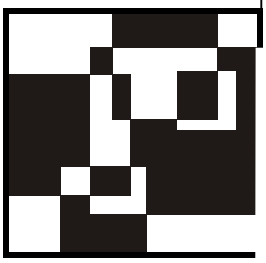
ГЛАВА 3



Синтаксис и программные конструкции VBA

3.1. Основы синтаксиса

Мы подошли к теме, которая многим пользователям покажется самой скучной, — синтаксис языка VBA. Относиться к этой теме, как мне кажется, следует так же, как к изучению азбуки или таблицы умножения: в самой азбуке или таблице умножения ничего интересного нет, но без их знания не удастся читать интересные книги или производить важные вычисления. Кроме того, VBA изначально проектировался и создавался как язык программирования, максимально дружелюбный по отношению к пользователю, который не является профессиональным программистом.

Для тех, кто хорошо знаком с обычным Visual Basic, в этой главе не будет почти ничего нового. Те, кто обладают опытом работы с любым другим современным языком программирования (C++, Java, Delphi, VBScript и JavaScript, Perl и т. п.) также освоят изложенный далее материал почти мгновенно. А тем, кто никогда не сталкивался ни с одним языком программирования, стоит просто выучить то, что изложено в этой главе, и постараться в течение какого-то времени активно применять полученные знания на практике, чтобы они не успели забыться. Потраченные усилия окупятся многократно, тем более что материала на самом деле не так и много — язык VBA очень прост.

Теперь рассмотрим некоторые общие моменты, связанные с синтаксисом языка VBA.

Синтаксис VBA, как понятно из самого названия этого языка (которое расшифровывается как Visual Basic for Applications), почти полностью совпадает с синтаксисом Visual Basic.

Основные синтаксические принципы этого языка следующие:

- VBA нечувствителен к регистру;
- чтобы закомментировать код до конца строки, используется одинарная кавычка (') или команда REM;
- символьные значения должны заключаться в двойные кавычки (");
- максимальная длина любого имени в VBA (переменные, константы, процедуры) — 255 символов;
- начало нового оператора — перевод на новую строку (точка с запятой, как в C, Java, JavaScript, для этого не используется);
- ограничений на максимальную длину строки нет (хотя в редакторе в строке помещается только 308 символов). Несколько операторов в одной строке разделяются двоеточиями:

```
MsgBox "Проверка 1" : MsgBox "Проверка 2"
```

- для удобства чтения можно объединить несколько физических строк в одну логическую при помощи пробела и знака подчеркивания после него:

```
MsgBox "Сообщение пользователю" _
    & vUserName
```

3.2. Операторы

Оператор — это наименьшая способная выполняться единица кода VBA. Оператор может объявлять или определять переменную, устанавливать параметр компилятора VBA или выполнять какое-либо действие в программе.

Арифметических операторов в VBA всего 7. Четыре стандартных: сложение (+), вычитание (–), умножение (*), деление (/), и еще три:

- возведение в степень (^). Например, `2^3 = 8`;
- целочисленное деление (\). Делит первое число на второе, отбрасывая (не округляя) дробную часть. Например, `5\2 = 2`;
- деление по модулю (Mod). Делит первое число на второе, возвращая только остаток от деления. Например, `5 Mod 2 = 1`.

Оператор присваивания в VBA — это знак равенства. Можно записывать так:

```
Let nVar = 10
```

а можно еще проще:

```
nVar = 10
```

Здесь не путайте знак равенства с оператором равенства. Последнее выражение означает "присвоить переменной `nVar` значение 10", а если строка выглядит так:

```
If (nVar = 10)
```

то это значит "если значение переменной `nVar` равно 10".

Если переменной нужно назначить объект, то делается это другими способами.

Операторов сравнения в VBA всего 8:

- равенство (=). Например, `If (nVar = 10)`;
- больше, чем (>) и меньше, чем (<). Например, `If (nVar > 10)`;
- больше или равно (>=) и меньше или равно (<=). Например, `If (nVar >= 10)`;
- не равно (<>). Например, `If (nVar<>10)`;
- сравнение объектов (`Is`). Определяет, ссылаются объектные переменные на один и тот же объект или на разные. Например, `If (obj1 is obj2)`;
- подобие (`Like`). Сравнивает строковый объект с шаблоном и определяет, подходит ли шаблон.

Операторы сравнения всегда возвращают `True` (если утверждение истинно) или `False` (если утверждение ложно).

Приведем некоторые особенности сравнения строковых значений:

- при сравнении строковых значений учитывается регистр;
- пробелы в строковых значениях также учитываются;
- при сравнении текстовых строк на больше/меньше по умолчанию сравниваются просто двоичные коды символов — какие больше или меньше. Если нужно использовать тот порядок, который идет в алфавите, то нужно воспользоваться командой

  ```
  Option Compare Text
  ```

Общий синтаксис оператора `Like` выглядит так:

Выражение1 `Like` *Выражение2*

При этом *Выражение1* — это любое текстовое выражение VBA, а *Выражение2* — шаблон, который передается оператору `Like`. В этом шаблоне можно использовать специальные подстановочные символы (табл. 3.1).

Очень часто при проверке нескольких условий используются логические операторы:

- `And` — логическое И. Должны быть истинными оба условия;
- `Or` — логическое ИЛИ. Должно быть истинным хотя бы одно из условий;

- `Not` — логическое отрицание. Возвращает `True`, если условие ложно;
- `Xor` — логическое исключение. В выражении `E1 Xor E2` возвращает `True`, если только `E1 = True` или только `E2 = True`, иначе — `False`;
- `Eqv` — эквивалентность двух выражений, возвращает `True`, если они имеют одинаковое значение;
- `Imp` — импликация, `E1 Imp E2` возвращает `False`, если `E1 = True` и `E2 = False`, иначе — `True`.

Помнить нужно про `And`, `Or`, `Not`, остальные логические операторы используются редко.

Таблица 3.1. *Подстановочные символы для оператора* `Like`

Подстановочный символ	Значение
#	Любая одна цифра от 0 до 9
*	Любое количество любых символов (включая нулевое)
?	Любой один символ
[a,b,c]	Любой один символ из приведенного в квадратных скобках списка
[!a,b,c]	Любой один символ, кроме приведенных в списке

Почти в любой программе VBA используются операторы конкатенации, т. е. слияния строковых значений. В VBA их два — (+) или (&). Рекомендуется всегда использовать оператор (&), потому что:

- при использовании (&) производится автоматическое преобразование числовых значений в строковые — нет опасности допустить ошибку;
- при использовании оператора (+) сложение строкового значения со значением типа `Null` дает `Null`.

Пример использования оператора (&):

```
MsgBox "Сообщение пользователю " & vUserName
```

Порядок применения операторов выглядит так: вначале в выражении вычисляются арифметические операторы, затем операторы конкатенации, следующими идут операторы сравнения и уже в самом конце логические. Если в выражении есть несколько операторов одного типа, то они применяются в обычном порядке — слева направо. При необходимости можно изменять порядок применения операторов при помощи круглых скобок.

3.3. Переменные и типы данных

Переменные — это контейнеры для хранения изменяемых данных. Без них не обходится практически ни одна программа. Для простоты переменную можно сравнить с номерком в гардеробе — вы сдаете в гардероб какие-то данные, в ответ вам выдается номерок. Когда вам опять потребовались эти данные, вы "предъявляете номерок" и получаете их. Пример работы с переменными в VBA может выглядеть так:

```
Dim nMyAge As Integer
nMyAge = nMyAge + 10
MsgBox nMyAge
```

Перед работой с переменной настоятельно рекомендуется ее объявить. Объявление переменной в нашем примере выглядит так:

```
Dim nMyAge As Integer
```

Расшифруем эту строку.

`Dim` — это область видимости переменной. В VBA предусмотрено 4 ключевых слова для определения области видимости переменных.

- `Dim` — используется в большинстве случаев. Если переменная объявлена как `Dim` в области объявлений модуля, то она будет доступна во всем модуле, если в процедуре — только на время работы этой процедуры.
- `Private` — при объявлении переменных в стандартных модулях VBA значит то же, что и `Dim`. Отличия проявляются только при создании своих классов (эта тема в данной книге не рассматривается).
- `Public` — такая переменная будет доступна всем процедурам во всех модулях данного проекта, если вы объявили ее в области объявлений модуля. Если вы объявили ее внутри процедуры, она будет вести себя как `Dim`.
- `Static` — такие переменные можно использовать только внутри процедуры. Эти переменные видны только внутри процедуры, в которой они объявлены, зато они сохраняют свое значение между разными вызовами этой процедуры. Обычно используются для накопления каких-либо значений. Например:

  ```
  Static nVar1 As Integer
  nVar1 = nVar1 + 1
  MsgBox nVar1
  ```

Если нет никаких особых требований, то имеет смысл всегда выбирать область видимости `Dim`.

Второе слово в нашем объявление (`nMyAge`) — это идентификатор (проще говоря, имя) переменной. Правила выбора имен в VBA едины для многих элементов (переменные, константы, функции, процедуры и т. п.):

- имя должно начинаться с буквы;
- не должно содержать пробелов и символов пунктуации (исключение — символ подчеркивания);
- максимальная длина — 255 символов;
- должно быть уникальным в текущей области видимости (подробнее *см. в разд. 3.3*);
- зарезервированные слова (те, которые подсвечиваются синим цветом в окне редактора кода) использовать нельзя.

При создании программ VBA настоятельно рекомендуется определиться с правилами, по которым будут присваиваться имена объектам — соглашение об именовании. Чаще всего используется так называемое венгерское соглашение (в честь одного из программистов Microsoft, Charles Simonyi, венгра по национальности):

- имя переменной должно начинаться с префикса, записанного строчными буквами. Префикс указывает, что именно будет храниться в этой переменной:
  - `str` (или `s`) — `String`, символьное значение;
  - `fn` (или `f`) — функция;
  - `sub` — процедура;
  - `c` (или все буквы имени заглавные) — константа;
  - `b` — `Boolean`, логическое значение (`True` или `False`);
  - `d` — дата;
  - `obj` (или `o`) — ссылка на объект;
  - `n` — числовое значение;
- имена функций, методов и каждое слово в составном слове должно начинаться с заглавной буквы:

```
MsgBox objMyDocument.Name
Sub CheckDateSub()
```

- в ранних версиях VB не было слова `Const`, все константы определялись как переменные, а для отличия их записывали заглавными буквами, между словами ставили символ подчеркивания, например `COMPANY_NAME`.

Многие программисты используют такой подход для обозначения констант и сейчас (но использование ключевого слова `Const` теперь обязательно — об этом будет рассказано в *разд. 3.4*).

Третья часть нашего объявления (`As Integer`) — это указание на тип данных нашей переменной. Тип данных определяет, данные какого вида можно будет хранить в этой переменной.

В VBA предусмотрены следующие типы данных:

- *числовые*:
  - `Byte` — целое число от 0 до 255;
  - `Integer` — целое число от –32 768 до 32 767;
  - `Long` — большое целое число от –2 147 483 648 до 2 147 483 647;
  - `Currency` — большое десятичное число с 19 позициями, включая 4 позиции после запятой;
  - `Decimal` — еще большее десятичное число с 29 позициями (после запятой можно использовать от 0 до 28 позиций);
  - `Single` и `Double` — значения с плавающей запятой (`Double` в 2 раза больше));

Внимание!

Попытка объявить переменную с типом `Decimal` (например, `Dim n As Decimal`) приведет к синтаксической ошибке. Чтобы получить возможность работать с типом `Decimal`, переменную нужно изначально объявить как `Variant` или вообще объявить без типа (`Dim n`), поскольку тип данных `Variant` используется в VBA по умолчанию. Если мы присвоим переменной с типом `Variant` числовое значение, для которого подходит только `Decimal`, VBA автоматически назначит ей этот тип данных. Кроме того, можно явно указать подтип `Decimal` для переменной типа `Variant` при помощи функции `CDec()`.

- *строковые* (`String` переменной длины (примерно до 2 млрд символов) и фиксированной длины (примерно до 65 400 символов));
- *дата и время* (`Date` — от 01.01.100 до 31.12.9999);
- *логический* (`Boolean` — может хранить только значения `True` и `False`);
- *объектный* (`Object` — хранит ссылку на любой объект в памяти);
- `Variant` — специальный тип данных, который может хранить любые другие типы данных.

Можно также использовать пользовательские типы данных, но их вначале нужно определить при помощи выражения `Type`. Обычно пользовательские

типы данных используются как дополнительное средство проверки вводимых пользователем значений (классический пример — почтовый индекс).

Приведем некоторые моменты, связанные с выбором типов данных для переменных:

- общий принцип — выбирайте наименьший тип данных, который может вместить выбранные вами значения. Если есть какие-то сомнения — выбирайте больший тип во избежание возникновения ошибок;
- если есть возможность, лучше не использовать типы данных с плавающей запятой (`Single` и `Double`). Работа с ними производится медленнее, кроме того, могут быть проблемы при сравнениях за счет округлений;
- при определении переменных можно использовать так называемые символы определения типа ((`%`) — `Integer`, (`$`) — `String` и т. п.). Например, в нашем примере можно закомментировать строку:

  ```
  ' Dim nVar1 As Integer
  ```

  а во второй строке написать:

  ```
  nVar1% = nVar1% + 1
  ```

  Такой подход является устаревшим и к использованию не рекомендуется.

При объявлении переменной можно не указывать ее тип. Например, наше объявление может выглядеть так:

```
Dim nVar1
```

В этом случае переменная будет автоматически объявлена с типом `Variant`.

В принципе, в VBA можно работать и без объявления переменных. Например, такой код

```
nVar1 = nVar1 + 1
MsgBox nVar1
```

будет вполне работоспособным. Если мы используем переменную в программе без ее объявления, то будет автоматически создана новая переменная типа `Variant`. Однако объявлять переменные нужно обязательно! И при этом желательно явно указывать нужный тип данных. Потому что:

- сокращается количество ошибок: программа с самого начала откажется принимать в переменную значение неправильного типа (например, строковое вместо числового);
- при работе с объектами подсказка по свойствам и методам действует только тогда, когда мы изначально объявили объектную переменную с нужным типом. Например, в Excel два варианта кода будут работать одинаково:

- первый вариант:

```
Dim oWbk As Workbook
Set oWbk = Workbooks.Add
```

- второй вариант:

```
Set oWbk = Workbooks.Add
```

Но подсказка по свойствам и методам объекта `oWbk` будет работать только в первом случае.

Все опытные разработчики вообще запрещают использовать переменные без явного их объявления. Для этого можно воспользоваться специальной командой компилятора (она помещается только в раздел объявлений модуля):

```
Option Explicit
```

а можно вставлять эту команду во все модули при их создании автоматически, установив в окне **Options** флажок **Require Variable Declarations** (меню **Tools | Options**, вкладка **Editor**).

Проиллюстрировать, зачем они это делают, можно на простом примере:

```
Dim n
n = n + 1
MsgBox п
```

С виду код не должен вызывать никаких проблем и просто выводить в окне сообщения единицу. На самом деле он выведет пустое окно сообщения. Причина спрятана очень коварно: в третьей строке `n` — это вовсе не английская буква N, а русская П. На вид в окне редактора кода отличить их очень сложно. В то же время компилятор VBA, встретив такой код, просто создаст новую переменную с типом данных `Variant`, у которой будет пустое значение. На выявление такой ошибки может потребоваться определенное время. В то же время при установленном `Option Explicit` проблема определяется мгновенно.

Можно объявить несколько переменных в одной строке, например, так:

```
Dim nVar1 As Integer, s1 As String
```

Присвоение значений переменным выглядит так:

```
nVar1 = 30
```

Если нужно увеличить уже существующее значение переменной, то команда может выглядеть так:

```
nVar1 = nVar1 + 1
```

В обоих примерах знак равенства означает не "равно", а "присвоить".

При присвоении значений переменным нужно помнить о следующем:

- строковые значения всегда заключаются в двойные кавычки:

```
sVar1 = "Hello"
```

- значение дата/время заключается в символы "решетка" (#):

```
dVar1 = #05/06/2004#
```

  Обратите внимание, что при присвоении значения даты/времени таким "явным" способом нам приходится использовать принятые в США стандарты: `05` в данном случае — это месяц, `06` — день. А отображение этого значения (например, в окне сообщения) будет зависеть от региональных настроек на компьютере пользователя;

- если нужно передать шестнадцатеричное значение, то перед ним ставятся символы (`&H`):

```
nVar1 = &HFF00
```

  Для передачи восьмеричного значения (что случается намного реже) нужно использовать набор символов (`&O`).

В переменных до присвоения им значений пользователем содержится:

- в переменных всех числовых типов данных — 0;
- в строковых переменных переменной длины — `""` (строка нулевой длины);
- в строковых переменных фиксированной длины — строка заданной длины с символами ASCII 0 (эти символы на экран не выводятся);
- в `Variant` — специальное пустое значение `Empty`. Произвести проверку на это значение (т. е. было ли присвоено значение переменной или нет) можно при помощи функции `IsEmpty()`;
- в `Object` — ничего (нет ссылки ни на один из объектов).

Задание для самостоятельной работы 3.1: Работа с переменными и операторами

Подготовка:

1. Создайте новую книгу Excel и сохраните ее как C:\LabVariablesOperators.xls. Введите в ячейки A1, A2 и A3 этой книги любые значения.
2. Откройте редактор Visual Basic в Excel и создайте в этой книге новый стандартный модуль *(см. разд. 2.2)*.

3. При помощи меню **Tools | References** добавьте в ваш проект ссылку на библиотеку Microsoft Word 11.0 Object Library.
4. Введите в созданном вами стандартном модуле следующий код:

```
Public Sub FromExcelToWord()
    MsgBox Range("A1").Text
    MsgBox Range("A2").Text
    MsgBox Range("A3").Text

    Dim oWord As Word.Application
    Dim oDoc As Word.Document
    Set oWord = CreateObject("Word.Application")
    oWord.Visible = True
    Set oDoc = oWord.Documents.Add()
    oDoc.Activate
    oWord.Selection.TypeText "Вставляемый текст"
End Sub
```

   Этот код должен выводить в окна сообщений значения ячеек A1, A2 и A3, а затем открыть Word и впечатать в начало нового документа строку `"Вставляемый текст"`.
5. Убедитесь, что код работает без ошибок.

ЗАДАНИЕ:

Измените код этой процедуры таким образом, чтобы вместо строки `"Вставляемый текст"` выводились значения ячеек A1, A2 и A3 вместе.

Ответ к заданию 3.1

Итоговый код может выглядеть так:

```
Public Sub FromExcelToWordAnswer()
    Dim sA1, sA2, sA3, sText As String
    sA1 = Range("A1").Text
    sA2 = Range("A2").Text
    sA3 = Range("A3").Text

    sText = sA1 + " " + sA2 + " " + sA3

    Set oWord = CreateObject("Word.Application")
    oWord.Visible = True
    Set oDoc = oWord.Documents.Add()
    oDoc.Activate
    oWord.Selection.TypeText sText
End Sub
```

3.4. Константы

Константы — еще один контейнер для хранения данных, но, в отличие от переменных, они не изменяются в ходе выполнения VBA-программы. Константы используют в следующих случаях:

- код становится более читаемым, убираются потенциальные ошибки;
- чтобы изменить какое-либо значение в коде (например, уровень налога), это нужно сделать всего один раз — в объявлении константы.

В VBA константы определяются при помощи ключевого слова `Const`:

```
Const COMP_NAME As String = "Microsoft"
```

Главное отличие констант от переменных заключается в том, что при попытке изменить значение константы в теле процедуры будет выдано сообщение об ошибке.

Константы очень удобны при работе с группами именованных элементов (дни недели, месяцы, цвета, клавиши, типы окон и т. п.). Они позволяют использовать в коде программы легко читаемые обозначения вместо труднозапоминаемых числовых кодов. Например, строки:

```
UserForm1.BackColor = vbGreen
```

и

```
UserForm1.BackColor = 65280
```

функционально одинаковы, но в чем смысл первой строки, догадаться гораздо легче.

В VBA встроено множество служебных констант: календарных, для работы с файлами, цветами, формами, типами дисков и т. п. Просмотреть их можно через справочную систему VBA: **Microsoft Visual Basic Documentation | Visual Basic Language Reference | Constants**. Про одну из констант (она находится в разделе **Miscellaneous Constants**) следует сказать особо: константа `vbCrLf` позволяет произвести переход на новую строку, например:

```
MsgBox "Первая строка" + vbCrLf + "Вторая строка"
```

Множество наборов констант встроено в объектные модели приложений Office, которые мы будем рассматривать в *гл. 10—15*.

3.5. Операторы условного и безусловного перехода

Операторы условного перехода — одни из самых важных и часто используемых элементов в языках программирования. Общий принцип их работы прост: проверяется соответствие каким-то условиям (истинность или лож-

ность каких-либо выражений) и в зависимости от этого выполнение программы направляется по одной или другой ветви. В VBA предусмотрено два оператора условного перехода: `If...Then` и `Select Case`.

3.5.1. Оператор *If ... Then*

Оператор `If...Then` — самый популярный у программистов. Полный его синтаксис выглядит так (необязательные части заключены в квадратные скобки):

```
If Условие Then
    Команды1
[ElseIf УсловиеN Then
    КомандыN]
[Else
    Команды2]
End If
```

При этом:

- *Условие* — выражение, которое проверяется на истинность. Если оно истинно, то выполняются *Команды1*, если ложно — *Команды2*;
- *УсловияN* — дополнительные условия, которые также можно проверить. В случае, если они выполняются (выражение *УсловияN* истинно), то выполняются *КомандыN*. Дополнительные условия (вместе с конструкцией `ElseIf`) можно повторять неограниченное количество раз, но если вам нужно реализовать проверку на соответствие большому количеству условий, то правильнее будет использовать конструкцию `Select Case` *(см. разд. 3.5.2)*.

Оператор `If...Then` применяется:

- когда нужно проверить на одно условие и в случае соответствия сделать какое-то действие:

```
If nTemperature < 10 Then
    MsgBox "Надеть куртку"
End If
```

- когда нужно сделать то же, что и в предыдущем примере, а в случае несоответствия выполнить другое действие:

```
If nTemperature < 10 Then
    MsgBox "Надеть куртку"
Else
    MsgBox "Надеть ветровку"
End If
```

- когда нужно проверить на соответствие несколько условий (обратите внимание на использование логических операторов):

```
If (nTemperature < 10) And (bRain = True) Then
    MsgBox "Надеть куртку и взять зонтик"
End If
```

- в случае, когда проверка первого условия вернула `False`, нужно проверить на соответствие еще несколько условий (удобно использовать `ElseIf`):

```
If (bIGoInCar = True) Then
    MsgBox "Одеться для машины"
ElseIf nTemperature < 10 Then
    MsgBox "Надеть куртку"
Else
    MsgBox "Можно идти в рубашке"
End If
```

  В этом примере, поскольку `bIGoInCar` — переменная типа `Boolean` и сама по себе принимает значения `True` или `False`, первая строка может выглядеть так:

```
If bIGoInCar Then ...
```

Приведу некоторые замечания по использованию `If...Then`:

- ключевое слово `Then` должно находиться в одной строке с `If` и условием. Если вы перенесете его на следующую строку, будет выдано сообщение об ошибке;
- если разместить команду, которую нужно выполнить при истинности проверяемого условия, на одной строке с `If` и `Then`, то `End If` можно не писать:

```
If nTemperature < 10 Then MsgBox "Надеть куртку"
```

  Если же вы используете несколько команд или конструкции `Else/ElseIf`, то `End If` в конце нужно писать обязательно, иначе возникнет синтаксическая ошибка;
- для выражения `If...Then` настоятельно рекомендуется использовать отступы для выделения блоков команд. Иначе читать код будет трудно;
- операторы `If...Then` можно вкладывать друг в друга:

```
If MyVar = 5 Then
    MsgBox "MyVar = 5"
    If MyVar = 10 Then
        MsgBox "MyVar = 10"
    End If
End If
```

3.5.2. Оператор *Select Case*

Оператор `Select Case` идеально подходит для проверки одного и того же значения, которое нужно много раз сравнить с разными выражениями. Синтаксис его очень прост:

```
Select Case Выражение
    Case Условие1
        Команды1
    [Case УсловиеN
        КомандыN]
    [Case Else
        Команды2]
End Select
```

Например:

```
Select Case sDayOfWeek
    Case "Понедельник"
        MsgBox "Салат из шпината"
    Case "Вторник"
        MsgBox "Салат из морской капусты"
    Case Else
        MsgBox "На этот день у нас ничего не предусмотрено"
End Select
```

Приведу некоторые замечания по поводу `Select Case`:

- строка:

  ```
  Case "Понедельник"
  ```

  на самом деле означает:

  ```
  Case sDayOfWeek = "Понедельник"
  ```

  Такое равенство подразумевается по умолчанию. Но вы можете использовать и другой оператор сравнения или целый набор операторов, например:

  ```
  Case 0 To 5, 15, Is > 55
  ```

  Такое выражение можно перевести как "Если проверяемое выражение попало в диапазон от 0 до 5 включительно, или равно 15, или больше 55".

  Слово `Is` при этом можно пропустить — компилятор VBA добавит это ключевое слово за вас. Несколько критериев в `Case` перечисляются через запятые и объединяются так, как работает оператор `Or`, т. е. выполнение пойдет по этой ветви, если тестируемое значение будет удовлетворять хотя бы одному из критериев;

- при использовании диапазона (`0 To 5`) включаются и границы диапазона (в данном случае `0` и `5`).

3.5.3. Оператор *GoTo*

Оператор `GoTo` — это оператор безусловного перехода, когда ход выполнения программы без проверки каких-либо условий перепрыгивает на метку в коде. Пример использования `GoTo` может выглядеть так:

```
GoTo EngineNotStarted
    ...
EngineNotStarted:
    MsgBox "Едем на метро"
    ...
```

Здесь `EngineNotStarted:` — это метка, для нее используется имя (выбираемое по правилам назначения имен для переменных), которое оканчивается двоеточием. Эта метка может находиться как до, так и после оператора `GoTo`. В любом случае, при выполнении оператора `GoTo` ход выполнения "перепрыгнет" на указанную в `GoTo` метку.

Иногда использование `GoTo` очень удобно, например, когда нам нужно добиваться от пользователя ввода правильного значения неизвестное число раз. Однако использовать `GoTo` категорически не рекомендуется, потому что код становится трудночитаемым. Чаще всего `GoTo` можно заменить на конструкцию `Do While...Loop` *(см. разд. 3.6)* или на вызов функции из самой себя.

Задание для самостоятельной работы 3.2: Работа с операторами условного перехода

Подготовка:

1. Создайте новую книгу Excel и сохраните ее как C:\LabCondConstructions.xls.
2. Откройте редактор Visual Basic в Excel и создайте в этой книге новый стандартный модуль.
3. Введите в этом модуле следующий код:

```
Public Sub IfThenSub()
    Dim nResult As Integer
    nResult = MsgBox("Нажмите кнопку", vbYesNo, "Окно сообщения")
    ThisWorkbook.Worksheets(1).Range("A1").Value = _
        "Вы нажали кнопку: " & nResult
    ThisWorkbook.Worksheets(1).Range("A1").Columns.AutoFit
End Sub
```

4. Запустите этот код на выполнение и убедитесь, что он выполняется без ошибок. Этот код должен вставлять в ячейку A1 первого листа вашей книги текстовое значение вида "Вы нажали кнопку: 6", в зависимости от того, какая кнопка была нажата в окне сообщения.

Примечание

Получить информацию о том, какие значения при нажатии какой кнопки возвращаются из окна сообщения, можно при помощи справки по функции `MsgBox()`.

ЗАДАНИЕ 3.2, А:

Измените код этой процедуры таким образом, чтобы вместо чисел в ячейку вписывалось строковое значение нажатой кнопки (например, "Вы нажали кнопку: Да"). Используйте при этом синтаксическую конструкцию `If...Then...Else`.

Ответ к заданию 3.2, А:

Итоговый код для вашей процедуры может быть таким:

```
Public Sub IfThenSubAnswer()
    Dim nResult As Integer
    nResult = MsgBox("Нажмите кнопку", vbYesNo, "Окно сообщения")
    If nResult = vbYes Then
        sResult = "Да"
    ElseIf nResult = vbNo Then
        sResult = "Нет"
    Else
        sResult = "Неизвестная кнопка"
    End If
    ThisWorkbook.Worksheets(1).Range("A1").Value = "Вы нажали кнопку: " _
        & sResult
    ThisWorkbook.Worksheets(1).Range("A1").Columns.AutoFit
End Sub
```

ЗАДАНИЕ 3.2, Б:

Замените в вашей процедуре строку:

```
nResult = MsgBox("Нажмите кнопку", vbYesNo, "Окно сообщения")
```

на:

```
nResult = MsgBox("Нажмите кнопку", vbAbortRetryIgnore, "Окно сообщения")
```

Измените вашу процедуру таким образом, чтобы она вставляла в ячейку A1 значения "Прервать", "Повторить" или "Пропустить", в зависимости от того, какая кнопка в окне сообщения была нажата. Используйте при этом синтаксическую конструкцию `Select Case`.

Ответ к заданию 3.2, Б:

Итоговый код для вашей процедуры может быть таким:

```
Private Sub SelectCaseAnswer()
    nResult = MsgBox("Нажмите кнопку", vbAbortRetryIgnore, "Окно
сообщения")
    Select Case nResult
        Case vbAbort
            sResult = "Отменить"
        Case vbRetry
            sResult = "Повторить"
        Case vbIgnore
            sResult = "Пропустить"
        Case Else
            sResult = "Неизвестная кнопка"
    End Select
    ThisWorkbook.Worksheets(1).Range("A1").Value = "Вы нажали кнопку: " _
        & sResult
    ThisWorkbook.Worksheets(1).Range("A1").Columns.AutoFit
End Sub
```

3.6. Работа с циклами

Циклы используются в ситуациях, когда нам нужно выполнить какое-либо действие несколько раз. Первая ситуация — мы знаем, сколько раз нужно выполнить определенное действие, в этом случае используется конструкция `For...Next`:

```
For iCounter = 1 to 10
    MsgBox "Счетчик: " & iCounter
Next
```

Чтобы указать, насколько должно прирастать значение счетчика, используется ключевое слово `Step`:

```
For iCounter = 1 to 10 Step 2
    MsgBox "Счетчик: " & iCounter
Next
```

Можно и уменьшать исходное значение счетчика:

```
For iCounter = 10 to 1 Step -2
    MsgBox "Счетчик: " & iCounter
Next
```

Для безусловного выхода из конструкции `For...Next` используется команда `Exit For`:

```
VStop = InputBox("Введите значение останова")
VInput = CInt(VStop)
For iCounter = 1 to 10
    MsgBox "Счетчик: " & iCounter
    If iCounter =VInput Then Exit For
Next
```

Очень часто в VBA требуется сделать какое-нибудь действие со всеми элементами коллекции или массива — перебрать все открытые документы, все листы Excel, все ячейки в определенном диапазоне и т. п. Для того чтобы пройти циклом по всем элементам коллекции, используется команда `For Each...Next`:

```
For Each oWbk in Workbooks
    MsgBox oWbk.Name
Next
```

При использовании этого приема можно очень просто найти и получить ссылку на нужный нам объект:

```
For Each oWbk in Workbooks
    If oWbk.Name = "Сводка.xls" Then
        Set oMyWorkBook = oWbk
        Exit For
    End If
Next
```

В этом случае мы проходим циклом по всем элементам коллекции `Workbooks` (т. е. по открытым рабочим книгам в Excel), для каждой книги проверяем ее имя и, если нашли книгу с именем `"Сводка.xls"`, получаем на нее ссылку и выходим из цикла. Коллекция рабочих книг — это специальная коллекция, которая умеет производить поиск в себе по имени элемента, поэтому, в принципе, можно было обойтись такой строкой:

```
Set oMyWorkBook = Workbooks("Сводка.xls")
```

Но для многих других коллекций без конструкции `For Each...Next` не обойтись.

Еще одна ситуация: когда мы не знаем точно, сколько раз должна быть выполнена та или другая команда — это зависит от какого-либо условия. В этом случае используются конструкции `Do While...Loop` и `Do Until...Loop`.

Конструкция `Do While...Loop` означает: выполнять какое-либо действие до тех пор, пока условие истинно:

```
Do While MyVar < 10
    MyVar = MyVar + 1
    MsgBox "MyVar = " & MyVar
Loop
```

Применений на практике — множество: пройти по всему набору записей, пока они не закончатся, требовать от пользователя ввести подходящее значение, пока он наконец не введет его, и т. п.

Внимание!

Если вы случайно запустили в своей программе бесконечный цикл, нажмите на клавиши <Ctrl>+<Break>. Откроется окно, аналогичное представленному на рис. 3.1, в котором вы сможете продолжить выполнение (кнопка **Continue**), завершить его (**End**) или открыть ваш код в отладчике (**Debug**).

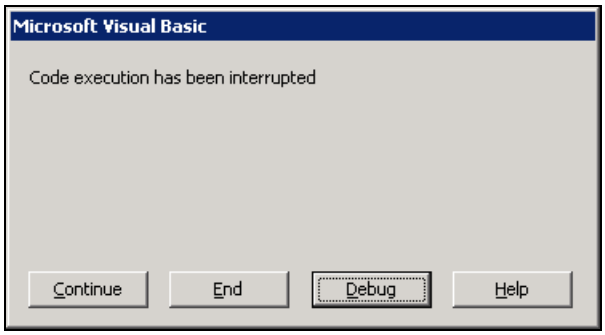


Рис. 3.1. Выполнение макроса остановлено по <Ctrl>+<Break>

Второй вариант — `Do Until...Loop`. Все выглядит точно так же, за одним исключением: цикл будет выполняться до тех пор, пока условие ложно.

```
Do Until MyVar >= 10
    MyVar = MyVar + 1
    MsgBox "MyVar = " & MyVar
Loop
```

Можно переписать цикл так, чтобы условие проверялось после завершения цикла:

```
Do
    MyVar = MyVar + 1
    WScript.Echo "MyVar = " & MyVar
Loop While MyVar < 10
```

В этом случае цикл будет выполнен, по крайней мере, один раз.

Немедленный выход из цикла можно произвести по команде `Exit Do`.

В VBA имеется также конструкция `While...Wend`. Это еще один вариант цикла, который оставлен для обратной совместимости с первыми версиями Visual Basic. Функциональные возможности — те же, что и у конструкции `Do While...Loop`:

```
While My Var < 10
    MyVar = MyVar + 1
    WScript.Echo "MyVar = " & MyVar
Wend
```

3.7. Массивы

Массивы используются для хранения в памяти множества значений. Вместо того чтобы объявлять множество похожих друг на друга переменных, часто гораздо удобнее воспользоваться массивом.

Объявление массива производится очень просто:

```
Dim MyArray(2) As Integer
```

Здесь `2` — это верхняя граница массива *(upper bound)*. По умолчанию нижней границе массива *(lower bound)* соответствует элемент с номером 0. Количество элементов, которое может хранить массив, — от 0 до верхней границы включительно. Наш массив может хранить три целочисленных элемента.

Если вам хочется, чтобы нижняя граница массива (и, соответственно, нумерация элементов) начиналась с 1, то в раздел объявлений модуля нужно внести команду:

```
Option Base 1
```

В принципе, тип данных для массива можно не объявлять:

```
Dim MyArray(2)
```

В этом случае для элементов массива будет использован тип `Variant`. Такой массив сможет хранить в себе элементы разных типов данных, но требования к памяти у него будут выше и работать он будет чуть медленнее по сравнению с массивом, для которого тип данных указан явно (например, `Integer` или `String`).

Присвоить значение отдельному элементу массива (в нашем случае — первому) можно очень просто:

```
MyArray(0) = 100
```

А затем это значение можно будет извлечь:

```
MsgBox MyArray(0)
```

Массивы могут быть многомерными, в частности, двумерными:

```
Dim MyArray(4, 9)
```

В каждой строке многомерного массива удобно хранить данные, относящиеся к одному объекту (например, имя сотрудника, уникальный номер, номер телефона). В VBScript в одном многомерном массиве может быть до 60 измерений.

Часто необходимо использовать *динамические* массивы, размер которых можно изменять в ходе выполнения. Динамический массив объявляется следующим образом:

```
Dim MyArray()        ' объявляем массив без верхней границы
ReDim MyArray(4)     ' изменяем размер массива
```

Команда `ReDim` не только изменяет размер массива, но и удаляет из него все старые значения. Чтобы старые значения сохранились, используется ключевое слово `Preserve`:

```
ReDim Preserve MyArray(7)
```

Однако если новый размер массива меньше, чем количество помещенных в него элементов, слово `Preserve` не поможет — часть данных все равно будет потеряна.

Массивы можно создавать и заполнять одновременно:

```
Dim MyArray
MyArray = Array(100, 200, 300, 400, 500)
```

Указывать размер массива необязательно — он будет автоматически настроен в соответствии с количеством передаваемых элементов.

Очистить массив можно командой `Erase`:

```
Erase MyArray
```

Массив фиксированной длины просто очищается, динамический массив разинициализируется — его придется инициализировать (определять размер) заново.

В динамических массивах часто не известно, сколько элементов в массиве. Для определения количества элементов используется функция `UBound()` (если массив одномерный или вас интересует размер первого измерения, то *измерение* передавать не надо):

```
UBound(имя_Массива [, измерение])
```

Как ни удивительно, но при программировании в VBA вам редко придется сталкиваться с массивами. Вместо них в объектных моделях приложений

Office обычно используются коллекции. *Коллекции* — это специальные объекты, которые предназначены для хранения наборов одинаковых элементов. Например, в Word предусмотрена коллекция `Documents` для хранения элементов `Document`, т. е. всех открытых документов, в Excel — коллекции `Workbooks` (открытые книги) и `Worksheets` (листы в книге) и т. п. Коллекции обычно удобнее, чем массивы: они изначально безразмерны и в них предусмотрен стандартный набор свойств и методов (метод `Add()` для добавления нового элемента, свойство `Count` для получения информации о количестве элементов, метод `Item()` для получения ссылки на нужный элемент). Для многих коллекций в объектных моделях кроме стандартных предусмотрен еще и набор специфических свойств и методов.

Задание для самостоятельной работы 3.3: Работа с циклами

Подготовка:

1. Создайте новый документ Word с именем C:\LabLoops.doc. Введите в этом документе несколько предложений с текстом.
2. Откройте редактор Visual Basic и создайте в этом документе новый стандартный модуль.

ЗАДАНИЕ 3.3, А:

Создайте в этом стандартном модуле процедуру `ForNextSub()`, которая выводила бы последовательно 10 окон сообщений с цифрами от 1 от 10.

Ответ к заданию 3.3, А:

Итоговый код процедуры может быть таким:

```
Public Sub ForNextSub()
    Dim i As Integer
    For i = 1 To 10
        MsgBox i
    Next
End Sub
```

или таким:

```
Public Sub ForNextSub2()
    Dim i As Integer
    i = 1
    Do While i <= 10
        MsgBox i
```

```
        i = i + 1
    Loop
End Sub
```

ЗАДАНИЕ 3.3, Б:

Создайте в этом программном модуле процедуру `ForEachSub()`, которая выводила бы в окна сообщений последовательно каждое слово в вашем документе LabLoops.doc.

Коллекцию (массив) всех слов документа можно получить при помощи конструкции `ThisDocument.Words`. Значение каждого слова можно получить при помощи свойства `Text` элемента этой коллекции.

Ответ к заданию 3.3, Б:

Итоговый код процедуры `ForEachSub()` может быть таким:

```
Public Sub ForEachSub()
    For Each oWord In ThisDocument.Words
        MsgBox oWord.Text
    Next
End Sub
```

3.8. Процедуры и функции

3.8.1. Виды процедур

Процедуры — это самые важные функциональные блоки языка VBA. В VBA вы можете выполнить только тот программный код, который содержится в какой-либо процедуре (обычной в стандартном модуле, событийной для элемента управления на форме и т. п.). Иногда начинающие пользователи пытаются записать команды прямо в область объявлений стандартного модуля и не могут понять, почему они не выполняются (сообщения об ошибке при этом не выдается — просто этот код становится "невидим" для компилятора). Причина проста: в разделе объявлений модуля (когда в верхних списках редактора кода показываются значения **General** и **Declarations**) могут быть только объявления переменных уровня модуля и некоторые специальные инструкции для компилятора. Весь остальной программный код должен находиться внутри процедур.

В VBA предусмотрены следующие типы процедур:

- процедура типа `Sub` (подпрограмма) — универсальная процедура для выполнения каких-либо действий:

```
Sub Farewell()
    MsgBox "Goodbye"
End Sub
```

Макрос в VBA — это процедура типа `Sub`, не имеющая параметров. Только макросы можно вызывать по имени из редактора VBA или из приложения Office. Все другие процедуры нужно вызывать либо из других процедур, либо специальными способами, о которых будет рассказано далее;

- процедура типа `Function` (функция) — набор команд, которые должны быть выполнены. Принципиальное отличие только одно: функция возвращает вызвавшей ее программе (или процедуре) какое-то значение, которое будет там использовано. Пример функции:

```
Function Tomorrow()
    Tomorrow = DateAdd("d", 1, Date())
End Function
```

и пример ее вызова:

```
Private Sub Test1()
    Dim dDate
    dDate = Tomorrow()
    MsgBox dDate
End Sub
```

В тексте функции необходимо предусмотреть оператор, который присваивает ей какое-либо значение. В нашем случае это строка:

```
Tomorrow = DateAdd("d", 1, Date())
```

В принципе, процедуры типа `Sub` тоже могут возвращать значения — при помощи переменных, передаваемых по ссылке *(см. разд. 3.8.4)*. Зачем же тогда нужны функции? Все очень просто: функцию можно вставлять практически в любое место программного кода. Например, наш последний пример может выглядеть намного проще:

```
Private Sub Test1()
    MsgBox Tomorrow()
End Sub
```

В VBA предусмотрены сотни встроенных функций (и гораздо большее количество функций предусмотрено в объектных моделях приложений Office). Даже в нашем примере используются две встроенные функции: `Date()`, которая возвращает текущую дату по часам компьютера, и `DateAdd()`, которая умеет прибавлять к текущей дате определенное количество дней, недель, месяцев, лет и т. п. Про встроенные функции будет рассказано в *разд. 3.9*.

В VBA имеются также *процедуры обработки событий* (event procedure) — процедуры типа `Sub` специального назначения, которые выполняется в случае возникновения определенного события, например, при открытии формы или нажатии на ней кнопки. Про события подробнее будет рассказано в *гл. 5*.

Есть еще процедуры типа `Property` (процедуры свойства). Они нужны для определения свойств создаваемого вами класса, а поскольку созданием своих классов мы заниматься не будем, то их можно не рассматривать.

3.8.2. Область видимости процедур

По умолчанию все процедуры VBA (за исключением процедур обработки событий) определяются как *открытые* (`Public`). Это значит, что их можно вызвать из любой части программы — из того же модуля, из другого модуля, из другого проекта. Объявить процедуру как `Public` можно так:

```
Public Sub Farewell()
```

или, поскольку процедура определяется как `Public` по умолчанию, то можно и так:

```
Sub Farewell()
```

Можно объявить процедуру локальной:

```
Private Sub Farewell()
```

В этом случае эту процедуру можно будет вызвать только из того модуля, в котором она расположена. Такое решение иногда может предотвратить ошибки, связанные с вызовом процедур, не предназначенных для этого, из других модулей.

Можно ограничить область видимости открытых процедур (тех, которые у вас определены как `Public`) в каком-то модуле рамками одного проекта. Для этого достаточно в разделе объявлений этого модуля вписать строку

```
Option Private Module
```

Если при объявлении процедуры использовать ключевое слово `Static`, то все переменные в этой процедуре автоматически станут статическими и будут сохранять свои значения и после завершения работы процедуры *(см. разд. 3.3)*. Например:

```
Private Static Sub Farewell()
```

3.8.3. Объявление процедур

Объявить процедуру можно вручную, например, добавив в код строку:

```
Private Sub Farewell()
```

При этом редактор кода автоматически добавит строку `End Sub` и линию-разделитель. А можно объявить процедуру, воспользовавшись меню **Insert | Procedure**. Разницы нет никакой.

3.8.4. Передача параметров

Параметры — это значения, которые передаются от одной процедуры к другой. В принципе, можно обойтись и без параметров, воспользовавшись только переменными уровня модуля, но при использовании параметров читаемость программы улучшается. Чтобы процедура имела возможность принимать параметры, ее вначале нужно объявить с параметрами. Далее приведен пример простой функции, которая складывает два числа и возвращает результат:

```
Function fSum(nItem1 As Integer, nItem2 As Integer)
    fSum = nItem1 + nItem2
End Function
```

Ее вызов может выглядеть так:

```
MsgBox fSum(3, 2)
```

В данном случае мы объявили оба параметра как обязательные, и поэтому попытка вызвать функцию без передачи ей какого-либо параметра (например, `fSum(3)`) приведет к ошибке "Argument not optional" — "Параметр не является необязательным". Чтобы можно было пропускать какие-то параметры, их нужно сделать необязательными. Для этой цели используется ключевое слово `Optional`:

```
Function fSum(nItem1 As Integer, Optional nItem2 As Integer)
```

В справке по встроенным функциям VBA необязательные параметры заключаются в квадратные скобки.

Для проверки того, был ли передан необязательный параметр, используется либо функция `IsMissing` (если для этого параметра был использован тип `Variant`), либо его значение сравнивается со значениями переменных по умолчанию (ноль для числовых данных, пустая строка для строковых и т. п.).

Вызов функции с передачей параметров может выглядеть так:

```
nResult = fSum(3, 2)
```

Однако здесь есть несколько моментов, которые необходимо рассмотреть.

В нашем примере мы передаем параметры по позиции, т. е. значение 3 присваивается первому параметру (`nItem1`), а значение 2 — второму (`nItem2`). Однако параметры можно передавать и по имени:

```
nResult = fSum(nItem1 := 3, nItem2 := 2)
```

Обратите внимание, что, несмотря на то, что здесь выполняется вполне привычная операция — присвоение значений, оператор присваивания используется не совсем обычный — двоеточие со знаком равенства (:=), как в C++. При использовании знака равенства возникнет ошибка.

Конечно, вместо явной передачи значений (`fSum(3, 2)`) можно использовать переменные. Однако что произойдет с переменными после того, как они "побывают" в функции, если функция изменяет их значения? Останутся ли эти значения за пределами функции прежними или изменятся?

Все зависит от того, как именно передаются параметры: *по ссылке* (по умолчанию, можно также использовать ключевое слово `byRef`) или *по значению* (ключевое слово `byVal`).

Если параметры передаются по ссылке, то фактически в вызываемую процедуру передается ссылка на эту переменную в оперативной памяти. Если эту переменную в вызываемой процедуре изменить, то значение изменится и в вызывающей функции. Это принятое в VBA поведение по умолчанию.

Если параметры передаются по значению, то фактически в оперативной памяти создается копия этой переменной и в вызываемую процедуру передается эта копия. Конечно же, чтобы вы не сделали с копией, на исходную переменную это никак не повлияет и в вызывающей процедуре не отразится.

Продемонстрировать разницу можно на простом примере:

```
Private Sub TestProc()
    'Объявляем переменную nPar1 и присваиваем ей значение
    Dim nPar1 As Integer
    nPar1 = 5

    'Передаем ее как параметр nItem1 функции fSum
    MsgBox fSum(nItem1:=nPar1, nItem2:=2)

    'А теперь проверяем, что стало в нашей переменной nPar1
    'после того, как она побывала в функции fSum
    MsgBox nPar1
End Sub

Function fSum(nItem1 As Integer, nItem2 As Integer)
    'Используем значение переменной
    fSum = nItem1 + nItem2
    'А затем ее меняем!
    nItem1 = 10
End Function
```

Проверьте, что будет, если поменять строку объявления функции:

```
Function fSum(nItem1 As Integer, nItem2 As Integer)
```

на другую:

```
Function fSum(byVal nItem1 As Integer, nItem2 As Integer)
```

Можно продемонстрировать компилятору VBA, что значение, возвращаемое функцией, нас совершенно не интересует. Для этого достаточно не заключать ее параметры в круглые скобки. Например, для нашей функции это может выглядеть так:

```
fSum 3, 2
```

Такой код будет работать совершенно нормально. Однако если нам потребуется все-таки узнать, что возвращает функция, то придется передаваемые ей параметры заключать в круглые скобки:

```
nResult = fSum(3, 2)
```

Для многих встроенных функций компилятор VBA в принципе не дает возможности игнорировать возвращаемое значение, заставляя помещать параметры в круглые скобки и принимать возвращаемое значение. В противном случае он сообщает о синтаксической ошибке.

3.8.5. Вызов и завершение работы процедур

С примерами вызова процедур и функций из кода мы уже сталкивались. Для этого достаточно записать имя процедуры или функции, передать ей необходимые параметры, а для функции еще и принять возвращаемое значение. Одни процедуры можно вызывать из других процедур. Но как дать пользователю возможность запустить самую первую процедуру?

В нашем распоряжении следующие возможности:

- создать макрос (т. е. специальную процедуру, не принимающую параметров, в модуле **NewMacros**) и запустить его по имени, кнопке или комбинацией клавиш *(см. разд. 1.4)*. Макрос затем может вызывать и другие процедуры;
- создать форму и воспользоваться набором событий этой формы и элементов управления на ней (об этом будет рассказано в *гл. 5*) или просто элементом управления на листе Excel или документе Word;
- назначить процедуре специальное имя (`AutoExec()`, `AutoNew()` и т. п.). Полный список таких специальных имен можно посмотреть в документации. Правда, поскольку раньше эти возможности активно использовались вирусами, в Office 2003 по умолчанию эти макросы запускаться не будут. Для того чтобы обеспечить им (и многим другим макросам) возможность

запуска, необходимо изменить установленный уровень безопасности в меню **Сервис | Макрос | Безопасность** или обеспечить цифровые подписи для ваших макросов;

- вместо специального имени для макроса использовать событие: например, событие запуска приложения, событие открытия документа и т. п. Это рекомендованный Microsoft способ обеспечения автоматического запуска программного кода. Подробнее про работу с событиями будет рассказано в *гл. 4* и *5*;
- можно запустить приложение из командной строки с параметром `/m` и именем макроса, например:

  ```
  winword.exe /mMyMacros
  ```

  Очень удобно в этом случае использовать ярлыки, в которых можно указать этот параметр запуска.

В VBA вполне допустима ситуация, когда функция запускает на выполнение саму себя. Однако подобных вызовов лучше избегать (по возможности заменяя их на циклы). Причина — проблемы с читаемостью и возможное (в случае бесконечного запуска) исчерпание оперативной памяти (переполнение стека), чреватое серьезными системными ошибками.

Для завершения выполнения процедуры в VBA предусмотрены конструкции `End` и `Exit`. Синтаксис их очень прост:

```
Exit Sub
End Sub
```

Делают они одно и то же — завершают работу текущей процедуры. Однако используются в разных ситуациях:

- оператор `End` — это завершение работы процедуры после того, как все сделано. После оператора `End` код процедуры заканчивается;
- оператор `Exit` — это немедленное завершение работы функции в ходе ее работы. Обычно помещается в блок оператора условного перехода, чтобы произвести выход, как только выяснилось, что функции по каким-то причинам дальше выполняться не нужно (например, дальше идет код обработчика ошибок).

Задание для самостоятельной работы 3.4: Работа с процедурами и функциями

ЗАДАНИЕ:

1. Создайте в модуле **NewMacros** шаблона Normal.dot новую функцию `fMultiply()`, которая бы:

- принимала в качестве входящих параметров два числа;
- перемножала их и возвращала полученное значение.

2. Создайте в модуле **NewMacros** шаблона Normal.dot новую процедуру `AutoNew()` со следующим кодом:

```
Public Sub AutoNew()
    Dim nMult1 As Integer
    Dim nMult2 As Integer
    Dim nResult As Integer
    nMult1 = CInt(InputBox("Введите первое число: "))
    nMult2 = CInt(InputBox("Введите второе число: "))
    nResult = 10
    Selection.InsertAfter nResult
    Selection.Collapse wdCollapseEnd
End Sub
```

3. Измените процедуру `AutoNew()` таким образом, чтобы она передавала значения переменных `nMult1` и `nMult2` функции `fMultiply()` и принимала от нее значение для переменной `nResult` (это значение должно использоваться вместо жестко определенного значения `10`).
4. Создайте в Word новый документ и убедитесь, что созданные вами процедуры и функции работают правильно.
5. Чтобы созданная вами процедура `AutoNew()` не мешала дальнейшей работе, закомментируйте весь ее код.

Ответ к заданию 3.4

1. Запустите Word и нажмите клавиши <Alt>+<F11>. В окне **Project Explorer** раскройте узел **Normal | Modules** и щелкните два раза левой кнопкой мыши на строке **NewMacros**.
2. Вставьте в модуль **NewMacros** следующие строки для функции `fMultiply()`:

```
Public Function fMultiply(nM1 As Integer, nM2 As Integer)
    fMultiply = nM1 * nM2
End Function
```

3. Код для процедуры `AutoNew()` может выглядеть так (измененный код выделен полужирным):

```
Public Sub AutoNew()
    Dim nMult1 As Integer
    Dim nMult2 As Integer
```

```
    Dim nResult As Integer
    nMult1 = CInt(InputBox("Введите первое число: "))
    nMult2 = CInt(InputBox("Введите второе число: "))
    nResult = fMultiply(nMult1, nMult2)
    Selection.InsertAfter nResult
    Selection.Collapse wdCollapseEnd
End Sub
```

4. Для того чтобы закомментировать код `AutoNew()`, выделите весь код этой процедуры (включая `Public Sub AutoNew()` и `End Sub`) и нажмите кнопку **Comment Block** на панели инструментов **Edit**.

3.9. Встроенные функции языка VBA

3.9.1. Что такое встроенные функции

В языке программирования VBA предусмотрено несколько десятков *встроенных функций*. Они доступны в любой программе на языке VBA, при этом безразлично, в среде какого программного продукта мы находимся — Excel, Word, Access или, к примеру, AutoCAD. Используются они очень активно, и во многих ситуациях без них не обойтись. Профессиональные программисты применяют их совершенно автоматически, а обычным пользователям хочется посоветовать потратить несколько часов на знакомство с ними, потому что без знания этих функций эффективно работать в VBA не получится. Дополнительным аргументом в пользу их изучения является то, что практически идентичный набор функций есть в обычном Visual Basic и VBScript, а многие из этих функций с теми же названиями и синтаксисом встречаются и в других языках программирования — C++, Delphi, Java, JavaScript и т. п.

В справке по VBA встроенные функции сгруппированы по буквам (рис. 3.2).

Многие слушатели на курсах задавали вопрос: а нет ли справки по этим функциям на русском языке? К сожалению, такой справки мне найти не удалось, поэтому попытаюсь привести краткую справку в этой книге. Далее будет рассказано про большинство активно используемых функций языка VBA (математические функции, такие как косинус или тангенс, которые в практической работе почти не используются, и финансовые функции мы рассматривать не будем). Полный синтаксис функций для экономии места приводиться не будет: главное — понимание, что делает каждая функция и в каких ситуациях ее можно использовать.

Функции в следующих разделах сгруппированы по своей функциональности. Если нужно найти информацию просто по имени функции, можно воспользоваться предметным указателем в конце книги.

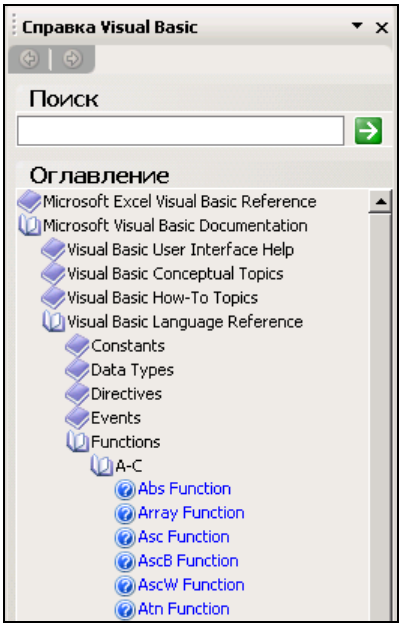


Рис. 3.2. Справка по встроенным функциям

3.9.2. Функции преобразования и проверки типов данных

В программах на VBA очень часто приходится преобразовывать значения из одного типа данных в другой. Приведу несколько типичных ситуаций, когда этим приходится заниматься:

- преобразование из строкового значения в числовое при приеме значения от пользователя через `InputBox()`;
- преобразование значения даты/времени в строковое, когда нам нужно отобразить дату или время единообразно вне зависимости от региональных настроек на компьютерах пользователей;
- преобразование значения из строкового в дату/время для применения специальных функций даты/времени.

Чаще всего для конвертации типов данных используются функции, имя которых складывается из префикса `'C'` (от слова *Convert*) и имени типа данных. Перечень этих функций следующий: `CBool()`, `CByte()`, `CCur()`, `CDate()`, `CDbl()`, `CDec()`, `CInt()`, `CLng()`, `CSng()`, `CStr()`, `CVar()`, `CVDate()`, `CVErr()`.

Просмотреть, что в итоге получилось, можно при помощи функции `TypeName()`, которая возвращает имя используемого типа данных, например:

```
nVar1 = CInt(InputBox("Введите значение"))
MsgBox TypeName(nVar1)
```

В данном случае эта функция вернет "Integer".

Кроме того, существует еще несколько полезных для конвертации функций.

- Str() — позволяет перевести числовое значение в строковое. Делает почти то же самое, что и CStr(), но при этом вставляет пробел перед положительными числами.
- Val() — "вытаскивает" из смеси цифр и букв только числовое значение. При этом функция читает данные слева направо и останавливается на первом нечисловом значении (допускается единственное нечисловое значение — точка, которая будет отделять целую часть от дробной). Очень удобно, когда у нас вперемежку с числовыми данными прописываются единицы измерения или валюта.
- IsNumeric() и IsDate() — проверяют значения на соответствие, чтобы не возникло ошибок при конвертации. Для проверки на соответствие специальным значениям можно использовать функции IsArray(), IsEmpty(), IsError(), IsMissing(), IsNull() и IsObject(). Все эти функции возвращают True или False в зависимости от результатов проверки переданного им значения.
- Hex() и Oct() — преобразовывают десятичные данные в строковое представление шестнадцатеричных и восьмеричных значений.

3.9.3. Строковые функции

Это наиболее часто используемые функции. Требуются они постоянно, и необходимо знать их очень хорошо.

- Asc() — эта функция позволяет вернуть числовой код для переданного символа. Например, Asc("D") вернет 68. Эту функцию удобно использовать для того, чтобы определить следующую или предыдущую букву. Обычно она используется вместе с функцией Chr(), которая производит обратную операцию — возвращает символ по переданному его числовому коду. Например, такой код в Excel позволяет написать в ячейки с A1 по A20 последовательно буквы русского алфавита от А до У:

```
Dim n, nCharCode As Integer
n = 1
nCharCode = Asc("A")
Do While n <= 20
    ActiveWorkbook.ActiveSheet.Range("A" & n).Value = Chr(nCharCode)
```

```
    n = n + 1
    nCharCode = nCharCode + 1
Loop
```

Варианты этой функции — `AscB()` и `AscW()`. `AscB()` возвращает только первый байт числового кода для символа, а `AscW()` возвращает код для символа в кодировке Unicode.

- `Chr()` — возвращает символ по его числовому коду. Помимо того, что используется в паре с функцией `Asc()` (см. предыдущий пример), без нее не обойтись еще в одной ситуации: когда нужно вывести служебный символ. Например, нам нужно напечатать в Word значение `"Газпром"` (в кавычках). Кавычка — это служебный символ, и попытка использовать строку вида:

  ```
  Selection.Text = ""Газпром""
  ```

  приведет к синтаксической ошибке. А вот так все будет в порядке:

  ```
  Selection.Text = Chr(34) & "Газпром" & Chr(34)
  ```

  Есть варианты этой функции — `ChrB()` и `ChrW()`. Работают аналогично таким же вариантам для функции `Asc()`.

- `InStr()` и `InStrRev()` — одни из самых популярных функций. Позволяют обнаружить в теле строковой переменной последовательность символов и вернуть ее позицию. Если последовательность не обнаружена, то возвращается 0. Функция `InStr()` ищет с начала строки, а `InStrRev()` — с конца.
- `Left()`, `Right()`, `Mid()` — позволяют взять указанное вами количество символов из существующей строковой переменной слева, справа или из середины соответственно.
- `Len()` — возвращает число символов в строке (длину строки). Часто используется с циклами, операциями замены и т. п.
- `LCase()` и `UCase()` — переводят строку в нижний и верхний регистры соответственно. Часто используются для подготовки значения к сравнению, когда регистр не важен (фамилии, названия фирм, городов и т. п.).
- `LSet()` и `RSet()` — заполняют одну переменную символами другой без изменения ее длины (соответственно слева и справа). Лишние символы обрезаются, на место недостающих подставляются пробелы.
- `LTrim()`, `RTrim()`, `Trim()` — убирают пробелы соответственно слева, справа или и слева, и справа.
- `Replace()` — заменяет в строке одну последовательность символов на другую.
- `Space()` и `String()` — возвращают строку из указанного вами количества пробелов или символов соответственно. Обычно используются для форма-

тирования вывода совместно с функцией `Len()`. Еще одна похожая функция — `Spc()`, которая используется для форматирования вывода на консоль. Она размножает пробелы с учетом ширины командной строки.

- `StrComp()` — сравнивает две строки.
- `StrConv()` — преобразует строку (в Unicode и обратно, в верхний и нижний регистры, первую букву слов заглавной и т. п.).
- `StrReverse()` — "переворачивает" строку, разместив ее символы в обратном порядке.
- `Tab()` — еще одна функция, которая используется для форматирования вывода на консоль. Размножает символы табуляции в том количестве, в котором вы укажете. Если никакое количество не указано, просто вставляет символ табуляции. Для вставки символа табуляции в строковое значение можно также использовать константу `vbTab`.

3.9.4. Функции для работы с числовыми значениями

Функций для работы с числовыми значениями в VBA очень много. Используются они реже, чем строковые функции, но во многих ситуациях без них не обойтись.

Еще один момент: если вы программируете на языке VBA, то, скорее всего, на вашем компьютере установлен Microsoft Office с Excel. В Excel есть свой собственный мощный набор встроенных функций для работы с числовыми значениями, которые вполне доступны из VBA. Если вы в моем списке не найдете ничего подходящего для вашей ситуации, возможно имеет смысл воспользоваться функциями Excel.

Кроме того, если в окне **Надстройки** (меню **Сервис | Надстройки**) установить флажок напротив строки **Пакет анализа**, в Excel будет добавлен дополнительный набор аналитических научных и финансовых функций, а если в том же окне установить флажок напротив **Analysis ToolPak — VBA**, то эти функции станут доступны из VBA (только внутри Excel, в котором установлена эта надстройка).

Далее приведены только универсальные функции VBA для работы с числовыми значениями. Эти функции доступны из любых приложений VBA.

- `Abs()` — эта функция возвращает абсолютное значение переданного ей числа (то же число, но без знака). Например, `Abs(3)` и `Abs(-3)` вернут одно и то же значение 3. Обычно используется тогда, когда нам нужно определить разницу между двумя числами, но при этом мы не знаем, какое число — первое или второе — больше. Результат вычитания может быть и

положительным и отрицательным. Чтобы он был только положительным, используется эта функция.

- `Int()`, `Fix()` и `Round()` — позволяют по-разному округлять числа. `Int()` возвращает ближайшее меньшее целое, `Fix()` отбрасывает дробную часть, `Round()` округляет до указанного количества знаков после запятой. При этом `Round()` работает не совсем правильно, в чем легко убедиться:

  ```
  MsgBox Round(2.505, 2)
  ```

  Поэтому на практике для округления лучше использовать `Format()` *(см. разд. 3.9.6)*:

  ```
  MsgBox Format(2.505, "#,##0.00")
  ```

- `Rnd()` и команда `Randomize` — используются для получения случайных значений (очень удобно, например, при генерации имен файлов). Обычный синтаксис при применении `Rnd()` выглядит так:

  ```
  случайное_число = Int(минимум + (Rnd() * максимум))
  MsgBox (Int(1 + (Rnd() * 100)))
  ```

  Однако перед вызовом функции `Rnd()` необходимо выполнить команду `Randomize` для инициализации генератора случайных чисел.

- `Sgn()` — позволяет вернуть информацию о знаке числа. Возвращает 1, если число положительное, –1, если отрицательное, и 0, если проверяемое число равно 0.

3.9.5. Функции для работы с датой и временем

Без функций даты и времени обычно обойтись очень сложно. Самые важные функции VBA для работы с датой/временем приведены далее.

- `Date()`, `Time()`, `Now()` — возвращают соответственно текущую системную дату, текущее системное время и дату и время одновременно. Установить их можно при помощи одноименного соответствующего оператора, например, так:

  ```
  Date = #5/12/2004#
  ```

- `DateAdd()` — добавляет к дате указанное количество лет, кварталов, месяцев и так далее до секунд.
- `DateDiff()` — возвращает разницу между датами (в единицах от лет до секунд).
- `DatePart()` — очень важная функция, которая возвращает указанную вами часть даты (например, только год, только месяц или только день недели).

- `DateSerial()` — создает значение даты на основе передаваемых символьных значений. То же самое делает функция `DateValue()`, но при другом формате принимаемых значений. Аналогичным образом (для времени) работают `TimeSerial()` и `TimeValue()`.
- `Day()` (а также `Year()`, `Month()`, `Weekday()`, `Hour()`, `Minute()`, `Second()`) — специализированные заменители функции `DatePart()`, которые возвращают нужную вам часть даты.
- `MonthName()` — возвращает имя месяца словами по его номеру. Возвращаемое значение зависит от региональных настроек. Если они русские, то вернется русское название месяца.
- `Timer()` — возвращает количество секунд, прошедших с полуночи.

Если нужно получить дополнительные возможности работы с датой/временем, то в вашем распоряжении объектная модель `Outlook`. Например, при помощи ее можно получить информацию о праздниках и рабочих/нерабочих днях большинства стран мира. Подробнее — в *гл. 13*.

3.9.6. Функции для форматирования данных

Для форматирования данных в вашем распоряжении функция `Format()` и целый набор функций, которые начинаются с префикса `Format...` (`FormatNumber()`, `FormatCurrency()`, `FormatDateTime()` и т. п.) Синтаксис функции `Format()` выглядит так:

```
Format(выражение, "формат")
```

Эта функция принимает *выражение* и форматирует его в соответствии с параметром *формат*.

Несколько примеров использования `Format()` (посмотрите сами, что получится):

```
Format(15/20, "Percent")
Format(Date, "Long Date")
Format(1, "On/Off")
Format(334.9, "###0.00")
Format("Просто текст", ">" )
```

Для остальных функций `Format...` то, что они делают, понятно из названий.

Особая ситуация — когда нужно, чтобы дата отображалась на компьютерах пользователей единообразно вне зависимости от региональных настроек. В качестве решения можно использовать функцию `DatePart()`, при помощи ее перевести дату "по частям" в текстовый формат и склеить нужным образом.

3.9.7. Функции для организации взаимодействия с пользователем

Во многих программах VBA необходимо обеспечить взаимодействие с пользователем — проинформировать его о чем-то и, возможно, получить от него ответную реакцию. В принципе, для пользователя можно просто вывести текст в окне приложения (например, в текущем документе Word) или воспользоваться формой и элементами управления. Как это делается — мы узнаем в *гл. 5*, посвященной работе с формами и элементами управления, и в *гл. 10—15*, в которых речь пойдет об объектных моделях приложений Office. В этом разделе мы рассмотрим только применение для этой цели встроенных функций VBA.

Самой простой способ вывести информацию пользователю — воспользоваться встроенной функцией VBA `MsgBox()`. Примеров применения этой функции в нашей книге уже было множество, а полный ее синтаксис выглядит так:

```
MsgBox(Текст [, кнопки] [, заголовок_окна] [, файл_справки,
метка_в_файле_справки])
```

Возможностей у `MsgBox()` достаточно много:

- можно отображать разное количество кнопок (**OK**, **Cancel**, **Abort**, **Retry**, **Ignore**, **Yes**, **No**);
- можно показывать символы *Critical* (красный круг с крестом), *Exclamation* (восклицательный знак в желтом треугольнике), *Question* (вопросительный знак), *Information* (буква "I");
- можно выбирать кнопку по умолчанию;
- можно делать окно модальным или обычным.

В зависимости от того, на какую кнопку нажал пользователь, функция возвращает соответствующее значение (всего 7 вариантов). Подробнее читайте в справке по VBA. Пример возврата значения от `MsgBox()` может быть таким:

```
Dim nVar As Integer
nVar = MsgBox ("Будем делать?", vbInformation + vbOKCancel, _
               "Демонстрационное окно сообщения")
```

Если значение `nVar` равно 1, то пользователь нажал **OK**, если 2, то **Отмена** (Cancel).

Иногда (например, при пакетной обработке данных) хотелось бы, чтобы окно сообщения через некоторое время закрывалось само собой. Это можно сделать при помощи метода `Popup()` объекта `Wscript.Shell`. Для этого в проект через меню **Tools | References** нужно добавить ссылку на Windows Script Host Object Model, а после этого использовать следующий код:

```
Dim oShell As New WshShell
oShell.Popup "Test", 5
```

В остальном функциональность получившего окна одинакова с `MsgBox()`. Код возврата, если пользователь не нажал ни на какую кнопку, равен –1.

Самый простой способ принять информацию от пользователя — воспользоваться функцией `InputBox()`. Все очень просто:

```
Dim InputName
InputName = InputBox("Введите Ваше имя")
MsgBox ("Вы ввели: " & InputName)
```

Для `InputBox()` можно указать текст приглашения, заголовок окна, значение по умолчанию, местонахождение окна и файл справки. Не забывайте, что все вводимое пользователем `InputBox()` автоматически переводит в тип данных `String`, может потребоваться преобразование.

Можно привлечь внимание пользователя звуковым сигналом. Для этой цели используется оператор `Beep`:

```
Dim i
For i = 1 To 3
    Beep
Next i
```

Если нужно обеспечить более сложное взаимодействие с пользователем, можно подумать о применении формы, возможностей самого документа Office или помощника (Office Assistant). Очень мощные возможности обеспечивает и применение объектной модели Internet Explorer.

3.9.8. Функции — заменители синтаксических конструкций

В VBA предусмотрено несколько функций, которые позволяют заменять синтаксические конструкции условного перехода, например `If...Then...Else` или `Select Case`. Каких-то преимуществ применение этих функций не дает (может быть, код станет на несколько строчек короче), но профессиональные программисты очень любят их использовать.

Начинающим программистам рекомендуется применять обычные синтаксические конструкции, чтобы не путаться. Однако для чтения чужого кода необходимо знать и эти функции.

- `Choose()` — принимает число (номер значения) и список значений. Возвращает значение из списка, порядковый номер которого соответствует передаваемому числу.

Например, вызов функции:

```
Choose(2, "Первый", "Второй", "Третий")
```

вернет "Второй".

- `IIf()` — расшифровывается как *Immediate If*, т. е. "немедленный *If*". Представляет собой упрощенный вариант `If...Else`, когда проверяется условие и возвращается одно из двух значений. Например:

```
IIf(n > 10, "Больше десяти", "Меньше или равно десяти")
```

- `Switch()` — принимает неограниченное количество пар типа "выражение = значение", проверяет каждое выражение на истинность и возвращает значение для первого выражения, которое оказалось истинным. Например:

```
Function Language (CityName As String)
    Language = Switch(CityName = "Москва", "русский", CityName = _
        "Париж", "французский", CityName = "Берлин", "немецкий")
End Function
```

3.9.9. Функции для работы с массивами

Как уже говорилось, при программной работе с приложениями Microsoft Office массивы используются редко. Вместо них применяются коллекции. Однако в VBA предусмотрены возможности и для работы с массивами.

- `Array()` — позволяет автоматически создать массив нужного размера и типа и сразу загрузить в него переданные значения:

```
Dim myArray As Variant
myArray = Array(10,20,30)
MsgBox A(2)
```

- `Filter()` — позволяет на основе одного массива получить другой, отфильтровав в исходном массиве нужные нам элементы.
- `LBound()`, `UBound()` — возвращают соответственно информацию о нижней границе массива (номер первого имеющегося в массиве значения) и о верхней границе (номер последнего имеющегося значения).
- `Join()` — соединяет множество строк, составляющих массив, в одну строковую переменную. В качестве разделителя по умолчанию используется пробел, но можно указать и свой разделитель. Обратная функция, создающая массив из одной строки, — `Split()`. Эти функции очень удобны, например, при обработке значений, полученных из базы данных, электронной таблицы, макетного файла и т. п.

3.9.10. Функции для работы с файловой системой

В VBA предусмотрен набор встроенных функций для выполнения различных операций с файлами, каталогами, дисками и прочими объектами файловой системы. Информация об этих функциях приведена далее. Но не забывайте, что помимо этих функций (общих для всех приложений, в которых используется VBA) у нас есть, во-первых, возможности, специфические для данного приложения (например, открытие и сохранение документа Word средствами объектной модели Word). Во-вторых, на любом компьютере под управлением Windows есть объектная библиотека Microsoft Scripting Runtime, очень простая и удобная для выполнения различных операций с файлами, каталогами и дисками. Можно добавить в проект VBA ссылку на нее и использовать все имеющиеся в ней возможности. Если, к примеру, мне нужно пройтись по всем файлам в данном каталоге и что-нибудь с ними сделать (например, загрузить в Excel все файлы отчетов, которые пришли из филиалов), я использую именно эту библиотеку. Справку по ней можно найти на сайте Microsoft (**www.microsoft.com/scripting**).

Далее приведены встроенные функции для работы с файловой системой, предусмотренные в VBA.

- `CurDir()` — функция, которая возвращает путь к текущему каталогу, в котором будут сохраняться файлы вашего приложения по умолчанию.
- `Dir()` — позволяет искать файл или каталог по указанному пути на диске.
- `EOF()` — при операции чтения или записи в файл на диске эта функция вернет `True`, если вы находитесь в конце файла.
- `Error()` — позволяет вернуть описание ошибки по ее номеру. Генерировать ошибку нужно при помощи метода `RaiseError()` специального объекта `Err` (*см. гл. 6*, в которой рассказывается про перехват ошибок и отладку).
- `FileAttr()` — позволяет определить, как именно был открыт вами файл в файловой системе: на чтение, запись, добавление, в двоичном или текстовом режиме и т. п.
- `FileDateTime()` — позволяет получить информацию о последнем времени обращения к указанному вами файлу. Если к файлу после создания ни разу не обращались, то функция вернет время создания файла.
- `FileLen()` — возвращает длину указанного вами файла в байтах.
- `FreeFile()` — позволяет определить следующую свободную цифру, которую можно использовать как номер файла при его открытии.
- `GetAttr()` — позволяет обратиться к файлу и получить информацию о его атрибутах (скрытый, доступен только для чтения, архивный и т. п.).

- `Input()` — позволяет считать информацию из открытого файла. Например, считать информацию из файла C:\text1.txt и вывести ее в окно сообщений можно так:

```
Dim MyChar
'Открываем файл функцией Open() на чтение
Open "c:\text1.txt" For Input As #1
Do While Not EOF(1)   ' Пока файл не кончился,
    ' получаем по одному символу и добавляем его к предыдущим
    MyChar = MyChar & Input(1, #1)
Loop
Close #1          'Закрываем файл
MsgBox MyChar     'Выводим его содержание в окно сообщения
```

  Вариант этой функции — `InputB()` — позволяет указать количество байт, которые надо считать из файла.

- `Loc()` — от *Location* (местонахождение) — возвращает число, которое определяет текущее место вставки или чтения в открытом файле. Похоже работает функция `Seek()`, но она возвращает информацию о позиции, с которой будет выполняться следующая операция чтения или вставки.

- `LOF()` — от *length of file* — позволяет определить длину открытого файла в байтах.

`Open` — это не функция, а команда VBA, но без нее операции чтения и записи с файлами на диске не произвести. Справку по ней можно найти по словосочетанию "Open Statement". Как минимум, ей нужно передать имя открываемого файла, режим открытия и номер файла (номер файла — это его идентификатор для передачи другим функциям, его назначаете вы сами). Например, чтобы открыть файл на чтение с возможностью одновременного обращения к нему других пользователей, можно использовать код вида:

```
Open "c:\file1.txt" For Output Shared As #1
```

3.9.11. Другие функции VBA

В этот раздел попали те встроенные функции языка VBA, которые я не смог отнести ни к одной другой категории.

- `DoEvents()` — это очень важная функция. Она позволяет на время отвлечься от выполнения какой-то операции VBA и передать управление операционной системе, чтобы обработать накопившиеся в операционной системе события (например, нажатия клавиш пользователем). После этого продолжение операции VBA продолжается. Если у вас выполняется очень длительная операция (поиск на дисках, обработка большого объема дан-

ных и т. п.) и вы хотите дать пользователю возможность быстро прервать эту операцию, можно выполнять эту команду, например, каждый раз после обработки определенной "порции" данных.

- `Environ()` — возвращает абсолютный путь для переменных окружения компьютера (полный список переменных, доступных на вашем компьютере, можно просмотреть, если в командной строке выполнить команду `SET`). Например, вам нужно записать что-то в файл во временном каталоге. Абсолютный путь к временному каталогу на вашем компьютере можно получить так:

```
MsgBox Environ("TEMP")
```

- `GetAllSettings()` — позволяет получить (в виде двумерного массива) из реестра все параметры, которые относятся к указанному вами приложению. Функция `SaveSetting()` позволяет записать информацию в реестр, а `DeleteSetting()` — удалить. `GetSetting()` позволяет получить информацию об определенном параметре. Замечу, что эти методы позволяют обращаться только к одному очень далекому уголку реестра в ветви `HKEY_CURRENT_USERS`. Обращаться к другим параметрам реестра при помощи этих методов бесполезно. Рекомендую для работы с реестром использовать объектную библиотеку Windows Script Host Object Model, которая также есть на любом компьютере под управлением Windows 2000, XP и 2003. Нужный объект называется `WSHShell`, методы — `RegRead()`, `RegWrite()` и `RegDelete()`. Справку по объектам этой библиотеки можно найти на сайте Microsoft (**www.microsoft.com/scripting**).
- `Partition()` — определяет, к какому диапазону из наборов значений относится переданное вами число, и возвращает описание этого диапазона (в виде строки). Обычно используется при выполнении запросов к базам данных.
- `QBColor()` — позволяет перевести обозначение цвета из старого номерного обозначения с возможными 16 значениями в RGB-код, который понимает VBA. Обычно используется при исправлении старых унаследованных программ.
- `RGB()` — еще одна функция для работы с цветом. Позволяет вернуть RGB-код, который можно использовать для присвоения цвета, приняв три значения для цветов: красного (Red), зеленого (Green) и синего (Blue). Значение для каждого из основных цветов могут варьироваться от 0 до 255. Например, самый зеленый из возможных цветов получится, если переданные этой функции значения будут выглядеть как `RGB(0, 255, 0)`.
- `Shell()` — позволяет запустить из VBA внешний программный файл и вернуть информацию о его Program ID в операционной системе. Обычно

применяется опытными разработчиками при использовании ими в программах возможностей Windows API. С практической точки зрения эту функцию можно использовать для запуска любых внешних программ из вашего приложения, хотя, с моей точки зрения, применение специальных объектов `WshShell` и `WshExec` из библиотеки Windows Script Host Object Model удобнее (можно передавать в окно клавиатурные комбинации, принимать и передавать значения через командную строку и т. п.). Эта библиотека есть на любом компьютере Windows, справку по ней можно найти на сайте **www.microsoft.com/scripting**.

- `TypeName()` — функция, которая возвращает имя типа данных для переданной ей переменной. Очень удобна для определения типа данных для значения, полученного из базы данных или путем вызова метода какого-то объекта.
- `VarType()` — делает почти то же самое, но вместо имени возвращает числовой код, который обозначает тип данных. Можно использовать для программных проверок типов данных для переменных.