Александр Чиртик
Алексей Гладкий

Excel

Трюки и эффекты

Откройте для себя новые возможности Excel

Алексей Гладкий

Excel. Трюки и эффекты

«Автор»

Гладкий А. А.

Excel. Трюки и эффекты / А. А. Гладкий — «Автор»,

Данная книга предназначена для пользователей Microsoft Excel и содержит описание приемов и методов работы, которые из-за своей специфичности недостаточно представлены (либо вообще не представлены) в пользовательской, справочной и иной соответствующей документации. Изучение приведенных в книге примеров позволит читателю открыть для себя не известные ранее возможности Excel. Предлагаемый материал легко усваивается благодаря тому, что излагается доступным и понятным языком.

Содержание

Алексей Анатольевич Гладкий, Александр Анатольевич Чиртик Excel. Трюки и эффекты

Введение

В настоящее время табличный редактор Excel, который является разработкой корпорации Microsoft и входит в состав пакета Microsoft Office, – один из самых популярных программных продуктов. Во многом это обусловлено возможностью применения Excel в самых разных отраслях: данную программу используют математики, IT-разработчики, инженеры, экономисты, бухгалтеры, аналитики, менеджеры и т. д. Такое распространение Excel объясняется широкими функциональными возможностями программы и вместе с этим простотой в использовании (удобный и понятный пользовательский интерфейс, возможность быстрого ввода и обработки данных, наглядность представления информации и др.).

Порядок использования программы подробно описывается в справочной подсистеме (для вызова справки достаточно нажать клавишу F1 на клавиатуре или кнопку Справка: Microsoft Offic Excel на ленте в окне приложения). Однако в процессе эксплуатации программы можно также использовать приемы и методы, которые в стандартной документации не рассматриваются либо рассматриваются поверхностно – как правило, потому, что они становятся известны только в результате активной эксплуатации Excel (то есть открываются опытным путем). Иногда они являются сюрпризом даже для самих разработчиков и открывают новые, порой самые неожиданные возможности программы. Описанию подобных трюков и посвящена эта книга.

Большинство описываемых в книге трюков и эффектов выполняется средствами языка VBA (для перехода к редактору VBA используется комбинация клавиш Alt+F11). Книга также содержит описание приемов, выполняемых «подручными» средствами, без программирования.

Структура книги

В главе 1 рассказывается об основах программирования на встроенном в пакет Microsoft Office языке Visual Basic for Applications (VBA). Главный упор сделан на описание синтаксиса и особенностей использования основных конструкций VBA.

В главе 2 рассматриваются нестандартные, но вместе с этим полезные и практичные решения, которые можно реализовать в рабочей области программы; здесь же рассказывается о приемах работы с файлами Excel (то есть с рабочей книгой). Приводится описание нестандартного использования формул Excel, а также порядок формирования и применения большого количества пользовательских функций. В отдельный раздел вынесено описание трюков и эффектов, которые применяются к ячейкам и диапазонам рабочего листа.

Глава 3 – самая объемная. В ней описывается большое количество самых разнообразных трюков и эффектов, относящихся к различным сферам использования программы. В частности, здесь рассказывается о создании пользовательских меню (как обычных, так и контекстных) и панелей инструментов, о нестандартных приемах форматирования, о работе с примечаниями и др. Большинство приведенных в данной главе трюков имеют практический характер, но есть и такие, реализация которых служит лишь развлекательным либо эстетическим целям.

У пользователей, часто работающих с диаграммами, наверняка вызовет интерес глава 4. В ней рассказывается, как быстро создать диаграмму с помощью макроса, построить диаграмму на основании данных нескольких рабочих листов, быстро изменить тип диаграммы, используя специально созданное контекстное меню, и др.

В главе 5 приведено несколько примеров создания небольших программ – как развлекательных, так и применяемых в работе, а также показаны приемы использования созданных программ.

В главе 6 содержится перечень полезных советов, которые пригодятся и начинающим, и опытным пользователям программы. Для удобства восприятия материал представлен в режиме «вопрос – ответ».

В приложении описаны наиболее часто используемые в приведенных в книге примерах стандартные объекты Excel: Application, Chart, Range, Workbook и Worksheet.

Общие положения

При написании книги использовалась версия Excel 2007. Тем не менее большинство приведенных в книге трюков работает и в других версиях программы.

Перед тем как вплотную приступать к изучению нестандартных приемов работы с Excel, кратко вспомним, для решения каких задач предназначено данное приложение и каковы его функциональные возможности, а также определимся с основными терминами и понятиями, которыми мы будем оперировать в дальнейшем.

Подразумевается, что данную книгу будут изучать пользователи, имеющие как минимум начальное представление об Excel. Тем не менее не будет лишним вспомнить, какие задачи решаются с помощью этой программы, а также ознакомиться с используемой в издании терминологией.

Назначение и функциональные возможности Microsoft Excel

Табличный редактор Microsoft Excel предназначен для решения следующих задач.

• Ввод и обработка табличных данных с использованием встроенных механизмов формул, функций, макросов и др.

• Анализ и управление данными (автоматический расчет итоговых и промежуточных данных, структуризация и консолидация данных, использование сводных таблиц, отчетов и др.).

• Импорт необходимых данных из различных источников (включая базы данных OLAP) и последующая их обработка. Поддержка XML-формата.

• Работа с графическими объектами и диаграммами.

• Взаимодействие и обмен данными с программой Lotus Notes, а также интеграция с другими программными продуктами («Галактика», «1С» и др.).

• Работа в Интернете (изменение данных на веб-странице, размещение данных Microsoft Excel в Сети, поддержка веб-файлов, гиперссылок и др.).

• Доступ к данным совместно с другими программами (Word, PowerPoint, Access и др.).

• Формирование самых разнообразных отчетов: аналитических, сводных, графических, в виде диаграмм и др.

• Выполнение стандартных функций Microsoft Office: печать документа, поиск данных и их замена, проверка наличия ошибок, защита информации и др.

• Создание приложений с применением языка программирования VBA.

С помощью Excel можно решать и другие задачи, возникновение которых обусловлено потребностями конкретного пользователя.

Используемая терминология

В данной книге используются следующие основные термины и понятия.

• Автофигура – готовая к использованию фигура заданной формы, которую можно добавлять на рабочий лист или в диаграмму. В Excel имеется встроенный набор автофигур.

• Диаграмма – визуальный способ представления числовых значений. Программа Excel поддерживает работу с различными видами диаграмм: круговыми, пузырьковыми, гистограммами, графиками и др.

• Имя – идентификатор, который предоставляет возможность ссылаться на какой-либо объект (ячейку, диапазон, формулу и т. д.).

• Комментарий – текст, который следует в программном коде сразу после символа «» вплоть до окончания данной строки и игнорируется при выполнении программы. Комментарий обычно включает в себя произвольную информацию вспомогательного характера, предназначенную для описания и пояснения определенных фрагментов кода либо всего кода.

• Контекстное меню – меню, содержащее список команд, которые предназначены для работы с конкретным объектом. Для вызова контекстного меню нужно щелкнуть на объекте правой кнопкой мыши или нажать комбинацию клавиш Shift+F10.

• Макрос – программа, которая написана на встроенном в Excel языке программирования Visual Basic for Applications (VBA). Переход в режим работы с макросами осуществляется с помощью команды Вид → Макросы.

• Массив – определенное количество ячеек либо значений, с которыми работают как с единым целым. Иначе говоря, массив – это группа элементов одного типа, которые имеют общее имя.

• Модуль – совокупность описаний, инструкций и процедур, сохраненная под общим именем в редакторе VBA.

• Надстройка – программа, внедренная в Excel для расширения функциональных возможностей. Чтобы подключить надстройки, следует в режиме настройки программы в разделе Настройки в поле Управление выбрать значение Настройки Excel и нажать кнопку Перейти, после чего в открывшемся окне установить требуемые флажки.

• Настройка – изменение ныне действующих параметров работы Microsoft Excel стандартными средствами, доступ к которым осуществляется из рабочего интерфейса Excel. Параметры работы программы можно разделить на два основных вида.

– Общие параметры – редактирование этих параметров приведет к соответствующим изменениям во всех рабочих книгах, в том числе и во вновь создаваемых.

– Локальные параметры – редактирование этих параметров вызовет соответствующие изменения только в текущей книге.

> **Примечание**
>
> Некоторые параметры работы Microsoft Excel можно изменить без использования стандартных средств.

• Область задач – элемент программы, предназначенный для быстрого выбора одной из нескольких связанных задач.

• Панель инструментов – панель, включающая в себя кнопки и иные элементы управления, которые используются для выполнения различных команд. Создание панелей инструментов осуществляется на вкладке Надстройки.

• Печать – вывод содержимого рабочей книги (полностью либо частично) на бумажный носитель с помощью принтера. На печать можно выводить следующие объекты: рабочую книгу, несколько рабочих книг, рабочий лист, несколько рабочих листов, диапазон ячеек на рабочем

листе, диапазон ячеек на нескольких рабочих листах, графические объекты, диаграммы. При этом существует возможность вывода на печать нескольких копий объекта за один сеанс.

• Пользовательский интерфейс – средство взаимодействия пользователя с программой. В состав пользовательского интерфейса входят лента с вкладками, группы, диалоговые окна и др. В Excel применяется стандартный пользовательский интерфейс Windows.

• Примечание – вспомогательная информация произвольного характера, относящаяся к определенной ячейке и хранящаяся независимо от содержимого этой ячейки. Чтобы добавить примечание к какой-либо ячейке, нужно выделить ее и выбрать в контекстном меню пункт Вставить примечание, после чего с клавиатуры ввести требуемый текст.

• Рабочая книга – файл, который создается, редактируется и сохраняется средствами Microsoft Excel. В большинстве случаев рабочая книга имеет расширение XLSX. Основной структурной единицей рабочей книги является рабочий лист (см. ниже).

• Рабочий лист – основной элемент рабочей книги, предназначенный для ввода, редактирования и хранения данных, а также для выполнения вычислений. По умолчанию в состав рабочей книги включены три рабочих листа. Основной структурной единицей рабочего листа является ячейка (см. ниже).

• Редактор VBA – интегрированная среда разработки, в которой осуществляется написание кодов (программирование) на языке VBA. Чтобы перейти в данный режим, необходимо нажать сочетание клавиш Alt+F11.

• Строка заголовка – стандартный элемент интерфейса многих приложений, расположенный в его верхней части. В данной строке отображается имя открытого документа.

• Строка формул – предназначена для ввода формул и редактирования содержимого ячеек.

• Форматирование – изменение отображения ячейки (ее «внешнего вида») либо представления данных, содержащихся в ячейке. Параметры форматирования ячейки не зависят от ее содержимого, и наоборот. Не стоит забывать, что после применения форматирования отображенное в ячейке значение может не совпадать с ее фактическим значением (наиболее характерный пример – округление: в ячейке хранится значение 0,24, но в соответствии с параметрами форматирования на экране может отображаться значение 0,2).

> **Совет**
>
> Точное фактическое значение, хранящееся в ячейке, при необходимости можно увидеть в строке формул, где оно отображается независимо от параметров форматирования.

• Формула – специальный инструмент Excel, предназначенный для расчетов, вычислений и анализа данных. Формула может включать в себя константу, оператор, ссылку, имя ячейки (диапазона) и функцию (см. ниже).

Операторы бывают трех видов.

– Арифметический оператор – предназначен для выполнения арифметических действий и выдающий в качестве результата числовое значение.

– Оператор сравнения – используется для сравнения данных и выдает в качестве результата логическое значение ИСТИНА или ЛОЖЬ.

– Текстовый оператор – применяется для объединения данных.

• Функция – готовая формула Microsoft Excel для расчетов, вычислений и анализа данных. Каждая функция может включать в себя константу, оператор, ссылку, имя ячейки (диапазона) и формулу. Пользовательская функция – это функция, написанная пользователем на языке VBA.

• Электронная таблица – интерактивная программа, состоящая из набора строк и столбцов, которые выводятся на экран в отдельном окне.

• Ячейка – наименьшая (элементарная) часть электронной таблицы, предназначенная для ввода и хранения информации. Каждая ячейка может содержать текст, число или формулу. Кроме того, при работе с ячейками используются следующие элементы.

– Адрес – это месторасположение (координаты) ячейки. Адрес состоит из буквы (номера) столбца и номера строки, на пересечении которых расположена данная ячейка.

– Ссылка – указание на адрес ячейки. Ссылки могут быть абсолютными (то есть не изменяющимися при перемещении и копировании ячейки), относительными (эти ссылки изменяются при перемещении и копировании ячейки) и смешанными. Внешняя ссылка – это ссылка на ячейку, расположенную в другой рабочей книге.

После того как мы вспомнили основные термины и понятия, используемые в Excel, можно приступать к изучению нестандартных приемов и методов работы с данной программой. И в первую очередь мы рассмотрим трюки, выполняемые в рабочей области.

Тексты программ на сайте издательства

Все приведенные в книге листинги (коды программ) можно загрузить с сайта издательства «Питер» по адресу http://www.piter.com/downLoad/978591180547/.

От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты dgurski@minsk.piter.com (издательство «Питер», компьютерная редакция). Мы будем рады узнать ваше мнение!

На сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

Глава 1
Краткое руководство по VBA

Цель данной главы – ознакомить читателя с основами программирования на языке Visual Basic for Applications (VBA), который встроен в пакет Microsoft Office.

Эта глава по форме изложения скорее похожа на справочник. Она рассчитана на пользователей, имеющих некоторый опыт создания программ на других объектно-ориентированных языках программирования, то есть представляющих, что такое переменная, константа, оператор, цикл, массив, класс, объект и т. д. Главный упор в данной главе сделан на описание синтаксиса и особенностей использования основных конструкций VBA, которые нужно знать для понимания приведенных в тексте листингов.

Знакомство с VBA

VBA – это язык программирования, поддерживаемый большинством приложений пакета Microsoft Office. Для запуска среды программирования VBA можно использовать сочетание клавиш Alt+F11.

Возможности VBA

Благодаря высокой степени интеграции в приложения язык VBA позволяет программисту легко применять существующие возможности Microsoft Office для решения специфических задач. Не менее легко VBA позволяет добавлять в приложения этого пакета новые возможности, увеличивая функциональность приложений и удобство работы с ними.

Применительно к Excel программирование на VBA позволяет реализовывать следующие возможности (естественно, это далеко не полный список):

- добавление функций для специфических расчетов;
- ускорение ввода данных в таблицу;
- автоматизацию типичных, часто выполняемых пользователем действий;
- создание команд меню, панелей инструментов как на основе уже имеющихся, так и выполняющих совершенно новые действия;
- повышение наглядности данных (например, с помощью различной окраски ячеек);
- автоматизацию обработки больших объемов данных;
- автоматизацию создания разнообразных отчетов, бланков и прочих действий, связанных с выбором заданной информации из большого объема исходных данных.

Реализация всех этих возможностей при использовании VBA очень часто достигается написанием небольшого количества достаточно простого программного кода.

Структура проекта VBA

Внешний вид редактора VBA с открытым проектом представлен на рис. 1.1.

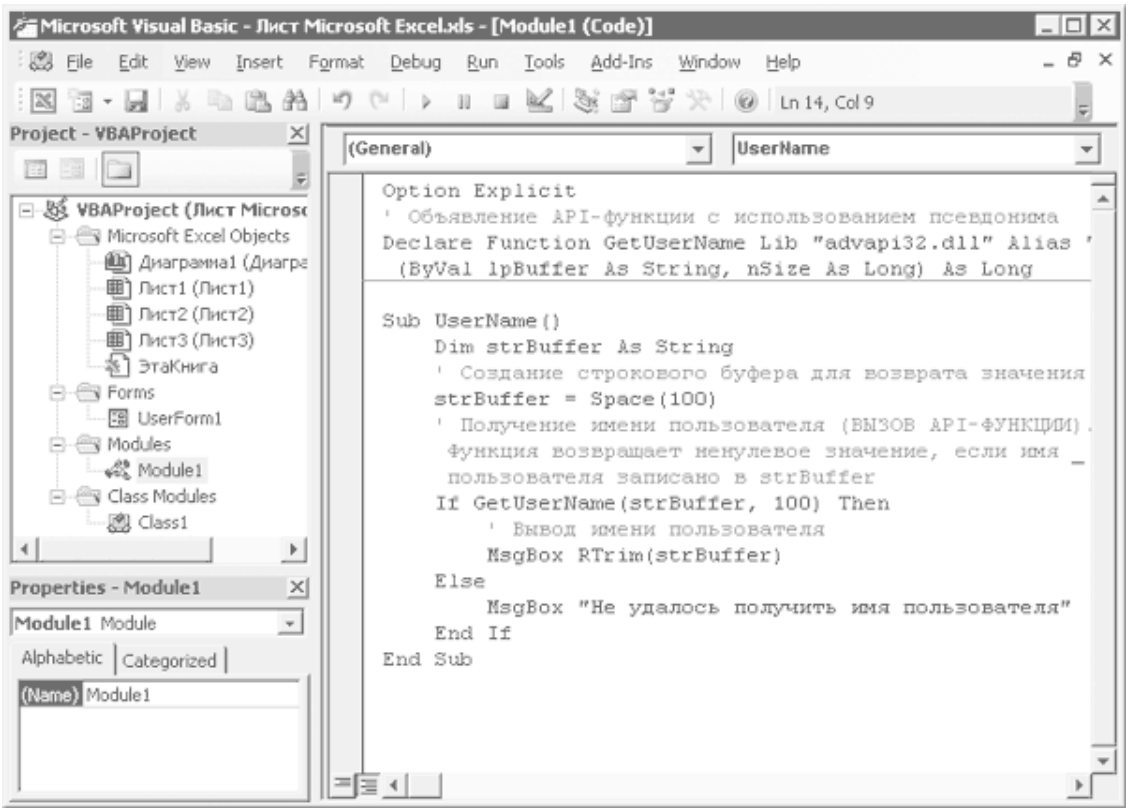

Рис. 1.1. Проект VBA

В редакторе открыты три окна: слева вверху – окно структуры проекта (Project), слева внизу – окно свойств (Properties) и справа – окно с текстом программы модуля. С помощью окна Project (Проект) можно просматривать структуру проекта, добавлять и удалять элементы проекта, открывать для редактирования содержимое модуля (двойным щелчком на значке соответствующего модуля). Окно Properties (Свойства) используется для задания свойств выделенных элементов проекта.

VBA-проект в Microsoft Excel может содержать следующие элементы:

- модули (стандартные модули VBA);
- модули класса;
- модули рабочей книги;
- модули рабочих листов;
- модули диаграмм;
- формы.

Стандартный модуль VBA

Стандартный модуль VBA – это элемент проекта, который содержит программный код, непосредственно используемый остальными элементами проекта (глобальные функции, переменные, константы и т. д.). В окне структуры проекта стандартные модули группируются в папку Modules.

Обычно в стандартном модуле записываются программы, которые не привязаны к конкретным объектам, таким как рабочий лист и рабочая книга. Именно в этом модуле записывается большинство примеров (трюков), рассматриваемых в последующих главах книги.

Модуль класса

Модуль класса – это модуль, в котором записывается программный код, реализующий работу пользовательских (созданных программистом) классов. В окне структуры проекта такие модули группируются в папку Class Modules.

Модуль рабочей книги

Модуль рабочей книги (ЭтаКнига в папке Microsoft Excel Objects) – это модуль класса, в котором реализуются дополнительные возможности по манипулированию рабочей книгой. Программы, записанные в этом модуле, могут напрямую обращаться к свойствам и методам объекта рабочей книги (см. описание модулей класса в конце главы). Отличием модуля рабочей книги от обычного модуля класса является то, что из программы на VBA нельзя создать экземпляр объекта рабочей книги – он создается автоматически.

Модуль рабочего листа

Модуль рабочего листа (папка Microsoft Excel Objects) – это модуль класса, в котором реализуются дополнительные возможности по манипулированию определенными рабочими листами книги. Программы, записанные в этом модуле, могут напрямую обращаться к свойствам и методам объекта рабочего листа. Из программы на VBA также нельзя создать экземпляр объекта конкретного рабочего листа – можно создать только новый рабочий лист с пустым модулем.

Модуль диаграммы

Модуль диаграммы (папка Microsoft Excel Objects) – это модуль класса, в котором реализуются дополнительные возможности по манипулированию диаграммами, вынесенными на отдельные листы рабочей книги. Особенности модуля диаграммы аналогичны особенностям модуля рабочего листа.

Форма

Форма (папка Forms) – это элемент проекта VBA, с помощью которого можно создавать диалоговые окна для взаимодействия с пользователем. Форма состоит из двух частей: модуля формы и собственно формы (диалогового окна).

Модуль формы – это модуль класса, в котором реализуется поведение формы. Использование этого модуля аналогично использованию обычного модуля класса. Для открытия модуля формы служит пункт View Code (Просмотр кода) контекстного меню, которое вызывается щелчком правой кнопки мыши на значке формы в окне Project (Проект).

Для задания внешнего вида диалогового окна используется редактор форм, который открывается с помощью пункта View Object (Просмотр объекта) контекстного меню. Форма, открытая для редактирования внешнего вида, показана на рис. 1.2.

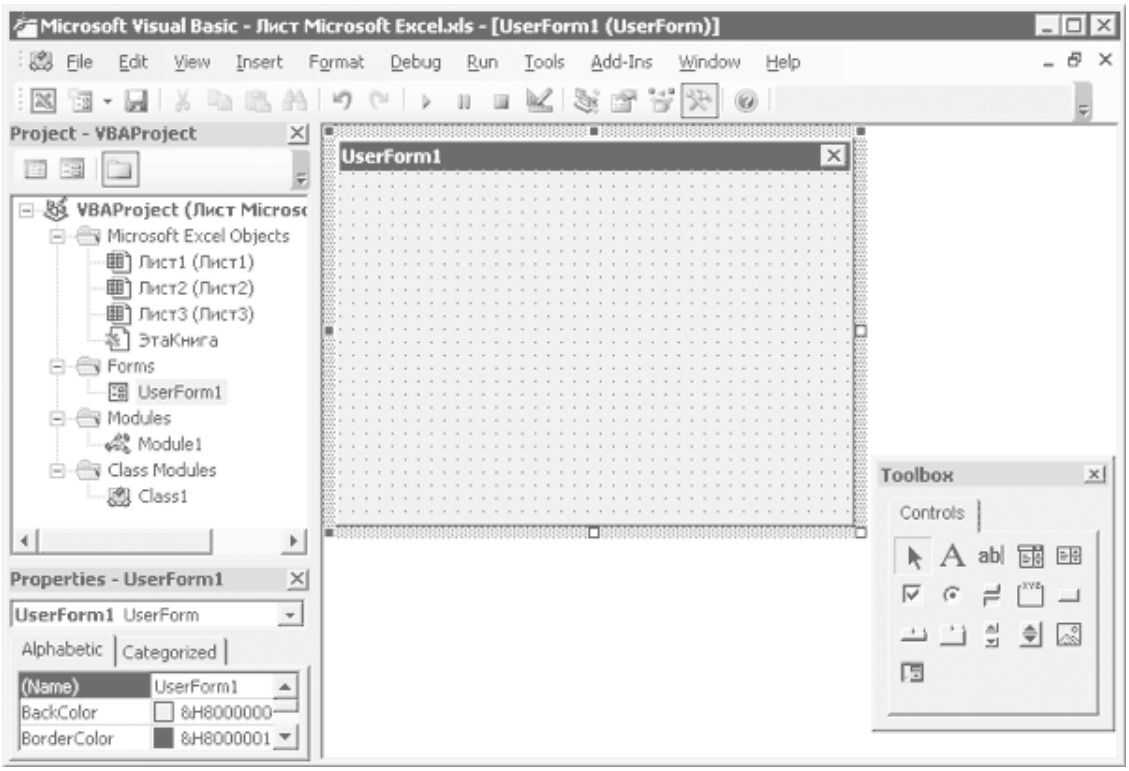

Рис. 1.2. Редактирование формы

Для изменения свойств открытой формы используется окно Properties (Свойства). В окне ТооШох (Панель инструментов) можно выбирать элементы управления, добавляемые в форму. После добавления любого элемента управленияс помощью двойнго щелчка кнопкой мыши на нем можно перейти к редактированию соответствующего кода в модуле формы.

Структура модуля VBA

При разработке любой программы на VBA программный код записывается в одном или нескольких модулях. Код, записанный в любом модуле VBA, имеет следующую структуру.

1. Объявления переменных, директивы (с использованием ключевого слова Option), объявления API-функций.

2. Объявления и реализация процедур и функций.

Соглашения, применяемые при описании синтаксиса VBA

В данном разделе приводится описание элементов, которые используются для формального задания синтаксиса конструкций языка VBA. Сведения об этих элементах приведены в табл. 1.1.

Таблица 1.1. Элементы описания синтаксических конструкций VBA

Элемент	Описание
Значение	Текст, написанный курсивом, обозначает произвольное значение (константное, такое как строка "D:\asd.txt", или значение переменной)
Инструкции	Это слово используется для обозначения произвольной последовательности инструкций программы. Может употребляться с номером, например «Инструкции1». Может также употребляться в составе словосочетания в его начале, например «Инструкции подпрограммы»
Open	Текст, записанный без использования курсива (кроме слова «Инструкции»), является ключевым словом или идентификатором VBA (названием инструкции, оператора, функций и т. д.)
[Необязательный_ Элемент]	Используется для указания необязательных элементов конструкции
Элемент1 | Элемент2	Применяется для указания возможности выбора употребляемого элемента конструкции (то есть использовать или *элемент1*, или *элемент2*)
...	Используется для указания возможности повторения последнего элемента конструкции бесконечное количество раз

Примечание

Кроме описанных в таблице элементов, при задании формата синтаксических конструкций и в текстах программ используется символ подчеркивания «». Он является стандартным символом VBA. Текст, который заканчивается символом подчеркивания, представляет собой одно целое с текстом на следующей строке.

Чтобы сказанное выше стало более понятным, ниже приведен пример формального задания синтаксической конструкции языка VBA. В качестве примера взят формат упрощенного объявления локальной переменной (смысл всех элементов данной конструкции будет полностью раскрыт в последующих разделах главы):

Dim | Static Имя [As Имя_типа][, Имя_переменной [As Имя_типа]]…

Приведенная запись означает, что строка объявления локальной переменной должна начинаться инструкцией Dim или Static. После инструкции должен следовать идеитификатор переменной. Необязательным элементом конструкции является указание типа переменной. Но если тип указывается, то значение в квадратных скобках (первых) должно быть использовано полностью, то есть ключевое слово As не должно применяться без указания имени типа. Объявления переменных можно продолжить в этой же строке без повторного использования инструкции Dim, но разделяя их запятой (см. вторые скобки). Подобные объявления можно

продолжать в строке до бесконечности (об этом говорит использование многоточия после вторых скобок).

Примеры объявлений переменных, удовлетворяющие указанному формату, приведены ниже:

```
Dim intPos As Integer
Dim varValue, intValue As Integer
Static strText As String
Static var1 As Variant, var2 As Variant, var3 As Variant
```

Комментарии в программе

В VBA предусмотрены два способа введения комментариев в программы. Первый – это использование ключевого слова Rem для обозначения начала комментария. Второй – использование вместо Rem апострофа (). Главным различием этих двух способов является то, что ключевое слово Rem должно находиться в начале строки программы. При этом вся строка является комментарием. Например:

```
Rem Объявление переменной
Dim intRes As Integer
Rem Присвоение значения переменной
intRes = 123
```

Комментарий же, вводимый с помощью апострофа, может быть расположен как в отдельной строке, так и на одной строке с другими инструкциями (в конце этой строки):

```
' Объявление переменной
Dim intRes As Integer
intRes = 123 Присвоение значения переменной
```

Все комментарии в VBA являются однострочными, но при необходимости их текст может быть перенесен на следующую строку с использованием символа подчеркивания:

```
' Длинный комментарий, текст которого не помещается _
в одной строке
```

или

```
Rem Длинный комментарий, текст которого не помещается _
в одной строке
```

Идентификаторы

Идентификаторами в VBA являются названия переменных, констант, функций, процедур, классов, типов данных и прочих элементов, не являющихся зарезервированными словами языка (названиями инструкций, операторов, встроенных функций и т. д.).

Среда разработки VBA поддерживает кодировку символов Unicode. Поддержка данной кодировки разработки означает, что программист может использовать в составе идентификаторов символы любого поддерживаемого алфавита (например, кириллицы).

При формировании идентификаторов необходимо учитывать следующее.

• Идентификатор должен состоять только из букв (любого алфавита), цифр и символа подчеркивания.

• Первым символом идентификатора должна быть буква.

> **Внимание!**
>
> VBA не различает регистр символов в идентификаторах. Это значит, что идентификаторы strmyText и strMyText будут представлять одну и ту же переменную. Это же справедливо и для идентификаторов процедур, функций, классов и т. д.

Рассмотрим примеры корректных идентификаторов VBA:

strText
CUSTOM_Data2
Функция_Суммы
РасчетПрибыли

Переменные

В данном разделе читатель ознакомится с основными особенностями использования переменных при написании программ на языке VBA.

Встроенные типы данных

VBA располагает множеством встроенных типов данных. Условно эти типы можно разделить на численные типы, строки, ссылки, типы для хранения даты и времени, объектные ссылки, массивы и особый тип для хранения значения любого типа, именуемый Variant.

Численные типы данных

Основные характеристики численных типов VBA приведены в табл. 1.2.

Таблица 1.2. Численные типы данных VBA

Тип	Объем памяти (байт)	Диапазон значений
Byte (байт)	1	От 0 до 255
Integer (целое)	2	От –32 768 до 32 767
Long (длинное целое)	4	От –2 147 483 648 до 2 147 483 647
Single (с плавающей точкой)	4	От –3.402 823E38 до –1.401 298E-45, от 1.401 298E-45 до 3.402 823E38
Double (с плавающей точкой двойной точности)	8	От –1.797 693 134 862 32E308 до — 4.940 656 458 412 47E-324, от 4.940 656 458 412 47E-324 до 1.797 693 134 862 32E308
Currency (масштабируемое целое)	8	От –922 337 203 685 477.5808 до 922 337 203 685 477.5807
Decimal (фиксированной длины, повышенной точности)	14	Без дробной части: +/ 79 228 162 514 264 337 593 543 950 335; с максимальной точностью: +/–7.922 816 251 426 433 759 354 395 033 5 (до 28 знаков после запятой)
Variant (для числовых значений)	16	Любое значение из указанных для остальных типов диапазонов

Примечание

Численный тип Decimal как самостоятельный тип на сегодняшний день не поддерживается. Однако его можно использовать в пределах типа Variant (о типе Variant будет рассказано далее).

Строки

Для хранения символьных данных в VBA реализована поддержка типа данных String (строка). В переменных этого типа могут храниться отдельные символы и большие фрагменты текста. Строки в VB A бывают двух видов: фиксированной и переменной длины. Разница между этими двумя типами строк понятна из их названий.

Строки фиксированной длины применяются, когда длина текста, который хранится в них, постоянна или не может превышать известный предел (например, для хранения отдельных символов). Строки фиксированной длины в ряде случаев обрабатываются быстрее строк переменной длины. Максимальный размер строки фиксированной длины – около 65 400 символов.

Строки переменной длины являются более гибким инструментом обработки текста в программах на VBA. Длина этих строк может динамически изменяться в зависимости от длины хранимого в них текста. Максимальная длина таких строк – около 2 млрд символов.

Дата и время

Для удобства работы со значениями даты и времени в VBA введен тип данных Date. Он позволяет задавать значения времени и даты в формате, удобном для восприятия, а также упрощает вычисления с временными интервалами. Этот тип данных, естественно, используется и в таблицах Excel.

Нужно заметить, что тип Date не является внутренним типом, используемым VBA для хранения даты и времени. Вместо него применяется тип Single (число с плавающей точкой). Целая часть этого числа – количество суток, прошедших с 30 декабря 1899 года, дробная – прошедшая часть текущих суток.

Тип данных Variant

В VBA предусмотрен один универсальный тип данных – Variant. Переменная этого типа может хранить значение любого поддерживаемого VBA типа (в том числе и ссылки на объекты, о которых будет рассказано ниже).

Однако при обработке переменных типа Variant тратится дополнительное время на определение и преобразование типа данных в этих переменных – самый универсальный тип данных VBA оказывается и самым медленным. Поэтому следует избегать слишком частого и неоправданного использования переменных этого типа (например, в качестве целочисленных итераторов, счетчиков и т. д.).

Когда переменная типа Variant пуста (ей не присвоено никакого значения), она заполняется специальным значением Empty.

Ссылки. Тип данных Object

Важно понимать, что в VBA переменные, предоставляющие доступ к объектам, являются только ссылками на эти объекты. В данном языке программирования невозможно получить сам объект (его двоичный код). Все операции по созданию, удалению объектов и манипулированию ими, осуществляются только с использованием ссылок.

Объекты, доступные из VBA, существуют, пока на них установлена хотя бы одна ссылка. Первая ссылка на объект устанавливается при его создании. В процессе работы можно как устанавливать новые ссылки на объект, так и удалять их с использованием специальной инструкции

Set. Пусть objRef – ссылка на некоторый объект. Тогда операция установления новой ссылки будет иметь такой вид:

```
Set objNewRef = objRef
```

Теперь objNewRef ссылается на тот же объект, что и objRef. Операция же удаления ссылок будет выглядеть следующим образом:

```
Set objRef = Nothing
Set objNewRef = Nothing
```

Если на объект не было других ссылок, кроме этих, то он будет удален.

Для доступа к объектам в VBA предусмотрен тип данных Object. Он является универсальным, так как может быть ссылкой на объект любого типа.

Объявление переменных

Для объявления переменных элементарных типов (не массивов) в блоке объявлений модуля используется следующая инструкция:

```
Public | Private [WithEvents] Имя_переменной [As [New] Имя_-
типа] _
[, Имя_переменной [As [New] Имя_типа]]…
```

Ключевые слова, записанные до первых квадратных скобок, задают область видимости переменных:

• Public – позволяет объявлять глобальные переменные и общие переменные-члены класса (о классах будет рассказано позже);

• Private – позволяет объявлять переменные, доступные только в одном модуле, и частные переменные-члены класса.

Для объявления переменных элементарных типов (не массивов) в процедурах или функциях используется такая инструкция (локальных переменных):

```
Dim | Static [WithEvents] Имя_переменной [As [New] Имя_типа] _
[, Имя_переменной [As [New] Имя_типа]]…
```

Ключевые слова, записанные до первых квадратных скобок, задают время жизни переменных:

• Dim – используется для объявления локальных переменных, которые уничтожаются после выхода из процедуры;

• Static – используется для объявления локальных переменных, значения которых сохраняются между вызовами процедуры или функции.

Ключевое слово WithEvents используется для объявления переменной-обработчика событий объекта.

Имя_переменной – это идентификатор объявляемой переменной.

Имя_типа – название типа данных объявляемой переменной.

Если конструкция [As [New] Имя_типа] не используется, то типом объявляемой переменной автоматически становится тип Variant.

Если используется ключевое слово New, то создается новый объект. New нельзя использовать совместно с WithEvents, а также при объявлении переменной типа Object и если тип Имя_типа не является объектным.

Ниже приведены примеры объявления переменных на VBA:

```
Public intData As Integer
Private intCount As Integer, varData
```

```
Dim strText As String
Static a, b, c
Dim objRef As Object
Dim objCtrl As New Control
```

Внимание!

При объявлении в одной строке нескольких переменных слово As относится только к переменной, непосредственно после идентификатора которой оно следует. Например, при объявлении Dim a, Ь, c As Integer переменные a и b будут иметь тип Variant, а переменная c – тип Integer.

Инициализация переменных

После того как переменная объявлена, VBA производит ее инициализацию указанным ниже образом.

• Переменным численных типов автоматически присваивается нулевое значение.

• Строки переменной длины после объявления являются пустыми (с нулевой длиной). Строки фиксированной длины заполняются нулевыми символами.

• Данные типа Date инициализируются значением 00:00:00 30.12.1899 (это равняется нулю при представлении даты в численном формате, о котором было рассказано выше).

• Все переменные типа Object и подобные (то есть ссылки на объекты определенного типа) принимают значение Nothing, если при их объявлении не создан новый объект (не использовалось New).

Явное и неявное объявление переменных

Рассмотренное выше объявление переменных называется явным.

VBA также поддерживает неявное объявление переменных. Под неявным объявлением подразумевается возможность использования переменной без ее объявления посредством инструкции Dim, Static, Private или Public. При первом обращении к такой переменной для нее автоматически выделяется память и происходит ее инициализация. Следует отметить, что все неявно объявленные переменные имеют тип Variant.

Использование неявного объявления переменных может как упростить написание программ, так и значительно усложнить процесс их отладки.

К примеру, можно очень долго разбираться, почему же после вычисления такого выражения, как dblSalaryAccount = dblSalaryAcount * 10. 5, значение переменной dblSararyAccount стало равным нулю, если до этого оно было равно 5.37 5. Все дело в небольшой ошибке в названии идентификатора переменной, в результате которой создается новая переменная dblSalaryAcount, которая инициализируется нулевым значением. В случае использования явного объявления переменных подобных ошибок в программе быть не может, потому что каждый раз при обнаружении необъявленного идентификатора VBA выдает ошибку.

Для включения требования обязательного объявления переменных используется директива Option Explicit. Чтобы данная директива добавлялась в каждый новый модуль автоматически, нужно с помощью меню Tools → Options (Сервис → Параметры) редактора VBA открыть диалоговое окно Options (Параметры) и на вкладке Editor (Редактор) установить флажок Require Variable Declaration (Явное описание переменных).

Константы

Как и в любом другом языке программирования, в VBA можно сопоставлять с идентификаторами константные значения. Объявление констант в языке VBA во многом сходно с объявлением переменных. Синтаксис инструкции объявления константы следующий:

[Public | Private] Const Имя_константы [As Имя_типа] = Значение

Два ключевых слова в первых скобках задают область видимости константы:

• Public – используется для объявления глобальных констант;

• Private – используется для объявления констант, доступных только в том модуле, где они объявлены.

По умолчанию, то есть когда не употреблены указанные ключевые слова, константа является Private.

Имя_константы – задает идентификатор константы.

Значение – константное значение (например, «Строка1», 1.245 ит. д.) либо выражение, в число аргументов которого не входят переменные и функции.

Если тип константы не задан, то он автоматически выбирается VBA исходя из ее значения. Ниже приводятся примеры объявления констант:

```
Const PI As Double = 3.14159265359
Public Const MyConstString = «MyConst»
Private Const НазваниеТаблицы As String = «Отчеты»
```

Операторы

Язык VBA содержит большое количество встроенных операторов, которые позволяют выполнять разнообразные действия над всеми встроенными в VBA типами. Операторы и их операнды по определенным правилам составляются в выражения. Данный раздел посвящен описанию операторов, предоставляемых VBA-программисту.

Операторы для работы с численными значениями

Информация обо всех операторах для работы с численными значениями приведена в табл. 1.3.

Таблица 1.3. Операторы для работы с численными значениями

Формат оператора	Описание
Результат = *Выражение*	Оператор присвоения правой части выражения левой
Результат = *Выражение1* + *Выражение2*	Оператор суммы
Результат = *Выражение1* – *Выражение2*	Оператор разности
Результат = *Выражение1* * *Выражение2*	Оператор произведения
Результат = *Выражение1* / *Выражение2*	Оператор деления
Результат = *Выражение1* \ *Выражение2*	Оператор целочисленного деления (возвращает целую часть от результата деления)
Результат = *Выражение1* Mod *Выражение2*	Оператор, возвращающий остаток от деления
Результат = *Выражение1* ^ *Выражение2*	Оператор возведения в степень. Основание степени — *Выражение1*, показатель степени — *Выражение2*

Примечание

Оператор «+» может использоваться и для соединения строк. Однако bVBA существует специальный оператор «&», выполняющий эту функцию. Рекомендуется использовать для соединения строк именно оператор «&», так как это способствует легкому визуальному отделению операций над строками от операций над другими типами данных, что, в свою очередь, улучшает читаемость кода.

Операторы сравнения

Результатом выполнения всех операторов сравнения является значение типа Boolean. Если операнды какого-либо оператора удовлетворяют его условию, то возвращается значение True, иначе возвращается значение False. Все операторы сравнения, поддерживаемые VBA, приведены в табл. 1.4.

Таблица 1.4. Операторы сравнения

Формат оператора	Описание
Результат = Выражение1 = Выражение2	Оператор равенства. Возвращает True, если значения выражений равны
Результат = Выражение1 > Выражение2	Оператор «больше». Возвращает True, если значение *Выражение1* больше значения *Выражение2*

Формат оператора	Описание
Результат = Выражение1 < Выражение2	Оператор «меньше». Возвращает True, если значение *Выражение1* меньше значения *Выражение2*
Результат = Выражение1 >= Выражение2	Оператор «больше либо равно». Возвращает True, если значение *Выражение1* больше или равно значению *Выражение2*
Результат = Выражение1 <= Выражение2	Оператор «меньше либо равно». Возвращает True, если значение *Выражение1* меньше или равно значению *Выражение2*
Результат = Выражение1 <> Выражение2	Оператор неравенства. Возвращает True, если значения выражений не равны

Описанные операторы сравнения могут принимать в качестве операндов значения выражений любого типа, то есть фактически оперируют с типом данных Variant. Если один из операндов равен Empty, то результатом выполнения операторов будет специальное значение NULL. Если операнды несравнимы, то при выполнении описанных выше операторов генерируется ошибка: «Несоответствие типа».

Режим сравнения строковых значений можно задать с помощью директивы Option Compare. Для Excel работают два варианта данной директивы: Option Compare Text (текст сравнивается без учета регистра символов) и Opt ion Compare Binary (сравниваются бинарные коды символов, при этом автоматически учитывается их регистр). По умолчанию сравнение строк происходит согласно директиве Option Compare Binary.

В VBA реализованы два специфических оператора Like и Is, которые тоже относятся к операторам сравнения. Они также возвращают значение типа Boolean как результат сравнения.

Оператор I s используется для определения, являются ли две ссылки ссылками на один и тот же объект. Этот оператор допускает использование в качестве операндов только ссылки на объекты. Формат данного оператора такой:

```
Результат = Ссылка1 Is Ссылка2
```

Оператор Like используется для проверки, удовлетворяет ли текст в строке заданному шаблону. Формат этого оператора следующий:

```
Результат = Строка Like Шаблон
```

Перечень символов, которые могут употребляться в строке шаблона, приведен в табл. 1.5.

Таблица 1.5. Перечень возможных символов в строке шаблона

Символы в шаблоне	Соответствующие символы в строке
*	Любое количество любых символов

Символы в шаблоне	Соответствующие символы в строке
?	Любой одиночный символ
#	Любая одиночная цифра
[*Список_символов*]	Любой одиночный символ из заданного списка
[!*Список_символов*]	Любой одиночный символ, не входящий в заданный список

В качестве примера определим, является ли строка «15 26 ОА» номером автомобиля серии ОА или ОО. Значение строки-шаблона для этого случая будет равно «## ## О[АО]», а результатом применения оператора «15 26 ОА» Like «## ## О[АО]» будет значение True.

Для задания непрерывного диапазона символов в квадратных скобках можно воспользоваться знаком «минус» (-). При этом символы необходимо указывать в возрастающем порядке (по номеру в алфавите): [A-Z], а не [Z-A].

Примечание

Если нужно, чтобы в шаблоне присутствовали специальные символы, приведенные в табл. 1.5, то необходимо заключить соответствующие знаки в скобки: ([), (]), (#), (?), (*).

Логические операторы

В VBA введены операторы, которые используются в составе логических выражений (например, условие в инструкции If-Then-Else, которая будет рассмотрена позже). Формат логических операторов VBA (кроме импликации) и их описание приведены в табл. 1.6 (все выражения, используемые в операторах, – логические, принимающие значение True или False).

Таблица 1.6. Логические операторы VBA

Формат оператора	Описание
Результат = *Выражение1* Eqv *Выражение2*	Эквивалентность. Возвращает True, если значения выражений равны
Результат = *Выражение1* And *Выражение2*	Логическое И. Возвращает True, если оба выражения имеют значение True
Результат = *Выражение1* Or *Выражение2*	Логическое ИЛИ. Возвращает True, если как минимум одно из выражений имеет значение True
Результат = *Выражение1* Xor *Выражение2*	Исключающее ИЛИ. Возвращает True, если выражения имеют противоположные значения
Результат = Not *Выражение*	Логическое НЕ. Возвращает значение, противоположное значению выражения *Выражение*

Массивы

При создании программ часто приходится оперировать большими количествами данных одного типа и имеющих одинаковый смысл. Для хранения таких данных используются массивы. Массив – это совокупность значений одного типа, объединенных в одной переменной.

Язык VBA предоставляет широкие возможности для использования массивов. В нем работа с массивами значительно упрощена. Например, при выполнении программы автоматически контролируется выход за пределы массива. Также VBA-программисту при работе с массивами не нужно заботиться о выделении и освобождении памяти.

Объявление массива

Для объявления массивов в VBA используются инструкции, формат которых приведен ниже:

```
Public | Private Имя_массива ([Размерность])[As Имя_типа] _
[, Имя_массива ([Размерность]) [As Имя_типа]]…
```

или

```
Dim | Static Имя_массива ([Размерность])[As Имя_типа] _
[, Имя_массива ([Размерность]) [As Имя_типа]]…
```

Первая инструкция используется для объявления массивов на уровне модуля, вторая – для объявления массива в процедуре или функции (все аналогично объявлению переменных).

Объявление массива отличается от объявления любой другой переменной тем, что при объявлении массива после идентификатора переменной указывается размерность. Если размерность не указана, то создается динамический массив, размер которого можно изменять во время выполнения программы. Для динамического массива инструкция объявления является формальной: чтобы этот массив можно было использовать, к нему необходимо применить инструкцию ReDim (об этой инструкции будет рассказано далее).

При указании размерности массива необходимо учитывать, что элемент Размерность имеет следующий формат:

```
Нижняя_граница To Верхняя_граница | Количество_элементов _
[,Нижняя_граница To Верхняя_граница | Количество_элементов]…
```

В VBA разрешено создавать многомерные массивы с количеством измерений не более 60. Размерности измерений массива разделяются запятой.

При задании размерности в виде Нижняягранща To Верхняягранща нужно явно указывать нижнюю и верхнюю границы измерения массива (например, 50 To 100).

При задании размерности можно также просто указывать требуемое количество элементов в данном измерении массива. При таком задании в качестве нижней границы измерения используется значение по умолчанию (об изменении этого значения будет рассказано далее).

Ниже приведены примеры объявлений массивов (переменного размера, двух одномерных и двух многомерных):

```
Dim avarValues()
Dim astrValues(1 To 10) As String, astrValues2(10) As String
Dim aintValues(1 To 10, 1 To 3) As Integer, aintValues(10, 3)
As Integer
```

Задание нижней границы по умолчанию

Как было сказано ранее, при указании размерности измерения массива может использоваться значение нижней границы по умолчанию. Для задания нижней границы, используемой по умолчанию, предназначена директива Option Base. Существуют только два варианта данной директивы:

```
Option Base 0
```

и

```
Option Base 1
```

Первый вариант устанавливает нижнюю границу равной нулю (используется по умолчанию), а второй – единице.

Изменение размера массива

Язык VBA позволяет изменять размер динамического массива во время выполнения программы. Кроме того, VBA дает возможность изменять количество измерений такого массива. Для этого используется инструкция ReDim, формат которой следующий:

```
ReDim [Preserve] Имя_массива ([Размерность])[As Имя_типа] _
[, Имя_массива ([Размерность]) [As Имя_типа]]…
```

Назначение элементов данной инструкции полностью аналогично назначению одноименных элементов инструкции Dim (при использовании ее для объявления массивов). Тип элементов массива можно указывать только в том случае, если Имямассива – это идентификатор переменной типа Variant.

При выполнении инструкции ReDim без использования ключевого слова Preserve значения всех элементов, которые ранее были в массиве, теряются. Ниже приведены примеры таких инструкций:

```
ReDim astrValues(1 To 10), aintValues(10, 20)
ReDim varArray(2 To 4) As Boolean
```

Использование Preserve позволяет изменять размер массива, не теряя значений его элементов. Однако использование данного ключевого слова налагает некоторые ограничения на возможности манипулирования массивами:

• нельзя изменять количество измерений массива;

• нельзя изменять размерности измерений массива, кроме размерности последнего измерения;

• можно изменять только верхнюю границу последнего измерения массива.

Давайте рассмотрим пример использования инструкции ReDim с ключевым словом Preserve:

```
' Первая инструкция ReDim для динамического массива
ReDim astrValues(1 To 5, 1 To 10)
…
' Увеличение размера массива
ReDim Preserve astrValues(1 To 5, 1 To 25)
…
' Уменьшение размера массива
ReDim Preserve astrValues(1 To 5, 1 To 15)
```

Определение границ массива

Так как VBA позволяет задавать произвольную нижнюю границу массива, при написании программ крайне удобно наличие возможности узнать границы массива во время выполнения программы. Для этой цели в VBA введены две функции, формат которых следующий:

```
LBound(Имя_массива[, Номер_измерения])
RBound(Имя_массива[, Номер_измерения])
```

Функция LBound позволяет получить нижнюю границу массива, а RBound – верхнюю. Обе функции принимают в качестве аргументов идентификатор массива и номер измерения, границу которого нужно получить. Нумерация измерений начинается с единицы. Если параметр Номеризмерения опущен, то его значение принимается равным единице. Обе функции возвращают значение типа Long.

Ниже приведен пример получения нижней и верхней границ первого измерения массива avarValues (значения сохраняются в переменных типа Long):

```
lngLBound = LBound(avarValues)
lngRBound = RBound(avarValues)
```

Доступ к элементам массива

Для доступа к элементам массива в VBA используется указание номера этого элемента в круглых скобках после идентификатора переменной массива. При этом номера измерений массива разделяются запятыми. Например (для одномерного и трехмерного массивов):

```
intNum = aintValues(16)
intNum = aintValues(12, 32, 3)
```

Использование переменной Variant при работе с массивами

Язык VBA поддерживает универсальный тип данных Variant, которому находится применение и при работе с массивами. Переменной этого типа можно присваивать массив. В результате этой операции в переменной Variant формируется копия массива. Далее с такой переменной можно работать либо как с обычной переменной, либо как с массивом (использовать доступ к элементам), например:

```
Dim aintValues(1 To 3) As Integer
Dim varArray
' Присвоение массива переменной типа Variant
varArray = aintValues
' Доступ к элементам массива
varArray(1) = 1
varArray(2) = 2
varArray(3) = 3
```

Возможность присвоения массива переменной типа Variant на самом деле широко используется в VBA при передаче массивов в функции и процедуры, а также при возврате функциями массивов.

Для определения того, содержит ли переменная типа Variant массив, можно использовать функцию IsArray, имеющую следующий формат:

```
IsArray(Переменная)
```

Данная функция возвращает значение типа Boolean: True – если в переменной с именем Переменная содержится массив, и False – в противном случае.

Использование функции Array для заполнения массива

В VBA имеется возможность быстрого заполнения массива значениями. Эта возможность реализована в функции Array. Ее формат такой:

```
Array(Список_элементов)
```

В качестве аргументов функция принимает список значений, разделенных запятой. Возвращает она заполненный заданными значениями массив, сохраненный в переменной типа Variant. Ниже приведен пример использования функции Array:

```
Dim varArray
' Заполнение массива значениями
varArray = Array(1, 2, 3, 4, 5)
```

Коллекции

Коллекции (они же семейства и множества) – это объекты, которые позволяют хранить произвольное количество элементов любого типа. Элементы в коллекции идентифицируются уникальным ключом, которым может быть не только номер элемента в коллекции, но и значение строкового или другого типа. При программировании на VBA различные коллекции используются очень часто. Например, к коллекции Workbooks нужно обращаться для получения ссылки на объект Workbook нужной рабочей книги, к коллекции Worksheets – для получения ссылки на объект Worksheet нужного рабочего листа и т. д.

В VBA коллекции реализованы во встроенном классе Collection. Создание объекта Collection ничем не отличается от создания объекта другого типа:

```
Dim col As New Collection
```

или

```
Dim col As Collection
Set col = New Collection
```

Добавление элементов

Для добавления элементов в коллекции реализован метод Add, имеющий следующий формат:

```
Ссылка. Add Элемент [, Ключ][, Добавить_перед][, Добавить_после]
```

Единственным обязательным параметром метода Add является значение добавляемого элемента. Элемент может быть константой или переменной любого типа, кроме типа, определенного пользователем. При добавлении элемента можно указать ключ, который будет однозначно идентифицировать элемент в коллекции. Ключ – это любое значение типа Variant.

По умолчанию новые элементы добавляются в конец коллекции. Для изменения порядка добавления элементов используются параметры Добавить_перед и Добавить_после, с помощью которых указывается номер или ключ того элемента, перед которым или после которого нужно вставить новый элемент. Нумерация элементов в коллекции начинается с единицы.

Ниже приведены примеры инструкций, добавляющих новые элементы в коллекцию:

```
col.Add «Value1», «Key1»
col.Add «Value2», «Key2»
col.Add «Value0»,"Key1"
```

Количество элементов в коллекции

Для определения количества элементов в коллекции используется свойство только для чтения Count. При его получении возвращается значение типа Long. Пример получения количества элементов коллекции приведен ниже:

```
lngCount = col.Count
```

Удаление элементов из коллекции

Для удаления элементов из коллекции используется метод Remove:

Ссылка. Remove Номер

Единственным параметром метода Remove является номер элемента в коллекции или его ключ. Например, для удаления значения «Value2», добавленного в коллекцию при рассмотрении метода Add, можно использовать инструкцию

```
col.Remove 3
```

или

```
col.Remove «Key2»
```

Доступ к элементам коллекций

Для получения значений элементов коллекции используется метод Item, который возвращает значение соответствующего элементу типа:

Ссылка. Item (Номер)

Единственным параметром метода I tern является Номер – это ключ элемента или его порядковый номер в коллекции. Например, для получения значения элемента с ключом «Keyl» (или номером 2) можно использовать следующие инструкции:

```
val = col.Item(«Key1»)
val = col.Item(2)
```

Следует отметить, что Item является для объекта Collection методом по умолчанию, поэтому значения элемента с ключом «Key 1» можно получать и таким образом:

```
val = col(«Key1»)
val = col(2)
```

Определяемые пользователем типы данных

Язык VBA предоставляет программисту достаточно большие возможности для создания и использования специфических типов данных: структур и перечислений.

Структуры

Структура – это сложный тип данных, представляющий собой совокупность переменных, хранящихся и обрабатываемых совместно. Переменные, входящие в состав структуры, называются полями (членами) структуры. В состав структуры могут входить переменные как стандартных, так и определенных пользователем типов данных.

Использование структур в программах зачастую позволяет уменьшить объем и сложность алгоритмов работы с данными. Например, для хранения большого количества записей с именами, адресами и телефонами потребовались бы три массива. При использовании же структуры с полями для имени, адреса и телефона для хранения всей информации потребуется всего один массив (VBA позволяет создавать массивы структур).

Описание структур

Для описания структуры в программе на VBA в блок деклараций модуля необходимо поместить следующую конструкцию:

```
[Public | Private] Type Имя_структуры
Поле1
Поле2
…
ПолеN
End Type
```

Ключевые слова Public и Private задают область видимости структуры (по умолчанию используется Public):

• Public – применяется для описания структуры, которую можно использовать (объявлять переменные этого типа) во всех модулях проекта; недопустимо в модулях класса;

• Private – применяется для описания структуры, которую можно использовать только в том модуле, где эта структура описана.

После ключевого слова Type следует имя описываемой структуры. Внутри блока Type-End Type помещаются объявления переменных-членов структуры. Эти объявления аналогичны объявлениям обычных переменных и отличаются только отсутствием в начале ключевых слов Dim, Static, Private или Public и тем, что в одной строке можно объявлять только одну переменную.

Пусть некоторой компании необходимо хранить данные об имени, фамилии, адресе, телефоне и дате рождения своих сотрудников. Совместно с этими данными нужно хранить информацию о проекте, в котором принимает участие каждый сотрудник. Ниже представлено описание структур, с помощью которых можно реализовать хранение требуемых данных.

```
Type ПроектИнформация
Название As String
Срок_завершения As Date
End Type
Type Сотрудник
```

```
Имя As String
Фамилия As String
Адрес As String
Телефон As String
Дата_рождения As Date
Проект As ПроектИнформация
End Type
```

Далее на этом примере рассмотрим особенности работы со структурами в программах на VBA.

Объявление переменных, содержащих структуры

Объявления переменных, содержащих структуры, выглядят точно так же, как объявления переменных другого типа. Ниже приведен пример объявления одной переменной, содержащей структуру Сотрудник:

```
Dim worker As Сотрудник
```

VBA позволяет создавать массивы любых типов данных, в том числе и структур:

```
Dim workers (15) As Сотрудник
```

В данном случае будет создан массив из 15 структур типа Сотрудник.

Примечание

При объявлении переменных, содержащих структуры, происходит автоматическая инициализация всех полей созданных структур.

Доступ к полям структур

Для доступа к содержимому полей структуры необходимо использовать символ «.» (точка). Ниже приведен пример получения значений полей с именем и телефоном сотрудника:

```
strFirstName = worker.Имя
strPhone = worker.Телефон
```

При доступе к массиву структур получение значений полей будет выглядеть следующим образом:

```
strFirstName = workers(15). Имя
strPhone = workers(15). Телефон
```

Получить значения полей вложенной структуры можно следующим образом (Проект – вложенная структура):

```
strName = worker.Проект. Название
datDate = worker.Проект. Срок_завершения
```

При работе со структурами необходимо помнить, что каждое поле структуры – это переменная, которой можно присваивать значение. Ниже приведен пример заполнения структуры с информацией о сотруднике:

```
worker.Имя = «Иван»
worker.Фамилия = «Иванов»
worker.Адрес = «ул. Первомайская, д. 100, кв. 5»
```

```
worker.Телефон = «(095) 200 00 00»
worker.Дата_рождения = «12.03.1978»
worker.Проект. Название = «План здания»
worker.Проект. Срок_завершения = «15.09.2005»
```

Содержимое полей структур можно использовать в любых корректных выражениях, например:

```
strFullName = "Имя: " & worker.Имя & ", фамилия: " & worker.Фамилия
```

При использовании заполненной чуть выше структуры (с информацией о сотруднике Иванове) строка strFullName в результате обработки выражения получит значение" Имя: Иван, фамилия: Иванов".

Перечисления

VBA позволяет определять целочисленные типы данных с ограниченным количеством значений – перечисления. Каждому значению перечисления соответствует идентификатор.

Использование перечислений, во-первых, позволяет оградить программиста от ошибок (не нужно знать значения элементов перечислений), а во-вторых, может повысить читаемость программного кода, так как вместо малоинформативных значений типа 167353b программе используются идентификаторы типа Actions ave. Использование перечислений также избавляет от необходимости создания глобальных целочисленных констант, которые используются только как значения параметров функций и процедур.

Описание перечислений

Для описания перечисления в блок деклараций модуля необходимо поместить следующую конструкцию:

```
[Public | Private] Enum Имя_перечисления
Идентификатор1 [= Значение1]
Идентификатор2 [= Значение2]
…
Идентификатор1 [= ЗначениеN]
End Enum
```

Ключевые слова Public и Private задают область видимости перечисления точно так же, как для структуры (см. выше).

После ключевого слова Enum следует имя описываемого перечисления. Внутри блока Enum—End Enum задаются идентификаторы значений перечисления и, если нужно, сами значения, которые сопоставляются с идентификаторами.

Если значение элемента перечисления явно не указывается, то оно автоматически формируется следующим образом:

• если элемент перечисления первый, то тогда ему присваивается нулевое значение;

• если элемент не первый, то его значение равняется значению предыдущего элемента, увеличенному на единицу.

Ниже приведен пример описания перечисления:

```
Enum MyEnum
value1
value2 = 100
value3
```

```
End Enum
```

В приведенном примере создается перечисление, содержащее три идентификатора и значения. При этом с идентификаторами значения сопоставлены следующим образом: value1 имеет значение 0, value2 – значение 100, а value3 – 101.

Использование перечислений

Объявление переменных для перечислений ничем не отличается от объявления переменных других типов. Ниже приведены примеры объявления переменной и массива переменных для перечисления MyEnum:

```
Dim EnumValue As MyEnum
Dim EnumValues(255) As MyEnum
```

Таким переменным можно присваивать любые численные значения, но можно (и даже нужно) использовать идентификаторы этих значений. Например:

```
EnumValue = value1
EnumValues(100) = value3
```

Идентификаторы значений элементов перечисления можно использовать во всех выражениях, в которых употребляются переменные с типом соответствующего перечисления. Например:

```
If EnumValue = value2 Then…
```

Здесь значением выражения EnumValue = value2 является True, если EnumValue имеет значение value2 (или 100), и False – в противном случае.

Управление выполнением программы

Язык VBA поддерживает ряд способов управления порядком выполнения инструкций программы в пределах функции или процедуры: инструкции безусловного и условного перехода, циклы. Большое количество этих инструкций и наличие различных вариантов обеспечивают максимально эффективное и удобное их использование при написании программ.

Циклы

В VBA реализовано несколько способов организации циклов. Их разнообразие и гибкость играют существенную роль в упрощении программ на языке VBA, а также во многом способствуют повышению наглядности программного кода.

VBA поддерживает четыре вида циклов: обычный цикл For-Next, цикл For Each-Next для просмотра элементов массивов и коллекций, циклы While-Wend и Do-Loop. Циклы различных видов могут быть вложены друг в друга. Рассмотрим подробно каждый из приведенных циклов.

Цикл For-Next

Цикл For-Next в VBA является самым простым и очень часто используемым. Формат данного цикла следующий:

```
For Счетчик = Начальное_значение To Конечное_значение [Step Шаг]
[Инструкции]
[Exit For]
[Инструкции]
Next [Счетчик]
```

Здесь Счетчик – это переменная-итератор любого численного типа. Начальное_значение, Конечное_значение, Шаг – численные значения или идентификаторы переменных численного типа. После ключевого слова Next можно (но не обязательно) указывать идентификатор итератора цикла, конец тела которого обозначает данное ключевое слово. Указывать идентификатор переменной-итератора после Next особенно удобно при организации сложных вложенных циклов.

В начале выполнения цикла итератору присваивается значение элемента Начальное_значение. Инструкции, записанные в теле цикла, выполняются до тех пор, пока значение итератора не превзойдет значение элемента Конечное_значение (станет больше или меньше его в зависимости от направления изменения итератора). Шаг и направление изменения итератора (увеличение или уменьшение) задаются элементом Шаг. Если шаг изменения итератора равен единице, то данный элемент можно опустить.

Для преждевременного выхода из цикла предусмотрена инструкция Exit For. При ее встрече в теле цикла выполнение программы переходит на следующую инструкцию после ключевого слова Next.

Ниже приведен пример трех вложенных циклов For-Next, итераторами которых являются целочисленные переменные i, j и k:

```
For i = 10 To 1 Step -1
For j = 1 To 20
For k = 10 To -10 Step -2
' Выполнение каких-то действий
```

```
…
Next k
Next j
Next i
```

Цикл For Each-Next

Цикл For Each-Next используется для просмотра всех элементов массива или коллекции. Формат данного цикла следующий:

```
For Each Элемент In Контейнер
[Инструкции]
[Exit For]
[Инструкции]
Next [Элемент]
```

Здесь Элемент – это идентификатор переменной-итератора, а Контейнер – идентификатор массива или коллекции. Для цикла For Each-Next допустимый тип итератора зависит от того, просматривается массив или коллекция. При просмотре массива итератор должен иметь тип Variant. При просмотре коллекции итератор может иметь тип Variant или быть ссылкой на объект.

После ключевого слова Next можно (но не обязательно) указывать идентификатор итератора цикла, конец тела которого обозначает данное ключевое слово.

Чтобы преждевременно выйти из цикла, можно использовать такую же инструкцию Exit For, как и для цикла For-Next.

Ниже приведен пример использования цикла For Each-Next для просмотра массива astrStrings:

```
For Each varItem In astrStrings
' Выполнение каких-то действий над элементом varItem
…
Next
```

Цикл While-Wend

While-Wend является самым простым циклом, с помощью которого можно осуществлять определенные действия до тех пор, пока выполняется заданное условие. Формат данного цикла следующий:

```
While Условие
[Инструкции]
Wend
```

Инструкции в теле цикла While-Wend выполняются до тех пор, пока логическое выражение Условие имеет значение True (значение этого выражения вычисляется при каждой итерации).

Ниже приведен пример организации цикла While-Wend:

```
While i < 100
' Действия в цикле
…
i = i + 3
```

```
Wend
```

Следует отметить, что цикл While-Wend является значительно упрощенным и ограниченным с точки зрения разнообразности способов его использования.

Цикл Do-Loop

Цикл Do-Loop предоставляет гораздо больше возможностей при организации циклических действий с проверкой логического условия, чем цикл While-Wend. Проверка логического условия окончания цикла может происходить в начале каждой итерации цикла, при этом формат цикла следующий:

```
Do [While | Until Условие]
[Инструкции]
[Exit Do]
[Инструкции]
Loop
```

Проверка условия может также происходить в конце каждой итерации цикла (тогда выполняется как минимум одна итерация цикла):

```
Do
[Инструкции]
[Exit Do]
[Инструкции]
Loop [While | Until Условие]
```

В приведенных форматах Условие – любое логическое выражение. При использовании ключевого слова While цикл выполняется до тех пор, пока Условие имеет значение True, а при использовании ключевого слова Until – пока Условие имеет значение False. Для выхода из цикла предусмотрена инструкция Exit Do.

Ниже приведен пример использования цикла Do-Loop:

```
Do While i < 100
i = i + 1
Do
j = j + 5
' Действия
…
Loop Until j > 200
Loop
```

Инструкции выбора

Язык VBA поддерживает инструкции, позволяющие осуществлять различные действия в зависимости от выполнения или невыполнения заданных условий, – инструкции выбора If-Then-Else и Select.

Инструкция If-Then-Else

Инструкция VBA If-Then-Else предоставляет возможность выбора одного из действий в зависимости от значений заданных логических выражений. Формат данной инструкции следующий:

```
If Выражение1 Then
[Инструкции1]
[ElseIf Выражение2 Then
[Инструкции2]]
…
[ElseIf ВыражениеN Then
[ИнструкцииN]]
[Else
[Инструкции]]
End If
```

Здесь Выражение1 – ВыражениеN – логические выражения. Если какое-либо из них истинно, то выполняются инструкции, находящиеся после соответствующего ключевого слова If или Elself. Если ни одно из выражений не является истинным, то выполняются инструкции, записанные после ключевого слова Else (если, конечно, это ключевое слово используется).

Рассмотрим пример использования инструкции If-Then-Else:

```
If intAction = 1 Then
' Выполнение сложения
res = a + b
ElseIf intAction = 2 Then
' Выполнение вычитания
res = a – b
ElseIf intAction = 3 Then
' Выполнение умножения
res = a * b
Else
' Заданное действие не поддерживается
'…
End If
```

В приведенном примере с помощью инструкции If-Then-Else выбирается одно из трех поддерживаемых действий для переменных а и Ь: сложение, вычитание или умножение. Действие, которое необходимо выполнять, определяется по содержимому переменной intAction. Если она имеет значение, отличное от 1, 2 и 3, то выполняются инструкции, следующие непосредственно после ключевого слова Else.

Язык программирования VBA также поддерживает упрощенный вариант инструкции If-Then-Else:

```
If Выражение Then [Инструкции1] [Else Инструкции2]
```

Здесь Выражение – это логическое выражение, при истинном значении которого выполняются инструкции после ключевого слова Then. Если Выражение не истинно, то выполняются инструкции после ключевого слова Else (если это ключевое слово используется). При использовании этой формы инструкции If-Then-Else следует учитывать, что она записывается в одну строку (или в несколько строк, но с использованием символа подчеркивания). Также необхо-

димо учитывать, что Инструкции и Инструкции1 представляют собой либо одну инструкцию VBA, либо несколько инструкций, разделенных двоеточием.

Если ключевое слово Else используется, то элемент Инструкции1 может отсутствовать.

Ниже приведены несколько примеров использования сокращенного варианта инструкции If-Then-Else:

```
If a = 1 Then a = 2 Else a = 1
If a = 1 Then a = 2 Else a = 1: b = b + 1
If a = 1 And b = 0 Then Else a = 1: b = b + 1
```

Инструкция Select Case

Select Case позволяет, подобно инструкции If-Then-Else, делать выбор выполняемых программой действий в зависимости от значения заданного аргумента. При большом количестве альтернатив данная инструкция работает быстрее инструкции If-Then-Else, так как значение проверяемого выражения вычисляется только один раз. Формат инструкции Select Case приведен ниже:

```
Select Case Проверяемое_выражение
[Case Список_выражений
[Инструкции]]…
[Case Else
[Инструкции]]
End Select
```

Здесь Проверяемое_выражение – это любое численное или строковое выражение. Список_выражений содержит неограниченное количество выражений, диапазонов значений и условий. Для более детального пояснения ниже приведен формат элемента Список_выражений:

```
Выражение | Мин_значение To Макс_значение | Is Оператор Выражение
_
[, Выражение | Мин_значение To Макс_значение | Is Оператор Выражение]…
```

Значения элементов приведенной конструкции следующие.

Выражение – это любое численное или строковое выражение (тип элемента Выражение должен соответствовать типу элемента Проверяемое_выражение).

Мин_значение To Макс_значение – используется для задания диапазона значений. Элементы Мин_значение и Макс_значение задают минимальное и максимальное значения диапазона соответственно.

• Is Оператор Выражение – используется для задания условий. Позволяет использовать в инструкции Select Case операторы сравнения. Элемент Оператор – это любой оператор сравнения VBA, кроме Is и Like. Элемент Выражение – это любое выражение, тип которого соответствует типу элемента Проверяемое_выражение.

При соответствии значения элемента Проверяемое_выражение одному из заданных выражений, при попадании значения этого элемента в один из диапазонов или при выполнении одного из заданных условий происходит выполнение инструкций, записанных после соответствующего ключевого слова Case. Если ни одна Case-конструкция не сработала, то выполняются инструкции после сочетания ключевых слов Case Else.

Допустим, что в программе необходимо проверять значение численной переменной intTestValue и выполнять одни действия, когда эта переменная имеет значение 1, 2, 3 или 5,

и другие действия – в противном случае. Приведенный ниже фрагмент программы позволяет решить поставленную задачу:

```
Select Case intTestValue
Case 1 To 3, 5
' Действия при значении переменной intTestValue,_
равном 1, 2, 3 или 5
Case Is < 1, Is > 3
' Действия при значении переменной intTestValue _
меньше 1 или больше 3
End Select
```

В данном примере необходимо обратить внимание на то, что значение 5 удовлетворяет обеим Case-конструкциям. При обработке инструкции Select Case VBA просматривает конструкции с ключевым словом Case в том порядке, в котором они следуют в программе. Поэтому в приведенном примере при значении переменной intTestValue, равном 5, выполняются инструкции после первого ключевого слова Case.

Не менее просто с помощью инструкции Select Case можно обрабатывать и строковые значения. Ниже приведен пример, в котором выполняются различные действия при значениях строковой переменной strTestValue, начинающихся со строчной и прописной букв латинского алфавита:

```
Select Case strTestValue
Case «a» To "z"
" Действия, если строка strTestValue начинается _
со строчной буквы латинского алфавита
Case «A» To "Z"
" Действия, если строка strTestValue начинается _
с прописной буквы латинского алфавита
Case Else
" Действия, если строка не начинается с символа _
латинского алфавита
End Select
```

Инструкции безусловного перехода

С помощью инструкций безусловного перехода можно приступать к выполнению части заданной программы без проверки каких-либо условий. К таким инструкциям относятся GoTo и пара GoSub-Return. Однако перед их рассмотрением необходимо ознакомиться еще с одним элементом языка VBA, без которого данные инструкции использоваться не могут, – с метками.

Метки

Метка – это идентификатор VBA или целое число, которое располагается в начале строки и заканчивается двоеточием. Метки используются для указания строк, на которые можно переходить с помощью инструкций GoTo и GoSub. Примеры меток приведены ниже:

```
100:
DoSomeAction:
Перерасчет:
```

После перехода на метку выполняются все инструкции, расположенные после нее до конца процедуры, функции, следующих инструкций GoTo, GoSub или до инструкции Return (см. далее).

Инструкция GoTo

Инструкция GoTo используется для простого перехода к выполнению программы после нужной метки. Формат инструкции следующий:

```
GoTo Имя_метки
```

Инструкции, расположенные после GoTo, выполняются только в том случае, если в программе существуют соответствующие инструкции GoTo или GoSub. Рассмотрим пример использования GoTo:

```
a = 15 + b
If a < 0 Then GoTo 10
' Выполнение действий для значения переменной a больше нуля
10:
' Выполнение действий для значения переменной a меньше нуля
```

Следует отметить, что частое использование инструкции GoTo в программе не рекомендуется, так как может сделать алгоритм слишком запутанным. GoTo нередко допустимо заменить инструкциями выбора либо вызовом процедуры или функции.

Пара инструкций GoSub-Return

Во времена старого доброго языка Basic инструкции GoSub и Return были незаменимы для программиста. Это было связано с тем, что Basic не был даже процедурным языком программирования: в нем не было процедур и функций, все инструкции записывались в виде единой программы. Чтобы не реализовывать несколько раз одинаковые действия, в этой большой программе выделялись отрезки кода, выполняющие типичные действия, – подпрограммы. Подпрограмма начиналась некоторой меткой и оканчивалась инструкцией Return.

При достижении инструкции Go Sub осуществлялся переход на указанную метку (аналогично инструкции GoTo) – начинала выполняться подпрограмма. При достижении инструкции Return происходил возврат из подпрограммы – выполнение программы продолжалось после последней инструкции Go Sub.

Пара инструкций GoSub-Return в языке VBA работает точно таким же образом, но переходы осуществляются только в пределах процедуры или функции. Формат инструкций GoSub-Return такой:

```
GoSub Имя_метки
[Инструкции]
Имя_метки:
[Инструкции подпрограммы]
Return
```

Ниже приведен пример использования инструкций GoSub-Return (в подпрограмме вычисляется квадрат длины гипотенузы прямоугольного треугольника):

```
a = 5
b = 4
GoSub Calculate
```

```
' Другие действия
…
Calculate:
' Подпрограмма
c2 = a ^ 2 + b ^ 2
Return
```

Следует отметить, что при процедурном, а тем более объектно-ориентированном программировании необходимость использования подпрограмм полностью отпала. Роль подпрограмм выполняют функции и процедуры.

Процедуры и функции

В языке VBA программист должен записывать все инструкции своей программы внутри специальных блоков: функций и процедур. Код внутри процедуры или функции представляет собой подпрограмму, выполняющую требуемые действия. Перед рассмотрением способов создания процедур и функций необходимо узнать, чем же различаются эти два вида подпрограмм в VBA.

Процедура – это подпрограмма, которая выполняет действия, не возвращая никакого значения в качестве результата либо возвращая некоторые значения путем изменения переданных ей параметров. Функция в дополнение к возможностям процедуры может возвращать некоторое результирующее значение.

Далее в этом разделе будут рассмотрены особенности создания и использования процедур и функций в программах на VBA.

Объявление процедур

Для объявления процедуры в VBA используется следующая конструкция:

```
[Private | Public] [Static] Sub Имя_процедуры [(Список_аргументов)]
[Инструкции]
[Exit Sub]
[Инструкции]
End Sub
```

Ключевые слова Private и Public данной конструкции задают область видимости процедуры.

• Public – применяется по умолчанию, позволяет создать процедуру, которую можно вызывать из любого места проекта VBA. При использовании в модуле класса она дает возможность создавать общую процедуру (метод) этого класса.

• Private – позволяет создать процедуру, которую можно вызывать только в том модуле VBA, где данная процедура объявлена. При использовании в модуле класса дает возможность создавать личную процедуру (метод) этого класса.

Если в объявлении процедуры используется ключевое слово Static, то значения всех локальных переменных данной процедуры сохраняются между ее вызовами. Это эквивалентно использованию инструкции Static вместо Dim при объявлении каждой локальной переменной внутри процедуры.

Имя_процедуры – это любой корректный идентификатор VBA, который будет употребляться в программе в случае необходимости вызова данной процедуры.

Список_аргументов – содержит описания аргументов, которые принимаются процедурой. Описания аргументов разделяются запятой и имеют следующий формат:

```
[Optional] [ByVal | ByRef] [ParamArray] Имя_аргумента[()] [As
Имя_типа] _
[= Значение_по_умолчанию]
```

Пояснения элементов, используемых в данной конструкции, приведены в табл. 1.7.

Таблица 1.7. Элементы описания аргумента процедуры

Элемент	Пояснение
Optional	Указывает, что аргумент необязательный. Если Optional используется, то все аргументы, записанные после, тоже должны быть необязательными. Данное ключевое слово не может использоваться совместно с ключевым словом ParamArray
ByVal	Указывает, что аргумент передается в процедуру по значению
ByRef	Указывает, что аргумент передается в процедуру по ссылке. Используется по умолчанию
ParamArray	Используется только при описании последнего аргумента процедуры и указывает, что этот аргумент является массивом из произвольного количества значений типа Variant. ParamArray нельзя использовать совместно с ключевыми словами ByVal, ByRef и Optional
Имя_аргумента	Идентификатор локальной переменной процедуры, в которой хранится значение соответствующего аргумента
( )	Указывает, что описываемый аргумент является массивом
As *Имя_типа*	Используется для явного указания типа аргумента. Если этот элемент не используется, то типом аргумента считается Variant
= *Значение_по_умолчанию*	Используется только совместно с ключевым словом Optional для присвоения необязательному аргументу значения по умолчанию. *Значение_по_умолчанию* — любое константное значение соответствующего типа. Для аргумента типа Object единственным допустимым значением является Nothing. Если данный элемент конструкции не используется, то происходит стандартная инициализация аргумента

Для выхода из процедуры предусмотрена инструкция Exit Sub. При ее достижении выполнение программы немедленно переходит к инструкции, следующей за вызвавшей процедуру инструкцией.

Ниже приведен пример процедуры, имеющей два аргумента, при этом второй аргумент необязательный и передается по ссылке:

```
Sub ProcedureExample(ByVal intNumber As Integer, Optional fFlag = True)
' Инструкции процедуры
…
End Sub
```

Проведенную процедуру можно модифицировать так, чтобы вместо необязательного второго параметра процедура принимала произвольное количество аргументов, из которых формируется массив:

```
Sub ProcedureExample(ByVal intNumber As Integer, ParamArray
varArray())
' Инструкции процедуры
…
End Sub
```

Внимание!

Два приведенных примера процедур в программе на VBA одновременно присутствовать не могут. Это обусловлено тем, что язык VBA не

поддерживает перегрузку процедур и функций (создание процедур и функций с одинаковыми именами, но с разными параметрами).

Вызов процедур

Для вызова процедуры в программе на VBA предусмотрена инструкция Call, формат которой приведен ниже:

```
[Call] Имя_процедуры [Список_аргументов]
```

Здесь элемент Имя_процедуры представляет собой идентификатор вызываемой процедуры. Если процедура принимает аргументы, то они должны быть указаны на месте элемента Список_аргументов через запятую. В качестве аргументов в вызывающей процедуре или функции используются константные значения или идентификаторы переменных соответствующих типов.

Эта инструкция позволяет также вызывать и функции, но при этом возвращаемое ими значение получить невозможно.

Примечание

Интересной особенностью инструкции Call является то, что само ключевое слово Call можно опускать. Если ключевое слово Call используется, то список аргументов процедуры необходимо заключать в скобки. В противном случае скобок быть не должно.

Пусть имеется процедура:

```
Sub ProcedureExample(ByVal intNumber As Integer, ParamArray varArray())
' Инструкции процедуры
…
End Sub
```

Пусть также имеется процедура TestExample, в которой необходимо вызывать процедуру ProcedureExample. Процедуру TestExample можно реализовать следующим образом:

```
Sub TestExample()
' Инструкции процедуры
…
' Вызов ProcedureExample
Call ProcedureExample(123, «Значение1», «Значение2», «Значение3»)
' Инструкции процедуры
…
End Sub
```

Если в TestExample не использовать ключевое слово Call, то вызов процедуры будет выглядеть так:

```
' Вызов ProcedureExample
ProcedureExample 123, «Значение1», «Значение2», «Значение3»
```

Далее, перед тем как рассматривать особенности передачи значений в процедуры, целесообразно рассмотреть создание и вызов функций. Это связано с тем, что передача параметров в процедуры и функции происходит одинаково.

Объявление функций. Возврат значения

Для объявления функций в VBA используется следующая конструкция:

```
[Private | Public] [Static] Function Имя_функции [(Список_аргументов)] _
[As Имя_типа]
[Инструкции]
[Имя_функции = Значение]
[Exit Functon]
[Инструкции]
[Имя_функции = Значение]
End Function
```

Приведенный формат объявления функции отличается от объявления процедуры использованием ключевого слова Function вместо Sub, возможностью указания типа возвращаемого функцией значения (после списка аргументов) и возможностью в теле функции присвоить значение переменной с идентификатором, соответствующим идентификатору этой функции (Имяфункции = Значение). При объявлении функций можно использовать все возможности, доступные при объявлении процедур.

Если тип возвращаемого функцией значения не указан, то подразумевается возвращение значения типа Variant.

Для возврата значения функцией необходимо в нужном ее месте присвоить соответствующее значение переменной с таким же идентификатором, как и идентификатор функции. Часто в функции может быть несколько точек, в которых возвращается значение. Если после получения результата нужно немедленно выходить из функции, то после присвоения Имяфункции = Значение используется инструкция Exit Function. Если на протяжении выполнения функции не было использовано присвоение Имяфункции = Значение, то возвращается значение по умолчанию для соответствующего типа данных (см. подраздел об инициализации переменных).

Ниже приведен пример функции, которая вычисляет квадратный корень из переданного ей аргумента (если аргумент меньше нуля, то возвращается значение -1, сигнализирующее об ошибке):

```
Function dhSQR(dblValue As Double) As Double
If dblValue < 0 Then
' Недопустимый аргумент функции
dhSQR = -1
Else
' Вычисление квадратного корня
dhSQR = Sqr(dblValue)
End If
End Function
```

Вызов функций

Для вызова функций допускается также использовать инструкцию Call, например:

```
Call dhSQR(16.324)
```

или

```
dhSQR 16.324
```

Однако при этом теряется возвращаемое функцией значение. Для использования возвращаемого значения идентификаторы функций необходимо включать в выражения справа от знака равенства или другого оператора. Тогда в момент вычисления значения выражения, в состав которого входит идентификатор функции, происходит вызов данной функции, а возвращенное ей значение подставляется в исходное выражение вместо идентификатора функции. Например, в результате обработки каждого из следующих выражений в переменную dblRes будет записано значение 5:

```
dblRes = dhSQR(25)
dblRes = 1 + dhSQR(16)
```

Точно таким же образом вызываются все встроенные функции VBA, например

IsArray, SQR и Array.

Особенности передачи параметров

При создании и использовании процедур и функций необходимо учитывать некоторые особенности передачи параметров в них. Они общие для процедур и функций. Рассмотрим данные особенности.

Позиционная передача параметров

Этот способ передачи параметров наиболее распространен и применяется практически во всех языках программирования. Во всех предыдущих примерах использовался именно позиционный способ передачи параметров в функции и процедуры. Суть данного способа в том, что при вызове процедуры или функции аргументы записываются в том порядке, в котором они указаны при ее объявлении. Пусть, например, необходимо использовать такую процедуру:

```
Sub Procedure(Optional intA As Integer = 25, Optional intB As Integer)
' Инструкции процедуры
…
End Sub
```

Вызов данной процедуры с использованием позиционной передачи параметров выглядит следующим образом:

```
Procedure 12, 56
```

или

```
Call Procedure (12, 56)
```

Отдельного внимания заслуживает передача необязательных параметров. Необязательные параметры можно пропустить, тогда им будет присвоено значение по умолчанию (см. подраздел об объявлении процедур). Ниже приведены примеры вызова процедуры Procedure с пропуском некоторых параметров по умолчанию:

```
Procedure 12 Пропущен второй параметр
Procedure, 12 Пропущен первый параметр
Procedure Пропущены оба параметра
```

Использование именованных параметров

Язык VBA поддерживает также передачу аргументов процедурам и функциям с использованием именованных параметров. Суть данного способа заключается в том, что при вызове функции или процедуры явно указываются имена параметров, которым присваиваются соответствующие значения. При этом порядок передачи не важен.

Для использованной выше процедуры Procedure вызов с применением именованных параметров выглядит следующим образом:

```
Procedure intA:=12, intB:=56
```

или

```
Procedure intB:=56, intA:=12
```

При использовании именованных параметров значительно упрощается передача необязательных параметров. Чтобы пропустить задание такого параметра, ему просто не нужно ничего присваивать при вызове функции или процедуры, например:

```
Procedure intB:=56
```

В данном примере не очень заметны преимущества использования именованных параметров. Другое дело, если необходимо использовать следующую функцию, задав значения только параметров arg3 и arg8:

```
Function dhManyArg(Optional arg1, Optional arg2, Optional arg3,
_
Optional arg4, Optional arg5, Optional arg6, Optional arg7, _
Optional arg8)
' Инструкции функции
…
End Function
```

Очевидно, что инструкция

```
varRes = dhManyArg(,,"text",,,,,142.23)
```

куда менее наглядна и понятна, чем инструкция

```
varRes = dhManyArg(arg3:="text",arg8:=142.23)
```

Передача аргументов по значению или ссылке

Рассмотрим, каким образом в вызываемой процедуре или функции может осуществляться доступ к передаваемым данным. В языке VBA существуют две возможности передачи аргументов: по значению и по ссылке.

При передаче аргумента по значению в вызываемой процедуре или функции создается локальная переменная, в которую копируется все переданное содержимое аргумента. Изменение значения этой локальной переменной никак не отражается на значении переменной, соответствующей аргументу в вызывающей процедуре или функции.

Ниже приведен пример процедуры, принимающей аргумент по значению:

```
Sub TestByVal(ByVal intArg As Integer)
' Какие-то действия, во время которых значение переменной _
intArg изменяется
```

```
...
End Sub
```

Допустим теперь, что в некоторой процедуре присутствует такая инструкция, как TestByVal intValue. После выполнения этой инструкции значение переменной intValue в вызывающей процедуре останется таким же, каким оно было до вызова процедуры TestByVal.

При передаче аргумента по ссылке дело обстоит иначе: при изменении значения переменной-аргумента в вызываемой процедуре или функции изменяется значение соответствующей переменной в вызывающей процедуре или функции.

Ниже приведен пример процедуры, принимающей аргумент по ссылке:

```
Sub TestByRef(ByRef intArg As Integer)
' Какие-то действия, во время которых значение переменной _
intArg изменяется
...
End Sub
```

Допустим, что теперь в другой процедуре присутствует такая инструкция, как TestByRef intValue. После выполнения данной инструкции в вызывающей процедуре значение переменной intValue будет отличаться от первоначального.

Передача аргументов по значению позволяет защитить данные вызывающей процедуры или функции от незапланированного изменения. В то же время передача аргументов по ссылке может использоваться для возврата значений процедурами, а также для возврата функциями более одного значения. Важным моментом является то, что передача больших объемов данных (например, длинных строк) по ссылке происходит значительно быстрее, чем по значению.

Определение и преобразование типов переменных

Данный раздел посвящен рассмотрению возможностей VBA для определения и изменения типа значений в переменных во время выполнения программы.

Определение типов переменных

В VBA предусмотрены возможности получения информации о типе любой переменной во время выполнения программы. Узнать тип переменной (или тип значения, содержащегося в переменной типа Variant) можно несколькими способами.

1. Для идентификации встроенного в VBA типа можно использовать функцию VarType. В качестве аргумента она принимает идентификатор переменной или константное значение некоторого типа. Возвращаемые этой функцией значения и их расшифровка приведены в табл. 1.8.

Таблица 1.8. Значения, возвращаемые функцией VarType

Значение	Константа VBA	Тип переменной или пояснение
0	vbEmpty	Неинициализированная переменная типа Variant
1	vbNull	В переменной типа Variant содержится значение Null
2	vbInteger	Integer
3	vbLong	Long
4	vbSingle	Single
5	vbDouble	Double
6	vbCurrency	Currency
7	vbDate	Date
8	vbString	String
9	vbObject	Object или ссылка на объект определенного типа (например, Application)
10	vbError	Код ошибки
11	vbBoolean	Boolean
12	vbVariant	Variant (возвращается только для массивов с элементами типа Variant)
13	vbDataObject	Объект для доступа к данным
14	vbDecimal	Decimal
17	vbByte	Byte
36	vbUserDefinedType	Тип данных, определенный пользователем
8192	vbArray	Массив

Примечание

Функция VarType возвращает значение vbArray только в сумме со значением, идентифицирующим тип элементов массива. Например, для массива строк функция возвратит значение 8192 + 8 = 8200. Значение же vbVariant возвращается только в сумме со значением vbArray и только для массивов с элементами типа Variant.

2. Для определения типа переменной можно использовать встроенную функцию TypeName. В качестве аргумента она принимает идентификатор переменной или константное значение некоторого типа. Возвращает данная функция строку (тип String) с именем типа аргумента, например «Integer», «String», «Workbook», «Object». Данная функция может вернуть некоторые специфические значения, описание которых приведено в табл. 1.9.

Таблица 1.9. Специфические значения, возвращаемые функцией TypeName

Значение	Пояснение
Error	Аргумент функции содержит код ошибки

Значение	Пояснение
Empty	Аргумент функции является пустой переменной типа Variant
Null	Аргумент функции содержит значение Null
Unknown	Не удалось определить имя типа для аргумента
Nothing	Аргумент функции — ссылка, которая не указывает ни на какой объект (имеет значение Nothing)

3. Для того чтобы определить тип объекта, на который указывает ссылка, допустимо использовать инструкцию TypeOf, имеющую следующий формат: TypeOf Ссылка Is Идентификатор_типа. Данная инструкция возвращает значение True, если ссылка с именем Ссылка указывает на объект, имя типа которого соответствует параметру Идентификатор_типа. В противном случае возвращается значение False. Например, если obj – ссылка на объект Worksheet, то в результате выполнения инструкции TypeOf obj Is Worksheet появится значение True.

Примечание

Инструкция TypeOf работает только для ссылок, имеющих значение, отличное от Nothing. Если в качестве параметра Идентификатор_типа используется Object, то результатом выполнения инструкции будет значение True независимо от типа объекта, на который указывает ссылка.

Преобразование типов

Чтобы типы можно было преобразовывать во время выполнения программы, в VBA предусмотрены специальные функции – функции преобразования типов данных. Все они принимают в качестве аргумента значение типа Variant и возвращают значение соответствующего типа. Ниже приведен формат функций преобразования типов данных:

CBool(Выражение)
CByte(Выражение)
CCur(Выражение)
CDate(Выражение)
CDbl(Выражение)
CDec(Выражение)
CInt(Выражение)
CLng(Выражение)
CSng(Выражение)
CStr(Выражение)
CVar(Выражение)

Далее приведены примеры использования этих функций (переменная varRes имеет тип Variant, а переменная strRes – тип String):

```
varRes = CDec(12.4635246) / CDec(3.14169265359)
strRes = CStr(12.3535)
```

В результате выполнения приведенных инструкций переменная varRes будет содержать значение типа Decimal (использование функции CDec – это единственный способ оперировать с типом данных Decimal), а в переменную strRes будет записано значение «12.3535».

Примечание

При использовании функции CBool необходимо помнить, что к значению True преобразуется любое значение аргумента, не равное нулю. Передавать в функцию CBool разрешается только численные значения. Интересным образом также ведут себя инструкции преобразования к целочисленным типам Clnt, CLng и CByte при наличии дробной части в аргументе. Эти функции округляют дробное число до ближайшего целого четного числа.

Файловый ввод/вывод

Язык VBA поддерживает некоторые возможности для организации файлового ввода/вывода, рассмотрению которых посвящается данный раздел.

Открытие файлов

Для открытия файла в VBA существует специальная инструкция Open, формат которой приведен ниже:

```
Open Имя_файла For Тип_доступа [Access Режим_доступа] [Блокировка] _
As [#]Дескриптор [Len=Длина_записи]
```

В табл. 1.10 даны описания элементов, используемых в приведенной конструкции.

Таблица 1.10. Элементы инструкции Open

Элемент	Описание
Имя_файла	Строка, содержащая полный или относительный путь к открываемому файлу
Тип_доступа	Задает режим, в котором VBA работает с открываемым файлом. Может принимать следующие значения: Append (добавление данных в конец файла), Binary (доступ к файлу в бинарном режиме), Input (последовательное чтение из файла в текстовом режиме), Output (последовательная запись в файл в текстовом режиме) и Random (произвольный доступ к файлу)
Режим_доступа	Задает операции, которые можно выполнять при работе с файлом. Может принимать значение Read (только чтение), Write (только запись) или Read Write (чтение и запись). Read нельзя использовать совместно с типом доступа Output, Write нельзя использовать совместно с Input, а Read Write — ни с Input, ни с Output
Блокировка	Позволяет запретить другим приложениям определенный вид доступа к открываемому файлу. Может принимать следующие значения: Lock Read (запрещает чтение, но разрешает запись в файл), Lock Write (запрещает запись в файл, но разрешает чтение), Lock Read Write (запрещает чтение и запись) и Shared (по умолчанию, разрешает чтение и запись)
Дескриптор	Целочисленное значение от 1 до 511, которое будет однозначно идентифицировать открытый файл
Длина_записи	Задает размер записи в байтах при использовании типа доступа Random. Для типов доступа Output, Input и Append задает количество байт, хранимых в буфере

Ниже приведены примеры инструкций открытия файла D:\MyTextFile.txt для произвольного доступа, для последовательного чтения и записи:

```
Open «D:\MyTextFile.txt» For Random Access Read Write As 1 Len = 100
Open «D:\MyTextFile.txt» For Input As 2
```

```
Open «D:\MyTextFile.txt» For Output As 3
```

Дескрипторы файлов. Функция FreeFile

В среде программирования VBA открытые файлы идентифицируются номерами – дескрипторами. Дескриптор каждого открытого файла должен быть уникальным. Как было видно из примеров открытия файла, программист может сам назначать дескрипторы открываемым файлам (при этом необходимо учитывать, что допустимый диапазон значений дескриптора – 1-511).

Для небольших и простых программ возможность назначать дескрипторы вручную очень удобна. Однако в больших проектах, в которых ведется работа с многими файлами, бывает достаточно сложно следить за правильностью назначения дескрипторов вручную. Для избавления программиста от необходимости контролировать правильность дескрипторов в VBA введена специальная функция FreeFile, имеющая следующий формат:

```
FreeFile ([Диапазон])
```

Данная функция возвращает значение типа Long, которое можно использовать в инструкции Open в качестве дескриптора открываемого файла. Единственным параметром данной функции является необязательный параметр Диапазон, который может иметь значение 1 или 0. Если значение параметра равно 0 (по умолчанию), то функция возвращает дескриптор файла из диапазона 1-255. Если же оно равно 1 – значение из диапазона 256–511. Если свободных дескрипторов в диапазоне нет, то функция возвращает нулевое значение.

Ниже приведен пример использования функции FreeFile:

```
Dim hFile As Long
hFile = FreeFile ' Получение дескриптора для файла
' Открытие файла
Open «D:\MyTextFile.txt» For Output As hFile
```

Закрытие файлов

После того как с открытым с помощью инструкции Open файлом выполнены необходимые действия, его нужно закрыть. Операция закрытия (или освобождения) является обязательной для всех объектов операционной системы, а не только для файлов. При закрытии файла освобождается его дескриптор, а другие приложения получают возможность работать с этим файлом, если он был заблокирован при открытии.

В VBA для закрытия файлов предусмотрены две инструкции: Reset и Close. Формат этих инструкций следующий:

```
Reset
Close [[#]Десктиптор [, [#]Дескриптор]...]
```

Инструкция Reset закрывает все файлы, открытые ранее с помощью инструкции Open. Инструкция Close закрывает только файлы с указанными дескрипторами, например:

```
Close 1, #3, hFile
```

Если при использовании инструкции Close дескрипторы закрываемых файлов не указаны, то она закрывает все открытые ранее файлы.

Чтение из файлов и запись в файлы

В VBA программисту предоставляется множество инструкций для чтения и записи данных при работе с файлами. Эти инструкции разделяются на три группы в соответствии с тем, при каком типе доступа к файлу они используются: последовательном, произвольном или бинарном.

Инструкции последовательного доступа

Описание инструкций последовательного доступа, используемых для работы с файлами, приведено в табл. 1.11.

Таблица 1.11. Инструкции последовательного доступа к файлу

Инструкция	Тип доступа	Описание
Write-#*Дескриптор*, _[*Значения*]	Output, Append	Записывает в файл, заданный параметром *Дескриптор*,

Инструкция	Тип доступа	Описание
		значения переменных или выражений, заданных параметром *Значения* (переменные и выражения разделяются запятой). Если *Значения* содержит несколько переменных или выражений, то при записи в файл их значения разделяются запятой. При этом строковые значения заключаются в кавычки. После записи всех значений в файл в него дописывается символ перехода на следующую строку
Print-# *Дескриптор*, _[*Значения*]	Output, Append	Записывает в файл, заданный параметром *Дескриптор*, значения переменных или выражений, заданных параметром *Значения*. Значения переменных или выражений записываются в одной строке файла и разделяются символом табуляции. После записи всех значений в файл в него дописывается символ перехода на следующую строку
Input-# *Дескриптор*, _*Переменные*	Input	Используется для чтения из файла данных, записанных инструкцией Write. Количество значений, считываемых из файла, определяется количеством переменных в списке *Переменные*
Line-Input-# *Дескриптор*, _ *Переменная*	Input	Считывает строку из файла. Часто используется для чтения из файла данных, записанных инструкцией Print

Ниже приведен пример использования данной функции для считывания из файла первых 10 символов:

```
Sub WriteToFile()
Open «D:\MyTextFile.txt» For Output As 1
' Запись данных в файл
Write #1, «Значение», «Value», 154.32
Print #1, «Слово1», «Слово2», 14.28464
Close 1
End Sub
```

Далее целесообразно привести пример процедуры, в которой осуществляется чтение записанных данных из файла:

```
Sub ReadFromFile()
Dim strVal1, strVal2, dblNumber
Dim strString
Open «D:\MyTextFile.txt» For Input As 1
' Чтение данных из файла
Input #1, strVal1, strVal2, dblNumber
Line Input #1, strString
Close 1
End Sub
```

Кроме приведенных в табл. 1.11 инструкций, в VBA имеется встроенная функция Input, позволяющая считывать из файла заданное количество символов:

```
Input(Количество_символов, [#]Дескриптор)
```

Ниже приведен пример использования данной функции для считывания из файла первых 10 символов:

```
Sub TestInput()
Dim strText As String
Open «D:\MyTextFile.txt» For Input As 1
" Чтение из файла первых 10 символов
strText = input(10, 1)
Close 1
End Sub
```

Инструкции произвольного доступа

При произвольном (Random) доступе файл представляется как совокупность записей, имеющих постоянную длину. Именно запись при данном типе доступа является элементарной единицей информации, которую можно считывать из файла или записывать в файл. Каждая запись имеет свой номер (нумерация начинается с единицы). Для работы с файлами при использовании произвольного доступа в VBA реализованы инструкции Put и Get для записи и чтения информации:

```
Put [#]Дескриптор, [Номер_записи], Переменная
Get [#]Дескриптор, [Номер_записи], Переменная
```

При выполнении инструкции Put значение переменной Переменная помещается в файл на место записи с номером Номерзаписи. Если номер записи не указывается, то данные помещаются в текущую запись файла.

Инструкция Get позволяет считать значение записи с номером Номерзаписи в переменную Переменная. Если номер записи не указан, то считывается текущая запись файла.

Рассмотрим пример, в котором две структуры сначала записываются в файл с помощью инструкции Put, а потом считываются из того же файла, но в обратном порядке:

```
Type Record
intVal As Integer
strName As String * 100
End Type
Sub TestRandomAccess()
Dim rec1 As Record, rec2 As Record
' Заполнение rec1 и rec2 значениями ...
Open «D:\MyRandomAccessFile.txt» For Random Access Read Write _
As 1 Len = Len(rec1)
' Запись данных в файл
Put 1, , rec1
Put 1, , rec2
' Теперь считывание данных из файла
Get 1, 2, rec2
Get 1, 1, rec1
Close 1
End Sub
```

Инструкции бинарного доступа

Бинарный (Binary) доступ к файлу по своей сути идентичен произвольному доступу с тем лишь различием, что запись в файле имеет длину 1 байт. При бинарном доступе к файлу используются те же инструкции Put и Get, что и при произвольном доступе. Также при бинарном доступе для чтения определенного количества байт может быть использована функция Input, о которой было рассказано выше.

Определение конца файла

На практике часто приходится сталкиваться с необходимостью чтения данных из файла, размер которого заведомо неизвестен. Если достигается конец файла, а после этого производится попытка прочитать из него данные, то генерируется ошибка. Для предотвращения подобных ситуаций можно использовать функции EOF и LOF:

```
EOF(Дескриптор)
LOF(Дескриптор)
```

Функция EOF возвращает значение True, если достигнут конец файла, заданного параметром Дескриптор, и False – в противном случае. Если функция EOF возвратила значение False, то читать из файла больше нельзя. Для файлов, открытых в режиме Output, функция EOF всегда возвращает значение True.

Функция LOF позволяет узнать длину файла, заданного параметром Дескриптор. Эта функция возвращает значение типа Long, отражающее длину открытого файла в байтах.

Определение текущей позиции файла

Для определения текущей позиции файла в VBA предусмотрены функции Loc и Seek, имеющие следующий формат:

Loc(Дескриптор)
Seek(Дескриптор)

Обе функции возвращают значение текущей позиции файла, заданного параметром Дескриптор. Однако каждая из этих функций имеет свои особенности.

Функция Loc для файлов, открытых в режиме Random, возвращает номер последней считанной или записанной записи. Для файлов, открытых в режиме Binary, – номер последнего считанного или записанного байта. Для файлов, открытых в режиме последовательного доступа, – текущую позицию в байтах, деленную на 128.

Функция Seek для файлов, открытых в режиме Random, возвращает номер записи, которая будет считана из файла или записана в файл при следующей операции чтения/записи. Для остальных файлов эта функция возвращает номер байта, с которого будет начинаться следующая операция чтения или записи.

Стандартные окна сообщений

Для вывода информации пользователю в арсенале VBA есть очень удобная функция MsgBox. Она позволяет отображать стандартное окно с сообщением (например, об ошибке). Функция MsgBox имеет следующий формат:

MsgBox(Текст_сообщения[, Стиль] [, Заголовок] [, Файл_справки, Индекс_темы])

Здесь Текстсообщения задает строку с текстом сообщения, Заголовок – строку с текстом, который отображается в строке заголовка окна, Файлсправки – имя справочного файла. Если задан аргумент Файлсправки, то должен быть задан аргумент Индекстемы, который идентифицирует тему из заданного файла справки, посвященную выводимому диалоговому окну.

Особого рассмотрения заслуживает аргумент Стиль – он задает значок окна сообщения, отображаемые в этом окне кнопки и другие полезные параметры стиля окна. В табл. 1.12 приведено описание значений, которые объединяются при задании аргумента Стиль с помощью оператора Or.

Таблица 1.12. Значения, используемые для формирования стиля окна

Константа VBA	Значение	Пояснение
vbOKOnly	0	Отображение только кнопки OK
vbOKCancel	1	Отображение кнопок OK и Cancel (Отмена)
vbAbortRetryIgnore	2	Отображение кнопок Abort (Стоп), Retry (Повтор) и Ignore (Пропустить)
vbYesNoCancel	3	Отображение кнопок Yes (Да), No (Нет) и Cancel (Отмена)
vbYesNo	4	Отображение кнопок Yes (Да) и No (Нет)
vbRetryCancel	5	Отображение кнопок Retry (Повтор) и Cancel (Отмена)
vbCritical	16	Отображение значка критической ошибки
vbQuestion	32	Отображение значка вопроса
vbExclamation	48	Отображение значка восклицания
vbInformation	64	Отображение значка информации
vbDefaultButton1	0	Выделяется первая кнопка (слева направо)
vbDefaultButton2	256	Выделяется вторая кнопка
vbDefaultButton3	512	Выделяется третья кнопка
vbDefaultButton4	768	Выделяется четвертая кнопка
vbApplicationModal	0	До закрытия окна пользователь не может работать с приложением
vbSystemModal	4096	До закрытия окна пользователь не может работать со всеми приложениями
vbMsgBoxHelpButton	16384	Отображение кнопки Help (Справка)
vbMsgBoxSetForeground	65536	Не отображать окно сообщения поверх всех других
vbMsgBoxRight	524288	Выравнивать текст по правому краю
vbMsgBoxRtlReading	1048576	Включить режим вывода текста справа налево

После того как пользователь закроет окно, функция возвратит значение, соответствующее нажатой в нем кнопке. Возможные значения, возвращаемые функцией MsgBox, и их объяснения приведены в табл. 1.13.

Таблица 1.13. Значения, возвращаемые функцией MsgBox

Константа VBA	Значение	Нажатая кнопка
vbOK	1	OK
vbCancel	2	Cancel (Отмена)
vbAbort	3	Abort (Стоп)
vbRetry	4	Retry (Повтор)

Константа VBA	Значение	Нажатая кнопка
vbIgnore	5	Ignore (Пропустить)
vbYes	6	Yes (Да)
vbNo	7	No (Нет)

Обработка ошибок времени выполнения

Иногда в процессе работы программы возникают ситуации, когда та или иная инструкция не может быть выполнена, например при попытке расчета значения выражения, в котором происходит деление на ноль, или при обращении к приводу компакт-дисков, когда диска в нем нет. В таких случаях генерируется ошибка времени выполнения. Если в программе не предусмотрен перехват ошибок, то будет выдано соответствующее сообщение об ошибке, а выполнение программы прекратится. Согласитесь, такое поведение программы является отнюдь не самым лучшим и дружественным по отношению к пользователю.

В VBA имеются возможности, позволяющие программе отслеживать возникновение ошибочных ситуаций и адекватно, с точки зрения программиста, на них реагировать.

Перехват ошибок

Для перехвата ошибок времени выполнения в VBA используется специальная инструкция On Error, вставляемая перед тем местом программы, в котором возможно возникновение ошибки. В распоряжение программиста предоставляются три разновидности этой инструкции:

```
On Error GoTo Метка
On Error Resume Next
On Error GoTo 0
```

Первый вариант инструкции On Error активизирует обработчик ошибок (см. подраздел об обработке перехваченных ошибок). При возникновении ошибки после этой инструкции выполнение программы продолжается с метки Метка.

Использование второго варианта позволяет игнорировать все ошибки: при возникновении любой ошибки инструкция, вызвавшая ошибку, пропускается, а выполнение программы продолжается со следующей инструкции.

Третий вариант инструкции On Error отключает перехват ошибок обработчиком, находящимся в выполняемой процедуре или функции.

Обработка перехваченных ошибок

Если в программе используется инструкция вида On Error GoTo Метка, то при возникновении ошибки после этой инструкции выполнение программы продолжается с метки Метка. Программный код, который начинается с данной метки и заканчивается (обычно, но не всегда и не обязательно) инструкцией Resume, называется обработчиком ошибок. В обработчике ошибок программист помещает действия, которые либо исправляют ошибку, либо информируют о ней пользователя. В конец обработчика ошибок обычно помещается один из вариантов инструкции Resume:

```
Resume [0]
Resume Next
Resume Метка
```

При использовании Resume [0] выполнение программы продолжается с той инструкции, в которой произошла ошибка. Если использовать вариант Resume Next, то выполнение программы продолжается со следующей инструкции после той, в которой произошла ошибка. Использование же варианта Resume Метка позволяет продолжить выполнение программы с указанной после Resume метки.

При обработке ошибок важно знать, что в распоряжении программиста всегда имеется глобальная ссылка с именем Err на объект ErrObject. Этот объект хранит подробную информацию о возникшей ошибке (номер ошибки, текст сообщения об ошибке и т. д.). В обработчике эту ссылку можно использовать для уточнения типа, источника ошибки, а также для получения других сведений.

Ниже приведен пример функции с обработчиком ошибок (она пытается записать текст в файл на гибком диске A:):

```
Function dhWriteToFloppy(strText As String) As Boolean
' Включение обработчика ошибок
On Error GoTo ErrHandler
' Выполнение операций с дискетой
Open «A:\Text.txt» For Output As 1
Write #1, strText
Close 1
' Действия выполнены успешно
dhWriteToFloppy = True
ExitFunc:
' Выход из функции до обработчика ошибок
Exit Function
ErrHandler:
' Закрытие файла, если его все-таки удалось открыть
Close 1
Dim strErrMessage As String
' Идентификация ошибки и формирование текста сообщения
Select Case Err.Number
Case 71
strErrMessage = «Нет диска в дисководе»
Case 70
strErrMessage = «Диск защищен от записи»
Case 61
strErrMessage = «Нет места на диске»
Case Else
strErrMessage = Err.Description
End Select
' Отображение сообщения об ошибке
MsgBox strErrMessage, vbExclamation, «Ошибка»
' Продолжение выполнения программы
dhWriteToFloppy = False
Resume ExitFunc
End Function
```

Если запись удается, то функция возвращает значение True. Если возникает ошибка, то выдается соответствующее сообщение, после чего функция возвращает значение False. На примере функции dhWriteToFloppy следует заметить, что при нормальном выполнении программы (без возникновения ошибок) обработчик ошибок выполняться не должен, что достигается выходом из функции до обработчика с помощью инструкции Exit Function.

Классы в VBA

Язык программирования VBA является объектно-ориентированным, хотя и не поддерживает наследование и полиморфизм. VBA-программист может работать с встроенными классами, а также создавать и использовать свои собственные классы.

Создание класса на VBA

Создание класса на VBA отличается от других языков программирования (таких как C ++), в которых описание классов во многом аналогично описанию структур.

В VBA для каждого класса в проект должен быть добавлен отдельный модуль, в который помещается код, реализующий работу класса, – модуль класса. Добавление нового модуля класса осуществляется с помощью команды меню Insert → Class Module (Вставить → Модуль класса) редактора Visual Basic. Имя модулю класса присваивается с помощью окна Properties (Свойства), которое показано на рис. 1.3.

Имя, которое присвоено добавленному модулю, и будет являться именем нового класса. В данном случае имя созданного класса – Class1. В качестве примера с помощью этого класса будет реализовано хранение ссылки на объект, а также хранение некоторой информации об объекте.

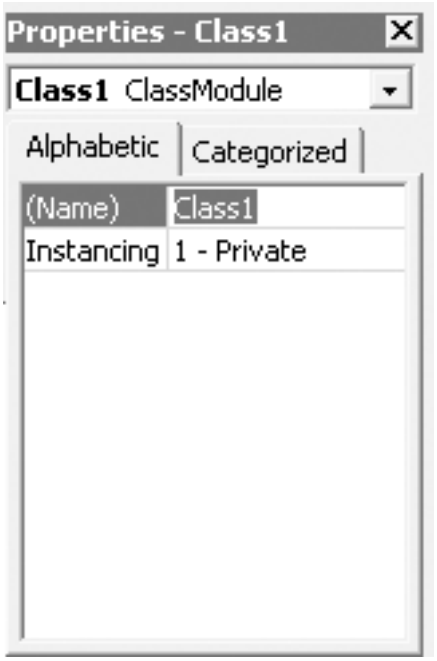

Рис. 1.3. Назначение имени классу

Свойства класса

Свойства для классов в VBA могут быть реализованы двумя способами. Первый способ – это использование в модуле класса общих переменных-членов (объявленных с атрибутом Public). Добавим таким способом свойство в созданный ранее класс Class1, в котором будет храниться строка с описанием данных, содержащихся в объекте-экземпляре этого класса:

```
Public strTag As String
```

Такой способ реализации свойств является самым простым, однако в нем не предусмотрена возможность контролировать правильность задания параметра и осуществлять какие-

либо действия при изменении его значения. Для решения этой проблемы можно использовать второй способ – создание процедур и функций, которые выполняются при установке и получении значений свойств соответственно. Для этих целей в модуле класса применяются обычные объявления процедур и функций, в которых используется ключевое слово Property.

Для получения значения свойства предназначена функция, объявленная с использованием Property Get:

```
[Public | Private] [Static] Property Get Имя_свойства ([Аргументы]) _
[As Имя_типа]
[Инструкции]
[Имя_свойства = Выражение]
[Exit Property]
[Инструкции]
[Имя_свойства = Выражение]
End Property
```

Для присвоения значения свойству, не являющемуся ссылкой на объект, предназначена процедура, объявленная с использованием Property Let:

```
[Public | Private] [Static] Property Let Имя_свойства
([Аргументы,]Значение)
[Инструкции]
[Exit Property]
[Инструкции]
End Property
```

Для присвоения значения свойству, являющемуся ссылкой на объект, предназначена процедура, объявленная с использованием Property Set:

```
[Public | Private] [Static] Property Set Имя_свойства
([Аргументы,]Значение)
[Инструкции]
[Exit Property]
[Инструкции]
End Property
```

Использование процедур и функций с ключевым словом Property очень удобно для создания свойств только для чтения (для этого свойства не реализуются Property Let и Property Set) и свойств только для записи (не реализуется Property Get).

Разберем реализацию свойств ObjectRef и ObjectType для рассматриваемого класса Class1 (частная переменная-член objRef используется для хранения установленной ссылки на объект):

```
Private objRef As Object
Property Set ObjectRef(objNewRef As Object)
' Задание ссылки хранимого объекта
Set objRef = objNewRef
End Property
Property Get ObjectRef() As Object
' Возврат ссылки на хранимый объект
Set ObjectRef = objRef
End Property
Property Get ObjectType() As String
' Возврат имени типа хранимого объекта
```

```
ObjectType = TypeName(objRef)
End Property
```

Методы класса

Любая функция или процедура, описанная в модуле класса, является методом этого класса. Методы делятся на общие (описаны с использованием Public) и частные (описаны с использованием Private).

Ниже приведена реализация метода для созданного нами класса Class1, при обращении к которому на экран выводится сообщение со значениями атрибутов класса:

```
Sub ShowInfo()
' Отображение окна со значением свойства strTag и именем типа _
объекта, на который хранится ссылка
MsgBox "strTag = " & strTag & vbCrLf & _
"Object type = " & ObjectType
End Sub
```

Использование класса в программе

Как было сказано в начале главы, операции со всеми объектами VBA осуществляет только с использованием ссылок. Объявление ссылок на объекты было рассмотрено в разделе, посвященном переменным в VBA. Здесь будет рассмотрено лишь применение объекта созданного ранее класса Class1. Для создания ссылки на объект можно использовать следующее объявление:

```
Dim obj As Class1
```

После создания ссылки сам объект создается с помощью инструкции Set:

```
Set obj = New Class1
```

Объявление переменной ссылки и создание объекта можно также совместить:

```
Dim obj As New Class1
```

Для доступа к свойствам и методам объекта используется точка, например:

```
obj.strTag = «Некоторый текст»
Set obj.ObjectRef = Nothing
MsgBox obj.ObjectType
obj.ShowInfo
```

Ниже приведен пример процедуры, которая использует реализованный класс

```
Class1:
Sub TestClass()
' Создание объекта
Dim obj As New Class1
' Установка свойств
Set obj.ObjectRef = New Collection
obj.strTag = "В этом объекте хранится ссылка на объект
Collection"
' Вызов метода
obj.ShowInfo
```

```
End Sub
```

В результате работы данной процедуры на экран будет выведено окно сообщения, показанное на рис. 1.4.

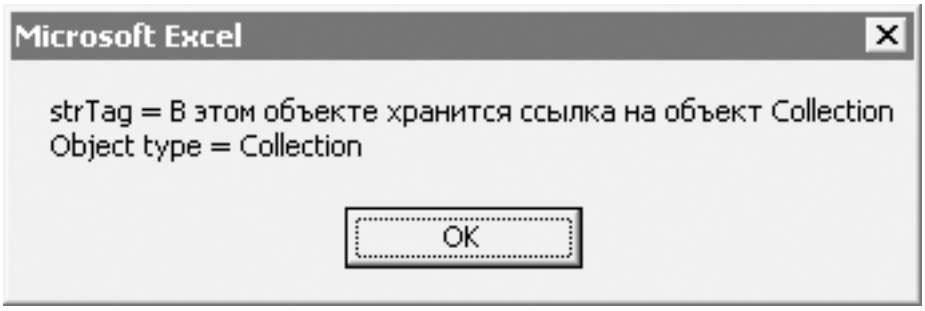

Рис. 1.4. Окно с информацией о свойствах объекта

Как можно заметить, в процедуре TestClass не происходит явного уничтожения ссылки на объект класса Class1. Дело в том, что ссылка obj – локальная переменная процедуры. А при выходе из процедуры данные всех локальных (не статических) переменных уничтожаются, в том числе удаляются и локальные ссылки на объекты.

Использование API-функций в VBA

Иногда даже при программировании на таком языке, как VBA, возникает необходимость использовать API-функции Windows. Эти стандартные функции действительно предоставляют программисту поистине огромные возможности – от управления отображением окон и кнопок до организации сетевого взаимодействия. Всего Windows API (Application Programming Interface) насчитывает около 1000 различных функций.

Объявление API-функций

Чтобы API-функцию можно было вызывать из программы на VBA, ее нужно объявить с использованием инструкции Declare:

```
[Public | Private] Declare Function Имя Lib «Библиотека» _
[Alias «Псевдоним»] [([Аргументы])] [As Имя_типа]
```

или, если API-функция не возвращает значения:

```
[Public | Private] Declare Sub Имя Lib «Библиотека» [Alias «Псевдоним»]
_
[([Аргументы])]
```

Данная инструкция помещается в блоке объявлений модуля. Ключевые слова Public и Private задают область видимости объявляемой API-функции аналогично обычной процедуре или функции. Единственной особенностью является то, что при объявлении API-функции в модуле класса нужно использовать Private. Назначение остальных элементов инструкции Declare поясняется в табл. 1.14.

Таблица 1.14. Элементы инструкции Declare

Элемент	Пояснение
Имя	Идентификатор, по которому в программе будет происходить обращение к API-функции аналогично обращению к обычной процедуре или функции. Если в инструкции Declare ключевое слово Alias не используется, то элемент *Имя* должен полностью совпадать с названием API-функции (с учетом регистра символов). Если Alias используется, то элемент *Имя* может быть любым корректным идентификатором, не используемым для другой процедуры или функции
Библиотека	Имя файла библиотеки (DLL), в которой находится объявляемая функция, например "User" или "User.dll"
Псевдоним	Если в инструкции Declare используется ключевое слово Alias, то *Псевдоним* должен полностью соответствовать имени API-функции в DLL. Использование Alias "*Псевдоним*" позволяет использовать API-функцию в программе на VBA под именем, отличающимся от настоящего
Аргументы	Описание списка аргументов, принимаемых функцией. Аналогично описанию аргументов процедур и функций VBA. Дополнительно при описании аргументов API-функции можно использовать тип данных Any для разрешения передачи в функцию значения любого типа
Имя_типа	Используется для задания типа возвращаемого API-функцией значения

Ниже приведен пример объявления API-функции получения имени текущего пользователя без использования псевдонима:

```
Declare Function GetUserNameA Lib «advapi32.dll» _
(ByVal lpBuffer As String, nSize As Long) As Long
```

а также с использованием псевдонима:

```
Declare Function GetUserName Lib «advapi32.dll» Alias
«GetUserNameA» _
(ByVal lpBuffer As String, nSize As Long) As Long
```

При использовании первой из приведенных инструкций для вызова функции нужно использовать имя GetUserNameA. При использовании второй – имя GetUserName.

Вызов API-функций

Вызов API-функций, объявленных с помощью инструкции Declare Function, ничем не отличается от вызова других функций: программист волен использовать инструкцию Call или употреблять функцию в выражениях. Если API-функция объявлена с использованием Declare Sub, то для вызова может применяться только инструкция Call (аналогично процедуре).

Для закрепления изложенного выше рассмотрим пример использования API-функции GetUserName для получения имени текущего пользователя компьютера:

```
' Объявление API-функции с использованием псевдонима
Declare Function GetUserName Lib «advapi32.dll» Alias
«GetUserNameA» _
(ByVal lpBuffer As String, nSize As Long) As Long
Sub UserName()
Dim strBuffer As String
' Создание строкового буфера для возврата значения функцией
strBuffer = Space(100)
' Получение имени пользователя (ВЫЗОВ API-ФУНКЦИИ). _
Функция возвращает ненулевое значение, если имя пользователя _
записано в strBuffer
If GetUserName(strBuffer, 100) Then
' Вывод имени пользователя
MsgBox RTrim(strBuffer)
Else
MsgBox «Не удалось получить имя пользователя»
End If
End Sub
```

Использование объектов Excel

Программирование на VBA в Microsoft Office чаще всего представляет собой управление объектами соответствующего приложения. Не является исключением и программирование в Excel. Данный раздел ознакомит читателя с основными объектами, встроенными в Excel. Эти объекты используются в подавляющем большинстве примеров (трюков), приведенных в дальнейших главах книги.

Объектная модель Excel

На рис. 1.5 представлена значительно упрощенная структура объектов, доступ к которым имеет программист на VBA.

Как видно из приведенного рисунка, корневым (главным) объектом, доступным в VBA, является Application. Используя ссылку на этот объект, можно манипулировать как самим запущенным приложением Excel, так и такими объектами, как рабочие книги, листы, диаграммы, окна, меню, панели инструментов, – Application предоставляет доступ ко всем объектам Excel.

Объект Application содержит большое количество вложенных объектов. Они могут быть и объектами, с которыми можно взаимодействовать непосредственно (как Assistant – объект для работы с помощником), и представлять собой коллекции, содержащие другие объекты.

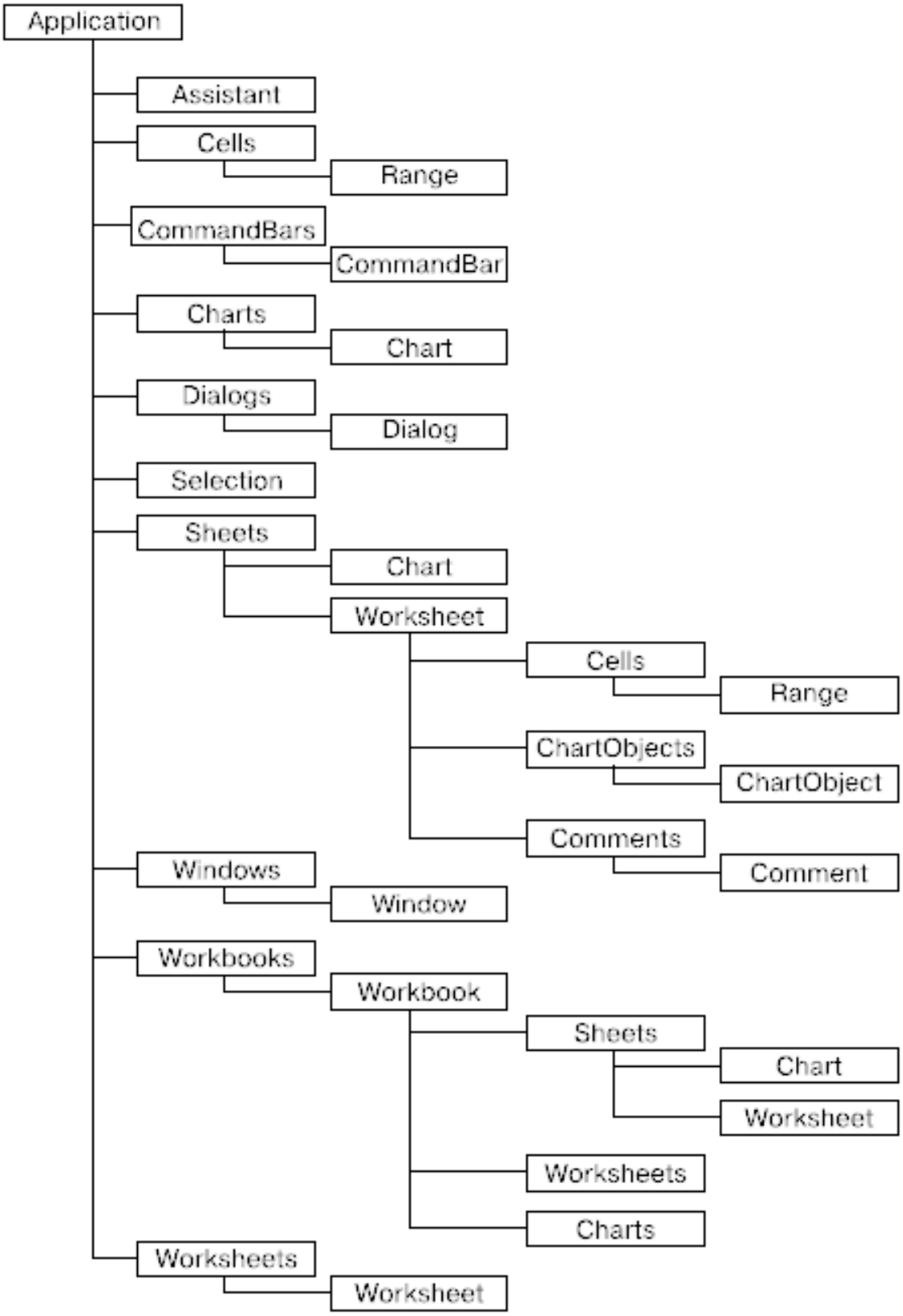

Рис. 1.5. Структура объектов Microsoft Excel

Ниже приведено описание некоторых особенно часто используемых коллекций:
• Cells – коллекция, содержащая все ячейки рабочего листа;
• CommandBars – коллекция, содержащая все меню и панели инструментов;
• Comments – коллекция, содержащая все примечания рабочего листа;

• ChartObjects – коллекция, содержащая все объекты-контейнеры внедренных в рабочий лист диаграмм (по одному объекту на каждую внедренную диаграмму);
• Charts – коллекция, содержащая все листы диаграмм рабочей книги;
• Dialogs – коллекция стандартных диалоговых окон Excel;
• Sheets – коллекция, содержащая все листы книги;
• Windows – коллекция всех отображаемых в Excel окон;
• Workbooks – коллекция, содержащая все открытые в Excel рабочие книги;
• Worksheets – коллекция, содержащая все рабочие листы книги.

Объект Selection (а вернее, свойство объекта Application) предоставляет доступ к данным, выделенным на активном листе рабочей книги. В Selection могут содержаться ссылки на объекты различного типа. Тип зависит от того, что именно выделено на листе (например, если выделены ячейки, то тип объекта Selection – Range).

Особого рассмотрения заслуживает объект Range. Он может содержать одну ячейку, диапазон ячеек или несколько диапазонов ячеек. Этот объект используется при необходимости получения или изменения значений в ячейках таблицы.

Подробная информация о наиболее часто используемых в книге объектах Excel приведена в приложении.

Доступ к объектам Excel из программы

Для доступа к объектам Excel в программах на VBA можно использовать глобальную ссылку на объект Application, которая имеет такой же идентификатор – Application. Например, получение ссылки на выделенные данные может выглядеть следующим образом:

```
Set objSel = Application.Selection
```

Необходимо отметить, что использование ссылки с именем Application во многих случаях подразумевается по умолчанию, поэтому предыдущий пример можно записать так:

```
Set objSel = Selection
```

Аналогичным образом осуществляется доступ к остальным объектам. При этом с коллекциями Excel, такими как Workbooks, Worksheets и пр., работают как с обычными коллекциями VBA, содержащими ссылки на объекты:

```
Worksheets(1).Name = «Sheet 1»
```

Объектом Application предоставляются также ссылки на активную рабочую книгу, активный рабочий лист этой книги, активную ячейку листа, активную диаграмму и т. д. (подобные ссылки объекта Application, а также других объектов рассмотрены в приложении). Эти ссылки нужны для обеспечения возможности быстрого использования информации активного объекта, например:

```
ActiveCell.Value = 15
```

или

```
ActiveSheet.Name = «This sheet is now activated»
```

Глава 2
Рабочая область Microsoft Excel

В данной главе мы рассмотрим порядок работы с основными элементами рабочей области Microsoft Excel – рабочей книгой, рабочим листом и ячейкой (диапазоном). Кроме того, здесь же поговорим о работе с формулами и пользовательскими функциями.

Рабочая книга

Как отмечалось ранее, рабочая книга представляет собой файл Microsoft Excel (обычно с расширением XLSX), в котором хранится и обрабатывается необходимая информация. Используя некоторые несложные приемы, можно расширить возможности рабочей книги. Об этом будет рассказано в текущем разделе.

Автозапуск любимого файла при загрузке Excel

Возможности программы предусматривают автоматический запуск требуемого файла одновременно с открытием Excel. Иначе говоря, при открытии Excel на экране отобразится не пустая рабочая книга (как обычно), а содержимое конкретного файла. Для достижения такого эффекта необходимо поместить требуемый файл в каталог автоматической загрузки – XLStart. Этот каталог расположен в папке с файлами Microsoft Office (например, по адресу C: \ Program Files\Microsoft Office\Office12\XLSTART). При необходимости можно поместить в указанный каталог несколько файлов – в результате при запуске Excel они автоматически будут открыты в разных окнах. Однако для настройки автоматического запуска нескольких файлов удобнее выполнить следующие действия.

1. Открыть все файлы, которые должны автоматически открываться вместе с запуском Excel.

2. На вкладке Вид в группе Окно выбрать команду Сохранить рабочую область и в открывшемся окне по обычным правилам Windows указать путь к каталогу автоматической загрузки (в нашем примере – C: \Program Files\Microsoft Office\Office12\XLStart), после чего нажать кнопку ОК.

В результате в каталог автозагрузки будет помещен файл с расширением XLW (это расширение файла рабочей области). Теперь при запуске Excel будут автоматически запускаться файлы, включенные в эту рабочую область.

Восстановление важной информации из испорченного файла

Использование трюка, описание которого приводится в данном подразделе, позволяет извлечь данные из испорченного файла с помощью встроенного в Excel механизма специальной вставки. Для этого необходимо выполнить следующие действия.

1. Создать две новые пустые книги.

2. В первой книге выделить диапазон ячеек и скопировать его в буфер.

3. Перейти ко второй книге.

4. Во второй книге выделить ячейку A1. На вкладке Главная выбрать из раскрывающегося списка кнопки Вставить (группа Буфер обмена) пункт Вставить связь.

Конец ознакомительного фрагмента.