Глава 53

Написание модулей на VBA

В этой главе...

- Создание программы на VBA
- Присоединение модулей, проектов и процедур
- Правильное расположение элементов VBA
- Добавление комментариев
- Описание и инициализация переменных, типов данных, операторов и массивов
- Работа со структурами управления, а также циклами

После ознакомления с основными концепциями VBA и знакомства с редактором Visual Basic, пора засучить рукава и приступить к написанию кода. В этой главе рассказывается о том, как программировать и тестировать программы в VBA.

Из чего строится программа

Программа VBA — не просто случайный набор инструкций для компьютера, а хорошо *организованный* случайный набор инструкций. Строки кода складываются в *процедуры*, из которых образуются *модули*, которые, в свою очередь, составляют *проекты*.

Пример программы

Чтобы представить элементы VBA более конкретно, рассмотрим модуль, приведенный в этом разделе. Он содержит вышеупомянутые элементы (за исключением проектов, поскольку модули входят в проекты, но не наоборот). В данном коде реализован поиск в выделенных ячейках рабочего листа Excel и форматирование ячеек, значения которых превышают 1000. К ним применяются границы, полужирный шрифт и красный цвет шрифта. Затем сообщается количество измененных ячеек (рис. 53.1).

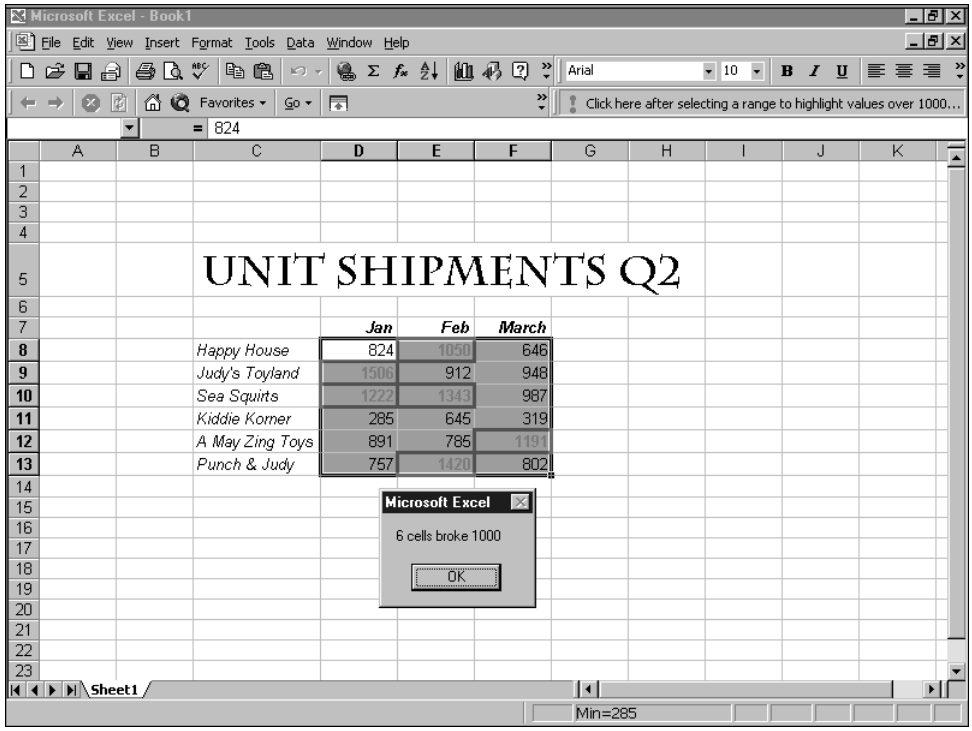
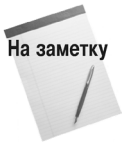


Рис.53.1. Рабочий лист Excel после запуска модуля

Чтобы проверить, как работает пример, запустите Excel, откройте новый рабочий лист и введите числа в ячейки; некоторые значения должны быть меньше 1000, некоторые больше. Затем выделите ячейки, содержащие цифры, и нажмите <Alt+F11> или выберите Сервис⇨Макрос⇨Редактор Visual Basic (Tools⇨Macro⇨Visual Basic Editor), чтобы запустить редактор. Затем откройте окна Project Explorer (Проект) и Properties (Свойства), нажав, соответственно, <Ctrl+R> и <F4>. Выберите Insert⇨Module (Вставка⇨Модуль), чтобы открыть пустое окно — чистый холст, на котором предстоит набрать код.



Автоматически присваиваемое название модуля отображается в поле Name (Имя) в окне Properties и в поле структуры Project Explorer. Чтобы изменить название модуля, выделите текущее имя в окне Properties и введите новое. Можно использовать любое, сколь угодно длинное имя, не содержащее пробелов и начинающееся с буквы. Информация о правилах и соглашениях об именовании приводится далее в этой главе.

В окне редактора кода (Code) введите следующий код:

```
Option Explicit
Const Cutoff As Integer = 1000
Dim objCell As Object
Sub HighlightBigNumbers()
   Dim intCellCounter As Integer
   intCellCounter = CountAndDoHighlights
   MsgBox (intCellCounter & " cells broke " & Cutoff)
End Sub

Function CountAndDoHighlights() As Integer
```

```
On Error GoTo BYE
   Dim z As Integer
   For Each objCell In Selection
      If objCell.Value > Cutoff Then
         With objCell.Font
            .Bold = True
            .Color = RGB(255, 0, 0)

         End With
         objCell.BorderAround ColorIndex:=3, _
            Weight:=xlThick
         z = z + 1
      End If
   Next objCell

CountAndDoHighlights = z

BYE:
End Function
```

Знакомство с иерархией VBA

После ознакомления с примером будет проще понять последующие определения и разобраться с описаниями блоков, из которых создаются программы на VBA.

Оператор — это наименьший элемент VBA-программы, который выполняет какое-либо действие. Оператор может объявлять или инициализировать переменную, задавать параметры компилятора VBA или выполнять операцию. Его можно сравнить с предложением: если оно не содержит необходимых компонентов, набор слов предложением назвать нельзя.

Процедура — наименьший элемент кода, которому можно дать название. Также процедура — наименьший участок кода, который можно выполнить отдельно. VBA различает два основных типа процедур: процедуры `Sub` (подпрограммы) и процедуры `Function` (функции). Процедура любого типа состоит из одного или нескольких операторов, заключенных между двумя специальными операторами: объявления процедуры в начале и оператора `End Sub` или `End Function` в конце.

Модуль — элемент, имеющий собственное название и состоящий из одной или нескольких процедур и объявлений, общих для всех процедур модуля. Хотя VBA и позволяет размещать все процедуры в одном модуле, более разумным представляется разнести процедуры по разным модулям, что упростит тестирование программы.

В VBA существуют два типа модулей. В основном, используются *стандартные модули* — для написания и запуска кода. Другой тип, *модули классов*, используется для описания стандартных объектов, а также их свойств и методов.

Проект состоит из всех модулей, форм и объектов, связанных с определенным документом, а также самого документа.

Определение программы

Итак, что такое программа? Программа — это завершенное правильно (или неправильно) работающее программное обеспечение. Программа состоит из одного или нескольких операторов, которые выполняются в порядке, определенном программистом.

Однако программа не является основным элементом VBA. VBA распознает процедуры, модули и проекты, но не программы как таковые. Программа VBA должна включать в себя как минимум одну процедуру, поскольку операторы VBA выполняются в составе процедур, но программа может содержать и большее количество процедур, заключенных в одном или нескольких модулях, которые, в свою очередь, входят в один или несколько проектов.

Снова о проектах

Чтобы создать проект, не нужно абсолютно ничего делать. Каждый документ в приложении VBA автоматически является проектом. Конечно, проект документа не будет содержать код или формы до тех пор, пока они не созданы в редакторе Visual Basic, или не записан макрос в самом приложении. Техника работы с проектами в целом описана в главе 52.

Модули

Согласно иерархии VBA, модуль — следующий за проектом. Модуль содержит одну или несколько процедур VBA, а в разделе объявлений находятся операторы, применяемые ко всему модулю.

Планирование модулей

Организация модулей не занимает много времени. Продумайте, сколько должно быть создано модулей и какие процедуры каждый из них должен содержать, учитывая следующие соображения.

- Процедуры могут *вызывать* или запускать процедуры, хранимые в других модулях. Также разрешается использовать переменные, объявляемые в других модулях.
- Использование процедур и переменных из других модулей является чуть более сложным процессом. Например, чтобы вызвать процедуру из другого модуля, нужно ввести также и имя модуля: `OtherModule.DoSomethingNow`.

В общем, лучше размещать связанные между собой процедуры в одном модуле. Как правило, модуль содержит процедуры одной программы VBA, без посторонних процедур, что упрощает программирование, поскольку не появляются трудности, связанные с вызовом процедур из других модулей.

Однако, если процедуры планируется использовать не только в одной программе, их можно организовать в соответствии с типом, например `MyMathProcedures` (математические процедуры) или `TextHandlingRoutines` (подпрограммы работы с текстом). Также можно создать модули `BrandNewProcedures` или `OldProceduresIMightNeedSomeday` (старые процедуры, которые могут пригодиться).

Добавление новых модулей в проект VBA

Чтобы создать новый модуль в редакторе Visual Basic, сначала убедитесь, что работа проводится с нужным проектом. В Project Explorer выберите соответствующий проект или любой из содержащихся в нем объектов (форма, объект приложения или

модуль). Чтобы добавить новый модуль, воспользуйтесь любым из приведенных далее способов.

- **Щелкните на кнопке Insert Module** (Вставить модуль). Это многофункциональная кнопка для вставки различных элементов. Если пиктограмма модуля невидима, щелкните на элементе со стрелкой справа от кнопки и выберите модуль из меню.
- **Щелкните правой кнопкой мыши в окне Project Explorer.** Убедитесь, что действие выполняется над нужным проектом и выберите в контекстном меню команду Insert⇨Module (Вставка⇨Модуль). Можно также воспользоваться основным меню, выбрав Insert⇨Module.

Редактор Visual Basic автоматически откроет окно редактора кода нового модуля. Он также наградит модуль новым названием. Чтобы изменить его, введите новое имя в окне Properties (см. главу 52).

Что к чему в новом модуле

Окно кода нового модуля содержит только один раздел объявлений (Declarations). Можно определить текущий раздел, посмотрев раскрывающийся список в правом верхнем углу окна кода.

В разделе объявлений можно вводить только два типа операторов.

- **Объявления переменных, констант и типов данных, определенных пользователем.** Они сообщают компилятору имя и тип каждого элемента (но не его значение). Переменные и константы, объявленные в разделе объявлений, могут использоваться в любой процедуре данного модуля.
- **Опции компилятора, которые отвечают за способ компиляции, используемый в VBA.** Например, `OPTION EXPLICIT` предписывает программисту прямо объявлять все переменные.

В разделе объявлений нельзя размещать операторы присваивания, которые выполняются при запуске кода. Например, нельзя присваивать переменным *значение*, что требует использование оператора присваивания, который должен размещаться в любой процедуре модуля. (Различия между выполняемыми операторами, операторами присваивания, объявлениями и опциями компилятора описываются в разделе "Использование операторов" далее в этой главе).

Операторы в разделе объявлений относятся к *коду уровня модуля*. Каждая процедура, добавляемая в модуль, является отдельным разделом. После добавления процедуры ее название появляется в раскрывающемся списке в правом верхнем углу окна Code, что позволяет легко переходить к процедуре, просто выбрав ее название из списка.

Стандартные модули и модули классов

Основным модулем VBA является стандартный модуль (в более ранних версиях VBA и Visual Basic известный как модуль кода). Такие модули содержат операторы, которые объявляют переменные и константы, определяющие пользовательские типы данных и устанавливающие опции компилятора. В этих модулях находятся также исполняемые операторы, которые и заставляют работать программу. В стандартном модуле можно использовать все объекты, к которым есть доступ, но создать новые в таком модуле нельзя.

Модули классов используются для определения собственных объектов. Чтобы создать модуль класса, выберите команду Insert⇨Class Module (Вставка⇨Модуль класса). После создания такого модуля в него нужно внести код, содержащий свойства и методы объекта. После этого объект можно вызывать из других модулей, как и обычный встроенный объект VBA.

Написание собственных классов — мощный инструмент VBA, о котором речь пойдет в главе 54.

Написание процедур

Процедуры — основной элемент кода на VBA. Для запуска кода можно, он должен содержаться в процедурах. Процедуры в VBA бывают двух типов: `Sub` и `Function` (см. врезку "Типы процедур"). Ниже приведены две простые процедуры разных типов:

```
Public Sub SubMarine()
MsgBox "Up Periscope!"
End Sub
Public Function FunkShun (Birthdate As Date)
FunkShun = DateDiff("yyyy", Birthdate, Date)
End Function
```

Как видно, каждая процедура содержит оператор объявления, одну или несколько строк кода и завершающий оператор `End`. Полностью эти элементы будут рассмотрены в разделах "Подробно о процедурах Sub" и "Подробно о процедурах Function".

Типы процедур

Ниже приводится описание процедур разных типов.

- ✧ Процедура `Sub` — универсальная процедура, на которой держится выполнение программ в VBA. Процедуры `Sub` — единственный тип процедур, который может запускаться самостоятельно. Однако процедура `Sub` может также запускать (или *вызывать*) другую процедуру.
- ✧ В процедуре `Function` могут выполняться любые операторы VBA. Однако она отличается от процедуры `Sub` тем, что в ней подсчитывается значение, которое возвращается в процедуру, запустившую `Function`.
- ✧ Процедура `Event` — специальный тип процедуры `Sub`, который используется для реакции на события, происходящие с формами и объектами приложения, такими как документы и само приложение. Особенности этих процедур описаны в главе 56.
- ✧ Процедура `Property` возвращает или присваивает значение свойству объекта. Процедуры `Property` описаны в главе 54.
- ✧ С технической точки зрения, макрос VBA представляет собой процедуру `Sub`, не требующую аргументов. Макросы — это процедуры `Sub`, которые могут запускаться в редакторе Visual Basic или пользовательском приложении VBA. Чтобы запустить процедуру `Sub`, имеющую аргументы, нужно вызвать ее из другой процедуры.

Создание новой процедуры

Перед созданием новой процедуры откройте в окне Code модуль, в котором будет храниться процедура. Создать новый модуль можно изложенным выше способом, а чтобы открыть существующий модуль дважды щелкните на его названии в окне Project Explorer.

Когда окно Code с загруженным в него модулем активно, можно начинать написание процедуры. Сначала введите оператор объявления, дополнительный код и оператор `End`, сообщающий о завершении процедуры. Это можно сделать двумя способами: через диалоговое окно Insert⇨Procedure (Вставка⇨Процедура) или набрав операторы вручную.

При использовании диалогового окна Insert⇨Procedure, определите имя процедуры и выберите ее тип (Subroutine, Function или Property), после чего задайте область видимости процедуры (Public или Private). По желанию, можно определить все локальные переменные как статические.

Ввод вручную или при помощи диалогового окна. Что лучше?

Рекомендуется вводить процедуры вручную, хотя бы потому, что диалоговое окно экономит время, только если вы *действительно* медленно печатаете. Кроме того, ввод вручную позволяет самостоятельно определять место в теле модуля для расположения процедуры — достаточно установить курсор там, где должна располагаться новая процедура, ввести оператор объявления процедуры и нажать <Enter>. Круглые скобки и завершающий оператор `End Sub` или `End Function` вводить не следует — редактор Visual Basic сделает это сам.

Заполнение процедуры

Основная работа по написанию процедуры заключается в написании операторов VBA между оператором объявления и оператором `End`. Не будем останавливаться на том, что должно содержаться в процедуре. Достаточно отметить, что операторы в процедуре выполняются в порядке следования, за исключением операторов перехода.

Подробно о процедурах Sub

В некоторых языках программирования основные процедуры вызывают меньшие процедуры, которые называются *подпроцедурами* или *подпрограммами*. Они также известны как процедуры `Sub`, потому что начинаются с `Sub` и заканчиваются `End Sub`. Хотя в VBA процедура `Sub` может быть вызвана другой процедурой, главная процедура программы *всегда* является процедурой `Sub`. Ниже приводится пример процедуры `Sub` с оператором объявления в начале, завершающим `End` в конце и несколькими операторами между ними:

```
Public Sub ASweetProcedure()
Dim ANiceMessage As String
ANiceMessage = "Have a nice day!"
Msgbox ANiceMessage
(другие операторы)
...
End Sub
```

Элементы объявления процедуры Sub

В операторе объявления, которым начинается `ASweetProcedure()`, первый термин, `Public`, определяет область видимости процедуры, которая может быть открытой и закрытой (`Public` и `Private`). `Public` устанавливается по умолчанию, т. е. это ключевое слово не обязательно вводить в объявлении, оно просто делает область видимости очевидной. Область видимости рассматривается в разделе "Область видимости" далее в этой главе.

Затем идет ключевое слово `Sub`, свидетельствующее об объявлении процедуры `Sub`. За ним следует имя процедуры, которое целиком зависит от пользователя. Это имя может быть любой длины, однако следует придерживаться предусмотренных стандартов, приведенных в разделе "Правила именования в VBA" далее в этой главе.

О параметрах

Объявление процедуры заканчивается парой круглых скобок. В них должны находиться *параметры*, если таковые передаются процедуре. В случае, если процедуре не передаются параметры, в круглых скобках ничего не пишется. Параметры описываются в разделе "Параметры" далее в этой главе.

Встроенные функции VBA

В VBA множество встроенных функций. Как и процедуры `Function`, функции VBA возвращают значение вызывающей процедуре. Для вызова встроенных функций и процедур `Function` применяются одинаковые методы. У них разные имена, но разница для пользователя заключается лишь в том, что для встроенных функций не нужно писать код. В разделе "Встроенные функции и операторы" дается обзор функций VBA.

Вызов процедур Sub

Запустить или *вызвать* любую процедуру `Sub` можно не зависимо от того, принимает она параметры из другой процедуры или нет. Чтобы вызвать процедуру `Sub`, добавьте оператор, состоящий из имени нужной процедуры, как во фрагменте кода, приведенном ниже. Строка `WashMyOldCar` вызывает процедуру `WashMyOldCar`:

```
...
MyOldCar = "Valiant"
WashMyOldCar
NumberOfCarwashTrips = NumberOfCarwashTrips + 1
...
```

Подробно о процедурах Function

Процедура `Function` может справляться с задачами процедуры `Sub`, но ее основной смысл — в вычислении значений. Процедура `Function` после выполнения возвращает значение вызывавшей ее процедуре, это значение может быть использовано в дальнейших вычислениях.

Ниже приведен пример процедуры `Function`:

```
Public Function DeFunct(x As Integer, y As Integer)
Dim z As Integer
z = x + y
```

```
DeFunct = x ^ z
End Function
```

Очевидно, что вид процедуры `Function` во многом повторяет процедуру `Sub`. Объявление начинается с необязательного параметра, определяющего область видимости процедуры (в данном случае, `Public`). Затем идут ключевое слово `Function`, имя процедуры и передаваемые процедуре параметры. (В четвертой строке кода переменная *x* возводится в степень *z*.) Функция заканчивается оператором `End Function`.

Разница между процедурами Sub и Function

Основное различие между процедурами `Sub` и `Function` заключается в том, что в процедуре `Function`, по крайней мере, один оператор определяет значение функции, а имя функции используется как переменная. В предыдущем примере это происходит в строке `DeFunct = x ^ z`. После выполнения данной строки `DeFunct` принимает значение, которое и будет возвращено вызывающей процедуре.

Вызов процедуры Function

Как правило, вызов процедуры `Function` происходит, когда какой-либо переменной присваивают ее в качестве значения. В следующем примере в первом выполняемом операторе переменная `Zpower` получает значение, возвращаемое процедурой `DeFunct`. Обратите внимание: параметры, передаваемые процедуре, следуют за ее именем и заключены в круглые скобки.

```
Sub DoDeFunct()
Zpower = DeFunct(3, 4)
MsgBox Zpower
End Sub
```

Параметры

Параметры — это значения, которые язык VBA должен *передать* от одной процедуры другой. Говорят, что вторая функция *принимает* параметры. Параметры передаются процедуре, чтобы изменить ее поведение в зависимости от существующих значений во время вызова.

Возьмем `DeFunct` — пример процедуры `Function`, приведенный в предыдущем разделе "Подробно о процедурах `Function`". Функция принимает два аргумента: `x` и `y`. В примере видно, что объявление переменных присутствует в объявлении функции. Переменные всегда содержатся в круглых скобках, следующих за именем функции. В этом *списке параметров* пользователь объявляет индивидуальные параметры так же, как и переменные этих типов, за исключением того, что отсутствует `Dim` или подобное зарезервированное слово (подробнее об объявлении индивидуальных переменных см. раздел "Работа с переменными").

Процедуре, принимающей параметры, они требуются для работы. Внутри процедуры параметры играют ту же роль, что и обычные переменные. Обратимся вновь к процедуре `DeFunct`. После объявления переменной `z` процедура присваивает `z` сумму значений `x` и `y`. Затем, чтобы высчитать значение, возвращаемое процедурой, следующая строка кода возводит `x` в степень `z`. Как видно, у `x`, `y` и `z` роли одинаковы. При желании можно, например, `z` возвести в степень `x`.

Зачем нужны параметры

Казалось бы, если параметры настолько схожи с переменными, зачем они нужны? Вообще, можно использовать переменные, чтобы достичь того же, что делается с помощью параметров. Но параметры имеют некоторые преимущества.

- При чтении кода параметры ясно показывают, какие значения процедура требует для работы.
- При написании кода параметры помогают уменьшить количество переменных, создаваемых вне процедур (в разделе Declarations).
- При вызове процедуры, которая принимает параметры, VBA *заставляет* пользователя присвоить параметрам значения, это гарантирует, что процедура получит правильные значения, нужные ей для работы.

Вызов процедур, которые принимают параметры

Способ вызова процедуры, которая принимает параметры, зависит от ситуации.

- Если используется процедура `Function`, которая возвращает значение, параметры разделяются запятыми и располагаются в круглых скобках: `ProceedsFromOldCar = SellOldCar ("Rambler Classic",1962)`
- Если вызывается процедура `Function` сама по себе и возвращаемый результат не важен или если вызывается процедура `Sub`, то параметры пишутся без круглых скобок: `SellOldCar "Studebaker", 1957`

Область видимости

У каждой процедуры VBA есть своя *область видимости* (scope). Область видимости определяет части программы, из которых может быть вызвана процедура. Представьте себе это следующим образом: область видимости процедуры решает, какие из частей программы могут "видеть" процедуру.

У процедур может быть три области видимости:

- По умолчанию процедуры VBA (кроме процедур обработки событий) *открытые* (public). Это значит, что их можно вызвать из любой части программы, из этого же модуля, из другого модуля или даже из другого проекта (если, конечно, программа содержит несколько проектов, как описывается в главе 51).
- При необходимости можно создать процедуры с *закрытой* (private) областью видимости. Такие процедуры видимы только из этого же модуля. Другими словами, закрытую процедуру можно вызвать только из другой процедуры, которая находится в том же модуле, но не из процедур других модулей.
- В программе VBA, которая содержит несколько проектов, можно создавать процедуры, доступные для всех модулей проекта, но доступ к ним со стороны других проектов будет закрыт.

У переменных тоже есть область видимости, которая работает подобным образом. О ней речь пойдет в разделе "Назначение области видимости переменных" далее в этой главе.

Назначение области видимости процедуры

Все, что нужно сделать для назначения области видимости процедуры, это ввести ключевое слово `Public` или `Private` в ее объявлении. Например:

```
Public Sub IKneadYou()
... (операторы процедуры)
End Sub

Private Function IKneedYou()
... (операторы процедуры)
End Function
```

Так как обычные процедуры (не обрабатывающие события) являются открытыми по умолчанию, не обязательно добавлять ключевое слово `Public`, чтобы сделать процедуру открытой. Однако, если программа содержит также закрытые процедуры, рекомендуется явно объявлять открытые процедуры, чтобы можно было определить область видимости каждой с первого взгляда.

Чтобы ограничить область видимости процедуры одним проектом (и сделать закрытой для других проектов), добавьте оператор `Option Private Module` в раздел Declarations модуля, в котором объявлена процедура. (Подробнее работа с этими операторами рассмотрена в разделах "Объявления" и "Опции компилятора".)

Использование закрытых процедур

Использование закрытых процедур помогает избежать ошибок. Поскольку процедуру можно вызвать только из модуля, в котором она располагается, это упрощает контроль за условиями вызова процедуры (под условиями подразумеваются значения переменных, используемых процедурой).

Сделать процедуру закрытой достаточно легко, но сначала продумайте необходимость такого шага. В конечном счете, VBA не *требует* вызывать процедуру из другого модуля только потому, что процедура открытая.

Основной причиной такого решения может быть защита. Пользователь может забыть, что создавал процедуру для использования внутри модуля. Если процедура закрытая, VBA не позволит вызвать ее из другого модуля. Преимущество заключается в том, что в программах большого размера и сложности, ограничение доступа к процедуре помогает отслеживать организацию программы.

Операторы

Процедуры состоят из *операторов*, самых маленьких значимых элементов кода. Большинство операторов занимают одну строку и, как правило, строка кода содержит один оператор. В VBA различают четыре типа операторов: объявления, присваивания, выполняемые операторы и опции компилятора. Операторы могут содержать ключевые слова, выражения, константы, операторы и переменные.

Объявления

Объявления сообщают компилятору VBA намерение пользователя применить в программе названный элемент — переменную, константу, тип данных, определенный пользователем, массив или процедуру. В объявлении определяется тип элемента и предоставляется дополнительная информация для компилятора. После объявления элемента его можно использовать в любом месте программы.

Следующий оператор объявляет переменную `MyLittleNumber` как целочисленную:

```
Dim MyLittleNumber As Integer
```

Следующий создает строковую (текстовую) константу с именем `UnchangingText`, которая состоит из символов "`Eternity`":

```
Constant UnchangingText = "Eternity"
```

Следующий оператор объявляет процедуру `Sub` с именем `HiddenProcedure`:

```
Private Sub HiddenProcedure()
```

Операторы присваивания

Операторы присваивания устанавливают значение переменной или свойства объекта равным какому-либо значению. Эти операторы состоят из трех частей: имя переменной или свойства, знак равенства и *выражение*, определяющее новое значение (выражения подробно обсуждаются в разделе "Выразительность").

Следующий оператор устанавливает значение переменной `MyLittleNumber` равным сумме переменной `SomeOtherNumber` и 12:

```
MyLittleNumber = SomeOtherNumber + 12
```

А этот оператор устанавливает для свойства `Color` объекта `AGraphicShape` значение `Blue`, которое, по всей видимости, было определено как константа:

```
AGraphicShape.Color = Blue
```

Выполняемые операторы

Выполняемые операторы не просто занимают место, они работают. Например, следующий оператор выполняет метод `Rotate` объекта `AGraphicShape`:

```
AGraphicShape.Rotate(90)
```

Следующий оператор выполняет функцию `Sqr`, которая вычисляет квадратный корень, над переменной `MyLittleNumber` и помещает результат в переменную `SquareRoot`:

```
SquareRoot = Sqr(MyLittleNumber)
```

Опции компилятора

Инструкции, контролирующие поведение компилятора VBA — это последний класс операторов. Из четырех операторов опций компилятора основным является

```
Option Explicit
```

Директива `Option Explicit` подробно описана в разделе "Выбор и использование типов данных".

К порядку! Этикет VBA

В этом разделе обсуждаются правила и соглашения по именованию элементов VBA, для написания понятного, читабельного, работающего кода. Речь пойдет о форматировании кода с целью придать ему большей ясности, а также о и добавлении комментариев.

Правила именования в VBA

VBA вводит правила для наименования *элементов программы* (переменные, константы процедуры, модули, формы, элементы контроля и проекты). Если ввести имя, не соответствующее этим правилам, при переходе курсора на другую строку редактор Visual Basic выдаст предупреждение.

- Имена должны начинаться с буквы. Однако в названии можно использовать цифры и символы подчеркивания, как в следующем примере: `Hidden_Variable3`
- Нельзя использовать в именах пунктуационные знаки, кроме символа подчеркивания. В именах VBA не допускается также использование следующих символов: ! @ & ' $ # ? , * . { } ( ) [ ] = + - ^ % / ~ < > : ;
- Имена не могут содержать пробелы.
- Длина имени не должна превышать 255 символов (для форм и элементов управления — 40 символов).
- Имя не должно повторять зарезервированных слов VBA, имен функций, операторов или методов.
- Нельзя использовать имя более одного раза в одной области видимости. Например, все процедуры одного модуля должны иметь разные имена. Процедура и переменная уровня модуля, объявленная в разделе Declarations, не могут иметь одинаковое имя. Однако можно использовать одинаковые имена для переменных, объявленных внутри разных процедур.

VBA игнорирует регистр, но сохраняет регистр, выбранный пользователем.

Соглашение об именовании

Учитывая изложенные ранее правила, элементу программы позволяется давать любое имя. Тем не менее, можно значительно упростить программирование, используя логическую систему именования. Один из методов — это использование коротких префиксов, описывающих тип элемента, за которыми идет описательное имя, начинающееся с заглавной буквы. В табл. 53.1 приведены общепринятые префиксы для большинства элементов VBA. Можно использовать их, а можно придумать свои. Главное, быть последовательным.

Таблица 53.1. Префиксы, используемые для именования элементов VBA

Префикс	*Тип элемента*	*Пример*
Переменные		
`byte`	Byte (байт)	`byteDaysInMonth`
`bool`	Boolean (булева)	`boolClearedStatus`
`int`	Integer (целое)	`intWeekOnChart`
`lng`	Long Integer (длинное целое)	`ingPopulation`
`sng`	Single	`sngRadius`

dbl	Double	dblParsecs
cur	Currency (денежный)	curUnitPrice
str	String (строка)	strLastName
dat	Date/Time (дата/время)	datBirthdate
var	Variant	varSerialNumber
obj	Object (объект)	objStampCollection
Элементы управления формы		
txt	Text box (текстовое поле)	txtEnterName
lbl	Label (надпись)	lblAnswerMessage
cmd	Command button (кнопка)	cmdCalculateInterestRate
mnu	Menu (меню)	mnuTools
cmb	Combo box (поле со списком)	cmbToyCategory
fra	Frame (фрейм)	fraHabitat
opt	Option button (переключатель)	optGasolinGrade
chk	Checkbox (флажок)	chkCaseSensitive
bas	Module (модель)	basTextFormatFunctions
frm	UserForm (пользовательская форма)	frmOptionsDialog

Как сделать код понятным

Этот раздел посвящен нескольким простым советам по написанию кода, в котором можно легко разобраться в последующем.

Использование отступов помогает организовать код

Придерживайтесь единого стиля. Компилятор VBA игнорирует отступы в начале строки, значит их можно использовать для смысловой организации строк кода. Итак, перед какими строками следует делать отступы и сколько? Основная задача — установка одинаковых отступов в связанных строках, чтобы связь стала очевидной. Также можно использовать отступы в строках, которые выполняются при наступлении определенных условий.

Например, VBA выполняет операторы, входящие в тело конструкций `If...Then...Else` или `Do...Loop`, или `For...Next` как группу, соответственно перед ними должен быть одинаковый отступ. Пример:

```
Do While intC <> 20
    intA = intA + 1
    If intA = intB Then
        intA = 5
        intB = 10
    Else
        intA = intB
        intC = 20
    End If
Loop
```

Обратите внимание: *управляющие структуры* типа `Do...Loop` `If...Then...Else` состоят, как минимум из двух элементов: один открывает структуру, другой завершает ее. В `Do...Loop` завершающим является оператор `Loop`; в `If...Then...Else` —это `End If`. Операторам, которые являются частью одной структуры, должны предшествовать одинаковые отступы, что позволяет четко видеть структуры и входящие в них операторы.

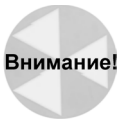
Чтобы свести вашу работу к минимуму, редактор Visual Basic автоматически проставляет отступы в каждой строке кода, отличные от отступов в предыдущих строках. Если в строке должен стоять отступ меньший, чем в предыдущей, нажмите клавишу <Backspace>. При желании можно отключить автоматическую расстановку отступов во вкладке Tab (Табуляция) диалогового окна Tools⇨Options (Сервис⇨Параметры).

Нет прокрутке! Использование символа переноса

Хотя короткие операторы проще для понимания, в редакторе Visual Basic вполне можно растягивать один оператор на несколько строк. Каждая строка, таким образом, становится легко читаемой, поэтому нет необходимости прибегать к горизонтальной прокрутке окна.

Чтобы продолжить оператор в следующей строке, поставьте в конце строки символ подчеркивания, _. Например, следующие три строки составляют один оператор:

```
sngWackyNumber = Cos (12 * 57.5 / Sqr( intMyTinyNumber + _
intMyBigNumber) + CustomDataMassage(sngrawinfo, 12) + _
(bytFirstTuesdayInAugust * curLastPayCheck) +1)
```



Убедитесь, что перед символом подчеркивания (иначе называемом *символом переноса*) поставлен пробел. В противном случае VBA выдаст сообщение об ошибке `invalid character` (Неверный символ). Не применяйте символ подчеркивания в строке, взятой в кавычки, — VBA выдаст сообщение об ошибке.

Немного о комментариях

Как и все серьезные языки программирования, VBA позволяет добавлять к коду комментарии, поясняющие цель каждого оператора или группы операторов. Комментарии просты в использовании и особенно полезны для начинающих программистов. При последующем просмотре кода комментарии напоминают назначение тех или иных операторов. Компилятор VBA полностью игнорирует комментарии пользователя. Комментарии присутствуют только в окне кода, а не в компилируемой программе. Они не удлиняют компилируемую программу и не замедляют ее. Комментарии не стоят пользователю ничего, кроме крошечного места на диске, поэтому можно применять их в свое удовольствие.

Как создают комментарии

Комментарии начинаются с апострофа. Все, что находится в строке справа от апострофа, является частью комментария. Можно размещать строки, полностью состоящие из комментариев, или помещать комментарии в тех же строках, что и обычный код. На рис. 53.2 показан код со множеством комментариев.

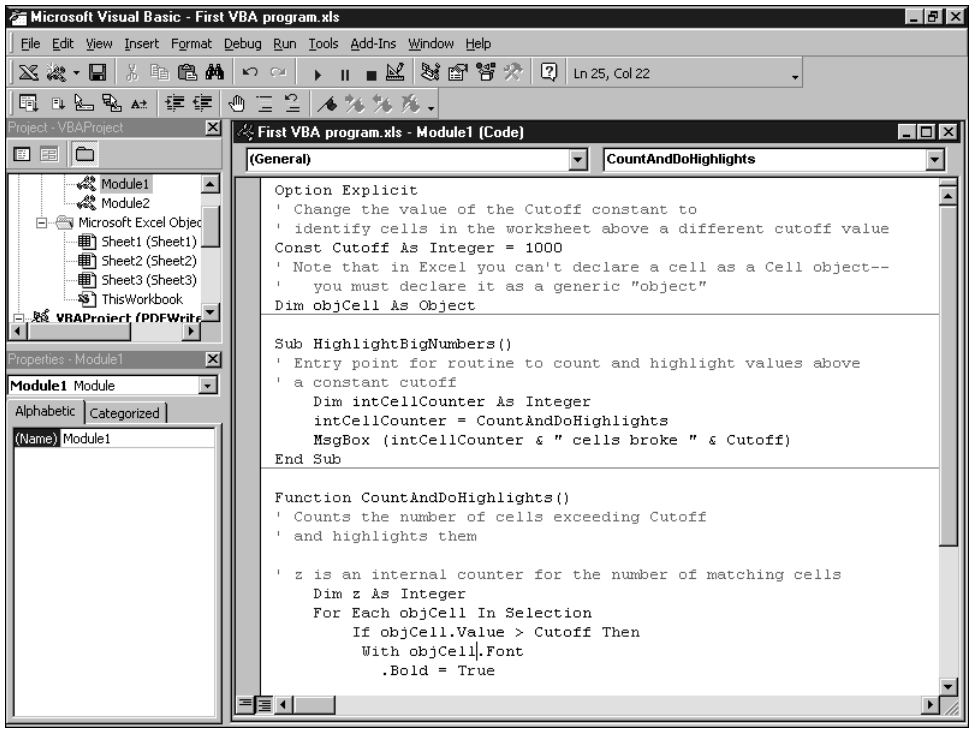


Рис. 53.2. Ваши комментарии?

Использование комментариев — хороший способ сделать операторы неактивными, не удаляя их, что может оказаться полезным при решении одной задачи двумя способами или если код содержит ошибки, к которым можно вернуться позже, а в данный момент необходимо протестировать модуль.

Длинные комментарии

Чтобы включить в программу длинный комментарий, состоящий из нескольких строк, поставьте апостроф в начале каждой строки. К счастью, VBA предоставляет команду, которая позволяет легко закомментировать сразу несколько строк. Она также предоставляет возможность *снять комментарии* с нескольких строк. Выделите нужные строки и щелкните на кнопке Uncomment Block (Снять комментарий с блока). С помощью кнопок Comment Block (Закомментировать блок) и Uncomment Block можно деактивизировать и активизировать любой фрагмент кода.

Следует отметить особенность работы этих кнопок. Если строка уже начинается с апострофа, кнопка Comment Block добавит второй и т.д. Также при щелчке на кнопке Uncomment Block будет удален только один апостроф. Поэтому, даже комментируя строки или снимая с них комментарии, можно быть уверенным, что настоящие комментарии останутся нетронутыми.

Работа с переменными

Чтобы в полной мере использовать потенциал VBA, программе необходимы *переменные* для хранения изменяющейся информации. Переменные — ключевой элемент в написании программ, который изменяет их поведение, основанное на текущей информации или изменении условий. Вообще, переменная — идентификационная метка для части информации, хранимой программой. Она напоминает театр, где зрители, оставляя в гардеробе шляпу, берут с собой номерок. Гардеробщик относит шляпу и вешает ее. Посетитель получает бирку с номером. На обратном пути гардеробщик берет номерок и возвращает шляпу. Во время следующего сеанса номерок используется уже для другой шляпы.

Объявление переменных

Наиболее распространенный способ употребления операторов объявления — объявление переменных. Переменные объявляют так:

```
Dim varAnyOldVariable
Private intIntegerVariable As Integer
Static strNewYearsResolution As String
```

Как правило, для объявления переменных используется оператор `Dim`. Зарезервированное слово `Dim` — сокращение от dimension (размер, объем). Идея заключается в том, что оператор `Dim` резервирует место для хранения информации, на которое ссылается переменная. Также для объявления переменных и определения их области видимости можно использовать ключевые слова `Public`, `Private` и `Static`. Область видимости рассмотрена в разделе "Определение области видимости переменной".

Где объявлять переменные

Объявить переменную можно:

- в разделе `Declarations` модуля (переменную уровня модуля);
- в любом месте процедуры (переменную уровня процедуры).

Выбор места объявления помогает определить, какие процедуры будут иметь доступ к переменной, другими словами — область видимости переменной. Если переменная объявляется на уровне модуля, она может использоваться любой процедурой модуля. Если же переменная объявлена в процедуре, ее можно использовать только в этой процедуре. Также для определения области видимости переменной можно воспользоваться ключевыми словами: `Public`, `Private` и `Static`. Подробнее об области видимости см. раздел "Определение области видимости переменной".

Хотя VBA и не возражает против объявления переменных среди исполняемых операторов процедуры, рекомендуется для разборчивости объявлять все переменные процедуры в ее начале. Вот как следующая процедура использует этот принцип:

```
Public Sub VariableProcedure ()
Dim strChildsName As String
Dim intToyCount As Integer
Dim curAllowance As Currency
strChildsName = InputBox ("Enter the child's name:")
' в следующих двух строках CountToys и NewAllowance
' это процедуры Function, предположительно определенные где-то
в коде
intToyCount = CountToys(strChildsName)
curAllowance = NewAllowance( strChildsName, intToyCount)
End Sub
```

Когда объявлять переменные

Как правило, переменные объявляются заранее, перед использованием в программе, чтобы сделать код более понятным. Другие преимущества появляются при использовании директивы компилятора `Option Explicit`, описанной в разделе "Использование явных объявлений" далее в этой главе.

По умолчанию VBA позволяет использовать переменные и без объявления. К примеру, если ввести **A = 7** где-нибудь в коде, VBA автоматически создаст переменную `A` и присвоит ей значение `7`. Переменные, которые создаются без объявления, автоматически объявляются как универсальные переменные (типа `Variant`), они описываются в следующем разделе.

Выбор и использование типов данных

При объявлении переменной ей можно просто присвоить имя. Следующий оператор предписывает VBA зарезервировать место для хранения переменной `varIable`, но не сообщает, какой тип информации должна хранить переменная:

```
Dim varIable
```

Можно объявить *тип данных* переменной открыто. Оператор

```
Dim sngMyOldSocks As Single
```

объявляет переменную `sngMyOldSocks` как переменную типа *single*, т. е. переменная содержит сравнительно небольшое *число с плавающей точкой* (тип, записываемый с помощью экспоненциального представления чисел в виде мантиссы и экспоненты, например 6,02 E 10^{23}).

VBA различает множество типов данных переменных, включая такие, как `Date` и `Currency`. Изучите все возможные типы, поскольку они представляют собой ключ к написанию эффективного кода. В табл. 53.2 предлагается описание типов данных. Далее в главе содержатся разделы, посвященные работе с различными типами данных.

Таблица 53.2. Типы данных в VBA

Тип данных	*Описание*	*Диапазон допустимых значений*
`Boolean`	Логические правда или ложь	`True` (-1) или `False` (0)
`Byte`	Наименьшее целое число	от 0 до 255
`Integer`	Небольшое целое число	от -32 768 до 32 767
`Long`	Большое целое число	от -2 147 483 648 до 2 147 483 647
`Single`	Число с плавающей точкой одинарной точности	от -3.402823E38 до -1.401298E45 для отрицательных чисел; от 1.401298E45 до 3.402823E38 для положительных чисел
`Double`	Число с плавающей точкой двойной точности	от -1.79769313486232E308 до -4.94065645841247E324 для отрицательных чисел; от 4.94065645841247E324 до 1.79769313486232E308 для положительных чисел
`Currency`	Большое точное число, 19 значимых цифр, включая четыре знака после десятичной точки	от -922,337,203,685,477.5808 до 922,337,203,685,477.5807
`Decimal`	Большое точное число, 29 значимых цифр, до 28 знаков после десятичной	от +/-79,228,162,514,264,337,593,543,950,335 без дроби; +/-

	точки	7.9228162514264337593543950335 с 28 дробными разрядами
`Date`	Дата и время	с 1 января 100 до 31 декабря 9999 год; от 00:00:00 до 23:59:59
`Object`	Объект VBA	Ссылка на любой объект
`String (Изменяемой длины)`	Последовательность текстовых символов изменяемой длины	от 0 до примерно 2 миллиардов (переменная длина)
`String (Фиксированной длины)`	Последовательность из определенного количества текстовых символов	от 1 до примерно 65 400 (фиксированная длина)
`Variant`	Любая величина	Любая числовая переменная до диапазона `Double`; тот же диапазон, что у `String` переменной длины
`User-defined` (требует оператор `Type`)	Группа переменных, используемая вместе как один элемент	Диапазон переменной согласно ее типу данных (см. предыдущие строки)

За и против использования типов данных

В зависимости от подхода, введение переменных определенного типа данных, отличного от `Variant`, может являться как хорошей идей, так и не очень. На практике это позволяет делать меньшие по объему и более быстрые программы, но, в большинстве случаев, скорость и размер программы не имеют решающего значения.

Смысл в том, что при объявлении переменной можно выбрать тип переменной, оптимально подходящий объему информации, который будет содержаться в этой переменной. Можно объявить переменную типа `Variant`, содержащую информацию любого типа. (Если тип переменной не объявлен, она по умолчанию `Variant`).

Переменные типа `Variant` занимают больший объем памяти, чем переменные других типов, и доступ к информации, содержащейся в них, замедляет работу программы. Однако это не так важно, если, конечно, программа не работает с действительно большим количеством переменных. Объявив все переменные как `Variant`, можно сократить количество ошибок и сделать код более гибким и легче изменяемым. К сожалению, в рамках данной книги этот вопрос не будет рассмотрен более подробно, но многие программисты рекомендуют при написании программ использовать исключительно переменные типа `Variant`.

Переменные типа `Variant` можно объявить так же, как и переменные других типов, например:

```
Dim varToutSuite As Variant
```

Однако, в связи с тем, что `Variant` — тип данных, используемый по умолчанию, можно обойтись простым присваиванием переменной имени при помощи оператора `Dim`:

```
Dim ToutSuite
```

Использование явных объявлений

Если решено объявлять каждую переменную как определенный тип данных или использовать `Variant` для всех переменных, настройте VBA так, чтобы требовалось явное их объявление, используя оператор

```
Option Explicit
```

Если расположить его в разделе `Declarations`, VBA будет выдавать сообщение об ошибке каждый раз при попытке использовать необъявленную переменную. После напоминания можно вернуться и добавить требуемое объявление.

Требование явных объявлений позволяет не допускать появления многих ошибок, связанных с неправильным набором. Вообразим ситуацию: код написан без использования в модуле `Option Explicit`. Однако все переменные объявлены явным образом. Где-то в теле процедуры, во время ввода имени очередной переменной, внезапный приступ ревматизма сыграл с пальцами программиста злую шутку. Компилятор VBA создаст новую переменную. Возможно, программа и заработает, но не стоит слишком удивляться, если текст вдруг станет пурпурным или программа сообщит, что население Земли только что достигло отметки 12.

Вместо того, чтобы писать `Option Explicit` самому, можно заставить VBA делать это автоматически: установите флажок Require Variable Declaration (Требовать объявление переменных) во вкладке Editor (Редактор) диалогового окна Tools⇨Options.

Определение области видимости переменной

Область видимости переменной определяет, где в программе доступна данная переменная, и зависит от двух факторов:

- места, где определена переменная (внутри процедуры или в разделе объявлений модуля — см. "Где объявляют переменные" ранее в этой главе)
- ключевого слова, используемого для объявления переменной (`Dim`, `Public`, `Private` или `Static`).

Если переменная объявлена в теле процедуры с использованием `Dim`, ее можно использовать только внутри процедуры. В другом месте программы VBA не узнает данную переменную. Переменные, объявленные при помощи оператора `Dim` в разделе объявлений модуля, доступны во всем модуле, но не в других модулях.

Закрытые переменные

Закрытыми называются переменные объявленные при помощи ключевого слова `Private`. Это слово работает так же, как и `Dim`. Два объявления, приведенных ниже, действуют одинаково:

```
Private strLouie As String
Dim strLouie As String
```

Поскольку `Private` и `Dim` работают одинаково, при желании можно забыть о существовании `Private`. Однако можно использовать `Private` вместо `Dim`, чтобы было ясно, что переменная доступна только в данном модуле.

Открытые переменные

Объявление переменной при помощи `Public` делает ее доступной для всего проекта. Например:

```
Public intBeulah As Integer
```

Ключевое слово `Public` действует, если переменная объявлена в разделе объявлений модуля. Хотя VBA и разрешает использование `Public` в процедурах, переменные объявленные таким образом, не будут доступны за пределами процедур, в которых они объявлены.

Преимущества статических переменных

Ключевое слово `Static` используется для объявления переменной внутри процедуры, если нужно, чтобы переменная оставалась в памяти и, что более важно, сохраняла свое значение, даже если процедура не запущена.

В следующем примере переменная `intLastingVariable` работает как счетчик, сохраняя количество запусков процедуры:

```
Sub TransientProcedure()
    Dim strTransientVariable As String
    Static intLastingVariable As Integer
    strTransientVariable = Format(Now(), "Medium Time")
    intLastingVariable = intLastingVariable + 1
    MsgBox "Время " & strTransientVariable & _
        ". " & "Процедура выполнялась " & _
        intLastingVariable & " раз."
End Sub
```

В этом примере оператор `intLastingVariable = intLastingVariable + 1` добавляет 1 к значению переменной каждый раз при запуске процедуры. Если же объявить переменную `intLastingVariable` при помощи ключевого слова `Dim` вместо `Static`, при запуске процедуры переменная всегда будет иметь одно и то же значение (ноль), что сделает выполнение процедуры бессмысленным.

Статические переменные могут быть объявлены только внутри процедуры. Если нужно, чтобы все переменные оставались в памяти по завершении работы процедуры, лучше всю процедуру объявить статической. Для этого в объявлении процедуры

поместите ключевое слово Static перед Sub или Function, определяющими тип процедуры. Вот пара примеров подобного подхода:

```
Private Static Sub DoItAll ()
Static Function DontDoVeryMuch(intTimeToWaste As Integer)
```

Обратите внимание: ключевое слово Static находится в объявлении процедуры после Private или Public, если, конечно, эти определения области видимости включены в объявление.

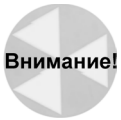
Объявление нескольких переменных в одной строке

Для экономии пространства можно объявлять в одной строке несколько переменных. Введите ключевое слово Dim и отделите объявляемые переменные запятыми. Помните, что нужно указывать тип данных для *каждой* объявляемой переменной, даже если все переменные — одного типа, например:

```
Dim intA As Integer, intD As Integer, intL As Integer
```

Можно также объявлять переменные разных типов:

```
Dim curNetWorth As Currency, datSecondTuesday As Date
```



Объявление переменных в одной строке увеличивает вероятность пропустить тип данных. Все переменные, для которых не определен тип, автоматически определяются как Variant. То есть, при использовании объявления Dim strX, strY, strZ As String переменные strX и strY будут рассматриваться компилятором как переменные типа Variant, а никак не строковые.

После объявления переменной в нее нужно вложить какую-либо информацию (внести начальную информацию в переменную, значит, *инициализировать* ее). Чтобы внести информацию в переменную, нужно *присвоить* ей значение. В переменной можно хранить различную информацию, присваивая переменной ее значение.

Присваивание значений

Чтобы присвоить переменной значение, используйте знак равенства. Например, чтобы сохранить в переменной intC цифру 3, введите

```
intC = 3
```

В VBA *операция присваивания* состоит из знака равенства между переменной (с левой стороны) и *выражения*, определяющего значение переменной (с правой стороны). В приведенном выше примере, выражение — просто цифра 3. Значения, определяемые прямо, как в примере, называются *буквальными* (literal) (см. раздел "Выразительность").

Возьмем другой пример оператора присваивания:

```
strQuote = "Не спрашивай, что твоя страна может сделать для
тебя," _
& " спроси, что ты можешь сделать для нее."
```

В данном случае, оператор присваивает текст с правой стороны от знака равенства переменной `strQuote`. Как и в предыдущем примере, информация в этом выражении состоит из буквальных значений и текста, который нужно поместить в переменную.

Однако, оператор разделен на две строки, т.е. текстовое выражение тоже должно быть разделено на две части. Знак `&` сообщает VBA, что их нужно соединить. Не важно, на сколько частей разбито выражение, VBA вычислит общее значение и *затем* присвоит его переменной.

Следует понимать, что пока оператор присваивания не выполнился, он, строго говоря, оператором не является. В математике в выражении вроде 2 + 2 = 4 объявляется, что две части выражения действительно равны. Оператор VBA же, *насильно приводит* переменную к значению выражения. Другой оператор может изменить значение переменной в любое время.

Использование переменных в операциях присваивания

Помимо литералов, можно присвоить переменной значение, включающее в себя другие переменные.

Оператор `curSalePrice = curCost * sngMargin` перемножает переменные `curCost` и `sngMargin` и присваивает это значение переменной `curSalePrice`. VBA выполняет вычисления, основываясь на значениях, содержащихся в переменных.

Использование функций в операциях присваивания

Функции и процедуры `Function` также могут использоваться в операциях присваивания, например:

```
strFavourite = InputBox("Какой Ваш любимый вкус?")
```

Любая функция или процедура `Function` возвращает значение. В данном примере функция `InputBox` выводит диалоговое окно, с определенным текстом и полем для ввода ответа. Этот ответ и будет значением, возвращаемым функцией в виде строки (подробнее о функции `InputBox` см. главу 56).

Выразительность

Выражение — набор операторов VBA, значение которых может быть высчитано и представляет собой число, строку или ссылку на объект. Выражение может состоять из одного или нескольких элементов, в любой последовательности:

- переменных, таких как `bytMonth` или `boolWinter`;
- литералов, таких как 1234 или "Это всего лишь проверка.";
- констант, замещающих выражения (см. далее в главе);
- функций VBA, например `InputBox()` или `Sqr()`;
- пользовательских процедур типа `Function`.

Если выражение содержит несколько из перечисленных элементов, они соединяются *операторами*, как знак +, или, в некоторых случаях, вложенными функциями и процедурами `Function`, заключенными в другие функции и процедуры `Function`. Если в выражении содержится более одного компонента, каждый компонент сам по себе является выражением и имеет значение.

Что содержится в переменной до того, как ей присвоено значение?

Когда VBA запускает процедуру, создается пространство для хранения переменных, и каждой из них присваивается значение "пусто". Как правило, перед использованием в операторах в переменные помещается некоторая информация. Однако вполне возможно (и это иногда используется) получить доступ к переменной, когда неизвестно, содержит ли она информацию пользователя.

Предположим, что в программе есть процедура, запускаемая при выполнении определенных условий. Предположим также, что эта процедура присваивает значение одной из переменных программы. В этом случае, чтобы проверить, выполнялась ли хоть раз первая процедура, другая процедура должна проверять, содержит ли переменная какое-либо значение. В табл. 53.3 приведены значения, которые содержатся в переменных до присвоения им значений.

Таблица 53.3. Значения типов данных по умолчанию

Тип данных	*Начальное значение*
Все числовые типы данных	0
`String` (изменяемой длины)	Строка нулевой длины ("")
`String`(фиксированной длины)	Строка определенной длины, состоящая из символа ASCII с кодом 0 (невидимый символ)
`Variant`	Пусто (специальное значение показывающее, что переменная ничего не содержит)
`Object`	Ничего (специальное значение показывающее, что переменной не присвоена ссылка на объект)

Работа с константами

Если в программе используется значение, которое *не* изменяется, это значение не нужно представлять переменной. Хотя всегда можно поместить в процедуры буквальные значения, лучше использовать константы.

Объявление констант

Для объявления констант используется оператор `Const`:

```
Const cstrPetsName As String = "Foo-foo"
Const cdtmTargetDate As Date = #6/23/01#
Const cblnUp As Boolean = True
```

Объявление констант очень похоже на объявление переменных. Разница в том, что значение константе присваивается на стадии объявления. Разрешается объявлять константы тех же типов данных, что и переменные, за исключением `Object`, определяемого пользователем и `decimal` (см. табл. 53.2).

Обратите внимание, что имя каждой константы в примере начинается с прописной буквы *c* (от constant). Этот метод используется, чтобы показать, что речь идет именно о константе, а не о переменной. Можно, конечно, воспользоваться любым другим префиксом.

Применяйте соглашения VBA и добавляйте в имена префиксы, основанные на имени пользователя или на названии проекта VBA. VBA и Visual Basic начинают константы с префикса vb, например `vbBlue` (код для цвета 16711680) или `vbKeyTab` (код клавиши <Tab>, 9). Приложения VBA имеют константы с префиксом приложения, например константа Excel `xlBarStacked` (представляет код линейной диаграммы с накоплением, 58).

Между прочим, в программе можно использовать константы, определенные VBA или приложением VBA. Для получения информации о предопределенных константах воспользуйтесь справкой или окном Object Browser (см. главу 52).

Преимущества использования констант

После объявления константы можно использовать ее имя вместо соответствующего значения. Например предположим, что существует программа, которая вычисляет размер зарплаты рабочего, исходя из размера его обуви Одним из методов написания части такой программы будет:

```
If bytSoeSize > 48 Then
    curJoesSalary 75000
End If
```

Проблемой такого подхода является "жесткое кодирование" размера зарплаты. Если инфляция приведет к росту зарплаты, придется перерыть весь код, чтобы программа работала корректно. Если значение встречается в программе более одного раза, придется внести изменения в каждом случае. Риск ошибки существенно возрастает, а работа может быть серьезно замедлена.

Ниже приведен пример кода, содержащего константу:

```
Const CcurTopSalaryStep As Currency = 75000
...
If bytShoeSize > 48 Then
    curJoesSalary = CcurTopSalaryStep
End If
```

Такое решение позволяет легко найти константу в верхней части модуля. Достаточно изменить здесь величину константы, и нужное значение будет автоматически подставлено во всей программе. Кроме того, код стал понятнее. Можно не спрашивать, "А что, собственно, означает цифра 75000?", т.к и без того видно, что Джо получит самую большую зарплату, если он носит ботинки 48 размера.

Можно, конечно, использовать вместо константы переменную. Однако переменные занимают место в памяти и, кроме того, существует риск изменения переменной "константы" в программе.

Использование констант для представления атрибутов

Константы удобны для работы с группой именованных элементов или характеристик, таких как дни недели (понедельник, вторник и т.д.) или вкус (сладкий, соленый, кислый и горький). Вместо работы с текстовыми строками в программе, проще пронумеровать каждый элемент и объявить константу, которая равна номеру, основанному на имени элемента, а затем ссылаться на элементы по имени. Ниже приведен код, который использует такую технику работы:

```
Const cbytSweet =1, cbytSalty = 2
Const cbytSour =3, cbytBitter = 4
Do While intTaste = cbytSour
    AddSweetener
    intTaste = CheckTaste()
Loop
```

Привет операторам!

В VBA *оператор* — специальный символ или ключевое слово в выражении, который объединяют два значения (подвыражения, если хотите) для получения нового результата. (Следует учитывать, что в русском языке для передачи значений слов statement (конструкция типа End If) и operator (например, символ сложения +) используется общий термин *оператор*. — *Прим.перев.*) Два значения расположены по сторонам оператора. В следующем выражении оператор сложения + складывает 3 со значением переменной `intA`:

```
intA+3
```

VBA делит операторы на три основные категории: арифметические, логические и операторы сравнения; существует несколько отдельно стоящих операторов, например, оператор *конкатенации* строк.

При использовании со строками оператор + выполняет конкатенацию, т.е. сцепляет две строки. Однако лучше использовать "настоящий" оператор конкатенации, символ `&`. VBA переводит выражение `"Меня зовут "& "Элли."` в "Меня зовут Элли."

А вот как работает оператор сравнения:

```
Tan(sngAngleA) <> 1.4
```

Символ <> — оператор "неравенства". Он проверяет, действительно ли два выражения не равны, и в зависимости от результата возвращает значение `True` или `False`. Если тангенс `sngAngleA` *не* равен 1.4, результатом выражения будет `True`. В противном случае, результатом будет `False`.

Приоритет операторов

В сложных выражениях, включающих в себя несколько операторов, VBA должен определить, какую операцию выполнять первой, какую второй и т.д. Выражение

```
intA + intB * intC
```

содержит два оператора: + (оператор сложения) и * (оператор умножения). На нормальном языке это будет звучать как "intA плюс intB умножить на intC".

Хотя символ * и является вторым оператором в выражении, выполняться он будет первым, поскольку его *приоритет* выше, чем у оператора сложения. Вначале VBA перемножит `intB` и `intC`, а затем прибавит полученный результат к `intA`. Как показывает данный пример, VBA следует установленной последовательности в расчете частей выражения, содержащих несколько операторов.

Чтобы изменить установленный порядок выполнения операторов, используйте круглые скобки. Если ввести

```
(intA + intB) * intC
```

VBA вначале сложит первые две переменные, а затем умножит значение `intC` на полученное число.

Каким же образом VBA решает, какие операторы выполнить первыми при отсутствии круглых скобок? Если выражение включает несколько операторов различных категорий, VBA высчитывает значения в каждой категории, используя следующий порядок.

1. Вначале — арифметические операторы и операторы конкатенации.
2. Затем — операторы сравнения.
3. В последнюю очередь обрабатываются логические операторы.

Внутри каждой категории VBA использует четкие правила для определения порядка выполнения операторов. Арифметические и логические операторы, а также операторы сравнения выполняются согласно приоритетов, описанных в табл. 53.4. VBA выполняет операторы сравнения слева направо. Если в выражении присутствует несколько операторов с одинаковым уровнем приоритета, VBA выполняет их слева направо.

Таблица 53.4. Операторы VBA и их приоритет*

Оператор	*Выполняемая операция*	*Подробности, комментарии*
Арифметические		
^	Возведение в степень	Возводит предшествующее значение в выражении в степень, равную следующему значению
-	Отрицание	Изменяет знак последующего значения
* или /	Умножение и деление	
\	Целочисленное деление	Делит и отбрасывает дробную часть, не округляя
Mod	Арифметический модуль	Делит и возвращает в качестве результата остаток от деления
+ или -	Сложение	
Конкатенации		
&	Конкатенация строк	
Сравнения		
=	Равенство	
<>	Неравенство	
<	Меньше	
>	Больше	
<=	Меньше или равно	
>=	Больше или равно	
Like	Сравнение строк с шаблоном	
Is	Проверка, ссылаются ли два элемента на один и тот же объект	
Логические		

Внутри каждой категории операторы будут выполняться в том порядке, в котором они следуют в списке.

Not	Логическое "нет"	см. "Когда использовать логические переменные" далее в этой главе
And	Логическое "и"	
Or	Логическое "или"	
Xor	Логическое "исключающее или"	
Eqv	Логическая эквивалентность	
Imp	Логическая импликация	

Сравнение величин

В VBA есть шесть операторов сравнения на все случаи для сравнения числовых и строчных значений и два специальных оператора сравнения — Like (для строк) и Is (для объектов). (Операторы сравнения сведены в табл. 53.4). Обратите внимание: VBA использует знак равенства =, как оператор сравнения, а также для присвоения переменным значений.

Результатом выражения, основанного на этих операторах, будет либо True, либо False. Например, в выражении, основанном на операторе <= (больше или равно):

```
intX <= 11
```

Если значение intX равно 12, результатом выражения будет False, поскольку 12 не меньше, чем 11.

Сравнение строк

Можно использовать операторы сравнения не только по отношению к числам, но и к строкам. Результатом выражения "Душистый горошек" = "Нарцисс" будет False — очевидно, что строки не равны. Однако в других случаях результаты сравнения не столь предсказуемы. Чтобы при сравнении строк получать нужные результаты, нужно понимать, какие правила использует VBA для определения большей строки.

Пока не определен другой метод, VBA использует "двоичный" метод сравнения. Две строки сравниваются на основе числовых кодов, которыми представлены символы в программе. В этой системе пунктуационные знаки имеют наименьшее значение, за ними следуют цифры, прописные буквы, строчные буквы и акцентированные символы. Поскольку коды строчных букв больше, чем у прописных, результатом обоих выражений "a" > "A" и "a" > "Z" будет True.

Чтобы воспользоваться другим, более интуитивным методом сравнения строк, включите в раздел объявлений модуля оператор Option

`Compare Text`. Если включена данная опция, строки сравниваются в алфавитном порядке, игнорируя регистр (однако акцентированные символы имеют большее значение, чем соответствующие символы без акцента). Учитывая вышесказанное, результатом всех последующих выражений будет `True`:

```
"a" = "A"
"a" < "Z"
"Aunt Hill" < "Zunt Hill"
```

При сравнении двух строк VBA начинает со сравнения первых символов каждой строки. Если символы различаются, "большей" считается та строка, в которой больший первый символ. Если первые символы одинаковые VBA сравнивает вторые символы и т.д.

Оператор `Like` сравнивает строку с шаблоном из групповых символов и используется, чтобы определить, входит ли строка в конкретный диапазон или нет. К сожалению, объем данной книги не позволяет детально описать эту функцию, но знать о ее существовании необходимо — это мощный инструмент для работы с текстом.

Использование операторов в коде

Результаты операции сравнения могут сохраняться в отдельной переменной, как правило типа `Boolean`, с помощью стандартной операции присваивания:

```
boolTheAnswerIs = 5 > 4
```

Так как 5 больше 4, результатом сравнения будет `True`, и VBA присвоит переменной `boolTheAnswerIs` значение `True`.

Ключевые слова `True` и `False`, на самом деле, являются встроенными цифровыми константами VBA и представляют собой, соответственно, значения -1 и 0. Можно присвоить результат сравнения любой числовой переменной.

Операторы сравнения часто используются в условных операторах для выбора той или иной ветви кода:

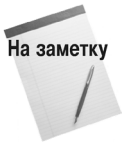
```
If intP <= intQ then
    SomethingWrong 'вызов процедуры обработки событий
End If
```

Объединение строк

Оператор конкатенации `&` служит для объединения строк. Его можно использовать вместе со строками, строковыми переменными и любыми функциями, возвращающими строковые значения. Он может применяться для создания длинных строк из нескольких строковых величин, как в данном примере:

```
strA = "Вы выбрали " & InputBox("Введите ответ:") & _
    " . Правильный ответ " & strAnswer & "."
```

После выполнения оператора переменная `strA` будет содержать следующее значение: "Вы выбрали Португалия. Правильный ответ Испания."



При создании больших строк из маленьких помните о необходимости вставки пробелов и знаков препинания в окончательный вариант строки.

Подробно о типах данных

В данном разделе описано, когда и как используются многочисленные типы данных VBA. Тип данных `Object` очень важен, однако довольно сложен, поэтому он рассматривается отдельно в главе 54.

Преобразование типов данных

Типы данных создавались для удобства, VBA хранит информацию в виде чисел. Вот почему преобразование типов не является проблемой для VBA. VBA содержит множество функций для преобразования одного типа данных в другой. Кроме того, VBA автоматически преобразует переменные одного типа данных в другой, всегда, когда это возможно и подразумевается контекстом. Например, оператор + добавляет строку к числу, если строка содержит только числа. Аналогично, если присвоить дробное значение переменной типа `Integer`, VBA автоматически округлит ее значение.

Тип Variant

Тип данных `Variant` представляет собой многоцелевой контейнер для хранения любой информации необходимой пользователю. `Variant` может хранить какой угодно тип информации, используемой в VBA, включая числовые значения, строки, дату, время и объекты. Более того, одна и та же переменная может содержать разные типы данных в разное время в рамках одной программы. Следующий код вполне понятен, однако не очень продуктивен:

```
Dim varAnythingGoes = 3
varAnythingGoes = "Я полагаю"
varAnythingGoes = #12/31/99 11:59:59 PM#
```

VBA не просто разрешает использование таких операторов, но и позволяет вычислять и отслеживать тип данных, содержащихся в переменной типа `Variant`. После выполнения последнего оператора в примере переменная `varAnythingGoes` отмечается как `Variant/Date`. Определить тип данных, содержащихся в переменной типа `Variant`, можно при помощи функции `TypeName`:

```
strVariantType = TypeName(varAnythingGoes)
```

После выполнения данного оператора переменная `strVariantType` будет содержать значение `"Date"`.

Переменные типа `Variant` очень удобны в силу своей гибкости. Вместо того, чтобы заботиться об используемых типах данных, можно объявить все переменные как `Variant` и присваивать им любые значения по мере надобности. Однако удобство использования дорого обходится с точки зрения скорости и занимаемого дискового

пространства. Подробнее см. раздел "Выбор и использование типов данных" ранее в этой главе.

Выбор числового типа данных

Если типы данных указываются явно, выбирайте минимальный из возможных, что позволит сделать программу быстрее, меньше и, в конечном счете, эффективнее. Естественно, переменная должна иметь достаточное пространство для хранения диапазона значений, которые могут в нее закладываться. Однако, если места для хранения очень много, это просто растрата памяти. В табл. 53.2 (ранее в этой главе) приведены возможные диапазоны значений для числовых типов данных.

Если программа во время работы выполняет какие-либо вычисления со значением переменной, переменная, содержащая *результат* этих вычислений, должна иметь достаточно места для его хранения. Это касается даже тех случаев, когда результат сохраняется не в той переменной, над которой выполняются вычисления.

Вот некоторые практические советы, касающиеся определенных типов данных.

- Для хранения целых чисел (без дробной части) используйте такие типы, как `Boolean`, `Byte`, `Integer` и `Long`.
- Для хранения чисел с плавающей точкой до 15 разрядов используйте типы `Single` или `Double`. Поскольку диапазон значений огромен, следует учитывать, что округление может привести к появлению ошибок, которые станут весьма значительными при операциях на различных числах. Чтобы присвоить переменной типа `Single` или `Double` значение 4.72 E 10^{-22}, используйте следующий формат (VBA определяет знак экспоненты по знаку + или - после E): `sngFloating = 4.72E-22`.
- Если нужны более точные вычисления, тип данных `Currency` позволяет хранить до 19 разрядов, а тип `Decimal` — до 29 (ни в одном из типов нет экспоненты). В текущей версии VBA, однако, тип `Decimal` не является самостоятельным, другими словами, нельзя объявить переменную этого типа. Использование `Decimal` возможно только в качестве подтипа `Variant`. Чтобы быть уверенным, что число хранится именно в этом типе, а не в каком-нибудь из типов данных с плавающей запятой, в операторе присваивания используйте функцию `CDec`.

Когда используют переменные типа Boolean

Переменные типа `Boolean` могут содержать только два значения: `True` (хранится как -1) или `False` (0). Переменные данного типа используют, когда нужно определить, какое из условий действует в данный момент. Например, переменная `boolIsOn` принимает значение `True`, если что-то включено и `False`, если выключено.

Также можно использовать переменные типа `Boolean` для определения константы со значением `True` или `False`. Имена переменных, в таком случае могут быть обычными, а константы прямо ссылаться на один из двух альтернативных вариантов, например:

```
Dim boolBellyButtonStyle As Boolean
Const Innie As Boolean = True
Const Outie As Boolean = False
If boolBellyButtonStyle = Outie Then
    TickleLightly
End If
```

Чтобы изменить значение переменной типа `Boolean` или свойства объекта на противоположное, используйте логический оператор `Not`. В Word, например, можно включить или выключить режим разметки страницы с помощью следующего кода:

```
ActiveWindow.DocumentMap = Not ActiveWindow.DocumentMap
```

Работа с переменными типа Currency

Основная причина использования переменных типа `Currency` — необходимость получения точных результатов при работе с финансовыми данными. Конечно, типы данных с плавающей точкой `Single` и `Double` могут хранить данные с дробной частью (как и тип данных `Currency`). Однако вычисления с использованием чисел с плавающей точкой приводят иногда к появлению небольших ошибок, поэтому при работе с деньгами можно случайно получить одного двух новых миллионеров.

Тип `Currency` используется не только для работы с деньгами, но и для хранения больших чисел типа `Long`, `Integer`, а также вычислений над ними с бо́льшей точностью, чем это предлагают типы данных для чисел с плавающей точкой. Переменные типа `Currency` могут содержать до 19 разрядов, 15 — в целой части и 4 — в дробной (точка фиксированная).

Не используйте тип `Currency` для форматированного вывода денежных данных. Лучше использовать функцию VBA `Format` с форматом "Currency", чтобы автоматически выводить такое числовое значение, как доллары, франки или какая-нибудь другая валюта.

Работа с переменными типа Date

Тип данных `Date` используется для работы с датами, временем или с тем и другим одновременно. VBA кодирует дату и время как число, например 35692.9201273148 (на первый взгляд, выглядит ужасно). Однако можно не обращать внимания на такие подробности и работать с датой и временем так же, как на бумаге или в текстовом процессоре.

Помните, что значение даты и времени нужно всегда вводить, заключая в символы #. Например, следующие операторы объявляют две переменные типа `Date` и присваивают им значения:

```
Dim datWeddingDay As Date, datTimeOfCeremony As Date
datWeddingDay = #4/20/99#
datTimeOfCeremony = #3:15:00 PM#
```

Как и в случае с валютой, VBA автоматически выводит числа согласно местным установкам. Выражение `Format (#10/24/89#, "Long date")` выдаст строку `"Tuesday, October 24, 1989"` в Соединенных Штатах, `"Вторник, 24 Октябрь, 1989г."` — в России.

Ввод значений даты

Значение даты можно вводить в формате, который вам больше подходит:

```
#09/1/2001#
#Sep 25, 93#
#Janua 9 1905#
```

Если редактор Visual Basic распознает введенное значение, он конвертирует его в "сокращенный" формат даты, определенный в панели управления Windows. Если год не указан, VBA использует текущий год. Изменение происходит как только курсор переходит на следующую строку, еще до запуска программы.

Значение времени

Время вводится в формате *#часы:минуты:секунды символ#*, где символ — это `AM` или `PM` (до или после полудня). Пример:

```
#10:45:00 PM#
#2:3:30 AM#
```

Незначащие нули, вроде `#01:02:03 PM#`, вводить не нужно, VBA добавит их, когда курсор перейдет на другую строку. Также не вводятся незначащие элементы — VBA их заполнит. Например, можно ввести секунды так `#0:0:23#`, VBA заменит это выражение на полное — `#12:00:23 AM#`.

Математические операции с датой и временем

Добавление или вычитание дат с помощью стандартных операторов VBA, конечно, *возможно*, но, к сожалениию, результат будет неожиданным. Например, `#3/19/2005# - #3/19/2004#` не равно "1 год", а равно `#12/30/1900#`. Объяснение заключается в способе, которым VBA хранит данные типа `Date`, но здесь он раскрываться не будет. Достаточно знать, что в VBA есть две функции `DateAdd()` и `DateDiff()`, которые позволяют выполнить все необходимые операции. За подробностями обратитесь к справочной системе.

Что касается операций со временем, вполне *можно* использовать обыкновенные арифметические операторы. Посмотрим на следующие выражения:

```
#07:15 AM# + #12:00# ' = #07:15:00 PM#
#07:15:00 AM# - #0:15 AM# ' = #07:00:00 AM#
#07:15:15 AM# + #0:0:30 AM# ' = #07:15:45 AM#
```

В примере показаны минимальные вводимые значения. Как всегда, VBA конвертирует их в полные значения: например, `#0:0:30 AM#` будет конвертировано в `#12:00:30 AM#`.

Использование строк

Учитывая, с какой легкостью VBA переводит информацию из одного типа в другой, возможно, строки понадобиться использовать немного реже. Если стоит задача

вывести на экран нестроковую величину в читабельной форме, нет необходимости преобразовывать ее в строку. Можно использовать число или дату и другую информацию, содержащуюся в переменной, в качестве аргумента функции или свойства объекта, хотя, по идее, в таких случаях используется строка.

Для чего *действительно* нужны строковые переменные так это для работы с буквами и знаками препинания, т. е. теми символами, которые нельзя представить в виде чисел.

Хороший стиль программирования предполагает явный перевод числовых величин в строки перед работой с ними как со строками, что уменьшает число ошибок и проясняет назначение операторов кода.

При присвоении значений строковой переменной использовать кавычки не обязательно. VBA и так приложит все усилия, чтобы конвертировать числа, дату или денежные величины в строки. Если программа выполнит код:

```
Dim strGString As String
strGString = #July 22, 1904#
```

переменная `strGString` будет содержать значение `7/22/1904`, пока другой оператор его не изменит. Однако лучше все-таки использовать кавычки, чтобы быть уверенным: переменная содержит именно то значение, которое подразумевалось.

Чтобы перевести числовые значения в соответствующие строковые величины и наоборот, используйте функции VBA `Chr` и `Asc`. `Chr` позволяет вставлять в строки символы, которые нельзя ввести с клавиатуры (например кавычки). `Asc` возвращает код *первого* символа в строке.

Работа с массивами

Очень часто возникает необходимость поработать с различными значениями одинакового типа как с группой. Такая группа называется *массивом* — структурой, состоящей из логически связанных элементов одного типа.

При работе с набором элементов альтернативой массиву может стать объект `Collection`, описанный в главе 54.

Предположим, имеется список чисел, представляющих цены, результаты проверки или дистанцию от Земли до некоторых астрономических объектов. Этот список написан на листе бумаги — каждый элемент в отдельной строке. Так выглядит простой массив. Вот пример:

Выигрышные номера лотереи

214236

891982

545273

000000

371453

941241

В этом списке каждый из похожих элементов имеет свое значение, но его трудно идентифицировать. Если нужно передать результат кому-то другому, можно сказать, что "это третий элемент в списке выигрышных номеров лотереи". Массивы VBA работают именно так.

Элементы массива

У каждого элемента в массиве VBA есть имя, отвечающее его позиции в списке. Чтобы получить доступ к элементу, нужно указать имя массива и *индекс* элемента — неотрицательное целое число определяющее, место элемента в массиве. Например, выражение `intLottoArray (3)` указывает на третий элемент (или четвертый, в зависимости от системы счисления) в массиве `intLottoArray`. Как можно догадаться, `int` в имени массива дает понять, что это массив целых чисел. Поэтому разумно предположить, что информация, хранимая в элементе `intLottoArray (3)`, тоже является целым числом.

Следует четко понимать следующие факты, касающиеся массивов.

- **Можно создавать массивы любого типа данных**. VBA может хранить в массивах строки, даты, денежные величины и любые числовые типы.
- **Любой массив может содержать элементы только одного типа данных**. Нельзя создать массив для хранения элементов типа `Date` и `String`. Однако тип данных `Variant` может хранить любую информацию, содержащуюся в VBA, т. е. для создания массива, способного хранить различную информацию, тип данных `Variant` вполне подойдет.

Размерность массива

Важная особенность массивов заключается в том, что они могут иметь несколько *размерностей*. Простой список — очевидный пример одномерного массива. Таблица со строками и столбцами —двухмерный массив. Массивы VBA могут иметь размерность до 60.

Объявления массивов

Массивы, как и переменные, должны объявляться перед первым использованием. Объявление массива похоже на объявление переменной с одной добавкой в операторе объявления: за именем переменной добавляются круглые скобки. Скобки могут быть пустыми, если размерность определяется позже. В противном случае, они должны содержать размер каждой размерности массива.

В этом примере объявляется массив типа `Currency`, но размерность его не определяется:

```
Public curPriceQuotes () As Currency
```

В следующем примере объявляется одномерный массив типа `Date`:

```
Dim datTimeOfImpact (12) As Date
```

Этот оператор объявляет четырехмерный массив типа `Integer`:

```
Dim intArrayOfInteger (34, 13, 29, 4) As Integer
```

Общее количество элементов в массиве равно произведению всех указанных чисел. Многомерные массивы могут содержать очень большие объемы данных.

Нумерация элементов массива и определение размера массива

Пока не определена другая система, элементы в массиве нумеруются с нуля, иначе говоря, нулевой элемент является первым элементом массива. Таким образом число, вводимое как размер массива, должно быть на 1 меньше количества элементов, которые будут в нем храниться. Если в массиве хранится 10 элементов, в качестве размера массива нужно ввести **9**.

При последующем доступе к элементам массива следует помнить о выбранной системе счисления. Обращение к `intArray (1)` на самом деле — обращение ко второму элементу массива. Если начинать отсчет элементов с нуля не удобно, можно использовать другое число, как правило 1. Чтобы все индексы массивов текущего модуля начинались с 1, добавьте оператор `Option Base 1` в разделе объявлений (перед процедурами). Этот оператор действует только на массивы модуля, поэтому, если нужно, чтобы все массивы проекта начинались с индекса 1, поместите этот оператор в начале каждого модуля проекта.

Объявление статических и динамических массивов

При определении размера массива во время объявления размер остается фиксированным, т.е. программа не сможет сделать его больше или меньше. Чтобы объявить *статический массив*,включите число элементов в каждой размерности массива в скобки в объявлении массива.

Статические массивы полезно использовать, если размер массива не будет изменяться в течение программы.

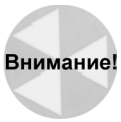
Чтобы объявить *динамический* массив, не следует объявлять размер массива в круглых скобках. *Динамический массив* используют если:

- невозможно определить размер массива перед запуском программы;
- известно, что размер массива будет изменяться в течении программы;
- необходимо освободить память для других переменных после использования массива. Большие массивы занимают большой объем памяти, который можно освободить если массив динамический.

Динамический массив не может хранить информацию, пока не определен его размер, для чего используется оператор `ReDim` (от "redimension" — переопределение массива), как в следующем примере:

```
Redim datBirthdays (intNumberOfBirthdays - 1)
```

Переопределять размер и размерность массива можно сколько угодно раз.



В большинстве случаев переопределение массива уничтожает его текущее содержимое.

Адресация элементов массива

Для работы с определенным элементом массива в коде, можно перебрать элементы массива с помощью *индекса*, заключенного в круглые скобки. Индекс содержит целую величину для каждой размерности массива. Например, выражение `strSayings (4,6)` определяет строковую информацию в строке `4`, столбца `6`, в двухмерном массиве строк.

Пользуясь этой системой, можно использовать элементы массива как обыкновенные переменные и выполнять следующие действия.

- Присваивать значение элементу массива. В этом примере значение присваивается элементу трехмерного массива `Currency`: `curBigDough (5,8,19) = 27.99`
- Присваивать значение элемента массива переменной: `datThatDate = datTheseDates (25, 10)`
- Использовать значение элемента массива в выражениях: `intA = 35 * (intB + intCounts (3,2))`

Можно использовать в качестве индекса буквальные значения, а можно использовать переменные, что дает возможность получить доступ к нужному в данный момент элементу массива. Последняя строка кода демонстрирует пример такого подхода:

```
Dim strTodaysFortune (29)
Dim intUserChoise As Integer
...' Здесь должен быть код, инициализирующий массив
intUserChoise = InputBox ("Чтобы узнать свою судьбу, введите"
_
   & "число от 1 до 30")
msgBox (strTodaysFortune (intUserChoise - 1))
```

Встроенные функции и операторы

Перед созданием пользовательских функций с нуля, нужно убедиться в том, что не будет изобретаться велосипед. VBA обладает набором встроенных команд, которые могут справиться с большим количество обычных задач. К сожалению, в данной книге

все эти функции рассмотрены не будут. Единственное, что можно порекомендовать — обратитесь к файлам справки VBA и справочникам по VBA и Visual Basic.

Между прочим, хотя встроенные функции рабочего листа Excel и не являются встроенными функциями VBA, их можно использовать в любой программе VBA, поместив в проект ссылку на библиотеку объектов Excel. В главе 57 описано, как добавлять ссылки на объекты, а в главе 30, объясняется как работать с функциями Excel в VBA.

Где находятся встроенные команды

VBA предоставляет для работы три типа встроенных команд.

- **Операторы.** Хотя термин "оператор" обычно подразумевает законченную инструкцию программы, VBA также понимает под операторами ключевые слова для выполнения некоторых рутинных операций. Некоторые из ключевых слов действуют как завершенные операторы. Например, оператор `Beep` задействует динамик компьютера. (Не следует слишком издеваться над ним.) Другие ключевые слова могут использоваться как часть целого оператора. Например, оператор `ChDir` (изменить директорию) не действует, пока в него не добавлен аргумент, определяющий директорию или папку, в которую нужно перейти: `ChDir("\Documents about Dreams")`.
- **Функции.** Встроенные функции работают так же как процедуры типа `Function`, т.е. возвращают значение. Часто функции используются для передачи данных переменной, как в примере с функцией тангенса `Tan`: `dblTangent = Tan(dblAnyOldAngle)`. Функции также используются для передачи значений в более сложных выражениях или в условных операторах, как, например: `If Tan (dblAcuteAngle) < 45 Then`.
- **Методы** встроенных объектов. Интересным методом этой группы является метод `Print` объекта `Debug`. Его применяют для перенаправления вывода в окно Immediate редактора Visual Basic, используя оператор вроде такого: `Debug.Print(strMessageFromMars)`.

В табл. 53.5 приведены примеры встроенных команд VBA, разделенных на три категории (операторы, функции и методы).

Таблица 53.5. Примеры встроенных функций, операторов и методов

Команда	*Тип*	*Действие*
`Randomize`	Оператор	Инициализирует генератор случайных чисел
`Sqr (number)`	Функция	Возвращает квадратный корень из переменной `number`
`Format (string)`	Функция	Форматирует строку согласно требованиям

		пользователя
`Date`	Оператор	Устанавливает системную дату
`Date`	Функция	Возвращает существующую системную дату
`Err.Raise`	Метод объекта `Err`	Генерирует ошибку при выполнении по номеру

Категории встроенных команд

Встроенные команды VBA делятся на следующие категории.

- **Форматирующие информацию.** Функция VBA `Format` форматирует любой из встроенных типов данных для отображения на экране или вывода на печать в соответствии с определенным шаблоном. С помощью функции `Format` можно определить в каком виде вывести переменную типа `Date`, которую VBA хранит как непонятное простому смертному число. Существует большое количество форматов вывода дат, такие как, например, 11/09/99 или пятница, июль 9, 2001. Если ни один из встроенных форматов не подходит, пользователь может создать свой.
- **Конвертирующие информацию.** Хотя VBA конвертирует данные и самостоятельно, по ходу выполнения, в VBA существуют также функции для открытого преобразования. Они используются для подстраховки автоматических преобразований VBA, для преобразований, которые VBA не может делать самостоятельно, и для того, чтобы писать самодокументируемый код. Примерами таких функций являются: `Cbyte`, `Fix`, `Hex` и `Val`.
- **Обработки строк.** VBA содержит большой набор операторов и функций для форматирования текстовых строк и работы с ними.
- **Работы с датой и временем.** VBA предлагает массу функций для определения текущей даты и времени, проведения вычислений с датами и извлечения из даты переменных, представляющих интерес (т.е. года, дня недели или часа).
- **Взаимодействия с пользователем.** Функции `MsgBox` и `InputBox` выводят простые диалоговые окна, которые позволяют проинформировать о чем-либо пользователей и получить от них отклик.
- **Выполнения математических и финансовых операций.** VBA содержит массу функций для работы с числами. Они могут совершать операции от самых простых (типа возвращения абсолютного значения) до сложных алгебраических и тригонометрических вычислений. Если ватерлиния уже под водой, возможно, финансовые функции VBA помогут удержать бизнес на плаву.
- **Работы с различными рутинными операциями.** Можно использовать большое количество команд для работы с файлами на диске, для вставки и проверки строк реестра Windows, а также работы с переменными и т.д.

Контроль потока

Управляющие структуры (control structures) — это операторы кода, которые, основываясь на условии действующем в данный момент работы программы,

определяют, какая процедура должна выполняться следующей. VBA предоставляет большой выбор управляющих структур, которые делятся на три основные группы: условные операторы, циклы и оператор `With`.

- **Условный оператор** определяет, какая из ветвей кода будет выполняться в зависимости от того, имеет условие значение `True` или `False`. Условные операторы VBA включают `If...Then` (при нескольких перестановках) и `Select Case`.
- **Цикл** раз за разом выполняет фрагмент кода либо определенное количество раз, либо до выполнения определенного условия. Когда известно число прохождений цикла, используйте цикл `For...Next`. Если код должен проверить какое-либо условие, чтобы определить, продолжать цикл или нет, используйте оператор `Do...Loop` (имеющий разные модификации). Для выполнения определенных действий над каждым объектом в библиотеке имеется цикл `For Each...Next`.
- **Оператор** `With` позволяет выполнить ряд действий над объектом, не называя объект во время каждого из них.

Управляющие операторы придают программе четкость, организованность, *структуру*. Они позволяют достаточно легко отслеживать ветви выполнения программы.

Из чего состоит структура управления

Что делает структуру управления "структурой" так это то, что она представляет собой не один оператор, а их набор. Оператор `If...Then` может служить моделью всех структур управления:

```
If a < b Then ' Если a меньше b, тогда
    b = a ' присвоить переменной b значение a
    a = c ' и присвоить переменной a значение c
End If ' Все, переход к дальнейшему выполнению программы
```

Основа структуры — открывающий оператор, который определяет тип и устанавливает условие, и оператор, который завершает структуру. Всю работу выполняют операторы, заключенные в теле структуры.

Все структуры управления имеют подобную структуру, за тем исключением, что в некоторых из них условие устанавливается последним оператором, а не первым.

Вложенные управляющие структуры

Когда речь идет об управляющих структурах, подразумевают, что *вложенные* находятся внутри других структур:, между начальным и завершающим операторами. VBA переходит к выполнению второй структуры еще до завершения первой. Вложенные структуры применяются для решения многих сложных проблем программирования. Можно создать столько уровней вложения, сколько необходимо.

В следующем примере цикл `Do While...Loop` является *вложенной* структурой по отношению к `If...Then`, а другая структура `If...Then`, в свою очередь, вложена в `Do While...Loop`:

```
If a < b Then
    Do While b > c
       b = b - 1
```

```
        If c > d Then
            d = a
        End If ' Завершение внутренней структуры If...Then
    Loop ' Завершение Do...While
End If ' Завершение внешней структуры If...Then
```

Выбор пути: использование условных операторов

Правильный выбор — основное понятие как в программном обеспечении, так и в реальной жизни. Хотя структуры выбора и просты по конструкции, они являются одним из самых мощных средств программирования. Они определяют, какой из блоков кода нужно исполнить, в чем, как говорил Роберт Фрост (Robert Frost), "и заключается все отличие".

Чтобы определить, выполнять или нет какой-либо блок кода, три управляющие структуры VBA просчитывают *условное выражение* написанное пользователем. В эти структуры входит цикл `Do...Loop`, условные операторы `If...Then` и `Select Case`. Далее речь пойдет об условных выражениях, используемых всеми тремя структурами. Такие управляющие структуры, как `For...Next` и `For Each...Next`, условные выражения не используют.

Как работают условные выражения

Структуры `If...Then`, `Select Case` и `Do...Loop` принимают решение о том, какую ветвь кода выполнять, основываясь на простом тесте: является ли значением условия `True` или `False`. Условием может быть любое выражение VBA. (Помните, что 0 соответствует в VBA значению `False`. Все другие значения соответствуют `True`.) Очень часто условные выражения используются вместе с оператором сравнения, который сравнивает значения двух подвыражений. Набор операторов сравнения VBA обсуждался ранее в этой главе. Однако составить общее представление можно, просмотрев простые операторы в табл. 53.6.

Таблица 53.6. Примеры условных выражений

Выражение	*Расшифровка*
`a < b`	Элемент `a` меньше `b`
`b = c`	Элемент `b` равен `c`
`colTBears("Henry") Is objCurrentBear`	Объект, хранимый в библиотеке `colTBears` как "`Henry`", тот же, на который ссылается переменная `objCurrentBear`
`sqr (1/x * 29.3234) >= CDbl (strNumber) + 12`	Квадратный корень из значения `(1/x * 29.3234)` (больше или равно числовому значению `(strNumber) + 12`

Использование в условиях логических операторов

Логический оператор рассчитывает значение каждого из подвыражений как `True` и `False` и затем сравнивает их согласно набору правил для получения конечного результата (также `True` или `False`). Наиболее важные логические операторы или те, действие которых понять проще — это `And`, `Or` и `Xor`. В табл. 53.7 показано, как они работают.

Таблица 53.7. Логические операторы

Оператор	*Когда возвращает True*	*Примеры*	*Результат*
`And`	Только если значения обоих подвыражений равны `True`	`3 * 2 = 6 And 12 > 11` `2 + 2 = 4 And 4-2 = 1`	`True` `False`
`Or`	Если значение любого из подвыражений равно `True`	`10 > 20 Or 20 > 10` `5 < 4 Or 6 < 5`	`True` `False`

`Xor`	Если значение одного подвыражения равно `True` (возвращает `False`, если оба выражения имеют значение `True` или `False`)	`5 + 5 < 9 Xor 5 + 5 = 10`
		`5 + 5 > 9 Xor 5 + 5 = 10`
		`5 + 5 < 9 Xor 5 + 5 <> 10`
			`True`
			`False`
			`False`

Использование операторов If...Then

Наиболее часто употребляемыми условными операторами являются `If...Then` и его вариации (`If...Then...Else` и `If...ElseIf`).

Основная форма: If...Then

В общем случае оператор `If...Then` выполняет блок кода, если условие равно `True`, и не делает ничего, если условие равно `False`. Синтаксис оператора:

```
If условие Then
(операторы,, которые должны исполняться, если условие истинно)
End If
```

Когда VBA выполняет оператор `If...Then` при истинном `условии` (равном `True`), то выполняются операторы между `If` и `End If`. Если `условие` ложно (равно `False`), операторы пропускаются, происходит переход к следующему оператору программы. Ключевое слово `Then` располагается в одной строке с `If` и условным выражением. Необходимо завершать оператор при помощи `End If`, иначе VBA выдаст сообщение об ошибке.

Что может идти в одной строке с If...Then

Когда структура `If...Then` содержит только один оператор, который должен выполняться при истинном условии, можно расположить его в одной строке с `If...Then`. В этом (и только в этом) случае оператор `End If` не нужен, на самом деле, его даже нельзя ставить. Например:

```
If curPrice > 20 Then MsgBox "Внимание! Цена чересчур
высокая!"
```

Этот код аналогичен:

```
If curPrice > 20 Then
   MsgBox "Внимание! Цена чересчур высока!"
End If
```

Использование операторов If...Then...Else

Если необходимо, чтобы программа выбирала между двумя участками кода в зависимости от условия, используйте оператор `If...Then...Else`. В этом случае, один участок будет выполняться при значении условия, равном `True`, а другое — если значение равно `False`. Оператор имеет следующий синтаксис:

```
If условие Then
(операторы, которые выполняются, если условие истинно (True))
Else
(операторы, которые выполняются, если условие ложно (False))
End If
```

Если `условие` истинно, VBA выполняет первый участок кода и пропускает остаток структуры вплоть до строки `End If`. С другой стороны, если `условие` ложно, выполняются только операторы из блока `Else`.

В следующем примере условное выражение проверяет является ли элемент управления формы VBA кнопкой. Если да, цвет кнопки меняется на красный. Все другие элементы управления окрашиваются голубым.

```
If TypeOf ctlCurrentControl Is CommandButton Then
   ctlCurrentControl.BackColor = &HFF& 'Красный
Else

   ctlCurrentControl.BAckCiolor = &HFFFF00 'Голубой
End If
```

В примере переменная `ctlCurrentControl` уже содержит ссылку на определенный элемент управления формы. Ключевое слово `TypeOf` позволяет проверить, является ли объект, на который ссылается переменная, объектом определенного типа, здесь — объектом `CommandButton`.

Подробнее о формах и элементам управления см. главу 56.

Особенности If...Then

Часто возникает необходимость проверить два или более условий, чтобы выбрать, какой участок кода должен быть выполнен. Этот процесс занимает много времени. Например, если человек пишет книгу (возьмем абсолютно случайный пример), он может думать следующим образом: "Если я закончу ее в нужный срок и если до тех пор у меня не закончатся деньги и если курс гривны к песо не поменяется, можно будет съездить в октябре на две недели в Мехико. Если же гривна упадет, придется довольствоваться Шепетовкой". В зависимости от выполняемых проверок, в структуру `If...Then` может понадобиться вставить несколько `Else If` или создать несколько уровней вложенных операторов `If...Then`.

Особенности. Часть I: использование операторов If...ElseIf

Ключевое слово `ElseIf` применяется для проверки нового условия только в случае, если нужно выполнить какие-либо операторы, когда первое условие *ложно*. Синтаксис оператора следующий:

```
If условие1 Then
(операторы, которые выполняются, если условие1 истинно)
ElseIf условие2
(операторы, которые выполняются, если условие1 ложно, а
условие2 истинно)
ElseIf условие3
(операторы, которые выполняются, если условие1 и условие2
ложные, а условие3 истинно)
...(дополнительные операторы ElseIf)
Else ' необязательный оператор
(операторы, которые выполняются, если все условия ложные)
End If
```

Количество операторов `ElseIf` не ограничено, оператор `Else` необязателен.

В структуре `If...ElseIf` выполняются только операторы, связанные с первым истинным условием. После их выполнения оставшиеся `ElseIf` и `Else` пропускаются. Оператор `Else` необязателен, однако если он используется, то должен располагаться последним в структуре, что, собственно, очень логично.

Особенности. Часть II: вложенные операторы If...Then

Вложенные операторы `If...Then` — разновидность оператора `If...ElseIf`. Они используются для проверки второго условия, чтобы решить, исполнять ли участок кода, но в случае, если первое условие истинно. Два вложенных оператора `If...Then` похожи на фразу, "Если X равен `True` и Y равен `True`, выполните A, B и C".

Можно вкладывать операторы `If...Then` в любые операторы `If...Then...Else`, `If...ElseIf` и наоборот в любой последовательности. Ниже схематически изображена пара вложенных операторов `If...Then`:

```
If условие1 Then
    If условие2 Then
    (операторы, которые выполняются, если условие1 и условие2
истинны)
    ElseIf условие3 Then
    (операторы, которые выполняются, если условие1 и условие3
истинны, а условие2 ложно)
    End If ' Завершает внутренний блок If...Then
(другие операторы, которые выполняются, если условие1 истинно,
несмотря на условие2)
Else
(операторы, которые выполняются, если условие1 ложно)
End If
```

Следующий простой пример вложенных операторов `If...Then` выводит сообщение с поздравлением за полученные высокие оценки при условии посещения занятий:

```
If sngGPA > 3.5 Then
    If sngUnits > 10 Then
       MsgBox "Декан внес Вас в свой список!"
    End If
End If
```

Особенности. Часть III: использование логических операторов в условиях

Использование логических операторов в условиях — элегантная альтернатива оператору `ElseIf` и вложенным структурам `If...Then`, но только если нужно выполнить одну из нескольких ветвей кода. Например, вернемся к коду предыдущего раздела. Той же цели можно достичь, применив один оператор `If...Then`:

```
If sngGPA > 3.5 And sngUnits > 10 Then
    MsgBox "Декан внес Вас в свой список!"
End If
```

Использование операторов Select Case

Операторы `If...ElseIf` и `If...Then` идеально подходят для проверки разных выражений перед тем, как принять решение, какой участок кода должен быть выполнен. Если, однако, нужно проверить одну и ту же *величину* при разных условиях, на помощь придет оператор `Select Case`, имеющий следующий синтаксис:

```
Select Case величина
Case значение1
(операторы, которые выполняются, если величина соответствует
значению1)
Case значение2
(операторы, которые выполняются, если величина соответствует
```

```
значению2)
... ' дополнительные операторы Case
Case Else ' необязательно
(операторы, которые выполняются, если величина не
соответствует ни значению1, ни значению2)
End Case
```

Условия проверки в операторах Select Case

Структура `Select Case` не использует прямо условные выражения, подобные описанным ранее. Вместо этого каждое выражение разбивается на две части, представленные в предыдущем примере *величиной* и *значением*. Если взять выражение

```
a + b > c
```

то *величиной* будет выражение в левой части оператора сравнения (`a + b`), а *значением* — вся правая часть, включая оператор (`> c`).

Пример оператора Select Case

Рассмотрим пример реального оператора `Select Case`. В последующем коде `objRollOfFilm` — объект, представляющий фотопленку, и имеет свойство `Type`, которое соответствует типу пленки:

```
Select Case objRollOfFilm.Type
Case "Слайдовая"
    intCountSlide = intCountSlide + 1
Case "Цветная"
    intCountColorPrint = intCountColorPrint + 1
Case "Черно-белая"
    intCountBWPrint = intCountBWPrint + 1
Case Else
    MsgBox "Неизвестный тип"
End Case
```

Здесь программу интересует только значение, возвращаемое свойством `Type`, которое следует сравнить с несколькими возможными вариантами. Оператор `Select Case` хорошо подходит для данной ситуации. Первый оператор `Case` в примере равносилен оператору `If objRollOfFilm.Type = "Слайдовая" Then`. То есть, если свойство объекта `Type` равно `Слайдовая`, программа переходит к следующему оператору, в противном случае — к оператору `Case` номер 2. Обратите внимание, что ожидаемый оператор = отсутствует во всех операторах `Case`. Так происходит потому, что в операторе `Select Case` равенство подразумевается под проводимым сравнением.

Оператор Case Else

Если значение свойства `Type` не равно ни одному из операторов `Case`, выполняется оператор `Case Else`, всегда последний в структуре `Select Case`. В данном случае `Case Else` выводит сообщение об ошибке. Оператор `Case Else` необязателен, поскольку возможен вариант, когда пользователь не пожелает что-либо предпринимать, если не одно из значений не подходит. Однако, стоит включить оператор `Case Else`, если нужно проинформировать пользователя о неожиданных значениях, хранимых программой.

Подробно о значениях оператора Case

Значения оператора `Case` в предыдущих примерах — простые проверки равенства. Однако, с помощью оператора `Case` можно проводить и более сложные проверки. Проще продемонстрировать это с помощью чисел. Предположим, оператор `Select Case` начинается со строки:

```
Select Case intPatientAge
```

В этом случае проверяемое значение — целочисленная переменная, представляющая возраст пациента в больнице. Данное значение будет проверяться в примерах, приведенных ниже.

- Можно сравнить значение с диапазоном:

```
Case 18 To 35
    Messages("Молодой").Print
```

Обратите внимание на оператор `To` между значениями, определяющими границы диапазона. Диапазон включает оба граничных значения и все значения между ними.

- Можно производить сравнения, используя операторы, отличные от =:

```
Case Is > 65
    Messages("Пожилой").Print
```

В этом случае следует использовать ключевое слово `Is` перед оператором сравнения. Вообще, не обязательно даже печатать `Is` — VBA добавит ключевое слово автоматически.

- В одном операторе `Case` можно проводить несколько сравнений:

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

    Messages("Напоминание о прививке").Print
```

Сравнения разделяйте запятыми. Оператор `Case` с несколькими сравнениями аналогичен выражению, построенному на операторах `Or`. Если величина проходит любое сравнение, будут выполнены следующие операторы.

Повторение операторов с использованием циклов

Структуры циклов используются для выполнения одного и того же участка кода несколько раз. Это основная операция, позволяющая выполнять множество математических вычислений, извлекать информацию и выполнять повторяющиеся действия с каждым элементом группы.

VBA имеет три основных типа циклов, описание которых приведено в табл. 53.8.

Таблица 53.8. Циклы

Тип цикла	*Принцип работы*
`Do...Loop`	До тех пор, пока условие истинно
`For...Next`	Определенное количество раз
`For Each...Next`	Для каждого объекта в библиотеке

При работе с вложенными циклами помните, что внутренний цикл завершается до завершения шага внешнего.

Циклы Do

Различные виды оператора `Do...Loop` предназначены для повторения участка кода до тех пор, пока не выполнено условие. Чтобы определить, продолжать цикл или нет, оператор `Do...Loop` вычисляет условное выражение описанного ранее и используемого в операторах `If...Then` типа. Структуры `Do...Loop` достаточно распространены. Они используются в следующих случаях.

- Вывод сообщения об ошибке, пока пользователь не введет соответствующее значение в диалоговом окне.
- Чтение данных из файла на диске до тех пор, пока не достигнут конец файла.
- Поиск и подсчет количества раз, которое меньшая строка встречается в большей.
- Остановка программы на определенный период.
- Выполнение операций над всеми элементами массива.
- В сочетании с оператором `If...Then` выполнение действий над элементами массива или библиотеки, которые отвечают определенным условиям.

Типы операторов Do...Loop

VBA предлагает пять разновидностей оператора `Do...Loop`, но принцип их действия очень похож (табл. 53.9).

Таблица 53.9. Циклы Do...Loop

Оператор	*Описание*
`Do...Loop`	Постоянно повторяет набор операторов, выходя из цикла при выполнении условия по команде `End Do`
`Do While...Loop`	Начинает и повторяет набор операторов, если условие истинно
`Do...Loop While`	Выполняет набор операторов и повторяет его до тех пор, пока условие истинно
`Do Until...Loop`	Начинает и повторяет набор операторов, только если условие ложно
`Do...Loop Until`	Выполняет набор операторов один раз и повторяет его, пока условие ложно

Использование оператора Do While...Loop

Цикл `Do While...Loop` имеет следующий синтаксис:

```
Do While условие
(операторы, которые выполняются, если условие истинно)
Loop
```

При выполнении оператора `Do While` VBA проверяет *условие*. Если условие ложно, остаток цикла игнорируется и выполнение программы продолжается со следующей строки после оператора `Loop`. Если условие истинно, VBA выполняет операторы, содержащиеся в цикле. По достижении оператора `Loop`, программа переходит к оператору `Do While` и вновь проверяет условие.

Как правило, один или несколько операторов в цикле могут изменять значение *условия*, т.е. значение может стать ложным. В таком случае VBA прерывает цикл и пропускает его. Однако, если *условие* истинно, цикл продолжается.

Процесс продолжается до тех пор, пока *условие* не станет ложным. Другими словами, ограничений на количество повторений цикла нет. Предположим, существует вечный источник питания и сверхнадежный компьютер, тогда цикл будет выполняться вечно, пока *условие* истинно.

Пример Do While...Loop (вообще-то два примера)

Следующий пример основан на двух операторах `Do While...Loop` и демонстрирует смену разрядов в числе. Практической пользы он не несет, но наглядно демонстрирует, как работают циклы `Do`:

```
Sub ReverseTheDigits()
Dim intOriginalNumber As Integer
Dim intOneDigit As Integer, strBackwardsNumber As String

Do While intOriginalNumber < 10
    intOriginalNumber = _
        InputBox("Введите целое число, большее 9.")
Loop

Do While intOriginalNumber
    intOneDigit = intOriginalNumber Mod 10
    strBackwardsNumber = strBackwardsNumber & intOneDigit
    intOriginalNumber = int(intOriginalNumber / 10)
Loop

MsgBoxstrBackwardNumber
End Sub
```

Объяснение примера

Первый цикл `Do` проверяет значение числа, введенного пользователем, с целью убедиться, что введено положительное число и в нем, как минимум, два разряда (иначе что менять?). Когда программа начинает цикл, значение переменной `intOriginal Number` равно нулю, поскольку ей не было присвоено другое значение. Нуль меньше 10, следовательно, условие истинно, и VBA выполняет цикл.

Цикл содержит один оператор — окно ввода, запрашивающее у пользователя подходящее число. Далее оператор `Loop` отсылает VBA к началу цикла, где происходит проверка введенного числа. Цикл прерывается, когда введено подходящее

число. (Обратите внимание: в данном примере отсутствуют серьезные проверки, например, является ли введенное значение целым числом и не превышает ли оно границу, определенную для целых чисел.)

Когда введено подходящее значение, VBA переходит к следующему циклу. Условием для выполнения этого цикла является значение переменной больше нуля. Поскольку любое значение, не равное нулю, является истинным, вместо `Do While intOriginalNumber > 0` можно написать просто `Do While intOriginalNumber` — оба оператора будут работать одинаково.

Простая трехшаговая процедура разделяет разряды числа справа налево и использует их в обратном порядке для создания новой строки. Необязательно знать, как работает код, чтобы понять назначение циклов, однако следует учесть некоторые моменты.

- В первой строке используется оператор `Mod` деления числа на 10 и присваивания остатка переменной `intOneDigit`. Так как происходит деление на 10, остаток — последнее (самое правое) число от первоначального значения.
- Во второй строке число, полученное в первой строке, добавляется в конец создаваемой строки.
- В третьей строке число вновь делится на 10, на этот раз сохраняя результат деления в исходной строке — таким образом отсекается правая цифра. Обратите внимание: перед присвоением значения переменной использована функция `Int`. Это необходимо, поскольку иначе VBA округлит результат, что может изменить первоначально заданные цифры.
- По мере выполнения цикла переменная `intOriginalNumber` станет равной 0 (после того, как будут обработаны все цифры, ведь любая цифра, деленная на 10 меньше 1, а значит, функция `Int` отбросит дробную часть). В VBA нуль — это `False`, поэтому цикл заканчивается и программа отображает обратное число.

Другие операторы Do

Различия в операторах `Do While...Loop` легко понять, если изучить основную форму. В этом разделе рассматриваются три альтернативных цикла `Do`. (Информация о четвертом находится в разделе "Когда использовать `Do` без `While` или `Until`".)

Do...Loop While

Разница между операторами `Do While...Loop` и `Do...Loop While` проста: в `Do While...Loop` условие находится в начале цикла, а в `Do...Loop While` оно расположено в конце.

В структуре `Do While...Loop` цикл начинается, если условие истинно. Если же условие ложно, операторы в цикле никогда не выполнятся. В структуре `Do...Loop While`, наоборот, цикл всегда выполняется, по крайней мере, один раз, когда программа впервые выполняет код. При проверке условия цикл повторяется, пока условие истинно. Структура `Do...Loop While` используется, когда цикл содержит оператор, устанавливающий значение условие перед его проверкой.

Структура `Do...Loop While` применяется и в другом случае, при выполнении действий над объектом (таким, как строка или массив), который может содержать больше одного элемента. Если известно, что объект содержит, по меньшей мере, один

элемент, возникает необходимость выполнения цикла как минимум один раз (а на самом деле столько раз, сколько элементов содержится в объекте).

Циклы Do Until

Операторы `Do While...Loop` и `Do Until...Loop` функционально равны. То есть можно выполнять одни и те же операторы с любым из них, меняя лишь условные выражения для того, чтобы выполнялось обратное условие. Если условие для оператора `Do While` выглядит как `A = B`, то оператор `Do Until` с условием `A <> B` будет работать абсолютно идентично. Операторы `Do...Loop While` и `Do...Loop Until` также являются взаимодополняющими.

Выход из цикла при помощи оператора Exit Do

Условие цикла `Do` предоставляют VBA информацию, необходимую для принятия решения о продолжении работы цикла. К сожалению, в реальном программировании не всегда все гладко. Иногда изменение условия в теле цикла требует досрочного выхода из него. Для это в VBA существует оператор `Exit Do`. Работая только в структурах `Do`, `Exit Do` прерывает цикл, переходя к оператору, следующему сразу за циклом. В следующем примере выполняется конкатенация двух строк, однако, если переменная содержит более одного символа, цикл прекращается:

```
Do While strA <= "Z"
    If len (strA) > 1 Then
       Exit Do
    End If
    strB = strB & strA
    strA = GetNextCharacter
Loop
```

Как правило, оператор `Exit Do` появляется в связке с оператором `If...Then` или `Select Case`, вложенным в цикл. Таким образом, цикл повторяется пока не произойдет какое-либо событие или величина не превысит допустимое значение. Также можно использовать `Exit Do` как средство отладки, чтобы пропускать временно код, не комментируя его.

Когда использовать Do без While или Until

Стандартные операторы `Do...While/Until...Loop` могут проверять условие в начале или конце цикла. Как быть, если требуется проверка значения в самом тексте?

В таком случае используется оператор `Do...Loop`, без `While` или `Until`. Этот способ предполагает наличие оператора `If` или `Select Case`, вложенного в цикл, а одна или несколько ветвей оператора условия включают оператор `Exit Do`, что позволяет программе выйти из цикла при выполнении некоторого условия.

Вот как схематически работает оператор `Do...Loop`:

```
Do
(операторы, которые выполняются с каждым шагом цикла)
    If условие Then
        Exit Do
    End If
(операторы, которые выполняются, только если цикл
продолжается)
Loop
```

Как видно, данный метод позволяет выполнить некоторые из операторов цикла, независимо от того, выполнено условие или нет. Его также можно использовать, если цикл нужно прервать при разных условиях. Часто используется несколько параметров для оценки введенного пользователем, как в следующем коде, где структура `Do...Loop` повторяет цикл, пока пользователь не введет подходящий вариант:

```
Sub GetAnswer()

strAnswer = InputBox("Введите Ваш вариант (A-E)")

Do
    If strAnswer = "" Then
        strAnswer = InputBox("Вы ничего не ввели." _
        & "Пожалуйста, введите букву от A до E")
    ElseIf Len(strAnswer > 1) Then
        strAnswer = InputBox("Ваш ответ должен быть" _
        & " одной буквой. Пожалуйста, попытайтесь " _
        & "снова")
    ElseIf strAnswer < "A" Or strAnswer > "E" Then
        strAnswer = InputBox("Вы ввели неправильный" _
        & " символ. Введите букву от A до E")
    Else
        Exit Do
    End If
Loop

End Sub
```

Эта программа выполняет оператор `Exit Do`, только когда выполнены все три условия.

Цикл For...Next

Если известно, сколько раз должен выполняться цикл, можно использовать цикл `For...Next`, указав число прохождений цикла и определяя `начальное` и `конечное` значения (выраженные целыми числами, переменными и даже сложными выражениями). По мере выполнения цикла переменная-`счетчик` сохраняет количество проходов. Когда счетчик становится равным `конечному` значению, цикл прекращается.

Упрощенно, синтаксис `For...Next` выглядит следующим образом:

```
For счетчик = начальное значение To конечное значение
(операторы, которые должны выполняться во время каждого
прохода цикла)
Next счетчик
```

Следующий пример использует окно Immediate для вывода сообщения во время каждого прохода цикла (в редакторе Visual Basic можно открыть окно Immediate с помощью комбинации клавиш <Ctrl+G>):

```
Sub CountToTen ()
Dim j As Integer
    For j = 1 To 10
        Debug.Print "Это проход №" & j
    Next j
End Sub
```

В предыдущем примере и `начальное`, и `конечное` значения являются числами. Когда цикл начинается, значение переменной `j` устанавливается равным 1, другими словами, `начальное значение` передается счетчику. При каждом проходе цикла оператор `Next j` увеличивает `j` на 1 и переходит в начало цикла. Когда `j` становится равным 10, цикл прекращается.

Подробнее о циклах For...Next

Код должен быть понятным, поэтому в качестве начального значения для цикла `For...Next` желательно использовать 1, если, конечно, нет серьезных причин для применения других значений. Такие причины заключаются в следующем. Одна из них — использование значения счетчика в самом цикле (но не изменение его). Если цикл работает с последовательно пронумерованными элементами, можно использовать в качестве начального и конечного значения реальные значения индекса элементов. Как правило, при работе с массивами начальное значение выставляется равным 0, что и описано в следующем разделе.

В операторе `Next`, который завершает цикл `For...Next`, переменная счетчика на самом деле не нужна — ключевое слово `Next` автоматически вычисляет следующее значение счетчика и отправляет VBA в начало структуры. Однако следует все-таки включать переменную счетчика в оператор `Next`. Тогда при работе с несколькими вложенными циклами можно будет ясно видеть, какому циклу принадлежит каждый оператор `Next`.

Не изменяйте значение счетчика в теле цикла `For...Next`. Поскольку счетчик — это переменная, возможен (и иногда необходим) вариант, когда его значение изменяется. Лучше не поддаваться искушению — цикл может пропустить нужные шаги или стать вечным.

Циклы For...Next и массивы

Циклы `For...Next`, возможно, наиболее удобны при работе с массивами. Рассмотрение циклов `For...Next` будет неполным без описания того, как они работают с объемами данных. Например, можно использовать цикл `For...Next` для занесения в массив вычисляемых значений:

```
Dim intArrayOfSquares (14) As Integer
For a =0 to 14
```

```
    intArrayOfSquares (a) = a * a
Next a
```

Код в этом примере начинается с объявления массива из 15 целых чисел (15, а не 14, так как VBA обычно начинает нумерацию массива с нуля). Затем используется цикл `For...Next` для присвоения значения каждому элементу массива, от 0 до 14. Переменная `a` служит не только счетчиком, но и *индексом* массива, указывая на элементы.

Вложенные циклы For...Next

Как и другие управляющие структуры VBA, циклы `For...Next` могут быть вложенными один в один или в другие управляющие структуры в любом количестве, например:

```
Dim sngR ' R означает случайное число
Randomize ' инициализирует генератор случайных чисел
For A = 1 To 5
    sngR = Rnd ()
    For B = 1 To 5
       Debug.Print sngR * Rnd ( )
    Next B
Next A
```

Рассмотрим код построчно. Он начинается с объявления переменной и инициализации генератора случайных чисел VBA. Затем начинается внешний цикл `For...Next`. Здесь VBA вызывает функцию `Rnd` для присвоения переменной `sngR` случайного значения. Затем выполняется внутренний цикл `For...Next`, в котором вычисляется пять других чисел. Результаты выводятся в окно Immediate. Внутренний цикл завершается после завершения всех пяти вычислений, и управление передается внешнему циклу. Наткнувшись на оператор `Next A`, VBA переходит в начало внешнего цикла, который четыре раза выполняет внутренний цикл.

Пример выглядит достаточно банальным, но можно заставить этот же код делать более полезную работу. Предположим, что необходимо написать мультимедийную программу, которая случайным образом выбирает пять компакт-дисков и проигрывает пять случайных треков на каждом.

Вложенные циклы `For...Next` также являются ключом к работе с многомерными массивами. Каждый цикл отвечает за одну размерность массива.

Выход с помощью оператора Exit For

Оператор `Exit For` обеспечивает возможность быстро прекратить цикл `For...Next`, еще до того, как достигнут конец цикла. Обычно используется с условными операторами (`If...Then` или `Select Case`), вложенными в основной цикл `For...Next`.

Одним из вариантов использования `Exit For` является проверка массива на наличие поврежденных данных и остановка процесса, если неверная величина найдена. В качестве примера предположим, что некий злобный умник смог вставить массив с неправильной информацией в прайс-лист. К счастью, известно, что перед этой информацией злодей оставил свой фирменный знак. При обновлении цен необходимо

убедиться, что данные не были подделаны. Следующий код сможет справиться с обеими задачами одновременно:

```
For p = 1 To varArraySize
    If varPriceArray (p) = "Здесь был Вася!" Then
        MsgBox "В массиве найдена поврежденная информация!"
        Exit For
    End If
    varPriceArray (p) = varPriceArray (p) * sngCOLA
Next p
```

For Each...Next

Вариант `For...Next`. Оператор VBA `For Each...Next` выполняет ряд операций с каждым объектом, хранимым в *библиотеке*.

Циклы `For Each...Next` описаны в главе 54.

Прерывание потока при помощи оператора GoTo

Если программа ведет себя непредсказуемо, можно напрямую направлять ее работу, передавая управление другому оператору в процедуре. Оператор `GoTo`, совместно со специальной меткой в месте назначения, позволяет свободно перемещаться в пределах процедуры. *Метка* — это оператор, который обозначает какое-либо место в программе. Чтобы ввести метку, просто поместите в код ее имя (см. соглашение об именовании в VBA) и двоеточие.

Пример использования оператора GoTo

В этом примере оператор `GoTo` переходит из основной части функции к метке `SpecialValue`, когда программа получает неверное значение:

```
Function GoToExample (ItemNumber As Integer)
    Dim intR As Integer
    Select Case ItemNumber
        Case 2412
            GoTo SpecialValue
        Case Is < CutOffValue
            DoSomething
        Case >= CutOffValue
            DoHArdlyAnything
    End Select
    (какие-нибудь операторы)
    GoToExample = intR
    Exit Function
SpecialValue:
    DoSomethingSpecial
    GoToExample = -intR
End Function
```

Опасности использования оператора GoTo

Использование оператора `GoTo` не приветствуется в программировании, поскольку приводит к появлению "макаронного кода", блуждающего по всей программе во время выполнения. Код, содержащий хотя бы несколько операторов `GoTo`, становится нечитабельным. Везде, где возможно, используйте управляющие структуры для контроля над выполнением программы.

Иногда, однако, оператор `GoTo` является наиболее простым путем заставить программу выполнять требуемые действия. Бывает, что мозги уже не вникают в запутанные хитросплетения вложенных циклов и условий, необходимых для реализации сложного набора требований. В таком случае `GoTo` поможет выбраться из лабиринта. Главное — не использовать его слишком часто.