Синтаксис и программные конструкции VBA

Содержание

1 Основы синтаксиса языка VBA

Главные правила синтаксиса VBA

Мы подошли к теме, которым многим пользователям на курсах кажется самой скучной: синтаксису 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 знания их не удастся читать интересные книги или производить важные вычисления. Кроме того, VBA изначально проектировался и создавался как язык программирования, максимально дружелюбный по отношению к пользователям, которые не являются профессиональными программистами.

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

Теперь — немного про 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. Некоторые основные синтаксические принципы этого языка:

- 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 умещается 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:

MsgBox "Сообщение пользователю" _

& vUserName

2 Операторы VBA

Операторы VBA: арифметические, логические, сравнения, присвоения

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 присвоения в VBA — знак равенства. Можно записывать так:

Let nVar = 10

а можно еще проще:

nVar = 10

Во втором случае не путайте знак равенства с оператором равенства.

Выражение

nVar = 10

значит "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 (> и <), например, *If (nVar > 10)*;
- больше или равно и меньше или равно (>= и <=), например, *If (nVar >= 10)*;
- не равно (<>), например, *If(nVar<>10)*;
- сравнение объектов (Is). Определяет, ссылаются объектные переменные на 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 или false — true, если утверждение истинно, и false, если ложно.

Немного про сравнение строковых значений:

- при сравнении строковых значений регистр учитывается;
- 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 можно воспользоваться командой

Option Compare Text

Чуть подробнее про оператор Like. Общий его синтаксис выглядит как

Выражение1 Like Выражение2

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

Табл. 3.1 Подстановочные символы для оператора LIKE

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

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, возвращает FALSE, если E1 = TRUE и E2 = FALSE, иначе — TRUE.

Помнить нужно про AND, OR, NOT, остальные логические операторы используются редко.

Почти в любой программе VBA используются операторы конкатенации. В VBA их два — + или &. Рекомендуется всегда использовать &, 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:

MsgBox "Сообщение пользователю" & vUserName

Порядок применения операторов можно регулировать при помощи круглых скобок.

3 Переменные и типы данных

Переменные VBA, объявление переменных, Option Explicit, правила именования, типы данных VBA, исходные значения переменных

Переменные — 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
```

Как расшифровать 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, она будет доступна во всем модуле, если в процедуре — только на время работы этой процедуры;
- *Private* — при объявлении переменных в VBA значит то же, что и Dim;
- *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/Private;
- *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 есть 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 и 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 — далее);
- зарезервированные слова (те, которые подсвечиваются другим 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;
    - c (или сделать все буквы заглавными) — 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 подчеркивания:

COMPANY_NAME

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

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

В VBA предусмотрены следующие типы данных:

- *числовые* (byte — целое число от 0 до 255, integer — целое число от -32768 до 32767, long — большое целое число, currency (большое десятичное число с 19 позициями, включая 4 позиции после запятой), decimal (еще большее десятичное число с 29 позициями), single и double — значение с плавающей запятой (double в два 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.

- *строковые* (string переменной длины (до примерно 2 млрд символов) и фиксированной длины (до примерно 65400 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 еще 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).

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 данных во избежание возникновения ошибок;
- если есть возможность, лучше не использовать типы данных с плавающей запятой (single и double). Работа с такими типами данных производится медленнее, кроме того, могут быть проблемы при сравнениях за счет округлений;
- если есть возможность, лучше не пользоваться типом Variant. Этот тип все равно приводится VBA к одному из других типов, но памяти для него требуется больше. Кроме того, в ходе такого неявного образования могут возникнуть ошибки;
- при определении переменных можно использовать так называемые символы определения типа (% — integer, $ — String и т.п.). Например, в нашем примере нужно закомментировать строку Dim nVar 1 As Integer, а во второй строке написать:

nVar1% = nVar1% + 1

Такой подход является устаревшим и к использованию не рекомендуется.

При объявлении переменных можно и 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. Почему:

- сокращается количество ошибок: программа с самого начала откажется принимать в переменную значение неправильно 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 во втором случае.

Все опытные разработчики вообще запрещают использование переменных 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 редактора кода флажок **Require Variable Declarations** (меню **Tools** -> **Options**, вкладка **Editor**).

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

Dim n

n = n + 1

MsgBox n

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.

Хорошее правило — объявлять переменные заблаговременно, а не когда они потребовались. Это позволяет сделать программу более читаемой и четко спланированной.

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

Dim n1 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 даты/времени заключаются в "решетки" — символы фунта:

dVar1 = #05/06/2004#

Обратите внимание, что при присвоении значения даты/времени таким "явным способом" нам придется использовать принятые в США стандарты: 05 в данном случае — это месяц, 06 — день. Отображение же 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

Что содержится в переменных до присвоения им значений?

- В переменных всех числовых типов данных — 0.
- В строковых переменных переменной длины — "" (строка нулевой длины).
- В строковых переменных фиксированной длины — строка данной длины с символами ASCII 0 (эти символы на экран не выводятся).
- В Variant — пустое значение.
- В Object — ничто (нет ссылки ни на один из объектов).

4 Константы

Константы, объявление, ключевое слово Const, встроенные константы, vbCrLf

Константы — еще один контейнер для хранения данных, но, в отличие от переменных, они не изменяются в ходе выполнения VBA-программы. Для чего нужны константы:

- код становится лучше читаемым/убираются потенциальные ошибки;
- чтобы изменить какое-либо значение, которое много раз используется в программе (например, уровень налога) — это можно сделать один раз.

В VBA константы определяются при помощи ключевого слова Const:

Const COMP_NAME As String = “Microsoft”

При попытке в теле процедуры изменить значение константы 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 Reference -> Constants. Про одну из констант (она находится в разделе Miscellaneous) 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, которые мы будем рассматривать в последних разделах этого курса.

5 Операторы условного и безусловного перехода

5.1 Операторы условного и безусловного перехода. Оператор *If… Then... Else*

Проверка условий в VBA, оператор If...Then... Else, вложенные конструкции If

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 ложность 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... Else и Select Case.

Оператор *If... Then...* Else — самый популярный у программистов. Полный его синтаксис выглядит так:

If Условие Then
Команды1

[ElseIf Условия N Then
Команды N]

[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.

Оператор If…Then... Else применяется:

- когда нужно проверить на соответствие одному условию и в случае соответствия сделать какое-то действие:

```
If nTemperature < 10 Then

MsgBox "Одеть 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 "Одеть куртку"

Else

MsgBox "Одеть ветровку"

End If
```

- когда нужно проверить на соответствие нескольким условиям (обратите внимание на использование логических операторов):

```
If (nTemperature < 10) And (bRain = True) Then

MsgBox "Одеть куртку и взять зонтик"

End If
```

- когда в случае, если первая проверка вернула False, нужно проверить на соответствие еще нескольким условиям (в этом случае удобно использовать ElseIf):

If (bIGoInCar = True) Then

MsgBox "Одеться для машины"

ElseIf nTemperature < 10 Then

MsgBox "Одеть 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 …

Некоторые замечания по использованию If…Then... Else:

- 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 "Одеть 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

5.2 Оператор *Select Case*

Проверка условий в VBA, оператор Select Case

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

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

Некоторые замечания по поводу Select Case:

- строка Case "Понедельник" на самом деле означает Case sDayOfWeek = "Понедельник", просто такое равенство подразумевается по умолчанию. Но вам ничего не мешает использовать другой оператор сравнения или целый набор таких операторов:

Case 0 To 5, 15, Is > 55

MsgBox "Напомнить о прививках"

- Слово Is при этом можно пропустить — компилятор VBA добавит это ключевое слово за Вас. Несколько критериев в Case объединяются аналогично операторы OR — то есть выполнение пойдет по этой ветви, если тестируемое значение будет удовлетворять хотя бы одному из критериев. Критерии для сравнения разделяются запятыми.
- при использовании диапазона (0 To 5) включаются и границы диапазона (в данном случае 0 и 5).

5.3 Оператор *GoTo*

Оператор GoTo в VBA, ситуации применения GoTo

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

GoTo EngineNotStarted

...

EngineNotStarted :

MsgBox "Едем на метро"

...

EngineNotStarted: — это метка, для нее используется имя (выбираемое по правилам назначения имен для переменных), которое оканчивается на двоеточие.

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

6 Работа с циклами

Циклы VBA, конструкция For...Next, конструкция For Each...Next, выход из цикла по Exit For, конструкции Do While и Do Until, выход из цикла по Exit Do, конструкция While...Wend

Циклы используются в ситуациях, когда нам нужно выполнить какое-либо действие несколько раз. Первая ситуация — мы знаем, сколько раз нужно выполнить какое-либо 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 элементам 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 (то есть открытым рабочим книгам в Excel), для каждой книги проверяем ее имя, и если мы нашли книгу с именем Сводка.xls, то мы получаем на нее ссылку и выходим из цикла. Коллекция рабочих книг — это специальная коллекция, которая умеет производить поиск в себе по имени элементов, поэтому в принципе можно было обойтись такой строкой:

Set oMyWorkBook = Workbooks("Сводка.xls")

Но для многих других коллекций без конструкции For Each 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. Используются конструкции Do While…Loop и Do Until…Loop.

Конструкция Do While 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 не 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, завершить его или открыть ваш код в отладчике.

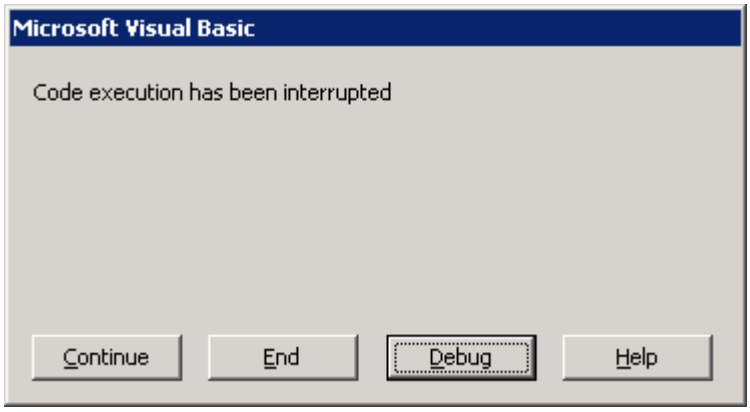

Рис. 3.1 Выполнение макроса остановлено по <Ctrl>+<Break>

Второй вариант — Do Until. 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:

While My Var < 10

MyVar = MyVar + 1

WScript.Echo "MyVar = " & MyVar

Wend

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

Такой массив может хранить три целочисленных элемента. 2 — это верхняя граница массива (upper bound). Количество элементов, которое может хранить массив, — от 0 до верхней границы включительно.

Если вам хочется, чтобы нумерация элементов в массиве 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.

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 вполне могут быть многомерными:

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 массиве может быть до 60 измерений.

Часто необходимы массивы динамические — те, размер которых можно изменять в ходе выполнения. Динамический массив объявляется следующим образом:

Dim MyArray () '- объявляем массив без верхней границы, эту строку можно ‘пропустить

ReDim MyArray (4) ' — изменяем размер массива

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

ReDim Preserve MyArray (7)

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

Массивы можно создавать и заполнять одновременно при помощи встроенной функции Array():

Dim MyArray

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

Указывать размер массива необязательно — он будет автоматически настроен в соответствии с кол-вом 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 кол-ва элементов используется функция UBound() (если массив одномерный или вас интересует размер первого измерения, то измерение передавать не надо):

UBound (имяМассива [, 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 — то есть 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) Подробнее про работу с коллекциями будет рассказано в главе 4 этой книги.

8 Процедуры и функции

8.1 Виды процедур

Процедуры (Sub) и функции (Function) VBA, объявление процедур и функций, макросы - специальный тип процедур VBA, типы процедур

Процедуры — это самые важные функциональные блоки языка VBA. В VBA вы можете выполнить только 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 (сообщений о ошибке при этом не выдается — просто этот код становится "невидим" для компилятора). Причина проста — в разделе объявлений модуля (когда в верхних списках показываются значения (General) и (Declarations) могут быть только объявления переменных уровня модуля и некоторые специальные инструкции для компилятора. Весь остальной программный код должен находится 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 просто процедура типа Sub, не имеющая параметров. Только макросы можно вызывать по имени из редактора VBA или приложения Office. Все другие процедуры нужно вызывать либо из других процедур, либо специальными способами, о которых будет рассказано ниже.

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

```
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 (об этом — ниже). 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 Test 1()

MsgBox Tomorrow()

End Sub

В VBA предусмотрены сотни встроенных функций (и гораздо большее число 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 ниже.

В VBA имеются также *процедуры обработки событий* (event procedure) — процедуры типа Sub специального назначения, которые выполняется в случае возникновении определенного события. Пример был приведен выше (Private Sub UserForm_Click()). Про события подробнее будет рассказано в модуле про формы и события.

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.

8.2 Область видимости процедур

Области видимости процедур VBA, объявления Public, Private, Static, команда Option Private Module

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 же 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. раздел про переменные). Пример:

Private Static Sub Farewell()

8.3 Объявление процедур

Объявление процедур в VBA

Объявить процедуру можно вручную, например, впечатав в коде строку

Private Sub Farewell ()

(редактор кода автоматически добавит строку *End Sub* и разделитель), а можно — на графическом экране, воспользовавшись меню **Insert** -> **Procedure**. Разницы не будет никакой.

8.4 Передача параметров

Передача параметров процедурам и функциям в VBA, необязательные (optional) параметры, передача по ссылке (ByRef) и по значению (ByVal), применение ссылок при передаче параметров

Параметры — значения, которые передаются от одной процедуры 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. Например, вот пример простой функции, которая складывает два числа и выводит результат:

Function fSum (nItem1 As Integer, nItem2 As Integer)

fSum = nItem1 + nItem2

End Function

Вызов ее 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, так: *MsgBox (fSum(3))*) приведет к ошибке "Argument not optional" — "Параметр не является необязательным". Чтобы можно было пропускать какие-то параметры, эти параметры можно 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, то есть значение 3 присваивается первому параметру (nItem1), а значение 2 — второму (nItem2). Однако параметры можно передавать и по имени:

nResult = fSum (nItem 1 := 3, nItem 2 := 2)

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

Конечно, вместо явной передачи значений (как у нас — 3 и 2) можно использовать переменные. Однако что произойдет с переменными после того, как они "побывают" в функции, если функция изменяет их значение? Останется ли это значение за пределами функции прежним или изменится?

Все зависит от того, как именно передаются параметры — *по ссылке* (по умолчанию, можно также использовать ключевое слово ByRef или *по значению* — нужно использовать 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 вызываемой процедуре передается эта копия. Конечно же, чтобы вы не сделали с этой 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 = nItem 1 + nItem 2

'А затем ее меняем!

nItem 1 = 10

End Function

Проверьте, что будет, если поменять строку объявления функции

Function fSum(nItem1 As Integer, nItem2 As Integer)

на следующую строку :

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

Можно продемонстрировать компилятору VBA, что то, что возвращает функция, наш совершенно не интересует. Для этого достаточно не заключать ее параметры в круглые скобки. Например, в случае со встроенной функцией MsgBox это может выглядеть так:

MsgBox "Test"

а для нашей функции —

fSum 3, 2

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

nTest = MsgBox("Test")

Для многих встроенных функций компилятор VBA в принципе не дает возможности игнорировать возвращаемое значение, заставляя помещать параметры в круглые скобки и принимать возвращаемое значение.

8.5 Запуск и завершение работы процедур

Способы запуска и завершения работы процедур в VBA, конструкция End Sub, конструкция Exit Sub

С примерами вызова процедур и функций из кода мы уже сталкивались:

nResult = fSum (3, 2)

Достаточно записать имя процедуры/функции, передать ей необходимые параметры, а для функции еще и принять необходимые значения. Одни процедуры можно вызывать из других процедур. Но как дать пользователю возможность запустить самую первую процедуру?

В нашем распоряжении следующие возможности:

- создать макрос (то есть специальную процедуру, не принимающую параметров, в модуле NewMacros) и запустить его по имени, кнопке или комбинации клавиш (см. раздел 1.4). Макрос затем может вызвать другие процедуры;
- создать форму и воспользоваться набором событий этой формы и элементов управления на ней (об этом рассказано в разделе, посвященном работе с формами и элементами управления), 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 Office2003 по умолчанию эти макросы запускаться не будут. Для того, чтобы обеспечить им (и многим другим макросам) возможность запуска, необходимо изменить установленный режим безопасности в меню **Сервис** -> **Макрос** -> **Безопасность**, или обеспечить подписи для ваших макросов (об этом будет говориться ниже).
- 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;
- можно запустить приложение из командной строки с параметром /m и именем макроса, например:

winword.exe /mMyMacros

Очень удобен в использовании ярлык Windows, в котором прописан этот параметр запуска.

В VBA вполне допустима ситуация, когда функция запускает на выполнение саму себя. Однако подобных ситуаций лучше избегать (по возможности заменяя их на циклы). Причина — проблемы с читаемостью и возможное (в случае бесконечного запуска функцией самой себя) 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 они 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 нельзя (например, дальше идет код обработчика ошибок).

9 Встроенные функции языка Visual Basic For Applications

9.1 Встроенные функции языка Visual Basic For Applications

Встроенные функции VBA, справка по встроенным функциям

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 знаниях этих функций эффективно работать в VBA не получится. Дополнительный аргумент в пользу их изучения: 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 (см. рис. 3.2).

Рис. 3.2 Справка по встроенным функциям

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 такую справку в этом курсе. Ниже будет рассказано про большинство активно используемых функций языка VBA (математические функции, которые в практической работе почти не используется, типа косинуса или тангенса, и финансовые функции мы рассматривать не будем). Полный синтаксис 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.

9.2 Функции преобразования и проверки типов данных

Встроенные функции преобразования и проверки типов данных VBA, функции CBool(), CByte(), CCur(), CDate(), CDbl(), CDec(), CInt(), CLng(), CSng(), CStr(), CVar(), CVDate(), CVErr(), Str(), Val(), IsNumeric(), IsDate(), IsEmpty(), IsError(), IsMissing(), IsNull(), IsObject(), IsArray(), Hex(), Oct()

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

- преобразование из строкового значение 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 выглядит как C (от слова Convert) + имя типа данных. Вот перечень этих функций: **CBool(), CByte(), CCur(), CDate(), CDbl(), CDec(), CInt(), CLng(), CSng(), CStr(), CVar(), CVDate(), CVErr()**. Просмотреть, что в итоге получилось, можно при помощи функции TypeName(), например:

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

MsgBox TypeName(nVar1)
```

Кроме того, 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 впереди для положительных чисел.

- *Val()* — "вытаскивает" из смеси цифр и букв только числовое значение. При этом эта 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.

Чтобы не возникло ошибок при конвертации, можно вначале проверять значения на возможность конвертации при помощи функций *IsNumeric()* и *IsDate()*. 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.

Для того, чтобы преобразовать десятичные данные в строковое представление шестнадцатеричных и восьмеричных значений, используются функции *Hex()* и *Oct()*. Для обратного преобразования специальных функций не предусмотрено, но вы можете указать компилятору VBA, что эти числа записаны в шестнадцатеричном или восьмеричном формате, записав их, например, как &O12 и &HA.

9.3 Строковые функции

Строковые функции VBA, Asc(), Chr(), InStr(), Len(), LCase(), UCase(), Replace(), Trim()

Это — наиболее часто используемые функции. Требуются они постоянно, и знать их необходимо 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()* — очень важная функция. Возвращает символ по его числовому коду. Помимо того, что используется в паре с функцией Asc() (см. пример по этой функции), 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""
```

сразу 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()* — одна из самых популярных функций. Позволяет обнаружить в теле строковой переменной последовательность символов и вернуть ее позицию. Если последовательность не обнаружена, то возвращается 0.
- *Left(), Right(), Mid()* — возможность взять указанное вами количество символов из существующей строковой переменной слева, справа или из середины соответственно.
- *Len()* — возможность получить число символов в строке. Часто используется с циклами, операциями замены и т.п.
- *LCase()* и *UCase()* — перевести строку в нижний и верхний регистры соответственно. Часто используется для подготовки значения к сравнению, когда при сравнении регистр не важен (фамилии, названия фирм, городов и т.п.).
- *LSet()* и *RSet()* — возможность заполнить одну переменную символами другой без изменения ее длины (соответственно слева и справа). Лишние символы обрезаются, на место недостающих подставляются пробелы.
- *LTrim(), RTrim(), Trim()* — возможность убрать пробелы соответственно слева, справа или и слева, и справа.
- *Replace()* — возможность заменить в строке одну последовательность символов на другую.
- *Space()* — получить строку из указанного вами количества пробелов; String() — получить строку из указанного вами количества символов (которые опять-таки указываются вами). Обычно используются для форматирования 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()* — возможность сравнить две строки.
- *StrConv()* — возможность преобразовать строку (в Unicode и обратно, в верхний и нижний регистр, сделать первую букву слов заглавной и т.п.).
- *StrReverse()* — "перевернуть" 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.

9.4 Функции для работы с числовыми значениями

Числовые функции VBA, функции Abs(), Int(), Fix(), Round(), Rnd(), команда Randomize

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 списке ниже не нашли ничего подходящего для вашей ситуации, возможно, есть смысл воспользоваться функциями Excel.

Кроме того, если в 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 Visual Basic на Application (только внутри Excel, в котором установлена эта надстройка).

Ниже 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 (читайте, 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():

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

- *Rnd()* и команда *Randomize* используются для получения случайных значений (очень удобно для генерации имен файлов и в других ситуациях). Обычный синтаксис при применении Rnd выглядит так:

случайное_число = Int(минимум + (Rnd() максимум))*

*MsgBox(Int(1 + (Rnd() * 100)))*

Настоятельно рекомендуется перед вызовом функции Rnd() 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.

9.5 Функции для работы с датой и временем

Функции VBA для работы с датой/временем, функции Date(), Time(), DateAdd(), DateDiff(), DatePart(), DateSerial(), Timer()

Без функций даты и времени обычно обойтись просто невозможно. Самые важные функции VBA для работы с датой/временем:

- *Date()* — возвращает текущую системную дату. Установить ее можно при помощи одноименного оператора, например, так:

Date = #5/12/2006#

- *Time()* возвращает текущее системное время, а *Now()* — дату и время вместе.
- *DateAdd()* — возможность добавить к дате указанное количество лет, кварталов, месяцев и так далее — вплоть до секунд.
- *DateDiff()* — возможность получить разницу между датами (опять таки 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()* — возможность создать значение даты на основе передаваемых символьных значений. То же самое делает *DateValue()*, отличия — в 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.

9.6 Функции для форматирования данных

Функции форматирования VBA, функция Format()

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

Format (выражение, "формат")

Несколько примеров применения 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 названия.

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 нее перевести дату "по частям" в текстовый формат и "склеить" ее нужным образом.

9.7 Функции для организации взаимодействия с пользователем

Взаимодействие с пользователем в VBA, функции MsgBox() и InputBox()

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 соответствующих главах. В этой части 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 кол-во кнопок (**OK**, **Cancel**, **Abort**, **Retry**, **Ignore**, **Yes**, **No**),

- можно показывать символы Critical, Warning, Question, Information,
- можно выбирать кнопку по умолчанию,
- можно делать окно модальным или обычным.

В зависимости от того, на какую пользователь кнопку нажал, такое значение возвращается приложению (всего 7 вариантов). Подробнее — в справке по VBA. Пример возврата значения от MsgBox():

```
Dim nVar As Integer

nVar = MsgBox ("Будем делать?", 65, "Демонстрационное окно сообщения")
```

Если значение nVar равно 1, то пользователь нажал OK, если 2, то 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 **References** нужно добавить ссылку на Windows Script Host Object Model (файл C:\WINNT\system32\wshom.ocx), 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 Input

Input = InputBox("Введите Ваше имя: ")

MsgBox (" Вы ввели: " & Input )
```

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 выполнить преобразование.

Можно привлечь внимание пользователем звуковым сигналом. Для этой цели используется оператор *Beep*:

```
Dim I

For I = 1 To 3

Beep

Next I
```

9.8 Функции — заменители синтаксических конструкций

Функции - заменители синтаксических конструкций VBA, функции Choose(), IIF(), Switch()

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, когда только это возможно.

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

- *Choose()* — принимает число (номер значения) и несколько значений. Возвращает значение, порядковый номер которого соответствует передаваемому числу. Например, Choose (2, "Первый", "Второй", "Третий") вернет "Второй".
- *IIF()* — расшифровывается как Immediate IF, то есть "Немедленный IF. Представляет из себя упрощенный вариант IF, когда проверяется условие и возвращается одно из двух значений. Пример:

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

- *Switch()* — принимает неограниченное количество пар типа выражение/значение, проверяет каждое выражения 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
```

9.9 Функции для работы с массивами

Функции VBA для работы с массивами, функции Array(), Filter(), LBound(), UBound(), Join(), Split()

Как уже говорилось, при программной работе с приложениями Microsoft Office массивы используются нечасто. Вместо них применяются коллекции. Однако в VBA предусмотрены и возможности 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()* — возвращает информацию о нижней границе массива (то есть номере первого имеющегося в нем значения), а *UBound()* — о верхней (номер последнего имеющегося значения).
- *Join()* — возможность слить множество строк из массива строк в одну строковую переменную. В качестве разделителя по умолчанию используется пробел, можно указать 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.

9.10 Функции для работы с файловой системой

Файловые функции VBA, Input(), FileLen(), EOF(), LOF(), Loc()

В VBA предусмотрен набор встроенных функций для выполнения различных операций с файлами, каталогами, дисками и прочими объектами файловой системы. Информация об этих функциях приведена ниже. Но не забывайте, что помимо этих возможностей (общих для всех приложений, в которых используется VBA) у нас есть также, 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 под названием Microsoft Scripting Runtime — очень простая и очень удобная для выполнения различных операций с файлами, каталогами и дисками. Можно добавить в проект VBA на нее ссылку и использовать все имеющиеся в ней возможности. Если, к примеру, мне нужно пройти 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**).

А это — встроенные функции для работой с файловой системой, предусмотренные в VBA:

- *CurDir()* — функция, которая возвращает путь к текущему каталогу, в котором будут сохраняться файлы нашего приложения по умолчанию.
- *Dir()* — позволяет искать файл или каталог по указанному пути на диске.
- *EOF()* — при операции записи в файл на диске эту функция вернет True, если вы находитесь в конце файла. Используется при записи в файл своего собственного формата. При сохранении документов Word, книг Excel и т.п. лучше использовать стандартные методы объектов этих документов: Save и SaveAs().
- *Error()* — позволяет вернуть описание ошибки по ее номеру. Генерировать ошибку нужно при помощи метода RaiseError() специального объекта Err (см. главу 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 это будет время создания файла.
- *FileLen()* — позволяет определить длину указанного вами файла в байтах.
- *FreeFile()* — позволяет определить следующую свободную цифру, которую можно использовать как номер файла при его открытии.
- *GetAttr()* — возможность обратиться к файлу к файловой системе и получить информацию об 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 Immediate можно так:

```
Dim MyChar

Open " c:\text1.txt" For Input As #1 'Открываем файл функцией Open() на чтение

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

Debug . Print MyChar 'Выводим его содержание в окно Immediate
```

- Вариант этой функции — *InputB()* позволяет указывать количество байт, которые надо скачать из файла.
- *Loc()* — от Location, то есть 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.

9.11 Прочие функции VBA

Функции VBA DoEvents(), Environ(), GetAllSettings(), Partition(), QBColor(), RGB(), Shell(), TypeName(), VarType()

В этот раздел попали те встроенные функции языка VBA, которые у меня не получилось отнести ни к одной другой категории.

- *DoEvents()* — 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 работает очень долгая операция (поиск на дисках, обработка большого объема данных 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()* — получить (в виде двухмерного массива) из реестра все параметры, которые относятся к указанному вами приложению. *SaveSetting()* позволяет записать эту информацию в реестр, *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()* — позволяет определить, 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 древнего номерного обозначения с возможными 16 значениями в RGB-код, который понимается VBA. Обычно используется при переделке унаследованных программ.
- *RGB()* — еще одна функция для работы с цветом. Позволяет вернуть цветовой код, который можно использовать для присвоения цвета в коде, 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 используется опытными разработчиками

при применении 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.