операции является значение типа **Boolean**, хотя некоторые логические операции могут иметь результатом особое значение **Null**, если один или больше операндов имеют значение **Null**. Поскольку VBA интерпретирует особое значение **Empty** как 0, операнды в логических выражениях, содержащие **Empty**, обрабатываются так, как если бы они содержали **Boolean**-значение **False**.

В таблице 3.6 приведены логические операторы, имеющиеся в VBA, вместе с синтаксисом и кратким описанием работы. (Е в этой таблице представляет собой любое допустимое выражение с результатом типа **Boolean**, такое как операция сравнения.)

Таблица 3.6. Логические операторы

Оператор	Синтаксис	Имя/Описание
And	E1 And E2	Конъюнкция. <b>True</b> , если оба E1 и E2 имеют значение <b>True</b> , иначе — <b>False</b> .
Or	E1 Or E2	Дизъюнкция. <b>True</b> , если одно выражение или оба (E1 и E2) являются равными <b>True</b> ; иначе — <b>False</b> .
Not	Not E1	Отрицание. <b>True</b> , если E1 имеет значение <b>False</b> ; <b>False</b> , если E1 является равным <b>True</b> .
Xor	E1 Xor E2	Исключение. <b>True</b> , если E1 является равным <b>True</b> или E2 является равным <b>True</b> ; иначе — <b>False</b> .
Eqv	E1 Eqv E2	Эквивалентность. <b>True</b> , если E1 имеет то же самое значение, что и E2; иначе — <b>False</b> .
Imp	E1 Imp E2	Импликация. <b>False</b> , когда E1 является равным <b>True</b> и E2 равно <b>False</b> ; иначе — <b>True</b> .

Логические VBA-операторы **And**, **Not** и **Or** подобны функциям рабочего листа **AND**, **NOT** и **OR**, встроенным в Excel. Однако Excel не имеет эквивалентных функций для VBA-операторов **Eqv**, **Imp** или **Xor**.

### Таблицы истинности

*Таблица истинности (truth table)* — это таблица, показывающая все возможные комбинации значений для определенного типа логического выражения и его результаты. Большинство таблиц истинности имеют три столбца. Первый столбец предназначен для значения первого операнда, второй — для значения второго операнда, а последний столбец всегда содержит значение результата выражения. Каждая строка в таблице предназначена для определенной комбинации значений. Рассмотрим следующую строку из таблицы истинности оператора **And**:

False True False

В этой строке первый операнд — это **False**, второй операнд — **True**, а результат операции **And** является равным **False**. Эта строка в таблице истинности означает, что результатом выражения **False And True** является **False**.

### And

Оператор **And** выполняет логическую конъюнкцию. Результатом операции **And** является значение **True**, только когда оба операнда являются равными **True**; иначе — **False**.

Оператор **And** имеет следующий синтаксис:

### Синтаксис

---

*Operand1 And Operand2*

*Operand1* и *Operand2* являются любыми допустимыми логическими выражениями VBA. [Логическое выражение (*logical expression*) — это любое выражение VBA, которое имеет результатом значение типа **Boolean: True** или **False**.]

---

Следующая таблица истинности показывает результаты операции **And**.

Operand1	Operand2	Результат выражения
True	True	True
True	False	False
False	True	False
False	False	False

Оператор **And** можно использовать не только для двух операндов. Результатом этого оператора с несколькими операндами будет значение **True**, если все операнды в выражении равны **True**. Например:

(2 > 5) And (7 > 3) And (3 > 1)

Это выражение имеет результатом **False**. Обратите внимание, что операнды в этом выражении являются выражениями сравнения; заметьте также круглые скобки вокруг выражений, которые составляют операнды оператора **And**. Круглые скобки указывают VBA на необходимость вычислить выражение внутри скобок перед вычислением других частей этого выражения. Кроме

того, круглые скобки делают выражение более читабельным, группируя вместе связанные части выражения. Скобки, используемые для группировки частей выражения в подвыражение, являются общей особенностью выражений всех типов: численных, строковых, дат и сравнения, так же как и логических выражений.

### Or

Оператор **Or** выполняет логическую дизъюнкцию и его часто называют *включающим «ИЛИ» (inclusive or)*. Результатом операции **Or** является значение **True**, когда хотя бы один из операндов равен значению **True**; иначе (если все операнды — **False**) результатом является значение **False**.

Оператор **Or** имеет следующий общий синтаксис:

### Синтаксис

---

*Operand1 Or Operand2*

*Operand1* и *Operand2* являются любыми допустимыми логическими выражениями VBA.

---

Следующая таблица истинности показывает результаты операции **Or**.

Operand1	Operand2	Результат выражения
True	True	True
True	False	True
False	True	True
False	False	False

Оператор **Or** можно использовать не только для двух операндов. Результатом оператора с несколькими операндами будет значение **True**, если хотя бы один операнд в выражении равен **True**. Например:

$(2 > 5) \text{ Or } (7 > 3) \text{ Or } (3 > 1)$

Это выражение имеет результатом **True**. Сравните с предыдущим примером.

Операторы **And** и **Or** можно комбинировать в одном выражении. Например, значение следующего выражения равно **True**:

$((2 > 5) \text{ Or } (7 > 3)) \text{ And } ((3 > 1) \text{ Or } (8 < 10))$

### Not

Оператор **Not** выполняет логическое отрицание; он инвертирует любое значение, к которому применяется. Оператор **Not** использует только один операнд и имеет результатом значение **True**, если операнд является равным **False**, или — **False**, если операнд — **True**.

Оператор **Not** имеет следующий общий синтаксис:

**Синтаксис**

*Not Operand*

*Operand* — это любое допустимое логическое выражение VBA.

Следующая таблица истинности показывает результаты операции **Not**:

Operand	Результат выражения
True	False
False	True

Подумайте над результатом следующего выражения и используйте среду VBA для проверки своих рассуждений.

`( (2 > 5) Or (7 > 3) ) And Not ( (3 > 1) Or (8 < 10) )`

Код для тестирования этого выражения совсем простой (см. листинг 3.13) и приведен только с целью напомнить о том, что мы изучаем программирование на VBA. Все, что мы обсуждаем, следует проверять, даже если вам кажется это очень понятным.

**Листинг 3.13.** Очень простой тест логического выражения

```
1: Sub list5_04()  
2:  
3:   dd = ((2 > 5) Or (7 > 3)) And Not ((3 > 1) Or (8 < 10))  
4:   gg = MsgBox(dd, vbOKOnly)  
5:  
6: End Sub
```

**Xor**

Оператор **Xor** выполняет логическое исключение. Чаще его называют *исключающим «ИЛИ»* (*exclusive or*). Результатом операции **Xor** является **True**, когда какой-либо операнд (но не оба), является равным **True**; иначе результатом будет значение **False**.

Оператор **Xor** имеет следующий общий синтаксис:

**Синтаксис**

*Operand1 Xor Operand2*

*Operand1* и *Operand2* — это любые допустимые логические выражения VBA.

Следующая таблица истинности показывает результаты операции **Xor**:

Operand1	Operand2	Результат выражения
True	True	False
True	False	True
False	True	True
False	False	False

Листинг 3.14 содержит код, полученный из кода листинга 3.13 заменой оператора **Or** на **Xor**. Проверьте, совпадает ли ожидаемый вами результат с тем, который выдает VBA.

#### Листинг 3.14. Отличие оператора **Or** от **Xor**

```
1: Sub list5_04()  
2:  
3:   dd = ((2 > 5) Xor (7 > 3)) And Not ((3 > 1) Xor (8 < 10))  
4:   gg = MsgBox(dd, vbOKOnly)  
5:  
6: End Sub
```

### **Eqv**

Оператор **Eqv** является операцией логической эквивалентности; использование оператора **Eqv** подобно тестированию на равенство. Результатом операции **Eqv** является значение **True**, когда оба операнда будут равными **True**; иначе — **False**.

Оператор **Eqv** имеет следующий синтаксис:

#### Синтаксис

*Operand1 Eqv Operand2*

*Operand1* и *Operand2* — это любые допустимые логические выражения VBA.

Следующая таблица истинности показывает результаты операции **Eqv**:

Operand1	Operand2	Результат выражения
True	True	True
True	False	False
False	True	False
False	False	True

Если любой операнд в выражении операции **Eqv** является равным **Null**, то результатом также будет значение **Null**.

## Imp

Оператор **Imp** выполняет логическую операцию импликации. Оператор **Imp** тестирует взаимосвязь между двумя логическими значениями, когда справедливость одного значения подразумевает справедливость другого значения; в качестве результата операция **Imp** имеет значение **True**, только когда второй операнд не противоречит первому. Оператор **Imp** имеет следующий синтаксис:

### Синтаксис

---

*Operand1 Imp Operand2*

*Operand1* и *Operand2* — это любые допустимые логические выражения VBA.

---

Следующая таблица истинности показывает результаты операции **Imp**.

Operand1	Operand2	Результат выражения
True	True	True
True	False	False
False	True	True
False	False	True

Результат логической импликации может быть равным **True**, даже когда первое условие является значением **False**, в то время как второе условие является значением **True**. Однако результат логической импликации не может быть значением **True**, если первое условие является равным **True**, а второе — **False**. Результаты логической импликации не являются такими же интуитивными и понятными, как результаты других логических операторов; к счастью, использование оператора логической импликации редко бывает необходимо.

## Конкатенация строк

VBA дает возможность объединять (склеивать) строки вместе для образования более длинных строк. Присоединение одной строки к другой называется *конкатенацией (concatenation)* строк. Конкатенация строк чрезвычайно полезна и довольно проста, хотя имеются некоторые трудности, о которых необходимо знать.

### Использование конкатенации строк

Конкатенацию строк чаще всего используют для формирования строк из различных источников в процедуре, чтобы создавать пользовательские сообщения и выводить на их экран. В листинге 3.15 приведена процедура VBA Excel, которая запрашивает у пользователя имя файла рабочей книги, открывает эту рабочую книгу и затем выбирает определенный рабочий лист (в данном случае — рабочий лист с именем *Отчет о продажах*).

**Листинг 3.15.** Конкатенация строк

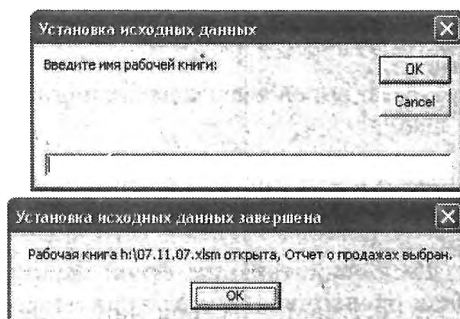
```
1: Sub Open2DataEntry()  
2: 'Демонстрация конкатенации строк  
3:  
4:   Const BoxTitle = "Установка исходных данных"  
5:   Const ShtName = "Отчет о продажах"  
6:  
7:   Dim FName As String  
8:  
9:   'получить от пользователя имя рабочей книги  
10:  FName = InputBox("Введите имя рабочей книги:", BoxTitle)  
11:  
12:  Workbooks.Open FileName:=FName           'открыть книгу  
13:  
14:  ActiveWorkbook.Sheets(ShtName).Select   'выбрать рабочий лист  
15:  
16:  'сообщение пользователю о том, что книга открыта  
17:  MsgBox "Рабочая книга " & FName & " открыта, " & ShtName & _  
18:        " выбран.", , BoxTitle & " завершена"  
19: End Sub
```

Заметьте, что оператор конкатенации используется только в строках 17–18. Вся остальная часть программы предназначена для описания строковых констант и подготовки строковых переменных, используемых в операторе **MsgBox**, в котором получается строка из нескольких констант и переменных.

Если вы выполняете эту процедуру и вводите имя файла при запросе на ввод, оператор **MsgBox** в строках 17–18 отображает диалоговое окно, показанное на рис. 3.7. (Этот пример предполагает, что вы действительно имеете рабочую книгу с именем **H:\07.11.07.xlsm** и в этой книге есть лист с именем **Отчет о продажах**; в противном случае VBA отображает ошибку времени исполнения. Если вы будете экспериментировать с этим листингом, убедитесь, что вы вводите правильное имя рабочей книги.)

**Рис. 3.7**

Оператор **InputBox** (строка 10) отображает окно для ввода наименования рабочей книги; оператор **MsgBox** (строки 17–18) отображает окно с сообщением о результате

**Операторы конкатенации строк**

VBA имеет два знака операции для конкатенации строк; в следующих параграфах описывается каждый из этих знаков и их использование.

Знак **&** можно использовать только для конкатенации строк; этот знак не имеет другого назначения или функции в VBA. Все примеры в этой книге ис-

пользуют знак & для конкатенации строк — это предпочтительный знак конкатенации строк, потому что он не оставляет никаких сомнений относительно того, какую операцию вы намереваетесь выполнить. Общий синтаксис знака & такой:

### Синтаксис

---

*Operand1 & Operand2 [& .Operand3...]*

*Operand1* и *Operand2* — любые допустимые строковые или численные выражения. Если один или оба операнда являются численными выражениями, VBA преобразует числа в строки перед выполнением операции конкатенации. Тип данных результата конкатенации строк — это всегда тип **String**. Если операнд в выражении конкатенации строк является равным **Null** или **Empty**, VBA интерпретирует этот операнд как строку нулевой длины (то есть, строку, которая не содержит никаких символов).

---

Если вы не отделяете знак & от имени переменной (перед ним), по крайней мере, одним пробелом, VBA «полагает», что вы хотите использовать символ & как символ определения типа **Long** вместо знака операции конкатенации. Эта ситуация имеет различные результаты: в некоторых случаях VBA преобразует переменную или результат выражения в число с типом данных **Long** (как в листинге 3.16, результат выполнения которого представлен на рис. 3.8), в других случаях отображает сообщение о какой-либо ошибке синтаксиса или времени исполнения. Если вы уже объявили все переменные с конкретными типами (что является хорошей практикой в программировании), VBA отобразит сообщение об ошибке.

### Листинг 3.16.

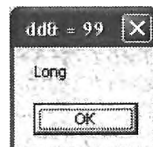
---

```
1: Sub list5_07()  
2:   dd& = 99  
3:   gg = MsgBox(TypeName(dd), vbOKOnly, " dd& = 99 ")  
4: End Sub
```

---

**Рис. 3.8**

Результат выполнения кода листинга 3.16



Для конкатенации строк можно также использовать знак операции «+». Знак операции «+» имеет синтаксис и требования к операндам такие же, как и знак &, только с одним главным отличием. Знак операции «+» имеет исторические корни в оригинальных версиях языка программирования BASIC. В этих ранних версиях языка BASIC не уделялось большого внимания различию между конкатенацией строк и сложением. Однако в VBA главным назначением знака «+» является арифметическое сложение. Всякий раз при обработке выражения, использующего знак «+», VBA сначала «пытается» выполнить арифметическое сложение. VBA выполняет конкатенацию строк со знаком «+», только



если один из операндов является строковым выражением и не может быть преобразован в число или если оба операнда являются строками.

Используйте знак `&` для конкатенации строк, чтобы избежать двусмысленности по поводу того, какую операцию необходимо выполнять, и чтобы обеспечить выполнение конкатенации строк VBA. Не используйте знак «+» для конкатенации строк, потому что выражения, использующие этот знак, могут быть неоднозначными для VBA и пользователей, которые будут читать ваш код.

## Приоритеты выполнения операций при вычислении сложных выражений

*Сложное (составное) выражение (complex expression)* — это любое выражение, образованное из двух или более выражений. Многие из записываемых вами выражений — это сложные выражения, особенно, если они определяют управление последовательностью выполнения кода в процедурах или представляют различные математические формулы.

При вычислении сложных выражений VBA следует таким правилам:

- Части выражения, заключенные в круглые скобки, всегда вычисляются в первую очередь. Если выражение, заключенное в круглые скобки, является другим сложным выражением, VBA применяет эти же правила к выражению в круглых скобках.
- Конкретные операции выполняются в зависимости от иерархии операторов. (Иерархия различных операторов VBA показана в табл. 3.7.)
- Когда операторы имеют равный уровень приоритета, они вычисляются в порядке слева направо.

VBA вычисляет выражения в следующем порядке:

- Знаки арифметических операций.
- Знаки конкатенации строк.
- Операторы сравнения.
- Логические операторы.

В таблице 3.7 приведена иерархия операторов VBA. Операторы/операции перечислены в порядке от самого высокого приоритета до самого низкого. Операторы, помещенные в одной и той же строке этой таблицы, имеют равный приоритет. Операторы с равным приоритетом вычисляются слева направо так, как они появляются в выражении.

**Таблица 3.7. Иерархия операторов/операций от наивысшего до самого низкого приоритета**

Оператор/знак	Комментарии
<b>^</b>	Возведение в степень, наивысший приоритет
<b>-</b>	Унарный минус
<b>*,/</b>	Умножение и деление имеют равные приоритеты; они вычисляются по мере появления в выражении слева направо.
<b>\</b>	
<b>Mod</b>	
<b>+, -</b>	Сложение и вычитание имеют равный приоритет; они вычисляются по мере появления в выражении слева направо.
<b>&amp;</b>	Всякая конкатенация строк выполняется после любых арифметических операций в выражении и перед любыми операциями сравнения или логическими операциями.
<b>&lt;, &lt;=, &gt;, &gt;=, Like=, &lt;&gt;, Is</b>	Все операторы сравнения имеют равные приоритеты и вычисляются по мере появления в выражении слева направо. Используйте круглые скобки для группирования операторов сравнения в выражениях.
<b>Not</b>	
<b>And</b>	
<b>Or</b>	
<b>Xor</b>	
<b>Eqv</b>	
<b>Imp</b>	

# Функции в Visual Basic

Вы уже пользовались встроенными VBA-функциями: **InputBox** и **MsgBox**. Теперь, когда вы знаете все о выражениях, пора научиться включать функции в выражения. При чтении этой главы не очень полагайтесь на правильность синтаксиса приведенных здесь функций. Печатные издания часто содержат ошибки. Ошибки встречаются и в справочных системах локализованных продуктов. Правильнее будет рассматривать эту главу как введение в мир функций и получение общего представления о функциях как необходимых составляющих системы программирования. Точный синтаксис каждой функции следует проверять в справочной системе VBA.

*Функция (function)* — это встроенная формула, выполняющая действия над выражениями и генерирующая значение. Функция всегда возвращает значение, которое VBA вставляет в программу в том месте, где появляется имя функции. Функции VBA делятся на несколько групп в зависимости от типа операции или вычисления, которое они выполняют. Функции используются для выполнения таких задач, как:

- Преобразование текстовых строк в другие типы данных.
- Получение данных о текстовых строках.
- Преобразование других типов в текстовые строки.
- Форматирование чисел или других типов данных для отображения.
- Манипулирование или генерирование значений данных.
- Получение информации о файлах, драйверах диска или среде, в которой VBA работает в данный момент времени.

В этой главе рассказывается о различных категориях функций и описываются подробно наиболее важные функции VBA и их использование. В последующих главах описываются другие функции VBA, перечисленные в этой главе и связанные с темами последующих глав. Кроме того, вы узнаете, что можно создавать собственные функции для использования в процедурах VBA.

В этой книге вы уже встречались с примерами процедур и функций VBA. Не путайте термины *функция* и *процедура*. Обычно процедура выполняет определенную задачу (или группу задач) подобно тому, как определенная команда меню в Excel, Word или другом приложении выполняет конкретную задачу. Функция оперирует одним или более значениями и возвращает некоторое результирующее значение (как формула в ячейке рабочего листа Excel).

## Использование функций в выражениях

Чтобы использовать функцию, просто вводите имя функции в оператор VBA вместе с любыми аргументами, которые требуются для этой функции, в том месте в операторе, где вам необходимо использовать результат функции. [Помещение имени функции в оператор VBA для активизации функции называют *вызовом (calling)* функции.] При использовании функций в выражениях следуйте таким правилам:

- ☐ Можно использовать результат функции как часть выражения.
- ☐ Можно присваивать результат функции какой-либо переменной.
- ☐ Можно использовать результат функции для предоставления значения в список аргументов другой процедуры или функции.
- ☐ Функции имеют списки аргументов, заключенные в круглые скобки.

В основном функцию можно использовать для предоставления значения в любом месте в любом операторе VBA, где может быть оправданно использование значения константы или переменной. В листинге 4.1 приведена процедура для демонстрации вызова функции в выражениях.

### Листинг 4.1. Использование функций

```
1: Sub FuncDemo()  
2:   Dim s_Date As String  
3:   Dim d_Time As Date  
4:   s_Date = CStr(Now)  
5:   d_Time = Time  
6:   MsgBox "Сегодня: " & s_Date  
7:   MsgBox "Текущее время: " & d_Time  
8: End Sub
```

В строках 2–3 объявлены две переменные: **s\_Date** с типом **String** и **d\_Time** с типом **Date**. В строке 4 переменная **s\_Date** получает значение текущей даты в виде строки, так как после вызова функции **Now** (получить системную дату) результирующее значение преобразуется функцией **CStr** в строку. Переменная **d\_Time** принимает значение системного времени. В строках 6–7 с помощью оператора **MsgBox** значения переменных выводятся на экран (рис. 4.1) в диалоговых окнах. Обратите внимание на то, что, поскольку выражение в строке 7 содержит оператор конкатенации строк (&), для вывода в диалоговом окне переменной **d\_Time** нет необходимости преобразования ее в строку. Поскольку эта синтаксическая «вольность» отсутствует во многих других языках программирования, я бы не советовал часто пользоваться этими VBA-возможностями. К тому же это не способствует стремлению к аккуратности при написании кода.

**Рис. 4.1**  
Результат выполнения кода  
листинга 4.1



## Аргументы и возвращаемое значение функции

Как вы уже видели в листинге 4.1, некоторые функции требуют одного или более аргументов, другие — нет. Функции, которые не требуют аргументов, обычно просто выбирают значение, недоступное иначе. Например, функция **Time**, возвращающая текущее системное время компьютера, не требует никаких аргументов, а функцию **CStr** вызывать без аргументов бессмысленно. Чтобы использовать функцию, не имеющую аргументов, просто введите имя функции в программу, как в следующих операторах:

```
СистемноеВремя = Time  
MsgBox Now
```

Другие функции требуют одного или более аргументов. Функция **Sqr**, например, возвращает корень квадратный числа; чтобы **Sqr** вычислила корень квадратный, необходимо предоставить число, как показано в следующих операторах:

```
r_Val = 16  
Root = Sqr(r_Val)
```

Значения предоставляются функции помещением их в список аргументов. Список аргументов следует заключать в круглые скобки и, если в списке находится более одного аргумента, отделять каждый аргумент запятой.

Тип данных значения, возвращаемого функцией, зависит от этой конкретной функции. Большинство функций возвращают значения типа **Variant**, хотя некоторые функции возвращают данные определенных типов, таких как **String**, **Double** и **Integer**. VBA во многих случаях может автоматически преобразовывать результат какой-либо функции в данные типа, совместимого с другими типами значений в выражении, содержащем эту функцию, точно, как VBA преобразует типы данных в присваивания переменных и вычислениях выражений. Правила совместимости типов данных, приведенные в предыдущей главе, которые применимы к переменным **Variant** и типизированным переменным и константам, применимы также к значениям, возвращаемым функциями.

## Игнорирование результата функции

Как правило, значение, возвращаемое функцией, должно быть каким-либо образом использовано: путем включения результата функции в выражение, в список аргументов, или путем присваивания его некоторой переменной. Однако в некоторых случаях можно не использовать возвращаемое функцией значение.

Ранее было упомянуто, что VBA-процедура **MsgBox** имеет необязательный второй аргумент, позволяющий определять, сколько командных кнопок появляется в окне сообщения. Можно использовать этот аргумент командных кнопок в процедуре **MsgBox** для отображения сообщения, которое задает пользователю вопрос и дает возможность ответить на него путем выбора командной кнопки в окне сообщения.

Если вы используете этот необязательный аргумент для **MsgBox**, то можно получить от **MsgBox** значение, указывающее, какую кнопку выбрал пользователь. В действительности, процедура **MsgBox** является функцией, хотя чаще всего используется как процедура.

До сих пор во всех примерах этой книги аргумент командных кнопок в операторах **MsgBox** был опущен. Если при вызове **MsgBox** аргумент командных кнопок не указывается, **MsgBox** отображает диалоговое окно, содержащее только одну кнопку. Поскольку диалоговое окно **MsgBox** до сих пор содержало только одну кнопку, не имело значения, каков результат функции **MsgBox**, и поэтому результат функции **MsgBox** игнорировался в коде.

VBA предоставляет несколько встроенных констант для определения кнопок в окне сообщения **MsgBox**. Одна из этих констант (**vbYesNo**) указывает **MsgBox**, что в отображаемое окно сообщения следует включить кнопку **Да (Yes)** и кнопку **Нет (No)**. Следующий оператор выводит диалоговое окно, показанное на рис. 4.2:

```
MsgBox "Вам видны две кнопки?" , vbYesNo, "Демонстрация кнопок"
```

При использовании этого оператора имеется одна проблема: результат пользовательского выбора никаким образом не возвращается; код по-прежнему не использует результат функции **MsgBox**.

Для выборки результата функции **Msgbox** необходимо изменить этот оператор: нужно заключить список аргументов в круглые скобки и изменить оператор так, чтобы он каким-то образом использовал результат функции. Обычно результат функции **MsgBox** присваивается какой-либо переменной и можно позже тестировать этот результат в другом операторе, чтобы определить, какую кнопку, фактически, выбрал пользователь процедуры. Следующая строка показывает оператор **MsgBox**, измененный так, что он возвращает значение:

```
Код_ответа = MsgBox ("Вам видны две кнопки?" , vbYesNo, _  
"Демонстрация кнопок")
```

При выполнении этого оператора VBA отображает то же самое диалоговое окно, показанное на рис. 4.2. Когда пользователь выбирает кнопку **Да (Yes)** или **Нет (No)**, VBA закрывает диалоговое окно и сохраняет число, соответствующее выбранной кнопке, в переменной **Код\_ответа**.

**Рис. 4.2**

Диалоговое окно **MsgBox**, демонстрирующее результат включения в список параметров аргумента кнопок



Два последних примера использования **MsgBox** иллюстрируют важный факт, касающийся функций: опуская круглые скобки вокруг списка аргументов функции, вы указываете VBA на то, что необходимо игнорировать результат функции. Когда круглые скобки опускаются, VBA интерпретирует вызов функции, как если бы это был вызов процедуры, и не возвращает результат.

Вернемся опять к примерам в начале этой главы и в функциях, использовавшихся в листинге 4.1, обратим внимание на то, что все операторы, вызывающие функцию VBA, включают круглые скобки вокруг списка аргументов функции, даже когда список аргументов содержит только один аргумент. Функции, не имеющие аргументов, не требуют круглых скобок при их вызове.

Нет смысла в игнорировании результата каждой функции VBA, и не всегда это желательно. Игнорирование результата функции полезно только при работе с функциями (такими как **MsgBox**), которые выполняют некоторую задачу,

а возвращение ими результата является только частью их работы. Результат функции игнорируют, когда хотят, чтобы функция выполняла некоторые действия, но результат этих действий необязательно использовать в коде. Функция **MsgBox**, например, выполняет отображение сообщения в диалоговом окне в качестве основной задачи. (В главе 5 показано, как использовать **MsgBox**, чтобы дать возможность пользователю программы делать выбор, влияющий на выполнение программы.)

При попытке игнорировать результат функции, не имеющей аргументов (такой, как функция **Now**), или любой другой функции, результат которой не может не использоваться, VBA отображает сообщение об одной из нескольких возможных ошибок времени исполнения в зависимости от конкретной функции, хотя обычно появляется сообщение об ошибке синтаксиса или несовпадения типов. Обычно VBA запрещает игнорирование результата функции, имя которой является ключевым словом VBA (такой, как **CStr** или других функций преобразования), и тех функций, единственным назначением которых является предоставление некоторого возвращаемого значения, например, математических.

## Использование именованных аргументов функции

Ранее в книге указывалось, что аргументы процедуры необходимо перечислять в определенном порядке. Также необходимо перечислять в определенном порядке и аргументы функции. Функции с несколькими аргументами используют положение аргумента в списке аргументов для определения того, какое значение получает конкретный аргумент. Например, для **MsgBox** первый аргумент является *сообщением*, второй — *это число и тип кнопок для диалогового окна*, третий — *заголовок диалогового окна*.

Несмотря на то, что второй аргумент **MsgBox**, определяющий командные кнопки для диалогового окна, является необязательным и был опущен во всех примерах и упражнениях до сих пор в книге, все же необходимо включать отмечающие запятые для второго аргумента в список аргументов. Пропуск отмечающих запятых для второго аргумента приводит к ошибке несовпадения типов.

Как вы могли уже заметить, легко нечаянно опустить отмечающие запятые или переставить значения аргументов в функциях, имеющих необязательные аргументы или несколько аргументов, несмотря на помощь свойства **Auto Quick** (Краткие сведения) Редактора VB. Пропуск или перестановка аргументов в списке аргументов функции могут привести к ошибкам несовпадения типов. Ошибка (что еще хуже) может быть не обнаружена.

Чтобы предотвратить ошибки программирования и сделать более легким использование функций, имеющих необязательные аргументы, VBA предоставляет альтернативу перечислению значений в списке аргументов в определенном порядке. Можно также передавать значения аргументов функции, используя *именованные аргументы* (*named arguments*) функций. Следующие строки показывают два оператора **MsgBox**, которые имеют один и тот же результат; первый оператор использует обычный метод перечисления аргументов, а второй — метод именованных аргументов (оба оператора игнорируют результат функции **MsgBox**):

```
MsgBox AnyMsg, , AnyTitle  
MsgBox Prompt:=AnyMsg, Title:=AnyTitle
```

Второй оператор использует именованные аргументы для *Prompt* (сообщение или запрос) и *Title* (определяет заголовок окна сообщения), присваивая значение каждому именованному аргументу.

Символ, который присваивает значение именованному аргументу (*:=*), не является в точности тем же обычным оператором присваивания (*=*). Если опустить двоеточие (*:*) во время присваивания значения именованному аргументу, VBA не обязательно обнаружит ошибку синтаксиса, но не может интерпретировать этот оператор правильно. Когда VBA выполняет этот оператор, он отображает одну из нескольких возможных ошибок времени исполнения, часто — ошибку несовпадения типов.

При использовании именованных аргументов нет необходимости включать отмечающие запятые для необязательных аргументов. Обратите внимание, во втором из приведенных выше операторов нет отмечающей запятой между аргументами для запроса и заголовка (в отличие от первого оператора). На самом деле, именованные аргументы не должны появляться в каком-либо определенном порядке. Во втором операторе можно поместить аргумент *Title* перед аргументом *Prompt*, как показано далее:

```
MsgBox Title:=ЗаголовокОкна, Prompt:=Сообщение
```

**MsgBox** все равно использует значение, присвоенное аргументу *Title*, в качестве заголовка диалогового окна. При использовании именованных аргументов VBA использует имя аргумента для определения того, какое значение этот аргумент представляет.

После столь долгого повествования о пользе функции **MsgBox** приведем, наконец, ее полный синтаксис.

## Синтаксис

---

```
MsgBox(Prompt [, Buttons] [, Title] [, HelpFile, Context])
```

Единственным обязательным аргументом для **MsgBox** является аргумент *Prompt*, который может быть любым строковым выражением и содержит информацию, выводимую в диалоговом окне. Все другие аргументы являются необязательными (об этом говорят квадратные скобки, в которые заключены эти аргументы вместе с разделяющими запятыми).

Аргумент *Title* содержит строку для заголовка диалогового окна. Без этого аргумента в заголовке будет выведено наименование host-приложения, из которого запускается программа на VBA.

Аргумент *HelpFile* — это строковое выражение, содержащее имя справочного файла Windows; обычно это — файл, который вы уже должны были создать с помощью Windows Help Compiler. Аргумент *Context* — это численное выражение, указывающее раздел в справочном файле, относящийся к отображаемому диалоговому окну, например, 0 обычно относится к оглавлению справочного файла.

Аргумент *Buttons* является численным выражением, которое задает количество и тип кнопок в диалоговом окне **MsgBox**. *Buttons* указывает также кнопку по умолчанию в диалоговом окне и, содержит ли это диалоговое окно стандартные значки Windows: **Critical**, **Information**, **Exclamation** или **Question** для предупредительных сообщений и запросов пользователя. В табл. 4.1 приведены значения аргументов-констант функции **MsgBox** и их назначение, а в табл. 4.2 — возвращаемые значения-константы функции.

---



Таблица 4.1. Аргументы-константы функции MsgBox

Константа	Назначение
vbAbortRetryIgnore	Отображает командные кнопки <u>С</u> топ, <u>П</u> овтор, <u>П</u> ропустить (Abort, Retry, Ignore)
vbApplicationModal	Для продолжения работы с приложением пользователь должен ответить на запрос (или закрыть) этого диалогового окна. Любые другие приложения Windows, работающие в фоновом режиме, продолжают выполняться.
vbCritical	Отображает в диалоге значок критического предупредительного сообщения (Critical Message) Windows (красный кружок).
vbDefaultButton1	Первая командная кнопка в диалоговом окне является кнопкой по умолчанию.
vbDefaultButton2	Вторая командная кнопка в диалоговом окне является кнопкой по умолчанию.
vbDefaultButton3	Третья командная кнопка в диалоговом окне является кнопкой по умолчанию.
vbDefaultButton4	Четвертая командная кнопка в диалоговом окне является кнопкой по умолчанию.
vbExclamation	Отображает значок («!») предупреждения (Warning Message); обычно используется для отображения важной информации или предупреждения, не требующего ответа.
vbInformation	Отображает значок («i») информации (Information Message); обычно используется для отображения важной информации, кроме предупреждения.
vbMsgBoxHelpButton	Добавляет к диалоговому окну кнопку Справка (Help); при щелчке на кнопке Справка открывается файл, который задан в аргументе HelpFile, в разделе, заданном аргументом Context.
vbOKCancel	Отображает кнопки ОК и Отмена (OK, Cancel).
vbOKOnly	Отображает только кнопку ОК; то же самое происходит при опускании аргумента Buttons.
vbQuestion	Отображает значок запроса (Query icon) Windows («?»); обычно используется, чтобы задать пользователю очень важный вопрос или выдать предупредительное сообщение, требующее ответа.
vbRetryCancel	Отображает кнопки Повтор и Отмена (Retry, Cancel).
vbSystemModal	При отображении диалоговое окно остается впереди других окон (даже при переключении на другое приложение), пока не будет закрыто.
vbYesNo	Отображает кнопки Да и Нет (Yes, No).
vbYesNoCancel	Отображает кнопки Да, Нет и Отмена (Yes, No, Cancel).

Таблица 4.2. Возвращаемые значения-константы функции MsgBox

Константа	Означает, что пользователь выбирает кнопку
vbAbort	Стоп (Abort)
vbCancel	Отмена (Cancel)
vbIgnore	Пропустить (Ignore)
vbNo	Нет (No)
vbOK	ОК
vbRetry	Повтор (Retry)
vbYes	Да (Yes)

Другой наиболее часто используемой для обмена данными с пользователем функцией является **InputBox**. В главе 3 был приведен краткий синтаксис этой функции, а ниже — полный:

### Синтаксис

```
InputBox(Prompt [, Title] [, Default] [, XPos][, YPos]
        [, HelpFile, Context])
```

*Prompt* — это любое строковое выражение. Аргумент *Prompt* является единственным обязательным аргументом для **InputBox**; все другие — необязательные.

*Title* — это строка, используемая в качестве заголовка для окна ввода (как и для функции **MsgBox**).

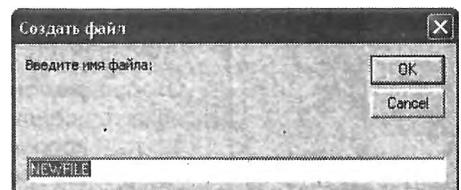
*Default* также является любым строковым выражением и используется как значение по умолчанию для пользовательского ввода.

Следующий оператор, например, запрашивает пользователя ввести имя файла и предлагает имя **NEWFILE** в качестве значения по умолчанию. Диалоговое окно, выводимое на экран этим оператором и отображающее значение по умолчанию, показано на рис. 4.3.

```
User_Input = InputBox("Введите имя файла: ", _
                      "Создать файл", "NEWFILE")
```

### Рис. 4.3

Используйте необязательные аргументы функции **InputBox** для задания значения по умолчанию



Как вы могли уже заметить, диалоговое окно **InputBox** отображается в центре экрана. Можно выводить это окно в другом месте на экране, особенно, если необходимо, чтобы были открыты и оставались видимыми другие диалоговые окна.

Аргументы *XPos* и *YPos* могут быть любыми численными выражениями. Эти аргументы позволяют указать, где в активном окне появляется окно ввода, и являются координатами верхнего левого угла диалогового окна: *XPos* — горизонтальное расстояние от левого края окна; *YPos* — это вертикальное расстояние от верхнего края окна. Оба расстояния измеряются в твипах (twips); один твип равен 1/20 точки

(точка — это измерение шрифта печати). Поскольку точка составляет  $1/72$  часть дюйма, то один твип приблизительно равен 0,0007 дюйма.

Будьте внимательны при задании положения диалогового окна **InputBox**. Вы можете задать позиции для аргументов *XPos* и *YPos* настолько большими, что диалоговое окно совсем не появится на экране, потому что его положение выйдет за правый или нижний край окна. При этом, хотя диалоговое окно и невидимо, оно является активным, ни один из элементов управления, которые вы видите на экране, не будет работать до тех пор, пока вы не отреагируете на «невидимое» диалоговое окно.

Последние два необязательных аргумента для функции **InputBox** — это *HelpFile* и *Context*. Они имеют то же назначение, что и подобные аргументы функции **MsgBox**.

Если вы задаете *HelpFile* или *Context*, необходимо задавать оба аргумента (эти аргументы заключены в одни квадратные скобки). Всякий раз, когда указывается справочный файл для окна ввода, VBA автоматически добавляет командную кнопку помощи в диалоговое окно. VBA не включает Windows Help Compiler; если вы хотите создать собственные пользовательские справочные файлы, необходимо от Microsoft получить Windows Help Compiler отдельно или использовать средства независимых поставщиков программного обеспечения.

---

При вызове **InputBox** можно использовать именованные аргументы. Для этого просто применяйте наименования аргументов: *Prompt*, *Title*, *Default*, *XPos*, *YPos*, *HelpFile* и *Context*. Следующий оператор выдает такое же диалоговое окно, что и показанное на рис. 4.3, но здесь применяются именованные аргументы:

```
User_Input = InputBox(Prompt:="Введите имя нового файла: ",  
                      Title:="Создать файл", Default:="NEWFILE")
```

Обратите внимание, что этот оператор имеет круглые скобки вокруг списка аргументов. Необходимо всегда заключать список аргументов в круглые скобки при использовании результата функции, независимо от того, используете ли вы именованные аргументы при вызове функции.

Нельзя смешивать именованные аргументы с обычным списком аргументов в одном и том же вызове функции. Необходимо использовать либо именованные аргументы, либо список обычных аргументов для каждого отдельного вызова функции.

Для определения имен аргументов функции используйте всплывающее окно **Auto Quick Info** (Краткие сведения), появляющееся при вводе имени функции (в Редакторе VBA). Можно также использовать **Object Browser** для вставки имени функции и добавления в список именованных аргументов в программном коде, как описано далее в этой главе.

## Использование других функций VBA

Встроенные функции VBA делятся на несколько категорий на основе общего назначения функций (математические, преобразования данных, даты и времени, строковые и работы с диском). В следующих разделах обсуждаются категории функций и описываются их действия.

К сожалению, невозможно описать подробно каждую функцию VBA в рамках этой книги. Однако большинство функций VBA, такие как математические функции, являются довольно ясными из их названия и не требуют под-

робного объяснения. Другие функции, такие как функции преобразования типа данных и обработки строк, описаны более подробно. VBA-функции обработки строк имеют важное значение, поэтому отдельный раздел этой главы посвящен способам их использования.

## Математические функции

VBA предоставляет стандартный набор математических функций. В табл. 4.3 приведены математические функции, имеющиеся в VBA. В этой таблице *N* означает любое численное выражение; все аргументы функций являются обязательными, если только не указано иначе.

Таблица 4.3. Математические функции VBA

Функции (аргументы)	Возвращает/действие
Abs( <i>N</i> )	Возвращает абсолютное значение <i>N</i> .
Atn( <i>N</i> )	Возвращает арктангенс <i>N</i> как угол в радианах.
Cos( <i>N</i> )	Косинус угла <i>N</i> , где <i>N</i> — это угол, измеренный в радианах.
Exp( <i>N</i> )	Возвращает константу <i>e</i> , возведенную в степень <i>N</i> . ( <i>e</i> — это основание натуральных логарифмов и она (приблизительно) равна 2,718282).
Fix( <i>N</i> )	Возвращает целую часть <i>N</i> . Fix не округляет число, а отбрасывает любую дробную часть. Если <i>N</i> является отрицательным, Fix возвращает ближайшее отрицательное целое большее, чем или равное <i>N</i> .
Int( <i>N</i> )	Возвращает целую часть <i>N</i> . Int не округляет число, а отбрасывает любую дробную часть. Если <i>N</i> является отрицательным, Int возвращает ближайшее отрицательное целое меньшее, чем или равное <i>N</i> .
Log( <i>N</i> )	Возвращает натуральный логарифм <i>N</i> .
Rnd( <i>N</i> )	Возвращает случайное число; аргумент является необязательным. Используйте функцию Rnd только после инициализации VBA-генератора случайных чисел оператором Randomize.
Sgn( <i>N</i> )	Возвращает знак числа: -1, если <i>N</i> — отрицательное; 1, если <i>N</i> — положительное; 0, если <i>N</i> равно 0.
Sin( <i>N</i> )	Возвращает синус угла; <i>N</i> — это угол, измеренный в радианах.
Sqr( <i>N</i> )	Возвращает корень квадратный из <i>N</i> . VBA отображает ошибку времени исполнения, если <i>N</i> — отрицательное.
Tan( <i>N</i> )	Возвращает тангенс угла; <i>N</i> — угол в радианах.

Функции **Fix** и **Int** укорачивают целые, то есть, они отбрасывают дробную часть числа без округления. Единственное различие между функциями **Fix** и **Int** — это то, как они обрабатывают отрицательные числа.

Дополнительные тригонометрические функции можно выводить из базовых математических функций VBA. Например, если необходимо вычислить котангенс угла, для его нахождения можно использовать формулу  $1/\tan(x)$ . В этой главе описывается, как писать собственные функции для создания производных математических функций.

Функции преобразования данных

Visual Basic предоставляет несколько функций для преобразования одного типа данных в другой. Используйте эти функции для устранения ошибок несовпадения типов и обеспечения явного контроля за типами данных в выражениях.

Например, при получении сообщения об ошибке несовпадения типов в определенном выражении можно преобразовать значения в выражении в типы, совместимые друг с другом, используя функции преобразования. Или же можно сохранять результат выражения в диапазоне численного типа **Single** (большинство численных выражений имеют результатом значение типа **Double**); в таком случае следует использовать функцию **CSng** для преобразования результата выражения в число типа **Single**, как показано в листинге 4.2.

Листинг 4.2. Использование функции CSng

```
1: Sub CSngDemo()  
2:   Dim d_Var1 As Double  
3:   Dim s_Var2  
4:   d_Var1 = 412  
5:   s_Var2 = CSng(352 / 17)  
6:   MsgBox TypeName(s_Var2)  
7: End Sub
```

Проверьте, как работает код листинга 4.2.

В табл. 4.4 приведены функции преобразования данных в VBA. В этой таблице N — это любое численное, S — любое строковое, а E — выражение любого типа. Аргументы каждой функции являются обязательными, если не указано иначе.

Таблица 4.4. Функции преобразования данных

Функция (аргументы)	Возвращает/действие
Asc (S)	Возвращает число кода символа, соответствующее первой букве строки S. Буква «А», например, имеет код символа 65.
Chr (N)	Возвращает строку из одного символа, соответствующего коду символа N, который должен быть числом между 0 и 255, включительно. Код символа 65, например, возвращает букву «А».
Format (E, S)	Возвращает строку, содержащую значение, представленное выражением E, в формате в соответствии с инструкциями, содержащимися в S.
Hex (N)	Возвращает строку, содержащую шестнадцатичное представление N.

Функция (аргументы)	Возвращает/действие
<b>Oct</b> (N)	Возвращает строку, содержащую восьмеричное представление N.
<b>RGB</b> (N, N, N)	Возвращает целое типа <b>Long</b> , представляющее значение основных цветов изображения. N в каждом аргументе должно быть целым в диапазоне 0–255, включительно. Аргументы (слева направо) — это значения для красного, зеленого и синего цвета.
<b>Str</b> (N)	Возвращает строку, эквивалентную численному выражению N.
<b>Val</b> (S)	Возвращает численное значение, соответствующее числу, представленному строкой S, которая должна содержать только цифры и одну десятичную точку, иначе VBA не может преобразовать ее в число. Если VBA не может преобразовать строку в S, то функция <b>Val</b> возвращает 0.
<b>CBool</b> (N)	Возвращает <b>Boolean</b> — эквивалент численного выражения N.
<b>CByte</b> (E)	Возвращает численное значение типа <b>Byte</b> (от 0 до 255); E — любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
<b>CCur</b> (E)	Возвращает численное значение типа <b>Currency</b> ; E — любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
<b>CDate</b> (E)	Возвращает значение типа <b>Date</b> . E может быть любым допустимым выражением (строкой или числом), представляющим дату в диапазоне 1/1/100 — 12/31/9999, включительно.
<b>CDbl</b> (E)	Возвращает численное значение типа <b>Double</b> ; E — любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
<b>Cint</b> (E)	Возвращает численное значение типа <b>Integer</b> ; E — любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
<b>CLng</b> (E)	Возвращает численное значение типа <b>Long</b> ; E — любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
<b>CSng</b> (E)	Возвращает численное значение типа <b>Single</b> ; E — любое допустимое численное или строковое выражение, которое может быть преобразовано в число.
<b>CStr</b> (E)	Возвращает значение типа <b>String</b> ; E — любое допустимое численное или строковое выражение.
<b>CVar</b> (E)	Возвращает значение типа <b>Variant</b> ; E — любое допустимое численное или строковое выражение.

Наиболее часто используемые функции — это функции (объединенные в конце табл. 4.4 в группу), начинающиеся с буквы C (от слова *conversion*), за которыми следует сокращение имени типа: **CStr**, **CSng**, **CDbl** и так далее.

## Функции даты и времени

VBA-функции даты и времени обычно используются для получения текущей даты и времени, разбиения значения даты на ее составляющие части или для преобразования строк и чисел в значения типа **Date**.

В таблице 4.5 приведены VBA-функции даты и времени и их действие. В этой таблице N — любое допустимое численное выражение, а D — любое допустимое выражения типа **Date** (включая значения типа **Date**, числа или строки, которые VBA может преобразовать в дату). Все аргументы функций в этой таблице являются обязательными, если не указано иначе.

**Таблица 4.5. Функции даты и времени**

Функции (аргументы)	Возвращает/действие
Date	Возвращает системную дату. Можно также использовать эту функцию как процедуру для установки системных часов компьютера. Более подробно можно узнать из справочной системы VBA.
Time	Возвращает системное время компьютера как значение типа <b>Date</b> . Можно также использовать эту функцию как процедуру для установки системных часов. Более подробно можно узнать из справочной системы VBA.
Now	Возвращает системную дату и время.
Year(D)	Возвращает целое, являющееся частью выражения типа <b>Date</b> и содержащее год. Год возвращается как число между 100 и 9999.
Month(D)	Возвращает целое, являющееся частью выражения типа <b>Date</b> , содержащее месяц. Месяц возвращается как число между 1 и 12, включительно.
Day(D)	Возвращает целое, являющееся частью выражения типа <b>Date</b> и содержащее день. День возвращается как число между 1 и 31, включительно.
Weekday(D)	Возвращает целое, содержащее день недели для выражения типа <b>Date</b> . День недели возвращается как число между 1 и 7, включительно; 1 — это воскресенье, 2 — понедельник и так далее.
WeekdayName (N1, B, N)	Возвращает строку с наименованием дня недели, номер которого задается параметром N1.
Hour(D)	Возвращает целое, содержащее часы как часть времени, содержащегося в выражении типа <b>Date</b> . Часы возвращаются как число между 0 и 23, включительно. Если выражение D не содержит значения времени, то <b>Hour</b> возвращает 0.
Minute(D)	Возвращает целое, содержащее минуты как часть времени в выражении типа <b>Date</b> . Минуты возвращаются как число между 0 и 59, включительно. Если выражение D не содержит значения времени, <b>Minute</b> возвращает 0.

Функции (аргументы)	Возвращает/действие
Second(D)	Возвращает целое, содержащее секунды как часть времени в выражении типа <b>Date</b> . Секунды возвращаются как число между 0 и 59, включительно. Если выражение D не содержит значения времени, <b>Second</b> возвращает 0.
DateAdd(S, N, D)	Возвращает значение [тип Variant (Date)], содержащее дату, к которой добавлен заданный интервал времени.
DateDiff(S, D1, D2[, N1 [, N2 ]])	Возвращает значение [тип Variant (Long)] числа временных интервалов между двумя определенными датами.
DatePart(S, D[, N1 [, N2 ]])	Возвращает определенную часть [тип Variant (Integer)] заданной даты.
DateSerial(N, N, N)	Возвращает значение последовательной даты для заданной даты. Слева направо аргументы представляют год, месяц и день. Аргумент года должен быть целым числом между 100 и 9999, месяца — между 1 и 12, дня — между 1 и 31 (все диапазоны являются включающими).
TimeSerial (N, N, N)	Возвращает значение последовательного времени. Слева направо аргументы представляют часы, минуты и секунды. Аргумент часов должен быть целым числом между 0 и 23, аргументы минут и секунд должны оба быть числами 0 и 59 (все диапазоны являются включающими).
DateValue(E)	Возвращает значение типа <b>Date</b> , эквивалентное дате, заданной аргументом E, который должен быть строкой, числом или константой, представляющей дату.
TimeValue(E)	Возвращает значение типа <b>Date</b> , содержащее время, заданное аргументом E, который может быть строкой, числом или константой, представляющей время.
Timer	Возвращает число, представляющее количество секунд от полуночи в соответствии с системным временем компьютера.

Некоторые из перечисленных функций уже использовались в примерах этой книги, другие — будут использоваться далее.

## Строковые функции

Строковые функции VBA часто применяются для нахождения заданных строк внутри других строк, для сравнения одной строки с другой и копирования выбранных частей строк. Строковые функции VBA используются довольно часто, потому что строковые данные очень важны и встречаются в каждом приложении VBA — Word, Excel, Access или другом host-приложении VBA. Часто необходимо манипулировать строковыми данными, полученными как пользовательский ввод функцией **InputBox**. В других случаях строковые данные появляются в коде VBA как имена файлов для документов Word, рабочих книг Excel, баз данных Access и других типов данных, сохраняемых в файлах на дисках.



В Word важность манипулирования строковыми данными очевидна: данные, сохраняемые в документах Word, в основном являются ничем иным, как строковыми данными. Строковые данные также важны в Excel в качестве имен рабочих листов, именованных диапазонов данных и так далее. В этом разделе приводятся имеющиеся в VBA строковые функции; в последующем разделе более подробно описывается, как использовать наиболее важные и полезные строковые функции.

В табл. 4.6, где приведены основные строковые функции VBA, N — это любое допустимое численное выражение, а S — это любое допустимое строковое выражение. Все аргументы функций являются обязательными, если не указано иначе.

Таблица 4.6. Строковые функции

Функция (аргумент)	Возвращает/действие
InStr([N1,] S1, S2[, N2])	Возвращает положение S2 в S1. N1 — начальное положение для поиска; N2 определяет тип сравнения. N1 и N2 необязательны. Если N2 опускается, то для поиска используется текущая установка <b>Option Compare</b> .
InStrRev(S1, S2 [, N1[, N2]])	Возвращает позицию появления строки S2 внутри S1, в направлении от конца (или N1) к началу строки. N2 определяет тип сравнения. Если N2 опускается, то для поиска используется текущая установка <b>Option Compare</b> .
LCase(S)	Возвращает строку (тип <b>String</b> ), содержащую копию S со всеми символами верхнего регистра, преобразованными в символы нижнего регистра.
Left(S, N)	Возвращает строку; копирует N символов из S, начиная с левого крайнего символа S.
Len(S)	Возвращает число символов в S, включая начальные и конечные пробелы.
LTRim(S)	Возвращает копию строки S после удаления символов пробела из левой части строки (начальные пробелы).
Mid(S, N1, N2)	Возвращает строку; копирует N2 символов из S, начиная с позиции символа в S, заданной аргументом N1. N2 является необязательным; если N2 опущен, то Mid возвращает все символы в строке S от позиции N1 до конца строки.
Right(S, N)	Возвращает значение типа <b>String</b> ; копирует N символов из S, начиная с правого крайнего символа S. Например, Right(«outright», 5) возвращает строку «right».
RTTrim(S)	Возвращает копию строки S после удаления символов пробела из правой части строки (конечные символы).
Space(N)	Возвращает строку пробелов длиной N символов.

Функция (аргумент)	Возвращает/действие
StrComp(S1, S2, N)	Сравнивает S1 с S2 и возвращает число, обозначающее результат сравнения: -1, если S1 < S2; 0, если S1 = S2; 1, если S1 > S2. N является необязательным и указывает, следует ли выполнять сравнение с учетом регистра. Если N опускается, строки сравниваются с использованием текущей установки <b>Option Compare</b> .
StrConv(S, N)	Возвращает строку, преобразованную в новую форму в зависимости от числового кода, заданного аргументом N. VBA предоставляет внутренние константы для использования с функцией StrConv; наиболее полезными являются: vbProperCase (преобразует строку так, что каждая буква, начинающая слово, становится заглавной), vbLowerCase (преобразует строку в буквы нижнего регистра) и vbUpperCase (преобразует строку в буквы верхнего регистра).
String(N, S)	Возвращает строку длиной N символов, состоящую из символа, заданного первым символом в S. Например, String(5, «x») возвращает строку «xxxxx».
Trim(S)	Возвращает копию строки S после удаления начальных и конечных символов пробела из этой строки.
UCase(S)	Возвращает S со всеми символами нижнего регистра, преобразованными в символы верхнего регистра.

Несколько перечисленных в табл. 4.6 функций преобразования типа данных относятся также к манипулированию строками: **Chr**, **Format**, **CStr**, в частности.

## Использование функций для манипулирования строками

Манипулирование строковыми данными является важной частью многих программ, особенно программ, которые взаимодействуют с пользователем. Интерактивные программы должны манипулировать строками по двум причинам: для формулировки сообщений, которые необходимо отображать для пользователя, и из-за того, что входные данные пользователя вводятся в вашу программу как строковые данные.

Чем лучше вы научитесь манипулировать строками, тем вероятнее, что вы будете способны отображать приемлемые, согласованные сообщения для пользователя вашей программы. Кроме того, чем лучше вы умеете манипулировать строками, тем вероятнее, что вы сможете успешно анализировать строки, вводимые пользователем вашей программы.

VBA имеет различные функции в качестве средств, помогающих манипулировать строковыми данными. В этом и следующем разделе описывается, как использовать наиболее важные строковые функции и некоторые из функций преобразования данных для выполнения как простых, так и сложных операций над строковыми данными в программах.

## Удаление ненужных символов

Иногда строки в программе содержат ненужные символы пробелов в конце или в начале строки. Эти начальные и конечные пробелы появляются по разным причинам.

Один из наиболее распространенных случаев появления начальных или конечных пробелов — когда используется функция **InputBox** для получения данных ввода от пользователя. **InputBox** возвращает весь текст, вводимый пользователем, включая любые лишние символы пробела. Если пользователь вводит лишние символы пробела перед или после фактического значения ввода, функция **InputBox** возвращает эти символы как часть вводимой строки.

Другая распространенная причина начальных и конечных символов — когда используется содержимое строковой переменной фиксированной длины. Строка фиксированной длины всегда имеет одну и ту же длину, и VBA при необходимости дополняет данные, присвоенные этой строковой переменной (обычно конечными пробелами), чтобы заполнить объявленную длину строки. В результате всякий раз, когда используется строка фиксированной длины, возникает вероятность того, что она будет содержать конечные пробелы.

Посторонние начальные или конечные пробелы в строке могут вызвать различные проблемы: некоторые — только «косметические», другие — более серьезные. Когда строки транслируются для вывода на экран, лишние пробелы могут привести к большим промежуткам в тексте, при этом отображается неприемлемый и трудночитаемый текст. В других случаях посторонние начальные и конечные пробелы могут повлиять на сравнения строк, на сообщаемую длину строки и на некоторые другие факторы. Это может привести к тому, что строковое значение не будет использоваться в каком-либо другом выражении или процедура выдаст ошибочный результат. В любом из этих случаев может возникнуть серьезная проблема.

В VBA имеются три функции, специально предназначенные для удаления нежелательных начальных и конечных пробелов из строки. Первая функция — **RTrim** — удаляет символы пробела из правой части строки (конечные пробелы). Вторая функция — **LTrim** — удаляет символы пробела из левой части строки (начальные пробелы). Третья функция — **Trim** — удаляет как начальные, так и конечные пробелы из строки.

Эти функции, на самом деле, не изменяют строку; вы передаете строку, из которой должны быть вырезаны пробелы, как аргумент функции, и функция возвращает *копию* строки с удаленными лишними пробелами. Процедура в листинге 4.3 демонстрирует использование функций удаления пробелов из строки.

### Листинг 4.3. Демонстрация функций **RTrim**, **LTrim** и **Trim**

```
1: Sub TrimDemo()  
2:   Dim ExSpace As String  
3:   ExSpace = "      mad dog      "  
4:   msg = "{" & ExSpace & "}"  
5:   MsgBox "{" & ExSpace & "}", , "До изменения"  
6:   MsgBox "{" & RTrim(ExSpace) & "}", , "Без правых пробелов"  
7:   MsgBox "{" & LTrim(ExSpace) & "}", , "Без левых пробелов"  
8:   MsgBox "{" & Trim(ExSpace) & "}", , "Без пробелов"  
9: End Sub
```

В строке 3 переменной **ExSpace** (объявленной в строке 2) присваивается строка « mad dog » (с начальными и конечными пробелами). В строке 4 используется функция **MsgBox** для отображения еще неизменной строки **ExSpace**. Строковое выражение для аргумента **MsgBox** конкатенирует пару фигурных скобок вокруг строки **ExSpace**, чтобы показать, имеются ли начальные или конечные пробелы в строке. Строки 5–7 листинга 4.3 — каждая использует одну из трех функций для удаления начальных и конечных пробелов из строки **ExSpace**; каждая строка использует **MsgBox** для отображения результата функции удаления пробелов. Результат работы кода представлен на рис. 4.4.

**Рис. 4.4**

Диалоговые окна как результат выполнения кода листинга 4.3



Следует использовать функцию **Trim** для удаления начальных и конечных пробелов из строк перед их сравнением; этим предотвращается влияние начальных и конечных пробелов на сравнение строк, что обычно не является существенным для пользователя, но имеет значение для VBA.

## Определение длины строки

Часто бывает необходимо знать длину строки (какое количество символов находится в строке), например, при форматировании сообщений для пользователя или при форматировании строковых данных, вводимых процедурой в рабочий лист Excel или документ Word. VBA имеет функцию **Len**, позволяющую получать длину строки. Следующий оператор показывает общий синтаксис для этой функции:

### Синтаксис

---

`Len(VString)`

*VString* — любое допустимое строковое выражение VBA. Например, следующий оператор показывает использование функции в операторе присваивания:

`StrLen = Len("VBA")`      'возвращает 3

---

Как и ранее, строки фиксированной длины представляют особый случай. Поскольку строка фиксированной длины имеет всегда одну и ту же длину, функция **Len** всегда возвращает объявленную длину строки. Например, при объявлении **FirstName** в качестве строковой переменной, имеющей длину 20 символов, функция **Len** всегда возвращает 20 как длину строки в **FirstName**;

даже если в переменной сохранено имя **Bob**, имеющее длину только 3 символа, все остальные символы из 20 являются символами пробела.

Обычно целью использования функции **Len** является определение того, сколько символов имеется в строке, исключая начальные или конечные пробелы. Вернемся к примеру с **FirstName**; если необходимо узнать фактическую длину имени, сохраненного в переменной, используйте оператор, подобный следующему:

```
NameLen = Len(Trim(FirstName))
```

В этом операторе функция **Trim** удаляет любые начальные или конечные пробелы из строки в переменной, а функция **Len** сообщает длину результирующей строки. Если **FirstName** содержит «Москва», то переменная **NameLen** будет иметь результатом значение 6.

## Сравнение и поиск строк

В главе 3 было описано, как сравнивать строки, используя операторы сравнения, и объяснялось действие установок **Option Compare Binary** и **Option Compare Text**.

VBA имеет также две функции для сравнения строк. Первая функция — **StrComp** просто сравнивает две разные строки. При некоторых обстоятельствах лучше использовать **StrComp** вместо операторов сравнения (**=**, **<**, **>**) для сравнения строк, потому что **StrComp** позволяет указывать, следует ли выполнять бинарное или текстовое сравнение (независимо от установки **Option Compare** модульного уровня) для этого определенного сравнения.

Например, если необходимо, чтобы большинство строковых сравнений игнорировали регистр (верхний или нижний) символов при сравнении строк, добавьте оператор **Option Compare Text** к этому модулю. После этого можно выполнить конкретное сравнение строк, являющееся чувствительным к регистру: используйте VBA-функцию **StrComp** и задавайте сравнение с учетом регистра.

### Использование функции StrComp

Общий синтаксис функции **StrComp** следующий:

#### Синтаксис

---

```
StrComp(String1, String2 [, Compare])
```

*String1* и *String2* — любые два строковых выражения, которые необходимо сравнить. Необязательный аргумент *Compare* может быть любой из следующих предопределенных констант:

- **vbBinaryCompare** — для сравнения двух строк с помощью бинарного (с учетом регистра) сравнения;
- **vbTextCompare** — для сравнения двух строк с помощью текстового (без учета регистра) сравнения;
- **vbDatabaseCompare** — имеет значение только в Microsoft Access; в этом случае строковое сравнение использует любой метод сравнения, установленный для текущей базы данных. При использовании константы **vbDatabaseCompare** в каком либо другом программном продукте VBA, а не в Access VBA, отображается сообщение об ошибке времени исполнения, констатирующее наличие недопустимого аргумента процедуры.

Если аргумент *Compare* опускается, **StrComp** использует текущую установку **Option Compare**. При выполнении **StrComp** сравниваются две строки с использованием заданного метода сравнения и возвращается одно из следующих значений:

- -1, если String1 меньше String2
- 0, если String1 и String2 равны друг другу
- 1, если String1 больше String2

Процедура в листинге 4.4 демонстрирует использование функции **StrComp**.

---

#### Листинг 4.4. Демонстрация функции **StrComp**

---

```
1: Sub Demo_STRComp()  
2:   Const Умолчание = " STRComp"  
3:   Dim СтрокаВвода As String  
4:   СтрокаВвода = InputBox(Prompt:="Введите некоторый текст:", _  
5:                           Title:="Сравнение строк", _  
6:                           Default:= Умолчание)  
7:   MsgBox StrComp(СтрокаВвода, Умолчание, vbTextCompare)  
8: End Sub
```

---

В строке 2 объявляется константа для использования в качестве текста по умолчанию в последующем операторе **InputBox**. В строке 3 объявляется строковая переменная **СтрокаВвода** для сохранения результата функции **InputBox**. Строки 4–6 являются одним оператором, вызывающим функцию **InputBox** для получения строки от пользователя. Вызов функции **InputBox** использует именованные аргументы и задает заголовок диалогового окна и предлагаемое значение по умолчанию. Результат функции **InputBox** присваивается переменной **СтрокаВвода**. Строка 7 содержит вызов функции **StrComp**; оператор использует **MsgBox** для вывода возвращаемого значения функции **StrComp** непосредственно на экран. В результате код определяет, принял ли пользователь текст по умолчанию или ввел отличную от этого текста строку.

#### Использование функции **InStr**

Другая VBA-функция сравнения строк, **InStr**, дает возможность определить, содержит ли одна строка другую строку. Эта функция полезна в ряде случаев. Например, используйте **InStr**, если необходимо определить, содержит ли введенная пользователем строка определенное слово. Другой пример: используйте **InStr**, если необходимо определить, содержит ли строка символы; не позволяющие преобразовать ее в число.

Синтаксис функции **InStr**:

#### Синтаксис

---

**InStr**([Start, ] String1, String2 [, Compare])

*String1* и *String2* — любые допустимые строковые выражения. **InStr** проверяет, содержится ли *String1* в *String2*. Необязательный аргумент *Start* является любым числовым выражением; этот аргумент (если используется) указывает положение символа в *String1*, с которого должна начинаться проверка. Необязательный аргумент *Compare* определяет, должна ли **InStr** использовать двоичное или текстовое сравнение. Допустимыми значениями для аргумента *Compare* функции **InStr** являются те же предопределенные константы, которые используются функцией **StrComp**: **vbBinaryCompare**, **vbTextCompare** и **vbDatabaseCompare**.

---

Функция **InStr** возвращает число, обозначающее положение символа в *String1*, где было обнаружено *String2*; если **InStr** не находит *String2* в *String1*, то она возвращает 0. Если *String1* (или *String2*) равно **Null**, то **InStr** возвращает **Null**.

В листинге 4.5 демонстрируется использование **InStr**.

#### Листинг 4.5. Демонстрация функции **InStr**

```
1: Sub Demo_InStr()  
2:   Const Умолчание = "InStr"  
3:   Dim СтрокаВвода As String  
4:   СтрокаВвода = InputBox(Prompt:="Введите некоторый текст:", _  
5:                           Title:="Сравнение строк", _  
6:                           Default:= Умолчание)  
7:   MsgBox InStr(1, СтрокаВвода, Умолчание, vbTextCompare)  
8: End Sub
```

В строке 7 проверяется, содержится ли в строке **СтрокаВвода** строка, обозначенная константой **Умолчание**. Используется текстовое сравнение, указано начальное положение для поиска. **InStr** начинает поиск строки **Умолчание** с первого символа строки **СтрокаВвода**.

### Разбиение строки на меньшие части

Во многих процедурах бывает необходимо разбить строку на составляющие части. Например, может понадобиться проанализировать вводимую пользователем строку, чтобы определить, содержит ли она более одного слова и, если содержит, разделить ее на отдельные слова. Примеры такой обработки строк встречаются в последующих главах.

#### Функция **Left**

VBA имеет три функции для выделения подстрок из больших строк. (*Подстрока* — это любая строка, которая является или может быть частью большей строки). Первая из них — это функция **Left**, возвращающая копию определенной части строки. Далее следует синтаксис функции **Left**:

#### Синтаксис

---

**Left**(*String*, *Length*).

Здесь *String* — любое допустимое строковое выражение, а *Length* — любое численное выражение. Функция **Left** возвращает копию *String*, начиная с первого символа и включая количество символов, заданных с помощью *Length*. Если *Length* является числом, большим, чем фактическая длина *String*, то **Left** возвращает все строковое выражение *String*.

---

В следующем операторе **Left** копирует первые 16 символов строки **OldStr** и возвращает эти символы как строку; этот оператор присваивает результат функции **Left** переменной **NewStr**. Если **OldStr** содержит строку «В пустыне чахлой и скупой», то **NewStr** будет содержать «В пустыне чахлой» (после выполнения этого оператора).

```
NewStr = Left(OldStr, 16)
```

## Функция Right

Другая VBA-функция подстроки — это функция **Right**. Далее следует синтаксис **Right**:

### Синтаксис

---

`Right(String, Length)`

Здесь *String* представляет любое допустимое строковое выражение, а *Length* — любое численное выражение. Функция **Right** возвращает копию *String*, начиная с последнего символа в строке и включая справа налево количество символов, указанное с помощью *Length*. Если *Length* является числом, большим, чем фактическая длина *String*, то **Right** возвращает все строковое выражение *String*. Функция **Right** всегда копирует символы от конца строки, действуя в направлении к началу строки. Если *Length* является числом, большим, чем фактическая длина *String*, то **Left** возвращает все строковое выражение *String*.

---

В следующем операторе функция **Right** возвращает последние восемь символов строки **OldStr**. Если переменная **OldStr** содержит строку «В пустыне чахлой и скупой», этот оператор сохраняет строку «и скупой» в переменной **NewStr**.

```
NewStr = Right(OldStr, 8)
```

В бизнес-приложениях функция **Right** может быть полезна, например, при выделении из некоторого ключевого поля номера накладной. Для того, чтобы создать уникальный номер накладной, которая создается на складе с кодом, например «1011», за определенную дату, можно сформировать номер как строку вида:

```
'1011101022001002'
```

Здесь первые четыре символа обозначают код склада, следующие восемь символов содержат дату, а последние три — номер накладной, начиная с первой в текущий день. Если код склада всегда содержит четыре символа, то для быстрого (с смысле краткости кода) выделения любого значения из всей строки можно использовать функцию **Mid** (см. далее). Если же код склада может содержать различное количество символов, то удобнее пользоваться функцией **Right**.

## Функция Mid

Можно извлечь подстроку из середины строки, а не из правой или левой ее части. Такая ситуация может возникнуть при извлечении отдельных слов из строки текста. Для извлечения подстроки из середины другой строки можно использовать функцию **Mid**. Функция **Mid** имеет следующий общий синтаксис:



## Синтаксис

---

```
Mid(String, Start [, Length])
```

Здесь *String* — любое строковое выражение, тогда как *Start* и *Length* — это любые численные выражения. Функция **Mid** возвращает копию *String*, начиная с положения символа в *String*, задаваемого с помощью аргумента *Start*. Необязательный аргумент *Length* определяет количество копируемых в **Mid** символов из *String*. Если *Start* содержит большее число, чем фактическая длина *String*, то **Mid** возвращает пустую строку.

---

Следующий оператор показывает пример функции **Mid**:

```
NewStr = Mid(OldStr, 10, 5)
```

Если переменная **OldStr** содержит строку «Беспечный сын природы», то этот оператор сохраняет строку «сын» в переменной **NewStr**.

### Использование символов, которые нельзя ввести с клавиатуры

Иногда необходимо включить в строку какой-либо символ, для которого нет соответствующей клавиши на клавиатуре, например, букву греческого языка, символ для йены или символ авторского права.

Может потребоваться также включить какой-либо символ, который уже имеет особое значение для VBA, такой как символ кавычек ("). Вы не можете включать символы, подобные кавычкам, непосредственно в строку, потому что VBA всегда «подразумевает», что этот символ начинает или заканчивает строку. Следующий оператор, например, приводит к ошибке времени исполнения или ошибке синтаксиса:

```
MsgBox "This "cannot" work"
```

Хотя этот оператор может предназначаться для вывода на экран строки, VBA не может выполнить его. Поскольку кавычки (") указывают VBA на то, что литеральная строка либо начинается, либо заканчивается, VBA разбивает вышеприведенный аргумент **MsgBox** на три части: строку **"This "**, имя переменной **cannot** и еще одну строку **"work"**.

Чтобы включить в строку символы, которые невозможно ввести с клавиатуры, или которые имеют особое значение для VBA, используйте VBA-функцию **Chr**. Функция **Chr** имеет следующий синтаксис:

## Синтаксис

---

```
Chr(Charcode)
```

Здесь *Charcode* — любое численное выражение, являющееся допустимым кодом для набора символов, используемого компьютером. Аргумент *Charcode* должен быть числом от 0 до 255. Как вы помните из обсуждения двоичного и текстового сравнения строк, компьютер сохраняет буквы во внутренней памяти как числа и использует схему, в которой каждый символ имеет собственный уникальный номер. Функция **Chr** принимает код отдельного символа в качестве аргумента и возвращает строку, содержащую соответствующий этому коду символ.

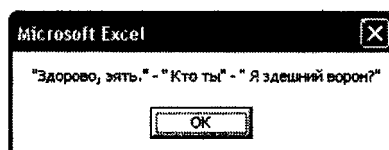
---

Чтобы просмотреть список кодов, которые распознает VBA, и соответствующих им символов, откройте справочную систему VBA и найдите раздел **Character sets**. Например, при поиске кода символа кавычек (") обнаруживаем, что кавычки имеют код 34. Используя функцию **Chr** для предоставления кавычек, следующий оператор отображает диалоговое окно, приведенное на рис. 4.5:

```
MsgBox Chr(34) & "Здорово, зять." & Chr(34) & " - " & Chr(34) & _  
" Кто ты" & Chr(34) & " - " & Chr(34) & _  
" Я здешний ворон?" & Chr(34)
```

Рис. 4.5

Пример использования функции **Chr**



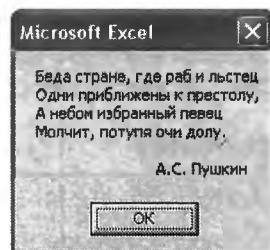
Можно также управлять форматированием отображаемых сообщений, добавляя специальные символы к строкам. Один из символов, которые можно получить с помощью функции **Chr**, — это символ возврата каретки (код символа 13). Это — символ, генерируемый компьютером всякий раз при нажатии клавиши **Enter** на клавиатуре и указывающий на начало новой строки при использовании его в тексте. Как показано на рис. 4.6, следующий оператор использует **Chr** для добавления символа возврата каретки в конкатенированную строку так, чтобы полученное диалоговое окно содержало несколько строк.

Диалоговое окно, показанное на рис. 4.6, может быть получено с помощью следующего (одного!) оператора:

```
MsgBox "Беда стране, где раб и льстец" & Chr(13) & _  
"Одни приближены к престолу," & Chr(13) & _  
"А небом избранный певец" & Chr(13) & _  
"Молчит, потупя очи долу." & Chr(13) & Chr(13) & _  
" А.С. Пушкин" & Chr(13)
```

Рис. 4.6

Использование функции **Chr** с константой 13 в качестве параметра открывает возможности форматирования текста в окне функции **MsgBox**



Поскольку символы, используемые для начала новой строки, являются очень важными при форматировании сообщений и других строковых данных, которыми манипулируют VBA-процедуры, VBA имеет несколько предопределенных констант для этих символов, чтобы не было необходимости использовать функцию **Chr**:

- **vbCr** — символ возврата каретки (код символа 13). Эта константа является эквивалентом выражения: **Chr(13)**. Включение **vbCr** в строку приводит к тому, что VBA и большинство приложений Windows начинают новую строку при выводе строки на экран. В некоторых случаях, например, при пересылке строки на принтер, символ возврата каретки просто приводит к перемещению курсора в начало текущей строки без перехода к следующей строке.
- **vbLf** — символ смещения на одну строку (код символа 10). Эта константа эквивалентна выражению: **Chr(10)**. Включение **vbLf** в строку приводит к тому, что VBA и большинство приложений Windows начинают новую строку при отображении строки. В некоторых случаях, например, при пересылке строки на принтер, символ смещения на одну строку просто приводит к перемещению курсора к следующей строке без возврата к левому краю области печати.
- **vbCrLf** — символ возврата каретки/смещения на одну строку; эквивалентен выражению: **Chr(13) & Chr(10)**. Помещение этого символа в строку приводит к тому, что VBA начинает новую строку при отображении строки. Например, при пересылке текста на принтер или для текстовых файлов DOS-формата, необходимо использовать **vbCrLf** для перемещения курсора к новой строке и к левому краю области печати.
- **vbNewLine** представляет символ(ы), используемый при создании новой строки для программной платформы, в которой выполняется ваша процедура VBA. Применяйте константу **vbNewLine** в процедурах, которые должны использоваться либо в Windows-, либо в Macintosh-версиях MS приложений.
- **vbTab** — символ табуляции (код символа 9). Этот символ создается при нажатии на клавишу **Tab** на клавиатуре. Константа **vbTab** эквивалентна выражению: **Chr(9)**. Можно включать символы табуляции в строки для выравнивания данных в столбцах.

С учетом вышесказанного последний оператор можно переписать в виде:

```
MsgBox "Беда стране, где раб и льстец" & vbCr & _  
      "Одни приближены к престолу," & vbCr & _  
      "А небом избранный певец" & vbCr & _  
      "Молчит, потупя очи долу." & vbCr & vbCr & _  
      "А.С. Пушкин" & vbCr
```

## Форматирование значений данных

Хотя VBA может автоматически преобразовывать любой тип данных в строку для отображения с помощью функции **MsgBox** или для вставки в рабочий лист Excel (документ Word или соответствующий объект другого приложения), формат данных, который выбирает VBA, может не совпадать с желаемым.

При преобразовании числа в строку VBA не добавляет в строку разделитель тысяч, символы доллара или другое числовое форматирование. Кроме того, если число очень большое или очень малое, VBA создает строку, представляющую это число в экспоненциальном формате. Например, VBA преобразует число **3145.25** в строку **«3145.25»**. Если это число представляет сумму в долларах, может оказаться предпочтительнее преобразовать его в строку, содержащую символ доллара и разделитель тысяч: **«\$3,145.25»**.

Аналогично, при преобразовании дат VBA всегда использует короткий формат даты и времени, используемый операционной системой компьютера, и всегда отображает и дату, и время. Вы можете использовать другой формат даты или времени или отображать только дату или только время.

Для получения почти любого формата дат при преобразовании чисел или дат в строки можно использовать функцию **Format**; можно даже использовать функцию **Format** для форматирования строковых данных в соответствии с определенным шаблоном. Можно также создавать пользовательские экранные форматы, если вам необходимо, чтобы данные появлялись в каком-либо особом формате.

VBA-функция **Format** идентична функции **Format** в Excel и использует те же символы-заполнители форматирования данных, что Excel и Access.

Синтаксис оператора **Format** следующий (приведен из справочной системы VBA):

### Синтаксис

**Format**(*Expression*[, *Format*[, *Firstdayofweek*[, *Firstweekofyear*]])

Здесь аргументы означают:

<i>Expression</i>	любое допустимое выражение (обязательный);
<i>Format</i>	допустимое выражение именованного или определенного пользователем формата (необязательный);
<i>Firstdayofweek</i>	константа, которая определяет первый день недели (необязательный);
<i>Firstweekofyear</i>	константа, которая определяет первую неделю года (необязательный).

Для аргументов *Firstdayofweek* и *Firstweekofyear* в VBA имеются именованные константы, о которых можно узнать из справочной системы VBA в разделе Date Constants.

Чтобы использовать функцию **Format**, можно либо задать предопределенный формат (называемый *именованным форматом (named format)*), либо создать образ определенного формата, используя комбинации особой группы символов, называемых *символами-заполнителями (placeholders)*. Используйте образ, который создается с помощью символов-заполнителей, если ни один из именованных форматов не отвечает вашим запросам. Примером этих методов использования функции **Format** могут быть следующие операторы **MsgBox** (каждый отображает одно и то же диалоговое окно, показанное на рис. 4.7); первый оператор использует именованный формат, второй — образ с символами-заполнителями:

```
MsgBox Format(#2/27/1976#, "Long Date")
MsgBox Format(#2/27/1976#, "dd mmmm yyyy")
```

Внимательный читатель заметит, что в русской версии сообщения в окнах все же немного отличаются.

**Рис. 4.7**

Использование функции **Format** для изменения строкового формата значения даты



Табл. 4.7 содержит доступные именованные форматы и объясняет их действие.

**Таблица 4.7. Именованные форматы для использования с функцией Format**

Именованный формат	Действие
General Date	Форматирует информацию о дате и времени в последовательное число даты, используя установки формата даты и времени для вашего компьютера. То же, что VBA-преобразование по умолчанию последовательных дат в строки.
Long Date	Форматирует в последовательной дате только часть, содержащую дату, используя установки компьютера для Long-формата даты.
Medium Date	Форматирует в последовательной дате только часть, содержащую дату, используя установки компьютера для Medium-формата даты
Short Date	Форматирует в последовательной дате только часть, содержащую дату, используя установки компьютера для Short-формата даты.
Long Time	Форматирует в последовательной дате только часть, содержащую время, используя установки компьютера для Long-формата времени.
Medium Time	Форматирует в последовательной дате только часть, содержащую время, используя установки компьютера для Medium-формата времени.
Short Time	Форматирует в последовательной дате только часть, содержащую время, используя установки компьютера для Short-формата времени.
General Number	Форматирует число в строку без каких-либо особых символов. Действие такое же, как VBA-преобразование по умолчанию чисел в строки.
Currency	Форматирует число с символом денежной единицы, разделителем тысяч и только двумя десятичными разрядами. Символ денежной единицы и десятичный разделитель определяются локальными установками Windows.
Fixed	Форматирует число так, чтобы всегда была, по крайней мере, одна цифра перед десятичным разделителем и, по крайней мере, две цифры после него.
Standard	Форматирует число с разделителем тысяч так, чтобы была, по крайней мере, одна цифра перед десятичным разделителем и, по крайней мере, две цифры после него.

Именованный формат	Действие
Percent	Форматирует число как процентное отношение, умножая его на 100 и добавляя символ процента. Например, 0.21 возвращается как 21%.
Scientific	Форматирует число в обычный экспоненциальный формат.
Yes/No	Использование этого формата приводит к тому, что функция <b>Format</b> возвращает строку «Да» («Yes»), если форматируемое число ненулевое, и строку «Нет» («No») для любого нулевого значения. Этот именованный формат наиболее часто используется со значениями типа <b>Boolean</b> .
True/False	Использование этого формата приводит к тому, что функция <b>Format</b> возвращает строку «Истина» («True»), если форматируемое число ненулевое, и строку «Ложь» («False») для любого нулевого значения. Этот именованный формат наиболее полезен со значениями типа <b>Boolean</b> .
On/Off	Использование этого формата приводит к тому, что функция <b>Format</b> возвращает строку «Вкл» («On»), если форматируемое число ненулевое, и строку «Выкл» («Off») для любого нулевого значения. Наиболее часто используется со значениями <b>Boolean</b> .

Форматы **long**, **medium** и **short date** и **time** можно изменять посредством **Панели управления** (Windows Control Panel). Можно также изменять символы, используемые для разделителя тысяч и десятичного разделителя (на **Панели управления**).

Если вам необходимо создавать пользовательские форматы для чисел, дат или времени, нужно создать строку, содержащую символы-заполнители, для задания образа форматирования, который должна будет использовать функция **Format** при преобразовании значений в строку. В табл. 4.8 приведены пользовательские символы-заполнители, применяемые для создания образов форматов с применением функции **Format**. Кроме того, в табл. 4.8 используется как пример численное значение 1234,5. Полужирным шрифтом выделены символы-заполнители, которые вводятся из окружающего текста.

**Таблица 4.8. Символы-заполнители для создания пользовательских форматов**

Символ-заполнитель	Действие
0	Цифровой символ, отображает цифру, если таковая находится в этой позиции, или 0, если — нет. Можно использовать символ 0 для отображения начальных нулей для целых чисел и конечных нулей в десятичных дробях; <b>00000.000</b> отображает 00124,500.
#	Цифровой символ, отображает цифру, если таковая находится в этой позиции, иначе — не отображает ничего. Символ-заполнитель # эквивалентен 0, кроме того, что начальные и конечные нули не отображаются; <b>#####.###</b> отображает 1234,5.

Символ-заполнитель	Действие
\$	Отображает знак доллара; <b>####,###.00</b> отображает \$1 234,50.
.	Десятичный символ-заполнитель, отображает десятичную точку в обозначенной позиции в строке символов-заполнителей 0; <b>#.###.##</b> отображает 1234,5
%	Символ процента, умножает значение на 100 и добавляет знак процента в позицию, указанную символами-заполнителями 0; <b>#0.00%</b> отображает число 0.12345 как 12,35% (12,345 округляется до 12,35).
, (запятая)	Разделитель тысяч, добавляет запятые как разделители тысяч в строках символов-заполнителей 0 и #. <b>###,###,###.00</b> отображает 1 234,50.
E- e-	Отображает значения в экспоненциальном формате со знаком порядка только для отрицательных значений; <b>#.####E-00</b> отображает 1,2345E03; 0,12345 отображается как 1,2345E-01.
E+ e+	Отображает значения в экспоненциальном формате со знаком порядка для положительных и отрицательных значений; <b>#.####E+00</b> отображает 1,2345E+03.
/	Отделяет день, месяц и год для форматирования значений дат. <b>mm/dd/yy</b> отображает 06/06/97. Символы «/» можно заменять на символы дефиса для отображения как 06-06-97.
m	Указывает, как отображать месяцы в датах; <b>m</b> отображает 2, <b>mm</b> — 02, <b>mmm</b> — фев, <b>mmm</b> — Февраль.
d	Указывает, как отображать дни в датах; <b>d</b> отображает 1, <b>dd</b> отображает 01, <b>ddd</b> — Пт, <b>dddd</b> — пятница.
y	Отображает день года как число от 1 до 366.
yy	Указывает, как отображать годы в датах; <b>yy</b> отображает 99, <b>yyyy</b> — 1999.
q	Отображает квартал года как число от 1 до 4.
w	Отображает день недели как число (1 — это воскресенье).
ww	Отображает неделю года как число от 1 до 54.
: (двоеточие)	Отделяет часы, минуты и секунды в значениях формата времени; <b>hh:mm:ss</b> отображает 02:02:02.
h	Указывает, как отображать часы; для значения времени 02:01:38 <b>h</b> отображает 2, <b>hh</b> отображает 02.
n	Минутный символ-заполнитель для времени; для значения времени 02:01:08 <b>n</b> отображает 1, а <b>nn</b> отображает 01.
s	Секундный символ-заполнитель для времени; для значения времени 02:01:08 <b>s</b> отображает 8, а <b>ss</b> отображает 08.
AM/PM	Отображает время в 12-часовом формате времени с добавленными AM и PM; <b>h:nn AM/PM</b> отображает 4:00 PM. Альтернативные форматы включают am/pm, A/P и a/p.

Символ-заполнитель	Действие
@	Символьный заполнитель, отображает пробел, если не имеется соответствующего символа в форматируемой строке. @@@@ отображает строку Hi с двумя начальными пробелами. (порядок заполнения по умолчанию — справа налево).
<	Отображает все символы в верхнем регистре.
>	Отображает все символы в нижнем регистре.

При использовании функции **Format** для форматирования строк и чисел можно создавать дополнительные секции в образе формата, чтобы изменять формат отображения в соответствии с форматируемым значением. Секции в образе формата разделяются точкой с запятой (;). Например, следующий образ содержит секции и форматирует отрицательные числа иначе, чем положительные:

```
$###,###,##0.00;$ (###,###,##0.00)
```

Вышеприведенный образ форматирует число **1234567,89** как **\$1 234 567.89** и форматирует **-1234567,89** как **\$(1 234 567.89)**.

Можно для форматирования строки иметь две секции в образе формата. Если образ формата содержит только одну секцию, этот образ применим ко всем форматируемым строкам. Если образ формата содержит две секции, первая секция применима к строковым данным, а вторая — к значениям **Null** и строкам нулевой длины (то есть, к строкам, которые не содержат символов, представленных с помощью ""). Рассмотрим следующий оператор, отображающий результат функции **Format**, используемый с образом из двух частей для строк.

```
MsgBox Format(strData, "(@@@) - @@@ - @@@@; не номер телефона")
```

Приведенный выше оператор отображает (510) - 555 - 1212 в результирующем окне сообщения, если переменная **strData** содержит «5105551212». Если переменная **strData** содержит строку нулевой длины (""), то предыдущий оператор отображает в окне сообщения строку «не номер телефона».

Для форматирования числовых значений можно иметь до четырех различных секций в образе формата. Первая секция используется для положительных чисел, вторая — для отрицательных чисел, третья — для нулевых значений, а четвертая — для значений **Null**. Следующий образ формата содержит четыре секции:

```
$###,###,##0.00;$ (###,###,##0.00); 0.00; "Null value"
```

Приведенный выше образ формата форматирует отрицательные числа с круглыми скобками, указывает, что нулевые значения должны быть показаны как **0.00**, и возвращает текстовое сообщение, если форматируемое значение равно **Null**. Этот образ формата форматирует число **1234567,89** как строку «\$1 234 567.89». Число **-1234567,89** форматируется в строку «\$(1 234 567.89)». Значение **0** форматируется в строку «0.00», тогда как значение **Null** форматируется в строку «Null value».

Кроме общей функции форматирования, вы можете использовать функции специального форматирования, приведенные в таблице 4.9.



Таблица 4.9. Функции специального форматирования

Функция	Назначение
FormatCurrency (E[, N [, I [, U [, G]]]])	Возвращает выражение, отформатированное как денежное (валютное) выражение с использованием значения, заданного на вкладке <b>Денежная единица</b> окна <b>Свойства: Язык и стандарты</b> , доступного из <b>Панели управления</b> .
FormatDateTime (D[,N])	Возвращает выражение, отформатированное, как дата или время.
FormatNumber (E [, N [, I [, U [, G]]]])	Возвращает выражение, отформатированное, как число.
FormatPercent (E [, N [, I [, U [, G]]]])	Возвращает выражение, отформатированное, как процентное отношение (умноженное на 100) с конечным знаком процента (%).

Функция **FormatCurrency** возвращает выражение, отформатированное, как денежное (валютное) выражение с использованием значения, заданного на вкладке **Денежная единица** окна **Свойства: Язык и стандарты**, доступного из **Панели управления**.

Синтаксис функции **FormatCurrency** следующий (приведен из справочной системы VBA):

#### Синтаксис

```
FormatCurrency(Expression[, NumDigitsAfterDecimal [,  
IncludeLeadingDigit [, UseParensForNegativeNumbers [,  
GroupDigits]]]])
```

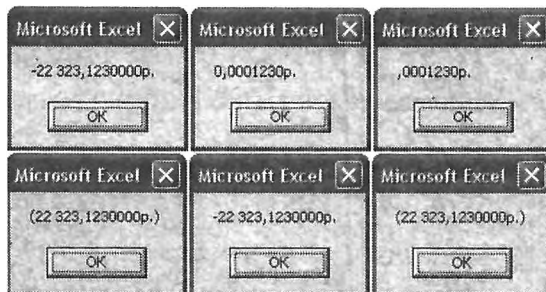
Здесь аргументы означают:

<i>Expression</i>	Выражение для форматирования (обязательный).
<i>NumDigitsAfterDecimal</i>	Численное значение, определяющее количество отображаемых знаков справа от десятичной точки. Значение по умолчанию для этого аргумента равно -1, что означает использование региональных настроек (необязательный).
<i>IncludeLeadingDigit</i>	Константа трех состояний, определяющая, следует ли отображать ведущий ноль для дробных чисел (необязательный).
<i>UseParensForNegativeNumbers</i>	Константа трех состояний, определяющая, следует ли помещать отрицательные значения внутри круглых скобок (необязательный).
<i>GroupDigits</i>	Константа трех состояний, определяющая, группировать ли цифры с помощью ограничителей, заданных региональными настройками (необязательный).

Аргументы *IncludeLeadingDigit*, *UseParensForNegativeNumbers* и *GroupDigits* могут принимать только значения констант: **vbTrue**, **vbFalse** и **vbUseDefault** (Использовать региональные настройки компьютера).

В качестве примера приведем код небольшой программы (листинг 4.6) и результат ее работы (рис. 4.8) в виде последовательно выдаваемых на экран диалоговых окон.

**Рис. 4.8**  
Пример использования функции **FormatCurrency**



**Листинг 4.6.** Демонстрация функции **FormatCurrency**

```

1: Sub MyMacro ()
2: 'Пример использования функции FormatCurrency
3:   Dim MyDouble1 As Double, MyDouble2 As Double
4:   MyDouble1 = -22323.123
5:   MyDouble2 = 0.000123
6:
7:   MsgBox FormatCurrency(MyDouble1, 7)
8:
9:   MsgBox FormatCurrency(MyDouble2, 7, vbTrue)
10:  MsgBox FormatCurrency(MyDouble2, 7, vbFalse)
11:
12:  MsgBox FormatCurrency(MyDouble1, 7, vbTrue, vbTrue)
13:  MsgBox FormatCurrency(MyDouble1, 7, vbTrue, vbFalse)
14:  MsgBox FormatCurrency(MyDouble1, 7, vbTrue, vbTrue, vbTrue)
15:
16: End Sub

```

Синтаксис функции **FormatDateTime** следующий (приведен из справочной системы VBA):

#### Синтаксис

```
FormatDateTime(Date[, NamedFormat])
```

Здесь аргументы означают:

- *Date* — дата для форматирования. (Обязательный);
- *NamedFormat* — Численное значение, указывающее необходимый формат. Если опущен, используется **vbGeneralDate** (необязательный).

Аргумент *NamedFormat* может принимать следующие значения:

- **VbGeneralDate** (значение равно 0) — отображать дату и/или время. Если в аргументе *Date* имеется часть даты, дата отображается в коротком формате. Если в аргументе *Date* имеется часть времени, она отображается в длинном формате;
- **vbLongDate** (значение равно 1) — отображать дату с использованием длинного формата, установленного на компьютере в региональных настройках;
- **vbShortDate** (значение равно 2) — отображать дату с использованием короткого формата, установленного на компьютере в региональных настройках;
- **vbLongTime** (значение равно 3) — отображать время с использованием временного формата, установленного на компьютере в региональных настройках;
- **vbShortTime** (значение равно 3) — Отображать время с использованием формата 24-hour (hh:mm).

В качестве примера приведем код небольшой программы (листинг 4.7) и результат ее работы (рис. 4.9) в виде последовательно выдаваемых на экран диалоговых окон.



Рис. 4.9. Пример использования функции **FormatDateTime**

#### Листинг 4.7. Демонстрация функции **FormatDateTime**

```
1: Sub MyMacro()  
2: 'Пример использования функции FormatDateTime  
3:   Dim MyDate As Date  
4:   MyDate = Now()  
5:  
6:   MsgBox FormatDateTime(MyDate, vbGeneralDate)  
7:   MsgBox FormatDateTime(MyDate, vbLongDate)  
8:   MsgBox FormatDateTime(MyDate, vbShortDate)  
9:   MsgBox FormatDateTime(MyDate, vbLongTime)  
10:  MsgBox FormatDateTime(MyDate, vbShortTime)  
11:  
12: End Sub
```

Синтаксис функции **FormatNumber** следующий (приведен из справочной системы VBA):

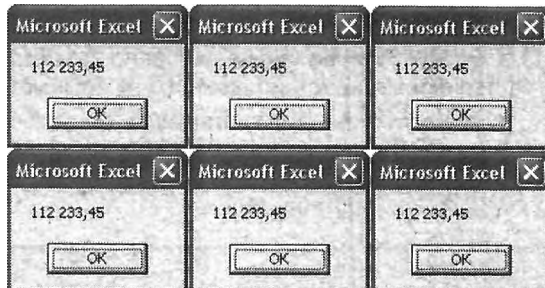
#### Синтаксис

```
FormatNumber(Expression[, NumDigitsAfterDecimal  
[, IncludeLeadingDigit [, UseParensForNegativeNumbers  
[, GroupDigits]]]])
```

Здесь аргументы означают то же, что и в функции **FormatCurrency**.

В качестве примера приведем код небольшой программы (листинг 4.8) и результат ее работы (рис. 4.10) в виде последовательно выдаваемых на экран диалоговых окон.

**Рис. 4.10**  
Пример использования функции **FormatNumber**



**Листинг 4.8.** Демонстрация функции **FormatNumber**

```

1: Sub MyMacro()
2: 'Пример использования функции FormatNumber
3:   Dim MyDouble As Double
4:   MyDouble = 112233.4455
5:
6:   MsgBox FormatNumber(MyDouble)
7:   MsgBox FormatNumber(MyDouble, vbFalse)
8:   MsgBox FormatNumber(MyDouble, vbTrue)
9:   MsgBox FormatNumber(MyDouble, , vbFalse)
10:  MsgBox FormatNumber(MyDouble, , vbTrue)
11:  MsgBox FormatNumber(MyDouble, , , vbFalse)
12: End Sub

```

Синтаксис функции **FormatPercent** следующий (приведен из справочной системы VBA):

#### Синтаксис

```

FormatPercent (Expression[,NumDigitsAfterDecimal
[,IncludeLeadingDigit [,UseParensForNegativeNumbers
[,GroupDigits]]]])

```

Здесь аргументы означают то же, что и в функции **FormatCurrency**.

В этой главе приведены не все функции, которые имеются в VBA. В VBA есть функции, которые позволяют «общаться» с другими приложениями, получать информацию об ошибках времени исполнения, получать информацию о массивах и манипулировать объектами различных host-приложений в Word и Excel. Некоторые из них описываются в других главах.

## Использование функций host-приложений

Кроме функций, встроенных в Visual Basic for Application, из кода VBA доступны некоторые функции host-приложения VBA. (Помните, что *host-приложение* — это приложение, в котором разрабатываются процедуры VBA, такие как Word, Excel, PowerPoint, Outlook или FrontPage).

Специфические функции host-приложения, доступные для VBA, зависят от определенного host-приложения, в котором выполняется работа.

В Excel, например, имеются различные функции, выполняющие математические, логические, финансовые и статистические операции над данными в рабочих листах. Многие (хотя и не все) из этих функции Excel доступны из кода VBA.

Функции host-приложения, доступные для VBA, не являются частью VBA, они являются частью host-приложения. Функции рабочих листов Excel, например, не являются частью языка программирования VBA; они являются частью host-приложения Excel. Не каждое host-приложение VBA содержит функции, которые можно использовать в VBA. Функции, доступные для VBA в одном host-приложении, могут не быть доступными в другом. Если вы намереваетесь писать процедуры VBA, которые могут выполняться любым host-приложением, не используйте функции из host-приложения, потому что они могут не быть доступными во всех приложениях. Например, если необходимо написать процедуру для использования в Word, Excel, Access или Microsoft Project, не используйте функций Excel в операторах VBA.

Чтобы использовать функцию, принадлежащую какому-либо host-приложению, обращайтесь к функции посредством программного объекта (**Application**). Объект **Application** VBA представляет host-приложение и все его ресурсы. (Объекты более подробно описываются далее.)

Помимо нескольких функций, преобразующих различные измерения набора текста в дюймы и наоборот, Word не имеет функций приложений, доступных в VBA. По этой причине в данной главе основное внимание при объяснении использования функций host-приложений уделяется использованию функций host-приложений в Excel.

Более вероятно, что вы будете использовать функции host-приложения в Excel, поскольку Excel разработан специально для манипулирования числовыми данными в формате рабочих листов. В результате Excel содержит много функций для использования в формулах рабочих листов. Функции рабочих листов Excel применяются для выполнения таких задач, как нахождение суммы целого столбца цифр, для выполнения финансовых вычислений (например, нахождение текущей величины займа) и для выполнения статистического анализа данных.

Хотя многие математические функции Excel дублируют математические функции VBA, Excel содержит намного больше специализированных статистических и финансовых функций, чем VBA. При программировании Excel-операций в Excel VBA может понадобиться использовать многие из функций рабочих листов, являющихся частью Excel.

В качестве примера следующий оператор использует Excel-функцию **Max**, возвращающую самое большое число в ее списке аргументов:

```
MsgBox Application.Max(9, 1, 3, 2) 'Отображает 9
```

В этом операторе обратите внимание на то, что за словом **Application** следует точка (.) и затем — имя функции **Max** без пробелов. Эта точка, называемая *точкой-разделителем* (*dot separator*), указывает на то, что оператор ссылается на функцию **Max**, которая является частью объекта **Application**. При использовании функций host-приложения в операторах VBA необходимо включать ключевое слово **Application** и точку-разделитель перед именем каждой функции. Например, чтобы использовать какие-либо функции рабочих листов Excel в операторах VBA, необходимо включать ключевое слово **Application** и точку-разделитель (.) перед именем каждой функции Excel.

Результат функции Excel нельзя игнорировать. Необходимо всегда включать круглые скобки в вызов функции Excel и всегда каким-либо образом использовать результат функции: как значение в выражении, аргумент для другой функции или процедуры или в операторе присваивания.

Если вы опытный пользователь рабочих листов Excel, вы могли уже заметить, что Excel имеет много функций, имеющих те же имена, что и некоторые функции, перечисленные в табл. 4.3–4.6. Эта ситуация может также наблюдаться и в других host-приложениях: host-приложение имеет функции, имена которых дублируют имена функций, имеющихся в VBA.

Поскольку необходимо всегда указывать ключевое слово **Application**, когда используется функция host-приложения, то не бывает неясно для VBA или пользователя, на какую функцию (VBA или приложения) ссылается оператор. Следующий фрагмент кода, например, показывает два оператора в Excel VBA:

```
Rslt = Log (AnyNum)
Rslt = Application.Log (AnyNum)
```

Первый оператор вызывает VBA-функцию **Log**; второй — вызывает Excel-функцию **LOG**. Чтобы использовать VBA-версию функции, просто пишите имя функции; чтобы использовать версию функции host-приложения, включайте ключевое слово **Application**.

Функции host-приложений, имеющие те же имена, что и функции VBA, не обязательно выполняют те же самые задачи и выдают те же самые результаты. Например, Excel-функция **LOG** отличается от VBA-функции **Log**, и эти функции возвращают разные ответы: Excel-функция **LN** — это функция, совпадающая с действием VBA-функции **Log**. Внимательно изучите действие и результат функции host-приложения перед тем, как использовать ее вместо функции VBA, в противном случае процедуры могут выдавать ошибочные результаты.

Не каждая функция host-приложения доступна VBA. Например, некоторые функции Excel, дублирующие функции VBA, недоступны, потому что в этом нет смысла. Конкретнее, Excel-функции **Date**, **Year**, **Month**, **Day**, **Hour**, **Minute** и **Second** дублируют VBA-функции **Date**, **Year**, **Month**, **Day**, **Hour**, **Minute** и **Second** как по своему действию, так и по назначению. Никакая из этих функций Excel недоступна для VBA. Иногда некоторые функции host-приложений недоступны для VBA независимо от того, дублируют они функции VBA или — нет.

Если вы не уверены, доступна ли определенная функция host-приложения для VBA, используйте **Object Browser**, чтобы проверить, включает ли список **Members** (Компонент) эту функцию, при выбранном **Application** в списке **Classes** (Классы) и при выбранном host-приложении в списке **Project/Library** (Проект/Библиотека). Если нужной функции нет в списке, то она недоступна для VBA.

Чтобы узнать, какие функции имеются в Excel (или любом другом приложении), и узнать, каково назначение и как использовать эти функции, обращайтесь к справочной системе и ищите раздел для слова *functions*.

В заключение отметим очень полезное дополнительное свойство функции **InputBox** приложения Excel, которого нет в VBA. Дело в том, что **InputBox** в Excel имеет необязательный параметр *Type*, задающий тип вводимого значения. Это можно использовать в качестве дополнительного контроля вводимой информации.

Параметр *Type* может принимать следующие значения:

Значение	Что означает
0	Формула
1	Число
2	Текст (строка)
4	Логическое значение (True или False)
8	Ссылка на ячейку, как объект диапазона
16	Значение ошибки, такое как #N/A
64	Массив значений

Вы можете использовать в качестве аргумента *Type* сумму доступных значений. Например, для ввода как текстовых, так и числовых значений можно задавать значение 1+2.

## Создание функций и функций-процедур

Итак вы уже узнали, что такое функция и как использовать встроенные функции VBA и host-приложений VBA, таких как Excel. Теперь вам предстоит узнать, как создавать собственные пользовательские функции, как работать с ними в host-приложениях VBA. В частности, вы узнаете, как сделать так, чтобы ваши собственные функции были доступны рабочим листам Excel.

Перед тем как начать писать собственные функции, вам необходимо ознакомиться с терминологией и понятиями, используемыми при обсуждении создания функций в VBA.

**Функция-процедура (или функция)** — это особый вид процедуры VBA, возвращающей результат. Пользовательские функции-процедуры, как и встроенные функции VBA, могут иметь необязательные и именованные аргументы. Функции можно использовать для обеспечения значениями выражений, (присваивания) или в качестве аргументов других функций и процедур. Создание новой функции состоит из написания программных операторов, которые определяют:

- 1) аргументы, используемые функцией;
- 2) действия, выполняемые функцией;
- 3) значение, возвращаемое функцией.

Для того чтобы можно было использовать ваши функции в Excel, необходимо выполнять некоторые дополнительные правила (ограничения).

Word не имеет встроенных функций, которые использовались бы подобно Excel-функциям рабочих листов. Word дает возможность вставлять в доку-

мент в виде полей информацию различных типов данных, таких как текущая дата и время, имя документа, оглавление и так далее. Word не содержит функций для выполнения статистических вычислений или других аналитических задач, для чего и предназначен Excel. Следовательно, функции, создаваемые с помощью VBA, имеют различное применение в Excel и Word. В Excel можно использовать функции VBA для расширения коллекции встроенных функций рабочих листов. Другие host-приложения VBA, такие как Access, также позволяют применять создаваемые вами VBA-функции для расширения и улучшения встроенной коллекции функций.

С «точки зрения» Excel используемые им функции-процедуры VBA являются *определенными пользователем функциями (user-defined functions)*. Этот термин позволяет отличать функции, написанные пользователем, от встроенных функций Excel. Хотя все определенные пользователем функции являются также функциями-процедурами, не все функции-процедуры отвечают требованиям определенной пользователем функции.

Функция-процедура — это наиболее общий термин для создаваемых пользователем функций; термин «определенная пользователем функция» описывает определенный тип функции-процедуры, который может использовать Excel. Нельзя использовать функцию-процедуру, которая не отвечает требованиям определенной пользователем функции в формулах рабочих листов Excel; можно использовать такие функции только в операторах ваших собственных процедур. Далее в этой главе будет использоваться термин «функция» вместо «функция-процедура», когда будет понятно, о чем идет речь.

Для записи функции-процедуры нельзя использовать макрорекордер, хотя можно редактировать записанный рекордером макрос и превращать его в функцию-процедуру.

### Написание функции-процедуры

Функции-процедуры очень похожи на процедуры VBA, которые вы уже умеете записывать. Основное различие между функцией-процедурой и другими процедурами, помимо того, что функции возвращают значение, а процедуры — нет, состоит в том, что вместо ключевых слов **Sub** и **End Sub**, с которыми вы уже знакомы, в функции-процедуре используются ключевые слова **Function** и **End Function**.

Функция-процедура имеет следующий синтаксис:

### Синтаксис

```
Function Name([Arglist]) [As Type]
    ' VBA Statements
    [Name = expression]
End Function
```

Каждая функция-процедура начинается ключевым словом **Function**, за которым следует имя функции, *Name* представляет имя, выбранное для этой функции. При написании имен функций необходимо соблюдать те же правила, что и при написании имен других идентификаторов в VBA: они должны начинаться с буквы, не могут содержать пробелов или каких-либо символов арифметических, логических операторов или операторов отношения и не могут дублировать ключевые слова VBA.



После имени функции следует список ее аргументов, который заключается в круглые скобки. Здесь *Arglist* представляет список аргументов функции и является необязательным.

*Type* — любой тип возвращаемого значения функции. Если только не определяется иначе, результат, который возвращает функция-процедура, имеет тип **Variant**. Как вы узнали ранее, значения, сохраняемые или обрабатываемые как тип **Variant**, занимают больше памяти, чем любой другой тип данных, и на их обработку уходит больше времени.

Следует задавать тип данных результата функции по тем же причинам, по каким задается тип переменных и констант: для ускорения выполнения кода, более эффективного использования памяти, получения более легкого и понятного кода и для нахождения ошибок программирования (при этом необходимо хорошо знать правила преобразования типов в VBA).

Необязательный элемент синтаксиса *Name = expression* представляет *присваивание функции (function assignment)*, которое указывает VBA, какое значение должна возвращать функция. Хотя эта часть функции является необязательной, следует всегда включать оператор присваивания в функции-процедуры. Наконец, объявление функции заканчивается ключевыми словами **End Function**.

---

Даже если функция не имеет аргументов, как VBA-функции **Now**, **Date** и **Time**, в объявлении функции необходимо использовать круглые скобки. Например, можно написать функцию-процедуру, возвращающую имя файла, содержащего текущую рабочую книгу, так, чтобы можно было вставлять имя файла в ячейку рабочего листа. Такой функции не нужны аргументы, ей не требуется никакая внешняя информация для выполнения работы, и она будет иметь подобное объявление:

```
Function ThisBookName()
```

Обычно функция предназначена для выполнения некоторого вычисления или другого манипулирования определенными данными и для возвращения результата этого манипулирования. Как вы уже знаете, информация встроенным функциям VBA передается заданием значений в списке аргументов функции. При объявлении функции-процедуры необходимо указывать имя каждого аргумента, передаваемого функции; имена аргументов в списке следует отделять друг от друга запятой и писать в соответствии с правилами, применяемыми к любому идентификатору VBA.

Имена, предоставляемые пользователем в списке аргументов, подобны переменным: они ссылаются на значение, предоставляемое в то время, когда вызывается функция, независимо от того, вызывается ли функция оператором VBA или *host*-приложением.

Имена аргументов имеют ту же область действия, что и переменные, объявляемые локально в функции-процедуре, то есть недоступны вне функции-процедуры, в списке аргументов которой они объявляются.

Вернемся к описанию синтаксиса и к строке, содержащей оператор *Name = expression*. Эта строка представляет *присваивание функции*; *Name* — имя функции, а *expression* — любое выражение, создающее значение, которое должна возвращать функция. Присваивание функции указывает VBA, какое значение будет возвращать функция. Заметьте, что присваивание функции использует имя функции-процедуры и присваивает ему значение, как если бы это была переменная. Функции-процедуры могут не иметь совсем или иметь один или несколько различных операторов присваивания функции.

Первый пример функции, который вы изучите, помогает манипулировать строками. Если необходимо определить длину строки, *исключая* начальные и конечные пробелы, можно использовать *вложенные (nested)* вызовы функции (один вызов функции внутри другого), подобные следующему:

```
StrLen = Len(Trim(AnyStr))
```

Этот тип операции хорошо подходит для функции-процедуры. Использовать один вызов пользовательской функции-процедуры проще и легче, чем повторять запись вложенного вызова функции. Листинг 4.9 показывает как раз такую простую функцию-процедуру **LenTrim**, которая возвращает длину строки, исключая начальные и конечные пробелы.

#### Листинг 4.9. Простая функция-процедура **LenTrim**

---

1: Function LenTrim(tStr)	'Определение функции
2:   LenTrim = Len(Trim(tStr))	'Возвращаемое значение
3: End Function	

---

В строке 1 содержится объявление функции **LenTrim**, начинающееся с обязательного ключевого слова **Function**. После имени функции открываются круглые скобки, что указывает VBA на начало списка аргументов. Далее следует имя аргумента (**tStr**). Имя аргумента сообщает VBA, что функции-процедуре при ее вызове должен передаваться один аргумент.

Как вы уже, наверное, успели заметить, при нажатии на клавишу **Enter** после ввода ключевого слова **Function**, имени функции и списка аргументов Редактор VB автоматически вставляет ключевые слова **End Function**, как и в случае, когда вы вводите ключевое слово **Sub** для создания процедуры. Если этого не происходит, возможно, при написании строки-заголовка была допущена ошибка синтаксиса.

Строка 2 функции **LenTrim** — это строка, которая выполняет всю (пока небольшую) работу функции и также содержит аргумент функции для **LenTrim**. При вычислении выражения **Len(Trim(tStr))** VBA принимает строку, полученную посредством аргумента **tStr**, и передает ее VBA-функции **Trim** для удаления начальных или конечных пробелов. Результат функции **Trim**, в свою очередь, используется как аргумент функции **Len**. Затем VBA присваивает результат функции **Len** имени функции **LenTrim**. **LenTrim** возвращает длину строки аргумента, исключая начальные или конечные пробелы.

Наконец, в строке 3 функция заканчивается ключевыми словами **End Function**. После выполнения этой строки VBA возвращается к оператору процедуры, вызвавшему функцию **LenTrim**, и вставляет результат функции **LenTrim** в этот оператор в том месте, где появляется имя функции.

Для вызова функции **LenTrim** используйте оператор, подобный следующему, который отображает результат **LenTrim** для строки " Excel 2007 ":

```
MsgBox LenTrim (" Excel 2007 ")
```

В этом операторе строка аргумента имеет четыре начальных и четыре конечных пробела, и длина строки (как указывает VBA-функция **Len**) равна 18 символам. Функция **LenTrim** сообщает длину строки без начальных и конечных пробелов; вышеуказанный оператор отображает число 10.

### Создание определенных пользователем функций для Excel

Функции-процедуры с некоторыми ограничениями на их действия называются *определенными пользователем функциями* (или сокращенно *UDF* — *user-defined functions*), и только их может использовать Excel в формуле ячейки рабочего листа.

Все ограничения определенных пользователем функций происходят из одного базового ограничения: **UDF не может никаким образом изменять среду Excel**. Это означает, что определенная пользователем функция не может выбирать, вставлять, удалять или форматировать никакие данные в рабочем листе, таблице или другом листе. UDF также не может добавлять, удалять или переименовывать листы или рабочие книги, не может изменять экранное представление и так далее. Например, нельзя использовать функцию-процедуру как определенную пользователем функцию в Excel, если она выделяет ячейки или изменяет каким-то образом текущий рабочий лист.

Кроме того, UDF не может устанавливать свойства объекта или использовать методы объекта; в большинстве случаев установка свойств объекта или использование его методов приводят к изменениям в среде Excel (объекты, методы и свойства описываются в главе 6). Определенная пользователем функция может осуществлять выборку значений свойств объекта и выполнять любые методы объекта, которые не изменяют среду Excel.

Обычно определенная пользователем функция должна только выполнять вычисления или манипулирование на основе данных, полученных из ее списка аргументов или выбранных из Excel. Функцию **LenTrim**, показанную в листинге 4.9, можно использовать как определенную пользователем функцию, поскольку она отвечает всем требованиям UDF.

Следует помнить о том, что ни VBA, ни Excel не отображают сообщение об ошибке, если в качестве UDF используется функция, в которой нарушены правила для определенных пользователем функций. Такая функция просто не может возвращать результат. Например, при попытке вставить значение в рабочий лист Excel, используя функцию-процедуру, нарушающую правила для определенных пользователем функций, ячейка с такой функцией отображает Excel-сообщение об ошибке **#VALUE!**, обозначающее, что функция или формула для этой ячейки не может возвращать допустимый результат.

VBA запрещает присваивание несовместимого типа результату функции в любой функции-процедуре, которая имеет объявленный тип данных для ее результата. Если, например, вы ошибочно написали оператор присваивания функции так, что присваивается, допустим, тип **Integer** результату функции с объявленным типом **String**, то VBA отображает ошибку несовпадения типа.

В случае присваивания типа данных, который не является тем же, что объявленный возвращаемый тип для функции-процедуры, но является совместимым, то VBA преобразует значение типа, определенного для функции при возврате результата функции. Например, если присваивается тип **Double** функции, результат которой был объявлен как **Long**, то VBA не выдает никакой ошибки, а просто преобразует **Double** в тип **Long** (при возвращении результата функции).

Объявление типа функции имеет еще один результат: если функция-процедура заканчивается без выполнения оператора присваивания функции, VBA возвращает строку нулевой длины для функций типа **String** и 0 — для функций-процедур, возвращающих численный тип. Нетипизированная функция-

процедура, которая заканчивается без выполнения оператора присваивания функции, возвращает результат типа **Variant**, содержащий специальное значение **Empty**.

В листинге 4.10 приведена функция **LenTrim**, модифицированная так, что она всегда возвращает значение типа **Long**.

---

**Листинг 4.10. Задание типа функции LenTrim**

---

```
1: Function LenTrim(tStr) As Long
2:   'возвращает длину tStr без ведущих и хвостовых пробелов
3:   LenTrim = Len(Trim(tStr))
4: End Function
```

---

Следует выбирать тип результата функций-процедур, требующий наименьшего объема памяти, но вмещающий полный диапазон возможных значений, которые может возвращать функция. Для функций общего назначения численных или математических наиболее характерным типом для результата функции является **Double**.

**Объявление типов данных для аргументов функции**

Если только не было определено иначе, VBA передает все аргументы в функцию-процедуру как типы **Variant**. Можно объявлять определенные типы данных для каждого аргумента в списке аргументов.

Аргументы с определенными типами используются по тем же известным причинам, по каким используются типизированные переменные или результаты функции. Определение типов аргументов для функции-процедуры также помогает пользователю при вызове функции вводить аргументы правильного типа в правильном порядке.

Для объявления определенных типов аргументов функции-процедуры, используйте ключевое слово **As**, за которым следует имя нужного типа данных после имени аргумента в списке аргументов. Листинг 4.11 показывает функцию **LenTrim**, модифицированную так, чтобы она принимала только данные типа **String** в аргументе **tStr**.

---

**Листинг 4.11. Определение типа данных аргумента функции LenTrim**

---

```
1: Function LenTrim(tStr As String) As Long
2:   'возвращает длину tStr без ведущих и хвостовых пробелов
3:   LenTrim = Len(Trim(tStr))
4: End Function
```

---

За исключением объявления функции-процедуры (в строке 1), листинг 4.11 идентичен листингу 4.10. После того как тип аргумента объявлен, VBA при вызове функции **LenTrim** допускает в качестве аргумента появление только значений типа **String**. Вызов **LenTrim** с каким-либо другим типом (даже **Variant**) в качестве аргумента вызывает ошибку несовпадения типа (конкретнее, *несовпадение типа аргумента — argument type mismatch*).

## Использование функций-процедур в VBA

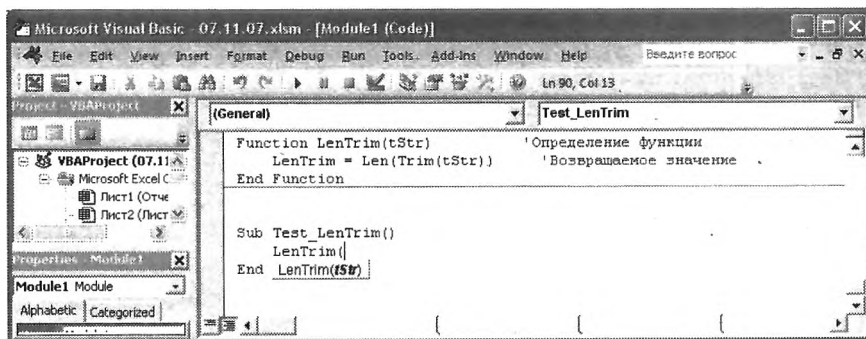
Используйте собственные функции-процедуры в операторах VBA так же, как любые встроенные функции VBA; все правила и условия по использованию встроенных функций применимы к пользовательским функциям-процедурам.

Помните, что при вызове функции необходимо включать список аргументов в круглых скобках, если только вы не собираетесь игнорировать результат функции. Как и в случае со встроенными функциями VBA, можно игнорировать результат вашей собственной функции, хотя это редко бывает необходимо.

Если нужно использовать именованные аргументы в ваших собственных функциях-процедурах, просто используйте имена из списка аргументов в объявлении функции-процедуры. Например, чтобы использовать именованный аргумент в вызове функции **LenTrim** (листинг 4.11), применяйте оператор, подобный следующему (**AnyStr** и **MyString** — строковые переменные):

```
AnyStr = SLen(tStr:=MyString)
```

Если вам трудно запомнить все именованные аргументы для одной из своих функций, убедитесь, что свойство **Auto Quick Info** (краткие сведения) включено. Если это так, Редактор VB отображает всплывающее окно с перечнем именованных аргументов вашей функции, как для встроенных функций VBA. На рис. 4.11 показано **Code Window**, в котором зафиксирован процесс ввода кода для вызова функции **LenTrim** из листинга 4.9; обратите внимание на всплывающее окно, отображающее имя функции, ее именованный аргумент.



**Рис. 4.11.** Свойство **Auto Quick Info** позволяет отображать всплывающее окно, показывающее список аргументов вашей собственной функции, как и для встроенных функций VBA

Свойство **Auto Quick Info** можно включать и отключать, выбирая **Tools | Options** (Сервис | Параметры) для отображения диалогового окна **Parameters** (Параметры). Щелкните на вкладке **Editor** (Редактор) для отображения опций Редактора VB и затем установите или отмените установку флажка **Auto Quick info** (краткие сведения).

### Использование инструмента *Object Browser* для нахождения функций-процедур

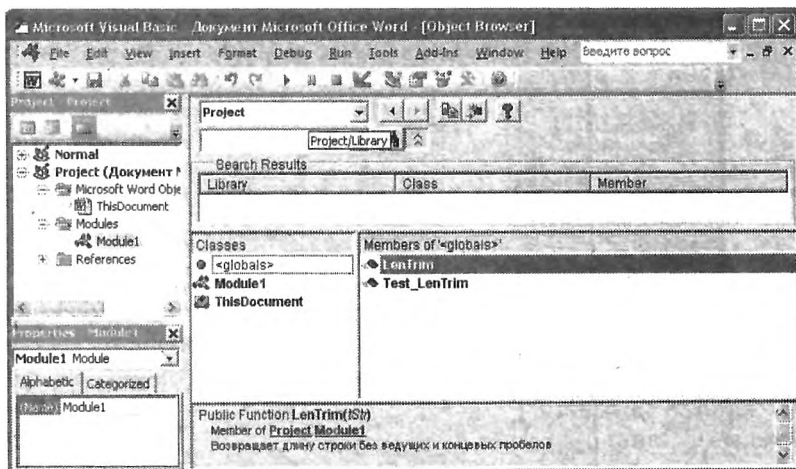
Если вы не можете вспомнить, в каком модуле сохранили некоторую функцию, можно использовать **Object Browser** для определения того, какие из ваших функций-процедур доступны в данный момент, и для быстрого отображения исходного VBA-кода вашей функции.

Чтобы использовать **Object Browser**, запустите его, как вы уже научились это делать: выберите команду **View | Object Browser** (Вид | Просмотр объектов) или щелкните кнопку **Object Browser** на панели Редактора VB для отображения окна **Object Browser**.

Вы уже знаете, как использовать **Object Browser** для отображения доступных функций VBA или функций host-приложений. Раскрывающийся список **Project/Library** (Проект/Библиотека) отображает все открытые в данный момент файлы приложения, как и всегда присутствующую опцию **VBA**. Для отображения пользовательских функций (и других процедур) в определенном проекте VBA (документе Word, рабочей книге Excel или каком-либо другом документе host-приложения) выполняйте следующие шаги:

1. Выберите наименование проекта в раскрывающемся списке **Project/Library**. На рис. 4.12 показано окно **Object Browser** с выбранным в **Project/Library**-списке элементом **Project** (в Excel подобный элемент называется **VBAProject**). Окно **Project Explorer** указывает имя файла, в котором сохранен проект (в круглых скобках после имени проекта). После того, как был выбран проект в окне **Project/Library**, список **Classes** (Классы) содержит перечень всех модулей и других объектов в выделенном проекте.

**Рис. 4.12**  
Используйте **Object Browser** для нахождения VBA-функций



2. Выберите модуль в списке **Classes**. После этого список **Members of <class>** (Компонент) содержит перечень всех процедур и функций, объявленных в выбранном модуле.

3. Выберите функцию или процедуру, исходный код которой необходимо просмотреть, в списке **Members of<class>** (Компонент).
4. Щелкните на кнопке **Show** (показать) для отображения исходного кода функции или процедуры. Редактор VB открывает соответствующий модуль и помещает курсор на первую строку после объявления функции или процедуры.

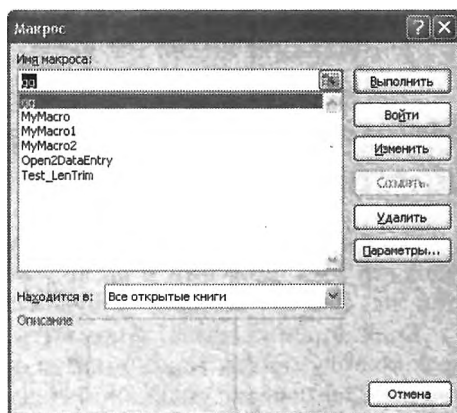
Можно также копировать текст из окна **Object Browser** для своих функций и процедур в буфер Windows (Clipboard) так же, как и в случае со встроенными функциями VBA.

Помните, что кнопка **Show** в окне **Object Browser** включается, только если исходный код для выбранной функции или другой процедуры доступен.

Не пытайтесь использовать диалоговое окно **Макрос** (открываемое с помощью команды **Разработчик | Макросы**) для выполнения функции-процедуры. VBA «ожидает», что функция-процедура предназначена для того, чтобы возвращать значение, поэтому нет смысла только в выполнении функции. По этой причине VBA не перечисляет функции-процедуры в списке **Имя макроса** в диалоговом окне **Макрос**. Это демонстрируется на рис. 4.13, где в списке макросов имеется только процедура (программа) **Test\_LenTrim**.

**Рис. 4.13**

VBA не перечисляет функции-процедуры в списке **Имя макроса** окна **Макрос**



### **Ввод описания функции-процедуры с помощью инструмента *Object Browser***

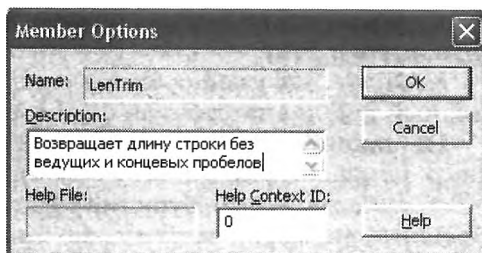
Вернемся к рис. 4.12. В списке **Members of<class>** (Компонент) выбрана функция **LenTrim** из листинга 4.11. Заметьте, как и в случае со встроенными функциями VBA и функциями host-приложения, имя функции и ее аргументы отображаются в нижней части окна **Object Browser** (под списками **Classes** и **Members**). Однако в отличие от встроенных функций VBA, никакого описания назначения функции не появляется. Для добавления описания к каждой из ваших функций или процедур можно использовать **Object Browser**. Описание, вводимое для функции с помощью окна **Object Browser**, появляется не только в окне **Object Browser**, но и в **Function Wizard** (мастере функций) Excel. Обеспечение функции-процедуры описанием помогает другим пользователям использовать вашу функцию в формуле ячейки рабочего листа.

Чтобы ввести описание для функции-процедуры, выполните следующие шаги:

1. Выберите функцию-процедуру в окне **Object Browser**, как было описано ранее.
2. Щелкните правой кнопкой мыши на функции, к которой необходимо добавить описание, и выберите **Properties** (Свойства) во всплывающем меню. Редактор VB отображает диалоговое окно **Member Options** (Параметры компонента), показанное на рис. 4.14.

**Рис. 4.14**

Вводите полезные описания для функций и процедур в диалоговое окно **Member Options**



3. Введите описание для функции (или процедуры) в текстовое окно **Description** (Описание). На рис. 4.14 показано описание функции **LenTrim**, уже введенное в текстовое окно **Description**.
4. Щелкните на **OK**. Вы вновь оказываетесь в окне **Object Browser**.

## Использование функций пользователя в рабочих листах Excel

Чтобы использовать свои функции (определенные пользователем) в Excel, вводите имя функции и ее аргументы как формулу в ячейку рабочего листа так же, как вы вводите любую встроенную функцию Excel.

Если вы не можете запомнить имя определенной пользователем функции или ее аргументы, можно найти эту функцию в списке **Function name** для категории **User-Defined**, указанной в списке **Function category** диалогового окна **Paste Function**. [Отобразить диалоговое окно **Paste Function** (Мастер функций) Excel можно, щелкая на кнопке **Paste Function** (Вставка функции) на панели инструментов или выбирая в меню **Insert | Function** (Вставка | Функция).]

Вставляйте функции VBA из диалогового окна **Paste Function**, как любые встроенные функции Excel. При необходимости в этом же окне вставляйте и значения для аргументов функции.

Если при использовании определенной пользователем функции (UDF) в ячейке рабочего листа отображаемым результатом является значение ошибки **#VALUE!**, внимательно проверьте код в вашей функции-процедуре, чтобы убедиться, что он не нарушает каких-либо правил для определенных пользователем функций. Ни Excel, ни VBA не препятствуют использованию в ячейке рабочего листа функции-процедуры, которая нарушает UDF-правила. Однако, поскольку функция-процедура нарушает UDF-правила, VBA не выполняет эту функцию, а возвращает ячейке значение ошибки **#VALUE!**.



При написании функции-процедуры помните, что особым назначением функции является возвращение какого-либо значения. Пишите простые и значимые функции. Обычно функция должна содержать программный код, необходимый только для выполнения некоторых вычислений или манипулирования данными; функция никогда не должна выполнять действия, не относящиеся непосредственно к созданию ее возвращаемого значения.

Пишите функцию всякий раз, когда вам необходимо использовать одну и ту же математическую формулу более двух-трех раз. Как показывают примеры в этой главе, функции-процедуры полезны также при манипулировании и другими данными, а не только числами.

Даже если выражение для какого-либо вычисления может быть совсем простым, все же следует писать функцию-процедуру, если вы используете это выражение часто. Таким образом вы устраняете вероятность ошибки при вводе выражения и даете этому выражению значащее имя.

Хотя можно игнорировать результаты функций-процедур, следует все же так писать функции-процедуры, чтобы игнорирование их результатов не имело смысла. Другими словами, функции не должны выполнять действия, кроме абсолютно необходимых для выдачи нужных возвращаемых результатов.

Не считайте VBA-функцию **MsgBox** примером для написания функций. Функция **MsgBox** выполняет уникальную задачу в VBA; в результате она делает намного большую работу с большим количеством опций, чем обычно выполняют функции. И, наоборот, считайте VBA-функции **StrComp**, **Mid**, функции преобразования данных **CStr** и **Cdbl** и так далее примером того, как должна действовать функция-процедура. Каждая функция выполняет одну задачу и возвращает один результат без изменения исходных данных, содержащихся в переданных ей аргументах.

Не пытайтесь использовать передаваемые по ссылке аргументы для создания функции, возвращающей более одного результата, используя возвращаемое значение функций и один или больше модифицированных аргументов для получения результатов. Функция никогда не должна изменять исходные данные в передаваемых ей аргументах. Далее в книге будет показано, что существуют обоснованные условия, при которых следует модифицировать значения в аргументах, передаваемых по ссылке; при этих условиях необходимо использовать процедуру, а не функцию. Следите за тем, чтобы ваши функции не изменяли передаваемые по ссылке аргументы; используйте ключевое слово **ByVal** для передачи аргументов по значению.

Многие ваши программы должны использоваться не только в программах или рабочих листах, для которых вы первоначально написали функцию-процедуру. Функция **LenTrim**, которая обсуждалась в этой главе, например, является функцией общего назначения и может быть полезной в целом ряде различных выражений в разных процедурах.

Поскольку функция-процедура в документе открытого проекта доступна для любого открытого в данный момент документа в одном и том же приложении, можно собрать универсальные функции в один проект. Объединяя функции-процедуры общего назначения в одном проекте, можно создать *библиотеку (library)* функций. Например, можно поместить все ваши универсальные Excel-функции-процедуры в файл рабочей книги с именем **MyFunctions.xls**. Другой пример: можно поместить все ваши универсальные Word-функции в документ шаблона с именем **MyFunctions.dot** и т.д.

Следите за тем, чтобы создаваемая вами функция содержала присваивание функции. То есть, чтобы функция возвращала результат, отличный от пустого значения по умолчанию типа **Variant**, строки нулевой длины типа **String** или нулевого численного значения. Следите за тем, чтобы программный код функции отвечал всем правилам для определенных пользователем функций, описанных в начале этой главы, если вы предполагаете использовать функцию как UDF в Excel.

## Создание функций для Excel

При написании функций-процедур для использования в качестве UDF в рабочих листах Excel необходимо знать несколько фактов, помимо общих требований для определенных пользователем функций:

- Определенные пользователем функции, которые будут использоваться в Excel, не должны иметь имена, похожие на записи ссылок на ячейку типа A1 или R1C1.
- Любые строковые (текстовые) данные, возвращаемые из VBA в Excel, не должны иметь более 255 символов в длину. Если UDF возвращает строку, имеющую больше 255 символов в длину, в ячейку рабочего листа, Excel укорачивает строку до максимальной длины 255 символов перед вставкой ее в ячейку.
- При написании UDF, возвращающей значение даты для Excel убедитесь, что задаете тип результата функции как **Date**. Excel применяет формат **Date** для результата функции в ячейке рабочего листа только, если результат имеет VBA-тип **Date**.

Существует еще одно обстоятельство, связанное с тем, в какой момент времени Excel фактически вызывает определенную пользователем функцию для вычисления или повторного вычисления формулы ячейки рабочего листа, частью которой является UDF.

По умолчанию Excel вызывает UDF для повторного вычисления формулы ячейки рабочего листа всякий раз при изменении значений, используемых для аргументов функции, во многом так же, как для любой формулы рабочего листа. Однако можно задать UDF так, чтобы Excel повторно вычислял ее при повторном вычислении любой ячейки в рабочем листе.

Следует пометить UDF как *изменяющуюся (volatile)* всякий раз, когда значения ее аргументов не получаются из значений других ячеек рабочего листа. Например, предположим, функция с именем **Общая\_Стоимость** вычисляет стоимость услуг поставщика сети Internet с учетом НДС (налог на добавленную стоимость). UDF может иметь два аргумента: один для определения ячейки рабочего листа, содержащей значение стоимости услуг без учета НДС, а другой — для определения ячейки рабочего листа, содержащей процентную ставку НДС. Поскольку аргументы функции **Общая\_Стоимость** определяют координаты ячейки, изменение содержимого ячейки не меняет значение аргументов функции **Общая\_Стоимость**. При редактировании содержимого ячеек общей стоимости или общей прибыли, Excel не вычисляет повторно функцию **Общая\_Стоимость**, поэтому функция **Общая\_Стоимость** может выдать ошибочные результаты. Если пометить эту UDF как изменяющуюся, она будет всегда отображать правильное значение, потому что Excel будет всегда повторно вычислять ее при каждом изменении любой ячейки в рабочем листе.

Определенная пользователем функция, которая перевычисляется при каждом изменении любой ячейки в рабочем листе Excel, называется *изменяющейся* функцией. Чтобы пометить UDF как изменяющуюся, добавьте следующий оператор к функции сразу за ее объявлением:

```
Application.Volatile
```

Этот VBA-оператор помечает UDF как изменяющуюся функцию, и такая функция, на самом деле, является особым типом процедуры. Эта процедура называется *методом (method)*, принадлежащим Excel (объекты, методы и свойства описываются в главе 6). Поскольку метод принадлежит Excel, необходимо при использовании метода **Volatile** задавать объект **Application**, точно так же, как необходимо задавать объект **Application** при использовании функции Excel в операторе VBA. Листинг 4.12 показывает изменяющуюся функцию с именем **Общая\_Стоимость**.

---

**Листинг 4.12.** Изменяющаяся определенная пользователем функция в Excel

---

```
1: Function Общая_Стоимость(Сумма As Currency, _  
2:   Ставка_НДС As Single) As Currency  
3:   Application.Volatile  
4:   Общая_Стоимость = Сумма * (Ставка_НДС / 100#) + Сумма  
5: End Function
```

---

Функция **Общая\_Стоимость** имеет два обязательных аргумента: **Сумма** и **Ставка\_НДС**. Один аргумент (**Сумма** — стоимость услуг без учета НДС) имеет тип **Currency**, другой (**Ставка\_НДС** процентная ставка НДС) — тип **Single**, так как представляет обычно число не более 100. Объявление функции заканчивается указанием того, что возвращаемое функцией значение имеет тип **Currency**, так как возвращает значения, представляющие деньги.

Строка 3 содержит оператор **Application.Volatile**; как и требуется, он является первым оператором в функции после объявления функции. Оператор **Application.Volatile** «регистрирует» определенную пользователем функцию с помощью Excel так, что Excel вызывает ее для перевычисления формулы ячейки рабочего листа, в которой появляется функция, каждый раз при перевычислении любой ячейки в рабочем листе.

Однако не следует злоупотреблять методом **Application.Volatile**. Если все функции рабочего листа будут изменяющимися, это может привести к ненужным перевычислениям, что увеличивает общее время, необходимое для перевычисления рабочего листа.

### **Пример разработки и использования функции рабочего листа**

В качестве еще одного примера рассмотрим способ представления некоторых данных с использованием функции пользователя. Необходимо решить следующую задачу. У нас имеется таблица с данными о товарах: наименования, цены и т.д. Эта таблица, в частности, может использоваться для формирования так называемых «прайс-листов». Цены на товары меняются так часто, что менеджеры едва успевают менять «ценники» на товары (небольшие листочки с информацией о наименовании и цены товара, а также даты формирования «ценника» — рис. 4.15).

**Рис. 4.15**

«Ценник» с информацией о товаре: наименование, цена (в руб.) и дата изменения цены

ЧП «Петров П.Б.»		
Bazookatone		
Цена	52 руб.	12.11.07

Эти «ценники» нам нужно формировать в одном из листов Excel (например, **Лист1**), поскольку таблицу с товарами из какого-либо источника данных также легко помещать именно в Excel (рис. 4.16).

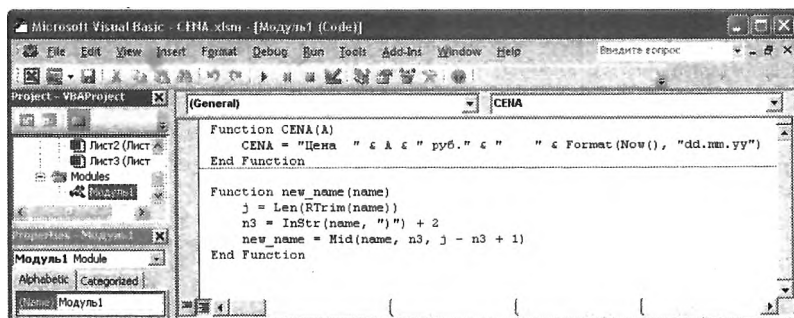
**Рис. 4.16**

Таблица с информацией о товарах (код, наименование, цены: в \$ и руб.)

	A	B	C	D	E	F	G
5	09801912021	(3DO) CPU Back	2,80	33,00			
6	00601911155	(3DO) Cyberdillo	2,00	52,00			
7	41711915521	(3DO) FIFA Soccer'95	2,60	65,00			
8	40711911573	(3DO) Fun n Games	2,70	33,00			
9	43701911295	(3DO) Micro Cosm	3,50	33,00			
10	26019093837	(3DO) Myst	2,00	65,00			
11	43701911320	(3DO) Night Trap (2cd)	6,80	65,00			
12	08601914313	(3DO) Olympic Soccer	3,50	33,00			
13	40711912525	(3DO) Pataank	2,70	33,00			
14	26019094442	(3DO) Perfect General	3,50	33,00			
15	26019094531	(3DO) PGA Tour'96	2,00	54,00			
16	03601911311	(3DO) Phychic Detective	2,60	65,00			
17	40711912002	(3DO) Real Pinbaal	2,70	33,00			

Для решения этой задачи следует к проекту **VBAProject(CENA.XLS)** добавить модуль и в этом модуле создать две функции, как показано на рис. 4.17.

**Рис. 4.17.**  
Функции **CENA** и **new\_name** в модуле проекта для форматирования данных в виде «ценников»



Функция **CENA** возвращает строку со словом «Цена», значением цены из четвертого столбца листа **Лист1**, словом (для приложений Windows это два слова) «руб.» и текущей датой.

Функция **new\_name** возвращает наименование товара без служебной информации, заключенной в круглые скобки. Для этого при помощи функции **InStr** находится позиция символа «)» (первое вхождение слева), а затем из

входной строки (полного имени) выделяются символы, находящиеся справа от символа «)».

Далее в другом рабочем листе (например, **Лист2**) необходимо заполнить, например, три ячейки первой строки заголовком ЧП «Петров П.В.» — наименование торгующей организации (рис. 4.15). В ячейки второй строки следует записать формулы **=new\_name(Лист1!B1)**, **=new\_name(Лист1!B2)** и **=new\_name(Лист1!B3)**, соответственно. В ячейки третьей строки остается поместить формулы **=CENA(Лист1!D1)**, **=CENA(Лист1!D2)** и **=CENA(Лист1!D3)**, соответственно (считаем, что цены в рублях записаны в колонке D). Сразу же после выполнения этих операций на этом листе должна появиться информация о товарах в том виде, как это показано на рис. 4.18.

**Рис. 4.18**

Необходимо заполнить только первые несколько строк с использованием функций **CENA** и **new\_name** для форматирования данных в виде «ценников»

	A	B	C
1	ЧП «Петров П.В.»	ЧП «Петров П.В.»	ЧП «Петров П.В.»
2	Stellar 7	Alone in the Dark 2	Bazookatone
3	Цена 32 руб. 12.11.07	Цена 50 руб. 12.11.07	Цена 52 руб. 12.11.07
4			
5			
6			

Осталось совсем немного — выделить ячейки **A1–D3**, поместить курсор мыши в правый нижний угол ячейки **D3** (курсор должен принять вид темного крестика), протащить курсор вниз на число ячеек кратное трем и с удовольствием наблюдать на экране довольно легко получившиеся «ценники» (рис. 4.19). Если данные о товарах изменились, следует только исправить их в листе **Лист1**.

**Рис. 4.19**

Задача форматирования данных в виде «ценников» решена

	A	B	C
1	ЧП «Петров П.В.»	ЧП «Петров П.В.»	ЧП «Петров П.В.»
2	Stellar 7	Alone in the Dark 2	Bazookatone
3	Цена 32 руб. 12.11.07	Цена 50 руб. 12.11.07	Цена 52 руб. 12.11.07
4	ЧП «Петров П.В.»	ЧП «Петров П.В.»	ЧП «Петров П.В.»
5	Cowboy Casino	CPU Back	Cyberdillo
6	Цена 49 руб. 12.11.07	Цена 33 руб. 12.11.07	Цена 52 руб. 12.11.07

# Изменение порядка выполнения операторов в VBA

До сих пор мы имели дело с процедурами и функциями, которые VBA выполняет только в линейном порядке. Во многом так же VBA выполняет записанный рекордером макрос: VBA начинает выполнение кода с первого оператора после строки объявления процедуры или функции и продолжает выполнять каждый оператор построчно до тех пор, пока не будет достигнут оператор **End Sub** или **End Function**, отмечающий конец определения этой процедуры или функции, или до тех пор, пока не возникнет runtime-ошибка (на это, очевидно, полагаться не стоит).

Подобные процедуры и функции, хотя и могут выполнять очень сложные задачи, но не могут менять порядок выполнения операторов-инструкций при определенных обстоятельствах. На самом деле, очень часто встречаются такие ситуации, когда необходимо, чтобы процедуры или функции выполняли различные действия при разных условиях. Например, если вы пишете процедуру, которая получает имя рабочей книги от пользователя, а затем открывает эту рабочую книгу, может понадобиться, чтобы ваша процедура имела возможность создать рабочую книгу, если книга еще не существует. В подобной ситуации необходимо, чтобы процедура выполняла те или иные операторы, предоставляя пользователю возможность создать рабочую книгу или отказаться от продолжения выполнения кода, если, например, пользователь просто еще не перезаписал необходимую ему рабочую книгу с переносного накопителя.

Команды, изменяющие порядок выполнения операторов, часто используют оценку конкретных элементов данных для выбора различных предопределенных действий. Например, можно написать процедуру, проверяющую, все ли числа столбца в рабочем листе находятся в диапазоне от 1 до 10. Эта процедура может проверять каждый элемент ввода в столбце отдельно и выполнять некоторый набор операторов, когда встречается элемент ввода вне указанного диапазона.

Изменение порядка выполнения операторов не всегда связано с реакцией на возникающие в процессе работы программы проблемы. Если вам необходимо, чтобы пользователь процедуры делал некоторый выбор по поводу последующего действия процедуры, можно использовать функцию **InputBox** для получения текстового ввода от пользователя или использовать функцию **MsgBox**, давая возможность пользователю делать выбор щелчком на команд-

ной кнопке в окне сообщения. После получения пользовательского выбора процедура должна выполнить действие, соответствующее этому выбору.

Конечно, процедуры не могут на самом деле «принимать решения» так же, как это делает человек. Однако процедуры «могут выбирать» предопределенные действия на основе простых условий и «принимать» относительно сложные решения, объединяя более простые решения, предоставленные разработчиком кода.

При использовании VBA-операторов изменения порядка выполнения кода определяется условие или набор условий, при которых VBA выполняет ту или иную *ветвь (branch)* кода процедуры. Поскольку такие операторы влияют на последовательность выполнения программы, их часто называют операторами *управления потоком (flow control)* или операторами *управления программой (program control)*, но на практике они более известны как операторы *условного и безусловного перехода (conditional и unconditional branching)*.

Оператор условного перехода — это структура, которая выбирает ту или иную ветвь кода процедуры на основе некоторого предопределенного условия или группы условий. *Оператор безусловного перехода* — это оператор, просто изменяющий последовательность выполнения кода процедуры независимо ни от какого конкретного условия. Условный переход используется гораздо чаще, чем безусловный.

При выполнении оператора условного перехода, такого как **If...Then**, VBA сначала оценивает указанное условие. Если условие равно **True**, VBA выполняет заданную группу операторов. Чтобы определить условие для оператора условного перехода, используется логическое выражение. При выполнении оператора безусловного перехода (**GoTo**) VBA немедленно начинает выполнение операторов, указанных командой перехода.

Критерии, на основе которых VBA «принимает решение» в операторах условного перехода, определяются путем создания логического выражения, описывающего условие, при котором необходимо выполнение или невыполнение определенной серии операторов. Для создания логических выражений в операторах перехода используются различные операторы сравнения и логические операторы VBA (которые описаны в главе 3).

## Простой выбор

Простейшими VBA-операторами изменения порядка выполнения кода являются операторы **If...Then** и **If...Then...Else**. Оператор **If...Then** позволяет VBA выбрать единственную альтернативную ветвь выполнения процедуры. Связанный с ним оператор **If...Then...Else** дает возможность VBA выбирать из двух альтернативных ветвей кода процедуры на основе оценки того, является ли указанное условие равным **True**.

Оператор **If...Then** позволяет выбрать единственную альтернативную ветвь кода в процедуре или функции.

Оператор **If...Then** имеет две различные формы синтаксиса. Простая форма — это однострочный оператор **If...Then**:

## Синтаксис

### *If Condition Then Statements*

*Condition* — любое логическое выражение, а *Statements* — один, несколько или ни одного оператора VBA; все операторы должны помещаться в одной и той же строке. При выполнении подобного оператора VBA сначала оценивает логическое выражение, представленное с помощью *Condition*; если это логическое выражение равно **True**, то выполняется оператор (или операторы) после ключевого слова **Then** до конца строки. Затем VBA возобновляет выполнение кода с первого оператора после строки, содержащей оператор **If...Then**.

Если логическое выражение, представленное с помощью *Condition*, равно **False**, то выполняется первый оператор в строке после строки, содержащей оператор **If...Then**, без выполнения альтернативной ветви. Следующий фрагмент процедуры показывает типичный однострочный оператор **If...Then**

```
If temperature > 100 Then MsgBox «Слишком горячо!»
```

Можно включать несколько операторов VBA в одну строку, отделяя каждый из них двоеточием (:), как показано в следующем примере:

```
Statement1 : Statement2 : StatementN
```

В этой строке *Statement1*, *Statement2* и *StatementN* каждый представляет один допустимый оператор VBA. Можно включать столько операторов в одну строку, сколько необходимо до максимальной длины строки для модуля. Однако строки со многими операторами трудно читать и понимать. Обычно следует помещать только один оператор в каждую строку.

В листинге 5.1 приведен код функции, которая принимает от пользователя новое наименование книги в виде строки и, если строка пустая (не содержит символов, кроме пробелов), выдает сообщение об этом.

### Листинг 5.1. Функция **GetBookName**

```
1: Function GetBookName() As String
2:   Dim lTitle As String, lPrmpt As String, lDflt As String
3:
4:   lTitle = "Ввод наименования книги"
5:   lPrmpt = "Введите наименование новой книги"
6:   lDflt = "Отчет о продажах"
7:
8:   'использование InputBox для получения имени файла.
9:   GetBookName = Trim(InputBox(prompt:=lPrmpt, _
10:                               Title:=lTitle, _
11:                               Default:=lDflt))
12:
13:   'проверить наличие строки ввода
14:   If Len(GetBookName) = 0 Then _
15:       MsgBox "Наименование не введено!"
16:
17: End Function           'GetBookName
```



Операция получения наименования функции (строка 9) совмещена с удалением ненужных пробелов посредством функции **Trim**. Функция **Trim** принимает результат функции **InputBox** и возвращает введенную строку без левых и правых пробелов.

Обратите внимание на строку 15, которая содержит однострочный оператор **If...Then** (символ перехода на новую строку не является причиной считать оператор неодноточным: это — только возможность написания более длинного оператора, чем позволяет размер страницы). При выполнении строки 15 VBA сначала оценивает условное выражение **Len(GetBookName) = 0**, которое возвращает значение **True**, если пользователь ввел строку, состоящую только из пробелов (здесь была использована ранее обсуждавшаяся функция **LenTrim**). В этом случае выполняется оператор:

```
MsgBox "Наименование не введено!"
```

Если выражение **Len(GetBookName) = 0** возвращает **False**, то VBA не выполняет оператор после ключевого слова **Then**.

Видимо, сейчас вам уже не трудно будет написать процедуру тестирования функции, поэтому мы не будем на этом останавливаться, а сразу перейдем к «критике» функции. Недостатком этой функции является то, что в случае ввода в диалоговом окне функции **InputBox** одних пробелов мы только получим об этом сообщение, а функция **GetBookName** возвратит нам пустую строку. Если мы только тестируем эту функцию, ничего плохого в этом случае не произойдет. А что, если нам нужно создать с полученным наименованием какой-либо объект: книгу, лист, файл? На этот случай необходимо всегда формировать какое-то запасное (по умолчанию) имя. На самом деле, в этом и заключается смысл создания оболочки вокруг функции **InputBox**. Если бы все пользователи были «правильными», можно было бы всегда обходиться функцией **InputBox**, поскольку «правильные» пользователи никогда не догадались бы в качестве имени новой книги ввести пробелы.

Итак, нам нужно, чтобы функция ввода наименования книги всегда возвращала непустую строку, даже если пользователь решил пошутить и сломать вашу программу. Для этого необходимо, кроме сообщения о том, что шутка оценена, присвоить возвращаемому значению функции имя по умолчанию, например, то, которое было ранее предложено пользователю, но отвергнуто им. Для этого рассмотрим следующую форму оператора **If...Then**.

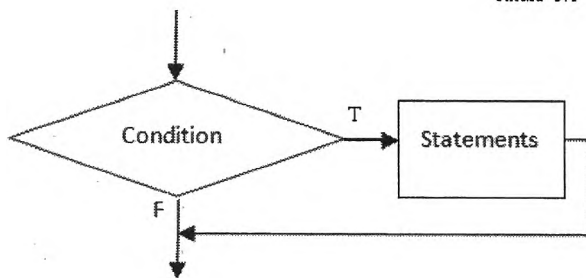
Вторая форма синтаксиса оператора **If...Then** называется *блоком (block)* оператора **If**. В блоке оператора **If...Then** условие и операторы записываются в отдельных строках, как показано в следующей синтаксической форме:

## Синтаксис

```
If Condition Then
    Statements
End If
```

Здесь *Condition*, как и в однострочном операторе **If...Then**, представляет логическое выражение, определяющее условие, при котором VBA следует выполнить альтернативные операторы. *Statements* — это один, несколько или ни одного оператора VBA; операторы могут находиться в одной или нескольких строках. Наконец, ключевые слова **End If** указывают VBA, что достигнут конец альтернативной ветви операторов. Ключевые слова **End If** должны появляться в отдельной строке, хотя в эту строку можно включать конечный комментарий (см. схему 5.1).

Схема 5.1



Как и в случае с однострочным оператором **If...Then**, VBA сначала оценивает логическое выражение, представленное с помощью *Condition*. Если это выражение равно **True**, VBA выполняет операторы в альтернативной ветви, начиная с первого оператора в строке, после строки, содержащей ключевые слова **If...Then**. VBA продолжает выполнение операторов в альтернативной ветви до тех пор, пока не достигнет ключевых слов **End If**. Затем продолжается выполнение операторов, начиная с первого оператора после **End If**.

Изменим код функции **GetBookName**, как показано в листинге 5.2.

#### Листинг 5.2. Функция **GetBookName2**

```

1: Function GetBookName2() As String
2:     Dim lTitle As String, lPrmpt As String, lDflt As String
3:
4:     lTitle = "Ввод наименования книги"
5:     lPrmpt = "Введите наименование новой книги"
6:     lDflt = "Отчет о продажах"
7:
8:     'использование InputBox для получения имени файла.
9:     GetBookName2 = Trim(InputBox(prompt:=lPrmpt, _
10:         Title:=lTitle, _
11:         Default:=lDflt))
12:
13:     'проверить наличие строки ввода
14:     If Len(GetBookName2) = 0 Then
15:         MsgBox "Наименование не введено!"
16:         GetBookName2 = lDflt 'имя по умолчанию
17:     End If
18:
19: End Function           'GetBookName
20:
21: Sub test_GetBookName2()
22:     'Тестирование функции GetBookName2
23:     NewBookName = GetBookName2()
24:     MsgBox NewBookName
25: End Sub
  
```

Листинг 5.2 содержит код уже обсужденной функции **GetBookName2**, которая никогда не возвращает пустую строку, и тестирующей процедуры, использующей **MsgBox** для отображения результата вызова **GetBookName2**.

В некоторых случаях при вводе наименования нового файла (книги) удобнее задавать конкретное имя по умолчанию, которое зависит от каких-либо дополнительных условий. Это требует ввода имени по умолчанию в качестве аргумента функции. Но, с другой стороны, хотелось бы оставить возможность использовать функцию ввода без аргументов. Поэтому прежде чем рассматривать далее более сложные конструкции операторов управления последовательностью выполнения кода, обсудим такую возможность VBA, как необязательные аргументы функции.

## Использование необязательных аргументов

Обычно аргументы, которые приводятся в списке аргументов функции-процедуры, должны предоставляться каждый раз, когда вызывается функция; эти аргументы являются *обязательными* (*required*). Вы уже знаете о необязательных аргументах функции по работе с VBA-функциями **InputBox** и **MsgBox**. Но, что самое замечательное, вы также можете использовать необязательные аргументы для ваших собственных функций-процедур.

При включении необязательных аргументов в список аргументов необходимо перечислять сначала все обязательные аргументы; после первого необязательного аргумента все последующие аргументы в этом списке должны быть также необязательными. Можно даже определять тип данных и значение по умолчанию для необязательных аргументов функции.

Для создания необязательного аргумента вставьте VBA-ключевое слово **Optional** перед именем этого аргумента в списке аргументов при объявлении функции-процедуры. Листинг 5.3 содержит очередную версию функции **GetBookName**, которая позволяет указать в качестве необязательного аргумента имя новой книги. С помощью встроенной функции **IsMissing** в коде определяется факт использования необязательного аргумента при вызове функции **GetBookName**. Если аргумент не был введен, функция использует свою внутреннюю строковую константу в качестве наименования книги.

### Листинг 5.3. Функция **GetBookName3**

```
1: Function GetBookName3(Optional lDflt) As String
2:     Dim lTitle As String, lPrmpt As String
3:
4:     lTitle = "Ввод наименования книги"
5:     lPrmpt = "Введите наименование новой книги"
6:
7:     'включен ли lDflt в список аргументов
8:     If IsMissing(lDflt) Then lDflt = "Книга1"
9:
10:    'использование InputBox для получения имени файла.
11:    GetBookName3 = Trim(InputBox(prompt:=lPrmpt, _
12:                                Title:=lTitle, _
13:                                Default:=lDflt))
14:
15:    'проверить наличие строки ввода
16:    If Len(GetBookName3) = 0 Then
17:        MsgBox "Наименование не введено!"
18:        GetBookName3 = lDflt 'имя по умолчанию
19:    End If
20:
```

```
21: End Function      'GetBookName
22:
23:
24: Sub test_GetBookName3()
25:     'Тестирование функции GetBookName2
26:
27:     'вызов функции без параметра
28:     NewBookName = GetBookName3()
29:     MsgBox "Наименование новой книги: " & NewBookName
30:
31:     'вызов функции с параметром
32:     NewBookName = GetBookName3(1Dflt:="Книга20")
33:     MsgBox "Наименование новой книги: " & NewBookName
34: End Sub
```

---

Продолжим рассмотрение операторов, позволяющих изменять последовательность выполнения кода процедуры или функции.

## Выбор ветви с помощью If...Then...Else

Оператор **If...Then** дает возможность задавать одну альтернативную ветвь операторов в процедуре. Однако часто бывает необходимо выбрать одну из двух различных ветвей операторов в зависимости от определенного условия. Для этого VBA предоставляет операторы **If...Then...Else** и **If...Then...ElseIf**.

Как и **If...Then**, VBA-оператор **If...Then...Else** имеет две формы: однострочную и блочную. Синтаксис однострочного оператора **If...Then...Else** следующий:

### Синтаксис

---

*If Condition Then Statements Else ElseStatements*

*Condition* — любое допустимое логическое выражение. *Statements* — один или несколько операторов VBA. Как и в однострочном операторе **If...Then**, все операторы и ключевые слова однострочного **If...Then...Else** должны находиться в одной и той же строке.

---

При выполнении однострочного оператора **If...Then...Else** VBA сначала оценивает логическое выражение, представленное с помощью *Condition*; если это выражение равно **True**, VBA выполняет операторы (представленные с помощью *Statements*) между ключевыми словами **Then** и **Else** и возобновляет выполнение кода с первого оператора после строки, которая содержит **If...Then...Else**.

Если логическое выражение, представленное с помощью *Condition*, равно **False**, VBA выполняет операторы после ключевого слова **Else** до конца строки (представленные с помощью *ElseStatements*) и продолжает выполнение кода с первого оператора после строки, содержащей **If...Then...Else**.

В следующей строке показан типичный пример однострочного оператора **If...Then...Else**:

```
If Вес > 100 Then MsgBox "Тяжело!" Else MsgBox "Не так тяжело!"
```

В этом примере, если значение переменной **Вес** больше 100, то условное выражение равно **True** и VBA выполняет оператор **MsgBox** для отображения сообщения «Тяжело!». Если переменная **Вес** содержит число, равное или мень-

шее 100, то рассматриваемое условие равно **False** и VBA выполняет оператор **MsgBox** после ключевого слова **Else** для отображения сообщения «Не так тяжело!».

Как видно из примера однострочного оператора **If...Then...Else**, однострочную форму не всегда просто читать. Кроме того, поскольку все элементы однострочного оператора **If...Then...Else** должны находиться в одной и той же строке, размер и количество операторов, которые можно включать в альтернативные ветви выполнения, ограничиваются имеющимся в строке местом.

Блок операторов **If...Then...Else** легче читать и понимать и, поскольку можно располагать операторы в разных строках внутри блока оператора **If...Then...Else**, он не имеет ограничения по размеру и числу операторов, которые можно помещать в альтернативные ветви. Блок оператора **If...Then...Else** имеет следующий синтаксис:

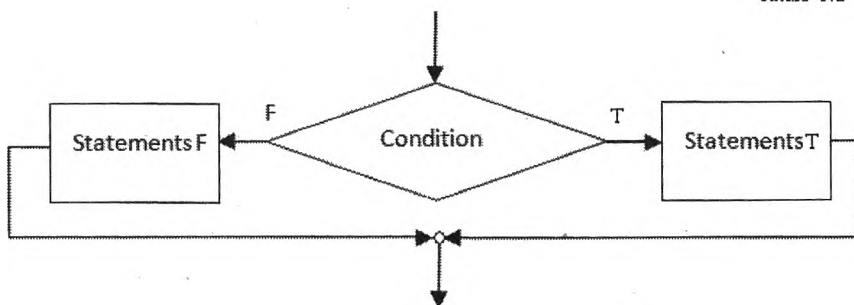
### Синтаксис

```
If Condition Then
    Statements
Else
    ElseStatements
End If
```

*Condition* — любое допустимое логическое выражение; *Statements* и *ElseStatements* представляют один, несколько или ни одного оператора VBA. При выполнении блока оператора **If...Then...Else** VBA сначала оценивает логическое выражение, представленное с помощью *Condition*. Если это выражение равно **True**, то VBA выполняет все операторы (представленные с помощью *Statements*) между ключевым словом **Then** и ключевым словом **Else**. Затем VBA возобновляет выполнение кода с первого оператора, появляющегося после ключевых слов **End If**, которые указывают на конец блока оператора **If...Then...Else**.

Если логическое выражение, представленное с помощью *Condition*, равно **False**, VBA выполняет все операторы (представленные с помощью *ElseStatements*) между ключевым словом **Else** и ключевыми словами **End If**. Затем VBA продолжает выполнение кода с первого оператора, появляющегося после ключевых слов **End If**. Как и в блоке оператора **If...Then**, ключевые слова **End If** должны помещаться в отдельной строке, хотя в эту строку можно добавлять конечный комментарий (см. схему 5.2).

Схема 5.2



Оператор **If...Then...Else** выбирает одну или другую ветвь, но никогда не выбирает обе ветви одновременно. В следующем примере (листинг 5.4) показан типичный блок оператора **If...Then...Else**:

#### Листинг 5.4. Функция **GetBookName4**

```
1: Function GetBookName4(Optional lDflt) As String
2:   Dim lTitle As String, lPrmpt As String, lVar As String
3:
4:   lTitle = "Ввод наименования книги"
5:   lPrmpt = "Введите наименование новой книги"
6:
7:   'включен ли lDflt в список аргументов
8:   If IsMissing(lDflt) Then lDflt = "Отчет о продажах"
9:
10:  'использование InputBox для получения имени файла.
11:  lVar = Trim(InputBox(prompt:=lPrmpt, _
12:                      Title:=lTitle, _
13:                      Default:=lDflt))
14:
15:  'проверить наличие строки ввода
16:  If Len(lVar) = 0 Then
17:    MsgBox "Наименование не введено!"
18:    GetBookName4 = lDflt 'имя по умолчанию
19:  Else
20:    GetBookName4 = lVar
21:  End If
22:
23: End Function           'GetBookName
```

Никаких преимуществ у этого кода по сравнению с кодом листинга 5.3 нет, но здесь был использован оператор **If...Then...Else** и строки 16–21 представляют собой более понятный код. Этот пример содержит тот же оператор, приводимый для однострочного **If...Then...Else**, но теперь использует форму блока.

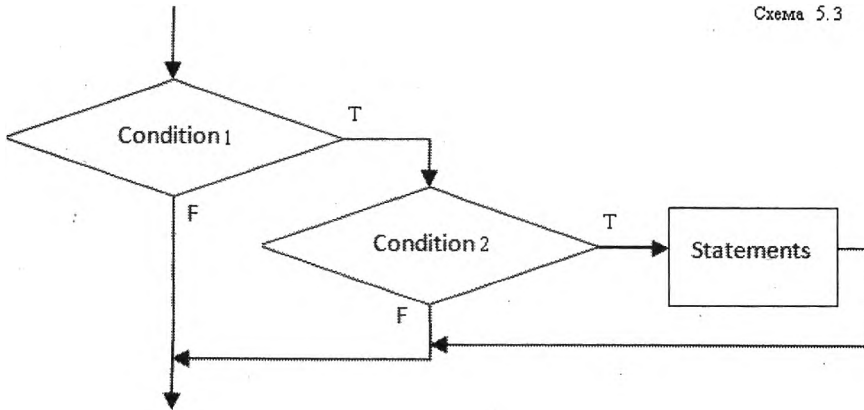
## Сложный выбор

Вы узнали, как создавать операторы перехода, выбирающие одну или одну из двух альтернативных ветвей кода процедуры. Однако часто бывает необходимо выполнить более сложный выбор в процедурах, выбирая между тремя, четырьмя и более ветвями.

### Вложенные операторы **If...Then**

При необходимости принятия более сложных решений можно помещать оператор **If...Then** или **If...Then...Else** внутрь другого оператора **If...Then** или **If...Then...Else**, что называется *вложением операторов (nesting)*. (Вложение означает помещение одного типа структуры управления выполнением кода внутрь другой.) (см. схему 5.3)

Схема 5.3



Хотя можно вкладывать и однострочные формы операторов **If...Then** и **If...Then...Else**, такие операторы трудно понимать. При вложении **If...Then** и **If...Then...Else** используйте для ясности блочную форму этих операторов.

Листинг 5.5 показывает простую процедуру для иллюстрации того, как работают вложенные операторы **If...Then...Else**. Процедура **EvalDiscount** получает от пользователя количество покупаемого клиентом товара (числовое значение), а затем оценивает это число и определяет скидку, на которую может рассчитывать клиент.

#### Листинг 5.5. Вложенные операторы **If...Then...Else**

```

1 Sub EvalDiscount()
2 'Определение скидки (в %) в зависимости от
3 'количества продаваемого товара
4   Dim lStr As String
5   Dim IntNum
6   IntNum = Application.InputBox( _
7       prompt:="Введите количество товара", _
8       Title:="Процедура EvalDiscount", _
9       Type:=1)
10  If Not (TypeName(IntNum) = "Boolean") Then
11
12      If IntNum > 1000 Then
13          lStr = "10"
14      Else
15          If IntNum > 500 Then
16              lStr = "6"
17          Else
18              lStr = "0"
19          End If
20      End If
21      MsgBox "Скидка " & lStr & "%"
22  Else
23      MsgBox "Количество не указано"
24  End If
25
26 End Sub
  
```

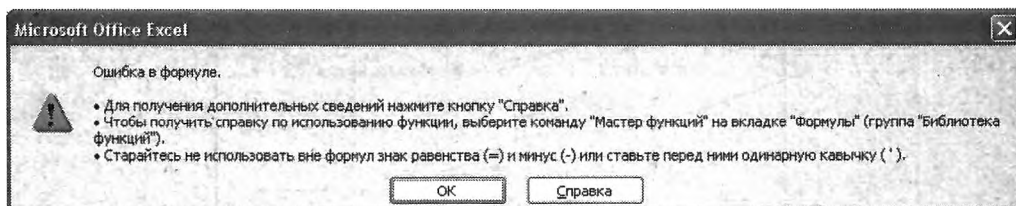
Написанная так процедура будет работать только в Excel, так как она использует метод **Application.InputBox**. Этот метод применен для того, чтобы не дать пользователю во время работы функции **InputBox** ввести что-либо, кроме числа. VBA отображает сообщение об ошибке, если пользователь вводит не число (рис.5.1), и ожидает до тех пор, пока пользователь не введет численное значение или не выберет кнопку **Cancel**. Если пользователь, не вводя данные в окне функции **Application.InputBox**, щелкнет на кнопке **OK**, VBA также выдаст сообщение об ошибке (рис.5.2).

**Рис. 5.1**

Сообщение Excel при неправильном вводе в окне функции **Application.InputBox**



Обратите внимание на степень защищенности программы листинга 5.5 от неправильных действий пользователя. Во-первых, функция **Application.InputBox** не дает возможности ввести в окно ввода нечисловое значение. Во-вторых, если пользователь использует «спасительную» кнопку **Cancel**, чтобы отказаться от ввода, программа (в строке 10) проверит результат ввода, который при щелчке на кнопке **Cancel** имеет тип **Boolean**, и выберет необходимую ветвь оператора **If...Then...Else**. При щелчке на кнопке **Cancel** выполнится оператор в строке 23 и на экран будет выдано сообщение о том, что количество товара не указано.



**Рис. 5.2.** Сообщение Excel при щелчке на кнопке **OK** без ввода данных в окне функции **Application.InputBox**

## Использование **If...Then...ElseIf**

VBA предоставляет сокращенную версию оператора **If...Then...Else**, являющуюся сжатым эквивалентом вложенных операторов **If...Then...Else**, показанных в листинге 5.5. Такой краткой формой является оператор **If...Then...ElseIf**.

Оператор **If...Then...ElseIf** имеет следующий синтаксис:

### Синтаксис

```
If Condition1 Then
    Statements
ElseIf Condition2
    ElseIfStatements
[Else
    ElseStatements]
End If
```

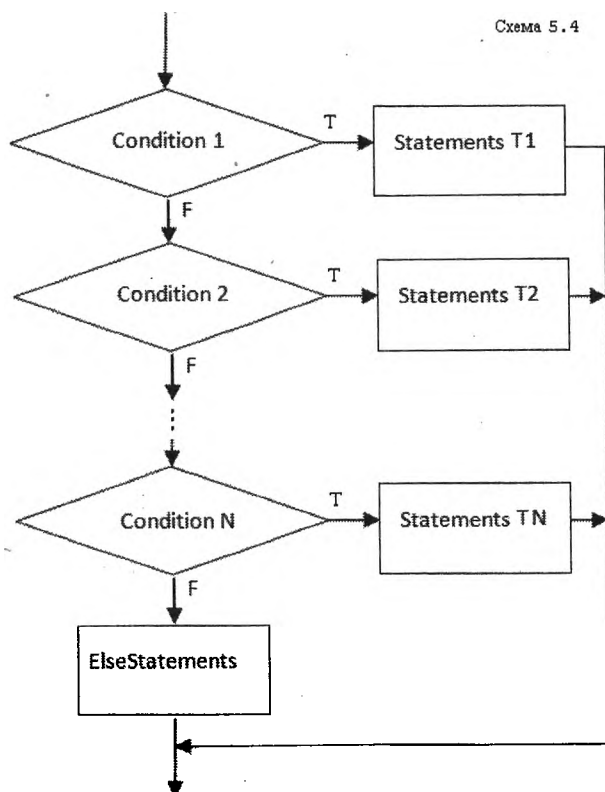


*Condition1* и *Condition2* — любые логические выражения; *Statements*, *ElseStatements* и *ElseStatements* — один, несколько или ни одного оператора VBA.

При выполнении оператора **If...Then...Elseif** VBA сначала оценивает логическое выражение, представленное с помощью *Condition1*. Если это выражение равно **True**, то VBA выполняет все операторы (представленные с помощью *Statements*) между ключевыми словами **Then** и **Elseif**. Затем VBA возобновляет выполнение кода с первого оператора после ключевых слов **End If**, которые указывают на конец оператора **If...Then...Elseif**.

Если логическое выражение *Condition1* равно **False**, VBA оценивает логическое выражение, представленное с помощью *Condition2*. Если выражение *Condition2* равно **True**, VBA выполняет все операторы (представленные с помощью *ElseStatements*) между ключевым словом **Elseif** и ключевыми словами **End If** (или необязательным **Else**). Затем VBA продолжает выполнение кода с первого оператора, появляющегося после ключевых слов **End If**. Ключевые слова **End If** должны находиться на отдельной строке.

Если логическое выражение *Condition2* равно **False**, то VBA пропускает *ElseStatements* и продолжает выполнение кода с первого оператора после ключевых слов **End If**. Можно по желанию включать оператор **Else** в оператор **If...Then...Elseif**. Если логическое выражение *Condition1* и логическое выражение *Condition2* равны **False** и имеется оператор **Else**, то VBA выполняет операторы, представленные с помощью *ElseStatements*. После выполнения операторов *ElseStatements* VBA продолжает выполнение кода с первого оператора после ключевых слов **End If** (см. схему 5.4).



В следующем листинге (5.6) показано, как с учетом вышеизложенного можно изменить строки 12–20 листинга 5.5.

#### Листинг 5.6. Использование If...Then...ElseIf

```
1 Sub EvalDiscount2()  
2 'Определение скидки (в %) в зависимости от  
3 'количества продаваемого товара  
4     Dim lStr As String  
5     Dim IntNum  
6     IntNum = Application.InputBox( _  
7         prompt:="Введите количество товара", _  
8         Title:="Процедура EvalDiscount2", _  
9         Type:=1)  
10     If Not (TypeName(IntNum) = "Boolean") Then  
11  
12         If IntNum > 1000 Then  
13             lStr = "10"  
14         ElseIf IntNum > 500 Then  
15             lStr = "5"  
16         Else  
17             lStr = "0"  
18         End If  
19         MsgBox "Скидка " & lStr & "%"  
20     Else  
21         MsgBox "Количество не указано"  
22     End If  
23  
24 End Sub
```

Этот оператор **If...Then...ElseIf** действует точно так же, как вложенный оператор **If...Then...Else** в строках 12–20 листинга 5.5, и с теми же результатами, только эта форма более компактна.

При желании можно включать более одного предложения **ElseIf** в оператор **If...Then...ElseIf**; все предложения **ElseIf** следуют перед **Else**. VBA выполняет операторы в предложении **ElseIf**, если только логическое выражение в **ElseIf** равно **True**.

Использование оператора **If...Then...ElseIf** является вопросом профессионального предпочтения. Многие программисты считают, что их код более понятен и удобен при использовании вложенных операторов **If...Then...Else** и избегают использования оператора **If...Then...ElseIf**. Первоначально оператор **If...Then...ElseIf** был добавлен в BASIC для обеспечения возможностей, которые обеспечиваются оператором **Select...Case** (описывается в следующем разделе); он остается в VBA для легкой трансляции существующих программ с BASIC на VBA. При выполнении выбора из нескольких вариантов оператор **Select...Case** превосходит вложенный оператор **If...Then...Else** или еще более компактный оператор **If...Then...ElseIf**.

#### Оператор Select...Case

Примеры использования вложенных операторов **If...Then...Else** и **If...Then...ElseIf** показывают, что эти операторы позволяют выбирать из трех ветвей кода, но что если необходимо выбрать между 5, 8 или 10 различными возможными действиями?

Для выполнения выбора из нескольких возможных ветвей кода можно вкладывать операторы **If...Then...Else** на много уровней вглубь, но уследить за ходом выполнения ветвей становится прогрессирующе труднее. Альтернативно, можно было бы добавить дополнительные операторы **ElseIf** в оператор **If...Then...ElseIf** с оператором **ElseIf** для каждой условной ветви. Оператор **If...Then...ElseIf** имеет подобную проблему: когда имеется много операторов **ElseIf**, оператор **If...Then...ElseIf** становится трудно читать и сопровождать.

К «счастью», VBA имеет условный оператор перехода для использования в случаях, когда необходимо выбирать из большого количества различных ветвей кода: оператор **Select Case**. Он работает во многом так же, как множество независимых операторов **If**, но он более понятен для того, кто пишет код, и того, кто читает этот код. Ключевые слова **Select Case** используются со многими операторами **Case**, где каждый оператор **Case** проверяет появление другого условия и выполняется только одна из ветвей **Case**. Ветвь **Case** может содержать один, несколько или ни одного оператора VBA.

Оператор **Select Case** имеет следующий синтаксис:

### Синтаксис

---

```
Select Case TestExpression
    Case ExpressionList1
        statements1
    Case ExpressionList2
        statements2
    .
    .
    .
    Case ExpressionListN

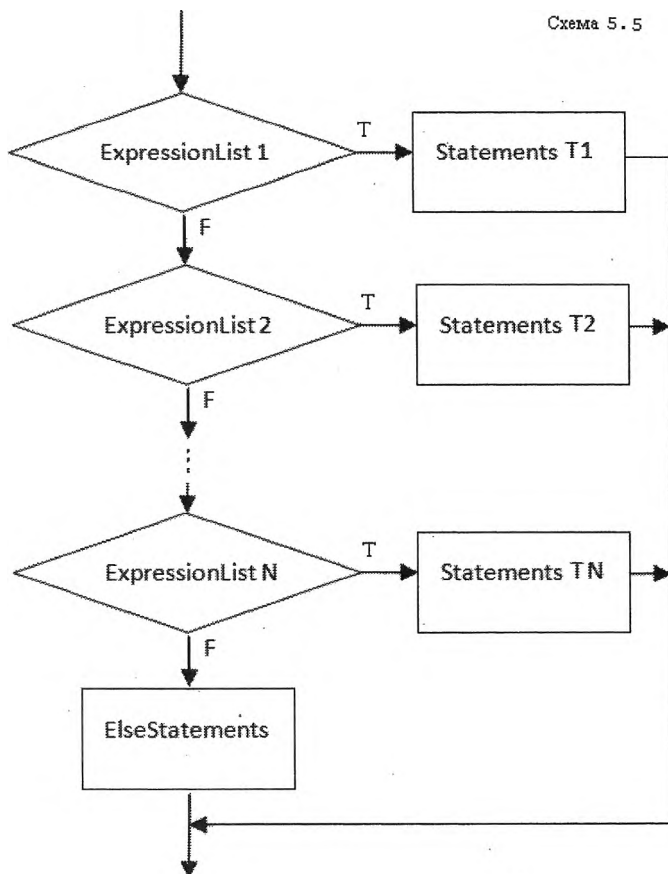
statementsN
[Case Else
    ElseStatements]
End Select
```

*TestExpression* — любое численное или строковое выражение. *ExpressionList1*, *ExpressionList2* и *ExpressionListN* — (каждый) представляют список логических выражений, отделенных запятыми. *Statements1*, *statements2*, *statementsN* и *ElseStatements* (каждый) представляют один, несколько или ни одного оператора VBA. В **Select Case** можно включать столько операторов **Case ExpressionList**, сколько необходимо (см. схему 5.5).

---

При выполнении VBA-оператора **Select Case** сначала оценивается *TestExpression*, а затем сравнивается результат этого выражения с каждым выражением, перечисленным в каждом *ExpressionList*. Если значение, представленное с помощью *TestExpression* совпадает с выражением в *ExpressionList* для одного из **Case**, VBA выполняет операторы для этого предложения **Case**. Если значение *TestExpression* совпадает более, чем с одним оператором **Case**, VBA выполняет только операторы в первом совпадающем предложении **Case**. Часто *TestExpression* — это просто имя одной переменной, математическое или численное, а не логическое выражение. Выражения в *ExpressionList* — это обычно логические выражения.

Схема 5.5



После завершения выполнения операторов в первом совпадающем операторе **Case** VBA продолжает выполнение кода с первого оператора после ключевых слов **End Select**, которые обозначают конец **Select Case**.

Если значение *TestExpression* не совпадает ни с каким из **Case**, а необязательный **Case Else** присутствует, VBA выполняет операторы, представленные с помощью *ElseStatements* перед переходом к оператору после **Select Case**.

В отдельных операторах **Case** выражение *ExpressionList* может состоять из одного или более выражений, отделяемых запятой. Список *ExpressionList* имеет следующий синтаксис:

### Синтаксис

*expression1, expression2, ..., expressionN*

Выражения в *ExpressionList* могут быть любыми численными, строковыми или логическими выражениями. Выражение в *ExpressionList* может также определять диапазон значений при использовании оператора **To**:

*expression1 To expression2*

Например, для определения диапазона чисел от 10 до 150 в операторе **Case ExpressionList** используется следующее выражение:

```
Case 10 To 150
```

Для выбора ветвей на основе сравнения, является ли *TestExpression* больше, меньше или равным какому-либо значению, или на основе какого-либо другого реляционного сравнения, используйте следующий синтаксис:

### Синтаксис

---

```
Is ComparisonOperator expression
```

В этой строке *ComparisonOperator* — это любые VBA-операторы отношения, кроме операторов **Is** и **Like**; *expression* — любое выражение VBA.

---

Для выполнения операторов в ветви **Case**, когда, например, *TestExpression* больше 10, используется следующее выражение:

```
Case Is > 10
```

Ключевое слово **Is** здесь — это не то же самое, что оператор **Is**. В этом операторе **Is** означает ссылку на *TestExpression*.

Листинг 5.7 представляет пример оператора **Select Case**.

### Листинг 5.7. Оператор **Select Case**

---

```
1 Sub EvalDiscount3()
2 'Определение скидки (в %) в зависимости от
3 'количества продаваемого товара
4 Dim lStr As String
5 Dim IntNum
6 IntNum = Application.InputBox( _
7     prompt:="Введите количество товара", _
8     Title:="Процедура EvalDiscount3", _
9     Type:=1)
10 If Not (TypeName(VarWeight) = "Boolean") Then
11     Select Case IntNum
12         Case Is > 1000
13             lStr = "10"
14         Case Is > 500
15             lStr = "5 "
16         Case Else
17             lStr = "0 "
18     End Select
19     MsgBox "Скидка " & lStr & "%"
20 Else
21     MsgBox "Количество не указано"
22 End If
23 End Sub
```

---

Обратите внимание на строки 11–18. После рассмотрения двух предыдущих примеров эти строки не могут не вызвать вздох облегчения. Эта конструкция хороша и простотой для понимания, и легкостью при наращивании новых условий. В остальном код этого примера переписан из предыдущих листингов.

## Безусловный переход

Оператор безусловного перехода, пожалуй, самый спорный оператор во всех языках программирования. В ранних языках он являлся почти единственным средством организации циклических выполнений блоков кода. Сторонники структурного программирования посвящают много времени критике программ и языков программирования, в которых используется этот оператор. Поэтому не очень увлекайтесь применением этого оператора в своем коде и старайтесь вспоминать о нем только в чрезвычайных ситуациях, когда все другие средства уже испробованы и не помогли.

Оператор безусловного перехода всегда изменяет порядок выполнения операторов в процедуре или функции VBA. При этом VBA не проверяет никаких условий (отсюда термин *безусловный* (*unconditional*)), а просто переходит к выполнению кода с другого места.

VBA имеет только один оператор безусловного перехода: **GoTo**. Существует очень мало причин для использования оператора **GoTo**; в действительности, процедуры, которые используют несколько операторов **GoTo**, трудны для понимания. Почти в каждом случае, когда вы можете использовать оператор **GoTo**, можно использовать один из операторов **If**, оператор **Select Case** или одну из структур организации циклов, о которых вы узнаете в главе 7, для выполнения той же задачи с большей легкостью и ясностью.

Оператор **GoTo** остался от ранних версий языка программирования BASIC, которые не имели сложного оператора принятия решений **Select Case**, описанного в этой главе, или мощных структур организации циклов. Программисты, работавшие с ранними версиями BASIC, использовали **GoTo** для имитации результатов более сложных операторов VBA.

Следует знать, как оператор **GoTo** работает в качестве оператора безусловного перехода, главным образом для того, чтобы понимать, как действует обработчик ошибок **GoTo**, но также потому, что вам может встретиться эта команда, используемая другими программистами.

Оператор **GoTo** имеет следующий синтаксис:

---

### Синтаксис

**GoTo** *line*

Выражение *line* представляет любую допустимую метку или номер строки в той же процедуре или функции, которая содержит оператор **GoTo**. При выполнении оператора **GoTo** VBA немедленно переходит к выполнению оператора в строке, определенной с помощью *line*.

---

Метка строки (*line label*) — это особый тип идентификатора, который задает определенную строку по имени. Метки строк имеют следующий синтаксис:

**Синтаксис**

*Name:*

Выражение *name* — любой допустимый идентификатор VBA. Метка строки может начинаться в любом столбце в строке, если она является первым непустым символом в строке. Номер строки (*line number*) — в основном, то же самое, что и метка строки, но он является просто номером, а не идентификатором. Использование номеров строк в процедурах VBA является крайне нежелательным, следует всегда использовать метку строки вместо них.

В листинге 5.8 показана процедура, демонстрирующая наиболее обоснованное использование оператора **GoTo** в процедуре VBA: эта процедура использует оператор **GoTo** для перехода к концу процедуры при выборе пользователем командной кнопки **Отмена** (Cancel) в окне ввода.

**Листинг 5.8. Пример использования оператора GoTo**

```

1 Sub EvalDiscount4()
2 'Определение скидки (в %) в зависимости от
3 'количества продаваемого товара
4   Dim lStr As String
5   Dim IntNum
6   IntNum = Application.InputBox( _
7       prompt:="Введите количество товара", _
8       Title:="Процедура EvalDiscount4", _
9       Type:=1)
10
11   If TypeName(VarWeight) = "Boolean" Then GoTo EndSubCancel
12
13   Select Case IntNum
14       Case Is > 1000
15           lStr = "10"
16       Case Is > 500
17           lStr = "5 "
18       Case Else
19           lStr = "0 "
20   End Select
21
22   MsgBox "Скидка " & lStr & "%"
23   GoTo EndSub
24
25 EndSubCancel:      'обработка отмены ввода в окне InputBox
26   MsgBox "Количество не указано"
27
28 EndSub:           'успешное окончание работы
29
30 End Sub

```

В строке 11 определяется, были ли введены данные в окне диалога. Если пользователем была выбрана кнопка **Отмена**, код продолжает выполняться со строки 25, на которую указывает метка **EndSubCancel**. Если же данные введены успешно, процедура продолжает выполняться со строки 13. В строке 23 снова встречается оператор безусловного перехода для того, чтобы обойти строку 26 и не выдавать сообщения о том, что количество не было введено.

Еще раз отметим, что так процедуры писать не нужно. Это — просто пример оператора **GoTo**. И лучше пока забыть о существовании этого оператора.

## Использование MsgBox для обеспечения возможности выбора

До сих пор мы использовали оператор **MsgBox** с целью отображения сообщений для пользователя в диалоговых окнах с заголовками. Как уже отмечалось в главе 4, при помощи необязательного аргумента *Buttons* можно использовать VBA-процедуру **MsgBox** как функцию для получения выбора от пользователя в ответ на сообщения или вопросы, которые отображает ваша процедура. Для многих простых вариантов выбора использование функции **MsgBox** для получения ответа от пользователя является гораздо более легким, чем получение текстового ввода с помощью функции **InputBox** и последующий анализ этого текста для определения того, какой выбор сделал пользователь.

При включении аргумента *Buttons* с необходимыми круглыми скобками оператор **MsgBox** работает подобно функции и отображает окно сообщения, содержащее различные командные кнопки. **MsgBox** возвращает численный результат, указывающий, какую командную кнопку выбрал пользователь. Число и тип командных кнопок, отображаемых диалоговым окном **MsgBox**, задается с помощью аргумента *Buttons*.

В листинге 5.9 показана простая процедура, которая демонстрирует использование аргумента *Buttons*.

**Листинг 5.9.** Использование **MsgBox** и аргумента **Buttons** для получения пользовательского ввода

---

```
1: Sub Demo MsgBoxFunction()  
2: 'Процедура демонстрирует MsgBox, используемую как функция  
3:  
4:   Const mTitle = "Демонстрация кнопок MsgBox"  
5:   Dim Resp As Integer  
6:  
7:   Resp = MsgBox(prompt:="Выберите кнопку", _  
8:                 Title:=mTitle, _  
9:                 Buttons:=vbYesNoCancel + vbQuestion)  
10:  Select Case Resp  
11:    Case Is = vbYes  
12:      MsgBox prompt:="Вы выбрали кнопку 'Да'", _  
13:            Title:=mTitle, _  
14:            Buttons:=vbInformation  
15:    Case Is = vbNo  
16:      MsgBox prompt:="Вы выбрали кнопку 'Нет'", _  
17:            Title:=mTitle, _  
18:            Buttons:=vbInformation  
19:    Case Is = vbCancel  
20:      MsgBox prompt:="Вы выбрали кнопку 'Отмена'", _  
21:            Title:=mTitle, _  
22:            Buttons:=vbCritical  
23:  End Select  
24: End Sub
```

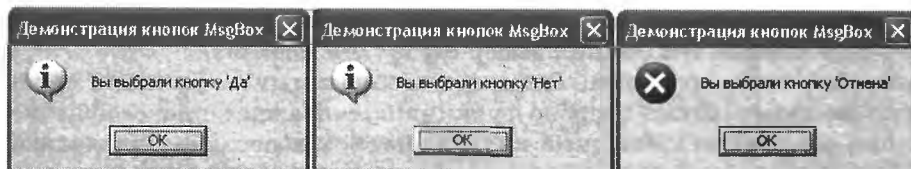
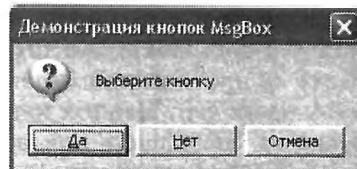
---



Процедура **Demo\_MsgBoxFunction** демонстрирует использование оператора **MsgBox** как функции, результаты использования различных аргументов *Buttons* и оценку значения, которое возвращает функция **MsgBox**. Диалоговое окно, в котором пользователю необходимо сделать выбор, представлено на рис. 5.3. Результаты работы кода этой процедуры при всех трех (по отдельности, конечно) выборах представлены на рис. 5.4.

**Рис. 5.3**

Окно кода листинга 5.9 предоставляет пользователю выбор из трех кнопок



**Рис. 5.4.** Такие сообщения можно получить при выборе различных кнопок в диалоговом окне функции **MsgBox**

Аргумент *Buttons* для **MsgBox** позволяет задавать количество и тип кнопок и наличие или отсутствие в окне сообщения одного из значков Windows для обозначения *предупредительного сообщения* (*Warning message*), *сообщения запроса* (*Query message*), *информационного сообщения* (*Information message*) и *критического предупредительного сообщения* (*Critical Warning message*). Можно также использовать аргумент *Buttons* для определения того, какая из отображаемых кнопок (кнопка 1, 2, 3 или 4) является кнопкой по умолчанию в окне сообщения. Каждый раз можно задавать только один тип кнопки, один значок и одну кнопку по умолчанию.

Чаще используйте в разрабатываемых вами интерфейсных окнах кнопки по умолчанию. Это экономит пользователям таких программ много времени, так как всегда можно для подачи команды приложению вместо мыши использовать клавиатуру. Понятно, что «ударить» по кнопке **Enter** на клавиатуре быстрее и легче, чем переместить мышь (не всегда в хорошем состоянии) в определенное место на экране и щелкнуть левой клавишей.

В строке 9 (часть оператора вызова функции **MsgBox**, начинающегося в строке 7) константа **vbYesNoCancel** определяет, что диалоговое окно **MsgBox** должно содержать три командные кнопки: кнопку **Yes** (Да), кнопку **No** (Нет) и кнопку **Cancel** (Отмена). Константа **vbQuestion** определяет, что окно сообщения должно содержать значок сообщения запроса (*Query message*) Windows.

Как только пользователь выбирает командную кнопку в окне сообщения, VBA возвращает численное значение, соответствующее выбору пользователя. В строке 7 результат функции **MsgBox** присваивается переменной **Resp**. VBA использует различные значения в зависимости от того, какую командную кнопку выбрал пользователь: одно значение для обозначения кнопки **Yes**, дру-

гое — для обозначения кнопки **No** и еще одно — для кнопки **Cancel**. Функция **MsgBox** может также отображать окна с кнопками **Abort** (Стоп), **Retry** (Повтор) и **Ignore** (Пропустить) в различных комбинациях. Поскольку каждая кнопка имеет свое определенное возвращаемое значение, VBA имеет несколько внутренних констант для представления возможных возвращаемых значений функции **MsgBox**. Полный список значений констант VBA, которые может возвращать **MsgBox**, приведен в главе 4.

В строке 10 начинается оператор **Select Case**, который оценивает значение, возвращаемое функцией **MsgBox** в строке 7 и сохраняемое в переменной **Resp**. Тестовое выражение для оператора **Select Case** — это сама переменная **Resp**, поэтому VBA сравнивает значение в переменной **Resp**, чтобы проверить, совпадает ли оно с каким-либо условием **Case** в операторе **Select Case**.

Помните, что можно увидеть все доступные внутренние константы, имеющие отношение к **MsgBox**, в окне **Object Browser** (Просмотр объектов). Для просмотра внутренних констант аргумента **Buttons** выберите **VBA** в списке **Project/Library** (Проект | Библиотека) в окне **Object Browser**, а затем выберите **VbMsgBoxStyle** в списке **Classes** (Классы); список **Members of 'VbMsgBoxStyle'** (Компонент) отображает внутренние константы для значений аргумента **Buttons**. Если вы выберете **VbMsgBoxResult** в списке **Classes**, список **Members of 'VbMsgBoxResult'** отображает внутренние константы для возвращаемого значения функции **MsgBox**.

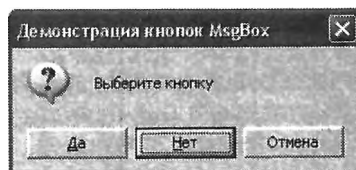
Остается один элемент, который можно задать с помощью аргумента **Buttons** для **MsgBox** и который не был включен в листинг 5.9: можно указать, является ли первая, вторая, третья или четвертая командная кнопка в окнах **MsgBox** кнопкой по умолчанию.

Как вы видели до сих пор, **MsgBox** обычно делает первую командную кнопку в своем окне кнопкой по умолчанию. Посмотрите на рис. 5.3 и обратите внимание, что кнопка **Да (Yes)** помечена как кнопка по умолчанию. Если после того, как VBA отобразит это диалоговое окно, пользователь нажимает на клавишу **Enter**, VBA действует так, как если бы пользователь выбрал кнопку **Да**.

Вам может понадобиться, чтобы кнопкой по умолчанию была какая-либо другая кнопка, а не первая в списке в окне сообщения. Например, если вы используете **MsgBox** для запроса от пользователя подтверждения удалить рабочий лист в рабочей книге, может быть предпочтительнее, чтобы командной кнопкой по умолчанию была кнопка **No**, а не кнопка **Yes**; поскольку удаленный рабочий лист не может быть восстановлен, имеет смысл помочь пользователю избежать нечаянного подтверждения удаления при простом нажатии на клавишу **Enter**.

**Рис. 5.5**

Сравните это диалоговое окно с окном, показанным на рис. 5.3. Заметьте, что кнопка **No** является теперь командной кнопкой по умолчанию как результат добавления **vbDefaultButton2** к аргументу **Buttons**



Чтобы изменить командную кнопку по умолчанию, добавьте одну из внутренних констант VBA — **vbDefaultButton1**, **vbDefaultButton2**,

**vbDefaultButton3** или **vbDefaultButton4** — к аргументу *Buttons*. Следующий фрагмент процедуры показывает оператор **MsgBox** из строк 7–9 листинга 5.9, модифицированный так, чтобы командной кнопкой по умолчанию в окне сообщения была кнопка **No**:

```
Resp = MsgBox(prompt:="Выберите кнопку", _  
              Title:=mTitle, _  
              Buttons:=vbYesNoCancel + vbQuestion + _  
              VbDefaultButton2)
```

При выполнении этого оператора VBA отображает диалоговое окно, показанное на рис. 5.5. Это окно идентично окну, показанному ранее на рис. 5.3, за исключением того, что добавление значения **vbDefaultButton2** к аргументу *Buttons* делает вторую кнопку (в данном случае — кнопку **No**) кнопкой по умолчанию.

## Дополнительные свойства процедур и функций

Теперь, когда вы знаете уже достаточно много о процедурах и функциях, а также об управлении выполнением кода на VBA, рассмотрим некоторые дополнительные свойства процедур и функций, использование которых открывает новые возможности для написания хорошего кода.

### Раннее окончание процедур, функций и целых программ

Существуют обстоятельства, такие как отмена окна ввода пользователем или отсутствие некоторых ожидаемых данных, при которых нет смысла продолжать выполнение процедуры или функции. Если у вас имеется программа, содержащая процедуры, которые вызывают другие функции и процедуры, вы можете задать условия, при которых вся программа должна прекращать выполнение: например, отсутствует рабочий лист, диапазон, документ и так далее.

VBA имеет операторы **Exit** и **End**, которые дают возможность либо завершать процедуру или функцию, либо останавливать всю программу.

### Использование оператора **Exit**

Для того чтобы процедура или функция прекратила выполнение, используется одна из двух доступных форм VBA-оператора **Exit**, в зависимости от того, необходимо ли завершить функцию или процедуру.

Оператор **Exit** имеет две синтаксические формы:

```
Exit Sub  
Exit Function
```

**Exit Sub** и **Exit Function** работают одинаково и имеют один и тот же результат; **Exit Sub** используется для окончания процедуры, а **Exit Function** — для окончания функции.

Сначала рассмотрим простой пример — листинг 5.10, который отличается от листинга 5.8 тем, что в нем нет оператора **GoTo**.

**Листинг 5.10.** Пример использования оператора **Exit Sub**

```
1: Sub EvalDiscount5()  
2: 'Определение скидки (в %) в зависимости от  
3: 'количества продаваемого товара  
4:     Dim lStr As String  
5:     Dim IntNum  
6:     IntNum = Application.InputBox( _  
7:         prompt:="Введите количество товара", _  
8:         Title:="Процедура EvalDiscount5", _  
9:         Type:=1)  
10:  
11:     If TypeName(VarWeight) = "Boolean" Then  
12:         MsgBox "Количество не указано"  
13:         Exit Sub  
14:     End If  
15:  
16:     Select Case IntNum  
17:         Case Is > 1000  
18:             lStr = "10"  
19:         Case Is > 500  
20:             lStr = "5 "  
21:         Case Else  
22:             lStr = "0 "  
23:     End Select  
24:  
25:     MsgBox "Количество: " & Str(IntNum) & ". Скидка " & lStr & "%"  
26:  
27: End Sub
```

В строках 11–14 этой процедуры содержится код проверки того, была ли выбрана пользователем в окне ввода кнопка **Cancel**. Если это — так, программа выдает сообщение о том, что количество не указано, и прекращает выполнение кода оператором **Exit Sub**. Для такой небольшой задачи это — самый подходящий код. Видимо, даже сторонники структурного программирования не найдут здесь причин для критики, хотя оператор **Exit Sub** также нельзя отнести к структурным языковым конструкциям.

В листинге 5.11 приведена процедура **MakeSalesRpt\_Chart**, в которой также для раннего окончания процедуры используются операторы **Exit Sub**. Причиной раннего окончания процедуры является отказ пользователя от ввода имени рабочего листа (в Excel). Можно только представить последствия выполнения кода без проверки того, ввел ли пользователь имя листа для дальнейшей его обработки!

**Листинг 5.11.** Использование оператора **Exit Sub**

```
1: Sub MakeSalesRpt_Chart()  
2: 'Запрашивает имя листа, содержащего исходные данные,  
3: 'затем запрашивает диапазон ячеек с данными для диаграммы.  
4: 'Далее запрашивается имя листа для помещения в него круговой  
5: 'диаграммы. Затем процедура создает диаграмму и использует  
6: 'метод ChartWizard для получения круговой диаграммы  
7:  
8:     Const sTitle = "Отчет о продажах"
```

```
9:
10: Dim SrcShtName As String
11: Dim SourceRng As String
12: Dim DestShtName As String
13:
14: 'получить имя исходного листа
15: SrcShtName = InputBox(prompt:= _
16:     "Введите имя листа, " & _
17:     "содержащего данные для диаграммы", _
18:     Title:=sTitle)
19: 'проверить: введено ли имя
20: If Len(Trim(SrcShtName)) = 0 Then
21:     MsgBox "Имя с данными не введено - конец работы"
22:     Exit Sub
23: End If
24: 'выбор листа, на который указал пользователь
25: Sheets(SrcShtName).Select
26:
27: 'получить диапазон (с данными)
28: SourceRng = InputBox(prompt:= _
29:     "Введите диапазон с данными для диаграммы, " & _
30:     "используя R1C1 формат:", _
31:     Title:=sTitle)
32: 'проверка: введен ли диапазон
33: If Len(Trim(SourceRng)) = 0 Then
34:     MsgBox "Диапазон данных не введен - конец работы"
35:     Exit Sub
36: End If
37:
38: 'получить имя листа для помещения в него диаграммы
39: DestShtName = InputBox(prompt:="Введите имя листа " & _
40:     "для диаграммы:", _
41:     Title:=sTitle)
42: 'проверка: не выбрана ли Отмена
43: If Len(Trim(DestShtName)) = 0 Then
44:     MsgBox "Лист для диаграммы не введен " & _
45:     "- конец работы"
46:     Exit Sub
47: End If
48:
49: 'выбор листа и создание диаграммы
50: Sheets(DestShtName).Select
51: ActiveSheet.ChartObjects.Add(96, 37.5, 234, 111).Select
52:
53: 'использование метода ChartWizard для создания диаграммы
54: With Sheets(SrcShtName)
55:     ActiveChart.ChartWizard Source:=.Range(SourceRng), _
56:         Gallery:=xlPie, _
57:         Format:=7, _
58:         PlotBy:=xlColumns, _
59:         CategoryLabels:=1, _
60:         SeriesLabels:=1, _
61:         HasLegend:=1, _
62:         Title:="Отчет о продажах"
63: End With
64: End Sub
```

Процедура **MakeSalesRpt\_Chart** является отредактированной версией записанного рекордером макроса Excel. Сначала был записан макрос для создания фрагмента диаграммы. Затем этот макрос был скопирован в другой модуль и отредактирован с добавлением объявления различных констант и переменных. В процедуру были добавлены операторы VBA для ввода данных от пользователя и оценки вводимых данных. (Заметьте, что процедура работает только в Excel.)

В переменной **SrcShtName** сохраняется (строки 15–18) имя рабочего листа, в котором содержатся данные для графического изображения, в **SourceRng** (строки 28–31) — диапазон данных для диаграммы, в **DestShtName** (строки 39–41) — имя рабочего листа для помещения в него диаграммы.

После каждого ввода данных результат операции ввода проверяется. Если данные не введены, работа процедуры заканчивается оператором **Exit Sub**.

Оператор **Exit Sub** приводит к тому, что VBA немедленно прекращает выполнение кода процедуры. После выполнения операторов **Exit Sub** VBA прекращает выполнение текущей процедуры и возвращается к выполнению той процедуры или функции, которая вызвала процедуру, содержащую оператор **Exit Sub**.

### Совет

---

Используйте оператор **Exit Sub** или **Exit Function** вместо **GoTo** для раннего окончания процедуры или функции. Оператор **Exit Sub** более очевидно показывает, что процедура закончит выполнение. И, следовательно, ее легче писать, читать и понимать.

---

## Использование оператора End

Вы уже знакомы с ключевыми словами **End Sub** и **End Function**, используемыми для сообщения VBA о достижении конца процедуры или функции. Эти ключевые слова-инструкции указывают VBA прекратить выполнение кода текущей процедуры или функции и возобновить выполнение кода процедуры или функции, которая вызвала текущую процедуру или функцию; если текущая процедура не была вызвана другой процедурой, VBA прекращает выполнение всех операторов.

Одним из хороших принципов программирования является принцип модульности. Процедура не должна содержать бесконечно длинный код, части которого выполняют совершенно не связанные между собой задачи. Следует помещать в каждую процедуру небольшой и функционально законченный код, который легко читать и сопровождать. Например, хорошо понимается код со следующей структурой:

```
Sub ГлавнаяПроцедура ()  
  
    'Вызов процедуры Процедура1  
    Call Процедура1  
  
    'Вызов процедуры Процедура2  
    Call Процедура2  
  
    'Вызов процедуры Процедура3  
    Call Процедура3
```

```
End Sub
```

```
Sub Процедура1()  
    'Код процедуры
```

```
End Sub
```

```
Sub Процедура2()  
    'Код процедуры
```

```
End Sub
```

```
Sub Процедура3()  
    'Код процедуры
```

```
End Sub
```

**Можно также встретить и следующий код:**

```
Sub ГлавнаяПроцедура()
```

```
    'Вызов процедуры Процедура1  
    Call Процедура1
```

```
End Sub
```

```
Sub Процедура1()  
    'Вызов процедуры Процедура2  
    Call Процедура2
```

```
End Sub
```

```
Sub Процедура2()  
    'Вызов процедуры Процедура3  
    Call Процедура3
```

```
End Sub
```

```
Sub Процедура3()  
    'Код процедуры
```

```
End Sub
```

Обратите внимание на то, что в этих фрагментах кода процедуры не возвращают никаких результатов своей работы. Это может быть только в случае, когда эти процедуры по очереди обрабатывают какие-то общие данные, выполняя над ними определенные действия. Результатами этих действий могут быть либо сами преобразованные данные, либо новые (и тоже общие) данные.

Представим такую ситуацию, когда обработка данных одной процедурой зависит от результатов обработки этих данных другой процедурой, выполняемой раньше. Вполне возможно, что по каким-то причинам (пользователь неправильно ввел имя листа с данными, вообще отказался от ввода, неправильно введены числовые и другие данные) одна из процедур не подготовила данные для обработки следующей процедурой. В этом случае весь процесс обработки данных следует прекратить и в дальнейшем никаких процедур не вызывать, т.е. просто закончить работу всей программы. При этом даже необязательно возвращаться в вызывающую процедуру, особенно если предусмотреть перед каждым выходом из программы вывод окна с сообщением о том, почему и где было прервано выполнение кода.

Для полного завершения выполнения программы с помощью VBA используйте ключевое слово **End** одно в отдельной строке:

```
End
```

При выполнении этого оператора VBA прекращает все выполнение операторов процедуры и функции. Любые имеющиеся переменные перестают существовать, и их значения теряются.

В листинге 5.12 показана функция **GetBookName**, которая полностью заканчивает любую программу, выполняемую VBA, если пользователь отменяет окно ввода.

#### Листинг 5.12. Прекращение выполнения программы оператором **End**

```
1: Function GetBookName(ByVal lDflt As String, _
2:     Optional Prmpt, _
3:     Optional lTitle) As String
4:     'включен ли параметр Prmpt в список аргументов?
5:     If IsMissing(Prmpt) Then Prmpt = "Введите имя книги:"
6:
7:     'если lDflt не является пустой строкой, преобразовать ее
8:     'к верхнему регистру, иначе присвоить lDflt имя
9:     If lDflt <> "" Then
10:        lDflt = UCase(lDflt)
11:     Else
12:        lDflt = "NEWFILE.XLS"
13:     End If
14:
15:     'использовать InputBox для получения имени файла
16:     GetBookName = InputBox(prompt:=Prmpt, _
17:        Title:=lTitle, _
18:        Default:=lDflt)
19:
20:
21:     'проверка: не была ли произведена отмена
22:     If Len(Trim(GetBookName)) = 0 Then
23:        MsgBox prompt:="Окончание программы", Title:=lTitle
24:        End
25:     End If
26: End Function
27:
28:
29: Sub TestGetBook_Name()
30:     'Тестирование функции GetBookName
31:
32:     MsgBox GetBookName(lDflt:="", _
33:        lTitle:="Тестирование выхода из программы")
34: End Sub
```

Пишите процедуры или функции, использующие оператор **End**, для отображения некоторого сообщения пользователю о том, что будет происходить и почему, перед оператором, который фактически содержит оператор **End**. В противном случае пользователи вашей процедуры могут не понять, почему процедура или программа, которую они используют, внезапно прекратила работу. Если программа перестанет выполняться вследствие какого-либо действия пользователя, такого как отмена окна ввода, без объяснения, пользователь процедуры может никогда не узнать, почему процедура заканчивается; в действительности, пользователи склонны считать, что это происходит из-за ошибки в процедуре.

Не забывайте закрывать рабочие книги или выполнять другую сервисную работу перед выполнением оператора **End**, чтобы пользователю программы не приходилось, например, доделывать наполовину законченные диаграммы или



иметь дело с рабочими книгами, которые имеют несохраненные данные, когда ваша программа прекращает выполнение.

## Дополнительные свойства необязательных аргументов

Необязательные аргументы функции могут объявляться с типом (иначе, по умолчанию, они имеют тип **Variant**). Тип необязательного аргумента объявляется по тем же причинам, что и тип обязательного аргумента: для того, чтобы правильно использовать функции и обнаруживать операторы, которые передают функции неверные значения.

Тип необязательного аргумента объявляется тем же способом, что и тип обязательного аргумента — с помощью ключевого слова **As**. Следующий фрагмент кода показывает объявление функции с типом необязательного аргумента `nChar` как **Long**:

```
Function UpCase(tStr As String, Optional nChar As Long) As String
```

Всякий раз при объявлении необязательного аргумента функции VBA резервирует место для этого аргумента. Поскольку необязательные аргументы могут присутствовать или не присутствовать во время вызова функции, можно указать VBA присваивать необязательному аргументу значение по умолчанию. VBA использует значение по умолчанию каждый раз, когда функция вызывается без этого включенного конкретного необязательного аргумента. Значение по умолчанию присваивается необязательному аргументу с помощью знака равенства (=) и предоставления константного значения так же, как при объявлении именованной константы.

```
Function GetBookName5(Optional lDflt As String = "Книга1") As String
```

Листинг 5.13 содержит функцию **GetBookName5**, полученную из подобной функции листинга 5.4. Здесь необязательный аргумент объявлен с типом и значением по умолчанию.

### Замечание

Функция **IsMissing** работает только с необязательными аргументами типа **Variant**. Для всех других типов необязательных аргументов VBA инициализирует значение аргумента, как для любой новой переменной (строки имеют нулевую длину и численные типы равны 0), а **IsMissing** всегда возвращает **False**. Чтобы обнаружить пропущенный типизированный необязательный аргумент, тестируйте его на равенство 0 или строке нулевой длины.

### Листинг 5.13. Функция **GetBookName5**

```
1: Function GetBookName5(Optional lDflt As String = "Книга1") As String
2:     Dim lTitle As String, lPrmpt As String, lVar As String
3:
4:     lTitle = "Ввод наименования книги"
5:     lPrmpt = "Введите наименование новой книги"
6:
7:     'использование InputBox для получения имени файла.
8:     lVar = Trim(InputBox(prompt:=lPrmpt, _
9:                           Title:=lTitle, _
10:                          Default:=lDflt))
```

```
11:
12:   'проверить наличие строки ввода
13:   If Len(lVar) = 0 Then
14:       MsgBox "Наименование не введено!"
15:       GetBookName5 = lDflt 'имя по умолчанию
16:   Else
17:       GetBookName5 = lVar
18:   End If
19:
20: End Function           'GetBookName
21:
22: 'Тестирование функции GetBookName5
23: Sub Test_GetBookName5()
24:     MsgBox_GetBookName5("Новая_Книга")
25: End Sub
```

---

Сравните этот код с кодом листинга 5.4. Неправда ли, этот код более изящен?

## Управление передачей аргументов

Теперь, когда вы знаете, как создавать функции-процедуры с типизированными и необязательными аргументами, кажется, что еще можно делать с аргументами, чтобы программы обладали большими возможностями? Оказывается, существуют механизмы, которые VBA использует для фактической передачи (и возвращения) данных-аргументов функции-процедуре, и вы можете управлять этими механизмами.

## Передача аргументов по ссылке и по значению

Существует два способа, которыми VBA передает информацию в функцию-процедуру: *по ссылке (by reference)* и *передача аргументов по значению (by value)*. По умолчанию VBA передает все аргументы по ссылке. При передаче данных функции посредством аргумента по ссылке VBA, на самом деле, только передает адрес памяти, который ссылается на исходные данные, определенные в списке аргументов функции во время вызова функции. Это означает, что если функция изменяет значения в любом из аргументов, исходные данные, переданные функции посредством этого аргумента, также изменяются.

При передаче аргумента по значению VBA делает копию исходных данных и передает эту копию функции. Если функция изменяет значение в аргументе, передаваемом по значению, изменяется только копия данных; исходные данные не изменяются. Передача по ссылке позволяет с помощью функции изменять исходные данные, передаваемые функции посредством аргумента; передача же по значению не позволяет изменять значение исходных данных.

Поскольку передача по ссылке позволяет функции изменять значение исходных данных ее аргументов, аргументы, передаваемые по ссылке, могут получить нежелательные побочные значения. Листинг 5.14 содержит код функции **Sconcat**, которая объединяет две строки, передаваемые ей как параметры по ссылке. Перед объединением строк из них удаляются начальные и хвостовые пробелы. При этом операторы удаления пробелов в качестве промежуточ-

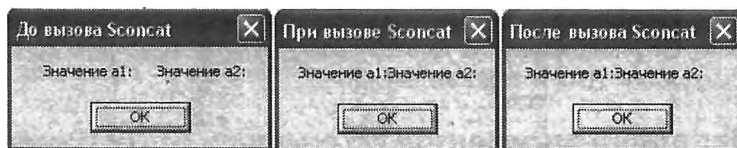
ных переменных используют сами аргументы, таким образом меняя их значения. Результат работы программы тестирования функции **Sconcat** приведен на рис. 5.6.

#### Листинг 5.14. Побочные результаты функции **Sconcat**

```
1: Function Sconcat(arg1 As String, arg2 As String) As String
2:   arg1 = Trim(arg1)
3:   arg2 = Trim(arg2)
4:   Sconcat = arg1 & arg2
5: End Function
6:
7: Sub Test_Sconcat()
8:   Dim a1 As String, a2 As String
9:   a1 = "   Значение a1:   "
10:  a2 = "   Значение a2:   "
11:  MsgBox a1 & a2, , "До вызова Sconcat"
12:  MsgBox Sconcat(a1, a2), , "При вызове Sconcat"
13:  MsgBox a1 & a2, , "После вызова Sconcat"
14: End Sub
```

Из рис. 5.6 видно, что после вызова функции **Sconcat** значения переменных **a1** и **a2** изменились.

**Рис. 5.6.**  
Результаты  
тестирования  
функции **Sconcat** из  
листинга 5.14



### Определение способа передачи аргумента

Понятно, что побочные результаты, вызванные изменением функцией значений в аргументах, передаваемых по ссылке, являются необязательными и в большинстве случаев — нежелательными. Они могут привести к серьезным проблемам в программах. Часто бывает трудно отследить проблемы, вызванные побочными результатами, потому что редко причина какого-либо изменения значения переменной является очевидной. (*Побочный результат [side effect]* — это общий термин для обозначения любого изменения в значении переменной в среде выполняющейся программы, которое не запрограммировано явно.)

Для предупреждения таких побочных результатов необходимо, чтобы функция работала с копией значения аргумента, а не с исходными данными. Можно было бы, конечно, объявить локальную переменную в функции-процедуре и копировать аргумент в эту локальную переменную. Копирование значений аргумента в локальные переменные функции обычно нежелательно. Дополнительные переменные увеличивают объем памяти, необходимой для функции, и количество выполняемой программистом работы. Дополнительные переменные могут также привести в беспорядок программный код, усложняя его до ошибочного.

Более легкий и предпочтительный способ обеспечить работу функции только с копиями значения аргумента, а не с исходным значением — это указать VBA, что аргумент должен передаваться по значению, а не по ссылке. (Помните, передача по значению дает функции копию значения аргумента; передача по ссылке открывает функции доступ к исходным данным в аргументе.)

Чтобы явно указать, передает VBA аргумент по значению или по ссылке, используйте ключевое слово **ByVal** или **ByRef** перед аргументом, для которого необходимо задать метод передачи. Как можно понять по именам ключевых слов, при использовании **ByVal** VBA передает аргумент по значению, а при **ByRef** — по ссылке.

Изменим объявление функции **Sconcat2** (листинг 5.15) с целью передачи ей аргументов по значению и протестируем получившийся код.

#### Листинг 5.15. Устранение побочных результатов функции **Sconcat** в функции **Sconcat2**

```
1: Function Sconcat2(ByVal arg1 As String, ByVal arg2 As String) As String
2:   arg1 = Trim(arg1)
3:   arg2 = Trim(arg2)
4:   Sconcat2 = arg1 & arg2
5: End Function
6:
7: Sub Test_Sconcat2()
8:   Dim a1 As String, a2 As String
9:   a1 = "   Значение a1:   "
10:  a2 = "   Значение a2:   "
11:  MsgBox a1 & a2, , "До вызова Sconcat2"
12:  MsgBox Sconcat2(a1, a2), , "При вызове Sconcat2"
13:  MsgBox a1 & a2, , "После вызова Sconcat2"
14: End Sub
```

Обратите внимание на то, что значения переменных **a1** и **a2** не изменились после вызова функции **Sconcat2**.

**Рис. 5.7.**  
Результаты  
тестирования  
функции **Sconcat2**  
из листинга 5.15



Не следует объявлять аргументы по значению (**ByVal**), если только функция-процедура не изменяет этот аргумент, потому что передача аргументов по значению может использовать больше памяти и времени для выполнения, чем при передаче аргументов по ссылке — передача аргументов по значению приводит к тому, что VBA делает копию данных для аргумента.

Объявляйте аргумент с ключевым словом **ByVal** для передачи его по значению, если у вас имеются сомнения, следует ли передавать этот конкретный аргумент по ссылке или по значению: потери в объеме памяти и скорости при передаче по значению не слишком велики. Кроме того, передача аргумента по значению эффективнее и делает код более понятным, чем создание вашей собственной копии значения аргумента в локальной переменной.

## Рекурсия

В конце этой главы, посвященной написанию пользовательских функций-процедур, необходимо объяснить еще одно важное понятие, хотя оно и не является непосредственно частью процесса создания функций-процедур VBA. Более того, этот материал предназначен не для начинающих программистов и может быть пропущен при первом чтении данной книги.

*Рекурсивная (recursive)* функция или процедура — это функция или процедура, которая вызывает сама себя. Почти во всех случаях, рекурсия является ошибкой программирования и приводит к полному сбою программы. Наиболее общим симптомом возникновения этой проблемы является ошибка из-за нехватки памяти или ошибка из-за нехватки памяти в стеке. *Стек (stack)* — это временная рабочая область компьютерной памяти. VBA использует стековую память для сохранения внутренних результатов выражений, копий аргументов функций, передаваемых по значению, результатов функций-процедур и в других случаях, когда необходима временная рабочая область.

### Примеры рекурсивных функций

Самый простой пример, который может продемонстрировать рекурсивную функцию, это — пример функции возведения числа в степень. Алгоритм функции **fPower** основан на том факте, что число, возведенное в степень  $n$ , равно этому же числу, умноженному само на себя в степени  $n-1$ . Например,  $2^3$  равно  $2 \times 2^{3-1}$  или  $2 \times 2^2$ .

Листинг 5.16 содержит код функции **fPower**, возвращающей степень числа. В качестве аргументов функция принимает число и степень, в которую нужно возвести это число.

#### Листинг 5.16. Функция **fPower**: пример рекурсии

```
1: Function fPower(num As Double, pwr As Integer) As Double
2:     'рекурсивное возведение в степень
3:     If pwr = 0 Then
4:         fPower = 1           'окончание рекурсии
5:     Else
6:         fPower = num * fPower(num, pwr - 1)    'рекурсия
7:     End If
8: End Function
9:
10: Sub Test_fPower()
11:     MsgBox fPower(2, 3)
12: End Sub
```

Функция **fPower** имеет два обязательных аргумента: **num** и **pwr**, и возвращает результат типа **Double**. В строке 6 функция вызывает сама себя. Чтобы понять, как работает функция **fPower**, рассмотрим порядок выполнения кода функции, вызванной процедурой **Test\_fPower**.

В этом случае при первом обращении аргумент **num** (и всегда) имеет значение 2, а **pwr** — значение 3. В строке 3 начинается оператор **If...Then**, который определяет момент прекращения рекурсии. Здесь проверяется, равно ли зна-

чение переменной **pwr** нулю. В этом вызове значение **pwr** равно 3, поэтому выполнение функции продолжается в строке 6.

В строке 6 выполняется присваивание функции; выражение, присваиваемое результату функции **fPower**, указывает на то, что результат функции равен значению **num**, умноженному на результат следующего вызова функции **fPower** со вторым аргументом, равным **pwr-1**.

Если подставить литеральные промежуточные значения для этого примера в данное выражение, вызов функции **fPower** в строке 6 в этом месте будет следующим:

```
fPower(2,2)
```

Это — второй вызов функции **fPower**; значение аргумента **pwr** теперь равно 2. В строке 3 проверяется, равно ли нулю значение аргумента **pwr**; если — нет, VBA снова выполняет присваивание функции в строке 6. Выражение в присваивании функции вызывает функцию **fPower** снова, передавая **pwr-1** в качестве второго аргумента.

Если мы опять подставим литеральные промежуточные значения для этого примера в выражение в строке 6, вызовом функции **fPower** в этом месте будет:

```
fPower(2,1)
```

В этом третьем вызове функции **fPower** значение **pwr** теперь равно 1. В строке 3 проверяется, не равно ли нулю значение аргумента **pwr**; если — нет, то присваивание функции в строке 6 выполняется третий раз, приводя в результате к четвертому вызову **fPower**.

В четвертом вызове функции **fPower** значение **pwr** равно 0; результатом проверки (в строке 3) на равенство нулю значения аргумента **pwr** является **True**, и VBA выполняет строку 4, заканчивая рекурсию. Строка 4 — это еще один оператор присваивания функции, на этот раз присваивающий число 1 как результат функции. (Любое число, возведенное в степень 0, равно 1).

В вызовах 1–3 функции **fPower** функция не возвратила никакого результата; VBA не может полностью вычислить выражение в операторе присваивания функции (строка 6) до тех пор, пока серия рекурсивных вызовов не прекращается в четвертом вызове функции **fPower**. Теперь, когда функция **fPower** возвращает результат, рекурсивный процесс прекращается, и каждый отдельный вызов функции **fPower** возвращает ее результат.

Четвертый вызов **fPower** возвращает число 1 в выражение в строке 6 третьего вызова функции **fPower**. Подставим литеральные промежуточные значения из примера, и это выражение будет равно  $2 \cdot 1$ . Результат, возвращаемый третьим вызовом функции **fPower**, равен 2.

VBA возвращает это значение 2 выражению присваивания функции во втором вызове функции так, что (подставляем литеральные промежуточные значения) это выражение равно теперь  $2 \cdot 2$ . Второй вызов функции **fPower** возвращает 4 выражению присваивания функции первого вызова функции **fPower**.

Снова подставляются литеральные значения, и выражение из первого вызова функции **Power** равно теперь  $2 \cdot 4$ . Наконец, исходный первый вызов функции **fPower** возвращает значение 8 — правильный результат возведения 2 в третью степень.

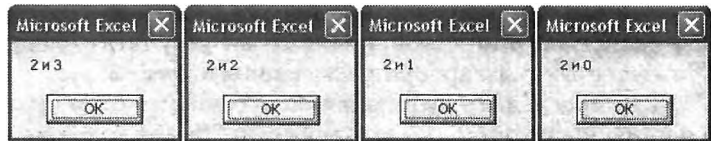
Если вы хотите проследить последовательность передачи аргументов функции **fPower**, вставьте перед строкой 3 следующий оператор:

```
MsgBox num & " и " & pwr
```

При этом в процессе тестирования предыдущего примера вы увидите последовательность окон, отображенную на рис. 5.8.

**Рис. 5.8**

Окна с промежуточными результатами работы функции **fPower**



На самом деле, нет необходимости писать функцию **fPower**: VBA оператор возведения в степень (^) имеет такой же результат. Функция **fPower** была выбрана для иллюстрации рекурсии, потому что она предоставляет самый простой пример работы рекурсии.

Рассмотрим пример функции нахождения наибольшего общего делителя двух целых чисел с помощью классического алгоритма Евклида, который заключается в следующем:

1) имеются два целых числа **a** и **b**; 2) если остаток от деления **a** на **b** равен нулю, то **b** — уже наибольший общий делитель, иначе выполнить: (**a0 = b**), (**a1 = a Mod b**), **a = a0**, **b = a1** и вернуться к началу пункта 2.

#### Листинг 5.17. Функция **GCD**: пример рекурсии

```
1: Function GCD(a As Integer, b As Integer) As Integer
2:   If (a Mod b) = 0 Then
3:     GCD = b
4:   Else
5:     GCD = GCD(b, (a Mod b))
6:   End If
7: End Function
8:
9: Sub Test_GCD()
10:  Dim d As Integer
11:  MsgBox GCD(a:=342, b:=612)
12: End Sub
```

Структура кода этой рекурсивной функции очень похожа на структуру предыдущего примера, поэтому, видимо, нет надобности анализировать ее так же подробно. В результате тестирования функции **GCD** при помощи процедуры **Test\_GCD** будет получен результат **18**, который и является наибольшим общим делителем исходных чисел.

Последний пример может пригодиться менеджеру, которому необходимо за определенный период времени (количество дней) определить начисление по кредиту, если базовая сумма изменяется на величину процентов каждый день. Если, например, процент по кредиту равен 0,05 (5%), а базовая сумма в начале периода равна 100000 рублей, то на за одни сутки базовая сумма изменится на следующее значение:  $100000 \times (0,05/365) = 13,7$ . Для определения изменения кредита за вторые сутки следует использовать следующее выражение:  $(100000 + 13,7) \times (0,05/365)$ . И т.д. Понятно, что это рекурсивный алгоритм и поэтому можно написать рекурсивную функцию **PerForDays** для его реализации (см. листинг 5.18).

**Листинг 5.18. Функция PerForDays: пример рекурсии**

```
1: Function PerForDays(  
2:   IntForDays As Integer, _  
3:   PerForOneDay As Double, _  
4:   SumForCredit As Double) _  
5:   As Double  
6:  
7:   If IntForDays = 0 Then  
8:     PerForDays = SumForCredit  
9:   Else  
10:    PerForDays =  
11:      Round(PerForDays(IntForDays - 1, PerForOneDay, _  
12:        SumForCredit) * (1# + PerForOneDay), 2)  
13:  
14:   End If  
15: End Function
```

В качестве параметров функции листинга 5.18 используются: IntForDays — количество дней, для которых определяется начисление на кредитную сумму; PerForOneDay — процент для одного дня (например, 0,05/365); SumForCredit — начальная сумма кредита.

**Как избежать случайной рекурсии и других проблем**

Теперь, когда вы знаете о том, как работает рекурсивная функция-процедура, вы готовы к рассмотрению недостатков рекурсивных программ.

Одним из наиболее очевидных недостатков рекурсивных функций и процедур является то, что их часто трудно понимать. Другой недостаток заключается в том, что они используют большой объем памяти; каждый раз, когда рекурсивная функция вызывает сама себя, VBA передает все ее аргументы снова и для результата функции резервируется область памяти. Объем памяти, используемый серией вызовов рекурсивной функции, может быть значительным в зависимости от размера аргументов (в байтах) и размера возвращаемого функцией значения (в байтах).

Почти всякую задачу, которую можно выполнить с помощью рекурсивной функции, можно также выполнить с помощью структуры цикла без дополнительных потерь памяти; структуру цикла часто легче понять, чем функцию, выполняющую работу посредством рекурсии. (Структуры цикла описываются в главе 7.)

Нечаянная рекурсия — это довольно распространенная ошибка программирования, особенно для начинающих программистов. Когда вы начинаете разрабатывать свои функции, легко ошибиться при написании оператора присваивания функции и нечаянно создать оператор, который вызывает функцию рекурсивно, вместо присваивания результата функции. Поскольку вы указываете VBA, какое значение должно быть возвращено как результат функции с помощью присваивания функции, легко также забыть, что имя функции не является переменной, и попытаться работать с именем функции как с именем переменной; такие попытки обычно приводят к случайным рекурсивным вызовам.

Если вы случайно создали рекурсивную функцию-процедуру, маловероятно, что рекурсия когда-либо закончится. Рекурсивные функции должны раз-



рабатываться очень тщательно, чтобы обязательно было какое-либо условие, прекращающее рекурсивные вызовы. Если функция не тестирует какое-либо условие, чтобы явно завершать рекурсию (как делает функция **fPower**), вызовы рекурсивной функции продолжаются до тех пор, пока у VBA не заканчивается память (с возникновением ошибки времени исполнения).

Обычно при выполнении функции, которая содержит нечаянный рекурсивный вызов, компьютер как бы прерывает работу на несколько секунд перед тем, как VBA отображает ошибку времени исполнения из-за нехватки памяти.

# 6

## Введение в объекты и коллекции

Эта глава посвящена одной из самых интересных тем в программировании. Материал, представленный здесь, может восприниматься по-разному в зависимости от уровня подготовленности читателя. Общеизвестным считается мнение о том, что объектно-ориентированное программирование легче воспринимается теми, кто не слишком много времени занимался традиционным (модульным) программированием. Поскольку эта книга в основном предназначена для тех, кто вообще никогда не программировал (для этого так подробно и разбираются многочисленные примеры), то имеется надежда, что и этот материал будет понятен и заинтересует читателя. По крайней мере, именно после прочтения этой главы можно начинать «по-настоящему» программировать в VBA, поскольку эта система является объектно-ориентированной.

VBA является идеальным инструментом для изучения основ объектно-ориентированного программирования, так как имеет встроенные объекты (рабочие книги и листы, ячейки, документы и выделенные фрагменты текста), их свойства и методы. Эти объекты — не выдумка, они реальны, их необходимость ощутима. Поэтому для иллюстрации объектов, свойств и методов не нужно обращаться к объектам, не имеющим никакого отношения к программированию, например, автомобилям с их свойствами (цвет, размер, скорость) и методами (изменение скорости, подача сигнала и т.д.). На примере объектов, свойств и методов, с которыми работает VBA, можно создавать свои собственные, полезные вам объекты. Можно к свойствам и методам встроенных объектов VBA добавлять новые, связанные с вашими задачами.

### Объекты

В середине 80-х годов была разработана новая концепция в компьютерном программировании, известная как *объектно-ориентированное программирование* (*object-oriented programming, OOP*). Популярность объектно-ориентированного программирования с годами возрастает; новый *Рабочий стол* (*desktop*) в Windows 95, Windows NT 4.0 и *Active Desktop* в Windows 98/2000/XP, фактически воплощают многие объектно-ориентированные принципы. Центральная идея объектно-ориентированного программирования заключается в том, что программное приложение (как и реальный мир вокруг нас) должно состоять из особых объектов, каждый из которых имеет собственные специфические качества и поведение.

Объектно-ориентированное приложение организует данные и выполняемые операторы программного кода в связанные объекты, что облегчает разработку, организацию и работу со сложными структурами данных и действиями, выполняемыми над этими данными (или с ними). Каждый объект в программном приложении содержит данные и операторы (методы), связанные вместе и образующие единый элемент. Большинство приложений содержат много различных типов объектов. Одни объекты могут состоять из других объектов, например, объект приложения Excel рабочая книга, состоит из объектов листов, которые, в свою очередь, состоят из ячеек или графических объектов.

В Word объектами являются документы, диапазоны текста, таблицы, графические объекты, диалоговые окна и само приложение Word.

То, что вы узнаете из этой главы, поможет вашей работе с VBA не только в Word и Excel. Все host-приложения VBA, такие как Access и Microsoft Project, имеют объекты, доступные для VBA таким же образом, как объекты Word и Excel. Объекты в других host-приложениях VBA существуют по тем же причинам, что и объекты в Word и Excel. Конкретные объекты в host-приложении VBA варьируют в зависимости от приложения. Объекты в Access, например, — все имеют отношение к базам данных и манипулированию базами данных, тогда как объекты в Excel связаны с понятиями рабочих листов, книг и так далее.

## Свойства объекта

Как и объекты реального мира, объекты VBA имеют различные присущие им качества или *свойства* (*properties*). Объекты в Word и Excel имеют свойства, определяющие их вид и поведение: текст в документе Word может отображаться или не отображаться полужирным шрифтом или курсивом, рабочий лист Excel может быть видимым или нет, строки в рабочем листе или таблице имеют свойство высоты, столбцы имеют свойство ширины и так далее.

Свойства регулируют вид и поведение объекта. Чтобы изменить вид и поведение объекта, необходимо изменить его свойства. В Excel можно изменить поведение рабочего листа, изменяя свойство вычисления с автоматического на ручное, или можно изменить вид рабочего листа, задав новый цвет для текста или графики в листе. В Word можно изменить поведение документа, изменяя свойство, которое регулирует показ орфографических ошибок во время набора текста; можно изменить вид документа, задав новые значения для свойств разметки страницы.

Чтобы узнать о текущем виде и поведении объекта, необходимо узнать о его свойствах. Вы можете определить диск, папку и имя рабочей книги Excel или документа Word, рассмотрев свойство **FullName** этой рабочей книги или документа.

Некоторые свойства объекта можно изменять, некоторые — нет. Например, в рабочей книге Excel можно изменить имя автора книги, изменяя свойство **Author**, но нельзя изменить свойство **Name** рабочей книги. (Свойство **Name** рабочей книги содержит имя файла на диске и не может быть изменено без создания нового дискового файла или переименования файла рабочей книги вне Excel).

Некоторые объекты имеют свойства с одними и теми же или подобными именами: объекты **Application**, **Workbook** и **Worksheet** в Excel — все имеют свойство **Name**. Каждый объект рабочего листа сохраняет данные для своих

свойств внутри самого рабочего листа вместе с пользовательскими данными. [*Пользовательские данные (user data)* — это любые данные, сохраняемые объектом, которые поступают непосредственно от пользователя подобно данным, содержащимся в ячейках рабочего листа.]

## Методы объекта

Объекты реального мира почти всегда имеют тот тип присущего им поведения или действия, который они могут выполнить. Объекты VBA также имеют поведение и возможности, называемые *методами (methods)*. Объект рабочей книги Excel, например, имеет встроенную способность добавлять к себе новый рабочий лист — *метод* добавления рабочих листов (называемый **Add**). Другой пример: документ Word имеет встроенную способность проверять орфографию текста в документе — *метод* проверки орфографии (называемый, как вы могли догадаться, **CheckSpelling**).

Методы изменяют значения свойств объектов; методы выполняют также действия с данными (или над данными), сохраняемыми объектом. Методы во многом похожи на процедуры VBA, с которыми вы уже знакомы, но связаны с объектом; к методам объекта можно обратиться, только используя объект. Это может казаться сложным, но на деле все проще. Для вызова метода объекта (функции обработки данных объекта) указывается не только наименование метода, но и объект (имя), которому принадлежит метод.

Одной из причин того, что объектно-ориентированное программирование стало популярной техникой проектирования, является то, что оно позволяет разработчикам программного обеспечения создавать более эффективные программы с совместно используемым кодом. Вместо сохранения отдельной копии кода для каждого метода каждого объекта VBA-объекты одного и того же типа совместно используют код своих методов.

Несмотря на то, что объекты одного и того же типа совместно используют код для своих методов, метод считается частью объекта; когда вы вызываете определенный метод для конкретного объекта, метод воздействует только на объект, посредством которого вы обращаетесь к этому методу.

После того как объект создан (если это — не встроенный объект), нет необходимости беспокоиться о том, как работают его методы и как объект сохраняет или манипулирует данными. Все, что нужно знать, — как указать объект и как указать метод, который необходимо использовать (или определенное свойство, которое нужно выбрать и изменить). Встроенный код для объекта VBA выполняет остальную работу автоматически.

## Классы объекта

Программные объекты группируются в иерархические *классы* или коллекции объектов, как объекты реального мира вокруг нас. Все объекты в одном и том же классе имеют одни и те же (или подобные) свойства и методы. Каждый класс объектов может содержать один или несколько подклассов. Объекты, принадлежащие к определенному классу объектов, называются *членами (members)* этого класса.

MS Word, Excel, Access, PowerPoint, Outlook и т.д. принадлежат к общему классу объектов — *host-приложений VBA*. При запуске Word на вашем компьютере вы создаете экземпляр Word. При наличии достаточных ресурсов памя-

ти в компьютере вы можете запустить Word более одного раза; каждый раз при запуске Word будет создаваться еще один экземпляр Word. Каждый раз при создании или открытии документа в Word создается экземпляр объекта **Document**. Аналогично, каждая копия Excel, выполняемая на компьютере, является экземпляром приложения Excel, а каждая рабочая книга, открытая в данный момент, является экземпляром объекта **Workbook**.

VBA дает возможность пользователю создавать собственные классы объектов добавлением модуля класса в проект. Ваш модуль класса должен содержать все определения свойств и методов для объекта. После описания нового класса вы можете создавать его экземпляры (объекты этого типа).

## Использование объектов

Операторы программ VBA, использующие объекты, обычно выполняют одно или несколько из следующих действий:

- определяют текущее состояние или статус объекта путем выборки значения, сохраняемого в определенном свойстве;
- изменяют состояние или статус объекта установкой значения, сохраненного в определенном свойстве;
- используют один из методов объекта, обеспечивая выполнение объектом одной из его встроенных задач.

Например, можно определить имя активного в данный момент рабочего листа в Excel, выполняя выборку строки, сохраняемой в свойстве **Name** рабочего листа. (Свойство **Name** рабочего листа содержит имя рабочего листа, как показано на ярлычке листа.) Чтобы изменить имя рабочего листа, необходимо присвоить новую строку свойству **Name** этого рабочего листа. Для добавления рабочего листа в рабочую книгу используется метод **Add** рабочей книги.

Прежде чем применять свойства и методы некоторого объекта, их следует сначала определить.

В операторах VBA используйте следующий общий синтаксис для определения свойства или метода объекта:

### Синтаксис

---

*Object.identifier*

*Object* — любая допустимая ссылка на объект. Объектные ссылки создаются заданием переменной для ссылки на объект или использованием методов или свойств объектов, возвращающих объектную ссылку. *Identifier* — любое допустимое имя свойства или метода; VBA отображает сообщение о runtime-ошибке при попытке использовать свойства или методы, которые не являются в действительности частью указанного объекта.

---

Если вы запустите в среде VBA Word процедуру листинга 6.1, то получите нечто подобное тому, что вы видите на рис. 6.1, т.е. в диалоговом окне функции **MsgBox** будет отображено наименование активного документа, хотя сама процедура может принадлежать другому документу (открытому, но не активному).

**Листинг 6.1. Активный документ**

```

1: Sub ActDocum()
2:   MsgBox "Активный документ: " & ActiveDocument.Name
3: End Sub

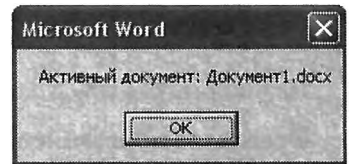
```

Обратите внимание на то, что точка (.) отделяет объектную ссылку от имени свойства или метода. В некотором смысле эта *точка-разделитель* также соединяет объектную ссылку с идентификатором свойства или метода. Поскольку вы обращаетесь к свойству или методу посредством объекта, необходимо определять объектную ссылку и идентификатор свойства или метода вместе. Точка-разделитель указывает VBA, где заканчивается объектная ссылка и где начинается идентификатор свойства или метода. В то же самое время точка-разделитель соединяет объектную ссылку и имя свойства или метода для образования единого идентификатора в операторе VBA.

В табл. 6.1 приведены несколько из наиболее важных объектов (с точки зрения программиста VBA) в Excel. В таблице показано имя объекта и краткое описание этого объекта. (Имейте в виду, что Excel содержит намного больше объектов, чем перечислено в табл. 6.1.)

**Рис. 6.1**

Результат выполнения кода листинга 6.1

**Таблица 6.1. Общие объекты Excel**

Объект	Описание
Application	Само приложение Excel (host-приложение)
Chart	Диаграмма в рабочей книге
Font	Этот объект содержит атрибуты шрифта и стиля для текста, отображаемого в рабочем листе
Name	Заданное имя для диапазона ячеек рабочего листа
Range	Диапазон ячеек (одна или более) или именованный диапазон в рабочем листе
Window	Любое окно в Excel; окна используются для отображения рабочих листов, диаграмм и т.д.
Workbook	Открытая рабочая книга
Worksheet	Рабочая таблица в книге

В табл. 6.2 перечислены наиболее важные объекты в Word. В таблице приводится имя объекта и краткое описание этого объекта. (Имейте в виду, что Word содержит намного больше объектов, чем перечислено в табл. 6.2.)

Таблица 6.2. Общие объекты Word

Объект	Описание
Application	Само приложение Word (host-приложение).
Bookmark	Отдельная закладка в документе.
Document	Открытый документ.
Font	Содержит атрибуты шрифта и стиля для текста, отображаемого в объекте.
Paragraph	Отдельный параграф в документе.
Range	Непрерывная область в документе.
Style	Встроенный или определенный пользователем форматирующий стиль в документе.
Table	Отдельная таблица в документе.
TableOfContents	Отдельная таблица содержания документа.
Template	Шаблон документа.
Window	Любое окно в Windows .

Из таблиц 6.1 и 6.2 видно, что Excel и Windows имеют похожие объекты.

### Использование свойств объектов

Свойства объектов можно использовать только двумя способами: *получать* значение свойства или *устанавливать* его. Как уже было упомянуто в начале этой главы, не все свойства объекта изменяемы. Свойства объектов, которые нельзя изменять, называют *свойствами, доступными только на чтение (read-only)*; свойства, которые можно устанавливать, называют *свойствами, доступными на чтение/запись (read-write)*.

Свойства обычно содержат численные, строковые, значения типа **Boolean**, хотя некоторые свойства могут возвращать значения типа **Object** или другие типы данных.

Обращение к свойству объекта имеет следующий синтаксис:

### Синтаксис

*Object.Property*

*Object* — допустимая объектная ссылка VBA, тогда как *Property* представляет любое допустимое имя свойства для объекта, на который выполняется ссылка.

Свойства используются в выражениях так же, как любое другое значение переменной или константы. Можно присваивать значение свойства переменной, использовать свойства объектов в выражениях как аргументы к функциям и процедурам или как аргументы для методов какого-либо объекта.

Чтобы присвоить некоторой переменной значение свойства объекта, используйте следующий синтаксис:

## Синтаксис

---

*Variable = Object.Property*

*Variable* — любая допустимая переменная, имеющая совместимый со свойством объекта тип; *Object* — любая допустимая ссылка на объект, а *Property* — любое допустимое имя свойства для объекта, на который выполняется ссылка.

---

В следующем примере строка, сохраняемая в свойстве **Name** рабочего листа Excel, на которую ссылается объектная переменная **aSheet**, присваивается переменной **AnyStr**:

```
AnyStr = aSheet.Name
```

Можно также использовать свойство объекта непосредственно в каком-либо выражении или в качестве аргумента функции или процедуры. Следующие строки представляют обоснованное использование свойства объекта (в каждой строке **InstSheet** является объектной переменной, заданной для ссылки на рабочий лист Excel):

```
MsgBox InstSheet.Name  
AnyStr = "Эта ячейка имеет имя: " & InstSheet.Name  
MsgBox LCase(InstSheet.Name)
```

Почти каждый объект в VBA имеет свойство, которое содержит его имя. Следующий оператор использует **MsgBox** для отображения свойства **FullName** в объекте рабочей книги Excel; свойство **FullName** содержит имя диска, путь к папке и имя файла рабочей книги:

```
MsgBox InstBook.FullName
```

В приведенном выше примере **InstBook** — это переменная, заданная для ссылки на объект открытой рабочей книги. Если **InstBook** ссылается на рабочую книгу с именем **Sales.xls** в папке **My Documents**, то окно сообщения, вызываемое приведенным выше оператором, отображает строку "C:\My Documents\SALES.XLS".

Чтобы задать свойство объекта, просто присвойте свойству новое значение, используя следующий синтаксис:

## Синтаксис

---

*Object.Property = Expression*

*Object* — любая допустимая объектная ссылка, *Property* — любое свойство объекта, на который выполняется ссылка, а *Expression* — любое выражение VBA, которое вычисляется до типа, совместимого со свойством.

---

Следующая строка, например, изменяет имя рабочего листа, на который ссылается объектная переменная **InstSheet**, присваивая значение свойству **Name** листа:

```
InstSheet.Name = "Третий квартал"
```



Следующий пример изменяет текст, отображаемый в строке состояния в нижнем левом углу окна приложения, присваивая строку свойству **StatusBar** объекта **Application** (объект **Application** — это host-приложение VBA, в данном случае — Excel):

```
Application.StatusBar = "Генерировать отчет за третий квартал"
```

Используйте свойство **Application.StatusBar** в своих процедурах для отображения сообщений о действиях, которые выполняет процедура, особенно если некоторые из этих действий занимают много времени (например, сортировка длинного списка). Добавляя сообщение в строку состояния, вы даете понять пользователю, что процедура все еще работает. Используйте оператор, подобный следующему:

```
Application.StatusBar = "Сообщение о текущих действиях"
```

Не забывайте возвращать управление строке состояния при работе в Excel. В Excel необходимо устанавливать свойство **Application.StatusBar** в **False**, когда процедура выполнена, иначе Excel продолжает отображать заданное вами сообщение строки состояния. Используйте оператор, подобный следующему, для удаления сообщения из строки состояния и возврата управления строке состояния:

```
Application.StatusBar = False
```

В табл. 6.3 перечислены некоторые из наиболее употребительных или полезных свойств объектов в Excel версии Visual Basic for Application. В таблице представлено имя свойства, тип и значение, а также объекты, которые имеют это свойство.

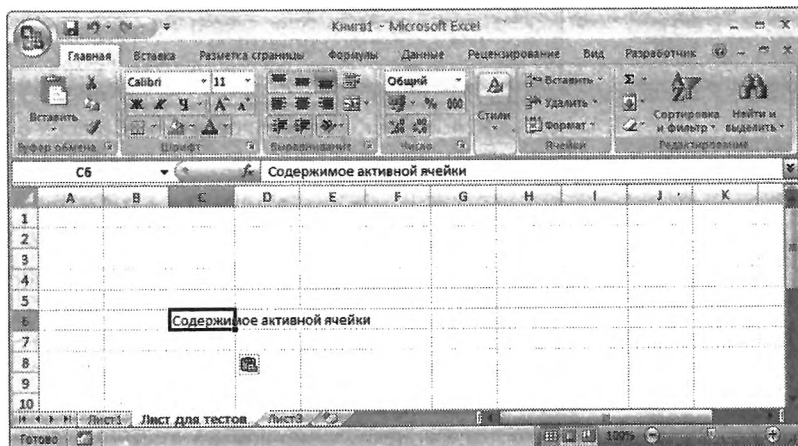
**Таблица 6.3. Наиболее употребительные свойства объектов в Excel**

Свойство	Тип/Что означает	Где найти
ActiveCell	Object: активная ячейка в рабочем листе	Application, Window
ActiveChart	Object: активная диаграмма	Application, Window, Workbook
ActiveSheet	Object: активный лист	Application, Window, Workbook
Address	Возвращает координаты ячейки указанного объекта	Range
Cells	Диапазон объекта Range	Application, Range, Worksheet
Charts	Коллекция диаграмм	Application, Workbook
Count	Integer: число объектов в коллекции	Все объекты коллекции
Formula	String: формула для ячейки рабочего листа	Диапазон
Index	Integer: число объектов в коллекции	Worksheet
Name	String: имя объекта	Application, Workbook и в других объектах

Свойство	Тип/Что означает	Где найти
Path	String: драйвер и каталог, в котором сохранен объект	AddIn, Application, Workbook
Saved	Boolean: сохранялась ли рабочая книга после последних изменений	Workbook
Selection	Object: текущий выделенный фрагмент	Application, Window
Sheets	Коллекция листов рабочей книги	Application, Workbook
StatusBar	String: сообщение в статусной строке	Application
ThisWorkBook	Object: рабочая книга, из которой выполняется текущая процедура	Application
Type	Integer: число, указывающее тип объекта	Window, Worksheet, Chart
Visible	Boolean: отображается или нет объект на экране	Application, Worksheet, Range и в других объектах
Value	(варьируется): действительное значение, отображаемое в ячейке	Range
Workbooks	Коллекция рабочих книг	Application
Worksheets	Коллекция рабочих листов	Application, Workbook

Некоторые свойства из таблицы 6.3 использованы в коде листинга 6.2. Что-бы посмотреть, как работает этот код, запустите Excel, выберите, например, лист с именем **Лист2** и переименуйте его в **Лист для тестов**. Введите в ячейку, например, **С6**, следующий текст **Содержимое активной ячейки**. В этот момент среда Excel готова к запуску кода листинга 6.2 и выглядит, как показано на рис. 6.2. Войдите в Редактор VBA и введите код листинга 6.2.

**Рис. 6.2**  
Среда Excel  
подготовлена для  
запуска кода  
листинга 10



Листинг 6.2. Активный лист, ячейка

```
1: Sub TestObject01()  
2:   'имя активного листа  
3:   MsgBox ActiveSheet.Name  
4:  
5:   'адрес активной ячейки:  
6:   MsgBox ActiveCell.Address  
7:  
8:   'содержимое активной ячейки  
9:   MsgBox ActiveCell.Formula  
10:  
11: End Sub
```

Процедура при помощи функции **MsgBox** по очереди отобразит окна, представленные на рис. 6.3.

**Рис. 6.3**  
Результат выполнения кода  
листинга 6.2



Как видите, даже небольшие знания об объектах дают некоторые дополнительные возможности для работы со средой Excel. Но все еще впереди.

В табл. 6.4 приведены некоторые из наиболее употребительных или полезных свойств объектов в Word. В таблице представлено имя свойства, тип, значение и объекты, которые имеют это свойство.

Таблица 6.4. Употребительные и полезные свойства объектов Word

Свойство	Тип/Значение	Имеется в объектах
ActiveDocument	Object: активный документ	Application
ActivePrinter	String: имя активного принтера	Application
ActiveWindow	Object: активное окно	Application, Document
Count	Long: число объектов в коллекции	Во всех объектах коллекций
Name	String: имя объекта	Application, Bookmark, Dictionary, Document и в других
Path	String: драйвер и путь к папке, в которой сохранен объект	AddIn, Application, Dictionary, Document и в других
Range	Object: часть документа, содержащаяся в указанном объекте	Bookmark, Paragraph, Selection, Table и в других
Saved	Boolean: был ли сохранен текст после последнего изменения	Document, template
Selection	Object: текущий выделенный фрагмент	Application, Pane, Window

Свойство	Тип/Значение	Имеется в объектах
StatusBar	String; сообщение строки состояния	Application
Visible	Boolean: отображается объект на экране или нет	Application, Border

Перед подробным изучением свойств и методов встроенных объектов приведем некоторые примеры использования свойств и методов из таблицы.

Запустите Word. Откройте два документа. В одном из них в Редакторе VBA создайте новый модуль и введите код листинга 6.3.

### Листинг 6.3. Активный документ

```

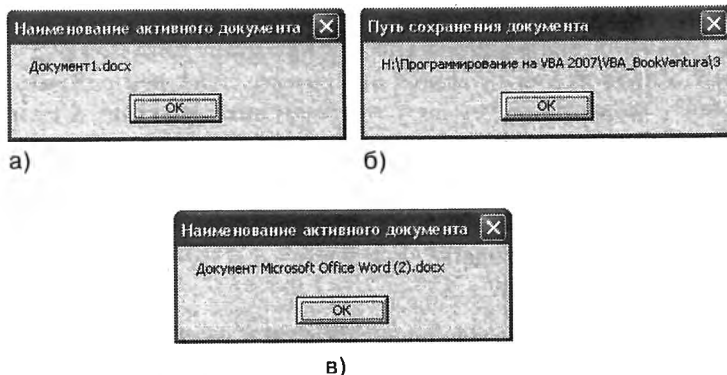
1: Sub ActDocum()
2:
3:     MsgBox ActiveDocument.Name, , _
4:         "Наименование активного документа"
5:
6:     MsgBox ActiveDocument.Path, , _
7:         "Путь сохранения документа"
8:
9:     ActiveDocument.Close    'закрыть активный документ
10:
11:     MsgBox ActiveDocument.Name, , _
12:         "Наименование активного документа"
13: End Sub

```

В строках 3–4 записан оператор, который выдает на экран имя активного документа (рис. 6.4 а) с помощью процедуры **MsgBox**. Следующий оператор (в строках 6–7) выдает окно, в котором указывается путь к месту хранения файла на диске (рис. 6.4 б). Не менее, чем два предыдущих, интересен оператор в строке 9. Этот оператор закрывает активный документ! Заметьте, это делается не при помощи привычного меню, а из кода VBA. Более того, предварительно пользователю (что необязательно и может быть отключено) задается вопрос, не следует ли сохранить последние изменения в документе (рис. 6.4 с).

**Рис. 6.4**

Результат работы кода листинга 6.3



### Использование методов объекта

Методы объекта используются в операторах VBA так, как использовались бы любые встроенные процедуры VBA.

Метод объекта имеет следующий синтаксис:

#### Синтаксис

---

*Object.Method*

Для методов объектов, имеющих обязательные и необязательные аргументы, используйте такой синтаксис:

*Object.Method Argument1, Argument2, Argument3 ...*

В обеих строках синтаксиса *Object* представляет любую допустимую объектную ссылку VBA, а *Method* — имя любого метода, принадлежащего объекту, на который выполняется ссылка. Во второй строке синтаксиса *Argument1, Argument2* и так далее представляют аргументы в списке аргументов метода. Как в случае с аргументами для вызова процедуры VBA, необходимо перечислять аргументы метода в определенном порядке, отделяя каждый аргумент в списке запятой и включая *отмечающие запятые (place holding commas)* для пропущенных необязательных аргументов. Метод может иметь один или несколько аргументов в своем списке или не иметь их совсем; аргументы метода могут быть обязательными или необязательными.

---

Пример: рабочие книги Excel имеют метод **Activate**, который делает рабочую книгу активной и активизирует первый лист в книге. Если задать переменную **InstBook** для ссылки на объект рабочей книги, то приведенный ниже оператор активизирует эту рабочую книгу (в последующих разделах этой главы описывается, как задать переменную для ссылки на объект):

```
InstBook.Activate
```

Хотя метод **Activate** не имеет аргументов, многие методы объектов имеют один или более аргументов. В следующем примере вызывается метод **SaveAs** объекта рабочей книги Excel; используется один обязательный аргумент и один из нескольких необязательных аргументов.

```
ActiveWorkbook.SaveAs Filename:="D\VBA2000\NEWFILE.xls", _  
                        FileFormat:=xlNormal
```

Word-объекты **Document** также имеют методы **Activate** и **SaveAs** с похожими аргументами. Первый оператор, приведенный ниже, активизирует документ, на который ссылается переменная **InstDoc**, а второй — использует метод **SaveAs** документа для сохранения активного документа под другим именем:

```
InstDoc.Activate  
ActiveDocument.SaveAs Filename:="E:\VBA2000\Chap10.doc"
```

Многие объекты имеют методы, которые возвращают значения так же, как это делает функция. Чтобы использовать значение, возвращаемое методом, необходимо поместить список аргументов метода в круглые скобки и включить вызов метода в оператор присваивания или другое выражение, точно так же, как при использовании функции. Можно также игнорировать результат, возвращаемый методом. Чтобы игнорировать результат метода (если он имеет ре-

зультат), вызовите метод без круглых скобок вокруг списка аргументов, как если бы метод не возвращал результата.

Пример: Excel-метод **Address** (который принадлежит объекту **Range**) возвращает адрес диапазона ячеек в рабочем листе как строку. В следующем примере показан оператор VBA, который использует метод **Address (myRange)** — это объектная переменная, которая ссылается на диапазон ячеек в рабочем листе):

```
MsgBox myRange.Address
```

Если переменная **myRange** ссылается на первую ячейку в рабочем листе, то оператор **MsgBox** в приведенной выше строке примера отображает строку **\$A\$1**.

Хотя в этом примере метод **Address** вызывается без каких-либо аргументов, он, на самом деле, имеет несколько необязательных аргументов. Эти необязательные аргументы определяют стиль адреса ячеек рабочего листа, возвращаемого методом, а также, являются ли координаты ячеек абсолютными или относительными. В следующем примере показан вызов метода **Address** с его третьим необязательным аргументом (который определяет стиль возвращаемых координат ячеек):

```
MsgBox myRange.Address(, , xlR1C1)
```

В этом операторе необходимо в список аргументов метода включать отмечающие запятые для пропущенных необязательных аргументов, как для любой другой процедуры или функции. Поскольку аргумент ссылочного стиля является третьим аргументом, ему предшествуют две отмечающие запятые в списке аргументов. **xlR1C1** — это внутренняя константа Excel, указывающая на то, что координаты ячеек рабочего листа используют стиль записи **R1C1**; если объектная переменная **myRange** ссылается на ячейку во второй строке и третьем столбце, то отображается строка **R2C3** (вторая строка, третья колонка).

Методы имеют также именованные аргументы, как и другие процедуры и функции VBA. Можно переписать последний пример, используя именованные аргументы, следующим образом:

```
MsgBox myRange.Address(ReferenceStyle:=xlR1C1)
```

Используйте именованные аргументы, чтобы было легче читать код VBA. В каждом из следующих примеров показан оператор, вызывающий метод **SaveAs** объекта рабочей книги Excel (этот метод не возвращает результат) для сохранения рабочей книги под другим именем (объектная переменная **InstBook** ссылается на рабочую книгу):

```
InstBook.SaveAs "NEWNAME.XLS", xlNormal, , , True
InstBook.SaveAs FileName:="NEWNAME.XLS", _
    FileFormat:=xlNormal, _
    CreateBackup:=True
```

Оба оператора используют только три из шести необязательных аргументов для метода **SaveAs**. Первый оператор использует стандартный список аргументов, второй — именованные аргументы. Намного легче понять назначение и действие второго обращения к методу **SaveAs**, чем первого.

Между методом объекта и любой другой процедурой VBA (встроенной или пользовательской) имеется одно существенное отличие — метод принадлежит определенному объекту и метод можно использовать, обращаясь к нему только

посредством этого объекта. Часто можно распознать вызов методов в коде VBA по тому факту, что метод связан со ссылкой на объект с помощью точки-разделителя, как показано в примерах этого раздела.

В табл. 6.5 приведены некоторые из наиболее употребительных или полезных методов в Excel. В таблице представлено имя, краткое описание метода и объекты, имеющие этот метод.

**Таблица 6.5. Общеупотребительные и полезные методы объектов Excel**

Метод	Назначение	Имеется в объектах
Activate	Активизирует объект	Window, Workbook, Worksheet, Range и в других объектах
Calculate	Выполняет вычисления в открытой рабочей книге, рабочем листе или диапазоне	Application, Range, Worksheet
Clear	Удаляет данные, сохраненные в указанном объекте	Range
Close	Закрывает указанный объект	Window, Workbook, Workbooks
Justify	Выравнивает текст, сохраненный в указанном объекте	Range
Run	Выполняет указанную процедуру или функцию	Application, Range
Save	Сохраняет файл рабочей книги	Application, Workbook
SaveAs	Сохраняет указанный объект в другом файле	Workbook, Worksheet
Select	Выбирает указанный объект	Range, Sheets, Worksheets
SendKeys	Пересылает нажатия клавиши в диалоговые окна в host-приложении	Application
Volatile	Регистрирует функцию как изменяющуюся (см. гл. 6)	Application

Вернитесь к листу с именем **Лист для тестов** в рабочей книге Excel, занесите какую-либо информацию в ячейки листа и выделите какую-нибудь ячейку, например, как показано на рис. 6.5. Выполните код листинга 6.4.

#### **Листинг 6.4. Метод Clear для ячейки и диапазона**

```

1: Sub TestMethods01()
2:     'тестирование метода Clear
3:
4:     'содержимое активной ячейки
5:     'до применения метода Clear
6:     MsgBox ActiveCell.Formula, , "До метода Clear"
7:
8:     'выполнения Clear
9:     ActiveCell.Clear

```

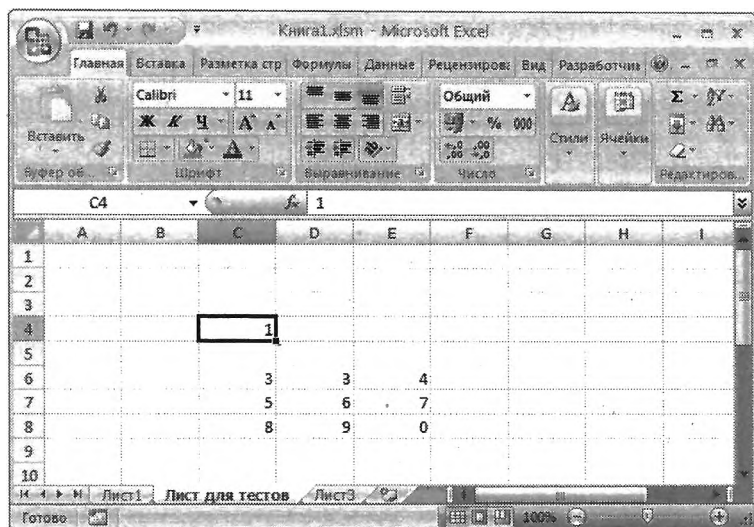
```

10:
11: 'содержимое активной ячейки
12: 'после применения метода Clear
13: MsgBox ActiveCell.Formula, , "После метода Clear"
14:
15: 'Очистить диапазон ячеек:
16: Range("C6:D7").Clear
17:
18: End Sub

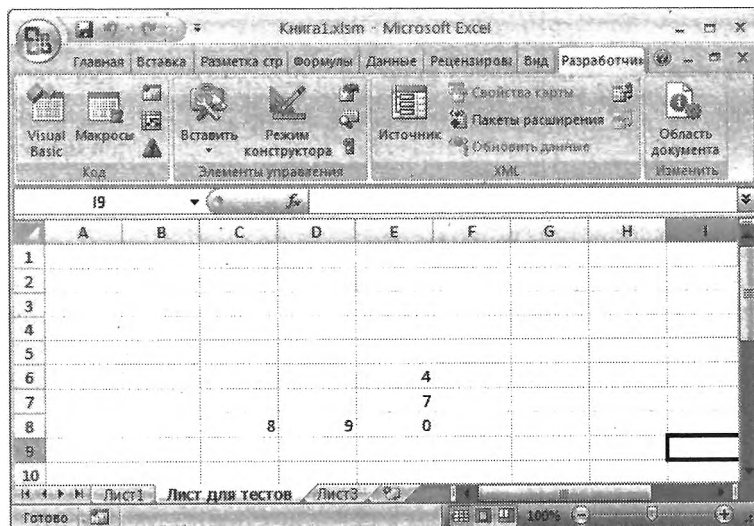
```

**Рис. 6.5**

Содержимое  
листа до выполнения  
кода листинга 6.4

**Рис. 6.6**

Содержимое листа  
после выполнения  
кода листинга 6.4



Сначала в результате выполнения кода этого листинга (строка 6) на экран выдается в диалоговом окне содержимое активной ячейки C4. Метод



**ActiveCell.Clear** очищает эту ячейку, а оператор в строке 13 выдает на экран пустое сообщение. Наконец, оператор в строке 16 очищает диапазон ячеек **C6:D7**. Рабочий лист имеет вид, представленный на рис. 6.6. Обратите внимание на то, как задается диапазон ячеек для выполнения метода **Clear**.

В табл. 6.6 приведены некоторые из наиболее употребительных или полезных методов в Word. В таблице приведено имя метода, краткое описание метода и объекты, имеющие этот метод.

**Таблица 6.6. Наиболее употребительные и полезные методы объектов Word**

Метод	Назначение	Имеется в этих объектах
Activate	Активизирует объект	Window, Document, Pane и в других объектах
CheckSpelling	Проверяет орфографию указанного документа или диапазона	Document, Range
Close	Закрывает указанный объект	Window, Document
Delete	Удаляет указанный объект или удаляет символы и слова	Bookmark, Range и в других объектах
GoTo	Перемещает курсор в начальное положение элемента, такого как страница, закладка или поле; возвращает объект Range	Document, Range, Selection
PrintOut	Печатает указанный объект	Application, Document, Envelope, Window
Run	Выполняет указанную процедуру или функцию	Application
Save	Сохраняет указанный объект	Document, Template
SaveAs	Сохраняет документ в другом файле	Document
Select	Выбирает указанный объект	Bookmark, Document, Field и в других объектах

Рассмотрим подробнее метод **SaveAs**, который уже использовался в примерах этой главы.

Для метода **SaveAs** необходимы дополнительные параметры, так как нет смысла сохранять документ под тем же именем методом, отличным от **Save** (на самом деле, **Save** тоже имеет необязательные параметры). Чтобы получить список параметров метода, можно воспользоваться системой **Auto Quick Info**. Как только вы наберете строку

```
ActiveDocument.SaveAs (
```

система подсказки выдаст на экран список всех параметров метода **SaveAs**, как это обычно делается для обычных функций (рис. 6.7).

Согласно выданному системой **Auto Quick Info** всплывающему окну метод **SaveAs** имеет довольно много аргументов, хотя все они — необязательные. Если интуитивно вам непонятно назначение параметров какой-либо функции

в окне **Auto Quick Info**, обратитесь за помощью к справочной системе. Для этого либо выделите, например, строку **ActiveDocument.SaveAs** и нажмите клавишу **F1**. В окне справочной системы вы получите исчерпывающую информацию о синтаксисе метода **SaveAs**:

## Синтаксис

```
expression.SaveAs(FileName, FileFormat, LockComments, Password,
AddToRecentFiles, WritePassword, ReadOnlyRecommended,
EmbedTrueTypeFonts, SaveNativePictureFormat, SaveFormsData,
SaveAsAOCELetter)
```

Все параметры метода, кроме *expression*, — необязательные и описываются в справочной системе следующим образом:

- *expression* — выражение, возвращающее объект **Document**.
- **FileName** — аргумент типа **Variant**. Является новым именем сохраняемого документа. По умолчанию это — текущая папка и имя файла. Если документ еще ни разу не сохранялся, используется имя по умолчанию (например, Doc1.doc). Если документ с именем **FileName** уже существует, он будет перезаписан без сообщения об этом пользователю.
- **FileFormat** — аргумент типа **Variant**. Формат, в котором сохраняется документ. Может быть одной из следующих **WdSaveFormat**-констант: **wdFormatDocument**, **wdFormatDOSText**, **wdFormatDOSTextLineBreaks**, **wdFormatEncodedText**, **wdFormatHTML**, **wdFormatRTF**, **wdFormatTemplate**, **wdFormatText**, **wdFormatTextLineBreaks** или **wdFormatUnicodeText**.
- **LockComments** — аргумент типа **Variant**. Чтобы блокировать документ для комментариев, должен иметь значение **True**.
- **Password** — аргумент типа **Variant**. Строка пароля для открытия документа.
- **AddToRecentFiles** — аргумент типа **Variant**. **True** — для добавления имени файла к списку недавно используемых документов в меню **File**.
- **WritePassword** — аргумент типа **Variant**. Строка пароля для сохранения изменений в документе.
- **ReadOnlyRecommended** — аргумент типа **Variant**. **True** — для того, чтобы Word устанавливал статус «только на чтение» при открытии документа.
- **EmbedTrueTypeFonts** — аргумент типа **Variant**. **True** — для сохранения TrueType-шрифтов вместе с документом.
- **SaveNativePictureFormat** — аргумент типа **Variant**. Если в документ импортировались графические изображения других платформ (например, Macintosh), **True** — для сохранения только Windows-версии импортированных изображений.
- **SaveFormsData** — аргумент типа **Variant**. **True** — для сохранения данных, введенных пользователем в форму как записи.
- **SaveAsAOCELetter** — аргумент типа **Variant**. Если документ имеет прикрепленный файл электронной почты, **True** — для сохранения документа как AOCE-письма (файл электронной почты сохраняется).
- **Encoding** — (перечислимый тип **MsoEncoding**). Определяет кодовую страницу (или набор символов) для документов, сохраняемых как некодированные текстовые файлы. По умолчанию — системная кодовая страница.

- **InsertLineBreaks** — аргумент типа **Variant**. Если документ сохраняется как текстовый файл, значение **True** используется для вставки в конец каждой строки текста символа конца строки.
- **AllowSubstitutions** — аргумент типа **Variant**. Если документ сохраняется как текстовый файл, значение **True** указывает приложению Word заменять некоторые символы на текст, который выглядит подобным образом. Например, символ, copyright-символ заменяется на (c). Значение по умолчанию — **False**.
- **LineEnding** — аргумент типа **Variant**. Способ, при помощи которого Word отмечает конец строки или параграфа в документе, сохраняемом как текстовый файл. Можно использовать следующие **WdLineEndingType**-константы: **wdCRLF** (значение по умолчанию) или **wdCROnly**.
- **AddBiDiMarks** — аргумент типа **Variant**. Значение **True** добавляет управляющие символы в выходной файл для сохранения двустороннего расположения (bi-directional layout) текста в оригинальном документе.

В справочной системе очень подробно приведено описание каждого метода, и следует почаще прибегать к ее услугам. Впрочем, Редактор VBA постоянно предлагает эту помощь в процессе вашей работы с ним.

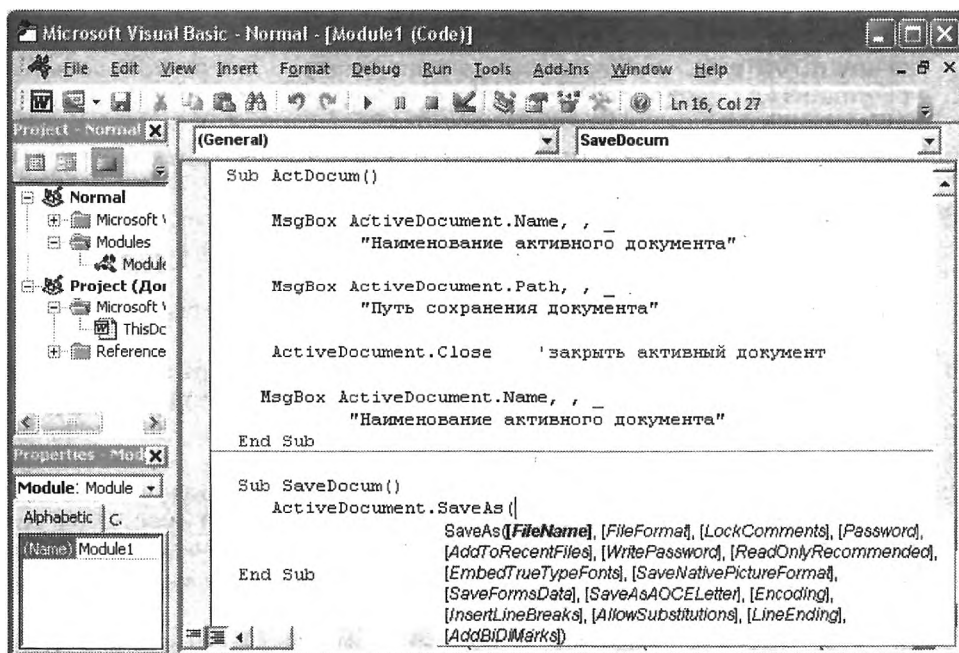


Рис. 6.7. Система Auto Quick Info для методов работает так же, как и для функций

## Объявление объектных переменных

Возможно, вы помните из главы, посвященной типам данных, что в дополнение к типам **Byte**, **Integer**, **Long**, **Single**, **Double** и **String** VBA также имеет тип **Object**. Переменные или выражения типа **Object** ссылаются на

объект VBA или на объект, принадлежащий host-приложению, такой как Excel-объекты **Workbook**, **Worksheet** и **Range** или Word-объекты **Document**, **Range**, **Paragraph**.

Как и в случае с другими типами VBA, можно объявлять переменные в модулях, процедурах и функциях с определенным типом **Object**, что показано в следующем операторе:

```
Dim myObject As Object
```

Можно задавать переменную **myObject**, создаваемую предшествующим оператором **Dim**, чтобы она содержала ссылку на любой объект VBA или объект host-приложения. Если вы собираетесь использовать переменную типа **Object** для некоторых специфических типов объектов, можно также объявлять объектную переменную для этого специфического типа объекта:

```
Dim InstBook As Workbook
```

Объектную переменную **InstBook**, создаваемую этим оператором **Dim**, можно использовать только для сохранения ссылок на объекты **Workbook**; при попытке присвоить переменной **InstBook** ссылку на объект **Range** или **Worksheet** VBA отображает сообщение об ошибке несовпадения типов. Аналогично, следующее предложение объявляет объектную переменную, которая может сохранять только объекты **Document**:

```
Dim InstDoc As Document
```

Можно использовать VBA-функцию **IsObject** для определения того, образует ли переменная или выражение допустимую объектную ссылку. Функция **IsObject** имеет следующий синтаксис:

## Синтаксис

---

```
IsObject (Object)
```

*Object* представляет переменную или выражение, которое необходимо протестировать; **IsObject** возвращает значение **True**, если *Object* является допустимой объектной ссылкой, и **False** — иначе.

---

Для определения объектного типа переменной, можно также использовать VBA-функцию **TypeName** (в предыдущих главах эта функция уже применялась).

## Объекты в выражениях

*Объектное выражение (object expression)* — это любое выражение VBA, которое определяет отдельный объект. Все объектные выражения должны вычисляться до единственной объектной ссылки (ссылки на объект); объектные выражения используются с единственной целью — создание ссылок на специфические объекты в ваших программах VBA.

Объектное выражение может состоять из объектных переменных, объектных ссылок или объектного метода или свойства, которое возвращает объект. Все последующие примеры являются правильными объектными выражениями (использующими объекты Excel):

Application	Имя объекта: ссылается на объект приложения
Application.ActiveSheet	Объектное свойство, которое возвращает ссылку на объект: активный лист
Application.Workbooks	Объектный метод, который возвращает коллекцию объектов: все открытые рабочие книги
ABook	Объектная переменная: инициализированная в операторе <b>Set</b> , ссылается на объект

Нельзя использовать переменные типа **Object** или объектные выражения в арифметических, логических или операциях сравнения. Объектная ссылка, созданная с помощью объектного выражения или сохраненная в объектной переменной, в действительности, является только адресом, указывающим место в памяти компьютера, где сохранен объект, на который выполняется ссылка. Поскольку объектная ссылка — это адрес памяти, арифметические, логические операторы и операторы сравнения не имеют смысла.

Перед использованием объектной переменной для ссылки на объект необходимо задать эту переменную, чтобы она содержала ссылку на нужный объект. Присваивание объектной ссылки объектной переменной отличается от присваиваний других переменных; чтобы присвоить объектную ссылку объектной переменной, используйте оператор **Set**.

Оператор **Set** имеет следующий синтаксис:

### Синтаксис

```
Set Var = Object
```

*Var* — это любая объектная переменная или переменная типа **Variant**. *Object* — любая допустимая объектная ссылка; это может быть другая объектная переменная или объектное выражение. Если *Var* — переменная, объявленная с каким-либо определенным типом (например, **Range** или **Workbook**), этот тип должен быть совместим с объектом, на который ссылается *Object*.

В следующем фрагменте программы типы переменной и объектов используются правильно (для объектов Excel):

```
Dim InstSheet As Worksheet
Set InstSheet = Application.ActiveSheet
```

Следующий фрагмент программы VBA имеет в результате ошибку несовпадения типов потому, что свойство **ActiveSheet** возвращает объект **Worksheet**, а не **Workbook**:

```
Dim InstBook As Workbook
Set InstBook = Application.ActiveSheet
```

Чтобы задать отдельный объект в выражении или объектную переменную для ссылки на этот объект, используйте методы и свойства, возвращающие объекты, такие как свойства **ActiveWorkbook** и **ActiveSheet** объекта **Application** или метод **Cells** объекта **Worksheet** (в Excel). Аналогичные принципы применимы в Word: используйте свойство **ActiveDocument** объекта **Application** для получения ссылки на текущий документ и так далее.

Хотя стандартные операторы сравнения (<, <=, >, >=, <>, =) не являются значимыми при использовании с объектами, VBA предоставляет один оператор сравнения, предназначенный исключительно для использования с объектными выражениями и переменными — оператор (операцию) **Is**.

Оператор **Is** имеет следующий синтаксис:

### Синтаксис

---

*Object1 Is Object2*

*Object1* и *Object2* — любые допустимые объектные ссылки. Используйте оператор **Is** для определения того, обозначают ли две объектные ссылки один и тот же объект. Результат операции сравнения **Is** равен **True**, если объектные ссылки являются одними и теми же, иначе — **False**.

---

В листинге 6.5 показана VBA-процедура для Excel, выполняющая резервную копию активной рабочей книги. Можно использовать подобную процедуру, если необходимо предоставить пользователю легкий способ сохранять копию активной рабочей книги под другим именем без изменения имени файла активной рабочей книги в памяти, как это делается с помощью команды **File | Save As** (Файл | Сохранить как...).

### Листинг 6.5. Процедура создания резервной копии активной рабочей книги

---

```
1: Sub SaveActiveBook()  
2:     'Создает копию активной рабочей книги под новым именем,  
3:     'используя метод SaveCopyAs. Новое имя имеет добавку "_bp"  
4:  
5:     Dim FName As String      'имя файла-копии  
6:     Dim OldComment As String 'комментарии  
7:  
8:     'сохранить комментарии исходного файла  
9:     OldComment = ActiveWorkbook.Comments  
10:  
11:    'добавить новые комментарии к backup-файлу  
12:    ActiveWorkbook.Comments = "Резервная копия файла " & _  
13:        ActiveWorkbook.Name & _  
14:        ", выполненная процедурой SaveActiveBook"  
15:  
16:    'Сформировать новое имя файла из исходного  
17:    FName = Left(ActiveWorkbook.Name, _  
18:        InStr(ActiveWorkbook.Name, ".") - 1) & "_bp.xlsm"  
19:  
20:    'добавление к имени файла пути  
21:    FName = ActiveWorkbook.Path & "\" & FName  
22:  
23:    ActiveWorkbook.SaveCopyAs Filename:=FName  
24:    ActiveWorkbook.Comments = OldComment 'восстановить комментарии  
25: End Sub
```

---

В этой процедуре используются два объекта Excel, их свойства и методы. Объектная ссылка **ActiveWorkbook** является свойством Excel-объекта **Application**, которое возвращает объектную ссылку на активную в данный

момент рабочую книгу. (В следующем разделе вы узнаете, что обычно можно опускать объектную ссылку для свойств и методов объекта **Application**.)

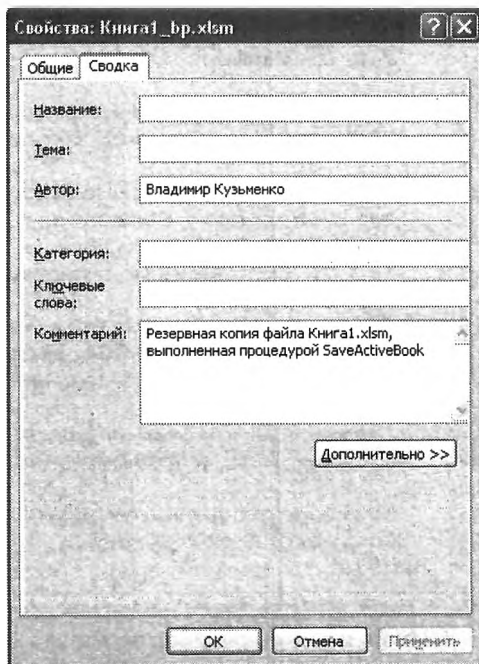
В строке 9 копируется строка свойства **Comments** объекта **ActiveWorkbook** в переменную **OldComment**. Затем в строках 12–14 задается новое значение для свойства **Comments** объекта **ActiveWorkbook** (это свойство изменяется и для исходной рабочей книги). Свойство **Комментарии** этого объекта содержит текст комментария, обычно вводимого вами в диалоговое окно **Свойства**, которое Excel отображает при сохранении рабочей книги в первый раз или при использовании команды **File | Properties** (Файл | Свойства). На рис. 6.8 приведено окно **Свойства** для файла, который был получен в результате резервного копирования. Обратите внимание на поле **Комментарии**. В этом поле должен находиться текст, который можно получить, если перед строкой 16 вставить оператор:

```
MsgBox ActiveWorkBook.Comments
```

В строках 17–19 формируется имя нового файла из имени текущего. Этот оператор использует функции **Left** и **InStr**, помогая создать новое имя файла для резервной копии активной рабочей книги. Функция **InStr** возвращает положение разделителя имени файла (.) в свойстве **Name** объекта **ActiveWorkbook**. (Свойство **Name** рабочей книги содержит имя файла рабочей книги). Результат функции **InStr** определяет, сколько символов функция **Left** копирует из свойства **Name** объекта **ActiveWorkbook**. Поскольку функция **InStr** возвращает позицию разделителя имени файла (.), **Left** возвращает имя файла, не включая разделитель и расширение. К полученному имени добавляется строка "\_bp.xlsm".

### Рис. 6.8

Убедитесь в правильности комментариев при резервном копировании активной рабочей книги



В строке 21 к пути, получаемому при помощи метода **ActiveWorkbook.Path**, добавляется имя нового файла.

Строка 23 использует метод **ActiveWorkbook.SaveCopyAs** для сохранения активной рабочей книги с новым именем файла на текущем диске и каталоге; имя файла активной рабочей книги в памяти остается тем же.

Наконец, в строке 24 восстанавливается оригинальное содержимое свойства **ActiveWorkbook.Comments**; файл активной рабочей книги теперь находится точно в таком же состоянии, в каком он был перед тем, как начинается выполнение этой процедуры.

## Ссылка на объекты с помощью With...End With

Как видно из листинга 6.5, процедуры могут ссылаться на один и тот же объект часто с помощью нескольких операторов в одной строке. Каждый оператор в листинге 6.5 в строках с 9 по 23 использует свойство или метод объекта, на который ссылается **ActiveWorkbook**. VBA предоставляет особую структуру — структуру **With...End With**, позволяющую ссылаться на свойства или методы, которые принадлежат одному и тому же объекту, без задания всей объектной ссылки каждый раз.

Структура **With...End With** имеет следующий синтаксис:

### Синтаксис

---

*With Object*

'операторы, использующие свойства и методы Object

*End With*

*Object* — это любая допустимая объектная ссылка.

---

В листинге 6.6 показана процедура **SaveActiveBook2**, на этот раз — со структурой **With...End With**.

### Листинг 6.6. Добавление к процедуре **SaveActiveBook** структуры **With...End With**

---

```
1: Sub SaveActiveBook2()  
2:   'Создает копию активной рабочей книги под новым именем,  
3:   'используя метод SaveCopyAs. Новое имя имеет добавку "_bp2"  
4:  
5:   Dim FName As String      'имя файла-копии  
6:   Dim OldComment As String  'комментарии  
7:  
8:   With ActiveWorkbook  
9:     'сохранить комментарии исходного файла  
10:    OldComment = .Comments  
11:  
12:    'добавить новые комментарии к backup-файлу  
13:    .Comments = "резервная копия файла " & .Name & _  
14:    " , выполненная процедурой SaveActiveBook2"  
15:  
16:    'Сформировать новое имя файла из исходного  
17:    FName = Left(.Name, InStr(.Name, ".") - 1) & "_bp2.xlsm"  
18:
```



```
19:     FName = .Path & "\" & FName
20:
21:     .SaveCopyAs Filename:=FName
22:     .Comments = OldComment 'восстановить комментарии
23: End With
24: End Sub
```

Эта версия процедуры работает точно так же, как показанная в листинге 6.5. Здесь используется структура **With...End With**. Строка 8 начинается с ключевого слова **With**, за которым следует объектная ссылка **ActiveWorkbook**, начиная таким образом всю структуру **With...End With**.

В строке 10 содержимое свойства **ActiveWorkbook.Comments** присваивается строковой переменной **OldComment**. Заметим, что на этот раз только точка-разделитель (.) включается в начало свойства **Comments**. Поскольку этот оператор находится внутри структуры **With ActiveWorkbook**, VBA «знает», что объектной ссылкой для свойства **.Comments** является **ActiveWorkbook**.

Все сказанное относится ко всем остальным операторам, использующим свойства объекта **ActiveWorkbook**.

Наконец, строка 23 завершает структуру **With...End With** ключевыми словами **End With**.

Не забывайте сохранять любые данные, которые хотите изменить только на некоторое время. Процедура **SaveActiveBook2**, например, изменяет свойство **Comments** активной рабочей книги. **SaveActiveBook2** использует переменную для сохранения оригинального комментария *перед* выполнением изменений, а затем восстанавливает оригинальный комментарий.

Не следует получать данные ввода от пользователя, если они вам действительно не нужны. Процедура **SaveActiveBook2** могла бы быть записана с оператором **InputBox** для получения имени файла вместо генерации этого имени автоматически. Автоматическое генерирование обеспечивает то, что резервные файлы, создаваемые этой процедурой, легко идентифицируются и пользователю не приходится выполнять ввод имени файла. Генерирование нового имени файла в процедуре устраняет также потенциальные проблемы, связанные с вводом неверного имени файла пользователем.

Очевидно, что набор кода листинга 6.6 требует меньших усилий, чем набор кода листинга 6.5; такой код к тому же легче читать и понимать. Упрощайте свой код, используя структуру **With...End With** всякий раз, когда у вас несколько программных операторов используют свойства или методы одной объектной ссылки.

## Работа с коллекциями объектов и контейнерами объектов

*Коллекция (collection)* объектов — это группа связанных объектов, таких как все рабочие листы в рабочей книге или все символы в параграфе. Объект в коллекции называется *элементом (element)* этой коллекции.

Сама коллекция является объектом; коллекции имеют собственные свойства и методы. Каждая коллекция, например, имеет свойство **Count**, которое возвращает число элементов в коллекции. Если в активной рабочей книге имеется 16 рабочих листов, то следующее выражение вычисляется до числа 16:

`Application.ActiveWorkbook.Worksheets.Count`

В этом выражении **Worksheets** — коллекция всех рабочих листов в рабочей книге, **ActiveWorkbook** — свойство Excel-объекта **Application**, возвращающее активную рабочую книгу, а **Count** — свойство коллекции **Worksheets**, возвращающее общее число рабочих листов в коллекции.

Это простое выражение помогает проиллюстрировать то, что одни объекты содержат другие объекты. *Контейнер (container)* — это любой объект, содержащий один или несколько других объектов. В данном примере **Application** содержит объект, на который ссылается **ActiveWorkbook**, содержащий, в свою очередь, коллекцию объектов **Worksheets**. Все контейнерные объектные ссылки соединяются вместе с помощью точки-разделителя (.) для образования одного объектного выражения.

Многие объекты содержат другие объекты различных типов: рабочая книга содержит объекты **Module**, **Chart** и **Worksheets**; рабочий лист может содержать объекты **DrawingObjects**, **ChartObjects** или **OLEObjects**. Иерархия объектов и контейнеров называется *объектной моделью (object model)*.

Объект **Application** и в Excel, и в Word (как, впрочем, и в других приложениях) называется *внешним* контейнером и содержит все другие объекты и коллекции. Например, Excel-объект **Application** содержит коллекции **Workbooks**, **AddIns** и другие; **Workbooks** — это коллекция всех открытых рабочих книг, а **AddIns** — коллекция всех установленных приложений-надстроек (Add-ins). Excel-коллекция **Workbooks**, в свою очередь, содержит коллекции **Worksheets**, **Charts** и т.д. Коллекция **Worksheets** содержит отдельные листы.

Word-объект **Application** содержит коллекции **Documents** — документы, **AddIns** — надстройки, **Dialogs** — диалоги, **Dictionaries** — словари и много других коллекций. Некоторые Word-коллекции **Application** сами содержат коллекции, например, коллекция **Documents** — это коллекция всех открытых документов.

Чтобы указать VBA, на какой объект необходима ссылка, может понадобиться определить контейнер объекта. Для ссылки на определенный лист рабочей книги Excel, например, может быть необходимо включить ссылку на рабочую книгу, которая содержит этот рабочий лист:

```
Workbook("Sales.xlsx").Worksheets("Отчет")
```

В данном объектном выражении используется Excel-коллекция **Workbooks** для ссылки на рабочую книгу **Sales.xls**, а затем используется коллекция **Worksheets**, содержащаяся в объекте рабочей книги **Sales.xls**, для ссылки на конкретный лист. (Как обозначаются отдельные элементы в коллекции, описывается немного дальше в этой главе.) Следующий пример выражения показывает еще более сложную объектную ссылку:

```
Application.Workbooks("Sales.xlsx").Worksheets("Отчет").Range("A1")
```

Здесь объектное выражение ссылается на ячейку **A1** в листе **Отчет** рабочей книги **Sales.xls**. Как видно из этих двух примеров, определение полной объектной ссылки посредством всех контейнеров объекта может быть довольно утомительным. К счастью, VBA позволяет опускать объектную ссылку **Application** почти для всех объектов, которые содержит объект **Application**. Если опускается объектная ссылка **Application**, VBA «полага-

ет», что имеется в виду `host`-приложение, и подставляет ссылку автоматически. Следующее объектное выражение ссылается на тот же объект, что и предыдущее выражение (ячейка **A1** в листе **Отчет** рабочей книги **Sales.xls**):

```
Workbooks("Sales.xlsx").Worksheets("Отчет").Range("A1")
```

Вам необходимо определять Excel-объектную ссылку **Workbooks** только тогда, когда ссылка может быть неоднозначной или при использовании встроенных функций рабочих листов Excel.

Если Excel-объектная ссылка **Workbooks** (или объектная ссылка **Documents** в Word) опускается, VBA обычно (но не всегда) «полагает», что имеется в виду текущая активная рабочая книга или документ. Следующее объектное выражение эквивалентно предыдущему при условии, что рабочая книга **Sales.xls** является активной:

```
Worksheets("Отчет").Range("A1")
```

Если **Sales.xls** — активная рабочая книга, а **Отчет** — активный лист, то следующее объектное выражение также эквивалентно предыдущему:

```
Range("A1")
```

Однако во многих процедурах невозможно определить с уверенностью, что какая-либо отдельная рабочая книга (рабочий лист или документ) является активной во время выполнения процедуры, поэтому, вероятно, потребуется определить, по крайней мере, некоторые из контейнеров для объекта, на который необходима ссылка.

Не забывайте использовать методы и свойства, возвращающие объекты (такие как **ActiveWorkbook** и **ActiveDocument**), чтобы делать контейнерные объектные ссылки короче. Использование структуры **With...End With** является идеальным способом избежать записи длинных объектных ссылок более одного или двух раз в одной процедуре.

Чтобы лучше понять контейнерные объекты и коллекции, необходимо помнить, что программный объект не является физическим предметом, находящимся в другом физическом предмете. Помните, что объект, содержащий другой объект, на самом деле просто имеет адрес памяти другого объекта. Контейнеры ссылаются на объекты, которые они «содержат», посредством их адресов. Точно так, как многие различные люди могут иметь ваш почтовый адрес, но вы не присутствуете физически в их доме, на любой отдельный программный объект могут ссылаться несколько других объектов через его адрес.

При необходимости сослаться на контейнер какого-либо объекта используйте свойство **Parent** этого объекта. Все объекты имеют свойство **Parent**, которое возвращает объектную ссылку на контейнер объекта. Например, следующее объектное выражение ссылается на Excel-объект **Application**, содержащий коллекцию **Workbooks**:

```
Workbooks.Parent
```

В табл. 6.7 перечислены несколько наиболее часто используемых коллекций объектов в Excel версии Visual Basic for Applications. В таблице приводится имя коллекции и краткое описание ее назначения.

В табл. 6.8 перечислены некоторые наиболее часто используемые коллекции объектов в Word версии VBA. В таблице приводится имя коллекции и краткое описание ее назначения.

Таблица 6.7. Общие коллекции Excel

Коллекция	Назначение
AddIns	Коллекция надстроек приложения.
Charts	Коллекция всех таблиц ( <b>Chart sheets</b> ) в рабочей книге.
ChartObjects	Коллекция всех объектов <b>Chart</b> в рабочем листе.
Dialogs	Коллекция всех диалогов в приложении.
Windows	Коллекция всех окон в приложении, независимо от того, отображается ли окно на экране.
Workbooks	Коллекция всех открытых в данный момент рабочих книг в приложении.
Worksheets	Коллекция всех рабочих листов в рабочей книге.

Таблица 6.8. Общие коллекции Word

Коллекция	Назначение
AddIns	Коллекция надстроек приложения.
Bookmarks	Коллекция всех закладок в документе, разделе или диапазоне (в зависимости от контейнера).
Characters	Коллекция всех символов в документе, разделе или диапазоне (в зависимости от контейнера). Каждый элемент в коллекции <b>Characters</b> является объектом <b>Range</b> , содержащим один символ.
Comments	Коллекция всех комментариев документа.
Dialogs	Коллекция диалогов приложения.
Documents	Коллекция всех открытых в данный момент документов.
Paragraphs	Коллекция объектов <b>Paragraph</b> в документе, разделе или диапазоне (в зависимости от контейнера).
Sections	Коллекция разделов в документе, разделе или диапазоне (в зависимости от контейнера).
Styles	Коллекция описаний стилей в документе для встроенных и определенных пользователем стилей.
Templates	Коллекция всех шаблонов приложения.
Windows	Коллекция всех окон в приложении или коллекция всех окон (отображающих документ в зависимости от контейнера).

## Добавление объектов к коллекциям

Большую часть времени вы будете работать с элементами, уже существующими в коллекции. Время от времени будет необходимо добавить какой-либо элемент в коллекцию. В Excel вы, вероятно, захотите создать новую рабочую книгу или добавить новый рабочий лист в рабочую книгу; в Word нужно будет создать новый документ или добавить новую закладку в документ. В этих слу-

чаях необходимо добавлять новый элемент в соответствующую коллекцию объектов.

Каждая коллекция имеет метод **Add** (это один из базовых методов коллекции), который добавляет новый элемент в эту коллекцию. Многие из методов **Add** имеют один или несколько аргументов, которые позволяют задавать различные начальные значения или условия для свойств новых объектов. Следующий оператор показывает метод **Add** коллекции **Workbooks**, используемый для создания новой рабочей книги Excel:

```
Workbooks.Add
```

Этот оператор создает новую рабочую книгу и делает ее активной. Следующий оператор показывает метод **Add** коллекции **Documents**, используемой для создания нового документа Word:

```
Documents.Add
```

Этот оператор создает новый документ и делает его активным.

## Ссылка на конкретные объекты в коллекции или контейнере

Для задания отдельного элемента в коллекции используйте следующий синтаксис:

### Синтаксис.

---

*Collection (Index)*

*Collection* — любое допустимое объектное выражение, ссылающееся на коллекцию. *Index* может быть либо строкой, либо целым, обозначающим конкретный необходимый элемент. Если аргумент *Index* является строкой, строка должна содержать текстовое имя объекта.

---

Например, в Excel можно использовать имя рабочего листа (показано на ярлычке листа), именованного диапазона в рабочем листе, файла рабочей книги, кнопки панели инструментов или команды меню и так далее в качестве *Index* для соответствующей коллекции. Например, для ссылки на рабочий лист с именем **Данные июля** в активной рабочей книге следует использовать такой оператор:

```
Worksheets("Данные июля")
```

В Word можно использовать имя файла документа, закладки, стиля и так далее в качестве аргумента *Index* для коллекции. Например, для ссылки на закладку с именем **Последняя закладка** в документе **Отчет за июль** нужно использовать следующий оператор:

```
Documents("Отчет за июль").Bookmarks("Последняя закладка")
```

Когда *Index* — целое, оно является номером элемента в коллекции. Каждый элемент нумеруется в том порядке, в котором он добавляется в коллекцию. Поэтому не существует легкого способа определения индексного номера любого данного элемента коллекции.

Обычно следует использовать текстовое имя для ссылки на элемент в коллекции; это не только позволяет точно определить, на какой элемент необходима ссылка, но и делает код более читабельным. **Worksheets("Отчет")** гораздо понятнее, чем **Worksheets(5)**.

Для ссылки на все элементы в коллекции не включайте никакого индекса. Следующий оператор закрывает все видимые открытые рабочие книги в Excel:

```
Workbooks.Close
```

Следует помнить о том, что методы и свойства в объекте **Application** принадлежат host-приложению, а не VBA. Когда вы работаете с различными host-приложениями VBA, некоторые из методов и свойств в объекте **Application** могут варьироваться в зависимости от конкретного host-приложения: Excel, Word и т.д.

Не забывайте, что объект **Application** предоставляет много полезных методов и свойств, а также предоставляет контейнер для объектов host-приложения. Например, помните, что Excel-функции рабочего листа, такие как **SUM**, **MIN**, **MAX** и так далее, доступны через объект **Application**. Word-объект **Application** содержит методы для преобразования дюймов в точки, точек в дюймы и другие.

Как уже отмечалось, объект **Application** может содержать методы с именами, дублирующими имена процедур или функций VBA. При использовании метода host-приложения, дублирующего имя процедуры или функции VBA, необходимо задавать объект **Application**, иначе VBA «полагает», что вы хотите использовать процедуру или функцию VBA.

Например, Excel-объект **Application** содержит метод **InputBox**. Excel-метод **InputBox** — это не то же самое, что VBA-функция **InputBox**: Excel-метод **Application.InputBox** имеет на один аргумент больше, чем VBA-метод **InputBox**; этот дополнительный аргумент позволяет ограничивать тип данных (численный, даты, текстовый), вводимый пользователем в диалоговое окно. Чтобы использовать Excel-метод **InputBox**, необходимо задать объект **Application**:

```
Application.InputBox
```

В противном случае VBA «полагает», что вы хотите использовать VBA-функцию **InputBox**.

## Фигуры в слое векторной графики

VBA поддерживает общую модель объектов слоя векторной графики в Microsoft Word, Microsoft Excel и Microsoft PowerPoint. Верхним уровнем этой модели является коллекция **Shapes**, содержащая все графические объекты (Shape-объекты), которые включаются в слой векторной графики. Кроме коллекции **Shapes**, можно в качестве коллекции нескольких Shape-объектов использовать коллекцию **ShapeRange**, объединяющую некоторое подмножество фигур в векторном слое (часть коллекции **Shapes**).

### Добавление к коллекции Shapes «автофигур»

Прежде чем работать с Shape-объектом, необходимо добавить его к (быть может, пока еще пустой) коллекции **Shapes**, используя ее метод **AddShapes**:

## Синтаксис

---

```
expression.AddShape(Type, Left, Top, Width, Height)
```

Здесь *expression* — любое **Shapes**-выражение; *Type* — константа (тип **Long**), определяющая тип фигуры и принимающая одно из **msoAutoShapeType**-значений (например, **msoShapeFlowchartData**, **msoShapeFlowchartMerge**, **msoShapeLeftBracket**, **msoShapeRectangle**, **msoShapeHeart**, **msoShapeRightTriangle**, **msoShapeHexagon**, **msoShapeHorizontalScroll**, **msoShapeWave** — всего около 140 констант); *Left* — положение (тип **Single**) левого угла автофигуры (измеряется в «поинтах»); *Top* — положение (тип **Single**) верхнего угла автофигуры; *Width* — ширина (тип **Single**) автофигуры; *Height* — высота (тип **Single**) автофигуры.

При добавлении в коллекцию **Shapes** фигура получает имя (свойство **Name**) по умолчанию, но его можно указать, используя следующий синтаксис:

```
expression.AddShape(Type, Left, Top, Width, Height).Name=<имя фигуры>
```

---

Для получения ссылки на всю коллекцию объектов в слое векторной графики необходимо использовать следующий синтаксис:

## Синтаксис

---

```
mDocument.Shapes.SelectAll
```

---

В листинге 6.7 приведен код, использующий метод **AddShape** для добавления фигур к коллекции **Shapes** текущего документа Word. На рис. 6.9 добавленные фигуры отображены в текущем документе.

### Листинг 6.7. Добавление фигур к документу

---

```
1: Sub ActDocumShapes()
2:   'добавление фигур к документу
3:
4:   'добавить к коллекции Shapes прямоугольник:
5:   ActiveDocument.Shapes.AddShape(msoShapeRectangle, _
6:     150, 150, 70, 70).Name = "Rectangle"
7:
8:   'добавить к коллекции Shapes правый треугольник:
9:   ActiveDocument.Shapes.AddShape(msoShapeRightTriangle, _
10:    300, 150, 70, 40).Name = "RightTriangle"
11:
12:   'добавить "горизонтально ракручивающийся лист":
13:   ActiveDocument.Shapes.AddShape(msoShapeHorizontalScroll, _
14:    450, 150, 70, 90).Name = "HorizontalScroll"
15:
16:   'выделить все фигуры коллекции:
17:   ActiveDocument.Shapes.SelectAll
18:
19: End Sub
```

---

Чтобы получить коллекцию **ShapeRange**, являющуюся подмножеством коллекции **Shapes**, необходимо использовать следующий синтаксис:

#### Синтаксис

---

`Shapes.Range(index)`

Здесь *index* — имя/индекс фигуры или массив имен/индексов фигур.

---

Чтобы получить коллекцию **ShapeRange**, представляющую все фигуры коллекции **Shapes**, необходимо использовать следующий синтаксис:

#### Синтаксис

---

`Selection.ShapeRange`

---

Чтобы получить коллекцию **ShapeRange**, представляющую все фигуры коллекции **Shapes**, необходимо использовать следующий синтаксис:

#### Синтаксис.

---

`Selection.ShapeRange(index)`

Здесь *index* — имя/индекс фигуры или массив имен/индексов фигур.

---

Например, при помощи кода листинга 6.8 можно «закрасить» фигуры, созданные кодом листинга 6.7. На рис. 6.10 отображены закрашенные фигуры.

#### Листинг 6.8. Использование коллекции **ShapeRange**

---

```

1: Sub ActDocumShapeRange()
2:   'Использование коллекции ShapeRange
3:
4:   ' "закрасить" первые две фигуры в коллекции Shapes:
5:   ActiveDocument.Shapes.Range(Array(1, 2)).Fill.PresetGradient _
6:     msoGradientHorizontal, 1, msoGradientLateSunset
7:
8:   'выделить все фигуры коллекции:
9:   ActiveDocument.Shapes.SelectAll
10:
11:   ' "закрасить" третью фигуру коллекции ShapeRange:
12:   ActiveWindow.Selection.ShapeRange(3).Fill.PresetGradient _
13:     msoGradientVertical, 1, msoGradientLateSunset
14:
15: End Sub

```

---



## Добавление к коллекции **Shapes** специальных фигур

Кроме добавления к коллекции **Shapes** документа (листа, презентации) «автофигуры» (методом **AddShape**), можно также добавлять специальные элементы оформления, используя такие методы, как **AddCallout**, **AddComment** и другие (см. таблицу 6.9).

**Таблица 6.9. Методы для добавления фигур в документы, листы или слайды**

Метод	Синтаксис	Назначение
<i>Word:</i>		
<b>AddCallout</b>	<code>AddCallout (Type As MsoCalloutType, Left As Single, Top As Single, Width As Single, Height As Single, [Anchor]) As Shape</code>	Добавляет в документ не имеющую границ выноску. Возвращает <b>Shape</b> -объект, представляющий эту выноску, и добавляет его в коллекцию <b>Shapes</b> .
<b>AddCanvas</b>	<code>AddCanvas (Left As Single, Top As Single, Width As Single, Height As Single, [Anchor]) As Shape</code>	Добавляет в документ «полотно» для рисования (drawing canvas). Возвращает <b>Shape</b> -объект, представляющий «полотно» для рисования, и добавляет его в коллекцию <b>Shapes</b> .
<b>AddCurve</b>	<code>AddCurve (SafeArrayOfPoints, [Anchor]) As Shape</code>	Возвращает в документ <b>Shape</b> -объект, который представляет кривую Безье.
<b>AddDiagram</b>	<code>AddDiagram (Type As MsoDiagramType, Left As Single, Top As Single, Width As Single, Height As Single, [Anchor]) As Shape</code>	Возвращает в документ <b>Shape</b> -объект, который представляет только что созданную диаграмму.
<b>AddLabel</b>	<code>AddLabel (Orientation As MsoTextOrientation, Left As Single, Top As Single, Width As Single, Height As Single, [Anchor]) As Shape</code>	Добавляет в документ надпись в виде прямоугольника с невидимыми границами и без заливки. Возвращает <b>Shape</b> -объект, который представляет эту надпись, и добавляет его в коллекцию <b>Shapes</b> .
<b>AddLine</b>	<code>AddLine (BeginX As Single, BeginY As Single, EndX As Single, EndY As Single, [Anchor]) As Shape</code>	Добавляет линию в документ. Возвращает <b>Shape</b> -объект, который представляет эту строку, и добавляет его в коллекцию <b>Shapes</b> .

Метод	Синтаксис	Назначение
<b>AddOLEControl</b>	AddOLEControl([ClassType], [Left], [Top], [Width], [Height], [Anchor]) As Shape	Создает элемент управления ActiveX (ранее известный как OLE-элемент управления). (Только в Word; в Excel и PowerPoint используйте AddOLEObject) Возвращает <b>Shape</b> -объект, который представляет новый элемент управления ActiveX.
<b>AddOLEObject</b>	AddOLEObject([ClassType], [FileName], [LinkToFile], [DisplayAsIcon], [IconFileName], [IconIndex], [IconLabel], [Left], [Top], [Width], [Height], [Anchor]) As Shape	Создает OLE-объект. Возвращает <b>Shape</b> -объект, который представляет новый OLE-объект.
<b>AddPicture</b>	AddPicture(FileName As String, [LinkToFile], [SaveWithDocument], [Left], [Top], [Width], [Height], [Anchor]) As Shape	Добавляет картинку в документ. Возвращает <b>Shape</b> -объект, который представляет эту картинку, и добавляет его в коллекцию <b>Shapes</b> .
<b>AddPolyline</b>	AddPolyline(SafeArrayOfPoints, [Anchor]) As Shape	Добавляет ломаную линию или многоугольник в документ. Возвращает <b>Shape</b> -объект, который представляет эту ломаную линию или многоугольник, и добавляет его в коллекцию <b>Shapes</b> .
<b>AddTextbox</b>	AddTextbox(Orientation As MsoTextOrientation, Left As Single, Top As Single, Width As Single, Height As Single, [Anchor]) As Shape	Добавляет в документ прикрепленную текстовую рамку в виде прямоугольника с невидимыми границами и без заливки. Возвращает <b>Shape</b> -объект, который представляет эту текстовую рамку, и добавляет его в коллекцию <b>Shapes</b> .

Метод	Синтаксис	Назначение
<b>AddTextEffect</b>	<code>AddTextEffect(PresetTextEffect As MsoPresetTextEffect, Text As String, FontName As String, FontSize As Single, FontBold As MsoTriState, FontItalic As MsoTriState, Left As Single, Top As Single, [Anchor]) As Shape</code>	Добавляет WordArt-объект в документ. Возвращает <b>Shape</b> -объект, который представляет этот WordArt-объект и добавляет его в коллекцию <b>Shapes</b> .
<i>Excel:</i>		
<b>AddCallout</b>	<code>AddCallout(Type As MsoCalloutType, Left As Single, Top As Single, Width As Single, Height As Single) As Shape</code>	Добавляет в документ не имеющую границ выноску. Возвращает <b>Shape</b> -объект, представляющий эту выноску.
<b>AddConnector</b>	<code>AddConnector(Type As MsoConnectorType, BeginX As Single, BeginY As Single, EndX As Single, EndY As Single) As Shape</code>	Создает соединительную линию. Возвращает <b>Shape</b> -объект, который представляет новую соединительную линию. Когда добавляется соединительная линия, она не соединена ни с чем.
<b>AddCurve</b>	<code>AddCurve(SafeArrayOfPoints) As Shape</code>	Когда применяется к объекту коллекции <b>Shapes</b> , возвращает <b>Shape</b> -объект, который представляет кривую Безье в электронной таблице.
<b>AddDiagram</b>	<code>AddDiagram(Type As MsoDiagramType, Left As Single, Top As Single, Width As Single, Height As Single) As Shape</code>	Создает диаграмму. Возвращает <b>Shape</b> -объект, который представляет новую диаграмму.
<b>AddForm</b>	<code>AddFormControl(Type As XlFormControl, Left As Long, Top As Long, Width As Long, Height As Long) As Shape</code>	Создает элемент управления Microsoft Excel. Возвращает <b>Shape</b> -объект, который представляет новый элемент управления.
<b>AddLabel</b>	<code>AddLabel(Orientation As MsoTextOrientation, Left As Single, Top As Single, Width As Single, Height As Single) As Shape</code>	Добавляет в документ надпись в виде прямоугольника с невидимыми границами и без заливки. Возвращает <b>Shape</b> -объект, который представляет эту новую надпись.

Метод	Синтаксис	Назначение
<b>AddLine</b>	AddLine( <i>BeginX</i> As Single, <i>BeginY</i> As Single, <i>EndX</i> As Single, <i>EndY</i> As Single) As Shape	Когда применяется к объекту коллекции <b>Shapes</b> , возвращает <b>Shape</b> -объект, который представляет новую линию в электронной таблице.
<b>AddOLEObject</b>	AddOLEObject([ <i>ClassType</i> ], [ <i>Filename</i> ], [ <i>Link</i> ], [ <i>DisplayAsIcon</i> ], [ <i>IconFileName</i> ], [ <i>IconIndex</i> ], [ <i>IconLabel</i> ], [ <i>Left</i> ], [ <i>Top</i> ], [ <i>Width</i> ], [ <i>Height</i> ]) As Shape	Создает <b>OLE</b> -объект. Возвращает <b>Shape</b> объект, который представляет этот новый <b>OLE</b> -объект.
<b>AddPicture</b>	AddPicture( <i>Filename</i> As String, <i>LinkToFile</i> As MsoTriState, <i>SaveWithDocument</i> As MsoTriState, <i>Left</i> As Single, <i>Top</i> As Single, <i>Width</i> As Single, <i>Height</i> As Single) As Shape	Создает картинку из существующего файла. Возвращает <b>Shape</b> -объект, который представляет новую картинку.
<b>AddPolyline</b>	AddPolyline( <i>SafeArrayOfPoints</i> ) As Shape	Создает ломаную линию или многоугольник. Возвращает <b>Shape</b> -объект, который представляет новую ломаную линию или многоугольник.
<b>AddTextbox</b>	AddTextbox( <i>Orientation</i> As MsoTextOrientation, <i>Left</i> As Single, <i>Top</i> As Single, <i>Width</i> As Single, <i>Height</i> As Single) As Shape	Создает прикрепленную текстовую рамку в виде прямоугольника с невидимыми границами и без заливки. Возвращает <b>Shape</b> -объект, который представляет новую текстовую рамку.
<b>AddTextEffect</b>	AddTextEffect( <i>PresetTextEffect</i> As MsoPresetTextEffect, <i>Text</i> As String, <i>FontName</i> As String, <i>FontSize</i> As Single, <i>FontBold</i> As MsoTriState, <i>FontItalic</i> As MsoTriState, <i>Left</i> As Single, <i>Top</i> As Single) As Shape	Создает WordArt-объект. Возвращает <b>Shape</b> -объект, который представляет новый WordArt-объект.
<b>PowerPoint:</b>		
<b>AddCallout</b>	AddCallout( <i>Type</i> As MsoCalloutType, <i>Left</i> As Single, <i>Top</i> As Single, <i>Width</i> As Single, <i>Height</i> As Single) As Shape	Создает не имеющую границ выноску. Возвращает <b>Shape</b> -объект, представляющий эту новую выноску

Метод	Синтаксис	Назначение
<b>AddComment</b>	AddComment([Left As Single = 1], [Top As Single = 1], [Width As Single = 145], [Height As Single = 145]) As Shape	Добавляет комментарий. Возвращает <b>Shape</b> -объект, который представляет новый комментарий.
<b>AddConnector</b>	AddConnector(Type As MsoConnectorType, BeginX As Single, BeginY As Single, EndX As Single, EndY As Single) As Shape	Создает соединительную линию. Возвращает <b>Shape</b> -объект, который представляет новую соединительную линию. Когда добавляется соединительная линия, она не соединена ни с чем.
<b>AddCurve</b>	AddCurve(SafeArrayOfPoints) As Shape	Создает кривую Безье. Возвращает <b>Shape</b> -объект, который представляет новую кривую.
<b>AddDiagram</b>	AddDiagram(Type As MsoDiagramType, Left As Single, Top As Single, Width As Single, Height As Single) As Shape	Возвращает <b>Shape</b> -объект, который представляет диаграмму, вставленную в слайд, master-слайд или slide range-слайд.
<b>AddLabel</b>	AddLabel(Orientation As MsoTextOrientation, Left As Single, Top As Single, Width As Single, Height As Single) As Shape	Создает надпись в виде прямоугольника с невидимыми границами и без заливки. Возвращает <b>Shape</b> -объект, который представляет эту новую надпись.
<b>AddLine</b>	AddLine(BeginX As Single, BeginY As Single, EndX As Single, EndY As Single) As Shape	Создает линию. Возвращает <b>Shape</b> -объект, который представляет эту новую линию.
<b>AddMediaObject</b>	AddMediaObject(FileName As String, [Left As Single], [Top As Single], [Width As Single = -1], [Height As Single = -1]) As Shape	Создает media-объект. Возвращает <b>Shape</b> -объект, который представляет новый media-объект.

Метод	Синтаксис	Назначение
<b>AddOLEObject</b>	AddOLEObject([Left As Single], [Top As Single], [Width As Single = -1], [Height As Single = -1], [ClassName As String], [FileName As String], [DisplayAsIcon As MsoTriState], [IconFileName As String], [IconIndex As Long], [IconLabel As String], [Link As MsoTriState]) As Shape	Создает <b>OLE</b> -объект. Возвращает <b>Shape</b> -объект, который представляет новый <b>OLE</b> -объект.
<b>AddPicture</b>	AddPicture(FileName As String, LinkToFile As MsoTriState, SaveWithDocument As MsoTriState, Left As Single, Top As Single, [Width As Single = -1], [Height As Single = -1]) As Shape	Создает картинку из существующего файла. Возвращает <b>Shape</b> -объект, который представляет новую картинку.
<b>AddPlaceholder</b>	AddPlaceholder(Type As PpPlaceholderType, [Left As Single = -1], [Top As Single = -1], [Width As Single = -1], [Height As Single = -1]) As Shape	Восстанавливает ранее удаленное поле для текста или графического объекта на слайде. Возвращает <b>Shape</b> -объект, который представляет это восстановленное поле.
<b>AddPolyline</b>	AddPolyline(SafeArrayOfPoints) As Shape	Создает ломаную линию или многоугольник. Возвращает <b>Shape</b> -объект, который представляет новую ломаную линию или многоугольник.
<b>AddShape</b>	AddShape(Type As MsoAutoShapeType, Left As Single, Top As Single, Width As Single, Height As Single) As Shape	Создает автофигуру (AutoShape). Возвращает <b>Shape</b> -объект, который представляет эту новую автофигуру.
<b>AddTable</b>	AddTable(NumRows As Long, NumColumns As Long, [Left As Single = -1], [Top As Single = -1], [Width As Single = -1], [Height As Single = -1]) As Shape	Добавляет объект-таблицу в слайд.

Метод	Синтаксис	Назначение
<b>AddTextbox</b>	AddTextbox(Orientation As MsoTextOrientation, Left As Single, Top As Single, Width As Single, Height As Single) As Shape	Создает прикрепленную текстовую рамку в виде прямоугольника с невидимыми границами и без заливки. Возвращает <b>Shape</b> -объект, который представляет новую текстовую рамку.
<b>AddTextEffect</b>	AddTextEffect(PresetTextEffect As MsoPresetTextEffect, Text As String, FontName As String, FontSize As Single, FontBold As MsoTriState, FontItalic As MsoTriState, Left As Single, Top As Single) As Shape	Создает WordArt-объект. Возвращает <b>Shape</b> -объект, который представляет новый WordArt-объект.
<b>AddTitle</b>	AddTitle() As Shape	Восстанавливает ранее удаленное заглавие слайда. Возвращает <b>Shape</b> -объект, который представляет восстановленное заглавие.

Фигуры коллекции **Shapes** после их добавления в коллекцию можно редактировать: перемещать, удалять, изменять размеры, изменять внешний вид (как это, например, делается в коде листинга 6.9), добавлять в них текст. Редактирование фигур осуществляется изменением их свойств. Объекты коллекции **Shapes** обладают как общими (**Left**, **Top**, **Height** и **Width**), так и специфическими (зависящими от типа объекта) свойствами. Например, код листинга 6.10 сначала добавляет объект **clock.avi** к первому слайду презентации, а затем (строки 12–15) изменяет его высоту (свойство **Height**) и ширину (свойство **Width**). Результат работы кода листинга 6.10 представлен на рис. 6.11.

#### Листинг 6.9. Изменение свойств фигуры коллекции **Shapes**

```

1: Sub ActDocumShapesRange()
2:   'Использование коллекции ShapesRange
3:
4:   'выделить все фигуры коллекции:
5:   ActiveDocument.Shapes.SelectAll
6:
7:   With ActiveDocument.Shapes(2)
8:     .Height = 200
9:     .Width = 200
10:   End With
11:
12: End Sub

```



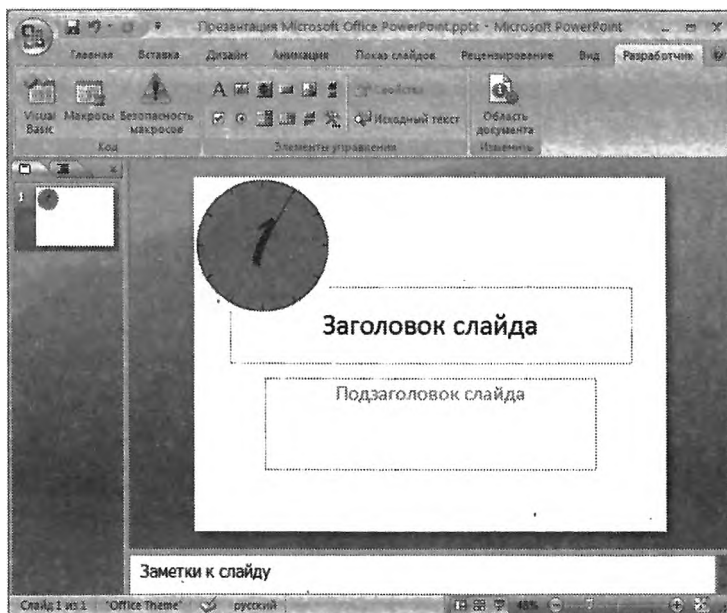
Рис. 6.9. Добавленные к коллекции **Shapes** фигуры отображены в текущем документе



Рис. 6.10. Добавленные к коллекции **Shapes** фигуры закрашены кодом листинга 6.8



**Рис. 6.11**  
Изменение  
размеров объекта  
коллекции **Shapes**  
кодом листинга 6.10



**Листинг 6.10.** Изменение свойств фигуры коллекции **Shapes**

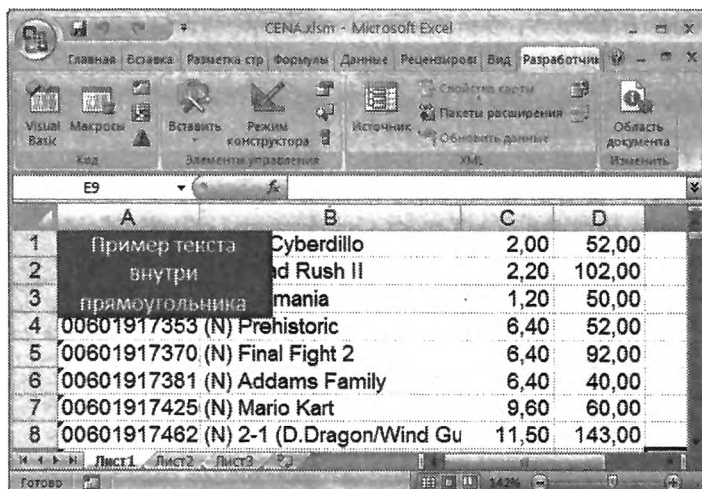
```

1: Sub PropertyChanging()
2: 'добавление объекта к коллекции Shapes
3: 'и изменение его размеров
4:
5:     Set myDocument = ActivePresentation.Slides(1)
6:
7:     'добавить объект к коллекции Shapes:
8:     myDocument.Shapes.AddMediaObject FileName:="g:\windows\clock.avi", _
9:     Left:=5, Top:=5, Width:=100, Height:=100
10:
11:     'изменение размеров объекта:
12:     With myDocument.Shapes(3)
13:         .Height = 200
14:         .Width = 200
15:     End With
16:
17: End Sub

```

Код листинга 6.11 при добавлении к коллекции **Shapes** прямоугольника помещает в него текст. Результат работы кода листинга 6.11 (в Excel) представлен на рис. 6.12.

**Рис. 6.12**  
Ввод текста для  
объекта коллекции  
**Shapes** кодом  
листинга 6.11



**Листинг 6.11.** Изменение свойств фигуры коллекции **Shapes**

```

1: Sub PropertyChanging2()
2: 'добавление прямоугольника к коллекции Shapes
3: 'и занесение в него текста
4:
5:     Set myDocument = Worksheets(1)
6:
7:     With myDocument.Shapes.AddShape(msoShapeRectangle, _
8:         0, 0, 100, 40).TextFrame
9:         .Characters.Text = "Пример текста внутри прямоугольника"
10:        .MarginBottom = 10
11:        .MarginLeft = 10
12:        .MarginRight = 10
13:        .MarginTop = 10
14:     End With
15: End Sub

```

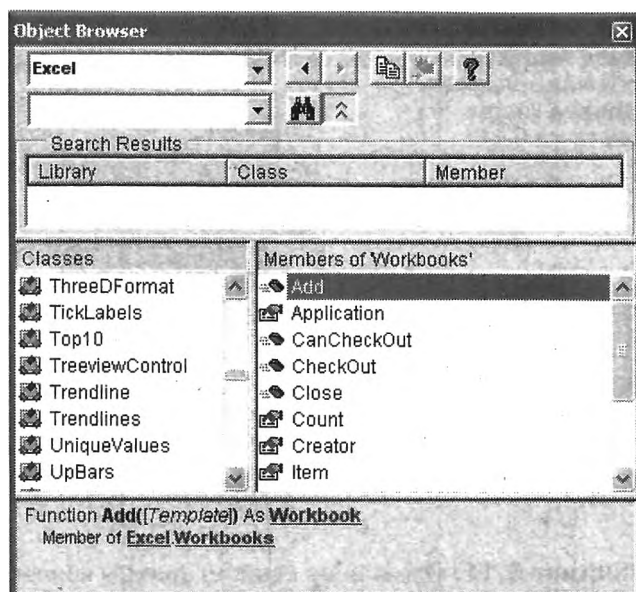
## Использование Object Browser для работы с объектами, методами и свойствами

Вы уже знаете, как использовать **Object Browser** для нахождения и получения справки VBA и функций host-приложений, процедур и констант. Подобным образом **Object Browser** используется с объектами, методами и свойствами; в действительности, основное назначение инструмента **Object Browser** (как предполагает его имя) — это обеспечивать возможность просмотра всех доступных объектов (с их различными методами и свойствами) в VBA и host-приложении.

Вы уже знаете, как запустить **Object Browser**: щелкните на кнопке **Object Browser** на панели инструментов Visual Basic или выберите команду **View | Object Browser** (Вид | Просмотр объектов) для отображения окна **Object Browser** (рис. 6.13).

**Рис. 6.13**

Используйте **Object Browser** для определения того, какие объекты доступны и какие свойства и методы принадлежат определенному объекту



Чтобы просмотреть список доступных объектов host-приложения и просмотреть список свойств и методов для каждого заданного объекта, выполните следующие шаги:

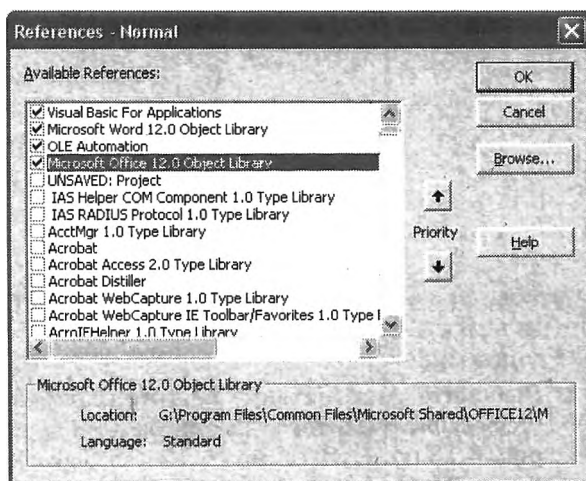
- Выберите библиотеку host-приложения в раскрывающемся списке **Project/Library** (Проект | Библиотека) в верхней части окна **Object Browser**. На рис. 6.13 показано открытое окно **Object Browser** в сеансе работы Редактора VB в Excel; библиотека Excel уже выбрана в раскрывающемся списке **Project/Library**. После выбора host-приложения в списке **Project/Library** список **Classes** (Классы) содержит все доступные для VBA объекты host-приложения.
- В списке **Classes** выберите объект, методы и свойства которого вас интересуют. На рис. 6.13 показана коллекция **Workbooks**.
- Чтобы получить более подробную информацию об объекте, выбранном вами в списке **Classes**, щелкните на кнопке со знаком вопроса (?) (в верхней правой части окна **Object Browser**) для доступа к справочной системе VBA. После выбора объекта в списке **Classes** список **Members of <class>** (Компонент <класс>) в окне **Object Browser** отображает все методы и свойства для этого объекта.
- В списке **Members of<class>** выберите нужный вам метод или свойство. **Object Browser** теперь отображает имя метода или свойства в нижней части окна; при выборе метода, имеющего аргументы, **Object Browser** перечисляет также все аргументы метода. На рис. 6.13 показан выбранный метод **Add** коллекции **Workbooks**.
- Щелкните на кнопке с вопросительным знаком в окне **Object Browser** для доступа к справочной системе VBA с целью получения подробной информации о методе или свойстве, выбранном в списке **Members of<class>**.

Чтобы вы могли различать методы и свойства в списке **Members of <class>**, **Object Browser** отображает особые значки слева от каждого элемента в списке. Свойства обозначаются значком с изображением руки, указывающей на индексную карту. Методы обозначаются значком, похожим на зеленый прямоугольник. Синий «мячик» обозначает, что свойство или метод является глобально доступным.

В заключение отметим, что по умолчанию VBA предлагает вам не все объекты, с которыми вы можете работать. Из любопытства и для более глубокого изучения открывающихся перед вами возможностей откройте окно **References** (меню **Tools | References**) в Редакторе VBA любого приложения (рис. 6.14).

**Рис. 6.14**

В окне **Available References** отмечены четыре библиотеки объектов, к объектам которых вы имеете доступ по умолчанию — они подключены



В окне **Available References** отмечены четыре библиотеки объектов, к объектам которых вы имеете доступ по умолчанию — они подключены. Все остальные пока не подключены, но «ждут», когда могут понадобиться вам.

# Повторение действий в Visual Basic: циклы и массивы

У нас осталась, пожалуй, последняя тема, без которой пока еще невозможно писать действительно полезные (но пока еще только простые) программы на языке VBA. Теперь, когда вы, возможно, научились выбирать различные действия на основе предопределенных условий, вы узнаете, что необходимо делать, чтобы процедуры VBA повторяли действия заданное количество раз или пока выполняется (или не выполняется) некоторое условие. В этой главе также описываются массивы. Массивы являются распространенным и полезным способом сохранения многих различных частей связанных данных. Массивы полезны при создании отсортированных и неотсортированных списков данных, при сохранении таблиц данных и для выполнения многих других задач. Циклы — это программные конструкции, без которых обработка массивов была бы довольно затруднительной.

## Команды организации циклов

Как уже вам стало ясно, макрорекордер позволяет записывать не все необходимые пользователю действия. Особенно это наглядно показано на примере операторов, изменяющих порядок выполнения кода процедур и функций. То же самое относится и к таким операторам, которые позволяют организовать циклическое повторение выполнения некоторых частей кода VBA.

Для организации циклов VBA предоставляет несколько мощных и гибких структур, позволяющих легко повторять различные действия. Программные структуры, приводящие к неоднократному повторению одного или нескольких операторов, называются *структурами организации циклов*, потому что поток выполнения операторов процедуры проходит циклично по одним и тем же операторам и обычно неоднократно.

Процесс выполнения всех операторов, заключенных в структуру цикла, один раз называется *итерацией (iteration)* цикла. Некоторые структуры цикла организуются так, что они всегда выполняются заданное количество раз. Структуры цикла, всегда выполняющиеся заданное количество раз, называются циклами с *фиксированным числом итераций (fixed iteration)*. Другие типы структур цикла повторяются переменное количество раз в зависимости от некоторого набора условий. Поскольку количество раз повторений этих гиб-

ких структур цикла является неопределенным, такие циклы называются *неопределенными циклами (indefinite loops)*.

Существуют два основных способа создания неопределенного цикла. Можно построить цикл так, что VBA будет тестировать некоторое условие (детерминант цикла) *перед* выполнением цикла: если условие для повторения цикла не равно **True**, VBA пропускает все операторы в цикле и прекращает цикл. Можно также построить цикл таким образом, что VBA будет тестировать условие детерминанта цикла *после* выполнения операторов в цикле.

Для цикла, в котором детерминант проверяется перед выполнением операторов внутри цикла, VBA определяет следующую последовательность действий:

- В верхней части цикла тестируется условие детерминанта.
- Если условия для выполнения цикла удовлетворяются, выполняются операторы внутри цикла.
- После выполнения последнего оператора внутри цикла VBA возвращается к началу цикла и снова оценивает условие детерминанта цикла.
- Если условия для выполнения цикла все еще удовлетворяются, VBA повторяет операторы внутри цикла и снова возвращается к началу цикла для тестирования детерминанта.
- Как только условия для выполнения цикла перестанут удовлетворяться, VBA останавливает выполнение цикла и начинает выполнение операторов после цикла.

Заметьте, что если условия для выполнения цикла не удовлетворяются, когда VBA тестирует условие детерминанта цикла первый раз, VBA не выполняет операторы внутри цикла *ни разу*; при этом тело цикла пропускается (не выполняется). *Тело (body)* цикла — это блок операторов VBA, находящийся между началом и концом цикла.

Для цикла, в котором детерминант проверяется после выполнения операторов в теле цикла, VBA определяет следующую последовательность действий:

- Выполняются все операторы в теле цикла.
- При достижении конца тела цикла, VBA оценивает условие детерминанта цикла.
- Если условия для выполнения цикла удовлетворяются, VBA возвращается к началу цикла и повторяет инструкции в теле цикла.
- В конце цикла VBA снова тестирует условие детерминанта цикла.
- Как только условия для выполнения цикла больше не удовлетворяются, VBA прекращает выполнение цикла и начинает выполнение операторов после цикла.

Заметьте, что VBA выполняет операторы в таком цикле, по крайней мере, один раз.

Можно также создать неопределенный цикл таким образом, чтобы он вообще не имел условия детерминанта; циклы, не имеющие условия детерминанта, повторяются бесконечно и называются *бесконечными циклами (infinite loops)*. Бесконечный цикл не заканчивается никогда; большинство бесконечных циклов являются результатом ошибок программирования, хотя имеется несколько случаев, когда удобно использовать бесконечные циклы.

## Повторение цикла фиксированное число раз: циклы For

Самой простой структурой цикла является фиксированный цикл. VBA предоставляет две различные структуры фиксированного цикла: **For...Next** и **For Each...Next**. Обе структуры фиксированного цикла называются циклами **For**, потому что они всегда выполняются *для (for)* заданного количества раз.

### Использование цикла For...Next

Первый из VBA-циклов **For** — это цикл **For...Next**. Используйте цикл **For...Next**, когда вам необходимо повторить действие или ряд действий заданное количество раз, известное до начала выполнения цикла.

Цикл **For...Next** имеет следующий синтаксис:

### Синтаксис

---

```
For counter = Start To End [Step StepSize]
    Statements
Next [counter]
```

Здесь *counter* — любая численная переменная VBA, обычно — переменная типа **Integer** или **Long**; *Start* — любое численное выражение и определяет начальное значение для переменной *counter*; *End* — это также численное выражение, определяет конечное значение для переменной *counter* (см. схему 7.1).

---

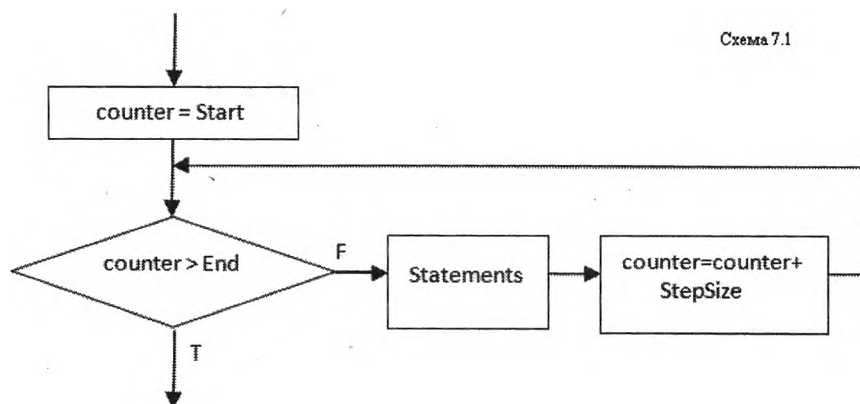
По умолчанию VBA увеличивает переменную *counter* на 1 каждый раз при выполнении операторов в цикле (считает количество циклов). Можно задавать другое значение (*StepSize*), на которое будет изменяться *counter*, включая необязательное ключевое слово **Step**. При включении ключевого слова **Step** необходимо задавать значение для изменения переменной *counter*. В указанной синтаксической конструкции *StepSize* представляет любое численное выражение и определяет значение для изменения переменной *counter*.

*Statements* представляет один, несколько или ни одного оператора VBA. Эти операторы составляют тело цикла **For**; VBA выполняет каждый из этих операторов при каждом выполнении цикла.

Ключевое слово **Next** сообщает VBA о том, что достигнут конец цикла; необязательная переменная *counter* после ключевого слова **Next** должна быть той же самой переменной *counter*, которая была задана после ключевого слова **For** в начале структуры цикла. Включайте необязательную переменную *counter* после ключевого слова **Next** для улучшения читабельности программного кода (особенно при использовании вложенных циклов **For...Next**) и для повышения скорости выполнения кода (иначе VBA придется потратить время на определение того, какая переменная *counter* является правильной, для ее изменения после достижения ключевого слова **Next**).

При выполнении цикла **For...Next** VBA поступает следующим образом:

- Присваивает значение, представленное *Star*, переменной *counter*.
- Выполняет все операторы, представленные с помощью *Statements*, пока не достигнет ключевого слова **Next**. Ключевое слово **Next** указывает VBA на то, что достигнут конец тела цикла.



- Изменяет переменную *counter* на величину *StepSize* (если включается не-обязательное ключевое слово **Step**). Если **Step** не определено, то VBA увеличивает переменную *counter* на 1.
- Возвращается к началу цикла и сравнивает текущее значение переменной *counter* со значением, представленным с помощью *End*. Если значение *counter* меньше или равно *End*, VBA выполняет цикл снова. Если значение *counter* больше *End*, VBA продолжает выполнение кода с первого оператора после ключевого слова **Next**.

### Использование **For...Next** с возрастающим счетчиком

В листинге 7.1 демонстрируется использование цикла **For...Next**. Эта процедура получает два числа от пользователя, складывает все числа в диапазоне, заданном этими двумя числами, а затем отображает результирующую сумму.

### Листинг 7.1. Демонстрация цикла **For...Next** с возрастающим счетчиком

```

1: Sub Demo_ForNext()
2:
3:   Dim k As Integer
4:   Dim sStart As String
5:   Dim sEnd As String
6:   Dim lSum As Long
7:
8:   sStart = InputBox("Введите целое число:")
9:   sEnd = InputBox("Введите другое целое число:")
10:
11:   lSum = 0
12:   For i = CInt(sStart) To CInt(sEnd)
13:     lSum = lSum + i
14:   Next i
15:
16:   MsgBox "Сумма чисел от " & sStart & _
17:     " до " & sEnd & " равна: " & lSum
18: End Sub
  
```



Переменная **i** используется как счетчик цикла **For...Next** (буквы **i, j, k, l, m**, и чаще всего применяются программистами в качестве счетчиков цикла, видимо, они больше других напоминают переменные целых типов); другие переменные используются для сохранения начального (**sStart**) и конечного значений (**sEnd**), полученных от пользователя, и для сохранения суммы чисел (**ISum**). Обратите внимание на типы этих переменных. Переменные **sStart** и **sEnd** в этой процедуре чаще всего используются для работы со строками (строки 8, 9, 16–17) и только в одном операторе (строка 12) нам понадобилось преобразовать их в числа. Переменная **ISum**, по существу, должна иметь тип числовой, причем как можно больший, потому что неизвестно, насколько большой диапазон будет вводиться пользователем, который будет испытывать этот код.

В процедуре предполагается, что первое вводимое число меньше, чем второе. Иначе тело цикла не выполнится ни разу и значение переменной **ISum** будет равно 0, так как она инициализируется этим значением в строке 11.

Со строки 12 начинается описание цикла **For...Next**. При обработке этой строки VBA сначала выполняет вызов функции **CInt**, которая преобразует пользовательский строковый ввод в числа типа **Integer**, а затем вставляет результирующие целые значения в оператор. VBA присваивает целый эквивалент значения, хранимого в **sStart**, переменной счетчика **i**. Затем значение в **i** сравнивается с целым эквивалентом числа, сохраненного в **sEnd**. Если **i** меньше или равно значению в **sEnd**, VBA выполняет тело цикла.

Этот цикл имеет только один оператор в своем теле: выражение сложения и присваивания (в строке 13). В строке складывается текущее значение **i** с текущим содержимым переменной **ISum**, а затем результат сложения присваивается переменной **ISum**.

Далее VBA переходит к оператору **Next** (в строке 14), который указывает на конец тела цикла **For**; VBA увеличивает переменную счетчика на 1 (по умолчанию) и возвращается к началу цикла (строка 12) и снова сравнивает значение в переменной счетчика **i** с числом, хранимым в переменной **sEnd**.

Как только значение в счетчике цикла превысит значение, определенное для конечного значения счетчика (а это зависит от вводимых данных), VBA прекращает выполнение цикла.

Теперь, когда цикл закончился, VBA продолжает выполнение кода с оператора **MsgBox** в строке 16, который отображает два введенных числа и сумму всех целых в этом диапазоне.

Если вы хорошо разобрались с циклом **For...Next**, то, наверное, заметили, что этот цикл (как, впрочем, и рассмотренные далее) можно реализовать оператором **If...Then...Else**. Чтобы это понять, достаточно сравнить диаграммы, сопровождающие синтаксис этих операторов. Дело только в том, что оператор **For...Next** и короче в записи, и быстрее при выполнении кода.

### Использование **For...Next** с убывающим счетчиком

Не всегда бывает нужно записывать циклы **For...Next** с возрастающим счетчиком. Иногда легче или полезнее писать цикл **For** с убывающим счетчиком. Чтобы цикл **For...Next** имел убывающий счетчик, используйте ключевое слово **Step** и отрицательное число (в синтаксической конструкции **StepSize** может иметь любой знак) для значения шага.

В листинге 7.2 показана процедура, полученная из предыдущего кода, но здесь добавлена возможность вводить значения шага цикла. Если при тестиро-

вании сначала ввести большее число, затем — меньшее и  $-1$ , то мы получим цикл **For...Next** с убывающим счетчиком. Понятно, что при вводе сначала меньшего числа, затем — большего и  $+1$ , мы получим обычный цикл с возрастающим счетчиком.

**Листинг 7.2.** Цикл **For...Next** с использованием шага изменения счетчика цикла

---

```
1: Sub Demo_ForNext()  
2:  
3:   Dim i As Integer  
4:   Dim sStart As String  
5:   Dim sEnd As String  
6:   Dim sStep As String  
7:   Dim uSum As Long  
8:  
9:   uStart = InputBox("Введите целое число:")  
10:  uEnd = InputBox("Введите другое целое число:")  
11:  sStep = InputBox("Введите шаг цикла (целое число):")  
12:  
13:  uSum = 0  
14:  For i = CInt(uStart) To CInt(uEnd) Step CInt(sStep)  
15:    uSum = uSum + i  
16:  Next i  
17:  
18:  MsgBox "Сумма чисел от " & uStart &  
19:    " до " & uEnd & " шагом " & sStep & " равна: " & uSum  
20: End Sub
```

---

Просматривая код листинга 7.2, помните, что при уменьшении счетчика цикла **For...Next** цикл выполняется, пока переменная счетчика больше или равна конечному значению, а когда счетчик цикла увеличивается, цикл выполняется, пока переменная счетчика меньше или равна конечному значению.

Приведем еще один пример с использованием цикла **For...Next** и обработки строковых переменных. Довольно часто в различных приложениях приходится создавать уникальные строки для использования их в качестве кодов некоторых объектов (например, в базах данных). Обычно для этих целей используют генераторы случайных (чаще всего — псевдослучайных) чисел и (при необходимости) переводят их в строки. Если получаемые строки используются и создаются в течение долгого времени, существует вероятность совпадения новых строк со старыми. Если уникальные строки (и соответствующие им другие данные) располагать в порядке возрастания (строки ведь тоже можно сравнивать), то их можно создавать так, что следующая создаваемая строка должна быть больше последней имеющейся в наборе, отличаясь только «на один символ». Например, для строк из четырех символов можно записать неравенства:

"A00X" < "A00Y" < "A00Z" < "A0010" ...

Код листинга 7.3 является примером функции, которая создает строку, отличающуюся от аргумента так, как отличаются друг от друга строки, приведенные выше.

**Листинг 7.3. Использование For ...Next — пример в Excel**

```

1: Function NetxCod(PrevCode As String) As String
2: 'функция возвращает код такой же длины, что и входной параметр
3: 'первый символ строки выбирается из диапазона символов,
4: 'отличающегося от диапазона для остальных символов
5:
6:     Dim mas1 As String, masi As String
7:     Dim s4 As String, s3 As String, s2 As String, s1 As String
8:     Dim lenmas1 As Integer, lenmas2 As Integer, lenmasNewCode
                                     As Integer
9:     Dim masNewCode() As String      'массив с новыми значениями
                                     'символов кода
10:
11:     'строки для выбора символов
12:     mas1 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"      'для первого символа
13:     masi = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ" 'для остальных
14:
15:     lenmas1 = Len(mas1)      'длина строки mas1
16:     lenmasi = Len(masi)      'длина строки masi
17:     lenmasNewCode = Len(PrevCode) 'длина входной строки
18:
19:     'заполнение массива символами входной строки:
20:     For i = 1 To lenmasNewCode
21:         ReDim Preserve masNewCode(i)
22:         masNewCode(i - 1) = Mid(PrevCode, i, 1)
23:     Next
24:
25:     'замена символов входной строки на новые,
26:     'начиная с последнего символа:
27:     For i = lenmasNewCode - 1 To 0 Step -1
28:         If i > 0 Then
29:             'положение i-го символа в заданном диапазоне:
30:             posi = InStr(1, masi, masNewCode(i))
31:             If posi = lenmasi Then
32:                 'символ последний в диапазоне:
33:                 masNewCode(i) = Mid(masi, 1, 1)
34:             Else
35:                 'выбор следующего символа из диапазона:
36:                 masNewCode(i) = Mid(masi, posi + 1, 1)
37:             Exit For
38:         End If
39:     Else
40:         'изменение первого символа в строке:
41:         posi = InStr(1, mas1, masNewCode(i))
42:         If posi = lenmas1 Then
43:             'аварийный, но вряд ли достижимый случай:
44:             MsgBox "Первый символ вышел из своего диапазона!"
45:         Else
46:             masNewCode(i) = Mid(mas1, posi + 1, 1)
47:         End If
48:     End If
49:
50:     Next
51:
52:     'формирование результата функции:
53:     NetxCod = ""
54:     For i = 0 To lenmasNewCode - 1

```

```

55:      NetxCde = NetxCde & masNewCde(i)
56:      Next
57:
58: End Function

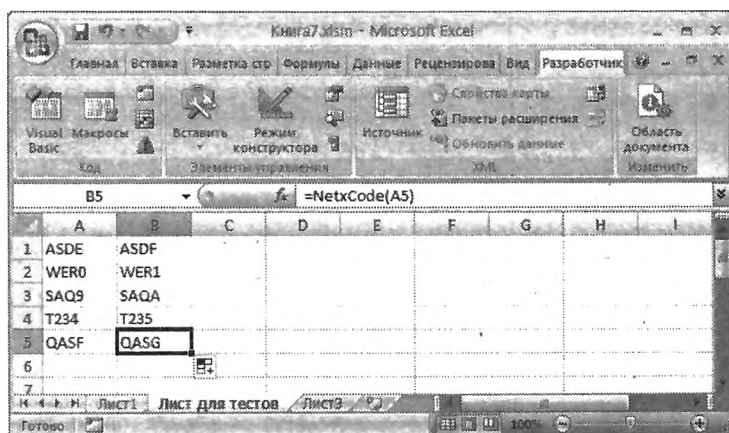
```

Основная идея алгоритма, реализованного этим кодом, заключается в следующем. Каждая позиция в генерируемой строке может содержать символы из некоторого диапазона символов<sup>1</sup>, который можно задавать произвольным образом, но символы диапазона должны быть упорядочены по возрастанию. С использованием цикла с отрицательным шагом (строки 27–50) входная строка просматривается, начиная с последнего символа; определяется позиция последнего символа в заданном для него диапазоне. Если текущий рассматриваемый символ не совпадает с последним символом допустимого диапазона (находится в середине диапазона), то в выходной строке только этот символ заменяется на следующий символ из диапазона; на этом работа алгоритма заканчивается. Если же рассматриваемый символ совпадает с последним символом заданного диапазона, то он в выходной строке заменяется на первый символ диапазона, с символом, стоящим во входной строке слева от рассматриваемого, производятся те же действия, т.е. он становится текущим рассматриваемым символом.

Для задания диапазона допустимых символов используются строки символов: **mas1** — допустимый диапазон начала строки (первый символ), **masi** — допустимый диапазон для остальных символов. Для положения текущего символа в диапазоне (в подстроке) используется функция **InStr**, позволяющая найти положение одной строки в другой строке. Функция **Mid** используется для выделения указанного (индексом) символа из строки.

На рис. 7.1 приведен пример использования функции: в колонке **A** помещены заданные строки (различной длины), а в колонке **B** — формулы **NextCde(Ai)**. Обратите внимание на то, что результирующие строки отличаются от исходных на один или два символа (в конце).

**Рис. 7.1**  
Использование  
функции листинга 7.3



<sup>1</sup> В общем случае для каждой позиции в строке можно задавать свой диапазон допустимых символов. В рассматриваемом коде только первый символ выходной строки имеет диапазон символов, отличный от диапазона для остальных символов, что определялось спецификой реальной задачи, которая послужила поводом для включения этого кода в книгу.

### Цикл For Each...Next

Второй цикл **For**, который имеется в VBA — это цикл **For Each...Next**. В отличие от цикла **For...Next** цикл **For Each...Next** не использует счетчик цикла. Циклы **For Each...Next** выполняются столько раз, сколько имеется элементов в определенной группе, такой как коллекция объектов или массив. (Коллекции объектов описываются в предыдущей, а массивы — далее в этой главе.) Другими словами, цикл **For Each...Next** выполняется один раз для каждого элемента в группе.

Цикл **For Each...Next** имеет следующий синтаксис:

#### Синтаксис

---

```
For Each Element In Group
    Statements
Next [Element]
```

*Element* — это переменная, используемая для итерации по всем элементам в определенной группе. *Group* — это объект коллекции или массив. Если *Group* — это объект коллекции, то *Element* должен быть переменной типа **Variant**, **Object** или заданным объектным типом, таким как **Range**, **Worksheet**, **Document**, **Paragraph** и так далее. Если *Group* — это массив, то *Element* должен быть переменной типа **Variant**; *Statements* — один, несколько или ни одного оператора VBA, составляющих тело цикла.

---

Цикл **For Each...Next** имеет меньше опций, чем цикл **For...Next**. Цикл **For Each...Next** всегда выполняется столько раз, сколько имеется элементов в определенной (указанной в операторе) группе.

В листинге 7.4 показана функция с именем **SheetExists**, использующая цикл **For Each...Next** для определения того, существует ли определенный лист в рабочей книге Excel.

#### Листинг 7.4. Использование For Each...Next — пример в Excel

---

```
1: Function SheetExists(sName As String) As Boolean
2:     'Возвращает True, если лист sName в активной книге
3:
4:     Dim aSheet As Object
5:     SheetExists = False
6:
7:     'цикл по всем листам со сравнением имен с sName,
8:     'сравнение текстовое
9:     For Each aSheet In ActiveWorkbook.Sheets
10:         If (StrComp(aSheet.Name, sName, 1) = 0) Then
11:             SheetExists = True 'найдено имя
12:             Exit For           'выход из цикла
13:         End If
14:     Next aSheet
15: End Function
16:
17: Sub Test_SheetExists()
18:     'тестирует функцию SheetExists
19:
20:     Dim uStr As String
```

```
21:
22:   uStr = InputBox("Введите имя листа " & _
23:                  "текущей книги:")
24:   If SheetExists(uStr) Then
25:     MsgBox "Лист '" & uStr & _
26:           "' в текущей книге."
27:   Else
28:     MsgBox "Листа '" & uStr & _
29:           "' НЕТ в текущей книге."
30:   End If
31: End Sub
```

Функция **SheetExists** имеет один обязательный аргумент **sName**, в котором функции передается строка, содержащая имя искомого листа. **SheetExists** возвращает значения типа **Boolean**: **True**, если лист с именем в **sName**, существует в текущей рабочей книге, иначе — **False**. Можно использовать подобную функцию в одной из процедур, чтобы убедиться в том, что рабочий лист (или диаграмма) существует, перед тем, как процедура выполнит команду обращения к этому листу. Если этого не предусмотреть, а лист действительно отсутствует, такая команда вызовет сообщение о runtime-ошибке и VBA прекратит выполнение программы. Функция, подобная **SheetExists**, позволяет писать процедуры так, чтобы они могли обнаруживать и исключать ошибки, которые могут возникнуть, например, когда пользователь вводит имя несуществующего рабочего листа.

В строке 4 объявляется объектная переменная **aSheet** для использования в цикле **For Each...Next**. В строке 5 результату функции **SheetExists** присваивается значение **False**. Теперь функция должна только установить, существует ли лист, и присвоить **True** результату функции в случае положительного результата. Если результату функции **SheetExists** не удалось присвоить значение **True**, значит лист не найден или случилось что-то еще, что подразумевает отсутствие листа.

В строке 9 начинается цикл **For Each...Next**, использующий объектную переменную **aSheet** как переменную *Element* для цикла. В качестве *Group* здесь выступает коллекция **ActiveWorkbook.Sheets**. Свойство **ActiveWorkbook** — это свойство объекта **Application** и оно возвращает объектную ссылку на текущую активную рабочую книгу. **Sheets** — это метод рабочей книги, который возвращает коллекцию всех листов в рабочей книге, включая рабочие листы и диаграммы.

В строке 10 начинается тело цикла, которое состоит из одного оператора **If...Then**. При выполнении строки 10 VBA сначала вызывает **StrComp** для сравнения строки в **sName** со строкой, сохраненной в свойстве **Name** листа, на который в данный момент ссылается **aSheet**. Этот вызов функции **StrComp** определяет, что сравнение должно быть текстовым, и, следовательно, на него не оказывают влияние различия в регистре строки в **sName** и строки в свойстве **Name** листа.

Если строка в **sName** совпадает со строкой в свойстве **Name** листа, VBA выполняет оператор в строке 11, который присваивает значение **True** результату функции. В строке 12 располагается оператор **Exit For**. Если вы еще не догадались, к чему приводит этот оператор, значит, вы еще не задумывались над тем, сколько же раз будет выполняться цикл в данном примере, даже если не-

обходимый лист найдется в первой же итерации. На самом деле **Exit For** — это оператор немедленного выхода из цикла. Он не зависит от текущей итерации. Данный пример, как никакой другой, подходит для демонстрации этого оператора. Как только строка в **sName** совпадет со строкой в свойстве **Name** листа, **VBA** выполнит операторы в строках 11–12 и дальнейшая необходимость в цикле отпадет, хотя его можно было бы без вреда для программы выполнять и далее. В данной функции досрочный выход из цикла экономит только время (слово «только» использовано здесь не в том смысле, что об этом не стоит беспокоиться; чаще всего в программах с хорошим пользовательским интерфейсом время — основная характеристика системы после надежности).

Когда заканчивается (или прерывается) выполнение цикла, **VBA** продолжает выполнение кода со строки 15, которой заканчивается функция **SheetExists**, и возвращает результат функции.

В строках 17–30 описана процедура **Test\_SheetExists** для тестирования функции **SheetExist**. Процедура использует только уже изученные вами операторы, и, видимо, нет необходимости в ее подробном разборе.

В листинге 7.5 показан еще один пример использования **For Each...Next**: на этот раз — для **Word**. Функция **BookmarkExists** из листинга 7.5 проверяет, существует ли определенная закладка в активном в данный момент документе, и если — существует, переводит курсор к этой закладке.

#### Листинг 7.5. Поиск закладки в документе Word

```
1: Function BookmarkExists(BkMark As String) As Boolean
2:     'Возвращает True, если закладка BkMark существует
3:     'в активном документе
4:
5:     Dim InstBkMark As Object
6:
7:     BookmarkExists = False 'предположим, закладка не найдена
8:
9:     'цикл по всем закладкам; сравнение имени каждой
10:    'закладки с BkMark; сравнение текстовое
11:    For Each InstBkMark In ActiveDocument.Bookmarks
12:        If (StrComp(InstBkMark.Name, BkMark, vbTextCompare) = 0) Then
13:            'перейти к закладке:
14:            Selection.GoTo What:=wdGoToBookmark, Name:=BkMark
15:            BookmarkExists = True
16:            Exit For
17:        End If
18:    Next InstBkMark
19: End Function
20:
21:
22: Sub Test_BookmarkExists()
23:     'тестирует функцию BookmarkExists
24:
25:     Dim uStr As String
26:
27:     uStr = InputBox("Введите имя закладки " & _
28:         "в текущем документе:")
29:     If BookmarkExists(uStr) Then
30:         MsgBox "Закладка '" & uStr & _
31:             "' НАХОДИТСЯ в этом документе"
```

```

32: Else
33:     MsgBox "Закладки '" & uStr & _
34:         "' НЕТ в этом документе" —
35: End If
36: End Sub

```

Функция **BookmarkExists** работает в основном так же, как и функция **SheetExists**. **BookmarkExists** имеет один обязательный аргумент **BkMark**, который содержит строку, представляющую имя закладки, поиск которой выполняется. **BookmarkExists** возвращает значение типа **Boolean**: **True**, если закладка, определяемая с помощью **BkMark**, находится в текущем документе, иначе — **False**.

В строке 5 объявляется объектная переменная **InstBkMark** для использования в цикле **For Each...Next**.

В строке 11 начинается цикл **For Each...Next**, использующий объектную переменную **InstBkMark** как переменную *Element* для цикла. В качестве *Group* выступает коллекция **ActiveDocument.Bookmarks**: **ActiveDocument** — это свойство объекта **Word Application**, которое возвращает объектную ссылку на активный в текущий момент документ. **Bookmarks** — это метод документа, возвращающий коллекцию всех закладок в документе.

В строке 12 начинается тело цикла, которое состоит из одного оператора **If...Then**. При выполнении строки 12 VBA сначала вызывает функцию **StrComp** для сравнения строки в **BkMark** со строкой, сохраненной в свойстве **Name** закладки, на которую в данный момент ссылается **InstBkMark**. При сравнении строк используется функция **StrComp** для текстового сравнения строк независимо от установок **Option Compare**.

Если строка в **BkMark** совпадает со строкой в свойстве **Name** закладки, VBA выполняет код в строках 14–15. В строке 15 присваивается значение **True** результату функции. С этим вы уже встречались в предыдущем листинге. В 14 строке располагается оператор перехода к найденной закладке. Этот оператор был получен из макроса, записанного рекордером, и состоит из вызова метода **GoTo** объекта **Selection**. Метод имеет два аргумента: **What**, которому присваивается значение встроенной константы **wdGoToBookmark** (выделенный фрагмент — закладка), и **Name** (наименование закладки); которому присваивается значение найденной метки.

Как и в предыдущем листинге, здесь используется «досрочный» выход из цикла, если закладка найдена.

Когда выполнение цикла для всех объектов в коллекции **ActiveDocument.Bookmarks** заканчивается, VBA продолжает выполнение кода со строки 19, которая заканчивает функцию **BookmarkExists** и возвращает результат функции.

В строках 22–36 объявляется процедура **Test\_BookmarkExists** для тестирования функции **BookmarkExists**.

## Циклы Do

VBA имеет чрезвычайно мощный оператор цикла для создания неопределенных структур цикла в процедурах и функциях. Обычно все VBA-неопределенные циклы строятся с помощью одного оператора неопределенного цикла: оператора **Do**. Оператор **Do** имеет так много опций и является настолько гибким, что, в действительности, он предоставляет четыре различных конструкции цикла в двух базовых категориях.



Две базовые категории конструкций цикла **Do** — это циклы, которые тестируют условие детерминанта до выполнения тела цикла, и циклы, которые тестируют условие детерминанта после выполнения тела цикла.

Условие детерминанта для неопределенного цикла задается с помощью логического выражения таким же образом, каким создаются логические выражения для использования с операторами **If...Then**.

VBA предоставляет два различных способа тестирования условия детерминанта для цикла. Можно построить цикл так, чтобы он выполнялся, пока условие детерминанта цикла будет равно **True**, и прекращал выполнение, когда это условие становится равным **False**. Можно также создать цикл так, чтобы он выполнялся, пока условие детерминанта цикла будет равно **False**, и прекращал выполнение, когда это условие станет равным **True**.

Чтобы цикл выполнялся, пока его условие детерминанта равно **True**, используйте в операторе цикла **Do** необязательное ключевое слово **While**. В этом случае VBA выполняет цикл, *пока (while)* условие детерминанта равно **True**, и прекращает выполнение цикла, как только это условие становится равным **False**.

Предположим, вам необходимо написать процедуру, помогающую пользователю вводить данные в рабочий лист. Для процедуры ввода данных обычно желательно, чтобы процедура повторяла некоторые операторы, получающие информацию от пользователя, пока еще есть данные для ввода, и прекращала повторение этих операторов, когда данных больше нет. Можно написать цикл **Do**, неоднократно получающий данные от пользователя, продолжающийся, пока пользователь вводит непустые строки, и прекращающий выполнение, как только пользователь вводит пустую строку. Условие детерминанта для этого цикла записывается так, чтобы логическое выражение было равно **True** (и поэтому цикл продолжал выполняться), пока пользовательский ввод не является пустой строкой. Если пользовательский ввод сохраняется в переменной с именем **uStr**, логическое выражение для детерминанта цикла может быть любым из следующих (оба логических выражения равны **True**, когда строка в **uStr** имеет ненулевую длину):

```
uStr <> ""  
Len(Trim(uStr)) <> 0
```

Для того чтобы цикл выполнялся, пока его условие детерминанта равно **False**, используйте в операторе цикла **Do** необязательное ключевое слово **Until**. VBA выполняет такой цикл до тех пор, *пока (until)* условие не будет равно **True**, то есть цикл продолжает выполняться, пока условие детерминанта равно **False**, и прекращает выполнение, как только это условие становится равным **True**.

Например, может понадобиться написать процедуру, продвигающуюся по столбцам рабочего листа, применяя определенное форматирование текста, и останавливающуюся в первом пустом столбце. Можно написать цикл **Do**, который повторяет операторы для форматирования заголовка столбца по достижении пустой ячейки. Условие детерминанта для этого цикла записывается так, что соответствующее логическое выражение равно **False**, пока текущая ячейка не является пустой, и равно **True**, когда текущая ячейка пустая. При использовании свойства **ActiveCell** объекта **Application** для возврата текущей активной ячейки и при использовании свойства **Value** для получения содержимого ячейки это логическое выражение для детерминанта может быть любым из следующих (оба выражения равны **True**, если текущая ячейка пустая):

```
ActiveCell.Value = ""  
Len(Trim(ActiveCell.Value)) = 0
```

Используется ли ключевое слово **While** или **Until** — зависит в основном от того, как вы представляете условие, определяющее, должен ли цикл выполняться, и от вашего умения строить логическое выражение для условия детерминанта цикла:

- ☐ используйте **While**, если хотите, чтобы цикл продолжал выполняться, пока заданное условие равно **True**.
- ☐ используйте **Until**, если хотите, чтобы цикл продолжал выполняться, пока заданное условие равно **False**.

## Как прервать выполнение макроса или процедуры

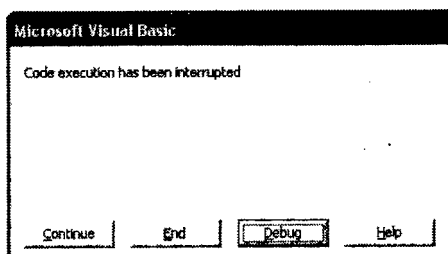
Поскольку одной из наиболее распространенных проблем, возникающих при работе с неопределенными циклами, является нечаянное создание бесконечного цикла, важно знать, как прерывать работу VBA, когда выполняются макросы и процедуры. Используйте один и тот же метод для прерывания записанных рекордером макросов или для прерывания процедур и функций, написанных вами самостоятельно.

Для прерывания выполнения VBA нажмите клавишу **Esc** или комбинацию клавиш **Ctrl+Break**. VBA заканчивает выполнение текущего оператора, переходит в состояние ожидания и отображает окно сообщения о runtime-ошибке, показанное на рис. 7.2. Командные кнопки в этом окне сообщения те же и имеют то же назначение, что и в любом другом окне сообщения о runtime-ошибке:

- ☐ *Continue*. Возобновляет VBA-выполнение программных операторов. Обычно не следует выбирать эту кнопку после прерывания бесконечного цикла, поскольку она просто запустит бесконечный цикл снова.
- ☐ *End*. Заканчивает программу. Все переменные и их содержимое теряются. Это командная кнопка, которую обычно следует выбирать после прерывания бесконечного цикла.
- ☐ *Debug*. Активизирует VBA-режим **Break** (режим прерывания), чтобы вы могли проверить содержимое переменных и по шагам пройти выполнение цикла для определения причины заикливания.
- ☐ *Help*. Активизирует справочную систему VBA для отображения информации о том, почему было прервано выполнение кода. Кнопка **Help** не очень полезна, если вы прервали выполнение кода нажатием на клавишу **Esc** или на **Ctrl+Break**; раздел справки констатирует, что выполнение кода было прекращено в результате нажатия на **Esc** или **Ctrl+Break**, и дает инструкции, как ввести режим **Break** или закончить выполнение кода.

**Рис. 7.2**

VBA отображает это окно, когда вы нажимаете **Esc** или **Ctrl+Break** для прерывания выполняющейся процедуры (или макроса)



Не следует прерывать VBA во время отображения диалогового окна; необходимо закрыть все окна ввода, окна сообщения или другие до того, как вы прервете процедуру.

В зависимости от конкретного цикла и операторов, которые в нем содержатся, VBA может не реагировать на одно нажатие на Esc или Ctrl+Break. В действительности, обычно необходимо нажимать и удерживать нажатой клавишу Esc для прерывания выполняющейся процедуры. Если вы удерживаете нажатой клавишу Esc, вы можете нечаянно закрыть окно сообщения о runtime-ошибке. В результате этого окно сообщения об ошибке может промелькнуть на экране слишком быстро, и вы не сможете его прочитать. Если такое случится, не стоит беспокоиться, вы можете понять, какой цикл был заблокирован, зная, насколько была выполнена процедура перед сбоем.

## Использование циклов, тестирующих условия до выполнения тела цикла

Для VBA-тестирования условия детерминанта цикла до выполнения тела цикла следует помещать логическое выражение для детерминанта цикла в начало блока операторов, составляющего тело цикла.

### Циклы Do While

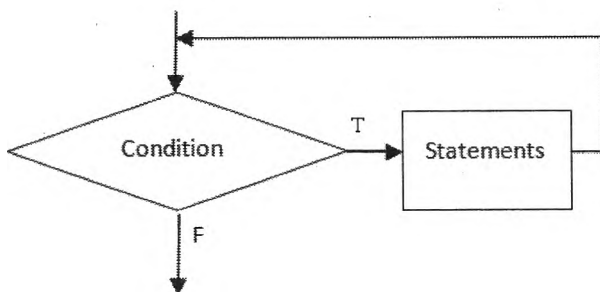
Первой конструкцией цикла, тестирующей свое условие детерминанта до выполнения цикла, является **Do While** со следующим синтаксисом:

#### Синтаксис

```
Do While Condition  
· Statements  
Loop
```

*Condition* — логическое выражение для детерминанта цикла; *Statements* — один, ни одного или несколько операторов, которые составляют тело цикла. Ключевое слово **Loop** после *Statements* указывает на окончание тела цикла и обозначает место, из которого VBA возвращается в начало цикла для проверки условия детерминанта (см. схему 7.2).

Схема 7.2



В операторе **Do While** выражение *Condition* находится в начале цикла, поэтому VBA проверяет условие детерминанта до выполнения цикла. Поскольку

в этой форме используется ключевое слово **While**, VBA выполняет цикл, пока логическое выражение, представленное с помощью *Condition*, равно **True**.

При выполнении цикла **Do While** сначала тестируется логическое выражение, представленное с помощью *Condition*; если оно равно **True**, VBA выполняет операторы, представленные с помощью *Statements*. При достижении ключевого слова **Loop** VBA возвращается в начало цикла и снова проверяет, равно ли **True** логическое выражение *Condition* (результат оценки). Если *Condition* равно **True**, VBA выполняет цикл снова; если выражение *Condition* равно **False**, VBA продолжает выполнение кода с любых операторов после ключевого слова **Loop**, т.е. прекращает работу цикла.

Заметьте, что если логическое выражение, представленное с помощью *Condition*, равно **False**, когда VBA выполняет оператор **Do While** в первый раз, VBA просто пропускает цикл, не выполняя его ни одного раза. Т.е. цикл **Do While** позволяет ни разу не выполнять операторы внутри него.

В листинге 7.6 показан простой пример цикла **Do While**, который неоднократно получает некоторое число от пользователя, останавливаясь только после того, как пользователь введет столько нечетных чисел, что их сумма превысит число 100.

#### Листинг 7.6. Демонстрация цикла **Do While**

---

```

1: Sub Test_DoWhile()
2: 'считает нечетные числа, вводимые пользователем;
3: 'запоминает строку с нечетными числами;
4: 'останавливается, когда их сумма нечетных чисел превысит 100;
5: 'выводит строку с нечетными числами в диалоговом окне
6:
7:   Const ocTitle = "Сумматор нечетных чисел"
8:
9:   Dim OddSum As Integer      'сумма нечетных чисел
10:  Dim OddStr As String       'строка с нечетными числами
11:  Dim Num                    'для приема данных пользователя
12:
13:  OddStr = ""                'инициализация выходной строки
14:  OddSum = 0                  'инициализация суммы OddSum
15:
16:  Do While OddSum < 100       'начало цикла
17:    Num = InputBox("Введите число:", ocTitle)
18:    If (Num Mod 2) <> 0 Then   'проверка на четность
19:      OddSum = OddSum + Num  'изменение суммы OddSum
20:      OddStr = OddStr & Num & " "
21:    End If
22:  Loop
23:
24:  'вывод строки с нечетными числами:
25:  MsgBox prompt:="Вы ввели следующие нечетные " & _
26:    "числа: " & Chr(13) & OddStr, _
27:    Title:=ocTitle
28: End Sub

```

---

Процедура **Test\_DoWhile** демонстрирует использование цикла **Do While**. Цикл в процедуре выполняется, пока сумма введенных пользователем нечетных чисел [*нечетное (odd)* число — это число, которое нельзя разделить на 2

без остатка] меньше 100. Очевидно, что эту задачу нельзя выполнить с помощью цикла **For...Next** (без дополнительного использования условного оператора).

Заметьте, что переменная **OddSum**, которая входит в условие детерминанта, изменяется только, когда пользователь вводит нечетное число. Поэтому цикл **Do While** в этой процедуре всегда будет выполняться разное количество раз: все зависит от пользовательского ввода.

Если вы не раз запустите эту процедуру, то, вероятно, обратите внимание, что в диалоговом окне ввода, кроме кнопки **ОК**, находится кнопка **Cancel**. Чем меньшие нечетные числа вы будете вводить для тестирования процедуры, тем чаще (так как сумма будет расти медленно) будете видеть, эти две кнопки. Когда-нибудь вы поддадитесь соблазну нажать на кнопку **Cancel**, и увидите, что программа прекратит работу с выдачей окна runtime-ошибки. Это обстоятельство не является таким уж неожиданным: программа рассчитана на «правильного» пользователя, который должен вводить четные и нечетные числа и не «капризничать». Конечно, если бы мы могли убрать из диалогового окна кнопку **Cancel**, то все проблемы были бы решены.

Выход из этой ситуации может быть один — проверять, не была ли нажата кнопка **Cancel** во время ввода чисел. В листинге 7.7 содержится код, который учитывает возможность отказа от ввода данных.

#### Листинг 7.7. Демонстрация цикла **Do While**

```

1: Sub Test_DoWhile2()
2: 'считает нечетные числа, вводимые пользователем;
3: 'запоминает строку с нечетными числами;
4: 'останавливается, когда их сумма нечетных чисел превысит число,
5: 'вводимое в диалоговом окне первым;
6: 'выводит строку с нечетными числами в диалоговом окне
7:
8:   Const ocTitle = "Сумматор нечетных чисел"
9:
10:  Dim Limit           'предел суммы нечетных чисел
11:  Dim OddSum As Integer 'сумма нечетных чисел
12:  Dim OddStr As String 'строка с нечетными числами
13:  Dim Num             'для приема данных пользователя
14:
15:  OddStr = ""          'инициализация выходной строки
16:  OddSum = 0           'инициализация суммы OddSum
17:
18:  Limit = InputBox("Введите максимальную сумму:", ocTitle)
19:  If Len(Limit) = 0 Then Exit Sub
20:
21:  Do While OddSum < Limit 'начало цикла
22:    Num = InputBox("Введите число:", ocTitle)
23:    If Len(Num) = 0 Then Exit Do
24:    If (Num Mod 2) <> 0 Then 'проверка на четность
25:      OddSum = OddSum + Num 'изменение суммы OddSum
26:      OddStr = OddStr & Num & " "
27:    End If
28:  Loop
29:
30:  'вывод строки с нечетными числами:
31:  MsgBox prompt:="Вы ввели следующие нечетные " & _

```

```
32:          "числа: " & Chr(13) & OddStr, _  
33:          Title:=ocTitle  
34: End Sub
```

---

Программа **Test\_DoWhile2** немного продвинута относительно своей предшественницы. Во-первых, она позволяет вводить в диалоге предел суммы нечетных чисел, после которого происходит выход из цикла (строка 18), во-вторых, в окне ввода можно использовать все (две) кнопки, так как операторы в строках 19 и 23 проверяют, не была ли нажата кнопка **Cancel**, и в первом случае (строка 19) прекращают выполнение процедуры, а во втором (строка 23) — цикла **Do While**.

Во всяком случае, если вы разрабатываете код не для себя, следует учитывать даже такие мелочи. Иначе пользователей ваших тестов будут больше занимать проблемы неработоспособности кода, чем его алгоритмическая сущность.

### Циклы **Do Until**

Вариант **Do While** оператора **Do** — это один из способов создания оператора **Do**, тестирующего условие детерминанта до выполнения тела цикла. Можно также использовать форму **Do Until** оператора **Do** для создания цикла, тестирующего условие детерминанта до выполнения тела цикла.

Оператор **Do Until** имеет следующий общий синтаксис:

### Синтаксис

---

```
Do Until Condition  
    Statements  
Loop
```

*Condition* — логическое выражение для детерминанта цикла, а *Statements* — это операторы VBA, составляющие тело цикла. Ключевое слово **Loop** после *Statements* указывает на конец тела цикла и также обозначает место, из которого VBA возвращается в начало цикла для проверки условия детерминанта (см. диаграмму синтаксиса цикла **Do While**).

---

VBA тестирует условие детерминанта цикла **Do Until** до выполнения операторов цикла. Поскольку эта форма включает ключевое слово **Until**, VBA выполняет цикл, пока логическое выражение, представленное выражением *Condition*, равно значению **False**.

При выполнении цикла **Do Until** VBA сначала тестирует логическое выражение, представленное с помощью *Condition*; если оно равно **False**, VBA выполняет операторы, представленные с помощью *Statements*. После достижения ключевого слова **Loop** VBA возвращается в начало цикла и снова проверяет, равно ли логическое выражение *Condition* значению **False**. Если *Condition* равно **False**, VBA выполняет цикл снова; если же оно равно **True**, VBA продолжает выполнение кода с операторов после ключевого слова **Loop**, т.е. прекращает работу цикла.

Если *Condition* равно **True**, когда VBA встречает оператор **Do Until** в первый раз, VBA пропускает цикл без его выполнения. Т.е. цикл **Do Until** позволяет ни разу не выполнять операторы внутри него.

В листинге 7.8 показан простой пример цикла **Do Until**, который неоднократно получает число от пользователя, прекращаясь, когда пользователь вводит четное число, большее 10.

### Листинг 7.8. Демонстрация цикла **Do Until**

```
1: Sub Test_DoUntil()  
2: 'принимает от пользователя числа, пока не будет  
3: 'введено четное число, большее десяти  
4:  
5:     Const evTitle = "Останов при четном, большем десяти"  
6:  
7:     Dim EvenFlag As Boolean  
8:     Dim Num  
9:  
10:    EvenFlag = False  
11:    Do Until EvenFlag  
12:        Num = InputBox("Введите число:", evTitle)  
13:        If Len(Num) = 0 Or TypeName(Num) <> "Integer" Then Exit Do  
14:        If (Num Mod 2) = 0 Then  
15:            If Num > 10 Then  
16:                MsgBox prompt:="Вы ввели четное число, " & _  
17:                    "большее, чем 10 - Цикл заканчивается", _  
18:                    Title:=evTitle  
19:                EvenFlag = True  
20:            Else  
21:                MsgBox prompt:="Вы ввели четное число, " & _  
22:                    "меньшее, чем (или равное) 10", _  
23:                    Title:=evTitle  
24:            End If  
25:        Else  
26:            MsgBox prompt:="Вы ввели нечетное число", _  
27:                Title:=evTitle  
28:        End If  
29:    Loop  
30:  
31:    MsgBox prompt:="Выполнение цикла прекращено", Title:=evTitle  
32: End Sub
```

Процедура **Test\_DoUntil** показывает, как построить цикл **Do Until**. Цикл в этой процедуре демонстрирует управляемый условием цикл и выполняется до тех пор, пока пользователь не введет четное число, большее 10. Условие, заканчивающее выполнение цикла — это ввод пользователем четного числа. На самом деле, этим условием является получение значения **True** переменной **EvenFlag**, что происходит (в строке 19), когда код обнаруживает с помощью вложенных операторов **If** (в строках 14–15) факт введения пользователем четного числа, большего 10.

Перед выполнением цикла переменной **EvenFlag** присваивается значение **False**, чем и обеспечивается, по крайней мере, первое выполнение цикла. Внутри цикла организуется прием числа, и анализ его на предмет четности и превышения 10. Если эти последние проверки дают положительный результат, выводится сообщение (строки 16–17) об окончании цикла и переменная **EvenFlag** получает значение **True**, что приводит к окончанию цикла.

В строке 13 находится оператор, предупреждающий возникновение runtime-ошибок при работе с программой пользователей, склонных к экспериментам.

## Использование циклов, тестирующих условия после выполнения тела цикла

Для VBA-тестирования условий после выполнения тела цикла необходимо поместить логическое выражение для детерминанта цикла в конец блока операторов, составляющих тело цикла, после ключевого слова **Loop**, которое сообщает о конце цикла.

### Циклы *Do...Loop While*

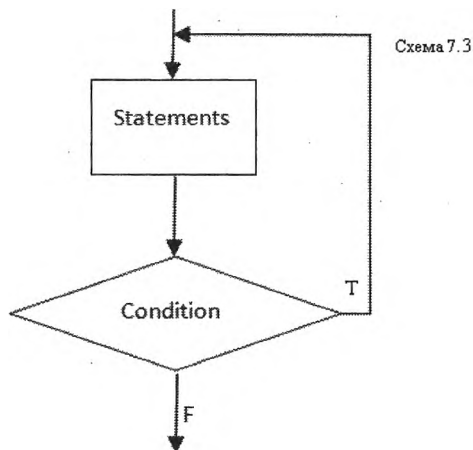
Первая рассматриваемая нами конструкция цикла, тестирующая условие детерминанта после выполнения тела цикла, — это **Do...Loop While**.

Оператор **Do...Loop While** имеет следующий синтаксис:

### Синтаксис

```
Do  
    Statements  
Loop While Condition
```

*Statements* — один, ни одного или несколько операторов VBA, составляющих тело цикла. Ключевое слово **Loop** после *Statements* обозначает конец тела цикла и также указывает место, из которого VBA возвращается в начало цикла; *Condition* представляет логическое выражение для детерминанта цикла (см. схему 7.3).



В этой форме синтаксиса оператора **Do...Loop While** VBA проверяет условие детерминанта после выполнения операторов цикла. Поскольку эта форма оператора **Do** использует ключевое слово **While**, VBA выполняет цикл, пока логическое выражение, представленное с помощью *Condition*, равно **True**.

При выполнении оператора **Do...Loop While** VBA сначала выполняет операторы, представленные с помощью *Statements*. Когда VBA достигает ключевых



слов **Loop While**, выполняется тестирование логического выражения, представленного с помощью *Condition*; если это выражение равно **True**, VBA возвращается в начало цикла и снова выполняет тело цикла. Когда VBA снова достигает ключевых слов **Loop While** в конце цикла, снова выполняется проверка, является ли логическое выражение *Condition* все еще равным **True**. Если *Condition* равно **True**, VBA выполняет цикл снова; если — нет, VBA продолжает выполнение кода с любых операторов после строки, содержащей ключевое слово **Loop**, т.е. прекращает работу цикла.

Заметьте, что независимо от значения логического выражения, представленного с помощью выражения *Condition*, этот цикл всегда выполняется, по крайней мере, один раз. (Сравните с циклами, проверяющими условие детерминанта до выполнения тела цикла.)

Процедура **Test\_DoLoopWhile** в листинге 7.9 использует конструкцию цикла **Do...Loop While** для получения в диалоговом окне данных от пользователя до тех пор, пока пользователь не введет слово «*exit*», указывая на то, что ввод данных завершен.

#### Листинг 7.9. Пример использования структуры **Do...Loop While**

```
1: Sub Test_DoLoopWhile()  
2: 'демонстрирует структуру Do...Loop While. Цикл  
3: 'повторяется, пока не будет введено слово "exit"  
4:  
5:     Const iTitle = "Ввод данных"  
6:     Dim uStr As String  
7:  
8:     Do  
9:         uStr = InputBox(prompt:="Введите строку " & _  
10:            "('exit' для окончания):", _  
11:            Title:=iTitle, _  
12:            Default:="exit")  
13:         'операторы обработки или проверки вводимых данных  
14:         'или сохранения данных в ячейках рабочего листа и т.д.  
15:         MsgBox prompt:="Вы ввели: " & uStr, Title:=iTitle  
16:     Loop While uStr <> "exit"  
17:  
18:     MsgBox prompt:="Ввод данных завершен.", _  
19:         Title:=iTitle, buttons:=vbExclamation  
20: End Sub
```

По-видимому, комментировать здесь нечего. К моменту рассмотрения этого листинга вы должны иметь знания, достаточные для того, чтобы понять представленный код.

#### Циклы **Do...Loop Until**

Для построения **Do**-цикла, тестирующего свое условие детерминанта после выполнения тела цикла можно также использовать форму **Do...Loop Until** оператора **Do**.

Оператор **Do...Loop Until** имеет следующий синтаксис:

### Синтаксис

---

```
Do
    Statements
Loop Until Condition
```

*Statements* представляет операторы VBA, составляющие тело цикла; *Condition* — логическое выражение для детерминанта цикла (см. диаграмму синтаксиса цикла **Do...Loop While**).

---

VBA проверяет условие детерминанта цикла **Do...Loop Until** после выполнения тела цикла. Поскольку эта форма **Do**-оператора использует ключевое слово **Until**, VBA выполняет цикл, пока логическое выражение, представленное с помощью *Condition*, равно **False**.

При выполнении оператора **Do...Loop Until** VBA сначала выполняет операторы, представленные с помощью *Statements*. Когда VBA достигает ключевого слова **Loop**, выполняется тестирование логического выражения, представленного с помощью *Condition*; если логическое выражение равно **False**, VBA возвращается в начало цикла и снова выполняет тело цикла. Когда VBA снова достигает ключевого слова **Loop** в конце цикла, снова выполняется проверка, является ли логическое выражение *Condition* равным **False**. Если *Condition* равно **False**, VBA выполняет цикл снова; если оно равно **True**, VBA продолжает выполнение кода с любых операторов после строки, содержащей ключевое слово **Loop**, т.е. прекращает работу цикла.

Заметьте, что этот цикл всегда выполняется, по крайней мере, один раз, независимо от значения логического выражения, представленного с помощью *Condition*. (Сравните с циклами, проверяющими условие детерминанта до выполнения тела цикла.)

Процедура **Test\_DoLoopWhile** в листинге 7.10 использует конструкцию цикла **Do...Loop While** для получения данных ввода от пользователя до тех пор, пока пользователь не введет слово «*exit*», указывая на то, что ввод данных завершен.

### Листинг 7.10. Пример использования структуры **Do...Loop Until**

---

```
1: Sub Test_DoLoopUntil()
2:     'демонстрирует структуру Do...Loop Until. Цикл
3:     'повторяется, пока не будет введено слово "exit"
4:
5:     Const iTitle = "Ввод данных"
6:     Dim uStr As String
7:
8:     Do
9:         uStr = InputBox(prompt:="Введите строку " & _
10:            " ('exit' для окончания):", _
11:            Title:=iTitle, _
12:            Default:="exit")
13:         'операторы обработки или проверки вводимых данных
14:         'или сохранения данных в ячейках рабочего листа и т.д.
15:         MsgBox prompt:="Вы ввели: " & uStr, Title:=iTitle
```

```
16: Loop Until uStr = "exit"
17:
18: MsgBox prompt:="Ввод данных завершен.", _
19: Title:=iTitle, buttons:=vbExclamation
20: End Sub
```

---

Код этого листинга практически ничем не отличается от кода предыдущего листинга, кроме того, что здесь используется структура **Do...Loop Until**, а не **Do...Loop While** (строка 16).

## Вложенные циклы

Можно помещать циклы внутри других циклов, аналогично тому, как можно помещать операторы **If...Then** один в другой. Помещение одной структуры цикла в другую называют *вложением (nesting)* циклов. Можно помещать структуры циклов любого типа (смешанные **For** и **Do** циклы) в любой уровень.

- При вложении циклов необходимо соблюдать следующие правила:
- При вложении циклов **For...Next** каждый цикл должен иметь свою уникальную переменную счетчика.
- При вложении циклов **For Each...Next** каждый цикл должен иметь свою уникальную *element*-переменную.
- Если вы используете оператор **Exit For** или **Exit Do** во вложенном цикле, этим оператором заканчивается только выполняемый в данный момент цикл; VBA продолжает выполнение следующего цикла более высокого уровня.

### Вложение циклов For

В листинге 7.11 показана простая процедура, которая использует вложенные циклы **For...Next**. Процедура **Test\_ForNext** предназначена для использования в Excel и применяет полужирный шрифт к первым *n* (вводится пользователем) строкам первых *m* (вводится пользователем) столбцов активного в данный момент рабочего листа. Внешний цикл в процедуре считает строки, а внутренний — столбцы.

#### Листинг 7.11. Вложенные циклы **For...Next**

---

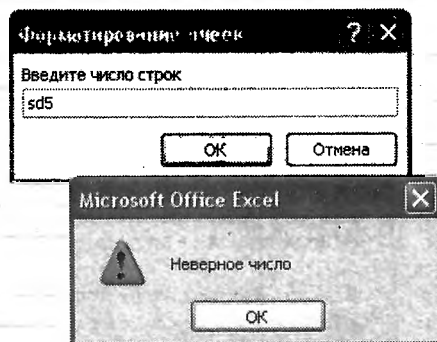
```
1: Sub Test_ForNext()
2: 'Применяет шрифт Arial, полужирный
3: 'к первым n строкам и m столбцам активной рабочей книги
4: Dim sTitle As String
5: Dim Rnum As Integer, Cnum As Integer
6:
7: Dim n As Integer, m As Integer
8: sTitle = "Форматирование ячеек листа"
9:
10: n = Application.InputBox(prompt:="Введите число строк", _
11: Title:="Форматирование ячеек", Type:=1)
12: If n = 0 Then Exit Sub
13:
14: m = Application.InputBox(prompt:="Введите число столбцов", _
```

```
15:                                     Title:="Форматирование ячеек", Type:=1)
16:   If m = 0 Then Exit Sub
17:
18:   Worksheets("Sheet1").Activate
19:   For Rnum = 1 To n
20:     For Cnum = 1 To m
21:
22:       Cells(Rnum, Cnum).Select
23:
24:       With Selection.Font
25:         .Name = "Arial"
26:         .FontStyle = "Bold"
27:         .Strikethrough = False
28:         .Superscript = False
29:         .Subscript = False
30:         .OutlineFont = False
31:         .Shadow = False
32:         .Underline = xlNone
33:         .ColorIndex = xlAutomatic
34:       End With
35:
36:     Next Cnum
37:   Next Rnum
38: End Sub
```

Процедура **Test\_ForNext** использует циклы **For...Next**, один, вложенный в другой, для применения полужирного шрифта Arial к ячейкам листа, начиная с первой и заканчивая в строке **n** и столбце **m**. Значения **n** и **m** вводятся в строках 10–11, 14–15 при помощи метода **Application.InputBox**, который, как уже отмечалось, позволяет сократить проверочный код пользователя использованием необязательного аргумента **Type**. При значении **Type**, равном 1, пользователь не сможет ввести в окне ввода ничего, кроме числа. Вы можете сами в этом убедиться. Набор чего-либо, не похожего на число, приведет к выдаче окна сообщения, подобного приведенному на рис. 7.3.

**Рис. 7.3**

Excel-метод **Application.InputBox** не позволяет вводить ничего, кроме того, что определено его необязательным аргументом **Type**



Кроме контроля за вводом, в процедуре проверяется, не введены ли нулевые или отрицательные значения для **n** и **m** (строки 12, 16). Работа процедуры продолжается только при введении положительных значений этих переменных.

Счетчик внешнего цикла **For...Next** изменяется от 1 до **n**, и процедура использует переменную счетчика внешнего цикла для предоставления координат строки каждый раз при выполнении цикла. Следовательно, внешний цикл обрабатывает **n** строк от начала рабочего листа.

Счетчик внутреннего цикла **For...Next** изменяется от 1 до **m**, и процедура использует переменную счетчика внутреннего цикла для предоставления координат столбца каждый раз при выполнении цикла. Следовательно, внутренний цикл обрабатывает в рабочем листе **m** столбцов слева направо.

В строке 22 используется метод **Cells** объекта **Application** Excel для возвращения объекта **Range**, определенного с использованием **Rnum** и **Cnum** в качестве координат строки и столбца. Оператор использует также метод **Select** объекта **Range** для выбора заданной ячейки в активном рабочем листе. Оператор **Cells** был скопирован из записанного рекордером макроса, который содержал только два действия: выбор ячейки и применение нужного форматирования. Оператор выбора ячейки был скопирован в эту процедуру, а литеральные значения из записанного рекордером оператора были заменены переменными **Rnum** и **Cnum**.

В строках 24–34 расположен оператор **With**, используемый для текущего выделенного фрагмента (ячейки). Объект **Selection.Font** содержит все установки шрифта для текущего выбора. Операторы внутри **With** (в строках 25–33) изменяют свойства объекта **Selection.Font** так, чтобы выбор имел нужный полужирный шрифт Arial. Эти операторы были также скопированы из записанного рекордером макроса.

## Вложенные циклы Do

В листинге 7.12 показана процедура, использующая вложенные циклы **Do**, чтобы пользователь мог выполнить ввод данных, необходимых для получения накладной. Внешний цикл в процедуре **MakeInv** получает номера накладных от пользователя до тех пор, пока пользователь не отменит окно ввода. Внутренний цикл в процедуре **MakeInv** получает элементы записей накладной до тех пор, пока пользователь не отменит окно ввода. Когда вы будете изучать эту процедуру, помните, что это просто эскизная процедура. На самом деле вам, вероятно, потребуется получить гораздо больше информации, чем используется здесь. Вам, конечно, предстоит также самим писать код для занесения вводимой информации в какие-либо хранилища данных. (Листинг 7.12 будет выполняться и в Excel, и в Word).

### Листинг 7.12. Вложенные циклы Do

```
1: Sub MakeInv()  
2: 'обеспечивает ввод данных для накладных и элементы  
3: 'записей для каждой накладной. Внешний цикл получает  
4: 'информацию об отдельной накладной, внутренний -  
5: 'о записях этой накладной (наименование и количество  
6: 'товара)  
7:  
8:     Const miTitle = ""  
9:  
10:    Dim InvNum As String * 5    'номер накладной  
11:    Dim ItemNum1 As String * 16  
12:    Dim ItemNum2 As String * 16
```

```
13: Dim ItemDone As Boolean
14: Dim InvcDone As Boolean
15: Dim pos As Integer
16:
17:
18: InvcDone = False
19: Do Until False
20:     InvcNum = InputBox(prompt:="Введите номер накладной:",
21:         Title:=miTitle)
22:     If Trim(InvcNum) = "" Then
23:         MsgBox prompt:="Отмена ввода накладной.",
24:             Title:="Ввод накладных", Buttons:=vbExclamation
25:         Exit Do
26:     End If
27:
28:     pos = 1
29:     Do Until False 'цикл для получения элемента строки
30:
31:         'Ввод наименования товара
32:         ItemNum1 = InputBox(prompt:=
33:             "Введите наименование товара",
34:             Title:="Накладная №" & InvcNum & "Позиция №" & pos)
35:         If Trim(ItemNum1) = "" Then
36:             MsgBox prompt:="Ввод позиций для накладной №" &
37:                 InvcNum & " завершен.",
38:                 Title:=miTitle
39:             Exit Do
40:         End If
41:
42:         'ввод количества товара
43:         ItemNum2 = InputBox(prompt:=
44:             "Введите количество товара",
45:             Title:="Накладная №" & InvcNum & "Позиция №" & pos)
46:         If Trim(ItemNum2) = "" Then
47:             MsgBox prompt:="Ввод позиций для накладной №" &
48:                 InvcNum & " завершен.",
49:                 Title:=miTitle
50:             Exit Do
51:         End If
52:
53:         'операторы для сохранения вводимых данных, получения
54:         'большей информации об элементе записи и т.д.
55:         MsgBox prompt:="Введенный элемент №" & pos & " " &
56:             ItemNum1 & " кол-во " & ItemNum2 &
57:             " в накладной №" & InvcNum,
58:             Title:="Контрольная выдача "
59:
60:         pos = pos + 1
61:     Loop
62: Loop
63:
64: MsgBox prompt:="Ввод накладной завершен", Title:=miTitle
65: End Sub
```

Процедура **MakeInv** демонстрирует применение вложенных **Do**-циклов на примере имитации ввода накладных. В процедуре использованы бесконечные циклы (строки 19, 29), выход из которых осуществляется оператором **Exit Do** (строки 25, 50). Накладная имеет номер и записи, включающие наименование товара и количество.

Номера накладных вводятся во внешнем цикле (начинается в строке 19). Сразу после ввода накладной производится проверка необходимости выхода из цикла (строки 22–26). Если пользователь не ввел номер или нажал кнопку **Cancel**, внешний цикл прекращается с предварительным сообщением об отмене ввода накладной (строки 23–24 ).

Перед началом выполнения внутреннего цикла переменной **pos** (счетчик количества позиций в накладной) присваивается значение 1 (строка 28). В конце внутреннего цикла (строка 60) значение счетчика увеличивается на единицу. В теле внутреннего цикла вводятся наименования (строки 32–34) и количества (строки 43–45) товаров. После ввода каждого из этих элементов производится проверка ввода (строки 35–40, 46–50), после которой выполнение внутреннего цикла может быть завершено операторами **Exit Do** (строки 39, 50). Если наименование и количество товаров введены успешно, в конце внутреннего цикла (строки 55–58) производится контрольная выдача введенных данных на экран.

## Пример использования коллекции в Word

Предыдущая глава была посвящена объектам и коллекциям приложений. Здесь мы рассмотрим пример использования коллекции **Selection** в Word (текущая выделенная часть) с применением цикла **For...Next**.

Для выделения в тексте данной книги наименований элементов управления диалоговых окон используется шрифт Arial, Bold. Например, кнопка **Cancel**. При первой верстке оказалось, что на фоне основного текста такой шрифт слишком велик. Правильнее было бы уменьшить его на один пункт. Так возникла задача — написать макрос, который должен уменьшить размер шрифта для слов, отформатированных при помощи Arial, Bold.

В листинге 7.13 приведен код процедуры, которая (в строке 6) определяет количество выделенных абзацев, а затем в цикле **For...Next** (строки 9–22) для каждого абзаца проверяет его стиль (свойство **Selection.Paragraphs(i).Style**). Если наименование стиля совпадает с «Обычный», «Рис\_текст» и др., то во вложенном цикле (строки 15–20) для каждого слова абзаца проверяется имя фонта. При совпадении его с Arial свойство **Font.Size** уменьшается на единицу (строка 17).

### Листинг 7.13. Форматирование отдельных слов документа

```
1: Sub ЗаменаArial()  
2: 'уменьшает размер слова, отформатированного как Aria, Bold  
3: Dim ZZ  
4:  
5: 'количество выделенных абзацев:  
6: ii = Selection.Paragraphs.Count()  
7:  
8: 'цикл по абзацам:  
9: For i = 1 To ii  
10: If Selection.Paragraphs(i).Style = "Обычный" Or  
11: Selection.Paragraphs(i).Style = "Рис_текст" Or _
```

```
12:           Selection.Paragraphs(i).Style = "Рис номер"
13:           Or Selection.Paragraphs(i).Style = "Основной текст"
14:           Or Selection.Paragraphs(i).Style = "No" Then
15:       For j = 1 To Selection.Paragraphs(i).Range.Words.Count
16:           If Selection.Paragraphs(i).Range.Words(j).Font.Name =
                                   "Arial" Then
17:               ZZ =
                   Selection.Paragraphs(i).Range.Words(j).Font.Size - 1
18:               Selection.Paragraphs(i).Range.Words(j).Font.Size = ZZ
19:           End If
20:       Next
21:   End If
22: Next
23:
24: End Sub
```

---

## Массивы

*Массив (array)* — это коллекция переменных, которые имеют общие имя и базовый тип. Массив является удобным способом хранения нескольких связанных элементов данных в едином контейнере для большего удобства и эффективности программирования. Все элементы данных, сохраняемых в массиве, должны иметь один и тот же тип; например, при создании массива для хранения типов **Integer** (**Double**, **String**, **Currency** и т.д.) все элементы данных, сохраненных в этом массиве, должны быть числами **Integer** (**Double**, **String**, **Currency** и т.д.).

Массив позволяет сохранять и манипулировать многими элементами данных посредством единственной переменной. Кроме уменьшения общего числа различных имен переменных, которые необходимо отслеживать, другим основным преимуществом использования массивов является то, что можно использовать циклы для легкой обработки различных элементов массивов. Объединяя массивы и структуры цикла (обычно **For...Next** или **For...Each**), можно написать небольшое число операторов, которые обрабатывают большой объем данных. Выполнение тех же задач с использованием отдельных переменных может потребовать написания сотен операторов.

## Размерность массива

В языках программирования обычно используются одномерные и многомерные массивы, одни из которых описывают относительно простые, а другие — более сложные объекты.

### Одномерные массивы

Наименее сложный массив — это просто список элементов данных; такого рода массив называется *простым (simple)* или *одномерным (single-dimensional)* массивом. Такой массив можно представить в виде таблицы (рис. 7.4). Каждый элемент данных, хранимых в массиве, называется *элементом (element)* массива. Массив на рис. 7.4 имеет 8 элементов; каждый элемент сохраняет число типа **Double**. Заметьте, что элементы в массиве пронумерованы от 0 до 7, что составляет 8 элементов. Такая система нумерации довольно распространена в программировании и называется нумерацией *с нулевой базой (zero-based)*.



Для доступа к данным, хранящимся в определенном элементе массива, следует указывать имя массива с последующим числом, называемым *индексом* (*subscript* или *index*) элемента. Индекс всегда заключается в круглые скобки. Например, если массив на рис. 7.4 имеет имя **DoubleArray**, то следующий оператор присваивает число 0.11 переменной **DoubleAny**:

```
DoubleAny = DoubleArray (6)
```

В этом операторе число 6 является индексом массива; заметьте, что он заключен в круглые скобки и не отделяется пробелами от имени массива. Поскольку нумерация элементов начинается с нуля, элемент, на который ссылается этот оператор, является, фактически, седьмым элементом массива **DoubleArray**.

10.2	11.2	22.1	1.1	21.3	123.0	0.11	1.1
0-й элемент	1-й элемент	2-й элемент	3-й элемент	4-й элемент	5-й элемент	6-й элемент	7-й элемент

**Рис. 7.4.** Одномерный числовой массив; одномерные массивы — это, в основном, просто списки данных одного и того же типа

Посмотрите снова на рис. 7.4 и обратите внимание, что элемент массива с номером 6 содержит значение 0.11. При выполнении приведенного выше оператора VBA выбирает значение 0.11 из указанного элемента массива и сохраняет это значение в переменной **DoubleAny** — точно так же, как в любом другом присваивании переменной.

Можно также использовать индекс всякий раз, когда необходимо сохранить данные в отдельном элементе массива. Например, следующий оператор сохраняет число 12.3 в восьмом элементе массива, показанном на рис. 7.4:

```
DoubleArray (7) = 12.3
```

При выполнении этого оператора VBA помещает значение 12.3 в указанный элемент массива, заменяя предыдущее содержимое этого элемента — точно так же, как в любом другом присваивании переменной. Можно использовать элемент массива в любом выражении VBA — точно так же, как используется значение константы или переменной в каком-либо выражении.

Одномерные массивы обычно используются для представления различных списков данных.

**Многомерные массивы**

Одномерные массивы хорошо подходят для представления простых списков данных. Однако часто бывает необходимо представить таблицы данных в программах с организацией данных в формате строк и столбцов, подобно ячейкам в рабочих листах Excel. Для этого необходимо использовать *многомерные* (*multi-dimensional*) массивы. Поскольку вы уже имеете опыт работы с ячейками Excel, то нет необходимости в подробном разъяснении сущности двумерного массива. Адрес каждой ячейки листа состоит из двух чисел (измерений), одно из которых (номер строки) является первым индексом, а второе (номер столбца) — вторым индексом массива. На примере организации листов Excel

можно представить и трехмерные массивы. Здесь третьим индексом массива может быть номер листа.

В VBA можно также создавать массивы, имеющие более трех измерений; фактически, VBA позволяет создавать массивы, имеющие до 60 измерений. Работа с массивами, имеющими 4 или более измерений, быстро становится запутанной; к счастью, вам, вероятно, не придется работать с такими массивами. Чаще всего в программировании используются одно- и двумерные массивы.

## Статические и динамические массивы

В следующем разделе вы узнаете, что число элементов в массиве обычно задается во время объявления массива. Объявление массива указывает VBA величину различных измерений массива (диапазон изменения каждого индекса). После того как массив объявлен, VBA выделяет достаточный объем памяти для всех элементов массива. Например, для массива на рис. 7.4 VBA разделит объем памяти, достаточный для 8 чисел типа **Double**.

Переменные типа массив подчиняются тем же правилам области действия, что и любые другие переменные.

VBA сохраняет зарезервированной область памяти для всех элементов в массиве, пока существует переменная типа массив. Подобные массивы называются *статическими* (*static*), потому что число элементов в массиве не меняется.

Выбор размера массива может быть затруднен, если неизвестно, сколько данных будет введено в массив, или если объем данных, собираемых для массива, значительно меняется. Если вам иногда приходится сохранять 100 значений, а иногда — только 10 значений, то потенциально напрасно расходуется область памяти, необходимая для сохранения 90 значений.

Для подобных ситуаций VBA поддерживает особый тип массивов, называемый *динамическим* (*dynamic*) *массивом*. Динамические массивы получили свое название, потому что можно изменять число элементов в массиве при выполнении VBA-программы. Динамический массив (в сочетании с правильным программированием) может увеличиваться или сжиматься (уменьшаться в размере), чтобы вмещать точно необходимое число элементов без напрасного расходования памяти. Для изменения размера динамического массива используйте оператор **ReDim**, описываемый далее в этой главе.

## Оператор Option Base

Обычно в VBA используются массивы с нулевой базой. В системе нумерации с нулевой базой индекс для первого элемента в любом измерении массива является равным 0; массив с 10 элементами имеет индексы от 0 до 9. Очевидно, что система нумерации с нулевой базой может быть непонятной, потому что индекс 0, в действительности, обозначает 1-й элемент массива, индекс 5 обозначает 6-й элемент массива и так далее.

Было бы гораздо удобнее, если бы элементы массива нумеровались, начиная с 1, а не с 0. Если бы нумерация элементов начиналась с 1, то индекс 1 обозначал бы 1-й элемент массива, индекс 5 — 5-ый и так далее.

VBA позволяет задавать начальное число для элементов массива. Можно задавать нижнее число для индексов массива при объявлении массива (описывается далее в этой главе), или использовать директиву компилятора **Option Base** для указания того, должна ли нумерация индексов начинаться с 0 или с 1.

Директива компилятора **Option Base** имеет следующий синтаксис:

### Синтаксис

---

`Option Base 0 | 1`

---

Оператор **Option Base** позволяет задавать 0 или 1 как начальное число по умолчанию для индексов массива. Если оператор **Option Base** не используется, VBA начинает нумерацию индексов массива с 0 (по умолчанию). Необходимо помещать оператор **Option Base** в область объявлений модуля перед объявлениями любых переменных, констант или процедур. Нельзя помещать оператор **Option Base** внутри процедуры. Можно иметь только один оператор **Option Base** в модуле. Оператор **Option Base** влияет на все массивы, объявляемые в модуле, независимо от того, являются ли они локальными в процедуре или объявляются на модульном уровне.

Следующие два оператора являются примерами директивы компилятора **Option Base**:

<code>Option Base 0</code>	'установка по умолчанию
<code>Option Base 1</code>	'индексы массивов начинаются с 1

### Объявление массивов

Вы уже знакомы с оператором **Dim**, используемым для объявления переменных. Этот же оператор используется и для объявления массивов. В действительности, ключевое слово **Dim** является сокращением слова *dimension* (измерение). В оригинальном языке программирования BASIC ключевое слово **Dim** использовалось исключительно для изменения размеров массивов (*dimensioning*), отсюда и сокращение **Dim**. Однако современный язык VBA расширил использование ключевого слова **Dim** до использования со всеми переменными. С помощью **Dim** можно объявлять как одномерные, так и многомерные массивы.

Объявление массива с помощью оператора **Dim** имеет следующий синтаксис:

### Синтаксис

---

`Dim VarName([Subscripts]) [As Type]`

*VarName* — любое имя для массива, удовлетворяющее VBA-правилам для имен идентификаторов. *Subscripts* — измерение (измерения) массива. Можно объявлять массивы, имеющие до 60 измерений. Для одномерного массива включается один *Subscripts*; для двумерного массива — два (отделенные друг от друга запятой) и так далее. Каждый *Subscripts* добавляет новое измерение в массив.

---

Можно также объявлять статические и динамические массивы, используя ключевые слова **Public**, **Private** и **Static** — точно так же, как для любой другой переменной и с тем же влиянием на область действия. Используйте показанный здесь синтаксис объявления массива и просто подставляйте ключевое слово **Public**, **Private** или **Static** вместо ключевого слова **Dim**, если необходимо.

Оператор *Subscripts* имеет следующий синтаксис:

### Синтаксис

---

```
[lower To] upper [, [lower To] upper]...
```

Здесь *lower* определяет нижний диапазон допустимых индексов для массива; *upper* — верхний предел. Заметьте, что только верхний предел является обязательным; часть *lower To* оператора *Subscripts* является необязательной. При определении только предела *upper* VBA нумерует элементы массива в зависимости от установки **Option Base**. Если действует установка **Option Base 1**, VBA нумерует элементы в массиве от 1 до *upper*; иначе — от 0 до *upper*.

---

Включение части *lower To* оператора *Subscripts* помогает сделать код более легким и понятным, а также выявить ошибки программирования. Это также позволяет определять иной начальный индекс для массива, чем 0 или 1. Например, можно создать массив с элементами, имеющими номера от 5 до 10 или от -5 до 0, в зависимости от конкретной выполняемой задачи.

Подобно обычным объявлениям переменных, можно объявлять определенный тип данных для массива, включая в объявление оператор **As type**. При этом *type* представляет любой допустимый тип VBA — **Currency**, **Double**, **String** и так далее. Можно также объявлять массивы, имеющие определенный пользователем тип (создание пользовательских типов в книге не рассматривается). Если опустить *type*, все элементы в массиве имеют тип **Variant**. VBA инициализирует элементы числовых массивов нулями и элементы строковых массивов пустыми строками.

Заметьте, что оператор *Subscripts* является необязательным. Для создания динамического массива не используйте оператор *Subscripts* (необходимо включать круглые скобки в объявление массива, независимо от того, определяется ли *Subscripts*).

Следующие примеры являются допустимыми объявлениями массива:

```
Dim str_array(1 To 100) As String
Dim variant_array()
Dim str_Multiplication(0 To 15, 0 To 15) As String
```

При объявлении массивов следует помнить, что включение оператора *Subscripts* в объявление массива создает статический массив с фиксированным числом элементов, пропуск оператора *Subscripts* в объявлении массива создает динамический массив, а установка **Option Base** может повлиять на общее число элементов в массиве.

### Использование массивов

После объявления массива использовать его в коде VBA довольно просто. Как уже объяснялось в начале этой главы, для доступа к элементу массива необходимо указать имя массива, за которым следует значение индекса, заключенное в круглые скобки.

Обращение к элементу массива имеет следующий синтаксис:

## Синтаксис

---

```
arrayName(validIndex1, [validIndex2]...)
```

Здесь *arrayName* — имя массива; *validIndex1* — допустимое значение индекса для первого измерения массива; *validIndex2* — допустимое значение индекса для второго измерения массива, если таковое имеется. Необходимо предоставлять значение индекса для каждого измерения массива при каждом обращении к какому-либо элементу в массиве.

---

Например, для двумерного массива необходимо всегда определять два индекса. Допустимым индексом является любая переменная VBA или выражение, имеющее результатом целое число в диапазоне объявленных измерений массива. Например, допустимым значением индекса для одномерного массива, объявленного с индексами 1–10, может быть любое выражение VBA, имеющее результатом целое число в диапазоне от 1 до 10. Использование меньшего или большего индекса, чем диапазон для определенного измерения в массиве, приводит к тому, что VBA отображает runtime-ошибку.

Следующий фрагмент кода показывает типичное объявление и использование массива.

```
Dim Factorial(0 To 30) As Double      'объявление массива из 31 элемента
Factorial(0) = 1                     'инициализировать 1-й элемент
For i = 1 To 30                       'начало цикла для работы с массивом
    Factorial(i) = i * Factorial(i - 1) 'изменить i-й элемент
Next i
```

Рассмотрим простой пример работы с одномерным массивом, представленный кодом листинга 7.14. Процедура в первом цикле **For...Next** принимает в диалоговом окне целые числа (для простоты без всякой проверки) и заносит их в элементы массива. Во втором цикле значения элементов массива заносятся в строку и выводятся на экран в диалоговом окне оператора **MsgBox**.

### Листинг 7.14. Работа со статическим одномерным массивом

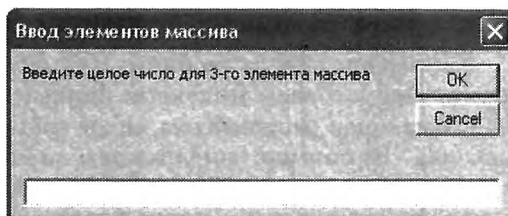
---

```
1: Sub DemoStatArray()
2:   Dim Int_Array(5) As Integer
3:   Dim str_msg As String
4:
5:   str_msg = ""
6:
7:   For i = 1 To 5
8:     Int_Array(i) = InputBox("Введите целое число для " &
9:       & i & "-го элемента массива", _
10:      "Ввод элементов массива")
11:   Next
12:
13:   For j = 1 To 5
14:     str_msg = str_msg & Int_Array(j) & ", "
15:   Next
16:
17:   MsgBox "Введено: " & str_msg, _
18:     , "Вывод ранее введенного массива"
19:
20: End Sub
```

---

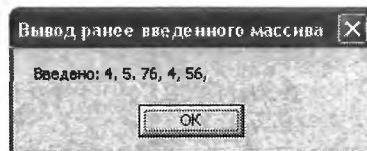
В строке 2 объявляется одномерный массив **Int\_Array** (размерностью 5 элементов) целых чисел **Int\_Array**. Далее (строка 3) объявлена рабочая переменная **str\_msg** для завершающего вывода данных из массива в диалоговом окне (строка 19). В первом цикле **For...Next** (строки 7–11) в массив **Int\_Array** записываются значения из окна ввода. Обратите внимание, как формируется значение аргумента *Prompt* функции **InputBox**. На рис. 7.5 приведен фрагмент ввода 3-го элемента массива.

**Рис. 7.5**  
Фрагмент ввода 3-го элемента массива



В строках 13–15 располагается второй цикл, который формирует строку из значений элементов массива для выдачи ее в диалоговом окне (рис. 7.6).

**Рис. 7.6**  
Результат работы кода листинга 7.14



Процедура **DemoStatArray2** в листинге 7.15 предназначена для демонстрации работы с двумерным массивом.

#### Листинг 7.15. Работа со статическим двумерным массивом

```

1: Sub DemoStatArray2()
2:   Dim Int_Array(3, 6) As Integer
3:   Dim str_msg As String
4:
5:   str_msg = ""
6:
7:   For i = 1 To 3
8:     For j = 1 To 6
9:       Int_Array(i,j)=InputBox("Введите целое число для "
10:      & " элемента (" & i & ", " & j & ")",
11:      "Ввод элементов массива; строка " & i)
12:     Next j
13:   Next i
14:
15:   For i = 1 To 3
16:     For j = 1 To 6
17:       str_msg = str_msg & Int_Array(i, j) & ", "
18:     Next j
19:
20:     str_msg = str_msg & Chr(13) 'перевод строки
21:   Next i

```

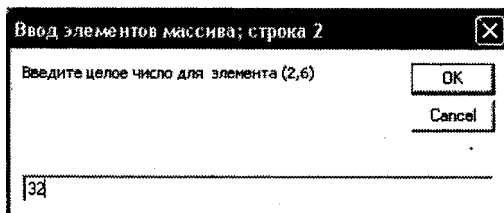
```
22:
23:   MsgBox "Введено: " & Chr(13) & str_msg, _
24:         , "Вывод ранее введенного массива"
25:
26: End Sub
```

Для инициализации массива используется вложенный цикл в строках 7–13. Обратите внимание на строки 12–13. Здесь, как ранее отмечалось, указывается, по какой переменной заканчивается цикл. Это необязательно, но лучше привыкнуть к такому стилю программирования с самого начала.

В окне ввода теперь для указания, какой элемент вводится в конкретный момент времени, выводятся оба индекса (рис. 7.7). Это уже довольно дружелюбный интерфейс для ввода двумерного массива: пользователю трудно не догадаться, чего от него «добивается» программа. Чтобы понять, как это достигается, внимательно проанализируйте оператор в строках 9–11.

**Рис. 7.7**

Дружелюбный интерфейс для ввода элемента двумерного массива

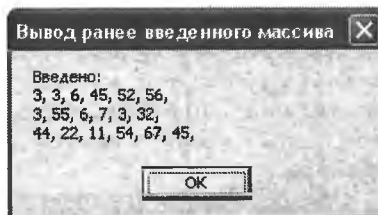


Вторая часть процедуры **DemoStatArray2** также состоит из вложенных циклов и, как и в предыдущем листинге, эта часть предназначена для выдачи инициализированного массива в окне **MsgBox** при помощи переменной **str\_msg**. Для осуществления этой задачи во внутреннем цикле (строки 16–18) формируется строка из элементов массива при постоянном первом индексе (переменная **i**). При этом меняется только второй индекс (переменная **j**). После заполнения строки, но перед изменением первого индекса к переменной **str\_msg** добавляется символ перехода на другую строку (строка 20).

Результат работы кода листинга 7.15 приведен на рис. 7.8.

**Рис. 7.8**

В результате работы кода листинга 7.15 инициализирован массив размерностью 3x6



## Использование ReDim с динамическими массивами

Как упоминалось ранее в этой главе, могут сложиться обстоятельства, при которых точно не известно, сколько элементов потребуется хранить в массиве. В VBA имеется возможность переопределять размерность массивов, а во время объявления не указывать размерность. Используя динамический массив, можно создавать массив такой большой или такой маленький, какой необходимо.

Динамические массивы создаются с помощью оператора **Dim**, затем их размер устанавливается с помощью оператора **ReDim** во время выполнения процедуры.

Оператор **ReDim** имеет следующий синтаксис:

### Синтаксис

---

```
ReDim [Preserve] varname(subscripts) [As type] [,
varname(subscripts) [As type]]
```

---

Необязательное ключевое слово **Preserve**, как предполагает его имя, приводит к тому, что VBA сохраняет данные в имеющемся массиве, когда изменяется размер массива с помощью **ReDim**; *varname* — имя существующего массива; *subscripts* — это измерения массива (синтаксис для оператора *subscripts* в операторе **ReDim** такой же, как для оператора **Dim**); *type* — любой тип VBA или определенный пользователем тип. Необходимо использовать отдельный оператор **As type** для каждого массива, который вы определяете.

---

Далее следуют допустимые примеры объявлений динамических массивов и возможные операторы **ReDim**, используемые с этими динамическими массивами:

Dim aMonth() As String	'объявляет динамический массив aMonth
ReDim aMonth(1 To 30)	'изменяет размер массива до 30 элементов
ReDim aMonth(31)	'изменяет размер массива до 31 элемента
ReDim Preserve aMonth(1 To 31)	'изменяет размер массива до 31
	'элемента, сохраняя содержимое
Dim Table() As Integer	'объявляет динамический массив
ReDim Table(3, 15)	'делает массив двумерным
ReDim Table(4, 20)	'изменяет размер двумерного массива
ReDim Preserve Table(4, 25)	'только изменяет последний размер
массива	
Dim Mas as Variant	'объявляет переменную типа Variant
ReDim Mas(20) As Integer	'создает массив 20 целых в Variant

Предыдущие примеры иллюстрируют некоторые важные моменты, относящиеся к динамическим массивам. Во-первых, можно изменять только последнее измерение многомерного массива, когда используется ключевое слово **Preserve**. Во-вторых, можно использовать **ReDim** для создания типизированного массива внутри переменной типа **Variant**. Поскольку переменные типа **Variant** могут содержать данные любого типа, можно использовать переменную типа **Variant** для сохранения динамического массива! (Использование переменной типа **Variant** для сохранения динамического массива дает возможность изменять размер массива с помощью **ReDim** и изменять тип данных массива.)

Обычно оператор **ReDim** используется для изменения размера динамического массива, который уже был объявлен ранее с помощью операторов **Dim**, **Private**, **Public** или **Static**. Можно использовать оператор **ReDim** для изменения числа элементов и измерений в динамическом массиве столько раз, сколько необходимо. Однако нельзя использовать оператор **ReDim** для изменения типа данных массива, если только массив не содержится в переменной типа **Variant** или сами элементы массива не имеют тип **Variant**. Если динамический массив сохраняется в переменной типа **Variant**, можно изменять тип данных, используя оператор **As type** в операторе **ReDim**.



### Замечание

Попытка использовать **ReDim** для статического массива (массив, измерения которого были явно определены в объявлении **Dim**, **Public**, **Private** или **Static**) приводит к тому, что VBA отображает runtime-ошибку.

Код листинга 7.16 является примером использования динамического одномерного массива и выполняет задачу ввода данных в массив, подобную решаемой в листинге 7.14. Это делается для того, чтобы концентрировать внимание читателя на отличиях использования разных типов массивов, а не деталях различных задач.

### Листинг 7.16. Работа с динамическим одномерным массивом

```
1: Option Base 1
2: Sub DemoDinArray()
3:     Dim Int_Array() As Integer
4:     Dim str_msg As String, i As Integer
5:     Dim Num
6:
7:     str_msg = ""
8:     i = 0
9:
10:    Do
11:        i = i + 1
12:        Num = InputBox("Введите целое число для " & i & "-го элемента массива", _
13:            "Ввод элементов массива")
14:        If Len(Num) = 0 Then Exit Do 'выход из цикла
15:
16:        'изменение размера массива
17:        'с сохранением элементов
18:        ReDim Preserve Int_Array(i)
19:
20:        'ввод данных в i-й элемент массива
21:        Int_Array(i) = Num
22:    Loop
23:
24:
25:    For j = 1 To i - 1
26:        str_msg = str_msg & Int_Array(j) & ", "
27:    Next
28:
29:    MsgBox "Введено: " & str_msg, _
30:        , "Вывод ранее введенного массива"
31:
32: End Sub
```

В строке 3 объявлен массив **Int\_Array** без указания размерности. Это подразумевает, что где-то в коде при помощи оператора **ReDim** размерность будет указана. Для ввода данных в массив используется бесконечный цикл **Do** (строки 10–23), так как количество вводимых данных заранее неизвестно. В цикле в диалоговом окне функции **InputBox** (строки 12–14) вводятся значения для элементов массива. Если значение введено (пользователь не отказался от ввода

посредством кнопки **Cancel**), размерность массива увеличивается (строка 19) с сохранением предыдущих значений элементов массива и очередному (новому) элементу массива присваивается введенное значение (строка 22). За количеством элементов в массиве «следит» целая переменная *i*, значение которой увеличивается в начале цикла. Если пользователь отказывается от ввода, оператор **Exit Do** (строка 15) прекращает работу цикла **Do**. При этом значение переменной *i* в этот момент оказывается на единицу больше размерности массива: оператор в строке 11 выполнен (увеличение *i*), а в строке 19 — нет (увеличение размерности массива).

Во второй части процедуры размерность массива уже известна (*i*–1), поэтому можно для формирования строки выдачи на экран использовать цикл с определенным количеством итераций, т.е. цикл **For...Next** (строки 25–27). Остальное по причине сходства с предыдущими двумя листингами должно быть понятно без комментариев.

В листинге 7.17 приведен код, который демонстрирует один из возможных способов работы с динамическим массивом.

#### Листинг 7.17. Работа с динамическим двумерным массивом

```

1: Option Base 1
2: Sub DemoDinArray2()
3:   Dim Int_Array1() As Integer
4:   Dim Int_Array() As Integer
5:   Dim str_msg As String
6:   Dim i As Integer, m As Integer, n As Integer
7:   Dim Num
8:
9:   str_msg = ""
10:  i = 0
11:
12:  Do
13:    i = i + 1
14:    Num = InputBox("Введите целое число для " & i & "-го элемента массива", _
15:                  "Ввод элементов массива")
16:
17:    If Len(Num) = 0 Then Exit Do 'выход из цикла
18:
19:    'изменение размера массива
20:    'с сохранением элементов
21:    ReDim Preserve Int_Array1(i)
22:
23:    'ввод данных в i-й элемент массива
24:    Int_Array1(i) = Num
25:  Loop
26:
27:  m = i - 1          'вторая размерность массива
28:
29:  'ввести первую размерность:
30:  n = InputBox("Введите количество строк массива ")
31:  ReDim Int_Array(n, m)
32:
33:  'запись в выходной массив уже введенной строки:
34:  For j = 1 To m
35:    Int_Array(1, j) = Int_Array1(j)

```

```
36: Next
37:
38:
39: 'теперь работаем с обычным двумерным массивом,
40: 'только первую строку не заполняем:
41: For i = 2 To n
42:     For j = 1 To m
43:         Int_Array(i, j) = InputBox("Введите целое число для " &
44:             & " элемента (" & i & ", " & j & ")", _
45:             "Ввод элементов массива; строка " & i).
46:     Next j
47: Next i
48:
49: 'подготовка строки выдачи
50: For i = 1 To n
51:     For j = 1 To m
52:         str_msg = str_msg & Int_Array(i, j) & ", "
53:     Next j
54:
55:     str_msg = str_msg & Chr(13) 'перевод строки
56: Next i
57:
58: MsgBox "Введено: " & Chr(13) & str_msg, _
59:     , "Вывод ранее введенного массива"
60:
61: End Sub
```

В первой части процедуры **DemoDinArray2** (строки 9–27) в динамический массив **Int\_Array1** записывается первая строка будущего многомерного массива и определяется его вторая (количество столбцов) размерность. Как только пользователь во время ввода первой строки отказался от ввода, формирование первой строки заканчивается. В этот момент программе известна вторая размерность будущего двумерного массива:  $m=i-1$ . В строке 30 функция **InputBox** запрашивает ввести первую размерность массива **Int\_Array**, а в строке 31 объявляется этот двумерный массив. В строках 33–36 данные из одномерного массива **Int\_Array1** переписываются в первую строку двумерного массива **Int\_Array**.

Со строки 41 программа почти ничем не отличается от кода листинга 7.15, начиная со строки 7. Отличие состоит только в том, что в последнем коде заполнение двумерного массива во вложенном цикле осуществляется со второй строки и пределы цикла заданы не литералами, а переменными.

В качестве еще одного примера использования динамического массива рассмотрим программу, при помощи которой можно пронумеровать выделенное количество строк. При этом эта нумерация будет отличаться от автонумерации Word тем, что строки, пронумерованные таким образом, ничем не отличаются от обычного текста и могут быть восприняты другими редакторами при передаче в них данного текста.

**Листинг 7.18. Нумерация строк**

---

```
1: ' НумерацияСтрок Macro
2: ' Macro recorded 28.04.01 by Владимир Кузьменко
3: '
4: ' Нумерует строки выделенного текста
5: ' Вызов: Ctrl+Alt+N
6:
7:     'количество выделенных строк:
8:     ii = Selection.Paragraphs.Count()
9:
10:    'объявление массива для хранения строк:
11:    Dim mas() As String
12:
13:    'цикл считывания строк и добавления нумерации:
14:    For i = 1 To ii
15:        ReDim Preserve mas(i)
16:        tt = Selection.Paragraphs(i)
17:        mas(i) = Str(i) & ":" & vbTab & tt
18:    Next
19:
20:    'удаление выделенного фрагмента:
21:    Selection.Cut
22:
23:    'цикл печати массива:
24:    For i = 1 To ii
25:        Selection.TypeText Text:=mas(i)
26:    Next
27:
```

---

Если вам необходимо начинать нумерацию с некоторого числа и вместо символа ":" использовать какой-либо другой, макрос можно переписать, например, так, как это сделано в следующем листинге.

**Листинг 7.19. Работа с динамическим одномерным массивом**

---

```
1. Sub НумерацияСтрок()
2. '
3. ' НумерацияСтрок Macro
4. ' Macro recorded 28.04.01 by Владимир Кузьменко
5. '
6. ' Нумерует строки выделенного текста
7. ' Вызов: Ctrl+Alt+N
8. '
9.
10.     BeginNumber = CInt(InputBox("Введите начальный номер строки", _
11.         "Ввод номера первой строки", 1))
12.
13.     NumberChar = InputBox("Введите символ разделения", _
14.         "Ввод символа разделения", ":")
15.
16.     'количество выделенных строк:
17.     ii = Selection.Paragraphs.Count()
18.
19.     'объявление массива для хранения строк:
20.     Dim mas() As String
```

```
21.  
22.      'цикл считывания строк и добавления нумерации:  
23.      For i = 1 To ii  
24.          ReDim Preserve mas(i)  
25.          tt = Selection.Paragraphs(i)  
26.          mas(i) = Str(i + BeginNumber - 1) & NumberChar & vbTab & tt  
27.      Next  
28.  
29.      'удаление выделенного фрагмента:  
30.      Selection.Cut  
31.  
32.      'цикл печати массива:  
33.      For i = 1 To ii  
34.          Selection.TypeText Text:=mas(i)  
35.      Next  
36.  End Sub
```

---

Как и в предыдущем макросе, в строке 26, кроме вводимого символа разделения номера и кода, используется символ табуляции (его роль играет константа `vbTab`). Это необходимо для того, чтобы строки кода были выровнены слева. Заметьте, что для нумерации последнего макроса в качестве символа разделения использовался символ ".".

## Функции **LBound** и **UBound**

Для контроля за размерами как статических, так и динамических массивов можно использовать одну или пару переменных для сохранения минимального и максимального индекса массива, как это делается в ранее рассмотренных листингах. Если для отслеживания верхнего и нижнего пределов индексов массива пользователь полагается на собственные переменные, то он полностью отвечает за обновление и точность этих переменных. В противном случае процедуры не будут работать правильно — будет обрабатываться меньше элементов, чем в действительности содержит массив, или будет получено сообщение об ошибке при попытке доступа к массиву с индексами, выходящими за фактический размер массива. Такие программные ошибки бывает трудно отслеживать.

VBA имеет пару функций, которые освобождают пользователя от необходимости вручную отслеживать верхний и нижний пределы массива — функции **LBound** и **UBound**. Эти функции возвращают нижнее и верхнее граничные значения индексов статического или динамического массива.

Функции **LBound** и **UBound** имеют следующий синтаксис:

### Синтаксис

```
LBound(arrayName [, dimension])  
UBound(arrayName [, dimension])
```

Функция **LBound** возвращает первый индекс массива, представленного с помощью *arrayName*. Функция **UBound** возвращает наибольший индекс массива, представленного с помощью *arrayName*. Можно использовать эти две функции как со статическими, так и с динамическими массивами. Аргумент *dimension* представляет целое число, определяющее измерение массива, для которого необходимо получить нижний или верхний предел. Если опустить *dimension*, VBA возвращает предел для первого измерения массива.

---

Следующие фрагменты кода являются примерами использования функций **LBound** и **UBound**:

```
Dim A(3 To 9) As String
For I = LBound(A) To UBound(A)
    A(I) = String(10, 65 + I)
Next I

Dim aMatrix(1 To 365, 1990 To 1999)
For i = LBound(aMatrix, 1) To UBound(aMatrix, 1)
    For j = LBound(aMatrix, 2) To UBound(aMatrix, 2)
        Matrix(i, j) = Rnd
    Next j
Next i
```

Заметьте, что во втором примере функции **LBound** и **UBound** используют необязательный аргумент *dimension*. Внешний цикл **For...Next** выполняется столько раз, сколько элементов имеется в первом измерении массива **aMatrix**, тогда как внутренний цикл **For...Next** выполняется столько раз, сколько элементов имеется во втором измерении массива **aMatrix**.

## Использование **Erase** для очистки или удаления массивов

VBA имеет особый оператор **Erase**, позволяющий выполнять одну из двух задач в зависимости от того, каким массивом манипулирует пользователь — статическим или динамическим. В случае статических массивов **Erase** позволяет очищать все элементы массива, в основном переустанавливая массив в то же самое состояние, какое он имел, когда VBA создавал его в оперативной памяти. В случае динамических массивов **Erase** позволяет полностью удалять массив и его содержимое из оперативной памяти.

Когда элементы массива заполнены, данные в массиве остаются (независимо от того, является данный массив статическим или динамическим) до тех пор, пока пользователь не присвоит новые значения элементам массива или пока VBA не освободится от массива. (Массивы следуют тем же правилам области действия и персистенции, что и любая другая переменная в VBA.)

При некоторых обстоятельствах может понадобиться очистить все значения в массиве, устанавливая числовые значения на 0, строковые значения на пустые строки и так далее. Обычно следует использовать цикл **For...Next** или **For...Each** для установки всех элементов в массиве на определенное значение. Следующие фрагменты кода показывают оба цикла **For...Next** и **For...Each**, используемые для инициализации всех значений в числовом массиве на 0:

```
For k = LBound(NumArray) To UBound(NumArray)
    NumArray(k) = 0
Next k

For Each Num In NumArray
    Num = 0
Next Num
```

Первый фрагмент кода использует цикл **For...Next** и функции **LBound** и **UBound** для прохождения в цикле по всем элементам массива; второй фрагмент кода использует **For...Each** для прохождения в цикле по каждому элементу массива, также устанавливая каждый элемент на 0. Хотя эти циклы до-

вольно короткие, можно выполнить ту же самую задачу для статического массива даже более эффективно с помощью единственного оператора **Erase**:

```
Erase NumArray
```

Массивы «имеют тенденцию» занимать относительно большой объем оперативной памяти — например, массив из 20 элементов занимает такой же объем памяти, как 20 отдельных переменных одного и того же типа. Поскольку массивам требуется так много памяти, следует удалять динамические массивы из оперативной памяти, когда они фактически не используются. (Статические массивы не могут удаляться из оперативной памяти до тех пор, пока VBA автоматически не освобождается от массива.)

VBA удаляет из памяти массивы, объявляемые локально в процедуре (так же, как и любые другие локальные переменные), каждый раз, когда процедура прекращает выполняться. Однако массивы, объявляемые на модульном уровне, существуют пока любая процедура в этом модуле выполняется. Если программа большая, вам может понадобиться восстановить ресурс памяти, используемой динамическими массивами модульного уровня. Оператор **Erase** позволяет делать именно это.

Оператор **Erase** имеет следующий синтаксис:

### Синтаксис

```
Erase array1 [, array2, ...]
```

Здесь *array1* и *array2* представляют любое допустимое имя массива VBA. Можно перечислить столько массивов в операторе **Erase**, сколько хотите, отделяя каждое имя массива запятой.

Оператор **Erase** удаляет из памяти динамические массивы, освобождая область памяти, ранее используемую этим массивом. При удалении динамического массива с помощью оператора **Erase** необходимо повторно создать массив с помощью оператора **ReDim** перед тем, как можно будет использовать этот определенный динамический массив снова. При попытке доступа к элементам в динамическом массиве, для которого был использован оператор **Erase**, без его переопределения VBA отображает сообщение о runtime-ошибке.

Поведение оператора **Erase** для статических массивов немного сложнее и зависит от конкретного типа элементов массива. В следующей таблице описывается действие оператора **Erase** со статическими массивами различных типов:

Тип статического массива	Действие оператора <b>Erase</b>
Любой числовой тип	Устанавливает элементы массива на 0.
Любой строковый тип	Устанавливает элементы массива на строку нулевой длины (""); устанавливает строки фиксированной длины как все символы пробелов.
<b>Variant</b>	Устанавливает элементы массива на <b>Empty</b> .

Тип статического массива	Действие оператора Erase
<b>Object</b>	Устанавливает элементы массива на <b>Nothing</b> .
Любой пользовательский тип	Устанавливает каждую переменную в пользовательском типе индивидуально: численные типы устанавливаются на 0, строковые — на строки нулевой длины, <b>Variant</b> — на <b>Empty</b> , <b>Object</b> — на <b>Nothing</b> .

## Использование массивов в качестве аргументов процедур и функций

VBA дает возможность передавать массивы в качестве аргументов процедур и функций. Эта возможность может быть очень полезной; поскольку нет необходимости задавать размер массива, который передается как аргумент, можно писать универсальные процедуры или функции обработки массива. Например, можно написать функцию, получающую числовой массив в качестве аргумента и возвращающую среднее значение всех чисел в массиве как результат функции. Еще один пример: можно написать процедуру, получающую массив в качестве аргумента, передаваемого по ссылке, и сортирующую этот массив. И в одном, и в другом примерах можно написать процедуру или функцию для работы с массивом любого размера — пять элементов, 10 элементов, 100 элементов и так далее.

Общий синтаксис для аргументов-массивов для процедур или функций такой же, как для любых аргументов процедур или функций:

### Синтаксис

```
[ByVal | ByRef] arrayname() As type
```

Как в случае аргументов процедур и функций, о которых вы уже знаете, ключевое слово **ByVal** указывает VBA передать аргумент-массив по значению, а ключевое слово **ByRef** указывает VBA, что следует передавать аргумент-массив по ссылке. Если **ByVal** и **ByRef** пропущены, VBA передает аргумент-массив по ссылке.

В этой синтаксической конструкции *arrayname()* представляет аргумент-массив; можно использовать любой допустимый идентификатор VBA в качестве имени аргумента-массива. Необходимо всегда включать пустые круглые скобки после *arrayname*; круглые скобки указывают VBA, что этот аргумент является массивом. Параметр *type* представляет любой допустимый тип VBA или определенный пользователем тип.

Следующий фрагмент кода показывает, как включать массивы в качестве аргументов для процедур и функций:

```
Sub ShowText(Lines(), NumLines As Integer)
    'получает массив типа Variant, передаваемый по ссылке
    ...
End Sub

Sub SortList(ByRef List() As String, NumLines As Integer)
    'получает массив типа String, передаваемый по ссылке
    ...
End Sub
```



Если вы помните, передача аргументов по значению (с помощью ключевого слова **ByVal**) приводит к тому, что VBA передает копию данных функции или процедуре. Не передавайте массивы по значению, если в этом нет особой необходимости — передача больших массивов по значению может быстро исчерпать ресурсы памяти вашего компьютера, приводя к runtime-ошибкам из-за нехватки памяти.

Если вы обратили внимание, в двух из трех последних примерах вместе с именами массивов в качестве параметров передаются целые переменные **NumLines**. Здесь подразумевается передача вместе с ссылкой на массив его размерности. Однако замечательным свойством VBA является то, что он позволяет посредством ранее рассмотренных нами функций **LBound** и **UBound** внутри процедуры, которая принимает массив в качестве аргумента, определить размерность переданного ей массива. Отметим, что далеко не все системы программирования обладают такой возможностью. Ниже (листинг 7.20) приведен пример передачи массива в качестве параметра процедуре, единственный оператор которой выдает в диалоговом окне размерность переданного ей массива.

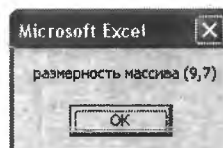
#### Листинг 7.20. Определение размерности массива-аргумента

```
1: Sub TestUBoundLBound(mas())
2:   MsgBox "размерность массива " & "(" & _
3:         UBound(mas, 1) & ", " & UBound(mas, 2) & ")"
4: End Sub
5:
6: Sub a()
7:   Dim arr(9, 7)
8:   Call TestUBoundLBound(arr)
9: End Sub
```

В результате работы этого кода на экране появляется диалоговое окно, представленное на рис. 7.9.

**Рис. 7.9**

Размерность массива, передаваемого в качестве аргумента, легко определяется в вызывающей процедуре



# 8

## Управление файлами с помощью VBA

Управление файлами — это одна из интереснейших задач для программиста в любой системе программирования. Более того, программировать в среде, состоящей из файлов, и не уметь работать с этими файлами (находить, копировать или удалять документ, рабочую книгу или другой файл, записывать в файл или читать из него данные) — не очень полезное занятие. Visual Basic for Applications предоставляет различные операторы, функции и методы для выполнения общих задач управления файлами.

### Управление файлами

В этом разделе описывается, что такое управление файлами, обсуждаются некоторые из типичных действий, которые можно выполнять как часть задач управления файлами, а также описываются VBA-функции, операторы и методы управления файлами.

### Что такое управление файлами

*Управление файлами (file management)* — термин, используемый для описания действий, которые выполняются с файлами, сохраненными на дисковых драйверах. Управление файлами включает действия, такие как копирование файлов, удаление неиспользуемых файлов для освобождения области дисковой памяти, перемещение файлов с одного диска (или папки) на другие и создание или удаление каталогов диска. Управление файлами включает также такие виды обработки, как просмотр списка файлов в папке для определения размера файла или даты и времени, когда этот файл был модифицирован в последний раз.

VBA позволяет выполнять наиболее общие задачи управления файлами под контролем процедуры или функции, которую вы можете написать сами.

Windows Desktop и справочная система используют термин *папка (folder)* для обозначения того, что опытные пользователи Windows или DOS знают как *каталоги (directories)*. В этой книге используется термин *папка* для согласованности с терминологией Windows; в некоторых случаях также используется термин *каталог*.

## Возможности VBA по управлению файлами

В табл. 8.1 приведены VBA-функции, операторы и методы управления файлами. В первом столбце таблицы находится ключевое слово VBA, во втором — указывается, предназначено ли ключевое слово для функции, оператора или объектного метода. Наконец, в третьем столбце содержится краткое описание назначения каждой функции, оператора или метода.

**Таблица 8.1. Функции, методы и операторы управления файлами**

Имя	Категория	Назначение
ChDir	Оператор	Изменяет текущий каталог (папку).
ChDrive	Оператор	Изменяет текущий драйвер диска.
CurDir	Функция	Возвращает текущий каталог (папку).
Dir	Функция	Возвращает имя каталога или файла, совпадающее с определенным именем файла (включая символы универсального сопоставления), передаваемым как строковый аргумент. Предназначена для нахождения одного или нескольких файлов на диске.
FileCopy	Оператор	Копирует файл.
FileDateTime	Функция	Возвращает значение типа Date, содержащее дату и время, когда этот файл был изменен последний раз.
FileLen	Функция	Возвращает длину файла в байтах.
GetAttr	Функция	Возвращает число, представляющее объединенные атрибуты файла или каталога диска, такие как System, Hidden и так далее.
GetOpenFileName	Метод	Отображает Excel-диалоговое окно Open и возвращает имя файла, выбранное пользователем. В Word не имеется.
GetSaveAsFileName	Метод	Отображает Excel-диалоговое окно (Save As) и возвращает имя файла, выбранное пользователем. В Word не имеется.
Kill	Оператор	Удаляет файлы с диска.
MkDir	Оператор	Создает каталог диска (папку).
Name	Оператор	Переименовывает или перемещает файл.
RmDir	Оператор	Удаляет каталог диска (папку).
SetAttr	Оператор	Устанавливает атрибуты файла.

Вы могли заметить, что в табл. 8.1 не приводятся аргументы ни для каких элементов (функций, методов, операторов). Несколько из этих функций, операторов и методов имеют довольно сложные списки аргументов и опций. Начиная со следующего раздела в этой главе, каждая VBA-функция, оператор или метод управления файлами описывается полностью со всеми аргументами и опциями.

Операторы, функции и объектные методы, имеющиеся в VBA, делятся на шесть различных функциональных частей:

- ☐ Получение или изменение атрибутов файла.
- ☐ Выборка или нахождение имен файлов.
- ☐ Получение или изменение текущего драйвера диска и папки или создание и удаление папок диска.
- ☐ Копирование или удаление файлов.
- ☐ Переименование или перемещение файлов.
- ☐ Получение информации о файлах, такой как длина файла, дата и время, когда этот файл был модифицирован последний раз.

## Атрибуты файла

Каждый файл, сохраненный на любом диске Windows (или более ранней версии DOS), имеет атрибуты (attributes). Независимо от конкретного типа драйвера диска (жесткий диск, гибкий диск, RAM-drive, ZIP-drive, Syquest-drive и так далее) Windows и DOS используют атрибуты файла для определения того, какие действия по управлению файлами допускаются для этого файла. Например, Windows (или любая другая операционная система) запрещает вам (и любым программам приложений) удалять, модифицировать или переименовывать файлы с атрибутом read-only.

Windows создает информацию об атрибутах файла, когда вы или программа приложения, работающая на вашем компьютере, создает новый файл. В каком-то смысле, атрибуты файла подобны свойствам объекта: атрибуты файла дают файлу некоторую характеристику в зависимости от конкретных атрибутов, которые он имеет. Атрибуты файла являются частью информации файла, которую Windows сохраняет на драйвере диска. Windows сохраняет информацию об атрибутах файла вместе с информацией об имени файла, размере, дате и времени.

Windows автоматически обновляет и сопровождает информацию об атрибутах файла. Чаще всего пользователи ничего не знают об атрибутах файла, и обычно нет причины, по которой им было бы необходимо знать, что такое атрибуты. В некоторых же случаях бывает полезно понимать и использовать атрибуты файла. В частности, необходимо понимать атрибуты файла и их значение для того, чтобы использовать все преимущества VBA-функции Dir.

Windows использует всего семь атрибутов файла для определения различных характеристик файла. Каждый отдельный атрибут может объединяться с другими атрибутами, кроме атрибута **Volume Label**, например, файл может иметь одновременно атрибуты **Hidden**, **System**, **Directory**, **Archive** и **Read-Only**. В следующем списке содержится имя каждого атрибута файла и описывается его значение.

- **Archive.** Атрибут **Archive** указывает, изменялся ли файл со времени, когда его резервировали последний раз с помощью backup-программы, такой как BACKUP Windows, или backup-программы других поставщиков, таких как Fastback!, BackIt, Norton Backup и других. Если файл имеет атрибут **Archive**, это означает, что для этого файла необходимо резервирование. Если файл не имеет атрибута **Archive**, то этот файл не изменялся со времени, когда он был резервирован последний раз.
- **Directory.** Если файл имеет атрибут **Directory**, это означает, что файл, в действительности, является каталогом или подкаталогом (папкой — в терминологии Windows). Каталог диска — это файл, который содержит информацию о других файлах; когда вы создаете каталог, Windows создает специальный файл каталога и дает ему атрибут **Directory**. Атрибут **Directory** сообщает Windows о том, что этот файл содержит информацию о других файлах и препятствует переименованию, копированию или удалению каталога.
- **Hidden.** Если файл имеет атрибут **Hidden**, Windows «скрывает» файл, не показывая его в большинстве случаев при отображении каталога, хотя Windows имеет опцию просмотра, которая отображает имена скрытых файлов.
- **Normal.** Атрибут файла **Normal** является, на самом деле, признаком отсутствия каких-либо специальных атрибутов. Так называемый **Normal**-атрибут (иногда называемый *general*) файла просто означает, что файл не имеет никаких других атрибутов, кроме, возможно, атрибута **Archive** (для указания, необходимо ли резервирование для этого файла).
- **Read-Only.** Атрибут **Read-Only** означает, что вы можете только читать из файла, но не можете изменять его. Windows 95/98 препятствует изменению, удалению или переименованию файла, который имеет атрибут **Read-Only**.
- **System.** Атрибут **System** указывает Windows, что файл является частью операционной системы компьютера. Как в случае с файлами **Read-Only**, Windows препятствует изменению файла, имеющего атрибут **System**. Кроме того, если вы создаете диск автозапуска DOS-командой SYS (или с Windows Control Panel), любые файлы, имеющие атрибут **System**, переносятся на новый диск автозагрузки.
- **Volume Label.** Атрибут **Volume Label** информирует Windows о том, что файл является меткой тома диска. [*Метка тома (volume label)* — это имя, которое вы даете жесткому диску (или дискете) при форматировании, использовании DOS-команды LABEL или изменении свойства **Label** в листе свойств диска.] Диск может иметь только одну метку тома одновременно.

Windows представляет каждый отдельный атрибут файла уникальным числом и сохраняет это число с информацией об имени и размере файла. Если файл имеет несколько атрибутов, Windows складывает кодовые числа для каждого атрибута и сохраняет их сумму. В следующем разделе описывается, как можно найти и интерпретировать кодовое число для атрибутов файла.

В табл. 8.2 перечисляются коды атрибутов файлов, которые использует Windows, и внутренние константы VBA для этих кодов атрибутов. Как и в случае с другими числовыми кодами, для которых VBA определяет константу,

следует использовать VBA-константу для кодового числа вместо самого кодового числа.

**Таблица 8.2. Константы атрибутов файла Visual Basic for Applications**

Константа VBA	Значение	Что означает
vbNormal	0	Normal.
vbReadOnly	1	Read-Only.
vbHidden	2	Hidden.
vbSystem	4	System.
vbVolume	8	Volume Label.
vbDirectory	16	Каталог или подкаталог диска.
vbArchive	32	Archive. Если установлен, указывает, что этот файл был изменен со времени последнего резервирования.
vbAlias	64	Указанное имя файла является алиасным. Доступна только в Macintosh.

**Получение атрибутов файла**

Для того чтобы определить, какие атрибуты имеет определенный файл, используйте VBA-функцию **GetAttr**.

Функция **GetAttr** возвращает число, содержащее сумму всех числовых кодов для атрибутов файла, и имеет следующий синтаксис:

**Синтаксис**

`GetAttr (pathname)`

Аргумент *pathname* — любое строковое выражение VBA, представляющее допустимое имя файла; *pathname* может включать буквенную метку диска и полный путь папки; если вы не включаете буквенную метку, **GetAttr** ищет указанный файл на текущем драйвере диска; если вы не включаете путь папки, функция **GetAttr** выполняет поиск в текущей папке.

В листинге 8.1 содержится пример, в котором используется функция **GetAttr** и показывается, как интерпретировать результат функции.

**Листинг 8.1. Использование функции **GetAttr** и интерпретация результата**

```
1: Sub ShowFA(fName As String)
2: 'отображает окно с сообщением об атрибутах
3: 'файла, имя которого передается как аргумент
4:
5:   Dim fAttr As Integer
6:   Dim mStr As String
7:
8:   fAttr = GetAttr(fName)
9:   mStr = UCase(fName)
10:  mStr = mStr & " has these attributes: " & vbCr
```

```
11: If (fAttr And vbReadOnly) Then
12:     mStr = mStr & "Read-Only" & vbCr
13: If (fAttr And vbHidden) Then
14:     mStr = mStr & "Hidden" & vbCr
15: If (fAttr And vbSystem) Then
16:     mStr = mStr & "System" & vbCr
17: If (fAttr And vbVolume) Then
18:     mStr = mStr & "Volume" & vbCr
19: If (fAttr And vbDirectory) Then
20:     mStr = mStr & "Directory" & vbCr
21: If (fAttr And vbArchive) Then
22:     mStr = mStr & "Archive" & vbCr
23: MsgBox mStr
24: End Sub
25:
26: Sub ListFileAttr()
27:
28:     Dim strPath As String
29:
30:     ShowFA fName:="c:\io.sys"
31:     ShowFA fName:="c:\msdos.sys"
32:
33:     strPath = "C:\Program Files\Microsoft Office\Office\XlStart"
34:     ShowFA fName:=strPath
35:
36: End Sub
```

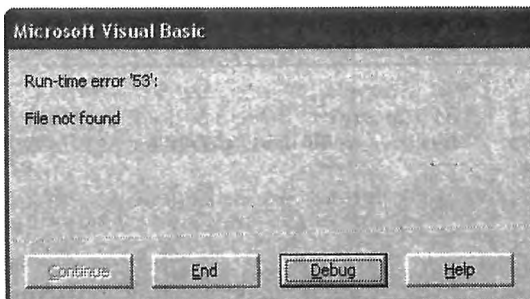
В листинге 8.1 содержатся две процедуры. Процедура **ShowFA** занимает строки 1–24, а процедура **ListFileAttr** — строки 26–36. Процедура **ListFileAttr** просто вызывает **ShowFA** несколько раз, передавая каждый раз другое имя файла.

Процедура **ShowFA** имеет один обязательный аргумент с именем **fName** и типом **String**. В строке 5 объявляется переменная **fAttr** типа **Integer** для сохранения значения, возвращаемого функцией **GetAttr**. В строке 6 объявляется переменная **mStr** типа **String** для сохранения сообщения, формируемого этой процедурой.

При выполнении оператора в строке 8 VBA вызывает **GetAttr**, передавая строку **fName** в качестве аргумента и сохраняя результат функции в переменной **fAttr**. Если строка в **fName** не содержит допустимого имени файла, VBA отображает сообщение о runtime-ошибке (рис. 8.1).

**Рис. 8.1**

Сообщения VBA при ошибке  
в аргументе функции **GetAttr**



Строки 9 и 10 формируют первую часть сообщения, которое отображает процедура **ShowFA**. В строке 9 используется VBA-функция **UCase** для преобразования имени файла в **fName** в верхний регистр по «косметическим» причинам.

В строках 11–22 находится ряд операторов **If...Then** оценивающих число, сохраненное в **fAttr**, которое содержит сумму всех кодовых чисел для атрибутов этого файла. Заметьте, что эти операторы **If...Then** не являются вложенными. Значение в **fAttr** тестируется шесть раз — один раз для каждого возможного значения атрибута. Помните, что значение атрибута, которое Windows 95/98 сохраняет с файлом, является суммой всех кодовых чисел для каждого из атрибутов, которые имеет файл. Если атрибут, наличие которого проверяет определенный оператор **If...Then**, присутствует, то строка, содержащая соответствующее имя атрибута, добавляется в строку сообщения, сохраняемую в **mStr**. Оператор **MsgBox** в строке 23 отображает строку сообщения, которую формируют предыдущие операторы.

В строке 26 содержится объявление процедуры **ListFileAttr**, которая просто вызывает процедуру **ShowFA** несколько раз, передавая ей различные имена файлов. Выполнение строки 30 приводит к тому, что **ShowFA** отображает атрибуты файла для файла **IO.Sys**. Этот файл является частью операционной системы Windows и всегда находится в корневом каталоге диска автозагрузки. Поскольку **IO.SYS** является частью операционной системы Windows, он имеет атрибут файла **System**; он имеет также атрибут **Read-Only** для предотвращения нечаянного удаления этого важного файла; кроме того, этот файл имеет атрибут **Hidden**, чтобы он обычно не появлялся в папке рабочего стола Windows или в списках каталога Проводника. При выполнении процедуры **ListFileAttr** последовательно отображаются диалоговые окна, показанные на рис. 8.2.

**Рис. 8.2**

Такие окна последовательно отображаются кодом листинга 8.1



В процедуре **ShowFA** нет тестирования для атрибута файла **Normal**, потому что атрибут файла **Normal** — это просто отсутствие любых других атрибутов.

Объявляйте все переменные, которые используете для сохранения кодов атрибутов файла, как переменные типа **Integer**, так кодовое число атрибута файла всегда находится в допустимом диапазоне типа **Integer**. Не следует также забывать, что коды атрибутов всегда обрабатываются как одно число, содержащее сумму всех отдельных кодовых чисел для атрибутов данного файла.



## Изменение атрибутов файла

Атрибуты файлов можно устанавливать из кода VBA. Например, можно изменить атрибуты файла шаблонного документа, чтобы включить атрибут **Read-Only** для предотвращения нечаянного удаления этого шаблонного документа.

Для изменения атрибутов файла предназначен оператор **SetAttr**, который выполняет ту же самую задачу, что и команда **Файл | Свойства** (доступная в любых окнах каталога Windows или с помощью щелчка правой кнопкой мыши на значке файла), а также DOS-команда **ATTRIB**.

Оператор **SetAttr** имеет следующий синтаксис:

### Синтаксис

---

`SetAttr pathname, attributes`

Здесь *pathname* — любое строковое выражение, содержащее допустимую спецификацию файла Windows [имя файла и (необязательно) полный путь к каталогу, и буквенную метку диска]; *attributes* — это численное выражение. Численное выражение для *attributes* должно быть числом между 1 и 255 и состоять из одного из кодовых чисел атрибута файла или суммы кодовых чисел атрибутов.

---

В листинге 8.2 показан пример того, как можно использовать оператор **SetAttr**.

### Листинг 8.2. Использование оператора **SetAttr** для изменения атрибутов файла

---

```
1: Sub SetReadOnly(fName As String)
2: 'устанавливает атрибут только на чтение
3:
4:     Dim fAttr As Integer
5:
6:     fAttr = GetAttr(fName)
7:
8:     'не устанавливать атрибут, если он уже установлен
9:     If Not CBool(fAttr And vbReadOnly) Then
10:         SetAttr fName, fAttr + vbReadOnly
11:     End If
12:
13:     MsgBox prompt:="Атрибут Read-only для " & _
14:             fName & " установлен.", _
15:             Title:="Delete Protection"
16: End Sub
17:
18:
19: Sub Test_SetReadOnly()
20:     Dim iName As String
21:     iName = Application.GetOpenFilename
22:     SetReadOnly iName
23:     ShowFA iName
24: End Sub
```

---

Процедура **SetReadOnly** защищает файлы от случайного удаления тем, что присваивает файлу атрибут **Read-Only**, не влияя на другие атрибуты.

**SetReadOnly** имеет один обязательный аргумент типа **String** с именем **fName**, который используется для передачи имени файла в процедуру. В строке 4 объявляется переменная **fAttr**; процедура использует эту переменную для сохранения числа для атрибутов файла.

В строке 6 используется функция **GetAttr** для получения атрибутов файла, определенного с помощью **fName**. В строках 9–11 содержится оператор **If**, выполняющий оператор **SetAttr**, если только этот файл не имеет уже атрибут **Read-Only**. В строке 10 содержится фактический вызов оператора **SetAttr**. Заметьте, что старые атрибуты файла арифметически складываются с константой **vbReadOnly** для задания новых атрибутов. Таким образом, оригинальные атрибуты файла сохраняются.

Наконец, в строках 13–15 содержится оператор **MsgBox**, отображающий сообщение пользователю о том, что атрибут **Read-Only** для файла установлен.

Процедура **Test\_SetReadOnly** тестирует процедуру **SetReadOnly**. В строке 21 используется Excel-метод **GetOpenFileName** (который будет описан в следующем разделе) для получения имени файла от пользователя. В строке 23 вызывается процедура **SetReadOnly** для задания атрибутов файла с целью включения атрибута **Read-Only**. Наконец, в строке 24 вызывается процедура **ShowFA** (приведенная в листинге 8.1) для отображения окна сообщения с новыми атрибутами файла.

Для добавления нового атрибута файла к имеющемуся набору атрибутов используйте оператор **Or**. Применение оператора **Or** для объединения атрибутов файла в побитовой операции исключает любые возможные проблемы, которые могут возникнуть при использовании арифметического сложения для объединения новых атрибутов с уже имеющимися. Результатом выражения **0 Or 1** является **1**; результатом выражения **1 Or 1** является также **1**. Поэтому можно объединять новый атрибут файла с имеющимся числом атрибута файла, используя выражение, подобное **OldAttr Or vbArchive**. Если атрибут **Archive** не задан в **OldAttr**, он будет задан как результат операции **Or**. Если атрибут **Archive** уже задан, он останется неизменным.

## Как находить файлы

В этом разделе сначала будет показано, как использовать VBA-функцию **Dir** для поиска в папке одного или нескольких файлов, совпадающих с определенным именем файла. Затем вы узнаете, как использовать VBA для включения Word- или Excel-собственных диалоговых окон **Open** (Открытие документа) и **Save As** (Сохранение документа) в ваши процедуры. Можно использовать Word- и Excel-встроенные диалоговые окна **Open** и **Save As** для предоставления возможности пользователю легко и правильно передавать имена файлов процедурам.

### Использование функции **Dir** для нахождения файлов

Иногда может быть необходимо проверить, содержит ли каталог диска один или несколько файлов, совпадающих с определенной спецификацией имени файла, такой как **\My Documents\\*.xls** или **\My Documents\\*.doc**. Для этого можно использовать VBA-функцию **Dir**. Функция **Dir** выполняет в VBA те же задачи, что и список файлов в окне папки Windows или DOS-команда **DIR**, —

дает возможность посмотреть, какие файлы сохранены в определенной папке и какие имеются другие папки.

Функция **Dir** имеет следующий общий синтаксис:

### Синтаксис

---

`Dir(pathname[, attributes])`

Здесь *pathname* представляет любое выражение типа **String**, имеющее результатом допустимое имя файла. Имя файла может содержать буквенную метку драйвера и полный путь папки. Имя папки может также включать символы универсального сопоставления файлов. Необязательный аргумент *attributes* является числом, представляющим атрибуты тех файлов, которые необходимо найти. При использовании *attributes* функция **Dir** выполняет поиск файлов, имеющих эти атрибуты. Если опустить аргумент *attributes*, функция **Dir** выполняет поиск обычных файлов, то есть любых файлов, кроме имеющих атрибуты **Hidden**, **Volume Label**, **Directory** или **System**.

При вызове функции **Dir** с аргументом *pathname* она возвращает строку, содержащую имя первого найденного ею файла, совпадающее с именем файла в аргументе *pathname*. Если имя файла содержит символы универсального сопоставления (\* или ?), то для того, чтобы найти все файлы в каталоге, совпадающие с этим именем файла, необходимо использовать функцию **Dir** в два этапа. Во-первых, вызывать **Dir** с аргументом *pathname* для получения первого совпадающего файла, затем вызывать **Dir** неоднократно без аргументов до тех пор, пока **Dir** не возвратит пустую строку. Как только **Dir** возвращает пустую строку, значит, больше не остается файлов, совпадающих с этим именем файла. При вызове **Dir** без указания имени файла, VBA выдает сообщение о runtime-ошибке.

---

В листинге 8.3 показан пример использования **Dir** для нахождения только одного файла.

### Листинг 8.3. Использование Dir для нахождения одного файла

---

```
1: Function IsFile(fName As String) As Boolean
2:   'возвращает Истина (True), если файл на диске найден,
3:   'иначе - Ложь (False)
4:   If (Dir(fName) <> "") Then
5:     IsFile = True
6:   Else
7:     IsFile = False
8:   End If
9: End Function
10:
11:
12: Sub Test_IsFile()
13:   Dim iName As String
14:   Dim str_msg As String
15:
16:   iName = InputBox(prompt:="Введите имя файла:", _
17:     Title:="Тестирование функции IsFile")
18:
19:   str_msg = "Файл " & iName & "; результат поиска: "
20:   If IsFile(iName) Then
21:     str_msg = str_msg & " найден"
```

```
22: Else
23:     str_msg = str_msg & " не найден"
24: End If
25:
26: MsgBox str_msg
27: End Sub
```

В строках 1–9 содержится функция **IsFile**, имеющая один обязательный аргумент **fName** типа **String**. **IsFile** возвращает результат с типом **Boolean**: **True**, если имя файла в **fName** находится на драйвере диска, иначе — **False**.

Работа функции **IsFile** очень проста. В строке 4 содержится оператор **If...Then...Else**, который вызывает функцию **Dir**, передавая содержимое **fName** в качестве имени файла, поиск которого необходимо выполнить. В этом обращении к функции **Dir** никакой аргумент для атрибутов не указывается, поэтому она будет находить любой файл, кроме файлов, имеющих атрибуты **Hidden**, **System**, **Directory** или **Volume Label**.

Поскольку назначением функции **IsFile** является просто определение того, существует ли определенный файл, возвращаемый результат функции **Dir** не используется, кроме сравнения его с пустой строкой. Если результат функции **Dir** не является пустой строкой, то **Dir** нашла указанный файл и VBA выполняет строку 5, которая присваивает результату функции значение **True**. Если результат функции **Dir** является пустой строкой, то никакого совпадающего файла не было найдено и VBA выполняет строку 7, которая присваивает **False** результату функции.

В строках 12–27 объявляется процедура для тестирования функции **IsFile**. Учитывая номер текущей главы, не будем подробно останавливаться на разборе этого кода. Если в таком простом коде что-то неясно, следует прочитать внимательнее предыдущие главы.

Функцию **Dir** можно также использовать для нахождения в папке всех файлов, совпадающих с определенным именем файла. Эта возможность функции **Dir** наиболее полезна, когда имя файла содержит символы универсального сопоставления. Можно использовать функцию **Dir** таким образом, если необходимо, скажем, найти все файлы шаблонов в определенной папке.

В листинге 8.4 показан пример того, как функция **Dir** используется для нахождения нескольких файлов.

#### Листинг 8.4. Использование функции **Dir** для нахождения нескольких файлов

```
1: Sub ShowFA(fName As String, Number As Integer)
2:     'отображает окно с сообщением об атрибутах
3:     'файла, имя которого передается как аргумент
4:
5:     Dim fAttr As Integer
6:     Dim mStr As String
7:
8:     fAttr = GetAttr(fName)
9:     mStr = UCase(fName)
10:    mStr = mStr & " has these attributes: " & vbCr
11:    If (fAttr And vbReadOnly) Then
12:        mStr = mStr & "Read-Only" & vbCr
13:    If (fAttr And vbHidden) Then
14:        mStr = mStr & "Hidden" & vbCr
```

```

15:   If (fAttr And vbSystem) Then _
16:       mStr = mStr & "System" & vbCr
17:   If (fAttr And vbVolume) Then _
18:       mStr = mStr & "Volume" & vbCr
19:   If (fAttr And vbDirectory) Then _
20:       mStr = mStr & "Directory" & vbCr
21:   If (fAttr And vbArchive) Then _
22:       mStr = mStr & "Archive" & vbCr
23:
24:   'запись в лист Excel:
25:   Cells(Number, 1).Value = mStr
26: End Sub
27:
28: Sub ShowAllFiles()
29: 'отображает имена и атрибуты всех файлов
30: 'в указанном каталоге Введите имя каталога:
31:
32:   Dim sAttr As Integer
33:   Dim fName As String
34:   Dim pName As String
35:   Dim fCount As Integer
36:
37:   pName = InputBox("Введите имя каталога:")
38:   If Trim(pName) = "" Then Exit Sub
39:   If Right(pName, 1) <> "\" Then pName = pName & "\"
40:
41:   sAttr = vbDirectory + vbArchive + vbReadOnly + _
42:       vbHidden + vbSystem
43:
44:   'получить первое имя файла и атрибуты
45:   fName = Dir(pName & "*.*", sAttr)
46:   If (fName <> "") And _
47:       ((fName <> ".") And (fName <> "..")) Then
48:       fCount = 1
49:       ShowFA pName & fName, fCount
50:   End If
51:
52:   Do While (fName <> "")
53:       fName = Dir()
54:       If (fName <> "") And _
55:           ((fName <> ".") And (fName <> "..")) Then
56:           fCount = fCount + 1
57:           ShowFA pName & fName, fCount
58:       End If
59:   Loop
60:
61:   MsgBox "Найдено файлов " & fCount
62: End Sub

```

В этом листинге сначала (строки 1–26) описана процедура **ShowFA**, которая уже встречалась в листинге 8.1. Здесь были внесены небольшие изменения: теперь процедура имеет еще один параметр **Number** типа **Integer** и результирующая строка выдается не в окне оператора **MsgBox** (при длинном списке файлов это будет утомлять пользователя даже в процессе тестирования

кода), а записывается в Excel-лист, причем **Number** указывает номер строки, в которую будет записываться результирующая строка.

Процедура **ShowAllFiles** запрашивает пользователя ввести имя папки (каталога), а затем использует функцию **Dir** для нахождения всех файлов в этом каталоге. В строках 32–35 объявляются несколько переменных. Переменная **sAttr** используется для сохранения числа атрибутов файла, **fName** — для временного хранения имен файлов, возвращаемых функцией **Dir**, **pName** — для сохранения пути каталога, который ищет эта процедура, **fCount** — для сохранения счетчика всех найденных файлов и в качестве параметра функции **ShowFA** для записи данных о файле в очередную строку листа.

В строке 39 проверяется, имеет ли путь (введенный пользователем в строке 37) каталога в конце строки символ разделителя пути (\), и добавляет его в конец пути, сохраненного в **pName**, если символ пропущен.

В строках 41–42 задается переменная **sAttr**, содержащая атрибуты файлов, поиск которых будет выполнять функция **Dir**. Заметьте, что **sAttr** задается так, что она содержит все возможные атрибуты файла, кроме **Volume Label**. При таких атрибутах **Dir** будет находить любой файл в заданном каталоге, включая любые подкаталоги в нем.

В строке 45 выполняется первый вызов функции **Dir** с использованием пути каталога в **pName** с добавленной в него строкой **"\*.\*"** (для нахождения всех файлов) и переменной **sAttr** (с атрибутами файлов). Результат функции **Dir** присваивается переменной **fName**.

В строках 46–50 содержится оператор **If . . . Then**, который оценивает строку, возвращаемую функцией **Dir** и сохраненную в переменной **fName**. Оператор проверяет: 1) является ли переменная **fName** пустой строкой, 2) не содержит ли **fName** строку, состоящую из одной точки (.), 3) не содержит ли **fName** строку, состоящую из двух точек (..). Эти особые строки используются системой файлов Windows для обозначения текущего (.) и родительского каталога (..). Хотя такие особые имена файлов являются каталогами и функция **Dir** будет сообщать об их существовании с таким атрибутом и спецификацией универсального сопоставления, какие используются в данном примере, они, на самом деле, не существуют на диске, и любая попытка получить их атрибуты с помощью **GetAttr** приведет к возникновению runtime-ошибки.

Если **fName** не является пустой строкой и не содержит специальных элементов (. или ..), VBA выполняет оператор в строке 48, который устанавливает счетчик найденных файлов в 1 (переменная **fCount**), и выполняет оператор в строке 49, который вызывает процедуру **ShowFA** для записи в очередную ячейку текущего листа открытой рабочей книги имени файла и его фактических атрибутов.

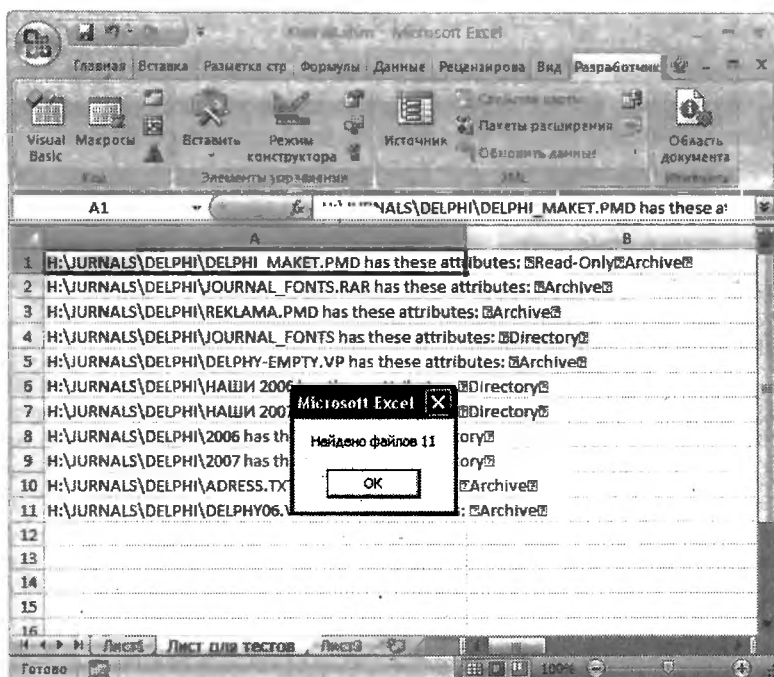
Теперь, когда первый совпадающий файл найден, необходимо использовать формат второй ступени функции **Dir**, чтобы найти остальные файлы в каталоге. В строках 52–60 содержится цикл **Do**, который выполняется, пока **fName** не является пустой строкой. Если хотя бы один совпадающий файл был найден в вызове функции **Dir** (в строке 45), то этот цикл будет выполняться, по крайней мере, один раз.

Обратите особое внимание на строку 53. Этот оператор снова вызывает функцию **Dir** без аргументов. При таком вызове функции **Dir** VBA «полагает», что пользователь хочет найти больше файлов, совпадающих со спецификацией для аргументов пути и атрибутов, которые использовались в послед-

нем вызове `Dir`. Функция `Dir` возвращает следующий файл в каталоге, совпадающий с предыдущими спецификациями имени и атрибутов. Если файлов больше нет, `Dir` возвращает пустую строку.

Когда все совпадающие файлы в каталоге будут найдены, `Dir` возвращает пустую строку и цикл прекращает выполняться. Оператор в строке 62 отображает окно сообщения, указывающее, сколько совпадающих файлов было найдено. На рис. 8.3 представлен результат работы кода этого листинга при введенной в окне `InputBox` строке «H:\JURNALS\Delphi» (конечно, на каком-то случайном компьютере).

**Рис. 8.3**  
Один из  
возможных  
результатов  
работы кода  
из листинга 8.4



## Использование встроенных диалоговых окон Excel для получения имен файлов

Excel-объект `Application` имеет два метода — `GetOpenFilename` и `GetSaveAsFilename`, которые можно использовать в Excel-коде VBA для упрощения задачи получения имени файла от пользователя. В следующих двух разделах описывается, как использовать эти два метода в процедурах.

### Использование метода `GetOpenFilename`

Во многих примерах в этой книге использовалась функция `InputBox` для получения имен файлов от пользователя процедуры. Хотя получение имени файла с помощью `InputBox` является обычной практикой, при этом не используются преимущества ориентированного на файл диалогового окна, такого как

Excel-окно **Open**, которое позволяет увидеть, какие файлы имеются на диске, и просматривать списки файлов различных драйверов диска и каталогов.

Можно использовать Excel-метод **GetOpenFilename** в Excel-коде VBA для отображения диалогового окна, которое выглядит и работает так же, как диалоговое окно, отображаемое Excel при выборе команды **File | Open** (Файл | Открыть). Метод **GetOpenFilename** возвращает строку, которая содержит имя файла, выбранного пользователем, включая буквенную метку диска и полный путь папки. Если пользователь отменяет диалоговое окно, **GetOpenFilename** возвращает **Boolean**-значение **False** в качестве результата.

Excel-метод **GetOpenFilename** имеет следующий синтаксис:

### Синтаксис

---

```
object.GetOpenFilename(fileFilter, filterIndex, title,  
    buttonText, multiSelect)
```

Здесь *object* — это ссылка на Excel-объект **Application**. Хотя ссылка *object* на Excel-объект **Application** является обязательной, все аргументы **GetOpenFilename** являются необязательными. Аргумент *fileFilter* представляет любое допустимое выражение **String**, специально форматированное для задания фильтров файла, перечисленных в окне раскрывающегося списка **Files of type** в диалоговом окне **Open**. Если аргумент *fileFilter* опущен, фильтр файла для раскрывающегося списка **Files of type** — это **All Files (\*.\*)**.

Аргумент *filterIndex* представляет любое численное выражение и указывает, какой фильтр файла должен использовать VBA как фильтр по умолчанию для раскрывающегося списка **Files of type** (Тип файлов). Если этот аргумент опустить или указать число, большее, чем фактическое количество фильтров файла, то первый фильтр файла становится фильтром по умолчанию. Аргумент *title* представляет любое выражение типа **String** и является заголовком, отображаемым VBA в диалоговом окне **Open**. Если опустить *title*, VBA отображает диалоговое окно с обычным **Open-заголовком**.

Аргумент *buttonText* — необязательный параметр типа **Variant**. Используется только в Macintosh.

Аргумент *multiSelect* представляет любое выражение или значение **Boolean**. Если *multiSelect* равно **True**, то диалоговое окно **Open** дает возможность пользователю выбирать множество имен файлов. Когда *multiSelect* равно **True**, **GetOpenFilename** возвращает массив, содержащий имена всех выбранных файлов.

Для определения строки фильтра файла форматируйте строку следующим образом:

```
"FilterName1,filespec1,FilterName2,filespec2,FilterNameN,filespecN"
```

Аргумент *filterName* представляет текст, который VBA отображает в окне раскрывающегося списка **Files of type**. Аргумент *filespec* представляет спецификацию файла, которую Windows использует для ограничения файлов, приводимых в окне **Open**. Можно вносить в список сколь угодно много фильтров файла.

---

В следующей строке показан пример строки фильтра файла:

```
"Templates (*.xlt), *.xlt, Workbooks (*.xlsx), *.xls"
```

Эта строка фильтра приводит к появлению двух опций (**Templates** и **Workbooks**) в окне раскрывающегося списка **Files of type**.

В листинге 8.5 показан пример, использующий метод **GetOpenFilename**.



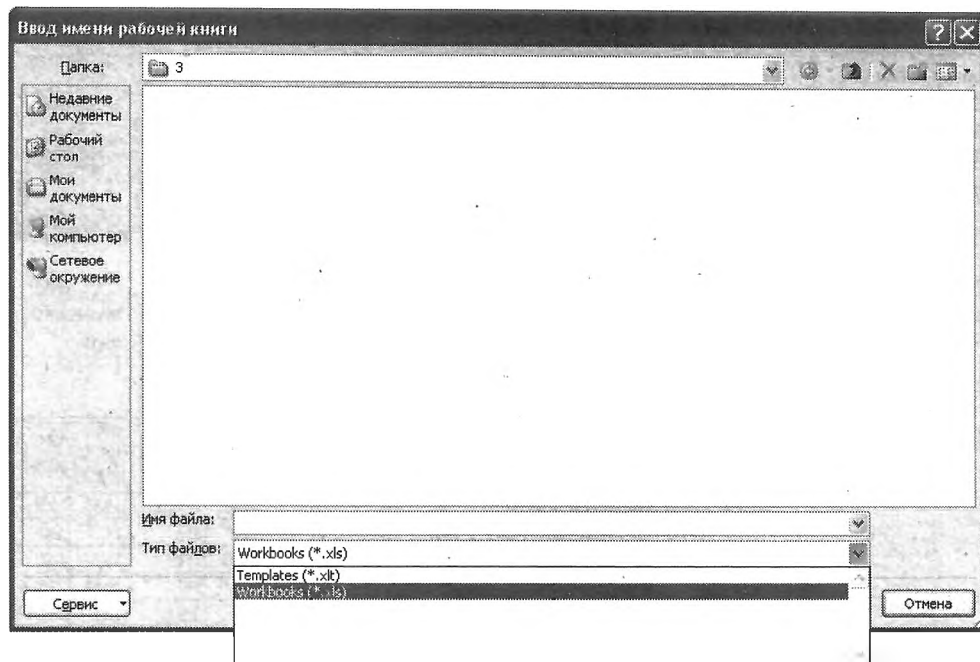
**Листинг 8.5.** Использование метода `GetOpenFilename` для получения имени файла от пользователя процедуры

```
1: Sub TestGetOpenFilename()  
2: 'открывает указанную рабочую книгу и выбирает  
3: 'лист, указываемый пользователем в окне функции InputBox  
4:  
5:     Const iTitle = "Ввод имени рабочей книги"  
6:     Dim ShtName As String  
7:  
8:     Const FilterList =  
9:         "Templates (*.xlt),*.xlt,Workbooks (*.xlsx),*.xlsm"  
10:  
11:     Dim fName As String  
12:  
13:  
14:     fName = Application.GetOpenFilename(Title:=iTitle, _  
15:         filefilter:=FilterList, _  
16:         filterindex:=2)  
17:  
18:  
19:     If fName = "False" Then 'для русской версии "Ложь"  
20:         MsgBox prompt:="Операция отменена!", _  
21:             Title:=iTitle  
22:         Exit Sub  
23:     End If  
24:  
25:     Workbooks.Open Filename:=fName  
26:  
27:     ShtName = InputBox("Введите наименование листа")  
28:     ActiveWorkbook.Sheets(ShtName).Select  
29:     MsgBox prompt:="Книга " & fName & _  
30:         " открыта, лист " & ShtName & _  
31:         " выбран ",  
32:         Title:=iTitle & " (Завершение)"  
33: End Sub
```

Процедура **TestGetOpenFilename** открывает рабочую книгу, выбранную пользователем в диалоговом окне с заголовком, определяемым константой **iTitle**, а затем выбирает рабочий лист с именем, указываемым в окне функции **InputBox**, в открытой рабочей книге.

В строках 8–9 объявляется константа **FilterList**, в дальнейшем передаваемая в качестве фильтра файла при обращении к методу **GetOpenfilename**.

В строках 14–16 содержится один оператор, вызывающий метод **GetOpenFilename** и присваивающий его результат переменной **fName**. При выполнении этого оператора VBA отображает диалоговое окно, показанное на рис. 8.4. На этом рисунке показано открытое окно списка **Тип файлов (Files of type)**, чтобы можно было видеть действие аргумента *fileFilter*. Заметьте, что диалоговое окно на рисунке идентично Excel-окну **Open** (Открытие документа), за исключением заголовка окна и содержимого окна списка **Тип файлов (Files of type)**. Поскольку аргумент *filterIndex* (строка 16) имеет значение 2, фильтром по умолчанию является фильтр **Workbooks (\*.xlsx)**, а не **Templates (\*.xlt)**.



**Рис. 8.4.** Процедура **TestGetOpenFilename** использует метод **GetOpenFilename** для отображения этого диалогового окна

Имя в диалоговом окне можно выбрать точно так же, как в Excel-окне **Open**. Когда вы выбираете кнопку **Open** (Открыть), метод **GetOpenFilename** возвращает строку, содержащую имя файла, буквенную метку диска и полный путь каталога (аргумент *multiSelect* не был использован, поэтому можно выбрать только одно имя файла). Если вы выберете **Cancel** (Отмена), **GetOpenfilename** возвращает **Boolean**-значение **False**. Поскольку в данном случае **fName** является переменной **String**, VBA автоматически преобразует **False** в строку «False».

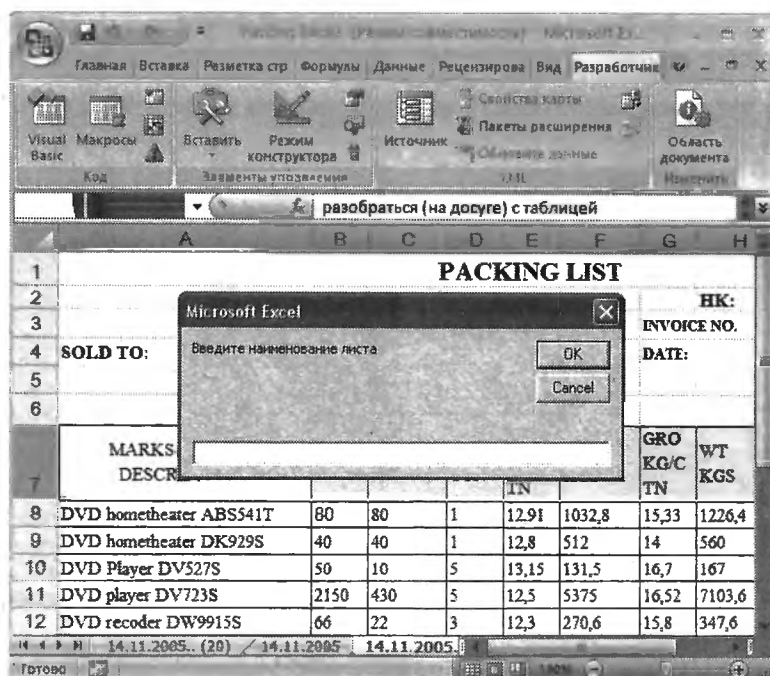
Если операторы в строках 19–23 не обнаружат факт отказа от ввода имени файла, то в строке 25 вызывается метод **Workbooks.Open**, открывающий выбранную пользователем рабочую книгу. В строке 27 с помощью функции **InputBox** пользователь может ввести наименование листа уже открытой рабочей книги. В это время ярлычки с именами листов будут видны пользователю (рис. 8.5).

Завершается процедура, как обычно, оператором **MsgBox** с сообщением об успешном выполнении задачи процедуры.

Не следует забывать, что **GetOpenFilename** возвращает строку, когда пользователь выбирает имя файла, и **Boolean**-значение **False** — при отмене диалогового окна. Если вы присваиваете результат метода **GetOpenFilename** переменной типа **Variant**, необходимо выполнять тестирование для значения **False**, а не строки «False».

Рис. 8.5

В момент ввода наименования необходимого листа ярлычки листов видны пользователю



Метод **GetOpenFilename** возвращает массив строк, если используется значение **True** для необязательного аргумента *multiSelect*, даже если пользователь выбирает только один файл.

## Использование метода **GetSaveAsFilename**

Вы можете использовать Excel-метод **GetSaveAsFilename** для отображения диалогового окна, которое выглядит и работает так же, как окно, отображаемое Excel, когда пользователь выбирает команду **File | Save As....** Метод **GetSaveAsFilename** также возвращает строку, содержащую имя файла, которое выбирает пользователь, включая буквенную метку диска и полный путь каталога. Если пользователь отменяет диалоговое окно, **GetSaveAsFilename** возвращает **Boolean**-значение **False**.

Метод **GetSaveAsFilename** имеет следующий синтаксис:

### Синтаксис

```
object.GetSaveAsFilename(initialFilename, fileFilter,
filterIndex, title, buttonText)
```

Метод **GetSaveAsFilename** имеет почти такие же аргументы и синтаксис, как метод **GetOpenFilename**. В этой синтаксической конструкции *object* — это ссылка на объект **Application**, *fileFilter* — выражение **String**, форматированное для задания фильтров файла, перечисленных в раскрывающемся списке **Save as type**; *filterIndex* — численное выражение, указывающее, какой фильтр файла должен использовать VBA в качестве фильтра по умолчанию, а *title* — выражение **String** для строки заголовка диалогового окна. Если пропускается аргумент *title*, VBA отображает диалоговое окно с обычным заголовком **SaveAs**.

Аргумент *buttonText* — необязательный параметр типа **Variant**. Используется только в Macintosh.

Метод **GetSaveAsFilename** имеет еще один аргумент — *initialFilename*, который представляет любое допустимое имя файла. Если указать этот необязательный аргумент, имя файла, которое задается для *initialFilename*, появляется в текстовом окне **File Name**, когда диалоговое окно **Save As** отображается первый раз.

Строки фильтра файла для метода **GetSaveAsFilename** определяйте таким же образом, каким форматируете их для **GetOpenFilename**.

В листинге 8.6 показан пример, использующий метод **GetSaveAsFilename**.

#### Листинг 8.6. Использование **GetSaveAsFilename** для получения имени файла от пользователя

```

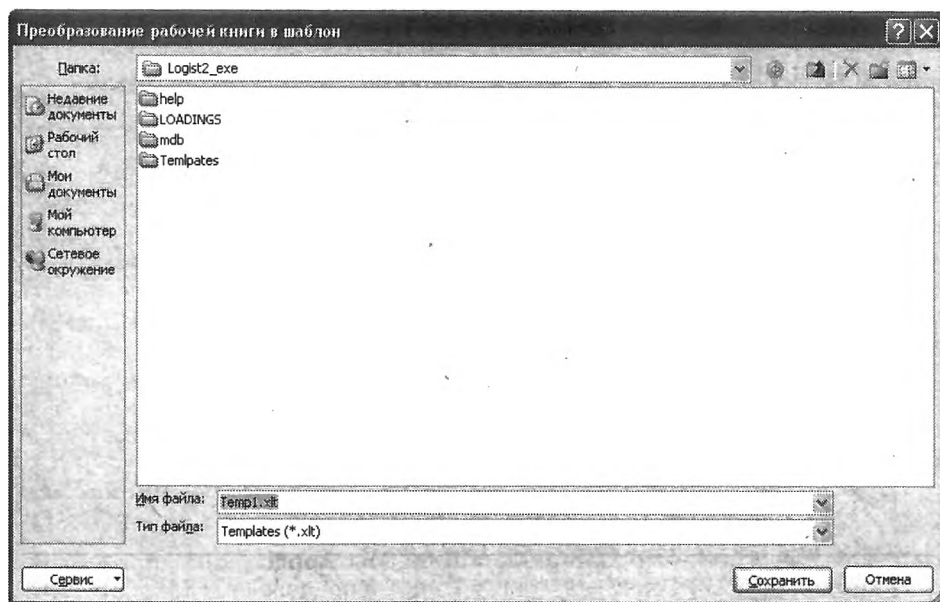
1: Sub ConvTemplate()
2:   'сохраняет текущую книгу как файл шаблона
3:
4:   Const FilterList := "Templates (*.xlt),*.xlt"
5:   Const iTitle = "Преобразование рабочей книги в шаблон"
6:   Static sN As String
7:   Static TCount As Variant
8:   Dim iName As String
9:
10:  If IsEmpty(TCount) Then
11:    TCount = 1
12:  Else
13:    TCount = TCount + 1
14:  End If
15:
16:  sN = "Temp" & CStr(TCount) & ".xlt"
17:  iName = Application.GetSaveAsFilename(initialfilename:=sN, _
18:    filefilter:=FilterList, _
19:    Title:=iTitle)
20:
21:  If iName = "False" Then
22:    MsgBox prompt:="Преобразование в шаблон отменено", _
23:      Title:=iTitle
24:  Else
25:    ActiveWorkbook.SaveAs Filename:=iName, _
26:      FileFormat:=xlTemplate
27:  End If
28: End Sub

```

Процедура **ConvTemplate** сохраняет текущую рабочую книгу как Excel-шаблонный файл и использует метод **GetSaveAsFilename** для получения имени для нового шаблонного файла. В строке 4 объявляется константа для фильтра файла; в строке 5 — константа для заголовков диалоговых окон, отображаемых этой процедурой.

В строках 6 и 7 объявляются **Static**-переменные: **sN** используется для имени нового шаблонного файла, а **TCount** — для нумерации шаблонных имен по умолчанию. Переменная **iName** используется для сохранения выбираемого пользователем имени файла.

В строках 10–16 устанавливается счетчик шаблонных файлов и формируется предлагаемое имя шаблонного файла. В строках 17–19 содержится один оператор, который вызывает метод **GetSaveAsFilename** и присваивает его результат переменной **iName**. При вызове этого метода VBA отображает диалоговое окно, показанное на рис. 8.6. Заметьте, что это окно точно такое же, как Excel-окно **Сохранение документа** (Save As), за исключением заголовка диалогового окна (поскольку он формируется процедурой и является аргументом метода **GetSaveAsFilename**) и содержимого списка **Тип файла** (Save as type).



**Рис. 8.6.** Процедура **ConvTemplate** использует метод **GetSaveAsFilename** для отображения этого диалогового окна

В строке 21 с помощью оператора **If...Then...Else** проверяется, не отменил ли пользователь диалоговое окно метода **GetSaveAsFilename**. Если пользователь выбрал кнопку **Cancel** (Отмена) [или нажал **Esc**], то выполняется оператор в строках 22–23, отображая окно сообщения, информирующее пользователя о том, что преобразование в шаблон было отменено. Иначе оператор в строках 25–26 вызывает метод **SaveAs**, сохраняя активную рабочую книгу как Excel-шаблонный файл.

Необходимо помнить, что и **GetOpenFilename**, и **GetSaveAsFilename** могут изменять текущий диск и папку. Можно использовать функцию **CurDir** для получения текущего диска и папки и сохранять результат в переменной, чтобы можно было переключаться снова на текущий диск и папку с помощью **ChDrive** и **ChDir** (описываются далее в этой главе).

## Использование встроенных диалоговых окон Word для получения имен файлов

Word не имеет тех же самых методов `GetOpenFilename` и `GetSaveAsFilename`, которые предоставляет Excel. Однако Word позволяет использовать VBA для отображения любых встроенных диалоговых окон — это те же самые диалоговые окна, которые отображаются, когда Word используется интерактивно. Можно, следовательно, применять собственные окна Word **Открытие документа** и **Сохранение документа (Save As)** в коде VBA для получения имен файлов от пользователя.

Использование встроенных диалоговых окон Word для получения имени файла от пользователя немного сложнее, чем использование Excel-методов `GetOpenFilename` и `GetSaveAsFilename`. Встроенные диалоговые окна Word доступны посредством свойства `Dialogs` Word-объекта `Application`. Свойство `Dialogs` возвращает коллекцию встроенных диалоговых окон Word. Для указания необходимого диалогового окна используются различные предопределенные константы Word (каждая из которых обозначается как `wdDialog` с последующим именем диалогового окна). Предопределенная Word-константа `wdDialogFileOpen` определяет Word-диалоговое окно **Открытие документа**, а константа `wdDialogFileSaveAs` — диалоговое окно **Сохранение документа**.

Для ссылки на одно из встроенных окон Word необходимо использовать следующий синтаксис:

### Синтаксис

---

```
Object.Dialogs(wdDialogConstant)
```

Здесь *Object* — объектная ссылка на Word-объект **Application**, которая является обязательной; *wdDialogConstant* — любая из предопределенных констант Word, которые определяют конкретное встроенное диалоговое окно.

---

Для просмотра списка констант, предопределенных для Word, и разделов справочной системы для этих констант используйте окно **Object Browser**. Word предоставляет более сотни констант — слишком много для подробного обсуждения в этой книге.

Для управления встроенными окнами Word необходимо сначала сослаться на конкретное диалоговое окно, которое будет использоваться, а затем вывести его на экран с помощью метода `Display` объекта `Dialog`. Метод `Display` отображает диалоговое окно; когда пользователь закрывает диалоговое окно, можно находить выбор пользователя из свойств объекта `Dialog`. При желании можно также задавать значения по умолчанию для установок диалогового окна, задавая свойства объекта `Dialog` перед отображением этого окна на экране.

Два наиболее важных свойства для диалоговых окон, определяемых константами `wdDialogFileOpen` и `wdDialogFileSaveAs`, являются также единственными общими для этих окон свойствами: `Name` и `Format`.

Свойство `Name` — это строка, сохраняющая имя файла, которое пользователь выбрал в окне **Открытие документа** или в окне **Сохранение документа**. Можно также предлагать имя файла в текстовом окне **Имя файла (File Name)** диалоговых окон

**Открытие документа и Сохранение документа**, присваивая значение свойству **Name** перед отображением этих диалоговых окон. Свойство **Format** указывает, какой тип документа открывается или сохраняется: документ или шаблон. Свойство **Format** может содержать значение одной из двух Word-констант типа файла: **wdTypeDocument** и **wdTypeTemplate**.

В целях сбора информации от пользователя для процедур VBA используйте метод **Display** объекта **Dialog**. Метод **Display** возвращает значение, указывающее, закрыл ли пользователь диалоговое окно щелчком кнопки **Открыть** или кнопки **Отмена**. В табл. 8.3 приведены возвращаемые значения метода **Display** и поясняется их смысл.

Таблица 8.3. Возвращаемые значения метода **Display**

Числовое значение	Что означает
-2	Диалоговое окно было закрыто с помощью кнопки <b>Отмена</b> .
-1	Диалоговое окно было закрыто щелчком кнопки <b>Открыть</b> в диалоговых окнах <b>wdDialogFileOpen</b> и <b>wdDialogFileSaveAs</b> .
0	Диалоговое окно было отменено.
> 0	Командная кнопка; 1 — для первой командной кнопки, 2 — для второй и так далее.

Теперь, когда вы понимаете основы использования Word-встроенных диалоговых окон **Открытие документа** и **Сохранение документа**, можно рассмотреть некоторые конкретные примеры работы этих окон.

Использование Word-диалогового окна **Open**

В листинге 8.7 показана процедура, использующая Word-встроенное диалоговое окно **Open** (Открытие документа) для получения имени файла документа от пользователя. Процедура открывает документ и помещает курсор в конец документа.

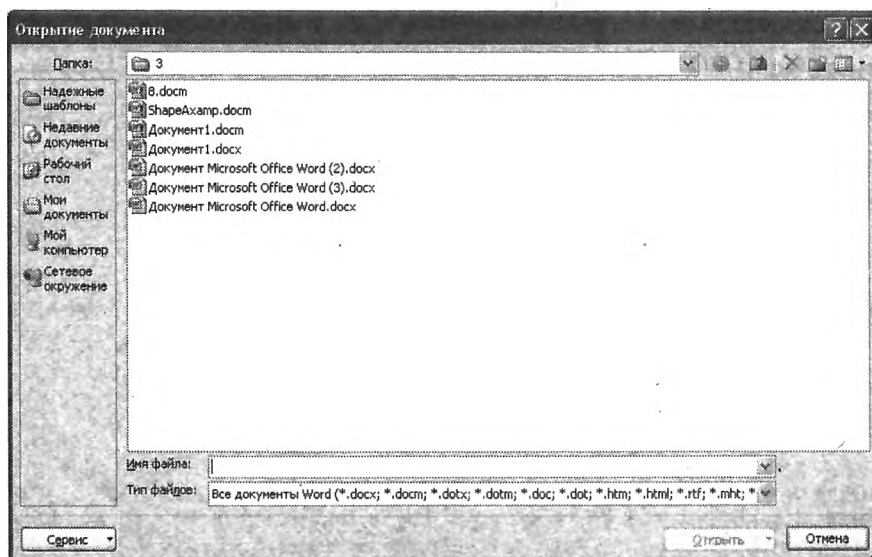
Листинг 8.7. Использование Word-встроенного диалогового окна **wdDialogFileOpen** для получения имени файла

```
1: Sub OpenWordFile()  
2: 'открывает указанный документ и перемещает  
3: 'указатель вставки в конец документа  
4:  
5:     Const iTitle = "Открытие файла"  
6:  
7:     Dim fName As String  
8:     Dim lRetVal As Long  
9:  
10:    With Application.Dialogs(wdDialogFileOpen)  
11:        lRetVal = .Display  
12:        fName = .Name  
13:    End With  
14:
```

```
15: If lRetVal <> -1 Then
16:     MsgBox prompt:="Операция отменена", _
17:         Title:=iTitle
18:     Exit Sub
19: End If
20:
21: Documents.Open FileName:=fName
22: Selection.EndKey unit:=wdStory
23:
24: MsgBox prompt:="Документ '" & fName & _
25:         "' открыт", _
26:         Title:=iTitle & " завершено"
27: End Sub
```

Процедура **OpenWordFile** использует только одну константу — **iTitle** — в качестве заголовков диалоговых окон, отображаемых этой процедурой (заголовок диалогового окна **Open** изменить невозможно). В строках 7 и 8 объявляются переменные, используемые процедурой: **fName** используется для сохранения имени файла, введенного пользователем, а **lRetVal** — для сохранения результата метода **Display**, чтобы процедура могла определить, отменил ли пользователь диалоговое окно Word, не делая выбора.

В строке 10 производится ссылка на диалоговое окно **Open** (Открытие документа), а строке 11 используется метод **Display** объекта **Dialog** для вывода на экран этого окна. В результате отображается диалоговое окно, показанное на рис. 8.7. Это окно идентично во всех отношениях окну, отображаемому Word, когда выбирается команда **Файл | Открыть (File | Open)**.



**Рис. 8.7.** Word-диалоговое окно **Open** (Открытие документа), отображаемое оператором в строке 11 листинга 8.7



Выполнение процедуры **OpenWordFile** приостанавливается на то время, пока открыто диалоговое окно; после закрытия окна **Открытие документа** (пользователем) значение, возвращаемое методом **Display**, сохраняется в **lRetVal** и выполнение кода возобновляется со строки 12. В этот момент свойство **Name** диалогового окна содержит имя файла, выбранное пользователем (или строку нулевой длины, если пользователь не выбрал имя файла). В строке 12 значение свойства **Name** диалогового окна сохраняется в **fName**.

В строках 15–19 оператор **If** оценивает возвращаемый методом **Display** результат, который был сохранен в переменной **lRetVal**. Значение **-1** указывает, что диалоговое окно было закрыто щелчком на кнопке **Открыть** (или эквивалентным действием, таким как двойной щелчок на имени файла). Если окно не было закрыто щелчком на кнопке **Открыть** или ее эквиваленте, то отображается сообщение о том, что операция была отменена, иначе — выполнение кода продолжается со строки 21.

В строке 21 выбранный файл открывается, а в строке 22 курсор помещается в конец документа. Наконец, в строках 24–26 отображается сообщение об успешном завершении операции.

## Использование Word-диалогового окна Сохранение документа

В листинге 8.8 показана процедура, использующая Word-встроенное диалоговое окно **Сохранение документа** (**Save As**) для получения имени файла документа от пользователя, предлагающее формат шаблона документа для файла, который должен быть сохранен. Затем процедура сохраняет документ как **.dot**-шаблон.

**Листинг 8.8.** Использование Word-встроенного диалогового окна **wdDialogFileSaveAs** для получения имени файла

---

```
1: Sub ConvWordTemplate()  
2: 'сохраняет текущий документ как файл шаблона  
3:  
4:   Const iTitle = "Преобразовать документ в шаблон"  
5:   Static sName As String  
6:   Static TCount As Variant  
7:  
8:   Dim iNameas As String  
9:   Dim lRetVal As Long  
10:  
11:  If IsEmpty(TCount) Then  
12:    TCount = 1  
13:  Else  
14:    TCount = TCount + 1  
15:  End If  
16:  
17:  sName = "Temp" & CStr(TCount) & ".dot"  
18:  With Application.Dialogs(wdDialogFileSaveAs)  
19:    .Name = sName  
20:    .Format = wdTypeTemplate  
21:    lRetVal = .Display  
22:    iName = .Name  
23:  End With
```

```
24:
25:   If lRetVal <> -1 Then
26:     MsgBox prompt:="Преобразование отменено",
27:           Title:=iTitle
28:   Else
29:     ActiveDocument.SaveAs FileName:=iName,
30:                           fileformat:=wdTypeTemplate
31:   End If
32: End Sub
```

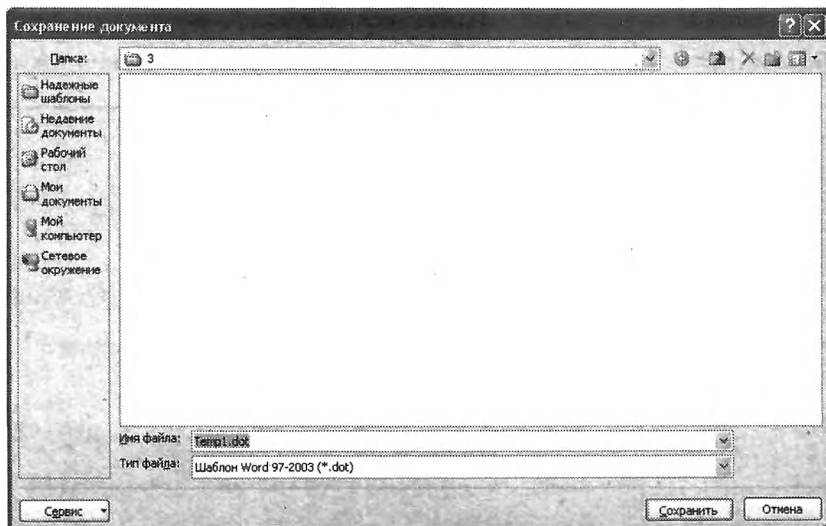
Процедура **ConvWordTemplate** сохраняет текущий документ как шаблонный Word-файл и использует диалоговое окно **Сохранение документа** для получения имени нового шаблонного файла.

В строках 11–17 устанавливается счетчик шаблонных файлов и формируется предлагаемое имя файла. В строке 18 начинается оператор **With**, который ссылается на диалоговый объект **Application.Dialogs(wdDialogFileSaveAs)**, то есть диалоговое окно, отображаемое Word, когда выбирается команда **Файл | Сохранить как (File | Save As)**.

В строках 19 и 20 устанавливаются некоторые предлагаемые умолчания для окна **Сохранение документа**. В строке 19 задается свойство **Name** диалогового окна, чтобы оно содержало предлагаемое имя файла шаблона (в **sName**). В строке 20 устанавливается свойство **Format** диалогового окна в значение **wdTypeTemplate**, это приводит к тому, что в диалоговом окне **Сохранение документа** в списке **Тип файла (Save as type)** выбирается тип файла **Шаблон документа (Document Template)** в качестве типа по умолчанию.

При выполнении оператора в строке 21 VBA отображает диалоговое окно, показанное на рис. 8.8. Заметьте, что это окно точно такое же, как Word-диалоговое окно **Сохранение документа**. Как только пользователь делает выбор в окне **Сохранение документа**, возвращаемое значение метода **Display** сохраняется в **lRetVal** и выполнение кода продолжается со строки 22, где выбранное пользователем имя файла (из свойства **Name** диалогового окна) помещается в переменную **iName**.

**Рис. 8.8**  
Word-диалоговое окно, отображаемое оператором в строке 21 листинга 8.8



В строках 29–30 документ сохраняется с использованием имени файла (в `iName`) и формата файла (`wdTypeTemplate`).

## Работа с дисками и папками

В этом разделе рассматривается, как использовать функции и операторы VBA для нахождения текущего диска и папки, как изменять текущий диск или папку и как создавать или удалять подпапки. Помните, *текущий диск* (*current drive*) — это диск, который использует Word или Excel, когда не указывается определенная буквенная метка диска. Аналогично, *текущая папка* (*current folder*) — это папка диска, которую Excel (или Word) использует, если не указывается определенная папка.

### Получение пути текущей папки и буквенной метки диска

Выборка текущего диска и пути папки довольно проста. Обе части информации можно получить, используя функцию `CurDir`. Функция `CurDir` возвращает строку, которая содержит полный путь текущей папки, включая буквенную метку диска. (`CurDir` является аббревиатурой слов *current directory*; помните, что папка и каталог — это одно и то же.)

Функция `CurDir` имеет следующий синтаксис:

#### Синтаксис

---

`CurDir[(drive)]`

Здесь *drive* представляет любое выражение типа **String** и указывает функции `CurDir`, текущая папка какого диска необходима пользователю. Если аргумент *drive* опущен, `CurDir` возвращает текущую папку текущего диска. Обычно *drive* содержит только одну букву; если пользователь передает строку с несколькими символами, `CurDir` использует первый символ строки как буквенную метку диска.

---

В листинге 8.9 показан пример использования функции `CurDir`.

#### Листинг 8.9. Использование функции `CurDir` для получения текущего каталога и диска

---

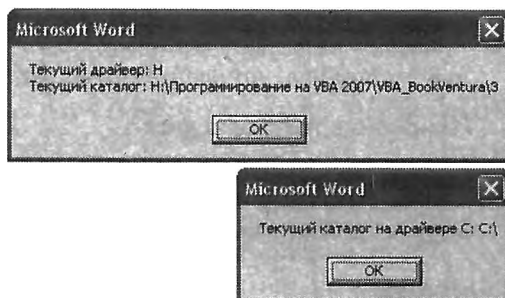
```
1: Sub ShowCurDriveDir()  
2: 'отображает текущий драйвер и каталог  
3:  
4: Dim DirName As String 'для имени каталога  
5: Dim DirLetter As String 'для буквы диска  
6:  
7: DirName = CurDir() 'получить текущую папку диска  
8:  
9: 'взять первый символ из DirName - буква диска:  
10: DirLetter = Left(DirName, 1)  
11:  
12: 'вывод на экран:  
13: MsgBox "Текущий драйвер: " & DirLetter & _  
14: Chr(13) & "Текущий каталог: " & DirName  
15:
```

```
16: 'получить текущий каталог на драйвере C:
17: DirName = CurDir("C")
18: MsgBox "Текущий каталог на драйвере C: " &
19: DirName
20: End Sub
```

Наличие большого количества комментариев и простота синтаксиса функции **CurDir**, видимо, не требуют анализа данного листинга. В дополнение можно предложить один из возможных результатов работы процедуры (на рис. 8.9).

**Рис. 8.9**

Такие окна могут получиться в результате работы кода из листинга 8.9



## Изменение текущей папки

Если вам необходимо, чтобы процедура VBA изменила текущую папку на какую-либо другую, используйте оператор **ChDir**. Если у вас есть опыт работы с DOS, вы поймете, что **ChDir** в VBA выполняет ту же задачу, что DOS-команды **CHDIR** и **CD**. (**ChDir** — это аббревиатура слов *change directory*).

Оператор **ChDir** имеет следующий синтаксис:

### Синтаксис

**ChDir** *path*

Аргумент *path* представляет любое выражение типа **String**, имеющее результатом допустимый путь папки; *path* может (необязательно) содержать буквенную метку диска. Если пользователь включает буквенную метку в аргумент *path*, **ChDir** изменяет текущую папку диска на указанную в *path* без изменения текущего диска.

В листинге 8.10 показан пример использования оператора **ChDir**.

### Листинг 8.10. Использование **ChDir**

```
1: Sub Test_ChDir()
2: 'демонстрирует оператор ChDir
3:
4: Dim oldDir As String 'для текущего каталога
5:
6: MsgBox "Текущий каталог: " & CurDir()
7:
8: 'сохранить текущий (старый) каталог:
```

```
9:   oldDir = CurDir()
10:
11:   'изменить каталог:
12:   ChDir "\Мои документы"
13:
14:   'вывести на экран новый каталог:
15:   MsgBox "Новый каталог: " & CurDir()
16:
17:   'вернуться к старому каталогу
18:   ChDir oldDir
19:   MsgBox "Текущий каталог: " & CurDir()
20:
21: End Sub
```

---

Эта процедура, по всей видимости, также не нуждается в комментариях.

### Изменение текущего диска

Для изменения текущего диска необходимо использовать оператор **ChDrive**. (**ChDrive** — это аббревиатура слов *change drive*.)

Оператор **ChDrive** имеет следующий синтаксис:

### Синтаксис

---

**ChDrive** *drive*

Аргумент *drive* — это любое выражение типа **String**, представляющее буквенную метку диска. Если *drive* содержит более одного символа, **ChDrive** использует только первый символ в строке для буквенной метки. Если для аргумента *drive* задается пустая строка, то текущий диск не изменяется. Если задается символ, не являющийся одной из букв от **A** до **Z**, VBA отображает runtime-ошибку. VBA отображает также runtime-ошибку, если задается буквенная метка для диска, которого фактически нет в данной компьютерной системе, хотя вы можете использовать буквенные метки дисков, соединенных с вашим компьютером через сеть.

---

В листинге 8.11 демонстрируется использование оператора **ChDrive**.

### Листинг 8.11. Использование **ChDrive** для изменения текущего диска

---

```
1: Sub Test_ChDrive()
2:   Dim oldDir As String
3:
4:   'запомнить текущий (старый) каталог:
5:   oldDir = CurDir()
6:
7:   'выдать на экран текущий каталог:
8:   MsgBox "Текущий каталог: " & oldDir
9:
10:  'изменить драйвер на A:
11:  ChDrive "D"
12:
13:  'выдать на экран новый драйвер и каталог:
14:  MsgBox "Новый драйвер и каталог: " & CurDir
15:
16:  'восстановить старый драйвер (используется
```

```
17: 'только первый символ строки, содержащий драйвер):
18: ChDrive oldDir
19:
20: 'восстановить старый каталог:
21: ChDir oldDir
22:
23: MsgBox "Текущий каталог: " & CurDir()
24: End Sub
```

---

## Создание дисковых папок

Вам может понадобиться, чтобы одна из процедур создавала новую дисковую подпапку для сохранения в ней новых файлов рабочей книги или документа или по каким-либо другим причинам. Для создания дисковой папки используйте оператор **MkDir**. Оператор **MkDir** выполняет ту же задачу, что DOS-команды MKDIR или MD или команда WINDOWS File | New | Folder (Файл | Создать | Папка). (**MkDir** — это аббревиатура слов *make directory*.)

Оператор **MkDir** имеет следующий синтаксис:

### Синтаксис

---

**MkDir** *path*

Аргумент *path* представляет любое выражение типа **String**, которое имеет результатом допустимый путь папки; *path* может (необязательно) содержать буквенную метку диска. Если буквенная метка диска не включена в аргумент *path*, **MkDir** создает новую папку на текущем диске. Если *path* указывает имеющуюся уже папку или включает недопустимые символы имени файла, VBA отображает сообщение о runtime-ошибке. При попытке создать подпапку в папке, которой не существует, VBA также отображает сообщение о runtime-ошибке. Использование **MkDir** не изменяет текущего диска или папки.

---

В листинге 8.12 показан пример использования оператора **MkDir**, не требующий никакого анализа.

### Листинг 8.12. Использование **MkDir** для создания нового дискового каталога

---

```
1: Sub Test_MkDir()
2: 'демонстрирует оператор MkDir
3:
4: Dim newDir As String
5:
6: 'новый (будущий) каталог:
7: newDir = "A:\test1"
8:
9: 'создать новый каталог:
10: MkDir newDir
11:
12: 'перейти к новому (только что созданному) каталогу:
13: ChDir newDir
14:
15: 'проверить результат перехода к новому каталогу:
16: MsgBox "Текущий каталог драйвера A: " & _
```

```
17:                                     CurDir("A")
18: End Sub
```

---

Перед запуском этой процедуры убедитесь, что в дисковом **A:** имеется диск, иначе VBA отобразит сообщение о runtime-ошибке.

### Удаление дисковых папок

Вам может понадобиться также, чтобы одна из процедур удаляла дисковую папку. Для удаления дисковой папки используйте оператор **RmDir**. Оператор **RmDir** выполняет ту же задачу, что и DOS-команды **RMDIR** или **RD**; в Windows используется команда **File | Delete** (Файл | Удалить) для удаления как файлов, так и папок. (**RmDir** — это аббревиатура слов *remove directory*.)

Оператор **RmDir** имеет следующий синтаксис:

### Синтаксис

---

**RmDir** *path*

Аргумент *path* представляет любое выражение **String**, имеющее результатом допустимый путь папки; *path* может (необязательно) содержать буквенную метку диска. Если буквенная метка не включена в аргумент *path*, **RmDir** удаляет папку на текущем диске. Если *path* определяет папку, которая еще не существует, или включает недопустимые символы имени файла, то VBA отображает сообщение о runtime-ошибке.

---

В листинге 8.13 показан простой пример использования оператора **RmDir**.

### Листинг 8.13. Использование **RmDir** для удаления папки

---

```
1: Sub Test_RmDir()
2:   'демонстрирует оператор RmDir
3:
4:   Dim delDir As String
5:
6:   'имя удаляемой папки:
7:   delDir = "A:\test1"
8:
9:   'если удаляемая папка - текущая, перейти
10:  'в корневую; НЕЛЬЗЯ УДАЛЯТЬ ТЕКУЩУЮ ПАПКУ
11:  If CurDir("A") = delDir Then ChDir "A:\"
12:
13:  'удалить папку A:\test1
14:  RmDir delDir
15:
16: End Sub
```

---

Предполагается, что была выполнена процедура в листинге 8.12 для создания папки **Test1** для диска **A:**. Если текущая папка диска **A:** является той же папкой, которая должна быть удалена, то в строке 11 вызывается функция **ChDir** для изменения текущей папки диска **A:** на корневую папку. Нельзя удалять дисковую папку, если она является текущей или пустой.

## Копирование и удаление файлов

Копирование и удаление файлов — это, вероятно, две наиболее часто выполняемые операции по управлению файлами. В этом разделе описывается, как можно копировать или удалять файлы под контролем процедур VBA.

### Копирование файлов

Для копирования файла используйте оператор **FileCopy**. Этот оператор VBA эквивалентен DOS-команде **COPY** или команде **File | Copy** (Файл | Копировать) в Windows.

Оператор **FileCopy** имеет следующий синтаксис:

#### Синтаксис

---

`FileCopy source, destination`

Как *source*, так и *destination* являются выражениями типа **String**, имеющие результатом допустимые имена файла. Они могут (необязательно) включать полный путь папки и буквенную метку диска. При попытке копировать файл в самого себя VBA выдает runtime-ошибку. VBA также выдает runtime-ошибку при попытке копировать файл, когда нет достаточного дискового пространства для сохранения копируемого файла.

---

В листинге 8.14 используется оператор **FileCopy**. Процедуру следует тестировать в Excel, так как она использует методы **GetOpenFilename** и **GetSaveAsFilename**.

#### Листинг 8.14. Процедура, использующая оператор **FileCopy**

---

```
1: Sub Test_CopyFiles()  
2: 'копирует файл, указанный пользователем, в файл  
3: 'с новым именем, драйвером или каталогом  
4:  
5: Dim sName As String  
6: Dim dName As String  
7:  
8: Do  
9: With Application  
10: sName = .GetOpenFilename(Title:=  
11: "File Copy - Source")  
12:  
13: If sName = "False" Then Exit Sub  
14:  
15: dName = .GetSaveAsFilename(Title:=  
16: "File Copy - Destination")  
17:  
18: If dName = "False" Then Exit Sub  
19:  
20: End With  
21:  
22: FileCopy Source:=sName, Destination:=dName  
23:
```



```
24: Loop
25:
26: End Sub
```

---

Эта процедура дает возможность пользователю выбирать файл-источник, диск, папку и имя файла-приемника, в который будет осуществляться копирование (иногда такой файл называют *целевым*). В строках 5 и 6 объявляются две переменные типа **String** для имен исходного и целевого файлов. В строке 8 начинается бесконечный цикл **Do** (цикл без детерминантного условия). Цикл в строках 8–24 выполняется до тех пор, пока пользователь не отменит одно из двух файловых диалоговых окон. В строке 9 начинается оператор **With**. В строках 10–11 содержится один оператор, вызывающий метод **GetOpenFilename** для отображения диалогового окна открытия файла и присваивающий имя файла (результат метода) переменной **sName**.

В строке 13 оператор проверяет, отменил ли пользователь диалоговое окно открытия файла. Если — да, процедура заканчивается. Иначе VBA выполняет оператор в строках 15 и 16, который использует метод **GetSaveAsFilename**, чтобы дать возможность пользователю выбрать имя файла, диск и папку для копируемого файла. В строке 18 проверяется, отменил ли пользователь диалоговое окно открытия файла. Если — да, выполнение процедуры прекращается. Иначе VBA продолжает выполнять код с оператора **FileCopy** в строке 22.

Когда VBA выполняет оператор **FileCopy** в строке 22, файл (имя которого сохранено в **sName**) копируется с новым именем на новый диск и в новую папку, сохраненные в **dName**. После копирования файла цикл **Do** повторяется.

## Удаление файла

Для удаления файла используйте оператор с драматическим именем **Kill**. Оператор **Kill** выполняет ту же задачу, что DOS-команда **DEL** или команда **File | Delete** (Файл | Удалить) в Windows.

Оператор **Kill** имеет следующий синтаксис:

### Синтаксис

---

**Kill** *pathname*

Аргумент *pathname* — это любое выражение типа **String**, имеющее результатом допустимую спецификацию имени файла; *pathname* может включать буквенную метку диска, полный путь папки и символы универсального сопоставления (\* и ?). Если *pathname* включает символы универсального сопоставления, **Kill** удаляет все файлы, совпадающие со спецификацией в *pathname*.

---

В листинге 8.15 показан пример оператора **Kill**.

**Листинг 8.15.** Использование Kill для удаления файлов

```
1: Sub Test_KillFiles()  
2: 'удаляет файлы, пока пользователь не отменит диалог  
3:  
4:   Dim fName As String  
5:   Dim Ans As Integer  
6:  
7:   Do  
8:  
9:     fName = Application.GetOpenFilename(Title:= _  
10:      "Delete File")  
11:  
12:    If fName = "False" Then Exit Sub  
13:  
14:    Ans = MsgBox(prompt:="Delete " & fName & "?", _  
15:      Title:="Delete File", _  
16:      Buttons:=vbQuestion + vbYesNo)  
17:  
18:    If Ans = vbYes Then  
19:      Kill fName  
20:    End If  
21:  
22:  Loop  
23: End Sub
```

Эта процедура использует метод **GetOpenFilename**, дающий возможность пользователю выбрать имя файла, подтверждает удаление, удаляет файл. Эти действия повторяются в цикле.

В строке 7 начинается бесконечный цикл **Do**, который выполняется до тех пор, пока пользователь не отменит файловое диалоговое окно. В строке 9 вызывается метод **GetOpenFilename** и его результат присваивается переменной **fName**. В строке 12 проверяется, отменил ли пользователь файловое диалоговое окно, и процедура завершается, если пользователь его отменил. В строках 14–16 содержится оператор **MsgBox**, запрашивающий пользователя подтвердить удаление выбранного файла. Если пользователь подтверждает удаление, VBA выполняет оператор **Kill** (в строке 19), который удаляет файл.

Не следует пытаться удалить файл открытой рабочей книги или документа, в таком случае VBA выдает runtime-ошибку. Проверяйте атрибуты файла перед попыткой удалить его. При попытке удалить файл, имеющий какой-либо из атрибутов **Hidden**, **System** или **Read-Only** VBA выдает runtime-ошибку. Используйте функцию **GetAttr** для выборки атрибутов файла и оператор **SetAttr** — для их изменения при необходимости удалять файлы с атрибутами **Hidden**, **System** или **Read-Only**.

## Переименование или перемещение файлов

Вам может понадобиться изменить имя файла или переместить файл в другую папку на том же диске. Для переименования файла или перемещения его в другую папку используйте оператор **Name**.

Оператор **Name** имеет следующий синтаксис:

### Синтаксис

---

`Name oldpathname As newpathame`

Аргументы *oldpathname* и *newpathname* — это выражения типа **String**, имеющие результатом допустимые имена файлов. Оба могут (необязательно) содержать полный путь папки, включая буквенную метку диска. Если пользователь включает буквенную метку, оба выражения *oldpathname* и *newpathname* должны включать одну и ту же буквенную метку диска, иначе VBA выдает runtime-ошибку. Если *oldpathname* и *newpathname* ссылаются на разные папки, VBA перемещает файл в новую папку и изменяет его имя.

---

В листинге 8.16 показана процедура, использующая оператор **Name** для переименования файлов.

### Листинг 8.16. Использование **Name** для переименования или перемещения файлов

---

```
1: Sub Test_RenameOrMoveFile()  
2: 'переименовывает или перемещает файл  
3:  
4:   Const iTitle = "Переименовать или удалить - "  
5:   Dim oldName As String  
6:   Dim newName As String  
7:   Dim oldDir As String  
8:  
9:   oldDir = CurDir()  
10:  With Application  
11:    oldName = .GetOpenFilename(Title:=iTitle & "Источник")  
12:    If oldName = "False" Then Exit Sub  
13:  
14:    newName = .GetSaveAsFilename(InitialFilename:=oldName,  
15:                                Title:=iTitle & "Новое имя")  
16:    If newName = "False" Then Exit Sub  
17:  End With  
18:  
19:  If Left(oldName, 1) = Left(newName, 1) Then  
20:    Name oldName As newName  
21:  Else  
22:    FileCopy oldName, newName  
23:    Kill oldName  
24:  End If  
25:  
26:  ChDrive oldDir  
27:  ChDir oldDir  
28: End Sub
```

---

Эта процедура переименовывает или перемещает файлы. Пользователь процедуры выбирает имя файла, папку и диск для оригинального файла и для нового имени и/или местоположения.

В строке 9 вызывается функция **CurDir**, результат которой сохраняется в переменной **oldDir**. В строке 10 начинается оператор **With** для объекта

**Application.** В строке 11 используется метод **GetOpenFilename** для того, чтобы пользователь мог выбрать файл, который будет переименован или перемещен. В строке 12 используется оператор **If** для проверки того, отменил ли пользователь диалоговое окно **GetOpenFilename**; если — да, то процедура прекращается.

В строке 14 вызывается метод **GetSaveAsFilename** для того, чтобы пользователь мог выбрать новое имя файла, диск и папку. Заметьте, что этот вызов **GetSaveasFilename** использует аргумент **InitialFilename** для заполнения предлагаемого нового имени файла. В строке 16 проверяется, отменил ли пользователь это диалоговое окно, и, если — отменил, процедура заканчивается.

В строке 19 начинается оператор **If...Then...Else**, который проверяет, выбрал ли пользователь другой диск для перемещения на него файла. Логическое выражение для оператора **If** в строке 19 использует функцию **Left** для возвращения первой буквы строк **oldName** и **newName**. Если эти две буквы одинаковые, то пользователь переименовывает или перемещает файл на том же самом диске и VBA выполняет оператор **Name** в строке 20 для переименования файла. Если пути папок в **oldName** и **newName** различны, **Name** перемещает файл в новую папку.

Если первые буквы **oldName** и **newName** различные, то пользователь выбрал перемещение файла на другой диск. Нельзя использовать **Name** для перемещения файлов на другой диск, поэтому процедура использует **FileCopy** для копирования файла на другой диск и затем использует **Kill** для удаления оригинального файла.

Наконец, операторы **ChDrive** и **ChDir** восстанавливают оригинальный диск и папку, которые были текущими при запуске процедуры. (Помните, методы **GetOpenFilename** и **GetSaveAsFilename** могут изменить текущий диск и папку.)

## Получение информации о файлах

Иногда бывает важно знать размер файла или дату и время, когда файл был модифицирован последний раз. Например, имея процедуру, резервирующую файлы рабочей книги, можно написать ее так, чтобы она проверяла, не заменяет ли она файл более старой версией.

### Получение времени и даты создания/модификации файла

Всякий раз, когда одна из программ приложения, такая как Word или Excel, изменяет дисковый файл, файловая система Windows записывает дату и время (в соответствии с часами данной компьютерной системы) с файлом, чтобы можно было определить, когда этот файл был модифицирован последний раз. Чтобы сделать эту информацию доступной в процедурах VBA, используйте функцию **FileDateTime**. Эта функция возвращает информацию о дате и времени из файла как VBA-значение типа **Date**.

Функция **FileDateTime** имеет следующий синтаксис:

### Синтаксис

---

**FileDateTime** (*pathname*)

Аргумент *pathname* — это любое выражение типа **String**, имеющее результатом допустимую спецификацию файла; *pathname* может необязательно включать буквенную метку диска и полный путь папки, но не может содержать символы универсального сопоставления (\* или ?).

---

В листинге 8.17 в следующем разделе показан пример, использующий функцию **FileDateTime**.

### Получение длины файла

Для того чтобы определить длину файла, используйте функцию **FileLen**. **FileLen** возвращает длину файла в байтах.

Функция **FileLen** имеет следующий синтаксис:

### Синтаксис

---

**FileLen** (*pathname*)

Аргумент *pathname* — это выражение типа **String**, имеющее результатом допустимую спецификацию имени файла; *pathname* может (необязательно) включать буквенную метку диска и полный путь папки, но не может включать символы универсального сопоставления. Если *pathname* определяет открытый файл, **FileLen** возвращает размер файла, каким он был, когда файл был сохранен на диске последний раз.

---

В листинге 8.17 показан пример, использующий функцию **FileLen**, а на рис. 8.10 — один из возможных результатов работы кода этого листинга.

### Листинг 8.17. Использование функций **FileDateTime** и **FileLen**

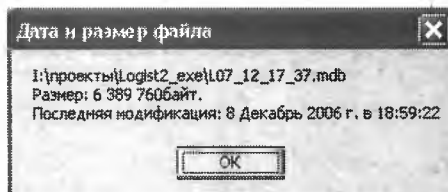
---

```
1: Sub ShowFDS(sName As String)
2:   'отображает размер и дату последней модификации файла
3:
4:   Dim msg1 As String
5:   Dim msg2 As String
6:   Dim fDate As Date
7:   Dim fLen As Long
8:
9:   fDate = FileDateTime(sName)
10:  fLen = FileLen(sName)
11:  msg1 = "Размер: " & Format(fLen, "###,###,###") & _
12:    "байт."
13:  msg2 = "Последняя модификация: " & _
14:    Format(fDate, "long date") & _
15:    " в " & Format(fDate, "long time")
16:  MsgBox Title:="Дата и размер файла", _
17:    prompt:=sName & Chr(13) & _
```

```
18:          msg1 & Chr(13) & msg2
19: End Sub
20:
21: Sub Test_ShowFDS()
22:   Dim sName As String
23:
24:   Do
25:     With Application
26:       sName = .GetOpenFilename(Title:="Дата/Размер файла")
27:     End With
28:     If sName <> "False" Then ShowFDS sName
29:   Loop Until sName = "False"
30: End Sub
```

**Рис. 8.10**

Процедура **ShowFDS** использует функции **FileLen** и **FileDateTime** для сбора информации о размере файла и о том, когда он был модифицирован последний раз



Процедура **ShowFDS** использует функции **FileLen** и **FileDateTime** для отображения окна сообщения, показывающего текущий размер файла и дату и время, когда этот файл модифицировался последний раз. **ShowFDS** имеет единственный аргумент **fName** типа **String**, используемый для указания процедуре **ShowFDS**, информация о каком файле должна быть отображена.

В строках 4–7 объявляется несколько переменных этой процедуры. Первые две переменные (**msg1** и **msg2**) используются для формирования текста сообщения, отображаемого процедурой. Переменные **fDate** и **fLen** используются для сохранения информации о дате и времени и информации о размере файла, соответственно. Заметьте, что **fLen** имеет тип **Long**, потому что длина файла может исчисляться миллионами байт.

В строке 9 вызывается функция **FileDateTime** с аргументом **sName** в качестве имени файла, а результат функции сохраняется в **fDate**. В строке 10 вызывается функция **FileLen** также с использованием **sName** для определения аргумента имени файла, а результат этой функции сохраняется в **fLen**.

В строках 11 и 12 содержится один оператор, который формирует первую часть сообщения, отображаемого этой процедурой, и сохраняет ее в **msg1**. В строках 13–15 содержится также один оператор, который формирует вторую часть сообщения, отображаемого этой процедурой, и сохраняет ее в **msg2**. Наконец, оператор **MsgBox** в строках 16–18 отображает имя файла, его размер и дату и время, когда этот файл модифицировался последний раз.

В строках 21–30 содержится процедура, тестирующая процедуру **ShowFDS**. Эта процедура использует метод **GetOpenFilename**, чтобы пользователь мог выбрать файл, а затем вызывает процедуру **ShowFDS**.

# Элементы диалоговых окон

До сих пор вы учились использовать диалоговые окна, которые встроены в VBA, а именно, функции **MsgBox** и **InputBox**. Хотя **MsgBox** и **InputBox** придают вашим программам гибкость, которой могут обладать только интерактивные программы, их возможности в известной степени ограничены. При разработке более сложных программ вы захотите выводить диалоговые окна, которые дают возможность пользователям ваших программ задавать при помощи одного диалогового окна сразу несколько опций, выбирать пункты из списка или вводить в одном окне несколько значений подобно тому, как это можно делать с помощью диалоговых окон, выводимых Word, Excel и другими приложениями Windows. Часто у вас будет возникать необходимость вместо встроенных окон, принадлежащих Word или Excel, использовать диалоговые окна, созданные специально для вашей программы.

VBA позволяет создавать и применять пользовательские<sup>1</sup> диалоговые окна в написанных вами программах и процедурах при помощи добавления в проект объекта **UserForm**. Используя VBA-формы пользователя (VBA User Forms), вы можете создавать диалоговые окна для вывода данных или получения значений от пользователя вашей программы именно в том виде, который требуется вашей программе. Например, вы можете вывести на экран диалоговое окно со списком различных вариантов формата даты и предоставить пользователю возможность выбрать из списка один из форматов.

Диалоговые окна позволяют вашей программе общаться с пользователем наиболее удобным образом, обеспечивая гибкую форму ввода и вывода данных. Из этой главы вы узнаете об основных элементах диалоговых окон и их свойствах, научитесь создавать простые диалоговые окна.

## Формы пользователя

Диалоговое окно в VBA создается добавлением в проект объекта **UserForm**. Объект **UserForm** — это пустое диалоговое окно. Настройку диалогового окна можно выполнить добавлением к объекту **UserForm** (обычно называют просто *форма*) элементов управления. Каждому объекту **UserForm** присущи определенные свойства, методы и события, которые он наследует от класса объектов **UserForm**. Каждый объект **UserForm** включает в себя также *модуль класса*, в который вы можете добавлять собственные методы и свойства или код обработки событий формы.

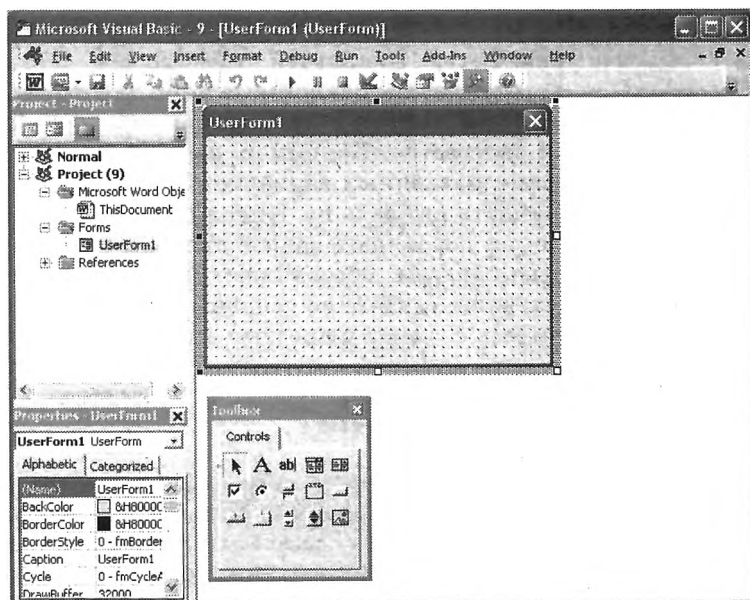
<sup>1</sup> В переводной литературе встречается термин «настраиваемые» (от англ. custom).

Первый шаг в создании пользовательского диалогового окна состоит в добавлении к проекту новой формы (объекта **UserForm**). Форма содержит рабочую область, в которую вы можете поместить элементы, необходимые для выполнения некоторого диалога пользователя с приложением. Поскольку объекты **UserForm** хранятся в коллекции **UserForms** проекта, они являются частью проекта.

Для добавления к проекту новой формы используйте команду VB-редактора **Insert | UserForm** (Вставка | UserForm). Редактор VB добавляет к текущему проекту новую форму, присваивая ей по умолчанию имя **UserFormN** и используя ту же систему нумерации, что и для модулей. Редактор VB выводит новую форму в *режиме разработки*, как показано на рис. 9.1. [В режиме разработки вы можете добавлять (или удалять) элементы управления к форме, устанавливать свойства формы или ее элементов управления и выполнять другие манипуляции с внешним видом формы в интерактивном режиме. Когда форма выведена и используется как часть выполняющейся программы, она находится в *режиме выполнения*.]

На рис. 9.1 показана добавленная в проект форма **UserForm1**. Заметим, что широкая рамка вокруг формы указывает на то, что форма выделена. Обратите внимание на сетку из точек на поверхности формы и **Панель элементов (Toolbox)**. Сетка из точек помогает выравнивать и контролировать размеры элементов управления, помещаемых на форму, и появляется только в режиме разработки. **Панель элементов** является «палитрой», с помощью которой вы можете выбирать элементы управления и добавлять их к форме. Панель, как правило, появляется только в том случае, если выбрана форма или один из ее элементов управления.

**Рис. 9.1**  
Новая UserForm  
в режиме разработки



Вы можете переименовать объект **UserForm** так же, как стандартный модуль или модуль класса. Для этого отредактируйте в **Properties Window** (окно свойств) свойство **Name** этого объекта.



Не оставляйте вновь созданной форме имя, присвоенное ей по умолчанию, например **UserForm1** или **UserForm2**. Напротив, переименовывайте новую форму сразу, как только ее создали.

Когда форма выводится на экран в режиме разработки, вы можете протестировать ее поведение, используя команду **Run | Run Sub/UserForm** (Запуск | Запуск подпрограммы/UserForm). После этого редактор VB выведет форму в режиме выполнения и все ее элементы управления будут активными. Однако следует помнить, что любой код, используемый формой, хранящийся не в модуле класса формы, не может быть инициализирован. При запуске формы инициализируется только код, находящийся в модуле класса формы. Переменные в стандартных модулях необязательно будут инициализироваться. В результате этого, если форма не является полностью независимой, некоторые из программ связанных с ней, могут не выполняться, выдавая сообщения о различных runtime-ошибках.

## Свойства объекта UserForm

Форма как объект имеет некоторые встроенные свойства, и вы можете устанавливать эти свойства или программным образом, или в **Properties Window** (окне свойств) редактора VB. Строго говоря, данные способы изменения свойств форм не являются эквивалентными. Некоторые из свойств могут быть установлены только посредством **Properties Window**. Программным способом свойства форм устанавливаются таким же образом, как и свойства других объектов: путем присвоения свойству нового значения. В таблице 9.1 перечислены свойства **UserForm**, на которые вам, скорее всего, придется ссылаться или изменять их.

**Таблица 9.1. Наиболее часто используемые свойства объектов UserForm**

Свойство	Описание
ActiveControl	Возвращает объектную ссылку на элемент управления, находящийся в фокусе в данный момент. Только для чтения.
BackColor	Целое типа <b>Long</b> определяет цвет фона формы. Самый простой способ установить это свойство — использовать <b>Properties Window</b> ; чтобы выбрать желаемый цвет (если необходимо), можно скопировать номер цвета из <b>Properties Window</b> в свою программу.
Caption	Текст, выводимый в качестве заголовка формы. Запись/Чтение.
Controls	Возвращает коллекцию всех элементов управления формы. Только для чтения.
Cycle	Определяет, должно ли нажатие клавиши табуляции вызывать последовательный выбор всех элементов управления во всех группах и на каждой странице многостраничных элементов управления или только в пределах текущей группы или страницы. Может содержать одну из двух встроенных констант: <b>fmCycleAllForms</b> или <b>fmCycleCurrentForm</b> . Чтение/Запись.

Свойство	Описание
Enabled	Содержит значение типа <b>Boolean</b> , указывающее, доступна ли форма. Если его значение равно <b>False</b> , ни один из элементов управления формы не доступен. Чтение/Запись.
Font	Возвращает ссылку на объект <b>Font</b> , посредством которого вы можете выбрать параметры шрифта формы или элемента управления.
ForeColor	То же самое, что и свойство BackColor, но устанавливает цвет используемый для переднего плана (обычно — это цвет текста) объекта формы.

## Методы объекта UserForm

Всякий раз создавая в проекте новый объект **UserForm**, вы создаете новый подкласс объекта **UserForm**. Любые процедуры или функции, написанные вами в разделе **General** (общий) модуля класса, относящегося к форме, становятся дополнительными методами для отдельного подкласса объекта. Вы также можете создать для формы новые свойства, добавив в ее модуль класса процедуры **Property Get** и **Property Let**. Вы можете создавать экземпляры подкласса вашей **UserForm** с помощью оператора **Dim** и ключевого слова **New**.

Однако чаще всего вы будете манипулировать объектом формы при помощи стандартных методов и свойств класса **UserForm** и при помощи собственных процедур обработки событий для определенной вами формы и ее элементов управления.

В табл. 9.2 перечислены наиболее часто используемые методы для объектов **UserForm**, которыми вы можете воспользоваться, и кратко изложены их свойства. Эти методы будут доступны для каждой формы, которую вы добавляете в свой проект.

**Таблица 9.2. Наиболее часто используемые методы для объектов UserForm**

Метод	Назначение
Copy	Копирует выделенный в элементе управления текст в буфер обмена Windows.
Cut	Вырезает выделенный в элементе управления текст и помещает его в буфер обмена Windows.
Hide	Скрывает UserForm, не выгружая ее из памяти, сохраняя значения элементов управления формы и всех переменных, объявленных в модуле класса формы.
Paste	Вставляет содержимое буфера обмена Windows в текущий элемент управления.
PrintForm	Выводит на используемый в Windows по умолчанию принтер изображение формы, включая все данные, введенные в элементы управления.

Метод	Назначение
Repaint	Перерисовывает форму, выведенную на экран. Используйте этот метод, если хотите перерисовать форму, не ожидая, когда она будет перерисована через обычный период времени.
Show	Выводит форму на экран. Если форма еще не загружена в память, то данный метод сначала ее загружает.

В этой главе при написании примеров процедур обработки событий нам понадобится метод **Show**. Поэтому рассмотрим его подробнее.

Синтаксис метода **Show**:

### Синтаксис

*FormName.Show*

В данной синтаксической конструкции *FormName* может быть любым объектом **UserForm** текущего проекта. *FormName* — имя формы в том виде, как оно отображается в **Project Explorer**. Например, если у вас есть форма **frmInsertFigure**, вывести ее на экран можно с помощью оператора:

*frmInsertFigure.Show*

Если форма в данный момент не загружена в память, метод **Show** загрузит ее и выведет на экран. Если форма уже загружена, метод **Show** просто выведет ее на экран. В любом случае метод **Show** выводит форму и затем передает ей управление. Форма будет оставаться на экране до тех пор, пока не будет выполнен метод **Hide** объекта **UserForm** или форма не будет выгружена при помощи оператора **Unload**.

Все формы VBA являются *модальными (modal)* приложениями, это означает, что вы не сможете выполнить какое-либо другое действие в приложении до тех пор, пока форма диалога не будет закрыта или скрыта.

Когда VBA выполняет метод **Show** для отображения формы диалогового окна, процедура, содержащая вызов метода **Show**, приостанавливается до тех пор, пока выведенная форма не будет закрыта пользователем. Однако VBA будет выполнять программу для любых событийных процедур в модуле класса формы.

### События и событийные процедуры

*Событие (event)* — это что-то, что может произойти с диалоговым окном или элементом управления диалогового окна. Типичные примеры событий: щелчок на кнопке, переключателе и т.д. Другие примеры событий: изменение содержимого окна редактирования или выбор элемента списка. Щелчок мышью, нажатие клавиши и действия внутренние для вашего компьютера, — все они *запускают* или, иными словами, влекут за собой события.

Использование событий позволяет создавать действительно диалоговые приложения. В таких приложениях все действия пользователя приводят к определенной реакции приложения, если эти действия предусмотрены и не заблокированы. Если вы хотите, чтобы ваши пользователи чувствовали себя комфортно при работе с вашими программами, начинайте создавать свои диалоговые окна с разработки событийных процедур.

Такие объекты, как формы и элементы управления приводят в действие, то есть делают доступными, некоторые события. Вы можете написать собственные VBA-процедуры, реагирующие на события. Такие процедуры, называются *событийными процедурами (event procedures)* или *процедурами обработки событий*.

Событийные процедуры следует записывать в модуль класса, который является частью **UserForm**. При этом такие процедуры должны иметь имена в виде *ObjectName\_EventName*, где *ObjectName* — имя формы или элемента управления, а *EventName* — имя события, с которым вы хотите работать. Такой формат имени позволяет VBA сопоставлять заданному событию требуемую процедуру.

Большая часть программы, которую вы записываете в модуль класса формы, будет связана с обработкой событий. В таблице 9.3 перечислены события, для которых вы можете написать процедуры обработки.

**Таблица 9.3. События объектов UserForm**

Событие	Синтаксис заголовка процедуры обработки события	Описание
Activate	Private Sub <b>object_Activate()</b>	Иницируется всякий раз, когда окно формы становится активным. Используйте это событие для обновления содержимого диалоговых элементов управления, чтобы отразить любые изменения, которые произошли, пока окно формы было неактивным.
Click	Private Sub <b>object_Click(index As Long)</b>	Иницируется всякий раз, когда по форме (любой ее части, не занятой элементами управления) щелкают мышью.
DblClick	Private Sub <b>object_DblClick(index As Long, ByVal Cancel As MSForms.ReturnBoolean)</b>	Иницируется всякий раз, когда по форме (любой ее части, не занятой элементами управления) дважды щелкают мышью.
Deactivate	Private Sub <b>object_Deactivate()</b>	Иницируется всякий раз, когда форма перестает быть активной.
Initialize	Private Sub <b>object_Initialize()</b>	Иницируется всякий раз, когда форма впервые загружается в память посредством выполнения оператора <b>Load</b> или с помощью метода <b>Show</b> . Используйте это событие для инициализации элементов управления формы при ее появлении на экране.
Resize	Private Sub <b>UserForm_Resize()</b>	Иницируется при изменении размеров формы.

Событие	Синтаксис заголовка процедуры обработки события	Описание
Terminate	Private Sub <b>object</b> _Terminate()	Иницируется всякий раз, когда форма выгружается из памяти. Используйте это событие для осуществления любых специальных служебных задач, которые необходимо выполнить прежде, чем переменные формы будут выгружены.

В дополнение к методам, свойствам и событиям, встроенным в объект **UserForm**, VBA предоставляет два оператора, которые особенно полезны при работе с объектами форм: **Load** и **Unload**. Вы можете использовать эти операторы для того, чтобы загрузить форму в память или же удалить ее оттуда.

### Синтаксис Load/Unload

Синтаксические конструкции для использования операторов **Load** и **Unload** выглядят следующим образом:

#### Синтаксис

---

Load *Object*  
Unload *Object*

Здесь *Object* представляет любую допустимую ссылку на объект **UserForm**.

---

Оператор **Load** загружает в память объект **UserForm** и запускает метод формы **Initialize**, но не выводит форму на экран. Когда форма загружена, вы можете использовать написанную на VBA программу для работы с объектом **UserForm**. Оператор **Unload** удаляет из памяти **UserForm**, а также все переменные формы. После того как форма выгружена, она перестает быть доступной для VBA-кода.

### Примеры программ модуля класса формы

Рассмотрим несколько простых примеров использования свойств и методов формы с помощью процедур обработки событий. Будем инициировать события, описанные в табл. 9.3, а в ответ на них с помощью окна функции **MsgBox** будем сообщать о типе события или менять видимые свойства формы. Начнем с события **Click**, так как процедуру обработки этого события легче всего создать: достаточно дважды щелкнуть на форме (используем созданную в начале главы форму) в режиме разработки, и вам будет предоставлен шаблон процедуры в виде:

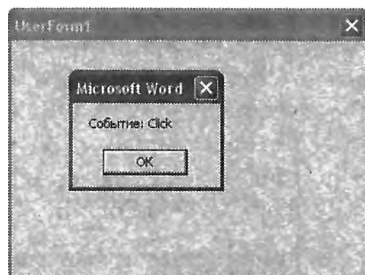
```
Private Sub UserForm_Click()
End Sub
```

Запишите после заголовка процедуры следующий оператор

```
MsgBox ("Событие: Click")
```

Запустите форму на выполнение командой **Run | Run Sub/UserForm**. После появления формы на экране щелкните на ней мышью, на экране отобразится сообщение, подобное приведенному на рис. 9.2.

**Рис. 9.2**  
Выполнение процедуры обработки события Click



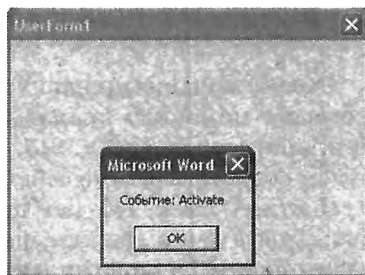
Если даже вам не часто будет нужно использовать подобное событие для формы, не забывайте о такой возможности. А еще лучше поступайте следующим образом. Всегда используйте это событие на случай, когда пользователь ваших форм при попытке щелкнуть на каком либо элементе управления не совсем точно позиционирует курсор мыши. Если при этом будет выполняться процедура обработки события **Click** для формы с выводом шутливого сообщения о недостаточной уверенности в действиях пользователя, с вашим приложением станет приятно работать. Вы также никогда не забудете о возможности использовать это событие, когда для этого появятся серьезные причины.

Продолжим рассмотрение событий и протестируем событие **Activate**. Запишите в модуле рассматриваемой формы следующую процедуру:

```
Private Sub UserForm_Activate()  
    MsgBox ("Событие: Activate")  
End Sub
```

Снова запустите форму на выполнение командой **Run | Run Sub/UserForm**. После появления формы на экране вы увидите сообщение о том, что произошло событие **Activate** (рис. 9.3).

**Рис. 9.3**  
Сразу после загрузки формы произошло событие Activate



Рассмотрим события **Deactivate** и **Terminate**. Первое из них инициируется всякий раз, когда форма перестает быть активной, а второе — когда форма выгружается из памяти. Процедура обработки события **Terminate** просто добавляется к имеющейся коллекции программ формы **UserForm1** и выполняется при завершении работы с формой. Для демонстрации же со-

бытия **Deactivate** необходимо, не выходя из приложения, сделать форму **UserForm1** неактивной. Это можно осуществить, если активной станет другая форма. И ее надо создать.

Создайте новую форму либо при помощи кнопки **Insert UserForm**, либо — командой **Insert | UserForm**. Пусть она имеет имя, которое ей присваивается по умолчанию: **UserForm2**. Помните, что следует всегда переименовывать формы для удобства сопровождения разработки, но для нашего тестирования событий мы этого делать не будем. Наметим план тестирования:

1. Загружаем форму **UserForm1**.
2. Событие **Click** формы **UserForm1** вызывает метод **Show** для загрузки формы **UserForm2**.
3. Загрузка/выгрузка формы **UserForm2** не сопровождается никакими событиями.
4. Форма **UserForm1** имеет процедуры обработки событий: **Click**, **Activate**, **Deactivate**, **Terminate**.

Для выполнения поставленной задачи в модуль формы **UserForm1** необходимо поместить код Листинга 9.1.

#### Листинг 9.1. Процедуры событий **UserForm1**

---

```
1: 'Обработчик события Click формы UserForm1
2: Private Sub UserForm_Click()
3:     'Вывод окна сообщения на экран:
4:     MsgBox ("Событие: Click. Выводим UserForm2")
5:     'Загрузка формы UserForm2:
6:     UserForm2.Show
7: End Sub
8:
9: 'Обработчик события Activate формы UserForm1
10: Private Sub UserForm_Activate()
11:     'Вывод окна сообщения на экран:
12:     MsgBox ("Событие: Activate для формы UserForm1")
13: End Sub
14:
15: Private Sub UserForm_Deactivate()
16:     'Вывод окна сообщения на экран:
17:     MsgBox ("Событие: Deactivate для формы UserForm1")
18: End Sub
19:
20: Private Sub UserForm_Terminate()
21:     'Вывод окна сообщения на экран:
22:     MsgBox ("Событие: Terminate для формы UserForm1")
23: End Sub
```

---

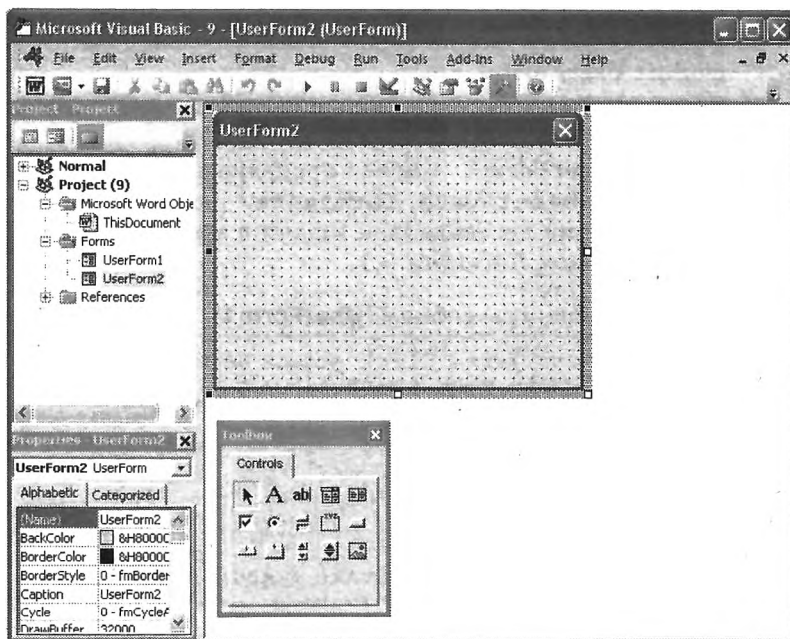
Следует отметить, что после создания новой формы, вы не сразу можете найти модуль формы **UserForm1**. Ваш экран может быть похожим на тот, который приведен на рис. 9.4.

Выберите команду **Run | Run Sub/UserForm**. Теперь вы имеете возможность проследить за инициализацией всех четырех событий. Сразу после загрузки **UserForm1** выдается сообщение «Событие: Activate для формы **UserForm1**» (процедура **UserForm\_Activate**). Щелчок на форме приводит к выводу сообщения «Событие: Click. Выводим **UserForm2**» (процедура **UserForm\_Click**), за-

грузке формы **UserForm2** (процедура **UserForm\_Click**) и выводу сообщения «Событие: **Deactivate** для формы **UserForm1**» (процедура **UserForm\_Deactivate**) в результате того, что форма **UserForm1** становится неактивной. Если бы мы описали процедуру события **Activate** и для формы **UserForm2**, то один щелчок мыши вызвал бы инициализацию трех событий. И, наконец, не забудьте обратить внимание на то, что при выгрузке из памяти формы **UserForm1** на экран выдается сообщение процедуры **UserForm\_Terminate**.

**Рис. 9.4**

После создания новой формы очень легко «потерять» предыдущую



Событие **Initialize** инициализируется при загрузке формы при помощи оператора **Load** или метода **Show**. При выполнении команды **Run | Run Sub/UserForm** это событие не инициализируется. Это событие следует использовать при первой загрузке формы для установки каких-либо свойств формы и ее элементов управления. В следующем примере (листинг 9.2) для этого события устанавливается свойство формы **BackColor**.

#### Листинг 9.2. Процедуры событий UserForm2

```

1: Dim sRED, sGREEN, sBLUE 'переменные для задания цвета формы
2:
3: 'Процедура обработки события Initialize
4: 'Инициализируется один раз: при загрузке
5: Private Sub UserForm_Initialize()
6:     'задаем начальный цвет формы
7:     sRED = 100
8:     sGREEN = 100
9:     sBLUE = 200
10:    UserForm2.BackColor = RGB(sRED, sGREEN, sBLUE)
11: End Sub

```



```
12:
13: 'Процедура обработки события Click
14: 'При каждой инициализации меняет цвет формы
15: 'При недопустимых значениях sRED, sGREEN, sBLUE выдает
    ошибку времени исполнения
16: Private Sub UserForm_Click()
17:     Dim I
18:
19:     'Меняем цвет формы:
20:     sRED = sRED + 20
21:     sGREEN = sGREEN + 10
22:     sBLUE = sBLUE - 20
23:     i = RGB(sRED, sGREEN, sBLUE)
24:     UserForm2.BackColor = I
25:
26:     'Изменить заголовок формы:
27:     UserForm2.Caption = "Цвет: " & Str(i)
28: End Sub
```

Для загрузки формы **UserForm2** используйте форму **UserForm1** с программами обработки событий из листинга 9.1.

В листинге 9.2 для установки свойства формы **BackColor** используется функция **RGB**, которая возвращает RGB-значение типа **Long**, используемое далее для присвоения свойству **UserForm2.BackColor**. Синтаксис функции **RGB**:

### Синтаксис

**RGB**(*red, green, blue*)

Именованные аргументы:

Аргумент	Описание
<i>Red</i>	Обязательный; тип: <b>Variant (Integer)</b> . Число в диапазоне 0–255; представляет красный компонент цвета.
<i>Green</i>	Обязательный; тип: <b>Variant (Integer)</b> . Число в диапазоне 0–255; представляет зеленый компонент цвета.
<i>Blue</i>	Обязательный; тип: <b>Variant (Integer)</b> . Число в диапазоне 0–255; представляет синий компонент цвета.

После вывода формы с именем **UserForm2** на экран событие **Click** будет приводить к изменению цвета и заголовка формы.

Мы не рассмотрели еще два события, связанные с формой: **DbtClick**, которое является аналогом **Click**, и **Resize** — событие, которое инициализируется при изменении размеров формы.

Рассмотрим последнее событие, так как первое должно быть понятным и в реализации похожим на ранее рассмотренные.

Размеры формы могут изменяться как во время диалога (если это предусмотрено разработчиком формы), так и программным способом. Для иллюст-

рации программы обработки события изменения формы удобнее использовать программный способ. Листинг 9.3 содержит две процедуры: первая обрабатывает событие **Click** (при этом изменяется свойство формы **Width**), вторая — событие **Resize**.

### Листинг 9.3. Обработка событий **Click** и **Resize**

```
1: Private Sub UserForm_Click() 'Можно UserForm_DblClick
2:   MsgBox ("Событие: Click")
3:   'Меняем размер формы:
4:   UserForm1.Width = 400
5: End Sub
6:
7: Private Sub UserForm_Resize()
8:   MsgBox ("Событие: Resize")
9: End Sub
```

## Элементы управления

Объект **UserForm** может содержать те же элементы управления, что и находящиеся в диалоговых окнах Word, Excel или других приложений Windows. *Элементы управления (controls)* — это элементы диалогового окна, которые дают возможность пользователю взаимодействовать с программой. Они включают в себя кнопки-переключатели, текстовые поля, линейки прокрутки, командные кнопки и так далее. В этом разделе вы познакомитесь со стандартными элементами управления, включенными в VBA, которые сможете добавлять в свои формы.

Многие производители программного обеспечения разрабатывают пакеты, расширяющие набор элементов управления. Вы можете добавлять элементы управления (известные как объекты **ActiveX** и **Automation**) к панели **Toolbox** (панели элементов), предварительно создавая ссылку из вашего проекта на библиотеку, содержащую расширенный набор элементов управления. После этого можно добавлять данные элементы управления на панель **Toolbox**.

Каждый элемент управления — это объект с определенными свойствами, методами и событиями. Как и для формы, их содержащей, вы можете устанавливать свойства элементов управления программным путем или посредством **Properties Window** редактора VB. В программе вы можете присваивать или восстанавливать значения свойств элементов управления так же, как для любых других объектов.

В табл. 9.4 перечислены стандартные элементы управления, включенные в VBA, и описано назначение каждого элемента. Как видно из этой таблицы, к стандартным относятся, практически, все элементы управления, которые вам встречались в приложениях Windows. (В зависимости от host-приложения VBA, в котором вы работаете, на панели **Toolbox** могут появляться дополнительные элементы управления, не перечисленные в табл. 9.4. Эти дополнительные элементы доступны посредством библиотек элементов управления, поставляемых с каждым из host-приложений.)

Таблица 9.4. Стандартные элементы управления, включенные в VBA

Элемент управления	Назначение
Label (надпись, метка)	Позволяет создавать заголовки элементов управления, которые не имеют собственных встроенных заголовков. Используйте этот элемент для того, чтобы поместить на форму статический текст, например, инструкции, советы по заполнению других диалоговых элементов управления.
TextBox (текстовое поле)	Окно редактируемого текста свободной формы для ввода данных. Может быть одно- или многострочным.
ComboBox (поле со списком)	Этот элемент управления объединяет окно редактирования и окно списка. Используйте его, когда хотите предложить пользователю выбрать значение, но при этом дать ему возможность ввести данные, отсутствующие в списке. Вы можете также ограничить выбор только теми значениями, которые появляются в ComboBox для эмуляции ниспадающего списка.
ListBox (список)	Отображает список значений, из которых пользователь может сделать выбор. Окна списка можно использовать, чтобы дать возможность пользователю выбрать только одно значение или же несколько.
CheckBox (флажок)	Стандартный флажок (квадратное окно, содержащее «галочку», если элемент выбран). Используйте флажки для выбора вариантов, которые не являются взаимоисключающими.
OptionButton (переключатель)	Стандартная кнопка-переключатель (круглое окно, при выборе в центре него находится черная точка). Используйте OptionButton, когда пользователю необходимо сделать выбор между «включено/выключено», «истина/ложь». Кнопки-переключатели, как правило, объединяются вместе при помощи рамки для создания группы переключателей.
ToggleButton (выключатель)	Выключатели служат для той же цели, что и флажки, но выводят установки в виде кнопки находящейся в «нажатом» или «отжатом» состоянии.
Frame (рамка)	Визуально и логически объединяет некоторые элементы управления (особенно флажки, переключатели и выключатели). Используйте Frame, чтобы показать пользователю, какие элементы управления в диалоговом окне связаны между собой, или чтобы выделить группу элементов управления, отделяя ее от остальных элементов управления, находящихся в диалоговом окне.
CommandButton (кнопка)	Используйте кнопки для выполнения таких действий, как Cancel (Отмена), Save (Сохранить), OK и так далее. Когда пользователь щелкает по кнопке, выполняется VBA-процедура, закрепленная за данным элементом управления.

Элемент управления	Назначение
TabStrip (набор вкладок)	Этот элемент управления состоит из области, в которую следует помещать другие элементы управления (такие как текстовые поля, флажки и так далее). Используйте элемент управления <b>TabStrip</b> для создания диалоговых вкладок, отображающих одни и те же данные в различных категориях.
MultiPage (набор страниц)	Этот элемент управления состоит из нескольких страниц. Вы можете выбрать любую из них, щелкнув по соответствующей вкладке. Используйте элемент управления <b>MultiPage</b> для создания диалоговых окон с вкладками, такими, например, как диалоговое окно, появляющееся при выборе команды <b>Tools   Options</b> (панели инструментов   настройка).
ScrollBar (полоса прокрутки)	Элемент управления <b>ScrollBar</b> позволяет выбирать линейное значение, аналогично тому, как это можно сделать при помощи счетчика.
SpinButton (счетчик)	Элемент управления <b>SpinButton</b> является специальной разновидностью текстового поля. Обычно счетчики используются для того, чтобы ввести число, дату или какие-либо иные последовательные величины, которые заведомо находятся в определенном интервале значений. Щелчок по указывающей вверх стрелке счетчика увеличивает значение в окошке, а щелчок по стрелке, направленной вниз, соответственно, уменьшает его.
Image (рисунок)	Элемент управления <b>Image</b> позволяет вывести на форме графическое изображение. Используйте <b>Image</b> для вывода графических изображений в любом из следующих форматов: *.bmp, *.cur, *.gif, *.ico, *.jpg, или *.wmf. Вы можете обрезать и масштабировать графическое изображение, чтобы подобрать размер элемента <b>Image</b> , но только не редактировать графическое изображение. Можно даже написать специальную VBA-процедуру, выполняющуюся, если пользователь щелкнет по элементу управления <b>Image</b> .

Обращение к элементам управления выполняется, в основном, через их свойства и с помощью процедур обработки событий, написанных для каждого элемента. В табл. 9.5 перечислены наиболее часто используемые свойства элементов управления, которые могут понадобиться для работы с вашими VBA-программами — свойства, которые позволяют изменять заголовок, определять состояние элемента управления (то есть обнаруживать установки, выполненные пользователем) и так далее.

**Таблица 9.5. Наиболее часто используемые свойства стандартных элементов управления**

Свойство	Где применяется	Описание
Accelerator	CheckBox, Tab, CommandButton, Label, Page, OptionButton, ToggleButton	Содержит символ, используемый, в качестве быстрой клавиши вызова, элемента управления. При «нажатии» Alt+<клавиша быстрого вызова> происходит выбор элемента управления.
BackColor	Все элементы	Число, представляющее определенный цвет фона элемента управления.
Caption	CheckBox, CommandButton, Frame, Label, OptionButton, ToggleButton, Page, Tab, UserForm	Для надписи — текст, отображаемый, элементом управления. Для других элементов управления — надпись, которая появляется на кнопке или вкладке или рядом с рамкой, флажком или переключателем.
Cancel	CommandButton	Задаёт кнопку отмены диалогового окна. При «нажатии» на эту кнопку или клавишу Esc диалоговое окно исчезает. Только одна кнопка формы может иметь данное свойство.
ControlTipText	Все элементы управления	Определяет текст, который отображается в виде всплывающей подсказки ( <b>ControlTip</b> , называемой также <b>ToolTip</b> ), когда указатель мыши помещается на элемент управления.
Default	CommandButton	Определяет заданную по умолчанию кнопку. Когда пользователь нажимает в процессе диалога клавишу Enter, эта кнопка ведет себя так, как если бы по ней щелкнули мышью.
Enabled	Все элементы управления	Хранит значение типа <b>Boolean</b> , определяющее доступен или нет элемент управления. Если <b>Enabled</b> имеет значение <b>False</b> , то элемент управления продолжает отображаться в диалоговом окне, но не может быть выбран.
ForeColor	Все элементы управления	То же самое, что и <b>BackColor</b> , но устанавливает цвет для переднего плана элемента управления — как правило, символов текста.
List	ComboBox, ListBox	Массив типа <b>Variant</b> (одно- или многомерный), представляет список содержащийся в элементе управления. Используйте методы элемента управления <b>AddItem</b> и <b>RemoveItem</b> для добавления или удаления пунктов списка.

Свойство	Где применяется	Описание
Max	ScrollBar, SpinButton	Переменная типа <b>Long</b> , определяющая максимальное значение счетчика, или значение, при котором полоса прокрутки находится в самом верху (для вертикальной полосы) или справа (для горизонтальной).
Min	ScrollBar, SpinButton	Переменная типа <b>Long</b> , определяющая минимальное значение счетчика или значение, при котором полоса прокрутки находится в самом низу (для вертикальной полосы) или слева (для горизонтальной).
Name	Все элементы управления	Содержит имя элемента управления. Вы можете установить данное свойство только с помощью <b>Properties Window</b> .
RowSource	ComboBox, ListBox	Задаёт источник, из которого <b>ComboBox</b> или <b>ListBox</b> «берет» список объекта. В Excel VBA <b>RowSource</b> обычно использует диапазон рабочего листа.
Selected	ListBox	Возвращает массив значений типа <b>Boolean</b> для списка, который допускает множественный выбор. Каждый элемент массива содержит по одному элементу, соответствующему каждому пункту списка. Если значение элемента в массиве <b>Selected</b> равно <b>True</b> , то соответствующий пункт списка выбран.
TabIndex	Все элементы управления	Число, указывающее положение элемента управления в порядке табуляции (может иметь значение от 0 до значения, равного количеству элементов управления на форме).
TabStop	Все элементы управления	Значение типа <b>Boolean</b> , указывающее может ли элемент управления быть выбран клавишей Tab. Если значение <b>TabStop</b> равно <b>False</b> вы, тем не менее, можете щелкнуть на элементе и таким образом его выбрать.
Value	Все элементы управления	Значение текущих установок элемента управления: текст в текстовом поле, какие выбраны флажки и переключатели, индекс выбранного раздела списка или число, указывающее текущее положение полосы прокрутки или счетчика.
Visible	Все элементы управления	Значение типа <b>Boolean</b> , указывающее, является ли элемент управления видимым.

В табл. 9.6 перечислены события элементов управления, для которых вы можете написать собственные процедуры обработки событий. Каждый элемент управления, который вы добавите в свою форму, будет иметь доступ к этим со-

бытия. В частности, для кнопок используется событие **Click**, а для того, чтобы проверить данные или обновить другие элементы управления, — **AfterUpdate** или **Change**.

**Таблица 9.6. Наиболее часто используемые события объектов управления**

Событие	Синтаксис процедуры обработки	Описание
AddControl	для элемента <b>Frame</b> : <pre>Private Sub object_AddControl()</pre> для элемента <b>MultiPage</b> : <pre>Private Sub object_AddControl(index As Long, ctrl As Control)</pre>	Иницируется всякий раз, когда к форме (или элементам <b>Frame</b> , <b>Page</b> , <b>MultiPage</b> ) добавляется какой-либо элемент управления.
AfterUpdate	<pre>Private Sub object_AfterUpdate()</pre>	Иницируется после обновления значения элемента управления.
BeforeUpdate	<pre>Private Sub object_BeforeUpdate(ByVal Cancel As MSForms.ReturnBoolean)</pre>	Иницируется после того, как было изменено значение элемента управления, но перед тем, как был обновлен сам элемент управления.
Change	<pre>Private Sub object_Change()</pre>	Иницируется всякий раз, когда изменяется значение элемента управления.
Click	<pre>Private Sub object_Click()</pre>	Иницируется всякий раз, когда по элементу управления щелкают мышью.
DbClick	<pre>Private Sub object_DbClick(ByVal Cancel As MSForms.ReturnBoolean)</pre>	Иницируется всякий раз, когда по элементу управления дважды щелкают мышью.
Enter	<pre>Private Sub object_Enter()</pre>	Иницируется всякий раз, когда выделяется элемент управления.
Exit	<pre>Private Sub object_Exit(ByVal Cancel As MSForms.ReturnBoolean)</pre>	Иницируется всякий раз, когда с элемента управления снимается выделение.
Error	<pre>Private Sub object_Error(Index As Long, ByVal Number As integer, ByVal Description As MSForms.ReturnString)</pre>	Иницируется всякий раз, когда элемент управления обнаруживает ошибку и не может вернуть информацию об ошибке в вызывающую программу.

Событие	Синтаксис процедуры обработки	Описание
KeyDown	Private Sub object_KeyDown (ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As fmShiftState)	Иницируется при нажатии пользователем какой-либо клавиши в тот момент, когда форма выполняется и имеет фокус.
KeyPress	Private Sub object_KeyPress (ByVal KeyANSI As MSForms.ReturnInteger)	Иницируется, когда пользователь нажимает алфавитно-цифровую клавишу.
KeyUp	Private Sub object_KeyUp (ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As fmShiftState)	Иницируется, когда пользователь отпускает клавишу.
Layout	Private Sub object_Layout()	Иницируется, когда изменяются размеры элемента <b>Frame</b> (или <b>MultiPage</b> ).
MouseDown, MouseUp	для элементов MultiPage и TabStrip:  Private Sub object_MouseDown( index As Long, ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single) Private Sub object_MouseUp( index As Long, ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)  для других элементов управления:  Private Sub object_MouseDown( ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single) Private Sub object_MouseUp( ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)	Иницируются при щелчке мышью. <b>MouseDown</b> : когда пользователь нажимает на клавишу мыши; <b>MouseUp</b> : когда пользователь отпускает клавишу мыши.



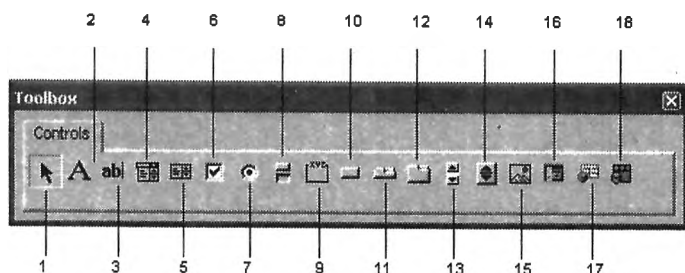
Событие	Синтаксис процедуры обработки	Описание
MouseMove	<div>для элементов MultiPage и TabStrip:</div> <pre>Private Sub object_MouseMove(index As Long, ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)</pre> <div>для других элементов управления:</div> <pre>Private Sub object_MouseMove(ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)</pre>	Иницируется, когда пользователь перемещает мышь.
SpinDown, SpinUp	<pre>Private Sub object_SpinDown() Private Sub object_SpinUp()</pre>	Событие <b>SpinDown</b> иницируется, когда пользователь щелкает стрелку «вниз» (или «влево») кнопки счетчика. Событие <b>SpinUp</b> иницируется, когда пользователь щелкает стрелку «вверх» (или «вправо») кнопки счетчика.
Zoom	<pre>Private Sub object_Zoom(index As Long, Percent As Integer)</pre>	Иницируется при изменении свойства <b>Zoom</b> .

## Использование Toolbox (панели элементов)

Прежде чем мы рассмотрим примеры обработки событий, связанных с элементами управления, вам следует научиться помещать элементы управления на форму и задавать им необходимые свойства.

Редактор VB в режиме разработки вместе с формой выводит на экран **Toolbox** (панель элементов) (если вывод этой панели не отключен при помощи меню **View | Toolbox**). На рис. 9.5 **Toolbox** показана в виде плавающего окна. Кнопки на панели **Toolbox** активизируют различные инструменты, которые позволяют помещать на форму элементы управления (на рис. 9.5 приведена панель **Excel**).

**Toolbox** (подобно всем другим панелям инструментов) можно настраивать. Если вы или другой пользователь установили дополнительные элементы управления независимых производителей или произвели настройку панели, она может выглядеть иначе, чем та, что показана на рис. 9.5. На рис. 9.5 присутствуют только стандартные элементы управления, поставляемые вместе с VBA.

**Рис. 9.5.**

Используйте Toolbox для размещения на форме элементов управления:

1 — Select Objects (выбор объектов), 2 — Label (надпись), 3 — TextBox (поле), 4 — ComboBox (поле со списком), 5 — ListBox (список), 6 — CheckBox (флажок), 7 — OptionButton (переключатель), 8 — ToggleButton (выключатель), 9 — Frame (рамка), 10 — CommandButton (кнопка), 11 — TabStrip (набор вкладок), 12 — MultiPage (набор страниц), 13 — ScrollBar (полоса прокрутки), 14 — SpinButton (счетчик), 15 — Image (рисунок), 16 — RefEdit, 17 — DataGrid (сетка данных), 18 — MSFlexGrid

Чтобы воспользоваться панелью **Toolbox** для добавления элементов управления к вашей форме, выполните следующие действия:

1. Щелкните на **Toolbox** по кнопке, соответствующей элементу управления, который вы хотите добавить к форме. Указатель мыши изменит форму на перекрестие, когда будет находиться на форме.
2. Установите перекрестие в то место формы, в котором хотите поместить верхний левый угол нового элемента управления.
3. Нажмите и удерживайте левую кнопку мыши.
4. Перемещайте мышь вниз и вправо, пока элемент управления не примет нужный размер, после чего отпустите кнопку мыши. Редактор VB вставит в форму элемент управления, и указатель мыши вновь примет форму стрелки.

Вы можете изменить размер самой формы. Щелкните по строке заголовка формы для ее выделения и затем, перемещая один из маркеров изменения размеров, увеличьте или уменьшите размер формы до желаемого. *Маркеры изменения размеров (sizing handles)* — это маленькие квадратики, которые появляются в углах и на серединах сторон графического объекта (наряду с толстой серой границей) при выделении объекта.

## Добавление к форме элементов управления

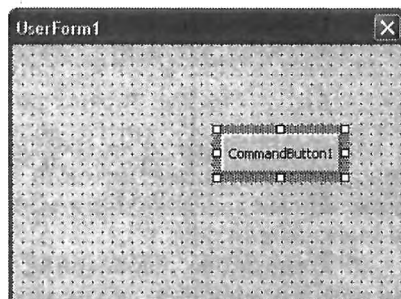
Когда вы создаете новую форму, редактор VB предоставляет чистую (без элементов управления) форму (смотри рис. 9.1). Вы можете добавить к форме элементы управления с использованием различных инструментов панели **Toolbox**.

Чтобы добавить к форме элемент управления, используйте панель **Toolbox**, как описано в предыдущем параграфе этой главы. В качестве примера добавления к форме конкретного элемента добавьте на новую форму кнопку, выполнив следующие действия:

1. Выберите команду **Insert | UserForm** (Вставка | UserForm). Редактор VB добавит новую форму, отображая ее в режиме разработки, и, кроме того, на экране появится **Toolbox**.
2. Щелкните на элементе **CommandButton** панели **Toolbox**. Кнопка перейдет в «утопленное» состояние, указывая выбранный вами элемент.
3. Поместите курсор мыши на форму. Курсор при этом изменит свою форму на перекрестие с прикрепленным к нему символом выбранного элемента.
4. Поместите перекрестие в ту точку формы, в которой должен быть верхний левый угол кнопки.
5. Переместите курсор мыши вниз и вправо, чтобы нарисовать элемент управления **CommandButton**. В процессе перемещения курсора мыши редактор VB отображает прямоугольный контур, показывающий размер кнопки.
6. Отпустите кнопку мыши, когда решите, что **CommandButton** имеет необходимый размер. Редактор VB создаст элемент управления **CommandButton** и поместит его на форму. На рис. 9.6 показано, как выглядит кнопка, сразу же после того, как была помещена на форму.

**Рис. 9.6**

**UserForm1** сразу же после добавления элемента **CommandButton**

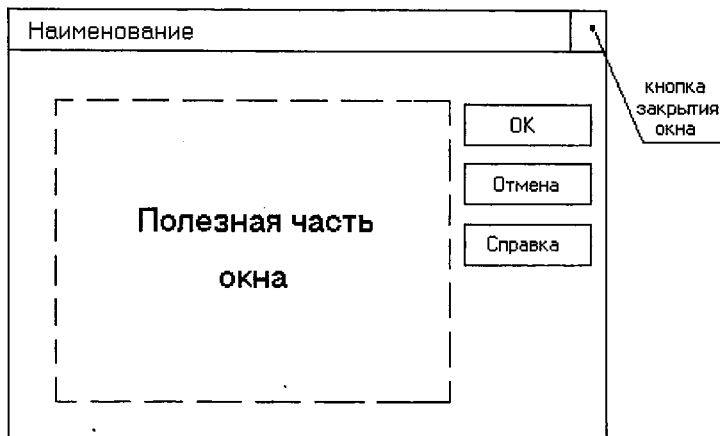


Добавив к форме кнопку, вы можете заметить, что верхний левый угол кнопки (также как и сама граница кнопки) автоматически выравнивается по шаблону сетки на форме. Это действие, называемое *привязка к сетке* (*snap to grid*), поможет вам выровнять размещенные на форме элементы управления и текст. Вы можете включать и отключать привязку к сетке, настраивать шаг сетки или скрывать ее, изменяя установки в диалоговом окне **Option** (Параметры) редактора VB. Выберите **Tools | Options** (Сервис | Параметры) и щелкните по вкладке **General** (общие), чтобы вывести **Form Grid Settings** (Параметры сетки в форме). Выберите или сбросьте необходимые опции.

Все элементы управления формы должны иметь уникальные имена. Эти имена следует использовать при ссылках на элемент управления в своей программе. Всякий раз, когда вы добавляете к форме новый элемент управления, VBA присваивает ему имя по умолчанию, состоящее из имени типа элемента и номера. Если кнопка, которую вы добавили в начале этого раздела, является первой кнопкой, она получит имя **CommandButton1**. Следующая добавленная вами кнопка будет **CommandButton2** и так далее. Если после этого вы добавите поле, оно получит имя **TextBox1**.

Перед тем как продолжить работу с формами, давайте договоримся о несложных правилах, которые помогут вам с первых шагов создавать диалоговые окна, почти похожие на профессиональные. Если вы до сих пор не обращали внимания на то, что все окна «солидных» разработчиков имеют общие черты, то сейчас как раз то время, когда это нужно сделать. На рис. 9.7 приведена схема, которой можно придерживаться при разработке своих первых форм.

**Рис. 9.7**  
Схема типичного  
Windows-окна



Наименование окна, как вы уже знаете, задается свойством **Caption** формы. Кнопка закрытия окна, если ничего специально не предпринимать, по умолчанию всегда будет в окне приложения. Если она вам не нужна, то лучше от нее избавиться.

В «полезной» части окна помещаются функционально необходимые элементы управления, которые определяются назначением окна. Здесь могут быть, например, элементы для ввода некоторой информации или изменений настроек приложения.

С кнопкой **ОК** обычно связывают событие, которое подводит некоторый итог. Обычно в профессиональных приложениях все, что вводится в диалоговом окне, не сразу вступает в действие, а только после того, как пользователь щелкнет кнопку **ОК**. Это дает возможность отказаться от недостаточно обдуманных действий. Разработчику таких окон не очень трудно скопировать данные из диалогового окна в несколько переменных прежде, чем присвоить их каким-либо свойствам элементов управления, а пользователю удобнее работать с окном, из которого всегда можно выйти без последствий.

Из предыдущего абзаца, видимо, понятно назначение кнопки **Отмена**. Дополнительной ее функцией является выход из процедуры обработки событий формы. Кнопка же **ОК** может разрешить пользователю продолжить путешествие «внутри» приложения, которое открылось данным окном (иногда говорят «панелью» или «диалогом»).

Кнопка **Справка** знакома всем «с детства». К сожалению, не так просто организовать помощь, подобную обычной Windows-справке, но можно сначала использовать эту кнопку для простых подсказок, которые создаются помещением на пустую форму элемента **Label** с текстом подсказки. В дальнейшем, изу-

чив способы построения сложных систем подсказок, можно создавать почти профессиональные диалоговые окна.

Этих нехитрых правил, конечно же, вам хватит при проектировании несложных форм. При создании «хороших» форм следует ознакомиться с рекомендациями фирмы Microsoft. Эти рекомендации имеют отношение к следующим аспектам:

- Размещение элементов управления на форме (в том числе и установка интервалов между элементами).
- Выравнивание меток (метки должны быть хорошими путеводителями формы).
- Использование стандартных шрифтов (часто пользователи собирают по «всему свету» интересные шрифты и забывают о том, что такие шрифты могут отсутствовать у других).
- Использование цвета.

При размещении на форме нескольких элементов управления, которые можно условно разделить на отдельные группы, следует использовать элемент **frame** (рамка). При этом желательно пользоваться свойством, определяющим заголовок рамки.

Для более легкого чтения полей, расположенных друг под другом, необходимо выравнивать их по левому краю.

Для всех элементов управления используйте в качестве шрифта нормальный Sans Serif 8-пунктов. Конечно, следует учитывать и запросы потенциального пользователя вашего продукта, но при подготовке тестового варианта для обсуждения сначала необходимо придерживаться рекомендаций Microsoft.

Самые обычные серые цвета, которые вы можете видеть во всех известных продуктах Windows, рекомендуются и для ваших приложений. Хотя в вашем распоряжении имеется большое количество цветов, все равно без особой нужды не следует изменять цвета по умолчанию. Если же вы можете предоставить пользователю работающего приложения самому изменять цвета формы и элементов управления на ней, это будет оптимальным вариантом.

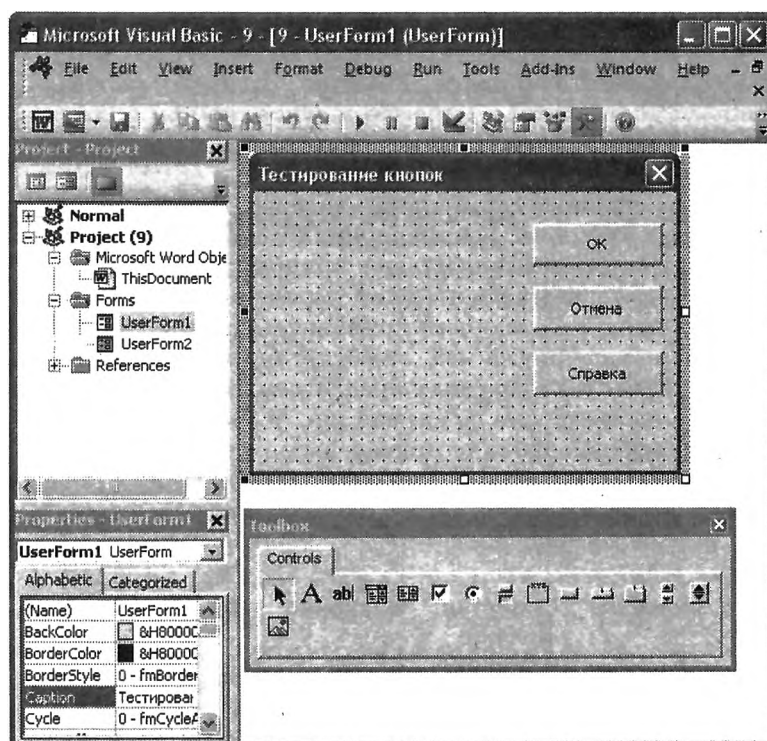
Итак, если вы согласны с предлагаемыми правилами, создайте свою первую форму в соответствии со схемой рис. 9.7, как это показано на рис. 9.8.

В следующей таблице приведены значения свойств элементов управления, которые были изменены. Для своих приложений в качестве документации вы тоже можете создавать такие таблицы для облегчения сопровождения своего программного продукта.

Тип элемента	Свойство, которое изменено	Значение	Примечание
UserForm	Name	Frmfirst	Имя формы, на которое можно ссылаться в коде.
	Caption	Тестирование кнопок	Заголовок окна (формы) в верхней части.
CommandButton	Name	CmdOK	Имя кнопки, на которое можно ссылаться в коде.

Тип элемента	Свойство, которое изменено	Значение	Примечание
	Caption	OK	Текст на кнопке.
	Default	True	При нажатии на клавишу Enter инициируется событие <b>Click</b> кнопки.
CommandButton	Name	CmdCancel	Имя кнопки, на которое можно ссылаться в коде.
	Caption	Отмена	Текст на кнопке.
	Cancel	True	При нажатии на клавишу Esc инициируется событие <b>Click</b> кнопки.
CommandButton	Name	CmdHelp	Имя кнопки, на которое можно ссылаться в коде.
	Caption	Справка	Текст на кнопке.

**Рис. 9.8**  
Форма с самыми необходимыми кнопками



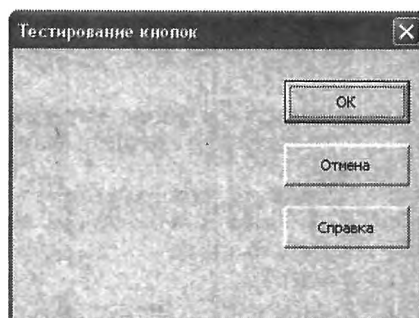
В VBA (как и в Microsoft Visual Basic) кнопки (элементы **CommandButton**) имеют специальное свойство **Cancel**, которое определяет кнопку **Отмена** (независимо от того, называется ли эта кнопка **Отмена**). Только одна кнопка формы может иметь свойство **Cancel**, установленное равным **True**. Если задать это

свойство для одной из кнопок, аналогичное свойство для всех остальных кнопок формы будет установлено равным **False**. Если форма содержит кнопку с установленным свойством **Cancel**, то нажатие клавиши **Esc** приводит к тому же результату, что и щелчок по кнопке.

Если вы запустите форму на выполнение (командой **Run | Run Sub/UserForm**), то ни щелчок на кнопке, ни нажатие клавиши **Esc** не приведут ни к какому результату, так как с кнопкой не связано никакого события. Но диалоговое окно появится на экране в режиме выполнения и будет создавать впечатление чего-то работающего (рис. 9.9).

**Рис. 9.9**

Такое окно тоже «имеет право» называться «диалоговым»



Если вы будете щелкать кнопки этого окна, то убедитесь в том, что окно совершенно «нечувствительно» к вашим манипуляциям. Единственная кнопка, которая будет реагировать на ваши действия, находится вверху справа (с изображением перекрестия). Эта кнопка закроет окно.

Чтобы кнопки, помещенные вами на диалоговой панели, «ожили», необходимо для каждой из них написать процедуры обработки событий **Click** (щелчок на кнопке). Начнем с кнопки **Отмена**, чтобы всегда иметь возможность выйти из приложения. Свяжем с щелчком на кнопке **Отмена** программу, которая выгрузит форму, что приведет к окончанию работы с диалоговым окном. Для создания процедуры обработки события дважды щелкните на кнопке **Отмена** в режиме разработки и в появившемся шаблоне процедуры обработки события введите оператор выгрузки формы, как это представлено в листинге 9.4.

#### **Листинг 9.4.** Процедура обработки события — щелчок на кнопке Отмена

```
1: Private Sub CmdCancel_Click()  
2:   Unload Me 'Выгружаем форму, заканчивая приложение  
3: End Sub
```

Снова запустите форму на выполнение. Вы увидите то же диалоговое окно, что и на рис. 9.9, но теперь это окно — «настоящее» диалоговое, потому что при щелчке на кнопке **Отмена** выполняется программа обработки события и окно закрывается. Более того, при нажатии на клавишу **Esc** вы получаете точно тот же результат! Понятно, что процедура обработки события **Click** (именно это событие и связано с кнопкой) может выполнить и более серьезную работу, чем простая выгрузка формы. Чтобы это было действительно понятно, добавим к процедуре вывод сообщения о том, что возможен выход из диалога (см. листинг 9.5).

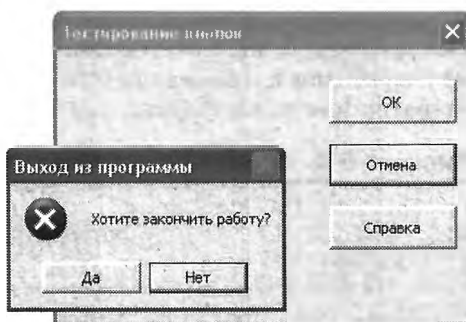
**Листинг 9.5.** Более «продвинутая» процедура обработки события — щелчок на кнопке Отмена

```
1: Private Sub CmdCancel_Click()  
2:   'объявление переменных  
3:   Dim Msg, Title, Response As String  
4:  
5:   'выдаваемое в окне MsgBox сообщение  
6:   Msg = "Хотите закончить работу?"  
7:  
8:   'состав кнопок и тип значка  
9:   Style = vbYesNo + vbCritical + vbDefaultButton2  
10:  
11:  'заголовок окна  
12:  Title = "Выход из программы"  
13:  
14:  'вызов функции MsgBox с возвращаемым значением  
15:  Response = MsgBox(Msg, Style,  
16:                    Title)  
17:  'анализ возвращаемого значения  
18:  If Response = vbYes Then  
19:    Unload Me    'выгрузка формы  
20:  End If  
21:  
22: End Sub
```

Если вы запустите форму с такой процедурой обработки события **Click** и щелкните на кнопке **Отмена** или нажмете клавишу **Esc**, то на экран будет выдано сообщение, подобное приведенному на рис. 9.10. Эта процедура стала более дружественной, поскольку не сразу заканчивает работу окна диалога. Случайный щелчок на кнопке **Отмена** или нажатие на клавишу **Esc** не приведут к немедленному окончанию работы с диалогом.

**Рис. 9.10**

Процедура обработки сообщения от кнопки **Отмена** стала более дружественной



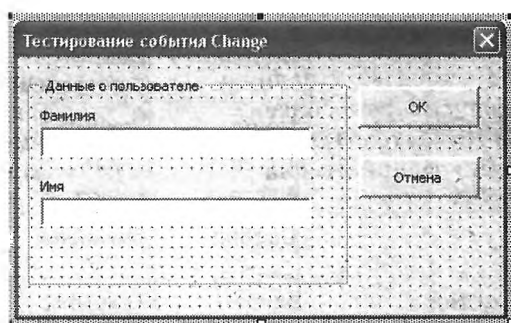
Теперь, когда у нас имеется возможность покинуть диалоговое окно, можно написать простые процедуры обработки событий для кнопок **OK** и **Справка**. Кнопка **OK** в начале разработки приложения может быть связана с процедурой, выдающей сообщение о том, что приложение пока не готово к эксплуатации. С кнопкой **Справка** для ваших первых приложений можно поступить по-разному: например, одним из решений может быть выдача длинного сообщения посредством функции **MsgBox**, а можно создать новое диалоговое окно



с элементом **Label**, свойством **Caption** которого будет также длинное сообщение о полезности приложения. Далее будет рассмотрено создание и вызов дополнительного диалогового окна приложения.

Очень полезной может оказаться процедура обработки события **Change** — при любом изменении свойств элементов управления (в конечном итоге, все, что происходит в диалоговом окне — это изменение тех или иных свойств) можно выполнять некоторые проверки на допустимость изменений и сообщать пользователю о возможных проблемах. Можно также использовать событие **Change** для подсказки пользователю о происходящих изменениях со свойствами элементов управления. В качестве примера создайте форму **UserForm1** так, чтобы она была похожа на представленную на рис. 9.11. Чтобы соответствующее форме диалоговое окно позволило продолжить работу с приложением, в этом окне необходимо в одном из текстовых полей ввести либо фамилию, либо имя пользователя. При изменении содержимого любого поля (свойство **Text**) инициируется процедура обработки события **Change**, в которой свойству **Enable** кнопки **OK\_button** (с заголовком **OK**) присваивается значение **True**. Кнопка **OK\_button** предназначена для изменения свойства **Caption** формы **UserForm2** и вывода этой формы на экран. Кнопка с заголовком **Отмена** всегда доступна и предназначена для выхода из программы. Свойство **Cancel** этой кнопки имеет значение **True**.

**Рис. 9.11**  
Форма для тестирования события **Change** для **TextBox**



Для создания такой формы выполните следующее:

1. Создайте новую форму с именем по умолчанию **UserForm1**. В **Properties Window** измените свойство **Caption** этого элемента на **Тестирование события Change**.
2. Поместите на форму **UserForm1** элемент **Frame**. По умолчанию его имя будет **Frame1**. Измените свойство **Caption** этого элемента на **Данные о пользователе**. (Размеры элемента не важны.)
3. Поместите на форму в область **Frame1** элемент **TextBox** (с именем по умолчанию **TextBox1**) и элемент **Label** с заголовком (свойство **Caption**) **Фамилия**. Имя метки для данного случая не имеет значения.
4. Поместите на форму в область **Frame1** элемент **TextBox** (с именем по умолчанию **TextBox2**) и элемент **Label** с заголовком **Имя**.
5. Поместите на форму (см. рис. 9.10) элемент **CommandButton** и измените его свойство **Name** на **Cancel\_button**, а свойство **Caption** на **Отмена**. Установите свойство **Cancel** в **True**, чтобы при нажатии на клавишу **Esc** программа выполняла ту же процедуру, что и при щелчке на кнопке **Отмена**.

6. Поместите на форму элемент **CommandButton** и измените его свойство **Name** на **OK\_button**, а свойство **Caption** — на **OK**. Измените свойство **Enabled** этой кнопки на **False**. Этим вы сделаете кнопку **OK** недоступной для пользователя после выдачи диалогового окна на экран в режиме выполнения. Чтобы кнопка стала доступной и приложение могло выполнить какую-либо работу с использованием кнопки **OK**, пользователь должен будет изменить содержимое одного из текстовых окон.
7. Откройте модуль класса формы **UserForm1** и запишите в него код, представленный в листинге 9.6.
8. Создайте форму **UserForm2** (без элементов управления) и запишите в модуль класса этой формы код, например, из листинга 9.2.

Тип элемента	Свойство, которое изменено	Значение	Примечание
UserForm	Name	UserForm1	Имя главной формы, на которое можно ссылаться в коде.
	Caption	Тестирование события Change	Заголовок окна (формы) в верхней части.
Frame	Name	Frame1	Имя, на которое можно ссылаться в коде.
	Caption	Данные о пользователе	
TextBox	Name	TextBox1	Имя, на которое можно ссылаться в коде.
Label	Name	Label1	
	Caption	Фамилия	Заголовок для текстового окна TextBox1.
TextBox	Name	TextBox2	Имя, на которое можно ссылаться в коде.
Label	Name	Label2	
	Caption	Имя	Заголовок для текстового окна TextBox2.
CommandButton	Name	Cancel_button	Имя кнопки, на которое можно ссылаться в коде.
	Caption	Отмена	Текст на кнопке: кнопка для окончания работы приложения.
CommandButton	Name	OK_button	
	Caption	OK	Текст на кнопке: кнопка для продолжения работы приложения.
	Enabled	False	Кнопка недоступна для пользователя сразу после выдачи диалогового окна на экран.
UserForm	Name	UserForm2	Имя формы, загружаемой из главной при щелчке на кнопке OK.

Выполните команду **Run | Run Sub/UserForm**. Если вы не будете редактировать (изменять) содержимое текстовых окон, вам будет доступна только кнопка **Выход**. Как только вы измените содержимое какого-либо текстового окна (**Фамилия** или **Имя**), инициируется событие **Change** и свойству **Enabled** кнопки **OK** будет присвоено значение **True** (строка 17 листинга 9.6). Так как с кнопкой **OK** связана процедура обработки события **Click**, которая загружает форму **UserForm2**, пользователь получает возможность дальнейшей работы. При этом свойству **Caption** формы присваивается сумма содержимого полей **TextBox1** и **TextBox2**.

#### Листинг 9.6. Обработка события **Change**

```
1: 'Обработка события Click кнопки Выход
2: Private Sub Cancel_button_Click()
3:     MsgBox ("Заканчиваем программу!")
4:     Unload Me
5: End Sub
6:
7: 'Обработка события Click кнопки OK
8: Private Sub OK_button_Click()
9:
10:     'Изменить заголовок диалогового окна UserForm2
11:     UserForm2.Caption = TextBox1.Text + TextBox2.Text
12:     UserForm2.Show           'загрузка окна UserForm2
13: End Sub
14:
15: 'Обработка события Change текстового окна с меткой Фамилия
16: Private Sub TextBox1_Change()
17:     Next_button.Enabled = True
18: End Sub
19:
20: 'Обработка события Change текстового окна с меткой Имя
21: Private Sub TextBox2_Change()
22:     Next_button.Enabled = True
23: End Sub
```

## Редактирование элементов управления на форме

Поместив на форму элемент управления, вы имеете возможность изменять его размеры, двигать, копировать, удалять, форматировать (изменять размер, стиль, гарнитуру шрифта) или изменять его свойства (такие как имя, заголовок и так далее).

### Выбор элементов управления

Чтобы отредактировать элемент управления, вы должны сначала выделить его щелчком мыши. Выделенный элемент имеет серую границу с маркерами изменения размеров, как показано для кнопки на рис. 9.5. Чтобы выделить несколько элементов управления сразу, удерживайте нажатой клавишу **Shift**, когда щелкаете по элементам управления, которые хотите выделить. Выделять сразу несколько элементов управления имеет смысл, если вы хотите выполнить форматирование, которое будет распространяться на все элементы формы. Это может быть, например, изменение размера шрифта, стиля или

гарнитуры. Выбор нескольких элементов управления может также оказаться полезным, если вы хотите переместить эти элементы одновременно.

Вы можете временно сгруппировать несколько элементов управления, воспользовавшись командой **Format | Group** (Формат | Группировать). Если элементы управления сгруппированы, они ведут себя как один объект, когда вы их выбираете, перемещаете, копируете или редактируете. Элементы управления остаются сгруппированными до тех пор, пока вы не примените к выделенной группе команду **Format | Ungroup** (Формат | Разделить).

### Перемещение элементов управления

Чтобы переместить элемент управления на новое место в форме, сначала выделите его. После этого поместите указатель мыши поверх любой части серой границы, окружающей выбранный элемент управления, за исключением маркеров изменения размеров. Теперь щелкните и перетащите этот элемент на новое место.

### Изменение размеров элементов управления

Чтобы изменить размер элемента управления, сначала выделите этот элемент. После этого поместите курсор мыши поверх любого из маркеров изменения размеров. Когда курсор находится на маркере изменения размера, он меняет свою форму на двойную стрелку, показывающую направление, в котором вы можете изменять размер выбранного элемента управления. Теперь нажмите (не отпуская) кнопку мыши и передвиньте маркер изменения размера на нужное место. В процессе перемещения редактор VB будет отображать контур, показывающий новые границы элемента управления. Когда элемент управления станет требуемого размера, отпустите кнопку мыши, редактор VB перерисует элемент управления в соответствии с вновь заданным размером.

### Копирование, вставка и удаление элементов управления

Один из самых простых способов создать несколько одинаковых элементов управления, таких как несколько флажков или кнопок, состоит в том, чтобы сначала создать один элемент управления, скопировать его, после чего поместить скопированный элемент в форму столько раз, сколько необходимо. Чтобы скопировать элемент управления, просто выберите его и затем воспользуйтесь командой **Edit | Copy** (Правка | Копировать) или нажмите **Ctrl+C**.

После того как элемент управления скопирован, вы можете вставить его в ту же или другую форму, используя команду **Edit | Paste** (Правка | Вставить) или нажав **Ctrl+V**. После выполнения операции вставки переместите элемент управления в предназначенное для него место.

Чтобы удалить элемент(ы) управления, выделите его (их), после чего нажмите клавишу **Delete**. Редактор VB удалит выделенный(-ные) элемент(-ы) управления.

### Редактирование или форматирование заголовков элементов управления

Для редактирования текста заголовков таких элементов управления, как кнопка, переключатель или флажок, просто щелкните по этому тексту. Редактор VB поместит в текст заголовка курсор вставки. Теперь можно редактиро-

вать текст, используя любые стандартные команды редактирования текста Windows, с которыми вы уже знакомы. Однако для того, чтобы изменить гарнитуру, стиль или размер шрифта, необходимо вывести для соответствующего элемента управления **Properties Window** (окно свойств) и изменить свойство **Font**.

Чтобы отредактировать текст заголовка формы, отредактируйте свойство формы **Caption** в **Properties Window**. А для форматирования текста заголовка формы воспользуйтесь свойством формы **Font**.

Некоторые элементы управления, такие как поля (**TextBox**), списки (**ListBox**) и поля со списком (**ComboBox**) не имеют встроенной надписи (или заголовка). Чтобы пометить такие элементы на форме, используйте элемент **Label**.

## Управление последовательностью перехода

В активном состоянии ваша форма будет вести себя подобно другим диалоговым окнам Windows. Пользователь вашего диалогового окна может переходить от одного элемента управления к другому вперед и назад, используя соответственно клавиши **Tab** или **Shift+Tab**. Порядок, в котором будут активизироваться элементы управления в ответ на нажатие пользователем клавиш **Tab** или **Shift+Tab**, называется *последовательностью перехода (tab order)* элементов управления. Обычно последовательность перехода для элементов управления выбирают так, чтобы она соответствовала (приблизительно) перемещению слева направо и сверху вниз. Вообще говоря, вы должны постараться так расположить элементы управления, чтобы последовательность перехода обеспечивала логическое перемещение от одного элемента управления диалогового окна к другому. Например, если пользователь вводит некоторые данные с использованием клавиатуры, то ему удобнее не прибегать к помощи мыши для перехода к следующим окнам ввода. В этом случае логичный для данного диалогового окна переход к различным элементам управления будет очень кстати.

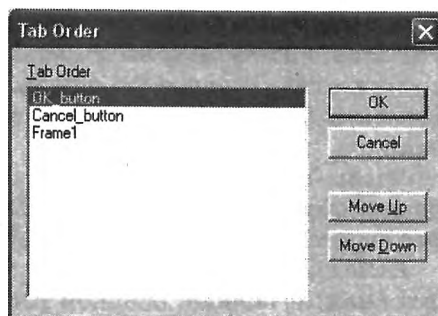
Редактор VB по умолчанию устанавливает для элементов управления последовательность перехода, определяемую порядком, в котором они были добавлены на форму. Очень часто, после того, как вы разместили на форме все элементы управления и несколько раз их передвинули, чтобы добиться приемлемого вида формы, установленная по умолчанию последовательность перехода уже больше не обеспечивает логической последовательности перемещения по элементам управления диалогового окна.

Для изменения последовательности перехода, выполните следующие действия:

1. Выберите команду **View | Tab Order** (Вид | Последовательность перехода); редактор VB отображает диалоговое окно **Tab Order** (Последовательность перехода), показанное на рис. 9.12. Список **Tab Order** выводит все элементы управления формы в существующей на данный момент последовательности.
2. В списке **Tab Order** выберите элементы, положение которых хотите изменить. (На рис. 9.12 выбран элемент управления с именем **TextBox1**.)

Рис. 9.12

Использование диалогового окна **Tab Order** для изменения последовательности перехода к элементам управления формы



- Используя кнопки **Move Up** (вверх) и **Move Down** (вниз), измените положение выбранного элемента управления в последовательности перехода. Щелчок по кнопке **Move Up** переместит элемент управления вверх по списку (то есть он будет активизироваться раньше), а щелчок по кнопке **Move Down** переместит этот элемент вниз по списку (то есть он будет активизироваться позже).
- Повторите пункты 2 и 3 для каждого элемента управления, пока не получите удовлетворяющую вас последовательность перехода для элементов управления формы.
- Выберите в диалоговом окне **Tab Order** кнопку **OK**, чтобы подтвердить сделанные вами изменения, после чего закройте окно.

Хотя элементы **Label** тоже появляются в диалоговом окне **Tab Order** и вы можете поменять их положение в списке, в VBA нельзя выбрать элемент управления **Label**. Этот элемент предназначен исключительно для вывода текста.

## Задание свойств формы и элементов управления в режиме разработки

Каждая форма и каждый элемент управления формы имеют собственный набор свойств точно так же, как и другие объекты VBA или host-приложения. Некоторые свойства формы и элемента управления проще и практичнее задать в режиме разработки, а не в коде VBA. Установленные вами свойства формы или элемента управления становятся для них новыми свойствами по умолчанию.

Например, хотя вы можете задать шрифт, цвет фона и цвет текста элемента управления с помощью VBA-кода, обычно нет необходимости изменять эти свойства в процессе выполнения кода. Вместо этого гораздо проще установить данные свойства в процессе разработки. Вы также можете задать для формы значения по умолчанию, задавая свойство **Value** элемента управления. Например, чтобы создать на форме флажок, который установлен по умолчанию, вы должны в режиме разработки, установить свойство **Value** для этого флажка равным **True**. Каждый раз, когда в процессе выполнения программы форма будет загружаться в память, флажок будет уже установлен.

Аналогично в процессе разработки, можно заполнить списки элементов управления, установив свойство **RowSource** списка элемента управления. В Excel вы можете с помощью свойства **RowSource** связать список с диапазоном ячеек листа и связать поле с ячейкой листа, установив свойство **ControlSource**.

Вы также можете задать значение по умолчанию для поля, введя текст непосредственно в элемент управления **TextBox** на форме. После этого каждый раз при выводе формы поле будет содержать введенный вами текст.

Некоторые свойства элементов управления, например, **Enabled** (доступность) или **Visible** (видимость на форме) удобно устанавливать в процессе выполнения кода. Если, например, в некоторое текстовое окно не введено имя файла, данные из которого необходимо считать, то кнопка для начала считывания вряд ли должна давать возможность запустить код для считывания, т.е. ее свойство **Enabled** должно иметь значение **False**, а не **True**. Или, например, нет смысла в кнопке с заголовком (свойство **Caption**) **Удалить**, если имя файла, который следует удалить, еще не введено в диалоговое окно.

Некоторые элементы управления вообще незачем отображать (свойство **Visible** должно быть равным **False**) до того момента, как они действительно станут необходимы.

В некоторых случаях бывает удобным одни и те же элементы управления использовать для разных целей, меняя, например, их свойство **Caption**. Если вы имеете кнопку с заголовком **Редактировать** и уже щелкнули на ней, будет вполне логично, если на этой кнопке появится заголовок **Сохранить** и код, соответствующий этой кнопке, с кода подготовки для редактирования (и изменения заголовка на **Сохранить**) изменится на код сохранения результатов редактирования (и изменения заголовка **Редактировать**).

В режиме разработки свойства формы или элемента управления устанавливаются в **Properties Window**, где перечислены все свойства объекта, которые вы можете установить в этом режиме. Конкретные свойства, появляющиеся в этом окне, зависят от выбранного вами элемента управления. На рис. 9.13 показано **Properties Window** для кнопки с именем (свойством **Name**) **CmdOK** и заголовком (**Caption**) **Вычислить** для некоторой формы.

**Properties Window** имеет две страницы (вкладки). Первая — показана на рис. 9.13 и содержит список свойств объекта в алфавитном порядке. Вторая страница, показанная на рис. 9.14, содержит список свойств, разделенных на категории. Заметьте, что различные категории выделены в **Properties Window** жирным шрифтом: **Appearance** (Вид), **Behavior** (Поведение), **Font** (Шрифт) и так далее. Вы можете свернуть или развернуть список свойств конкретной категории, щелкнув по квадратику слева от заголовка каждой из категорий.

**Рис. 9.13**  
**Properties Window** для кнопки, показывающее список свойств в алфавитном порядке

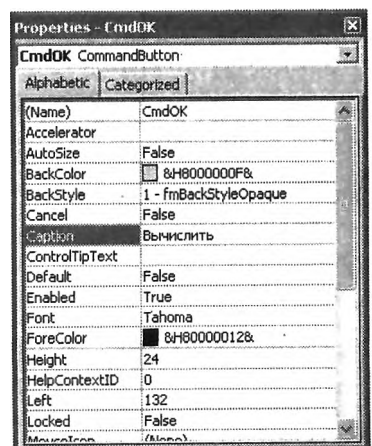
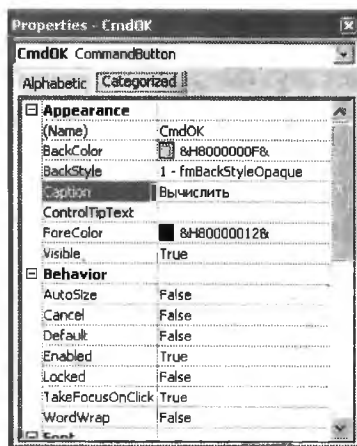


Рис. 9.14

**Properties Window** для кнопки, показывающее список свойств по категориям



Чтобы задать свойство формы или элемента управления, щелкните в поле, соответствующем свойству, которое хотите изменить. Для большинства свойств появится раскрывающийся список. Щелкните по кнопке списка и выберите из списка нужное значение свойства. Другие свойства (такие как **Name** и **Caption**) дают возможность ввести любой необходимый текст. Некоторые свойства при их выборе выводят кнопку с многоточием (...). Щелчок по такой кнопке открывает дополнительные диалоговые окна, помогающие задать это свойство.

Чтобы изменить свойства формы или элемента управления, выполните следующие действия:

1. Выберите элемент управления, свойства которого хотите изменить. Чтобы выделить форму, щелкните где-нибудь на поверхности формы, не занятой элементами управления.
2. Выберите команду **View | Properties Window** (Вид | Окно свойств) для отображения **Properties Window**. (Вы можете сделать это и щелчком на кнопке **Properties Window** на панели инструментов редактора VB или, нажав **F4**.)
3. Установите необходимое свойство для выбранного элемента управления.
4. Щелкните по кнопке **Close** (закрыть) в верхнем левом углу окна **Properties Window**, чтобы его закрыть.

Итак, вы познакомились с основными положениями создания и управления диалоговыми окнами и их элементами. Более полную информацию о каждом элементе управления можно получить из справочной системы VBA.

## Использование дополнительных элементов управления

Вам очень скоро могут понадобиться элементы управления, которых нет на **Toolbox**. Например, самыми полезными элементами управления являются элементы, отображающие данные в таблице — сетке (в Excel панель **Toolbox** содержит элементы типа сетки, но таких элементов можно найти больше, чем содержит **Toolbox** по умолчанию). Другая группа часто используемых элементов — элементы управления для работы с внешними базами данных. И так далее.

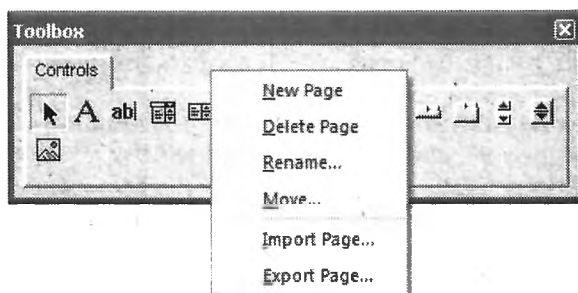


Многие элементы управления по умолчанию не помещаются на **Toolbox**. Этот «недостаток» легко исправить, воспользовавшись диалоговым окном, которое «отвечает» за то, какие именно элементы управления должны находиться на **Toolbox**. Но прежде чем «отправляться на поиски» дополнительных элементов управления, подготовьте себе место на **Toolbox** для этих дополнительных элементов. Если этого не сделать, то дополнительные элементы будут размещаться на вкладке **Controls** панели **Toolbox**. Вы, конечно, могли этого и не заметить, но **Toolbox**, действительно, содержит вкладку и позволяет вам добавлять и удалять свои вкладки для размещения на них элементов управления.

Добавить вкладку на **Toolbox** очень просто: правым щелчком на пустом месте панели **Toolbox** вызовите контекстное меню (рис. 9.15) и выберите команду **New Page**. В результате выполнения этой команды к **Toolbox** будет добавлена вкладка с именем по умолчанию (рис. 9.16).

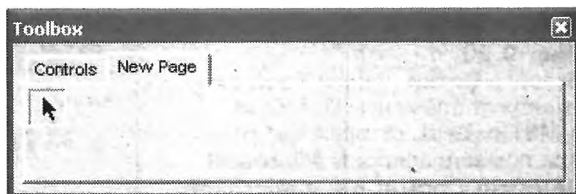
**Рис. 9.15**

Для добавления вкладки к **Toolbox** правым щелчком на пустом месте этой панели вызовите контекстное меню и выберите команду **New Page**



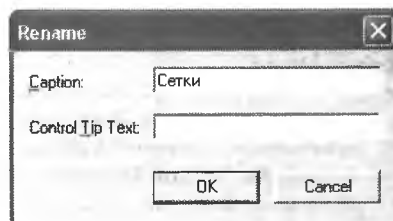
**Рис. 9.16**

В результате выполнения команды **New Page** к **Toolbox** будет добавлена вкладка с именем по умолчанию



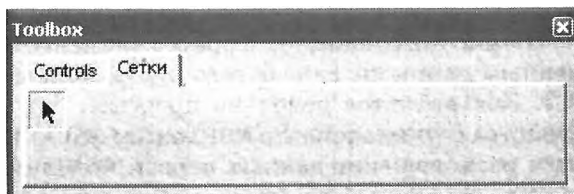
**Рис. 9.17**

В этом окне можно переименовать любую вкладку



**Рис. 9.18**

Вкладка имеет наименование **Сетки**

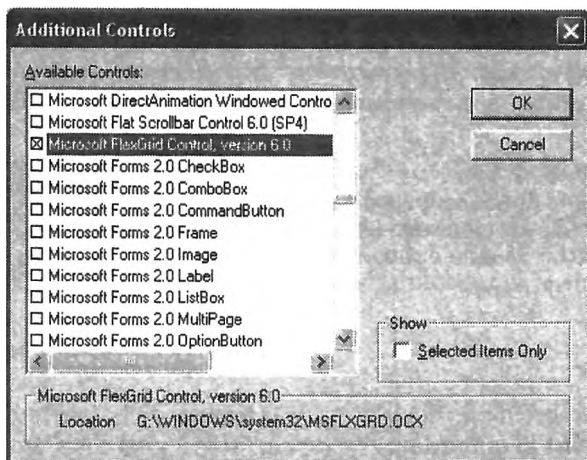


При помощи того же контекстного меню можно переименовать любую вкладку в окне **Rename** (рис. 9.17). Например, на рис. 9.18 вкладка имеет наименование **Сетки**.

Для добавления элементов управления на текущую вкладку следует выбрать команды **Tools | Additional Controls** и в появившемся окне **Additional Controls** (рис. 9.19) указать группу необходимых элементов управления (фактически, DLL-файл). На рис. 9.20 вкладка **Сетки** панели **Toolbox** содержит элементы **DataGrid** и **MSFlexGrid**, которые выбраны при помощи флажков **Microsoft DataGrid Control 6.0** и **Microsoft FlexGrid Control, version 6.0**, соответственно.

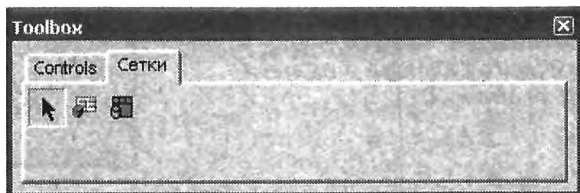
**Рис. 9.19**

В окне **Additional Controls** следует указать группу необходимых элементов управления



**Рис. 9.20**

Вкладка **Сетки** панели **Toolbox** содержит элементы **DataGrid** и **MSFlexGrid**, которые выбраны при помощи флажков **Microsoft DataGrid Control 6.0** и **Microsoft FlexGrid Control, version 6.0**, соответственно

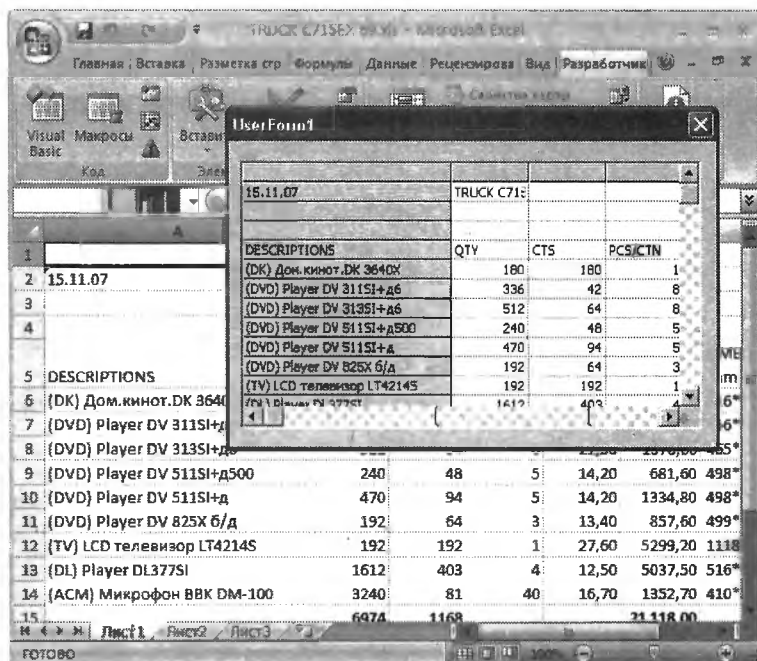


### Замечание

Наименование окна **Additional Controls**, вообще-то, не означает, что оно управляет наличием на панели **Toolbox** только дополнительных (additional) элементов. Откройте окно **Additional Controls** для вкладки **Controls** и можете увидеть список всех выделенных флажков для элементов управления, помещенных на эту вкладку по умолчанию.

На рис 9.21 показана форма с элементом управления **MSFlexGrid**, заполненным данными из рабочего листа. Код модуля формы приведен в листинге 9.7. Код является довольно простым: сначала в строках (6–9) описываются свойства сетки (элемента **MSFlexGrid**) — количество строк и колонок, ширина и расположение данных первой колонки.

**Рис. 9.21**  
Форма с элементом управления **MSFlexGrid**, заполненным данными из рабочего листа



В строках 12–20 при помощи двух циклов (один из которых вложен в другой) выполняется считывание данных из листа с именем **Лист1** текущей рабочей книги и заполнение сетки.

#### Листинг 9.7. Заполнение сетки данными из рабочего листа

```
1: Private Sub UserForm_Initialize()  
2: ' код заполнения сетки данными из рабочего листа  
3:  
4:     'размеры сетки:  
5:     MSFlexGrid1.Rows = 20 'кол-во строк  
6:     MSFlexGrid1.Cols = 7 'кол-во столбцов  
7:     MSFlexGrid1.ColWidth(0) = 2600 'ширина первого столбца  
8:     MSFlexGrid1.ColAlignment(0) = Left 'выравнивание данных  
9:  
10:    'заполнение сетки данными:  
11:    For j = 0 To MSFlexGrid1.Rows - 1  
12:        MSFlexGrid1.Row = j 'текущая строка  
13:        For I = 1 To MSFlexGrid1.Cols  
14:            MSFlexGrid1.Col = I - 1 'текущий столбец  
15:            'содержимое ячейки сетки:  
16:            MSFlexGrid1.Text =  
17:                Worksheets("Лист1").Cells(j + 1, I).Value  
18:        Next  
19:    Next  
20: Next  
21:  
22: End Sub
```

# Управление host-приложениями VBA

Вы уже знаете, что Visual Basic for Applications — язык программирования, предназначенный в основном для работы с имеющимся программным обеспечением и для расширения его возможностей. Приложения Microsoft Office имеют множество различных объектов, доступных VBA. Свойства и методы этих объектов позволяют посредством VBA-кода управлять, практически, любым аспектом работы приложений Microsoft Office. Из этой главы вы узнаете, как посредством VBA-кода управлять основными операциями в Word и Excel, а именно:

- Как работать с объектами Excel: **Workbook** и **Worksheet**.
- Как использовать методы объекта Excel, возвращающие объекты **Range**, включая то, как задать и выделить диапазоны ячеек в листе Excel.
- Как ввести в ячейки листа Excel значения и формулы.
- Как работать с объектами Word: **Document** и **Template**.
- Как использовать методы объекта Word, возвращающие разделы документов, включая объекты **Paragraphs**, **Characters** и **Range**.
- Как добавить, вырезать, скопировать или вставить данные в лист Excel или документ Word.

## Работа с Excel

В первой части этой главы вы узнаете, как управлять наиболее важными и основными объектами Excel. Во-первых, вы узнаете об объектах **Workbook** и **Worksheet**, а затем научитесь управлять информацией, содержащейся в объекте **Worksheet**. Далее из этой же главы вы узнаете, как управлять аналогичными объектами в Word.

### Возвращение объекта *Workbook*

В Excel каждая книга является объектом **Workbook**, а **Workbooks** — коллекцией всех открытых книг в текущем рабочем сеансе Excel. Для ссылок на конкретную книгу следует использовать коллекцию **Workbooks** объекта **Application**:

## Синтаксис

`Workbooks (Index)`

Index может быть:

- ☐ численным выражением, представляющим книгу, которую необходимо использовать (число 1 означает первую книгу, открытую в текущем рабочем сеансе, 2 — вторую и так далее);
- ☐ строковым выражением, представляющим имя открытой книги, которую вы хотите использовать.

Более распространенный и обычно наиболее удобочитаемый способ — использование в качестве *Index* текстовой строки. Листинг 10.1 содержит соответствующий пример.

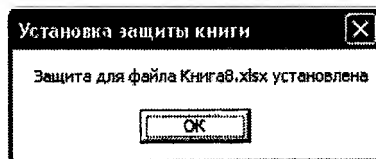
### Листинг 10.1. Использование коллекции **Workbooks**

```
1: Sub SetWorkbookProtection()  
2: 'Установка защиты книги  
3:  
4: Workbooks("Книга8.xlsx").Protect Structure:=True, Windows:=True  
5:     MsgBox "Защита для файла Книга8.xlsx установлена", , _  
6:         "Установка защиты книги"  
7: End Sub
```

Выражение в строке 4, **Workbooks("Книга8.xlsx")**, возвращает объектную ссылку на объект **Workbook** с именем **Книга8.xlsx**. Метод **Protect** устанавливает защиту книги. Оператор **MsgBox** (в строках 5–6) выводит сообщение, информирующее о том, что для данной книги была установлена защита (рис. 10.1). (Предполагается, что книга с именем **Книга8.xlsx** в данный момент открыта. Если это — не так, то вы получите сообщение о runtime-ошибке.)

**Рис. 10.1**

Сообщение о результате выполнения кода листинга 10.1



### Как открыть книгу

Если необходимая вам книга еще не открыта, воспользуйтесь методом **Open** для загрузки ее в память:

## Синтаксис

`Workbooks.Open (Filename)`

Аргумент *Filename* — текстовая строка, представляющая полный путь к книге: диск, папка и имя файла. Если вы не зададите имя диска или папку, Excel будет искать файл на текущем диске и в той папке, которая открыта в данный момент.

Пример использования метода **Open** представлен в листинге 10.2.

**Листинг 10.2.** Применение метода **Open** для открытия книги

```
1: Sub OpenWorkbook()  
2: 'Предлагает открыть и затем открывает ее  
3:  
4:     Dim Workbookname As String  
5:  
6:     Workbookname = _  
7:         InputBox("Введите полный путь к книге:")  
8:     If Workbookname <> "" Then  
9:         Workbooks.Open FileName:=Workbookname  
10:    End If  
11: End Sub
```

В этой процедуре переменная **WorkbookName** объявлена с типом **String** (строка 4). Функция **InputBox** предлагает пользователю ввести имя файла книги (строки 6 и 7). Оператор **If...Then** (в строке 8) проверяет, не отменил ли пользователь диалог ввода. Если — нет, другими словами, если строка **WorkbookName** не пуста, метод **Open** использует **WorkbookName** для открытия файла (строка 9).

**Как создать новую книгу**

Если в вашей программе требуется создать новую книгу, воспользуйтесь методом **Add** коллекции **Workbooks**:

**Синтаксис**

```
Workbooks.Add([Template])
```

Необязательный аргумент *Template* определяет тип книги, создаваемой Excel. Если вы опустите *Template*, Excel создаст книгу, заданную по умолчанию и содержащую некоторое количество чистых листов. Количество чистых листов в новой книге задается на вкладке **Общие** (General) в диалоговом окне **Параметры** (Options) (вызвать данное диалоговое окно можно, выбрав команду **Сервис | Параметры** (Tools | Options)). Вы можете также задать количество чистых листов в новой книге при помощи свойства **Application.SheetsInNewWorkbook**.

Для того чтобы создать новую книгу, основываясь на существующем шаблоне, подставьте в качестве аргумента *Template* строковое значение, определяющее имя шаблона. Если необходимый вам шаблон расположен не в загрузочной или не в альтернативной загрузочной папке, включайте в аргумент *Template* полный путь: имя диска, папку и имя шаблона.

Чтобы создать книгу, содержащую только один лист, используйте одну из следующих встроенных констант аргумента *Template* (эти константы определены в классе **xlWBATemplate**): **xlWBATWorksheet**, **xlWBATChart**, **xlWBATExcels4MacroSheet** или **xlWBATExcels4IntlMacroSheet**.

Листинг 10.3 демонстрирует пример процедуры, создающей новую книгу.

**Листинг 10.3.** Использование метода **Add** для создания новой книги

```

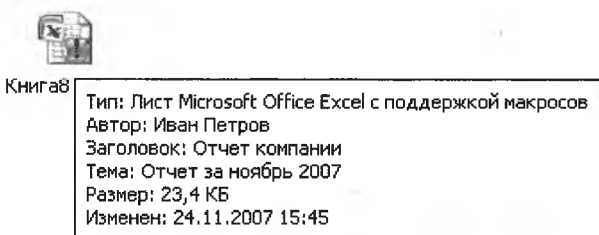
1: Sub CreateMonthlyReport()
2: 'Создание новой книги на основе файла-шаблона
3:
4:     Dim FilePath As String, FileName As String
5:
6:     FilePath = "Н:\Программирование на VBA 2007\VBA_BookVentura\3"
7:
8:     FileName = FilePath & "\Книга8.xlsm"
9:     Workbooks.Add Template:=FileName
10:        With ActiveWorkbook
11:            .Title = "Отчет компании"
12:            .Subject = "Отчет за ДЕКАБРЬ 2003"
13:            .Author = "Иван Петров"
14:        End With
15: End Sub

```

Код листинга 10.3 создает новую книгу (строка 9) на базе шаблона **Book15Template.xls** — обычный Excel-файл, который был предварительно создан и обработан. Любая новая книга автоматически становится активной книгой. В строках с 10 по 14 для добавления в файл новой книги некоторой общей информации — **Title**, **Subject** и **Author** — используется объект **ActiveWorkbook**. Для просмотра результатов работы кода строк 9–13 используйте или всплывающую Explorer-подсказку (см. рис. 10.2), или вкладку **Summary** диалогового окна **Properties** для данного файла.

**Рис. 10.2**

Общую информацию о файле можно увидеть, если просто на некоторое время поместить на его значке курсор мыши

**Как сделать книгу активной**

Если ваша VBA-программа работает с несколькими одновременно открытыми книгами, вам может понадобиться переключаться из одной книги в другую для вывода на экран, например, отчета или введенных данных.

Чтобы сделать открытый документ активным, используйте метод **Activate**:

**Синтаксис**

```
Object.Activate
```

В данном случае *Object* представляет любую допустимую объектную ссылку на лист, который необходимо сделать активным.

В листинге 10.4 — пример процедуры, использующей метод **Activate**.

**Листинг 10.4.** Использование метода **Activate** для переключения к книге

```
1: Sub ActivateTest()  
2:   'Изменение активной книги без вывода на экран  
3:  
4:   Dim SaveBook As String  
5:  
6:   SaveBook = ActiveWorkbook.Name  
7:  
8:   Application.ScreenUpdating = False  
9:   Workbooks("Балансовый отчет 1207.XLSX").Activate  
10:  
11:   ' код, выполняющий заполнение файла Балансовый отчет 1202.XLS  
12:   ' . . .  
13:  
14:   Workbooks(SaveBook).Activate  
15:   Application.ScreenUpdating = True  
16: End Sub
```

Эта простая процедура демонстрирует два принципа хорошего программирования. Во-первых, по возможности следует скрывать от пользователя промежуточные операции, выполняемые вашей программой. Например, если процедура содержит операторы, форматирующие диапазоны ячеек и введенные данные, выполняйте данные действия «за кулисами». Выводите для пользователя на экран лишь конечный результат. Сделать это можно, присваивая свойству **ScreenUpdating** объекта **Application** значение **False**.

Во-вторых, если вы устанавливаете книгу активной, потому что это требуется вашей программе, а не для того, чтобы пользователь мог увидеть другой файл, ваша процедура должна по завершении «возвращать» пользователя в то же самое место, где он находился при ее старте. Особенно это относится к малоопытным пользователям, которые приходят в замешательство, обнаружив, что по выполнении процедуры они оказываются совсем в другой книге, не имея, при этом, никаких разумных предположений для объяснения этого факта.

Назначение представленной в листинге 10.4 процедуры — переключиться из текущей книги в другую книгу (**Балансовый отчет 1202.XLS**), произвести в ней программным образом некоторые изменения, а затем вернуться к первоначальной книге. Сначала (в строке 6) имя текущей активной книги сохраняется в строковой переменной **SaveBook**. Далее, чтобы избавить пользователя от наблюдения за деталями выполнения процедуры, в строке 8 свойству **ScreenUpdating** присваивается значение **False**. В строке 9 книга **Балансовый отчет 1202.XLS** устанавливается активной. Строки, следующие далее, должны выполнять действия в книге **Балансовый отчет 1202.XLS**. Реальная программа, которая вносит изменения в **Балансовый отчет 1202.XLS**, в данном примере опущена. Для того чтобы «вернуть» пользователя в то же место, в котором он находился перед запуском процедуры, в строке 14 активной становится первоначальная книга. В строке 15 свойству **ScreenUpdating** присваивается значение **True**.

### Как сохранить книгу

Если ваша VBA-программа вносит в книгу изменения, вы должны предложить пользователю сохранить их. Сделать это довольно просто, или включив встроенную команду **Сохранить** в меню вашей программы, или добавив кноп-



ку **Сохранить** на панель инструментов программы, если программа имеет меню или панели инструментов.

Возможны случаи, когда у вас возникнет потребность сохранить книгу под управлением процедуры. Например, вы захотите создать собственную версию команды **Файл | Сохранить**. Более того, вы, возможно, попытаетесь предостеречь начинающего пользователя от выполнения неудачной последовательности команд, таких как закрытие книги, выход из программы или из Excel без сохранения результатов работы.

Для сохранения книги следует пользоваться методом **Save**:

---

### Синтаксис

*Object.Save*

В данном случае, *Object* — это объектная ссылка на открытый объект **Workbook**, который следует сохранить.

---

Существует также очень удобный метод **SaveCopyAs**. Этот метод сохраняет копию указанной книги на диск, никак не влияя на эту книгу в памяти.

Общий синтаксис метода **SaveCopyAs** выглядит следующим образом:

---

### Синтаксис

*Object.SaveCopyAs (Filename)*

*Object* — это объектная ссылка на объект **Workbook**, который будет сохранен, а *Filename* — строковое выражение для имени, которое следует использовать при сохранении копии книги.

---

В листинге 10.5 приведен код, использующий методы **Save** и **SaveCopyAs**.

---

### Листинг 10.5. Использование методов **Save** и **SaveCopyAs**

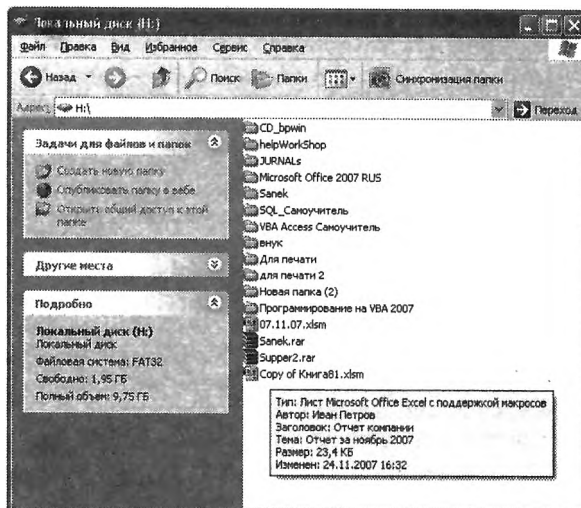
```
1: Sub BackUpToZIP()  
2:   'Сохраняет активную книгу и делает ее копию на ZIP-диске H:  
3:  
4:   Const SaveDrv = "H:"  
5:  
6:   With ActiveWorkbook  
7:     If Not .Saved Then .Save  
8:     .SaveCopyAs Filename:=SaveDrv & "\Copy of " & .Name  
9:   End With  
10: End Sub
```

---

Эта процедура сохраняет активную книгу и создает резервную копию на логическом диске **H** (рис.10.3). Процедура начинается с объявления единственной константы с именем диска. Оператор **With** выполняет несколько команд для объекта **ActiveWorkbook**.

**Рис. 10.3**

Копия активной книги на диске H — дополнительная гарантия сохранения результатов вашей работы



Тест **If...Then** (строка 7) проверяет свойство **Saved** книги. Если оно равно **False**, в книге имеются изменения, которые еще не были сохранены, поэтому процедура вызывает метод **Save**.

Имя резервной копии файла создается конкатенацией значения константы **SaveDrv**, литеральной константы **"\Copy of "** и свойства **Name** книги. Результат используется в качестве аргумента **Name** метода **SaveCopyAs** (строка 8), который сохраняет копию книги.

Метод **Save** не следует использовать для новой книги (которая прежде ни разу не сохранялась). Если вы все же сделаете это, Excel сохранит новую книгу под ее текущим именем, например, **КНИГА1.XLS**. Чтобы при сохранении присвоить книге имя используйте метод **SaveAs**:

## Синтаксис

```
Object.SaveAs([FileName] [,FileFormat] [,Password] [,
WriteResPassword] [, ReadOnlyRecommended] [, CreateBackup] [,
AccessMode] [, ConflictResolution] [, AddToMru])
```

**Object** — это объектная ссылка на объект **Workbook**, а **Filename** — строковое выражение для имени, которое следует использовать при сохранении копии книги.

**FileFormat** (тип **Variant**) — применяемый при сохранении файла формат. Для существующего на диске файла по умолчанию используется последний заданный формат, для нового файла — формат используемой версии Excel.

**Password** (тип **Variant**) — строка (не более 15-ти символов), которая используется в качестве пароля при открытии файла.

**ReadOnlyRecommended** (тип **Variant**) — аргумент, который при значении, равном **True**, приводит к появлению диалогового окна с рекомендацией открывать файл только на чтение.

**CreateBackup** (тип **Variant**) — аргумент, который при значении, равном **True**, приводит к созданию резервной копии сохраняемой книги.

**AccessMode** — значение одной из следующих констант:

- **xlExclusive** (монопольный доступ);

- ☐ **xlNoChange** (не менять режим доступа, действует по умолчанию);
- ☐ **xlShared** (многопользовательский доступ).

При использовании константы **xlExclusive** файл может быть открыт для записи только одним пользователем. Другой пользователь получит в диалоговом окне сообщение о том, что файл уже открыт. С другой стороны, константа **xlShared** дает возможность сразу нескольким пользователям работать с одним файлом.

При одновременной работе нескольких пользователей с одним файлом в любой момент времени любой пользователь может сохранять свои изменения в файле. При этом, если какой-либо другой пользователь уже сохранял данные на диске, эти данные попадают в файл того, кто делает в данный момент сохранение своих данных. Однако при такой совместной работе с одним файлом не всегда можно просто сохранить свои данные в файле на диске. Если вы, например, пытаетесь одновременно с другим пользователем изменить содержимое одной и той же ячейки, можете получить сообщение о конфликте доступа.

*ConflictResolution* — одна из следующих констант:

- ☐ **xlUserResolution** (отображать диалоговое окно разрешения конфликтов);
- ☐ **xlLocalSessionChanges** (автоматически принимать пользовательские изменения);
- ☐ **xlOtherSessionChanges** (принимать другие изменения вместо локальных пользовательских).

*AddToMru* (тип **Variant**) — аргумент, который (при равенстве значению **True**) указывает на то, что файл добавляется в список недавно используемых файлов. По умолчанию этот аргумент имеет значение **False**.

Метод **SaveAs** имеет и другие параметры, о которых вы можете узнать из справочной системы.

---

### Как закрыть книгу

После завершения работы с книгой, следует ее закрыть, чтобы освободить память и избежать «засорения» экрана. Закрыть книгу можно, воспользовавшись одной из форм метода **Close**:

### Синтаксис

---

```
Workbooks.Close  
Object.Close (SaveChanges)
```

При использовании первой синтаксической формы можно просто закрыть все открытые книги. Excel предлагает сохранить изменения в книгах перед тем, как их закрыть, если это необходимо. Вторая синтаксическая форма позволяет закрыть конкретную книгу, на которую указывает *Object*. Аргумент *SaveChanges* используется следующим образом:

- ☐ Если *SaveChanges* равен **True**, Excel сохраняет книгу автоматически, перед тем как ее закрыть.
  - ☐ Если *SaveChanges* равен **False**, Excel закрывает книгу без сохранения изменений.
  - ☐ Если вы опустите аргумент *SaveChanges*, Excel предложит (в случае необходимости) выполнить сохранение.
- 

Листинг 10.6 демонстрирует использование метода **Close**.

**Листинг 10.6. Использование метода Close**

```
1: Sub CloseAll()  
2: 'Закрывает все открытые книги и предлагает сохранить сделанные  
3: ' в них изменения  
4: Const qButtons = vbYesNo + vbQuestion  
5:  
6: Dim Book As Workbook  
7: Dim Ans As Integer  
8: Dim MsgPrompt As String  
9:  
10: For Each Book In Workbooks  
11:     If Not (Book.Name = ThisWorkbook.Name) Then  
12:         If Not Book.Saved Then  
13:             MsgPrompt = "Сохранить изменения в " & Book.Name & "?"  
14:             Ans = MsgBox(prompt:=MsgPrompt, Buttons:=qButtons)  
15:             If Ans = vbYes Then  
16:                 Book.Close SaveChanges:=True  
17:             Else  
18:                 Book.Close SaveChanges:=False  
19:             End If  
20:         Else  
21:             Book.Close  
22:         End If  
23:     End If  
24: Next Book  
25: End Sub
```

Данная процедура закрывает все открытые книги (за исключением книги, содержащей процедуру) и предлагает пользователю сохранить изменения для каждой книги, имеющей несохраненные изменения. Используйте процедуру, подобную этой, вместо метода **Workbooks.Close**, когда вам необходимо, чтобы пользователь не отменял операцию сохранения. Встроенная в Excel команда **Сохранить** имеет кнопку **Отмена**. После объявления константы и нескольких переменных (строки с 4 по 8) процедура начинает цикл **For Each** для обработки всех книг в коллекции **Workbooks**.

Поскольку закрытие книги, содержащей выполняемую в данный момент процедуру, приведет к ее остановке, строка 11 содержит оператор **If...Then**, который проверяет, не имеет ли книга, обрабатываемая в данный момент, то же самое имя, что и книга, содержащая процедуру **CloseAll**. Если — да, то часть кода, выполняющая закрытие книги пропускается; строки с 12 по 22 выполняются только в том случае, если книга не содержит процедуры **CloseAll**. (Свойство **ThisWorkbook** возвращает ссылку на книгу, содержащую выполняемый в данный момент код.)

Если книга содержит изменения, которые не были сохранены (строка 12), функция **MsgBox** (строка 13) запрашивает пользователя, необходимо ли сохранять изменения. Если пользователь выбирает **Yes** (то есть переменная **Ans** получит значение **vbYes**), процедура запускает метод **Close** с аргументом **SaveChanges**, равным значению **True** (строка 16). В противном случае процедура закрывает книгу с **SaveChanges**, равным значению **False** (строка 18). Если в книге отсутствуют несохраненные изменения, процедура запустит метод **Close** без аргументов (строка 21).

## Работа с объектами **Worksheet**

Объекты **Worksheet** содержат набор свойств и методов, которые вы можете использовать в своей программе. Эти свойства и методы позволяют делать листы активными и скрывать их, добавлять в книгу новые листы, а также перемещать, копировать, переименовывать или удалять их. В следующих разделах описывается каждая из этих операций над листом.

### Возвращение значения объекта **Worksheet**

Каждый лист является объектом **Worksheet**, а все листы данной книги образуют коллекцию **Worksheets**. Для ссылок на заданный лист используйте коллекцию **Worksheets** объекта **Workbook**:

### Синтаксис

---

*Object.Worksheets (Index)*

*Object* представляет ссылку на объект **Workbook**, содержащий лист. *Index* может быть:

- ☐ Числом, представляющим лист, который следует использовать. Число 1 означает первый лист в книге, 2 — второй лист и так далее.
  - ☐ Имя листа, который следует использовать, в виде строки. Это имя, которое появляется на вкладке листа.
- 

Листинг 10.7 представляет пример наиболее частого способа использования метода **Worksheets** — когда лист идентифицируется строкой текста.

### Листинг 10.7. Использование коллекции **Worksheets**

---

```
1: Sub SetWorksheetProtection()  
2:   'Активизирует защиту листа  
3:  
4:   Workbooks("Балансовый отчет 1207.xlsx").Worksheets("Расходы").Protect _  
5:       Contents:=True, Scenarios:=True  
6:   MsgBox "Защита листа 'Расходы' активизирована."  
7: End Sub
```

---

Объект **Workbooks("Балансовый отчет 1207.XLSX").Worksheets("Расходы")** возвращает ссылку на лист с именем **Расходы** в книге **Балансовый отчет 1207.xlsx**. Метод **Protect** задает защиту для содержимого и сценария. Процедура **MsgBox** информирует пользователя о том, что защита листа активизирована.

Для работы рассматриваемой программы необходимо, чтобы в коллекции **Workbook** имелась та книга, на которую выполняется ссылка — **Балансовый отчет 1202.xlsx**. Если это — не так, VBA выдаст сообщение об runtime-ошибке 9 «Subscript out of range», которое вряд ли поможет вам сразу понять причину ошибки.

Хорошим правилом при программировании является проверка наличия объекта в коллекции перед попыткой его использования. В следующем листинге защита листа книги **Балансовый отчет 1202.xlsx** выполняется в том случае, если эта книга находится в коллекции **Workbooks**. Наличие самого листа в книге не проверяется, хотя можно было бы проверить и это.

**Листинг 10.8. Использование коллекции Worksheets**

---

```
1: Sub SetWorksheetProtection()  
2: ' Активизирует защиту листа с проверкой  
3: ' наличия книги в коллекции Workbooks  
4: Dim ProtYes As Boolean  
5: ProtYes = False  
6:  
7: For Each Book In Workbooks  
8: If (Book.Name = "Балансовый отчет 1207.xlsx") Then  
9: Book.Worksheets("Расходы").Protect _  
10: Contents:=True, Scenarios:=True  
11: MsgBox "Защита листа 'Расходы' активизирована "  
12: ProtYes = True  
13: Exit For  
14: End If  
15: Next Book  
16:  
17: If Not ProtYes Then MsgBox "Книга не найдена!"  
18:  
19: End Sub
```

---

В строках 4 и 5 объявляется и инициализируется переменная **ProtYes**, которая используется в качестве индикатора наличия необходимой книги в коллекции **Workbooks**. В цикле **For Each** (строки 7–15) для каждой книги проверяется свойство **Name**. При совпадении этого свойства со строкой "**Балансовый отчет 1202.xlsx**" выполняется защита листа **Расходы** (строка 9), переменной **ProtYes** присваивается значение **True** (книга найдена), происходит выход из цикла при помощи оператора **Exit For**. В строке 17 выдается сообщение об отсутствии необходимой книги в коллекции **Workbooks**, если переменная **ProtYes** содержит значение **False**.

Если вам необходимо сослаться на активный лист, используйте свойство книги **ActiveSheet**. Если вам нужно сослаться на *все* страницы книги (или вы не уверены, какой тип листа необходимо выбрать), используйте вместо коллекции **Worksheets** коллекцию **Sheets**. Коллекция **Sheets** содержит не только листы, но и диаграммы.

**Как сделать лист активным**

Большинство книг содержат более одного листа, поэтому вашей программе может понадобиться переключаться с одного листа на другой. Например, отобразить один лист для ввода данных и затем переключиться на другой лист, чтобы просмотреть отчет или диаграмму. Выполнять переключение между листами можно при помощи метода **Activate**, имеющего следующий синтаксис:

**Синтаксис**

---

*Object*.Activate

*Object* представляет ссылку на объект **Worksheet**, который следует сделать активным.

---

Листинг 10.9 содержит пример использования метода **Activate**.

**Листинг 10.9.** Использование метода **Activate** для перехода на заданный лист

---

```

1: Sub DisplaySheet()
2:   ' Устанавливает активным лист "Расходы"
3:
4:   Dim Ans As Integer, qBtns As Integer
5:   Dim mPrompt As String
6:
7:   mPrompt = "Хотите просмотреть ""Расходы""?"
8:   qBtns = vbYesNo + vbQuestion + vbDefaultButton2
9:   Ans = MsgBox(prompt:=mPrompt, Buttons:=qBtns)
10:  If Ans = vbYes Then
11:    Workbooks("Балансовый отчет 1203.xlsx").Worksheets("Расходы").Activate
12:  End If
13: End Sub

```

---

В данной процедуре пользователю задается вопрос, хочет ли он просмотреть определенный лист, в данном случае — **Расходы** (строки 7–9 программы). Если пользователь выбирает кнопку **Yes**, то лист с именем **Расходы** становится активным (строка 11).

Для выбора листа используйте метод **Select**:

```
Object.Select
```

Данный метод полезен для создания трехмерных ссылок и определения листов, которые следует вывести на печать. *Трехмерная ссылка (three-dimensional reference)* — это ссылка на диапазон ячеек более чем одного листа.

Не следует без необходимости делать лист активным. Для большинства свойств и методов объекта **Worksheet** вы можете просто воспользоваться методом **Worksheets** для ссылок на лист, который нужно использовать. Например, следующий оператор возвращает стандартную ширину листа **График баланса**, не делая этот лист активным:

```
StdWidth = Worksheets("График баланса").StandardWidth
```

### Как создать новый лист

Коллекция **Worksheets** имеет метод **Add**, который вы можете использовать для добавления в книгу новых листов. Синтаксическая конструкция применения данного метода следующая:

### Синтаксис

---

```
Object.Worksheets.Add([Before] [, Count] [, Type])
```

*Object* представляет ссылку на объект **Workbook**, в который добавляется новый лист. Аргумент *Before* задает лист, *перед* которым добавляется новый лист, а аргумент *After* — лист, *после* которого добавится новый лист. (Нельзя использовать аргументы *Before* и *After* одновременно в одном и том же операторе.) Если опустить оба аргумента: как *Before*, так и *After*, VBA добавит новый лист перед активным листом. *Count* — количество вновь добавляемых листов. (Если опустить *Count*, метод **Add** добавит один лист.) *Type* — тип добавляемого листа. Существуют следующие возможные варианты аргумента *Type* (определенные в классе **XlSheetType**): **xlWorksheet** (устанавливается по умолчанию), **xlChart**, **xlExcel4MacroSheet** или **xlExcel4IntlMacroSheet**.

---

### Как переименовать лист

Имя листа — это текст, который появляется на ярлычке листа. В случае необходимости переименовать лист следует изменить свойство **Name** листа:

---

#### Синтаксис

*Object.Name*

*Object* — это лист, который будет переименован.

---

### Копирование и перемещение листа

Если вам необходимо расположить листы книги в определенном порядке, используйте методы **Copy** и **Move**. Оба метода имеют одинаковый синтаксис:

---

#### Синтаксис

*Object.Copy*([Before] [, After])

*Object.Move*([Before] [, After])

*Object* — объектная ссылка на лист, который необходимо скопировать или переместить. *Before* задает лист *перед* которым, а *After* — лист *после* которого будет помещен копируемый или перемещаемый лист. (Нельзя использовать аргументы *Before* и *After* одновременно в одном операторе.) Если опустить как *Before*, так и *After*, VBA создаст для копируемого или перемещаемого листа новую книгу.

---

Листинг 10.10 содержит процедуру, использующую метод **Move**.

---

#### Листинг 10.10. Использование метода **Move** для перемещения листа

```
1: Sub CreateWorksheet()  
2:   'Создание нового листа и перемещение  
3:   'его в конец книги  
4:  
5:   Dim NewSheet As String  
6:  
7:   Worksheets.Add before:=Worksheets(Worksheets.Count)  
8:   NewSheet = ActiveSheet.Name  
9:   With Worksheets(NewSheet)  
10:      .Move after:=Worksheets(Worksheets.Count)  
11:      .Activate  
12:   End With  
13:  
14: End Sub
```

---

С помощью метода **Add** в книгу добавляется лист (строка 7). В данном случае метод **Add** вызывается с необязательным аргументом *Before* для того, чтобы добавить новый лист непосредственно перед последним листом книги. Номер последнего листа книги определяется в строке 7 с помощью свойства **Count**, которое возвращает количество листов в книге. После чего новое название листа сохраняется в переменной **NewSheet**.



В строке 10 с помощью метода **Move**, новый лист помещается за последним листом книги. (Для того чтобы получить количество листов в книге, вновь используется свойство **Count**.) После этого новый лист устанавливается активным (строка 11).

Если вы разрабатываете приложение, в котором часто приходится создавать и перемещать листы, будет неплохой практикой устанавливать перед этими операциями команды отключения обновления экрана (**Application.ScreenUpdating = False**). При этом следует не забывать включать обновление экрана, если ваша процедура его отключала (**Application.ScreenUpdating = True**).

Помните, что вы можете добавить лист и после любого заданного листа. Процедура в листинге 10.10 добавляет лист перед последним листом и затем перемещает его, поскольку Excel не позволяет добавлять листы в конец книги. Использование аргумента *After*, совместно с методом **Add** позволяет добавить новый лист после любого существующего листа, за исключением последнего.

### Как удалить лист

В целях поддержания управляемости книгами и экономии места на диске ненужные листы следует удалять. Это особенно актуально в случаях, когда созданные вами приложения используют для хранения промежуточных результатов временные листы. Для удаления листа используйте метод **Delete**:

### Синтаксис

---

*Object.Delete*

*Object* — ссылка на лист, который следует удалить.

---

Листинг 10.11 демонстрирует пример использования метода **Delete**.

### Листинг 10.11. Использование метода Delete для удаления листа

---

```
1: Sub DeleteTemporarySheets()  
2:   'Удаление всех временных листов  
3:  
4:   Dim Sheet As Worksheet  
5:  
6:   Application.DisplayAlerts = False  
7:  
8:   For Each Sheet In Workbooks("Балансовый отчет 1207.xlsx").Worksheets  
9:     If InStr(1, Sheet.Name, "Временный") Then  
10:       Sheet.Delete  
11:     End If  
12:   Next Sheet  
13:  
14:   Application.DisplayAlerts = True  
15: End Sub
```

---

Эта процедура проверяет каждый лист книги и удаляет все ранее созданные временные листы. Процедура «полагает», что временные листы должны называться *Временный1*, *Временный2* и так далее.

Процедура объявляет объектную переменную **Sheet** (типа **Worksheet**) и задает свойство **DisplayAlerts** объекта **Application**, равным значению **False** (строка 6). Таким образом подавляется обычное диалоговое Excel-окно подтверждения, которое появляется всякий раз, когда вы пытаетесь удалить лист. В строке 8 начинается цикл **For Each**, перебирающий все листы книги **Балансовый отчет 1202.xlsx**. Функция **InStr** проверяет каждый лист на наличие в названии листа строки «**Временный**». Если такая строка обнаруживается, то (в строке 10 кода) этот лист удаляется. Когда данный процесс завершается, свойство **DisplayAlerts** вновь получает значение **True** (строка 14).

Процедура в листинге 10.11 будет работать правильно, если **Option Compare Text** является текущей глобальной установкой сравнения.

## Методы, возвращающие объекты **Range**

Наибольшую долю при работе с листом составляют: ввод информации, копирование данных, а также выполнение форматирования, включая работу с ячейками, диапазонами ячеек и их именами. Нет ничего удивительного в том, что так много методов объекта **Excel** в **VBA**, в конечном счете, выполняют какие-либо действия с диапазоном ячеек.

При обычной диалоговой работе с **Excel** прежде чем работать с диапазоном ячеек листа, диапазон необходимо выделить. Для процедуры **VBA** это означает, что вы должны сослаться на диапазон ячеек листа, перед тем как сможете что-нибудь с ним сделать. Наиболее распространенным из всех объектов **Excel** является объект **Range**. Объект **Range** может быть одной ячейкой, строкой, колонкой, набором ячеек и даже трехмерным диапазоном значений (то есть набором ячеек нескольких листов).

### Использование **Range** метода

Наиболее легкий и прямой способ задать ячейку или диапазон ячеек состоит в использовании метода **Range**:

### Синтаксис

---

*Object*.Range (*Name*)

*Object* — ссылка на объект **Worksheet**, который содержит диапазон ячеек. Если вы опустите *Object*, **VBA** будет «считать», что метод применяется к объекту **ActiveSheet**. Аргумент *Name* — ссылка на диапазон ячеек или имя диапазона, введенное как текст. Метод **Range** работает также с именованными диапазонами ячеек.

---

Код листинга 10.12 демонстрирует пример использования метода **Range**.

### Листинг 10.12. Использование метода **Range**

---

```
1: Sub FormatRangeFont()  
2: 'С помощью метода Range задается шрифт для диапазона ячеек  
3:  
4: With Worksheets("Лист1")  
5:     .Range("A1:L1").Font.Size = 24  
6:     .Range("A1:L1").Font.Bold = True  
7:     .Range("A1:L1").Font.Name = "Times New Roman"
```

```
8:      End With
9:      End Sub
```

---

В этом примере все три оператора (строки с 5 по 7) возвращают диапазон ячеек **A1:L1** листа **Лист1**. (Обратите внимание на то, что диапазон ячеек следует заключать в кавычки.)

Процедура листинга 10.12 устанавливает некоторые атрибуты шрифта для заданного диапазона ячеек. **Font** — свойство объекта **Range**. **Font** возвращает ссылку на объект **Font**, связанный с заданным диапазоном значений. Свойства объекта **Font** — те же самые, что и атрибуты шрифта, которые можно задать при помощи диалогового окна **Шрифт**: полужирный, курсив, подчеркнутый и другие. Ниже в качестве примера представлено несколько<sup>1</sup> свойств объекта **Font** (во всех случаях **Object** — объект типа **Range**):

- **Object.Font.Bold**: Устанавливает (или отменяет) полужирный стиль шрифта (**True** или **False**).
- **Object.Font.Italic**: Устанавливает (или отменяет) курсив (**True** или **False**).
- **Object.Font.Underline**: Устанавливает эффект подчеркивания включенным или выключенным. Значение свойства **Underline** может быть равно одной из встроенных констант, определенных в классе **xlUnderlineStyle**: **xlUnderlineStyleNone**, **xlUnderlineStyleSingle**, **xlUnderlineStyleDouble**, **xlUnderlineStyleSingleAccounting** или **xlUnderlineStyleDoubleAccounting**.
- **Object.Font.Name**: Устанавливает наименование шрифта. Значение свойства **Name** определяется, например, в виде строки **"Arial"**.
- **Object.Font.Size**: Устанавливает размер шрифта (в пунктах).

В Excel имеется альтернативная синтаксическая конструкция для метода **Range**, применение которой требует двух аргументов:

## Синтаксис

---

*Object.Range(Cell1, Cell2)*

Как обычно, **Object** — является объектом типа **Worksheet**, содержащим диапазон значений ячеек. Аргумент **Cell1** определяет верхний левый угол диапазона, а **Cell2** — правый нижний угол. Каждый аргумент может быть текстовым адресом ячейки, объектом **Range**, состоящим из одной ячейки, или же целым столбцом или строкой.

---

Преимущество использования данной синтаксической конструкции заключается в разделении угловых значений в различных аргументах, что позволяет независимо изменять значение для каждого из «углов» под управлением процедуры. Например, вы можете задавать имена переменных, таких как **UpperLeft** и **LowerRight**, и возвращать объекты **Range** различных размеров в виде

`Range(UpperLeft, LowerRight)`

---

<sup>1</sup> Для того чтобы просмотреть полный список свойств **Font**, используйте инструмент **Object Browser**.

Не следует использовать в методе **Range** координаты диапазонов, если доступны их имена. Имена диапазонов облегчают восприятие и отладку программы.

## Использование метода Cells

Хотя для того чтобы выполнить ссылку на одну ячейку, можно воспользоваться методом **Range**, такую же возможность дает и метод **Cells**, который является более гибким:

### Синтаксис

---

*Object.Cells(RowIndex, ColumnIndex)*

*Object* — ссылка или на **Worksheet**, или на объект **Range**, содержащий ячейку, с которой необходимо работать. Если *Object* опущен, метод применяется к объекту **ActiveSheet**.

*RowIndex* — номер строки, в которой находится ячейка. Если *Object* — лист, то *RowIndex*, равный 1, ссылается на строку 1 листа. Если *Object* — диапазон значений, то *RowIndex*, равный 1, ссылается на первую строку диапазона.

*ColumnIndex* — номер столбца, в котором расположена ячейка. Вы можете ввести или букву (как буквенную константу или как строковую переменную), или число, задающее столбец. Если *Object* — лист, то *ColumnIndex*, равный "A" или 1, ссылается на столбец A листа. Если *Object* — диапазон значений, то *ColumnIndex*, равный "A" или 1, ссылается на первый столбец диапазона.

---

В листинге 10.13 приведен пример использования метода **Cells**.

### Листинг 10.13. Использование метода Cells

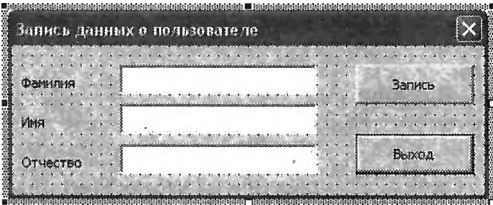
---

```
1: Sub CmdWrite_Click()
2:   'Ввод данных из диалогового окна формы пользователя в лист
3:
4:   Dim I As Integer
5:   Dim DBNewRow As Integer
6:   Dim NewRange As String
7:
8:   With Range("База_данных")
9:     'находится новая строка данных
10:    DBNewRow = .Row + .Rows.Count
11:
12:    'ввод данных из диалогового окна в ячейки новой строки
13:    For I = 1 To .Columns.Count
14:      Cells(DBNewRow, (I + .Column) - 1).Value =
15:        UserForm1.Controls("TextBox" & CStr(I)).Text
16:    Next I
17:
18:    'расширение диапазона базы данных
19:    NewRange = "=" & .Parent.Name & "!"
20:    NewRange = NewRange & Cells(.Row, .Column).Address & ":"
21:    NewRange = NewRange &
22:      Cells(DBNewRow, (.Column + .Columns.Count) - 1).Address
23:    Names("База_данных").RefersTo = NewRange
24:  End With
25: End Sub
```

---

Эта процедура — часть приложения поддержки базы данных. Пользователь вводит в «базу данных» информацию при помощи диалогового окна (подобного тому, что показано на рис. 10.4). Процедура `CmdWrite_Click` вызывается при нажатии кнопки **Запись** диалогового окна. Процедура `CmdWrite_Click` записывает новые данные в лист в конец диапазона ячеек базы данных и увеличивает диапазон ячеек базы данных для включения новой строки.

**Рис. 10.4**  
Диалоговое окно (в режиме разработки) для ввода данных



Описание элементов управления, на которые ссылается код процедуры листинга 10.13, приведено в следующей таблице:

Тип элемента	Свойство	Значение	Примечание
UserForm	Name	UserForm1	Имя формы, на которое можно ссылаться в коде.
	Caption	Запись данных	Заголовок окна (формы) в верхней части.
CommandButton	Name	CmdWrite	Имя кнопки, на которое можно ссылаться в коде.
	Caption	Запись	Текст на кнопке.
	Default	True	При нажатии на клавишу Enter инициируется событие Click кнопки.
CommandButton	Name	CmdExit	Имя кнопки, на которое можно ссылаться в коде.
	Caption	Выход	Текст на кнопке.
	Cancel	True	При нажатии на клавишу Esc инициируется событие Click кнопки.
Label	Name	Label1	Имя метки, на которое можно ссылаться в коде.
	Caption	Фамилия	Текст на метке.
Label	Name	Label2	Имя метки, на которое можно ссылаться в коде.
	Caption	Имя	Текст на метке.
Label	Name	Label3	Имя метки, на которое можно ссылаться в коде.
	Caption	Отчество	Текст на метке.

Тип элемента	Свойство	Значение	Примечание
TextBox	Name	TextBox1	Имя поля, на которое можно ссылаться в коде.
TextBox	Name	TextBox2	Имя поля, на которое можно ссылаться в коде.
TextBox	Name	TextBox3	Имя поля, на которое можно ссылаться в коде.

Пример работающего окна для ввода данных при помощи кода листинга 10.13 приведен на рис. 10.5.

**Рис. 10.5**

Процедура **CmdWrite\_Click** листинга 10.13 работает с данными, вводимыми с использованием диалогового окна, подобного этому

В строках с 4 по 6 объявляются переменные, используемые в процедуре. Строка 8 начинается оператором **With**, задающим диапазон ячеек, на которые ссылается имя **База\_данных**. В строке 10 переменной **DBNewRow** присваивается значение, соответствующее строке, расположенной сразу же за диапазоном ячеек базы данных — это строка, в которой должны быть сохранены новые данные.

Цикл **For...Next** (в строках с 13 по 16) записывает значения из диалогового окна формы пользователя в лист. Счетчик цикла (**I**) «пробегаёт» значения от 1 до числа столбцов диапазона ячеек, полученного при нахождении свойства **Count** коллекции **Columns** для объекта **Range**. (Коллекция **Columns** содержит все столбцы объекта **Range**, а его свойство **Count** возвращает число столбцов.)

Внутри цикла метод **Cells** (строка 14) возвращает каждую ячейку в новой строке и свойство **Value** ячейки устанавливается в значение, равное соответствующему тексту в элементе управления окна редактирования. Предполагается, что каждый элемент управления окна редактирования в форме пользователя имеет имя (по умолчанию) **TextBoxN**, где **N** — номер текстового окна в последовательности помещения его на форму и соответствующего столбца в диапазоне ячеек базы данных. Таким образом, в строке 15 собирается имя элемента управления окна редактирования, из которого будет преобразовано значение данных посредством конкатенации строки, эквивалентной переменной текущего счетчика цикла, с буквенной константой **"TextBox"**. (Вместо нескольких имен текстовых окон можно использовать один массив таких окон с одним именем и разными индексами.)

Чтобы включить новую строку в «базу данных», необходимо изменить диапазон ячеек «базы данных». Переопределить диапазон, на который ссылается конкретное имя, можно, изменив свойство **RefersTo** этого имени. (Именованные диапазоны ячеек листа, являются объектами типа **Name** и доступны посредством коллекции **Names** листа.) Свойство **RefersTo** должно содержать

строку вида "*=sheetreference*", где *sheet* — имя листа, содержащего именованный диапазон, а *reference* — адрес в виде **R1C1**.

В строках с 19 по 23 собирается соответствующим образом отформатированная строка, задающая новый диапазон «базы данных». В строке 19, чтобы получить имя листа, содержащего данный объект **Range**, к адресу диапазона с помощью свойства **Parent** объекта **Range** добавляется имя листа. Далее (в строке 20) с помощью свойства **Address** объекта **Cells**, возвращающего строку с адресом ячейки, на которую сделана ссылка, добавляются координаты верхнего левого угла нового диапазона ячеек (в данном случае это те же самые строка и столбец, на которые указывает текущая ссылка диапазона **База\_данных**). В строках 21–22 вновь с использованием свойства **Address** (на этот раз — для ячейки, которая находится на пересечении новой строки и прежнего числа столбцов в диапазоне ячеек «базы данных») добавляются координаты нижнего правого угла нового диапазона ячеек. Наконец, (в строке 23) свойству **RefersTo** присваивается новое значение диапазона **База\_данных**, на который ссылаются посредством коллекции **Names**.

### Использование метода **Offset**

При определении объектов **Range** конкретный диапазон адресов, который необходимо использовать, часто бывает неизвестен. Например, у вас может возникнуть необходимость сослаться на ячейку, которая расположена на две строки ниже и один столбец правее от активной ячейки. Хотя вы, в общем-то, можете определить адрес активной ячейки и вычислить адрес ячейки, которая вам необходима, в Excel VBA предусмотрен более легкий и более гибкий путь — использование метода **Offset**. Метод **Offset** возвращает объект типа **Range**, который смещен относительно заданного диапазона ячеек на заданное количество строк и столбцов.

### Синтаксис

---

```
Object.Offset([RowOffset] [,ColumnOffset])
```

*Object* является ссылкой на исходный объект **Range**. *RowOffset* — число строк, на которое смещается *Object*. Вы можете использовать положительное число (для смещения вниз), отрицательное число (для смещения вверх) или 0 (для использования той же строки). Если вы опустите *RowOffset*, будет использовано значение по умолчанию — 0.

---

Листинг 10.14 демонстрирует процедуру, использующую метод **Offset**.

### Листинг 10.14. Использование метода **Offset** для ссылки на диапазон ячеек

---

```
1: Sub SelectData()  
2:   'Выделяется область данных диапазона базы данных  
3:  
4:   Dim DBRows As Integer  
5:  
6:   Worksheets("Sheet1").Select  
7:   With Range("База_данных")  
8:     DBRows = .Rows.Count  
9:     .Offset(1, 0).Resize(DBRows - 1, .Columns.Count).Select  
10:   End With  
11: End Sub
```

---

Данная процедура выделяет область данных диапазона ячеек с именем **База\_данных**, то есть диапазон, в который предполагается включать только данные (без заголовков столбцов), опуская для этого из текущего выбора первую строку именованного диапазона. Выполнение данной процедуры полезно, если необходимо выполнить над данными в таблице глобальные операции, такие как сортировка или форматирование.

В строке 6 выбирается лист с именем **Sheet1**, а оператор **With** в строке 7 задает для этого листа объект **Range** с именем **База\_данных**. В строке 8 с помощью свойства **Rows.Count** объект **Range** получает число строк диапазона ячеек, которое сохраняется в переменной **DBRows**.

В строке 9 метод **Offset** (с аргументами 1 и 0) возвращает диапазон, который является смещением диапазона **База\_данных** на одну строку вниз. Поскольку этот диапазон включает (предположительно) пустую строку, находящуюся за диапазоном **База\_данных**, вы должны эту дополнительную строку удалить. Чтобы выполнить данную операцию, можете воспользоваться свойством **Resize**, изменяющим размер диапазона:

## Синтаксис

---

```
Object.Resize([RowSize] [,ColumnSize])
```

*Object* — объектная ссылка на диапазон, размер которого изменяется. *RowSize* — число строк в возвращаемом диапазоне. *ColumnSize* — число столбцов в возвращаемом диапазоне.

---

В строке 9 при помощи выражения **Resize(DBRows-1, .Columns.Count)** удаляется ненужная нижняя строка смещенного диапазона.

Метод **Select** (в строке 9) выделяет новый диапазон ячеек.

### Другие методы и свойства, возвращающие диапазоны ячеек

Методы **Range**, **Cells** и **Offset** являются наиболее часто используемыми методами для возвращения объектов **Range**, но они далеко не единственные. На примере кода листинга 10.14 вы познакомились с тем, как с помощью метода **Resize** можно вернуть диапазон заданного размера. Вот лишь некоторые методы и свойства, возвращающие объекты типа **Range**:

- **[cellRef]**. Вы можете вернуть одну ячейку, поместив ссылку на нее в квадратные скобки. Например, **[A1].Font.Size=16** устанавливает размер шрифта ячейки **A1** в значение **16**. Обратите внимание на то, что в данном случае для ссылки на ячейку не используется никаких кавычек.
- **Object.Rows (Index)**. Данный метод возвращает строку листа или диапазона ячеек, заданных при помощи *Object*. Если вы опустите *Object*, VBA использует **ActiveSheet**. *Index* — номер строки. Если *Object* является листом, *Index*, равный 1, ссылается на строку 1 листа. Если же *Object* является диапазоном, то *Index*, равный 1, ссылается на первую строку диапазона.
- **Object.EntireRow**. Данное свойство возвращает целую строку или строки, содержащие диапазон ячеек, заданных при помощи *Object*.



- **Object.Columns (Index)**. Данный метод возвращает столбец листа или диапазона ячеек, заданных при помощи *Object*. Если вы опускаете *Object*, VBA использует *ActiveSheet*. *Index* — буква или номер столбца. Если *Object* является листом, *Index*, равный "A" или 1, ссылается на столбец A листа. Если же *Object* является диапазоном ячеек, то *Index*, равный "A" или 1, ссылается на первый столбец диапазона.
- **Object.EntireColumn**. Данное свойство возвращает целый столбец или столбцы, содержащие диапазон ячеек, заданных при помощи *Object*.
- **Object.CurrentRegion**. Данное свойство возвращает *текущую область* диапазона *Object*. Текущая область определяется как область, окружающая текущую ячейку или диапазон ячеек, ограниченный пустыми строками сверху и снизу и пустыми столбцами слева и справа.

## Работа с Cells и Ranges

Теперь, когда вы знаете, как возвращать объект **Range**, вы можете использовать все преимущества применения многочисленных свойств и методов **Range**. В следующем разделе рассматриваются некоторые из этих свойств и методов. Для того чтобы увидеть все методы и свойства объекта **Range**, пользуйтесь инструментом **Object Browser** и соответствующим данному объекту разделом справочной системы.

### Выделение ячейки или диапазона ячеек

Для выделения ячейки или диапазона ячеек используйте метод **Select**:

### Синтаксис

---

*Object*.Select

*Object* — ссылка на объект типа **Range**, который выделяется.

---

Листинг 10.15 демонстрирует пример использования данного метода.

### Листинг 10.15. Использование метода **Select** для выделения диапазона ячеек

---

```
1: Sub CreateChart()  
2:   'Создается диаграмма на основе данных диапазона Продажа  
3:  
4:   With Workbooks("Отчеты.xlsx").Worksheets("Лист2")  
5:     .Activate  
6:     .Range("Продажа").Select  
7:   End With  
8:  
9:   Charts.Add  
10: End Sub
```

---

Данная процедура создает из выбранного диапазона ячеек новую диаграмму. В строке 5 активизируется лист **Workbooks("Отчеты.XLS ").Worksheets(Лист2)**. В строке 6 на листе выделяется диапазон ячеек с именем **Продажа**. В строке 9 для создания диаграммы выполняется метод **Add** объекта **Charts**.

### Свойства Value и Formula

Если вам необходимо получить содержимое ячейки или ввести данные в диапазон ячеек, VBA предлагает два свойства объекта **Range**: **Value** и **Formula**. Синтаксис для использования этих свойств следующий:

#### Синтаксис

---

*Object.Value*  
*Object.Formula*

*Object* — объектная ссылка на объект типа **Range**, с которым необходимо работать. Для того чтобы получить содержимое ячейки, руководствуйтесь следующими правилами:

- Если вам необходимо значение, содержащееся в ячейке, используйте свойство **Value**. Например, если ячейка **A1** содержит формулу **=2\*2**, то выражение **Range("A1").Value** возвращает значение 4.
  - Если вам необходима формула, содержащаяся в ячейке, используйте свойство **Formula**. Например, если ячейка **A1** содержит формулу **=2\*2**, то выражение **Range("A1").Formula** возвращает текстовую строку **"=2\*2"**.
- 

Для того чтобы ввести данные в ячейку или диапазон ячеек, вы можете применять как **Value**, так и **Formula**. Листинг 10.16 демонстрирует несколько примеров такого использования.

#### Листинг 10.16. Использование методов Value и Formula для ввода данных

---

```
1: Sub CreateCalculator()  
2:   'Выплаты по ссуде в листе с именем Лист3  
3:  
4:   Worksheets("Лист3").Select  
5:  
6:   With Range("A1")  
7:     'введение заголовков:  
8:     .Value = "Расчет выплат по ссуде"  
9:     .Font.Bold = True  
10:    .Font.Italic = True  
11:    .Font.Size = 16  
12:    .Offset(1).Value = "Тариф"  
13:    .Offset(2).Value = "Период"  
14:    .Offset(3).Value = "Количество"  
15:    .Offset(5).Value = "Платеж"  
16:  
17:    'ввод формата чисел и формулы:  
18:    .Offset(1, 1).NumberFormat = "0.00%"  
19:    .Offset(3, 1).NumberFormat = "#,##0.00"; ($#,##0)"  
20:    .Offset(5, 1).NumberFormat = "$#,##0.00"; [Red] ($#,##0.00)"  
21:    .Offset(5, 1).Formula = "=PMT($B$2/12, $B$3*12, $B$4)"  
22:  End With  
23:  
24: End Sub
```

---

Данная процедура помещает на **Лист3** активной книги простой калькулятор для расчета выплат по ссуде. Оператор **With** использует ячейку **A1** как начальную. Свойство **Value** этой ячейки устанавливается содержащим текст "**Расчет выплат по ссуде**" (строка 8); кроме того, устанавливаются некоторые параметры шрифта (строки 9–11). В следующих четырех строках (12–15) используется свойство **Value** ячеек **A2**, **A3**, **A4** и **A6** для ввода заголовков. Обратите внимание на использование метода **Offset**. Код строк 18–20 задает формат ячеек **B2**, **B4** и **B6**. Выполняется данная операция при помощи свойства **NumberFormat** для диапазона ячеек, на который существует ссылка:

```
Object.NumberFormat = FormatString
```

*Object* — является объектной ссылкой на ячейку или диапазон, которые необходимо отформатировать, *FormatString* — текстовая строка, определяющая форматирование.

Наконец, код строки 21 использует свойство **Formula** для введения в ячейку **B6** формулы платежа.

### Определение имени диапазона ячеек

Имена диапазонов ячеек в VBA являются **Name**-объектами. Для того чтобы их задать, необходимо использовать метод **Add** коллекции **Names**, которая обычно является коллекцией имен, определенных в книге. Ниже представлена сокращенная синтаксическая конструкция использования метода **Add** коллекции **Names**. Всего данный метод имеет одиннадцать аргументов — все необязательные. Полный синтаксис метода следует смотреть в справочной системе, а легче всего получить доступ к нужному разделу справочной системы при помощи **Object Browser**.

### Синтаксис

---

```
Names.Add([Name][, RefersTo][, RefersToR1C1])
```

Аргумент *Name* — любое строковое выражение, задающее имя, которое вы хотите использовать для нового именованного диапазона. *RefersTo* и *RefersToR1C1* — аргументы, описывающие диапазон, на который ссылается имя. Аргументы используются следующим образом:

- *RefersTo*: Используйте данный аргумент для ввода диапазона ячеек, описываемого в **A1**-стиле, например: "**=Sales!\$A\$1:\$C\$6**".
  - *RefersToR1C1*: Используйте данный аргумент, когда диапазон ячеек описывается или в стиле **R1C1**, например, "**=Sales!R1C1:R6C3**", или когда описание диапазона является методом или свойством, возвращающим диапазон, например, **Range("A1:C6")** или **Selection**.
- 

Листинг 10.17 демонстрирует применение метода **Add** коллекции **Names**.

### Листинг 10.17. Задание имени диапазона при помощи метода **Add** коллекции **Names**

---

```
1: Sub CreateNameRange()  
2: 'создает именованный диапазон ячеек по координатам,  
3: 'вводимым пользователем  
4:
```

```
5: Const strTaskName = "Имя диапазона"
6:
7: Dim strTopLeft As String
8: Dim strBottomRight As String
9: Dim RngName As String
10: Dim strRefersToR1C1 As String
11:
12: Worksheets("Лист1").Select
13:
14: 'ввести диапазон и имя:
15: strTopLeft = InputBox(prompt:="Верхний левый угол " & _
16:     "в формате R1C1:", _
17:     Title:=strTaskName)
18: strBottomRight = InputBox(prompt:="Нижний правый угол " & _
19:     "в формате R1C1:", _
20:     Title:=strTaskName)
21: RngName = InputBox(prompt:="Имя нового диапазона:", _
22:     Title:=strTaskName)
23:
24: 'создать имя диапазона ячеек:
25: strRefersToR1C1 = "=Лист1!" & strTopLeft & ":" & strBottomRight
26: Names.Add Name:=RngName, RefersToR1C1:=strRefersToR1C1
27:
28: 'выделить новый диапазон:
29: Range(RngName).Select
30:
31: End Sub
```

Данная процедура не очень сложна — она содержит всего лишь несколько операторов **InputBox** для того, чтобы получить координаты диапазона ячеек (верхний левый угол и нижний правый) и его имя. Затем, используя предложенные координаты и имя, процедура создает именованный диапазон ячеек.

### Как вырезать, скопировать и очистить данные

Если ваша процедура должна выполнять некоторые основные, наиболее часто используемые операции редактирования листа, то это можно делать методами **Cut**, **Copy** и **Clear**. Синтаксис первых двух из них следующий:

### Синтаксис

```
Object.Cut([Destination])
Object.Copy([Destination])
```

**Object** — объектная ссылка на объект типа **Range**, который необходимо вырезать или скопировать. **Destination** — ячейка или диапазон, в который нужно поместить содержимое вырезанного или скопированного диапазона. Если вы опустите аргумент **Destination**, данные будут вырезаться или копироваться в буфер обмена Windows.

Для очистки диапазона ячеек вы можете использовать метод **Cut** с аргументом назначения или без него или воспользоваться любым из следующих методов:

**Синтаксис**


---

```
Object.Clear
Object.ClearContents
Object.ClearFormats
Object.ClearNotes
```

В каждом случае *Object* является объектной ссылкой на диапазон ячеек, который следует очистить. Метод **Clear** очищает все — содержимое, форматирование и примечания. **ClearContents** очищает содержимое *Object*, **ClearFormats** — форматирование *Object*, **ClearNotes** удаляет из *Object* примечания.

---

Листинг 10.18 демонстрирует пример применения методов **Cut**, **Copy** и **Clear**.

**Листинг 10.18. Использование методов Cut, Copy и Clear**


---

```
1: Sub CopyToTemp()
2:   'Копирует данные во временный лист
3:
4:   Dim LastCell As Range
5:   Dim oldSheet As String
6:
7:   Application.ScreenUpdating = False
8:
9:   oldSheet = ActiveSheet.Name           'имя текущего листа
10:  Worksheets.Add                        'создается временный лист
11:  ActiveSheet.Name = "Временный"
12:  Worksheets("Лист1").Select             'выбирается Лист1
13:
14:  Set LastCell = Range("A1").SpecialCells(xlLastCell)
15:  With Worksheets("Временный")
16:    .Cells.Clear
17:    Range("A1", LastCell).Copy Destination:=.Range("A1")
18:  End With
19:
20:  Sheets(oldSheet).Select                'восстанавливается исходный лист
21:  Application.ScreenUpdating = True
22:
23: End Sub
24:
25:
26: Sub RestoreFromTemp()
27:   'Восстанавливаются данные из временного листа
28:
29:   Dim LastCell As Range
30:   Dim oldSheet As String
31:
32:   Application.ScreenUpdating = False
33:   oldSheet = ActiveSheet.Name
34:
35:   Worksheets("Лист1").Select
36:   With Worksheets("Временный")
37:     Set LastCell = .Range("A1").SpecialCells(xlLastCell)
38:     .Range("A1", LastCell).Cut Destination:=.Range("A1")
39:   End With
```

```
40:
41:     Sheets(oldSheet).Select
42:     Application.ScreenUpdating = True
43:
44: End Sub
```

---

Код состоит из двух процедур: **CopyToTemp** (строки 1–23) и **RestoreFromTemp** (строки 26–44). Процедура **CopyToTemp** добавляет лист с именем **Временный** и выбирает лист с именем **Лист1**. Далее она копирует все данные из активного листа и сохраняет их в листе **Временный**. Для возврата последней ячейки активного листа и сохранения объектной ссылки результирующего объекта **Range** в переменной **LastCell** данная процедура использует метод **SpecialCells** с аргументом **xlLastCell** (строка 14). Затем с использованием оператора **With** над листом **Временный** выполняются два действия: код строки 16 использует метод **Clear** для очистки всего листа (метод **Cells** без аргументов возвращает все ячейки листа), код строки 17 использует метод **Copy** для копирования диапазона, заданного с помощью **A1** (верхний левый угол) и **LastCell** (нижний правый угол); «получателем» является ячейка **A1** листа **Временный**.

Вторая процедура, **RestoreFromTemp**, выполняет обратные действия. На этот раз **LastCell** устанавливается на последнюю ячейку листа **Временный** (строка 37). Диапазон от **A1** до **LastCell** вырезается и помещается в ячейку **A1** активного листа (строка 38). Активным является лист **Лист1**, выбранный в строке 35.

Для вставки данных из буфера обмена в лист используйте с объектами **Worksheet** или **Range** методы **Paste** или **PasteSpecial**. Синтаксис метода **Paste** следующий:

#### Синтаксис

---

*Object.Paste ([Destination], [Link])*

*Object* — любая допустимая объектная ссылка на лист. Необязательный аргумент *Destination* — любой объект диапазона ячеек. Данные вставляются, начиная с верхнего левого угла диапазона. Если вы опустите аргумент *Destination*, данные вставляются, начиная с выбранной в данный момент ячейки. Использование аргумента *Destination* аналогично использованию команды **Правка | Вставить** (Edit | Paste). Если необязательный аргумент *Link* равен значению **True**, то должна быть задана ссылка на источник данных. Использование аргумента *Link* аналогично использованию команды **Правка | Специальная вставка** (Edit | Paste Special) с установленным флажком **Связать** (Link). Одновременное использование аргументов *Destination* и *Link* в одном операторе недопустимо.

---

## Работа с объектами Word

До сих пор в данной главе обсуждалась работа только с объектами, доступными в Excel VBA. Word также создает многочисленные объекты, доступные VBA. В этой части главы вы узнаете, как управлять большинством наиболее важных и распространенных объектов Word. Во-первых, вы узнаете об объектах **Document** и **Template**, а затем о том, как управлять информацией, содержащейся в объекте **Document**.

## Работа с объектами Document

В Word используется формат документа, в котором объекты и события полностью доступны VBA, что дает возможность управлять документом и его объектами при помощи VBA-кода. Такой формат документа Word позволяет выполнять наиболее распространенные задачи обработки текста быстрее и легче. В следующих нескольких разделах рассмотрены наиболее часто используемые объекты, методы и свойства Word.

### Доступ к объекту Document

В Word каждый документ является объектом типа **Document**, а все открытые документы текущей сессии Word образуют коллекцию **Documents**. Для ссылок на конкретный документ следует использовать коллекцию **Documents** объекта **Application**.

### Синтаксис

---

**Documents** (*Index*)

*Index* может быть:

- Численным выражением, представляющим документ, который необходимо использовать. Число 1 означает первый документ, открытый в ходе текущей рабочей сессии, 2 — второй открытый документ и так далее.
- Строковым выражением, представляющим имя открытого документа, который нужно использовать.

Наиболее распространенным и в большинстве случаев наиболее удобным способом использования *Index* является текстовая строка.

---

Листинг 10.19 демонстрирует пример использования коллекции **Documents**.

### Листинг 10.19. Использование коллекции **Documents** для возвращения документа

---

```
1 Sub SetDocumentActivate()  
2 'Делает (открытый) документ активным  
3  
4 Documents("H:\Документ10_1.docm").Activate  
5 MsgBox "H:\Документ10_1.docm становится активным!"  
6  
7 End Sub
```

---

Выражение в строке 4, **Documents("C:\F.DOC")**, возвращает объектную ссылку на документ **C:\F.DOC**. Метод **Activate** делает указанный документ активным. Оператор **MsgBox** в строке 5 выводит об этом сообщение. (Документ **C:\F.DOC** в момент выполнения процедуры должен быть открыт, в противном случае вы получите сообщение о runtime-ошибке.)

Для ссылок на активный документ нужно применять **Application**-свойство **ActiveDocument**. Свойство **ThisDocument** следует использовать для ссылок на документ, содержащий выполняемую в данный момент процедуру, особенно если вы намереваетесь преобразовать свою программу в *Word-надстройку* (*add-in*)<sup>1</sup>.

### Как открыть документ

Если необходимый вам документ не открыт, для загрузки его в память используйте метод **Open**:

### Синтаксис

---

```
Documents.Open (Filename)
```

Аргумент *Filename* является текстовой строкой, представляющей полный путь к документу: логический диск, папка и имя файла. Если вы не задали диск или имя папки, Word будет искать документ на текущем диске или в текущей папке.

---

Листинг 10.20 демонстрирует действие метода **Open**.

### Листинг 10.20. Использование метода Open для открытия документа

---

```
1: Sub OpenTest()  
2:   'Предлагает открыть документ и затем его открывает  
3:  
4:   Dim DocumentName As String  
5:  
6:   DocumentName = _  
7:     InputBox("Введите полный путь к документу")  
8:   If DocumentName <> "" Then  
9:     Documents.Open FileName:=DocumentName  
10:  End If  
11:  
12: End Sub
```

---

В данной процедуре переменная **DocumentName** объявляется как **String** (строка 4), а функция **InputBox** предлагает пользователю ввести имя документа (строки 6 и 7). Оператор **If...Then** (в строке 8) проверяет, не отменил ли пользователь диалог ввода. Если — нет (другими словами, если переменная **DocumentName** не пуста), метод **Open** использует ее для того, чтобы открыть заданный документ (строка 9).

Чтобы избежать ошибок при открытии существующего на диске файла (пользователь вряд ли введет в диалоговом окне функции **InputBox** правильное имя файла), следует использовать встроенное в Word диалоговое окно **Open**. Вы можете использовать встроенное в Word диалоговое окно **Open** при помощи коллекции **Dialogs**, как показано ниже (**Dialogs** содержит все диалоговые окна, встроенные в Word):

```
Application.Dialogs(wdDialogFileOpen).Show
```

---

<sup>1</sup> До появления локализованной версии Office 97 использовался термин «надстройка», в Office 97 появился термин «дополнение», тем не менее Microsoft Press пользуется привычным русским пользователям термином «надстройка».



### Как создать новый документ

Если вашей процедуре необходимо создать новый документ, используйте метод **Add** коллекции **Documents**, имеющий следующий синтаксис:

#### Синтаксис

---

```
Documents.Add([Template], [NewTemplate])
```

Оба аргумента метода **Documents.Add** не являются обязательными. *Template* представляет строковое выражение, задающее имя шаблона документа, на котором должен основываться новый документ. Аргумент *Template* наряду с именем файла может включать имя диска и путь к папке. Включать полный путь необходимо, если шаблон документа расположен в папке, отличной от той, в которой по умолчанию хранятся шаблоны Word. Если вы опустите аргумент *Template*, Word создаст новый документ, основываясь на шаблоне *Normal.dot*.

Аргумент *NewTemplate* может быть любым выражением типа **Boolean**. Если он равен значению **True**, Word создает новый документ в виде шаблона. По умолчанию значение этого аргумента равно значению **False**.

---

Листинг 10.21 демонстрирует пример процедуры, создающей новый документ.

#### Листинг 10.21. Использование метода **Add** для создания нового документа

---

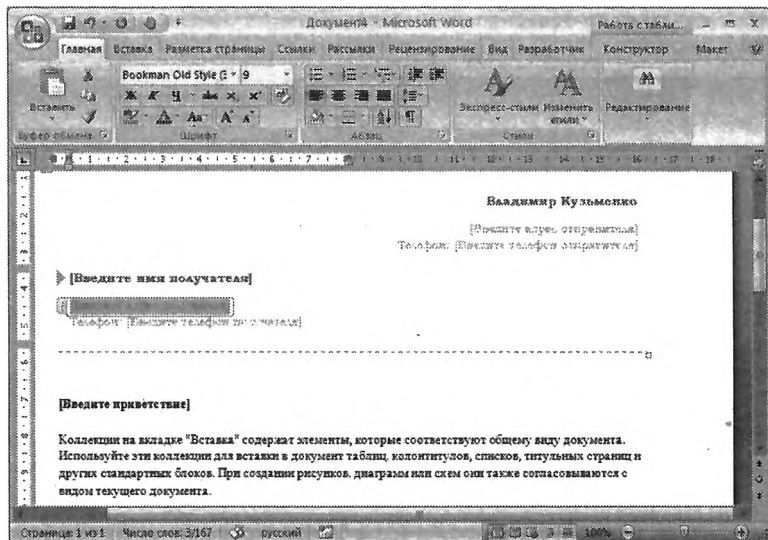
```
1: Sub CreateNewLetter()  
2:   'Создает документ на базе шаблона Professional Letter  
3:  
4:   Dim dPath As String  
5:  
6:   dPath = "G:\Program Files\Microsoft Office\Templates\1049\  
7:     Documents.Add Template:=dPath & "OriginLetter"  
8:   With ActiveDocument  
9:     .BuiltInDocumentProperties("Title") = _  
10:       "Письмо из России"  
11:     .BuiltInDocumentProperties("Subject") = _  
12:       Str(Date)  
13:     .BuiltInDocumentProperties("Author") = _  
14:       Application.UserName  
15:   End With  
16:  
17: End Sub
```

---

Код листинга 10.21 создает новый документ на основании шаблона **OriginLetter**, поставляемого вместе с Word и Microsoft Office. Любой новый документ автоматически становится активным. В строках с 8 по 15 используется объект **ActiveDocument** для добавления некоторой общей информации, относящейся к новому документу. Свойства документа **Title**, **Subject** и **Author** хранятся в коллекции **BuiltInDocumentProperties** объекта **Document**. В результате на экране появляется заготовка письма (рис. 10.6). Свойства активного документа, устанавливаемые в строках кода 8–15, можно увидеть в диалоговом окне **Свойства**.

**Рис. 10.6**

В результате работы кода процедуры CreateNewLetter на экране появляется заготовка письма



### Как сделать документ активным

Если ваша VBA-программа хранит несколько документов, открытых одновременно, вам может понадобиться переключаться с одного документа на другой. Для того чтобы сделать какой-либо из открытых документов активным, используйте метод **Activate**:

### Синтаксис

*Object*.Activate

*Object* представляет собой ссылку на документ, который следует сделать активным.

Листинг 10.22 представляет пример процедуры, использующей метод **Activate**.

### Листинг 10.22. Использование метода **Activate** для переключения к документу

```

1: Sub ActivateTest()
2:   'Активизирует два документа без изменения вывода на экран
3:
4:   Dim SaveDoc As String
5:
6:   SaveDoc = ActiveDocument.Name
7:
8:   Application.ScreenUpdating = False 'не обновлять экран
9:   Documents("TestData.XML").Activate
10:
11:   'действие, выполненное над TestData.docm
12:   With Selection
13:     .TypeText Text:="Документ данных установлен активным"
14:     .TypeParagraph

```

```
15:      End With
16:
17:      Documents(SaveDoc).Activate
18:      Application.ScreenUpdating = True 'обновлять экран
19:
20: End Sub
```

---

Процедура листинга 10.22 демонстрирует два принципа хорошего стиля программирования: скрывать от пользователя промежуточные операции и всегда «возвращать пользователя» в то место, из которого он запустил программу. Назначение процедуры листинга 10.22 — переключиться из текущего документа в другой документ (TestData.xml), произвести в нем (программно) определенные изменения и затем вернуться к исходному документу. В первую очередь, имя текущего активного документа сохраняется в строковой переменной **SaveDoc** (в строке 6). Далее, чтобы избавить пользователя от наблюдения за подробностями выполнения процедуры, в строке 8 свойство **ScreenUpdating** устанавливается в значение **False**. В строке 9 активизируется документ **TestData.doc**. Строки с 12 по 15 представляют действия, выполняемые над документом **TestData.docm**. Данный фрагмент программы просто добавляет в документ некоторый текст. Чтобы «вернуть пользователя» обратно в исходный документ, в строке 17 документ вновь устанавливается активным. А в строке 18 свойство **ScreenUpdating** вновь получает значение **True**.

Не следует без особой необходимости активизировать документ. Как правило, вы можете выполнить изменения или получить информацию о документе, просто ссылаясь на соответствующий объект **Document**. Например, чтобы выяснить, какой шаблон присоединен (если вообще какой-нибудь присоединен) к **TestData.XML**, вы можете использовать следующий оператор (не делая документ активным):

```
DocTemplate = Document("TestData.XML").AttachedTemplate
```

### Как сохранить документ

Если ваш VBA-код изменяет содержимое документа, вы должны быть уверены, что дадите пользователю возможность сохранить эти изменения, или обеспечить их сохранение с помощью вашей программы. Хотя вы можете включить встроенную в Word команду **Save** в меню и панели инструментов, вы, возможно, захотите сохранять документ под управлением процедуры, особенно если нужно не дать пользователю возможность отказаться от изменений, произведенных вашим VBA-кодом.

Для сохранения документа используйте метод **Save** объекта **Document**:

### Синтаксис

---

*Object*.Save

*Object* представляет ссылку на открытый объект типа **Document**, который необходимо сохранить.

---

Можно воспользоваться и родственным методом **SaveAs**. Данный метод сохраняет документ под новым именем и изменяет имя документа. (Метод имеет

шестнадцать необязательных аргументов, здесь для простоты указаны только два из них. В справочной системе вы можете найти полный синтаксис этого метода.)

## Синтаксис

```
Object.SaveAs([FileName], [FileFormat])
```

*Object* — любая допустимая ссылка на открытый документ типа **Document**. Необязательный аргумент *FileName* — строковое выражение, указывающее новое имя, под которым должен быть сохранен документ. *FileName* может включать полный путь (включая диск и папку). Если опустить полный путь, документ сохраняется в текущей папке на текущем диске. Если вы полностью опустите аргумент *FileName*, документ сохранится под текущим именем, если документ с подобным именем уже существует, он будет переписан без предупреждения.

Необязательный аргумент *FileFormat* определяет формат, в котором будет сохранен документ. Аргумент *FileFormat* может быть любой из встроенных констант (определенных в классе **WdSaveFormat**):

<b>wdFormatDocument</b>	Документ сохраняется как «обычный Microsoft Word-документ» (используется по умолчанию).
<b>wdFormatDOSText</b>	Документ сохраняется как неформатированный текст. Преобразует символы (маркеры) окончания раздела, страницы и новой строки в маркеры абзаца. Используется ANSI-набор символов.
<b>wdFormatDOSTextLineBreaks</b>	Документ сохраняется как неформатированный текст. Преобразует символы (маркеры) окончания раздела, страницы и новой строки в маркеры абзаца. Обычно используется для преобразования текста с целью отправки электронной почтой.
<b>wdFormatEncodedText</b>	Документ сохраняется как кодированный текстовый файл (для указания кодовой страницы используется аргумент <i>Encoding</i> ).
<b>wdFormatFilteredHTML</b>	Текст документа сохраняется посредством HTML-тегов с минимальным форматированием с использованием каскадных таблиц стилей. Результирующий документ может быть просмотрен Web-браузером.
<b>wdFormatHTML</b>	Текст и форматирование сохраняется посредством HTML-тегов, чтобы документ можно было просмотреть Web-браузером.
<b>wdFormatRTF</b>	Все форматирование сохраняется.
<b>wdFormatTemplate</b>	Документ сохраняется как Word-шаблон.
<b>wdFormatText</b>	Текст сохраняется, форматирование — нет. Преобразует символы окончания раздела, страницы и новой строки в маркеры абзаца. Используется ANSI-набор символов. Обычно используется, если целевая программа не может прочитать никакой другой формат.

<b>wdFormatTextLineBreaks</b>	Текст сохраняется, форматирование — нет. Преобразует символы окончания раздела, страницы и новой строки в маркеры абзаца. Обычно используется для преобразования текста с целью отправки электронной почтой.
<b>wdFormatUnicodeText</b>	Документ сохраняется как Unicode-текстовый файл.
<b>wdFormatWebArchive</b>	Текст, рисунки и форматирование сохраняются как однофайловая Web-страница.
<b>WdFormatXML</b>	Текст и форматирование сохраняются с использованием Extensible Markup Language (XML) Word XML-схемы.

Листинг 10.23 демонстрирует пример процедуры, использующей как **Save**, так и **SaveAs**.

#### Листинг 10.23. Применение методов **Save** и **SaveAs**

```

1: Sub CreateTemplate()
2:   'На базе активного документа создается новый шаблон документа
3:
4:   Dim NewName As String
5:
6:   With ActiveDocument
7:     If Not .Saved Then .Save
8:       NewName = Left(.Name, Len(.Name) - 3) & ".dot"
9:       .SaveAs FileName:=NewName, FileFormat:=wdFormatTemplate
10:    End With
11:
12: End Sub

```

Процедура этого листинга сохраняет активный документ, а затем сохраняет копию активного документа как шаблон. Начинается код с объявления переменной, предназначенной для сохранения имени файла нового шаблона. Оператор **With** в строках с 6 по 10 содержит операторы, связанные с объектом **ActiveDocument**.

Оператор **If...Then** (строка 7) проверяет свойство **Saved** документа. Если оно равно значению **False**, документ содержит изменения, которые еще не были сохранены, поэтому процедура вызывает метод **Save**.

Имя файла шаблона создается в строке 8 конкатенацией полного имени документа, за исключением последних трех символов, со стандартным расширением для имени файла шаблона **".dot"**. (Имя документа получается из его свойства **Name**.) Результат используется в качестве аргумента для метода **SaveAs** (строка 9), который сохраняет копию документа в виде файла шаблона. Обратите внимание на использование аргумента **FileFormat**, имеющего значение **wdFormatTemplate** — данное значение аргумента сообщает Word о том, что необходимо сохранить документ как шаблон.

Для проверки того, был ли документ когда-либо сохранен, следует использовать свойство **Path** объекта **Document**. Если **Path** возвращает пустую строку (""), то документ никогда прежде не сохранялся.

### Как закрыть документ

Завершив работу с документом, вы должны закрыть его для освобождения памяти и других системных ресурсов. Закрыть документ можно, используя одну из следующих форм метода **Close**:

### Синтаксис

---

```
Documents.Close  
Object.Close([SaveChanges][, OriginalFormat ] [,  
RouteDocument ])
```

Первая синтаксическая форма просто закрывает все открытые документы. Word предлагает перед закрытием документа (если это необходимо) сохранить сделанные в нем изменения.

Вторая синтаксическая форма закрывает конкретный документ, на который указывает *Object*. Аргумент *SaveChanges* может быть равен одной из перечисленных ниже встроенных констант (определенных в классе **WdSaveOptions**): **wdDoNotSaveChanges** (документ закрывается без сохранения изменений), **wdPromptToSaveChanges** (Word запрашивает, нужно ли сохранять изменения), **wdSaveChanges** (изменения сохраняются). Аргумент *OriginalFormat* определяет формат сохранения для документа и может быть одной из следующих **WdOriginalFormat**-констант: **wdOriginalDocumentFormat**, **wdPromptUser** или **wdWordDocument**. Аргумент *RouteDocument* определяет необходимость передачи документа следующему получателю.

---

В листинге 10.24 приведен код с использованием метода **Close**.

### Листинг 10.24. Применение метода **Close**

---

```
1: Sub CloseAll()  
2: 'Закрывает все документы и предлагает сохранить изменения  
3:  
4:     Const qButtons = vbYesNo + vbQuestion  
5:  
6:     Dim Doc As Document  
7:     Dim Ans As Integer  
8:     Dim MsgPrompt As String  
9:  
10:    For Each Doc In Documents  
11:        If Not (Doc.Name = ThisDocument.Name) Then  
12:            If Not Doc.Saved Then  
13:                MsgPrompt = "Сохранить изменения в " & Doc.Name & "?"  
14:                Ans = MsgBox(prompt:=MsgPrompt, Buttons:=qButtons)  
15:                If Ans = vbYes Then  
16:                    Doc.Close SaveChanges:=wdSaveChanges  
17:                Else  
18:                    Doc.Close SaveChanges:=wdDoNotSaveChanges  
19:                End If  
20:            Else  
21:                Doc.Close
```

```
22:           End If
23:       End If
24:   Next Doc
25:
26: End Sub
```

---

Процедура листинга 10.24 закрывает все открытые документы (*за исключением* документа, содержащего саму процедуру) и спрашивает у пользователя, нужно ли выполнять сохранение внесенных изменений для каждого из документов, имеющих несохраненные изменения. Используйте подобную процедуру вместо метода **Documents.Close**, если необходимо заблокировать возможность отмены операции пользователем (встроенный в Word запрос **Save** имеет кнопку **Cancel**).

После объявления константы и нескольких переменных, процедура запускает цикл **For Each** для обработки всех документов коллекции **Documents**.

Поскольку закрытие документа, содержащего выполняемую в данный момент процедуру, приведет к завершению ее выполнения, в строке 11 находится оператор **If...Then**, проверяющий, не совпадает ли имя обрабатываемого в данный момент документа с именем документа, содержащего процедуру **CloseAll**. Если они совпадают, часть программы, закрывающая документ, пропускается; строки с 12 по 22 выполняются только в том случае, если документ не содержит процедуры **CloseAll** (свойство **ThisDocument** возвращает ссылку на документ, содержащий выполняемую в данный момент программу).

Если документ содержит изменения, которые не были сохранены (строка 12), функция **MsgBox** «спрашивает», нужно ли сохранять изменения. Если пользователь выбирает кнопку **Yes** (то есть в переменную **Ans** возвращается значение константы **vbYes**), процедура запускает метод **Close** с аргументом **SaveChanges**, равным значению константы **wdSaveChanges** (строки 15 и 16). В противном случае процедура закрывает документ с аргументом **SaveChanges**, равным значению константы **wdDoNotSaveChanges** (строка 16). Если документ не содержит несохраненных изменений, процедура выполняет метод **Close** без аргументов (строка 21).

## Работа с объектами **Template**

В Word каждый шаблон документа является объектом типа **Template**, а **Templates** — коллекцией всех открытых шаблонов. Далее рассматривается, как выполнять основные операции с объектами **Template**.

### Как создать шаблон

Коллекция **Templates** не имеет метода **Add**. Поэтому для добавления к ней нового шаблона нужно создать этот шаблон, сначала создав объект **Document**. Вы можете создать новый шаблон документа одним из следующих способов:

- Использовать метод **Documents.Add** с аргументом *NewTemplate*, равным **True**, для одновременного создания нового документа и преобразования его в объект **Template**.
- Использовать метод **SaveAs** объекта **Document** с аргументом *FileFormat*, заданным как **wdFormatTemplate**, для сохранения документа в виде шаблона (в листинге 10.23 это уже использовалось).

### Как загружать и выгружать общие шаблоны

Как вы уже можете знать, единственный способ сделать код в проекте VBA Word доступным всем открытым документам — это сохранить его в шаблоне документа, загруженном как общий. По этой причине вам, возможно, захочется поместить процедуру текущего документа в шаблон, загруженный как общий, делая таким образом процедуры и другие ресурсы данного шаблона доступными всем открытым документам.

Коллекция **Templates** не имеет метода **Open**. Вместо этого следует загружать шаблон документа как общий посредством коллекции **AddIns** объекта **Applications** Word. (**AddIns** — коллекция всех доступных в Word надстроек, независимо от того, были они установлены или нет.) Поскольку они являются общими ресурсами для программ, панелей команд, стилей и других элементов, Word рассматривает общие шаблоны как разновидность надстройки.

Если вы знаете, что ваша программа больше не нуждается в ресурсах конкретного шаблона, то можете выгрузить общий шаблон для освобождения системных ресурсов компьютера.

Чтобы загрузить шаблон как общий, используйте метод **Add** коллекции **AddIns** в следующей синтаксической форме:

### Синтаксис

---

```
Object.Add(FileName [, Install])
```

*Object* представляет ссылку на объект **AddIns**. Аргумент *FileName* является обязательным. Это строковое выражение, содержащее имя файла шаблона, включая полный путь.

Аргумент *Install* (необязательный) управляет установкой надстройки: при равенстве значению **True** (значение по умолчанию) надстройка устанавливается, при **False** — добавляется в список надстроек, но не устанавливается.

---

Для того чтобы выгрузить общий шаблон, используйте метод **Delete** коллекции **AddIns** в следующей синтаксической форме:

### Синтаксис

---

```
AddIns(Name).Delete
```

*Name* — строковое выражение, задающее имя шаблона, который нужно выгрузить, или численное выражение, указывающее номер индекса шаблона в коллекции. Как и для большинства других коллекций, следует пользоваться именами шаблонов, чтобы создавать удобочитаемые и надежные программы.

---

Листинг 10.25 содержит две демонстрационные процедуры: одна — загружает шаблон как общий, а другая — выгружает общий шаблон.



**Листинг 10.25. Загрузка и выгрузка общего шаблона**


---

```

1: Sub LoadGlobalTemplate()
2: 'Загружает и определяет общий шаблон
3:
4:     Dim strName As String
5:
6:     strName = InputBox(prompt:="Введите имя шаблона:" & _
7:         vbCr & "(включая полный путь)", _
8:         Title:="Загружается общий шаблон")
9:
10:    If Trim(strName) = "" Then Exit Sub
11:    AddIns.Add FileName:=strName
12:
13: End Sub
14:
15:
16: Sub UnloadGlobalTeplate()
17: 'Выгружает шаблон, заданный как общий
18:
19:     Dim strName As String
20:
21:     strName = InputBox(prompt:="Введите имя шаблона:", _
22:         Title:="Выгрузка общего шаблона")
23:
24:    If Trim(strName) = "" Then Exit Sub
25:    AddIns(strName).Delete
26:
27: End Sub

```

---

**Как присоединить шаблон**

Если вам необходимо задать присоединенный к документу шаблон, используйте свойство **AttachedTemplate** объекта **Document** в следующей синтаксической форме:

**Синтаксис**


---

*Object*.AttachedTemplate = *Template*

Здесь *Object* — любая допустимая ссылка на объект типа **Document**. *Template* может быть как объектной ссылкой на уже открытый **Template**-объект, так и строковым выражением, задающим имя файла шаблона, который необходимо присоединить. Свойство **AttachedTemplate** всегда возвращает объектную ссылку на объект типа **Template**. Если документ не содержит присоединенного шаблона, свойство **AttachedTemplate** возвращает ссылку на **Normal.dot**.

---

Следующий фрагмент программы присоединяет шаблон к активному документу (предполагается, что переменная **TestTemplate** содержит строку, включающую полный путь к файлу шаблона):

```
ActiveDocument.AttachedTemplate = TestMyTemplate
```

Для отключения от активного документа всех шаблонов достаточно использовать следующий код:

```
ActiveDocument.AttachedTemplate = ""
```

## Компоненты объекта Document

Документ в Word представляет собой конструкцию довольно свободной формы: он состоит из символов, слов, предложений и абзацев. Документ может содержать колонтитулы, сноски и прочие элементы. Как следствие, число и тип объектов, используемых для управления содержимым документа, может сначала вызвать замешательство.

Каждый документ состоит из нескольких *областей* (*stories*). Каждая область документа соответствует конкретному разделу документа, такому как колонтитул, сноска основной текст документа, примечания и так далее. Многие методы объекта **Document** позволяют выбрать конкретную область. Многие объекты также обладают свойствами, из которых можно извлечь значение, указывающее, какая область объекта документа в нем содержится. VBA Word предоставляет несколько внутренних констант (определенных в классе **WdStoryType**), которые вы можете использовать для задания конкретной области документа: **wdMainTextStory**, **wdPrimaryFooterStory**, **wdPrimaryHeaderStory** и другие.

Текст внутри области документа может рассматриваться как коллекция символов, слов, предложений и абзацев. Word предоставляет коллекцию объектов для каждого из этих представлений: **Characters**, **Words**, **Sentences** и **Paragraphs**. Ниже дана краткая характеристика этих коллекций:

- **Characters**. Коллекция символов. В зависимости от того, из какого объекта вы получили доступ к коллекции, она может содержать некоторые или все символы области документа. Объект для *одного символа* отсутствует. Каждый элемент коллекции **Characters** является объектом типа **Range**, включающим только один символ. (Объекты **Range** Word будут рассмотрены далее в этом разделе.)
- **Words**. Коллекция слов. В зависимости от того, из какого объекта вы получили доступ к коллекции **Words**, она может содержать некоторые или все слова области документа. В Word отсутствует объект «слово». Каждый элемент коллекции **Words** является объектом типа **Range**, включающим только одно слово. (Слово — это группа символов, отделенная пробелами или знаками пунктуации.)
- **Sentences**. Коллекция предложений. В зависимости от того, из какого объекта вы получили доступ к коллекции **Sentences**, она может содержать некоторые или все предложения области документа. Подобно коллекциям **Characters** и **Words** объект «одно предложение отсутствует». Каждый элемент коллекции **Sentences** является объектом типа **Range** (который будет описан далее в этой главе), включающим единственное предложение.
- **Paragraphs**. Коллекция абзацев. В зависимости от того, **Paragraphs**-коллекция какого именно объекта используется, может содержать некоторые или все абзацы области документа. Каждый элемент коллекции **Paragraphs** является объектом типа **Paragraph**. Вы можете использовать свойства и методы объекта **Paragraph** для изменения стиля, увеличения или уменьшения интерлиньяжа и других, связанных с абзацем задач. Каждый объект типа **Paragraph**, в свою очередь, содержит объект типа **Range**, который содержит текст параграфа.

Представленный обзор коллекций показывает, что доступ к тексту для каждой из этих коллекций, в конечном счете, осуществляется с помощью объекта **Range**. Объект **Range** является одним из основных объектов Word и появляется в виде свойства во многих других объектах Word.

Каждый объект **Range** представляет непрерывную часть области документа, определяемую положениями начального и конечного символов. Объект **Range** может не содержать ни одного, содержать один или много символов и может представлять весь документ или любую его часть. Используя методы и свойства объекта **Range**, можно добавлять текст, форматировать его, добавлять поля или выполнять любые другие действия. Объект **Range** также предоставляет методы, позволяющие задавать или изменять диапазон символов, на который ссылается конкретный объект **Range**, и определять, в каком текстовом блоке документа находится диапазон.

Другим основным объектом Word является объект **Selection**. Объект **Selection** представляет курсор вставки в окне, отображающем определенную часть документа. В каждом окне документа может находиться только один объект **Selection**, и только один объект **Selection** может быть активным в любой конкретный момент времени. Объект **Selection** используется для того, чтобы добавлять текст в то место, на которое указывает курсор вставки, применять форматирование, выделять текст для копирования или вырезания или любой другой задачи, которую можно выполнить интерактивно с помощью курсора вставки, вводя текст, щелкая, или перетаскивая его. Объект **Selection** может быть как непрерывной областью документа, так и сжатым до курсора вставки. В следующем разделе обсуждается, как с помощью свойств и методов объекта **Selection** выполнять основные операции с документом.

В дополнение к только что описанным основным компонентам документа каждый объект **Document** также содержит несколько других коллекций, таких как **Fields** (коллекция полей, включая поля для простановки почтовых реквизитов в документе), **TablesOfContents** и другие.

## Как задать диапазон

Прежде чем вы сможете работать с объектом **Range**, следует задать в документе диапазон текста, с которым необходимо работать. Объект **Range** может являться курсором вставки, не содержащим ни одного символа, или быть непрерывной областью документа. Вы можете создать в документе более одного объекта **Range**.

### Как получить ссылку на объект

Существует несколько различных способов сослаться на конкретный диапазон документа. Можно задать диапазон непосредственно или получить его при помощи одной из коллекций документа: **Paragraphs**, **Sentences** или **Words**.

Для того чтобы задать диапазон в области основного текста документа непосредственно, воспользуйтесь методом **Range**, применив следующую синтаксическую конструкцию:

**Синтаксис**

---

*Object*.Range(*Start*, *End*)

Здесь *Object* — любая допустимая ссылка на объект **Document**. *Start* и *End* — целые значения типа **Long**, задающие положение начального и конечного символов диапазона, соответственно. Метод **Range** возвращает объект **Range**.

---

В следующем фрагменте программы метод **Range** используется для получения диапазона, начинающегося перед первым символом и заканчивающегося после двадцатого символа активного документа (**AnyRange** — объектная переменная):

```
Set AnyRange = ActiveDocument.Range(Start:=0, End:=20)
```

Ссылку на диапазон можно получить и с помощью свойства **Range** любой из коллекций диапазона, упоминавшихся ранее: **Paragraphs**, **Sentences** или **Words**. Представленные ниже примеры выражений демонстрируют, как это сделать:

```
ActiveDocument.Sentences(3)  
ActiveDocument.Paragraphs(2)  
ActiveDocument.Words(10)
```

Первое выражение возвращает объект **Range**, ссылающийся на третье предложение активного документа. Второе — возвращает объект **Range**, ссылающийся на второй абзац активного документа, а третье выражение возвращает объект **Range**, ссылающийся на десятое слово активного документа.

Следующие два примера выражений немного сложнее. Первое из помещенных ниже выражений возвращает объект **Range**, ссылающийся на первое слово второго абзаца активного документа. Коллекция **Paragraphs** используется для возвращения ссылки на второй абзац документа, а коллекция **Word**, связанная с диапазоном объекта **Paragraph**, — для возвращения ссылки на первое слово абзаца. Второе выражение возвращает объект типа **Range**, ссылающийся на первое слово первого предложения активного документа. Коллекция **Sentences** используется для возвращения ссылки на первое предложение документа, а коллекция **Words** данного предложения — для возвращения объекта **Range**, ссылающегося на первое слово в предложении.

```
ActiveDocument.Paragraphs(2).Range.Words(1)  
ActiveDocument.Sentences(1).Words(1)
```

Код листинга 10.26 демонстрирует пример процедуры, использующей различные диапазоны.

**Листинг 10.26. Использование объектов Range**

---

```
1: Sub TestRange()  
2:   'В активном документе первое слово каждого  
3:   'абзаца, имеющего стиль "Основной" и длину не менее  
4:   'трех слов, выделяется курсивом  
5:  
6:   Dim mParagraph As Paragraph  
7:  
8:   For Each mParagraph In ActiveDocument.Paragraphs  
9:     With mParagraph
```

```

10:         If .Style = "Основной" Then
11:             If .Range.Words.Count >= 3 Then
12:                 .Range.Words(1).Italic = True
13:             End If
14:         End If
15:     End With
16: Next mParagraph
17:
18: End Sub

```

Процедура **TestRange** выделяет курсивом первое слово каждого абзаца в главной области документа, если он содержит три или более слов и если стиль данного абзаца — **Основной**.

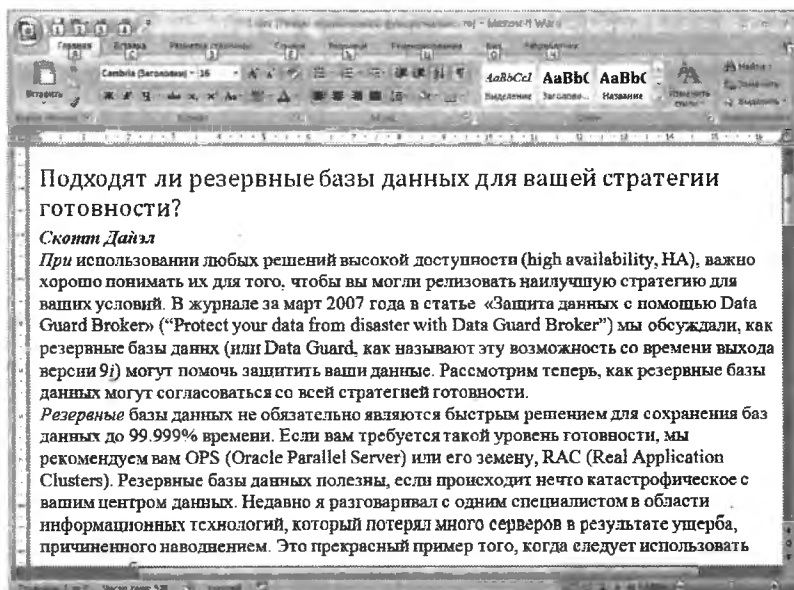
В строке 8 начинается цикл **For Each**, который перебирает все абзацы коллекции **Paragraphs** документа. Строка 9 начинается с оператора **With**, включающего несколько операторов, ссылающихся на свойства и методы текущего объекта **Paragraph**. В строке 10 используется оператор **If** для определения того, действительно ли абзац имеет стиль **Основной текст**. Текущий стиль абзаца определяется при помощи свойства **Style**.

В строке 11 используется оператор **If** для определения того, содержит ли абзац более двух слов. Поскольку символ конца абзаца считается словом, для любого абзаца, содержащего как минимум два слова, не считая символа конца абзаца, выражение в строке 11 будет равно значению **True**.

В строке 12 первое слово абзаца выделяется курсивом (рис. 10.7). Сначала при помощи свойства **Range** объекта **Paragraph** возвращается диапазон, на который ссылается данный абзац. В свою очередь, коллекция **Words** используется для того, чтобы вернуть объект типа **Range**, ссылающийся на первое слово в диапазоне абзаца. Чтобы выделить слово курсивом, свойству **Italic** объекта **Range** присваивается значение **True**.

**Рис. 10.7**

В результате работы кода процедуры **TestRange** в активном документе первое слово каждого абзаца, имеющего стиль «Основной текст» и длину не менее трех слов, выделяется курсивом



### Как выполнить ссылку на диапазон

Размер области, на которую ссылается конкретный объект **Range**, можно изменять. Например, вы можете сначала получить диапазон, который ссылается на один абзац, а затем изменить область данного диапазона, включив в него два следующих абзаца. Область документа, на которую ссылается объект **Range**, можно расширить или уменьшить, и даже сжать диапазон до курсора вставки.

Чтобы изменить область, на которую ссылается диапазон, используйте следующую синтаксическую конструкцию для метода **SetRange**:

### Синтаксис

---

```
Object.SetRange(Start, End)
```

Здесь *Object* — любая допустимая ссылка на объект **Range** или **Selection**. Оба аргумента *Start* и *End* являются обязательными, каждый из них является целым числом типа **Long**, представляющим новые позиции начального и конечного символов диапазона, соответственно. Следующий фрагмент программы демонстрирует, как объектная переменная **MyRange** сначала устанавливается для ссылки на первое слово активного документа, а затем переопределяется и включает уже первые два слова:

```
Set MyRange = ActiveDocument.Words(1)
AnyRange.SetRange Start:=AnyRangeStart, _
    End:= ActiveDocument.Words(2).End
```

Вместо переопределения диапазона напрямую вы можете расширить или уменьшить диапазон до заданного размера или «свернуть» его полностью.

---

Метод **MoveStart** изменяет положение начального символа диапазона, в то время как метод **MoveEnd** изменяет положение конечного символа диапазона. Для увеличения или уменьшения области, на которую ссылается диапазон, вы можете перемещать конечную точку диапазона, используя методы **MoveEnd** или **MoveStart** при помощи следующих синтаксических форм:

### Синтаксис

---

```
Object.MoveEnd([Unit], [Count])
Object.MoveStart([Unit], [Count])
```

В данном синтаксическом выражении *Object* — любая допустимая ссылка на объект **Range**. Как для метода **MoveEnd**, так и для метода **MoveStart** аргументы *Unit* и *Count* являются необязательными. Аргумент *Unit* задает элемент, для которого будет изменяться начало или конец диапазона. *Unit* может быть одной из констант **WdUnits**, перечисленных в таблице 10.1. Если вы опустите аргумент *Unit*, методы **MoveStart** и **MoveEnd** по умолчанию будут использовать **wdCharacter**.

Аргумент *Count* — число элементов (units), на которое должна быть сдвинута конечная точка. Если вы используете для аргумента *Count* положительное число, конечная точка перемещается в документе вперед, если вы используете для аргумента *Count* отрицательное число, конечная точка перемещается назад.

---

Ниже приведены операторы, демонстрирующие применение методов **MoveStart** и **MoveEnd** (**AnyRange** является объектной переменной, содержащей ссылку на объект **Range**):

```
AnyRange.MoveStart Unit:=wdParagraph, Count:= -1
AnyRange.MoveEnd Unit:=wdWord, Count:=-2
AnyRange.MoveEnd Count:=6
AnyRange.MoveStart Count:=8
```

**Таблица 10.1 Встроенные константы WdUnits**

Константа	Описание
wdCell	Ячейка таблицы.
wdCharacter	Символ, включая скрытые и непечатаемые символы.
wdColumn	Столбец таблицы.
wdLine	Строка текста. Вы можете использовать данную константу только для аргумента <i>Unit</i> , если используемый метод применяется к объекту <b>Selection</b> .
wdParagraph	Абзац. Абзац ограничивается символом конец абзаца (§). Символ конца абзаца не отображается на экране пока вы не щелкните кнопку <b>Показать все символы</b> (Show All) панели инструментов Word.
wdRow	Строка таблицы.
wdSection	Раздел документа.
wdSentence	Предложение. Предложение ограничивается знаками пунктуации, такими как точка (.), знак вопроса (?) и восклицательный знак (!).
wdStory	Область документа.
wdTable	Таблица.
wdWord	Слово. Слово ограничено пробелами и символами пунктуации.

Иногда вам может понадобиться расширить диапазон от его текущего значения до размеров, включающих следующий, больший по величине компонент документа, например, расширить диапазон курсора вставки, чтобы он включал слово, в котором находится курсор вставки.

Чтобы расширить диапазон, применяйте метод **Expand** в следующей синтаксической форме:

### Синтаксис

```
Object.Expand([Unit])
```

Здесь *Object* — представляет любую допустимую ссылку на объект **Range** или **Selection**. Необязательный аргумент *Unit* определяет элемент, до которого расширяется диапазон, и может быть равен одной из констант **WdUnits**, перечисленных в таблице 10.1. Если вы опустите аргумент *Unit*, метод **Expand** использует по умолчанию значение константы **wdWord**.

Ниже представлен фрагмент кода, выделяющий десятое предложение документа и затем расширяющий диапазон для включения в него всего абзаца, в котором находится десятое предложение (**MyRange** — объектная переменная):

```
Set MyRange = ActiveDocument.Sentences(10)
MyRange.Expand Unit:=wdParagraph
```

## Работа с объектом Selection

Для большинства действий, связанных с управлением текстом документа, вы, вероятно, остановитесь на объекте **Selection**. Используя объект **Selection** можно создать VBA-программу, выполняющую с курсором вставки любые операции, которые можно выполнить в Word интерактивно: добавить текст, переместить курсор вставки, выделить текст и другие.

Выполнить ссылку на объект **Selection** очень просто: необходимо только использовать свойство **Selection** объектов **Application**, **Pane** или **Window**. В большинстве случаев используется свойство **Application.Selection**, чтобы получить ссылку на выделенный в данный момент фрагмент в активном документе.

Как уже упоминалось в этой главе, в любой момент времени может быть активным только один объект **Selection** и в любом окне, отображающем документ, может находиться только один объект **Selection**. Это вполне очевидно, поскольку объект **Selection**, по существу, является курсором вставки, который отображается на экране при интерактивной работе в Word.

Вы можете связать объект **Selection** с любым диапазоном, используя метод **Select** объекта **Range**:

```
MyRange.Select
```

Данный оператор выделяет область документа, на которую ссылается объектная переменная **MyRange**. Объект **Selection** теперь также совпадает с диапазоном, на который ссылается **MyRange**.

## Как переместить или «свернуть» объекты Selection и Range

Вы уже знаете, как определить или переопределить диапазоны и как связать текущий выделенный фрагмент с конкретным диапазоном. Объекты **Range** и **Selection** имеют несколько методов, позволяющих перемещать диапазон или выделенный фрагмент. Чаще всего эти методы используются для перемещения объекта **Selection** перед добавлением и выделением текста. В действительности, существует пара методов, которые доступны исключительно в объекте **Selection**, они не имеют аналогов в объекте **Range**.

Чтобы поместить объект **Selection** в начале или в конце текущей строки текста, столбца таблицы, строки таблицы или области документа и при желании выделить данную строку текста, столбец или строку таблицы или же область, можно использовать методы **HomeKey** или **EndKey** в следующей синтаксической форме:



## Синтаксис

```
Object.HomeKey([Unit], [Extend])
Object.EndKey([Unit], [Extend])
```

В обоих случаях *Object* представляет собой допустимую ссылку на объект **Selection**. В зависимости от конкретного значения используемого аргумента метод **HomeKey** оказывает на выделенный фрагмент такое же действие, как и нажатие клавиш **Home**, **Shift+Home**, **Ctrl+Home** или **Ctrl+Shift+Home**. Аналогично метод **EndKey** оказывает на выделенный фрагмент такое же действие, что и нажатие в различных комбинациях клавиши **End**.

Необязательный аргумент *Unit* задает элемент, в конце которого должен помещаться выделенный фрагмент (в конце строки, столбца и т.д.). *Unit* может быть значением одной из следующих констант **WdUnits**: **wdStory**, **wdLine**, **wdColumn** или **wdRow**. Если аргумент *Unit* опустить, то как **HomeKey**, так и **EndKey** используют по умолчанию значение константы **wdLine** (в конце строки).

Необязательный аргумент *Extend* определяет, должен ли выделенный фрагмент быть расширен или должен ли курсор вставки быть перемещен. В качестве *Extend* можно использовать значение одной из встроенных констант **WdMovementType**: **wdMove** или **wdExtend**. Использование **wdMove** приведет к «сворачиванию» выделенного фрагмента до курсора вставки, после чего курсор вставки переместится на начало или конец заданного элемента документа, определяемого аргументом *Unit*. Использование **wdExtend** оказывает такое же действие, как нажатие на клавиши **Home** или **End** при удерживаемой нажатой клавише **Shift**. Использование **wdExtend** приведет к тому, что выделенный фрагмент будет расширен до конца заданного элемента. Если вы опустите аргумент *Extend*, в качестве значения по умолчанию будет использована константа **wdMove**.

В представленном ниже примере каждый из двух операторов использует один из только что описанных методов. Выполнение первого оператора «сворачивает» выделенный фрагмент до курсора вставки и затем перемещает курсор вставки в начало текущей строки. Выполнение второго оператора расширяет выделенный фрагмент до конца текущей строки.

```
Selection.HomeKey
Selection.EndKey Extend:=wbExtend
```

Хотя объекты **Range** и **Selection** имеют много различных методов для выполнения перемещений, возможно, наиболее мощным из них является метод **GoTo**. Метод **GoTo** позволяет помещать объекты **Range** и **Selection** в любое место в пределах текущей области и перемещать **Selection** или **Range** по документу вперед или назад на заданное количество символов, слов или абзацев.

Для перемещения объектов **Selection** или **Range** используйте метод **GoTo** в следующей синтаксической форме:

## Синтаксис

```
Object.GoTo([What], [Which], [Count], [Name])
```

Здесь *Object* — любая допустимая ссылка на объекты **Selection**, **Range** или **Document**. Если *Object* ссылается на **Selection**, то метод **GoTo** возвращает объект **Range**, представляющий новое положение выделенного фрагмента после его «сворачивания» до курсора вставки и перемещения в позицию символа, расположенного непосредственно перед заданным положением. Если *Object* ссылается на **Range** или **Document**, метод **GoTo** просто возвращает объект **Range**, представляющий начальное положение для заданной области.

Все аргументы метода **GoTo** являются необязательными. Аргумент *What* задает тип элемента, в который должен быть перемещен выделенный фрагмент, и может быть любой из встроенных констант **WdGoToItem**. В таблице 10.2 перечислены некоторые из констант **WdGoToItem**. Для того чтобы просмотреть полный список констант, доступных в классе **WdGoToItem**, используйте Object Browser.

Аргумент *Which* указывает, в какой элемент должен быть перемещен выделенный фрагмент. *Which* может быть равен любой из встроенных констант класса **WdGoToDirection**: **wdGoToAbsolute**, **wdGoToFirst**, **wdGoToLast**, **wdGoToNext**, **wdGoToPrevious**, **wdGoToRelative**.

Аргумент *Count* задает число элементов документа, на которое должен быть перемещен выделенный фрагмент. По умолчанию значение аргумента *Count* равно 1. Использование аргумента *Count* подразумевает использование аргумента *What*, указывающего, куда необходимо выполнить перемещение, и аргумента *Which*, задающего, какой тип перемещения необходимо выполнить.

Последний аргумент *Name* определяет имя области, в которую должен быть перемещен выделенный фрагмент. Вы можете использовать аргумент *Name* только в том случае, если аргумент *What* также используется и равен при этом значению одной из следующих констант: **wdGoToBookmark**, **wdGoToComment**, **wdGoToField** или **wdGoToObject**.

В результате выполнения приведенного ниже оператора выделенный фрагмент перемещается на следующую страницу:

```
Selection.GoTo What:=wgGoToPage
```

Результатом выполнения следующего оператора будет перемещение выделенного фрагмента на последнюю страницу документа:

```
Selection.GoTo What:=wgGoToPage, Which:=wdGoToLast
```

В результате выполнения приведенного ниже оператора выделенный фрагмент перемещается на две строки вниз относительно текущего местоположения:

```
Selection.GoTo What:=wgGoToLine, Which:=wdGoToNext, Count:=2
```

Таблица 10.2 Встроенные константы **WdGoToItem**

Константа	Описание
<b>wdGoToBookmark</b>	Закладка.
<b>wdGoToField</b>	Поле, например поле даты или номера страницы.
<b>wdGoToHeading</b>	Заголовок, то есть текст, отформатированный с помощью одного из стилей заголовка: Заголовок1, Заголовок2 и так далее.
<b>wdGoToLine</b>	Строка документа.
<b>wdGoToPage</b>	Страница документа.
<b>wdGoToPercent</b>	Местоположение в документе, выраженное в процентах от длины документа.

В дополнение к изменению положения выделенного фрагмента вы можете «свернуть» выделенный фрагмент (или диапазон) до курсора вставки. Для этого используйте метод **Collapse** в следующей синтаксической форме:

### Синтаксис

---

*Object.Collapse* ([*Direction*])

Здесь *Object* — любая допустимая ссылка на объекты **Selection** или **Range**. Необязательный аргумент *Direction* задает направление, в котором диапазон или выделенный фрагмент должны быть «свернуты». Он может быть равен одной из констант класса **WdCollapseDirection**: **wdCollapseEnd** или **wdCollapseStart**. По умолчанию используется значение константы **wdCollapseStart**.

---

Следующие операторы демонстрируют применение метода **Collapse**:

```
ActiveDocument.Paragraphs(3).Range.Select  
Selection.Collapse Direction:=wdCollapseEnd
```

Первый оператор выделяет третий абзац активного документа. Второй оператор «сворачивает» выделенный фрагмент до курсора вставки, помещенного в конец предыдущего выделенного фрагмента.

### Добавление текста

Основным назначением программ подготовки текста является хранение и обработка текста. Одной из основных задач, выполняемых при интерактивной работе с Word, является ввод текста в документ. Точно также, одна из наиболее распространенных задач, которую вам придется выполнять под управлением VBA-кода, будет заключаться в том, чтобы добавлять текст. Объекты **Selection** и **Range** имеют несколько методов, предоставляющих возможность добавлять текст. В первую очередь рассмотрим методы добавления текста объекта **Selection**, поскольку вам чаще всего придется иметь дело именно с этим объектом.

Простейший способ добавить текст в документ состоит в использовании метода **TypeText** объекта **Selection**, сделать это можно при помощи следующей синтаксической конструкции:

### Синтаксис

---

*Object.TypeText* ([*Text*])

*Object* — любая ссылка на объект **Selection**. Аргумент *Text* является обязательным, он может быть любым строковым выражением, и это именно тот текст, который добавляется в документ. Метод **TypeText** помещает текст в то место, где расположен курсор вставки, как если бы вы ввели его с клавиатуры. Если **Selection** содержит диапазон и свойство **ReplaceSelection** равно значению **True**, то добавляемый текст заменяет выделенный фрагмент. В противном случае текст помещается перед курсором вставки объекта **Selection**. При установке Word свойство **ReplaceSelection** по умолчанию принимает значение **True**.

---

Свойство **ReplaceSelection** соответствует флажку заменять выделенный фрагмент (Typing Replace Selection) на вкладке Правка диалогового Word-окна Параметры, которое можно открыть при помощи команды Сервис | Параметры. Можно также задать это свойство при помощи объекта **Options**.

Если метод **TypeText** добавляет текст так, как будто вы ввели его с клавиатуры, то, очевидно, что объекту **Selection** необходим метод, эквивалентный нажатию клавиши **Enter**, для того, чтобы можно было добавлять новый абзац. Этот метод так и называется **TypeParagraph**. Синтаксическая конструкция этого метода выглядит следующим образом:

### Синтаксис

---

*Object.TypeParagraph*

*Object* — ссылка на объект **Selection**. Метод **TypeParagraph** помещает в документ новый абзац аналогично тому, как это происходит при нажатии на клавишу **Enter**. Если объект **Selection** — диапазон, содержимое диапазона заменяется новым абзацем. Если **Selection** — курсор вставки, добавляется новый абзац, а курсор вставки перемещается в документе вперед.

---

Чтобы случайно не заменить выделенный фрагмент, используйте метод **Collapse** для «сворачивания» выделенного фрагмента в курсор вставки.

Возможно, вам понадобится добавлять текст в диапазон, который не является выделенным. В этом случае следует использовать методы **InsertAfter**, **InsertBefore**, **InsertParagraphAfter** и **InsertParagraphBefore**. Все эти методы применимы как к объектам **Selection**, так и к объектам **Range**.

Для добавления текста в начало или конец диапазона используйте методы **InsertBefore** или **InsertAfter** со следующим синтаксисом:

### Синтаксис

---

*Object.InsertBefore(Text)*

*Object.InsertAfter(Text)*

Здесь *Object* — ссылка на объект **Selection** или **Range**. *Text* — строковое выражение для текста, который необходимо добавить. Аргумент *Text* обязателен для каждого из методов. Заданный текст помещается в диапазон, и диапазон расширяется, включая в себя добавленный текст. Оправдывая свое имя, метод **InsertBefore** помещает текст в начало диапазона или выделенного фрагмента, а **InsertAfter** помещает текст соответственно в конец диапазона или выделенного фрагмента.

Применение метода **InsertAfter** имеет некоторые особенности, требующие дополнительных пояснений. Если выделенный фрагмент или диапазон включает целый абзац, метод **InsertAfter** вставляет текст как новый абзац. Так происходит потому, что выделенный фрагмент, включающий целый абзац, также включает и символ конца абзаца. Чтобы вставить текст в конец абзаца, необходимо или «свернуть» диапазон до курсора вставки в позиции перед символом конца абзаца, или таким образом изменить конечную точку диапазона, чтобы он содержал на один символ меньше.

---

---

**Замечание**

Для всех методов, добавляющих текст — **TypeText**, **InsertAfter** и **InsertBefore**, — можно использовать функцию **Chr** и встроенные константы VBA (**vbCr**, **vbLf**, **vbCrLf**, **vbTab**) как часть строки аргумента *Text* для добавления символов, которые нельзя набрать на клавиатуре, или символов, которые имеют в VBA специальное значение (например, кавычки).

---

Так же, как метод **TypeText** имеет сопутствующий ему метод **TypeParagraph**, который можно использовать для добавления нового абзаца, методы **InsertBefore** и **InsertAfter** имеют сопутствующие методы для добавления нового абзаца перед областью или после области, на которую ссылается определенный диапазон:

---

**Синтаксис**

*Object*.InsertParagraphBefore  
*Object*.InsertParagraphAfter

В каждой синтаксической конструкции *Object* — ссылка на объект **Selection** или на объект **Range**. Метод **InsertParagraphBefore** добавляет символ конца абзаца перед диапазоном или выделенным фрагментом, в то время как метод **InsertParagraphAfter** добавляет символ конца абзаца в конец диапазона или выделенного фрагмента. Ни тот, ни другой метод не имеет аргументов. Как и в случае с другими методами **Insert...**, диапазон или выделенный фрагмент расширяется для включения добавленного символа конца абзаца.

---

Листинг 10.27 демонстрирует использование методов **InsertBefore**, **InsertAfter** и **InsertParagraphAfter**.

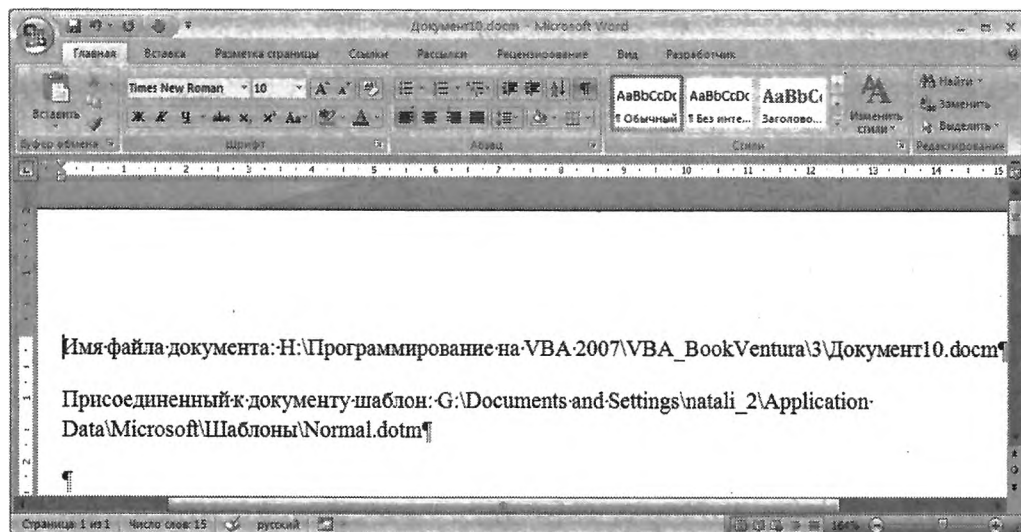
---

**Листинг 10.27. Применение методов InsertAfter, InsertBefore, InsertParagraphAfter**

---

```
1: Sub InsertDocumentInformation()  
2:   'Помещает в начало документа два абзаца  
3:   'Первый абзац содержит имя файла, в котором хранится документ  
4:   'Второй абзац содержит имя файла-шаблона документа  
5:  
6:   Dim aRange As Range  
7:  
8:   Set aRange = ActiveDocument.Range(Start:=0, End:=0)  
9:  
10:  With aRange  
11:    .InsertBefore Text:="Имя файла документа: " & _  
12:      ActiveDocument.FullName  
13:    .InsertParagraphAfter  
14:    .InsertAfter Text:="Присоединенный к документу шаблон: " & _  
15:      ActiveDocument.AttachedTemplate.FullName  
16:    .InsertParagraphAfter  
17:  End With  
18:  
19: End Sub
```

---



**Рис. 10.8.** Код процедуры **InsertDocument-Information** помещает в начало текущего документа два абзаца: первый абзац содержит имя файла, в котором хранится документ, второй абзац содержит имя файла-шаблона документа

## Как вырезать, скопировать, вставить и удалить текст

Вам обязательно понадобится не только вводить, но и вырезать, копировать, вставлять или удалять текст в документе. Для выполнения этих задач, как объекты **Selection**, так и объекты **Range** имеют соответствующие методы.

Чтобы вырезать, скопировать или вставить выделенный фрагмент *в* или *из* буфера обмена Windows, можно использовать один из следующих методов:

### Синтаксис

```
Object.Cut  
Object.Copy  
Object.Paste
```

Для каждого из этих методов *Object* — любая ссылка на объект **Selection** или **Range**. При использовании вами метода **Cut** содержимое диапазона или выделенного фрагмента вырезается из документа и помещается в буфер обмена Windows. Объекты **Selection** или **Range** остаются в документе, «сворачиваясь» до курсора вставки. Метод **Copy** копирует содержимое диапазона или выделенного фрагмента в буфер обмена Windows. Объекты **Selection** или **Range** при этом не изменяются.

Метод **Paste** вставляет содержимое буфера обмена Windows в заданный диапазон или выделенный фрагмент. Если диапазон или выделенный фрагмент не является курсором вставки, то текст, вставляемый из буфера обмена, заменяет содержимое заданного диапазона или выделенного фрагмента. Когда вы используете метод **Paste** с объектом **Range**, диапазон расширяется, включая в себя содержимое вставленного из буфера обмена текста. При использовании же вами метода **Paste** с объектом **Selection** выделенный фрагмент располагается после вставленного из буфера обмена текста; выделенный фрагмент не расширяется.

Листинг 10.28 содержит процедуру, которая при помощи методов **Cut** и **Paste** объекта **Selection** меняет местами два абзаца документа. Код листинга 10.29 при помощи методов **Cut** и **Paste** объекта **Range** меняет местами два слова в документе.

**Листинг 10.28. Использование методов **Cut** и **Paste** совместно с объектом **Selection****

---

```
1: Sub TransposeParagraphs()  
2: 'Меняет местами текущий абзац и абзац, следующий за ним.  
3: 'Используется объект Selection.  
4:  
5: With Selection  
6:     .Expand Unit:=wdParagraph  
7:     .Cut  
8:     .Move Unit:=wdParagraph  
9:     .Paste  
10: End With  
11:  
12: End Sub
```

---

**Листинг 10.29. Использование **Cut** и **Paste** совместно с объектом **Range****

---

```
1: Sub TransposeWords()  
2: 'Меняет местами слово, на которое помещен курсор вставки,  
3: 'и слово, следующее за ним  
4:  
5: Dim MyRange As Range  
6:  
7: If Selection.Type <> wdSelectionIP Then Exit Sub  
8:  
9: Set MyRange = Selection.Range  
10: With MyRange  
11:     .Expand Unit:=wdWord  
12:     .Cut  
13:     .Move Unit:=wdWord  
14:     .Paste  
15: End With  
16: End Sub
```

---

Для удаления из документа текста используйте метод **Delete**. Синтаксическая конструкция для его использования следующая:

**Синтаксис**

---

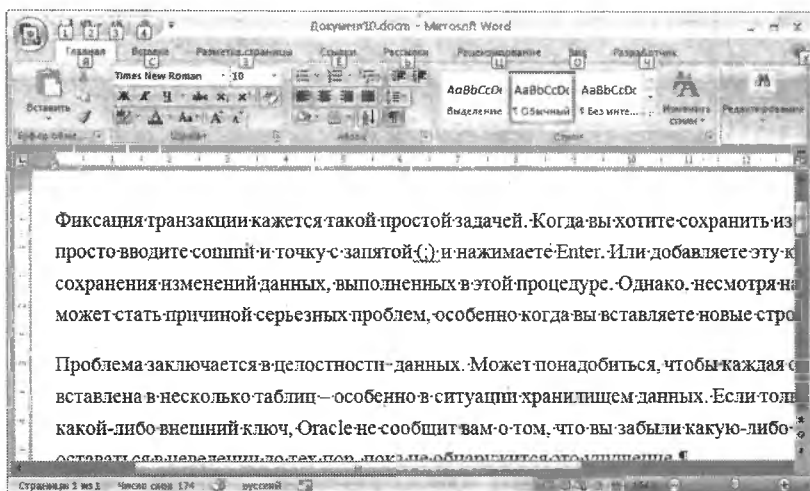
*Object.Delete([Unit], [Count])*

*Object* — любая ссылка на объекты **Selection** или **Range**. Оба аргумента *Unit* и *Count* являются необязательными. Аргумент *Unit* может быть равен значению **wdCharacter** или **wdWord**. По умолчанию значение *Unit* равно **wdCharacter**. Аргумент *Count* задает количество символов или слов (в зависимости от значения аргумента *Unit*), которые должны быть удалены. Если объекты **Range** или **Selection** ссылаются на диапазон текста, метод **Delete** удаляет содержимое данного диапазона.

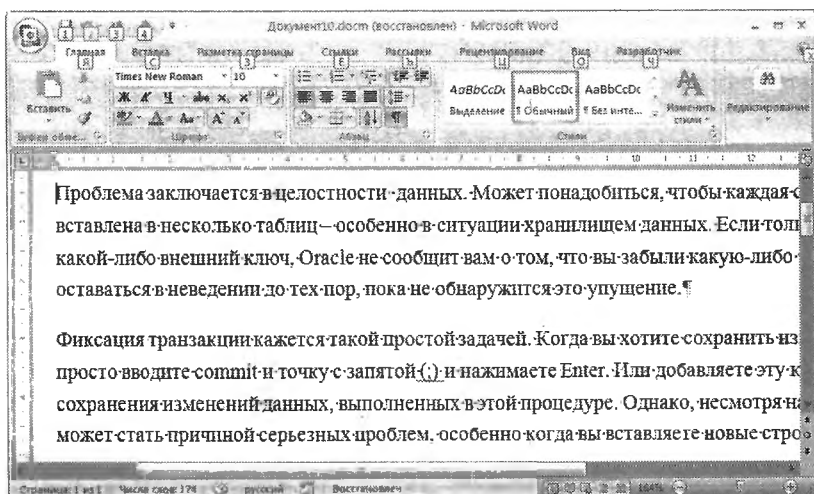
---

Если вы используете **Delete** как функцию, она возвращает значение, указывающее число удаленных элементов текста, или — 0, если операция удаления не удалась.

Для удаления заданного числа символов или слов до или после диапазона или выделенного фрагмента сначала его необходимо «свернуть» до курсора вставки. Для того чтобы удалить элементы текста после курсора вставки, следует использовать в качестве аргумента *Count* положительное число, а чтобы удалить элементы текста до курсора вставки — отрицательное. На рис. 10.9 приведена часть документа до применения процедуры **TransposeParagraphs**, а на рис. 10.10 — после ее применения.



**Рис. 10.9.** Код процедуры **TransposeParagraphs** меняет местами текущий абзац и абзац, следующий за ним (документ до применения процедуры **TransposeParagraphs**)



**Рис. 10.10.** Код процедуры **TransposeParagraphs** меняет местами текущий абзац и абзац, следующий за ним (документ после применения процедуры **TransposeParagraphs**)



# 11

## Отладка VB-кода. Поиск и устранение ошибок

К сожалению, начинающие программировать недостаточно серьезно подходят к вопросам, о которых пойдет речь в этой главе. Даже некоторые опытные программисты утверждают, что в методах отладки нет необходимости — «нужно просто писать код без ошибок». На практике же получается, что отсутствие навыков в вопросах отладки приводит к большим потерям времени.

Как бы вы ни старались, кода без ошибок не бывает. Каждая последняя найденная вами ошибка должна всего лишь придавать вам уверенности в том, что скоро вы найдете еще одну (снова «последнюю») ошибку.

Процесс нахождения *ошибок* (*bugs*) в приложении называется *отладкой* (*debugging*). В этой главе рассматриваются средства отладки Visual Basic и вопросы «управления» ошибками времени исполнения.

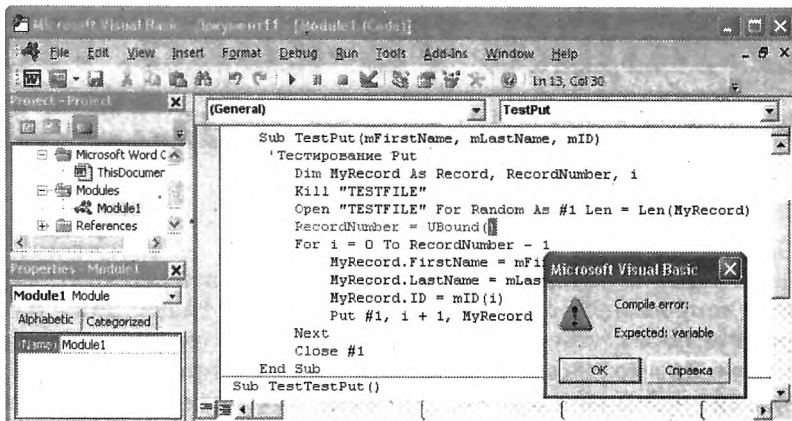
### Типы ошибок

Все ошибки, которые могут возникать при создании и эксплуатации приложения, можно разделить на: *синтаксические ошибки* (*syntax errors*), *ошибки компиляции* (*compile errors*), *ошибки времени исполнения* или *runtime-ошибки* (*runtime errors*). Иногда говорят еще и о логических ошибках, которые связывают с тем, что программа выполняет не то, что хотел разработчик, но для системы программирования это не является ошибкой.

*Синтаксические ошибки* возникают в результате неправильного ввода кода. Это самые простые ошибки: они обнаруживаются редактором кода Visual Basic сразу и являются результатом несоблюдения разработчиком синтаксиса языка или просто невнимательности при вводе кода. Например, можно случайно написать идентификатор для какой-либо переменной, совпадающий с ключевым словом, или неправильно указать аргументы для встроенной функции. К синтаксическим, например, относятся орфографические ошибки при вводе ключевого слова VBA, имени переменной или подпрограммы. Редактор VB «способен» распознавать большинство синтаксических ошибок по мере того, как вы редактируете или вводите текст программы. (VBA выполняет проверку синтаксиса, когда вы перемещаете курсор вставки с вновь отредактированной или набранной строки). Другие синтаксические ошибки могут проявляться как ошибки при трансляции. В общем, все, на что редактор кода Visual Basic (во время ввода кода) реагирует выводом окна сообщения (рис. 11.1), сопровождаемым звуковым сигналом, — это синтаксические ошибки. Обратите внимание на то, что сам редактор VB называет синтаксиче-

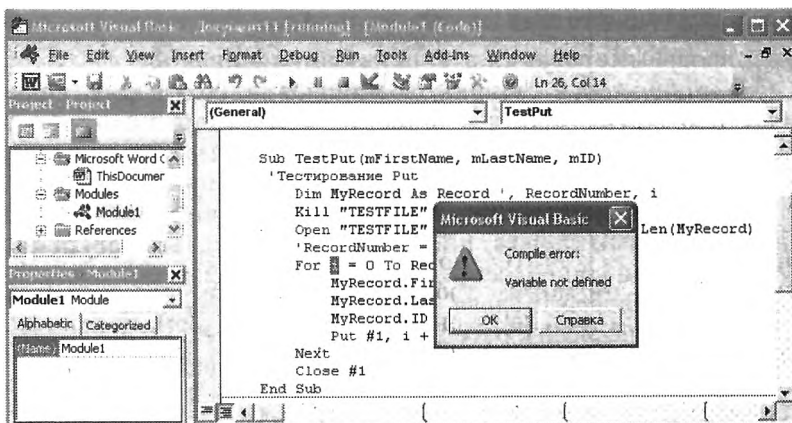
скую ошибку *ошибкой компиляции (compile error)* — редактор VB проверяет синтаксис запуском компилятора, и ошибка происходит во время компиляции, хотя это не означает, что ошибка не синтаксическая.

**Рис. 11.1**  
Обычно так редактор VB сообщает об ошибках синтаксиса



*Ошибки компиляции* возникают при попытке запустить подпрограмму, содержащую оператор, который VBA не в состоянии правильно откомпилировать. Если у VBA возникают какие-либо трудности в процессе компиляции написанного вами кода, то он выдает сообщение об ошибке и не создает оттранслированного кода. Например, если в вашем модуле содержится оператор **Option Explicit** и используется переменная без ее предварительного объявления (с помощью оператора **Dim**), VBA перемещает курсор вставки в точку, где обнаружена переменная, которую забыли объявить, и выдает сообщение об ошибке компиляции (рис. 11.2). Заметьте, что в отличие от синтаксической ошибки эта ошибка обнаруживается при попытке выполнить код процедуры.

**Рис. 11.2**  
Эта ошибка была обнаружена при попытке выполнить код процедуры



*Ошибки времени исполнения или runtime-ошибки* возникают, когда VBA встречает выражения или операторы, которые не удастся вычислить или выполнить, например, содержащие недопустимые команды, недопустимые аргу-

менты в процедурах и функциях, или недопустимые математические операции. Эти ошибки проявляются в процессе выполнения программного кода. Вот типичные примеры, приводящие к появлению runtime-ошибок:

- выражения, в результате которых происходит математическое переполнение;
- несоответствие типов переменных в выражениях присваивания или в аргументах подпрограммы;
- попытка открыть несуществующие файлы;
- попытка выполнить деление на ноль.

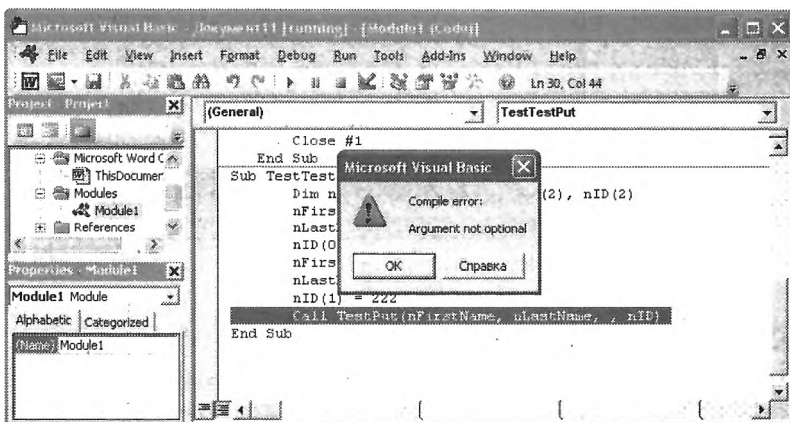
На рис. 11.3 в строке кода для вызова процедуры записи массивов **nFirstName**, **nLastName** и **nID** в файл с произвольным доступом

```
Call TestPut(nFirstName, nLastName, , nID)
```

оказалась лишняя запятая. VBA, как «мог», проанализировал строку и «решил», что мы хотели вызвать процедуру с пропущенным аргументом. На самом деле, редактор VBA не может абсолютно точно указать причину ошибки, поэтому не всегда можно понять из диагностики, какая же ошибка произошла в действительности.

**Рис. 11.3**

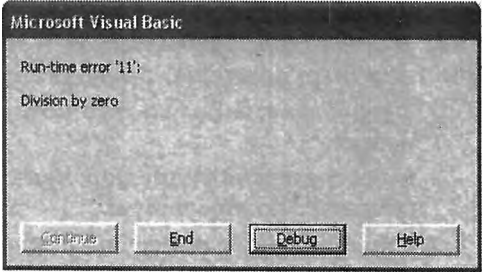
Ошибка в вызове процедуры



Конечно же, «классическим» примером ошибки времени исполнения является попытка деления на ноль, при возникновении которой выдается диалоговое окно, представленное на рис. 11.4. Следует отметить, что попытка разделить на ноль необязательно связана с невнимательным делением чего-либо на число ноль. Это может произойти (и чаще всего так и бывает) в результате того, что некоторая переменная (или выражение), являющаяся делителем, непосредственно перед делением будет оценена как нулевое значение.

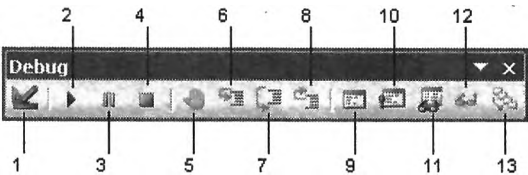
Если вы описали переменную как число и забыли присвоить ее значение (конечно, отличное от нуля), вы можете ожидать, что при делении какого-либо выражения на эту переменную может произойти ошибка от деления на ноль. Вы также могли и не забыть присвоить делителю отличное от нуля значение, но при наличии в коде различных ветвей ветвь, содержащая инициализацию ненулевого делителя, могла не выполниться.

**Рис. 11.4**  
Ошибка деления на ноль



Средства отладки

Рассмотрим средства отладки кода, предоставляемые системой VBA. Как и все современные системы, VBA имеет функции интерактивной отладки кода. Во-первых, это — панель **Debug**, которая представлена на рис. 11.5. Панель инструментов **Debug** не выводится на экран автоматически. Чтобы ее вывести, выберите в меню команду **View | Toolbars | Debug** (Вид | Панели инструментов | Отладка). (Когда панель инструментов выведена на экран, в меню слева от опции **Debug** появляется «галочка».) Панель инструментов **Debug** можно перемещать так же, как и любую другую панель инструментов и настраивать ее подобно панелям инструментов Windows-приложений.



**Рис. 11.5.** Панель **Debug**

- 1 – Exit Design Mode; 2 – Run Sub/UserForm; 3 – Break; 4 – Reset; 5 – Toggle Breakpoint; 6 – Step Into; 7 – Step Over; 8 – Step Out; 9 – Locals Window; 10 – Immediate Window; 11 – Watch Window; 12 – Quick Window; 13 – Call Stack

Наименования кнопок, указанных на рисунке, появляются на экране рядом с панелью, когда курсор мыши на несколько секунд остается на соответствующей кнопке. Назначение кнопок приведено в следующей таблице.

Кнопка	Назначение
Exit Design Mode	Переводит design-режим в состояние off или on.
Run Sub/UserForm	Служит для переключения из режима разработки в режим выполнения ( <b>Run</b> ) или из режима прерывания в режим выполнения ( <b>Continue</b> ). В режиме прерывания название кнопки меняется на <b>Continue</b> .
Break	Останавливает выполнение приложения и переключает систему отладки в режим прерывания.

Кнопка	Назначение
Reset	Очищает стек выполнения и переменные модульного уровня. Завершает выполнение кода.
Toggle Breakpoint	Устанавливает или отменяет в текущей строке точку останова. Здесь можно прервать программу, просмотреть некоторые данные и после продолжить выполнение кода.
Step Into	Выполняет следующую строку кода с заходом в процедуры. Код процедуры выполняется по шагам, как и до входа в процедуру.
Step Over	Выполняет следующую строку кода без захода в процедуры. Код процедуры выполняется как одна команда.
Step Out	Выполняет оставшуюся часть текущей процедуры и останавливает работу программы на следующей строке вызывающей процедуры.
Locals Window	Отображает текущие значения локальных переменных в отдельном окне.
Immediate Window	Позволяет выполнить нужные операторы или просмотреть значения переменных в режиме прерывания.
Watch Window	Открывает окно с текущими значениями выбранных выражений.
Quick Watch	Отображает текущее значение выражения в режиме прерывания.
Call Stack	Отображает в режиме прерывания окно со списком вызванных, но еще не выполненных процедур (стек вызовов).

### Замечание

Панель **Debug** по умолчанию не появляется на экране. Вы сами можете вызвать ее, выполнив команды: **View, Toolbars, Debug**.

## Режимы отладки

Чтобы тестировать и отлаживать (находить ошибки) приложение, вам необходимо понять, какие режимы работы используются при создании нового приложения. Основная часть работы происходит в *режиме разработки (design time)*. После того как разработчик приложения выбирает команды **Run, Start**, редактор VB переходит в *режим исполнения (run time)*. Этот режим еще не является «боевым», поскольку поддерживается средой редактора VB и разработчик имеет возможность тем или иным способом приостановить выполнение кода в определенном месте для выполнения отладочных операций, т.е. перейти в режим прерывания. В *режиме прерывания (break mode)* программа не выполняется, но занимает память; здесь вы можете просматривать и изменять значения переменных и выражений и даже перемещаться по коду, выполняя один или группу операторов.

Основное время при создании приложения занимает режим разработки. Здесь к приложению добавляются формы, элементы управления на формах,

модули кода и так далее. В этом режиме используется только одно средство отладки — редактор кода, но, кроме того, вы можете расставить точки останова и определить контрольные выражения.

В заголовке главного окна Visual Basic всегда отображается режим, в котором в данный момент времени находится система: в заголовке окна приложения после имени документа в квадратных скобках помещаются слова **design**, **running**, **break**. Для переключения режимов можно, кроме меню, использовать кнопки **Start**, **Break** и **End** панели **Debug**.

## Режим останова

В процессе отладки программы у вас часто будет возникать потребность, проследить за тем, как именно VBA выполняет операторы вашей программы и какие значения принимают некоторые переменные на разных этапах ее выполнения. Режим останова редактора VB предоставляет возможность более близкого «взаимодействия» с написанным вами VBA-кодом по сравнению с обычным его выполнением (когда код выполняется от начала до конца с максимальной скоростью, которую может обеспечить VBA, и вы не в состоянии уследить ни за тем, какие части кода исполняются в данный конкретный момент, ни за тем, чему в этот момент равны значения интересующих вас переменных). В режиме же останова вы получаете возможность исполнения кода вашей программы или отдельных его частей построчно или по одной подпрограмме за шаг, как в замедленной съемке. Пооператорное выполнение программы называется *пошаговым (single-stepping)* проходом.

В VBA перевести редактор VB в режим останова можно следующими способами:

- Щелчком на кнопке **Debug** (Отладка) в диалоговом окне runtime-ошибки.
- Задав одну или несколько точек останова.
- Вставив в текст программы оператор **Stop**.
- Воспользовавшись командой **Debug | Step Into** (Отладка | Шаг с заходом), кнопкой **Break** или **Step Into** на панели **Debug** (или нажав F8).
- Нажав Esc или Ctrl+Break, чтобы прервать выполнение программы.

## Переход в режим останова из окна сообщения об ошибке

Всякий раз, когда в процессе выполнения одной из ваших VBA процедур или функций происходит runtime-ошибка, появляется диалоговое окно ошибки, подобное тому, что показано на рис. 11.4 (конкретное сообщение об ошибке может быть другим, в зависимости от того, какая именно ошибка произошла). В диалоговом окне выводится номер ошибки и дается краткое пояснение характера ошибки. Кроме того, это окно имеет несколько кнопок.

В диалоговом окне, появляющемся при возникновении runtime-ошибки, имеются следующие кнопки:

- **Continue** (Продолжить) — продолжить выполнение прерванного VBA-кода. В случае возникновения runtime-ошибки управляющая кнопка **Continue**, как правило, отключена (см. рис. 11.4).
- **End** (Завершить) — завершить выполнение VBA-кода и выйти из режима останова.

- **Debug (Отладка)** — вывести оператор VBA, который привел к возникновению runtime-ошибки; в случае необходимости открываются соответствующие проект и модуль. VBA остается в режиме останова, позволяя разработчику VBA-приложения проверить значения переменных, проследить цепочку подпрограмм, приведшую к подпрограмме, исполняемой в данный момент, и ряд других параметров связанных с отладкой. Используйте кнопку **Debug** для быстрого перехода к оператору, выполнение которого привело к появлению ошибки.
- **Help (Справка)** — вывести диалоговое окно со справочной информацией, относящейся к возникшей runtime-ошибке.
- Для того чтобы перейти в режим останова из диалогового окна сообщения о runtime-ошибке, просто нажмите на кнопку **Debug**. Редактор VB открывает **Code Window** (окно кода), в котором появится выделенный желтым цветом оператор, приведший к появлению runtime-ошибки.

## Точки останова

Для того чтобы обнаружить причину появления runtime- и логических ошибок, в большинстве случаев приходится прибегать к пошаговому выполнению операторов разрабатываемой программы. Пошаговый проход программы, позволяет либо точно узнать, какой оператор выполнялся в момент возникновения runtime-ошибки, либо дает возможность проконтролировать выполнение группы операторов, в корректности исполнения которых разработчик сомневается.

Пошаговый проход всех операторов подпрограммы, а тем более программы в целом, может оказаться занятием довольно утомительным и требующим больших затрат времени. Желание или возможность выполнить пошаговый проход всех операторов программы возникает нечасто. Редактор VB позволяет выполнить большую часть кода с максимальной скоростью, переходя в режим останова только тогда, когда это необходимо для выполнения отдельных операторов кода. (Помните, чтобы выполнить пошаговый проход кода, необходимо находиться в режиме останова.)

**Точка останова (breakpoint)** — это специально помеченная строка программы. Когда VBA доходит до точки останова, происходит его переключение из режима обычного выполнения программы в режим останова. Использование точек останова позволяет выполнить большую часть программы с полной скоростью с переходом в режим останова, только когда VBA достигает определенных операторов, выполнение которых вы хотите проверить более тщательно. Точка останова «действует» до тех пор, пока вы ее не удалите или не закроете проект, содержащий модуль, в котором она установлена.

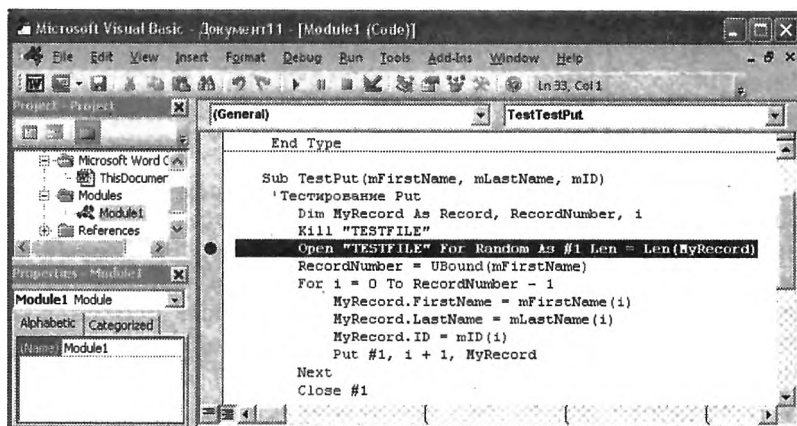
Точку останова можно поместить в любой строке исходного кода, содержащей исполняемый оператор. Обычно точка останова задается перед оператором, который, как вы знаете (или предполагаете), является причиной возникновения проблем. Для того чтобы задать точку останова, необходимо выполнить следующие шаги:

1. Выведите на экран VBA модуль и подпрограмму, содержащие операторы, для которых вы хотите задать пошаговое выполнение.

2. В **Code Window** (окне программы), поместите курсор вставки на строке, начиная с которой нужно, чтобы VBA переключился из обычного выполнения программы в режим останова. Эта строка станет новой точкой останова и должна содержать выполняемый оператор VBA.
3. Выберите команду **Debug | Toggle Breakpoint** (Отладка | Точка останова), чтобы вставить точку останова. [Вы также можете нажать клавишу F9 или воспользоваться кнопкой **Toggle Breakpoint** (Точка останова) на панели инструментов **Debug** (Отладка) редактора VB.] VBA выделяет выбранную в качестве точки останова строку (темно красным цветом), как показано на рис. 11.6; а рядом со строкой у левой границы **Code Window** появляется (темно-красная) точка.

**Рис. 11.6**

VBA выделяет выбранную в качестве точки останова строку, а рядом со строкой у левой границы окна **Code Window** появляется точка



Чтобы удалить точку останова, необходимо проделать те же самые действия, что и для ее вставки. Если применить команду **Toggle Breakpoint** к строке, содержащей точку останова, VBA удалит эту точку. Для того чтобы удалить все точки останова в модуле, необходимо выбрать команду **Debug | Clear All Breakpoints** (Отладка | Снять все точки останова) или нажать клавиши Ctrl+Shift+F9.

## Использование оператора Stop

Точки останова, заданные при помощи команды **Toggle Breakpoint**, «действуют» только в течение текущего рабочего сеанса. Иногда, особенно в случае сложных программ, отладка может потребовать нескольких рабочих сеансов. Поэтому вы, возможно, захотите установить «многоразовые» точки останова. Именно для этой цели целесообразно использовать ключевое слово **Stop**.

Когда VBA встречает оператор **Stop**, выполнение программы приостанавливается, отображаются модуль и процедура, содержащие выполняемый в данный момент оператор, и происходит переключение в режим останова. При использовании оператора **Stop** создается *постоянная точка останова*, то есть она, фактически, становится частью VBA-кода. Единственный способ удалить точку останова, созданную при помощи оператора **Stop**, — это удалить сам оператор **Stop** из исходного кода.



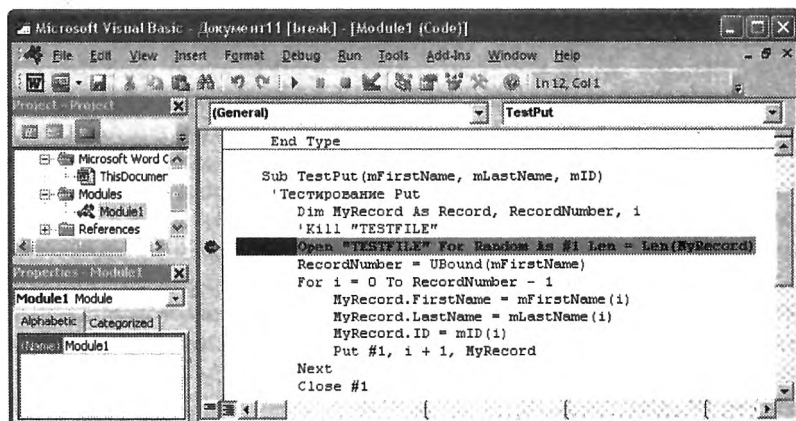
Как правило, оператор **Stop** вставляют в текст программы в случаях, когда нужна постоянная точка останова. После завершения отладки программы необходимо отредактировать текст программы, удалив из него операторы **Stop**.

### Переход в режим останова с помощью команды **Step Into**

Вы уже знаете, что можете запустить выполнение процедуры непосредственно из исходного кода, используя команду **Run | Run Sub/UserForm** или щелкнув (с помощью мыши) кнопку **Run Sub/UserForm** на панели инструментов **Debug**. Подобным же образом можно запустить на выполнение любую процедуру в режиме останова. Просто поместите курсор вставки внутрь процедуры, которую хотите выполнить в режиме останова, и выберите команду **Debug | Step Into** (Отладка | Шаг с заходом). (Вы также можете нажать **F8** или щелкнуть мышью по расположенной на панели инструментов **Debug** кнопке команды **Step Into**.) VBA перейдет в режим останова и покажет текущий исполняемый оператор подпрограммы. Как упоминалось выше, в режиме останова текущий исполняемый оператор выделяется в **Code Window** желтым цветом и стрелкой у левой границы текста кода (как показано на рис. 11.7).

**Рис. 11.7**

В режиме останова текущий исполняемый оператор выделяется в **Code Window** желтым цветом и стрелкой у левой границы текста кода



### Переход в режим останова прерыванием исполнения кода

Как вы уже знаете, VBA прерывает выполнение кода, когда пользователь нажимает клавишу **Esc** (или комбинацию **Ctrl+Break**). Когда вы прерываете исполнение кода, VBA выдает диалоговое окно ошибки с сообщением, что выполнение кода было прервано, и предлагает те же самые возможности выбора, что и диалоговое окно, появляющееся при возникновении runtime-ошибки: **Continue**, **End**, **Debug** и **Help**. Чтобы перейти к выполнению программы в режиме останова, щелкните на кнопке **Debug**. VBA перейдет в режим останова и укажет оператор, выполнявшийся в тот момент, когда вы прервали процедуру.

### Выход из режима останова

Часто возникает необходимость «пройти» часть программы в пошаговом режиме, после чего завершить выполнение оставшейся части в обычном режиме. Возможно, в некоторых случаях вы также захотите как выйти из режима ос-

танова, так и завершить выполнение программы, чтобы получить возможность, внести исправления в текст программы для устранения выявленных в режиме останова ошибок.

Для того чтобы выйти из режима останова и продолжить выполнение программы в обычном режиме, воспользуйтесь командой **Run | Continue** (Запуск | Продолжить). VBA выйдет из режима останова и продолжит выполнение программы до тех пор, пока не произойдет ее нормальное завершение. Если VBA встретится с точкой останова или оператором **Stop**, прежде чем достигнет конца программы, то произойдет переход в режим останова. (Выйти из режима останова и продолжить выполнение программы, можно также нажатием F5 или щелчком мыши по кнопке **Continue** на панели инструментов **Debug**.)

Для того чтобы выйти из режима останова и одновременно завершить выполнение всех программ, выберите команду **Run | Reset** (Запуск | Сброс). VBA выйдет из режима останова и прервет дальнейшее выполнение программы, все переменные потеряют свои значения, а все процедуры будут удалены из памяти. (Вы можете также выйти из режима останова и завершить выполнение программы, щелкнув мышью по кнопке **Reset** на панели **Debug**.)

## Использование команды Step Into

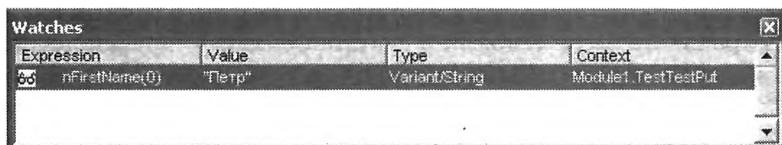
Теперь, вы знаете, как перейти в режим останова и как из него выйти, поэтому можно приступить к изучению использования команды **Step Into** (шаг с заходом) для пошагового выполнения программы в режиме останова. Пошаговое выполнение операторов позволяет в процессе отладки проследить за исполнением отдельных процедур, функций или программы в целом. При пошаговом проходе исходного кода с использованием команды **Step Into** VBA входит в тело каждой вызываемой процедуры или функции, давая возможность выполнить в пошаговом режиме и их операторы тоже. Пошаговое выполнение программы с использованием команды **Step Into** позволяет проследить за выполнением всех операторов главной процедуры и всех операторов каждой функции или процедуры, вызываемых из основной программы.

В **Code Window** операторы, которые будут выполняться следующими, выделены желтым цветом. Если код не такой простой, как в процедуре **TestTestPut**, и имеет условные операторы или операторы циклов, то пошаговый проход программы, выполняемый таким образом, может в некоторых случаях помочь понять, как работают используемые разработчиком приложения программные структуры. Перед выполнением следующей команды можно, помещая курсор вставки на переменные или выражения, в окне **ToolTip** просмотреть значение переменной или выражения.

Режим подсказки значений данных (позволяющий просматривать значения выражений в окне типа **ToolTip**) по умолчанию включен. Если по каким-то причинам на вашем компьютере подсказки не работают, выберите команду **Tools | Options** и на вкладке **Editor** окна **Options** выберите флажок **Auto Data Tips**.

## Наблюдение за значениями выражений

Если в процессе пошагового выполнения операторов кода вас интересует не только последовательность выполнения операторов, но и изменение значений переменных, то вас вряд ли удовлетворит возможность просматривать значения переменных (и даже выражений) в окне **ToolTip**. Редактор VB имеет окно **Watches**, в котором можно просматривать значения выражений по мере пошагового выполнения кода. Например, на рис. 11.8 приведено окно **Watches**, в котором отображается значение переменной **nFirstName(0)** процедуры **TestTestPut** модуля **Module1**.



**Рис. 11.8.** В режиме пошагового выполнения кода в **Watch Window** можно наблюдать за изменением значений выражений (переменных)

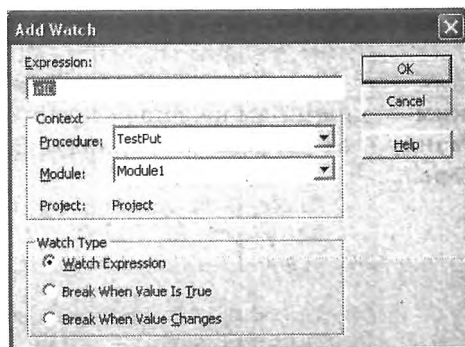
Для просмотра значений выражений (переменных) процедуры необходимо занести (добавить) эти выражения в окно **Watches**. Это можно выполнить двумя способами, предварительно выделив выражение (для переменной можно просто поместить на ней курсор вставки):

- Из контекстного меню (правой кнопки мыши) выбрать опцию **Add Watch**.
- Из меню **Debug** выбрать опцию **Add Watch**.

В любом случае на экране появляется диалоговое окно (рис. 11.9) **Add Watch**, в котором в самом простом случае можно просто щелкнуть на кнопке **OK**, после чего в окне **Watches** (если ранее его не было на экране, то оно выводится) появляется строка, содержимое которой соответствует содержимому окна **Add Watch** (рис. 11.10): совпадают поля **Expression** и **Context**. В поле **Value** окна **Watches** либо отображается значение выражения, либо указывается, что выражение находится «вне контекста» (<Out of context>), либо (для массивов) выводится сообщение о том, что «индекс выходит за пределы допустимого диапазона» (<subscript out of range>).

**Рис. 11.9**

Диалоговое окно **Add Watch**, в котором в самом простом случае можно щелкнуть на кнопке **OK**



Expression	Value	Type	Context
nFirstName(0)	"Петр"	Variant/String	Module1.TestTestPut
nID	333	Variant/Variant(0 to 2)	Module1.TestTestPut
nID(0)	333	Variant/Integer	Module1.TestTestPut
nID(1)	222	Variant/Integer	Module1.TestTestPut
nID(2)	Empty	Variant/Empty	Module1.TestTestPut

**Рис. 11.10.** В режиме пошагового выполнения кода в окне **Watch** можно наблюдать за изменением значений выражений (переменных)

Вернемся к окну **Add Watch**. Как уже можно понять, в секции **Expression** указывается выражение (переменная), в секции **Context** — наименование процедуры и модуля, где находится выражение. Если перед вызовом окна **Add Watch** добавляемое выражение уже указано (выделено), то в секции **Context** ничего изменять не нужно. Если в момент вывода диалогового окна **Add Watch** курсор вставки не указывал ни на какую переменную, то в текстовом окне **Expression** необходимо будет ввести выражение (переменную). В любом случае окна комбинированных списков **Procedure** и **Module** в секции **Context** будут содержать имена текущих процедуры и модуля, на коде которых находится курсор вставки. Конечно, можно воспользоваться этими списками для выбора других доступных модулей и процедур.

Секция **Watch Type** окна **Add Watch** предоставляет дополнительные возможности для отладки. При помощи переключателя **Break When Value Is True** (останов, если значение выражения истинно) этой секции можно использовать окно **Add Watch** не в пошаговом режиме выполнения кода, а в обычном. Для подобной же цели можно использовать и переключатель **Break When Value Changes** (останов при изменении значения выражения). Как уже должно быть понятно, режимом выполнения кода в этих случаях может быть обычный режим. При соответствующих изменениях указанных выражений произойдет переход в режим останова.

Таким образом, если задать в окне интересные разработчика выражения, можно попеременно задавая то пошаговый, то обычный режим выполнения приложения, просматривать изменения значений выражений.

На рис. 11.11 показаны **Code Window** и окно **Watches** при пошаговом выполнении кода приведенного ниже листинга. Наблюдаемыми переменными являются два массива: **nFirstName** и **nID**. Оба массива уже получили значения для первых элементов, но **nFirstName** получил также значение для второго элемента, а **nID** — нет.

### Листинг 11.1. Код для изучения работы с окном **Watches**

```

1: Option Explicit
2: Type Record ' Определенный пользователем тип
3:     ID As Integer
4:     FirstName As String * 20
5:     LastName As String * 20
6: End Type
7:
8: Sub TestPut(mFirstName, mLastName, mID)
9:     'Тестирование Put

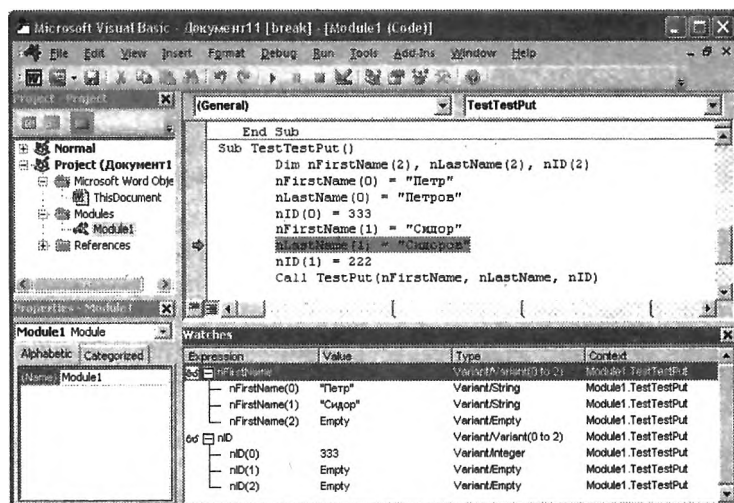
```

```

10: Dim MyRecord As Record, RecordNumber, i
11: Kill "TESTFILE"
12: Open "TESTFILE" For Random As #1 Len = Len(MyRecord)
13: RecordNumber = UBound(mFirstName)
14: For i = 0 To RecordNumber - 1
15:     MyRecord.FirstName = mFirstName(i)
16:     MyRecord.LastName = mLastName(i)
17:     MyRecord.ID = mID(i)
18:     Put #1, i + 1, MyRecord
19: Next
20: Close #1
21: End Sub
22: Sub TestTestPut()
23: Dim nFirstName(2), nLastName(2), nID(2)
24: nFirstName(0) = "Петр"
25: nLastName(0) = "Петров"
26: nID(0) = 333
27: nFirstName(1) = "Сидор"
28: nLastName(1) = "Сидоров"
29: nID(1) = 222
30: Call TestPut(nFirstName, nLastName, nID)
31: End Sub

```

**Рис. 11.11**  
Окно **Code** и **Watches**  
при пошаговом  
выполнении кода  
приведенного ниже  
листинга



## Редактирование наблюдаемого выражения

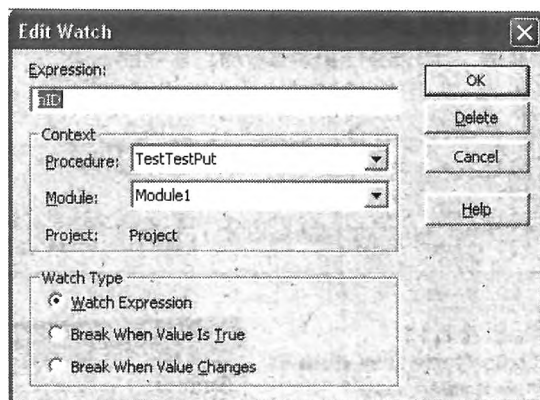
Наблюдаемые в окне **Watches** выражения можно редактировать после того, как они уже добавлены в это окно. Чтобы отредактировать контрольное выражение, можно выполнить следующее:

1. В окне **Watches** выделите выражение, которое хотите отредактировать.
2. Выберите команду **Debug | Edit Watch** (Отладка | Изменить контрольное значение). Эта команда активизирует диалоговое окно **Edit Watch**, которое отличается от окна **Add Watch** только заголовком и дополнительной кнопкой **Delete** (Удалить).

3. Выполните необходимые исправления в выражении, а также, если необходимо, внесите изменения в разделы **Context** и **Watch Type**. Заметьте, что выражение для контроля может вообще не встречаться в отлаживаемом коде. Важно только, чтобы оно содержало выражения, входящие в код. Например, вы можете остановить выполнение цикла за несколько итераций до указанной в качестве последней или наблюдать изменение не самой переменной некоторой программы, а результат, например, ее деления по модулю на какое-либо число.
4. Щелкнув кнопку **OK**, закройте диалоговое окно **Edit Watch**. VBA внесет в окно **Watches** соответствующие исправления.

**Рис. 11.12**

Окно **Edit Watch** отличается от окна **Add Watch** только заголовком и дополнительной кнопкой **Delete**



## Удаление наблюдаемого выражения

По мере работы с наблюдаемыми переменными и выражениями вы можете обнаружить, что список в окне **Watches** имеет тенденцию разрастаться, и в какой-то момент поймете, что значения далеко не всех находящихся в нем выражений, действительно, необходимо контролировать. Поэтому вам может понадобиться удалить их из окна **Watches**. Удалить контрольное выражение можно одним из двух способов:

1. Выделите выражение, которое хотите удалить из окна **Watches**, и нажмите клавишу **Del**. VBA удалит выделенное выражение из окна без предупреждения.
2. Выделите выражение, которое хотите удалить, затем выберите команду **Debug | Edit Watch**. В появившемся окне **Edit Watch** щелкните кнопку **Delete**. VBA удалит выделенное выражение из окна **Watches**.

## Использование команды Step Over

Если код вашего приложения состоит из большого количества процедур, а предполагаемый характер ошибки — неправильный алгоритм вызова этих процедур, то пошаговое выполнение каждой вызываемой процедуры может оказаться достаточно утомительным. Во всяком случае, нет необходимости выполнять по шагам код процедуры, в которой вы абсолютно уверены (хотя, конечно, программист должен быть абсолютно уверен только в том, что в лю-

бой процедуре в любой момент времени содержится, по крайней мере, одна ошибка). Но даже, если вы «правильно» не уверены во всех процедурах, на каком-то этапе разработки приложения многие процедуры выполняют то, что от них требуется, и их пошаговое выполнение имеет мало смысла.

Отладчик VBA предоставляет в дополнение к команде **Step Into** (шаг с заходом) команду **Step Over** (шаг с обходом). При использовании команды отладки **Step Over**, если встречается оператор, вызывающий процедуру или определенную пользователем функцию, VBA не переходит к пошаговому выполнению кода вызванной подпрограммы. Вместо этого он выполняет код вызванной подпрограммы с обычной скоростью и продолжает пошаговое выполнение с первого оператора, следующего за вызовом подпрограммы.

Если вы уверены в нормальной работе процедур, вызываемых из некоторой управляющей процедуры, возможность избежать пошагового выполнения подпрограмм помогает сосредоточиться на отладке операторов текущей процедуры. Кроме того, за счет отказа от пошагового выполнения там, где в этом нет необходимости, можно сэкономить затрачиваемое на отладку время.

## Использование окна Locals

Окно **Locals** (локальные переменные), хотя и не связано с окном **Watches**, но выполняет похожую функцию (по умолчанию не появляется на экране и вызывается командой **View | Locals Window**). Оно выводит все локальные (или находящиеся в активной области видимости) переменные выполняемой в данный момент процедуры или функции. На рис. 11.14, например, показаны значения локальных переменных процедуры **TestTestPut** приведенного ранее листинга.

Окно **Locals** содержит три колонки: **Expression** (Выражение) (имя переменной или объекта), **Value** (Значение) (текущее значение переменной или свойство объекта) и **Type** (Тип) (тип данных или объекта). Обратите внимание на прямоугольники слева от элементов **mFirstName**, **mLastName** и **mID** в окне **Locals** на рис. 11.13. Они аналогичны прямоугольникам в других раскрывающихся древовидных списках операционной системы Windows. Если в прямоугольнике содержится знак «плюс», это означает, что, щелкнув по нему мышью, можно открыть древовидную диаграмму со свойствами и значениями объекта. Значок «минус» показывает, что диаграмма уже раскрыта и, щелкнув по нему, вы можете ее свернуть. Если щелкнуть по значку «плюс» слева от элемента **mFirstName**, появится список всех элементов массива и значения, которые каждый из них имеет в настоящий момент. Окно **Locals** удобно использовать для наблюдения за текущими значениями и структурой переменных и объектов выполняемой в данный момент процедуры, функции или модуля.

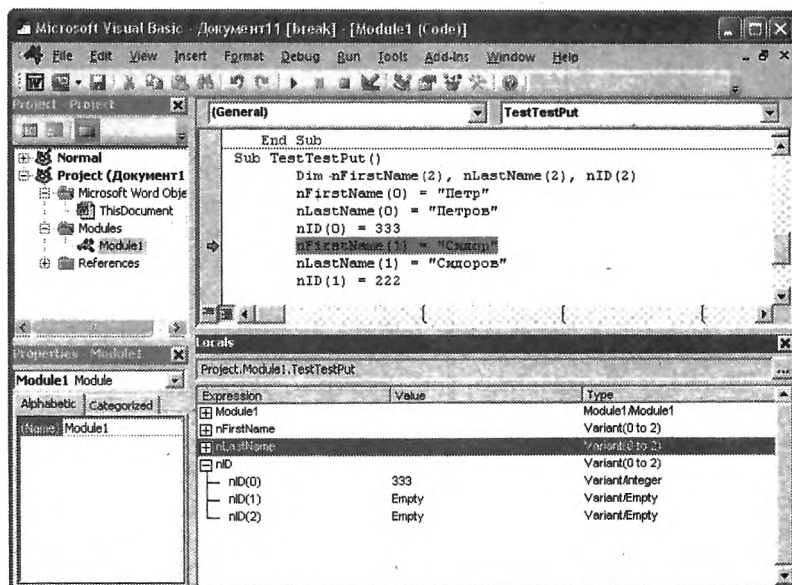
## Трассировка вызовов процедур

При отладке программ довольно часто возникает необходимость определить точную *цепочку* или последовательность процедур, приведшую к выполнению конкретного оператора, который порождает ошибки. Знание цепочки вызовов процедур может оказаться полезным, поскольку часто играет важную роль в оп-

ределении причины ошибки. Например, вы можете обнаружить оператор, приводящий к ошибке деления на ноль, но не знаете, почему одна из переменных в операторе содержит ноль. Знание точной цепочки вызовов процедур поможет вам определить, что процедура была вызвана, например, с пропущенным необязательным параметром или просто с неверным значением требуемого аргумента.

VBA позволяет просматривать цепочку вызовов процедур посредством диалогового окна **Call Stack** (Стек вызова). Чтобы вывести окно **Call Stack** на экран (в режиме останова), либо выберите команду **View | Call Stack** (Вид | Стек вызова), либо щелкните мышью по кнопке **Call Stack** на панели инструментов **Debug**, либо нажмите **Ctrl+L**. На экране появится диалоговое окно **Call Stack**, пример которого показан на рис. 11.14 (последовательность вызовов процедур может быть иной). Как видно из рис. 11.14, диалоговое окно **Call Stack** выводит список последовательности вызовов процедур, приводящей к выполняемой в данный момент процедуре или функции.

**Рис. 11.13**  
Окно **Locals** при  
пошаговом  
выполнении кода  
приведенного  
ранее листинга



Вы, наверное, уже заметили, что диалоговое окно **Call Stack** содержит две кнопки — **Show** (Показать) и **Close** (Заккрыть). Кнопка **Close** просто закрывает диалоговое окно. Кнопка **Show** выполняет очень полезную функцию и служит для того, чтобы можно было найти оператор, вызвавший исполняемую в данный момент подпрограмму. Кнопкой **Show** можно воспользоваться для проверки значений аргументов, переданных выполняемой процедуре. Для этого откройте диалоговое окно **Call Stack**, выберите имя процедуры, которая вызвала текущую процедуру, и щелкните командную кнопку **Show**. VBA закроет диалоговое окно **Call Stack** и отметит в окне **Code** оператор, содержащий вызов текущей процедуры. Используя кнопку **Show** и диалоговое окно **Call Stack**, можно пройти в обратном направлении вдоль всей цепочки вызовов подпрограмм.

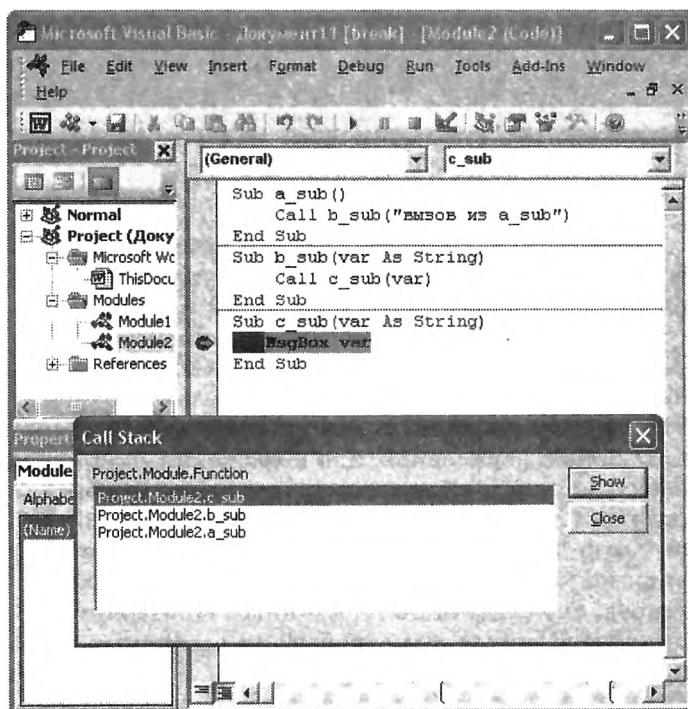


## Использование окна Immediate

Иногда даже сочетания пошагового выполнения программы с использованием наблюдаемых переменных и трассировки вызовов подпрограмм бывает недостаточно, чтобы найти содержащуюся в программе ошибку. В редакторе VB предусмотрено еще одно средство для нахождения ошибок в вашей программе — окно **Immediate** (Проверка) редактора VB является редактором свободного формата, позволяющим контролировать значения переменных и выражений, производить вычисления, изменять значения переменных и проверять результаты выполнения функций. Все эти функции поддерживаются, когда программа находится в режиме останова. Для того чтобы воспользоваться окном **Immediate**, выберите **View | Immediate Window** или введите с клавиатуры комбинацию **Ctrl+G**, или щелкните на кнопке **Immediate Window** на панели инструментов **Debug**. VBA выведет диалоговое окно **Immediate**.

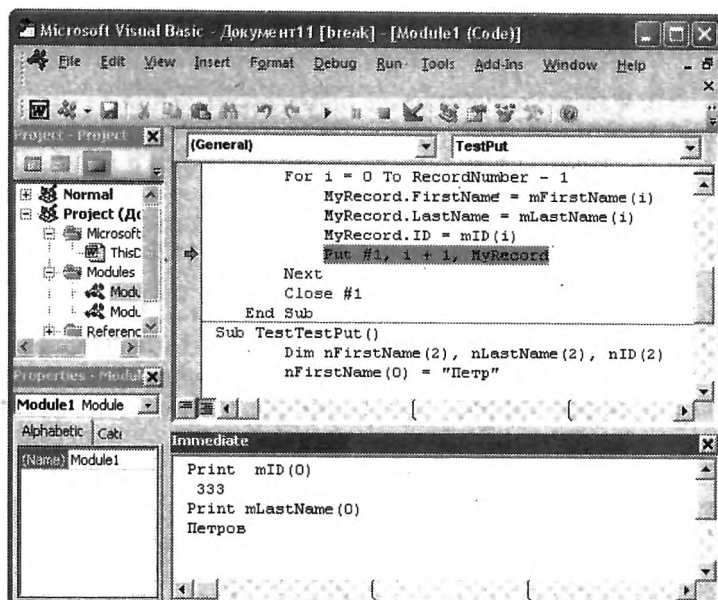
**Рис. 11.14**

Окно **Call Stack** и код, посредством которого можно получить такое окно



Чтобы вывести в окне **Immediate** интересующую вас информацию, используйте оператор **Print** (или **?**). Оператор **Print** вводится в окне **Immediate** вместе со списком переменных или выражений. Достаточно набрать команду **Print**, перечислить интересующие вас переменные и выражения, а затем нажать клавишу **Enter**. VBA вычислит значения переменных, перечисленных после команды **Print**, и выведет результаты в строке следующей за командой **Print**. На рис. 11.15 показан пример работы в окне **Immediate**.

**Рис. 11.15**  
Просмотр данных в окне  
**Immediate**



## Оператор Debug.Print

Объект VBA Debug имеет метод **Print**, позволяющий вывести информацию в окно **Immediate** в процессе выполнения программы. Для этого необходимо добавить в программу VBA операторы, вызывающие метод **Debug.Print**.

Метод **Debug.Print** используется, как правило, в сочетании с уже изученным в этой главе оператором **Stop**. Сочетание этих двух элементов позволяет выполнять программы в обычном (неотладочном) режиме, получая при этом отладочную информацию аналогичную той, которую дает использование наблюдаемых переменных или выражений. Как правило, бывает необходимо добавить один или несколько операторов, вызывающих метод **Debug.Print**, выводящих значения определенных переменных или выражений в окно **Immediate**. Хотя, при выполнении программы в нормальном режиме окно **Immediate** невидимо, как только VBA встретит оператор **Stop** и переключится в режим останова, оно сразу станет доступным и позволит вам просмотреть всю информацию, выведенную включенными в программу операторами **Debug.Print**. Используя этот технический прием, можно сэкономить массу отладочного времени за счет отказа от пошагового выполнения кода. Просмотрев информацию, содержащуюся в окне **Immediate**, вы можете возобновить выполнение программы в обычном режиме.

## Оператор Debug.Assert

Объект VBA Debug имеет метод **Assert**, приостанавливающий выполнение кода на строке, содержащей этот метод, при выполнении некоторого условия. Синтаксис метода следующий:

## Синтаксис

---

`Debug.Assert Booleanexpression`

Аргумент *Booleanexpression* — логическое выражение, имеющее результатом значение **True** или **False**.

---

Этот метод, практически, работает как оператор **Stop**, но выполняется, когда его аргумент принимает значение **False**.

## Обработчик ошибок

Обработчик ошибок — это код для перехвата и обработки ошибок в вашем приложении. Использование ваших собственных обработчиков ошибок говорит о том, что вы заметно продвинулись по пути изучения программирования в системе Windows и очень скоро пользователи ваших программ перестанут получать сообщения об ошибках на непонятном языке сообщений Windows.

Создание своего собственного обработчика ошибок состоит из следующих шагов:

- Установки ловушки прерываний (error trap) сообщением места кода, куда следует перейти при возникновении ошибки (адрес обработчика). Этот шаг осуществляется посредством оператора **On Error**, который делает доступным перехват прерываний и указывает метку, за которой начинается обработчик прерываний.
- Написания процедуры обработки прерываний, связанных с ошибками вашего приложения.
- Обеспечения выхода из процедуры обработки прерывания.

Установка ловушки прерываний осуществляется при помощи оператора **On Error**, который при этом указывает на обработчик прерывания. Ловушка остается активной на время работы процедуры (функции), в которой она установлена, т.е. до того, как выполнится один из операторов **Exit Sub**, **Exit Function**, **Exit Property**, **End Sub**, **End Function** или **End Property** этой процедуры (функции). В любой момент времени может быть доступна только одна ловушка в данной процедуре. Но вы можете создавать много ловушек и активизировать их в разное время. Ловушку можно сделать неактивной применением специального типа команды **On Error — On Error GoTo 0**.

Простой синтаксис оператора **On Error GoTo** имеет вид:

## Синтаксис

---

`On Error GoTo label`

Здесь *label* — метка (отмечаемая в коде двоеточием за ней), за которой начинается обработчик.

---

Итак, первым оператором блока кода обработки прерывания по ошибке является метка с двоеточием за ней. Далее без всяких ограничивающих блоки операторов следует писать код, который, кстати, необязательно должен обра-

батывать ошибки: вы можете просто констатировать факт наличия ошибки. Во-первых, не всегда можно идентифицировать ошибку, а во-вторых, не всякую ошибку можно исправить. Вы можете, например, только предположить, что произошло, и попытаться дать рекомендации пользователю, как избежать повторения какой-либо ошибки. Для кодов-примеров мы, конечно, будем точно знать типы ошибок, потому что сами будем их инициировать.

Перед меткой, задающей начало обработчика, должен находиться оператор выхода из процедуры (функции). Иначе даже при отсутствии ошибки код обработчика будет выполняться. В листинге 11.2 приведен пример обработчика, который выполняется в том случае, если пользователь введет в текстовом окне не число, а что-либо отличное от числа.

---

### Листинг 11.2. Пример обработчика ошибок

---

```
1: Private Sub Command1_Click()  
2:     Dim z, y As Integer  
3:     On Error GoTo err  
4:     y = InputBox("Введите число")  
5:     z = y + 10  
6:  
7:     MsgBox z  
8:     MsgBox "Спасибо!"  
9:     Exit Sub  
10:  
11: err:  
12:     MsgBox ("Просили же Вас ввести число!")  
13: End Sub
```

---

Как видно из этого примера, здесь никто не собирався идентифицировать ошибку: в таком простом коде можно было только предположить, что всему виной будет пользователь, который не может пользоваться окном ввода целого числа. Тем не менее, в Visual Basic есть возможность узнать тип ошибки (как ее «для себя» определяет Visual Basic) посредством объекта **Err** с несколькими свойствами, из которых для нас наиболее полезны **Number** — номер ошибки, и **Description** — описание ошибки. При этом в документации MSDN есть замечание с рекомендацией не очень доверять свойству **Description**, а использовать только свойство **Number**. Например, код листинга 11.2 можно дополнить анализом произошедшей ошибки (см. листинг 11.3).

---

### Листинг 11.3. Анализ ошибки в обработчике

---

```
1: Private Sub Command1_Click()  
2:     Dim z, y As Integer  
3:     On Error GoTo err  
4:     y = InputBox("Введите число")  
5:     z = y + 10  
6:  
7:     MsgBox z  
8:     MsgBox "Спасибо!"  
9:     Exit Sub  
10:  
11: err:  
12:     If Err.Number = 13 Then
```

```

13:      MsgBox ("Просили же Вас ввести число!")
14:      End If
15:  End Sub

```

Если вы предполагаете, что в течение работы с процедурой могут произойти самые разные ошибки, можно в коде обработчика использовать оператор **Select Case** для возможных действий, исправляющих ситуацию.

Как вы могли уже заметить, в листингах 11.2–11.3 сразу после выполнения кода обработчика вся процедура завершается. Такой обработчик хорош только в качестве примера того, как не нужно поступать. Если мы по всяким пустякам будем завершать процедуры, то с таким приложением будет трудно работать.

Итак, нам нужен способ обработать ошибку или, в крайнем случае, сообщить о ней, но обязательно как-то продолжить выполнение процедуры.

## Выход из блока обработки прерывания по ошибке

Для выхода из блока обработки ошибки можно ничего не предпринимать, и работа функции будет завершена. Но поскольку с таким образом завершившей свою работу процедурой (или функцией) могут быть связаны другие части кода, видимо, не всегда будет правильно просто отказаться от дальнейшего выполнения процедуры. По крайней мере, надо как-то сообщить о том, что процедура завершилась из-за ошибки. Например, если функция должна была вернуть результат (некоторых вычислений), используемый в дальнейших расчетах, то аварийный выход по ошибке дает мало пользы. Ошибка может произойти и при открытии файла. Если не сообщить об этом при выходе из блока обработки ошибки, то найдется такая функция, которая попытается считать данные из неоткрытого файла. И так далее.

Для выхода из блока обработки ошибок предназначен оператор **Resume**, функции которого приведены в следующей таблице.

Оператор	Описание
Resume [0]	Выполнение кода возобновляется с того оператора, который вызвал ошибку. Используйте этот оператор для повторения кода после исправления ошибки.
Resume Next	Возобновляет выполнение кода с оператора, следующего непосредственно за оператором, вызвавшим ошибку. Если ошибка произошла вне процедуры, содержащей блок обработки ошибок, выполнение кода возобновляется с оператора, находящегося за вызовом процедуры, вызвавшей ошибку, если в этой процедуре нет своего обработчика.
Resume line	Возобновляет выполнение кода с указанной метки <i>line</i> . Метка должна находиться в процедуре, содержащей обработчик ошибок.
Err.Raise Number:= number	Эмулирует ошибку времени исполнения. Когда этот оператор выполняется внутри блока обработки ошибок, Visual Basic обращается к списку вызовов (последовательность вызова процедур) для поиска другого обработчика.

Пример использования оператора **Resume** приведен в листинге 11.4. Это самая простая форма оператора, при которой происходит возврат к строке кода, где произошла ошибка. По крайней мере, в данном случае этого вполне достаточно, поскольку после правильного ввода числа программа будет работать нормально.

---

**Листинг 11.4.** Использование в обработчике оператора **Resume**

---

```
1: Private Sub Command1_Click()  
2:     Dim z, y As Integer  
3:     On Error GoTo err  
4:     y = InputBox("Введите число")  
5:     z = y + 10  
6:  
7:     MsgBox z  
8:     MsgBox "Спасибо!"  
9:     Exit Sub  
10:  
11: err:  
12:     If err.Number = 13 Then  
13:         MsgBox ("Просили же Вас ввести число!")  
14:         Resume  
15:     End If  
16: End Sub
```

---

В листингах 11.5 и 11.6 приведены примеры использования оператора **Resume Next**. Причем в первом из них выполнится присвоение переменной числа 9999, если пользователь «забыл», как вводятся числа. Затем код продолжается со следующей строки (за оператором **InputBox**).

В листинге 11.6 обработчик более «лоялен» к пользователю, давая ему три попытки для ввода числа. Поэтому здесь используется и оператор **Resume**, и **Resume Next**. Для этого в код введен счетчик **co**, который при загрузке формы (процедура **Form\_Load**) инициализируется значением 1. В блоке обработки ошибок значение счетчика увеличивается на единицу. Как только пользователь использует три попытки ввода числа, но не введет число, оператор **Resume Next** прекратит пользовательские «мучения», как это делается (только гораздо раньше) в предыдущем листинге.

---

**Листинг 11.5.** Использование в обработчике оператора **Resume Next**

---

```
1: Private Sub Command1_Click()  
2:     Dim z, y As Integer  
3:     On Error GoTo err  
4:     y = InputBox("Введите число")  
5:     z = y + 10  
6:  
7:     MsgBox z  
8:     MsgBox "Спасибо!"  
9:     Exit Sub  
10:  
11: err:  
12:     If err.Number = 13 Then  
13:         MsgBox ("С Вашего разрешения введем 9999")  
14:         y = 9999
```

```
15:      Resume Next
16:  End If
17: End Sub
```

---

### Листинг 11.6. Использование в обработчике оператора **Resume Next**

---

```
1:  Public co As Integer
2:  Private Sub Command1_Click()
3:      Dim z, y As Integer
4:      On Error GoTo err
5:      y = InputBox("Введите число")
6:      z = y + 10
7:
8:      MsgBox z
9:      MsgBox "Спасибо!"
10:     Exit Sub
11:
12:  err:
13:     If err.Number = 13 Then
14:         If co = 2 Then
15:             MsgBox ("С Вашего разрешения введем 9999")
16:             y = 9999
17:             Resume Next
18:         Else
19:             co = co + 1
20:             MsgBox ("У Вас осталось " & _
21:                 (3 - co) & " попытки")
22:             Resume
23:         End If
24:     End If
25: End Sub
26:
27: Private Sub Form_Load()
28:     co = 0
29: End Sub
```

---

Очень простой пример использования оператора **Resume line** приведен в следующем листинге.

### Листинг 11.7. Использование в обработчике оператора **Resume line**

---

```
1:  Public co As Integer
2:  Private Sub Command1_Click()
3:      Dim z, y As Integer
4:      On Error GoTo err
5:      y = InputBox("Введите число")
6:      z = y + 10
7:
8:      MsgBox z
9:      MsgBox "Спасибо!"
10:     Exit Sub
11:
12:  err_exit:
13:     MsgBox ("Что-то у Вас не получается")
```

```
14: Exit Sub
15:
16: err:
17: If err.Number = 13 Then
18:     If co = 2 Then
19:         Resume err_exit
20:     Else
21:         co = co + 1
22:         MsgBox ("У Вас осталось " & _
23:             (3 - co) & " попытки")
24:         Resume
25:     End If
26: End If
27: End Sub
28:
29: Private Sub Form_Load()
30:     co = 0
31: End Sub
```

---

В листинге 11.8 блок кода обработки ошибок находится не в той процедуре, где происходит ошибка при делении числа на ноль. Visual Basic при помощи списка вызовов возвращается в процедуру **A**, поскольку именно там был активизирован обработчик ошибок. Оператор **Resume Next** инициировал переход к строке кода, которая находится за оператором вызова процедуры **B**.

#### Листинг 11.8. Обработчик ошибок в одной из процедур

---

```
1: Private Sub Command1_Click()
2:     Call A
3: End Sub
4:
5: Sub A()
6:     On Error GoTo err
7:     Call B
8:     MsgBox ("В любом случае процедура а завершается")
9:     Exit Sub
10: err:
11:     MsgBox ("Где-то произошла ошибка!")
12:     Resume Next
13: End Sub
14:
15: Sub B()
16:     Call C
17: End Sub
18:
19: Sub C()
20:     Dim d As Integer
21:     d = 1 / 0
22: End Sub
```

---



## Программная обработка ошибок при помощи оператора On Error

Оператор **On Error** позволяет обрабатывать ошибки, возникающие во время выполнения кода, программно. Синтаксис оператора **On Error** следующий:

### Синтаксис

```
On Error GoTo line  
On Error Resume Next  
On Error GoTo 0
```

Синтаксис оператора **On Error** имеет следующие формы:

<b>On Error GoTo line</b>	Делает доступным обработчик ошибок, который начинается со строки, определенной аргументом <i>line</i> . Аргумент <i>line</i> может быть меткой или числом. При возникновении ошибки управление выполнением программы передается на строку кода с меткой <i>line</i> . Таким образом происходит активизация обработчика ошибок. Метка <i>line</i> должна находиться в той же процедуре, что и оператор <b>On Error</b> .
<b>On Error Resume Next</b>	Указывает на то, что при возникновении ошибки управление выполнением программы передается на оператор, следующий непосредственно за тем, в котором произошла ошибка. Этот оператор позволяет продолжить выполнение программы, несмотря на возникновение runtime-ошибки. Оператор <b>On Error Resume Next</b> становится неактивным при вызове другой процедуры.
<b>On Error GoTo 0</b>	Отключает пользовательский обработчик ошибок в данной процедуре. Этот оператор не указывает метку <b>0</b> в качестве начала обработчика ошибки, даже если в процедуре действительно имеется такая метка. Без этого оператора обработчик ошибок «отключается» автоматически после окончания работы процедуры.

Пользовательские обработчики ошибок являются составной частью «дружественного» интерфейса. Если во время работы пользователя с вашим приложением возникает не предусмотренная вами ситуация, то в любом случае не стоит оставлять пользователя «наедине» с системой VBA, не настолько дружелюбной, как хороший разработчик приложений. VBA «знает» очень мало о том, в каких местах вашего программного кода и какие могут возникать ошибки. Разработчик же приложения должен знать все «тонкие» места в коде и ожидать те или иные ошибки. Даже если приложение просто пытается открыть файл, который нечаянно «убили», VBA-сообщение об отсутствии файла не так информативно, как могло бы быть сообщение разработчика, которое не только должно сообщить об отсутствии файла, но и предложить возможные пути его восстановления или замены старой копией.

Следует различать состояния «доступность» (enabled) и «активность» (active) обработчика ошибок. Обработчик доступен, если оператор **On Error** выполнен, но ошибка еще не возникла. Обработчик активен, когда им выполняется обработка некоторой возникшей ошибки. Если ошибка возникает, когда обработчик активен (обрабатывает ранее возникшую ошибку), эта ошибка не обрабатывается и управление передается в вызывающую процедуру. Если вызывающая процедура имеет обработчик ошибок, находящийся в состоянии «enabled», обработчик переходит в состояние «active» (т.е. обработки ошибки). Если обработчик вызывающей процедуры находится в состоянии «active», то управление передается предыдущей вызывающей процедуре с обработчиком в состоянии «enabled», но не «active». Т.е. в цепочке вызывающих процедур ищется обработчик ошибок в состоянии «enabled», готовый обработать ошибку.

Обычно в процедурах обработки ошибок для определения причины ошибки используется свойство **Number** объекта **Err**. Сообщение об ошибке (довольно «скупое»), связанное с **Err.Number**, содержится в **Err.Description**. Поэтому обычно на стадии отладки после метки, на которую передается управление выполнением программы в случае возникновения (ожидаемой) ошибки, помещается следующий оператор:

```
MsgBox "Error " & Err & ": " & Err.Description
```

В процедуре листинга 11.9 используется **On Error Resume Next** для того, чтобы при обнаружении отсутствия в системе экземпляра приложения Excel программа не завершалась с ошибкой.

---

**Листинг 11.9.** Обработчик ошибок для защиты программы от прерывания

---

```
1: Sub TestError()  
2:   Dim xlApp As Object  
3:   Dim xlSheet As Object  
4:   On Error Resume Next  
5:   Set xlApp = Nothing  
6:   Set xlApp = GetObject(, "Excel.Application")  
7:   If Err.Number <> 0 Then  
8:     Set xlApp = CreateObject("Excel.Application")  
9:   End If  
10:  Err.Clear  
11:  
12:  Set xlBook = xlApp.Workbooks.Add  
13:  Set xlSheet = xlBook.WorkSheets(1)  
14:  xlSheet.Application.Visible = True  
15: End Sub
```

---

# Приложение А

## Операторы в VBA

---

### AppActivate

---

Активизирует окно приложения (application window).

#### Синтаксис

**AppActivate** *title*[, *wait*]

Оператор **AppActivate** имеет следующие именованные аргументы:

- |              |   |
|--------------|---|
| <i>title</i> | Обязательный. Строковое выражение, определяющее строку заголовка для окна активизируемого приложения. ID-задачи, возвращаемый функцией <b>Shell</b> , может использоваться в качестве параметра для <i>title</i> .  |
| <i>wait</i>  | Необязательный. Тип <b>Boolean</b> . Значение, определяющее, должно ли вызывающее приложение иметь фокус перед активизацией другого приложения. Если этот параметр равен значению <b>False</b> (по умолчанию), указанное приложение немедленно активизируется, даже если вызывающее приложение не имеет фокуса. Если этот параметр равен значению <b>True</b> , вызывающее приложение ожидает получение фокуса, после чего активизируется указанное приложение. |

Оператор **AppActivate** переносит фокус на именованное приложение или окно, но не воздействует на их максимизацию или минимизацию. Для запуска приложения (которое затем можно активизировать) можно использовать функцию **Shell**. **Shell** запускает исполняемую программу и возвращает значение, представляющее ID-задачи, если запуск выполнен успешно, или нуль — в противном случае.

В процессе определения того, какое приложение следует активизировать, аргумент *title* сравнивается со строкой заголовка каждого выполняющегося приложения. Если точного совпадения нет, то активизируется любое приложение, строка заголовка которого начинается с *title*. Если одновременно выполняются несколько экземпляров приложения с заголовком *title*, активизируется один из них, выбранный произвольно.

#### Пример

```
Sub Call_Excel()  
'в качестве аргумента оператора AppActivate можно использовать  
'возвращаемое значение функции Shell  
    MyAppID = Shell("e:\Program Files\Microsoft  
                  Office\Office11\EXCEL.EXE", 1)  
    AppActivate MyAppID  
End Sub
```

---

### Beep

---

Выдает звуковой сигнал посредством встроенного динамика.

#### Синтаксис

**Beep**

Частота и длительность сигнала зависят от оборудования и программного обеспечения компьютера.

### Пример

```
Function MyBeep(n As Integer)
    ' n - "длительность" звучания
    For I = 1 To n
        Beep
    Next
End Function
```

---

### Call

---

Передаёт управление (вызывает) **Sub**-процедуре, **Function**-процедуре или процедуре DLL-библиотеки.

### Синтаксис

[**Call**] *name* [*argumentlist*]

Синтаксис оператора **Call** состоит из элементов:

<b>Call</b>	Необязательный; (ключевое слово). Если указан, список аргументов <i>argumentlist</i> необходимо заключать в круглые скобки.
<i>name</i>	Обязательный. Имя вызываемой процедуры.
<i>argumentlist</i>	Необязательный. Список разделённых запятыми переменных, массивов или выражений для передачи вызываемой процедуре в качестве аргументов. Компоненты списка <i>argumentlist</i> могут включать ключевые слова <b>ByVal</b> или <b>ByRef</b> для описания способа передачи аргументов (по значению или по ссылке) процедуре.

При вызове процедуры ключевое слово **Call** использовать необязательно. Но, если вы используете **Call** для вызова процедуры, которая принимает аргументы, необходимо список аргументов заключать в круглые скобки. Если ключевое слово **Call** не используется, скобки вокруг списка аргументов также не нужны.

### Пример

В этом примере описаны три процедуры (с именами **a\_sub**, **b\_sub** и **c\_sub**). Процедура **a\_sub** вызывает процедуру **b\_sub** и передаёт ей в качестве аргумента строку. Процедура **b\_sub** вызывает процедуру **c\_sub** и передаёт ей в качестве аргумента принятую от процедуры **a\_sub** строку. Процедура **c\_sub** с помощью **MsgBox** выводит содержимое переданной из **b\_sub** строки.

```
Sub a_sub()
    Call b("Привет от процедуры a!")
End Sub

Sub b_sub(StrMy As String)
    c_sub (StrMy)
End Sub

Sub c_sub(StrMy As String)
    MsgBox StrMy
End Sub
```

---

**ChDir**

---

Изменяет текущий каталог или папку.

**Синтаксис**

**ChDir** *path*

Обязательный аргумент *path* — строковое выражение, которое определяет новый текущий каталог или папку. Аргумент *path* может включать имя драйвера. Если имя драйвера не указывается, используется текущий драйвер.

Оператор **ChDir** изменяет каталог по умолчанию, но не драйвер по умолчанию. Например, если драйвер по умолчанию — F, следующий оператор изменит каталог по умолчанию на драйвере E, но F останется драйвером по умолчанию:

```
ChDir "E:\TMP"
```

**Пример**

```
ChDir "MYDIR"
```

```
ChDir "D:\WIN2000\SYSTEM"
```

---

**ChDrive**

---

Изменяет текущий драйвер.

**Синтаксис**

**ChDrive** *drive*

Обязательный аргумент *drive* — строковое выражение, определяющее имеющийся драйвер. Если аргумент *drive* — строка из нескольких символов, оператор **ChDrive** использует только первую букву.

**Пример**

```
ChDrive "F"
```

---

**Close**

---

Закрывает ввод/вывод для файла, открытого с использованием оператора **Open**.

**Синтаксис**

**Close** [*filenumberlist*]

Необязательный аргумент *filenumberlist* может быть списком номеров файлов (*file numbers*) со следующим синтаксисом (*filenumber* — допустимый номер файла):

```
[[#]filenumber] [, [#]filenumber] ...
```

Если *filenumberlist* не указан, оператор **Close** закрывает все активные файлы, открытые оператором **Open**.

Когда вы закрываете файлы, открытые для **Output** или **Append**, окончательный буфер вывода записывается в буфер операционной системы для этого файла. Все буферное пространство, связанное с этим файлом, удаляется, и связь файла с его файловым номером прекращается.

**Пример**

В этом примере в цикле **For...Next** открываются три файла для записи, в каждый файл записывается текстовая строка; после окончания цикла при помощи оператора **Close** файлы закрываются.

```

Sub CloseTest()
    Dim I, FileName
    For I = 1 To 3
        FileName = "TEST" & I           ' создать имя файла
        Open FileName For Output As #I   ' открыть файл
        Print #I, "запись в файл " & FileName ' записать строку в файл
    Next I
    Close
    ' закрыть все три файла
End Sub

```

---

## Const

---

Объявляет константы.

### Синтаксис

[**Public** | **Private**] **Const** *constname* [**As** *type*] = *expression*

Синтаксис оператора **Const** включает три элемента:

- |                   |   |
|-------------------|---|
| <b>Public</b>     | Необязательный. Ключевое слово, используемое на уровне модуля для объявления констант, которые доступны всем процедурам во всех модулях. Нельзя использовать в процедурах.  |
| <b>Private</b>    | Необязательный. Ключевое слово, используемое на уровне модуля для объявления констант, которые доступны только в том модуле, в котором объявлены. Нельзя использовать в процедурах.                                       |
| <i>constname</i>  | Обязательный. Имя константы; должно отвечать стандартным соглашениям об именовании переменных.  |
| <i>type</i>       | Необязательный. Тип (данных) константы; может быть типом <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>String</b> или <b>Variant</b> . |
| <i>expression</i> | Обязательный. Литерал, другая константа или любая комбинация, включающая все арифметические или логические операции, кроме <b>Is</b> .  |

Константы по умолчанию являются закрытыми. В процедурах константы всегда являются закрытыми; их «видимость» не может быть изменена. В стандартных модулях видимость по умолчанию констант модульного уровня может быть изменена с использованием ключевого слова **Public**. Однако в модулях классов константы могут быть только закрытыми, их видимость нельзя изменить ключевым словом **Public**. Константы, объявляемые в **Sub**-, **Function**- или **Property**-процедурах, локальны для этих процедур. Константы, объявленные вне процедуры, «видимы» в модуле, в котором объявлены.

Для объединения нескольких объявлений констант в одной строке следует отделять их друг от друга запятыми.

### Пример

```

Const IntMyVar1 = 429
Public Const StrMyVar1 = "Word"
Private Const IntMyVar2 As Integer = 15
Const StrMyVar1 = "Excel", DblPi As Double = 3.14

```

---

## Date

---

Устанавливает системную текущую дату.

### Синтаксис

**Date** = *date*

Для системы Microsoft Windows 9X обязательный элемент синтаксиса *date* может быть в диапазоне от 1 января 1980 года до 13 декабря 2099 года. Для Microsoft Windows NT этот диапазон имеет верхнюю границу 31 декабря 2079 года. Для Macintosh *date* может принимать значения от 1 января 1904 года до 5 февраля 2040 года.

### Пример

В следующем примере объявляется константа **DateMy** и переменная **DateNow** для сохранения текущей системной даты. С помощью функции **Now** в переменной запоминается текущая дата. Затем оператор **Date** (с аргументом **DateMy**) изменяет системную дату, а **MsgBox** выдает в диалоговом окне подтверждение выполнения оператора **Date**. Далее оператор **Date** с аргументом **DateNow** восстанавливает системную дату, а **MsgBox** в диалоговом окне дает возможность убедиться в «восстановлении справедливости» по отношению к системной дате.

```
Sub DateTest()
    Const DateMy = #2/11/1986#
    Dim DateNow
    DateNow = Now()      'запомнить текущую дату (и время)
    Date = DateMy        'установить новую дату
    MsgBox Now()         'проверить изменение даты на новую
    Date = DateNow       'восстановить правильную дату
    MsgBox Now()         'проверить изменение даты сохраненную
End Sub
```

---

## Declare

---

Используется на уровне модуля для объявления ссылок на внешние процедуры в DLL-библиотеке.

### Синтаксис 1

```
[Public | Private] Declare Sub name Lib <libname>
[Alias <aliasname>] [(arglist)]
```

### Синтаксис 2

```
[Public | Private] Declare Function name Lib <libname>
[Alias <aliasname>] [(arglist)] [As type]
```

Синтаксис оператора **Declare** включает следующие элементы:

- |                 |   |
|-----------------|---|
| <b>Public</b>   | Необязательный. Используется для объявления процедур, которые доступны всем другим процедурам во всех модулях.  |
| <b>Private</b>  | Необязательный. Используется для объявления процедур, которые доступны только внутри модуля, содержащего эти объявления.  |
| <b>Sub</b>      | Необязательный (должно быть либо <b>Sub</b> , либо <b>Function</b> ). Указывает, что процедура не возвращает значения.  |
| <b>Function</b> | Необязательный (должно быть либо <b>Sub</b> , либо <b>Function</b> ). Указывает, что процедура возвращает значение, которое может быть использовано в каком-либо выражении. |

<i>name</i>	Обязательный. Любое допустимое имя процедуры.
<b>Lib</b>	Обязательный. Указывает, что DLL- или код-источник содержит процедуру, которая объявляется.
<i>libname</i>	Обязательный. Имя DLL- или код-источника, который содержит объявляемую процедуру.
<b>Alias</b>	Необязательный. Указывает, что вызываемая процедура имеет другое имя в DLL. Это полезно, когда имя внешней процедуры совпадает с ключевым словом или когда имя DLL-процедуры совпадает с <i>public</i> -переменной, константой или именем другой процедуры в той же области действия. <b>Alias</b> также используется, если некоторые символы в имени DLL-процедуры недопустимы стандартными соглашениями об DLL-именовании.
<i>aliasname</i>	Необязательный. Имя процедуры в DLL- или код-источнике. Если первый символ не является символом #, <i>aliasname</i> — имя точки входа в процедуру в DLL. Если # — первый символ, все последующие символы указывают порядковый номер точки входа в процедуру.
<i>arglist</i>	Необязательный. Список переменных, представляющих аргументы, передаваемые в процедуру при вызове.
<i>type</i>	Необязательный. Тип значения, возвращаемого <b>Function</b> -процедурой; может быть <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>String</b> (только переменной длины), <b>Variant</b> , типом, определенным пользователем, или объектным типом.

Синтаксис аргумента *arglist* состоит из следующих элементов:

[**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *varname*[( )] [**As** *type*]

<b>Optional</b>	Необязательный. Указывает, что аргумент — необязательный. Если используется, все последующие аргументы в <i>arglist</i> должны быть также необязательными и объявляться с ключевым словом <b>Optional</b> . Если используется <b>ParamArray</b> слово <b>Optional</b> не может использоваться ни для какого аргумента.
<b>ByVal</b>	Необязательный. Указывает, что аргумент передается по значению.
<b>ByRef</b>	Указывает, что аргумент передается по ссылке (в Visual Basic по умолчанию).
<b>ParamArray</b>	Необязательный. Используется только в качестве последнего аргумента в списке <i>arglist</i> для указания того, что последний аргумент является <b>Optional</b> -массивом <b>Variant</b> -элементов. Ключевое слово <b>ParamArray</b> позволяет использовать переменное число аргументов и не может применяться с <b>ByVal</b> , <b>ByRef</b> или <b>Optional</b> .
<i>varname</i>	Обязательный. Имя переменной, передаваемой процедуре; должно удовлетворять стандартным соглашениям об именовании переменных.
( )	Обязательны для переменных-массивов. Указывают, что <i>varname</i> — массив.



*type* Необязательный. Тип аргумента, передаваемого процедуре; может быть **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String** (только переменной длины), **Object**, **Variant**, типом, определенным пользователем, или объектным типом.

Для **Function**-процедур тип данных процедуры определяет тип возвращаемого значения. Пустые скобки указывают, что **Sub**- или **Function**-процедура не имеет аргументов. Если процедура имеет список аргументов, их тип и количество проверяются при вызове процедуры.

**Deftype**

Используется на уровне модуля для определения типа по умолчанию переменных, аргументов, передаваемых процедурам, и возвращаемого типа **Function**- и **Property Get**-процедур, имена которых начинаются с определенных символов.

**Синтаксис**

```
DefBool letterrange[, letterrange] ...
DefByte letterrange[, letterrange] ...
DefInt letterrange[, letterrange] ...
DefLng letterrange[, letterrange] ...
DefCur letterrange[, letterrange] ...
DefSng letterrange[, letterrange] ...
DefDb1 letterrange[, letterrange] ...
DefDec letterrange[, letterrange] ...
DefDate letterrange[, letterrange] ...
DefStr letterrange[, letterrange] ...
DefObj letterrange[, letterrange] ...
DefVar letterrange[, letterrange] ...
```

Обязательный аргумент *letterrange* имеет следующий синтаксис:

```
letter1[-letter2]
```

Аргументы *letter1* и *letter2* определяют диапазон имен, для которого устанавливается тип данных по умолчанию. Каждый аргумент представляет первый символ имени переменной, аргумента, **Function**- или **Property Get**-процедуры и может быть любым алфавитным символом.

Имя оператора определяет тип данных:

оператор	тип данных	оператор	тип данных
DefBool	Boolean	DefDb1	Double
DefByte	Byte	DefDate	Date
DefInt	Integer	DefStr	String
DefLng	Long	DefObj	Object
DefCur	Currency	DefVar	Variant
DefSng	Single		

Если в коде указывается буквенный диапазон, он обычно определяет тип данных для переменных, которые начинаются с букв первых 128-и символов символического набора. Однако когда задается буквенный диапазон A–Z, устанавливается значение по умолчанию на определенный тип данных для всех переменных, включая переменные, которые начинаются с международных символов из расширенной части символического набора (128–255).

После того как диапазон A-Z определен, можно в дальнейшем переопределять любой поддиапазон переменных, используя операторы **Deftype**. Если в следующем операторе **Deftype** окажется символ из ранее используемого диапазона, это вызовет ошибку. Однако при этом можно, используя оператор **Dim** со спецификатором **As type**, явно указать для некоторой переменной ее тип. Например, если вы указали, что переменные, начинающиеся с символов диапазона i-n, по умолчанию будут иметь тип **Integer** (именно эти символы очень часто используются для именования переменных циклов **For...Next**), для конкретной переменной **iTaxRate** можно указать тип **Double**:

```
DefInt A-Z
Dim iTaxRate As Double
```

Оператор **Deftype** не оказывает воздействия на элементы типов, определенных пользователем, поскольку эти элементы должны объявляться явно.

### Пример

```
DefInt I-K
DefStr L-Z
```

---

## DeleteSetting

---

Удаляет секцию или ключевую настройку (key setting) из раздела приложения в Windows-реестре.

### Синтаксис

```
DeleteSetting appname, section[, key]
```

Синтаксис оператора **DeleteSetting** включает следующие именованные аргументы:

<i>appname</i>	Обязательный. Строковое выражение, содержащее наименование приложения или проекта (project), для которого предназначена эта секция или ключевая настройка.
<i>section</i>	Обязательный. Строковое выражение, содержащее наименование секции или ключевой настройки для удаления. Если указываются и <i>appname</i> , и <i>section</i> , то удаляется заданная секция и все связанные с ней настройки.
<i>key</i>	Необязательный. Строковое выражение, содержащее наименование ключевой настройки для удаления.

### Пример

Далее приведены две процедуры, одна из которых (**TestSave**) создает секцию и ключевые настройки в Реестре, а другая (**TestDelete**) удаляет всю секцию. Выполните первую процедуру и запустите программу **Regedit.exe** (из меню Пуск). Найдите секцию **MyApp** и посмотрите установленные ключи. Затем запустите процедуру **TestDelete** и убедитесь в том, что секция **MyApp** действительно удалена.

```
Sub TestSave()
' поместить некоторые настройки в Реестр
SaveSetting appname:="MyApp", Section:="Startup", _
    Key:="Top", setting:=75
SaveSetting "MyApp", "Startup", "Left", 50
End Sub

Sub TestDelete()
' удалить секцию из Реестра
```

```
DeleteSetting "MyApp"
End Sub
```

---

## Dim

---

Объявляет переменные и резервирует память.

### Синтаксис

```
Dim [WithEvents] varname([[subscripts]]) [As [New] type] [,
[WithEvents] varname([[subscripts]]) [As [New] type]] ...
```

Синтаксис оператора **Dim** включает следующие элементы:

**WithEvents** Необязательный. Ключевое слово, которое указывает, что *varname* — объектная переменная, используемая для реакции на событие, инициированное ActiveX-объектом. **WithEvents** допускается только в модулях класса. Отдельных переменных с использованием ключевого слова **WithEvents** можно объявлять столько, сколько необходимо, но массив, используя его, создавать нельзя. Также нельзя использовать ключевое слово **New** при наличии **WithEvents**.

*varname* Обязательный. Имя переменной; должно соответствовать стандартным соглашениям об именовании переменных.

*subscripts* Необязательный. Размерность переменной-массива; объявить можно до 60 размерностей. Аргумент *subscripts* использует следующий синтаксис:

```
[lower To] upper [, [lower To] upper] ...
```

*lower* — нижний предел допустимых индексов массива; *upper* — верхний предел (является обязательным) для индексов массива. При наличии в операторе **Dim** только *upper* нижний предел определяется в зависимости от установки оператора **Option Base**.

**New** Необязательный. Ключевое слово, позволяющее неявно создать объект. Если вы используете **New** при объявлении объектной переменной, создается новый экземпляр объекта, поэтому нет необходимости использовать оператор **Set** для назначения объекту ссылки.

*Type* Необязательный. Тип данных переменной; может быть **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String** (только переменной длины), **String \* length** (для строк фиксированной длины), **Object**, **Variant**, типом, определенным пользователем, или объектным типом.

Переменные, объявляемые оператором **Dim** на уровне модуля, доступны всем процедурам этого модуля. На уровне процедуры переменные доступны только внутри этой процедуры. Оператор **Dim** следует использовать на уровне модуля или процедуры для объявления типа переменной. Например, следующая переменная объявляется как **String**:

```
Dim StrName As String
```

Оператор **Dim** можно использовать и для объявления переменной объектного типа. В следующем примере объявляется новый экземпляр объекта **Worksheet**:

```
Dim WrkShNew As New Worksheet
```

Если ключевое слово **New** не используется при объявлении объектной переменной, то переменной, которая ссылается на объект, должна быть присвоена ссылка на существующий объект с использованием оператора **Set**. До тех пор, пока этого не сделано, объектная переменная будет иметь значение **Nothing**, которое указывает, что объектная переменная не ссылается ни на какой конкретный экземпляр объекта.

Оператор **Dim** можно использовать с пустыми круглыми скобками для объявления динамического массива. Используя затем оператор **ReDim**, можно указать размерность и число элементов массива.

### Do...Loop

Повторяет блок операторов до тех пор, пока некоторое условие имеет значение **True** или пока оно не станет равным **True**.

#### Синтаксис

```
Do [{While | Until} condition]
    [statements]
    [Exit Do]
    [statements]
Loop
```

или:

```
Do
    [statements]
    [Exit Do]
    [statements]
Loop [{While | Until} condition]
```

Синтаксис оператора **Do Loop** включает следующие элементы:

**condition**      Необязательный. Численное или строковое выражение, имеющее значение **True** или **False**. Если **condition** имеет значение **Null**, оно интерпретируется как **False**.

**statements**      Один или более операторов, которые повторяются, пока **condition** — **True**, или до тех пор пока оно не станет равным **True**.

В теле оператора (цикла) может быть любое количество операторов **Exit Do**, которые являются дополнительными средствами для окончания цикла (выхода из оператора **Do...Loop**). Оператор **Exit Do** часто используется внутри **Do...Loop** с применением оператора **If...Then**, который проверяет некоторое дополнительное условие для выхода из цикла.

### End

Заканчивает процедуру или блок.

#### Синтаксис

```
End
End Function
End If
End Property
End Select
End Sub
End Type
End With
```

Синтаксис оператора **End** имеет следующие формы:

<b>End</b>	Немедленно прекращает выполнение. Не является обязательным, но может располагаться в любом месте процедуры для окончания работы этой процедуры, закрытия файлов, открытых оператором <b>Open</b> , и «очистки» (обнуление в соответствии с типом) переменных.
<b>End Function</b>	Обязателен для окончания <b>Function</b> -процедуры.
<b>End If</b>	Обязателен для окончания блока оператора <b>If...Then...Else</b> .
<b>End Property</b>	Обязателен для окончания <b>Property Let</b> -, <b>Property Get</b> - или <b>Property Set</b> -процедуры.
<b>End Select</b>	Обязателен для окончания оператора <b>Select Case</b> .
<b>End Sub</b>	Обязателен для окончания <b>Sub</b> -процедуры.
<b>End Type</b>	Обязателен для окончания определения пользовательского типа (оператора <b>Type</b> ).
<b>End With</b>	Обязателен для окончания оператора <b>With</b> .

При выполнении оператор **End** «переустанавливает» все переменные уровня модуля и все статические локальные переменные во всех модулях. Чтобы сохранить значения этих переменных используйте оператор **Stop**.

Оператор **End** останавливает выполнение кода аварийно, без вызова событий (и соответствующих им программ обработки) **Unload**, **QueryUnload** или **Terminate**, а также какого-либо другого кода. Объекты, созданные с помощью модулей класса, уничтожаются, файлы, открытые оператором **Open**, закрываются, память, используемая программой, освобождается.

### Пример

В этом примере при загрузке формы с помощью функции **InputBox** запрашивается пароль. Если пароль введен неправильно, программа завершает работу. Конечно, этот пример следует использовать только в учебных целях, так как пароль, находящийся рядом с кодом его запроса, легко извлечь.

```
Sub Form_Load
    Dim strPassword, StrPw
    strPassword = "Привет Шишкину!"
    StrPw = InputBox("Введите пароль:")
    If StrPw <> strPassword Then
        MsgBox "К сожалению, пароль неверен!"
    End If
    'полезные операторы процедуры:
    ...
End Sub
```

---

### Enum

---

Объявляет тип для перечисления.

**Синтаксис**

```
[Public | Private] Enum name
membername [= constantexpression]
membername [= constantexpression]
.
.
End Enum
```

Синтаксис оператора **Enum** включает следующие элементы:

<b>Public</b>	Необязательный. Указывает, что тип <b>Enum</b> является видимым во всем проекте. Типы <b>Enum</b> являются <b>Public</b> по умолчанию.
<b>Private</b>	Необязательный. Указывает, что тип <b>Enum</b> является видимым только внутри модуля, в котором он определяется.
<i>name</i>	Обязательный. Имя типа <b>Enum</b> . Имя должно быть допустимым для Visual Basic идентификатором.
<i>membername</i>	Обязательный. Допустимый для Visual Basic идентификатор, определяющий имя, посредством которого будет известен составной элемент типа <b>Enum</b> .
<i>constantexpression</i>	Необязательный. Значение элемента (приводится к <b>Long</b> ). Если <i>constantexpression</i> не указан, этому значению присваивается либо ноль (если это первый <i>membername</i> ), либо на единицу больше, чем значение непосредственно предыдущего <i>membername</i> .

Переменные перечисления — это переменные, объявленные при помощи оператора **Enum**. В качестве типа **Enum** могут быть объявлены как переменные, так и аргументы. Элементы типа **Enum** инициализируются константными значениями. Эти значения не могут быть изменены в режиме исполнения программы и могут включать как положительные, так и отрицательные значения.

Оператор **Enum** можно использовать только на уровне модуля. Сразу после описания **Enum**-типа этот тип можно использовать для объявления переменных, параметров процедур или типов данных, возвращаемых процедурами. **Public Enum**-типы в модуле класса не являются членами этого класса, хотя они записываются в библиотеку типов. Типы **Enum**, определенные в стандартных модулях, записываются в библиотеки типов. Типы **Public Enum** с одним и тем же именем не могут определяться и в стандартном, и модуле класса, поскольку они совместно используют (разделяют) одно и то же пространство имен. Если два типа **Enum** в различных библиотеках типов имеют одно и то же имя, но различные элементы, ссылка на переменную типа зависит от того, какая библиотека типов имеет более высокий приоритет в **References**.

---

**Erase**

Повторно инициализирует элементы фиксированного массива и освобождает память, занимаемую элементами динамического массива.

**Синтаксис**

```
Erase arraylist
```

Обязательный аргумент *arraylist* — один (или более разделенных запятыми) массив.

Оператор **Erase** действует по разному для массивов фиксированного и переменного размера (динамических). Массивы фиксированного размера не уничтожаются: занимаемая ими память не освобождается. Оператор **Erase** устанавливает элементы массивов фиксированного размера следующим образом:

Тип массива	Действие оператора <b>Erase</b>
Фиксированный массив чисел	Устанавливает каждый элемент массива в ноль.
Фиксированный массив строк (переменной длины)	Присваивает каждому элементу массива значение строки нулевой длины ("").
Фиксированный массив строк (фиксированной длины)	Устанавливает каждый элемент массива в ноль.
Фиксированный <b>Variant</b> -массив	Устанавливает каждый элемент массива в <b>Empty</b> .
Массив данных определенных пользователем типов	Устанавливает каждый элемент, как-будто это — отдельная переменная.
Массив объектов	Присваивает каждому элементу значение <b>Nothing</b> .

Оператор **Erase** освобождает память, занимаемую динамическим массивом. Прежде чем ваша программа сможет снова ссылаться на динамический массив, необходимо снова объявить массив с использованием оператора **ReDim**.

## Error

Моделирует появление ошибки.

### Синтаксис

**Error** *errornumber*

Обязательный аргумент *errornumber* может быть любым допустимым номером ошибки.

Оператор **Error** поддерживается для обратной совместимости. При разработке нового кода (особенно при создании объектов) для генерации run-time-ошибок используйте метод **Raise** объекта **Err**.

Если *errornumber* указывается, оператор **Error** вызывает обработчик ошибок (error handler) после того, как свойства **Err**-объекта принимают следующие значения по умолчанию:

<b>Number</b>	Значение, определенное как аргумент для оператора <b>Error</b> . Может быть любым допустимым номером ошибки.
<b>Source</b>	Наименование текущего проекта Visual Basic.
<b>Description</b>	Строковое выражение, относящееся к возвращаемому значению функции <b>Error</b> для определенного <b>Number</b> , если эта строка существует. Если строка не существует, <b>Description</b> содержит строку нулевой длины ("").
<b>HelpFile</b>	Драйвер, путь и имя соответствующего Help-файла Visual Basic.
<b>HelpContext</b>	ID соответствующего Help-файла Visual Basic.
<b>LastDLLError</b>	Ноль.

---

**Event**


---

Объявляет определенное пользователем событие.

**Синтаксис**

**[Public] Event** *procedurename* [(*arglist*)]

Синтаксис оператора **Event** включает следующие элементы:

**Public** Необязательный. Определяет «видимость» события во всем проекте (используется по умолчанию).

*procedurename* Обязательный. Имя события; должно соответствовать стандартным соглашениям об именовании переменных.

Аргумент *arglist* имеет следующий синтаксис:

**[ByVal | ByRef]** *varname*[()] [**As** *type*]

**ByVal** Необязательный. Указывает, что аргумент передается по значению.

**ByRef** Необязательный. Указывает, что аргумент передается по ссылке (по умолчанию).

*varname* Обязательный. Имя переменной — аргумент, передаваемый процедуре; должно соответствовать стандартным соглашениям об именовании переменных.

*Type* Необязательный. Тип (данных) аргумента, передаваемого процедуре; может быть **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String** (только переменной длины), **Object**, **Variant**, типом, определенным пользователем, или объектным типом.

После объявления события для его инициализации используется оператор **RaiseEvent**:

```
Event LogonCompleted (UserName as String)

Sub
    RaiseEvent LogonCompleted("AntoineJan")
End Sub
```

Аргументы для событий можно создавать так же, как это делается для обычных процедур, хотя имеются следующие исключения: события не могут иметь именованные аргументы, **Optional**-аргументы или **ParamArray**-аргументы. События не возвращают значение.

---

**Exit**


---

Прерывает выполнение блока **Do...Loop**, **For...Next**, **Function**-, **Sub**- или **Property**-кода.

**Синтаксис**

```
Exit Do
Exit For
Exit Function
Exit Property
Exit Sub
```



Синтаксис оператора **Exit** имеет следующие формы:

- Exit Do** Предназначен для прекращения работы оператора **Do...Loop** и может использоваться только внутри этого оператора. Оператор **Exit Do** «передает управление» выполнением программы оператору, следующему за словом **Loop** оператора **Do...Loop**. Если **Exit Do** используется во вложенных операторах **Do...Loop**, то «управление передается» во внешний **Do...Loop** по отношению к тому, в котором встретился оператор **Exit Do**.
- Exit For** Обеспечивает способ выхода из цикла **For**. Может использоваться в циклах **For...Next** или **For Each...Next**. Оператор **Exit For** «передает управление» выполнением программы оператору, следующему за словом **Next**. При использовании внутри вложенного цикла **For** оператор **Exit For** «передает управление» во внешний **For**-цикл по отношению к тому, в котором встретился оператор **Exit For**.
- Exit Function** Немедленно прекращает выполнение **Function**-процедуры, в которой встречается. «Управление передается» оператору, следующему за тем, который вызвал **Function**-процедуру.
- Exit Property** Немедленно прекращает выполнение **Property**-процедуры, в которой встречается. «Управление передается» оператору, следующему за тем, который вызвал **Property**-процедуру.
- Exit Sub** Немедленно прекращает выполнение **Sub**-процедуры, в которой он находится. «Управление передается» оператору, следующему за тем, который вызвал **Sub**-процедуру.

Не следует путать **Exit** и **End** операторы. Оператор **Exit** не определяет конец структуры.

#### Пример

```
Sub ExitStatementTest()  
Dim I, MyNum  
Do  
    For I = 1 To 1000  
        MyNum = Int(Rnd * 1000)  
        Select Case MyNum  
            Case 7: Exit For  
            Case 29: Exit Do  
            Case 54: Exit Sub  
        End Select  
    Next I  
Loop  
End Sub
```

---

## FileCopy

---

Копирует файл.

#### Синтаксис

**FileCopy** *source*, *destination*

Синтаксис оператора **FileCopy** включает следующие именованные аргументы:

- Source** Обязательный. Строковое выражение, определяющее имя файла, который будет копироваться; может включать каталог (папку) и драйвер диска.

**Destination** Обязательный. Строковое выражение, определяющее имя файла, в который будет производиться копирование.

Оператор **FileCopy** нельзя использовать для текущего открытого файла.

### Пример

```
FileCopy "File1", "File2"
```

## For Each...Next

Повторяет группу операторов для каждого элемента массива или коллекции.

### Синтаксис

```
For Each element In group
    [statements1]
    [Exit For]
    [statements2]
Next [element]
```

Синтаксис оператора **For...Each...Next** включает следующие элементы:

- element** Обязательный. Переменная, используемая для итерации по элементам коллекции или массива. Для коллекций *element* может быть только **Variant**-переменной, обобщенной объектной переменной или определенной объектной переменной. Для массива *element* может быть только **Variant**-переменной.
- group** Обязательный. Имя коллекции объектов или массива (исключая массив типов, определенных пользователем).
- statements** Необязательный. Один или более операторов, которые выполняются для каждого элемента в *group*.

Блок **For...Each** выполняется, если в *group* имеется, по крайней мере, один элемент. Все операторы цикла выполняются для каждого элемента в *group*. После этого «управление передается» оператору, следующему за ключевым словом **Next**.

### Пример

```
Sub ForEachtest()
    Dim a() As Integer, i As Integer, j As Integer
    i = 1
    Do While Not i = 0
        i = Int(InputBox("Введите целое число"))

        If i > 0 Then
            ReDim Preserve a(i)
            For j = 1 To i
                a(j) = j
            Next
            Call PrintArray(a)
        End If

        Loop
    End Sub

    Sub PrintArray(aa)
        'для печати массива aa не нужно знать его размер
```

```

For Each dd In aa
    Debug.Print dd
Next
Debug.Print "-----"
End Sub

```

---

## For...Next

---

Повторяет группу операторов определенное число раз.

### Синтаксис

```

For counter = start To end [Step step]
    [statements1]
    [Exit For]
    [statements2]
Next [counter]

```

Синтаксис оператора **The For...Next** содержит следующие элементы:

<i>counter</i>	Обязательный. Числовая переменная, используемая в качестве счетчика цикла (loop counter). Тип этой переменной не может быть типом <b>Boolean</b> или элементом массива.
<i>start</i>	Обязательный. Начальное значение для <i>counter</i> .
<i>end</i>	Обязательный. Конечное значение для <i>counter</i> .
<i>step</i>	Необязательный. Шаг изменения значения <i>counter</i> после каждого цикла. Если не указывается, по умолчанию используется значение 1.
<i>statements</i>	Необязательный. Один или более операторов, выполняемых определенное число раз.

Аргумент *step* может быть как положительным, так и отрицательным. После выполнения всех операторов в цикле значение *step* добавляется к значению *counter*. После этого либо все операторы цикла выполняются снова, либо цикл прекращается и «управление передается» следующему за словом **Next** оператору. Не следует пытаться изменять значение *counter* внутри цикла. Это может затруднить чтение и отладку кода.

В качестве альтернативных выходов из цикла **For...Next** можно внутри цикла располагать любое количество операторов **Exit For**.

### Пример

```

Sub Fortest()
    Dim a(33) As Integer, j As Integer
    For j = 1 To 33
        a(j) = j
    Next
End Sub

```

---

## Function

---

Объявляет имя, аргументы и код тела **Function**-процедуры.

### Синтаксис

```

[Public | Private | Friend] [Static] Function name [(arglist)] [As type]
    [statements1]
    [name = expression1]

```

```
[Exit Function]
[statements2]
[name = expression2]
End Function
```

Синтаксис оператора **Function** включает следующие элементы:

<b>Public</b>	Необязательный. Указывает, что <b>Function</b> -процедура доступна всем другим процедурам во всех модулях. Если используется в модуле, который содержит <b>Option Private</b> , процедура недоступна вне этого проекта.
<b>Private</b>	Необязательный. Указывает, что <b>Function</b> -процедура доступна другим процедурам только в модуле, в котором объявлена.
<b>Friend</b>	Необязательный. Используется только в модуле класса. Указывает, что <b>Function</b> -процедура доступна всем модулям проекта.
<b>Static</b>	Необязательный. Указывает, что локальные переменные <b>Function</b> -процедуры сохраняются между вызовами процедуры. Атрибут <b>Static</b> не действует на переменные, объявленные вне <b>Function</b> , даже если они используются в коде процедуры.
<i>name</i>	Обязательный. Имя <b>Function</b> -процедуры; должно отвечать стандартным соглашениям об именовании переменных.
<i>arglist</i>	Необязательный. Список переменных, представляющих передаваемые процедуре аргументы.
<i>type</i>	Необязательный. Тип данных, возвращаемый <b>Function</b> -процедурой; может быть <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>String</b> , <b>Object</b> , <b>Variant</b> или типом, определенным пользователем.
<i>statements</i>	Необязательный. Любая группа операторов, которая выполняется при вызове <b>Function</b> -процедуры.
<i>expression</i>	Необязательный. Возвращаемое значение <b>Function</b> -процедуры.

Аргумент *arglist* должен соответствовать следующему синтаксису:

```
[Optional] [ByVal | ByRef] [ParamArray] varname[( )] [As type] [= defaultvalue]
```

<b>Optional</b>	Необязательный. Указывает, что аргумент необязательный. Если используется, все последующие аргументы в <i>arglist</i> должны быть также необязательными и объявляться с ключевым словом <b>Optional</b> . Если применяется <b>ParamArray</b> , слово <b>Optional</b> не может использоваться ни для какого аргумента.
<b>ByVal</b>	Необязательный. Указывает, что аргумент передается по значению.
<b>ByRef</b>	Необязательный. Указывает, что аргумент передается по ссылке. Используется по умолчанию.
<b>ParamArray</b>	Необязательный. Используется только в качестве последнего аргумента в списке <i>arglist</i> для указания того, что последний аргумент является <b>Optional</b> -массивом <b>Variant</b> -элементов. Ключевое слово <b>ParamArray</b> позволяет использовать переменное число аргументов и не может применяться с <b>ByVal</b> , <b>ByRef</b> или <b>Optional</b> .

<i>varname</i>	Обязательный. Имя переменной, передаваемой процедуре; должно удовлетворять стандартным соглашениям об именовании переменных.
<i>type</i>	Необязательный. Тип аргумента, передаваемого процедуре; может быть <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>String</b> (только переменной длины), <b>Object</b> , <b>Variant</b> или определенным объектным типом. Если параметр не описан как <b>Optional</b> , может быть указан тип, определенный пользователем.
<i>defaultvalue</i>	Необязательный. Любая константа или константное выражение. Допускается только для аргументов, описанных как <b>Optional</b> . Если типом является <b>Object</b> , то явное значение по умолчанию может быть только значением <b>Nothing</b> .

Если явно не определено использование **Public**, **Private** или **Friend**, **Function**-процедура является **Public** по умолчанию. Если не используется **Static**, значения локальных переменных не сохраняются между вызовами. Ключевое слово **Friend** может быть использовано только в модулях класса. Однако **Friend**-процедуры могут быть доступны процедурам в любом модуле проекта.

**Function**-процедуры могут быть рекурсивными. Это означает, что они могут вызывать сами себя. Иногда в результате рекурсивных вызовов может происходить переполнение программного стека. Ключевое слово **Static** обычно не используется при описании рекурсивных **Function**-процедур.

Весь выполняемый код помещается в теле **Function**-процедуры. Нельзя определить **Function**-процедуру внутри другой **Function**-, **Sub**- или **Property**-процедуры.

Оператор **Exit Function** приводит к немедленному завершению **Function**-процедуры. При этом программа продолжает выполняться с оператора, следующего за тем, который вызвал **Function**-процедуру. В качестве альтернативных выходов внутри **Function**-процедуры может встречаться любое количество операторов **Exit Function**.

Подобно **Sub**-процедуре **Function**-процедура — это отдельная процедура, которая может принимать аргументы, выполнять ряд операторов и изменять значения ее аргументов. Однако в отличие от **Sub**-процедуры выражение с вызовом **Function**-процедуры может появляться в правой части операции присваивания в составе некоторого выражения. Например, как функции **Chr**, **Cos** и так далее. Для вызова функции используется ее имя, за которым в круглых скобках помещаются значения аргументов функции.

Чтобы функция возвращала значение, необходимо в теле (любом месте) функции поместить оператор, присваивающий это значение имени функции. Причем таких операторов может быть несколько. Если имени функции не присваивается внутри функции никакого значения, функция возвращает значение по умолчанию: функция, объявленная как числовая, возвращает 0, строковая функция — строку нулевой длины, **Variant**-функция — значение **Empty**, а объектная функция — значение **Nothing**.

### Пример

В этом примере приведена функция проверки наличия файла на диске (**IsDiskFile**) и процедура, тестирующая эту функцию.

```
Function IsDiskFile(fName As String) As Boolean
' возвращает Истина (True), если файл на диске найден,
' иначе - Ложь (False)
If (Dir(fName) <> "") Then
```

```

        IsDiskFile = True
    Else
        IsDiskFile = False
    End If
End Function

Sub TestIsDiskFile()
    Dim namefun As String
    namefun = InputBox("Введите имя файла")
    If IsDiskFile(namefun) Then
        MsgBox ("Файл на диске!")
    Else
        MsgBox ("Файл на диске не найден!")
    End If
End Sub

```

---

## Get

---

Считывает данные из открытого файла на диске и помещает их в переменную.

### Синтаксис

**Get** [#]*filenumber*, [*recnumber*], *varname*

Синтаксис оператора **Get** включает следующие элементы:

<i>filenumber</i>	Обязательный. Любое допустимое имя файла.
<i>recnumber</i>	Необязательный. <b>Variant (Long)</b> . Номер записи (record number) [для режима <b>Random</b> ] или номер байта (byte number) [для режима <b>Binary</b> ], с которого начинается чтение данных.
<i>varname</i>	Обязательный. Допустимое имя переменной, в которую считываются данные.

Данные, считываемые оператором **Get**, обычно предварительно записываются посредством оператора **Put**. Первая запись (или байт) в файле находится в позиции 1, вторая — в позиции 2 и так далее. Если аргумент *recnumber* опускается, происходит считывание следующей записи (или байта) после той, с которой использовались операторы **Get** или **Put** (или указанной функцией **Seek**). Не следует забывать, что для пропуска аргумента нужно использовать две запятые:

```
Get #2,,StrBuffer
```

Для файлов, открытых в **Random**-режиме, применяются следующие правила:

- Если длина считываемых данных меньше, чем длина, указанная в опции **Len** оператора **Open**, **Get** считывает последующие записи на границах длин записей. Пространство между концом одной записи и началом следующей записи заполняется имеющимся содержимым файлового буфера.
- Если считываемая переменная является строкой переменной длины, оператор **Get** сначала считывает 2-байтовый дескриптор (descriptor), содержащий длину строки, а затем — сами данные, которые помещаются в переменную. Поэтому длина записи (record), определяемая опцией **Len** в операторе **Open**, должна быть, по крайней мере, на 2 байта больше, чем действительная длина строки.

- Если считываемая переменная — **Variant** числового типа, **Get** считывает 2 байта, идентифицирующие **VarType**, а затем — сами данные, которые помещаются в переменную. Длина записи, определяемая опцией **Len** в операторе **Open**, должна быть, по крайней мере, на 2 байта больше, чем действительное число байтов, необходимое для сохранения переменной.
- Если считываемая переменная — **Variant**, значение **VarType** которой равно 8 (**String**), оператор **Get** считывает 2 байта, идентифицирующие **VarType**, 2 байта, указывающие длину строки, а затем — сами строковые данные. Длина записи, определяемая опцией **Len** в операторе **Open** должна быть, по крайней мере, на 4 байта больше, чем действительная длина строки.
- Если считываемая переменная — динамический массив, оператор **Get** считывает дескриптор, длина которого равна  $2 \text{ «плюс» } 8$ , умноженное на число размерностей массива ( $2 + 8 * \text{NumberOfDimensions}$ ). Длина записи, определяемая опцией **Len** в операторе **Open**, должна быть больше или равна сумме количества байтов, необходимых для хранения массива, и длины дескриптора. Например, объявленный ниже массив требует при записи на диск 218 байтов —  $18 (2 + 8 * 2)$  байтов занимает дескриптор и  $200 (5 * 20 * 2)$  необходимо для хранения массива:

```
Dim MyArray(1 To 5, 1 To 20) As Integer
```

- Если считываемая переменная — массив фиксированного размера, оператор **Get** считывает только данные.
- Если считываемая переменная — переменная любого другого типа (не строка переменной длины или **Variant**), **Get** считывает только данные. Длина записи, определяемая опцией **Len** в операторе **Open**, должна быть больше или равна длине считываемых данных.
- Оператор **Get** считывает элементы типов, определенных пользователем, как если бы они считывались отдельно, за исключением того, что между ними нет заполняющих элементов. На диске динамический массив определенного пользователем типа (записанный оператором **Put**) предваряется дескриптором, длина которого равна  $2 \text{ «плюс» } 8$ , умноженное на число размерностей:  $2 + 8 * \text{NumberOfDimensions}$ .

Длина записи, определяемая опцией **Len** в операторе **Open**, должна быть больше или равна сумме всех байтов, необходимых для чтения отдельных элементов, включая любые массивы и их дескрипторы.

Для файлов, открытых в **Binary**-режиме, применяются все **Random**-правила, за исключением:

- **Len**-опция (параметр) оператора **Open** не действует. Оператор **Get** считывает все переменные с диска непрерывно, т.е. без заполняющих символов между записями.
- Для любого массива, кроме массива пользовательского типа, **Get** считывает только данные. Дескриптор не читается.
- **Get** считывает строки переменной длины, которые не являются элементами пользовательских типов, без дескриптора 2-байтовой длины. Число считываемых байтов равно числу символов, имеющихся в строке.

### Пример

В следующем примере описывается пользовательский тип данных (**Record**), процедура записи (**TestPut**) данных в файл и процедура чтения (**TestGet**) данных из файла.

```

Type Record                                ' Определенный пользователем тип
    ID As Integer
    FirstName As String * 20
    LastName As String * 20
End Type

Sub TestPut()
    'Тестирование Put
    Dim MyRecord As Record, RecordNumber

    Kill "TESTFILE"
    Open "TESTFILE" For Random As #1 Len = Len(MyRecord)

    MyRecord.FirstName = "Петр"
    MyRecord.LastName = "Петров"
    MyRecord.ID = 333
    Put #1, 2, MyRecord

    MyRecord.FirstName = "Сидор"
    MyRecord.LastName = "Сидоров"
    MyRecord.ID = 111
    Put #1, 3, MyRecord

    MyRecord.FirstName = "Иван"
    MyRecord.LastName = "Иванов"
    MyRecord.ID = 222
    Put #1, 1, MyRecord

    Close #1
End Sub

Sub TestGet()
    'тестирование Get
    Dim MyRecord As Record, Position
    Open "TESTFILE" For Random As #1 Len = Len(MyRecord)
    For i = 1 To 3
        Get #1, i, MyRecord
        Debug.Print MyRecord.FirstName & " : " & MyRecord.ID
    Next
    Close #1
End Sub

```

---

### GoSub...Return

---

Передаёт (и возвращает) управление выполнением процедуры подпрограмме (subroutine) внутри процедуры.

#### Синтаксис

**GoSub** *line*

[statements of procedure]

*line*

[statements of subroutine]

**Return**

Аргумент *line* может быть любой строковой или числовой меткой.

Операторы **GoSub** и **Return** можно использовать в любом месте процедуры, но **GoSub** и соответствующий ему оператор **Return** должны быть в одной и той же процедуре. Подпрограмма может иметь более одного оператора **Return**.



### Пример

В этом примере (практически, бессмысленном с математической точки зрения) приведена процедура, внутри которой для большей наглядности имеются две подпрограммы (именно подпрограммы с общими переменными!). Обратите внимание на то, что после оператора **Return** выполняется оператор **Debug.Print Num**.

```
Sub GosubDemo3()  
    Dim Num  
    Num = InputBox("Введите целое число")  
  
    Select Case Num  
        Case Is > 0  
            GoSub MyRoutine1  
        Case Is < 0  
            GoSub MyRoutine2  
    End Select  
  
    Debug.Print Num  
    Exit Sub  
MyRoutine1:  
    Debug.Print "значение/2:"  
    Num = Num / 2  
    Return  
MyRoutine2:  
    Debug.Print "(100+значение)/2"  
    Num = 100 + Num  
    Num = Num / 2  
    Return  
End Sub
```

---

### GoTo

Приводит к безусловной передаче управления выполнением программы на указанную строку внутри процедуры.

#### Синтаксис

**GoTo** *line*

Обязательный аргумент *line* может быть любой строковой или числовой меткой.

Оператор **GoTo** передает управление только на строку в процедуре, в которой этот оператор описан. С точки зрения структурного программирования этот оператор является «вредным», наносящим ущерб читабельности, создающим определенные трудности для отладки и сопровождения программного кода. Практика программирования показывает, что без этого оператора можно вполне обходиться. Чаще всего в VB этот оператор используется в программах обработки ошибок.

---

### If...Then...Else

Выполняет группу операторов в зависимости от значения некоторого условия.

#### Синтаксис 1

**If** *condition* **Then** [*statements*] [**Else** *elsestatements*]

**Синтаксис 2**

```

If condition Then
    [statements1]
    [ElseIf condition-n Then
        [elseifstatements] ...
    [Else
        [elsestatements2]]
End If

```

Оператор **If...Then...Else** состоит из следующих элементов:

<i>condition</i>	Обязательный. Одно или более выражений следующих двух типов: численное или строковое выражение, которое приводится к значениям <b>True</b> или <b>False</b> . Если значение <i>condition</i> — <b>Null</b> , интерпретируется как <b>False</b> .
<i>statements</i>	Необязательный в блочной форме (block form); обязательный в однострочной форме (single-line form), которая не имеет ветви <b>Else</b> . Один или более операторов; выполняются, если <i>condition</i> имеет значение <b>True</b> .
<i>condition-<i>n</i></i>	Необязательный. То же, что и <i>condition</i> .
<i>elseifstatements</i>	Необязательный. Один или более операторов, выполняемых, если соответствующее <i>condition-<i>n</i></i> имеет значение <b>True</b> .
<i>elsestatements</i>	Необязательный. Один или более операторов, выполняемых, если ни одно из предыдущих выражений <i>condition</i> или <i>condition-<i>n</i></i> не равны значению <b>True</b> .

Самая простая форма оператора — однострочная. Например, следующая строка кода приводит к выводу диалогового окна с текстом «а больше, чем d», если в момент выполнения этой строки значение переменной **a** больше, чем значение переменной **d**:

```
If a > d Then MsgBox "а больше, чем d"
```

Если необходимо выполнить более одного оператора в однострочной структуре, можно записать их все в одной строке, используя в качестве разделителя символ двоеточия.

Блочная структура обеспечивает большую гибкость, чем однострочная форма, и обычно легче читается и отлаживается. Следующие строки эквиваленты предыдущей строке кода:

```

If a > d Then
    MsgBox "а больше, чем d"
End If

```

**Пример**

В этой процедуре обработки события — выбор кнопки-переключателя — **Public**-переменной **np** присваивается тип подразделения. Далее вызывается функция загрузки наименований подразделений выбранного типа, например, в окно списка.

```

Private Sub OptionCklad_Click()
    If OptionCklad.Value Then
        np = 2          'тип подразделения -- СКЛАД
    Else

```

```
        np = 4          'тип подразделения -- ОПТОВЫЙ МАГАЗИН
End If

    t = ListBox1Load(np)
End Sub
```

**Implements**

Определяет интерфейс (interface) или класс (class), внедряемый в модуль класса, в котором используется этот оператор.

**Синтаксис**

```
Implements [InterfaceName | Class]
```

Обязательный аргумент *InterfaceName* или *Class* — имя интерфейса или класса в библиотеке типов (type library), методы которого будут внедряться соответствующими методами в VB-классе.

Интерфейс — коллекция прототипов, представляющая те члены (методы и свойства), которые инкапсулирует интерфейс; это означает, что он содержит только объявления для процедур-членов (member procedures). Класс обеспечивает реализацию (implementation) всех методов и свойств одного или более интерфейсов. Классы обеспечивают код, используемый при вызове любой функции контроллером (controller) класса. В Visual Basic любой член, который не является явно членом внедренного (implemented) интерфейса, — неявно член интерфейса по умолчанию.

**Input #**

Считывает данные из открытого файла последовательного доступа и присваивает их переменным.

**Синтаксис**

```
Input # filenumber, varlist
```

Синтаксис оператора **Input #** состоит из следующих элементов:

- filenumber*            Обязательный. Любой допустимый файловый номер.
- varlist*                Обязательный. Список разделенных запятыми переменных, которым присваиваются значения, считываемые из файла — массив или объектная переменная.

Данные, считываемые с помощью оператора **Input #**, — обычно то, что записано в этот файл оператором **Write #**. Используйте этот оператор только для файлов, открытых в режимах **Input** или **Binary**.

При чтении стандартной строки или числовых данных результат присваивается переменной без преобразования данных. В следующей таблице показано, как интерпретируются другие входные данные:

данные	значение, назначаемое переменной
Ограничивающая запятая или пустая строка	Empty
#NULL#	Null
#TRUE# или #FALSE#	True или False

данные	значение, назначаемое переменной
#yyyymm-dd hh:mm:ss#	Дата и/или время, представленное выражением
#ERROR errornumber#	errornumber (переменная — Variant, помеченная как ошибка)

Двойные кавычки ("" ) во входных данных игнорируются.

**Пример**

Чтобы выполнить следующий пример, необходимо предварительно создать файл с именем «IWFILE» и при помощи оператора Write # занести в него данные в формате: <строка>,<число>.

```
Sub TestInput()  
  Dim StrMy, NumberMy  
  Open "IWFILE" For Input As #1      'открыть файл  
  Do While Not EOF(1)                'цикл до конца файла  
    Input #1, StrMy, NumberMy        'считать данные в две переменные  
    MsgBox StrMy & " : " & NumberMy  'результат -> на экран  
  Loop  
  Close #1      'закрыть файл  
End sub
```

**Kill**

Оператор Kill удаляет файлы с диска.

**Синтаксис**

```
Kill pathname
```

Обязательный аргумент *pathname* — строковое выражение, которое определяет одно или более файловых имен для удаления. Аргумент *pathname* может включать каталог (или папку) и имя драйвера.

В Microsoft Windows оператор Kill поддерживает использование символов множественной подстановки \* или ?.

**Пример**

```
Kill "TestFile.TXT"  
Kill "*.BAK"
```

**Let**

Присваивает значение выражения переменной или свойству.

**Синтаксис**

```
[Let] varname = expression
```

Синтаксис оператора Let состоит из следующих элементов:

- Let**                    Необязательный. Явное использование ключевого слова Let — это оригинальный стиль, который обычно не применяется.
- varname*                Обязательный. Имя переменной или свойства; должен соответствовать стандартным соглашениям об именовании переменных.
- expression*            Обязательный. Значение, присваиваемое переменной или свойству.

Значение выражения может быть присвоено переменной или свойству, если только оно имеет тип данных, совместимый с типом переменной. Нельзя присвоить строковое выражение (string expression) числовой переменной (numeric variable); нельзя присвоить числовое выражение строковой переменной.

Переменные типа **Variant** могут присваиваться строчным или числовым переменным. Однако обратное не всегда верно. Строковой переменной может быть присвоена любая **Variant**-переменная, кроме **Null**, но числовой переменной может быть присвоена только та **Variant**-переменная, значение которой может интерпретироваться как число. Для определения того, может ли **Variant**-переменная быть преобразована в число, можно использовать функцию **IsNumeric**.

### Пример

```
Dim StrMy, IntMy
Let StrMy = "Опять — весна..."
Let IntMy = 25
```

---

### Line Input #

Считывает одну строку из файла последовательного доступа и присваивает эту строку **String**-переменной.

### Синтаксис

**Line Input #** *filenumber*, *varname*

Синтаксис оператора **Line Input #** состоит из следующих элементов:

*filenumber*            Обязательный. Допустимый файловый номер.

*varname*              Обязательный. Допустимое имя переменной типа **Variant** или **String**.

Данные, считываемые оператором **Line Input #**, обычно записываются в файл оператором **Print #**.

Оператор **Line Input #** считывает из файла по одному символу, пока не встретит символ «возврата каретки» (**Chr(13)**) или последовательность «возврат каретки» — «перевод строки» (**Chr(13) + Chr(10)**). Последовательность «возврат каретки» — «перевод строки» (carriage return—linefeed) не добавляется к считываемой символической строке.

### Пример

В качестве примера приведены две процедуры: для записи данных в файл (**TestPrint**) и для чтения данных из файла (**TestInput**). Запустите на выполнение первую процедуру. Затем откройте в Редакторе VB окно **Immediate** и выполните вторую процедуру для просмотра содержимого тестового файла.

```
Sub TestPrint()
'запись данных в файл TESTFILE
Dim TextLine
Open "TESTFILE" For Output As #1
Print #1, "Желал я душу освежить,"
Print #1, "Бывалой жизнью пожить"
Print #1, "В забвенье сладком близ друзей"
Print #1, "Минувшей юности моей"
Print #1,
Print #1, Tab(10); "А.С. Пушкин"
Close #1
End Sub
```

```

Sub TestInput()
'чтение данных из файла TESTFILE
  Dim TextLine
  Open "TESTFILE" For Input As #1
  Do While Not EOF(1)
    Line Input #1, TextLine
    Debug.Print TextLine
  Loop
  Close #1
End Sub

```

---

## Load

---

Загружает, но не показывает объект.

### Синтаксис

**Load** *object*

Шаблон *object* представляет объектное выражение (object expression), которое преобразуется к объекту в Applies To list.

При загрузке объект помещается в память, но остается невидимым. Чтобы объект стал видимым, необходимо выполнить его метод **Show**. Если объект остается невидимым, с ним невозможно «общаться».

### Пример

```

Private Sub UserForm_Initialize()
  Load UserForm2
  UserForm2.Show
End Sub

```

---

## Lock, Unlock

---

Управляет доступом ко всему или части файла, открытого с использованием оператора **Open**.

### Синтаксис

**Lock** [#] *filenumber* [, *recordrange*]

**Unlock** [#] *filenumber* [, *recordrange*]

Синтаксис оператора **Lock** (и **Unlock**) состоит из следующих элементов:

<i>filenumber</i>	Обязательный. Любой допустимый файловый номер.
<i>recordrange</i>	Необязательный. Диапазон записей для открытия (закрытия) доступа.

Аргумент *recordrange* имеет следующий синтаксис:

*recnumber* | [*start*] **To** *end*

*recnumber* Номер записи (файл в режиме **Random**) или номер байта (файл в режиме **Binary**), на который устанавливается или отменяется доступ.

*start* Номер первой записи (или байта) для открытия или закрытия доступа.

*end* Номер последней записи (или байта) для открытия или закрытия доступа.

Операторы **Lock** и **Unlock** используются в среде, где к одному и тому же файлу необходим доступ из нескольких процессов (*processes*), и всегда используются вместе. Аргументы этих операторов должны точно совпадать.

Первая запись (или байт) в файле находится в позиции 1, вторая запись (или байт) — в позиции 2 и т.д. Если указывается (в операторах **Lock**, **Unlock**) только одна запись, то только к этой записи будет открыт или закрыт доступ. Если определяется диапазон записей и опускается начальная запись (*start*), режим доступа задается для всех записей с первой до указанной последней записи (*end*). Использование **Lock** (**Unlock**) без аргумента *resnumber* управляет доступом ко всему файлу.

---

## LSet

---

Выполняет левое выравнивание строки внутри строковой переменной или копирует переменную одного пользовательского типа в другую переменную пользовательского типа, отличного от первого.

### Синтаксис

```
LSet stringvar = string  
LSet varname1 = varname2
```

Синтаксис оператора **LSet** состоит из следующих элементов:

<i>stringvar</i>	Обязательный. Имя строковой переменной.
<i>string</i>	Обязательный. Строковое выражение, которое будет выравниваться внутри <i>stringvar</i> .
<i>varname1</i>	Обязательный. Имя переменной пользовательского типа, в которое будет производиться копирование.
<i>varname2</i>	Обязательный. Имя переменной пользовательского типа, из которого будет производиться копирование.

Оператор **LSet** заменяет все левые символы в *stringvar* пробелами. Если *string* длиннее, чем *stringvar*, **LSet** помещает в *stringvar* только «наиболее левые» символы из *string*, количество которых равно длине строки *stringvar*.

### Пример (см. пример для RSet)

```
Dim StrMyVar  
StrMyVar = "0123456789"      ' инициализация для задания длины строки  
Lset StrMyVar = "<-Left"      ' StrMyVar теперь: "<-Left"  "
```

---

## Mid

---

Выполняет замену указанного числа символов в **Variant**-переменной (**String**) на символы другой строки.

### Синтаксис

```
Mid(stringvar, start[, length]) = string
```

Синтаксис оператора **Mid** состоит из следующих элементов:

<i>stringvar</i>	Обязательный. Имя строчной переменной для модификации.
<i>start</i>	Обязательный; <b>Variant (Long)</b> . Позиция символа в <i>stringvar</i> , с которой начинается заменяемая часть строки.

<i>length</i>	Необязательный; <b>Variant (Long)</b> . Число заменяемых символов.
<i>string</i>	Обязательный. Строковое выражение для замены.

Число заменяемых символов всегда меньше или равно числу символов с *stringvar*.

Не следует путать оператор **Mid** с функцией **Mid**, которая возвращает значение [тип **Variant (String)**], содержащее указанное число символов из строки, и имеет похожий синтаксис.

### Примеры

```
Dim StringMy
StringMy = "Мама Катю мыла"
Mid(StringMy, 5, 3) = "Сашу"           'StringMy = "Мама Сашу мыла"
Mid(StringMy, 5) = "Катю"              'StringMy = "Мама Катю мыла"
Mid(StringMy, 5) = "кашу варила"       'StringMy = "Мама кашу варила"
'функция Mid возвращает значение       'Мама кашу не варила':
SMidWords = Mid(StringMy, 1, 9) & " не " & Mid(StringMy, 11, 6)
```

---

### MkDir

---

Создает новый каталог (или папку).

#### Синтаксис

**MkDir** *path*

Обязательный аргумент *path* — строковое выражение, определяющее создаваемый каталог (или папку). Аргумент *path* может включать драйвер. Если драйвер не указывается, **MkDir** создает новый каталог (или папку) на текущем драйвере.

#### Пример

```
MkDir "NewDir"      'создать каталог на текущем драйвере
```

---

### Name

---

Переименовывает дисковый файл, каталог (папку).

#### Синтаксис

**Name** *oldpathname* **As** *newpathname*

Синтаксис оператора **Name** состоит из следующих элементов:

<i>oldpathname</i>	Обязательный. Строковое выражение, определяющее текущее имя и местонахождение файла; может включать каталог (папку) и драйвер диска.
<i>newpathname</i>	Обязательный. Строковое выражение, определяющее новое имя и местонахождение файла; может включать каталог (папку) и драйвер диска.

Оператор **Name** переименовывает файл и перемещает его в другой каталог (папку). Оператор **Name** может перемещать файл с одного драйвера (диска) на другой, но он может переименовать только существующий каталог (папку), если в обоих аргументах указан один и тот же драйвер. Оператор **Name** не может создать новый файл, каталог (папку).

Использование оператора **Name** с открытым файлом вызовет ошибку. Аргументы оператора **Name** не могут включать символ множественной подстановки (\*) и одиночной подстановки (?).



**Пример**

```
Dim StrOldName As String, StrNewName As String
StrOldName = "OLDFILE"
StrNewName = "NEWFILE"
Name StrOldName As StrNewName
```

---

**On Error**

---

Делает доступным обработчик ошибок и определяет его положение внутри процедуры; используется также и для запрета обработки ошибок.

**Синтаксис**

```
On Error GoTo line
On Error Resume Next
On Error GoTo 0
```

Синтаксис оператора **On Error** имеет следующие формы:

- |                             |   |
|-----------------------------|---|
| <b>On Error GoTo line</b>   | Делает доступным обработчик ошибок, который начинается со строки, определенной аргументом <i>line</i> . Аргумент <i>line</i> может быть меткой или числом. При возникновении ошибки управление выполнением программы передается на строку кода с меткой <i>line</i> . Таким образом происходит активизация обработчика ошибок. Метка <i>line</i> должна находиться в той же процедуре, что и оператор <b>On Error</b> . |
| <b>On Error Resume Next</b> | Указывает на то, что при возникновении ошибки управление выполнением программы передается на оператор, следующий непосредственно за тем, в котором произошла ошибка. Этот оператор позволяет продолжить выполнение программы, несмотря на возникновение run-time-ошибки. Оператор <b>On Error Resume Next</b> становится неактивным при вызове другой процедуры.  |
| <b>On Error GoTo 0</b>      | Отключает пользовательский обработчик ошибок в данной процедуре. Этот оператор не указывает метку <b>0</b> в качестве начала обработчика ошибки, даже если в процедуре действительно имеется такая метка. Без этого оператора обработчик ошибок «отключается» автоматически после окончания работы процедуры.   |

Пользовательские обработчики ошибок являются составной частью «дружественного» интерфейса. Если во время работы пользователя с вашим приложением возникает непредвиденная вами ситуация, то в любом случае не стоит оставлять пользователя «наедине» с системой VBA, не настолько дружественной, как хороший разработчик приложений. VBA «знает» очень мало о том, в каких местах вашего программного кода и какие могут возникать ошибки. Разработчик же приложения должен знать все «тонкие» места в коде и ожидать те или иные ошибки. Даже если приложение просто пытается открыть файл, который нечаянно «убили», VBA-сообщение об отсутствии файла не так информативно, как могло бы быть сообщение разработчика, которое не только должно сообщить об отсутствии файла, но и предложить возможные пути его восстановления или замены этого файла его старой копией.

Следует различать состояния «доступность» («enabled») и «активность» («active») обработчика ошибок. Обработчик доступен, если оператор **On Error** выполнен, но ошибка еще не возникла. Обработчик активен, когда им выполняется

обработка некоторой возникшей ошибки. Если ошибка возникает, когда обработчик активен (обрабатывает ранее возникшую ошибку), эта ошибка не обрабатывается и управление передается в вызывающую процедуру. Если вызывающая процедура имеет обработчик ошибок, находящийся в состоянии «enabled», он переходит в состояние «active» (т.е. обработки ошибки). Если обработчик вызывающей процедуры находится в состоянии «active», то управление передается предыдущей вызывающей процедуре с обработчиком в состоянии «enabled», но не «active». Т.е. в цепочке вызывающих процедур ищется обработчик ошибок в состоянии «enabled», готовый обработать ошибку.

Обычно в процедурах обработки ошибок для определения причины ошибки используется свойство **Number** объекта **Err**. Сообщение об ошибке (довольно «скупое»), связанное с **Err.Number**, содержится в **Err.Description**. Поэтому обычно на стадии отладки после метки, на которую передается управление выполнением программы в случае возникновения (ожидаемой) ошибки, помещается следующий оператор:

```
MsgBox "Error " & Err & ": " & Err.Description
```

### Пример

Следующая функция использует **On Error** и предназначена для создания базы данных на SQL Server. Если подключение по каким-либо причинам невозможно, функция выдаст (при помощи функции **MsgBox**) сообщение, из которого можно будет понять причину неудачи.

```
Function DataBaseCreate(DataBaseName As String) As Boolean
'простая Функция создания базы данных на SQL Server
'DataBaseName - имя базы
'
Dim wrk As Workspace, cnn As Connection, strConnect As String

On Error GoTo Err_DataBaseCreate

'строка подключения
strConnect = "ODBC;DSN=finan;UID=sa;PWD="

Set wrk = DBEngine.CreateWorkspace("NewODBCDirect", _
    "sa", "", dbUseODBC)
'устанавливаем тип драйвера курсора как
wrk.DefaultCursorDriver = dbUseClientBatchCursor

'открываем соединение
Set cnn = wrk.OpenConnection("finan", dbDriverNoPrompt, _
    False, strConnect)

'создать, наконец, указанную базу на сервере
cnn.Execute "CREATE DATABASE " & DataBaseName
cnn.Execute "CREATE TABLE T2 " & _
    "(C_KOD VARCHAR(12) NOT NULL, " & _
    "C_NAMET VARCHAR(30) )"

Exit_DataBaseCreate:
cnn.Close
Exit Function
Err_DataBaseCreate:

If Err.Number = 3146 Then
'такой номер выдается при попытке создать уже имеющуюся базу
```

```

    MsgBox "Проверьте наличие такой базы на сервере"
Else
    'вывод номера и описания ошибки
    MsgBox "Error " & Err.Number & ": " & Err.Description
End If
Resume Exit_DataBaseCreate
End Function

```

В следующей процедуре используется **On Error Resume Next** для того, чтобы при определении отсутствия в системе экземпляра приложения Excel программа не завершалась с ошибкой.

```

Sub TestError()
    Dim xlApp As Object
    Dim xlSheet As Object
    On Error Resume Next
    Set xlApp = Nothing
    Set xlApp = GetObject(, "Excel.Application")
    If Err.Number = 0 Then
        Set xlApp = CreateObject("Excel.Application")
    End If
    Err.Clear

    Set xlBook = xlApp.Workbooks.Add
    Set xlSheet = xlBook.WorkSheets(1)
    xlSheet.Application.Visible = True
End Sub

```

## On...GoSub, On...GoTo

«Передают управление» выполнением программы на одну из нескольких определенных строк кода в зависимости от некоторого значения.

### Синтаксис

```

On expression GoSub destinationlist
On expression GoTo destinationlist

```

Синтаксис операторов **On...GoSub** и **On...GoTo** включает следующие элементы:

**Expression** Обязательный. Любое численное выражение, которое приводится к целому числу в диапазоне от 0 до 255, включительно. Если *expression* — не является целым, оно предварительно округляется.

**destinationlist** Обязательный. Список номеров строк (line numbers) или строковых меток (line labels), разделенных запятыми.

Значение *expression* определяет, на какую строку в списке *destinationlist* выполняется переход. Если значение *expression* меньше чем 1, или больше чем число элементов в списке, то происходит следующее:

если <i>expression</i>	то
равно 0	«Управление передается» на оператор, следующий за операторами <b>On...GoSub</b> и <b>On...GoTo</b> .
больше числа элементов в списке	«Управление передается» на оператор, следующий за операторами <b>On...GoSub</b> и <b>On...GoTo</b> .
отрицательно	Иницируется ошибка.
больше, чем 255	Иницируется ошибка.

В одном списке можно смешивать номера строк и строковые метки. Можно использовать неограниченное количество номеров строк и строковых меток.

### Пример

```
Sub OnGosubGotoDemo()
Dim Number, MyString
    Number = 2

    On Number GoSub Sub1, Sub2
        ' On...GoSub.
    On Number GoTo Line1, Line2
Exit Sub

Sub1:
    MyString = "In Sub1" : Return
Sub2:
    MyString = "In Sub2" : Return
Line1:
    MyString = "In Line1"
Line2:
    MyString = "In Line2"
End Sub
```

---

### Open

Открывает файл для ввода/вывода.

#### Синтаксис

**Open** *pathname* **For** *mode* [**Access** *access*] [*lock*] **As** [#]*filenumber*  
[**Len**=*reclength*]

Синтаксис оператора **Open** состоит из следующих элементов:

<i>pathname</i>	Обязательный. Строковое выражение, которое определяет имя файла; может включать каталог (папку) и драйвер.
<i>mode</i>	Обязательный. Ключевое слово, определяющее режим файла: <b>Append</b> , <b>Binary</b> , <b>Input</b> , <b>Output</b> или <b>Random</b> .
<i>access</i>	Необязательный. Ключевое слово, определяющее операции, выполняемые с файлом: <b>Read</b> , <b>Write</b> или <b>Read Write</b> .
<i>lock</i>	Необязательный. Ключевое слово, определяющее ограничения для других процессов: <b>Shared</b> , <b>Lock Read</b> , <b>Lock Write</b> и <b>Lock Read Write</b> .
<i>Filenumber</i>	Обязательный. Допустимый файловый номер в диапазоне от 1 до 511, включительно. Для получения следующего допустимого файлового номера следует использовать функцию <b>FreeFile</b> .
<i>reclength</i>	Необязательный. Число, меньшее или равное 32767 (байтов). Для файлов, открытых в режиме прямого доступа ( <i>mode</i> = <b>Random</b> ), это значение равно длине записи. Для последовательного доступа это значение равно числу буферизованных символов.

Перед выполнением файловых операций чтения/записи файл необходимо открыть. Оператор **Open** выделяет буфер ввода/вывода для файла и определяет режим доступа к этому буферу. Если файл, указанный аргументом *pathname*, не существует, он создается для режимов **Append**, **Binary**, **Output** или **Random**.

Опция *Len* при *mode* со значением **Binary** игнорируется.

### Пример

```
Sub OpenTest()  
    Dim I, FileName  
    For I = 1 To 3  
        FileName = "TEST" & I           'создать имя файла  
        Open FileName For Output As #I   'открыть файл  
        Print #I, "запись в файл " & FileName 'записать строку в файл  
    Next I  
    Close  
    ' закрыть все три файла  
End Sub
```

---

## Option Base

Используется на модульном уровне для объявления нижней границы массива по умолчанию.

### Синтаксис

**Option Base {0 | 1}**

По умолчанию нижняя граница массива — это значение 0. При использовании этого оператора необходимо располагать его в модуле до объявления процедур. Оператор **Option Base** может появляться в модуле только один раз и предшествовать объявлению массивов. При этом он воздействует только на нижнюю границу (может быть определена при помощи функции **LBound**) массивов в модуле, в котором он определен.

---

## Option Compare

Используется на модульном уровне для объявления метода по умолчанию при сравнении строчных данных.

### Синтаксис

**Option Compare {Binary | Text | Database}**

Оператор **Option Compare** следует располагать в модуле перед определением процедур.

Оператор **Option Compare** определяет метод сравнения строк (**Binary**, **Text** или **Database**) для модуля. Если модуль не содержит оператор **Option Compare**, по умолчанию при текстовом сравнении используется метод **Binary**.

При использовании **Option Compare Binary** сравнение строк основывается на двоичном представлении символов, т.е. на порядке расположения символов в таблице двоичного представления. В Microsoft Windows порядок расположения символов определяется кодовой страницей.

Для **Option Compare Text** двоичный эквивалент для каждого символа не используется. При этом буквы верхнего регистра эквивалентны буквам нижнего регистра.

**Option Compare Database** может использоваться только в Microsoft Access. При этом сравнение строк основывается на параметре, определенном локальным ID базы данных, где возникают эти сравнения.

---

## Option Explicit

Используется на модульном уровне для обязательного явного объявления всех переменных в модуле.

**Синтаксис****Option Explicit**

Оператор **Option Explicit** должен располагаться в модуле до объявления процедур.

При использовании оператора **Option Explicit** необходимо явно объявлять все переменные с помощью операторов **Dim**, **Private**, **Public**, **ReDim** или **Static**. Если этого не сделать, возникнет ошибка времени компиляции.

Если оператор **Option Explicit** не используется, все необъявленные переменные имеют тип **Variant**.

**Пример**

```
Option explicit
Dim VarMy
IntMy = 103      ' эта строка вызовет ошибку
VarMy = 101      ' эта строка не вызовет ошибку
```

**Option Private**

При использовании в *host*-приложениях, которые допускают ссылки на множество проектов, **Option Private Module** препятствует ссылкам на содержимое модуля из внешней среды этого проекта. В *host*-приложениях, которые не допускают таких ссылок, например, отдельные версии Visual Basic, оператор **Option Private** не действует.

**Синтаксис****Option Private Module**

Оператор **Option Private** на модульном уровне должен появляться перед описанием процедур. Если модуль включает оператор **Option Private Module**, *public*-элементы, переменные, объекты и определяемые пользователем типы, объявленные на модульном уровне, остаются доступными внутри проекта, содержащего этот модуль, но недоступными другим приложениям или проектам.

**Пример**

```
Option private Module
```

**Print #**

Записывает отформатированные для отображения данные в файл с последовательным доступом.

**Синтаксис**

```
Print #filenumber, [outputlist]
```

Синтаксис оператора **Print #** включает следующие элементы:

*Filenumber*            Обязательный. Любой допустимый файловый номер.

*outputlist*            Необязательный. Выражение или список выражений для печати.

Аргумент *outputlist* имеет следующий синтаксис:

```
[[Spc(n) | Tab(n)]][expression] [charpos]
```

**Spc**(*n*)                Используется для вставки в вывод символов пробела; *n* — это число вставляемых символов пробелов.

<b>Tab(<i>n</i>)</b>	Используется для позиционирования точки (курсора) вставки в определенную колонку, где <i>n</i> — номер колонки. Использование <b>Tab</b> без аргументов приводит к позиционированию точки вставки в начало следующей зоны печати (print zone).
<i>expression</i>	Численные или строковые выражения для печати.
<i>charpos</i>	Определяет точку вставки для следующего символа. Чтобы позиционировать точку вставки сразу после последнего отображенного символа, используйте точку с запятой. Для позиционирования точки вставки в определенную колонку используйте <b>Tab(<i>n</i>)</b> . Использование <b>Tab</b> без аргументов приводит к позиционированию точки вставки в начало следующей зоны печати. Если <i>charpos</i> опускается, очередной символ печатается на следующей строке.

Данные, записываемые оператором **Print #**, обычно считываются из файла при помощи операторов **Line Input #** или **Input**. Если *outputlist* не используется, а после *filename* включается разделитель списка, в файл записывается пустая строка. Несколько записываемых в файл выражений (на одной строке кода) могут разделяться либо пробелами, либо двоеточиями.

### Пример

```
Sub TestPrint()
'запись данных в файл TESTFILE
Dim TextLine
Open "TESTFILE" For Output As #1
Print #1, "Нет ни в чем вам благодати;"
Print #1, "С счастьем у вас разлад:"
Print #1, "И прекрасны вы некстати,"
Print #1, "И умны вы невпопад."
Print #1,
Print #1, Tab(10); "А.С. Пушкин"
Close #1
End Sub
```

---

### Private

---

Используется на модульном уровне для объявления *private*-переменных и выделения для них памяти.

### Синтаксис

```
Private [WithEvents] varname[[subscripts]] [As [New] type]
[, [WithEvents] varname[[subscripts]] [As [New] type]] ...
```

Синтаксис оператора **Private** включает следующие элементы:

**WithEvents** Необязательный. Ключевое слово, которое определяет *varname* — объектная переменная для реакции на события, вызываемые ActiveX-объектом. **WithEvents** допускается только в модулях класса. Это ключевое слово нельзя использовать при объявлении массивов. Нельзя использовать ключевое слово **New** с ключевым словом **WithEvents**.

*varname* Обязательный. Имя переменной.

<i>subscripts</i>	<p>Необязательный. Размерности массива-переменной; можно объявлять до 60 размерностей. Аргумент <i>subscripts</i> имеет следующий синтаксис:</p> <p>[<i>lower To</i>] <i>upper</i> [, [<i>lower To</i>] <i>upper</i>] ...</p> <p>Если явно не указывается <i>lower</i>, то нижняя граница массива определяется оператором <b>Option Base</b>. Если же оператор <b>Option Base</b> в коде не используется, значение нижней границы по умолчанию равняется 0.</p>
<b>New</b>	<p>Необязательный. Ключевое слово, позволяющее создание объекта. Использование <b>New</b> при объявлении объектной переменной приводит к созданию нового экземпляра объекта и ссылки на него, поэтому в операторе <b>Set</b> (назначает ссылку на объект) для этого объекта нет необходимости.</p>
<i>Type</i>	<p>Необязательный. Тип (данных) переменной; может быть <b>Byte</b>, <b>Boolean</b>, <b>Integer</b>, <b>Long</b>, <b>Currency</b>, <b>Single</b>, <b>Double</b>, <b>Date</b>, <b>String</b> (только переменной длины), <b>String * length</b> (только фиксированной длины), <b>Object</b>, <b>Variant</b>, типом, определенным пользователем, или объектным типом.</p>

Оператор **Private** используется для объявления переменных (объектных переменных), доступных только в модуле, в котором они объявлены. Например, в следующей строке объявлены переменная целого типа **j** и объектная переменная типа **Table** с именем **newT**.

```
Private j As Integer, newT As New Table
```

Если при объявлении объектной переменной ключевое слово **New** не используется, то оператор **Private** не создает новый экземпляр объекта, а создает, практически, ссылочную переменную, которая пока не ссылается ни на какой конкретный объект (участок памяти). Чтобы таким образом объявленная объектная переменная ссылалась на объект, необходимо оператором **Set** указать этот объект. До тех пор, пока этого не сделано, объектная переменная имеет специальное значение **Nothing**, которое указывает на то, что переменная не ссылается ни на какой объект.

Если при объявлении переменных (объектных переменных) тип переменной не указывается, VB по умолчанию объявляет для нее тип **Variant**. Из синтаксиса оператора **Private** следует, что с его помощью можно объявлять массивы (динамические — в том числе).

---

## Property Get

---

Объявляет имя, аргументы и код, формирующие **Property**-процедуру, которая считывает (извлекает) значение свойства (property).

### Синтаксис

```
[Public | Private | Friend] [Static] Property Get name [(arglist)] [As type]
    [statements]
    [name = expression]
    [Exit Property]
    [statements]
    [name = expression]
End Property
```



Значение элементов синтаксиса оператора **Property Get**:

<b>Public</b>	Необязательный. Указывает на то, что процедура <b>Property Get</b> доступна всем другим процедурам во всех модулях. Если используется в модуле, содержащем оператор <b>Option Private</b> , эта процедура не доступна вне проекта.
<b>Private</b>	Необязательный. Указывает на то, что процедура <b>Property Get</b> доступна другим процедурам в модуле, в котором она объявлена.
<b>Friend</b>	Необязательный. Используется только в модуле класса. Указывает на то, что процедура <b>Property Get</b> видима во всем проекте.
<b>Static</b>	Необязательный. Указывает на то, что локальные переменные <b>Property Get</b> -процедуры сохраняют свои значения между вызовами. <b>Static</b> -атрибут не оказывает действия на переменные, которые объявляются вне процедуры <b>Property Get</b> , даже если они используются в этой процедуре.
<i>name</i>	Обязательный. Имя процедуры <b>Property Get</b> ; должно соответствовать стандартным соглашениям об именовании переменных, за исключением того, что в одном модуле имя может быть таким же, как и в процедуре <b>Property Let</b> или <b>Property Set</b> .
<i>arglist</i>	Необязательный. Список разделенных запятыми переменных, представляющих аргументы, передаваемые процедуре <b>Property Get</b> при ее вызове. Имя и тип каждого аргумента в процедуре <b>Property Get</b> должны быть теми же, что и в соответствующей процедуре <b>Property Let</b> (если она имеется).
<i>type</i>	Необязательный. Тип (данных) значения, возвращаемого процедурой <b>Property Get</b> ; может быть <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>String</b> (кроме фиксированной длины), <b>Object</b> , <b>Variant</b> , пользовательским типом и массивом.
<i>statements</i>	Необязательный. Любая группа операторов, которая выполняется в теле процедуры <b>Property Get</b> .
<i>expression</i>	Необязательный. Значение свойства, возвращаемое процедурой, определяемой оператором <b>Property Get</b> .

Аргумент *arglist* имеет следующий синтаксис:

```
[Optional] [ByVal | ByRef] [ParamArray] varname[( )] [As type]
[= defaultvalue]
```

<b>Optional</b>	Необязательный. Указывает, что аргумент является необязательным. Если используется, все последующие аргументы в <i>arglist</i> должны быть необязательными и объявляться с использованием слова <b>Optional</b> .
<b>ByVal</b>	Необязательный. Указывает, что аргумент передается по значению (by value).
<b>ByRef</b>	Необязательный. Указывает, что аргумент передается по ссылке (by reference). В Visual Basic способ <b>ByRef</b> используется по умолчанию.

<b>ParamArray</b>	Необязательный. Используется только как последний аргумент в <i>arglist</i> для указания того, последний аргумент — это <b>Optional</b> -массив <b>Variant</b> -элементов. Ключевое слово <b>ParamArray</b> позволяет использовать переменное число аргументов и не может применяться с <b>ByVal</b> , <b>ByRef</b> или <b>Optional</b> .
<i>varname</i>	Обязательный. Имя переменной, представляющей аргумент; должно соответствовать стандартным соглашениям об именовании переменных.
<i>type</i>	Необязательный. Тип аргумента, передаваемого процедуре; может быть типом <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>String</b> (фиксированной длины), <b>Object</b> , <b>Variant</b> или объектным типом. Если параметр — не <b>Optional</b> , может быть указан также пользовательский тип.
<i>defaultvalue</i>	Необязательный. Любая константа или константное выражение. Допустима только для <b>Optional</b> -параметров. Если тип — <b>Object</b> , явным значением по умолчанию может быть только <b>Nothing</b> .

Если явно не указано использование **Public**, **Private** или **Friend**, процедура **Property** является **Public** по умолчанию. Если не используется **Static**, локальные переменные не сохраняют свои значения между вызовами процедуры. Ключевое слово **Friend** можно использовать только в модуле класса. Однако **Friend**-процедура может быть доступна любой процедуре модуля проекта.

При программировании классов некоторые данные класса (обычно они называются свойствами) можно защитить от прямого обновления и даже считывания. Пользователь класса (в качестве которого выступают программы) может не иметь непосредственного доступа к данным класса, но может обращаться к ним посредством интерфейсных процедур. В качестве таких процедур в VB используются процедуры с наименованиями **Property Get** и **Property Let**. Особенно важным является возможность проверки корректности данных, записываемых в закрытые элементы класса при помощи кода процедур **Property Let**.

### Пример

Пусть, например, необходимо разработать класс, в котором будут содержаться такие сведения о накладной: дата, номер накладной, код отправителя товара, код получателя товаров. Код для описания данных (в модуле класса в разделе объявлений) может иметь вид:

```
Private Dm_Date As Date
Private Im_Num As Integer
Private Cm_Sou As String
Private Cm_Wne As String
```

Для получения значения закрытой переменной **Dm\_Date** как общедоступного свойства **D\_Date** можно написать следующий код:

```
Public Property Get D_Date() As Date
    D_Date = Dm_Date
End Property
```

Точно так же можно написать код для получения остальных свойств объекта класса:

```
Public Property Get I_Num() As Integer
    I_Num = Im_Num
End Property
```

```

Public Property Get C_Sou() As String
    C_Sou = Cm_Sou
End Property
Public Property Get C_Wne() As String
    C_Wne = Cm_Wne
End Property

```

---

## Property Let

---

Объявляет имя, аргументы и код, формирующие **Property Let** процедуру, которая присваивает значение свойству.

---

## Синтаксис

---

```

[Public | Private | Friend] [Static] Property Let name (
    [arglist,] value)
    [statements]
[Exit Property]
    [statements]
End Property

```

Значения элементов синтаксиса оператора **Property Let**:

<b>Public</b>	Необязательный. Указывает на то, что процедура <b>Property Let</b> доступна всем другим процедурам во всех модулях. Если используется в модуле, содержащем оператор <b>Option Private</b> , эта процедура недоступна вне проекта.
<b>Private</b>	Необязательный. Указывает на то, что процедура <b>Property Let</b> доступна другим процедурам в модуле, в котором она объявлена.
<b>Friend</b>	Необязательный. Используется только в модуле класса. Указывает на то, что процедура <b>Property Let</b> видима во всем проекте.
<b>Static</b>	Необязательный. Указывает на то, что локальные переменные <b>Property Let</b> -процедуры сохраняют свои значения между вызовами. <b>Static</b> -атрибут не оказывает действия на переменные, которые объявляются вне процедуры <b>Property Let</b> , даже если они используются в этой процедуре.
<i>name</i>	Обязательный. Имя процедуры <b>Property Let</b> ; должно соответствовать стандартным соглашениям об именовании переменных, за исключением того, что в одном модуле имя может быть таким же как и в процедуре <b>Property Get</b> или <b>Property Set</b> .
<i>arglist</i>	Обязательный. Список разделенных запятыми переменных, представляющих аргументы, передаваемые процедуре <b>Property Let</b> при ее вызове. Имя и тип каждого аргумента в процедуре <b>Property Let</b> должны быть теми же, что и в соответствующей процедуре <b>Property Get</b> (если она имеется).
<i>value</i>	Обязательный. Переменная, значение которой присваивается свойству. При вызове процедуры этот аргумент появляется в правой части вызывающего выражения. Тип переменной <i>value</i> должен совпадать с типом возвращаемого значения соответствующей <b>Property Get</b> -процедуры.
<i>statements</i>	Необязательный. Любая группа операторов, которая выполняется в теле процедуры <b>Property Let</b> .

Аргумент *arglist* имеет следующий синтаксис:

```
[Optional] [ByVal | ByRef] [ParamArray] varname[( )] [As type]
[= defaultvalue]
```

<b>Optional</b>	Необязательный. Указывает, что аргумент является необязательным. Если используется, все последующие аргументы в <i>arglist</i> должны быть необязательными и объявляться с использованием слова <b>Optional</b> .
<b>ByVal</b>	Необязательный. Указывает, что аргумент передается по значению.
<b>ByRef</b>	Необязательный. Указывает, что аргумент передается по ссылке. В Visual Basic способ <b>ByRef</b> используется по умолчанию.
<b>ParamArray</b>	Необязательный. Используется только как последний аргумент в <i>arglist</i> для указания того, последний аргумент — это <b>Optional</b> -массив <b>Variant</b> -элементов. Ключевое слово <b>ParamArray</b> позволяет использовать переменное число аргументов и не может применяться с <b>ByVal</b> , <b>ByRef</b> или <b>Optional</b> .
<i>varname</i>	Обязательный. Имя переменной, представляющей аргумент; должно соответствовать стандартным соглашениям об именовании переменных.
<i>type</i>	Необязательный. Тип аргумента, передаваемого процедуре; может быть типом <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>String</b> (фиксированной длины), <b>Object</b> , <b>Variant</b> или объектным типом. Если параметр — не <b>Optional</b> , может быть указан также пользовательский тип.
<i>defaultvalue</i>	Необязательный. Любая константа или константное выражение. Допустима только для <b>Optional</b> -параметров. Если тип — <b>Object</b> , явным значением по умолчанию может быть только <b>Nothing</b> .

Каждый оператор **Property Let** должен определять, по крайней мере, один аргумент процедуры, которую этот оператор описывает. Этот аргумент (или последний, если их несколько) содержит значение, которое присваивается свойству при вызове процедуры, определенной оператором **Property Let**.

Если явно не указано использование **Public**, **Private** или **Friend**, процедура **Property** является **Public** по умолчанию. Если не используется **Static**, локальные переменные не сохраняют свои значения между вызовами процедуры. Ключевое слово **Friend** можно использовать только в модуле класса. Однако **Friend**-процедура может быть доступна любой процедуре модуля проекта.

Оператор **Exit Property** (их может быть более одного в процедуре) приводит к немедленному выходу из **Property Let**-процедуры. Выполнение программы продолжается с оператора, следующего за тем, который вызвал **Property Let**-процедуру.

Подобно **Function**- и **Property Get**-процедурам **Property Let**-процедура — это отдельная процедура, которая может принимать аргументы, выполнять набор операторов и изменять значения своих аргументов. Однако в отличие от **Function**- и **Property Get**-процедур, которые возвращают значение, **Property Let**-процедуры можно располагать только в левой части выражения присваивания свойства или оператора **Let**.

### Пример

Для класса из примера оператора **Property Get** можно написать следующие процедуры **Property Let**:

```
Public Property Let D_Date(ByVal DatNew As Date)

    'код для проверки корректности даты
    'можно, например, исключить выходные и праздники
    ...
    Dm_Date = DatNew

End Property
Public Property Let I_Num(ByVal NumNew As Integer)

    'код для проверки корректности номера накладной
    ...
    Im_Num = NumNew
End Property
Public Property Let C_Sou(ByVal SouNew As String)

    'код для проверки корректности кода источника
    'если такой код в базе данных отсутствует, функция
    'может об этом сообщить
    ...
    Cm_Sou = SouNew
End Property
Public Property Let C_Wne(ByVal WneNew As String)

    'код для проверки корректности кода приемника
    'если такой код в базе данных отсутствует, функция
    'может об этом сообщить
    ...
    Cm_Wne = WneNew
End Property
```

---

### Property Set

---

Объявляет имя, аргументы и код, формирующие **Property**-процедуру, которая устанавливает ссылку на объект.

#### Синтаксис

```
[Public | Private | Friend] [Static] Property Set name ([arglist,]
reference)
    [statements]
    [Exit Property]
    [statements]
End Property
```

Синтаксис оператора **Property Set** состоит из следующих элементов:

- |                |   |
|----------------|---|
| <b>Public</b>  | Необязательный. Указывает на то, что процедура <b>Property Set</b> доступна всем другим процедурам во всех модулях. Если используется в модуле, содержащем оператор <b>Option Private</b> , эта процедура недоступна вне проекта. |
| <b>Private</b> | Необязательный. Указывает на то, что процедура <b>Property Set</b> доступна другим процедурам в модуле, в котором она объявлена.  |
| <b>Friend</b>  | Необязательный. Используется только в модуле класса. Указывает на то, что процедура <b>Property Set</b> видима во всем проекте.   |

<b>Static</b>	Необязательный. Указывает на то, что локальные переменные <b>Property Set</b> -процедуры сохраняют свои значения между вызовами. <b>Static</b> -атрибут не оказывает действия на переменные, которые объявляются вне процедуры <b>Property Set</b> , даже если они используются в этой процедуре.
<i>name</i>	Обязательный. Имя процедуры <b>Property Set</b> ; должно соответствовать стандартным соглашениям об именовании переменных, за исключением того, что в одном модуле имя может быть таким же, как и в процедуре <b>Property Get</b> или <b>Property Let</b> .
<i>arglist</i>	Обязательный. Список разделенных запятыми переменных, представляющих аргументы, передаваемые процедуре <b>Property Set</b> при ее вызове.
<i>reference</i>	Обязательный. Переменная, содержащая объектную ссылку, используемую в правой части присвоения объектной ссылки.
<i>statements</i>	Необязательный. Любая группа операторов, которые должны выполняться в процедуре.

Аргумент *arglist* имеет следующий синтаксис:

```
[Optional] [ByVal | ByRef] [ParamArray] varname([ ]) [As type]
[= defaultvalue]
```

<b>Optional</b>	Необязательный. Указывает, что аргумент является необязательным. Если используется, все последующие аргументы в <i>arglist</i> должны быть необязательными и объявляться с использованием слова <b>Optional</b> .
<b>ByVal</b>	Необязательный. Указывает, что аргумент передается по значению.
<b>ByRef</b>	Необязательный. Указывает, что аргумент передается по ссылке. В Visual Basic <b>ByRef</b> используется по умолчанию.
<b>ParamArray</b>	Необязательный. Используется только как последний аргумент в <i>arglist</i> для указания того, что последний аргумент — это <b>Optional</b> -массив <b>Variant</b> -элементов. Ключевое слово <b>ParamArray</b> позволяет использовать переменное число аргументов и не может применяться с <b>ByVal</b> , <b>ByRef</b> или <b>Optional</b> .
<i>varname</i>	Обязательный. Имя переменной, представляющей аргумент; должно соответствовать стандартным соглашениям об именовании переменных.
<i>Type</i>	Необязательный. Тип аргумента, передаваемого процедуре; может быть <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>String</b> (фиксированной длины), <b>Object</b> , <b>Variant</b> или объектным типом. Если параметр — не <b>Optional</b> , может быть указан также пользовательский тип.
<i>defaultvalue</i>	Необязательный. Любая константа или константное выражение. Допустима только для <b>Optional</b> -параметров. Если тип — <b>Object</b> , явным значением по умолчанию может быть только <b>Nothing</b> .

Каждый оператор **Property Set** должен определять, по крайней мере, один аргумент процедуры, которую этот оператор описывает. Этот аргумент (или последний, если их несколько) содержит объектную ссылку для свойства при вызове процедуры, определенной оператором **Property Set**. Это — *reference* в предыдущем синтаксисе. Может быть **Optional**.

Если явно не указано использование **Public**, **Private** или **Friend**, процедура **Property** является **Public** по умолчанию. Если не используется **Static**, локальные переменные не сохраняют свои значения между вызовами процедуры. Ключевое слово **Friend** можно использовать только в модуле класса. Однако **Friend**-процедура может быть доступна любой процедуре модуля проекта. Весь исполняемый код должен быть в процедуре. Нельзя определять процедуру **Property Set** внутри другой **Property**-, **Sub**- или **Function**-процедуры.

Оператор **Exit Property** (их может быть более одного в процедуре) приводит к немедленному выходу из **Property Set**-процедуры. Выполнение программы продолжается с оператора, следующего за тем, который вызвал **Property Set**-процедуру.

Подобно **Function**- и **Property Get**-процедурам, **Property Set**-процедура — это отдельная процедура, которая может принимать аргументы, выполнять набор операторов и изменять значения своих аргументов. Однако в отличие от **Function**- и **Property Get**-процедур, которые возвращают значение, **Property Set**-процедуры можно располагать только в левой части выражения присваивания свойства или оператора **Set**.

---

**Public**

---

Используется на модульном уровне для объявления **public**-переменных и выделения памяти для них.

**Синтаксис**

```
Public [WithEvents] varname([[subscripts]]) [As [New] type]
[, [WithEvents] varname([[subscripts]]) [As [New] type]] ...
```

Синтаксис оператора **Public** содержит следующие элементы:

<b>WithEvents</b>	Необязательный. Ключевое слово, которое указывает, что <i>varname</i> — объектная переменная, используемая для реакции на событие, инициированное ActiveX-объектом. <b>WithEvents</b> допускается только в модулях класса. Отдельных переменных с использованием ключевого слова <b>WithEvents</b> можно объявлять столько, сколько необходимо, но массив, используя его, создавать нельзя. Также нельзя использовать ключевое слово <b>New</b> при наличии <b>WithEvents</b> .
<i>varname</i>	Обязательный. Имя переменной; должно соответствовать стандартным соглашениям об именовании переменных.
<i>subscripts</i>	Необязательный. Размерность переменной-массива; можно объявить до 60 размерностей. Аргумент <i>subscripts</i> использует следующий синтаксис:  [ <i>lower To</i> ] <i>upper</i> [, [ <i>lower To</i> ] <i>upper</i> ] ...  <i>lower</i> — нижний предел допустимых индексов массива; <i>upper</i> — верхний предел (является обязательным) для индексов массива. При наличии в операторе <b>Dim</b> только <i>upper</i> нижний предел определяется в зависимости от установки оператора <b>Option Base</b> .

<b>New</b>	Необязательный. Ключевое слово, позволяющее неявно создать объект. Если вы используете ключевое слово <b>New</b> при объявлении объектной переменной, создается новый экземпляр объекта, поэтому нет необходимости использовать оператор <b>Set</b> для назначения объекту ссылки.
<i>type</i>	Необязательный. Тип данных переменной; может быть <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>String</b> (только переменной длины), <b>String</b> * <i>length</i> (для строк фиксированной длины), <b>Object</b> , <b>Variant</b> , типом, определенным пользователем, или объектным типом.

Переменные, объявляемые оператором **Public**, доступны всем процедурам всех модулей во всех приложениях, если используется оператор **Option Private Module**, при действии которого переменные имеют статус **Public** только в том проекте, в котором они находятся. Оператор **Public** нельзя применять в модуле класса для объявления переменной-строки фиксированной длины.

Оператор **Public** можно использовать для объявления типа переменной. Например, следующий оператор объявляет переменную типа **Integer**:

```
Dim IntName As Integer
```

Оператор **Public** можно использовать также для объявления переменной объектного типа:

```
Dim WrkShNew As Worksheet
```

или

```
Dim WrkShNew As New Worksheet
```

Если при объявлении объектной переменной ключевое слово **New** не используется, то чтобы объектная переменная ссылалась на объект, необходимо оператором **Set** указать этот объект. До тех пор, пока этого не сделано, объектная переменная имеет специальное значение **Nothing**, которое указывает на то, что переменная не ссылается ни на какой объект.

Оператор **Public** можно использовать (вместо оператора **Dim**) и для объявления массивов. После объявления динамического массива (при помощи пустых круглых скобок) следует для указания размерности массива применять оператор **ReDim**.

## Put

Записывает данные из переменной в файл диска.

### Синтаксис

```
Put [#]filename, [recnumber], varname
```

Синтаксис оператора **Put** включает следующие элементы:

<i>Filename</i>	Обязательный. Любой допустимый файловый номер.
<i>recnumber</i>	Необязательный. <b>Variant (Long)</b> . Номер записи (для режима доступа <b>Random</b> ) или номер байта (для режима доступа <b>Binary</b> ) для записи.
<i>varname</i>	Обязательный. Имя переменной, содержащей записываемые в файл данные.

Данные, записанные оператором **Put**, обычно можно прочитать оператором **Get**.



Первая запись (или байт) в файле находится в позиции 1, вторая запись (или байт) — в позиции 2 и т.д. Если аргумент *resnumber* не указывается, номер записи (или байта) определяется как результат работы последнего оператора **Get** (или **Put**) или функции **Seek**.

Для файлов, открытых в **Random**-режиме, применяются следующие правила:

- Если длина записываемых данных меньше, чем длина, указанная в опции **Len** оператора **Open**, **Put** записывает последующие записи на границах длин записей. Пространство между концом одной записи и началом следующей записи заполняется имеющимся содержимым файлового буфера.
- Если записываемая переменная является строкой переменной длины, оператор **Put** записывает сначала 2-байтовый дескриптор (descriptor), содержащий длину строки, а затем — переменную. Поэтому длина записи, определяемая опцией **Len** в операторе **Open**, должна быть, по крайней мере, больше на 2 байта, чем действительная длина строки.
- Если записываемая переменная — **Variant** числового типа, **Put** записывает 2 байта, идентифицирующие **VarType** переменной **Variant**, а затем — само значение переменной. Длина записи (record), определяемая опцией **Len** в операторе **Open**, должна быть, по крайней мере, на 2 байта больше, чем действительное число байтов, необходимое для сохранения переменной.
- Если записываемая переменная — **Variant**, значение **VarType** которой равно 8 (**String**), оператор **Put** записывает 2 байта, идентифицирующие **VarType**, 2 байта, указывающие длину строки, а затем — сами строковые данные. Длина записи, определяемая опцией **Len** оператора **Open**, должна быть, по крайней мере, на 4 байта больше, чем действительная длина строки.
- Если записываемая переменная — динамический массив, оператор **Put** записывает дескриптор, длина которого равна 2 «плюс» 8, умноженное на число размерностей массива ( $2 + 8 * \text{NumberOfDimensions}$ ). Длина записи, определяемая опцией **Len** в операторе **Open**, должна быть больше или равна сумме количества байтов, необходимых для записи массива, и длины дескриптора. Например, объявленный ниже массив требует при записи на диск 218 байтов —  $18 (2 + 8 * 2)$  байтов занимает дескриптор и  $200 (5 * 20 * 2)$  необходимо для хранения массива:

```
Dim MyArray(1 To 5, 1 To 20) As Integer
```

- Если записываемая переменная — массив фиксированного размера, оператор **Put** записывает только данные.
- Если записываемая переменная — переменная любого другого типа (не строка переменной длины или **Variant**), **Put** записывает только данные. Длина записи, определяемая опцией **Len** в операторе **Open**, должна быть больше или равна длине записываемых данных.
- Оператор **Put** записывает элементы типов, определенных пользователем, как если бы они записывались отдельно, за исключением того, что между ними нет заполняющих элементов. На диске динамический массив определенного пользователем типа (записанный оператором **Put**) предваряется дескриптором, длина которого равна 2 «плюс» 8, умноженное на число размерностей:  $2 + 8 * \text{NumberOfDimensions}$ . Длина записи, определяемая опцией **Len** в операторе **Open**, должна быть больше или равна сумме всех байтов, необходимых для чтения отдельных элементов, включая любые массивы и их дескрипторы.

Для файлов, открытых в **Binary**-режиме, применяются все **Random**-правила, кроме:

- **Len**-опция (параметр) оператора **Open** не действует. Оператор **Put** записывает все переменные на диск непрерывно, т.е. без заполняющих символов между записями.
- Для любого массива, кроме массива пользовательского типа, **Put** записывает только данные. Дескриптор не записывается.
- **Put** записывает строки переменной длины, которые не являются элементами пользовательских типов, без дескриптора 2-байтовой длины. Число записываемых байтов равно числу символов, имеющихся в строке.

### Пример

В следующем примере описывается пользовательский тип данных (**Record**) и процедура записи (**TestPut**) данных в файл.

```
Type Record      ' Определенный пользователем тип
    ID As Integer
    FirstName As String * 20
    LastName As String * 20
End Type

Sub TestPut()
    'Тестирование Put
    Dim MyRecord As Record, RecordNumber

    Kill "TESTFILE"
    Open "TESTFILE" For Random As #1 Len = Len(MyRecord)

    MyRecord.FirstName = "Петр"
    MyRecord.LastName = "Петров"
    MyRecord.ID = 333
    Put #1, 2, MyRecord

    MyRecord.FirstName = "Сидор"
    MyRecord.LastName = "Сидоров"
    MyRecord.ID = 111
    Put #1, 3, MyRecord

    MyRecord.FirstName = "Иван"
    MyRecord.LastName = "Иванов"
    MyRecord.ID = 222
    Put #1, 1, MyRecord

    Close #1
End Sub
```

---

### RaiseEvent

---

Иницирует событие, объявленное на модульном уровне в классе (class), форме (form) или документе (document).

#### Синтаксис

**RaiseEvent** *eventname* [(*argumentlist*)]

Обязательный аргумент *eventname* — имя события, объявляемого внутри модуля; должно соответствовать стандартным соглашениям об именовании переменных.

Синтаксис оператора **RaiseEvent** включает следующие элементы:

<i>eventname</i>	Обязательный. Имя события.
<i>argumentlist</i>	Необязательный. Разделяемый запятыми список аргументов, в качестве которых могут использоваться переменные, массивы или выражения. Если аргументов нет, скобки не должны использоваться.

## Randomize

Инициализирует генератор случайных чисел.

### Синтаксис

**Randomize** [*number*]

Необязательный аргумент *number* — значение типа **Variant** или любое числовое выражение.

Оператор **Randomize** использует *number* для инициализации генератора случайных чисел функции **Rnd**, задавая новое seed-значение. Если аргумент *number* опускается, то в качестве нового seed-значения используется значение системного таймера.

### Пример

Следующий код генерирует строку из случайных чисел в диапазоне от 1 до 6.

```
Dim MyValue, MyString As String
Randomize
MyString = ""
For I = 0 To 9
    MyValue = Int((6 * Rnd) + 1)
    MyString = MyString & " " & MyValue
Next
```

## ReDim

Используется на уровне процедуры для переопределения размеров динамический массивов.

### Синтаксис

**ReDim** [**Preserve**] *varname(subscripts)* [**As type**] [, *varname(subscripts)* [**As type**]] ...

Синтаксис оператора **ReDim** состоит из следующих элементов:

<b>Preserve</b>	Необязательный. Ключевое слово, используемое для сохранения данных в существующем массиве при изменении последней размерности.
<i>varname</i>	Обязательный. Имя переменной.
<i>subscripts</i>	Обязательный. Размерности массива-переменной; можно объявлять до 60 размерностей. Аргумент <i>subscripts</i> имеет следующий синтаксис: <i>[lower To] upper</i> [, <i>[lower To] upper</i> ] ... Если <i>lower</i> явно не указан, нижняя граница массива определяется оператором <b>Option Base</b> . Если в коде не используется оператор <b>Option Base</b> , нижняя граница считается равной нулю.

*type*            Необязательный. Тип (данных) переменной (массива); может быть **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String** (для строк переменной длины), **String \* length** (для строк фиксированной длины), **Object**, **Variant**, определенным пользователем типом или объектным типом.

Оператор **ReDim** используется для определения и переопределения размера динамического массива, который перед этим уже был формально объявлен при помощи операторов **Private**, **Public** или **Dim** с пустыми скобками (вместо указания размерностей).

Оператор **ReDim** можно применять повторно для изменения числа элементов и размерностей массива. Однако нельзя при помощи оператора **ReDim** изменить тип массива (не для **Variant**-массива). Если массив содержит данные типа **Variant**, тип его элементов можно изменить (предложением **As type**); это не относится к оператору **ReDim**, в котором используется ключевое слово **Preserve**, не допускающее изменения типа данных.

При использовании ключевого слова **Preserve** можно изменить размер только последней размерности; количество самих размерностей изменить нельзя.

#### Пример

```
Dim MyArray() As Integer           ' объявление динамического
массива
Redim MyArray(10)                  ' выделение памяти для массива
...
Redim Preserve MyArray(25)         ' добавление 15-ти элементов
```

---

#### Rem

Используется для включения в код комментариев.

#### Синтаксис

**Rem** *comment*

Можно также использовать следующий синтаксис:

' *comment*

Необязательный аргумент *comment* — текст комментария. Между ключевым словом **Rem** и *comment* обязательно следует помещать пробел.

---

#### Reset

Закрывает все файлы на диске, которые были открыты с использованием оператора **Open**.

#### Синтаксис

**Reset**

Оператор **Reset** закрывает все активные файлы, открытые оператором **Open**, и записывает содержимое всех файловых буферов на диск.

---

#### Resume

Возобновляет выполнение кода после того, как заканчивает работу процедура обработки ошибки.

**Синтаксис**

**Resume** [0]  
**Resume Next** .  
**Resume** *line*

Синтаксис оператора **Resume** может быть одной из следующих форм:

- Resume**            Если ошибка возникла в той же самой процедуре, в которой находится обработчик ошибок, выполнение кода возобновляется с того же оператора, который вызвал ошибку. Если ошибка возникла в вызванной процедуре, выполнение кода возобновляется с оператора, который последним вызвал процедуру, содержащую обработчик ошибок.
- Resume Next**      Если ошибка возникла в той же самой процедуре, в которой находится обработчик ошибок, выполнение кода возобновляется с оператора, находящегося сразу за тем, который вызвал ошибку. Если ошибка возникла в вызванной процедуре, выполнение кода возобновляется с оператора, непосредственно следующего за тем, который последним вызвал процедуру, содержащую обработчик ошибок (или оператор **On Error Resume Next**).
- Resume line**        Выполнение кода возобновляется со строки *line*, заданной в качестве аргумента. Аргумент *line* — метка или номер строки в той же самой процедуре, где находится обработчик.

---

**Примеры**

Далее приведены примеры на все упомянутые в таблице формы оператора **Resume**.

```
Private Sub Test_Resumel()  
    'Использование в обработчике оператора Resume  
    Dim z, y As Integer  
    On Error GoTo Err  
    y = InputBox("Введите число")  
    z = y + 10  
  
    MsgBox z  
    MsgBox "Спасибо!"  
    Exit Sub  
  
Err:  
    If Err.Number = 13 Then  
        MsgBox ("Просили же Вас ввести число!")  
        Resume  
    End If  
End Sub
```

---

```
Private Sub Test_Resume2()  
    'Использование в обработчике оператора Resume Next  
    Dim z, y As Integer  
    On Error GoTo Err  
    y = InputBox("Введите число")  
    z = y + 10  
  
    MsgBox z  
    MsgBox "Спасибо!"
```

```

Exit Sub

Err:
    If Err.Number = 13 Then
        MsgBox ("С Вашего разрешения введем 9999")
        y = 9999
        Resume Next
    End If
End Sub

-----
Public co As Integer
Private Sub Test_Resume3()
    'Использование в обработке оператора Resume line
    Dim z, y As Integer
    On Error GoTo Err
    y = InputBox("Введите число")
    z = y + 10

    Debug.Print z
    Debug.Print "Спасибо!"
    Exit Sub

err_exit:
    Debug.Print "Что-то у Вас не получается"
    Exit Sub

Err:
    If Err.Number = 13 Then
        If co = 2 Then
            Resume err_exit
        Else
            co = co + 1
            Debug.Print ("У Вас осталось " & (3 - co) & " попытки")
            Resume
        End If
    End If
End Sub

Private Sub Form_Load()
    co = 0
End Sub

'Обработчик ошибок в одной из процедур
Private Sub Test_Resume4()
    Call A
End Sub

Sub A()
    On Error GoTo Err
    Call B
    Debug.Print "В любом случае процедура а завершается"
    Exit Sub

Err:
    Debug.Print "Где-то произошла ошибка!"
    Resume Next
End Sub

Sub B()
    Call C

```

```
End Sub

Sub C()
    Dim d As Integer
    d = 1 / 0
End Sub
```

---

## RmDir

---

Удаляет существующий каталог (или папку), не содержащий файлов.

### Синтаксис

**RmDir** *path*

Обязательный аргумент *path* — строковое выражение, которое определяет удаляемый каталог (или папку). Путь может включать драйвер (по умолчанию — текущий). Если каталог содержит файлы (т.е. — не пустой), при выполнении оператора возникает ошибка, которую можно «перехватить» и обработать оператором **On Error**; обработчик, например, может дать пользователю возможность либо удалить файлы каталога (оператором **Kill**), либо отказаться от удаления каталога, если пользователь «поспешил»).

### Пример

```
RmDir "WINDOWS"           'пользователь не любит WINDOWS
```

---

## RSet

---

Выполняет правое выравнивание в строковой переменной. Не может быть использован с типами, определенными пользователем.

### Синтаксис

**RSet** *stringvar* = *string*

Синтаксис оператора **RSet** состоит из следующих элементов:

<i>stringvar</i>	Обязательный. Имя строковой переменной.
<i>string</i>	Обязательный. Строковое выражение, которое выравнивается внутри <i>stringvar</i> .

Если *stringvar* длиннее, чем *string*, оператор **RSet** заменяет все символы с левой стороны строки *stringvar* пробелами.

### Пример (см. пример для LSet)

```
Dim StrMyVar
StrMyVar = "0123456789"      ' инициализация для задания длины строки
Rset StrMyVar = "Right->"    ' StrMyVar теперь: "    Right->"
```

---

## SaveSetting

---

Сохраняет или создает элемент настройки (ключевую настройку) определенного приложения в Windows-реестре.

### Синтаксис

**SaveSetting** *appname*, *section*, *key*, *setting*

Синтаксис оператора **SaveSetting** имеет следующие именованные аргументы:

<i>appname</i>	Обязательный. Строковое, содержащее имя приложения или проекта, для которого предназначены настройки.
<i>section</i>	Обязательный. Строковое выражение, содержащее наименование секции, в которой сохраняется ключевая настройка.
<i>key</i>	Обязательный. Строковое выражение, содержащее имя настройки.
<i>setting</i>	Обязательный. Выражение, содержащее значение ключевой настройки.

---

## Seek

---

Устанавливает позицию для следующей операции чтения/записи в открытом (при помощи оператора **Open**) файле.

### Синтаксис

**Seek** [#]*filenumber*, *position*

Синтаксис оператора **Seek** состоит из следующих элементов:

<i>Filenumber</i>	Обязательный. Любое допустимое число.
<i>position</i>	Обязательный. Число в диапазоне 1 – 2,147,483,647, включительно, которое указывает, номер элемента (записи) для операции чтения/записи.

### Пример

В следующем примере описывается пользовательский тип данных (**Record**), процедура записи (**TestPut**) данных в файл и процедура чтения (**TestSeek**) данных из файла.

```
Type Record                                ' Определенный пользователем
тип
    ID As Integer
    FirstName As String * 20
    LastName As String * 20
End Type

Sub TestPut()
'Tестирование Put
    Dim MyRecord As Record, RecordNumber

    Open "TESTFILE" For Random As #1 Len = Len(MyRecord)

    MyRecord.FirstName = "Петр"
    MyRecord.LastName = "Петров"
    MyRecord.ID = 333
    Put #1, 2, MyRecord

    MyRecord.FirstName = "Сидор"
    MyRecord.LastName = "Сидоров"
    MyRecord.ID = 111
    Put #1, 3, MyRecord

    MyRecord.FirstName = "Иван"
    MyRecord.LastName = "Иванов"
    MyRecord.ID = 222
    Put #1, 1, MyRecord
```



```

    Close #1
End Sub

Sub TestSeek()
'тестирование Get
    Dim MyRecord As Record, MaxSize As Integer

    'чтение записей без Seek:
    Open "TESTFILE" For Random As #1 Len = Len(MyRecord)
    For i = 1 To 3
        Get #1, i, MyRecord
        Debug.Print MyRecord.FirstName & " : " & MyRecord.ID
    Next
    Close #1

    'чтение с Seek:
    Open "TESTFILE" For Random As #1 Len = Len(MyRecord)
    'определить число записей в файле:
    MaxSize = LOF(1) \ Len(MyRecord)
    'чтение (и печать) записей в цикле в обратном порядке
    For RecordNumber = MaxSize To 1 Step -1
        Seek #1, RecordNumber 'Установка позиции
        Get #1, , MyRecord 'чтение записи
        Debug.Print MyRecord.FirstName & " : " & MyRecord.ID
    Next RecordNumber
    Close #1
End Sub

```

---

## Select Case

---

Выполняет одну или несколько групп операторов в зависимости от тестируемого выражения.

### Синтаксис

```

Select Case testexpression
    [Case expressionlist-n
        [statements-n]] ...
    [Case Else
        [elsestatements]]
End Select

```

Синтаксис оператора **Select Case** состоит из следующих элементов:

- Testexpression.** Обязательный. Любое числовое или строковое выражение.
- expressionlist-n** Обязателен в случае использования **Case**. Список (с разделителями) из одной или более следующих форм:
- ```

expression
expression To expression
Is comparisonoperator expression

```
- Ключевое слово **To** определяет диапазон значений, в котором меньшее значение должно быть перед ключевым словом **To**. Ключевое слово **Is** следует использовать с операторами сравнения (кроме, **Is** и **Like**) для указания диапазона значений. Если этот аргумент не используется, автоматически подставляется ключевое слово **Is**.
- statements-n** Необязательный. Один или более операторов, выполняемых, если **testexpression** совпадает с любым значением из **expressionlist-n**.
- elsestatements** Необязательный. Один или более операторов, выполняемых, если **testexpression** не совпадает ни с каким **Case**-предложением.

Если *testexpression* совпадает с любым из **Case**-выражений *expressionlist*, выполняются операторы *statements*, следующие за этим **Case**-предложением (до следующего **Case**-предложения или последнего **Case**-предложения — до **End Select**). Управление выполнением программы передается оператору, следующему за предложением **End Select**. Если *testexpression* совпадает с *expressionlist* в нескольких **Case**-предложениях, выполняются операторы, соответствующие первому совпадению. Предложение **Case Else** используется для указания группы операторов *elsestatements*, которые следует выполнять, если ни в одном **Case**-предложении не нашлось совпадений.

Конструкция **Select Case** допускает вложения.

## SendKeys

Посылает одно или более нажатий клавиш активному окну, как-будто они выполнены с клавиатуры.

### Синтаксис

**SendKeys** *string* [, *wait*]

Синтаксис оператора **SendKeys** включает следующие именованные аргументы:

|               |                                                                                                                                                                                                                                                                                                               |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>string</i> | Обязательный. Строковое выражение, определяющее посылаемые нажатия клавиш.                                                                                                                                                                                                                                    |
| <i>wait</i>   | Необязательный. Значение типа <b>Boolean</b> , определяющее режим ожидания (wait mode). Если — <b>False</b> (по умолчанию), управление возвращается процедуре немедленно после передачи клавиш. Если — <b>True</b> , нажатия клавиш должны быть обработаны прежде, чем управление будет возвращено процедуре. |

Каждая клавиша представляет один или более символов. Для указания одиночного символа клавиатуры используйте сам символ. Например, для представления символа N используйте строку "N". Для представления нескольких символов используйте строку, например, "ABCD".

Знаки «плюс» (+), «каре» (^), «процент» (%), «тильда» (~), «парные скобки» ( ) имеют для функции **SendKeys** специальные значения. Для указания любого из этих символов заключайте их в фигурные скобки ({}). Например, для определения знака «плюс» используйте строку "{+}". Квадратные скобки ([ ]) не имеют специального значения для функции **SendKeys**, но их также следует заключать в фигурные скобки.

Для указания символов, которые не отображаются при нажатии на них (ENTER или TAB), и символов, которые представляют действие, а не символ, используйте следующие коды:

| Клавиши        | Коды                            | Клавиши  | Коды  |
|----------------|---------------------------------|----------|-------|
| BACKSPACE      | {BACKSPACE}, {BS}<br>или {BKSP} | TAB      | {TAB} |
| BREAK          | {BREAK}                         | UP ARROW | {UP}  |
| CAPS LOCK      | {CAPSLOCK}                      | F1       | {F1}  |
| DEL или DELETE | {DEL} или {DELETE}              | F2       | {F2}  |
| DOWN ARROW     | {DOWN}                          | F3       | {F3}  |

| Клавиши        | Коды               | Клавиши | Коды  |
|----------------|--------------------|---------|-------|
| END            | {END}              | F4      | {F4}  |
| ENTER          | {ENTER} или ~      | F5      | {F5}  |
| ESC            | {ESC}              | F6      | {F6}  |
| HELP           | {HELP}             | F7      | {F7}  |
| HOME           | {HOME}             | F8      | {F8}  |
| INS или INSERT | {INS} или {INSERT} | F9      | {F9}  |
| LEFT ARROW     | {LEFT}             | F10     | {F10} |
| NUM LOCK       | {NUMLOCK}          | F11     | {F11} |
| PAGE DOWN      | {PGDN}             | F12     | {F12} |
| PAGE UP        | {PGUP}             | F13     | {F13} |
| PRINT SCREEN   | {PRTSC}            | F14     | {F14} |
| RIGHT ARROW    | {RIGHT}            | F15     | {F15} |
| SCROLL LOCK    | {SCROLLLOCK}       | F16     | {F16} |

Для определения комбинаций клавиш с клавишами SHIFT, CTRL и ALT предваряйте код клавиши следующими знаками:

| Клавиша | Знак |
|---------|------|
| SHIFT   | +    |
| CTRL    | ^    |
| ALT     | %    |

Для указания того, что некоторая комбинация клавиш SHIFT, CTRL и ALT остается нажатой в то время, как нажимаются другие клавиши, заключайте коды этих клавиш в скобки. Например, чтобы указать, что при нажатой клавише SHIFT «нажимаются» клавиши **Е** и **Я**, следует использовать строку "+(ЕЯ)". А чтобы указать, что при нажатой клавише SHIFT нажимается клавиша **Е**, а затем — только **Я** (без SHIFT), следует использовать строку "+ЕЯ".

Для определения повторяющихся клавиш используйте следующую форму {клавиша число}. Например, {LEFT 43} означает нажатие клавиши LEFT ARROW (стрелка влево) 43 раза; {Y 12} означает нажатие клавиши Y 12 раз.

**Пример**

В этом примере с использованием функции Shell запускается приложение Microsoft Outlook и «нажимается» кнопка Send/Receive сочетанием клавиш Alt и C.

```
Sub TestSendKeys()  
    'Запустить почтовую программу  
    ReturnValue = Shell("OUTLOOK.EXE", 1)  
    'Нажать на кнопку Send/Receive  
    SendKeys "%C", True  
End Sub
```

---

**Set**

---

Присваивает объекту ссылку на переменную или свойство.

**Синтаксис**

**Set** *objectvar* = {[**New**] *objectexpression* | **Nothing**}

Синтаксис оператора **Set** состоит из следующих элементов:

|                         |                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>objectvar</i>        | Обязательный. Имя переменной или свойства; должно соответствовать стандартным соглашениям об именовании переменных.                                                                                                                                                                                                                                                                  |
| <b>New</b>              | Необязательный. <b>New</b> обычно используется для неявного создания объекта. Если <b>New</b> используется вместе с <b>Set</b> , создается новый экземпляр класса. Если <i>objectvar</i> содержал ссылку на некоторый объект, эта ссылка удаляется, когда присваивается новая. Ключевое слово <b>New</b> нельзя использовать для создания нового экземпляра встроенного типа данных. |
| <i>objectexpression</i> | Обязательный. Выражение, состоящее из имени объекта, другой объявленной переменной того же объектного типа или функции (метода), которая возвращает объект того же типа.                                                                                                                                                                                                             |
| <b>Nothing</b>          | Необязательный. Прекращение связи <i>objectvar</i> с любым указанным объектом.                                                                                                                                                                                                                                                                                                       |

Переменная *objectvar* должна быть объектного типа, совместимого с объектом, который ей присваивается. Операторы **Dim**, **Private**, **Public**, **ReDim** и **Static** только объявляют переменную, которая ссылается на объект. Действительное присваивание ссылки на определенный объект выполняет оператор **Set**.

**Пример**

```
Dim xlBook As Object
Dim xlSheet As Object
Set xlApp = CreateObject("Excel.Application")
Set xlBook = xlApp.Workbooks.Add
Set xlSheet = xlBook.WorkSheets(1)
```

---

**SetAttr**

---

Устанавливает информацию об атрибутах для файла.

**Синтаксис**

**SetAttr** *pathname*, *attributes*

Синтаксис оператора **SetAttr** включает следующие именованные аргументы:

|                   |                                                                                                      |
|-------------------|------------------------------------------------------------------------------------------------------|
| <i>pathname</i>   | Обязательный. Строковое выражение, определяющее имя файла (может включать каталог и драйвер диска).  |
| <i>attributes</i> | Обязательный. Константное или числовое выражение, которое представляет сумму определенных атрибутов. |

Аргумент *attributes* может принимать следующие значения:

| Константа         | Значение | Назначение                                     |
|-------------------|----------|------------------------------------------------|
| <b>VbNormal</b>   | 0        | Normal (по умолчанию) [обычный].               |
| <b>VbReadOnly</b> | 1        | Read-only [для чтения/записи].                 |
| <b>VbHidden</b>   | 2        | Hidden [скрытый].                              |
| <b>VbSystem</b>   | 4        | System [системный] файл.                       |
| <b>VbArchive</b>  | 32       | Файл изменялся со времени последней архивации. |

### Пример

```
SetAttr "C:\Мои документы\TESTFILE", vbHidden
SetAttr "C:\Мои документы\TESTFILE", vbHidden + VbSystem
```

### Static

Используется на уровне процедуры для объявления переменных и выделяет для них память. Переменные, объявляемые при помощи оператора **Static**, сохраняют свои значения в течение всего времени выполнения программы.

#### Синтаксис

```
Static varname([([subscripts])]) [As [New] type] [,
varname([([subscripts])]) [As [New] type]] ...
```

Синтаксис оператора **Static** включает следующие элементы:

- varname** Обязательный. Имя переменной; должно соответствовать стандартным соглашениям об именовании переменных.
- subscripts** Необязательный. Размерность переменной-массива; можно объявить до 60 размерностей. Аргумент *subscripts* использует следующий синтаксис: *[lower To] upper* [, *[lower To] upper*] ...  
*lower* — нижний предел допустимых индексов массива; *upper* — верхний предел (является обязательным) для индексов массива. При наличии в операторе **Dim** только *upper* (нижний предел) определяется в зависимости от установки оператора **Option Base**.
- New** Необязательный. Ключевое слово, позволяющее неявно создать объект. Если вы используете **New** при объявлении объектной переменной, создается новый экземпляр объекта, поэтому нет необходимости использовать оператор **Set** для назначения объекту ссылки.
- type** Необязательный. Тип данных переменной; может быть типом **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String** (только переменной длины), **String \* length** (для строк фиксированной длины), **Object**, **Variant**, типом, определенным пользователем, или объектным типом.

Во время выполнения кода модуля переменные, объявленные оператором **Static**, сохраняют свои значения до «сброса» или перезапуска модуля. Переменные, объявленные оператором **Static** в модуле класса, сохраняют свои значения в каждом экземпляре класса до удаления экземпляра. Переменные, объявленные оператором **Static** в модуле формы, сохраняют свои значения до закрытия формы.

---

## Stop

---

Останавливает выполнение кода.

### Синтаксис

#### Stop

Для приостановки выполнения кода (в целях отладки) можно помещать оператор **Stop** в любом месте программы. Использование этого оператора подобно использованию точки прерывания (breakpoint). Оператор **Stop** останавливает выполнение программы, но в отличие от оператора **End** этот оператор не закрывает файлы и не очищает переменные (за исключением компилированных исполняемых файлов).

### Пример

```
Sub TestStop()
    Dim I
    For I = 1 To 10
        Debug.Print I & "    Для продолжения выполните Run|Continue"
        Stop
    Next I
End Sub
```

---

## Sub

---

Объявляет имя, аргументы и код тела **Sub**-процедуры.

### Синтаксис

```
[Private | Public | Friend] [Static] Sub name [(arglist)]
    [statements]
    [Exit Sub]
    [statements]
End Sub
```

Синтаксис оператора **Sub** включает следующие элементы:

|                   |                                                                                                                                                                                                                           |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Public</b>     | Необязательный. Указывает, что <b>Sub</b> -процедура доступна всем другим процедурам во всех модулях. Если используется в модуле, который содержит оператор <b>Option Private</b> , эта процедура недоступна вне проекта. |
| <b>Private</b>    | Необязательный. Указывает, что процедура доступна другим процедурам только того модуля, в котором она объявлена.                                                                                                          |
| <b>Friend</b>     | Необязательный. Используется только в модуле класса. Указывает, что <b>Sub</b> -процедура видима во всем проекте.                                                                                                         |
| <b>Static</b>     | Необязательный. Указывает, что локальные переменные процедуры сохраняют свои значения между вызовами.                                                                                                                     |
| <i>name</i>       | Обязательный. Имя <b>Sub</b> -процедуры; должно соответствовать стандартным соглашениям об именовании переменных.                                                                                                         |
| <i>arglist</i>    | Необязательный. Список переменных, представляющих аргументы, передаваемые процедуре при ее вызове.                                                                                                                        |
| <i>statements</i> | Необязательный. Любой набор операторов, которые будут выполняться при вызове процедуры.                                                                                                                                   |

Синтаксис параметра *arglist* имеет вид и включает:

```
[Optional] [ByVal | ByRef] [ParamArray] varname[( )] [As type]
[= defaultvalue]
```

|                     |                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Optional</b>     | Необязательный. Ключевое слово, указывающее, что аргумент является необязательным. Если используется, все последующие аргументы в <i>arglist</i> должны быть также необязательными и объявляться с ключевым словом <b>Optional</b> . Если применяется <b>ParamArray</b> , слово <b>Optional</b> не может использоваться ни для какого аргумента.                                                     |
| <b>ByVal</b>        | Необязательный. Указывает, что аргумент передается по значению.                                                                                                                                                                                                                                                                                                                                      |
| <b>ByRef</b>        | Необязательный. Указывает, что аргумент передается по ссылке. Используется по умолчанию.                                                                                                                                                                                                                                                                                                             |
| <b>ParamArray</b>   | Необязательный. Используется только в качестве последнего аргумента в списке <i>arglist</i> для указания того, что последний аргумент является <b>Optional</b> -массивом <b>Variant</b> -элементов. Ключевое слово <b>ParamArray</b> позволяет использовать переменное число аргументов и не может применяться с <b>ByVal</b> , <b>ByRef</b> или <b>Optional</b> .                                   |
| <i>varname</i>      | Обязательный. Имя переменной, передаваемой процедуре; должно удовлетворять стандартным соглашениям об именовании переменных.                                                                                                                                                                                                                                                                         |
| <i>type</i>         | Необязательный. Тип аргумента, передаваемого процедуре; может быть типом <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>String</b> (только переменной длины), <b>Object</b> , <b>Variant</b> или объектным типом. Если параметр не описан как <b>Optional</b> , может быть указан тип, определенный пользователем. |
| <i>defaultvalue</i> | Необязательный. Любая константа или константное выражение. Допускается только для аргументов, описанных как <b>Optional</b> . Если типом является <b>Object</b> , то явное значение по умолчанию может быть только значение <b>Nothing</b> .                                                                                                                                                         |

**Sub-процедуры** — это основные программные единицы на языке Visual Basic. Они обеспечивают функционирование диалоговых форм, выполняют действия общего характера, вызывая при необходимости другие **Sub-процедуры** и **Function-процедуры**, обеспечивают работу с экземплярами классов.

Если явно не определено использование **Public**, **Private** или **Friend**, **Sub-процедура** является **Public** по умолчанию. Если не используется **Static**, значения локальных переменных не сохраняются между вызовами. Ключевое слово **Friend** может быть использовано только в модулях класса. Однако **Friend-процедуры** могут быть доступны процедурам в любом модуле проекта.

**Sub-процедуры** могут быть рекурсивными. Это означает, что они могут вызывать сами себя. Иногда в результате рекурсивных вызовов может происходить переполнение программного стека. Ключевое слово **Static** обычно не используется при описании рекурсивных **Sub-процедур**.

Весь выполняемый код помещается в теле процедуры. Нельзя определить **Function-процедуру** внутри другой **Function-**, **Sub-** или **Property-процедуры**.

Оператор **Exit Sub** приводит к немедленному завершению **Sub-процедуры**. При этом программа продолжает выполняться с оператора, следующего за тем, который вызвал **Sub-процедуру**. В качестве альтернативных выходов внутри **Sub-процедуры** может встречаться любое количество операторов **Exit Sub**.

Подобно **Function**-процедуре **Sub**-процедура — это отдельная процедура, которая может принимать аргументы, выполнять ряд операторов и изменять значения ее аргументов. Однако в отличие от **Function**-процедуры имя **Sub**-процедуры не может появляться в правой части операции присваивания в составе некоторого выражения.

---

**Time**

---

Устанавливает системное время.

**Синтаксис**

**Time** = *time*

Обязательный аргумент *time* — любое числовое, строковое выражение или любая комбинация, представляющая время. Если *time* — строка, оператор **Time** «пытается» преобразовать ее во время, используя разделители времени, которые определены в вашей системе.

**Пример**

```
TimeMy = #4:30:20 PM#  
Time = TimeMy
```

---

**Type**

---

Используется на уровне модуля для определения типа данных пользователя.

**Синтаксис**

```
[Private | Public] Type varname  
    elementname [[subscripts]] As type  
    [elementname [[subscripts]] As type]  
    ...  
End Type
```

Синтаксис оператора **Type** включает следующие элементы:

- |                    |                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Public</b>      | Необязательный. Используется для объявления пользовательских типов, доступных всем процедурам во всех модулях всех проектов.                                                                                                                                                                                                                                                       |
| <b>Private</b>     | Необязательный. Используется для объявления пользовательских типов, доступных только в модуле, где они объявляются.                                                                                                                                                                                                                                                                |
| <i>varname</i>     | Обязательный. Имя определенного пользователем типа; должно соответствовать стандартным соглашениям об именовании переменных.                                                                                                                                                                                                                                                       |
| <i>elementname</i> | Обязательный. Имя элемента пользовательского типа. Имена элементов также должны соответствовать стандартным соглашениям об именовании переменных.                                                                                                                                                                                                                                  |
| <i>subscripts</i>  | Если аргумент <i>lower</i> явно не указывается, нижняя граница массива определяется оператором <b>Option Base</b> . Нижняя граница равна 0, если оператор <b>Option Base</b> не используется.                                                                                                                                                                                      |
| <i>type</i>        | Обязательный. Тип данных элемента; может быть типом <b>Byte</b> , <b>Boolean</b> , <b>Integer</b> , <b>Long</b> , <b>Currency</b> , <b>Single</b> , <b>Double</b> , <b>Date</b> , <b>String</b> (только переменной длины), <b>String</b> * <i>length</i> (для строк фиксированной длины), <b>Object</b> , <b>Variant</b> , типом, определенным пользователем, или объектным типом. |



После того как с использованием оператора **Type** (можно использовать только на модульном уровне) описан пользовательский тип, вы можете объявлять переменные этого типа в любом месте внутри области действия описания типа. При этом следует использовать операторы **Dim**, **Private**, **Public**, **ReDim** или **Static**. В стандартных модулях и модулях класса типы пользователя являются public-типами по умолчанию, что легко изменить, используя ключевое слово **Private**.

### Пример

```
Type Record      ' Определенный пользователем тип
    ID As Integer
    FirstName As String * 20
    LastName As String * 20
End Type
```

---

## Unload

---

Удаляет объект из памяти.

### Синтаксис

**Unload** *object*

Обязательный аргумент *object* является объектным выражением, которое вычисляется до объекта в **Applies To** list.

### Пример

```
Private Sub CanselCommand_Click()
    Unload me 'удалить из памяти текущий диалог
End Sub
```

---

## While...Wend

---

Выполняет серию операторов, пока некоторое условное выражение является равным **True**.

### Синтаксис

```
While condition
    [statements]
Wend
```

Синтаксис оператора **While...Wend** включает следующие элементы:

- |                   |                                                                                                                                                                                                                   |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>condition</i>  | Обязательный. Числовое или строковое выражение, результатом вычисления которого являются значения <b>True</b> или <b>False</b> . Если <i>condition</i> — <b>Null</b> , то оно интерпретируется как <b>False</b> . |
| <i>statements</i> | Необязательный. Один или более операторов, выполняемых, если условию является равным значению <b>True</b> .                                                                                                       |

Если *condition* является равным значению **True**, выполняются все операторы между словами **While** и **Wend**. Управление выполнением программы передается в начало структуры **While...Wend** и значение *condition* снова проверяется. Если оно остается равным значению **True**, процесс повторяется, иначе управление передается на оператор, следующий за словом **Wend**. Оператор **While...Wend** допускает вложения.

**Пример**

```
Sub TestWhileWend()  
    Dim i As Integer  
    i = 10  
    While i < 20  
        i = i + 2  
        MsgBox i  
    Wend  
    MsgBox "Последнее выданное значение должно быть равно 20"  
End Sub
```

---

**Width #**

Устанавливает ширину выводимой в файл строки (файл открывается оператором **Open**).

**Синтаксис**

**Width #** *filenumber*, *width*

Синтаксис оператора **Width #** включает следующие элементы:

|                   |                                                                                                                                                                                                                                                                                                                                |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>filenumber</i> | Обязательный. Любой допустимый файловый номер.                                                                                                                                                                                                                                                                                 |
| <i>width</i>      | Обязательный. Числовое выражение в диапазоне 0–255, включительно, которое указывает количество символов, появляющихся в строке перед началом новой строки (т.е. количество символов в строке). Если значение <i>width</i> равно 0, то длина строки не имеет ограничений. По умолчанию значение <i>width</i> является равным 0. |

---

**With**

Выполняет несколько операторов для единственного объекта или пользовательского типа.

**Синтаксис**

```
With object  
    [statements]  
End With
```

Синтаксис оператора **With** включает следующие элементы:

|                   |                                                                           |
|-------------------|---------------------------------------------------------------------------|
| <i>Object</i>     | Обязательный. Имя объекта или пользовательского типа.                     |
| <i>Statements</i> | Необязательный. Один или более операторов, которые должны быть выполнены. |

Оператор **With** позволяет выполнить несколько операторов для определенного объекта без повторного указания имени объекта. Так, например, удобно сразу изменить несколько свойств некоторого объекта.

---

**Write #**

Записывает данные в файл с последовательным доступом.

**Синтаксис**

**Write #** *filenumber*, [*outputlist*]

Синтаксис оператора **Write #** состоит из следующих элементов:

|                   |                                                                                                           |
|-------------------|-----------------------------------------------------------------------------------------------------------|
| <i>filename</i>   | Обязательный. Любой допустимый файловый номер.                                                            |
| <i>outputlist</i> | Необязательный. Одно или более (разделенных запятыми) числовых или строковых выражений для записи в файл. |

Данные, которые записываются в файл оператором **Write #**, обычно считываются оператором **Input #**. Если аргумент *outputlist* не используется, а после *filename* помещается запятая, в файл записывается пустая строка.

Чтобы при использовании оператора **Write #** для записи в файл данные корректно считывались оператором **Input #**, выполняются следующие правила:

- Числовые данные всегда записываются с использованием точки в качестве десятичного разделителя.
- Для данных типа **Boolean** печатается либо **#TRUE#**, либо **#FALSE#**.
- Данные типа **Date** записываются в файл с использованием универсального формата даты. Если компонента даты или времени отсутствует или нулевая, в файл записывается только предоставленная часть.
- В файл ничего не записывается, если значение *outputlist* имеет значение **Empty**. Однако для **Null**-данных записывается **#NULL#**.

Для **Error**-данных в файле появляется **#ERROR errorcode#**.

В отличие от оператора **Print #** оператор **Write #** вставляет запятые между элементами и кавычки вокруг строк. Вам не нужно использовать явно разделители в списке. Оператор **Write #** вставляет символ новой строки, т.е. **Chr(13) + Chr(10)**, после того, как в файл будет записан последний символ из *outputlist*.

### Пример

```
Sub TestPrint()  
'запись данных в файл TESTFILE  
    Dim TextLine  
    Open "TESTFILE" For Output As #1  
    Write #1, "Напрасно я ищу повсюду развлечения,      "  
    Write #1, "Пестреет и жужжит толпа передо мной ..."  
    Write #1, "Но сердце холодно, и спит воображение:  "  
    Write #1, "Они все чужды мне, и я им всем чужой!    "  
    Write #1,  
    Write #1, Tab(10); "М.Ю. Лермонтов"  
    Close #1  
End Sub  
  
Sub TestInput()  
Dim StrMy  
Open "TESTFILE" For Input As #1 'открыть файл  
Do While Not EOF(1)             'цикл до конца файла  
    Input #1, StrMy              'считать данные  
    Debug.Print StrMy            'результат -> на экран  
Loop  
Close #1                        'закрыть файл  
End Sub
```

# Приложение Б

## Функции в VBA

---

### Abs

---

Возвращает абсолютное значение своего аргумента. (Абсолютное значение числа — это его значение без знака.)

#### Синтаксис

**Abs** (*number*)

Обязательный аргумент *number* может быть любым допустимым арифметическим выражением. Если *number* содержит **Null**, возвращается также **Null**; если — неинициализированная переменная, то возвращается нулевое значение.

---

### Array

---

Возвращает **Variant**-массив, значениями которого являются элементы аргумента-списка.

#### Синтаксис

**Array** (*arglist*)

Обязательный аргумент *arglist* — это разделенный запятыми список значений, которые назначаются элементам массива. Значение нижней границы (индекса) массива определяется опцией оператора **Option Base**.

#### Примеры

|                       |                                  |
|-----------------------|----------------------------------|
| Dim A As Variant      | ' объявление A как типа Variant  |
| A = Array(10, 20, 30) | ' теперь A — массив              |
| B = A(2)              | ' в B записывается значение A(2) |

---

### Asc

---

Возвращает числовой (тип **Integer**) код первого символа строки-аргумента.

#### Синтаксис

**Asc** (*string*)

Обязательный аргумент *string* — любое допустимое строковое выражение. Если аргумент не содержит ни одного символа, вызывается ошибка времени исполнения.

Диапазон возвращаемых значений: 0 – 255 для не DBCS-систем и –32768 – 32767 для DBCS-систем.

Функция **AscB** используется с байтовыми данными, содержащимися в строке, и возвращает не символьный код для первого символа, а первый байт. Функция **AscW** Unicode-код символа, если Unicode-символы поддерживаются данной платформой.

### Примеры

```
Dim MyNumber
MyNumber = Asc("B")           'возвращает 66
MyNumber = Asc("a")           'возвращает 97
MyNumber = Asc("ABCD")        'возвращает 65
```

### Atn

Возвращает арктангенс (тип **Double**) заданного аргумента.

#### Синтаксис

**Atn** (*number*)

Обязательный аргумент *number* — переменная типа **Double** или любое допустимое арифметическое выражение. Функция **Atn** (обратная функции **Tan**) принимает отношение двух сторон прямоугольного треугольника и возвращает значение соответствующего угла в радианах (в диапазоне от  $-\pi/2$  до  $\pi/2$ ).

#### Пример

```
pi = 4 * Atn(1)      ' pi
```

### CallByName

Выполняет метод некоторого объекта или устанавливает или возвращает свойство объекта.

#### Синтаксис

**CallByName** (*object*, *procname*, *calltype*, [*args()*])

Функция **CallByName** принимает следующие именованные аргументы:

|                 |                                                                                             |
|-----------------|---------------------------------------------------------------------------------------------|
| <i>object</i>   | Обязательный; тип — <b>Variant (Object)</b> . Имя объекта, для которого вызывается функция. |
| <i>procname</i> | Обязательный; тип — <b>Variant (String)</b> . Строка с именем свойства или метода объекта.  |
| <i>calltype</i> | Обязательный; Константа типа <b>vbCallType</b> , представляющая тип вызываемой процедуры.   |
| <i>args()</i>   | Необязательный; тип — <b>Variant (Array)</b> .                                              |

#### Пример

Для проверки работы этой функции поместите на форме три кнопки с именами, которые Редактор VB задаст им по умолчанию, и напишите простые программы обработки событий:

```
Private Sub CommandButton1_Click()
    MsgBox "Свойство Enabled кнопки: " &
        CallByName(CommandButton2, "Enabled", VbGet)
End Sub
Private Sub CommandButton3_Click()
    If CallByName(CommandButton2, "Enabled", VbGet) Then
        CallByName CommandButton2, "Enabled", VbLet, False
    Else
        CallByName CommandButton2, "Enabled", VbLet, True
    End If
End Sub
```

Когда вы запустите это приложение (командой **Run** | **Run Sub/UserForm**), кнопка с именем **CommandButton1** позволит вывести на экран свойство **Enabled** кнопки **CommandButton2**, а кнопка с именем **CommandButton3** будет последовательно изменять свойство **Enabled** кнопки **CommandButton2**. Обратите внимание на различие в синтаксисе функции при установке и считывании свойства.

## Функции преобразования типов

### Синтаксис

**CBool** (*expression*)  
**CByte** (*expression*)  
**CCur** (*expression*)  
**CDate** (*expression*)  
**Cdbl** (*expression*)  
**CDec** (*expression*)  
**CInt** (*expression*)  
**CLng** (*expression*)  
**CSng** (*expression*)  
**CStr** (*expression*)  
**CVar** (*expression*)

Функции преобразуют данные из одних типов в другие. Обязательный аргумент *expression* — любое строковое или численное (арифметическое) выражение.

### Возвращаемые типы

Тип возвращаемых данных определяется именем функции:

| Функция      | Тип возвращаемых данных | Функция     | Тип возвращаемых данных |
|--------------|-------------------------|-------------|-------------------------|
| <b>CBool</b> | <b>Boolean</b>          | <b>CInt</b> | <b>Integer</b>          |
| <b>CByte</b> | <b>Byte</b>             | <b>CLng</b> | <b>Long</b>             |
| <b>CCur</b>  | <b>Currency</b>         | <b>CSng</b> | <b>Single</b>           |
| <b>CDate</b> | <b>Date</b>             | <b>CStr</b> | <b>String</b>           |
| <b>Cdbl</b>  | <b>Double</b>           | <b>CVar</b> | <b>Variant</b>          |
| <b>CDec</b>  | <b>Decimal</b>          |             |                         |

Обычно функции преобразования типов данных используются разработчиком VBA-приложения для документирования преобразований в коде, поскольку преобразования, выполняемые по умолчанию, могут быть не понятны при чтении кода.

## Choose

Выбирает и возвращает значение из списка аргументов.

### Синтаксис

**Choose** (*index*, *choice-1* [, *choice-2*, ... [, *choice-n*]])

Функция **Choose** принимает следующие аргументы:

- index*            Обязательный. Численное выражение, результатом которого является значение от 1 до числа возможных вариантов для выбора.
- choice*            Обязательный. **Variant**-выражение, содержащее один из возможных выборов.

**Пример**

```
Dim Ind As Integer
Ind = 2
'функция MsgBox выдаст диалоговое окно с "United"
MsgBox Choose(Ind, "Speedy", "United", "Federal")
```

---

**Chr**

---

Возвращает **String**-значение (строку), содержащее символ, код которого указан в качестве аргумента.

**Синтаксис**

**Chr** (*charcode*)

Обязательный аргумент *charcode* — это **Long**-значение кода символа. Диапазон значений аргумента: 0 – 255. Однако в DBCS-системах этот диапазон: от –32768 до 65535.

Функция **ChrB** используется с байтовыми данными, содержащимися в строке, и всегда возвращает не символ, который может быть одним или двумя байтами, а единственный байт. Функция **ChrW** возвращает строку, содержащую Unicode-символ, если Unicode-символы поддерживаются данной платформой.

---

**Cos**

---

Возвращает значение (тип **Double**) косинуса угла-аргумента.

**Синтаксис**

**Cos** (*number*)

Обязательный аргумент *number* — допустимое численное выражение, представляющее угол в радианах.

---

**CreateObject**

---

Создает и возвращает ссылку на ActiveX-объект.

**Синтаксис**

**CreateObject** (*class*, [*servername*])

Функция **CreateObject** имеет следующие аргументы:

|                   |                                                                                                                                                                            |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>class</i>      | Обязательный; тип <b>Variant (String)</b> . Имя приложения и класс объекта для создания.                                                                                   |
| <i>servername</i> | Необязательный; тип <b>Variant (String)</b> . Имя сервера сети, где должен быть создан объект. Если <i>servername</i> — пустая строка (""), используется локальная машина. |

Аргумент *class* использует синтаксис *appname.objecttype*, где:

|                   |                                                                                      |
|-------------------|--------------------------------------------------------------------------------------|
| <i>appname</i>    | Обязательный; тип <b>Variant (String)</b> . Имя приложения, предоставляющего объект. |
| <i>objecttype</i> | Обязательный; тип <b>Variant (String)</b> . Тип или класс объекта для создания.      |

Любое приложение, поддерживающее Automation, предоставляет, по крайней мере, один тип объекта. Например, приложение текстового процессора может предоставить объект **Application**, **Document** и **Toolbar**.

Чтобы создать ActiveX-объект, присвойте объект, возвращаемый функцией **CreateObject**, объектной переменной:

```
Dim Excel_Sheet As Object
Set Excel_Sheet = CreateObject("Excel.Sheet")
```

Объявление объектной переменной при помощи **As Object** создает переменную, содержащую ссылку на объект любого типа. Связывание этой ссылки с конкретным объектом происходит при выполнении программы (позднее связывание).

### Пример

Чтобы создать объектную переменную при помощи раннего связывания, объявляйте объектную переменную как определенный ID-класс:

```
Dim xlApplication As Excel.Application
Dim xlBook As Excel.Workbook
Dim xlSheet As Excel.WorkSheet
Set xlApplication = CreateObject("Excel.Application")
Set xlBook = xlApplication.Workbooks.Add
Set xlSheet = xlBook.Worksheets(1)
```

---

## CurDir

Возвращает значение типа **Variant (String)**, содержащее текущий путь (current path).

### Синтаксис

**CurDir** [(drive)]

Необязательный аргумент *drive* — строковое выражение, определяющее драйвер. Если этот аргумент не задан или является пустой строкой (""), функция **CurDir** возвращает путь для текущего драйвера.

### Пример

```
Dim My_Path
My_Path = CurDir           ' Returns "C:\WIN2000\SYSTEM".
My_Path = CurDir("C")     ' Returns "C:\WIN2000\SYSTEM".
My_Path = CurDir("D")     ' Returns "D:\WORD".
```

---

## CVErr

Возвращает значение (тип **Variant**) подтипа **Error**, содержащее номер ошибки, определенный пользователем.

### Синтаксис

**CVErr** (errornumber)

Обязательный аргумент *errornumber* — любое допустимое error-число.

Функцию **CVErr** следует использовать для описания ошибок в пользовательских процедурах. Например, если вы создали функцию с несколькими аргументами, то можно при этом написать вспомогательную функцию, которая проверяет количество передаваемых аргументов и нахождение значений аргументов в допустимых диапазонах. В случае нарушения правил передачи параметров вы можете использовать функцию **CVErr**, для возвращения номера возникшей ошибки, который вы сами и зададите.



---

**Date**


---

Возвращает значение [тип **Variant (Date)**], содержащее текущую системную дату.

**Синтаксис****Date**

Можно также использовать эту функцию как процедуру для установки системных часов компьютера.

---

**DateAdd**


---

Возвращает значение [тип **Variant (Date)**], содержащее дату, к которой добавлен заданный интервал времени.

**Синтаксис**

**DateAdd**(*interval*, *number*, *date*)

Функция **DateAdd** принимает следующие именованные аргументы:

|                 |                                                                                                                                      |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>interval</i> | Обязательный. Строковое выражение, определяющее вид интервала, который необходимо использовать.                                      |
| <i>number</i>   | Обязательный. Числовое выражение, определяющее количество добавляемых интервалов. Может быть как положительным, так и отрицательным. |
| <i>date</i>     | Обязательный. Значение [ <b>Variant (Date)</b> ], определяющее дату, к которой добавляется временной интервал.                       |

Аргумент *interval* может принимать следующие значения:

| значение | описание                | значение | описание              |
|----------|-------------------------|----------|-----------------------|
| yyyy     | Year (год)              | w        | Weekday (день недели) |
| q        | Quarter (квартал)       | ww       | Week (неделя)         |
| m        | Month (месяц)           | h        | Hour (час)            |
| y        | Day of year (день года) | n        | Minute (минута)       |
| d        | Day (день)              | s        | Second (секунда)      |

Функция **DateAdd** может использоваться для добавления (или вычитания) определенного временного интервала к указанной дате (из указанной даты). Например, с использованием функции **DateAdd** можно узнать значение даты через 40 дней после текущей даты или время через 50 секунд от текущего времени.

---

**DateDiff**


---

Возвращает значение [тип **Variant (Long)**] числа временных интервалов между двумя определенными датами.

**Синтаксис**

**DateDiff**(*interval*, *date1*, *date2*[, *firstdayofweek*[, *firstweekofyear*]])

Функция **DateDiff** принимает следующие именованные аргументы:

|                        |                                                                                                                                                      |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>interval</i>        | Обязательный. Строковое выражение, которое является временным интервалом (единицей измерения).                                                       |
| <i>date1, date2</i>    | Обязательный; <b>Variant (Date)</b> . Две даты для вычисления.                                                                                       |
| <i>firstdayofweek</i>  | Необязательный. Константа, которая определяет первый день недели (день по умолчанию зависит от национальных настроек).                               |
| <i>firstweekofyear</i> | Необязательный. Константа, которая определяет первую неделю года. Если не указана, первой неделей считается неделя, в которой имеется дата 1 января. |

Значения аргумента *interval* приведены в таблице для функции **DateAdd**.

Значения аргумента *firstdayofweek* могут быть следующими:

| константа          | значение | описание                                                                            |
|--------------------|----------|-------------------------------------------------------------------------------------|
| <b>vbUseSystem</b> | 0        | Использовать национальные языковые [National Language Support (NLS)] API-настройки. |
| <b>vbSunday</b>    | 1        | Sunday (Воскресенье).                                                               |
| <b>vbMonday</b>    | 2        | Monday (Понедельник).                                                               |
| <b>vbTuesday</b>   | 3        | Tuesday (Вторник).                                                                  |
| <b>vbWednesday</b> | 4        | Wednesday (Среда).                                                                  |
| <b>vbThursday</b>  | 5        | Thursday (Четверг).                                                                 |
| <b>vbFriday</b>    | 6        | Friday (Пятница).                                                                   |
| <b>vbSaturday</b>  | 7        | Saturday (Суббота).                                                                 |

Значения аргумента *firstweekofyear* могут быть следующими:

| константа              | значение | описание                                                                           |
|------------------------|----------|------------------------------------------------------------------------------------|
| <b>vbUseSystem</b>     | 0        | Использовать NLS API-настройки.                                                    |
| <b>vbFirstJan1</b>     | 1        | Начинать с недели, в которой имеется дата 1 января (по умолчанию).                 |
| <b>vbFirstFourDays</b> | 2        | Начинать с первой недели, которая имеет, по крайней мере, четыре дня в новом году. |
| <b>vbFirstFullWeek</b> | 3        | Начинать с первой полной недели года.                                              |

Функция **DateDiff** используется для определения количества указанного вида (дней, недель и т.д.) временных интервалов между двумя датами. При этом следует учитывать значение последнего аргумента.

## DatePart

Возвращает определенную часть [тип **Variant (Integer)**] заданной даты.

### Синтаксис

**DatePart**(*interval, date[, firstdayofweek[, firstweekofyear]]*)

Функция **DatePart** принимает следующие именованные аргументы:

|                        |                                                                                                                                         |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <i>interval</i>        | Обязательный. Строковое выражение с обозначением необходимого интервала.                                                                |
| <i>date</i>            | Обязательный. Тип <b>Variant (Date)</b> . Значение даты.                                                                                |
| <i>firstdayofweek</i>  | Необязательный. Константа, задающая первый день недели. Значение по умолчанию определяется национальными настройками системы.           |
| <i>firstweekofyear</i> | Необязательный. Константа, задающая первую неделю года. Если не задается, первая неделя считается той, в которой имеется дата 1 января. |

Значения аргумента *interval* приведены в таблице для функции **DateAdd**. Значения аргумента *firstdayofweek* приведены в таблице для функции **DateDiff**.

Аргумент *firstweekofyear* может принимать следующие значения:

| константа              | значение | описание                                                                           |
|------------------------|----------|------------------------------------------------------------------------------------|
| <b>vbUseSystem</b>     | 0        | Использовать настройку NLS API.                                                    |
| <b>vbFirstJan1</b>     | 1        | Начинать с той недели, в которой будет первое января (по умолчанию).               |
| <b>vbFirstFourDays</b> | 2        | Начинать с первой недели, которая имеет, по меньшей мере, четыре дня в новом году. |
| <b>vbFirstFullWeek</b> | 3        | Начинать с первой полной недели года.                                              |

Функция **DatePart** может быть использована для вычисления некоторой даты и интервала времени. Например, вы можете вычислить день недели или текущий час.

#### Пример

```
Dim DateVar As Date
DateVar = InputBox("Введите дату:")
MsgBox "Квартал: " & DatePart("q", DateVar)
```

Если в диалоговом окне функции **InputBox** ввести, например, **22/09/2007**, то в диалоговом окне **MsgBox** будет выведена строка **"Квартал: 3"**

---

#### DateSerial

---

Возвращает значение даты [тип **Variant (Date)**] для заданных года, месяца и дня.

#### Синтаксис

**DateSerial**(year, month, day)

Функция **DateSerial** принимает следующие именованные аргументы:

|              |                                                                                                 |
|--------------|-------------------------------------------------------------------------------------------------|
| <i>year</i>  | Обязательный; тип <b>Integer</b> . Число (арифметическое выражение) в интервале от 100 до 9999. |
| <i>month</i> | Обязательный; тип <b>Integer</b> . Любое арифметическое выражение.                              |
| <i>day</i>   | Обязательный; тип <b>Integer</b> . Любое арифметическое выражение.                              |

### Пример

Следующая строка имеет результатом диалоговое окно с текстом «12.02.07».

```
MsgBox DateSerial(2007, 2, 12)
```

---

## DateValue

---

Возвращает значение типа **Date**, эквивалентное дате, заданной аргументом, который должен быть строкой, числом или константой, представляющей дату.

### Синтаксис

**DateValue** (*date*)

Обязательный аргумент *date* — обычно строковое выражение, представляющее дату от 1 января 100 года до 31 декабря 9999 года. Этот аргумент может быть также любым выражением, которое вычисляется до значения даты.

Если *date* является строкой, включающей только числа, разделенные так называемыми разделителями даты (*date separators*), **DateValue** распознает месяц, день и год в соответствии с форматом **Short Date**, который определен для вашей системы. **DateValue** также распознает строку, которая содержит дату с полным или коротким именем месяца (например, «февраль 12, 1969»).

---

## Day

---

Возвращает значение [тип **Variant (Integer)**] (между 1 и 31 включительно), представляющее день месяца.

### Синтаксис

**Day** (*date*)

Обязательный аргумент *date* — любое значение типа **Variant**, численное выражение, строковое выражение или их комбинация, которая представляет дату. Если аргумент содержит значение **Null**, функция возвращает **Null**.

Возвращаемое значение зависит от значения свойства **Calendar** в вашей системе.

---

## DDB

---

Возвращает значение (**Double**) амортизации (*depreciation*) активов за определенный период времени с использованием балансового метода «двойного списания» (*double-declining*) или другого (вами указанного) метода.

### Синтаксис

**DDB** (*cost*, *salvage*, *life*, *period*[, *factor*])

Именованные аргументы функции **DDB**:

|                |                                                                                                                    |
|----------------|--------------------------------------------------------------------------------------------------------------------|
| <i>cost</i>    | Обязательный. Тип <b>Double</b> . Определяет начальную стоимость активов.                                          |
| <i>salvage</i> | Обязательный. Тип <b>Double</b> . Определяет стоимость активов в конце периода амортизации (остаточная стоимость). |
| <i>life</i>    | Обязательный. Тип <b>Double</b> . Определяет длительность периода амортизации собственности.                       |
| <i>period</i>  | Обязательный. Тип <b>Double</b> . Определяет период, для которого вычисляется амортизация активов.                 |

*factor* Необязательный. Тип **Variant**. Определяет норму снижения стоимости (амортизации). Если аргумент отсутствует, используется значение 2 (метод «двойного списания»).

Аргументы *life* и *period* должны выражаться в одних и тех же единицах, например, если *life* задается в месяцах, то и *period* также должен быть задан в месяцах. Все аргументы должны быть положительными числами.

Функция **DDb** использует следующую формулу для вычисления амортизации за заданный период:

$$\text{Амортизация} / \text{period} = ((\text{cost} - \text{salvage}) * \text{factor}) / \text{life}$$

**Dir**

Возвращает значение типа **String**, представляющее имя файла (каталога или папки), которое совпадает с шаблоном, файловым атрибутом или меткой тома драйвера.

**Синтаксис**

**Dir**[(*pathname*[, *attributes*])]

Функция **Dir** имеет следующие аргументы:

- pathname* Необязательный. Строковое выражение, которое определяет имя файла. Может включать каталог (папку) и имя драйвера диска. Если функция не находит *pathname*, возвращается строка нулевой длины ("").
- attributes* Необязательный. Константа или числовое выражение, являющееся суммой атрибутов файла.

Аргумент *attributes* принимает следующие значения:

| константа          | значение | описание                                                                                     |
|--------------------|----------|----------------------------------------------------------------------------------------------|
| <b>vbNormal</b>    | 0        | Обычный (Normal) (по умолчанию) — файлы без атрибутов.                                       |
| <b>vbReadOnly</b>  | 1        | Только на чтение (Read-only).                                                                |
| <b>vbHidden</b>    | 2        | Скрытый (Hidden).                                                                            |
| <b>VbSystem</b>    | 4        | Системный файл (System file).                                                                |
| <b>vbVolume</b>    | 8        | Метка тома (Volume label); если указаны любые другие атрибуты, <b>vbVolume</b> игнорируется. |
| <b>vbDirectory</b> | 16       | Каталог или папка.                                                                           |
| <b>vbAlias</b>     | 64       | Алиасное имя (на Macintosh).                                                                 |

В Microsoft Windows **Dir** поддерживает использование символов (\*) и (?) для поиска множества файлов.

В первом вызове функции **Dir** следует обязательно задавать аргумент *pathname*, иначе будет сгенерирована ошибка. Аргумент *pathname* также должен использоваться, если указываются атрибуты файлов.

**Dir** возвращает первое имя файла, которое совпадает с *pathname*. Для получения следующих файлов, совпадающих с *pathname*, следует вызвать функцию **Dir** снова, но без аргументов. После того как функция в последних вызовах возвратит

все совпавшие файлы, функция возвратит пустую строку (""). В связи с этим удобно записывать имена файлов, возвращаемых функцией **Dir**, в массив и в дальнейшем, например, сортировать его.

---

## Environ

---

Возвращает значение типа **String**, связанное с переменной среды операционной системы. На Macintosh не работает.

### Синтаксис

**Environ** ({*envstring* | *number*})

Функция **Environ** принимает следующие именованные аргументы:

|                  |                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------|
| <i>envstring</i> | Необязательный. Строковое выражение, содержащее имя переменной среды.                         |
| <i>number</i>    | Необязательный. Числовое выражение, соответствующее номеру строки в таблице переменных среды. |

Если значение аргумента *envstring* не может быть найдено в таблице переменных среды, функция возвращает строку нулевой длины (""). Иначе функция возвращает текст, присвоенный определенному параметру *envstring*, текст, который следует за знаком равенства (=) в таблице переменных среды для определенной переменной.

---

## EOF

---

Возвращает значение **True** при достижении конца файла, открытого для прямого или последовательного доступа.

### Синтаксис

**EOF** (*filenumber*)

Обязательный аргумент *filenumber* является **Integer**-значением, содержащим допустимый номер файла. Эту функцию следует использовать перед попыткой доступа к концу файла.

Пока конец файла (открытого для **Random**- или **Binary**-доступа) не достигнут, функция возвращает значение **False**. При этом из файла можно получать данные.

Если файл был открыт для **Binary**-доступа, попытка чтения записей из него функцией **Input** при условии, что **EOF** возвращает значение **True**, приведет к генерации ошибки. Используйте функции **LOF** и **LOC** вместо **EOF** при чтении бинарных файлов функцией **Input** или используйте **GET** с функцией **EOF**. С файлами, открытыми для **Output**, функция **EOF** всегда возвращает **True**.

---

## Error

---

Возвращает error-сообщение, соответствующее заданному номеру ошибки.

### Синтаксис

**Error** [ (*errornumber*) ]

Необязательный аргумент *errornumber* может быть любым допустимым error-номером. Если *errornumber* — допустимый error-номер, но не определен, функция возвращает сообщение «Application-defined or object-defined error.» Если аргумент отсутствует, возвращается сообщение, относящееся к последней run-time-ошибке. Если же *errornumber* равен 0, функция возвращает строку нулевой длины ("").

**Exp**

Возвращает значение (тип **Double**), определяющее число *e* (основание натурального логарифма), возведенное в степень.

**Синтаксис**

**Exp**(*number*)

Обязательный аргумент *number* — **Double**- или любое допустимое численное выражение. Если значение аргумента превышает число 709.782712893, возникнет ошибка. Константа *e* равна приблизительно 2.718282.

**FileAttr**

Возвращает значение (тип **Long**), представляющее файловый режим для файла, открытого с использованием оператора **Open**.

**Синтаксис**

**FileAttr**(*filenumber*, *returntype*)

Функция **FileAttr** имеет следующие именованные аргументы:

*filenumber*      Обязательный; тип **Integer**. Любой допустимый файловый номер.

*returntype*      Обязательный; тип **Integer**. Число, указывающее на тип возвращаемой информации. Для определения файлового режима следует задать значение 1. Для получения дескриптора файла операционной системы укажите значение 2 (только для 16-битной системы!).

При значении аргумента *returntype* равном 1, можно получить следующие результаты:

| режим  | значение | режим  | значение |
|--------|----------|--------|----------|
| Input  | 1        | Append | 8        |
| Output | 2        | Binary | 32       |
| Random | 4        |        |          |

**FileDateTime**

Возвращает значение [тип **Variant (Date)**], которое указывает дату и время создания или последней модификации файла.

**Синтаксис**

**FileDateTime**(*pathname*)

Обязательный аргумент *pathname* является строковым выражением, определяющим имя файла, которое может включать каталог (папку) и драйвер.

**Пример**

Следующая строка приводит к выдаче в окне **MsgBox** информации о времени создания файла "TESTFILE".

```
MsgBox FileDateTime("TESTFILE")
```

## FileLen

Возвращает значение (тип **Long**) длины (размера) файла в байтах.

### Синтаксис

**FileLen** (*pathname*)

Обязательный аргумент *pathname* является строковым выражением, определяющим имя файла, которое может включать каталог (папку) и драйвер. Если указанный в аргументе файл открыт при вызове функции **FileLen**, возвращаемое значение имеет отношение к длине файла непосредственно перед открытием.

Чтобы получить длину открытого файла, используйте функцию **LOF**.

### Пример

Следующая строка приводит к выдаче в окне **MsgBox** информации о длине файла **TESTFILE**:

```
MsgBox FileLen ("TESTFILE").
```

## Filter

Возвращает массив (начинающийся с нулевого индекса), содержащий поднабор строчного массива на основе заданного критерия.

### Синтаксис

**Filter** (*sourcearray*, *match*[, *include*[, *compare*]])

Функция **Filter** имеет следующие именованные аргументы:

*sourcearray*      Обязательный. Одномерный массив, в котором производится поиск.

*match*              Обязательный. Строка для поиска.

*include*            Необязательный. Значение типа **Boolean**, определяющее, какое значение возвращать: включающее или не включающее шаблон *match*. Если *include* — **True**, функция **Filter** возвращает поднабор массива, который содержит *match* как подстроку. Если *include* — **False**, **Filter** возвращает поднабор массива, который не содержит *match* как подстроку.

*compare*            Необязательный. Численное значение, указывающее тип сравнения.

Аргумент *compare* может иметь следующие значения:

| константа                 | значение | назначение                                                                                |
|---------------------------|----------|-------------------------------------------------------------------------------------------|
| <b>vbUseCompareOption</b> | -1       | Выполнять сравнение с использованием настроек оператора <b>Option Compare</b> .           |
| <b>vbBinaryCompare</b>    | 0        | Выполнять бинарное сравнение.                                                             |
| <b>vbTextCompare</b>      | 1        | Выполнять текстовое сравнение.                                                            |
| <b>vbDatabaseCompare</b>  | 2        | Только для Microsoft Access. Выполнять сравнение, основанное на информации в базе данных. |



Если ни одного совпадения с шаблоном *match* в *sourcearray* не обнаруживается, функция **Filter** возвращает пустой массив. Если *sourcearray* — **Null** или — не одномерный массив, возникает ошибка.

---

**Fix**

---

Возвращает целую часть числа.

**Синтаксис**

**Fix** (*number*)

Обязательный аргумент *number* имеет тип **Double** или является допустимым численным выражением. Если *number* — **Null**, возвращается **Null**.

И функция **Int**, и функция **Fix** удаляют дробную часть аргумента *number* и возвращают целое значение. Отличаются эти функции при работе с отрицательным аргументом: **Int** возвращает ближайшее меньшее отрицательное число, а **Fix** — ближайшее большее отрицательное число.

**Примеры**

```
Dim My_Number as Integer
My_Number = Int(19.8)      ' результат 19
My_Number = Fix(19.2)      ' результат 19

My_Number = Int(-19.8)     ' результат -20
My_Number = Fix(-19.8)     ' результат -19

My_Number = Int(-19.2)     ' результат -20
My_Number = Fix(-19.2)     ' результат -19
```

---

**Format**

---

Возвращает строку [тип **Variant (String)**], содержащую значение, отформатированное согласно инструкциям, находящимся в *format*-выражении.

**Синтаксис**

**Format** (*expression* [, *format* [, *firstdayofweek* [, *firstweekofyear* ]]])

Функция **Format** принимает следующие аргументы:

- expression*            Обязательный. Любое допустимое выражение.
- format*                Необязательный. Допустимое выражение именованного или определенного пользователем формата.
- firstdayofweek*        Необязательный. Константа, которая определяет первый день недели.
- firstweekofyear*        Необязательный. Константа, которая определяет первую неделю года.

Аргумент *firstdayofweek* может принимать следующие значения:

| константа          | значение | описание                       |
|--------------------|----------|--------------------------------|
| <b>vbUseSystem</b> | 0        | Использовать настройку NLS API |
| <b>VbSunday</b>    | 1        | Sunday                         |

|                    |   |           |
|--------------------|---|-----------|
| <b>vbMonday</b>    | 2 | Monday    |
| <b>vbTuesday</b>   | 3 | Tuesday   |
| <b>vbWednesday</b> | 4 | Wednesday |
| <b>vbThursday</b>  | 5 | Thursday  |
| <b>vbFriday</b>    | 6 | Friday    |
| <b>vbSaturday</b>  | 7 | Saturday  |

Аргумент *firstweekofyear* может принимать следующие значения:

| константа              | значение | описание                                                                       |
|------------------------|----------|--------------------------------------------------------------------------------|
| <b>vbUseSystem</b>     | 0        | Использовать настройку NLS API.                                                |
| <b>vbFirstJan1</b>     | 1        | Начинать с недели, в которой имеется дата первое января.                       |
| <b>vbFirstFourDays</b> | 2        | Начинать с первой недели, в которой имеется, по крайней мере, четыре дня года. |
| <b>vbFirstFullWeek</b> | 3        | Начинать с первой полной недели года.                                          |

Более подробно функция **Format** описана в главе 4.

**FormatCurrency**

Возвращает выражение, отформатированное как денежное (валютное) выражение с использованием значения, заданного на вкладке **Денежная единица** окна **Свойства: Язык и стандарты**, доступного из **Панели управления**.

**Синтаксис**

**FormatCurrency**(*Expression* [, *NumDigitsAfterDecimal* [, *IncludeLeadingDigit* [, *UseParensForNegativeNumbers* [, *GroupDigits*]]]])

Функция **FormatCurrency** принимает следующие аргументы:

|                                    |                                                                                                                                                                                                                   |
|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Expression</i>                  | Обязательный. Выражение для форматирования.                                                                                                                                                                       |
| <i>NumDigitsAfterDecimal</i>       | Необязательный. Численное значение, определяющее количество отображаемых знаков справа от десятичной точки. Значение по умолчанию для этого аргумента равно -1, что означает использование региональных настроек. |
| <i>IncludeLeadingDigit</i>         | Необязательный. Константа трех состояний, определяющая, следует ли отображать ведущий ноль для дробных чисел.                                                                                                     |
| <i>UseParensForNegativeNumbers</i> | Необязательный. Константа трех состояний, определяющая, следует ли помещать отрицательные значения внутри круглых скобок.                                                                                         |

*GroupDigits*

Необязательный. Константа трех состояний, определяющая, группировать ли цифры с помощью ограничителей, заданных региональными настройками.

Аргументы *IncludeLeadingDigit*, *UseParensForNegativeNumbers* и *GroupDigits* могут принимать следующие значения:

| константа           | значение | описание                                        |
|---------------------|----------|-------------------------------------------------|
| <b>vbTrue</b>       | -1       | <b>True</b>                                     |
| <b>vbFalse</b>      | 0        | <b>False</b>                                    |
| <b>vbUseDefault</b> | -2       | Использовать региональные настройки компьютера. |

Когда один или более необязательных аргументов пропускается, значения для пропущенных аргументов используются из региональных настроек компьютера.

**FormatDateTime**

Возвращает выражение, отформатированное, как дата или время.

**Синтаксис**

**FormatDateTime** (*Date* [, *NamedFormat*])

Функция **FormatDateTime** принимает следующие аргументы:

*Date*                      Обязательный. Дата для форматирования.

*NamedFormat*            Необязательный. Численное значение, указывающее, необходимый формат. Если опущен, используется **vbGeneralDate**.

Аргумент *NamedFormat* может принимать следующие значения:

| константа            | значение | описание                                                                                                                                                                                                    |
|----------------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>vbGeneralDate</b> | 0        | Отображать дату и/или время. Если в аргументе <b>Date</b> имеется часть даты, дата отображается в коротком формате. Если в аргументе <b>Date</b> имеется часть времени, она отображается в длинном формате. |
| <b>vbLongDate</b>    | 1        | Отображать дату с использованием длинного формата, установленного на компьютере в региональных настройках.                                                                                                  |
| <b>vbShortDate</b>   | 2        | Отображать дату с использованием короткого формата, установленного на компьютере в региональных настройках.                                                                                                 |
| <b>vbLongTime</b>    | 3        | Отображать время с использованием временного формата, установленного на компьютере в региональных настройках.                                                                                               |
| <b>vbShortTime</b>   | 4        | Отображать время с использованием формата 24-hour (hh:mm).                                                                                                                                                  |

---

**FormatNumber**


---

Возвращает выражение, отформатированное, как число.

**Синтаксис**

**FormatNumber** (*Expression* [, *NumDigitsAfterDecimal* [, *IncludeLeadingDigit* [, *UseParensForNegativeNumbers* [, *GroupDigits*]]]])

Функция **FormatNumber** принимает следующие аргументы:

|                                    |                                                                                                                                                                                                                   |
|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Expression</i>                  | Обязательный. Выражение для форматирования.                                                                                                                                                                       |
| <i>NumDigitsAfterDecimal</i>       | Необязательный. Численное значение, определяющее количество отображаемых знаков справа от десятичной точки. Значение по умолчанию для этого аргумента равно -1, что означает использование региональных настроек. |
| <i>IncludeLeadingDigit</i>         | Необязательный. Константа трех состояний, определяющая, следует ли отображать ведущий ноль для дробных чисел.                                                                                                     |
| <i>UseParensForNegativeNumbers</i> | Необязательный. Константа трех состояний, определяющая, следует ли помещать отрицательные значения внутри круглых скобок.                                                                                         |
| <i>GroupDigits</i>                 | Необязательный. Константа трех состояний, определяющая; группировать ли цифры с помощью ограничителей, заданных региональными настройками.                                                                        |

Аргументы *IncludeLeadingDigit*, *UseParensForNegativeNumbers* и *GroupDigits* могут принимать следующие значения:

| константа           | значение | описание                                        |
|---------------------|----------|-------------------------------------------------|
| <b>vbTrue</b>       | -1       | <b>True</b>                                     |
| <b>vbFalse</b>      | 0        | <b>False</b>                                    |
| <b>vbUseDefault</b> | -2       | Использовать региональные настройки компьютера. |

Когда один или более необязательных аргументов пропускается, значения для пропущенных аргументов используются из региональных настроек компьютера.

---

**FormatPercent**


---

Возвращает выражение, отформатированное, как процентное отношение (умноженное на 100) с конечным знаком процента (%).

**Синтаксис**

**FormatPercent** (*Expression* [, *NumDigitsAfterDecimal* [, *IncludeLeadingDigit* [, *UseParensForNegativeNumbers* [, *GroupDigits*]]]])

Функция **FormatPercent** имеет следующие аргументы:

|                                    |                                                                                                                                                                                                                   |
|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Expression</i>                  | Обязательный. Выражение для форматирования.                                                                                                                                                                       |
| <i>NumDigitsAfterDecimal</i>       | Необязательный. Численное значение, определяющее количество отображаемых знаков справа от десятичной точки. Значение по умолчанию для этого аргумента равно -1, что означает использование региональных настроек. |
| <i>IncludeLeadingDigit</i>         | Необязательный. Константа трех состояний, определяющая, следует ли отображать ведущий ноль для дробных чисел.                                                                                                     |
| <i>UseParensForNegativeNumbers</i> | Необязательный. Константа трех состояний, определяющая, следует ли помещать отрицательные значения внутри круглых скобок.                                                                                         |
| <i>GroupDigits</i>                 | Необязательный. Константа трех состояний, определяющая, группировать ли цифры с помощью ограничителей, заданных региональными настройками.                                                                        |

Аргументы *IncludeLeadingDigit*, *UseParensForNegativeNumbers*, и *GroupDigits* могут принимать следующие значения:

| константа           | значение | описание                                        |
|---------------------|----------|-------------------------------------------------|
| <b>vbTrue</b>       | -1       | <b>True</b>                                     |
| <b>vbFalse</b>      | 0        | <b>False</b>                                    |
| <b>vbUseDefault</b> | -2       | Использовать региональные настройки компьютера. |

Когда один или более необязательных аргументов пропускается, значения для пропущенных аргументов используются из региональных настроек компьютера.

**FreeFile**

Возвращает значение (тип **Integer**), представляющее следующий файловый номер, доступный для использования оператором **Open**.

**Синтаксис**

**FreeFile**[(*rangenumber*)]

Необязательный аргумент *rangenumber* (тип **Variant**) определяет диапазон для возвращаемых файловых номеров. Если этот аргумент задается нулевым (по умолчанию), то возвращается файловый номер в диапазоне 1 – 255, включительно. Для диапазона 256 – 511 используется значение 1.

**FV**

Возвращает будущее значение (тип **Double**) ежегодных поступлений (annuity) на базе периодических, фиксированных платежей (fixed payments) и фиксированной процентной ставки (fixed interest rate).

**Синтаксис****FV**(*rate*, *nper*, *pmt*[, *pv*[, *type*]])Функция **FV** имеет следующие именованные аргументы:

|             |                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>rate</i> | Обязательный. Тип <b>Double</b> . Процентная ставка за период. Например, если вы получаете ссуду на покупку машины из расчета 10 процентов годовых [annual percentage rate (APR)] и вносите ежемесячные платежи, то ставка за период должна составлять 0.1/12 или 0.0083.                                                                                            |
| <i>nper</i> | Обязательный. Тип <b>Integer</b> . Общее число периодов платежей в ежегодных поступлениях (annuity). Например, если вы вносите ежемесячные платежи в счет погашения четырехлетней ссуды на покупку автомашины, в вашей ссуде будет всего 4 × 12 (или 48) периодов платежей.                                                                                          |
| <i>pmt</i>  | Обязательный. Тип <b>Double</b> . Платеж, который должен быть внесен за каждый период. Платежи обычно состоят из основного платежа (principal) и и платежа по процентам (interest), которые не изменяются на протяжении выплаты взноса.                                                                                                                              |
| <i>pv</i>   | Необязательный. Тип <b>Variant</b> . Настоящая величина [или твердая (lump sum)] ряда будущих платежей. Например, когда вы берете в долг деньги, чтобы купить автомобиль, сумма займа (loan amount) — это настоящая величина для кредитора ежемесячных платежей за автомобиль, которые вы будете вносить. Если аргумент отсутствует, то он подразумевается равным 0. |
| <i>type</i> | Необязательный. Тип <b>Variant</b> . Определяет, когда должен производиться платеж. Используется 0, если платеж должен производиться в конце периода платежа или 1, если срок платежа наступает в начале периода платежа. Если отсутствует, подразумевается 0.                                                                                                       |

Ежегодные поступления, аннуитет (annuity) — это ряд фиксированных наличных платежей, вносимых в течение некоторого периода времени. Поступления могут быть ссудой, такой как ипотека (home mortgage) или инвестицией, такой как план ежемесячных накоплений (monthly savings plan).

Аргументы *rate* и *nper* должны вычисляться с использованием периодов платежей, выраженных в одних и тех же единицах. Например, если *rate* вычисляется с использованием месяцев, *nper* должен также вычисляться в месяцах.

Для всех аргументов наличные платежи, такие как депозиты (deposits to savings) представляются отрицательными числами; наличные поступления, такие как дивиденды, представляются положительными числами.

---

**GetAllSettings**

---

Возвращает список ключевых настроек и их значения в Windows реестре (или файле инициализации приложения на Macintosh).

**Синтаксис****GetAllSettings**(*appname*, *section*)Функция **GetAllSettings** принимает следующие именованные аргументы:

|                |                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------|
| <i>appname</i> | Обязательный. Строковое выражение, содержащее имя приложения или проекта, ключевые установки которого требуются. |
|----------------|------------------------------------------------------------------------------------------------------------------|

*section* Обязательный. Строковое выражение, содержащее имя секции, ключевые настройки которой необходимо получить. Функция возвращает двумерный массив строк, содержащий все ключевые настройки в указанной секции и соответствующие им значения.

Функция **GetAllSettings** возвращает неинициализированное **Variant**-значение, если не задан либо аргумент *appname*, либо *section*.

---

### GetAttr

---

Возвращает значение (тип **Integer**), представляющее атрибуты файла, каталога или папки.

#### Синтаксис

**GetAttr** (*pathname*)

Обязательный аргумент *pathname* является строковым выражением, определяющим имя файла, которое может включать каталог (или папку) и драйвер.

Значения, возвращаемые функцией **GetAttr**, представляют собой сумму следующих значений атрибутов:

| константа          | значение | описание                                                      |
|--------------------|----------|---------------------------------------------------------------|
| <b>vbNormal</b>    | 0        | Обычный (Normal).                                             |
| <b>vbReadOnly</b>  | 1        | Только на чтение (Read-only).                                 |
| <b>vbHidden</b>    | 2        | Скрытый (Hidden).                                             |
| <b>vbSystem</b>    | 4        | Системный файл (System file).                                 |
| <b>vbDirectory</b> | 16       | Каталог или папка (Directory or folder).                      |
| <b>vbArchive</b>   | 32       | Файл изменился с момента архивации (недоступен на Macintosh). |
| <b>vbAlias</b>     | 64       | Определяет имя файла как алиасное. Только для Macintosh.      |

Чтобы определить установленные атрибуты, необходимо использовать оператор **And** для побитового сравнения возвращаемого функцией значения и некоторого шаблона с указанием конкретного атрибута. Если результатом сравнения будет нулевое значение, это означает, что атрибут не установлен.

---

### GetObject

---

Возвращает ссылку на объект, поддерживаемый ActiveX-компонентом.

#### Синтаксис

**GetObject** ([*pathname*] [, *class*])

Функция **GetObject** принимает следующие именованные аргументы:

*pathname* Необязательный; тип **Variant (String)**. Полный путь и имя файла, содержащего объект. Если *pathname* опускается, необходимо указывать аргумент *class*.

*class* Необязательный; тип **Variant (String)**. Строка — класс объекта.

Аргумент *class* использует синтаксис *appname.objecttype* и имеет две составляющие:

*appname*            Обязательный; тип **Variant (String)**. Имя приложения, обеспечивающего объектом.

*objecttype*        Обязательный; тип **Variant (String)**. Тип или класс объекта.

Используйте функцию **GetObject** для доступа к ActiveX-объекту из файла и назначайте объект объектной переменной. Оператор **Set** применяйте для присваивания объектной переменной возвращаемого функцией **GetObject** значения. Например:

```
Dim COBject As Object
Set COBject = GetObject("C:\CAD\SCHEMA.CAD")
```

Если *pathname* является строкой нулевой длины (""), функция **GetObject** возвращает экземпляр нового объекта указанного типа. Если аргумент *pathname* опускается, **GetObject** возвращает текущий активный объект указанного типа. Если ни одного указанного объекта не существует, возникает ошибка.

### Пример

В следующей процедуре функция **GetObject** используется для того, чтобы определить наличие в системе экземпляра приложения Excel.

```
Sub Test GetObject()
    Dim xlApp As Object
    Dim xlSheet As Object

    On Error Resume Next
    Set xlApp = Nothing
    Set xlApp = GetObject(, "Excel.Application")
    If Err.Number < 0 Then
        Set xlApp = CreateObject("Excel.Application")
    End If
    Err.Clear

    Set xlBook = xlApp.Workbooks.Add
    Set xlSheet = xlBook.WorkSheets(1)
    xlSheet.Application.Visible = True
End Sub
```

---

## GetSetting

Возвращает значение ключевой настройки определенного приложения в Windows-реестре.

### Синтаксис

**GetSetting**(*appname*, *section*, *key*[, *default*])

Функция **GetSetting** принимает следующие именованные аргументы:

*appname*            Обязательный. Строковое выражение, содержащее имя приложения или проекта, настройки которого необходимы.

*section*            Обязательный. Строковое выражение, содержащее наименование секции, в которой следует искать ключевую настройку.

*key*                Обязательный. Строковое выражение, содержащее имя настройки.



**default**            Необязательный. Выражение, содержащее возвращаемое значение, если в настройке не установлено никакого значения. Если этот параметр опускается, используется строка нулевой длины ("").

Если какой-либо из элементов, указанных в аргумента функции **GetSetting**, не существует, функция возвращает *default*.

**Hex**

Возвращает строку, представляющую шестнадцатиричное значение числа.

**Синтаксис**

**Hex** (*number*)

Обязательный аргумент *number* является любым допустимым числовым (или строковым) выражением.

Если *number* не является целым числом, сначала выполняется округление до целого.

| если <i>number</i> | функция возвращает |
|--------------------|--------------------|
| <b>Null</b>        | <b>Null</b>        |
| <b>Empty</b>       | Ноль (0)           |

**Примеры**

```
My_Hex = Hex(5)            ' Returns 5
My_Hex = Hex(10)          ' Returns A
My_Hex = Hex(459)         ' Returns 1CB
My_Hex = Hex(459.4)       ' Returns 1CB
My_Hex = Hex(459.7)       ' Returns 1CC
```

**Hour**

Возвращает значение [тип **Variant (Integer)**], определяющее целое число в диапазоне от 0 до 23, включительно, представляющее час дня.

**Синтаксис**

**Hour** (*time*)

Обязательный аргумент *time* — любое значение типа **Variant**, численное выражение, строковое выражение или любая их комбинация, представляющая время. Для аргумента *time*, содержащего **Null**, функция возвращает **Null**.

**IIf**

Возвращает значение одного из двух своих аргументов в зависимости от оцениваемого выражения.

**Синтаксис**

**IIf**(*expr*, *truepart*, *falsepart*)

Функция **IIf** имеет следующие (все обязательные) именованные аргументы:

|                  |                                                                                              |
|------------------|----------------------------------------------------------------------------------------------|
| <i>expr</i>      | Оцениваемое выражение.                                                                       |
| <i>truepart</i>  | Значение или выражение, которое возвращается, если <i>expr</i> имеет значение <b>True</b> .  |
| <i>falsepart</i> | Значение или выражение, которое возвращается, если <i>expr</i> имеет значение <b>False</b> . |

### Пример

```
x = 200
MsgBox IIf(x > 1000, "Много", "Мало")           ' выдача слова "Мало"
```

---

## Input

Функция **String** возвращает строку символов из файла, открытого в режиме **Input** или **Binary**.

### Синтаксис

**Input**(*number*, [#]*filenumber*)

Функция **Input** имеет следующие (обязательные) аргументы:

|                   |                                                                                            |
|-------------------|--------------------------------------------------------------------------------------------|
| <i>number</i>     | Любое допустимое выражение (возвращающее число), определяющее число возвращаемых символов. |
| <i>filenumber</i> | Допустимый номер файла.                                                                    |

Данные, считываемые функцией **Input**, предварительно должны быть записаны в файл посредством функции **Print #** или **Put**.

В отличие от оператора **Input #** функция **Input** возвращает все прочитанные символы, включая запятые, символы возврата каретки, перехода на другую строку, кавычки и ведущие пробелы.

### Пример

В этом примере открывается текстовый файл и в цикле происходит посимвольное считывание и вывод в окно **Immediate**.

```
Open "F:\ZZ\WWW.TXT" For Input As #1 ' открыть файл
Do While Not EOF(1)                  ' цикл до конца файла
    CharOne = Input(1, #1)           ' считать символ
    Debug.Print CharOne              ' отобразить в Immediate-окне
Loop
Close #1                             ' закрыть файл
```

---

## InputBox

Отображает запрос в диалоговом окне, ожидает ввода пользователем строки (или щелчка на кнопке окна), а затем возвращает строку из поля ввода окна.

### Синтаксис

**InputBox**(*prompt* [, *title*] [, *default*] [, *xpos*] [, *ypos*] [, *helpfile*, *context*])

Функция **InputBox** принимает следующие именованные аргументы:

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>prompt</i>   | Обязательный. Любое строковое выражение, отображаемое в качестве подсказки. Максимальная длина этой строки примерно 1024 символа (в зависимости от используемой ширины символов). Если необходимо, чтобы <i>prompt</i> -аргумент содержал более одной строки, следует использовать символы разделения строк: символ возврата каретки ( <b>Chr(13)</b> ) и символ перехода на другую строку ( <b>Chr(10)</b> ) или комбинацию этих символов ( <b>Chr(13) &amp; Chr(10)</b> ). |
| <i>title</i>    | Необязательный. Строка, используемая в качестве заголовка для окна ввода. Если этот аргумент опускается, в качестве заголовка выводится наименование приложения, в котором вызывается функция.                                                                                                                                                                                                                                                                               |
| <i>default</i>  | Необязательный. Является любым строковым выражением и используется как значение по умолчанию для пользовательского ввода. Если опускается, строка ввода в окне отображается пустой.                                                                                                                                                                                                                                                                                          |
| <i>xpos</i>     | Необязательный. Численное значение, определяющее (в твипах) горизонтальное расстояние от левого края окна до верхнего левого угла диалогового окна. Если этот аргумент не указан, диалоговое окно центрируется горизонтально.                                                                                                                                                                                                                                                |
| <i>ypos</i>     | Необязательный. Численное значение, определяющее (в твипах) вертикальное расстояние от верхнего края окна до верхней границы диалогового окна. Если этот аргумент не указывается, диалоговое окно позиционируется по вертикали примерно на одну треть от нижней части экрана.                                                                                                                                                                                                |
| <i>helpfile</i> | Необязательный. Строковое выражение, которое содержит имя справочного Windows-файла для обеспечения контекстной справки. Если этот аргумент используется, необходимо включать и аргумент <i>context</i> .                                                                                                                                                                                                                                                                    |
| <i>context</i>  | Необязательный. Численное выражение, которое указывает раздел в справочном файле, относящийся к отображаемому окну.                                                                                                                                                                                                                                                                                                                                                          |

## InStr

Возвращает значение [тип **Variant (Long)**], определяющее позицию первого вхождения одной строки внутри другой.

### Синтаксис

**InStr**(*[start]*, *string1*, *string2*[, *compare*])

Функция **InStr** принимает следующие аргументы:

|                |                                                                                                                                                                                                                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>start</i>   | Необязательный. Числовое выражение, задающее начальную позицию для поиска. Если аргумент не указывается, то поиск начинается с первого символа. Если аргумент <i>start</i> содержит значение <b>Null</b> , возникает ошибка. Аргумент <i>start</i> является обязательным, если указывается аргумент <i>compare</i> . |
| <i>string1</i> | Обязательный. Строковое выражение, в котором выполняется поиск.                                                                                                                                                                                                                                                      |
| <i>string2</i> | Обязательный. Строковое выражение, поиск положения которого выполняет функция.                                                                                                                                                                                                                                       |

*compare*                    Необязательный. Определяет тип сравнения строк. Если не опускается, то используется сравнение согласно оператору **Option Compare**.

Аргумент *compare* может принимать следующие значения:

| константа                 | значение | описание                                                                                          |
|---------------------------|----------|---------------------------------------------------------------------------------------------------|
| <b>vbUseCompareOption</b> | -1       | Выполняется сравнение согласно установкам посредством оператора <b>Option Compare</b> .           |
| <b>vbBinaryCompare</b>    | 0        | Выполняется двоичное сравнение.                                                                   |
| <b>vbTextCompare</b>      | 1        | Выполняется текстовое сравнение.                                                                  |
| <b>vbDatabaseCompare</b>  | 2        | Только для Microsoft Access. Выполняется сравнение, основанное на информации в вашей базе данных. |

Функция **InStr** возвращает следующие значения:

| если                               | InStr возвращает | если                                    | InStr возвращает                         |
|------------------------------------|------------------|-----------------------------------------|------------------------------------------|
| <i>string1</i> имеет нулевую длину | 0                | <i>string2</i> не найдена               | 0                                        |
| <i>string1</i> — <b>Null</b>       | <b>Null</b>      | <i>string2</i> найдена в <i>string1</i> | Позиция, в которой произошло совпадение. |
| <i>string2</i> имеет нулевую длину | <i>start</i>     | <i>start</i> > <i>string2</i>           | 0                                        |
| <i>string2</i> — <b>Null</b>       | <b>Null</b>      |                                         |                                          |

**InStrRev**

Возвращает позицию появления одной строки внутри другой, начиная с конца строки.

**Синтаксис**

**InstrRev**(*stringcheck*, *stringmatch*[, *start*[, *compare*]])

Функция **InstrRev** принимает следующие именованные аргументы:

- stringcheck*            Обязательный. Строка, в которой выполняется поиск.
- stringmatch*            Обязательный. Строковое выражение, поиск которого выполняется.
- start*                    Необязательный. Численное выражение, устанавливающее начальную позицию для поиска. Если не указывается, используется значение -1, что означает начало поиска с последнего символа строки. Значение **Null** вызывает появление ошибки.
- compare*                Необязательный. Численное значение, задающее тип сравнения. Если не указывается, выполняется двоичное сравнение.

Аргумент *compare* может принимать следующие значения:

| константа                 | значение | описание                                                                                          |
|---------------------------|----------|---------------------------------------------------------------------------------------------------|
| <b>vbUseCompareOption</b> | -1       | Выполняется сравнение согласно установкам посредством оператора <b>Option Compare</b> .           |
| <b>vbBinaryCompare</b>    | 0        | Выполняется двоичное сравнение.                                                                   |
| <b>vbTextCompare</b>      | 1        | Выполняется текстовое сравнение.                                                                  |
| <b>vbDatabaseCompare</b>  | 2        | Только для Microsoft Access. Выполняется сравнение, основанное на информации в вашей базе данных. |

Функция **InStrRev** возвращает следующие значения:

| если                                            | InStrRev возвращает                     |
|-------------------------------------------------|-----------------------------------------|
| <i>stringcheck</i> имеет нулевую длину          | 0                                       |
| <i>stringcheck</i> — <b>Null</b>                | <b>Null</b>                             |
| <i>stringmatch</i> имеет нулевую длину          | <i>start</i>                            |
| <i>stringmatch</i> — <b>Null</b>                | <b>Null</b>                             |
| <i>stringmatch</i> не найдена                   | 0                                       |
| <i>stringmatch</i> найдена в <i>stringcheck</i> | Позиция, в которой произошло совпадение |
| <i>start</i> > <b>Len(stringmatch)</b>          | 0                                       |

## Int

Возвращает целую часть числа.

### Синтаксис

**Fix** (*number*)

Обязательный аргумент *number* — число (тип **Double**) или допустимое числовое выражение. Если *number* содержит **Null**, функция возвращает **Null**.

## IPmt

Возвращает значение (тип **Double**) платежей по процентам для данного периода ежегодных поступлений на базе периодических, фиксированных платежей и фиксированной процентной ставки.

### Синтаксис

**IPmt** (*rate*, *per*, *nper*, *pv*[, *fv*[, *type*]])

**Именованные аргументы функции IPmt:**

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>rate</i> | Обязательный. Тип <b>Double</b> . Процентная ставка за период. Например, если вы берете заем для покупки автомобиля из расчета 10 процентов годовых (annual percentage rate, APR) и делаете ежемесячные платежи, ставка за период составляет 0.1/12, или 0.0083.                                                                                                                                                                                   |
| <i>per</i>  | Обязательный. Значение типа <b>Double</b> . Период платежа в диапазоне 1– <i>nper</i> .                                                                                                                                                                                                                                                                                                                                                            |
| <i>nper</i> | Обязательный. Значение типа <b>Double</b> . Общее число периодов платежей в ежегодных поступлениях (annuity). Например, если вы вносите ежемесячные платежи в счет погашения четырехлетней ссуды, ваша ссуда имеет всего $4 \times 12$ (или 48) периодов платежа.                                                                                                                                                                                  |
| <i>pv</i>   | Обязательный. Значение <b>Double</b> . Текущая стоимость или общая сумма всех будущих платежей с настоящего момента. Например, когда вы получаете заем, чтобы купить автомобиль, сумма займа – это текущее значение (present value) для кредитора.                                                                                                                                                                                                 |
| <i>fv</i>   | Необязательный. Значение <b>Variant</b> . Будущее значение (future value) или баланс наличности (cash balance), которого необходимо достичь после того, как вы внесете окончательный платеж. Например, будущая величина ссуды равна \$0, потому что это ее величина после окончательного платежа. Однако если вы хотите накопить \$50,000 за 18 лет, то \$50,000 – это будущее значение. Если аргумент отсутствует, то он предполагается равным 0. |
| <i>type</i> | Необязательный. Значение <b>Variant</b> . Определяет, когда должен производиться платеж. Используется 0, если срок платежа наступает в конце периода платежа или используется 1, если срок платежа наступает в начале периода. Если аргумент отсутствует, он предполагается равным 0.                                                                                                                                                              |

Ежегодные поступления (annuity) — это ряд фиксированных наличных платежей (fixed cash payments), внесенных в течение некоторого периода времени. Поступления могут быть ссудой, такой как ипотека (home mortgage) или инвестиции (investment), такие как план ежемесячных накоплений (monthly savings plan).

Аргументы *rate* и *nper* должны вычисляться с использованием периодов платежей, выраженных в одних и тех же единицах. Например, если *rate* вычисляется с использованием месяцев, *nper* также должен вычисляться в месяцах.

Для всех аргументов выплаченные наличные платежи, такие как депозиты (deposits to savings), представляются отрицательными числами; прибыли, такие как дивиденды (dividend checks), представляются положительными числами.

---

**IRR**

Возвращает (значение типа **Double**) внутреннюю скорость оборота для серии периодических операций с наличными.

**Синтаксис**

**IRR**(*values*() [, *guess*])

**Именованные аргументы функции IRR:**

|                 |                                                                                                                                                                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>values()</i> | Обязательный. Массив (тип <b>Double</b> ) со значениями, для которых определяется внутренняя скорость оборота средств. Массив должен содержать, по меньшей мере, одно отрицательное значение (выплата) и одно — положительное (поступление). |
| <i>guess</i>    | Необязательный. Ваша оценка (тип <b>Variant</b> ) возвращаемого функцией результата. Если отсутствует, то полагается равным значению 0.1 (10%).                                                                                              |

Функция **IRR** использует порядок значений массива для интерпретации порядка денежных выплат или поступлений, поэтому следует внимательно указывать этот порядок.

---

**IsArray**

---

Возвращает значение (тип **Boolean**), указывающее, является ли переменная массивом.

**Синтаксис**

**IsArray** (*varname*)

Обязательный аргумент *varname* — идентификатор, определяющий переменную.

Функция **IsArray** возвращает значение **True**, если переменная-аргумент является массивом. В противном случае возвращается значение **False**.

**Примеры**

```
Dim Array1(1 To 5) As Integer, Array2, Rez
Array2 = Array(1, 2, 3)
Rez = IsArray(Array1)      ' True
Rez = IsArray(Array2)     ' True
```

---

**IsDate**

---

Возвращает значение (тип **Boolean**), указывающее, может ли выражение-аргумент быть конвертировано в дату.

**Синтаксис**

**IsDate** (*expression*)

Обязательный аргумент *expression* — значение (тип **Variant**), содержащее выражение даты или любое строковое выражение, распознаваемое как дата или время.

Функция **IsDate** возвращает значение **True**, если выражение-аргумент является датой или может быть представлен как дата; иначе возвращается значение **False**.

**Примеры**

```
Dim DateExp1, DateExp2, NoDate, Rez
DateExp1 = "February 12, 1969"
DateExp2 = #2/12/69#
NoDate = " "
Rez = IsDate(NoDate)      ' False
Rez = IsDate(DateExp1)   ' True
Rez = IsDate(DateExp2)   ' True
```

---

## IsEmpty

---

Возвращает значение (тип **Boolean**), указывающее, была ли переменная-аргумент инициализирована.

### Синтаксис

**IsEmpty** (*expression*)

Обязательный аргумент *expression* — значение (тип **Variant**), содержащее численное или строковое выражение. Но поскольку функция **IsEmpty** используется для определения того, инициализирована ли отдельная переменная, аргумент *expression* — чаще всего имя единственной переменной.

Функция **IsEmpty** возвращает значение **True**, если переменная инициализирована или ей явно было присвоено значение **Empty**; иначе — функция возвращает значение **False**. Значение **False** также возвращается всегда, когда *expression* содержит более одной переменной.

### Примеры

```
Dim My_Var, Rez
MyCheck = IsEmpty(My_Var)      ' True

My_Var = Null
MyCheck = IsEmpty(My_Var)      ' False

My_Var = Empty
MyCheck = IsEmpty(My_Var)      ' True
```

---

## IsError

---

Функция возвращает значение (тип **Boolean**), указывающее, является ли выражение значением ошибки (error-значением).

### Синтаксис

**IsError** (*expression*)

Обязательный аргумент *expression* может быть любым допустимым выражением. Error-значения создаются преобразованием действительных чисел в error-значения с использованием функции **CVErr**. Функция **IsError** используется для определения того, представляет ли числовое выражение ошибку. **IsError** возвращает значение **True**, если аргумент *expression* указывает на ошибку, и — **False** в противном случае.

---

## IsMissing

---

Возвращает значение (тип **Boolean**), указывающее, был ли необязательный аргумент (типа **Variant**) передан в процедуру (указан ли при вызове).

### Синтаксис

**IsMissing** (*argname*)

Обязательный аргумент *argname* содержит имя необязательного аргумента процедуры. Используйте функцию **IsMissing** для определения того, были ли определены при вызове процедуры необязательные аргументы. Функция **IsMissing** возвращает значение **True**, если для определенного аргумента не было при вызове процедуры указано никакого значения; иначе — возвращается значение **False**. Если функция **IsMissing** возвращает значение **True** для некоторого аргумента, использование отсутствующего аргумента в коде может вызвать определенную пользователем ошибку.



---

## IsNull

---

Возвращает значение (тип **Boolean**), которое указывает, содержит ли выражение недопустимые данные (**Null**).

### Синтаксис

**IsNull** (*expression*)

Обязательный аргумент *expression* — (тип **Variant**) численное или строковое выражение. Функция **IsNull** возвращает значение **True**, если *expression* — **Null**; иначе — **IsNull** возвращает **False**. Если *expression* содержит более одной переменной, **Null** в любой переменной приводит к тому, что для всего выражения функция возвращает значение **True**.

Значение **Null** указывает, что **Variant** содержит недопустимые данные. **Null** — это не то же самое, что **Empty**, которое указывает на то, что переменная еще не была инициализирована. Это также не то же самое, что строка нулевой длины ("").

Обычно следует использовать функцию **IsNull** для определения, содержит ли выражение значение **Null**.

### Примеры

```
Dim My_Var, Rez
Rez = IsNull(My_Var)      ' False

My_Var = Null
Rez = IsNull(My_Var)      ' True

My_Var = ""
Rez = IsNull(My_Var)      ' False
```

---

## IsNumeric

---

Возвращает значение (тип **Boolean**), указывающее, является ли результатом вычисления выражения численное значение.

### Синтаксис

**IsNumeric** (*expression*)

Обязательный аргумент *expression* — значение (тип **Variant**), содержащее численное или строковое выражение. Функция **IsNumeric** возвращает значение **True**, если все *expression* распознается как число, иначе — возвращается значение **False**. Если *expression* является датой, функция возвращает значение **False**.

### Примеры

В этом примере используется функция **IsNumeric** для определения, может ли переменная быть оценена как число.

```
StrVar = "763"
BooleanVar = IsNumeric(StrVar)      'возвращает True
```

---

## IsObject

---

Возвращает значение (тип **Boolean**), указывающее, представляет ли идентификатор объектную переменную.

### Синтаксис

**IsObject** (*identifier*)

Обязательный аргумент *identifier* — имя переменной. Функция **IsObject** возвращает значение **True**, если аргумент *identifier* является переменной, объявленной как **Object**-тип или как тип любого допустимого класса, или если *identifier* — **Variant**-переменная **VarType vbObject**, или объект типа, определенного пользователем; иначе — возвращается значение **False**. Функция **IsObject** возвращает значение **True**, даже если переменной было присвоено значение **Nothing**.

---

## Join

Возвращает строку, объединяя несколько подстрок, находящихся в массиве.

### Синтаксис

**Join**(*sourcearray*[, *delimiter*])

Функция **Join** принимает следующие именованные аргументы:

|                    |                                                                                                                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>sourcearray</i> | Обязательный. Одномерный массив, содержащий подстроки, которые следует объединить.                                                                                                                                                               |
| <i>delimiter</i>   | Необязательный. Символ, используемый для разделения подстрок в возвращаемой строке. Если параметр опускается, используется символ пробела (" "). Если этот аргумент является строкой нулевой длины (""), подстроки объединяются без разделителя. |

### Примеры

```
Dim array_str(3)
array_str(1) = "Первый"
array_str(2) = "Второй"
array_str(3) = "Третий"
MsgBox Join(array_str, ":")
```

'вывод: "Первый:Второй:Третий "

---

## LBound

Функция возвращает значение (тип **Long**), содержащее наименьшее значение индекса массива.

### Синтаксис

**LBound**(*arrayname*[, *dimension*])

Функция **LBound** принимает следующие аргументы:

|                  |                                                                                                                                                                                                                                                                            |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>arrayname</i> | Обязательный. Имя переменной-массива.                                                                                                                                                                                                                                      |
| <i>dimension</i> | Необязательный; тип <b>Variant (Long)</b> . Целое число, определяющее, границу какой из размерностей следует возвратить. Для первой размерности указывайте значение 1, для второй — 2, и так далее. Если аргумент <i>dimension</i> опускается, он предполагается равным 1. |

Функцию **LBound** можно использовать совместно с функцией **UBound** для определения размера массива, поскольку последняя возвращает наибольшее значение индекса массива.

### Пример

```
Dim Rez
Dim A(4 To 100, 0 To 3, -5 To 4)
Rez = LBound(A, 1)      'возвращает 4
Rez = LBound(A, 2)      'возвращает 0
Rez = LBound(A, 3)      'возвращает -5
```

---

## LCase

---

Возвращает значение (тип **String**), которое преобразовано к нижнему регистру.

### Синтаксис

**LCase** (*string*)

Обязательный аргумент *string* — любое допустимое строковое выражение. Если *string* содержит **Null**, возвращается **Null**. К нижнему регистру преобразуются только символы верхнего регистра. Остальные символы остаются без изменений.

### Пример

```
Dim Ustring, Lstring
Ustring = "Hello Nike-2"
Lstring = Lcase(Ustring) ' возвращает "hello nike-2"
```

---

## Left

---

Возвращает значение [тип **Variant (String)**], содержащее указанное количество символов с левой стороны строки.

### Синтаксис

**Left** (*string*, *length*)

Функция **Left** принимает следующие именованные аргументы:

- |               |                                                                                                                                                                                                                                                               |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>string</i> | Обязательный. Строковое выражение, из которого возвращается указанное количество символов. Если аргумент <i>string</i> содержит <b>Null</b> , возвращается также <b>Null</b> .                                                                                |
| <i>length</i> | Обязательный; тип <b>Variant (Long)</b> . Численное выражение, определяющее количество возвращаемых символов строки. Если это значение равно 0, возвращается строка нулевой длины (""). Если аргумент больше или равен длине строки, возвращается вся строка. |

### Примеры

```
Dim StrMY
StrMY = Left("Hello World", 2) 'возвращает "He"
StrMY = Left("Hello World", 6) 'возвращает "Hello "
StrMY = Left("Hello World", 20) 'возвращает "Hello World"
```

---

## Len

---

Возвращает значение (тип **Long**), содержащее количество символов (длину) строки или число байт, необходимых для сохранения переменной.

### Синтаксис

**Len** (*string* | *varname*)

Функция **Len** принимает следующие аргументы:

- |                |                                                                                                                                                                                                                                                |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>string</i>  | Любое допустимое строковое выражение. Если аргумент содержит <b>Null</b> , функция возвращает <b>Null</b> .                                                                                                                                    |
| <i>varname</i> | Любое допустимое имя переменной. Если <i>varname</i> содержит <b>Null</b> , функция возвращает <b>Null</b> . Если <i>varname</i> — <b>Variant</b> , функция <b>Len</b> «рассматривает» его как строку и возвращает число символов в аргументе. |

**Loc**

Возвращает значение (тип **Long**), определяющее текущую позицию чтения/записи в открытом файле.

**Синтаксис**

**Loc** (*filenumber*)

Обязательный аргумент *filenumber* — любой допустимый файловый номер (тип **Integer**).

В следующей таблице приведены режимы файлового доступа и соответствующие им возвращаемые значения функции **Loc**.

| режим             | возвращаемое значение                                                         |
|-------------------|-------------------------------------------------------------------------------|
| <b>Random</b>     | Номер последней записи, которая считывалась из файла или записывалась в файл. |
| <b>Sequential</b> | Текущая байтовая позиция в файле, деленная на число 128.                      |
| <b>Binary</b>     | Позиция последнего считанного или записанного байта.                          |

**Примеры**

В этом примере открывается текстовый файл и в цикле происходит посимвольное считывание и вывод в окно **Immediate** считанного символа и номера текущей байтовой позиции в файле.

```

Open "F:\ZZ\WWW.TXT" For Input As #1      'открыть файл
Do While Not EOF(1)                        'цикл до конца файла
    CharOne = Input(1, #1)                 'считать символ
    MyLocation = Loc(1)                     'текущая позиция
    Debug.Print CharOne; MyLocation         'отобразить в Immediate-окне
Loop
Close #1                                   ' закрыть файл

```

**LOF**

Возвращает значение (тип **Long**), представляющее размер (в байтах) файла, открытого с использованием оператора **Open**.

**Синтаксис**

**LOF** (*filenumber*)

Обязательный аргумент *filenumber* — число (тип **Integer**), содержащее допустимый номер файла.

В качестве примера см. код для функции **Loc**.

**Log**

Возвращает значение (тип **Double**), определяющее натуральный логарифм числа.

**Синтаксис**

**Log** (*number*)

Обязательный аргумент *number* — число (тип **Double**) или допустимое численное выражение, большее нуля. Натуральный логарифм — это логарифм с основанием *e*, где константа *e* равна примерно **2.718282**.

---

**LTrim**


---

Функция возвращает значение [тип **Variant (String)**], содержащее копию указанной строки без ведущих (leading) пробелов.

**Синтаксис**

**LTrim**(*string*)

Обязательный аргумент *string* — любое допустимое строковое выражение. Если аргумент содержит **Null**, функция возвращает **Null**.

---

**Mid**

Возвращает значение [тип **Variant (String)**], содержащее указанное число символов из строки.

**Синтаксис**

**Mid**(*string*, *start*[, *length*])

Функция **Mid** принимает следующие именованные аргументы:

|               |                                                                                                                                                                                                                                                                          |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>string</i> | Обязательный. Строковое выражение (тип <b>String</b> ), из которого возвращается определенное число символов. Если аргумент <i>string</i> содержит <b>Null</b> , функция возвращает <b>Null</b> .                                                                        |
| <i>start</i>  | Обязательный; тип <b>Long</b> . Символьная позиция в строке <i>string</i> , с которой следует начинать выделение строки. Если значение аргумента <i>start</i> больше длины заданной строки, функция возвращает строку нулевой длины.                                     |
| <i>length</i> | Необязательный; тип <b>Variant (Long)</b> . Число возвращаемых из строки символов. Если опускается или если символов в строке меньше, чем <i>length</i> , включая символ в позиции <i>start</i> , функция возвращает все символы с позиции <i>start</i> до конца строки. |

**Примеры**

```
SFirstWord = Mid("Mid Function Example", 1, 3)      ' "Mid"
SLastWord = Mid("Mid Function Example", 14 )      ' "Example"
SMidWords = Mid("Mid Function Example", 5, 8)      ' "Function"
```

---

**Minute**

Функция возвращает [тип **Variant (Integer)**] целое число от 0 до 59, включительно, которое представляет минуты часа.

**Синтаксис**

**Minute**(*time*)

Обязательный аргумент *time* — численное (тип **Variant**) выражение, строковое выражение или любая комбинация, представляющая время. Если аргумент *time* содержит **Null**, функция возвращает **Null**.

**Примеры**

```
DTime = #4:35:17 PM#
DMinute = Minute(DTime)      ' возвращаемое значение - 35
```

---

**MIRR**

---

Возвращает (тип **Double**) модифицируемую внутреннюю скорость оборота для ряда периодических операций с наличными.

**Синтаксис**

**MIRR**(*values()*, *finance\_rate*, *reinvest\_rate*)

Именованные аргументы функции **MIRR**:

|                      |                                                                                                                                                                                                                                              |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>values()</i>      | Обязательный. Массив (тип <b>Double</b> ) со значениями, для которых определяется внутренняя скорость оборота средств. Массив должен содержать, по меньшей мере, одно отрицательное значение (выплата) и одно — положительное (поступление). |
| <i>finance_rate</i>  | Обязательный. Норма прибыли (тип <b>Double</b> ), выплачиваемой за денежные средства, находящиеся в наличном обороте.                                                                                                                        |
| <i>reinvest_rate</i> | Обязательный. Норма прибыли (тип <b>Double</b> ), получаемой за денежные средства, находящиеся в наличном обороте при инвестировании.                                                                                                        |

Функция **MIRR** учитывает как стоимость инвестиций, так и доход от реинвестирования, и использует порядок значений массива для интерпретации порядка денежных выплат или поступлений. Следует внимательно указывать этот порядок.

---

**Month**

---

Возвращает целое значение [тип **Variant (Integer)**] в диапазоне от 1 до 12, включительно, представляющее месяц года.

**Синтаксис**

**Month**(*date*)

Обязательный аргумент *date* — (тип **Variant**) численное выражение, строковое выражение или любая комбинация, которая представляет дату. Если аргумент *date* содержит **Null**, функция возвращает **Null**. Возвращаемое значение функции зависит от значения свойства **Calendar**.

**Примеры**

```
DDateMy = #February 12, 1969#  
DMonthMy = Month(MyDate)      ' возвращает 2
```

---

**MonthName**

---

Возвращает строку, указывающую определенный месяц.

**Синтаксис**

**MonthName**(*month*[, *abbreviate*])

Функция **MonthName** имеет следующие аргументы:

|                   |                                                                                                                                                                                                                                                            |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>month</i>      | Обязательный. Числовое представление месяца. Например, Январь (January) — это 1, Февраль (February) — 2 и т.д.                                                                                                                                             |
| <i>abbreviate</i> | Необязательный. Тип <b>Boolean</b> . Значение, которое указывает, следует ли использовать сокращение для отображения имени месяца. Если аргумент опущен, то по умолчанию принимается значение <b>False</b> , которое означает, сокращение не используется. |

## MsgBox

Отображает сообщение в диалоговом окне, ожидает нажатия на кнопку пользователем и возвращает значение (тип **Integer**), указывающее, какую кнопку выбрал пользователь.

### Синтаксис

**MsgBox** (*prompt* [, *buttons*] [, *title*] [, *helpfile*, *context*])

Функция **MsgBox** имеет следующие именованные аргументы:

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>prompt</i>   | Обязательный. Строковое выражение, отображаемое в диалоговом окне. Строка может содержать примерно 1024 символа (в зависимости от используемой ширины символов). Если необходимо, чтобы выражение <i>prompt</i> состояло из нескольких строк, можно для разделения строк использовать символы возврата каретки ( <b>Chr(13)</b> ), перехода на другую строку ( <b>Chr(10)</b> ) или комбинацию этих символов ( <b>Chr(13) &amp; Chr(10)</b> ). |
| <i>buttons</i>  | Необязательный. Числовое выражение, являющееся суммой значений, которые определяют количество и тип кнопок для отображения, используемый значок, кнопку по умолчанию и модальность диалогового окна. Если не используется, принимается равным значению 0.                                                                                                                                                                                      |
| <i>title</i>    | Необязательный. Строковое выражение для заголовка диалогового окна. Если не используется, в заголовке отображается наименование приложения.                                                                                                                                                                                                                                                                                                    |
| <i>helpfile</i> | Необязательный. Строковое выражение, которое идентифицирует Help-файл для обеспечения контекстно-зависимой помощи. Если этот аргумент используется, <b>context</b> также должен использоваться.                                                                                                                                                                                                                                                |
| <i>context</i>  | Необязательный. Числовое выражение, указывающее раздел (в справочном файле), относящийся к отображаемому диалоговому окну.                                                                                                                                                                                                                                                                                                                     |

Аргумент *buttons* может принимать следующие значения (константы):

| константа                 | значение | описание                                                        |
|---------------------------|----------|-----------------------------------------------------------------|
| <b>vbOKOnly</b>           | 0        | Отображать только кнопку <b>OK</b> .                            |
| <b>vbOKCancel</b>         | 1        | Отображать кнопки <b>OK</b> и <b>Cancel</b> .                   |
| <b>vbAbortRetryIgnore</b> | 2        | Отображать кнопки <b>Abort</b> , <b>Retry</b> и <b>Ignore</b> . |
| <b>vbYesNoCancel</b>      | 3        | Отображать кнопки <b>Yes</b> , <b>No</b> и <b>Cancel</b> .      |
| <b>vbYesNo</b>            | 4        | Отображать кнопки <b>Yes</b> и <b>No</b> .                      |
| <b>vbRetryCancel</b>      | 5        | Отображать кнопки <b>Retry</b> и <b>Cancel</b> .                |
| <b>vbCritical</b>         | 16       | Отображать значок <b>Critical Message</b> .                     |
| <b>vbQuestion</b>         | 32       | Отображать значок <b>Warning Query</b> .                        |
| <b>vbExclamation</b>      | 48       | Отображать значок <b>Warning Message</b> .                      |
| <b>vbInformation</b>      | 64       | Отображать значок <b>Information Message</b> .                  |

|                              |         |                                                                                                                                               |
|------------------------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>vbDefaultButton1</b>      | 0       | Первая кнопка — кнопка по умолчанию.                                                                                                          |
| <b>vbDefaultButton2</b>      | 256     | Вторая кнопка — кнопка по умолчанию.                                                                                                          |
| <b>vbDefaultButton3</b>      | 512     | Третья кнопка — кнопка по умолчанию.                                                                                                          |
| <b>vbDefaultButton4</b>      | 768     | Четвертая кнопка — кнопка по умолчанию.                                                                                                       |
| <b>vbApplicationModal</b>    | 0       | Приложение — модальное; пользователь должен сначала «ответить» на диалоговое окно, прежде чем сможет продолжить работу с текущим приложением. |
| <b>vbSystemModal</b>         | 4096    | Система — модальная; все приложения ожидают «ответа» на диалоговое окно.                                                                      |
| <b>vbMsgBoxHelpButton</b>    | 16384   | Добавляет кнопку <b>Help</b> к диалоговому окну.                                                                                              |
| <b>VbMsgBoxSetForeground</b> | 65536   | Определяет диалоговое окно как окно переднего плана.                                                                                          |
| <b>vbMsgBoxRight</b>         | 524288  | Текст выравнивается по правой границе.                                                                                                        |
| <b>vbMsgBoxRtlReading</b>    | 1048576 | Определяет, будет ли появляться текст справа-налево в системах Hebrew и Arabic.                                                               |

В этой таблице первая группа констант (0–5) описывает типы кнопок, отображаемых в диалоговом окне; следующая группа (16, 32, 48, 64) — стили значков; третья группа (0, 256, 512) определяет кнопку умолчания (кнопка, которая будет «срабатывать» при нажатии на клавишу **Enter**); четвертая группа (0, 4096) задает модальность диалогового окна.

Функция **MsgBox** возвращает следующие значения:

| константа       | значение | выбранная кнопка |
|-----------------|----------|------------------|
| <b>vbOK</b>     | 1        | <b>OK</b>        |
| <b>vbCancel</b> | 2        | <b>Cancel</b>    |
| <b>vbAbort</b>  | 3        | <b>Abort</b>     |
| <b>vbRetry</b>  | 4        | <b>Retry</b>     |
| <b>vbIgnore</b> | 5        | <b>Ignore</b>    |
| <b>vbYes</b>    | 6        | <b>Yes</b>       |
| <b>vbNo</b>     | 7        | <b>No</b>        |

Если используются оба аргумента *helpfile* и *context*, для вызова справочного файла, связанного с диалоговым окном **MsgBox**, достаточно нажать на клавиатуре клавишу F1. Если диалоговое окно содержит кнопку **Cancel**, то вместо щелчка на этой кнопке можно нажимать на клавишу Esc на клавиатуре.

## Now

Возвращает текущую дату и время [тип **Variant (Date)**], используя системные дату и время компьютера.

### Синтаксис

**Now** ( )



---

**NPer**

---

Возвращает количество (тип **Double**) периодов выплаты для данного вклада на основе периодических фиксированных выплат и фиксированной процентной ставки.

**Синтаксис**

**NPer**(rate, pmt, pv[, fv[, type]])

Функция **NPer** имеет следующие именованные аргументы:

- rate* Обязательный. Ставка (тип **Double**) — процентная ставка за период. Например, если вы получаете заем для покупки автомашины из расчета 10 процентов годовых (APR) и вносите ежемесячные платежи, то ставка за период составит 0.1/12 или 0.0083.
- pmt* Обязательный. Выплата (тип **Double**), производимая в каждый период. Выплаты обычно состоят из основного платежа и платежа по процентам и не изменяются в течение срока займа.
- pv* Обязательный. Тип **Double**. Текущее значение или величина сегодня всех будущих платежей или поступлений. Например, если вы получаете заем для покупки автомашины, сумма займа является текущим значением для кредитора ежемесячных платежей, которые вы будете вносить.
- fv* Необязательный. Значение типа **Variant**. Будущее значение или баланс наличности, который необходимо достичь после последнего платежа. Например, будущее значение займа равно \$0 поскольку это — его значение после внесения последнего платежа. Однако если вы хотите накопить \$50,000 в течение 18 лет на образование вашего ребенка, то \$50,000 будет будущим значением. Если аргумент опущен, предполагается равным 0.
- type* Необязательный. Значение типа **Variant**, обозначающее, когда должна производиться выплата. Используется 0, если срок платежа наступает в конце периода платежа, или 1, если выплата должна производиться в начале периода платежа. Если аргумент опущен, предполагается равным 0.

Ежегодные поступления (по займу и т.д., аннуитет) — это ряд фиксированных наличных платежей в течение периода времени. Аннуитет может быть ссудой, такой как ипотека (home mortgage) или вкладами, такими как план ежемесячной экономии (monthly savings plan).

Для всех аргументов наличные выплаты, такие как вклады (deposits to savings), представляются отрицательными числами, наличные поступления, такие как дивиденды (dividend checks), представляются положительными числами.

---

**NPV**

---

Возвращает значение типа **Double**, определяющее чистый текущий объем вклада на базе ряда периодических операций с наличностью (платежей и поступлений) и учетной ставки (discount rate).

**Синтаксис**

**NPV**(rate, values())

Именованные аргументы функции **NPV**:

- rate* Обязательный. Тип **Double**, определяет учетную ставку за период.

*values()* Обязательный. Массив типа **Double**. Определяет значения операций с наличностью. Массив должен содержать, по меньшей мере, одно отрицательное значение (платеж) и одно положительное значение (поступление).

Чистый текущий объем вклада — это текущее значение ряда будущих платежей и поступлений.

Функция **NPV** использует порядок значений в массиве для определения порядка платежей и поступлений. Убедитесь, что значения ваших платежей и поступлений введены в правильной последовательности.

Инвестиция, значение которой вычисляет функция **NPV**, начинается за один период до даты первой наличной операции (денежного взноса) и заканчивается последним значением наличной операции (денежного взноса) в массиве.

Вычисление чистого текущего значения вклада основывается на будущих наличных операциях (денежных взносах). Если ваш первый денежный взнос производится в начале первого периода, то первое значение следует добавить к значению, возвращаемому функцией **NPV**, но не включать в список аргументов *values()*.

Функция **NPV** аналогична функции **PV** (текущее значение), за исключением того, что функция **PV** допускает, чтобы денежные взносы производились либо в конце, либо в начале периода. В отличие от переменных значений денежных взносов в функции **NPV**, значения денежных взносов функции **PV** должны быть фиксированными в течение всего периода инвестиции.

## Oct

Возвращает значение [тип **Variant (String)**], являющееся восьмеричным представлением числа.

### Синтаксис

**Oct** (*number*)

Обязательный аргумент *number* — любое допустимое численное или строковое выражение. Если аргумент *number* не является целым числом, его значение округляется до целого, а затем производится преобразование к восьмеричному представлению.

Функция **Oct** возвращает следующие значения:

| если <i>number</i> | функция <b>Oct</b> возвращает |
|--------------------|-------------------------------|
| <b>Null</b>        | <b>Null</b>                   |
| <b>Empty</b>       | Нуль (0)                      |
| Любое другое число | Восьмеричные цифры            |

### Примеры

```
OctRep = Oct(4)           '      4
OctRep = Oct(9)           '     11
OctRep = Oct(459)        '   713
```

## Partition

Возвращает значение [тип **Variant (String)**], показывающее диапазон (из серии диапазонов), внутри которого находится заданное число.

**Синтаксис**

**Partition**(*number*, *start*, *stop*, *interval*)

Функция **Partition** принимает следующие именованные аргументы:

|               |                                                                                                                         |
|---------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>number</i> | Обязательный. Целое число, которое необходимо оценить относительно диапазонов.                                          |
| <i>start</i>  | Обязательный. Целое число — начало всего диапазона чисел. Это число не может быть меньше нуля.                          |
| <i>stop</i>   | Обязательный. Целое число — конец всего диапазона чисел. Это число не может быть равным или меньшим, чем <i>start</i> . |

Каждый диапазон создается с использованием аргументов *start*, *stop* и *interval*.

Функцию **Partition** удобно использовать в запросах (queries). Вы можете создать select-запрос, который покажет, как много заказов попадает в различные диапазоны, например, значения заказов от 1 до 1000, от 1001 до 2000 и т.д.

Если любой из аргументов функции — **Null**, функция **Partition** возвращает **Null**.

**Примеры**

|                                |                   |
|--------------------------------|-------------------|
| Var = Partition(3, 1, 20, 5)   | 'возвращает 1:5   |
| Var = Partition(14, 1, 20, 5)  | 'возвращает 11:15 |
| Var = Partition(14, 1, 20, 10) | 'возвращает 11:20 |

**Pmt**

Возвращает значение типа **Double**. Вычисляет величину выплаты по ссуде на основе периодических, фиксированных платежей и фиксированной процентной ставки.

**Синтаксис**

**Pmt**(*rate*, *nper*, *pv*[, *fv*[, *type*]])

Именованные аргументы функции **Pmt**:

|             |                                                                                                                                                                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>rate</i> | Обязательный. Тип <b>Double</b> . Процентная ставка за период. Например, если вы получаете заем на покупку автомашины из расчета 10 процентов годовых (APR) и вносите ежемесячные платежи, ставка за период составляет 0.1/12, или 0.0083.                          |
| <i>nper</i> | Обязательный. Значение типа <b>Integer</b> . Общее число периодов выплат ежегодных поступлений (аннуитет). Например, если вы делаете ежемесячные выплаты по четырехлетнему займу, ваш заем имеет в сумме 4×12 (или 48) периодов выплат.                             |
| <i>pv</i>   | Обязательный. Тип <b>Double</b> . Текущее значение (или общая сумма), которое составит ряд будущих платежей. Например, если вы берете в долг деньги для покупки автомашины, сумма займа является текущим значением для кредитора выплат, которые вы будете вносить. |

|             |                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>fv</i>   | Необязательный. Тип <b>Variant</b> . Будущая сумма или баланс наличности, которого необходимо достичь, после того как вы внесете последний платеж. Например, будущее значение займа равно \$0, потому что это его значение после последней выплаты. Однако если вы хотите накопить \$50,000 в течение 18 лет на образование вашего ребенка, то будущей суммой является \$50,000. Если опущен, то предполагается равным 0. |
| <i>type</i> | Необязательный. Тип <b>Variant</b> . Определяет, когда должна производиться выплата. Используется 0, если платеж должен производиться в конце периода, или 1, если платеж должен производиться в начале периода. Если опущен, предполагается равным 0.                                                                                                                                                                    |

Ежегодные поступления (annuity) — это ряд фиксированных наличных платежей, производимых в течение периода времени. Аннуитет может быть ссудой, такой как ипотека (home mortgage), или инвестицией, такой как план ежемесячных накоплений (monthly savings plan).

Аргументы *rate* и *nper* должны вычисляться с использованием периодов выплат, выраженных в одних и тех же единицах. Например, если *rate* вычисляется с использованием месяцев, *nper* также должен вычисляться с использованием месяцев.

Для всех аргументов наличные выплаты, такие как депозиты и накопления, представляются отрицательными числами; наличные поступления, такие как дивиденды-чеки (dividend checks), представляются положительными числами.

---

## PPmt

---

Возвращает значение типа **Double**. Величина основного платежа на данный период займа на основе периодических, фиксированных платежей и фиксированной процентной ставки.

### Синтаксис

**PPmt**(*rate*, *per*, *nper*, *pv*[, *fv*[, *type*]])

Именованные аргументы функции **PPmt**:

|             |                                                                                                                                                                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>rate</i> | Обязательный. Тип <b>Double</b> . Процентная ставка за период. Например, если вы получаете заем на покупку автомашины из расчета 10 процентов годовых (APR) и вносите ежемесячные платежи, ставка за период равна 0.1/12, или 0.0083.         |
| <i>per</i>  | Обязательный. Тип <b>Integer</b> . Задаёт период в интервале 1– <i>nper</i> .                                                                                                                                                                 |
| <i>nper</i> | Обязательный. Тип <b>Integer</b> . Это — общее число периодов выплат годовой ссуды. Например, если вы вносите ежемесячные платежи в счет погашения четырехлетней ссуды на автомашину, ваш заем имеет всего 4 × 12 (или 48) периодов платежей. |
| <i>pv</i>   | Обязательный. Тип <b>Double</b> . Это — текущее значение или значение сегодня ряда будущих платежей или поступлений. Например, если вы берете заем для покупки автомашины, сумма займа является текущим значением для кредитора.              |

- fv* Необязательный. Тип **Variant**. Это — будущая сумма или баланс наличности, которого необходимо достичь после того, как вы внесете последний платеж. Например, будущая сумма ссуды равна \$0, таково ее значение после последнего платежа. Однако если вы хотите накопить \$50,000 в течение 18 лет на образование вашего ребенка, то будущим значением является \$50,000. Если опущен, предполагается равным 0.
- type* Необязательный. Тип **Variant**, определяет, когда должен производиться платеж. Используется 0, если платеж должен производиться в конце периода, или используется 1, если делать выплату необходимо в начале периода. Если опущен, предполагается равным 0.

Ежегодные поступления (annuity) — это ряд фиксированных наличных платежей, производимых в течение периода времени. Аннуитет может быть ссудой, такой как ипотека (home mortgage), или инвестицией, такой как план ежемесячных накоплений (monthly savings plan).

Аргументы *rate* и *nper* должны вычисляться с использованием периодов выплат, выраженных в одних и тех же единицах. Например, если *rate* вычисляется с использованием месяцев, *nper* также должен вычисляться с использованием месяцев.

Для всех аргументов наличные выплаты, такие как депозиты и накопления, представляются отрицательными числами; наличные поступления, такие как дивиденды-чеки (dividend checks), представляются положительными числами.

---

## PV

---

Возвращает значение типа **Double**, определяющее текущий объем вклада (займа) на основе будущих периодических, фиксированных платежей и фиксированной процентной ставки.

### Синтаксис

**PV**(*rate*, *nper*, *pmt*[, *fv*[, *type*]])

Именованные аргументы функции **PV** :

- rate* Обязательный. Тип **Double**. Процентная ставка за период. Например, если вы получаете ссуду на покупку автомашины из расчета 10 процентов годовых (APR) и вносите ежемесячные платежи, то процентная ставка за период равна 0.1/12, или 0.0083.
- nper* Обязательный. Тип **Integer**. Это — общее число периодов платежей годовой ссуды. Например, если вы вносите ежемесячные платежи в счет погашения четырехлетней ссуды, ваша ссуда имеет всего 4×12 (или 48) периодов.
- pmt* Обязательный. Тип **Double**. Это — платеж, производимый в каждый период. Платежи обычно включают основные платежи и платежи по процентам, которые не изменяются за все время годовых поступлений.
- fv* Необязательный. Тип **Variant**, обозначает будущее значение или наличный баланс, которого необходимо достичь после того, как будет внесен последний платеж. Например, будущая стоимость займа равна \$0, потому что таково его значение после последней выплаты. Однако если вы хотите накопить \$50,000 за 18 лет на образование вашего ребенка, то будущее значение равняется \$50,000. Если опущен, предполагается равным 0.

**type** Необязательный. Тип **Variant**, обозначает, когда должен производиться платеж. используется 0, если платеж должен производиться в конце периода, или 1, если платеж должен производиться в начале периода. Если опущен, предполагается равным 0.

Ежегодные поступления (annuity) — это ряд фиксированных наличных платежей, производимых в течение периода времени. Аннуитет может быть ссудой, такой как ипотека (home mortgage), или инвестицией, такой как план ежемесячных накоплений (monthly savings plan).

Аргументы *rate* и *nper* должны вычисляться с использованием периодов выплат, выраженных в одних и тех же единицах. Например, если, *rate* вычисляется с использованием месяцев, *nper* также должен вычисляться с использованием месяцев.

Для всех аргументов наличные выплаты, такие как депозиты и накопления, представляются отрицательными числами; наличные поступления, такие как дивиденды-чеки (dividend checks), представляются положительными числами.

---

## QBColor

---

Возвращает значение (тип **Long**) кода RGB-цвета.

### Синтаксис

**QBColor** (*color*)

Обязательный аргумент *color* — целое число в диапазоне 0–15:

| число | цвет                | число | цвет                        |
|-------|---------------------|-------|-----------------------------|
| 0     | Black (черный)      | 8     | Gray (серый)                |
| 1     | Blue (голубой)      | 9     | Light Blue (светло голубой) |
| 2     | Green (зеленый)     | 10    | Light Green                 |
| 3     | Cyan (бирюзовый)    | 11    | Light Cyan                  |
| 4     | Red (красный)       | 12    | Light Red                   |
| 5     | Magenta (малиновый) | 13    | Light Magenta               |
| 6     | Yellow (желтый)     | 14    | Light Yellow                |
| 7     | White (белый)       | 15    | Bright White                |

Аргумент *color* представляет значение цвета, используемого ранними версиями системы Basic (такими как Microsoft Visual Basic for MS-DOS и Basic Compiler). Начиная с младшего значащего байта, возвращаемое значение определяет red-, green- и blue-значения, используемые для установки соответствующего цвета в RGB-системе для Visual Basic for Applications.

### Примеры

```
Sub ChangeBackColor (IntColorCode As Integer, FormMy As Form)
    FormMy.BackColor = QBColor(IntColorCode)
End Sub
```

Процедура в этом примере позволяет менять свойство формы **BackColor**. Имя формы (как и код цвета) передается как параметр.

---

**Rate**


---

Возвращает значение типа **Double**, определяющее процентную ставку за период при выплате годовых поступлений (annuity).

**Синтаксис**

**Rate**(*nper*, *pmt*, *pv*[, *fv*[, *type*[, *guess*]]])

Полное описание аргументов *nper*, *pmt*, *pv*, *fv* и *type* см. в справке по функции **PV**. Именованные аргументы функции **Rate**:

|              |                                                                                                                                                                                                                                                                 |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>nper</i>  | Обязательный. Значение типа <b>Double</b> , обозначает общее число периодов выплат годовых поступлений. Например, если вы вносите ежемесячные платежи по четырехлетнему займу на покупку автомашины, то ваш заем имеет всего 4 × 12 (или 48) периодов платежей. |
| <i>pmt</i>   | Обязательный. Тип <b>Double</b> . Это — платеж, вносимый в каждый период. Платежи обычно состоят из основного платежа и платежа по процентам и не меняются в течение всего периода выплат.                                                                      |
| <i>pv</i>    | Обязательный. Тип <b>Double</b> . Это — текущее значение или сумма сегодня ряда будущих платежей или поступлений. Например, когда вы берете заем для покупки автомашины, сумма займа является настоящим значением для кредитора.                                |
| <i>fv</i>    | Необязательный. Тип <b>Variant</b> . Это — будущее значение или баланс наличности, которого необходимо достичь после последнего платежа. Например, будущее значение для займа равняется \$0. Если аргумент <i>fv</i> опущен, то он предполагается равным 0.     |
| <i>type</i>  | Необязательный. Тип <b>Variant</b> . Число 0, если платеж должен производиться в конце периода, или 1, если платеж должен производиться в начале периода. Если аргумент опущен, он предполагается равным 0.                                                     |
| <i>guess</i> | Необязательный. Тип <b>Variant</b> . Это — предполагаемая величина нормы, возвращаемая функцией <b>Rate</b> . Если аргумент <i>guess</i> опущен, то он предполагается равным 0.1 (10 процентам).                                                                |

Ежегодные поступления (annuity) — это ряд фиксированных наличных платежей, производимых в течение периода времени. Аннуитет может быть ссудой, такой как ипотека (home mortgage), или инвестицией, такой как план ежемесячных накоплений (monthly savings plan).

Для всех аргументов наличные выплаты, такие как депозиты и накопления, представляются отрицательными числами; наличные поступления, такие как дивиденды-чеки (dividend checks), представляются положительными числами.

**Rate** вычисляется методом последовательного приближения и может не иметь решения или иметь несколько решений. Начиная со значения *guess*, функция **Rate** выполняет циклически вычисления до тех пор, пока результат не будет в пределах 0.00001 процента. Если после 20 итераций **Rate** не может найти результат, функция заканчивается неуспешно. Если ваше значение *guess* равно 10 процентам и функция **Rate** заканчивается неуспешно, попробуйте использовать другие значения для *guess*.

## Replace

Возвращает строку, в которой определенная подстрока заменена на другую подстроку указанное число раз.

### Синтаксис

**Replace**(*expression*, *find*, *replace*[, *start*[, *count*[, *compare*]]])

Функция **Replace** принимает следующие именованные аргументы:

|                   |                                                                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>expression</i> | Обязательный. Строковое выражение, содержащее подстроку для замены.                                                                                             |
| <i>find</i>       | Обязательный. Подстрока, которую необходимо найти.                                                                                                              |
| <i>replace</i>    | Обязательный. Строка для замены.                                                                                                                                |
| <i>start</i>      | Необязательный. Позиция внутри <i>expression</i> , с которой начинается поиск подстроки. Если опускается, используется значение 1.                              |
| <i>count</i>      | Необязательный. Необходимое число подстановок подстроки. Если опускается, по умолчанию используется число -1, означающее выполнение всех возможных подстановок. |
| <i>compare</i>    | Необязательный. Числовое значение, определяющее тип сравнения. См. следующую таблицу.                                                                           |

Аргумент *compare* может принимать следующие значения:

| константа                 | значение | описание                                                                                             |
|---------------------------|----------|------------------------------------------------------------------------------------------------------|
| <b>vbUseCompareOption</b> | -1       | Выполнять сравнение с использованием установки оператора <b>Option Compare</b> .                     |
| <b>vbBinaryCompare</b>    | 0        | Выполнять двоичное сравнение.                                                                        |
| <b>vbTextCompare</b>      | 1        | Выполнять текстовое сравнение.                                                                       |
| <b>vbDatabaseCompare</b>  | 2        | Только для Microsoft Access. Выполнять сравнение, основываясь на информации в некоторой базе данных. |

Функция **Replace** возвращает следующие значения:

| если                                            | функция <b>Replace</b> возвращает                                     |
|-------------------------------------------------|-----------------------------------------------------------------------|
| <i>expression</i> — нулевой длины               | строку нулевой длины ("" )                                            |
| <i>expression</i> — <b>Null</b>                 | ошибку                                                                |
| <i>find</i> — нулевой длины                     | копию строки <i>expression</i>                                        |
| <i>replace</i> — нулевой длины                  | копию строки <i>expression</i> со всеми удалениями строки <i>find</i> |
| <i>start</i> > <b>Len</b> ( <i>expression</i> ) | строку нулевой длины                                                  |
| <i>count</i> — 0                                | копию строки <i>expression</i>                                        |



Возвращаемое значение функции **Replace** — строка с выполненными подстановками.

### Пример

```
vexpression = "Привет всем, кто добрый, и ничего всем, кто не добрый"
vFind = "добрый"
vReplace = "хороший"
'замена всех слов "добрый" на слово "хороший":
MsgBox Replace(vexpression, vFind, vReplace)
vReplace = ""
'удаление всех слов "добрый"
MsgBox Replace(vexpression, vFind, vReplace)
```

## RGB

Возвращает целое число (тип **Long**), представляющее значение RGB-цвета.

### Синтаксис

**RGB**(*red*, *green*, *blue*)

Функция **RGB** принимает следующие именованные аргументы:

- red*            Обязательный; тип **Variant (Integer)**. Число в диапазоне 0–255, включительно, которое представляет красную (*red*) компоненту цвета.
- green*          Обязательный; тип **Variant (Integer)**. Число в диапазоне 0–255, включительно, которое представляет зеленую (*green*) компоненту цвета.
- blue*            Обязательный; **Variant (Integer)**. Число в диапазоне 0–255, включительно, которое представляет голубую (*blue*) компоненту цвета.

Значение RGB-цвета определяет относительную интенсивность красного, зеленого и голубого для получения необходимого цвета, который будет отображаться на экране.

Если значение некоторого аргумента функции **RGB** превышает 255, оно заменяется на значение 255.

В следующей таблице приведены стандартные цвета и соответствующие им значения красного, зеленого и голубого цветов.

| цвет    | Red-значение | Green-значение | Blue-значение |
|---------|--------------|----------------|---------------|
| Black   | 0            | 0              | 0             |
| Blue    | 0            | 0              | 255           |
| Green   | 0            | 255            | 0             |
| Cyan    | 0            | 255            | 255           |
| Red     | 255          | 0              | 0             |
| Magenta | 255          | 0              | 255           |
| Yellow  | 255          | 255            | 0             |
| White   | 255          | 255            | 255           |

Значения RGB-цветов, возвращаемых функцией, совместимы с используемыми в операционной системе Macintosh. Они могут использоваться в контексте

Microsoft-приложения для Macintosh, но не могут применяться, когда взаимодействующие цвета меняются непосредственно для операционной системы Macintosh.

## Right

Возвращает строку [тип **Variant (String)**], содержащую определенное количество символов правой части указанной строки.

### Синтаксис

**Right** (*string*, *length*)

Функция **Right** принимает следующие именованные аргументы:

- string*      Обязательный. Строковое выражение, из правой части которого возвращаются символы. Если *string* содержит **Null**, функция возвращает **Null**.
- length*      Обязательный; тип **Variant (Long)**. Численное выражение, указывающее число возвращаемых символов. Если этот аргумент равен значению 0, функция возвращает строку нулевой длины (""). Если *length* содержит значение, равное или большее, чем длина строки, возвращается вся строка.

Для определения длины строки (аргумента *string*) следует использовать функцию **Len**.

### Примеры

В этом примере используется функции **Right** для возвращения указанного числа символов от правого конца строки.

```
Dim StringAny, StrRez
StringAny = "Hello World"
StrRez = Right (StringAny, 1)      ' возвращает "d"
StrRez = Right (StringAny, 6)      ' возвращает " World"
StrRez = Right (StringAny, 20)     ' возвращает "Hello World"
```

## Rnd

Возвращает значение (тип **Single**), содержащее случайное число.

### Синтаксис

**Rnd** [ (*number*) ]

Необязательный аргумент (тип **Single**) *number* — любое допустимое числовое выражение. Возвращаемые значения функции **Rnd**:

| если <i>number</i> | функция <b>Rnd</b> возвращает                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------|
| меньше нуля        | одно и то же число каждый раз, используя <i>number</i> для инициализации генератора случайных чисел |
| больше нуля        | следующее случайное число в последовательности случайных чисел                                      |
| равно нулю         | самое последнее сгенерированное число                                                               |
| не используется    | следующее случайное число в последовательности случайных чисел                                      |

Функция **Rnd** возвращает значение, меньшее 1, большее или равное 0. Значение аргумента *number* определяет способ генерации функцией случайного числа:

для любого заданного числа-инициализатора последовательности случайных чисел (seed) генерируется одна и та же последовательность, поскольку каждый последующий вызов функции **Rnd** использует предыдущее число как инициализатор для генерации следующего случайного числа. Перед вызовом функции **Rnd** следует использовать оператор **Randomize** без аргументов для инициализации генератора случайных чисел, использующего системное время в качестве числа-инициализатора.

Иногда бывает необходимо получать последовательность случайных чисел в некотором числовом диапазоне. Для этого можно использовать следующую формулу:

```
Int((upperbound - lowerbound + 1) * Rnd + lowerbound)
```

Здесь *upperbound* — наибольшее число диапазона, *lowerbound* — наименьшее число диапазона.

### Примеры

```
Dim MyValue, MyString As String
Randomize
MyString = ""
For I = 0 To 9
    MyValue = Int((6 * Rnd) + 1)
    MyString = MyString & " " & MyValue
Next
```

Приведенный код генерирует строку из случайных чисел в диапазоне от 1 до 6. Результатом работы этого кода могут быть, например, следующие строки:

```
"1 5 2 6 5 5 5 5 4 1"; "5 6 1 3 5 4 6 1 4 3"
```

---

### Round

---

Возвращает число, округленное до определенного количества десятичных знаков.

#### Синтаксис

```
Round(expression [,numdecimalplaces])
```

Функция **Round** принимает следующие аргументы:

|                         |                                                                                                                                                                                                          |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>expression</i>       | Обязательный. Численное выражение, которое округляется.                                                                                                                                                  |
| <i>numdecimalplaces</i> | Необязательный. Число, указывающее, до скольких десятичных знаков (справа от десятичной точки) следует выполнять округление. Если этот аргумент опускается, функция <b>Round</b> возвращает целое число. |

---

### RTrim

---

Функция возвращает значение [тип **Variant (String)**], содержащее копию указанной строки без конечных (trailing) пробелов.

#### Синтаксис

```
RTrim(string)
```

Обязательный аргумент *string* — любое допустимое строковое выражение. Если аргумент содержит **Null**, функция возвращает **Null**.

---

**Second**


---

Возвращает значение [тип **Variant (Integer)**], определяющее целое число между 0 и 59, включительно, представляющее секунды в минуте.

**Синтаксис**

**Second** (*time*)

Обязательный аргумент *time* — любое **Variant**-, численное, строковое выражение или любая их комбинация, которое представляет время. Если аргумент *time* содержит **Null**, функция также возвращает **Null**.

**Пример**

```
Time1 = #3:45:20 PM#
MySecond = Second(Time1) ' переменная получает значение 20
```

---

**Seek**

Возвращает значение (тип **Long**), определяющее текущую позицию чтения/записи в файле, открытом с использованием оператора **Open**.

**Синтаксис**

**Seek** (*filenumber*)

Обязательный аргумент *filenumber* — допустимый файловый номер (тип **Integer**).

Функция **Seek** возвращает значения между 1 и 2147483647 (что эквивалентно  $2^{31} - 1$ ), включительно.

В следующей таблице описаны возвращаемые значения для каждого режима файлового доступа.

| режим                                | возвращаемое значение                                                                                                          |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <b>Random</b>                        | Номер следующей записи (record) для чтения или записи.                                                                         |
| <b>Binary, Output, Append, Input</b> | Позиция байта, для которой имеет место следующая операция. Первый байт в файле имеет позицию 1, второй байт — позицию 2 и т.д. |

---

**Sgn**

Возвращает значение [тип **Variant (Integer)**], указывающее знак числа.

**Синтаксис**

**Sgn** (*number*)

Обязательный аргумент *number* может быть любым допустимым численным выражением. Возвращаемые значения функции **Sgn**:

| если <i>number</i> | функция возвращает |
|--------------------|--------------------|
| больше нуля        | 1                  |
| равно нулю         | 0                  |
| меньше нуля        | -1                 |

---

Обратите внимание: знак аргумента определяет знак возвращаемого значения функции.

## Shell

Запускает исполняемую программу и возвращает значение [тип **Variant (Double)**], представляющее ID-задачи, если запуск выполнен успешно, или нуль — в противном случае.

### Синтаксис

**Shell** (*pathname* [, *windowstyle*])

Функция **Shell** принимает следующие именованные аргументы:

- pathname* Обязательный; тип **Variant (String)**. Наименование (имя) программы, которую следует запустить, и необходимые аргументы и ключи; можно включать каталог или папку и драйвер диска. На Macintosh можно использовать функцию **MacID** для определения сигнатуры приложения вместо его имени. Например, так на Macintosh можно с помощью сигнатуры запустить Microsoft Word: `Shell MacID("MSWD")`
- windowstyle* Необязательный; тип **Variant (Integer)**; соответствует стилю окна, в котором запускается приложение.
- Если аргумент *windowstyle* опускается, программа «стартует» минимизированной (если такое состояние в ней предусмотрено) с фокусом. На Macintosh (System 7.0 и более поздняя) аргумент *windowstyle* определяет, получает ли запускаемое приложение фокус.

Аргумент *windowstyle* может иметь следующие значения:

| константа                 | значение | описание                                                                                                          |
|---------------------------|----------|-------------------------------------------------------------------------------------------------------------------|
| <b>vbHide</b>             | 0        | Окно скрыто, фокус передается на скрытое окно. На платформе Macintosh эта константа не используется.              |
| <b>vbNormalFocus</b>      | 1        | Окно имеет фокус и восстановлено до оригинальных размеров и положения на экране.                                  |
| <b>vbMinimizedFocus</b>   | 2        | Окно отображается как значок с фокусом.                                                                           |
| <b>vbMaximizedFocus</b>   | 3        | Окно максимизировано и с фокусом.                                                                                 |
| <b>vbNormalNoFocus</b>    | 4        | Окно восстановлено к его предыдущим размерам и положению на экране. Текущее активное окно остается быть активным. |
| <b>vbMinimizedNoFocus</b> | 6        | Окно отображается как значок. Текущее активное окно остается активным.                                            |

Если функция **Shell** успешно запускает исполняемый файл, она возвращает ID запущенной программы — уникальный номер, который идентифицирует выполняемую программу. Если функция **Shell** не может запустить программу, возникает ошибка.

По умолчанию функция **Shell** запускает другие программы асинхронно. Это означает, что программа, запущенная этой функцией, не должна завершаться перед той, из которой была выполнена функция **Shell**.

### Пример

Так, например, можно запустить Windows-программу «Лазерный проигрыватель» (переменная `ReVal` может получить при этом значение, например, `-532327`):

```
RetVal = Shell("C:\WINDOWS\cdplayer.EXE", vbNormalFocus)
```

---

## Sin

---

Возвращает значение (тип **Double**) синуса угла (в диапазоне от -1 до 1).

### Синтаксис

**Sin** (*number*)

Обязательный аргумент *number* — **Double**- или любое численное выражение, которое представляет значение угла в радианах.

---

## SLN

---

Возвращает значение типа **Double**, обозначающее величину непосредственной амортизации имущества за один период.

### Синтаксис

**SLN** (*cost*, *salvage*, *life*)

Именованные аргументы функции **SLN**:

*cost*            Обязательный. Тип **Double**, обозначает начальную стоимость имущества.

*salvage*        Обязательный. Тип **Double**. Это — стоимость имущества в конце периода амортизации.

*life*            Обязательный. Тип **Double**. Это — продолжительность периода использования (амортизации).

Период снижения стоимости (depreciation period ) должен выражаться в одних и тех же единицах, что и аргумент *life*. Все аргументы должны быть положительными числами.

---

## Space

---

Возвращает строку [тип **Variant (String)**], содержащую указанное количество пробелов.

### Синтаксис

**Space** (*number*)

Обязательный аргумент *number* — это необходимое количество пробелов в возвращаемой строке. Функция **Space** полезна при форматировании выходных данных и очистке строк фиксированной длины.

### Пример

```
MsgBox = "ПрЮвет" & Space(5) & "зайцу!"
```

---

## Spc

---

Используется с оператором **Print #** или методом **Print** (в окне **Immediate**) для позиционирования выходных данных.

### Синтаксис

**Spc** (*n*)

Обязательный аргумент *n* — это количество пробелов для вставки перед следующим отображаемым на экране или печатаемым выражением в списке.

Если  $n$  меньше, чем ширина выходной строки, информация выводится сразу за указанным количеством пробелов. Если  $n$  больше, чем ширина выходной строки, функция **Spc** вычисляет следующую позицию для печати, используя формулу:

*текущая\_позиция + (n Mod ширина)*

Например, если текущая позиция для печати — 25, ширина выходной строки — 75, и вы определяете **Spc(90)**, выдача данных начнется с позиции (текущая позиция + остаток от 90/75). Если разность между текущей позицией и шириной выходной строки меньше, чем  $n$  (или  $n \bmod \text{ширина}$ ), функция **Spc** выполняет переход к началу следующей строки и генерирует число пробелов, равное  $n - (\text{ширина} - \text{текущая\_позиция})$ .

При использовании метода с пропорциональным шрифтом ширина символов пробела с использованием функции **Spc** — это всегда среднее значение ширины всех символов для выбранного шрифта. Однако нет никакой связи между числом печатаемых символов и количеством колонок фиксированной ширины, которые эти символы занимают. Например, символ W занимает больше места, чем колонка фиксированной ширины, а символ l — меньше.

## Split

Возвращает одномерный массив (индексируемый с нуля), содержащий определенное число подстрок.

### Синтаксис

**Split**(*expression*[, *delimiter*[, *limit*[, *compare*]])

Функция **Split** принимает следующие именованные аргументы:

|                   |                                                                                                                                                                                                                                                                                                                               |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>expression</i> | Обязательный. Строковое выражение, содержащее подстроки и ограничители. Если <i>expression</i> — строка нулевой длины (""), функция <b>Split</b> возвращает пустой массив, не содержащий ни элементов, ни данных.                                                                                                             |
| <i>delimiter</i>  | Необязательный. Строковый символ, используемый в качестве разделителя подстрок. Если этот аргумент не указан, в качестве разделителя используется символ пробела. Если <i>delimiter</i> — строка нулевой длины, функция возвращает массив из одного элемента, содержащего всю строку, заданную аргументом <i>expression</i> . |
| <i>limit</i>      | Необязательный. Число возвращаемых подстрок; -1 указывает на необходимость вернуть все подстроки.                                                                                                                                                                                                                             |
| <i>compare</i>    | Необязательный. Числовое значение, определяющее тип сравнения при оценке подстрок. (См. следующую таблицу.)                                                                                                                                                                                                                   |

Аргумент *compare* может иметь следующие значения:

| константа                 | значение | описание                                                                                |
|---------------------------|----------|-----------------------------------------------------------------------------------------|
| <b>vbUseCompareOption</b> | -1       | Выполнять сравнение с использованием настройки оператора <b>Option Compare</b> .        |
| <b>vbBinaryCompare</b>    | 0        | Выполнять бинарное сравнение.                                                           |
| <b>vbTextCompare</b>      | 1        | Выполнять текстовое сравнение.                                                          |
| <b>vbDatabaseCompare</b>  | 2        | Только для Microsoft Access. Выполнять сравнение, основанное на информации базы данных. |

---

**Sqr**

---

Возвращает значение (тип **Double**), определяющее корень квадратный из числа-аргумента.

**Синтаксис**

**Sqr** (*number*)

Обязательный аргумент *number* — **Double**- или любое допустимое числовое выражение, большее или равное нулю.

**Пример**

```
Sqr3 = Sqr(0)      'возвращает 0
Sqr1 = Sqr(9)      'возвращает 3
Sqr2 = Sqr(23)     'возвращает 4.79583152331272
Sqr4 = Sqr(-9)     'генерируется ошибка времени исполнения
```

---

**Str**

---

Возвращает строку [тип **Variant (String)**], представляющую число.

**Синтаксис**

**Str** (*number*)

Обязательный аргумент *number* — значение (тип **Long**), содержащее любое допустимое числовое выражение. При преобразовании числа в строку всегда резервируется ведущий пробел для знака числа. Если *number* — положительное число, возвращается строка с ведущим пробелом.

Для преобразования числовых значений в форматированные строки (в даты, время, валюты и т.д.) следует использовать функцию **Format**, которая в отличие от **Str** не включает в строку ведущий пробел для знака числа.

**Примеры**

```
MyString = Str(975)      'возвращает строку " 975"
MyString = Str(-449.63)  'возвращает строку "-449.63"
MyString = Str(479.011)  'возвращает строку " 479.011"
```

---

**StrComp**

---

Возвращает значение [тип **Variant (Integer)**] с результатом строкового сравнения.

**Синтаксис**

**StrComp** (*string1*, *string2* [, *compare*])

Функция **StrComp** принимает следующие именованные аргументы:

|                |                                                                                                                                                                                                                           |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>string1</i> | Обязательный. Любое допустимое строковое выражение.                                                                                                                                                                       |
| <i>string2</i> | Обязательный. Любое допустимое строковое выражение.                                                                                                                                                                       |
| <i>compare</i> | Необязательный. Определяет тип строкового сравнения. Если <i>compare</i> — <b>Null</b> , генерируется ошибка. Если <i>compare</i> не указывается, тип сравнения определяется настройкой оператора <b>Option Compare</b> . |



Аргумент *compare* может принимать следующие значения:

| константа                 | значение | описание                                                                                |
|---------------------------|----------|-----------------------------------------------------------------------------------------|
| <b>vbUseCompareOption</b> | -1       | Выполнять сравнение с использованием настройки оператора <b>Option Compare</b> .        |
| <b>vbBinaryCompare</b>    | 0        | Выполнять бинарное сравнение.                                                           |
| <b>vbTextCompare</b>      | 1        | Выполнять текстовое сравнение.                                                          |
| <b>vbDatabaseCompare</b>  | 2        | Только для Microsoft Access. Выполнять сравнение, основанное на информации базы данных. |

Функция **StrComp** возвращает следующие значения:

| если                                            | StrComp возвращает |
|-------------------------------------------------|--------------------|
| <i>string1</i> меньше, чем <i>string2</i>       | -1                 |
| <i>string1</i> эквивалентна <i>string2</i>      | 0                  |
| <i>string1</i> больше, чем <i>string2</i>       | 1                  |
| <i>string1</i> или <i>string2</i> — <b>Null</b> | <b>Null</b>        |

### Примеры

```
Dim Str1, Str2, RezComp
Str1 = "BCDF"
Str2 = "bcdF"
RezComp = StrComp(Str2, Str1)           ' 1
RezComp = StrComp(Str1, Str2, 1)       ' 0
RezComp = StrComp(Str1, Str2, 0)       '-1
```

### StrConv

Возвращает значение [тип **Variant (String)**], конвертированное указанным способом.

#### Синтаксис

**StrConv**(*string*, *conversion*, *LCID*)

Функция **StrConv** принимает следующие именованные аргументы:

|                   |                                                                                                 |
|-------------------|-------------------------------------------------------------------------------------------------|
| <i>string</i>     | Обязательный. Конвертируемое строковое выражение.                                               |
| <i>conversion</i> | Обязательный. Тип <b>Integer</b> . Суммарное значение, определяющее тип преобразования.         |
| <i>LCID</i>       | Необязательный. <i>LocaleID</i> , если отличается от системного <i>LocaleID</i> (по умолчанию). |

Аргумент *conversion* может принимать следующие значения:

| константа          | значение | описание                                           |
|--------------------|----------|----------------------------------------------------|
| <b>vbUpperCase</b> | 1        | Конвертировать строку в символы верхнего регистра. |
| <b>vbLowerCase</b> | 2        | Конвертировать строку в символы нижнего регистра.  |

|                      |     |                                                                                                      |
|----------------------|-----|------------------------------------------------------------------------------------------------------|
| <b>vbProperCase</b>  | 3   | Конвертирует первый символ каждого слова строки в символ верхнего регистра.                          |
| <b>vbUnicode</b>     | 64  | Конвертирует строку в Unicode, используя кодировку страницы системы по умолчанию (Не для Macintosh). |
| <b>vbFromUnicode</b> | 128 | Конвертирует строку из Unicode в кодировку страницы системы по умолчанию (Не для Macintosh).         |

## StrReverse

Возвращает строку с обратным порядком символов по сравнению с оригинальной строкой.

### Синтаксис

**StrReverse** (*expression*)

Аргумент *expression* — строка, символы которой должны изменить порядок следования. Если *expression* является строкой нулевой длины (""), функция также возвращает строку нулевой длины. Если *expression* — **Null**, возникает ошибка.

## String

Возвращает строку [тип **Variant (String)**], содержащую определенной длины строку повторяющихся символов.

### Синтаксис

**String** (*number*, *character*)

Функция **String** принимает следующие именованные аргументы:

- number*                      Обязательный; тип **Long**. Длина возвращаемой строки. Если *number* содержит **Null**, функция также возвращает **Null**.
- character*                    Обязательный; тип **Variant**. Код символа или строковое выражение, первый символ которого используется для построения возвращаемой строки. Если *character* содержит **Null**, функция возвращает **Null**.

Если *character* содержит код, больший чем 255, функция **String** предварительно преобразует это значение в следующей формуле:

*character* **Mod** 256

### Примеры

```
String1 = String(5, "*")           ' возвращает "*****"
String2 = String(5, 42)            ' возвращает "*****"
String3 = String(10, "A")          ' возвращает "AAAAAAAAAA"
String4 = String(5, 299)           ' возвращает "+++++"

```

## Switch

Оценивает список выражений и возвращает значение (тип **Variant**) или выражение, связанное с первым выражением в списке, равным значению **True**.

### Синтаксис

**Switch** (*expr-1*, *value-1* [, *expr-2*, *value-2* ... [, *expr-n*, *value-n*]])

Функция **Switch** принимает следующие аргументы:

- expr*      Обязательный. **Variant**-выражение, которое необходимо оценить.
- value*      Обязательный. Значение или выражение, которое возвращается, если соответствующее выражение равно значению **True**.

Список аргументов функции **Switch** содержит пары выражений и значений. Выражения оцениваются слева направо, и значение, соответствующее первому выражению, которое оценивается как **True**, возвращается.

Функция **Switch** возвращает **Null**, если ни одно из выражений не равно значению **True** или если первое из выражений, равное значению **True**, имеет соответствующее ему значение **Null**.

---

## **SYD**

Возвращает значение годовой амортизации активов за определенный период.

### **Синтаксис**

**SYD**(*cost*, *salvage*, *life*, *period*)

Функция **SYD** имеет следующие (все обязательные, тип **Double**) именованные аргументы:

- cost*      Начальная стоимость имущества.
- salvage*    Остаточная стоимость имущества в конце периода амортизации (*useful life*).
- life*      Время эксплуатации (период амортизации).
- period*    Период, за который вычисляется амортизация.

Аргументы *life* и *period* должны выражаться в одних и тех же единицах, например, если *life* задается в месяцах, то и *period* также должен быть задан в месяцах. Все аргументы должны быть положительными числами.

---

## **Tab**

Используется с оператором (statement) **Print #** или методом **Print** для позиционирования выходных данных.

### **Синтаксис**

**Tab** [ (*n*) ]

Необязательный аргумент *n* — число колонок, на которые необходимо сместить начало вывода информации на экране или принтере. Если аргумент отсутствует, функция **Tab** перемещает курсор вставки в начало следующей зоны печати. Это позволяет использовать функцию **Tab** вместо запятой там, где запятая применяется как десятичный разделитель.

Если текущая позиция печати на текущей строке больше, чем *n*, функция **Tab** пропускает *n*-ую колонку на следующей строке вывода. Если *n* меньше, чем 1, функция **Tab** продвигает позицию печати к первой колонке. Если значение аргумента *n* больше, чем ширина выходной строки, следующая позиция печати вычисляется по формуле:

$n \bmod width$

---

**Tan**

---

Возвращает значение (тип **Double**) тангенса угла.

**Синтаксис**

**Tan** (*number*)

Обязательный аргумент *number* — **Double**- или любое допустимое числовое выражение, которое представляет значение угла в радианах.

**Примеры**

```
Angle = 1.3                'значение угла в радианах
Cotangent = 1 / Tan(Angle) 'значение котангенса
```

---

**Time**

---

Возвращает значение [тип **Variant (Date)**] системного времени.

**Синтаксис**

**Time**

**Пример**

```
TimeSystem = Time 'возвращает системное время
```

---

**Timer**

---

Возвращает значение (тип **Single**), представляющее число секунд, прошедших с начала суток.

**Синтаксис**

**Timer**

В Microsoft Windows функция **Timer** возвращает дробные части секунд.

---

**TimeSerial**

---

Возвращает значение [тип **Variant (Date)**], содержащее время для указанных часа, минут и секунд.

**Синтаксис**

**TimeSerial** (*hour*, *minute*, *second*)

Функция **TimeSerial** принимает следующие именованные аргументы:

|               |                                                                                                                                  |
|---------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>hour</i>   | Обязательный; тип <b>Variant (Integer)</b> . Число между 0 (12:00 А.М.) и 23 (11:00 Р.М.), включительно, или числовое выражение. |
| <i>minute</i> | Обязательный; тип <b>Variant (Integer)</b> . Любое числовое выражение в диапазоне от 0 до 59, включительно.                      |
| <i>second</i> | Обязательный; тип <b>Variant (Integer)</b> . Любое числовое выражение в диапазоне от 0 до 59, включительно.                      |

**Примеры**

```
Time1 = TimeSerial(1, 30, 17) 'значение для Time1 1:30:17
```

---

**TimeValue**


---

Возвращает значение [тип **Variant (Date)**], содержащее время.

**Синтаксис**

**TimeValue** (*time*)

Обязательный аргумент *time* — это обычно строковое выражение, представляющее время от 0:00:00 (12:00:00 А.М.) до 23:59:59 (11:59:59 Р.М.), включительно. Если *time* содержит **Null**, функция возвращает **Null**.

**Пример**

Msg = "Московское время: " & TimeValue("5:30:17 PM")

В этом примере переменная **Msg** получает значение 'Московское время: 17:30:17'.

---

**Trim**


---

Функция возвращает значение [тип **Variant (String)**], содержащее копию указанной строки без ведущих и конечных пробелов.

**Синтаксис**

**Trim** (*string*)

Обязательный аргумент *string* — любое допустимое строковое выражение. Если аргумент содержит **Null**, функция возвращает **Null**.

---

**TypeName**


---

Возвращает строку (тип **String**), которая содержит информацию о типе переменной.

**Синтаксис**

**TypeName** (*varname*)

Обязательный аргумент *varname* — имя переменной. Строка, содержащая функцию **TypeName**, может иметь следующее значение:

| возвращаемая строка | переменная                                   |
|---------------------|----------------------------------------------|
| <b>Object type</b>  | Объект с типом <i>objecttype</i>             |
| <b>Byte</b>         | Байтовое значение                            |
| <b>Integer</b>      | Целое                                        |
| <b>Long</b>         | Длинное целое                                |
| <b>Single</b>       | Число с плавающей запятой одинарной точности |
| <b>Double</b>       | Число с плавающей запятой двойной точности   |
| <b>Currency</b>     | Значение валюты                              |
| <b>Decimal</b>      | Десятичное значение                          |
| <b>Date</b>         | Значение даты                                |
| <b>String</b>       | Строка                                       |
| <b>Boolean</b>      | Логическое значение                          |

| возвращаемая строка | переменная                                           |
|---------------------|------------------------------------------------------|
| <b>Error</b>        | Значение ошибки                                      |
| <b>Empty</b>        | Неинициализированная переменная                      |
| <b>Null</b>         | Нет допустимых данных                                |
| <b>Object</b>       | Объект                                               |
| <b>Unknown</b>      | Объект с неизвестным типом                           |
| <b>Nothing</b>      | Объектная переменная, которая не ссылается на объект |

Если *varname* — массив, функция возвращает строку с типом элементов массива и двумя круглыми скобками.

### Примеры

```
Dim NullVar, VarType
Dim StrVar As String
Dim IntVar As Integer
Dim CurVar As Currency
Dim IntArray (1 To 25) As Integer
NullVar = Null
VarType = TypeName(StrVar)      ' возвращает "String"
VarType = TypeName(IntVar)      ' возвращает "Integer"
VarType = TypeName(CurVar)      ' возвращает "Currency"
VarType = TypeName(IntArray)    ' возвращает "Integer()"
VarType = TypeName(NullVar)     ' возвращает "Null"
```

## UBound

Возвращает значение (тип **Long**), содержащее наибольшее значение индекса для указанной размерности массива.

### Синтаксис

**UBound**(*arrayname*[, *dimension*])

Функция **UBound** имеет следующие аргументы:

- arrayname*      Обязательный. Имя переменной-массива.
- dimension*      Необязательный; Тип **Variant (Long)**. Целое число, указывающее размерность, для которой определяется верхняя граница индекса. Для первой размерности используется число 1, для второй — 2 и т.д. Если аргумент не указывается, по умолчанию используется значение 1.

Функция **UBound** обычно используется с функцией **LBound** для определения размера массива, поскольку **LBound** позволяет найти наименьшее значение индекса указанной размерности.

### Примеры

```
Dim IntArray(1 To 12, 5 To 17, 10 To 20)
Dim DouArray(13)
IntUpper = UBound(DouArray)      'возвращает 13
IntUpper = UBound(IntArray, 1)   'возвращает 12
IntUpper = UBound(IntArray, 2)   'возвращает 17
```

## UCase

Возвращает значение [тип **Variant (String)**], содержащее заданную строку, символы которой преобразованы к верхнему регистру.

### Синтаксис

**UCase** (*string*)

Обязательный аргумент *string* — любое допустимое строковое выражение. Если *string* содержит **Null**, функция возвращает **Null**.

### Примеры

```
Dim LowerCase, UpperCase
StringLowerCase = "Hello World!"
'Следующий оператор возвращает строку 'HELLO WORLD! '
StringUpperCase = UCase(StringowerCase)
```

## Val

Возвращает число, содержащееся в строке.

### Синтаксис

**Val** (*string*)

Обязательный аргумент *string* — любое допустимое строковое выражение.

Функция **Val** прекращает чтение строки на первом символе, который не распознается как часть символа. Такие символы, как символ доллара, запятая, не распознаются. Однако функция распознает символы основания системы счисления &O (для восьмеричной) и &H (для шестнадцатеричной). Пробелы, знаки табуляции отсекаются.

### Примеры

```
IntValue = Val("9459")           'возвращается 9459
IntValue = Val(" 9 45 9")        'возвращается 9459
IntValue = Val("124 and 537")    'возвращается 24
```

## VarType

Возвращает целое, указывающее подтип переменной.

### Синтаксис

**VarType** (*varname*)

Обязательный аргумент *varname* — значение (тип **Variant**), содержащее любую переменную, исключая переменные типов, определенных пользователем.

Возвращаемые значения функции **VarType**:

| константа        | значение | описание                              |
|------------------|----------|---------------------------------------|
| <b>vbEmpty</b>   | 0        | <b>Empty</b> (не инициализирована)    |
| <b>vbNull</b>    | 1        | <b>Null</b> (недопустимые данные)     |
| <b>vbInteger</b> | 2        | Целое                                 |
| <b>vbLong</b>    | 3        | Длинное целое                         |
| <b>vbSingle</b>  | 4        | С плавающей точкой одинарной точности |

| константа                | значение | описание                                                 |
|--------------------------|----------|----------------------------------------------------------|
| <b>vbDouble</b>          | 5        | С плавающей точкой двойной точности                      |
| <b>vbCurrency</b>        | 6        | Значение валюты                                          |
| <b>vbDate</b>            | 7        | Значение даты                                            |
| <b>vbString</b>          | 8        | Строка                                                   |
| <b>vbObject</b>          | 9        | Объект                                                   |
| <b>vbError</b>           | 10       | Значение ошибки                                          |
| <b>vbBoolean</b>         | 11       | Логическое значение                                      |
| <b>vbVariant</b>         | 12       | <b>Variant</b> (используется только с variant-массивами) |
| <b>vbDataObject</b>      | 13       | Объект для доступа к данным                              |
| <b>vbDecimal</b>         | 14       | Десятичное значение                                      |
| <b>vbByte</b>            | 17       | Байтовое значение                                        |
| <b>vbUserDefinedType</b> | 36       | <b>Variant</b> -данные пользовательского типа            |
| <b>vbArray</b>           | 8192     | Массив                                                   |

Функция **VarType** никогда не возвращает значение для **vbArray** «само по себе». Оно всегда добавляется к некоторому другому значению для указания на массив определенного типа. Константа **vbVariant** возвращается только в конъюнкции с **vbArray** для индикации того, что аргумент функции **VarType** является массивом типа **Variant**.

### Примеры

```

IntVar = 159
DateVar = #2/11/73#
StrVar = "Random number generator"
MyRez = VarType(IntVar)      'возвращает 2 (vbInteger)
MyRez = VarType(DateVar)    'возвращает 7 (vbDate)
MyRez = VarType(StrVar)     'возвращает 8 (vbString)

```

### Weekday

Возвращает значение типа **Variant (Integer)**, являющееся частью аргумента (типа **Date**) и содержащее день недели.

#### Синтаксис

**Weekday** (*date*, [*firstdayofweek*])

Функция **Weekday** принимает следующие именованные аргументы:

- |                       |                                                                                                                                                                                       |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>date</i>           | Обязательный. Тип <b>Variant</b> . Численное, строковое выражение или их комбинация, представляющая дату. Если <b>date</b> содержит <b>Null</b> , функция возвращает <b>Null</b> .    |
| <i>firstdayofweek</i> | Необязательный. Константа, которая задает первый день недели; если не указан, по умолчанию используется <b>vbSunday</b> или <b>vbMonday</b> (в зависимости от национальных настроек). |



В качестве аргумента *firstdayofweek* следует использовать следующие константы:

| константа          | значение | назначение                      |
|--------------------|----------|---------------------------------|
| <b>vbUseSystem</b> | 0        | Использовать настройки NLS API. |
| <b>vbSunday</b>    | 1        | Sunday                          |
| <b>vbMonday</b>    | 2        | Monday                          |
| <b>vbTuesday</b>   | 3        | Tuesday                         |
| <b>vbWednesday</b> | 4        | Wednesday                       |
| <b>vbThursday</b>  | 5        | Thursday                        |
| <b>vbFriday</b>    | 6        | Friday                          |
| <b>vbSaturday</b>  | 7        | Saturday                        |

Функция **Weekday** возвращает следующие значения:

| константа          | значение | назначение |
|--------------------|----------|------------|
| <b>vbSunday</b>    | 1        | Sunday     |
| <b>vbMonday</b>    | 2        | Monday     |
| <b>vbTuesday</b>   | 3        | Tuesday    |
| <b>vbWednesday</b> | 4        | Wednesday  |
| <b>vbThursday</b>  | 5        | Thursday   |
| <b>vbFriday</b>    | 6        | Friday     |
| <b>vbSaturday</b>  | 7        | Saturday   |

Результат функции зависит от значения свойства *Calendar*.

**Пример**

```
My_Date = #February 13, 1969#  
My_Week_Day = Weekday(My_Date) ' My_Week_Day содержит 5
```

**WeekdayName**

Возвращает строку с наименованием дня недели, используя номер дня недели.

**Синтаксис**

```
WeekdayName (weekday, abbreviate, firstdayofweek)
```

Функция **WeekdayName** принимает следующие аргументы:

- weekday*            Обязательный. Числовое обозначение дня недели. Численное значение каждого дня недели зависит от значения *firstdayofweek*.
- abbreviate*        Необязательный. Тип **Boolean**. Указывает на необходимость использовать сокращенное название дня недели. По умолчанию — **False**, что означает использовать полное название.
- firstdayofweek*    Необязательный. Численное значение, определяющее первый день недели.

Аргумент *firstdayofweek* может принимать следующие значения:

| константа          | значение | назначение                      |
|--------------------|----------|---------------------------------|
| <b>vbUseSystem</b> | 0        | Использовать настройки NLS API. |
| <b>vbSunday</b>    | 1        | Sunday                          |
| <b>vbMonday</b>    | 2        | Monday                          |
| <b>vbTuesday</b>   | 3        | Tuesday                         |
| <b>vbWednesday</b> | 4        | Wednesday                       |
| <b>vbThursday</b>  | 5        | Thursday                        |
| <b>vbFriday</b>    | 6        | Friday                          |
| <b>vbSaturday</b>  | 7        | Saturday                        |

**Пример**

```
MsgBox WeekdayName(2)           'в диалоговом окне — "Вторник"
MsgBox WeekdayName(2, True)     'в диалоговом окне — "Вт"
MsgBox WeekdayName(2, True, vbSunday) 'в диалоговом окне — "Пн"
```

**Year**

Возвращает значение типа **Variant (Integer)**, являющееся частью аргумента (типа **Date**) и содержащее год.

**Синтаксис**

**Year** (*date*)

Обязательный аргумент *date* — любое выражение (**Variant**, численное, строковое или любая комбинация), представляющее дату. Если аргумент *date* содержит **Null**, функция возвратит **Null**. Результат функции зависит от значения свойства **Calendar**.

**Пример**

```
Dim My_Date, My_Year
My_Date = #February 11, 1960#
My_Year = Year(My_Date)           ' My_Year содержит 1960
```

# Предметный указатель

!

&, 139

:=, 148

## A

ABook, 250

Abort, 215

Abs, 152

Accelerator, 373

Activate, 242, 364, 366, 400, 407, 427

ActiveCell, 238, 287

ActiveChart, 238

ActiveControl, 361

ActiveDocument, 240, 250, 256, 425

ActivePrinter, 240

ActiveSheet, 238, 250

ActiveWindow, 240

ActiveWorkbook, 250–253, 256

ActiveWorkbook.Path, 253

Add, 258, 399, 408, 420, 426

– Watch, 461

AddCallout, 262, 264–265

AddCanvas, 262

AddComment, 262, 266

AddConnector, 264, 266

AddControl, 375

AddCurve, 262, 264, 266

AddDiagram, 262, 264, 266

AddForm, 264

AddIns, 255, 257, 433

AddLabel, 262, 264, 266

AddLine, 262, 265–266

AddMediaObject, 266

AddOLEControl, 263

AddOLEObject, 263, 265, 267

AddPicture, 263, 265, 267

AddPlaceholder, 267

AddPolyline, 263, 265, 267

Address, 238, 243

AddShape, 267

AddShapes, 259

AddTable, 267

AddTextbox, 263, 265, 268

AddTextEffect, 264–265, 268

AddTitle, 268

AfterUpdate, 375

Alphabetic, 43

And, 134

Application, 191, 235–236, 250–251, 255, 259, 334

Application.ActiveSheet, 250

Application.InputBox, 205

Application.StatusBar, 238

Application.Workbooks, 250

Archive, 323–324

argument, 73, 76

argument list, 76

array, 109, 302

As, 184

Asc, 153

Assert, 468

Atn, 152

AttachedTemplate, 434

attribute, 323

Author, 400

Auto Data Tips, 76

– Indent, 75

– indenting, 75

– Quick Info, 73, 76

## B

BackColor, 361, 368–369, 373

BeforeUpdate, 375

body, 276

Bookmark, 236

Bookmarks, 47, 257

Boolean, 89, 107, 109, 116, 118

branch, 196

Break, 454, 456

Break mode, 455  
breakpoint, 457  
Buttons, 76, 213  
ByRef, 225  
Byte, 87, 109, 115  
ByVal, 189, 225, 319

## C

Call Stack, 455, 466  
Cancel, 373, 382  
Caption, 361, 373, 380  
CBool, 154  
CByte, 154  
CCur, 154  
CDate, 154  
CDBl, 154  
Cells, 238, 250, 413  
Change, 375, 385, 387  
character code, 129  
Characters, 257, 397, 435  
Chart, 235, 255  
ChartObjects, 255, 257  
Charts, 238, 257  
ChDir, 322, 347, 355  
ChDrive, 322, 348, 355  
CheckBox, 371  
Chr, 153, 165–166  
Cint, 154, 279  
Clear, 46, 246, 421  
ClearContents, 422  
ClearFormats, 422  
ClearNotes, 422  
Click, 364–365, 369, 375  
CLng, 154  
Close, 404, 431  
– and Return to ..., 46  
Code, 48  
– Window, 44, 62, 71–73  
Collapse, 444–445  
collection, 254  
ComboBox, 371, 389  
CommandButton, 371, 379  
comment, 64  
Comments, 252, 254, 257  
comparison operation, 89

compile error, 451–452  
compiler directive, 104  
compiling, 78  
Complete Word, 47  
concatenate, 89  
concatenation, 138  
Const, 106  
constant, 105  
container, 255  
Context, 462  
Continue, 80, 456  
control, 370  
Controls, 361  
ControlSource, 390  
ControlTipText, 373  
Copy, 46, 362, 409, 421, 447  
Cos, 152  
Count, 238, 240, 254–255  
Critical Warning message, 214  
CSng, 153–154  
CStr, 145, 154  
CurDir, 322, 346, 354  
Currency, 87–88, 90, 109, 115, 169  
current directory, 346  
– drive, 346  
– folder, 346  
Cut, 362, 421, 447–448  
CVar, 154  
Cycle, 361

## D

Date, 85, 107–109, 115–116, 124–125, 144, 155, 178, 181  
date expression, 115  
DateAdd, 156  
DateDiff, 156  
DatePart, 156  
DateSerial, 156  
DateValue, 156  
Day, 155, 178  
DblClick, 364, 369, 375  
Deactivate, 364, 366  
Debug, 54, 80, 456  
Debug.Print, 468  
Debugging, 50

declaration, 64  
Default, 373  
Definition, 48  
Delete, 410, 433, 448  
Description, 64, 470  
design time, 455  
Dialog, 342  
Dialogs, 255, 257, 341  
Dictionaries, 255  
Dim, 249, 305  
dimension (измерение), 305  
Dir, 322–323, 329–330  
directory, 321, 323–324  
Display, 341–342  
Do, 286–287  
– While, 289, 292  
Do...Loop Until, 296  
– While, 294  
docked, 43  
Document, 236, 397  
Documents, 257, 424  
Double, 87–88, 109, 115, 118, 124  
DrawingObjects, 255  
dynamic, 304

## E

Edit, 54  
element, 254  
Empty, 114  
Enable, 385  
Enabled, 362, 373, 387, 391  
End, 80, 216, 456  
– Function, 180, 182, 195, 469  
– Property, 469  
– Select, 209  
– Sub, 64, 72, 180, 195, 469  
– With, 254  
EndKey, 441  
Enter, 375  
Eqv, 137  
Erase, 317  
Err.Description, 476  
Err.Number, 476  
Error, 375  
event, 363

event procedures, 364  
executing, 23  
Exit, 216, 375  
– Design Mode, 454  
– Function, 216, 469  
– Property, 469  
– Sub, 217, 469  
Exp, 152  
Expand, 440  
explicit, 94  
Export File, 45  
expression, 114, 465

## F

False, 89, 109, 116  
Fields, 436  
File, 45  
– management, 321  
FileCopy, 322, 351, 355  
FileDateTime, 322, 355  
FileLen, 322, 356  
Find, 46  
– Next, 47  
Fix, 152  
Fixed, 169  
– iteration, 275  
Fixed-length, 98  
floating point number, 88  
– window, 43  
flow control, 196  
folder, 321  
Font, 235–236, 362, 389  
Font.Size, 301  
For Each...Next, 277, 283  
For...Next, 277, 301  
ForeColor, 362, 373  
Format, 153, 168, 341  
FormatCurrency, 173  
FormatDateTime, 174  
FormatNumber, 175  
FormatPercent, 176  
Formula, 238, 419  
Frame, 371  
Full Module View, 44  
function, 49, 112, 143, 180

function result, 114  
fuzzy search, 130

## G

General, 45  
– Date, 169  
– Number, 169  
GetAttr, 322, 325  
GetAttrOpenFilename, 322, 329, 334, 335, 337, 353, 355  
GetAttrSaveasFilename, 322, 334, 338, 355  
GoTo, 196, 211, 442

## H

Height, 268  
Help, 80  
Hex, 153  
Hidden, 323–324  
Hide, 362  
HomeKey, 441  
host-приложение, 54  
Hour, 155, 178

## I

identifier, 92  
If...Then, 196  
If...Then...Else, 196, 201  
If...Then...ElseIf, 201, 205  
Ignore, 215  
Image, 372  
Immediate, 467  
– Window, 455, 467  
Imp, 138  
implicit variable declaration, 93  
Import File, 45  
indefinite loop, 276  
Indent, 47  
index, 238, 303  
infinite loop, 276  
Information, 148  
Information message, 214  
InitialFilename, 355  
Initialize, 364–365, 368  
InputBox, 112, 150, 195, 213, 259  
InsertAfter, 445

InsertBefore, 445  
InsertParagraphAfter, 445–446  
InsertParagraphBefore, 445–446  
InStr, 157, 162, 252, 282  
InStrRev, 157  
Int, 152  
Integer, 87, 90, 107, 109, 115, 118, 126  
intrinsic constant, 110  
Is, 132, 210, 251  
IsMissing, 200  
IsObject, 249  
iteration, 275

## K

KeyDown, 376  
KeyPress, 376  
KeyUp, 376  
Kill, 322, 352

## L

Label, 371, 380, 385, 389  
Last Position, 48  
Layout, 376  
LBound, 315  
LCase, 157  
Left, 157, 163, 252, 268  
Len, 157, 160, 182  
LenTrim, 182  
Let, 119  
Like, 130  
line label, 211  
line-continuation character, 67  
List, 373  
– Constants, 47  
– Properties/Methods, 47  
ListBox, 371, 389  
literal constant, 105  
Load, 365  
Locals, 465  
Locals Window, 455  
Log, 152, 178  
logical expression, 116  
Long, 87, 90, 109, 115, 126  
– Date, 169  
– Time, 169

Loop, 289

LTrim, 157, 159

## M

Max, 259, 374

Medium Date, 169

– Time, 169

Members of object, 62

method, 191, 233

Mid, 157, 164, 189, 282

Min, 259, 374

Minute, 155, 178

MkDir, 322, 349

Module, 71, 255

Month, 155, 178

More Buttons, 54

MouseDown, 376

MouseMove, 377

MouseUp, 376

Move, 409–410

MoveStart, 439

MSFlexGrid, 394

MsgBox, 73–74, 148, 195, 213

multi-dimensional, 303

MultiPage, 372

## N

Name, 235, 238, 240, 252, 322, 341, 354, 374, 409

named argument, 147

– constant, 105

Names, 420

nesting, 203

Next, 277

Normal, 324

Not, 135

Now, 155, 181

Null, 126

– string, 107

Number, 470

NumberFormat, 420

numeric expression, 115

## O

Object, 48, 109, 132, 236, 248, 250

– Browser, 59

– expression, 116, 249

– List, 45

– model, 255

Object-oriented programming, OOP, 231

Oct, 154

Offset, 416

OLEObjects, 255

On Error, 469, 475

– – GoTo, 469

– – Resume Next, 476

On/Off, 170

On-the-fly, 93

Open, 329, 398, 425

operand, 115

Option Base, 304–305

Option Compare, 129–130

Option Explicit, 103, 452

Optional, 200

OptionButton, 371

Or, 135

Outdent, 47

## P

Pane, 441

Paragraph, 236

Paragraphs, 257, 397, 435

Parameter Info, 47

Parent, 256

parsing, 77

Paste, 46, 362, 423, 447–448

PasteSpecial, 423

Path, 239–240

Percent, 170

persistence, 103

predefined constant, 110

Print, 46, 81, 467

PrintForm, 362

Private, 305

procedure, 44, 71

– View, 44

procedure-level scope, 99

Project, 43

– Explorer, 43, 71, 73, 81

Project/Library, 62

Prompt, 76, 148

Properties Window, 43, 72, 361

Property, 49

– Get, 362

– Let, 362

Public, 305, 310

## Q

Query message, 214

Quick Info, 47, 76

Quick Watch, 455

## R

Range, 235–236, 240, 397, 411–412, 419, 436

real, 88

Real-Only, 323–324, 328

Record Macro, 64

ReDim, 304, 310

Redo, 46

References, 53

relational operator, 127

Remove item, 46

Repaint, 363

Replace, 47

ReplaceSelection, 444–445

required, 200

Reset, 455

Resize, 364, 369–370, 417

Resume, 471

– line, 473

– Next, 472

Retry, 215

RGB, 154, 369

Right, 157, 164

Rmdir, 322, 350

Rnd, 152

RowSource, 374, 390

RTrim, 157, 159

Run Sub/UserForm, 454

– time, 455

running, 23

runtime error, 79, 451

runtime-ошибка, 79, 451–452

## S

Save, 402, 428

– As, 329

– project, 45

SaveAs, 246, 403

SaveCopyAs, 402

Saved, 239–240, 430

Scientific, 170

scope, 99

ScreenUpdating, 401

ScrollBar, 372

Search, 61

Search Results, 61

Second, 156, 178

Sections, 257

Select, 418, 441

– All, 46

Select...Case, 207

Selected, 374

Selection, 239–240, 301, 441

Selection.Font, 299

Sentences, 435

Set, 250

SetAttr, 322, 328

SetRange, 439

Sgn, 152

ShapeRange, 259

Shapes, 259

Sheets, 239, 407

Short Date, 169

ShortTime, 169

Show, 363, 466

side effect, 224

Sin, 152

Single, 87–88, 109, 115, 118

Single-dimensional, 302

Single-stepping, 456

sizing handle, 378

snap to grid, 379

source code, 35

Space, 157

SpinButton, 372



SpinDown, 377  
SpinUp, 377  
Sqr, 145, 152  
Stack, 455  
Standard, 54, 71, 169  
statement, 119  
static, 304, 305  
StatusBar, 238–239, 241  
Step, 277  
– Into, 455, 456, 460, 465  
– Out, 455  
– Over, 455, 465  
Stop, 456, 458  
Str, 154  
StrComp, 158, 161, 189, 286  
StrConv, 158  
string, 89  
String, 89, 107, 109, 115–116, 144, 158  
– expression, 115  
Style, 236  
Styles, 257  
Sub, 49, 64, 72, 74, 180  
Subject, 400  
subprocedure, 71  
subscript, 303  
SUM, 259  
syntax, 77  
– error, 451  
– errors, 77  
System, 323–324

## T

tab order, 389, 390  
TabIndex, 374  
Table, 236  
TableOfContents, 236  
TableOfContents, 436  
TabStop, 374  
TabStrip, 372  
Tan, 152  
Template, 236, 397, 432  
Templates, 257, 432  
Terminate, 365–366  
TextBox, 371, 389, 391  
ThisDocument, 425, 432

ThisWorkbook, 239  
three-dimensional reference, 408  
Time, 155, 181  
Timer, 156  
TimeSerial, 156  
TimeValue, 156  
Title, 77, 148, 400  
– bar, 43  
To, 209  
– Breakpoint, 455, 458  
Toggle Folders, 43  
ToggleButton, 371  
Toolbox, 377  
Top, 268  
Trim, 158–159, 198  
True, 89, 109, 116  
True/False, 170  
truth table, 134  
Type, 239, 465  
Type-mismatch, 116  
– mismatch error, 80  
TypeName, 249  
TypeParagraph, 445–446  
TypeText, 444, 446

## U

UBound, 315  
UCase, 158  
UDF, 183  
unary minus, 123  
Underline, 412  
Undo, 46  
Unload, 365  
Until, 287  
UseForm, 54  
user-defined function, 180, 183  
UserForm, 359, 362, 365

## V

Val, 154  
Value, 239, 374, 419, 465  
variable, 78, 90  
Variant, 90, 96, 116, 125, 184  
VBA statement, 64  
vbAbort, 150

vbRetryIgnore, 149  
 vbAlias, 325  
 vbApplicationModal, 149  
 vbArchive, 325  
 VBA-проект, 43  
 vbCancel, 150  
 vbCr, 167  
 vbCritical, 149  
 vbCrLf, 167  
 vbDatabaseCompare, 161  
 vbDefaultButton1, 149, 215  
 vbDefaultButton2, 149, 215  
 vbDefaultButton3, 149, 216  
 vbDefaultButton4, 149, 216  
 vbDirectory, 325  
 vbExclamation, 149  
 vbGeneralDate, 175  
 vbHidden, 325  
 vbIgnore, 150  
 vbInformation, 149  
 vbLf, 167  
 vbLongDate, 175  
 vbLongTime, 175  
 vbMsgBoxHelpButton, 149  
 vbNewLine, 167  
 vbNo, 150  
 vbNormal, 325  
 vbOK, 150  
 vbOKCancel, 149  
 vbOKOnly, 149  
 vbQuestion, 149, 214  
 vbReadOnly, 325, 329  
 vbRetry, 150  
 vbRetryCancel, 149  
 vbShortDate, 175  
 vbShortTime, 175  
 vbSystem, 325  
 vbSystemModal, 149  
 vbTab, 167  
 vbTextCompare, 161  
 vbVolume, 325  
 vbYes, 150  
 vbYesNo, 149  
 vbYesNoCancel, 149, 214  
 View Code, 43

– Definition, 62  
 – Object, 43  
 Visible, 239, 241, 374, 391  
 Visual Basic for Applications, 16  
 Volatile, 191  
 Volume Label, 323–324, 333

## W

Warning message, 214  
 Watch Window, 455  
 Watches, 461, 464–465  
 wdCell, 440  
 wdCharacter, 440  
 wdCollapseDirection, 444  
 wdCollapseEnd, 444  
 wdColumn, 440, 442  
 wdDialog, 341  
 wdDialogFileopen, 341  
 wdDialogFileSaveAs, 341  
 wdFormatTemplate, 433  
 wdGoToBookmark, 443  
 wdGoToField, 443  
 wdGoToHeading, 443  
 wdGoToLine, 443  
 wdGoToObject, 443  
 wdGoToPage, 443  
 wdGoToPercent, 443  
 wdLine, 440  
 wdMainTextStory, 435  
 wdMove, 442  
 wdParagraph, 440  
 wdPrimaryFooterStory, 435  
 wdPrimaryHeaderStory, 435  
 wdRow, 440  
 wdSaveFormat, 429  
 wdSection, 440  
 wdSentence, 440  
 wdStory, 440  
 wdStoryType, 435  
 wdTable, 440  
 wdTypeDocument, 342  
 wdTypeTemplate, 342, 345  
 wdUnits, 442  
 wdWord, 440  
 Weekday, 155

WeekdayName, 155  
What, 286  
While, 287  
Width, 268, 370  
Window, 235–236  
Windows, 257  
With...End With, 64, 253–254  
Words, 435  
Workbook, 235, 250, 397  
Workbooks, 239, 257  
Worksheet, 235, 250, 397, 406  
Worksheets, 239, 255, 257, 408

## X

xlExclusive, 403  
xlLocalSessionChanges, 404  
xlNoChange, 404  
xlOtherSessionChanges, 404  
xlShared, 404  
xlUnderlineStyle, 412  
xlUnderlineDouble, 412  
xlUnderlineDoubleAccounting, 412  
xlUnderlineNone, 412  
xlUnderlineSingle, 412  
xlUnderlineSingleAccounting, 412  
xlUserResolution, 404  
xlWBATChart, 399  
xlWBATemplate, 399  
xlWBATExcel4IntlMacroSheet, 399  
xlWBATExcel4MacroSheet, 399  
xlWBATWorksheet, 399  
Xor, 136

## Y

Year, 155, 178  
Yes/No, 170

## Z

zero-based, 302  
Zoom, 377

## A

автоматический отступ, 75  
автоотступ, 75  
аргумент (argument), 73, 75

## Б

байт (byte), 83  
бесконечный цикл (infinite loop), 276  
бинарный метод сравнения, 129  
бит (bit), 83  
блок (block) оператора, 198  
– кода обработки прерывания, 469  
булево значение, 89

## В

ветвь процедуры, 196  
вложение операторов, 203  
внешний контейнер, 255  
внутренняя константа, 110  
вызов функции, 144  
выражение, 114

## Д

действительное число, 88, 125  
динамический массив, 304  
директива компилятора (compiler directive), 104  
дочернее окно, 43

## И

идентификатор (identifier), 92  
именованная константа (named constant), 105  
именованный аргумент, 147  
– формат (named format), 168  
индекс массива, 303  
информационное сообщение, 214  
исключающее «ИЛИ» (exclusive or), 136  
исходный код макроса, 35  
итерация цикла, 275

## К

каталог, 321  
класс, 233  
кнопка-переключатель (toggle), 35  
код символа, 129  
коллекция (collection), 254  
Комментарий, 64  
компиляция, 78  
конкатенация, 89, 138  
константа (constant), 105

контейнер, 255  
критическое предупредительное  
сообщение, 214

## Л

литеральная константа (literal  
constant), 105  
логическое выражение, 116  
локальная область действия  
переменной, 101  
– переменная, 102

## М

макрокоманда, 16  
макрос, 15, 71  
маркер изменения размеров, 378  
массив, 302  
метод, 191  
– объекта, 233  
модальное приложение, 363  
модуль класса, 359

## Н

неопределенный цикл, 276  
нечеткий поиск, 130  
неявное объявление переменной, 93  
нулевая база, 302

## О

область действия (scope), 92, 99  
– – процедурного уровня  
(procedure-level scope), 99  
– объявлений модуля, 101  
общий шаблон, 37  
объект Err, 470  
объектная модель, 255  
объектное выражение (object  
expression), 116, 249  
объектно-ориентированное  
программирование, 231  
обязательный аргумент (required  
argument), 112  
ограниченное ключевое слово  
(restricted keyword), 92  
одномерный (single-dimensional)  
массив, 302  
операнд, 115

оператор, 119  
– Is, 251  
– On Error, 469, 475  
– Resume, 471  
– – line, 473  
– – Next, 472  
– VBA, 64  
– безусловного перехода, 196  
– условного перехода, 196  
операция отношения, 127  
– сравнения, 89  
определенная пользователем функция,  
180, 183  
относительные ссылки, 34  
ошибка времени исполнения, 451, 452  
– компиляции, 451–452  
– несовпадения типов, 116  
– несоответствия типа, 80  
– синтаксиса, 77  
ошибки времени исполнения, 79

## П

папка, 321  
переменная (variable), 78, 90  
Персистенция (persistence), 103  
плавающее окно, 43  
подпроцедура, 71  
последовательность перехода, 389  
постоянная точка останова, 458  
предопределенная константа, 110  
предупредительное сообщение, 214  
привязка к сетке, 379  
прикрепленное окно, 43  
простой массив, 302  
процедура, 71, 143  
– обработки событий, 364

## Р

Рабочий стол (desktop), 231  
режим исполнения, 455  
– прерывания, 455  
– разработки, 455  
результат функции, 114  
– функции (function result), 114  
рекурсивная (recursive) функция, 226

**С**

- свойства, 232
- свойство объекта, 232
- символ определения типа (type-definition character), 97
  - продолжения строки, 67
- символ-заполнитель (placeholder), 168
- синтаксис (syntax), 77
- синтаксическая ошибка, 451
- синтаксический анализ, 77
- сложное (составное) выражение (complex expression), 141
- событие, 363
- событийная процедура, 364
- сообщение запроса, 214
- список аргументов, 76
- статический массив, 304
- стек (stack), 226
- строка заголовка, 43
  - объявления, 64
  - состояния, 27, 32–33
  - фиксированной длины, 98
- строковое выражение, 115

**Т**

- таблица истинности (truth table), 134
- тело макроса, 64
  - цикла, 276
- тип даты, 115
  - последовательных дат (serial Dates), 86
- типизированная переменная, 96
- точка останова, 457
- трехмерная ссылка, 408

**У**

- унарный минус (unary minus), 123
- управление потоком, 196
- управление файлами, 321

**Ф**

- функция (function), 112, 143
- функция-процедура, 179

**Ч**

- числа с плавающей точкой (floating point numbers), 88
- численное выражение, 115
- число двойной точности (double-precision number), 88
  - одинарной точности, 88
  - с плавающей точкой, 125
  - с фиксированной точкой (fixed-point number), 88
- член класса (class member), 233

**Ш**

- шаг с заходом, 460, 465
  - с обходом, 465

**Э**

- экспоненциальное представление, 85
- элемент управления, 370
- явное объявление переменной, 94

# VBA

В.Г. Кузьменко

## Эффективное использование

Книга содержит курс и примеры программ на языке программирования Visual Basic в среде приложений Microsoft Office версий от 2000 до 2007 и предназначена для начинающих программировать в среде Windows с использованием объектов приложений Microsoft Office. Материал книги можно использовать для создания простых макросов с целью автоматизации часто применяемых наборов операций при работе с приложениями Microsoft Office, для написания новых функций (например, функций рабочего листа в приложении Excel), а также для разработки законченных Windows-приложений с обработкой данных с использованием диалоговых окон, обеспечивающих пользователя современными интерфейсными средствами.

Материал книги, содержащийся в приложениях, является удобным справочником при работе как с VBA, так и с обычным VB.

- Запись и выполнение макросов в приложениях Microsoft Office 2000-2007
- Использование встроенных и написание собственных процедур и функций
- Объекты и коллекции, введение в объектно-ориентированное программирование
- Управление внешними файлами, создание, использование, удаление файлов и каталогов
- Элементы диалоговых окон, введение в программирование Windows-интерфейса пользователя
- Управление host-приложениями
- Средства отладки VBA-кода
- Справочник операторов и встроенных функций Visual Basic

978-5-9518-0444-0



9 785951 804440