

АВТОМАТИЗАЦИЯ РУТИНЫ В **EXCEL** **VBA**

ЛАЙФХАКИ ДЛЯ ОБЛЕГЧЕНИЯ
СКУЧНЫХ РАБОЧИХ ЗАДАЧ



ВИКТОР ШИТОВ

- ЭФФЕКТИВНЫЕ РЕШЕНИЯ
ДЛЯ ФИНАНСОВЫХ ЗАДАЧ
- СОЗДАНИЕ ПРОСТЫХ
ВИЗУАЛИЗАЦИЙ ДАННЫХ
- ПРОДУКТИВНАЯ РАБОТА
С ФАЙЛАМИ MICROSOFT



ВИКТОР ШИТОВ

АВТОМАТИЗАЦИЯ
рутины в
EXCEL.
VBA

ЛАЙФХАКИ ДЛЯ ОБЛЕГЧЕНИЯ
СКУЧНЫХ РАБОЧИХ ЗАДАЧ

- ЭФФЕКТИВНЫЕ РЕШЕНИЯ
для финансовых задач
- СОЗДАНИЕ ПРОСТЫХ
визуализаций данных
- ПРОДУКТИВНАЯ РАБОТА
с файлами Microsoft



Москва 2023

УДК 004.67
ББК 32.973.26-018.2
Ш64

Шитов, Виктор Николаевич.

Ш64 Автоматизация рутины в Excel VBA : лайфхаки для облегчения скучных рабочих задач / Виктор Шитов. — Москва : Эксмо, 2023. — 448 с. — (Excel для всех).

ISBN 978-5-04-180209-7

Часто работаете с большим количеством данных? Знаете основные приемы в Excel, но еще не пробовали использовать язык Visual Basic? Тогда самое время разобраться в нем и перестать тратить время на операции, которые можно выполнить одним кликом. С помощью этого руководства вы научитесь правильно использовать синтаксис VBA, сделать тривиальные действия в Excel автоматическими, быстро обрабатывать любые данные и визуализировать их в подходящем формате, легко подстраивать функции VBA под свои нужды и писать любой код самостоятельно.

УДК 004.67
ББК 32.973.26-018.2

ISBN 978-5-04-180209-7

© Шитов В.Н., текст, 2023
© Оформление. ООО «Издательство «Эксмо», 2023

Оглавление

https://t.me/it_boooks

Введение	9
-----------------	----------

1

Введение в офисное программирование	11
--	-----------

Цель разработки	12
Область применения	12
Язык программирования	12
Среда разработки	13
Поддержка ООП	13
Преимущества офисного программирования	13

2

Макросы. Использование макрорекордера	15
--	-----------

Назначение макросов	16
Запись макросов. Использование макрорекордера	17
Воспроизведение макроса	19
Редактирование макроса	26

3

Интерфейс редактора Visual Basic	47
---	-----------

Строка меню	48
Инструментальные панели	49
Окно проектов	50
Форма	51
Создание экранных форм VBA	52
Дополнительные элементы управления	101
Краткое описание дополнительных элементов управления	116
Вставка элементов управления на форму	119
Окно кода	121
Окно выполнения	122
Панель элементов управления	122
Настройка редактора VBA	123

4

Синтаксис VBA	131
----------------------	------------

Общие положения	132
Типы данных	136
Объявление переменных	140
Локальные переменные	151

Объявление статических переменных	152
Объявление констант	154
Создание пользовательских типов	158
Преобразования типов	158
Верхние и нижние колонтитулы	160
Диапазоны в VBA	162
Выражения VBA	169
Операторы VBA	170
Свойства объектов	185
Свойства книги	188
Свойства листа	189
Свойства диаграммы	193
Свойства формы	194
Свойства элементов управления	196
Примеры создания экранных форм	202
Режим конструктора	214
Функция диалога MsgBox	219
Функция InputBox	227
Метод InputBox	231
Выравнивание элементов управления на форме	235
Практическая работа № 1. Преобразование типов данных	239
Практическая работа № 2. Программирование линейных и разветвляющихся алгоритмов	241

5

Организация циклов

245

Оператор цикла For-Next	246
Оператор цикла Do-While	248
Оператор цикла Do-Until	250
Конструкция For Each-Next	251
Практическая работа № 3. Программирование циклических вычислительных процессов	253

6

Пользовательские подпрограммы в VBA

255

Функция	256
Процедуры-функции	258
Аргументы функций	264
Процедуры VBA	265
Вызов процедуры	271
Аргументы процедур	277
Процедуры свойств	280
Практическая работа № 4. Программирование с использованием функций	283

7**Модули VBA****287**

Типы модулей: модули форм, стандартные модули, модули классов	288
Области видимости переменных	289
Модульные переменные	289
Глобальные переменные	290

8**Структурные типы данных****293**

Объявление массивов	294
Практическая работа № 5. Программирование циклических вычислительных процессов с использованием массивов	304
Практическая работа № 6. Программирование с использованием составных пользовательских типов данных	306

9**Объектная модель компонентов Microsoft Office****309**

Коллекции объектов в VBA	310
Свойства объектов	311

10**Создание форм. Обработка событий****315**

События VBA	316
Блокировка и разблокировка событий	426
Практическая работа № 7. Разработка пользовательских диалоговых окон (форм)	428

11**Интеграция с внешними приложениями****433**

Основы автоматизации	434
Ссылка на библиотеку объектов приложения-сервера	434
Отладка	436

Заключение**441****Список литературы****443**

Введение

Microsoft Excel — электронные таблицы, которые предназначены для вычислений, анализа и построения разнообразных отчетов.

Сегодня этот программный продукт — абсолютный лидер среди аналогичных программ электронных таблиц по своей простоте и вместе с тем колоссальным возможностям.

Электронные таблицы Microsoft Excel разработаны компанией Microsoft и обновляются примерно раз в два года. В каждой новой версии Microsoft Excel расширяются возможности этой программы. Все, чему пользователь учится в одной программе, он найдет и в следующей версии, плюс что-то новое.

Даже самая совершенная программа не может охватить всего многообразия реальных ситуаций и задач. Чтобы вы могли самостоятельно решать такие задачи, в программе Microsoft Excel существуют макросы.

Как правило, в учебниках по программе Microsoft Excel об этом понятии говорится вскользь, и это неудивительно: к изучению макросов пользователи подходят уже подготовленными специалистами по Microsoft Excel. Чтобы создавать макросы, нужно видеть проблему, которая решается нестандартным способом.

Умение самостоятельно решать такие проблемы позволит вам не только почувствовать себя крутым специалистом, которых компании зазывают к себе на работу, но и самой организации сэкономить много финансовых средств, которые были бы потрачены на поиск программистов.

Программирование в среде Excel осуществляется на языке Visual Basic for Applications.

Слово *Basic* — это аббревиатура (Beginners Allpurpose Symbolic Instruction Code).

1

Введение в офисное программирование

Цель разработки

Офисное программирование предназначено для разработки приложений, используемых для автоматизации офисной деятельности. У приложений различное назначение: Microsoft Office, OpenOffice, LibreOffice, CorelDRAW Graphics Suite.

Цель — совершенствовать работу с документами под требования конкретного пользователя. Бывают макросы постоянного и разового применения. Вопрос в том, как выгоднее и быстрее обработать информацию: с помощью макроса или вручную.

Область применения

Макрос входит в состав документов: или одного, или встраивается в шаблон документа. В последнем случае макрос встраивается в каждый создаваемый документ. Макрос может изменять настройки документа или использоваться для обработки других документов. Может, например, изменять параметры шрифта, ориентацию страниц, создавать разделы, изменять ориентацию страниц в каждом из разделов, исправлять одни буквы на другие — і на и, ъ на е, ѿ на ф в дореволюционных книгах и т. д. Как самостоятельная программа — отдельно от документа, рабочей книги, презентации — макрос не используется.

Язык программирования

В офисном программировании, как правило, используется язык Visual Basic for Applications (VBA). В некоторых приложениях (OpenOffice, LibreOffice, CorelDRAW Graphics Suite) используется версия языка StarBasic — в связи с другим набором объектов.

Язык Visual Basic for Applications — специализированная версия языка Visual Basic.

Среда разработки

Среда разработки — приложения Microsoft Office. Каждое имеет свой набор объектов: Документ ((Documents) в Word, Рабочий лист (WorkSheets) и Рабочая книга (Workbook) в Excel) и т. д.

Макросы создаются двумя способами: включением макрорекордера и ручным программированием. Макрорекордер записывает все действия, а программные строки записываются автоматически. Запись макроса вручную выполняется с помощью специального редактора, который загружается из риббона Разработчик.

Поддержка ООП

Язык VBA поддерживает объектно-ориентированное программирование. Документы, листы, книги, презентации, методы и события — это объекты. Например, многие цвета объектов имеют свои имена и/или константы, по ним эти цвета можно вызвать. Каждый элемент на форме и сама форма тоже объекты, у них есть свойства, которые можно изменить.

Объектно-ориентированное программирование — основа визуального программирования, то есть свойства объектов можно изменять не только в тексте программы, но и с помощью специальной панели. Но эта возможность может повлечь ошибки у неопытных пользователей. Поэтому многие преподаватели не разрешают студентам в начале обучения изменять свойства объектов на панели свойств, а использовать их только при обработке события инициализации формы.

Преимущества офисного программирования

Возможность использовать макрорекордер позволяет пользователю изменять документ, не привлекая программистов.

Пользователь может продолжать работать в привычном приложении, получая новые возможности.

Многие приложения (OpenOffice, LibreOffice, Mozilla Firefox) могут изменять возможности с помощью расширений и дополнений, их разрабатывают сторонние программисты. Для этих приложений организованы специальные интернет-магазины и библиотеки, можно скачать и установить такие дополнения.

2

Макросы. Использование макрорекордера

https://t.me/it_boooks/2

Назначение макросов

Макрос — это последовательность команд, которые выполняются в Microsoft Office, OpenOffice, LibreOffice, Corel Graphic Suite. В этих приложениях макросы можно создавать на языках Visual Basic for Applications, его версии StarBasic, многих других. Макросы — самостоятельные программные продукты, но способны выполнятся только внутри одного из приложений. Они могут быть потенциально опасными, поэтому о существовании макросов обычно предупреждают в устаревших форматах: DOC, XLS и др. В новейших форматах (DOCX, XLSX) макросы сохранить невозможно: для этого предусмотрены специальные форматы (DOCM, XSLM). Буква *M* в формате говорит о специализации формата для хранения макросов. Такое разделение форматов по назначению в некоторой степени защищает пользователей от макровирусов при скачивании файлов из интернета.

Во многих приложениях для создания макросов используется язык Visual Basic for Applications (VBA) или его версии (StarBasic), но могут использоваться и другие языки: Python, BeanShell, JavaScript, Lisp (если это предусмотрено). Во многих приложениях, хотя об этом и не сообщается явно, используется язык запросов SQL.

Макросы нужны для автоматизации часто повторяющихся операций. На практике наибольшее применение макросов встречается в электронных таблицах. Поэтому в примерах будут использованы именно электронные таблицы (на примере Microsoft Excel). Рассмотрим несколько простейших примеров.

1. Отделу кадров часто приходится вводить информацию с персональными данными сотрудников: фамилию, имя, отчество. Для выбора ширины столбца можно разработать макрос.
2. В электронных таблицах нужно форматировать заголовки, шапку таблицы: размер шрифта, начертание и т. д.
3. Оформление бланков и отчетов. В столбцах можно создать формулы. Пользователю остается заполнить бланк или отчет числами, по которым немедленно будут сделаны расчеты.

4. Электронная библиотека Максима Мошкова предлагает множество дореволюционных книг. С помощью макрорекордера можно отредактировать текст в современном написании (без ятей и ижиц) и сделать книги удобными для чтения и конвертирования в электронные форматы (FB2, EPUB, PDF, DjVu).

Для создания макросов есть два способа: автоматическое создание команд по принципу магнитофона (авторекордер) и ручное создание в редакторе Visual Basic for Applications. Оба метода взаимосвязаны: в любой момент один может дополнить другой. Например, можно записать некоторую последовательность команд с помощью макрорекордера, а затем отредактировать полученный текст программы в редакторе VBA.

Запись макросов. Использование макрорекордера

Создадим пример. Введем пять записей с ФИО в трех столбцах (строку 7 не вводите), а затем включим макрорекордер, и для каждого столбца будет определена оптимальная ширина (рис. 1).

	A	B	C	D		A	B	C
1	Фамилия	Имя	Отчество		1	Фамилия	Имя	Отчество
2	Амиррова	Дарья	Руслановна		2	Амиррова	Дарья	Руслановна
3	Гаджимурат Амина		Байрамбековна		3	Гаджимуратова Амина		Байрамбековна
4	Копытина Екатерина	Дмитриевна			4	Копытина Екатерина	Дмитриевна	
5	Айнуллина Рушан	Раисович			5	Айнуллин Рушан	Раисович	
6	Коловатой Олеся		Александровна		6	Коловатова Олеся		Александровна
7	Байрамов Тельман		Бюньямутдинович		7	Байрамов Тельман		Бюньямутдинович

Рисунок 1. Столбцы до и после применения макроса

Алгоритм создания макроса: выполните команду **Разработчик** ⇒ **Запись макроса**: укажите имя нового макроса (рис. 2), создайте описание назначения макроса, нажмите на кнопку **OK**.

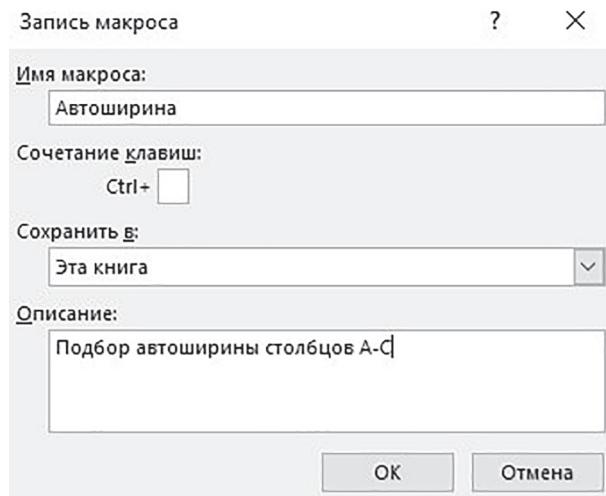


Рисунок 2. Диалоговое окно Запись макроса

Установите автоширину каждого из трех столбцов двойным щелчком между заголовками столбцов, выполните команду **Разработчик** ⇒ **Остановить запись**. Макрос готов, и его программный код можно просмотреть командой **Разработчик** ⇒ **Макрос** ⇒ **Изменить**. Появится следующий программный код:

```
Sub Автоширина ()  
'  
' Автоширина Макрос  
' Подбор автоширины столбцов  
'  
  
Columns ("A:A").EntireColumn.AutoFit  
Columns ("B:B").EntireColumn.AutoFit  
Columns ("C:C").EntireColumn.AutoFit  
End Sub
```

Команда (AutoFit) устанавливает автоширину (Columns) для указанного столбца. Обратите внимание на имена ячеек (A:A, B:B, C:C) — автоширина подбирается по всему столбцу.

Обратите внимание на двойные компьютерные кавычки ("") — в тексте программ допускаются только такие кавычки. Другие типы кавычек (<, >) не допускаются. Распространенная ошибка среди начинающих программистов — вместо одной двойной кавычки часто вводят две одиночные кавычки, найти такую ошибку очень сложно. Такие ошибки при использовании макрорекордера допущены не будут, так как он знает, какие кавычки допустимы, а вот при ручном написании текста программы ошибки с кавычками могут быть. Часто недопустимые кавычки оказываются в тексте программ при:

- 1) копировании текста программы из интернета;
- 2) при сканировании и оптическом распознавании текста, а потом при копировании этого текста и его вставке в свою программу.

Воспроизведение макроса

Добавьте новую строку с длинными ФИО. В примере (рис. 1) фамилия и отчество нового работника (строка 7) не умещаются по ширине столбцов. Выделите ячейку A1.

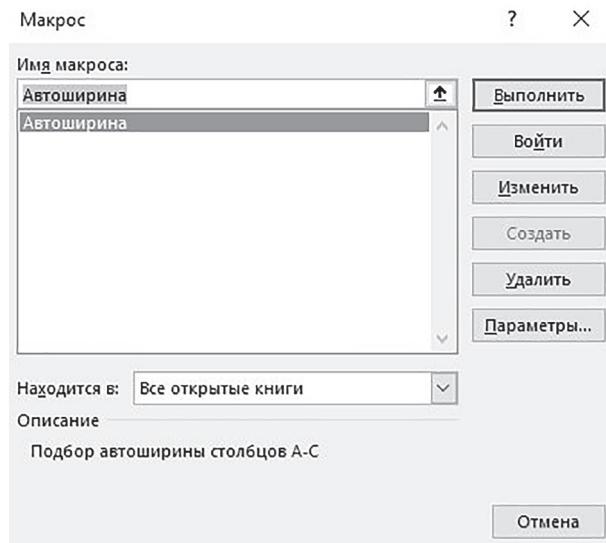


Рисунок 3. Выбор макроса

Для воспроизведения макроса выполните команду **Разработчик** ⇒ **Макрос** ⇒ **Выполнить**. В открывшемся диалоговом окне (рис. 3) выберите имя макроса, который необходимо воспроизвести, и щелкните по нему левой клавишей. После выбора макроса в списке он появляется в поле **Имя макроса**. Чтобы воспроизвести выбранный макрос, необходимо нажать на кнопку **Выполнить**.

Макрос: создание заголовка таблицы

Рассмотрим другой пример, в котором в ячейке A5 вводится заголовок таблицы. Создадим макрос, который должен выполнять следующие действия:

- вводить текст *Отчет за II квартал*;
- устанавливать размер шрифта в 20 пунктов;
- форматировать текст полужирным и курсивным начертанием;
- устанавливать автоширину столбца (рис. 4).

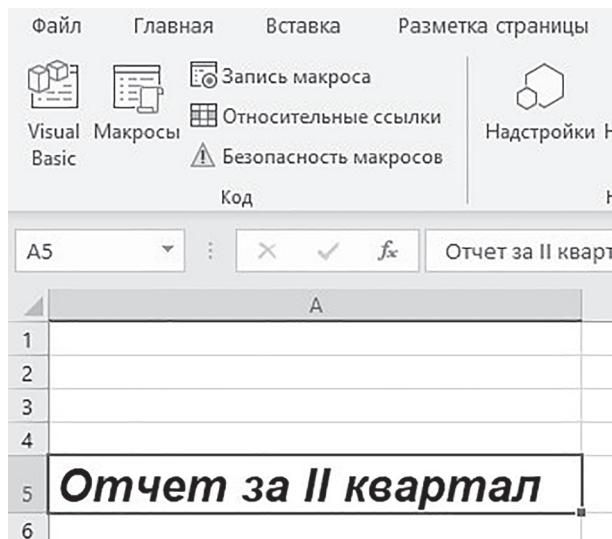


Рисунок 4. Оформленный заголовок

Алгоритм создания макроса в этом примере ничем не отличается от предыдущего: выполняем команду **Разработчик** ⇒ **Макрос** ⇒ **Начать запись**, задаем имя новому макросу,

вводим неформатированный текст, форматируем ячейку полужирным и курсивным начертанием, устанавливаем автоширину столбца, выполняем команду **Разработчик** ⇒ **Макрос** ⇒ **Остановить запись**. Макрос готов. Его программный код можно просмотреть командой **Разработчик** ⇒ **Макрос** ⇒ **Макросы** и в открывшемся диалоговом окне **Макрос** нажать на кнопку **Изменить**. Появится следующий программный код:

```
Sub Макрос2 ()  
'  
' Макрос2 Макрос  
' Макрос записан 25.11.2022 (Шитов)  
'  
  
'  
  
    Columns("A:A").EntireColumn.AutoFit  
    Range("A5").Select  
    ActiveCell.FormulaR1C1 = "Отчет за II квартал"  
    With ActiveCell.Characters(Start:=1,  
Length:=19).Font  
        .Name = "Arial Cyr"  
        .FontStyle = "полужирный курсив"  
        .Size = 20  
        .Strikethrough = False  
        .Superscript = False  
        .Subscript = False  
        .OutlineFont = False  
        .Shadow = False  
        .Underline = xlUnderlineStyleNone  
        .ColorIndex = xlAutomatic  
    End With  
    Rows("5:5").Select  
End Sub
```

Из всего программного кода отметим только значения булева типа, которые могут принимать следующие значения: или *True* (*истина*), или *False* (*ложь*). Опция *True* означает, что данный параметр включен, а опция *False* — параметр выключен.

Как видно из текста программных кодов, есть повторяющиеся команды:

```
Columns ("A:A").EntireColumn.AutoFit
```

Даже если мы ничего не понимаем в программировании, все равно понятно, что для столбца (Columns) устанавливается автоширина (AutoFit), пример работы с которой рассмотрен чуть выше.

В тексте этой программы задан текст, который мы должны вводить в ячейке. С одной стороны, удобно для ввода повторяющихся и не изменяющихся записей. Но в этом примере в следующем квартале текст надписи, наверное, будет «Отчет за III квартал», и нам придется вручную изменять его. Учтем, что человек часто забывает исправлять подобные ошибки, поэтому в нашем примере правильнее вообще не вводить текст. Для этого в программном коде удалим текст между кавычками, оставив только кавычки: "". Будет проводиться форматирование ячейки, а требуемый текст можно потом ввести вручную, уже с заданными параметрами форматирования.

Создайте пустую строку перед последней строкой макроса End Sub. Для этого поместите курсор перед этой строкой, нажмите на клавишу Enter на клавиатуре. Вставьте в пустой строке команду:

```
ActiveCell.FormulaR1C1 = ""
```

Снова выполните макрос: теперь ширина столбца A изменяется, а потом текст удаляется, но ширина столбца не изменяется.

Воспроизведение макроса

Выполните команду **Разработчик** ⇒ **Макрос** ⇒ **Макросы**. В открывшемся диалоговом окне выберите имя макроса и нажмите на кнопку **Выполнить**.

Внешне работа макроса не очень видна, так как он только подготовил ячейки для ввода информации: отформатировал шрифт, но текста в ячейке нет. Если теперь начнем вводить текст, он будет вводиться с полужирным и курсивным начертанием, размером 20 пунктов и другими выбранными параметрами.

Макрос: создание балансового отчета

Рассмотрим более сложный пример — макрос по составлению баланки для создания балансового отчета. В интернете нашли шаблон бюджета и сохранили под именем *Балансовый отчет.xls* (рис. 5).

Создайте новую книгу, задайте ей имя (у нас это Книга2).

Имена рабочих книг специально оговариваются, так как в тексте программы будут фигурировать два имени: Книга2 и *Балансовый отчет.xls*. Была открыта книга *Балансовый отчет.xls*. Выполните команду **Вид** ⇒ **Расположить** и в открывшемся диалоговом окне **Расположение окон** установите переключатель **Расположить** ⇒ **Сверху вниз**.

1

Выполните команду **Разработчик** ⇒ **Макрос** ⇒ **Начать запись**. Задайте имя макросу. Можете оставить его по умолчанию — Макрос1.

2

Ведите в ячейку F1 текст: Балансовый отчет. Выделите эту ячейку, измените размер шрифта на 14 пунктов. Выделите текст и задайте ему полужирное и курсивное начертание. Выделите ячейки с A1 по F1. На риббоне **Главная** нажмите на кнопку **Объединить** и поместить в центре (□).

3

В ячейку A3 введите текст: Описание. Выделите ячейки с A3 по E5. Выполните команду **Главная** ⇒ **Формат** ⇒ **Формат ячеек**. В открывшемся диалоговом окне **Формат ячеек** перейдите на вкладку **Выравнивание**. Установите флажок в индикаторе **Объединение ячеек**. В раскрывающемся списке **Выравнивание по вертикали** выберите значение **По верхнему краю**. Нажмите **OK** для закрытия окна **Формат ячеек**.

4

В ячейке G3 введите текст: Начальный баланс. Выделите ячейки G3, H3, I3. Выполните команду **Главная** ⇒ **Формат** ⇒ **Формат ячеек**. В открывшемся диалоговом окне **Формат ячеек** перейдите на вкладку **Выравнивание**. Установите флажок в индикаторе **Объединение**

ячеек. Нажмите **OK** для закрытия окна **Формат ячеек**.

- 5** Выделите ячейку A7 и введите текст: Дата.
- 6** В ячейку B7 введите текст: Описание статьи. Выделите ячейки с B7 по G7. На риббоне Главная нажмите на кнопку **Объединить и поместить в центре** ().
- 7** В ячейках H7, I7 и J7 введите соответственно текст: Получено, Платеж и Баланс.
- 8** В ячейке G18 введите текст: Получено. Выполните команду Главная ⇒ Формат ⇒ Формат ячеек. В открывшемся диалоговом окне **Формат ячеек** перейдите на вкладку **Выравнивание**. Установите флажок в индикаторе **Объединение ячеек**. В раскрывающемся списке **Выравнивание по горизонтали** выберите значение **По правому краю** (отступ).
- 9** В ячейке G19 введите текст: Платежи. Выполните команду Главная ⇒ Формат ⇒ Формат ячеек. В открывшемся диалоговом окне **Формат ячеек** перейдите на вкладку **Выравнивание**. Установите флажок в индикаторе **Объединение ячеек**. В раскрывающемся списке **Выравнивание по горизонтали** выберите значение **По правому краю** (отступ).
- 10** В ячейке G20 введите текст: Текущий баланс. Выполните команду Главная ⇒ Формат ⇒ Формат ячеек. В открывшемся диалоговом окне **Формат ячеек** перейдите на вкладку **Выравнивание**. Установите флажок в индикаторе **Объединение ячеек**. В раскрывающемся списке **Выравнивание по горизонтали** выберите значение **По правому краю** (отступ).
- 11** Выделите ячейки с H18 по H20. На риббоне Главная нажмите на треугольную кнопку, раскрывающую список кнопки **Цвет заливки** (). Выберите светлобирюзовый.

12

Выделите ячейки с H8 по H17. На риббоне **Главная** нажмите на треугольную кнопку, раскрывающую список кнопки **Цвет заливки**. Выберите серый 25%.

13

Выделите ячейки с I8 по I17. На риббоне **Главная** нажмите на треугольную кнопку, раскрывающую список кнопки **Цвет заливки**. Выберите светло-желтый.

14

Выделите ячейки с J8 по J17. На риббоне **Главная** нажмите на треугольную кнопку, раскрывающую список кнопки **Цвет заливки**. Выберите бирюзовый.

На этом можем отключить создание макроса.

После создания макроса нужно выполнить команду **Разработчик** ⇒ **Макрос** ⇒ **Остановить запись**. Воспроизводить этот макрос не нужно, так как он все равно работать пока не будет.

A	B	C	D	E	F	G	H	I	J
1									
2									
3	Описание						Начальный баланс		
4									
5									
6									
7	Дата	Описание статьи		Получено	Платеж	Баланс			
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18				Получено					
19				Платежи					
20				Текущий баланс					

Рисунок 5. Балансовый отчет, созданный с помощью макроса

При создании макроса мы умышленно забыли некоторые вопросы. Например, форматирование ячеек в диапазоне с A8 по J17 и с H18 по H20; суммирование в ячейках с H18 по H20; выравнивание по центру надписи *Дата* и т. д. Форматирование должно было производиться разными числовыми форматами. В этом макросе мы этого не сделали, чтобы показать в дальнейшем, как производить изменения макроса. Рано или поздно

проблема изменения макросов все равно возникнет, поэтому легче изменить существующий макрос, чем создавать новый. Тем более что при создании нового макроса можно внести ошибки.

Макрос готов. Его программный код можно просмотреть командой **Разработчик** ⇒ **Макрос** ⇒ **Макросы** и в открывшемся диалоговом окне **Макрос** нажать на кнопку **Изменить**. Появится программный код.

Редактирование макроса

Первый макрос, который мы создали (изменение ширины столбцов), не требует никаких изменений: он очень простой. Второй (форматирование заголовка) выявил недостаток: он не объединяет и не центрирует заголовок. Идти путем, которым мы шли в первом макросе, — устанавливать автоширину — неправильно: будет расширен весь столбец, а мы этого совсем не хотим.

Чтобы заставить компьютер поработать вместо нас и показать, что же нам нужно делать, создадим небольшой макрос. Запустите запись макроса (**Разработчик** ⇒ **Макрос** ⇒ **Начать запись**), назовите его любым именем (после того, как посмотрим текст программы и скопируем ее в наш макрос (2), все равно удалим).

Выделите ячейки с A5 по E5. Нажмите на кнопку **Объединить и поместить в центр**, которая находится на инструментальной панели **Форматирование**. Остановите макрос. Выполните команду **Разработчик** ⇒ **Макрос** ⇒ **Макросы**. В открывшемся диалоговом окне выберите имя макроса, который мы создали только что, и нажмите на кнопку **Изменить**.

Выделите весь программный код этого макроса, кроме первых строк с комментарием и последней строки End Sub, и скопируйте его в буферную память (сочетание Ctrl + C). Откройте макрос по форматированию заголовка (для этого выполните команду **Разработчик** ⇒ **Макрос** ⇒ **Макросы**. В открывшемся диалоговом окне выберите имя макроса и нажмите на кнопку **Изменить**). Найдите последнюю строку макроса End Sub и установите перед ней (End Sub) указатель мыши, нажмите

Enter, чтобы создать одну пустую строку. Установите указатель мыши в начало пустой строки и нажмите Shift + Insert для вставки содержимого буферной памяти. Программный код макроса по объединению и центрированию строки будет дописан в конец макроса заголовка. Найдите строку Rows("5:5").Select (она находилась в первоначальном коде перед командой End Sub) и закомментируйте ее. Установить комментарий в VBA означает установить символ одиночной кавычки ('). Все, что находится правее от одиночной кавычки, будет считаться комментарием, а не программой. Таким образом, строка Rows("5:5").Select не будет учитываться при работе программы.

Удалите макрос по объединению и центрированию ячеек. Для этого выполните команду **Разработчик** ⇒ **Макрос** ⇒ **Макросы**. В открывшемся диалоговом окне выберите имя макроса и нажмите **Удалить**. Программа спросит вас о подтверждении удаления. Нажмите **Да**.

Полный программный код макроса по форматированию, объединению и центрированию ячеек будет такой:

```
Sub Макрос()
'
' Макрос2 Макрос
' Макрос записан 22.11.2022 (Шитов)
'

'
Columns("A:A").EntireColumn.AutoFit

Range("A5").Select
ActiveCell.FormulaR1C1 = ""
With ActiveCell.Characters(Start:=1,
Length:=19).Font
.Name = "Arial Cyr"
.FontStyle = "полужирный курсив"
.Size = 20
.Strikethrough = False
.Superscript = False
.Subscript = False
.OutlineFont = False
```

```
.Shadow = False
.Underline = xlUnderlineStyleNone
.ColorIndex = xlAutomatic
End With
' Rows("5:5").Select           так
          оформляется комментарий (')
Range("A5:E5").Select
With Selection
    .HorizontalAlignment = xlCenter
    .VerticalAlignment = xlBottom
    .WrapText = False
    .Orientation = 0
    .AddIndent = False
    .IndentLevel = 0
    .ShrinkToFit = False
    .ReadingOrder = xlContext
    .MergeCells = False
End With
Selection.Merge
End Sub
```

Рассмотрим следующий (созданный нами ранее) макрос.

При выполнении макроса Балансовый отчет необходимо открывать документ, с которого мы рисовали наш макрос. Так как для прокрутки экрана требовалось переключаться с одной книги на другую, эти действия также были записаны в макрос. Нужно найти начало этой ошибки и превратить программные строки в комментарий — и эти команды выполнятся не будут. Чтобы оформить строку комментарием, нужно в самом начале строки ввести одиночную кавычку('). Комментарием необходимо оформить следующие программные строки:

```
' Windows("Балансовый отчет.xlt").Activate
' ActiveWindow.SmallScroll Down:=5
' Windows("Книга2").Activate
```

Разберем эти строки.

В первой происходит активация книги *Балансовый отчет.xlt*. Во второй — задается величина прокрутки на линейке

прокрутки. В третьей строке происходит активация документа Книга2. Книга уже активирована, значит, эта команда не нужна.

Найдем следующий фрагмент, в котором происходит переключение на книгу *Балансовый отчет.xlt*, а затем на Книга2. Сделайте эти программные строки комментарием. Необходимо первым символом в программной строке (точнее — в любом месте строки перед текстом программы в этой строке) сделать символ ('):

```
' Windows("Балансовый отчет.xlt").Activate  
' ActiveWindow.SmallScroll Down:=12  
' Windows("Книга2").Activate
```

Просмотрите свой программный код. Если встретятся обращения к книге *Балансовый отчет.xlt*, а затем к книге Книга2, оформите эти строки как комментарий.

Может возникнуть вопрос: а почему нельзя их удалить — вместо того чтобы оформлять комментарием? В принципе, можно сделать и так, но человеку свойственно забывать собственные ошибки. Поэтому, если у вас возникнет такая же ошибка в будущем, можно открыть текст программы рассмотренного выше макроса и вспомнить, как эту ошибку исправить.

Продолжим анализ нашего программного кода. При создании макроса нам пришлось постоянно перемещаться с помощью линейки прокрутки. Это было необходимо по многим причинам, например, чтобы переместиться в книге Книга2 то вверх, то вниз. Мы создавали сложный документ, таких передвижений было не избежать. В нашем коде это помечалось командами такого типа:

```
ActiveWindow.SmallScroll Down:=7  
ActiveWindow.ScrollRow = 7
```

Если запустим макрос с такими программными строками, увидим, как в процессе выполнения макроса лист книги дергается то вниз, то вверх. Поэтому такие программные строки также необходимо оформить в виде комментария:

```
' ActiveWindow.SmallScroll Down:=7
' ActiveWindow.ScrollRow = 7
' ActiveWindow.SmallScroll Down:=1
' ActiveWindow.ScrollRow = 7
' ActiveWindow.ScrollRow = 5
' ActiveWindow.ScrollRow = 4
' ActiveWindow.ScrollRow = 3
' ActiveWindow.ScrollRow = 2
' ActiveWindow.ScrollRow = 1
' ActiveWindow.ScrollRow = 2
' ActiveWindow.ScrollRow = 3
' ActiveWindow.ScrollRow = 4
' ActiveWindow.ScrollRow = 5
' ActiveWindow.ScrollRow = 6
' ActiveWindow.ScrollRow = 7
' ActiveWindow.ScrollRow = 8
```

В тексте программы есть еще одна строка такого же типа. Ее также необходимо оформить в виде комментария:

```
' ActiveWindow.SmallScroll Down:=-3
```

После внесенных изменений в текст программы макроса нам не нужно заранее открывать файл *Балансовый отчет.xlt*, и лист книги Книга2 не дергается при работе макроса.

При создании макроса Балансового отчета мы как бы случайно (а на самом деле, конечно, не случайно) забыли сделать много другой работы:

- отформатировать ячейки в столбце «Дата» числовым форматом *Дата*;
- отформатировать ячейки в столбцах Получено, Платеж, Баланс числовым форматом **Финансовый** или **Денежный**;
- отформатировать итоговые ячейки с H18 по H20 числовым форматом **Финансовый** или **Денежный**;
- оформить рамками таблицы;

- написать формулы расчета сумм значений в ячейках с H18 по H20 и т. д.

Как объяснить? Во-первых, наш макрос и так довольно сложен для начинающих пользователей. Мы только что рассматривали неточности в ходе его создания. Если усложнить макрос при его создании, был риск не создать его никогда: число ошибок и неточностей на каждом шаге только бы увеличивалось. Во-вторых, постоянно придется изменять макросы, и нужно научиться этому.

На самом деле ничего сложного в усовершенствовании макроса нет. Вы можете сделать это самостоятельно и убедиться: это действительно так. При доработке макроса форматирования заголовка было так: создали отдельный самостоятельный макрос, который делает только то, что мы упустили из виду при создании исходного макроса (форматирование ячеек в столбце Дата числовым форматом Дата). Затем открываем этот макрос, копируем программные строки (это строки без строк комментария и последней строки End Sub) и вставляем их в наш исходный макрос перед строкой End Sub. Так по очереди можем устранить все недостатки исходного макроса.

Пусть вас не смущает длина программы: на выполнение макроса это не сильно влияет. Даже длинный программный код, который мы первоначально создавали, идет довольно быстро, а после усовершенствования стал идти значительно быстрее.

Мы рассмотрели, как исключить отдельные программные строки из процесса выполнения макроса. Это несложно. Сложнее — добавить программные строки в текст ранее созданного.

Одно из решений — создать макрос заново и внести изменения. Рассмотрим недостатки этого метода. Во-первых, при создании нового макроса вы можете внести в него не только новые конструкционные изменения, но и новые ошибки. Во-вторых, должны постоянно помнить о последовательности выполнения операций. Нет гарантий, что не забудете хотя бы один шаг. Наконец, при создании такого большого и сложного макроса, как Балансовый отчет, опять придется удалять ненужные программные строки (которые были созданы при просмотре разных документов, прокрутке линейки и т. д.). Поэтому после создания макроса таким

путем вам придется опять тратить много времени на его анализ и вычищать новые мелкие конструкционные недостатки.

Наконец, нет никакой гарантии, что через некоторое время опять что-то не изменится в законодательстве.

Поэтому такой путь не является оптимальным. Самое простое решение — добавлять программные сроки в уже готовый макрос. Ранее созданный макрос мы сохранили под именем Макрос1.

Приступая к доработке, сначала запускаем макрос создания балансового отчета. Включите запись нового макроса, например под именем МакросФорматБаланс. Выполните следующие действия.

- 1** Выделите ячейки с A8 по A17. Выполните команду **Главная** ⇒ **Формат** ⇒ **Формат ячеек**. В окне **Формат ячеек** на вкладке **Число** в списке **Числовые форматы** выберите значение **Дата**. В списке **Тип** выберите формат в виде **ДД.ММ.ГГ**. Нажмите на кнопку **OK**.
- 2** Выделите ячейки с H8 по H17. Выполните команду **Главная** ⇒ **Формат** ⇒ **Формат ячеек**. В окне **Формат ячеек** на вкладке **Число** в списке **Числовые форматы** выберите значение **Финансовый**. Нажмите **OK**.
- 3** Выделите ячейки с I8 по I17. Выполните команду **Главная** ⇒ **Формат** ⇒ **Формат ячеек**. В окне **Формат ячеек** на вкладке **Число** в списке **Числовые форматы** выберите значение **Финансовый**. Нажмите **OK**.
- 4** Выделите ячейки с J8 по J17. Выполните команду **Главная** ⇒ **Формат** ⇒ **Формат ячеек**. В окне **Формат ячеек** на вкладке **Число** в списке **Числовые форматы** выберите значение **Финансовый**. Нажмите **OK**.
- 5** Выделите ячейки с H18 по H20. Выполните команду **Главная** ⇒ **Формат** ⇒ **Формат ячеек**. В окне **Формат**

ячеек на вкладке **Число** в списке **Числовые форматы** выберите значение **Финансовый**. Нажмите **OK**.

6

Выделите ячейки с H4 по H6. Выполните команду **Главная** ⇒ **Формат** ⇒ **Формат ячеек**. В окне **Формат ячеек** на вкладке **Число** в списке **Числовые форматы** выберите значение **Финансовый**. Нажмите **OK**.

7

Выделите ячейку H18. На инструментальной панели **Стандартная** нажмите на кнопку **Автосумма** (Σ). Установите указатель левой мыши в ячейку H8, нажмите левую клавишу мыши и, не отпуская ее, протащите до ячейки H17 включительно. Отпустите мышь. Так мы указываем диапазон суммирования. Опять нажмите на кнопку **Автосумма**.

8

Выделите ячейку H19. Нажмите **Автосумма**, укажите диапазон суммирования и опять нажмите на кнопку **Автосумма**.

9

Выделите ячейку H20. Нажмите **Автосумма**, укажите диапазон суммирования и опять нажмите на кнопку **Автосумма**.

Отключите запись макроса МакросФорматБаланс. Выполните команду **Разработчик** ⇒ **Макрос** ⇒ **Макросы**. Выделите макрос МакросФорматБаланс. В окне **Макрос** нажмите **Изменить**, чтобы просмотреть текст созданной нами программы:

```
Sub МакросФорматБаланс()
'
' МакросФорматБаланс Макрос
' Макрос записан 27.11.2022 (Шитов)
'

    Range("A8:A17").Select
    Selection.NumberFormat = "dd/mm/yy;@"
    Range("H8:H17").Select
    Selection.NumberFormat = "_($* #,##0.00_); _($* (#,##0.00);_($* ""-""??_);_(@_)"
'
```

```
Range("I8:I17").Select
Selection.NumberFormat = "_($* #,##0.00_); _($*
(#,##0.00);_($* """?_);_(@_)"
Range("J8:J17").Select
Selection.NumberFormat = "_($* #,##0.00_); _($*
(#,##0.00);_($* """?_);_(@_"
Range("H18:H20").Select
Selection.NumberFormat = "_($* #,##0.00_); _($*
(#,##0.00);_($* """?_);_(@_"
Range("H4:H6").Select
Selection.NumberFormat = "_($* #,##0.00_); _($*
(#,##0.00);_($* """?_);_(@_"
Range("H18").Select
Selection.FormulaR1C1 = "=SUM(R[-10]C:R[-1]C)"
Range("H19").Select
Selection.FormulaR1C1 = "=SUM(R[-11]C[1]:R[-2]
C[1])"
Range("H20").Select
Selection.FormulaR1C1 = "=SUM(R[-12]C[2]:R[-3]
C[2])"
End Sub
```

Задача — добавить текст программы только что созданного макроса МакросФорматБаланс к тексту программы Макрос1, который создает Балансовый отчет. Для этого нужно выделить исполняемую часть макроса МакросФорматБаланс, вырезать ее (или скопировать, если текст макроса МакросФорматБаланс потребуется для дальнейшей работы) и вставить в текст макроса Макрос1. Исполняемой частью макроса являются программные строки, за исключением строк Sub [Имя макроса], End Sub, строк с комментарием. Из них Sub [Имя макроса] и End Sub нельзя копировать ни в коем случае. Строки с комментарием не влияют ни на что, поэтому их не копируем только для экономии места.

В данном случае фрагмент, который мы хотим добавить к макросу Макрос1, начинается со строки Range("A8:A17").Select и заканчивается строкой Selection.FormulaR1C1 = "=SUM(R[-12]C[2]:R[-3]C[2])" включительно.

Откройте текст программы макроса Макрос1 командой **Разработчик** ⇒ **Макрос** ⇒ **Макросы**. Выделите Макрос1. В окне **Макрос** нажмите на кнопку **Изменить**. Установите курсор в начало последней строки End Sub, нажмите на Enter для создания пустой строки. Из макроса МакросФорматБаланс скопируйте исполняемый фрагмент программы. Это можно сделать так: выделите строки, которые собираетесь скопировать, и нажмите на сочетание клавиш Ctrl + C (или Ctrl + X, если необходимо эти строки вырезать из макроса). На пустую строку в макросе Макрос1 вставьте скопированный в буферную память фрагмент. Для этого можно нажать на сочетание Shift + Insert (или Ctrl + V).

Для вырезания, копирования и вставки фрагмента в программе Microsoft Visual Basic имеются также специальные кнопки Cut, Copy, Paste и команды раздела меню Edit с этими же именами: Cut, Copy, Paste. Лично я отдаю предпочтение сочетанию клавиш, так как это осуществляется только с использованием клавиатуры и без участия мыши.

В данном случае мы решили вставлять фрагмент программы в самый конец макроса Макрос1. Форматирование происходит после того, как Балансовый отчет уже создан. Но если вы будете создавать макрос для других целей, ничто не мешает вам вставлять фрагмент программы в другие части макроса — в начало, в середину или в другое место. Нужно только понимать, что вставляемый фрагмент может разорвать выполнение созданного ранее макроса. Пока вы не умеете определять место вставки в тело макроса, и мы вставляем наш фрагмент в конец ранее созданного.

Создав макрос, мы вдруг вспомнили, что забыли оформить Балансовый отчет рамками. Это обычная ситуация — вам не раз придется дорабатывать свой макрос. Постоянно изменяется ситуация, для которой мы изначально создавали программы. Появляются другие отделы-заказчики или организации-заказчики, которые начинают понимать нужность и простоту наших отчетов, изменяется законодательство и т. д.

Приступая к доработке макроса, сначала запускаем макрос создания балансового отчета. Затем включаем запись нового макроса, например под именем МакросРамки. Выделите заполняемые ячейки в Балансовом отчете с A1 по J20. Выполните

команду **Главная** ⇒ **Формат** ⇒ **Формат ячеек**. В окне **Формат ячеек** перейдите на вкладку **Граница** и нажмите на кнопку **Внешние и Внутренние**. Нажмите **OK**. Щелкните по любой ячейке для снятия ранее сделанного выделения (я щелкнул по G21, как вы увидите позже). Отключите запись макроса МакросРамки. Выполните команду **Разработчик** ⇒ **Макрос** ⇒ **Макросы**. Выделите макрос МакросРамки. В окне **Макрос** нажмите на кнопку **Изменить**, чтобы просмотреть текст созданной программы:

```
Sub МакросРамки()
    ' МакросРамки Макрос
    ' Макрос записан 27.11.2022 (Шитов)
    '



    Range("A1:J20").Select
        Selection.Borders(xlDiagonalDown).LineStyle =
xlNone
        Selection.Borders(xlDiagonalUp).LineStyle =
xlNone
        With Selection.Borders(xlEdgeLeft)
            .LineStyle = xlContinuous
            .Weight = xlThin
            .ColorIndex = xlAutomatic
        End With
        With Selection.Borders(xlEdgeTop)
            .LineStyle = xlContinuous
            .Weight = xlThin
            .ColorIndex = xlAutomatic
        End With
        With Selection.Borders(xlEdgeBottom)
            .LineStyle = xlContinuous
            .Weight = xlThin
            .ColorIndex = xlAutomatic
        End With
        With Selection.Borders(xlEdgeRight)
            .LineStyle = xlContinuous
            .Weight = xlThin
        End With
    End Sub
```

```

    .ColorIndex = xlAutomatic
End With
With Selection.Borders(xlInsideVertical)
    .LineStyle = xlContinuous
    .Weight = xlThin
    .ColorIndex = xlAutomatic
End With
With Selection.Borders(xlInsideHorizontal)
    .LineStyle = xlContinuous
    .Weight = xlThin
    .ColorIndex = xlAutomatic
End With
Range("G21").Select
End Sub

```

Осталось только вырезать (или скопировать) текст исполняемой программы и вставить этот фрагмент в макрос Балансового отчета. Чтобы открыть текст макроса, опять выполните в Microsoft Excel команду **Разработчик** ⇒ **Макрос** ⇒ **Макросы**. Выделите Макрос1. В окне **Макрос** нажмите на кнопку **Изменить** и перейдите в самый конец текста программы. Установите курсор перед последней строкой (End Sub) и нажмите на Enter для создания пустой строки. В макросе МакросРамки выделите и вырежьте текст программы (начиная со строки Range("A1:J20").Select и заканчивая строкой Range("G21").Select включительно). Вырезать фрагмент программы можно с использованием сочетания Ctrl + X. Вставьте вырезанный фрагмент в подготовленную ранее пустую строку в макросе Макрос1. Это можно сделать с помощью сочетания клавиш Shift + Insert (или Ctrl + V). Если макрос МакросРамки более не нужен, то и сохранять его не нужно. Если он потребуется в дальнейшем, из него нужно не вырезать фрагмент программы, а копировать путем выделения и использования клавиш Ctrl + C.

Поработайте с готовым Балансовым отчетом. Вводите в ячейки данные, изменяйте их, удаляйте. Балансовый отчет реагирует на все изменения, отражая их как в одной ячейке, так и в итоговых.

Макрос окончательно готов. После нескольких дополнительных доработок (удаление и вставка новых фрагментов в основной

макрос) вы получили навыки корректировки макросов. Что касается корректировки текста макроса, этому будет посвящена вся оставшаяся часть книги.

Использование личной книги макросов

При создании макроса всегда предлагаются три варианта сохранения макроса.

Личная книга макросов — будет доступен для любой книги Excel.

Новая книга — макрос создается для новой книги Excel.

Эта книга — создается только для этой книги Excel. Для других этот макрос будет недоступен.

Чтобы макрос был доступен во всех книгах, необходимо сохранить его в Личной книге макросов.

Пока в ней не сохранен ни один макрос, книга недоступна. Поэтому сохраните в Личной книге макросов любой макрос.

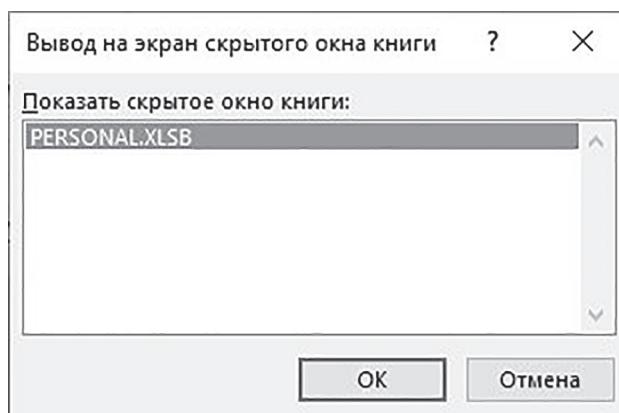


Рисунок 6. Отображение скрытого окна Personal.xlsb

С открытием существующей или созданием новой книги Личная книга макросов всегда присутствует, но по умолчанию скрыта. Чтобы ее открыть, необходимо выполнить команду **Вид ⇒ Отобразить окно**. Откроется диалоговое окно **Вывод**.

на экран скрытого окна книги (рис. 6). Выберите в списке Показать скрытое окно книги имя Personal и нажмите OK.

Первоначально в Личной книге макросов всего один модуль, но можно добавить и новые модули, и новые рабочие листы.

Вполне вероятна ситуация, когда макрос мог быть создан в текущей книге или в новой книге, а не в Личной книге макросов. Поэтому может потребоваться перетащить этот макрос в Личную книгу.

Для этого откройте книгу, в которой хранится макрос. Выполните команду Вид ⇒ Отобразить окно. Выберите в диалоговом окне Вывод на экран скрытого окна книги значение Personal.xlsb, нажмите OK. Переийдите в книгу, где хранится макрос, не входящий в Личную книгу. Выполните команду Разработчик ⇒ Макрос ⇒ Макросы. В диалоговом окне Макрос нажмите на кнопку Изменить, после чего войдете в модуль редактора Visual Basic, где хранится текст программы. Выделите все строки программы, включая первую строку Sub Имя_Макроса() и последнюю End Sub. Нажмите Ctrl + X, чтобы вырезать текст программы. Можно воспользоваться и другими командами вырезания, но так быстрее. Переийдите в Личную книгу макросов. Откройте редактор Visual Basic. В окне проектов Project — VBAProject найдите проект VBAProject(Personal.xlsb). Если слева от имени этого проекта находится символ раскрытия дерева вхождений в виде плюса (+), щелкните по этому знаку, чтобы его развернуть. Список проекта будет развернут, а плюс станет минусом (-). Выделите модуль Module. На инструментальной панели редактора Visual Basic есть кнопка со списком Insert Имя_элемента (вторая кнопка слева) (Module). Раскройте список и выберите значение Module. После этого модуль Module, который был в проекте, будет переименован в Module1, а новый модуль станет Module2. Новый модуль после его создания раскрывается автоматически. (Если автоматически не раскрывается или нужно раскрыть какой-либо модуль, щелкните по нему дважды левой клавишей мыши). Нажмите Shift + Insert (или Ctrl + V) для вставки содержимого буферной памяти, а поскольку в буферной памяти находится текст программы, который мы вырезали из предыдущего макроса, то и будет вставлен текст этой программы. Если

вы пользуетесь другими способами вставлять содержимое буферной памяти, можете вставить программу своим способом. Осталось перейти в Excel, выделить книгу Personal, если она не выделена, и сохранить книгу, в которой раньше был макрос. Но перед сохранением нужно обязательно выделить книгу *Personal.xlsb* и выполнить команду **Вид ⇒ Скрыть окно**, чтобы скрыть Личную книгу макросов.

Если в дальнейшем будете выполнять этот макрос, увидите в диалоговом окне ссылку на Личную книгу макросов *Personal.xlsb*. Этот макрос принадлежит уже не отдельной Книге, а Личной книге макросов.

Относительные и абсолютные ссылки в макросах

По умолчанию в макросах используются абсолютные ссылки. Но можно использовать и относительные.

Чтобы сравнить макросы, использующие абсолютные и относительные ссылки, создадим два макроса — первый с абсолютными, второй с относительными ссылками. Примеры будут одинаковые, чтобы их можно было сравнить.

Так как мы создавали макросы только с абсолютными ссылками, никакие дополнительные действия не нужны.

Чтобы записать первый макрос, выполните следующие действия.

- 1** Перейдите в ячейку A1, как обычно при открытии нового листа.
- 2** Выполните команду **Разработчик ⇒ Макрос ⇒ Начать запись**.
- 3** Сохраните макрос под именем **МакросЗоопарк**.
- 4** В ячейку A1 введите: План поставки экзотических животных в зоопарк *Алиса и Базилио*.

- 5** В ячейку С3 введите: 1 полугодие.
- 6** В ячейку D3 введите: 2 полугодие.
- 7** В ячейку А4 введите: Свинорогий спиногрыз.
- 8** В ячейку А5 введите: Пупырчатый камнедав.
- 9** В ячейку А6 введите: Чешуйчатый грибожуй.
- 10** В ячейку А8 введите: Итого.
- 11** Выделите ячейки с А1 по К1.
- 12** Выполните команду **Разработчик** ⇒ **Макрос** ⇒ **Остановить запись**.

Если теперь выполнить команду **Разработчик** ⇒ **Макрос** ⇒ **Макросы**, в открывшемся диалоговом окне **Макрос** выбрать записанный макрос **МакросЗоопарк** и нажать на кнопку **Изменить**, увидим такой программный код:

```
ActiveCell.FormulaR1C1 = _
    "План поставки экзотических животных
    в зоопарк Алиса и Базилио"
Range("C3").Select
ActiveCell.FormulaR1C1 = "1 полугодие"
Range("D3").Select
ActiveCell.FormulaR1C1 = "2 полугодие"
Range("A4").Select
ActiveCell.FormulaR1C1 = "Свинорогий спиногрыз"
Range("A5").Select
ActiveCell.FormulaR1C1 = "Пупырчатый камнедав"
```

```
Range("A6").Select  
ActiveCell.FormulaR1C1 = "Чешуйчатый грибожуй"  
Range("A8").Select  
ActiveCell.FormulaR1C1 = "Итого:"  
Range("A1:K1").Select
```

Чтобы сравнить полученный макрос с абсолютными ссылками с другим макросом с относительными ссылками, необходимо получить этот макрос. Выполните следующие действия.

- 1** Перейдите в ячейку A1, как обычно бывает при открытии нового листа.
- 2** Выполните команду **Разработчик** ⇒ **Макрос** ⇒ **Начать запись**.
- 3** Сохраните макрос под именем **МакросЗоо2**.
- 4** На риббоне **Разработчик** нажмите на кнопку **Относительные ссылки**.
- 5** Выполните действия с п. 4 по п. 12, которые выполнялись при создании макроса с абсолютными ссылками.

Если теперь выполнить команду **Разработчик** ⇒ **Макрос** ⇒ **Макросы**, в открывшемся диалоговом окне **Макрос** выбрать записанный макрос **МакросЗоо2** и нажать на кнопку **Изменить**, то увидим такой программный код:

```
ActiveCell.FormulaR1C1 = _  
    "План поставки экзотических животных  
в зоопарк Алиса и Базилио"  
    ActiveCell.Offset(2, 2).Range("A1").Select  
    ActiveCell.FormulaR1C1 = "1 полугодие "  
    ActiveCell.Offset(0, 1).Range("A1").Select  
    ActiveCell.FormulaR1C1 = "2 полугодие "  
    ActiveCell.Offset(1, -3).Range("A1").Select  
    ActiveCell.FormulaR1C1 = "Свинорогий спиногрыз"  
    ActiveCell.Offset(1, 0).Range("A1").Select  
    ActiveCell.FormulaR1C1 = "Пупырчатый камнедав"
```

```
ActiveCell.Offset(1, 0).Range("A1").Select  
ActiveCell.FormulaR1C1 = "Чешуйчатый грибожуй"  
ActiveCell.Offset(2, 0).Range("A1").Select  
ActiveCell.FormulaR1C1 = "Итого:"  
ActiveCell.Offset(-7, 0).Range("A1:K1").Select
```

Различия в двух макросах видны с первого взгляда. Во-первых, набор самих команд совершенно другой. Во-вторых, в первом макросе адреса ячеек указываются в явном виде, а во втором макросе ячейки указываются в виде адресов смещения курсора.

Макрос, созданный на основе абсолютных ссылок, ориентирует-ся по адресам ячеек, указанных в макросе. Создавая первый макрос, я специально сделал небольшую ошибку, установил курсор в ячейку A1 до включения записи макроса. Если мы установим курсор в ячейку A1 и выполним команду **Разработчик ⇒ Макрос ⇒ Макросы**, в открывшемся диалоговом окне **Макрос** выберем записанный макрос МакросЗоопарк и нажмем на кнопку **Выполнить**, все действия будут выполнены правильно. Если мы установим курсор в любую другую ячейку, кроме A1, в эту ячейку будет записана строка: *План поставки экзотических животных в зоопарк Алиса и Базилио* (курсив мой), далее, в ячейках, адреса которых находятся в макросе МакросЗоопарк, будут выведены остальные значения. Последняя команда в обоих макросах — выделение ячеек с A1 по K1. Если в первом макросе макрос начал выполняться не с ячейки A1, такого выделения не будет.

Во втором макросе нет адресов конкретных ячеек, вывод значений происходит в матрице со сдвигом относительно предыдущей ячейки, с которой производилось выполнение макроса.

В макросе на основе относительных ссылок невозможна ситуация, которая сложилась в примере с ячейкой A1. Если макрос, созданный на основе абсолютных ссылок, должен выполняться с ячейки A1, для макроса, созданного на основе относительных ссылок, совершенно все равно, с какой ячейки начнется выполнение. Просто адрес этой ячейки станет отправной точкой, от которой будет организован вывод данных. Рассмотрим это подробнее на основе адресов ячеек, на основании которых строился второй макрос.

Перед записью макроса курсор находился в ячейке A1. Следующая запись создавалась в ячейке C3, то есть со сдвигом в матрице 2 ячейки вниз и 2 ячейки вправо. Поэтому и адрес сдвига в этой строке записан как (2, 2). Далее запись создавалась в ячейке D3, то есть со сдвигом относительно ячейки C3 вниз на 0 ячеек и вправо на 1 ячейку. Поэтому адрес сдвига будет (0, 1). Далее запись происходит в ячейке A4, а это сдвиг относительно ячейки D3 вниз на одну ячейку и влево на 3 ячейки. А что такое сдвиг влево? Это отрицательный сдвиг, так как является возвратом. Следовательно, адрес сдвига будет (1, -3).

Теперь, когда мы поняли получение адреса сдвига в матрице ячеек (как положительного, так и отрицательного), можем сказать, что первое число в адресе указывает на вертикальный сдвиг, а второе — на горизонтальный.

Рассмотрим различие команд в обоих макросах.

Метод **Offset** производит сдвиг ячеек относительно предыдущей ячейки. Как это происходит, мы изучили только что.

Синтаксис этого метода следующий:

expression.Offset(RowIndex, ColumnOffset)

Где:

expression — обязательный аргумент, выражение, которое возвращает объект **Range**;

RowIndex — необязательный аргумент, тип **Variant**, количество строк (положительное, отрицательное или 0 (нуль)), на которое диапазон должен быть перемещен. Положительные величины — перемещение вниз, отрицательные величины — перемещение вверх. Значение по умолчанию — 0;

ColumnOffset — необязательный аргумент, тип **Variant**, количество столбцов (положительное, отрицательное или 0 (нуль)), на которое диапазон должен быть сдвинут. Положительные величины являются сдвигом вправо, отрицательные величины — сдвиг влево. Значение по умолчанию — 0.

Так как при создании макросов можно применять и абсолютные, и относительные ссылки, возникает вопрос: какие ссылки лучше использовать. Нужны и те и другие в зависимости от того, какие вопросы требуется решить. Если требуется, чтобы заполнялись строго указанные ячейки, лучше применять абсолютные ссылки. Если необходим какой-то маневр и некоторая свобода действий, лучше использовать относительные.

3

Интерфейс редактора Visual Basic

https://t.me/it_boooks/2

Интерфейс редактора Visual Basic состоит из сложного и взаимосвязанного набора составных частей. Он включает в себя.

- Стока меню.
- Инструментальные панели.
- Окно проектов.
- Окно кода Code.
- Окно выполнения.
- Окно формы.
- Панель элементов управления Toolbox.
- Окно Watch.

Кроме этих элементов интерфейса есть много других второстепенных составных частей.

Строка меню

В разделе меню **File** представлены команды по сохранению редактируемого макроса, его печати, операций импорта и экспорта, удалению компонента приложения — формы, рабочего листа, диаграммы, модуля и т. д.

В разделе меню **Edit** — команды по редактированию, поиску, замене, сдвигу текста относительно левого края. Здесь же представлены информационные команды по методам, функциям, константам и т. д.

В разделе меню **View** — команды по открытию или закрытию окон.

В разделе меню **Insert** представлены команды по вставке в приложение формы, модуля, класса модуля и т. д.

В разделе меню **Format** представлены команды по форматированию элементов управления, в частности, по выравниванию, группировке.

В разделе меню **Debug** представлены команды по отладке приложения.

В разделе меню **Run** — команды по компилированию приложения.

В разделе меню **Tools** представлены команды по настройке параметров и опций редактора VBA.

В разделе меню **Add-Ins** представлена команда создания менеджера приложения.

В разделе меню **Window** — команды по организации окон в редакторе VBA.

В разделе меню **Help** — команды по справочной системе VBA.

Инструментальные панели

В верхней части приложения под меню находятся панели инструментов. Их можно настраивать так же, как и любую инструментальную панель.

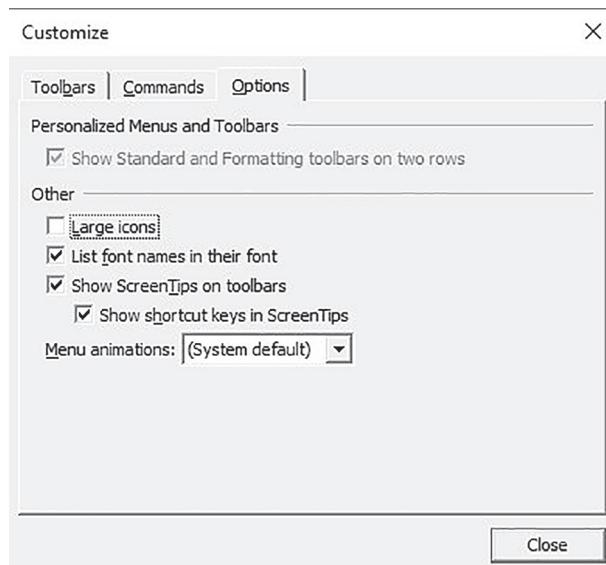


Рисунок 7. Диалоговое окно Customize

По умолчанию открыта только одна инструментальная панель — **Standard**. Чтобы открыть другие, нужно подвести указатель мыши к любой кнопке или свободной области инструментальной панели или полосы меню (она также считается инструментальной панелью) и щелкнуть правой клавишей. Открывается контекстное меню с именами основных инструментальных панелей и командой **Customize**. При выполнении команды **Customize** открывается одноименное диалоговое окно.

Окно **Customize** можно открыть также командой **View ⇒ Toolbars ⇒ Customize** (Вид ⇒ Инструментальные панели ⇒ Настройка) (рис. 7).

На вкладке **Commands** предлагаются кнопки по разделам, примерно соответствующим меню. Если нужна кнопка, которой нет на инструментальных панелях, подхватите ее с вкладки **Commands** и перетащите на одну из открытых панелей. Указатель мыши будет показывать точку вставки. Отпустите мышь.

Окно проектов

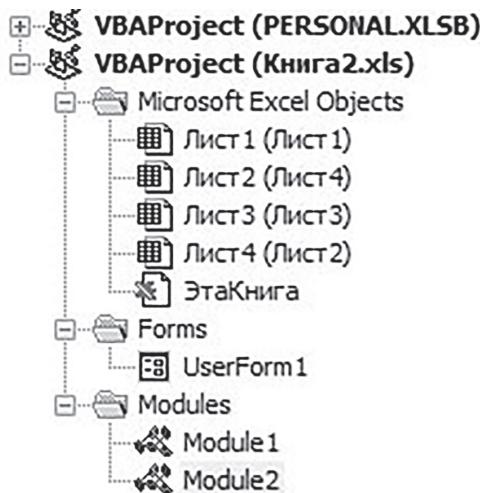


Рисунок 8. Окно проектов

В окне проекта отображаются все элементы приложения: формы, модули, классы и т. п., сгруппированные по категориям (рис. 8). В VBA все разрабатываемые приложения называются проектами. Проект содержит несколько групп компонентов (рабочие книги, листы, формы, модули и т. п.). Все приложения VBA строятся по модульному принципу, поэтому и объектный код состоит не из одного большого файла, а из нескольких частей. Несколько приложений могут объединяться в группы.

Для записи проекта выберите команду **File ⇒ Save Книга№**. После этого можно задать конкретное имя проекта, который сохраняется не как самостоятельный, а как проект, входящий в какую-то Книгу Excel.

Форма

Форма — это панель, на которой расположены все остальные элементы управления (рис. 9). Элементы управления располагаются либо на форме, либо на рабочем листе. Располагать элементы управления на форме удобнее, чем на рабочем листе. Форму можно открыть, когда это удобно, и закрыть после того, как выполнены действия, запускаемые с этой формы.

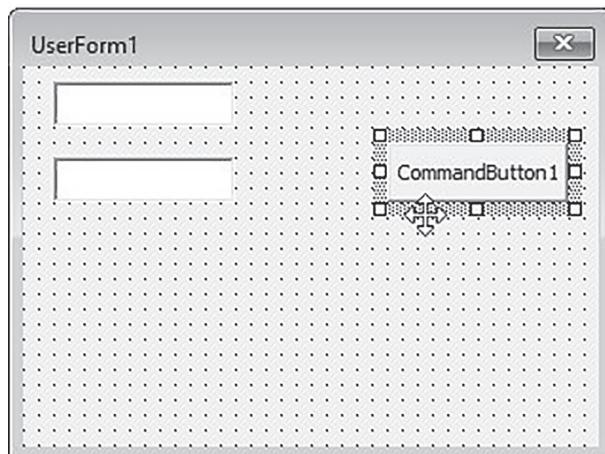


Рисунок 9. Форма

На форме расположена сетка, позволяющая точно расположить элементы управления относительно друг друга. Размер сетки определяется из вкладки **General** диалогового окна **Options**. Узлы сетки привязывают объекты, располагающиеся на форме.

Чтобы отменить выделение всех элементов управления, щелкните по пустому месту на форме. Чтобы выбрать элементы управления в контейнере, сначала отмените выделение контейнера.

Создание экранных форм VBA

В какой бы программе вы ни работали, в ней, как правило, имеются диалоговые окна. В этом разделе мы научимся создавать такие диалоговые окна.

Вставка новой формы

В открывшемся редакторе Visual Basic на инструментальной панели **Standard** найдите кнопку с раскрывающимся списком **Insert UserForm** (это имя устанавливается по умолчанию). Раскройте список. Выберите значение **UserForm**. Этим вы создадите форму, на которую будете помещать элементы управления.

Другой способ вставить форму — выполнить команду **Insert ⇒ UserForm**.

Первой новой форме по умолчанию присваивается имя **UserForm1**. Каждая следующая форма в приложении будет увеличивать свое имя на единицу. Имя формы можно менять в окне свойств **Properties**. Для этого на вкладке **Alphabetic** найдите самую верхнюю строку (**Name**). В поле свойства напишите новое имя формы (если это необходимо) английскими или русскими буквами.

В окне **Properties** есть вкладки **Alphabetic** и **Categorized**. Разница между ними в том, что на вкладке **Alphabetic** все свойства рассортированы по алфавиту, а на вкладке **Categorized** — по категориям. По умолчанию открыта вкладка **Alphabetic**. С какой вкладкой работать, решает пользователь.

Многие путают имя формы с заголовком формы. Имя формы (**Name**) — имя, под которым происходит обращение к форме в программе, и под этим же именем форма фигурирует в окне проектов. Заголовок формы (**Caption**) — текст в системной полосе формы.

На вставленной форме нет ни одного элемента управления, так как программа не знает, какие из них конкретно вам потребуются. На форме есть разметочная сетка, на выходной форме она не будет видна. Обычно устанавливаемые элементы управления привязывают к верхнему левому углу ячейки сетки.

Поверхность формы может быть контейнером для вывода на нее картинки. При этом картинку можно выводить в виде обоев, то есть повторяющейся непрерывной картинки. Выбирают ее с помощью свойства **Picture**, при нажатии на которое в правой части визуализируется кнопка с многоточием. При нажатии на нее выводится обычное диалоговое окно для выбора картинки.

В одном приложении может быть несколько форм. Они открываются и закрываются командами, которые необходимо написать.

Рассмотрим, как создать приложения (создаются две формы). И на первой, и на второй будет по одной командной кнопке **CommandButton**, этого вполне достаточно. Кнопка на форме **UserForm1** будет открывать **UserForm2**, а кнопка на **UserForm2** — закрывать форму **UserForm2**. Форма **UserForm1** в нашем приложении будет основной, а форма **UserForm2** — вспомогательной. Поэтому совершенно естественно, что управление открытием формы **UserForm2** будет находиться на **UserForm1**. Закрыть **UserForm2** можно как с первой, так и со второй формы. Но так как необходимо задействовать и форму **UserForm2**, мы будем управлять закрытием этой формы с нее же.

Обработчик щелчка кнопки, расположенной на форме **UserForm1**, будет такой:

```
Private Sub CommandButton1_Click()
UserForm2.Show
End Sub
```

Обработчик щелчка кнопки на форме UserForm2 будет такой:

```
Private Sub CommandButton1_Click()  
Unload UserForm2  
End Sub
```

Рассмотрим эти две процедуры.

В первой форма **UserForm2** открывается методом **Show**. Существует несколько способов открыть форму. Но в нашем случае наиболее подходящий — с помощью метода **Show**. Имя загружаемой формы указано в явном виде.

При открытии методом **Show** первая форма будет недоступна до тех пор, пока не закроем **UserForm2**. Поэтому мы поступили логично, управляя закрытием формы **UserForm2** с нее же.

Во второй процедуре используется команда **Unload**, выгружающая форму из памяти. Есть два способа указать выгружаемую форму: указать ее имя явно или вместо имени — слово **Me** (то есть Я или Меня). Так как управление выгрузкой происходит с формы **UserForm2**, можно использовать оба варианта. При выгрузке **UserForm2** с формы **UserForm1** нужно явно указать имя **UserForm2**. Если, пытаясь выгрузить форму **UserForm2** (при другом методе, не **Show**), использовать **Me**, вместо **UserForm2** будет выгружена форма **UserForm1**.

Команда **Unload** не является обязательной: закрыть форму можно щелчком по кнопке **Закрыть** на системной полосе. Но чтобы приложение выглядело доработанным, конечно, необходимо предусматривать закрытие с помощью специальных кнопок или щелчков, например, по форме.

В некоторых программах, которые я видел у своих знакомых, программирующих на VBA, выгрузка формы происходит не щелчком по кнопке на форме, а щелчком по самой форме, то есть они пишут следующий код:

```
Private Sub UserForm_Click()  
Unload UserForm2  
End Sub
```

В этом случае при щелчке по форме она закрывается. Несмотря на относительную простоту этой части приложения (нет лишней кнопки на форме), этот вариант не очень удобен: любой щелчок (например, случайный мимо кнопки, которая должна выполнять какой-нибудь расчет, выборку или сортировку) приведет к закрытию формы и, скорее всего, к потере произведенных настроек.

Кроме метода **Show** имеется оператор **Load**. Но он не отображает форму визуально, а только загружает ее в память. Для визуализации формы все равно нужен метод **Show**. Например:

```
Private Sub UserForm_Initialize()
    Load UserForm2
    UserForm2.Show
End Sub
```

Эта процедура записана для формы **UserForm1**. При загрузке **UserForm1** она не видна, видна форма **UserForm2**. Только после удаления **UserForm2** становится видна форма **UserForm1**.

Как правило, при загрузке формы записывают и оператор **Load** для загрузки ее в память, и метод **Show** для визуализации. Это необходимо, чтобы прорисовка формы происходила быстрее.

Рассмотрим метод **Hide**, который скрывает объекты, в данном случае форму. Метод скрывает форму от визуализации, но не удаляет ее из памяти. В следующем примере мы будем закрывать форму **UserForm2** щелчком по ней. Щелкните дважды по форме **UserForm2** для создания первоначального программного кода и впишите текст, чтобы процедура выглядела следующим образом:

```
Private Sub UserForm_Click()
    UserForm2.Hide
End Sub
```

Обратите внимание, форма скрывается гораздо быстрее, чем закрывается: ее не нужно выгружать из памяти. Различие во времени заметно даже на быстродействующем компьютере,

хотя в этом примере мы пользовались на обеих формах всего одной-двумя кнопками. На сложных формах время будет существенно отличаться в зависимости от того, какой метод вы выбираете, **Hide** или **Unload**.

Вместо записи:

```
UserForm2.Hide
```

можно использовать запись:

```
Me. Hide
```

Она закрывает текущую (активную) форму.

Очень часто метод **Hide** используется при запуске процедуры, после которой форма временно не нужна. Например:

```
Private Sub CommandButton2_Click()
Worksheets("Лист1").Range("A17").Select
Selection.Formula = "=Sum((A1):(A15))"
MsgBox "Расчет произведен!"
Me. Hide
End Sub
```

В этом примере производится расчет суммы ячеек с A1 по A15, результат заносится всегда в ячейку A17, после чего сообщается, что расчет произведен. Форма становится невидимой, но из памяти не выгружается. Чтобы открыть невидимую форму, можно использовать метод **Show**, при этом время на визуализацию формы, загруженной в память, минимально.

Основные элементы управления

На панели **Toolbox** находятся 15 элементов управления, которые загружаются при создании нового приложения:

- **Label** — надпись или метка;
- **TextBox** — текстовое поле;
- **ComboBox** — раскрывающийся список;
- **ListBox** — линейный список;
- **CheckBox** — индикатор с флажком;
- **OptionButton** — переключатель;

- **ToggleButton** — кнопка с эффектом нажатого состояния;
- **Frame** — рамка;
- **CommandButton** — командная кнопка;
- **TabStrip** — страница с закладкой;
- **MultiPage** — многостраничное окно;
- **ScrollBar** — полоса прокрутки;
- **SpinButton** — счетчик;
- **Image** — изображение;
- **RefEdit** — поле интервала.

Label — надпись или метка

Элемент управления **Label** предназначен для вывода текста, как символьного, так и числового. Текст может быть задан заранее и оставаться неизменным, а может меняться в зависимости от того, какая команда была выполнена. Изменение текста с клавиатуры и другими ручными способами не допускается.

Текст задается в свойстве **Caption**. Первоначально он задан так же, как и имя, — **Label1**. Порядковый номер элемента управления указывается сзади него автоматически. Следовательно, если мы возьмем одну метку **Label**, она получает имя **Label1**, если вторую — **Label2**, и т. д. Если вы не хотите, чтобы указанный текст показывался в метке, сотрите его либо впишите туда свой. После компиляции он отразится в метке. Все параметры шрифта находятся в свойстве **Font**. Чтобы открыть его, надо щелкнуть на значке «...» слева от этого свойства. Если свойство **Font** не выделено, то и значок «...» слева от этого свойства не виден. Значок показывает, что после нажатия на него будет открыто какое-то диалоговое окно. После нажатия на этот значок откроется знакомое нам диалоговое окно **Шрифт**, в котором можно выбрать **Шрифт**, **Размер**, **Начертание**, а также **Видоизменение** шрифта — **Подчеркнутый** или **Зачеркнутый**. Здесь же можно просмотреть образец выбранного. Нажмем на кнопку **OK** — загрузятся сделанные изменения, и закроется окно **Шрифт**.

Свойство **Autosize** определяет авторазмер. Изначально установлено в **False**, что запрещает устанавливать авторазмер. Если надпись в метке слишком длинная, она продолжается на следующих строках. При установке свойства **Autosize** в **True** ширина

элемента управления **Label** подстраивается под ширину текста надписи.

Первоначальное имя элемента управления задается в свойстве **(Name)**. Если посмотреть на это имя сразу после установки элемента управления на форму, это будет **Label1**. Имя можно оставлять заданное, можно менять. При изменении используются как английские, так и русские буквы, а также цифры и символ подчеркивания **(_)**. Другие символы задавать нельзя. Если случайно задать другие недопустимые символы, появится сообщение об ошибке и недопустимый символ будет удален из имени автоматически.

Свойство **WordWrap** определяет допустимость переноса продолжения строки. Чтобы иметь такую возможность, установите свойство **Autosize** в **False**, а свойство **WordWrap** — в **True**. Здесь нужны подробности. Если установим **Autosize** в **True**, а **WordWrap** — в **False**, надпись будет продолжаться вправо, даже если ширина элемента управления меньше этой надписи. Если установим **Autosize** в **True** и **WordWrap** — в **True**, длинная надпись не будет увеличиваться вправо, а будет переноситься на другие строки при достижении ширины элемента управления. То есть свойство **WordWrap** как бы затеняет свойство **Autosize**. Автоширина хотя и включена, но не работает. Если же **Autosize** установлена в **False**, независимо от значения свойства **WordWrap** ширина элемента управления не изменяется, а не умещающаяся часть надписи обрезается. Например, нужно вывести надпись «Сарстройтранссарай». Но по ширине умещаются только символы «Сарстройтранс». Следовательно, эти символы и будут выведены в надписи, а *сарай* обрежется. Поэтому при работе с элементом управления **Label** необходимо следить за свойством **Autosize**.

Свойство **BorderStyle** определяет наличие рамки вокруг надписи: выберете значение **fmBorderStyleNone** — рамки не будет, значение **fmBorderStyleSingle** — вокруг надписи появится рамка. Цвет рамки определяет **BorderColor**. При выделении этого свойства в левой части появляется маленькая кнопка с треугольным значком. Нажав на нее, увидим список цветовых констант. Если ни один из вариантов вас не устраивает, есть другие способы управления цветом, например функция **RGB**.

Цвет текста определяется свойством **ForeColor**. Цвет фона — свойством **BackColor**. Выбираем цветовое решение так же, как и в **BorderColor**.

Выравнивание текста надписи управляется свойством **TextAlign**. По умолчанию выбрано значение **fmTextAlignLeft**, то есть выравнивание по левому краю. Другие значения этого свойства: **fmTextAlignCenter** — выравнивание по центру и **fmTextAlignRight** — выравнивание по правому краю.

Свойство **SpecialEffect** определяет внешний вид элемента управления **Label**:

- **fmSpecialEffectFlat** — специального эффекта нет;
- **fmSpecialEffectRaised** — элемент управления приподнят над формой;
- **fmSpecialEffectSunken** — элемент управления вдавлен в форму;
- **fmSpecialEffectEtched** — рамка вокруг элемента управления утоплена в форму;
- **fmSpecialEffectBump** — рамка вокруг элемента управления приподнята над формой.

Свойство **Enabled** определяет доступность элемента управления. При значении **True** он доступен, а при **False** — недоступен.

Свойство **Visible** определяет видимость элемента управления. При значении **True** элемент управления виден на форме или на другой панели, а при значении **False** — невидим.

Свойства **Width** и **Height** определяют ширину и высоту элемента управления в пикселях. Эти свойства можно менять как вручную, задавая соответствующие значения в полях, так и с помощью белых маркеров, окружающих элемент управления при его выделении.

Свойство **Picture** определяет рисунок (точнее, значок-пиктограмму) на элементе управления **Label**. Если выделить это свойство, в левой части появляется маленькая кнопка с многоточием «...». Нажав на нее, откроем диалоговое окно **Load Picture**, с помощью которого можно выбрать изображение в надписи (если, конечно, у вас есть изображения такого размера).

Свойство **PicturePosition** определяет порядок выравнивания пиктограммы в надписи и текста надписи:

- **fmPicturePositionLeftTop** — изображение слева, текст надписи справа в верхнем углу;
- **fmPicturePositionLeftCenter** — изображение слева, текст надписи справа и по центру;
- **fmPicturePositionLeftBottom** — изображение слева, текст надписи справа в нижнем углу;
- **fmPicturePositionRightTop** — изображение справа, текст надписи слева в верхнем углу;
- **fmPicturePositionRightCenter** — изображение справа, текст надписи слева и посередине;
- **fmPicturePositionRightBottom** — изображение справа, текст надписи слева в нижнем углу;
- **fmPicturePositionAboveTop** — изображение вверху, текст надписи под изображением справа;
- **fmPicturePositionAboveCenter** — изображение вверху, текст надписи под изображением в центре;
- **fmPicturePositionAboveBottom** — изображение вверху, текст надписи под изображением в нижней части;
- **fmPicturePositionBelowTop** — изображение внизу, текст надписи над изображением в верхней части;

- **fmPicturePositionBelowCenter** — изображение внизу, текст надписи над изображением в центре;
- **fmPicturePositionBelowBottom** — изображение внизу, текст надписи над изображением в нижней части;
- **fmPicturePositionCenter** — и текст надписи, и изображение посередине, перекрывая друг друга.

Свойство **ControlTipText** определяет надпись (подсказку, хint), которая будет появляться при подводе указателя мыши к данному элементу управления.

Свойство **MousePointer** определяет вид указателя мыши при подводе к данному элементу управления:

- **fmMousePointerDefault** — по умолчанию;
- **fmMousePointerArrow** — стрелка;
- **fmMousePointerCross** — тонкий черный крестик;
- **fmMousePointerIBeam** — I-образный указатель;
- **fmMousePointerSizeNESW** — двунаправленная стрелка, из нижнего левого угла в верхний правый угол;
- **fmMousePointerSizeNS** — двунаправленная стрелка, снизу вверх;
- **fmMousePointerSizeNWSE** — двунаправленная стрелка, из верхнего левого угла в нижний правый угол;
- **fmMousePointerSizeWE** — двунаправленная стрелка, справа налево;
- **fmMousePointerUpArrow** — тонкая черная стрелка, направленная снизу вверх;
- **fmMousePointerHourGlass** — песочные часы;

- **fmMousePointerNoDrop** — черный круг, перечеркнутый из верхнего левого угла в нижний правый угол;
- **fmMousePointerAppStarting** — белая стрелка с песочными часами;
- **fmMousePointerHelp** — белая стрелка с черным вопросительным знаком;
- **fmMousePointerSizeAll** — черная крестообразная стрелка;
- **fmMousePointerCustom** — пользовательский указатель.

Свойство **MouseIcon** позволяет выбрать значок иконки, который будет появляться при подведении указателя мыши к элементу управления и вместо указателя мыши, если в свойстве **MousePointer** выбрано значение **fmMousePointerCustom**. То есть вместо указателя мыши по умолчанию будет появляться пользовательский указатель.

Свойство **TabStop** определяет отключение элемента управления из обхода клавишей Tab. Если это свойство установлено в **True**, элемент управления будет выделен при нажатии на клавишу Tab в порядке очередности, установленном в списке элементов управления в диалоговом окне **Tab Order**. Если **TabStop** установлен в **False**, этот элемент управления отключается от обхода клавишей Tab. Для элементов управления **Label** это актуально: этот элемент управления предназначен только для вывода текста, ему не нужно передавать фокус, поэтому ему по умолчанию устанавливают **TabStop** в **False**.

Этот элемент управления мы рассмотрели первым, поэтому пришлось подробно остановиться на свойствах. Многие из них будут встречаться в других элементах управления. Там, где они будут повторяться, рассматривать их заново не станем: можете просмотреть их назначение и возможные значения в этом элементе управления.

В элементе управления **Label** выводится строковая информация, то есть с типом данных **String**.

TextBox — текстовое поле

Элемент управления TextBox предназначен для ввода и вывода на него как одностороннего, так и многострочного текста самого разного характера — строк, чисел, символов.

В текстовом поле, как это обычно делается в среде Windows, можно также выделять текст теми же способами, что и в других программах, с помощью мыши или клавиатуры. С помощью клавиатуры это можно сделать, например, нажав на клавишу Shift и удерживая ее, а затем нажимая клавиши ⇧, ⇩, ⇪, ⇫.

У внешнего оформления BorderStyle два подсвойства: если выбрано значение fmBorderStyleNone, рамки вокруг текстового поля нет; если fmBorderStyleSingle — вокруг текстового поля появляется рамка. Для вывода какой-либо первоначальной информации есть свойство Text. Принцип его действия аналогичен свойству Caption элемента управления Label. Есть свойство Value, оно дублирует текст свойства Text и по идеи должно хранить число. Но так как в VBA имеется тип данных Variant, тип данных определяется автоматически как наиболее подходящий.

Многие свойства имеют такое же назначение и такие же значения, что и элемент управления Label. Например, BackColor определяет цвет фона текстового поля, но появилось и новое свойство BackStyle, которое определяет прозрачность цвета фона. Свойство BackStyle может принимать значения:

- fmBackStyleTransparent — цвет фона игнорируется и принимается цвет фона родителя (Parent), на котором находится данный элемент управления. Если этот элемент управления находится на форме, принимает цвет формы, так как форма в данном случае и есть родитель;
- fmBackStyleOpaque — принимается цвет фона, заданный свойством BackColor.

Свойство MultiLine задает многострочность в элементе управления. Если выбрано значение False, разрешается только одна строка. Если выбрано True, строк может быть больше.

В многострочном поле для перехода на новую строку можно использовать клавишу Enter. Но следует помнить, что для некоторой кнопки, возможно, установлено свойство **Default = True**. Поэтому нажатие клавиши Enter вызовет срабатывание этой кнопки. Тогда для перехода на новую строку надежнее использовать Ctrl + Enter или Shift + Enter. Если одна из кнопок имеет свойство **Default = True**, а свойство **MultiLine** в элементе управления **TextBox** равно **False**, при нажатии Ctrl + Enter или Shift + Enter будет немедленно, без всяких предупреждений, выполнено действие, предусмотренное при нажатии на кнопку.

Есть свойство **EnterKeyBehavior**, в котором при установленном **True** переход на новую строку осуществляется простым нажатием на Enter. Каждое нажатие на клавишу Enter или на Ctrl + Enter или Shift + Enter переводит курсор на новую строку до тех пор, пока вы явно не укажете на необходимость выхода за пределы элемента управления, например щелчком по форме или любому другому элементу управления. Приведу пример. Допустим, есть элемент управления **TextBox**, в котором свойства **MultiLine** и **EnterKeyBehavior** установлены в **True**, и кнопка, где установлено свойство **Default = True**. При прочих равных условиях нажатие на Enter запустит действие, заложенное в этой кнопке. Если мы находимся в элементе управления **TextBox**, нажатие на Enter добавит новую строку. Чтобы нажать на кнопку, в которой установлено свойство **Default = True**, нужно или непосредственно нажать на эту кнопку, или щелкнуть по форме или любому другому элементу управления (кроме элемента управления **TextBox**). После этого каждое нажатие на Enter будет запускать действия, заложенные в кнопку, где установлено свойство **Default = True**. Кнопки будем изучать немного позднее, там и познакомимся ближе с понятием **Default = True**. Кнопка с таким свойством и значением может быть выбрана только одна.

Независимо от того, какое значение выбрано в свойстве **MultiLine**, **ScrollBars** определяет наличие линеек прокрутки в окне элемента управления **TextBox**. Необходимо помнить, что линейки прокрутки появляются только в том случае, если текстовая строка не умещается в том направлении, в котором вы выбрали линейку. Здесь можно выбрать одно из следующих значений:

- **fmScrollBarsNone** — линейки прокрутки отсутствуют;
- **fmScrollBarsHorizontal** — имеется горизонтальная линейка прокрутки;
- **fmScrollBarsVertical** — имеется вертикальная линейка прокрутки;
- **fmScrollBarsBoth** — имеются и горизонтальная, и вертикальная линейки прокрутки.

Еще раз напоминаю, что заданные линейки автоматически появляются только при необходимости. Если линейки прокрутки не нужны, то есть поле не заполнено по ширине или по высоте, линейки не видны.

Элемент управления **TextBox** — один из самых распространенных. Практически в каждом приложении приходится указывать программе номер листа, имя книги, начальную или конечную ячейку и т. д.

Для нашего примера перенесите на форму две кнопки **CommandButton** и два текстовых поля **TextBox**. Назначение примера следующее: при нажатии на кнопку **CommandButton1** будет открыт Лист3, а сам Лист3 переместится в списке листов на первое место. При нажатии на кнопку **CommandButton2** в текстовом поле **TextBox1** будет выдано число листов в книге (включая диаграммы на отдельных листах и т. д.), во втором текстовом поле **TextBox2** будет выдано имя первого листа в списке листов. Программный код для первой кнопки будет такой:

```
Private Sub CommandButton1_Click()
Sheets("Лист3").Select
Sheets("Лист3").Move Before:=Sheets(1)
End Sub
```

Программный код для второй кнопки будет такой:

```
Private Sub CommandButton2_Click()
TextBox1.Text = ActiveWorkbook.Sheets.Count
TextBox2.Text = ActiveWorkbook.Sheets(1).Name
End Sub
```

ComboBox — раскрывающийся список

Раскрывающийся список **ComboBox** — это комбинированный список, комбинация двух элементов управления: списка со значениями и текстового поля. Поля со списком используются в том случае, если заранее трудно определить значения, которые нужно включить в список, или список содержит слишком много элементов. В таком списке нужное значение можно не только выбирать, но и вводить непосредственно в поле ввода. Новое значение после ввода автоматически помещается в список.

Раскрывающийся список **ComboBox** более компактен, чем похожий на него линейный список **ListBox**.

В списке **ComboBox** разрешается выбрать только одно значение из этого списка. Множественный выбор, то есть одновременный выбор сразу нескольких значений, в списке **ComboBox** не разрешен.

Свойство **Text** содержит текст последнего выбранного элемента раскрывающегося списка.

Свойство **ListRows** задает число строк, которые видны в раскрывающемся списке **ComboBox** без прокрутки. Если в списке меньше **ListRows** строк, полоса прокрутки не устанавливается, так как она не нужна.

Свойство **ColumnCount** отображает число столбцов в элементе управления **ComboBox**. По умолчанию в раскрывающемся списке один столбец, но можно сделать и несколько.

Свойство **DropButtonStyle** определяет внешний вид кнопки, расположенной в раскрывающемся списке. Здесь можно выбрать следующие значения:

- **fmDropButtonStylePlain** — нет никакого значка;
- **fmDropButtonStyleArrow** — треугольный значок, установлен по умолчанию;
- **fmDropButtonStyleEllipsis** — знак многоточия;

- **fmDropButtonStyleReduce** — символ подчеркивания (_).

Свойство **SpecialEffect** определяет внешний вид элемента управления **ComboBox**:

- **fmSpecialEffectFlat** — специального эффекта нет;
- **fmSpecialEffectRaised** — элемент управления приподнят над формой;
- **fmSpecialEffectSunken** — элемент управления вдавлен в форму;
- **fmSpecialEffectEtched** — рамка вокруг элемента управления утоплена в форму;
- **fmSpecialEffectBump** — рамка вокруг элемента управления приподнята над формой.

Свойство **ListStyle** определяет стиль выбора записей. Вы можете выбрать один из двух стилей представления для **ComboBox**. Если **fmListStylePlain**, каждый пункт находится в отдельной строке; пользователь выбирает пункт, выделяя одну строку. Если стиль **fmListStyleOption**, кнопка выбора или индикатор с флагжком появляется в начале каждой строки. С этим стилем пользователь выбирает пункт, щелкнув кнопку выбора или индикатора с флагжком.

ListBox — линейный список

Линейный список **ListBox** позволяет выбирать из списка один или несколько элементов. Можно добавлять новые элементы или удалять существующие. Если не все могут одновременно отобразиться в поле линейного списка, в нем автоматически появляются полосы прокрутки.

Свойство **Multiselect** — это возможность множественного выбора значений в линейном списке **ListBox**. Если множественный выбор запрещен, можно выбрать в списке только одно значение, если разрешен — одновременно выбрать несколько значений. Максимальное число выбранных значений при этом ограничено только их числом в списке.

Свойство **Multiselect** может принимать следующие значения:

- **fmMultiselectSingle** — множественный выбор невозможен. Щелчком мыши или нажатием клавиши пробела в списке можно выбрать только один элемент;
- **fmMultiselectMulti** — простой множественный выбор. Элементы списка выбираются щелчком мыши или нажатием клавиши пробела.
- **fmMultiselectExtended** — расширенный множественный выбор. Можно выбрать несколько элементов с помощью мыши или клавиш управления курсором с использованием клавиш Shift и Ctrl. При использовании мыши без клавиатуры можно выделить только один элемент в списке.

При множественном выборе свойство **Text** содержит текст последнего выбранного элемента линейного списка.

Свойство **ListStyle** определяет стиль выбора записей. Можно выбрать один из двух стилей представления для **ListBox**. Каждый обеспечивает различные пути, чтобы выбирать пункты в списке. Если **fmListStylePlain**, каждый пункт находится в отдельной строке; выбираем пункт, выделяя одну или более строк. Если **fmListStyleOption**, кнопка выбора или индикатор с флагшком появляются в начале каждой строки. С этим стилем выбираем пункт, щелкнув кнопку выбора или индикатора с флагшком. Индикаторы с флагшками появляются при условии, что свойство **MultiSelect** установлено в **True**.

Сочетание свойств **ListStyle** и **MultiSelect** дает различные комбинации представления строк на элементе управления **ListBox**.

Для **ListBox** значение свойства **Text** должно соответствовать существующему значению в списке.

Кроме свойств, которые находятся в окне **Properties**, существует еще несколько, которых в этом окне нет. Они задаются программным путем. Рассмотрим их и создадим примеры с использованием этих свойств.

Текущее количество элементов в списке сохраняется в свойстве **ListCount**.

Свойство **ListCount** только для чтения. **ListCount** — количество колонок, которые вы можете переместить. **ListRows** — максимум, чтобы отображаться сразу. **ListCount** всегда на единицу больше, чем самая большая величина для свойства **RowIndex**, поскольку индексные номера начинаются с 0 и счет пунктов начинается с 1. Если ни один пункт не выбран, **ListCount** — 0 и **RowIndex** — 1.

Свойство **Selected** возвращает или устанавливает состояние выбора пунктов в **ListBox**. Синтаксис этого свойства следующий:

object.Selected(index) [= Boolean]

Где:

object — обязательный аргумент, возвращаемый объект;

index — обязательный аргумент, целое с диапазоном от 0 до числа, на единицу меньше, чем количество пунктов в списке;

Boolean — необязательный аргумент, выбор пункта;

True — если выбор сделан и **False** — если выбор не сделан.

Выданное свойство полезно, когда пользователь может делать множественный выбор. Это свойство можно использовать, чтобы определять выбранные колонки при множественном выборе списка значений. Это свойство можно использовать, чтобы выбирать или отменять выбор колонки в списке. Значение по умолчанию — в текущем состоянии выбора **ListBox**. **Value** или **RowIndex** рекомендованы при единственном выборе списка. В этом случае **RowIndex** возвращает индекс выбранного пункта. При множественном выборе **RowIndex** возвращает индекс строки, содержащейся в пределах прямоугольника фокуса — независимо от того, какое значение действительно выбрано. Если линейный список свойства **MultiSelect** установлен в **None**, только одно значение может быть установлено в свойстве **Selected** в **True**.

Нам потребуется на форме один элемент управления **ListBox**, одно текстовое окно **TextBox**, несколько кнопок

CommandButton. Назначение примера следующее: в **ListBox** создадим список, с которым будем проводить какие-то манипуляции. В случае необходимости — выводить полученное значение в текстовое окно **TextBox**, а кнопками будем инициировать выполнение процедур.

Вначале создадим записи в списке **ListBox1**. Напишите обработчик щелчка первой кнопки:

```
Private Sub CommandButton1_Click()
    ListBox1.AddItem "Иванов Петр Сидорович"
    ListBox1.AddItem "Штириц Максим Исаевич"
    ListBox1.AddItem "Попандопуло Кузьма Егорович"
    ListBox1.AddItem "Голубков Леонид Мавродьевич"
    ListBox1.AddItem "Воробьянинов Аполидек Харонович"
End Sub
```

Конечно, это не лучший способ занести записи в список. Гораздо проще делать это при создании формы, то есть с применением события **Activate**. Но мы еще не изучали события и напишем таким способом, который нам известен.

Используем метод **AddItem**, который добавляет строку в список.

Вторая кнопка будет удалять все записи из списка. Для этого воспользуемся методом **Clear**. Обработчик второй кнопки будет такой:

```
Private Sub CommandButton2_Click()
    ListBox1.Clear
End Sub
```

Первое свойство, которое изучим, — **List(i)**. Оно возвращает текст элемента списка по его индексу, где *i* — номер индекса. Нумерация индексов начинается с 0. Поэтому программный код

```
Private Sub CommandButton3_Click()
    TextBox1.Text = ListBox1.List(3)
End Sub
```

возвращает нам выделенную строку в линейном списке после нажатия третьей кнопки. Возвращается текст в записи с индексом 3 (**List(3)**).

Чтобы узнать номер индекса выделенной строки, нужно для четвертой кнопки написать следующий программный код:

```
Private Sub CommandButton4_Click()
    TextBox1.Text = ListBox1.ListIndex
End Sub
```

В реальном списке находятся десятки и сотни записей, поэтому определение индекса вручную невозможно. Как сказано выше, нумерация индекса начинается с 0. Если пользователю необходимо узнать номер строки, а не индекса, в привычной для нас нумерации, начиная с 1, необходимо переписать программный код так:

```
Private Sub CommandButton5_Click()
    TextBox1.Text = ListBox1.ListIndex + 1
End Sub
```

Если пользователь хочет узнать по номеру строки содержимое этой строки, необходимо перенести на форму текстовое поле **TextBox2**, в котором будем задавать номер строки. Напишите обработчик этой кнопки:

```
Private Sub CommandButton6_Click()
    TextBox1.Text = ListBox1.List(TextBox2.Text + 1)
End Sub
```

Свойство **ListFillRange** возвращает или устанавливает необходимый диапазон рабочего листа, чтобы заполнять список. Microsoft Excel читает содержимое каждой ячейки в диапазоне и включает величины ячейки в список. Он прослеживает изменения в диапазоне ячеек. Если список создан методом **AddItem**, это свойство возвращает пустую строку ("").

Свойство **ColumnCount** отображает число столбцов в элементе управления **ListBox**. По умолчанию в линейном списке один столбец, но можно сделать и несколько. Установите это свойство в окне свойств **Properties** равным 3. В нашем примере теперь будут три столбца. Добавьте новую кнопку на форму. Напишите следующий программный код:

```
Private Sub CommandButton7_Click()
ListBox1.Column(0, 0) = "Иванов"
ListBox1.Column(1, 0) = "Петр"
ListBox1.Column(2, 0) = "Сидорович"
ListBox1.Column(0, 1) = "Штирлиц"
ListBox1.Column(1, 1) = "Максим"
ListBox1.Column(2, 1) = "Исаевич"
ListBox1.Column(0, 2) = "Попандопуло"
ListBox1.Column(1, 2) = "Кузьма"
ListBox1.Column(2, 2) = "Егорович"
ListBox1.Column(0, 3) = "Голубков"
ListBox1.Column(1, 3) = "Леонид"
ListBox1.Column(2, 3) = "Мавродьевич"
ListBox1.Column(0, 4) = " Воробьянинов"
ListBox1.Column(1, 4) = " Аполидевк"
ListBox1.Column(2, 4) = " Харонович"
End Sub
```

В этом примере заполняются данными все три столбца. Столбцов несколько (более одного), но это не означает, что линейный список с данными, заполнив первый столбец, начинает заполнять второй столбец, затем третий и т. д. Все столбцы относятся к одной записи. В этом примере все данные в одной строке относятся к одной записи независимо от того, в каком столбце они находятся. Например, «Иванов» «Петр» «Сидорович» находится в трех столбцах, но в одной строке. Одна строка — одна запись.

Основное событие списка — **Click**. Пользователь с помощью мыши или клавиш курсора выбирает элемент в списке.

CheckBox — индикатор с флажком

Элемент управления **CheckBox** широко используется в создании визуальных приложений. Он предназначен для включения или выключения каких-либо опций.

Начнем с рассмотрения свойства **TripleState**. Самостоятельного значения оно не имеет, всего лишь определяет, сколько опций может быть у другого свойства этого элемента управления, **Value** — три или два. Если **TripleState = False**, как это установлено изначально, **Value** может принимать только два

значения (есть флажок, нет флажка), если **TripleState** = **True**, то три (есть черный флажок, есть серый флажок, нет флажка).

Рассмотрим эти значения **Value**. Но прежде уясним: оно определяет графический значок в элементе управления **CheckBox**: черная галочка, серая галочка и пустое окно индикатора. Таким образом, индикатор может использоваться как выбор между двумя или тремя вариантами.

Максимально можно выбрать три значения флажка, но в свойстве **Value** можно выбрать только **False** (0) или **True** (1). Третье положение (серый недоступный флажок) по умолчанию не предусмотрено. Любое отличное от 0 значение трактуется как **True** (1). Причем любое отличное от 0 значение понимается буквально: все равно, задаете вы 1 или 10, 0,1 или 10 — все это трактуется как **True** (1).

Третье положение (серый недоступный флажок) можно установить только программно. В этом случае положение меняется. Заданными значениями являются только 0 и 1, а остальные трактуются уже как третье положение флажка (серый недоступный флажок), поэтому программно можно записать как

```
CheckBox1.Value = 2
```

так и:

```
CheckBox1.Value = -2
```

Щелчок по элементу управления **CheckBox** ничем не отличается от щелчка по кнопке. Если для этого щелчка будет написан программный код, он будет выполнен. Щелчок можно делать по всей площади элемента управления, а не только по окну с флажком.

Надпись на элементе управления задается в свойстве **Caption**.

Свойство **Locked** определяет возможность изменить флажок в индикаторе. Если **Locked** установлено в **False**, флажок можно изменять. Если свойство **Locked** установлено в **True**, положение флажка изменять нельзя.

Свойство **Alignment** имеет два положения:

- **fmAlignmentLeft** — надпись на элементе управления находится слева, а окно для флажка справа;
- **fmAlignmentRight** — надпись на элементе управления находится справа, а окно для флажка слева.

По умолчанию выбрано значение **fmAlignmentRight**.

Свойство **SpecialEffect** задает внешний вид окна для флажка. Здесь могут задаваться следующие значения:

- **fmSpecialEffectFlat** — окно плоское с рамкой;
- **fmSpecialEffectSunken** — окно объемное, углубленное в форму без окантовки рамкой.

Элемент управления **CheckBox** может быть размещен на форме как один, так и несколько, так как положение флагков в одном элементе управления не зависит от состояния флажка в другом таком же элементе управления.

Выберем для примера следующие элементы управления: одну метку **Label**, одну кнопку **CommandButton**, один индикатор с флагком **CheckBox**. Если в индикаторе установлен флагок, в надписи **Label1** выдается одна надпись, а если флагка нет — другая. Программный код будет следующий:

```
Private Sub CommandButton6_Click()
If CheckBox1.Value = True Then
    Label1.Caption = "Выбрать ячейки A1: F10"
Else
    Label1.Caption = "Выбрать ячейки D1: J10"
End If
End Sub
```

Выше говорилось, что элемент управления **CheckBox** можно использовать как кнопку, при щелчке по которой происходят какие-то действия. Сотрите написанный код и удалите с формы кнопку **CommandButton1**. Щелкните дважды по элементу

управления **CheckBox1**, чтобы создать начальный программный код, напишите программный код:

```
Private Sub CheckBox1_Click()
If CheckBox1.Value = True Then
    Label1.Caption = " Выбрать ячейки A1: F10"
Else
    Label1.Caption = " Выбрать ячейки D1: J10"
End If
End Sub
```

Эти две программы работают одинаково, поэтому для **CheckBox** не обязательно использовать другие элементы управления, так как индикатор с флагком сам может выполнять какие-то процедуры.

OptionButton — переключатель

Элемент управления **OptionButton** используется только в наборе, так как использование одного такого элемента управления в приложении бессмысленно.

Свойства **OptionButton** очень похожи на свойства элемента управления **CheckBox**. Наличие флажка также определяется в **Value**. Флажок в **OptionButton** может принимать только два положения: есть флажок / нет флажка. Все элементы управления **OptionButton** на родителе (Parent), например на рамке **Frame**, автоматически объединяются в группу, в которой допустим только один флажок в группе или ни одного при загрузке, если свойство **Value** у всех элементов управления в группе **OptionButton** равно **False**.

Чтобы разделить элементы управления **OptionButton** на группы, в которых переключение флагков происходило бы независимо от положения флагка в другой группе, необходимо каждую группу переключателей **OptionButton** выделить специальным образом. Для этого предназначено свойство **GroupName**. По умолчанию оно не задано, следовательно, все переключатели объединяются в одну группу, в которой есть одно пустое имя «». Предположим, у нас на форме шесть переключателей. Их нужно разделить на две группы. В первой четыре переключателя, во второй — два. Для четырех переключателей в свойстве **GroupName** нужно задать любое

имя, английскими или русскими буквами, а во второй группе — другое имя. Задавать имя группы во всех переключателях без ошибок довольно трудно, а один ошибочный символ создаст другую группу. Поэтому имя группы нужно вводить без ошибок. Для этого выделим всю группу переключателей следующим образом. Если они расположены смежно (друг за другом), выделение каждого элемента управления производим с нажатием на Shift и, не отпуская клавишу, щелкаем указателем мыши по первому переключателю, а затем по последнему. При этом в группу будут включены все элементы управления, находящиеся между первым и последним элементом управления.

Если в группу необходимо включить несмежные элементы управления, создаем выделенную группу с нажатием на клавишу Ctrl и, не отпуская, на каждый переключатель, включаемый в группу. После этого можно задать единое имя группы переключателей: всего один раз заполнить свойство **GroupName** в группе. Затем разъединить выделенную группу щелчком по любому месту формы или по любому другому элементу управления.

Другой способ создать группу переключателей — расположить их на разных панелях, при этом нужно помнить, что форма также считается панелью. Следовательно, наш пример можно выполнить так: четыре переключателя оставить на форме, а вторую группу расположить на панели **Frame**. Частный случай последнего решения — расположить обе группы переключателей на двух панелях **Frame**.

Создадим следующий пример: при щелчке по первому переключателю в индикаторе **CheckBox** флажок будет сниматься, а при щелчке по второму переключателю в индикаторе **CheckBox** — устанавливаться. Для этого перенесите на форму два элемента управления **OptionButton** и один **CheckBox**.

Обработчик первого переключателя будет такой:

```
Private Sub OptionButton1_Click()
    CheckBox1.Value = 0
End Sub
```

Обработчик второго переключателя будет такой:

```
Private Sub OptionButton2_Click()
CheckBox1.Value = 1
End Sub
```

Рассмотрим следующий пример. В нем в зависимости от того, какое положение переключателя выбрано, в надпись **Label1** попадает имя (Name) одного из элементов управления. Для этого нам понадобится одна метка (надпись) **Label**, два переключателя **OptionButton**, одна кнопка **CommandButton**, один элемент управления **MultiPage**. По умолчанию в **MultiPage** две страницы. Добавьте еще две. Для этого в верхнем правом месте элемента управления найдите свободный участок, закрашенный белым цветом, и щелкните в нем правой клавишей мыши. В появившемся контекстном меню выберите команду **New Page** (добавить новую страницу). Напишите следующий программный код для кнопки **CommandButton1**:

```
Private Sub CommandButton5_Click()
Dim MyProba As Object
Dim ControlsIndex As Integer
If OptionButton1.Value = True Then
Set MyProba = Controls.Item(ControlsIndex)
Label1.Caption = MyProba.Name
ControlsIndex = ControlsIndex + 1
If ControlsIndex >= Controls.Count Then
ControlsIndex = 0
End If
ElseIf OptionButton2.Value = True Then
Set MyProba = MultiPage1.Pages _
.Item(MultiPage1.Value)
Label1.Caption = MyProba.Name
End If
End Sub
```

Если флажок установлен в первом переключателе, в метке **Label1** выводится имя метки **Label1**, если во втором, в метке **Label1** выводится имя открытой страницы элемента управления **MultiPage1**. Поскольку в элементе управления четыре страницы, листая их, можно заметить, что выводится имя страницы, а не имя всего элемента управления **MultiPage1**.

ToggleButton — кнопка с эффектом нажатого состояния

Внешне (да и названием) кнопки **ToggleButton** напоминают командные кнопки **CommandButton**, но в основном они используются для другого — показывают состояние выбора пункта. Используйте **ToggleButton**, чтобы показывать выбор пункта. Если **ToggleButton** — обязательный источник данных, то показывает текущее значение этого источника данных как Yes/No, True/False, On/Off или некоторый другой выбор двух установочных параметров. Если пользователь выбирает **ToggleButton**, текущая установка — Да, Истина; если не выбирает **ToggleButton**, установка — No, Ложь. Если **ToggleButton** не затемнена (то есть доступна), то положение этой кнопки готово к исполнению, а пользователь может управлять кнопкой: нажимать или отжимать ее. Если **ToggleButton** показывает нажатие/ненажатие, но затемнена, то пользователю не разрешено изменять интерфейс, то есть управлять положением кнопки. Недоступность (затемненность) элементов управления создается для того, чтобы пользователь случайно не нанес себе вред, нажав на такие элементы управления. Если такие элементы управления все-таки нужны, то необходимо предусмотреть разблокировку недоступных элементов управления.

Можно также использовать **ToggleButton** в рамке (Frame), чтобы выбирать один или несколько вариантов нажатий из группы связанных кнопок. Например, можно создать форму со списком доступных пунктов с **ToggleButton**. Пользователь может выбирать конкретный подходящий пункт **ToggleButton**. Чуть выше приводился пример с доступной (незатемненной) кнопкой и затемненной (недоступной). Если на форме созданы недоступные кнопки с эффектом нажатия, то необходимо предусмотреть возможность разрешить изменение положения кнопки **ToggleButton**. Например, в рамке (Frame) можно предложить кнопку **ToggleButton**, которая включает и выключает доступность остальных элементов управления на рамке (Frame).

Чтобы была более понятна работа этой кнопки, рассмотрим это на конкретном примере: в приложении MS Word такой кнопкой

является кнопка **Отобразить все знаки**. Если кнопка нажата, то непечатаемые символы показываются в документе. Если кнопка не нажата, то непечатаемые символы в документе не видны. Такими же кнопками в MS Word являются **Маркеры**, **Нумерация**, **Многоуровневый список**, **Полужирный**, **Курсив**, **Подчеркнутый** и множество других кнопок.

Рассмотрим пример с рамкой, на которой находятся взаимосвязанные кнопки с эффектом нажатия. В MS Word такими взаимосвязанными кнопками являются **Нумерация** и **Многоуровневый список**. Если, не нажимая на кнопку **Нумерация**, нажать на кнопку **Многоуровневый список**, то будет автоматически нажата кнопка **Нумерация**, потому что эти кнопки связаны между собой, хотя многоуровневый список будет применен.

Встроенным свойством **ToggleButton** является свойство **Value**. Встроенным событием **ToggleButton** является событие **Click**.

По умолчанию свойство **Value** установлено в **False**. Это означает, что при загрузке формы кнопка отжата, то есть на нее не нажали. Если установить свойство **Value** в **True**, кнопка будет выглядеть нажатой.

Свойство **TripleState** показывает, сколько вариантов может принимать положение кнопки — 2 или 3. Если **TripleState** установлено в **False**, кнопка может принимать только два значения: Отжата, Нажата. Если **TripleState** установлено в **True**, то три положения. Третье положение — Полунажата. Если проанализировать эту возможность, становится понятно, что этот элемент управления очень похож на индикатор с флагжком **CheckBox**.

Свойство **Locked** определяет возможность изменения состояния кнопки. Если **Locked** установлена в **False**, кнопка может принимать как нажатое, так и отжатое положение. Если свойство **Locked** установлено в **True**, заданное состояние кнопки фиксируется и изменить его нельзя (можно, но только программным путем).

Разработаем несколько примеров с использованием элемента управления **ToggleButton**.

Перенесите на форму несколько кнопок **ToggleButton** (для примеров нам потребуются три кнопки), одну командную кнопку **CommandButton**, надпись **Label**.

В первом примере при нажатии на командную кнопку **CommandButton1** фокус будет передаваться в **ToggleButton1**, а вторая кнопка **ToggleButton2** будет нажата. Напишите программный код:

```
Private Sub CommandButton1_Click()
    ToggleButton1.SetFocus
    ToggleButton2.Value = True
End Sub
```

Во втором примере в зависимости от состояния **ToggleButton1** будет происходить либо одно, либо другое действие.

```
Private Sub CommandButton2_Click()
    If ToggleButton1.Value = True Then
        Label1.Caption = "Выбрать ячейки A1: F10"
    Else
        Label1.Caption = "Выбрать ячейки D1: J10"
    End If
End Sub
```

Если бы кнопка могла принимать три значения, а не два, логику выражения нужно было бы просто дополнительно усложнить.

В принципе, кнопки **ToggleButton** можно использовать и как обычные командные кнопки. Рассмотрим пример:

```
Private Sub ToggleButton3_Click()
    Label1.Caption = "Наш пример выполнен!"
End Sub
```

Но на практике для выполнения команд подобного рода кнопки **ToggleButton** не используются: **CommandButton** имеет только одно положение — отжатое, так как после нажатия возвращается в обычное, то есть в отжатое состояние. Кнопка **ToggleButton** имеет более чем одно положение и поэтому неудобна для таких операций. В то же время **ToggleButton** и **CheckBox** явно дублируют друг друга во многих отношениях.

Frame — рамка

Frame (Рамка) — это один из элементов контейнеров (или панелей). Его назначение — объединить в группу несколько элементов управления. Объекты, объединенные с помощью рамки, можно перемещать как единое целое, активизировать и деактивизировать, делать видимыми или невидимыми. Некоторые элементы часто сами нуждаются в контейнере. Например, обычно переключатели (OptionButton) в форме объединяются в одну группу. Чтобы создать вторую группу опций, нужно требуемые переключатели объединить в элементе-контейнере.

Для объединения объектов в группу сначала создают контейнер Frame (Рамка), затем добавляют в него нужные элементы управления. Если требуемые элементы уже находятся в форме, их достаточно переместить в элемент-контейнер. Чтобы проверить, действительно ли элемент принадлежит контейнеру Frame (Рамка), достаточно переместить этот контейнер, элемент управления, принадлежащий контейнеру, будет перемещаться вместе с ним. В окне свойств Properties при выделении контейнера Frame (Рамка) выводятся только те элементы управления, которые расположены в этом контейнере, и сам контейнер. При выделении формы выводится список расположенных на форме элементов управления и панелей-контейнеров. Если, например, создать на форме Frame (Рамка), а на нее поместить элементы управления Label (Надпись) и CommandButton (Командная кнопка), при выделении формы в окне свойств Properties будут видны только сама форма (UserForm) и Frame (Рамка). Элементы управления Label (Надпись) и CommandButton (Командная кнопка) при этом не будут выведены в список окна Properties, так как не являются самостоятельными — они принадлежат контейнеру Frame (Рамка). Если создать другую панель Frame (Рамка), элементы управления, расположенные на ней, также будут видны в списке окна Properties только при выделении этого контейнера.

Один контейнер Frame (Рамка) может располагаться не только непосредственно на форме, но и на поверхности другого контейнера Frame (Рамка). В этом случае принцип действия остается тот же самый: Frame2 является собственностью Frame1, а не формы UserForm1. Поэтому при выделении контейнера Frame2 выделяется и его собственник — Frame1.

Нумерация **TabIndex** в контейнере производится самостоятельно и независимо от нумерации **TabIndex**. Если открыть диалоговое окно **TabOrder**, можно увидеть подтверждение этого — нумерация внутри каждого контейнера, независимо от нумерации в других контейнерах и на форме.

Рассмотрим основные свойства элемента управления **Frame** (Рамка).

Свойство **Caption** устанавливает заголовок панели **Frame** (Рамка), заменяя заголовок по умолчанию **FrameN**, где N — номер панели в порядке создания в приложении.

Из других свойств наиболее важными являются **Enabled** (Доступность) и **Visible** (Видимость). Остальные свойства второстепенны и используются для внешнего оформления панели **Frame** (Рамка). Обычно они не изменяются и принимаются по умолчанию, так как чаще всего программист работает только с элементами управления, принадлежащими панели **Frame** (Рамка).

Программирование с участием элементов управления, расположенных на панелях (контейнерах) **Frame** (Рамка), ничем не отличается от программирования с участием элементов управления на форме.

Рассмотрим следующий пример. Перенесите на форму одну рамку **Frame**, на нее — одну кнопку **CommandButton1** и одну надпись **Label1**. На форму перенесите кнопку **CommandButton2** и метку **Label2**. На рамку **Frame1** перенесите вторую рамку и сократите ее размеры так, чтобы она уместилась на первой рамке и не мешала просматривать уже расположенные там элементы управления. Лучше всего это сделать на форме: перенести рамку **Frame2** не сразу на рамку **Frame1**, а сначала на форму, здесь уменьшить размеры этой рамки до нужных, а затем переместить вторую рамку на первую. После установки второй рамки на первой на второй рамке поместите надпись **Label3**. Таким образом, у нас имеется элемент управления **Label** на форме, на первой и на второй рамках. Этот пример позволит понять, что рамка не влияет на работу других элементов управления. Рамка предназначена

всего лишь для внешнего оформления и для группировки элементов управления.

Программный код будет следующий:

```
Private Sub CommandButton2_Click()
Label1.Caption = "Диапазон A1: D10 найден"
Label2.Caption = "Данные диапазона допустимы"
Label3.Caption = "Произведен расчет суммы ячеек A1: D10"
End Sub
```

Как видно из этого примера, принцип программирования не зависит от того, где расположен элемент управления — на форме, на панели или на панели второго или следующего порядка.

CommandButton — командная кнопка

Этот элемент управления используется для того, чтобы начать, прервать или закончить какой-либо процесс. Кнопка встречается во всех приложениях Windows.

Главное событие для кнопки — **Click**. Для непосредственного вызова события **Click** нужно дважды щелкнуть по кнопке — создается заголовок программного кода. Между строк этого кода можно записать свой программный код. Во всех примерах (кроме работы с некоторыми другими элементами управления, например, **CheckBox** или **OptionButton**) мы использовали для инициирования каких-то действий именно кнопку **CommandButton** как самый распространенный элемент управления для вызова события **Click**.

Свойство **Caption** определяет заголовок кнопки. Заголовок можно писать английскими или русскими буквами, цифрами и любыми символами. Заголовок кнопки оформляется шрифтом **Font** и цветом **ForeColor**.

Свойство **Default** определяет, что кнопка активна по умолчанию. Если оно равно **True**, нажатие Enter автоматически генерирует событие **Click** этой кнопки независимо от того, какой элемент имеет фокус. Присваивать значение **True** этому свойству можно только для одной кнопки в форме — нажатие клавиши Enter перехватывается и передается ей. Обычно

кнопка по умолчанию — OK. Свойство **Cancel** похоже на свойство **Default**. Оно обеспечивает перехват клавиши Esc и вызов события **Click** для соответствующей кнопки. Обычно это свойство у кнопок с надписью Cancel (Отмена).

Очень интересно свойство **TakeFocusOnClick**, в котором определяется поведение кнопки при передаче в нее фокуса. Если выбрано значение **True**, кнопка получает фокус и показывает это — будто бы окружена точечной рамкой. Если выбрано **False**, кнопка не получает фокус и показывает это — нет точечной рамки. Свойство **TakeFocusOnClick** определяется только в том случае, если пользователь щелкает по этой кнопке. Если управление передается кнопке клавишей Tab в порядке очередности или прямо методом **SetFocus**, кнопка берет фокус независимо от величины **TakeFocusOnClick**.

Свойство **TabStop** определяет отключение этого элемента управления из обхода клавишей Tab. Если оно установлено в **True**, элемент управления будет выделен при нажатии на Tab в порядке очередности, установленной в списке элементов управления в диалоговом окне **Tab Order**. Если **TabStop** установлен в **False**, этот элемент управления отключается от обхода клавишей Tab.

В кнопках часто приходится использовать свойство **Accelerator**. Оно позволяет задать один символ, который становится подчеркнутым в заголовке кнопки. Это означает, что сочетание Alt с символом, указанным в свойстве **Accelerator**, позволяет нажимать на кнопку не указателем мыши, а с помощью клавиатуры.

Так как на кнопках часто располагаются пиктограммы, актуально свойство **Picture**, которое позволяет их выбирать (если, конечно, у вас есть коллекция таких пиктограмм). **PicturePosition** определяет выравнивание пиктограмм и текста на кнопках. Свойство **PicturePosition** может принимать следующие значения:

- **fmPicturePositionLeftTop** — изображение слева, текст надписи справа в верхнем углу;
- **fmPicturePositionLeftCenter** — изображение слева, текст надписи справа и посередине;

- **fmPicturePositionLeftBottom** — изображение слева, текст надписи справа в нижнем углу;
- **fmPicturePositionRightTop** — изображение справа, текст надписи слева в верхнем углу;
- **fmPicturePositionRightCenter** — изображение справа, текст надписи слева и посередине;
- **fmPicturePositionRightBottom** — изображение справа, текст надписи слева в нижнем углу;
- **fmPicturePositionAboveTop** — изображение вверху, текст надписи внизу в верхнем углу;
- **fmPicturePositionAboveCenter** — изображение вверху, текст надписи внизу в центре;
- **fmPicturePositionAboveBottom** — изображение вверху, текст надписи внизу в нижнем углу;
- **fmPicturePositionBelowTop** — изображение внизу, текст надписи вверху в верхнем углу;
- **fmPicturePositionBelowCenter** — изображение внизу, текст надписи вверху в центре;
- **fmPicturePositionBelowBottom** — изображение внизу, текст надписи вверху в нижнем углу;
- **fmPicturePositionCenter** — и текст надписи, и изображение посередине, перекрывая друг друга.

Размещение пиктограммы обычно не вызывает никаких сложностей, а вот удаление ее с поверхности кнопки может стать проблемой. (Если пиктограммы нет, в свойстве **Picture** находится значение (**None**). Если вы выбрали иконку, значение (**Icon**), если рисунок, значение (**Bitmap**) и т. д.) Для удаления пиктограммы с поверхности кнопки установите указатель мыши сразу после открывающей круглой скобки в значении свойства **Picture** затем нажмите один раз

на Delete. Свойство **Picture** получает значение (**None**), а пиктограмма исчезает.

Составим следующий пример. Перенесите на форму одну кнопку **CommandButton**. В свойстве **Caption** задайте OK. Напишите следующий программный код:

```
Private Sub CommandButton1_Click()
    If CommandButton1.Caption = "OK" Then
        'Это изменяет заголовок
        CommandButton1.Caption = "Cancel"
        CommandButton1.Accelerator = "C"
        'Устанавливает Accelerator в Alt + C
    Else
        CommandButton1.Caption = "OK"
        CommandButton1.Accelerator = "O"
        'Устанавливает Accelerator в Alt + O
    End If
End Sub
```

В этом примере при нажатии на кнопку OK заголовок переименуется в Cancel, а буква *C* станет подчеркнутой. Если у кнопки заголовок Cancel, он переименуется в OK, подчеркнется буква *O*. Это происходит при каждом нажатии на кнопку.

Один из наиболее важных методов элемента управления **CommandButton** — **SetFocus**. Он передает фокус в указанный объект (делает этот элемент активным). Синтаксис **SetFocus** следующий:

object.SetFocus

Где **object** — объект, в который передается фокус. Обязательный аргумент.

Если передать фокус невозможно, он возвращается на предшествующий объект, и генерируется ошибка. Например, фокус невозможно передать элементу управления **Label** (Надпись), так как этот элемент управления только выводит текст и не может стать активным элементом управления. В этом случае будет сгенерирована ошибка.

По умолчанию установка фокуса на управление не активизирует управляющее окно и не устанавливает его поверх других элементов управления. Метод **SetFocus** остается в силе для пустой рамки (Frame), а также для рамки (Frame), которая содержит другие элементы управления. Пустая рамка (Frame) возьмет фокус сама, и любые последующие клавищные события относятся к ней. В рамке (Frame), которая содержит другие элементы управления, фокус перемещается на первое управление в рамке (Frame), и последующие клавищные события относятся к элементу, который получил фокус.

Чтобы показать это на примере, создайте новое приложение. На форму перенесите элемент управления **Frame1** (Рамка). На **Frame** перенесите элементы управления в такой последовательности: надпись **Label1**, затем кнопку **CommandButton1**. После этого на форму перенесите одну кнопку **CommandButton2**. В **CommandButton1** установите свойство **Default** в **True**. Для **CommandButton2** напишите обработчик:

```
Private Sub CommandButton2_Click()
Frame1.SetFocus
End Sub
```

Этот обработчик передает фокус управления в рамку **Frame1**. Но там уже расположены другие элементы управления, и поэтому команда передается на первый же элемент управления, в который этот фокус можно передать. Передача происходит в порядке расположения элементов на рамке (панели). Первый элемент управления — надпись **Label1**, но он не может получить фокус. Следующий, помещенный на рамку, — кнопка **CommandButton1**, в нее и будет передано управление. Поэтому программный код можно переписать так:

```
Private Sub CommandButton2_Click()
CommandButton1.SetFocus
End Sub
```

Как видно из примеров, требуется четко указывать адресный элемент управления. В первом примере мы могли добавить еще одну кнопку и случайно изменить **TabIndex**, сделав индекс новой кнопки первым — и тогда управление было бы передано

в эту кнопку, а та, куда мы непрямо передавали до этого, фокус не получит. Во втором же примере этого не произойдет никогда: мы явно указываем имя элемента управления (в данном примере кнопку **CommandButton1**), который получает фокус.

TabStrip — набор закладок

Внешне этот элемент управления напоминает многостраничный документ **MultiPage**, его мы будем рассматривать в следующем разделе. Между ними существует принципиальное отличие.

MultiPage — многостраничный документ, а **TabStrip** — одностораничный, со множеством закладок-кнопок. Нажимаем на кнопку — происходят какие-то действия.

Внешний вид кнопок (или того, что кажется кнопками) на элементе **TabStrip** определяется свойством **Style**. Здесь можно выбрать следующие значения:

- **fmTabStyleTabs** — страничный переключатель;
- **fmTabStyleButtons** — кнопочный переключатель;
- **fmTabStyleNone** — отсутствие переключателей.

Ориентация кнопок переключателей страниц определяется свойством **TabOrientation**. Здесь можно выбрать следующие значения:

- **fmTabOrientationTop** — сверху;
- **fmTabOrientationBottom** — внизу;
- **fmTabOrientationLeft** — слева;
- **fmTabOrientationRight** — справа.

В свойстве **Value** хранится индекс каждой закладки. Об этом впереди.

Создадим пример. Дважды щелкните по **TabStrip**, чтобы создать первоначальный код. У вас будет сформирован программный код:

```
Private Sub TabStrip2_Change()  
End Sub
```

Нам такой код не нужен. В списке **Procedure** (в окне **Code** в правом верхнем углу) выберите **Click**, щелкните по нему. Предыдущий программный останется незаполненным, будет сформирован другой программный код, где в событии **Click** будет находиться аргумент **Index**. Между строками этого программного кода напишите свой:

```
Private Sub TabStrip1_Click(ByVal Index As Long)  
If Index = 0 Then  
    MsgBox "Первая закладка"  
End If  
  
If Index = 1 Then  
    MsgBox "Вторая закладка"  
End If  
End Sub
```

Разберем этот код. Закладок по умолчанию у нас две (**Tab1** и **Tab2**), и логических блоков будет два. В первом блоке интересуемся закладкой с индексом **Index**, равным 0. Почему именно 0, а не 8, или не 666? Значение я беру из **Value** каждой закладки. На закладке **Tab1** значение **Value** равно 0, а на закладке **Tab2** равно 1. Если появится закладка **Tab3**, **Value** у нее будет 2.

В каждом логическом блоке определяется индекс кнопки, которую вы нажали, в зависимости от этого будут производиться действия. Предположим, требуется рассчитать две важные формулы. Одну закладываем в **Tab1**, вторую — в **Tab2**. При нажатии одной из кнопок программа будет определять ее индекс и выполнять расчет.

Предположим, вы создали следующую закладку — **Tab3**, но не включили ее в логический расчет. В этом случае кнопка на элементе управления будет, а вот действия она выполнять не будет: программа не знает, что именно нужно делать при щелчке по этой закладке.

MultiPage — многостраничное окно

Этот элемент управления используется очень часто. Он позволяет значительно расширить площадь формы без линеек прокрутки. Как правило, всевозможные опции задаются именно с помощью таких элементов управления.

MultiPage содержит набор одной или более страниц. Страница — это не закладка, как у **TabStrip**. Каждая страница (Page) MultiPage — форма, содержит собственные элементы управления и может иметь уникальный формат. Обычно у страниц в MultiPage есть переключатели, так что пользователь может выбрать индивидуальные страницы.

По умолчанию MultiPage включает две страницы — **Page1** и **Page2**. Каждая — объект страницы (Page), вместе они представляют набор страниц (Page) MultiPage. Если вы добавляете больше страниц, они становятся частью того же самого набора (Page).

Рассмотрим основные свойства этого элемента управления.

Свойства **Captions** можно изменять не только для элемента управления в целом, но и каждой страницы отдельно. Бывает путаница, поэтому постарайтесь сразу определить, что выделили — весь элемент MultiPage или одну из его страниц.

В MultiPage три контекстных меню. Одно на теле элемента; второе — на кнопках многостраничного окна и свободной области за кнопками, третье — на границе элемента. Набор команд в каждом различный. В расположенных на кнопках элемента управления находятся команды для добавления, удаления, переименования и перемещения страниц MultiPage. Добавление новой страницы осуществляется командой **New Page** из контекстного меню кнопок.

Внешний вид кнопок (или того, что нам кажется кнопками) на элементе MultiPage определяется свойством **Style**. Можно выбрать следующие значения:

- **fmTabStyleTabs** — страничный переключатель;
- **fmTabStyleButtons** — кнопочный переключатель;

- **fmTabStyleNone** — отсутствие переключателей.

Ориентация кнопок переключателей страниц определяется свойством **TabOrientation**. Здесь можно выбрать следующие значения:

- **fmTabOrientationTop** — сверху;
- **fmTabOrientationBottom** — внизу;
- **fmTabOrientationLeft** — слева;
- **fmTabOrientationRight** — справа.

Из свойств страниц **Page** отметим **Index**. Оно определяет порядок следования страниц. Индексация начинается с 0. Таким образом, **Page1** по умолчанию имеет **Index = 0**, **Page2** — **Index = 1** и т. д. Для изменения порядка следования страниц необходимо изменить их индексацию. Двойная индексация не допускается.

На страницах **Page** нужно рассмотреть свойство **TransitionEffect**. Оно определяет визуальный эффект, чтобы использовать замену одной страницы на другую (таблица 1). Это свойство может принимать следующие значения.

Таблица 1. Значения свойства TransitionEffect

Константа	Значение	Описание
fmTransitionEffectNone	0	Без специальных эффектов
fmTransitionEffectCoverUp	1	Новая страница закрывает старую, которая перемещается снизу на верх
fmTransitionEffectCoverRightUp	2	Новая страница закрывает старую, которая перемещается из нижнего левого угла в правый верхний угол

Таблица 1. Значения свойства TransitionEffect. Окончание

Константа	Значение	Описание
fmTransitionEffectCoverRight	3	Новая страница закрывает старую, которая перемещается с левого края направо
fmTransitionEffectCoverRightDown	4	Новая страница закрывает старую, которая перемещается из верхнего левого угла в нижний правый угол
fmTransitionEffectCoverDown	5	Новая страница закрывает старую, которая перемещается сверху вниз
fmTransitionEffectCoverLeftDown	6	Новая страница закрывает старую, которая перемещается из правого верхнего угла в нижний левый угол
fmTransitionEffectCoverLeft	7	Новая страница закрывает старую, которая перемещается справа налево
fmTransitionEffectCoverLeftUp	8	Новая страница закрывает старую, которая перемещается из нижнего правого угла в верхний левый угол
fmTransitionEffectPushUp	9	Новая страница выводит старую страницу из видимости, перемещаясь снизу наверх
fmTransitionEffectPushRight	10	Новая страница выводит старую страницу из видимости, перемещаясь слева направо
fmTransitionEffectPushDown	11	Новая страница выводит старую страницу из видимости, перемещаясь сверху вниз
fmTransitionEffectPushLeft	12	Новая страница выводит старую страницу из видимости, перемещаясь справа налево

Синтаксис свойства **TransitionEffect** следующий:

object.TransitionEffect [= fmTransitionEffect]

Где:

object — обязательный аргумент, возвращаемый объект;
fmTransitionEffect — необязательный аргумент, эффект перехода, который вы хотите установить между страницами.

Вместе с **TransitionEffect** используется свойство **TransitionPeriod**, в котором задается скорость перерисовки страницы Page. Это значение задается в миллисекундах, 1 секунда как 1000.

Для примера перенесите на каждую страницу несколько элементов управления и убедитесь, что работа с каждой страницей не отличается от работы с этими же элементами управления на форме.

Перенесите на каждую вкладку по одной надписи **Label** и по одной командной кнопке **CommandButton**. Разместите элементы на каждой вкладке в разных местах. На вкладке **Page1** в свойстве **TransitionEffect** выберите значение **fmTransitionEffectCoverLeftDown**, в свойство **TransitionPeriod** установите значение 1000. На вкладке **Page2** в **TransitionEffect** выберите значение **fmTransitionEffectCoverRight**, в свойство **TransitionPeriod** установите значение 1000.

На вкладке **Page1** для кнопки **CommandButton1** напишите следующий программный код:

```
Private Sub CommandButton1_Click()
MultiPage1.Pages.Add.Name = Page3
MultiPage1.Value = 1
Label2.Caption = "Вставка страницы"
End Sub
```

Во-первых, попробуйте теперь просто понажимать страницы **Page1** и **Page2**. При переходе с одной страницы на другую будет происходить небольшая мультипликация. Во-вторых, при нажатии на кнопку **CommandButton1** будет создаваться третья вкладка. Автоматически открывается вкладка **Page2**, в надписи **Label2** сообщается, что наше задание выполнено.

ScrollBar — полоса прокрутки

Элемент управления ScrollBar — это полосы прокрутки окна. Многие элементы управления (например, **TextBox**, **ComboBox**, **ListBox**) используют их, причем от разработчика не требуется написать для этого программный код — достаточно выбрать вид линейки прокрутки: вертикальную, горизонтальную или обе одновременно. Полоса прокрутки как самостоятельный элемент управления VBA хотя и предназначена для выполнения аналогичных функций, но не выполняет автоматически каких-либо действий, то есть ее поведение необходимо программировать.

Диапазон прокрутки определяется свойствами **Min** и **Max** полосы прокрутки. Значение **Min** всегда соответствует верхнему концу полосы, **Max** — нижнему (для вертикальной полосы прокрутки). При прокрутке содержимого окна сверху вниз значение свойств **Value** увеличивается. Чтобы изменить направление, достаточно поменять местами значения свойств **Min** и **Max**.

Свойство **SmallChange** определяет величину передвижения бегунка на линейке прокрутки при каждом щелчке по кнопкам, расположенным на концах линейки. Каждый щелчок изменяет значение **Value**.

Если пользователь щелкнет между бегунком и какой-либо из кнопок, значение **Value** полосы прокрутки и положение бегунка изменяются на величину, определяемую **LargeChange**.

Ориентация линейки определяется свойством **Orientation**. Можно выбрать следующие значения:

- **fmOrientationAuto** — автоматическая ориентация линейки;
- **fmOrientationVertical** — вертикальная ориентация линейки;
- **fmOrientationHorizontal** — горизонтальная ориентация линейки.

Для примера перенесите на форму кнопку **CommandButton**. Нажимая на **CommandButton1** на открытом рабочем листе, создаете линейку прокрутки. Лист может быть открыт любой. Линейка прокрутки все равно будет создана на листе, который

указан в тексте программы. Напишите обработчик щелчка кнопки:

```
Private Sub CommandButton1_Click()
Set RRR = Worksheets(1).Shapes.
AddFormControl(xlScrollBar, _
    Left:=25, Top:=25, Width:=15, Height:=200)
With RRR.ControlFormat
    .LinkedCell = "E10"
    .Max = 100
    .Min = 0
    .LargeChange = 10
    .SmallChange = 4
End With
End Sub
```

В этом примере создается объект RRR. Вместо RRR можете взять любое имя на любом языке. Затем создается линейка с параметрами Left:=25, Top:=25, Width:=15, Height:=200.

В ячейке E10 (LinkedCell = «E10») выводится текущее положение бегунка. Каждый щелчок на концевые кнопки перемещает бегунок на 4 единицы (SmallChange = 4). Каждый щелчок по линейке прокрутки перемещает бегунок на 10 единиц (LargeChange = 10).

SpinBox — счетчик

Этот элемент управления предназначен для инкрементов и декрементов. Щелчок по **SpinBox** изменяет только величину **SpinBox**. Вы можете написать код, который он использует, чтобы корректировать отображаемую величину другого управления: использовать **SpinBox**, чтобы изменять месяц, день, или год, перемещаться через диапазон величин или список пунктов, изменять величину, отображенную в текстовом окне.

Чтобы отображать величину, скорректированную **SpinBox**, вы должны назначить величину **SpinBox** в отображаемую часть управления — как, например, свойство **Caption** элемента управления **Label** или свойство **Text** элемента управления **TextBox**. Чтобы создавать горизонтальный или вертикальный **SpinBox**, перетащите калибровку ручек **SpinBox** горизонтально или вертикально.

Встроенное свойство для **SpinButton — Value**. Встроенное событие для **SpinButton — Change**.

Событие **Change** происходит, когда изменяется свойство **Value**.

Ориентация счетчика определяется свойством **Orientation**.

Здесь можно выбрать следующие значения:

- **fmOrientationAuto** — автоматическая ориентация счетчика;
- **fmOrientationVertical** — вертикальная ориентация счетчика;
- **fmOrientationHorizontal** — горизонтальная ориентация счетчика.

Диапазон изменения счетчика определяется свойствами **Min** и **Max**. Значение **Min** всегда соответствует верхнему пределу счетчика, **Max** — нижнему (для вертикального счетчика). При прокрутке содержимого окна сверху вниз значение свойства **Value** увеличивается. Чтобы изменить направление изменения свойств **Value**, достаточно поменять местами значения свойств **Min** и **Max**.

Свойство **SmallChange** определяет величину, на которую изменяется значение счетчика при каждом щелчке по кнопкам на концах счетчика. Каждый щелчок по крайним кнопкам изменяет значение **Value**.

Для примера перенесите на форму кнопку **SpinButton1** (можно использовать и обычную кнопку **CommandButton**). При нажатии **SpinButton1** в открытой рабочей книге на листе 2 создается счетчик. Лист при этом может быть открыт любой. Счетчик все равно будет создан на том листе, который указан. Напишите обработчик щелчка кнопки:

```
Private Sub SpinButton1_Change()  
Set RRR = Worksheets(2).Shapes.  
AddFormControl(xlSpinner,_  
Left:=25, Top:=25, Width:=15, Height:=30)  
With RRR.ControlFormat  
.LinkedCell = "E10"
```

```
.Max = 100  
.Min = 0  
.SmallChange = 1  
End With  
End Sub
```

В данном примере создается объект RRR. Вместо RRR вы можете взять любое имя на любом языке. Затем создается линейка с параметрами Left:=25, Top:=25, Width:=15, Height:=30.

В этом примере в ячейке E10 (LinkedCell = «E10») выводится текущее положение бегунка. Каждый щелчок на концевые кнопки изменяет значение счетчика на одну единицу (SmallChange = 1).

Так как мы очень часто используем элемент управления **SpinButton** на форме, создадим пример такого использования. Перенесите на форму один счетчик **SpinButton** и одно текстовое поле **TextBox**. Мы должны выдавать значения счетчика в текстовое поле, где оно будет отражаться. В свойствах счетчика **SpinButton** задайте: **Max** = 100; **Min** = -100. Расположите эти элементы управления рядом друг с другом. Например, текстовое поле **TextBox** слева, а счетчик **SpinButton** справа. Выделите **TextBox**, нажмите на клавишу Shift и, не отпуская клавишу, щелкните по счетчику **SpinButton**. У нас выделены оба элемента управления. Отпустите Shift и щелкните по выделенным объектам правой клавишей мыши. Выберите команду **Group** и нажмите на нее. Оба элемента управления оказались сгруппированными. Теперь при перемещении одного из них другой также будет перемещен.

Обращаю ваше внимание: мы не создали новый объединенный элемент управления, а всего лишь сгруппировали эти элементы. Если щелкнем по ним, сначала будет выделен общий элемент (будто это единое целое). Если щелкнем еще раз по одному из элементов, он выделится, но выделение группы не изменится. Это особенность группы. Вас это уже не должно удивлять: мы изучали элемент управления **Frame**, где выделение происходит аналогичным образом. В группе мы можем управлять какими-то общими свойствами, но можем выделять каждый элемент и изменять только его свойства.

Щелкните дважды счетчик **SpinButton** и напишите программный код:

```
Private Sub SpinButton1_Change()  
    TextBox1.Text = SpinButton1.Value  
End Sub
```

В этой процедуре мы присваиваем свойству **Text** элемента **TextBox1** текущее значение **Value** счетчика **SpinButton1**. Если вместо свойства **Text** элемента **TextBox1** возьмем свойство **Value**, будет то же самое:

```
TextBox1.Value = SpinButton1.Value
```

Элемент управления **TextBox** более всего подходит для выводения информации о значениях счетчика, хотя для этих целей можно использовать не только другие элементы управления, но и саму форму. В форме есть свойство **Caption**, можно выводить информацию в это свойство:

```
Private Sub SpinButton1_Change()  
    UserForm1.Caption = SpinButton1.Value  
End Sub
```

При щелчке по счетчику в заголовке формы будет отражаться его текущее значение.

В наборе дополнительных элементов управления есть элемент **UpDown**, который похож на только что рассмотренный.

Image — изображение

Элемент управления **Image** создан для отображения рисунков.

Главным свойством **Image** является **Picture**.

Свойство **PictureSizeMode** определяет масштаб изображения в окне **Image**. Свойство может принимать следующие значения:

- **fmPictureSizeModeClip** — реальный масштаб изображения;
- **fmPictureSizeModeStrech** — изображение занимает всю площадь окна **Image**;

- **fmPictureSizeModeZoom** — изображение масштабируется как квадрат, где ширина и высота равны минимальному размеру окна **Image**.

Свойство **PictureAlignment** определяет положение изображения в окне **Image**, если оно помещается в этом окне, не занимая всю его площадь. Значения этого свойства могут быть следующими:

- **fmPictureAlignmentTopLeft** — начиная с левого верхнего угла окна **Image**;
- **fmPictureAlignmentTopRight** — начиная с правого верхнего угла окна **Image**;
- **fmPictureAlignmentCenter** — в центре окна **Image**;
- **fmPictureAlignmentBottomLeft** — начиная с левого нижнего угла окна **Image**;
- **fmPictureAlignmentBottomRight** — начиная с правого нижнего угла окна **Image**.

Свойство **BorderStyle** определяет наличие рамки вокруг окна **Image**. Это свойство может принимать следующие значения:

- **fmBorderStyleNone** — рамки нет;
- **fmBorderStyleSingle** — рамка есть.

Свойство **SpecialEffect** определяет внешний вид окна **Image**. Это свойство может принимать следующие значения:

- **fmSpecialEffectFlat** — специального эффекта нет;
- **fmSpecialEffectRaised** — окно **Image** приподнято над формой;
- **fmSpecialEffectSunken** — окно **Image** вдавлено в форму;
- **fmSpecialEffectEtched** — рамка вокруг окна **Image** утоплена в форму;

- **fmSpecialEffectBump** — рамка вокруг окна **Image** приподнята над формой.

Свойства **SpecialEffect** и **BorderStyle** взаимосвязаны. Если в **BorderStyle** выбирается значение **fmBorderStyleSingle**, в **SpecialEffect** автоматически выбирается **fmSpecialEffectFlat**. Если в свойстве **fmSpecialEffectFlat** выбирается любое значение, кроме **fmSpecialEffectFlat**, в **BorderStyle** автоматически выбирается значение **fmBorderStyleNone**.

В элементе управления **Image** есть очень интересное свойство **PictureTiling**. Установите **PictureSizeMode** в значение **fmPictureSizeModeClip**. Если оно принимает значение **False**, в окне **Image** изображение выводится как одно изображение, а если равно **True**, площадь окна **Image** заливается плиткой (или обоями) в виде выбранного изображения в реальном масштабе.

Самый простой способ подгонки размера изображения под размер окна **Image** — установить свойство **AutoSize** в **True**.

RefEdit — поле интервала

Этот элемент управления отображает адрес диапазона ячеек, который был введен или выбран в одном или более рабочих листах. Чтобы выбрать диапазон, щелкните кнопкой на элементе управления, чтобы закрыть форму, выберите диапазон, затем снова щелкните кнопкой на управлении, чтобы открыть форму.

Если элемент управления **RefEdit** вставлен на рабочий лист, можно связать содержание управления с ячейками в любом рабочем листе в этой рабочей книге.

Встроенное свойство для управления **RefEdit** — **Value**. Встроенное событие для управления **RefEdit** — **BeforeDragOver**.

Вы не можете использовать управление **RefEdit** в модуле формы. Можете использовать свойство **ShowModal**, чтобы устанавливать форму как модальную.

```
UserForm1.RefEdit1.ControlTipText = _  
    "Выбранный диапазон данных в рабочем листе"  
UserForm1.Show
```

Рассмотрим пример щелчка по **CommandButton1** в **UserForm1**. Процедура обработки события отображает адрес выбранного диапазона рабочего листа, затем заканчивает программу.

Перенесите на форму одну командную кнопку **CommandButton** и одно поле диапазона **RefEdit**. На рабочем листе Excel создайте какие-нибудь значения, которые мы будем включать в диапазон. Перейдите в редактор VBA. Для щелчка по кнопке напишите следующий программный код:

```
Private Sub CommandButton1_Click()
    MsgBox RefEdit1.Value
End
End Sub
```

Скомпилируйте проект. На форме щелкните по кнопке **RefEdit**. Форма свернется. На рабочем листе выделите диапазон ячеек. Опять щелкните по кнопке элемента управления **RefEdit**. Форма будет восстановлена. Щелчок по **CommandButton1** вызывает диалоговое окно **MsgBox**, в котором сообщается диапазон выбранных ячеек и находится одна кнопка для закрытия. При щелчке по этой кнопке закрывается и окно **MsgBox**, и окно формы.

Дополнительные элементы управления

Дополнительные элементы управления расположены в диалоговом окне **Additional Controls**. Для его вызова в окне **Toolbox** щелкните правой клавишей по любой инструментальной кнопке с элементом управления. Обращаю внимание: для вызова диалогового окна **Additional Controls** щелкать нужно именно по кнопке с элементом управления, а не по имени страницы **Controls**. В появившемся контекстном меню необходимо выбрать команду **Additional Controls**. В открывшемся диалоговом окне **Additional Controls** в имени элемента управления, который вы собираетесь перенести в окно **Toolbox**, нужно установить флажок в индикаторе этого элемента (находится слева от имени).

Дополнительные элементы управления нужно создавать на новой странице, а не на странице **Controls**. Для этого щелкните правой клавишей мыши по имени страницы **Controls**. В контекстном меню выберите команду **New Page**. В окне **Toolbox** будет создана страница **New Page**. При необходимости можно имя новой страницы **New Page** переименовать с помощью команды **Rename**, удалить с помощью команды **Delete Page** и т. д.

Конкретный набор дополнительных элементов управления может быть различным на каждом конкретном компьютере. Это зависит от многих причин, в том числе и от конкретного содержания, и версии ActiveX, инсталлированной у вас на компьютере.

Предупреждаю, что, к сожалению, многие элементы управления, которые будут находиться в диалоговом окне **Additional Controls**, не будут работать: на их использование нужно специальное разрешение.

ProgressBar

ProgressBar — это индикатор, который отображает, насколько продвинулся какой-либо процесс (копирование, перемещение, загрузка или сохранение файлов). Этот элемент по своим свойствам очень похож на линейку прокрутки **ScrollBar**. Он также имеет свойства **Min**, **Max** и **Value**, которые устанавливают границы области диапазона и текущее значение.

Определить значение свойств **Value** предоставляется программисту, так как элемент управления **ProgressBar** не может отслеживать продвижение процесса. Свойство **Value**, которое во многих элементах управления выведено в окно свойств **Properties**, непосредственно в этом окне не видно (оно здесь не нужно, но его можно задать программным путем).

Свойство **Appearance** определяет объемность внешнего вида элемента управления. Оно может принимать следующие значения:

- **ccFlat** — плоский вид;
- **cc3D** — объемный вид.

Свойство **Orientation** устанавливает ориентацию ползунка:

- **ccOrientationHorizontal** — по горизонтали;
- **ccOrientationVertical** — по вертикали.

Scrolling определяет внешний вид внутреннего заполнения индикатора:

- **ccScrollingStandard** — дискретный вид делений;
- **ccScrollingSmooth** — непрерывный вид делений.

Свойство **Custom** открывает диалоговое окно **Property Pages**, в котором можно не только установить основные свойства элемента управления **ProgressBar**, но и выбрать новую иконку, которая будет выводиться во время работы **ProgressBar**, конечно, если они у вас имеются. Но скажу по своему опыту: нарисовать новые иконки несложно, рисование происходит по пикселям, а пиксели можно сильно увеличить.

Этот элемент управления необходимо устанавливать по значению **Value**. Рассмотрим очень простой пример:

```
Private Sub CommandButton1_Click()
    ProgressBar1.Value = 50
End Sub
```

Заполнение закрашенной области индикатора происходит в соответствии со значением **Value**.

StatusBar

StatusBar — полоса состояния, как правило, находящаяся в нижней части окна, на которую выводится служебная информация: номер строки и/или столбца, номер ячейки, дата, время, включение различных клавиш клавиатуры и многое другое.

Свойство **Style** устанавливает вид полосы состояния: она может быть простой, состоять из одной или нескольких частей:

- **sbrNormal** — полоса из более чем одной индикаторной панели;

- **sbrSimple** — полоса из одной индикаторной панели.

Основные свойства задаются в диалоговом окне **Property Pages**. Открытие диалогового окна **Property Pages** производится нажатием на кнопку свойства **Custom**.

На вкладке **General** определяются общие для всей полосы состояния **StatusBar** параметры: **Style** (Стиль), **MousePointer** (Вид указателя мыши), **SimpleText** (Текст полосы при **Style = sbrSimple**).

На вкладке **Panels** определяются аргументы каждой панели и возможность создания и удаления других индикаторных панелей в **StatusBar**.

Для создания новой панели нужно нажать на кнопку **Insert Panel** (Вставка панели). Нумерация панелей начинается от 1. Полоса состояния может содержать до 16 отдельных панелей. При попытке вставить 17-ю будет выдано сообщение, что достигнут предел в нумерации. Просмотр имеющихся на полосе **StatusBar** панелей осуществляется с помощью счетчика **Index**. Удалить панель на полосе состояния можно кнопкой **Remove Panel**.

Текст в панели задается в поле **Text**. В поле **ToolTipText** задается подсказка (или хint), которая появляется при наведении указателя мыши на панель в общей полосе **StatusBar**. Например, если на панели с индексом (**Index**) 2 мы введем в это свойство текст **Текущая ячейка**, при наведении на эту панель всплывет указанный текст, а при наведении на панель с индексом 1 — другая подсказка, заданная для этого индекса, или не будет вообще никакой подсказки, если она в этом индексе не задана.

В поле **Key** указывается ключ панели. Так как понятие ключа нам пока ни о чем не говорит, рассмотрим его подробнее при составлении примеров.

В раскрывающемся списке **Alignment** выбирается вариант выравнивания. Оно может быть в трех вариантах: **По левому краю**, **По центру** и **По правому краю**.

В списке **Style** предлагаются наиболее часто применяемые стили вывода информации на панель полосы состояния:

- **sbrText** — выводится из поля **Text** диалогового окна **Property Pages** на вкладке **Panels**;
- **sbrCaps** — выводится индикация CAPS;
- **sbrNum** — выводится индикация NUM;
- **sbrIns** — выводится индикация INS;
- **sbrScrl** — выводится индикация SCRL;
- **sbrTime** — выводится индикация текущего времени;
- **sbrDate** — выводится индикация текущей даты;
- **sbrKana** — выводится индикация KANA.

При изменении состояния кнопок на клавиатуре CAPS, NUM, INS, SCRL их высвечивание на панелях меняется: при включении этих режимов индикаторы полностью высвечены; при отключении становятся окрашенными в бледно-серые цвета, что означает недоступность или отключение.

В списке **Bevel** находятся значения внешнего вида панелей полосы состояния. Здесь следующие значения:

- **sbrNoBevel** — рамок нет;
- **sbrInset** — панель вдавлена в форму;
- **sbrRaised** — панель возвышается над формой.

Индикаторы **Enabled** и **Visible** определяют доступность и видимость соответствующих панелей на полосе состояния. Если флагок установлен в индикатор, свойство включено. Если флагок сброшен, свойство отключено.

На панель можно выводить как текст, так и текст и изображение. По умолчанию вывод картинок отключен, так как не известно, какую картинку выводить. Если необходимо включить картинку, нужно нажать на **Browse** на панели **Picture**. При нажатии на **Browse** открывается диалоговое окно **Select Picture**, в котором можно выбрать иконку (если они у вас есть). Если нет, их можно нарисовать, это несложно. Картинка располагается на панели слева, а текстовое сообщение — справа.

Если выбранная иконка больше не нужна, ее можно отключить, нажав на кнопку **No Picture**.

На панели есть два элемента — текст и иконка, поэтому возникают вопросы: как их выровнять? возможно ли наложение, пересечение друг с другом? Особенно это касается выравнивания (**Alignment**) с левого края. Не нужно об этом беспокоиться — картинка и текст не пересекаются и не закрывают друг друга. При выравнивании слева сначала прорисовывается картинка, и только потом идет текст сообщения.

На вкладке **Шрифт** диалогового окна **Property Pages** — выбор шрифтовых параметров.

На вкладке **Рисунок** окна **Property Pages** — выбор иконки указателя мыши, если свойство **MousePointer** элемента управления **StatusBar** установлено в **ccCustom**.

Для выбора изменений на вкладках диалогового окна **Property Pages** нужно нажать на клавишу **Применить** и затем **OK**.

Создадим пример. Перенесите на форму одну **CommandButton**. Напишите программный код для обработки щелчка по этой кнопке:

```
Private Sub CommandButton1_Click()
oldStatusBar = Application.DisplayStatusBar
Application.DisplayStatusBar = True
Application.StatusBar = "Пожалуйста, подождите..."
Workbooks.Open Filename:="C:\Program Files\Microsoft
Office\Office10\1049\FUNCS.XLS"
Application.StatusBar = False
```

```
Application.DisplayStatusBar = oldStatusBar  
End Sub
```

Сначала запоминается состояние полосы, затем загружается указанный файл. Во время загрузки на полосу состояния выводится сообщение, затем загружается предыдущее состояние полосы, а информационная надпись меняется. Вы можете познакомиться в файле FUNC.S.XLS с русскими терминами и их английскими эквивалентами.

В следующем примере создадим полосу состояния для нашей формы. Перенесите на форму две кнопки **CommandButton**, одну полосу состояния **StatusBar**. Щелчок по одной кнопке — и полоса будет превращаться в однопанельную, по другой — в двухпанельную. Напишите следующий программный код:

```
Private Sub CommandButton1_Click()  
StatusBar1.Style = sbrSimple  
StatusBar1.SimpleText = "Выбор сделан"  
End Sub  
  
Private Sub CommandButton2_Click()  
StatusBar1.Style = sbrNormal  
StatusBar1.Panels.Add.Style = sbrDate  
End Sub
```

Щелчок по второй кнопке создает новую панель на полосе состояния с текущей датой.

Поговорим теперь о ключе **Key**. Ключ — это, говоря проще, пароль, который мы задаем, чтобы отличить одну панель на полосе состояния от другой. Для примера будем использовать две панели на полосе состояния. В одной задайте ключ **Key** равным QQ, а в другой — WW. В качестве ключей используется любой набор символов.

Для программы вся панель представляет единое целое, поэтому, чтобы отличить одну от другой, необходимо задать ключи.

Перенесите на форму текстовое поле **TextBox**, чтобы немного усложнить задачу, но заодно и показать, что все работает

по-прежнему. Щелкните дважды по **StatusBar**, чтобы создать первоначальный код. Перепишите следующий программный код:

```
Private Sub StatusBar1_PanelClick(ByVal Panel As  
MSComctlLib.Panel)  
If Panel.Key = "QQ" Then  
    TextBox1.Text = "Пример работает!!!"  
    MsgBox "Работает панель QQ"  
End If  
If Panel.Key = "WW" Then  
    TextBox1.Text = "Второй пример работает!!!"  
    MsgBox "Работает панель WW"  
End If  
End Sub
```

При щелчке по первой панели полосы состояния выдается одно сообщение, а при щелчке по второй — другое состояние.

Обратите внимание: весь программный код для полосы состояния находится в одной процедуре. Программа различает конкретные панели по ключу.

ToolBar

ToolBar — это панель с различными кнопками, свойства которых определяет программист.

Это происходит в диалоговом окне **Property Pages**. Диалоговое окно **Property Pages** открываем кнопкой **Custom**.

На вкладке **General** определяются общие для всей полосы состояния **ToolBar** параметры: изображение указателя мыши **MousePointer**, источник картинок **ImageList**, стиль **Style**, выравнивание и т. д.

На вкладке **Buttons** создаются кнопки на инструментальной панели **ToolBar**. Каждое нажатие на **Insert Button** создает новую кнопку с очередным индексом, а каждое нажатие на **Remove Button** удаляет ту, индекс (**Index**) которой определен счетчиком **Index**.

В поле **Caption** задается заголовок кнопки. В поле **Key** задается ее ключ.

В списке **Style** выбирается вид кнопки:

- **tbrDefault** — кнопка по умолчанию (типа **CommandButton**);
- **tbrCheck** — двухходовая кнопка (нажато — отжато), аналог **CheckBox**;
- **tbrButtonGroup** — групповая кнопка (в группе кнопок может быть нажата только одна, остальные обязательно отжимаются), аналог **OptionButton**;
- **tbrSeparator** — разделитель (сепаратор) между кнопками;
- **tbrPlaceholder** — разделитель между кнопками;
- **tbrDropDown** — кнопка со списком.

В списке **Value** находятся значения, определяющие стартовое положение кнопки с индексом. Кнопка может быть или отжата — значение **tbrUnpressed**, или нажата — значение **tbrPressed**.

Так как это довольно сложный элемент управления, создадим пример с подробным объяснением каждого действия.

Сложность в следующем. Все кнопки на инструментальной панели имеют событие **Click**: какую бы мы ни нажали, происходит одно и то же событие. Главная сложность — идентифицировать конкретную кнопку на этой панели. Я уверен, вы уже попробовали, но какую бы ни нажимали, происходит одно и то же действие. Для новичков, впервые сталкивающихся с этой панелью, первоначальные ощущения обычно не очень приятные, так как непонятно — что делать и как идентифицировать отдельные кнопки.

Идентификация осуществляется с помощью ключа **Key**. Создайте несколько кнопок на инструментальной панели **ToolBar** с помощью **Insert Button**. В примере будут использованы две кнопки. Задайте каждой заголовки **Caption**: для первой **Can (Cancel)**, для второй **OK**. Задайте ключи **Key**: для первой кнопки **ZZ**, для второй **XX**. Ключи берутся с потолка, там обычно находится

одновременно много и ключей, и всякой другой нужной нам всячины. Оставьте **Style** кнопок по умолчанию: **tbrDefault**. Щелкните по инструментальной панели два раза, чтобы создать исходный программный код. Напишите следующий код:

```
Private Sub Toolbar1_ButtonClick(ByVal Button As  
MSComctlLib.Button)  
If Button.Key = "ZZ" Then  
MsgBox "Работает кнопка с ключом ZZ"  
End If  
If Button.Key = "XX" Then  
MsgBox "Работает кнопка с ключом XX"  
End If  
End Sub
```

Щелчки всех кнопок на инструментальной панели обрабатываются в одной процедуре. Если нажать кнопку с ключом **ZZ**, происходит одно действие — сообщение *Работает кнопка с ключом ZZ*, если нажимается кнопка с ключом **XX**, выдается сообщение *Работает кнопка с ключом XX*.

Если после этого перенести на форму различные текстовые элементы управления (**Label**, **TextBox**), убедитесь, что в них также можно посыпать сообщения, считывать данные и т. д. с помощью кнопок инструментальной панели. Попробуйте создать такие примеры самостоятельно, это просто.

Рассмотрим очень сложный вопрос — перенесение изображений на кнопки инструментальной панели **ToolBar**. По умолчанию изображений на кнопках нет. Управление изображениями на кнопках находится в **ImageList**. Если в дополнительных элементах управления он у вас есть, перенесите его на форму. Выделите **ImageList**, найдите свойство **Custom** и откройте диалоговое окно **Property Pages**. Перейдите на вкладку **Images**. Здесь выбираются картинки для кнопок. Если у вас есть хоть какой-нибудь набор картинок, нажмите на **Insert Picture**. Откроется диалоговое окно **Select Picture**, здесь можно выбрать файл с картинкой для кнопки. После выбора очередной картинки окно **Select Picture** закрывается, вы возвращаетесь на вкладку **Images**. Новая картинка попадает в список **Images**, расположенный над полосой кнопок

на вкладке **Images**. Для примера достаточно выбрать две картинки. При выделении каждой выводятся ее параметры. Из параметров нужно знать только **Index** и **Key**. Если **Index** задается автоматически по мере добавления картинок в список **Images**, **Key** вводится вручную. Предположим, что мы задали **Key** для первой картинки AAA, а для второй SSS.

На инструментальной панели **Toolbar** создаем две инструментальные кнопки с теми же значениями **Key**. Значения **Key** и в **ImageList**, и в **Toolbar** должны совпадать.

Вывод картинок на инструментальных кнопках должен произвольиться во время загрузки формы. Следовательно, нам потребуется событие **Activate** (подробнее разберем позже). Чтобы вызвать событие **Activate**, дважды щелкните по форме — сформируем первоначальный код. При этом будет сформирован код для события **UserForm_Click**. Нам это событие не нужно. В окне **Code** (Код) в верхнем правом раскрывающемся списке будет высвечена надпись **Click**. Раскройте этот список и выберите значение **Activate**. Щелкните по нему. Сформируется другой первоначальный код, между строк которого напишите свой программный код. Общий вид программы будет выглядеть так:

```
Private Sub UserForm_Activate()
Set Toolbar1.ImageList = ImageList1
Dim myButton As Variant
For Each myButton In Toolbar1.Buttons
If myButton.Key <> Empty Then
myButton.Image = myButton.Key
myButton.Description = myButton.Key
myButton.ToolTipText = myButton.Key
End If
Next
End Sub
```

После компиляции приложения на инструментальной панели на кнопках будут картинки.

TreeView

TreeView — элемент управления для отображения иерархических структур.

Управление свойствами удобно проводить в диалоговом окне **Property Pages**, которое можно открыть в **Custom**.

Из свойств в диалоговом окне **Property Pages** на вкладке **General** можно особо отметить **tvwTreelinesPlusMinusPictureText**, которое оформляет элементы в списке зависимых элементов с помощью символов плюс (+) и минус (-). Щелчок по плюсу разворачивает список, щелчок по минусу — сворачивает. Символы плюс и минус находятся только там, где есть зависимые элементы: нет зависимых элементов — нет плюса и минуса.

Надписи изменяются двойным щелчком (щелкайте медленно!) по имени элемента в наборе. После этого надпись оказывается выделенной в рамке. Изменение возможно лишь в том случае, если свойство **LabelEdit** установлено в **tvwAutomatic**. Чтобы запретить изменения, нужно установить свойство **LabelEdit** в **tvwManual**.

Для вставки дерева при загрузке формы используется метод **Add**, который имеет следующий синтаксис:

```
TreeView(Номер).Nodes.Add(Relative, Relationship, Key, Text,  
Image, SelectedImage)
```

Где:

Relative — свойство **Index** или **Key** существующего объекта;
Relationship — вид отношения к указанному в **Relative** объекту;

Key — уникальная строка-идентификатор;

Text — надпись;

Image — графическое изображение (индекс из **ImageList**);

SelectedImage — графическое изображение выбранного объекта (индекс из **ImageList**).

Изменять заголовки элементов можно с помощью свойств **BeforeLabelEdit** и **AfterLabelEdit**:

```
Private Sub TreeView1_AfterLabelEdit(Cancel As  
Integer, NewString As String)
```

```
End Sub
```

или

```
Private Sub TreeView1_BeforeLabelEdit(Cancel As Integer)  
End Sub
```

Например, можно написать так:

```
Private Sub TreeView1_AfterLabelEdit(Cancel As Integer, NewString As String)  
Cancel = True  
MsgBox NewString & "Элемент не найден"  
End Sub
```

В события передаются один или два аргумента: **Cancel**, определяющий возможность прекращения редактирования, и **NewString** — новое значение надписи.

Из событий можно еще остановиться на **Expand** и **Collaps**, вызываемых при разворачивании и сворачивании ветви. Аргумент **Node** (узел ветви) содержит ссылку на соответствующую ветвь. Событие **NodeClick** — это аналог события **Click**, но для конкретной ветви узла, ссылка на которую передается в процедуру обработки этого события.

Свойство **Checkboxes** при значении **True** отображает ветви в виде индикаторных флагков (**CheckBox**). **FullRowSelect** позволяет задать или прочесть значение, определяющее полный выбор ветви. Свойство **Scroll** определяет, будут ли отображаться полосы прокрутки, а **SingleSel** задает возможность разворачивания ветви при ее выборе.

RichTextBox

RichTextBox — текстовое окно, позволяющее вводить многострочный текст в формате RTF — это текст, похожий на формат Word.

Основное свойство **Text**.

Задание аргументов осуществляется в диалоговом окне **Property Pages**, которое можно открыть в свойстве **Custom**. На вкладке **General** в поле **FileName** выбираем файл, который можно загрузить в элемент управления: нужно нажать на кнопку

с многоточием (находится справа от этого поля). В окно **RichTextBox** можно загрузить файлы с форматом *.TXT и *.RTF.

Для более удобного просмотра текста используйте полосы прокрутки, они устанавливаются на вкладке **Appearance**.

Slider

Slider — это элемент управления-ползунок, устанавливающий значения из заданного диапазона. По свойствам очень похож на линейку прокрутки **ScrollBar**. Он также имеет свойства **Min**, **Max** и **Value**, которые устанавливают область диапазона и текущее значение.

Свойство **Orientation** устанавливает ориентацию ползунка:

- **ccOrientationHorizontal** — по горизонтали;
- **ccOrientationVertical** — по вертикали.

Свойство **SelectRange** определяет высоту внутренней полосы бегунка. Если **SelectRange** установлен в **False**, внутренняя полоса имеет вид узкой прорези. Если **SelectRange** установлен в **True**, внутренняя полоса шире.

Свойство **TickStyle** определяет внешний вид указателя бегунка и расположения цены деления. Он может принимать следующие значения:

- **sldBottomRight** — вниз и вправо, деления находятся в направлении указателя бегунка;
- **sldTopLeft** — вверх и влево, деления находятся в направлении указателя бегунка;
- **sldBoth** — равномерный прямоугольный бегунок, деления с обеих сторон бегунка;
- **sldNoTicks** — без цены деления.

Свойство **TickFrequency** определяет частоту меток шкалы бегунка.

Свойство **SelStart** определяет начальное положение указателя на бегунке.

Свойство **LargeChange** определяет величину, на которую будет изменяться значение указателя бегунка, если пользователь станет нажимать на клавиатуре клавиши быстрого перемещения **PageUp** и **PageDown**.

Свойство **SmallChange** определяет величину, на которую будет увеличиваться значение указателя бегунка, если пользователь нажимает на клавиатуре клавиши **↔**, **⇨**, **↑**, **↓**.

Так как этот элемент управления очень похож на **ScrollBar**, рассмотрим аналогичный пример, в котором при щелчке на одну кнопку указатель бегунка перемещается на деление вперед, а при щелчке по другой кнопке — на деление назад. Перенесите на форму две **CommandButton**, один бегунок **Slider** и одно поле **TextBox**. Смысль примера в том, чтобы, нажимая на одну из кнопок, изменить местоположение указателя бегунка на единицу, вывести полученное значение в текстовое поле (так как получение визуальных числовых значений очень удобно для понимания происходящего) и изменить внешний вид указателя бегунка. Для первой кнопки напишем следующий обработчик:

```
Private Sub CommandButton1_Click()
Slider1.Value = Slider1.Value + 1
Slider1.TickStyle = sldTopLeft
TextBox1.Text = Slider1.Value
End Sub
```

Обработчик второй кнопки также меняет внешний вид указателя на бегунке:

```
Private Sub CommandButton2_Click()
Slider1.Value = Slider1.Value - 1
Slider1.TickStyle = sldBoth
TextBox1.Text = Slider1.Value
End Sub
```

Свойства элемента управления **Slider** можно задавать в диалоговом окне **Property Pages**, которое открывается из окна **Properties**

при нажатии свойства **Custom**. Изучая предыдущие элементы управления, мы изучали такие же окна для других элементов, и вы уже можете самостоятельно разобраться с этим окном: оно в основном повторяет их вкладки и свойства.

ChartSpace

ChartSpace — очень простая для изучения и очень мощная система построения диаграмм.

Построить диаграмму можно в диалоговом окне **Property Pages**.

Перечень вкладок в диалоговом окне зависит от выбора на первой вкладке — **Источник данных**. Если выбирается значение **Лист**, две другие вкладки называются **Лист данных** и **Тип**.

Если выбран вариант **Запрос** или **таблица базы данных**, две другие вкладки называются **Данные** и **Тип**.

В первой есть некоторые различия: если выбран **Лист**, в пункте 2 кнопка называется **Лист данных**, если вариант **Запрос** или **таблица базы данных**, кнопка называется **Подключение**.

Оба варианта в принципе похожи, поэтому рассмотрим один — вариант **Лист**. Нажмите на кнопку **Лист данных**. Перейдите на вкладку **Лист данных** и введите данные. Перейдите на вкладку **Тип** и выберите тип диаграммы. Нажмите на кнопку **OK**.

У диаграммы есть контекстное меню, где находятся команды для работы с ней. Команда **Команды и параметры** содержит параметры для работы с цветом, объемом и т. д. Кроме этого можно подключать инструментальную панель для работы с диаграммами.

Краткое описание дополнительных элементов управления

Элемент управления **Calendar** предназначен для работы с датами. Дату выводит свойство **Value**. Настройка основных свойств производится в диалоговом окне **Property Pages**. Как правило, изменять нужно **Первый день недели** (в разных странах первый день недели разный).

Для работы с датами и временем есть еще два элемента управления — **Date and Time Picker Control** и **MonthView Control**.

DateTimePicker — поле со списком, позволяет отображать дату и/или время. Этот элемент управления может функционировать в двух режимах. Режим раскрывающегося календаря (установлен по умолчанию) позволяет выбирать дату в календаре; режим форматирования времени — выбирать поле в отображаемой дате (день, месяц, год и т. п.) и изменять его значение, нажимая кнопки на счетчике **UpDown** (в правой части поля со списком). В последнем случае в диалоговом окне **Property Pages** на вкладке **General** нужно установить флагок в индикатор **UpDown**.

Важными являются свойства **Format** и **CustomFormat**, определяющие формат отображения даты и времени. Свойство **Format** может принимать одно из следующих значений: **dtpLongDate** (дата отображается в длинном формате), **dtpShortDate** (дата отображается в кратком формате), **dtpTime** (формат времени) или **dtpCustom** (пользовательский формат). Если значение **Format** равно **dtpCustom**, при отображении даты и/или времени применяется пользовательский формат, определяемый свойством **CustomFormat**.

Среди событий отметим **CloseUp**, происходящее после закрытия ниспадающего календаря. Другое важное событие — **Change**, происходит при изменении содержимого элемента управления.

Элемент управления **MonthView** позволяет выбрать дату или последовательность дат с помощью графического представления календаря. С помощью свойств **MinDate** и **MaxDate** разработчик может ограничить диапазон дат, с которыми будет работать пользователь. Свойство **MultiSelect** определяет выбор непрерывной последовательности дат, а **MaxSelCount** — длину этой последовательности в днях. Для считывания или установки начальной и конечной дат предназначены свойства **SelStart** и **SelEnd**. **MonthColumns** и **MonthRows** позволяют указать число месяцев, которые будут отображаться на экране (по горизонтали и по вертикали), тогда можно просматривать одновременно на экране календарь на несколько месяцев. Этот элемент управления можно также связать с определенным полем базы данных, для чего тоже предусмотрен набор свойств.

Среди событий этого элемента управления выделим **DateClick** и **DateDblclick**, которые происходят после соответственно одинарного или двойного щелчка на дате. Обрабатывая их и используя свойство **Value**, программист может определить значение даты, выбранной пользователем.

Элемент управления **UpDown** — счетчик, похожий на счетчик **SpinButton**.

Элемент управления **Spreadsheet** предназначен для редактирования листов Excel на форме. Данные можно корректировать, выбирать рабочий лист, вводить формулы, импортировать в рабочую книгу Excel.

ImageList не используется самостоятельно. Он предоставляет другим элементам управления список графических образов. **ImageList** невидим в форме при выполнении программы. Он предоставляет информацию для других элементов управления и не имеет собственных событий. Отдельные изображения **ImageList** просто используются другими элементами управления.

FlatScrollBar — аналог полосы прокрутки **ScrollBar**. Может отображаться в одном из трех видов, определяемых значением свойства **Apperance**: трехмерная полоса прокрутки (**fsb3D**), плоская двумерная полоса прокрутки (**fsbFlat**), плоская двумерная полоса прокрутки с ползунком, который становится трехмерным при прохождении над ним указателя мыши (**fsbTrack3D**).

Animation — элемент отображения анимации. При этом могут воспроизводиться только несжатые AVI-файлы без звука или сжатые с использованием технологии Run Length Encoding (RLE). Важными являются методы **Open** и **Play**, с помощью которых можно загружать и воспроизводить видеофайлы.

CoolBar используется для создания конфигурируемых пользователем панелей элементов.

ImageCombo похож на элемент управления **ComboBox** с одним дополнительным преимуществом: вы можете добавлять графические объекты.

ListView — элемент для отображения списка каких-либо элементов, но без иерархической структуры.

Вставка элементов управления на форму

На панели элементов управления **Toolbox** находится кнопка **Select Objects**. По умолчанию эта кнопка нажата. При нажатии на любую другую кнопку на этой панели **Select Objects** отжимается, а нажимается та, на которую вы нажали. Нажатая на панели **Toolbox** кнопка какого-то элемента управления не означает, что выбор элемента управления сделан. Можно нажать на тот элемент управления, который вам действительно нужен. При этом предыдущая кнопка отжимается, а нажимается новая. Если вы вообще передумали выбирать какой-либо элемент управления, нажатие на кнопку **Select Objects** отменяет выбор любого из них.

Когда выбор сделан (кнопка с именем и пиктограммой нажата), элемент управления можно переносить на форму. Щелкните по форме. Точка, по которой щелкнули, — это верхняя левая точка элемента управления на форме.

Его можно переместить в любое другое место формы. Выделите его щелчком левой клавиши по любому месту элемента управления. Вокруг образуется рамка с восемью белыми квадратными маркерами. Если провести указателем мыши по рамке вокруг элемента управления, указатель мыши принимает вид крестообразной стрелки. Теперь его можно перемещать по форме. Нажмите левую клавишу и, двигая мышь, переместите элемент управления в нужное место.

Белые маркеры вокруг рамки предназначены для измерения размеров элемента управления. Проведите указателем мыши по маркеру — появляется двунаправленная стрелка, которая показывает направление изменения размеров. Размеры можно изменять в тех направлениях, куда показывают стрелки. Дождитесь появления двунаправленной стрелки и, щелкнув левой клавишей, потяните в нужную сторону. При нажатии на левую

клавишу вокруг элемента управления появляется пунктирная рамка. При перемещении в ту или иную сторону она будет показывать изменяемые размеры элемента управления, хотя он сам размеры изменять не будет. Отпустите левую клавишу — пунктирная рамка исчезнет, а элемент управления примет заданные размеры. При этом рамка выделения и белые маркеры останутся в этом же элементе управления.

Для удаления элемента управления с формы необходимо его выделить и выполнить одно из следующих действий:

- нажать на клавишу **Delete**;
- щелкнуть правой клавишей мыши по элементу управления, из контекстного меню выбрать команду **Delete** или **Cut**;
- выполнить команду **Edit ⇒ Delete** или **Cut**.

Все элементы управления, помещенные на форму, заносятся в специальный список **Tab Order** в той последовательности, в которой переносились на форму. В этой последовательности будет осуществляться переход от одного элемента управления к другому при нажатии на клавишу Tab. Но в дальнейшем она может измениться. Например, мы можем ввести элементы управления в следующей последовательности: **TextBox1**, **Label1**, **TextBox2** и **TextBox3**. Это может потребоваться, чтобы вводить два числа в **TextBox1** и **TextBox2** и выдавать результат в **TextBox3**. Нам удобно после ввода первого числа в **TextBox1** после нажатия на клавишу Tab вводить второе число в **TextBox2**. Но мы не сможем этого сделать, так как после элемента управления поместили на форму не **TextBox2**, а **Label1**. Порядок перехода от одного элемента управления к другому можно изменить в окне **Tab Order**.

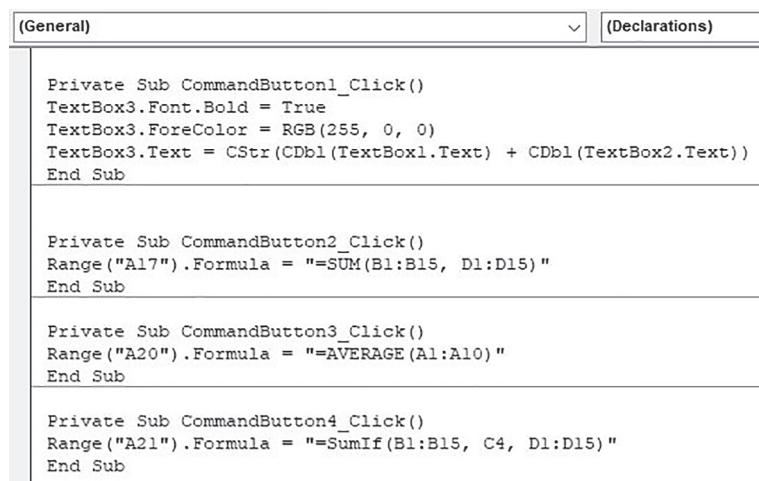
Выделите форму и щелкните по ней правой клавишей мыши, чтобы вызвать контекстное меню. Выберите команду **Tab Order**.

Откроется одноименное диалоговое окно. Чтобы переместить один элемент управления относительно других, нужно выделить его щелчком левой клавиши и с помощью кнопок **Move Up** или **Move Down** переместить вверх или вниз. Одно нажатие

на **Move Up** или **Move Down** перемещает выделенный элемент управления вверх или вниз на один уровень. Если он достиг самого верхнего или самого нижнего уровня, кнопки **Move Up** или **Move Down** не перемещают его в списке вверх или вниз — это физически невозможно. После изменения порядка следования элементов управления необходимо нажать **OK** для фиксации сделанных изменений и закрытия окна **Tab Order**.

Окно кода

Сразу после запуска VBA это окно не отображается. Тем не менее оно едва ли не самое важное в VBA: именно в нем вводится программный код (рис. 10). Код в VBA разделяется на процедуры и обычно непосредственно связан с определенными элементами управления. Это позволяет открыть окно кода двойным щелчком на элементе управления в форме.



(General) ▾ (Declarations)

```
Private Sub CommandButton1_Click()
    TextBox3.Font.Bold = True
    TextBox3.ForeColor = RGB(255, 0, 0)
    TextBox3.Text = CStr(CDbl(TextBox1.Text) + CDbl(TextBox2.Text))
End Sub

Private Sub CommandButton2_Click()
    Range("A17").Formula = "=SUM(B1:B15, D1:D15)"
End Sub

Private Sub CommandButton3_Click()
    Range("A20").Formula = "=AVERAGE(A1:A10)"
End Sub

Private Sub CommandButton4_Click()
    Range("A21").Formula = "=SumIf(B1:B15, C4, D1:D15)"
End Sub
```

Рисунок 10. Окно кода с записанной программой

Число строк кода (формы, модуля и т. п.) ограничивается 65 534. Это ограничение несущественно, поскольку в большинстве программ число строк значительно меньше.

Окно выполнения

Окно **Immediate** отображает информацию, получаемую из отладочных утверждений или строк в вашем коде или из команд, набранных в окне **Immediate**. Чтобы отобразить окно **Immediate** в меню **View** (Вид), выберите окно **Immediate** (CTRL + G).

Чтобы выполнять код в окне **Immediate**, наберите строки кода в окне **Immediate**. Нажмите Enter, чтобы выполнить утверждение.

Используйте окно **Immediate** для тестирования проблемного или вновь написанного кода, запроса или изменения величины переменной при прогоне приложения. Пока выполнение тестирования приостановлено, назначьте переменной новое значение, как будто это записывается в обычном коде.

Утверждения окна **Immediate** выполняются в контексте, то есть как будто бы они введены в специальный модуль.

Панель элементов управления

Панель **Toolbox** содержит по умолчанию 15 элементов управления (рис. 11). Все они — объекты, которые можно переместить на форму или на рабочий лист.

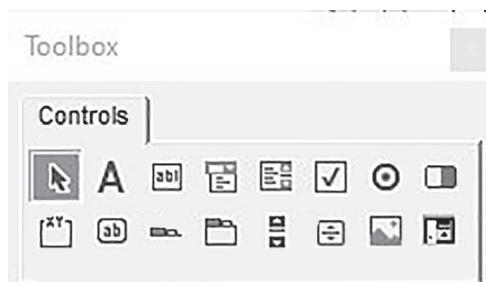


Рисунок 11. Панель с элементами управления

Кроме основных элементов управления, существует огромное число других. Чтобы просмотреть и выбрать другие элементы управления, щелкните правой клавишей по любой кнопке **Toolbox**. В контекстном меню выберите команду **Additional Controls**, появится одноименное диалоговое окно. В окне для визуализации элемента управления установите флажок в индикаторе, он слева от имени элемента управления. Сделав выбор, окно **Additional Controls** необходимо корректно закрыть (нажать **OK**). Элемент управления окажется на панели **Toolbox**.

Настройка редактора VBA

Для настройки опций редактора VBA используется диалоговое окно **Options**, которое открывается командой **Tools ⇒ Options**. В диалоговом окне **Options** есть четыре вкладки: **Editor**, **Editor Format**, **General** и **Docking**.

Вкладка Editor

На вкладке **Editor** на панели **Code Settings** (Кодовые параметры) находятся следующие опции (рис. 12):

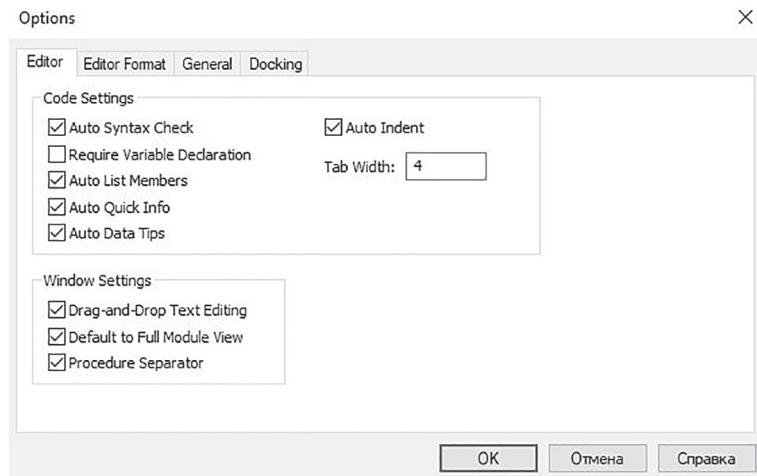


Рисунок 12. Диалоговое окно Options. Вкладка Editor

- 1** **Auto Syntax Check** (Автоматическая проверка синтаксиса) — устанавливает, будет ли выводиться окно с сообщением при обнаружении ошибки в тексте программы. Если флагок установлен, окно с сообщением об ошибке выводится. Если флагок снят, окно не выводится, а ошибка помечается специальным цветом.
- 2** **Require Variable Declaration** (Обязательное объявление переменных) — устанавливает обязательность объявления переменных. Если флагок установлен, в модуль записывается оператор **Option Explicit** (Явное объявление), и объявление всех переменных в этом случае обязательно. Если флагок снят, необъявленным переменным по умолчанию будет присвоен тип данных **Variant**. Если флагок в опции установлен, это означает, что обязательное объявление переменных требуется только в создаваемых модулях. В созданных ранее действуют те правила объявления переменных, которые были при создании этих модулей.
- 3** **Auto List Members** (Автоматическое создание членов списка) — устанавливает автоматическое появление списка свойств, методов и констант при вводе имени объектов. Если флагок установлен, список появляется сразу после того, как ввели последний символ этого объекта и точку, чтобы указать свойство, метод и т. д. Если флагок снят, список не отображается.
- 4** **Auto Quick Info** (Подсказка, Хint) — устанавливает возможность высовчивания подсказки о синтаксисе функции или метода, используемых в программной строке. Если флагок установлен, подсказка выдается. Если сброшен, не выдается.
- 5** **Auto Data Tips** (Значение данных) — устанавливает возможность высовчивания подсказки о состоянии переменных. Если флагок установлен, подсказка выдается. Если снят, не выдается.

6

Auto Indent (Автоматический отступ слева) — устанавливает отступ слева такой же, каким он был в предыдущей строке после нажатия на Enter, то есть при создании новой строки. Если флажок установлен, автоматический отступ слева создается, если снят, не создается.

7

Поле **Tab Width** (Отступ слева) — устанавливает отступ слева в программной строке после нажатия клавиши Tab. По умолчанию отступ составляет 4 символа. Если есть необходимость, число символов отступа слева можно изменить: введите другое число. Для отмены отступа слева нажмите или на клавишу Backspace, если курсор находится справа от отступа, или Delete, если курсор слева от отступа. Или нажмите на Shift + Tab.

На панели **Window Settings** (Параметры окна) находятся следующие опции.

1

Drag-and-Drop Text Editing (Метод *Перетащить и бросить* для текста) — устанавливает для текста возможность перетаскивания методом *Перетащить и бросить*. Выделите фрагмент текста программы, подхватите этот фрагмент левой клавишей и перенесите на новое место. Если флажок установлен, метод *Перетащить и бросить* применять можно. Если сброшен — нельзя.

2

Default to Full Module View (Просмотр всего модуля по умолчанию) — устанавливает возможность просмотра редактируемого модуля. Если флажок установлен, модуль можно просматривать весь (в пределах видимости на экране). Если часть программы модуля не видна, ее можно просмотреть с помощью линеек прокрутки. Если флажок сброшен, просматривать можно только одну процедуру. Для быстрого перехода из одного режима в другой в окне кода (Code) в левой нижней части есть две кнопки: **Procedure View** (Просмотр

процедуры) и **Full Module View** (Просмотр всего модуля), которые позволяют изменять просмотр модуля.

3

Procedure Separator (Разделитель (сепаратор) для процедур) — определяет разделительные линии между процедурами. Если флажок установлен, при создании новой процедуры она отделяется от других (или от другой) горизонтальными разделительными линиями. Они позволяют быстрее сориентироваться в определении границ процедуры. После последней процедуры горизонтальная разделительная линия не создается, ее нижняя граница видна и так.

Вкладка Editor Format

На вкладке **Editor Format** находятся следующие опции (рис. 13):

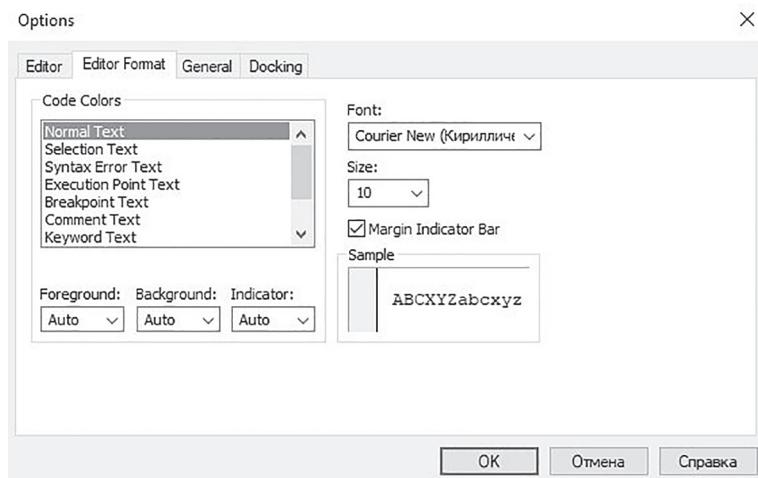


Рисунок 13. Диалоговое окно Options. Вкладка Editor Format

1

линейный список **Code Colors** (Кодовые цвета) — выбирается имя текста, для которого необходимо перенастроить цвета. В VBA используются несколько различных текстов: основной, текст ошибки, текст комментария и т. д. По умолчанию всем уже установлены цвета. Пользователь может их переопределить. Это может понадобиться, например, тем, кто привык

работать с другими языками программирования, где цветовое решение другое;

- 2 раскрывающиеся списки **Foreground** (Основной цвет), **Background** (Цвет фона), **Indicator** (Индикатор) — выбираются цвета для переопределения цвета шрифта, фона и индикатора. Индикаторы — значки на вертикальной индикаторной полосе, расположенной слева от кода. Подробнее см. опцию **Margin Indicator Bar** (Поле индикаторной полосы);
- 3 раскрывающиеся списки **Font** (Шрифт) и **Size** (Размер шрифта) — позволяют выбирать шрифт и размер шрифта, которым будет набираться текст программы. Как правило, для набора программ выбираются равнотолщинные шрифты, так как многие опции настроены именно на такие шрифты (например, на предыдущей вкладке рассматривали опцию **Tab Width** (Отступ слева), в которой отступ измеряется числом символов);
- 4 опция **Margin Indicator Bar** (Поле индикаторной полосы) — предназначена для визуализации вертикальной индикаторной полосы, на которую в ходе отладки выдаются различные метки. Если флажок установлен, полоса видна, сброшен — не видна.

Вкладка General

На вкладке **General** находятся следующие опции (рис. 14):

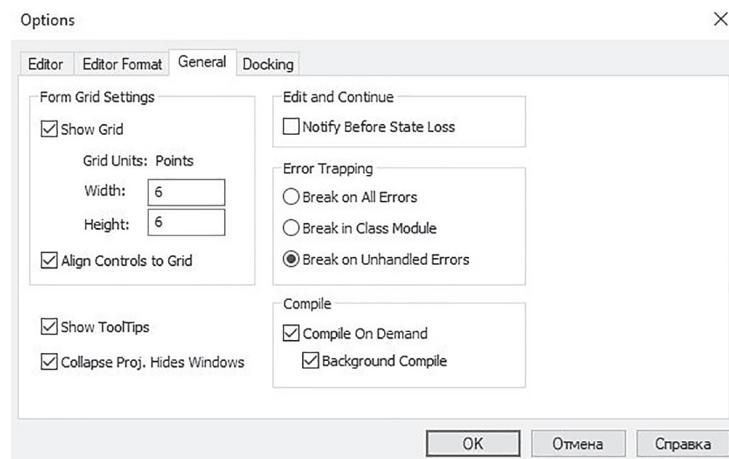


Рисунок 14. Диалоговое окно Options. Вкладка General

На панели **Form Grid Settings** (Параметры сетки формы) находятся опции:

- 1** **Show Grid** (Сетка привязки) — предназначена для визуализации сетки привязки, расположенной на форме. Если флажок установлен, сетка видна, сброшен — не видна. Сетка на форме видна только во время проектирования. Независимо от того, видна сетка или нет, она на форме все равно присутствует. По узлам сетки происходит выравнивание элементов управления. В выходной форме сетка привязки не видна и не действует;
- 2** параметры сетки привязки устанавливаются в текстовых полях **Width** (Ширина) и **Height** (Высота);
- 3** **Align Controls to Grid** (Выравнивание элементов управления на сетке) — выравнивает элементы управления по верхней левой точке сетки привязки. Если флажок установлен, привязка к указанной точке осуществляется автоматически;
- 4** на панели **Edit and Continue** (Редактировать и продолжить) находится опция **Notify Before State Loss**

(Уведомить перед прогоном) — отображение сообщений, вызывающих весь модульный уровень переменных, который должен быть восстановлен для прогона проекта.

На панели **Error Trapping** (Обнаружение ошибок) находятся опции, определяющие действия редактора VBA на нахождение ошибок при компиляции программы:

- 1 Break on All Errors** (Прерывание при всех ошибках) — программа прерывается при обнаружении любых ошибок;
- 2 Break in Class Module** (Прерывание в модуле класса) — любая ошибка, произведенная в модуле класса, заставляет проект вводить способ прерывания на строке кода, который вызвал эту ошибку;
- 3 Break on Unhandled Errors** (Прерывание на ошибках Unhandled) — любая ошибка заставляет проект вводить способ прерывания.

На панели **Compile** (Компиляция) находятся опции:

- 1 Compile On Demand** (Компиляция по запросу) — проект полностью компилируется;
- 2 Background Compile** (Фоновая компиляция) — использование компилирования проекта в фоновом режиме в течение времени выполнения, пока он полностью не завершится.

На вкладке **General** находятся отдельные опции:

- 1 Show ToolTips** (Показывать ToolTips) — отображение ToolTips для инструментальных кнопок;
- 2 Collapse Proj. Hides Windows** (Скрывать окна при свертывании проекта) — автоматически закрывает проект, форму, объект или модульное окно, когда проект попадает в Project Explorer.

Вкладка Docking

На вкладке Docking находятся следующие опции, фиксирующие окна в заданных местах экрана (рис. 15):

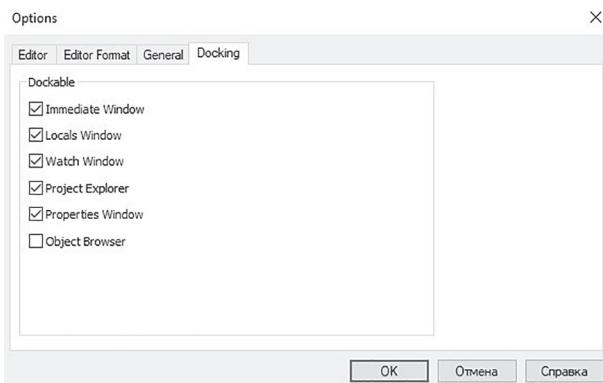


Рисунок 15. Диалоговое окно Options. Вкладка Docking

- 1 **Immediate Window** (Окно Immediate) — фиксирование окна Immediate;
- 2 **Locals Window** (Локальное окно) — фиксирование окна Locals;
- 3 **Watch Window** (Окно Watch) — фиксирование окна Watch;
- 4 **Project Window** (Окно проекта) — фиксирование окна Project;
- 5 **Properties Window** (Окно Properties) — фиксирование окна Properties;
- 6 **Object Browser** (Окно просмотра объектов) — фиксирование окна Object Browser.

Если флагки сброшены с названия соответствующих окон, данное окно располагается в любом месте экрана, то есть становится плавающим, что может создать неудобства при работе с редактором VBA.

4

Синтаксис VBA

Общие положения

Язык Visual Basic for Applications является объектно-ориентированным языком. Это означает, что все элементы рассматриваются как объекты, имеющие определенные свойства. В один объект могут входить другие, рангом ниже. Самым верхним объектом является приложение, например Excel или Word.

Свойства объекта можно рассматривать как его характеристики. Чтобы было понятнее, рассмотрим свойство **Цвет** (Color). Создадим наипростейший пример, в котором будем управлять цветом. Выполните команду **Разработчик** ⇒ **Visual Basic**. В открывшемся редакторе Visual Basic на инструментальной панели **Standard** найдите кнопку с раскрывающимся списком **Insert UserForm** (✉). Раскройте список. Выберите значение **UserForm**. Создана форма, на которую будут помещаться элементы управления.

На форму **UserForm1** перенесите одну надпись (или метку) **Label**, которая примет имя **Label1**, и две **CommandButton**, которые примут имена **CommandButton1** и **CommandButton2**. Элементы управления переносятся с панели **Toolbox**: подхватите элемент с панели **Toolbox** и перетащите на форму. Пока мышь не отпущена, можно перемещать элемент управления по форме. По умолчанию верхний левый угол элемента управления привязывается к одному из узлов сетки на форме, но при перемещении по форме могут притягиваться и другие углы элемента.

Выделите элемент управления **Label1** на форме и в свойстве **Caption**, которое можно найти в диспетчере **Properties** — **Label1**, напишите любую фразу на русском языке, она будет выводиться в надписи при открытии формы. Так как текст фразы безразличен, можете написать слово План, Отчет или Введите число.

Выделив поочередно **CommandButton1** и **CommandButton2**, в свойстве **Caption** напишите на каждой кнопке. Например, OK и Cancel (или + или -, если создается калькулятор).

При выделении формы в диспетчере **Properties** — **UserForm1** также имеется свойство **Caption**. Напишите в этом свойстве имя приложения, например **Наша форма**.

Каждый элемент управления при выделении окружается восемью маркерами. Потянув за маркер, можно уменьшить или увеличить размер элемента управления. Например, слово *План* вполне поместится в существующие границы, а если взять надпись *Отчет по отгрузке подшипников за II квартал* на метке **Label** — нет: в этом случае надпись разместится на нескольких строках. Чтобы уместить ее на одной строке, нужно увеличить горизонтальные размеры элемента управления **Label**: потяните за правый или левый маркер (или за оба маркера поочередно). Определить сразу нужный размер элемента управления довольно трудно, окончательное решение о подгонке его границ принимается после компиляции проекта. Иногда такую подгонку приходится выполнять неоднократно.

При выделении формы вокруг нее также появляются маркеры. Но маркеры формы немного отличаются от маркеров элементов управления. Если маркеры элементов управления — это белые квадратики, то маркеры формы имеют вид как белых, так и черных квадратиков. Разница в том, что белые способны изменять размеры формы, а черные — нет.

В программном коде можно указать имена элементов управления, переменных, функций и т. д. — без учета регистра. В процессе компиляции или проверки эти имена анализируются, и в случае, если они написаны не в соответствии с регистром, исправляются, как только курсор будет установлен в другое место, вне границ этого имени. Первая командная кнопка в нашем приложении будет выводить одну заданную надпись заданного цвета на форму, а вторая — выводить другую заданную надпись другого цвета.

Чтобы программа предоставила место для написания кода и сгенерируала некоторый начальный код, нужно выполнить двойной щелчок левой клавишей по элементу управления, при нажатии на который должна выполняться команда. Чтобы сгенерировать первоначальный код для **CommandButton1**, щелкните по ней дважды. Откроется окно для ввода программного кода, где уже записаны программные строки только что созданной процедуры. Между строк есть мигающий символ курсора. Это означает, что с этой позиции — и начиная с этой строки — можно вводить программный код.

```
Private Sub CommandButton1_Click()
```

```
End Sub
```

Созданные автоматически строки являются границами процедуры. Впишите строки так, чтобы они были расположены между сгенерированным кодом. Создание новой строки производится нажатием на клавишу Enter:

```
Private Sub CommandButton1_Click()
Label1.ForeColor = RGB(0, 255, 255)
Label1.Caption = "Первая программа!!!"
End Sub
```

Для написания обработчика для второй кнопки необходимо также сначала щелкнуть дважды левой клавишей мыши по кнопке и между сгенерированным кодом вписать код:

```
Private Sub CommandButton2_Click()
Label1.ForeColor = RGB(255, 0, 0)
Label1.Caption = "Вторая программа!!!"
End Sub
```

Рассмотрим эти строки.

Во второй строке каждого обработчика в данном примере указывается цвет. Цвет можно задавать либо с помощью специальных констант, либо, как в этом примере, — с помощью цветовой модели RGB. Цветовая модель RGB создает цвет на основе трех цветов: красного (**Red**), зеленого (**Green**) и синего (**Blue**). Значения каждого цвета могут изменяться от 0 до 255. Если задан цвет RGB(255, 0, 0), это чистый красный цвет. Белый задается как RGB(255, 255, 255).

Для работы с элементом управления необходимо указать его имя. Так как необходимо вывести надпись на элемент управления **Label1**, указывается это имя. Особенно внимательно нужно относиться к номерам элементов управления. На форме может быть расположена не одна надпись **Label**, как в этом примере, а несколько. Поэтому необходимо точно указать имя элемента управления, которое включает стандартное имя элемента управления (в нашем случае это **Label**) и номер его на форме

(в нашем случае это 1). Полное имя — **Label1**. Имя нужно указывать полностью: не **Label**, а **Label1**. При ошибочном написании имени элемента управления **Label** будет выдано сообщение об ошибке, так как такого элемента управления в списке **Properties** нет.

Через точку нужно указать свойство, которое необходимо изменить. Так как изменение цвета текста осуществляется с помощью свойства **ForeColor**, оно указывается после точки, справа от **Label1**.

В третьей строке каждого обработчика выводится текст сообщения, которое появляется в элементе управления **Label1** после нажатия соответствующей командной кнопки. Он заключается в парные кавычки (а не две одиночные! Это одна из распространенных ошибок, найти которые сложно). Если в надписи не нужно использовать текст, то есть ввести число символов 0, все равно необходимо указать текст, только нулевой длины.
Запись была бы такой:

```
Label1.Caption = ""
```

Текст нулевой длины все равно заключается в кавычки.

Так как функция RGB используется часто, в таблице 2 приведены наиболее часто встречающиеся сочетания чистых цветов на основе трех основных цветов цветовой модели RGB.

Таблица 2. Основные комбинации цветов

Цвет	Красный (Red)	Зеленый (Green)	Синий (Blue)
Черный	0	0	0
Синий	0	0	255
Зеленый	0	255	0
Голубой	0	255	255
Красный	255	0	0
Пурпурный	255	0	255
Желтый	255	255	0
Белый	255	255	255

После ввода программного кода необходимо проверить наличие или отсутствие ошибок — с помощью команды **Debug** ⇒ **Compile VBAProject**. Для создания готового приложения или выполните команду **Run** ⇒ **Run Sub/UserForm**, или нажмите на одноименную кнопку (▶) на инструментальной панели **Standard**.

Самостоятельно измените цвет и опробуйте работу программы.

Если она работает правильно, закройте скомпилированную форму. В редакторе Visual Basic сохраните проект: либо нажмите на кнопку **Save Книга** (номер), либо выполните команду **File ⇒ Save Книга** (номер). Вместо **Книга** (номер) задайте реальное имя книги. Обратите внимание, что создается не самостоятельное приложение, а Книга в формате Excel.

Типы данных

VBA поддерживает следующие типы переменных (таблица 3).

Таблица 3. Типы данных в VBA

Double	Число с плавающей запятой двойной точности	От -1.79769313486231E308 до -4.94065645841247E-324 для отрицательных значений; от 4.94065645841247E-324 до 1.79769313486232E308 для положительных значений
Integer	Целое число	От 32768 до +32767
Long	Длинное целое число	От 2147483648 до +2147483647
Object	Объект	Любая ссылка на объект
Single	Число с плавающей запятой	От -3.402823E38 до -1.401298E-45 для отрицательных значений; от 1.401298E-45 до 3.402823E38 для положительных значений
String	Строковая переменная	Приблизительно до 65500 (16-разрядный код) или до 2x1032 (32-разрядный код)
Variant	Все типы	Null, Error, числовой Double, String, Object или массив
User-defined	Пользовательский тип	Определяется пользователем

Данные типа **Boolean** могут содержать только значения **True** или **False**. В VBA значению **True** соответствует 1, а **False** — 0. Если переменной этого типа присваивается значение 0, переменная содержит **False**. Все другие значения типа **Boolean** подразумевают **True**. Не считите мою последнюю фразу ошибкой. Действительно, иногда встречаются значения не только 0 или 1, но и 2, и 4 и т. д.

Данные типа **Byte**, **Integer**, **Long** содержат лишь целые цифровые значения из различных диапазонов. Если переменной такого типа присваивается 2,3, то возвращается 2, если 2,9 — возвращается 3.

Данные типа **Currency** служат для представления чисел с плавающей точкой в денежном формате, но число разрядов после десятичного знака ограничено четырьмя. В Excel десятичным символом является запятая, а в редакторе VBA используется точка. Конфликтов в приложениях при этом не возникает, так как используются стандарты России в Excel и стандарты США — в редакторе VBA.

Данные типа **Double** и **Single** содержат числа с плавающей точкой из разных диапазонов значений. Этого достаточно при выполнении денежных расчетов.

Тип **Decimal** может использоваться только внутри переменной типа **Variant**, то есть объявить переменную типа **Decimal** напрямую нельзя. Они позволяют использовать числа с количеством знаков после запятой от 0 до 28, но не могут объявляться непосредственно.

Данные типа **Date** предназначены для обработки информации о дате и времени. Дата и/или время заключается между двумя символами **#**. При вводе данных типа **Date** можно использовать парные кавычки (**"**). При этом подразумевается установленная система формата даты и времени.

Данные типа **Object** служат для хранения других объектов.

Данные типа **String** служат для хранения строк. Каждый символ, сохраненный в переменной типа **String**, занимает один байт памяти. Длина строки в 32-разрядных системах (Windows) — около 1032 символов. Текстовая строка заключается в парные кавычки.

Тип **Variant** является типом по умолчанию. Он устанавливает тип данных в зависимости от содержимого в переменной. Если в такой переменной содержится число или строка, переменная типа **Variant** принимает соответствующий тип данных. Если ее содержимое — число 10, она принимает тип **Integer**; если 1,2 — тип **Double**; если текст — тип **String**. Переменная типа **Variant** изменяет свой тип во время выполнения программы.

Может показаться непонятным, как тип данных **Variant** изменяет свой тип во время работы программы. Поэтому рассмотрим следующие примеры.

В примере в первой процедуре во 2-й и 3-й строках инициализируются переменные **ZZZ** и **XXX**, но типы им не присваиваются. Следовательно, автоматически им присваивается тип **Variant**. Во всех приведенных примерах используется функция **TypeName**, которая возвращает тип переменной, указанной

в круглых скобках. В первом примере используем три переменные, в том числе и **YYY**, которую также не объявляли. Если в первой процедуре попытаемся выяснить тип переменной **YYY**, увидим, что выводится не **Variant**, который ожидали увидеть, а **Integer**. Дело в том, что **Variant** изменяет свое значение и подстраивает его под наиболее подходящий тип данных. Так как здесь используются только целые величины, тип подстраивается как **Integer**.

```
Private Sub CommandButton1_Click()
    ZZZ = 100
    XXX = 200
    YYY = ZZZ * XXX
    TextBox1.Text = YYY
    TextBox2.Text = TypeName(ZZZ)
End Sub
```

В следующем примере явно объявляется переменная **SSS** с типом данных **Variant**. Но если попытаться выяснить тип данных переменной **SSS**, увидим, что он стал **Double**, так как число со знаками после запятой более всего подходит под этот тип.

```
Private Sub CommandButton2_Click()
    Dim SSS As Variant
    SSS = 100.25
    TextBox2.Text = TypeName(SSS)
End Sub
```

В следующем примере явно объявляется переменная **Фамилия** с типом данных **Variant**. Эта переменная инициализируется как текст. Переменная **ZZZ** вообще не объявлена, а только инициализирована целым числом. Следовательно, по умолчанию она получит тип данных **Variant**. Если попытаться определить тип данных в переменных, участвующих в этой процедуре, увидим, что переменная **Фамилия** получила тип **String**, а переменная **ZZZ** получила тип данных **Integer**.

```
Private Sub CommandButton3_Click()
    Dim Фамилия As Variant
    Фамилия = "Швейк Йозеф
    ZZZ = 100
```

```
TextBox1.Text = TypeName(Фамилия)
TextBox2.Text = TypeName(zzz)
End Sub
```

Объявление переменных

Переменная — это именованная область памяти, предназначенная для хранения данных. Необходимо соблюдать следующие правила:

- 1) имя переменной должно начинаться с буквы;
- 2) максимальная длина имени — 255 символов;
- 3) имена могут содержать буквы, цифры и символ подчеркивания (_). Другие символы не допускаются;
- 4) имя переменной не может быть словом, зарезервированным в VBA.

Имя переменной может состоять как из английских букв, так и из русских. Это выгодно отличает язык Visual Basic for Applications от других языков программирования, где неанглийские буквы (кириллические или буквы национальных алфавитов) запрещены в именах переменных.

Регистр в имени переменной не учитывается. Поэтому можно написать: *Zena*, *zena*, *ZeNa*, *ZenA*; *Сумма* и *сумма*. Все эти способы написания имени переменной не вызовут ошибки. Можно, например, при объявлении переменной написать *Итого*, а использовать ее как *итого*. Но это касается только регистра, а не добавления других символов в переменную. Так, если объявили имя переменной как *НачальнаяЦена*, ее можно применять в выражении как *начальнаяцена*, но нельзя как *Начальная_Цена*. В последнем варианте был добавлен символ (_), которого не было при объявлении имени переменной.

Чтобы пользоваться переменными, нужно их объявить. Объявить переменные можно явно и неявно.

Явное объявление переменных осуществляется с использованием оператора **Dim**. Синтаксис объявления переменных следующий:

```
Dim [WithEvents] varname([subscripts]) [As [New]
type] [, [WithEvents] varname([subscripts]) [As
[New] type]] . . .
```

Где:

WithEvents — необязательный аргумент, ключевое слово, которое определяет, что **varname** используется как объектная переменная, реагирующая на события, инициированные объектом Active X. **WithEvents** используется только в модулях класса. Нельзя использовать **New** с **WithEvents**;

varname — обязательный аргумент, имя переменной;

subscripts — необязательный аргумент, длина переменной массива; может быть объявлено до 60 значений. Аргумент списка использует следующий синтаксис:

```
[lower To] upper [, [lower To] upper] . . .;
```

New — необязательный аргумент, ключевое слово, которое подразумевает создание объекта. Если используется **New**, объявляя объектную переменную, новый объект создается при этом в первой ссылке, так что нельзя использовать утверждение **Set** (Набор), чтобы назначать объектную ссылку. Ключевое слово **New** не может быть использовано, чтобы объявлять переменные любого существенного типа данных, не может быть использовано, чтобы объявлять зависимые объекты, и не может быть использовано с **WithEvents**;

type — необязательный аргумент, тип данных переменной; может быть **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (к настоящему времени не поддерживается), **Date**, **String** (для переменной длины строк), **String *** установленной длины (для фиксированной длины строк), **Object**, **Variant**, определенный тип пользователя, или объектный тип. Используйте отдельно слово **As** для каждого типа, для каждой объявляемой переменной.

Рассмотрим несколько примеров явного объявления переменных:

```
Dim Zena As Double  
Dim Kolich As Integer  
Dim Привет As String * 20
```

Синтаксис не запрещает писать на одной строке несколько переменных, но лучше все-таки каждую переменную писать на отдельной строке. Это, во-первых, гораздо нагляднее, во-вторых, позволяет после объявления переменной справа от нее прокомментировать назначение этой переменной: уже через месяц программист забудет назначение переменной, а его преемник и вообще не сможет в этих переменных сразу разобраться.

В синтаксисе объявления переменной не предусмотрена ее инициализация (присвоение ей значения). Это происходит в другой программной строке. Например:

```
Dim Zena As Double  
Zena = 100.25
```

В какой последовательности инициализировать переменные, не имеет значения. Например, можно сначала объявить все, а затем инициализировать их в любой последовательности или объявлять каждую переменную отдельно, а в следующей строке инициализировать ее. Но нельзя сначала инициализировать переменную, а затем объявлять ее. Это вызовет сообщение об ошибке — двойное объявление переменной. Предположим, сначала переменная была инициализирована. В этом случае она автоматически получает и тип данных. Переменная получает тип по умолчанию **Variant**. Если далее попытаться объявить тип этой переменной, получается, что она была объявлена дважды — сначала неявно, а затем явно. В этом случае будет выдано сообщение о двойном объявлении переменной (**Duplicate declaration in current scope**).

Переменные, объявленные с ключевым словом **Dim** на модульном уровне, пригодны для всех процедур в пределах модуля. Переменные, объявленные на уровне процедуры, доступны только в пределах процедуры.

Используйте **Dim** утверждение в модульном или процедурном уровне, чтобы объявлять тип данных переменной. Например, следующее утверждение объявляет переменную как **Integer**:

```
Dim NumberOfEmployees As Integer
```

Также используйте **Dim** утверждение, чтобы объявлять объектный тип переменной. Следующее объявление объявляет переменную как новый рабочий лист:

```
Dim X As New Worksheet
```

Если ключевое слово **New** не используется при объявлении объектной переменной, переменная, имеющая отношение к объекту, должна быть объявлена как существующий объект, использовавший утверждение **Set**. Пока не будет назначен объект, объявленная объектная переменная имеет специальную пустую величину **Nothing**, которая указывает, что она не ссылается на конкретный объект.

Можно также использовать **Dim**-утверждение с пустыми круглыми скобками, чтобы объявлять динамический массив. После того как он объявлен, необходимо использовать утверждение **ReDim** в пределах процедуры, чтобы определять количество элементов в массиве. Если попытаться повторно объявить размер для переменной массива, чей размер был определен явно в **Private**, **Public** или **Dim**-утверждении, будет сгенерирована ошибка.

Если тип данных или объектный тип не указывается и нет утверждения **DefType** в модуле, переменная является **Variant** по умолчанию. Такое объявление типа называется неявным объявлением типа данных.

Рассмотрим следующие примеры.

1. Ошибка в имени переменной:

```
Private Sub CommandButton1_Click()
Dim AAA As Double
AAA = 100
TextBox1.Text = AAB + 25
End Sub
```

Во второй строке этой процедуры объявляется переменная AAA с типом данных **Double**. В третьей строке процедуры переменная AAA инициализируется и получает значение 100. В четвертой строке процедуры в текстовое поле **TextBox1** выводится результат расчета. Но в правой части этой программной строки произошла ошибка, вместо переменной AAA с типом **Double** используется переменная AAB. Происходит следующее: переменная AAB получает тип **Variant** (так как она не была объявлена), а значение переменной AAB инициализируется нулем (так как никакой другой явной инициализации для этой переменной объявлено не было). В итоге получается не 125 с типом **Double**, а 25 с типом **Variant**.

2. Необъявленные переменные:

```
Private Sub CommandButton1_Click()
    ZZZ = 100
    XXX = 200
    YYY = ZZZ * XXX
    TextBox1.Text = YYY
End Sub
```

В данном примере ни в одной переменной тип данных в явном виде не объявлен. В этом случае все они получают тип **Variant**. Переменная YYY первоначально не инициализирована и получает свое значение во время расчета.

3. Объявление перечня переменных:

```
Dim AAA, SSS, DDD, FFF As Double
```

Типичная ошибка. Тип **Double** получит только переменная FFF, так как после ее имени следует ключевое слово **As** и тип этой переменной **Double**, а переменные AAA, SSS, DDD получат тип **Variant**. Правильное объявление типов данных должно быть следующее:

```
Dim AAA As Double
Dim SSS As Double
Dim DDD As Double
Dim FFF As Double
```

Не является ошибкой объявление всех переменных в одной строке:

```
Dim AAA As Double, SSS As Double, DDD As Double, FFF  
As Double
```

Но объявление переменных на отдельных строках более наглядно и удобно для последующей работы с этим приложением.

Пока переменные не инициализированы, числовые переменные инициализированы на 0, переменная длина строки инициализирована на нулевую длину строки («»), и фиксированная длина строки заполнена нулями. Различные переменные инициализированы на **Empty** (Пустой). Каждый элемент переменной типа **Variant** инициализирован, как будто он указан в отдельной переменной.

Когда используется **Dim**-утверждение в процедуре, обычно необходимо помещать его в начало процедуры.

Рассмотрим несколько примеров неправильного объявления имен переменных:

```
55Zena ' начальным символом в переменной должна быть  
буква, а не цифра;  
_Zena ' начальным символом в переменной должна быть  
буква, а не символ подчеркивания;  
Zena& ' имени переменной могут быть только буквы,  
цифры и символ подчеркивания (_);  
Zena Kwartal ' переменная не может включать пробелы,  
поэтому правильнее было бы записать: ZenaKwartal или  
Zena_Kwartal;  
Rem ' в качестве имен переменных нельзя брать  
зарезервированные слова языка Visual Basic for  
Applications. В данном случае была сделана попытка  
взять в качестве имени переменной зарезервированное  
слово Rem, которое обозначает начало комментария  
и является синонимом символа (').
```

Грамматическая ошибка в имени переменной приведет к синтаксической ошибке. Так, если объявлено имя переменной

Zena, а использовано имя *Zent*, переменная *Zent* будет объявлена как тип **Variant**. Но так как инициализации переменной *Zent* может и не быть, она будет принята за 0. Соответственно и выражение, в котором участвует ошибочная переменная *Zent*, может быть равно 0. Поэтому хотя формально сообщения об ошибке не будет, но фактически она появится. Такая ошибка будет не объявлена, о ее наличии трудно будет догадаться. Например, если объявить переменную *Zena* как **Double**, ошибочная переменная *Zent* получит тип **Variant**. А это не одно и то же. Использование типа **Variant** приводит к увеличению времени работы приложения.

Указывать явный тип данных при объявлении не обязательно, он может устанавливаться просто добавлением знака типа к имени переменной.

Не все типы данных располагают собственными знаками. Типы, которые имеют специальные символы, представлены в таблице 4.

Таблица 4. Знаки типов переменных

Тип переменной	Знак	Пример
Integer	%	AAA%
Long	&	Super&
Single	!	Itogo!
Double	#	Zena#
Currency	@	Summa@
String	\$	Name\$

Использовать эти знаки в настоящее время не рекомендуется, так как они есть в VBA только для совместимости с предыдущими версиями.

Примеры объявления переменных со специальными знаками:

```
Private Sub CommandButton2_Click()
Dim Kol%
Kol = 10
Dim Итого_Сумма#
```

```
Итого_Сумма = 25
Dim Фамилия$
Фамилия = "Абулмуслимов Ахмед Азаматович"
TextBox1.Text = (Итого_Сумма * Кол) + 100
TextBox2.Text = Фамилия
End Sub
```

Малейшие ошибки в объявленных и используемых именах переменных приводят к появлению **Variant**. Появление этого типа данных не очень важно, так как неудобства, связанные с более длительным временем обработки программы и несколько большим объемом использования памяти, покрываются удобством автоматического объявления переменных. Но это часто ведет к фатальным ошибкам. В серьезных приложениях нужно стараться отказываться от неявного объявления переменных. Вернемся к одному из рассмотренных ранее примеров:

```
Private Sub CommandButton1_Click()
Dim AAA As Double
AAA = 100
TextBox1.Text = ААВ + 25
End Sub
```

В этой процедуре вместо объявленной переменной ААА используем необъявленную переменную ААВ. Здесь это фатальная ошибка. Программа не сможет обнаружить ошибку в имени переменной. Все подобные расчеты обречены на ошибку (так как вместо исходного значения 100 будет использоваться значение 0), хотя пользователь об этом, возможно, даже не будет знать. Необъявление переменных может повлечь за собой всевозможные ошибки. Для обнаружения ошибок такого рода необходимо отказаться от неявного объявления переменных. Предлагаются два решения:

1) в начало модуля VBA вручную записывать строку:

Option Explicit

Декларация Option Explicit требует обязательного объявления переменных:

- 2) в диалоговом окне **Options** на вкладке **Editor** установить флагок в индикатор **Require Variable Declaration**. После этого добавление строки **Option Explicit** в модуль будет происходить автоматически.

Если хотя бы одна переменная не будет объявлена явным образом, будет выдано об этом сообщение.

Чтобы вновь получить возможность неявного объявления переменных, нужно или вручную удалить строку **Option Explicit** из первой строки модуля, или сбросить флагок из опции **Require Variable Declaration** в диалоговом окне **Options** на вкладке **Editor**.

Язык VBA позволяет использовать другие способы неявного объявления переменных. Например, можно использовать оператор вида **Deftype**.

Его можно использовать на модульном уровне для задания по умолчанию: 1) типов данных для переменных; 2) аргументов возврата в процедуры; 3) процедур возвращаемого типа для **Function** и **Property Get**, чьи имена начинаются с определенных символов.

Синтаксис оператора вида **Deftype** следующий:

```
DefBool letterrange[, letterrange] . . .
DefByte letterrange[, letterrange] . . .
DefInt letterrange[, letterrange] . . .
DefLng letterrange[, letterrange] . . .
DefCur letterrange[, letterrange] . . .
DefSng letterrange[, letterrange] . . .
DefDbl letterrange[, letterrange] . . .
DefDec letterrange[, letterrange] . . .
DefDate letterrange[, letterrange] . . .
DefStr letterrange[, letterrange] . . .
DefObj letterrange[, letterrange] . . .
DefVar letterrange[, letterrange] . . .
```

Необходимый аргумент **letterrange** имеет следующий синтаксис:

```
letter1[-letter2]
```

Где **letter1** и **letter2** — аргументы, определяют диапазон имен, для которых устанавливается тип данных по умолчанию. Каждый аргумент представляет первую букву и последнюю букву переменной, аргумента, процедуры **Function** или имя процедуры **Property Get** и может быть любым символом алфавита. Регистр символов в **letterrange** не имеет значения.

Имя утверждения определяет тип данных (таблица 5).

Таблица 5. Соответствие операторов вида Deftype и типа данных

Оператор вида Deftype	Тип данных
DefBool	Boolean
DefByte	Byte
DefInt	Integer
DefLng	Long
DefCur	Currency
DefSng	Single
DefDbl	Double
DefDec	Decimal (к настоящему времени не поддерживается)
DefDate	Date
DefStr	String
DefObj	Object
DefVar	Variant

Например, в программном фрагменте следующего сообщения объявляется переменная строки:

```
DefStr A-Z
```

Это означает, что ВСЕ переменные в области видимости, начинающиеся с *A* и по *Z* по умолчанию, получают тип STR.

Утверждение **Deftype** влияет только на модуль, где оно использовано.

Если диапазон с *A* по *Z* определен, нельзя изменить переопределение любых подтипов переменных, использовавших утверждения **Deftype**. Как только диапазон определен и если включается ранее определенный символ в другое утверждение

Deftype, генерируется ошибка. Тем не менее можно явно определить тип данных любой переменной, определена она или нет, используя **Dim** утверждение с **As**. Например, можно использовать следующий код на модульном уровне, чтобы определить переменную как **Double**, даже если по умолчанию тип данных является **Integer**:

```
DefInt A-Z  
Dim TaxRate As Double
```

В данном примере все переменные, начинающиеся с **A** по **Z**, по умолчанию объявляются с типом **Integer**. Переменная **TaxRate** тоже входит в этот диапазон, но имеет тип **Double**. Поэтому мы объявляем эту переменную в явном виде.

Утверждения **Deftype** не влияют на элементы определенных типов данных, поскольку элементы должны быть объявлены явно.

Операторы, приведенные в таблице 5, можно использовать только в секции объявления: (**General**) (**Declarations**).

```
DefDb1 A-Z  
Private Sub CommandButton1_Click()  
AAA = 100  
BBB = AAA / 5  
TextBox1.Text = BBB  
TextBox2.Text = TypeName(AAA)  
End Sub
```

Обратите внимание, переменные **AAA** и **BBB** внутри процедуры не объявлены. В итоге тип данных у обеих переменных будет выдан как **Double**, так как было объявлено, что в диапазоне имен с **A** по **Z** необъявленные переменные получают тип **Double**, а не **Variant**.

Чтобы стало более понятным действие утверждения **Deftype**, замените в примере имя переменной **AAA** именем **ФФФ**. Действие будет другое — переменная **ФФФ** получит тип переменной **Integer**, хотя объявляется диапазон с **A** по **Z**, в диапазон попадают только английские символы (с 0 по 127), а русскоязычные символы (с 128 по 255 код символов)

не попадают. Переменная ФФФ получает тип Integer потому, что присваиваемое значение (100) — целое число.

Немного усложним задачу: переменной BBB дадим имя ЯЯЯ. Имя ЯЯЯ также выходит за пределы английского алфавита, поэтому тип переменной ЯЯЯ сначала будет Variant. При выполнении любого математического действия, кроме сложения (+), тип Variant будет преобразован в Double, если при делении получится дробная часть и Integer, если результат деления будет в целых числах.

Рассмотрим другой пример:

```
DefDb1 A-C
Private Sub CommandButton1_Click()
ФФФ = 100
BBB = ФФФ / 5
TextBox1.Text = BBB
TextBox2.Text = TypeName(ФФФ)
End Sub
```

В этом примере тип переменной ФФФ сначала будет Variant, а затем Integer. Дело в том, что был объявлен диапазон действия типа Double с символа А по символ С (оба символа английские), а ФФФ явно выходит за эту область (русские буквы следуют после английских), и поэтому переменная по умолчанию сначала получит тип Variant, а затем будет выбран наиболее близкий к получаемому значению тип (ФФФ = 100, то есть тип Integer).

Кроме оператора Dim при объявлении переменной можно пользоваться ключевыми словами Static, Private и Public.

Локальные переменные

Локальная переменная объявляется в одной из процедур. Границы процедуры ограничены строками:

```
Private Sub CommandButton1_Click()
End Sub
```

Время жизни локальной переменной протекает на уровне процедуры, в которой были объявлены эти переменные. Инициализация таких переменных протекает также в границах одной процедуры и не зависит от инициализации переменных с таким же именем в других процедурах. В качестве примера возьмем следующий программный код:

```
Private Sub CommandButton1_Click()
Dim AAA As Integer
AAA = 100
Dim BBB As Integer
BBB = AAA / 5
TextBox1.Text = BBB
End Sub

Private Sub CommandButton2_Click()
Dim AAA As Integer
AAA = 10
Dim BBB As Integer
BBB = AAA * 5
TextBox1.Text = BBB
End Sub
```

Здесь для событий щелчков по **CommandButton1** и **CommandButton2** написаны две процедуры. В каждой объявляются переменные AAA. В каждой процедуре переменная инициализируется своим числом. В данном примере объявляем переменную AAA как **Integer**, но могли бы объявить в каждой процедуре и с различными типами данных. При нажатии на кнопки **CommandButton1** и **CommandButton2** получаем значения, исходя из инициализации в каждой процедуре. В этом примере переменные AAA в первой процедуре и AAA во второй — совершенно различные переменные и не оказывают друг на друга никакого влияния.

Объявление статических переменных

Статические переменные можно объявлять только в процедуре. При попытке объявить статическую переменную на модульном или на глобальном уровне будет выдано сообщение об ошибке.

```
Private Sub CommandButton1_Click()
Static AAA As Double
AAA = AAA + 1
BBB = AAA / 5
TextBox1.Text = BBB
End Sub
```

Статические переменные могут возвращать значение после окончания выполнения процедуры.

Чем отличается нестатическая переменная от статической?

Рассмотрим пример. Создайте новое приложение. Перенесите на форму две **CommandButton** и два поля **TextBox**. Первая кнопка будет иллюстрировать пример со статической переменной, вторая — с нестатической. Результаты будут выводиться в текстовые поля **TextBox**. Для статической переменной программный код следующий:

```
Private Sub CommandButton1_Click()
Static VVV As Double
VVV = VVV + 1
TextBox1.Text = VVV
End Sub
```

Для нестатической переменной программный код такой:

```
Private Sub CommandButton2_Click()
Dim VVB As Double
VVB = VVB + 1
TextBox2.Text = VVB
End Sub
```

При работе со статической переменной каждый щелчок по **CommandButton1** увеличивает значение переменной на 1.

При работе с нестатической переменной (кнопка **CommandButton2**) значение будет неизменным.

Может сложиться ложное представление, что обычные переменные вообще не могут быть использованы как, например, накопители. Это не так. Вот пример. Объявим переменную **QQQ** как модульную. С каким словом объявлять эту переменную — **Dim**,

Public или **Private** — не имеет никакого значения, так как используется один модуль. Для обработки события (щелчок по **CommandButton1**) создайте следующий программный код:

```
Private QQQ As Currency  
Private Sub CommandButton1_Click()  
Dim ZZZ As Double  
ZZZ = 2  
QQQ = QQQ + ZZZ  
TextBox1.Text = QQQ  
End Sub
```

Переменная **QQQ** объявляется на модульном уровне, поэтому она не включена в процедуру **CommandButton1_Click**. Как видно из примера, переменная **QQQ** способна накапливать текущее значение. Значение переменной **QQQ** после выхода из процедуры **CommandButton1_Click** не удаляется.

Создадим другую процедуру, например **CommandButton2_Click**, где нужно выводить текущее значение переменной **QQQ** после k-того нажатия кнопки **CommandButton1**:

```
Private Sub CommandButton2_Click()  
TextBox2.Text = QQQ  
End Sub
```

Убедитесь, что это действительно так.

Объявление констант

Основное отличие констант от переменных состоит в том, что их значение нельзя изменять в процессе выполнения программы. Они сохраняют значение, присвоенное при разработке. Области видимости для констант определяются так же, как и для переменных. Константы бывают локальные, модульные и глобальные.

Константы объявляют для использования вместо буквальных значений. Например, вместо буквального числа 3,14159265358979323846 можно объявить константу **PI**, которой присваивается значение **П**. Представьте, что в программе

десять раз используется число Π . При использовании числа, а не константы, во-первых, будет потеряна масса времени на ввод такого длинного числа. Во-вторых, рано или поздно можно сделать ошибку при вводе. В-третьих, при изменении константы необходимо в явном виде исправлять все эти значения, а значит, можно ввести ошибку.

Другой пример. Используются тарифные ставки станочников по разрядам. Если использовать числовые значения ставок, при их изменении придется переписывать текст программы, и опять нет гарантий, что не будет ошибок при вводе.

Константы избавляют от этих проблем: число (или текст) присваивается всего один раз, а дальше используется только имя константы.

Синтаксис объявления констант следующий:

```
[Public | Private] Const constname [As type] =  
expression
```

Где:

1. **Public** — необязательный аргумент, ключевое слово использования на модульном уровне для объявления константы, пригодной для всех процедур, во всех модулях. Недопустимо в процедурах;
2. **Private** — необязательный аргумент, ключевое слово использования на модульном уровне для объявления константы, доступной только в пределах модуля, в котором сделана декларация. Не допускается в процедурах;
3. **constname** — обязательный аргумент, имя константы; следует за стандартным присваиванием имен соглашений;
4. **type** — необязательный аргумент, тип данных константы; может быть **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (к настоящему времени не поддерживается), **Date**, **String** или **Variant**. Используйте отдельно **As** для типа каждой объявленной константы;

5. **expression** — обязательный аргумент, Литерал, другая константа или любая комбинация, которая включает любые арифметические или логические операторы, кроме **Is**.

Одновременно с объявлением константе присваивается и значение. Этим константа принципиально отличается от переменной.

Константы закрыты по умолчанию. В пределах процедур константы всегда закрыты; их видимость не может изменяться. В стандартных модулях видимость модульного уровня констант может быть изменена с использованием ключевого слова **Public**. В модулях класса константы могут быть только закрытыми, их видимость не может быть изменена даже с использованием ключевого слова **Public**.

Чтобы объединять несколько постоянных деклараций на той же строке, разделите каждое объявление константы запятой.

Нельзя использовать переменные, определившие функции пользователя или встроенные функции Visual Basic (как, например, **Chr**) в выражениях, назначенных в константах.

В отличие от переменных, константы не могут быть нечаянно изменены во время работы программы. Если явно не объявляется постоянный тип, использовавший **As** тип, константа имеет тип данных, который наиболее подходит для данного выражения.

Константы, объявленные в **Sub**, **Function** или процедуре **Property**, являются локальными в этой процедуре. Константа, объявленная за пределами процедуры, определена для всего модуля, в котором она объявлена. Можно использовать такие константы в любом выражении.

Рассмотрим некоторые варианты объявления констант:

```
Const AAA As Double = 10
Const BBB As Double = 10 / 2
Const QQQ As Double = AAA / BBB
Private Sub CommandButton1_Click()
Dim CCC As Currency
CCC = AAA * 2
```

```
TextBox1.Text = CCC  
TextBox2.Text = QQQ  
End Sub
```

Первые строки объявляют три константы: AAA, BBB и QQQ в разделе модуля (General) (Declarations).

Далее следует процедура, в которой используются эти константы.

Если в процессе работы программы случайно попытаться изменить значение константы, будет выдано сообщение об ошибке (**Assignment to constant not permitted**). Сообщение выдается не при компиляции, а при попытке изменить значение константы, например при нажатии на кнопку, когда делается попытка такого изменения. Возьмем константы из предыдущего примера. Попытаемся присвоить переменной, объявленной константой, другое значение. Перенесите на форму еще одну кнопку **CommandButton2** и напишите для нее следующий обработчик:

```
Private Sub CommandButton2_Click()  
AAA = 100  
End Sub
```

При попытке нажать на **CommandButton2** будет выдано сообщение об ошибке: переменной с константой нельзя изменить значение.

Переменные с одним именем можно объявлять на разных уровнях. Переменная нижнего уровня затеняет переменную верхнего уровня. С константами все обстоит по-другому — они не допускают дублирования своего имени именем переменной на одном уровне. Но если попытаться объявить, например, на модульном уровне константу, а на уровне процедуры — переменную с тем же именем, ошибки выдано не будет. Программа посчитает, что в процедуре действует переменная, а в остальных процедурах — константа с этим же именем.

При дублировании имен константы на разных уровнях происходит затенение константой нижнего уровня константы более высокого уровня.

В языке VBA много предопределенных констант. Их объявлять не нужно, просто вставляйте в программу как готовые значения. В справочной системе смотрите их по запросу Visual Basic Constants.

В свойствах элементов управления — большой выбор констант. Для примера изменим внешний вид в элементе управления **TextBox1** с помощью **CommandButton1**. Обработчик будет следующий:

```
Private Sub CommandButton1_Click()
    TextBox1.SpecialEffect = 6 'fmSpecialEffectBump
End Sub
```

В качестве константы можно выбрать как числовое (в данном примере 6), так и текстовое значение (в данном примере **fmSpecialEffectBump** оформлена в виде комментария).

Создание пользовательских типов

В языке VBA предусмотрена возможность создания пользовательских типов данных. Пользовательские типы — это типы, определенные самим пользователем. Создание пользовательских типов данных — первый и необходимый шаг к созданию объектов.

Например, какой тип данных имеет ручка? Прежде чем ответить, определим параметры ручки: тип чернильного наполнителя, цвет, длина, толщина, вес, наличие надписей на ней, возможность выдвижения чернильного наконечника из корпуса ручки и т. д.

Совершенно понятно, что с одним параметром к ручке не подойти. Она обладает целым рядом параметров, которые должны рассматриваться в целом, например длина и ширина, цвет кнопки и цвет краев, параметры шрифта на этой кнопке и т. д.

Преобразования типов

В ходе работы часто приходится преобразовывать одни типы данных в другие. Для этого предназначены функции преобразования типов. Каждая функция преобразует выражение в специфический тип данных. Синтаксис следующий:

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

Требуемый аргумент выражения — любое выражение строки или числового выражения. Имя функции определяет возвращаемый тип, как показано в таблице 6.

Таблица 6. Функции преобразования типов и их возвращаемые значения

Функция	Возвращаемый тип	Диапазон для аргумента выражения
CBool	Boolean	Любая допустимая строка или числовое выражение
CByte	Bite	От 0 до 255
CCur	Currency	От -922,337,203,685,477.5808 до 922,337,203,685,477.5807
CDate	Date	Любое допустимое выражение даты
CDbl	Double	От -1.79769313486231E308 до -4.94065645841247E-324 для отрицательных значений; от 4.94065645841247E-324 до 1.79769313486232E308 для положительных значений
CDec	Decimal	+/-79,228,162,514,264,337,593,543,950,335 для масштабированных нулевых номеров, то есть числа без десятичных порядков. Для чисел с 28 десятичными порядками диапазон +/-7.9228162514264337593543950335. Минимальное возможное не равное нулю число 0.00000000000000000000000000000001
CInt	Integer	От -32,768 до 32,767; доли округлены

Таблица 6. Функции преобразования типов и их возвращаемые значения. Окончание

Функция	Возвращаемый тип	Диапазон для аргумента выражения
CLng	Long	От -2,147,483,648 до 2,147,483,647; доли округлены
CSng	Single	От -3.402823E38 до -1.401298E-45 для отрицательных значений; от 1.401298E-45 до 3.402823E38 для положительных значений
CStr	String	Возвращает строку от аргумента выражения
CVar	Variant	Тот же диапазон как Double для числовых аргументов. Тот же диапазон как String для нечисловых аргументов

Если выражение, заданное в функцию, находится за пределами получаемого диапазона типа данных, генерируется ошибка.

Верхние и нижние колонтитулы

При создании документов Microsoft Excel часто требуется оформлять верхние и нижние колонтитулы. Они содержат различную служебную информацию: дату распечатки или создания, нумерацию страниц, общее число страниц в документе и т. д. В программах все это можно предусмотреть с помощью специальных кодов.

Следующее специальное форматирование кодов (таблица 7) может быть включено как часть заголовка и свойств верхнего и нижнего колонтитулов (`LeftHeader`, `CenterHeader`, `RightHeader`, `LeftFooter`, `CenterFooter`, `RightFooter`).

Таблица 7. Коды верхнего и нижнего колонтитулов

Format code	Description
<code>&L</code>	Выравнивание по левому краю
<code>&C</code>	Выравнивание по центру
<code>&R</code>	Выравнивание по правому краю
<code>&E</code>	Подчеркивание двойной линией
<code>&X</code>	Надстрочный индекс
<code>&Y</code>	Подстрочный индекс

&B	Полужирное начертание
&I	Курсивное начертание
&U	Подчеркнутое начертание
&S	Зачеркнутое начертание
&D	Вывод даты
&T	Вывод времени
&F	Вывод наименования документа
&A	Наименование рабочей книги
&P	Вывод номера страниц
&P + номер	Печатает номер страницы плюс определенное число
&P - номер	Печатает номер страницы минус определенное число
&&	Печатает единственный амперсанд
& "Имя шрифта"	Печатает символы, которые находятся в определенном шрифте. Не забудьте включать двойные кавычки
&n	Печатает символы, которые последуют в заданном шрифтовом размере. Используйте двухзначное число, чтобы задавать размер в пунктах
&N	Вывод общего числа страниц в документе

Рассмотрим несколько примеров задания параметров в колонтизтулах.

Этот пример печатает дату в верхнем левом углу каждой страницы:

```
Worksheets("Лист1").PageSetup.LeftHeader = "&D"
```

Этот пример печатает дату, номер страницы и всего страниц наверху каждой страницы:

```
Worksheets("Лист1").PageSetup.CenterHeader = "&D  
page &P of &N"
```

Этот пример печатает имя файла в верхнем правом углу каждой страницы:

```
Worksheets("Лист1").PageSetup.RightHeader = "&F"
```

Этот пример печатает номер страницы в левом более низком углу каждой страницы:

```
Worksheets("Лист1").PageSetup.LeftFooter = "&P"
```

Этот пример печатает имя рабочей книги и номер страницы внизу каждой страницы:

```
Worksheets("Лист1").PageSetup.CenterFooter = "&F  
page &P"
```

Этот пример печатает номер страницы в более низком правом углу каждой страницы:

```
Worksheets("Лист1").PageSetup.RightFooter = "&P"
```

Диапазоны в VBA

Речь пойдет о разных понятиях, но они обозначаются одним словом **Range**. Объект **Range** и свойство **Range** — разные, хотя и родственные понятия. Объект **Range** находится в объекте **Worksheet** (Рабочий лист) и представляет собой или одну отдельную ячейку, или набор ячеек на одном рабочем листе. Свойство **Range** является одним из средств ссылки на этот объект. На объект **Range** можно сослаться и посредством двух других свойств: **Cells** и **Offset**. Но наименования этих свойств не совпадают с именем объекта. Поэтому я вынужден был сделать это вступление прежде, чем приступить к изучению свойств объекта-диапазона **Range**.

Свойство Range

Свойство **Range** имеется у следующих объектов:

- Range, Application, Worksheet;
- AllowEditRange;
- AutoFilter, Hyperlink, PivotCell, SmartTag;
- GroupShapes, Shapes.

В каждой из этих групп синтаксис свойства **Range** различный. Нужно четко представлять себе объект, для которого вы хотите использовать это свойство.

Рассмотрим синтаксис этого свойства только для объекта **Range**, хотя он и совпадает с синтаксисом для объектов **Application** и **Worksheet**.

Синтаксис свойства **Range** для объекта **Range** следующий:

expression.Range(*Cell1*, *Cell2*)

Где:

- **expression** — обязательный аргумент, выражение, которое возвращает один из вышеуказанных объектов;
- **Cell1** — обязательный аргумент, тип **Variant**, имя диапазона. Это может быть ссылкой в стиле A1 на языке макроса. Оно может включить оператор диапазона (двоеточие), оператор пересечения (пробел) или оператор союза (запятая). Оно может также включать знаки доллара, но они будут проигнорированы. Вы можете использовать локальное определенное имя в любой части диапазона. Если используете именованную область, она должна быть объявлена на языке макроса;
- **Cell2** — необязательный аргумент, тип **Variant**, ячейка в верхнем левом и нижнем правом углах диапазона. Может быть объектом **Range**, который содержит единственную ячейку, целый столбец или целую строку. Или это может быть строкой (String), которая называет единственную ячейку на языке макроса.

Следующий пример устанавливает величину ячейки A1 на Лист1 в 5,67891:

```
Worksheets("Лист1").Range("A1").Value = 5.67891
```

Этот пример создает формулу в ячейке D1 на Лист1:

```
Worksheets("Лист1").Range("D1").Formula =
"=10*RAND()"
```

Этот пример проводит проверку в ячейках A1:E10 на Лист1. Если одна из ячеек имеет величину менее чем 0,0001, код заменяет эту величину 0 (нуль):

```
For Each AAA In Worksheets("Лист1").Range("A1:E10")
If AAA.Value < 0.0001 Then
```

```
    AAA.Value = 0
End If
Next AAA
```

Этот пример проводит проверку в диапазоне, названном Diapazon, и отображает количество пустых ячеек в диапазоне:

```
ValueRange = 0
For Each AAA In Range("Diapazon")
    If AAA.Value = "" Then      ' поиск пустой ячейки
        ValueRange = ValueRange + 1
    End If
Next AAA
MsgBox "Есть " & ValueRange & " пустых ячеек в этом
диапазоне"
```

Этот пример устанавливает шрифтовой курсивный стиль в ячейках A1:C5 в Лист1:

```
Worksheets("Лист1").Range(Cells(1, 1), Cells(5, 3)).
Font.Italic = True
```

В следующем примере задается одно и то же значение ячайкам в диапазоне A1:C15:

```
ActiveSheet.Range("A1:C15").Value = 100
```

или:

```
Range("A1", "C15") = 100
```

Предполагается, что речь идет о текущем рабочем листе. Если лист не текущий, будет сгенерирована ошибка.

В следующем примере возвращается адрес ячейки, находящейся на пересечении двух диапазонов.

```
Range("D1:D10", "A1:E1") = 100
```

Очевидно, что речь идет о ячейке D1, которая находится на пересечении этих двух диапазонов. Если такого пересечения нет, возвращается ошибка.

В следующем примере указываются четыре ячейки, не связанные друг с другом диапазоном:

```
Range("D1, A5, E7, D10") = 100
```

Во всех примерах мы использовали лишь несколько программных строк. Чтобы превратить эти строки в полноценную процедуру, необходимо только выбрать ей имя. Например, для последнего примера можно использовать событие **Activate** для рабочего листа. В этом случае программный код будет выглядеть следующим образом:

```
Private Sub Worksheet_Activate()
Range("D1, A5, E7, D10") = 100
End Sub
```

Чтобы не было вопросов о возможных значениях аргумента свойства Range, приведем несколько примеров (таблица 8).

Таблица 8. Примеры выбора ячеек, строк, столбцов

Значение	Описание
Range("A1")	Ячейка A1
Range("A1:B5")	Ячейка с A1 по B5
Range("C5:D9,G9:H16")	Выбор нескольких диапазонов
Range("A:A")	Столбец A
Range("1:1")	Строка 1
Range("A:C")	Столбец с A по C
Range("1:5")	Строки с 1 по 5
Range("1:1,3:3,8:8")	Строки 1, 3, и 8
Range("A:A,C:C,F:F")	Столбцы A, C, и F

Свойство Cells

Свойство Cells есть у нескольких объектов: **Range**, **Application**, **Worksheet**. И хотя синтаксис описания этого свойства во всех этих объектах похож, возвращает разные объекты: **Range**, **Application**, **Worksheet** соответственно.

Синтаксис свойства Cells такой:

expression.Cells(NumRow, NumColumn)

expression.Cells(NumCell)
expression.Cells

Где:

- **expression** — обязательный аргумент, выражение, которое возвращает объект **Range**;
- **NumRow** — номер строки;
- **NumColumn** — номер столбца;
- **NumCell** — номер ячейки.

Этот пример устанавливает шрифтовой размер для ячейки D5 на Лист1 в 14 пунктов:

```
Worksheets("Лист1").Cells(5, 4).Font.Size = 20
```

Ячейка D5 находится на пересечении пятой строки и четвертого столбца.

На рабочем листе находится 65 536 строк и 256 столбцов. Поэтому номера строк и столбцов не должны выходить из указанного диапазона.

Этот пример удаляет содержимое из первой ячейки на Лист1:

```
Worksheets("Лист1").Cells(1).ClearContents
```

Первой ячейкой на листе Лист1 является ячейка A1. Если бы мы захотели удалить содержимое третьей ячейки, должны были бы указать это так:

```
Worksheets("Лист1").Cells(3).ClearContents
```

Третьей ячейкой у нас будет ячейка C1.

Номер ячейки лежит в интервале от 1 до 16 777 216. Это значение вычисляется так: 65 536 строк * 256 столбцов. Ячейки нумеруются с ячейки A1 по строкам вправо. Поэтому последний

номер ячейки в первой строке будет 256. Во второй строке первая ячейка будет иметь номер 257 (256 + 1).

Следующий пример устанавливает шрифт Arial и шрифтовой размер для каждой ячейки в Лист1 в 8 пунктов:

```
With Worksheets("Лист1").Cells.Font
    .Name = "Arial"
    .Size = 8
End With
```

Этот пример анализирует ячейки A1:J4 в Лист1. Если ячейка содержит величину менее чем 0,0001, пример заменяет эту величину на 0 (нуль):

```
For rwIndex = 1 To 4
    For colIndex = 1 To 10
        With Worksheets("Лист1").Cells(rwIndex, colIndex)
            If .Value < 0.001 Then .Value = 0
        End With
    Next colIndex
Next rwIndex
```

Этот пример устанавливает шрифтовой курсивный стиль для ячеек A1:C5 в Лист1:

```
Worksheets("Лист1").Activate
Range(Cells(1, 1), Cells(5, 3)).Font.Italic = True
```

Этот пример проверяет столбец данных в именованной области myRange. Если ячейка имеет ту же величину, что и ячейка выше ее, пример отображает адрес ячейки, которая содержит двойные данные:

```
Set r = Range("myRange")
For n = 1 To r.Rows.Count
    If r.Cells(n, 1) = r.Cells(n + 1, 1) Then
        MsgBox "Двойные данные в: " & r.Cells(n + 1,
1).Address
    End If
Next n
```

При использовании третьей синтаксической формы возвращаются все ячейки рабочего листа. Это может потребоваться, например, для очистки содержимого рабочего листа.

Свойство Offset

Свойство **Offset** есть у объектов **Range** и **TickLabels**. Для каждого из этих объектов — свой синтаксис. Мы будем рассматривать это свойство только для объекта **Range**. Синтаксис **Offset** следующий:

```
expression.Offset (RowOffset, ColumnOffset)
```

Где:

- **expression** — обязательный аргумент, выражение, которое возвращает объект **Range**;
- **RowOffset** — обязательный аргумент, тип **Variant**, количество строк (положительное, отрицательное или 0 (нуль)), на которое диапазон должен быть перемещен. Положительные величины — перемещение вниз, отрицательные величины — перемещение вверх. Значение по умолчанию — 0.
- **ColumnOffset** — обязательный аргумент, тип **Variant**, количество столбцов (положительное, отрицательное или 0 (нуль)), на которое диапазон должен быть перемещен. Положительные величины являются перемещением направо, отрицательные — перемещением налево. Значение по умолчанию — 0.

Этот пример активизирует ячейку на три столбца вправо и на три строки от активной ячейки на Лист1:

```
Worksheets("Лист1").Activate  
ActiveCell.Offset(rowOffset:=3, columnOffset:=3).Activate
```

или:

```
Worksheets("Лист1").Activate  
ActiveCell.Offset(3, 3).Activate
```

Обе записи идентичны.

При создании макроса в главе 1 при использовании относительных ссылок мы уже применяли это свойство.

Создадим пример, в котором будет идти заполнение ячеек данными с той ячейки, которая будет активной в момент запуска процедуры. Ячейки будут заполняться данными в столбец. Смещение будет происходить на 1 в столбце и на 0 в строке. Аргументы в круглых скобках будут выглядеть как (1, 0).

```
ActiveCell.FormulaR1C1 = "99"  
ActiveCell.Offset(1, 0).Select  
ActiveCell.FormulaR1C1 = "100"  
ActiveCell.Offset(1, 0).Select  
ActiveCell.FormulaR1C1 = "109"
```

или в упрощенном виде:

```
ActiveCell = 99  
ActiveCell. Offset(1, 0) = 100  
ActiveCell. Offset(2, 0) = 109
```

В последнем случае, как видите, нам приходится определять координаты смещения от исходной ячейки.

Выражения VBA

В выражениях VBA производятся математические вычисления, операции присваивания, логические операции, операции сравнения. В выражениях VBA используются также функции. Так как мы изучаем принципы использования VBA для Excel, необходимо специально отметить, что в VBA нельзя использовать русифицированные имена функций из Excel, так как в VBA есть встроенные функции с английскими именами. Поэтому используются только эти функции.

Операторы VBA

Операторы, используемые в VBA, представлены в таблице 9.

Таблица 9. Операторы VBA

Оператор	Описание оператора
+	Сложение
-	Вычитание
*	Умножение
/	Деление
^	Возведение в степень
&	Конкатенация (складывание или склеивание строк), используется также оператор +
\	Целочисленное деление
Mod	Вычисление остатка от деления одного числа на другое
=	Равно, присваивание
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
<>	Не равно
And	Логическое умножение (конъюнкция) двух выражений
Eqv	Логическая операция эквивалентности двух выражений
Imp	Логическая операция импликации (или следования) двух выражений
Not	Логическое отрицание
Or	Логическое сложение (дизъюнкция) двух выражений
Xor	Логическая операция «исключающее или» над двумя выражениями
Like	Используется для сравнения двух строк
Is	Используется для сравнения двух объектов, ссылающихся на переменную

Большинство операторов знакомо. Рассмотрим малознакомые.

Возведение в степень

Возведение в степень имеет синтаксис:

$$\text{result} = \text{number} ^ \text{exponent}$$

Где:

- **result** — результат, обязательный аргумент, любая числовая переменная;
- **number** — число, обязательный аргумент, любое числовое выражение;
- **exponent** — экспонента (показатель), обязательный аргумент, любое числовое выражение.

Например:

```
Dim MyValue  
MyValue = 2 ^ 2      ' Возвращает 4.  
MyValue = 3 ^ 3 ^ 3      ' Возвращает 19683  
MyValue = (-5) ^ 3      ' Возвращает -125
```

Целочисленное деление

Целочисленное деление имеет синтаксис:

result = number1\number2

Где:

- **result** — результат, обязательный аргумент, любая числовая переменная;
- **number1** — первое число, обязательный аргумент, любое числовое выражение;
- **number2** — второе число, обязательный аргумент, любое числовое выражение.

Например:

```
Dim MyValue  
MyValue = 11 \ 4      ' Возвращает 2.  
MyValue = 9 \ 3      ' Возвращает 3.  
MyValue = 100 \ 3      ' Возвращает 33.
```

Оператор Mod

Оператор Mod — вычисление остатка от деления одного числа на другое имеет синтаксис:

$$\text{result} = \text{number1 Mod number2}$$

Где:

- **result** — результат, обязательный аргумент, любая числовая переменная;
- **number1** — первое число, обязательный аргумент, любое числовое выражение;
- **number2** — второе число, обязательный аргумент, любое числовое выражение.

Например:

```
Dim MyResult
MyResult = 10 Mod 5      ' Возвращает 0
MyResult = 10 Mod 3      ' Возвращает 1
MyResult = 12 Mod 4.3    ' Возвращает 0
MyResult = 12.6 Mod 5    ' Возвращает 3
```

Операторы сравнения

Операторы сравнения имеют следующий синтаксис:

$$\begin{aligned} \text{result} &= \text{expression1 comparisonoperator expression2} \\ \text{result} &= \text{object1 Is object2} \\ \text{result} &= \text{string Like pattern} \end{aligned}$$

Где:

- **result** — результат, обязательный аргумент, любая числовая переменная;
- **expression** — выражение, обязательный аргумент, любое выражение;
- **comparisonoperator** — оператор сравнения, обязательный аргумент, любой оператор сравнения;

- **object** — объект, обязательный аргумент, любое объектное имя;
- **string** — строка, обязательный аргумент, любое выражение строки;
- **pattern** — образец, обязательный аргумент, любое выражение строки или диапазона символов.

Операторы **Like** и **Is** будут рассмотрены в следующих разделах.

Следующая таблица 10 содержит список операторов сравнения и условия, которые определяют результат — **True**, **False** или **Null**.

Таблица 10. Определение результатов сравнения

Оператор	Если True	Если False	Если Null
< (Менее чем)	expression1 < expression2	expression1 >= expression2	expression1 или expression2 = Null
<= (Менее чем или равно)	expression1 <= expression2	expression1 > expression2	expression1 или expression2 = Null
> (Больше чем)	expression1 > expression2	expression1 <= expression2	expression1 или expression2 = Null
>= (Больше чем или равно)	expression1 >= expression2	expression1 < expression2	expression1 или expression2 = Null
= (Равно)	expression1 = expression2	expression1 <> expression2	expression1 или expression2 = Null
<> (Не равно)	expression1 <> expression2	expression1 = expression2	expression1 или expression2 = Null

Например:

```
Dim MyResult, Var1, Var2
MyResult = (45 < 35)      ' Возвращает False.
MyResult = (45 = 45)      ' Возвращает True.
MyResult = (4 <> 3)       ' Возвращает True.
MyResult = ("5" > "4")    ' Возвращает True.

Var1 = "5": Var2 = 4      ' Инициализация переменных.
MyResult = (Var1 > Var2)  ' Возвращает True.
```

```
Var1 = 5: Var2 = Empty  
MyResult = (Var1 > Var2)      ' Возвращает True.
```

```
Var1 = 0: Var2 = Empty  
MyResult = (Var1 = Var2)      ' Возвращает True.
```

При изучении функций мы изучим оператор двоеточие-равно (`:=`), который используется при присваивании значений в функциях (методах).

Операторы конкатенации & и +

И тот и другой операторы используются и для других целей. Например, `&` участвует в объявлении типа `Long`, а `+` используется в операциях сложения.

Операторы конкатенации имеют синтаксис:

```
result = expression1 & expression2  
result = expression1 + expression2
```

Где:

- **result** — результат, обязательный аргумент, любая переменная `String` или `Variant`;
- **expression1** — первая строка, обязательный аргумент, любое выражение;
- **expression2** — вторая строка, обязательный аргумент, любое выражение.

Обратите внимание, что синтаксису конкатенации со знаком плюс (+) все равно, что складывать, — числа или строки. Основным символом конкатенации объявляется символ `&`, но лично мне проще использовать для этого символ `+`, так как он более понятен и легко запоминается.

Например:

```
Dim MyStr  
MyStr = "Здравствуй," & " Вася"      ' Возвращает  
"Здравствуй, Вася".
```

```

MyStr = "Check " & 123 & " Check"      ' Возвращает
"Check 123 Check".
Var1 = "34": Var2 = "6"      ' Инициализация
переменных как строки.
MyNumber = Var1 + Var2      ' Возвращает "346"
(Конкатенация строки).

```

Логический оператор And

Логический оператор **And** имеет синтаксис:

$$\text{result} = \text{expression1 And expression2}$$

Где:

- **result** — результат, обязательный аргумент, любая числовая переменная;
- **expression1** — первое выражение, обязательный аргумент, любое выражение;
- **expression2** — второе выражение, обязательный аргумент, любое выражение (таблица 11).

Таблица 11. Результаты применения оператора And

Если expression1	И expression2	To result
True	True	True
True	False	False
True	Null	Null
False	True	False
False	False	False
False	Null	False
Null	True	Null
Null	False	False
Null	Null	Null

Например:

```

Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null      ' Инициализация
переменных.

```

```

MyCheck = A > B And B > C      ' Возвращает True.
MyCheck = B > A And B > C      ' Возвращает False.
MyCheck = A > B And B > D      ' Возвращает Null.
MyCheck = A And B      ' Возвращает 8 (поразрядное
сравнение).

```

Логический оператор Eqv

Логический оператор Eqv имеет синтаксис:

$$\text{result} = \text{expression1 Eqv expression2}$$

Где:

- **result** — результат, обязательный аргумент, любая числовая переменная;
- **expression1** — первое выражение, обязательный аргумент, любое выражение;
- **expression2** — второе выражение, обязательный аргумент, любое выражение.

Если любое выражение **Null**, результат также **Null**. Если ни одно выражение не **Null**, результат определен согласно следующей таблице 12.

Таблица 12. Результаты оператора Eqv

Если expression1	И expression2	To result
True	True	True
True	False	False
False	True	False
False	False	True

Например:

```

Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null      ' Инициализация
переменных.

```

```

MyCheck = A > B Eqv B > C      ' Возвращает True.
MyCheck = B > A Eqv B > C      ' Возвращает False.
MyCheck = A > B Eqv B > D      ' Возвращает Null.
MyCheck = A Eqv B      ' Возвращает -3 (поразрядное
сравнение) .

```

Логический оператор Imp

Логический оператор **Imp** имеет синтаксис:

$$result = expression1 \text{ Imp } expression2$$

Где:

- **result** — результат, обязательный аргумент, любая числовая переменная;
- **expression1** — первое выражение, обязательный аргумент, любое выражение;
- **expression2** — второе выражение, обязательный аргумент, любое выражение.

Следующая таблица 13 иллюстрирует результат определений оператора **Imp**.

Таблица 13. Результаты оператора Imp

Если expression1	И expression2	To result
True	True	True
True	False	False
True	Null	Null
False	True	True
False	False	True
False	Null	True
Null	True	True
Null	False	Null
Null	Null	Null

Например:

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null      ' Инициализация
переменных.
MyCheck = A > B Imp B > C      ' Возвращает True.
MyCheck = A > B Imp C > B      ' Возвращает False.
MyCheck = B > A Imp C > B      ' Возвращает True.
MyCheck = B > A Imp C > D      ' Возвращает True.
MyCheck = C > D Imp B > A      ' Возвращает Null.
MyCheck = B Imp A      ' Возвращает -1 (поразрядное
сравнение).
```

Логический оператор Not

Логический оператор **Not** имеет синтаксис:

result = Not *expression*

Где:

- **result** — результат, обязательный аргумент, любая числовая переменная;
- **expression** — выражение, обязательный аргумент, любое выражение.

Следующая таблица 14 иллюстрирует результат определений оператора **Not**.

Таблица 14. Результаты оператора Not

Если <i>expression</i>	To <i>result</i>
True	False
False	True
Null	Null

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null      ' Инициализация
переменных.
MyCheck = Not(A > B)      ' Возвращает False.
MyCheck = Not(B > A)      ' Возвращает True.
```

```
MyCheck = Not(C > D)      ' Возвращает Null.  
MyCheck = Not A          ' Возвращает -11 (поразрядное  
сравнение).
```

Логический оператор Or

Логический оператор **Or** имеет синтаксис:

$$\text{result} = \text{expression1 Or expression2}$$

Где

- **result** — результат, обязательный аргумент, любая числовая переменная;
- **expression1** — первое выражение, обязательный аргумент, любое выражение;
- **expression2** — второе выражение, обязательный аргумент, любое выражение (таблица 15).

Таблица 15. Результаты оператора Or

Если expression1	И expression2	To result
True	True	True
True	False	True
True	Null	True
False	True	True
False	False	False
False	Null	Null
Null	True	True
Null	False	Null
Null	Null	Null

Например:

```
Dim A, B, C, D, MyCheck  
A = 10: B = 8: C = 6: D = Null      ' Инициализация  
переменных.  
MyCheck = A > B Or B > C      ' Возвращает True.  
MyCheck = B > A Or B > C      ' Возвращает True.  
MyCheck = A > B Or B > D      ' Возвращает True.
```

```
MyCheck = B > D Or B > A      ' Возвращает Null.
MyCheck = A Or B      ' Возвращает 10 (поразрядное
сравнение) .
```

Логический оператор Xor

Логический оператор **Xor** имеет синтаксис:

$$[result =] expression1 \text{Xor} expression2$$

Где:

- **result** — результат, обязательный аргумент, любая числовая переменная;
- **expression1** — первое выражение, обязательный аргумент, любое выражение;
- **expression2** — второе выражение, обязательный аргумент, любое выражение.

Если одно и только одно выражение оценивается в **True**, результат является **True**. Тем не менее, если любое выражение **Null**, результат также **Null**. Когда ни одно выражение не **Null**, результат определен согласно следующей таблице 16.

Таблица 16. Результаты оператора Xor

Если expression1	И expression2	To result
True	True	False
True	False	True
False	True	True
False	False	False

Например:

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null      ' Инициализация
переменных.
MyCheck = A > B Xor B > C      ' Возвращает False.
MyCheck = B > A Xor B > C      ' Возвращает True.
MyCheck = B > A Xor C > B      ' Возвращает False.
```

```
MyCheck = B > D Xor A > B      ' Возвращает Null.  
MyCheck = A Xor B      ' Возвращает 2 (поразрядное  
сравнение) .
```

Оператор Like

Оператор **Like** имеет следующий синтаксис:

result = *string* Like *pattern*

Где:

- **result** — результат, обязательный аргумент, любая числовая переменная;
- **string** — строка, обязательный аргумент, любое выражение строки;
- **pattern** — образец, обязательный аргумент, любое выражение строки, соответствующее образцу-сопоставлению соглашений, описанному в примечаниях.

Примечания

Если совпадение образца строки, результат является **True**; если нет совпадения, результат **False**. Если или строка, или образец **Null**, результат **Null**.

Поведение оператора **Like** зависит от утверждения Option Compare (Сравнение Выбора). Встроенная сравниваемая строка для каждого модуля — Option Compare Binary (Двоичный код Сравнения Выбора).

Двоичный код сравнения выбора заканчивается сравнениями строки, основанными на правилах сортировки внутренних двоичных представлений символов. Порядок сортировки определен кодовой страницей. В следующем примере показан типичный двоичный порядок сортировки:

A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø

Текст сравнения выбора заканчивается сравнениями строки, основанными на нечувствительности к регистру. Когда вы

сортируете те же символы, использовавшие текст сравнения выбора, производится следующий текстовый порядок сортировки:

$$(A=a) < (\grave{A}=\grave{a}) < (B=b) < (E=e) < (\grave{E}=\grave{e}) < (Z=z) < (\emptyset=\emptyset)$$

Встроенное сопоставление образца обеспечивает разностороннее средство для сравнений строки. Образец для сопоставления характеристик позволяет использовать символы шаблона, литералы в любой комбинации, чтобы соответствовать строкам. Следующая таблица 17 показывает символы, используемые в образцах, и чему они соответствуют.

Таблица 17. Символы, используемые в операторе Like

?	Любой единственный символ
*	Нуль или дополнительные символы
#	Любая единственная цифра (0–9)
[charlist]	Любой единственный символ в charlist
[!charlist]	Любой единственный символ не в charlist

Группа с одним или более символами (charlist), указанная в квадратных скобках ([]), может быть использована, чтобы соответствовать любому единственному символу в строке, и может включать почти любой символьный код, включая цифры.

Чтобы соответствовать специальным символам, квадратную скобку ([]), знак вопроса (?), знак диез, или шарп (#), звездочку (*) указывайте в скобках. Скобка справа ()) не может быть использована в пределах группы, чтобы соответствовать себе, но может быть использована за пределами группы как индивидуальный символ.

Используйте дефис (-), чтобы разделить верхние и нижние границы диапазона, charlist может определить диапазон символов. Например, [A-Z], если соответствующая символьная позиция в строке содержит любые символы верхнего регистра в диапазоне от A до Z.

Другие важные правила для сопоставления образца включают следующее.

- Восклицательный знак (!) в начале charlist означает: в условии задано, что любой символ, за исключением того символа в charlist, может быть обнаружен в строке. Если он использован за пределами скобок, восклицательный знак соответствует себе.
- Дефис (-) может появиться или в начале (после восклицательного знака, если он использован один) или от charlist до сопоставления с собой. В любой другой позиции дефис использован, чтобы идентифицировать диапазон символов.

Когда диапазон символов определен, символы должны появиться в возрастающем порядке сортировки (от самого низкого до самого верхнего). [A-Z] — правильный образец, а [Z-A] — неправильный.

Символьная последовательность [] считается нулевой длиной строки («»).

Например:

```
Dim MyCheck
MyCheck = "aBBBB" Like "a*a"      ' Возвращает True.
MyCheck = "F" Like "[A-Z]"        ' Возвращает True.
MyCheck = "F" Like "[!A-Z]"       ' Возвращает False.
MyCheck = "a2a" Like "a#a"        ' Возвращает True.
MyCheck = "aM5b" Like "a[L-P]#[!c-e]"'
' Возвращает True.
MyCheck = "BAT123khg" Like "B?T*"
' Возвращает True.
MyCheck = "CAT123khg" Like "B?T*"
' Возвращает False.
```

Оператор Is

Оператор Is имеет следующий синтаксис:

183

result = object1 Is object2

Где:

- **result** — результат, обязательный аргумент, любая числовая переменная;

- **object1** — первый объект, обязательный аргумент, любое объектное имя;
- **object2** — второй объект, обязательный аргумент, любое объектное имя.

Если **object1** и **object2** ссылаются на один и тот же объект, результат является **True**; если этого нет, результат **False**.

В следующем примере A ссылается на тот же объект, что и B:

Set A = B

Следующий пример делает A и B ссылающимися на тот же объект, что и C:

Set A = C

Set B = C

Например:

```
Dim MyObject, YourObject, ThisObject, OtherObject,  
ThatObject, MyCheck  
Set YourObject = MyObject      ' Передаваемые  
объектные ссылки.  
Set ThisObject = MyObject  
Set ThatObject = OtherObject  
MyCheck = YourObject Is ThisObject      ' Возвращает  
True.  
MyCheck = ThatObject Is ThisObject      ' Возвращает  
False.  
' Assume MyObject <> OtherObject  
MyCheck = MyObject Is ThatObject      ' Возвращает  
False.
```

Приоритеты операторов

В заключение необходимо напомнить, что операторы выполняются не в порядке их расположения в выражении, а по специальным правилам. Большинство этих правил вам известны, так как они используются и в обычной алгебре, и в высшей математике и т. д.

Каждый оператор имеет свой приоритет выполнения при прочих равных условиях. Эти приоритеты можно вывести в таблицу 18.

Таблица 18. Приоритеты операторов

Оператор	Приоритет	Название оператора
$^$	1	Возведение в степень
*	2	Умножение
/	2	Деление
+	3	Сложение
-	3	Вычитание
&	4	Конкатенация (склеивание)
=	5	Логическое сравнение «равно»
<	5	Логическое сравнение «больше чем»
>	5	Логическое сравнение «меньше чем»
\geq	5	Логическое сравнение «больше чем или равно»
\leq	5	Логическое сравнение «меньше чем или равно»
\neq	5	Логическое сравнение «не равно»

Так как во многих случаях подобные приоритеты могут не устраивать пользователя, их можно менять с помощью круглых скобок. Число круглых скобок может быть неограниченным, но общее число символов в строке ограничено.

Фигурные и квадратные скобки для участия в формулах не допускаются.

Число открывающих круглых скобок должно быть равно числу закрывающих. Если в выражении число открывающих и закрывающих скобок не равно, VBA выдаст сообщение об ошибке и предложит ее исправить.

Свойства объектов

Свойства объектов находятся в окне **Properties** — Имя элемента управления, или объекта.

Так как в ходе программирования мы будем постоянно пользоваться свойствами объекта, необходимо подробно остановиться

на этом понятии. Свойства объекта — это основные характеристики, которыми обладает объект.

Чтобы просмотреть или редактировать свойства объекта, необходимо прежде всего выделить этот объект. Выделение производится либо щелчком левой клавиши по этому объекту в окне **Project**, либо выбирается из списка элементов управления, который можно просмотреть в окне **Properties** (Свойства), если мы работаем с элементами управления. В верхней части этого окна, сразу под системной полосой, есть раскрывающийся список, где находятся все элементы управления, расположенные на форме, и сама форма. Второй способ кажется очень сложным и долгим, но иногда он единственный, чтобы найти нужный элемент управления. Дело в том, что в сложном приложении одна панель с элементами управления может располагаться на другой. Например, если нажата Кнопка1, видна одна панель, а вторая не видна. Если нажата Кнопка2, первая панель не видна, а вторая — видна. В этом случае выделить нижнюю панель (расположенную на панели раньше второй) со всеми элементами управления не представляется возможным.

Следует особо отметить, что в раскрывающемся списке элементов управления и формы, который расположен под системной полосой, находятся только те элементы управления, которые помещены на данную панель. Поясню это на примере. Форма — основная родительская панель для всех элементов управления, расположенных как на самой форме, так и на других панелях. В нашем примере и Панель1, и Панель2 (со всеми элементами управления) будут видны, если предварительно щелкнуть по пустому месту формы, а затем раскрыть список окна **Properties** (Свойства) (рис. 16). Если предварительно выделить одну из панелей, в раскрывающемся списке можно будет видеть не все элементы управления, расположенные на форме, а только те, что расположены непосредственно на выделенной форме, включая и саму выделенную форму. Элементы управления, помещенные на каждую из панелей **Frame** (Рамка) в списке формы, не видны, так как они помещены на форму не прямо, а косвенно, то есть находятся на элементе управления **Frame** (Рамка), а не на форме. Здесь используется такое понятие, как **Parent** (Родитель, или владелец). Форма — владелец всех элементов управления, которые помещены непосредственно

на нее, но она не является прямым владельцем элементов управления, расположенных на других владельцах, например на **Frame** (Рамка). Поэтому элементы управления, расположенные на других владельцах, и не видны в списке элементов управления формы.

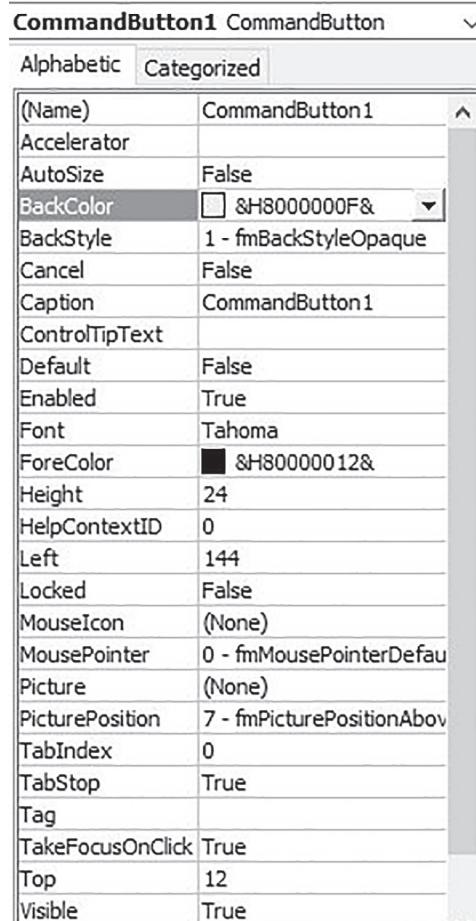


Рисунок 16. Окно Properties – CommandButton1

Окно **Properties** (Свойства) включает выбранный к настоящему времени объект. Видимы только объекты из активной формы.

Окно свойств состоит из двух столбцов: в правом перечислены названия свойств, а в левом — их значения. Редактирование

свойства осуществляется либо вручную (например, ввод имени элемента), либо выбором соответствующего поля из списка, либо при помощи диалогового окна настройки свойства.

Окно свойств состоит из двух вкладок:

- **Alphabetic Tab** — в алфавитном порядке включает все свойства для выбранного объекта, который может быть изменен в проекте, а также их текущее значение. Вы можете изменить установку свойств, выбирая имя свойства или выбирая новое значение;
- **Categorized Tab** — включает все свойства для выбранного объекта категорий. Например, **BackColor**, **Caption**, и **ForeColor** — в категории **Appearance** (Внешний вид). Вы можете сократить список, чтобы видеть категории, можете расширять категорию, чтобы видеть свойства. Когда вы расширяете или уменьшаете список, видите значок плюс (+) или минус (-) слева от имени категории.

Окно **Properties** (Свойства) включает свойства выделенного объекта. Если выделяются сразу несколько элементов управления, в окне **Properties** (Свойства) выводятся только их общие свойства. Такими свойствами, как правило, являются **Enable** (Доступность), **Visible** (Видимость), **MousePointer** (Вид указателя мыши) и т. д.

Свойства книги

Password

Возвращает или устанавливает пароль, чтобы открывать определенную рабочую книгу. Читает/пишет **String**.

Свойство **Password** имеет следующий синтаксис:

expression.Password

Где **expression** — обязательный аргумент, выражение, которое возвращает один из объектов в **Applies**.

В этом примере открытую рабочую книгу сохранили под именем Password.xls, установили пароль для нее, затем книга закрывается. Этот пример устанавливает для файла Password.xls, записанного на Рабочем столе пароль как Parol:

```
Sub UsePassword()
    Dim wkbOne As Workbook
    Set wkbOne = Application.Workbooks.Open("C:\Users\vikto\OneDrive\Рабочий стол\Password.xls")
    wkbOne.Password = "Parol"
    wkbOne.Close
End Sub
```

Свойство **Password** является удобочитаемым и возвращает «*****».

В VBAProject выделите объект *Эта книга* двойным щелчком. Введите текст программы, нажмите на кнопку Run Sub. Нажмите Save (Сохранить). Закройте книгу с сохранением изменений. Откройте книгу Password.xls, в окне Введите пароль (рис. 17) введите Parol. Нажмите на кнопку OK.

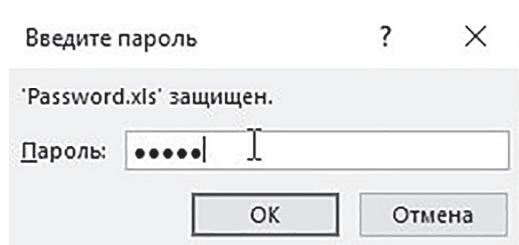


Рисунок 17. Введите пароль

Свойства листа

Рассмотрим лишь некоторые свойства рабочего листа.

Свойство **Visible** определяет видимость рабочих листов. Значение **xlSheetVisible** делает рабочий лист видимым. Значение **xlSheetHidden** скрывает рабочий лист. Но данные по-прежнему доступны для различных расчетов. Например, предположим,

что на Лист3 в ячейках с A1 по A15 находятся данные, которые не должны изменяться случайными лицами. Есть смысл скрыть эти данные. В открытом и видимом Лист1 в ячейках с этими же адресами (с A1 по A15) находятся другие данные, которые являются изменяемыми. Мы можем записать в ячейках с B1 по B15 формулу:

$$=\text{Лист3!A(i)} + \text{Лист1!A(i)}$$

С помощью программы можем управлять изменением этого свойства. Для создания примера перенесите на форму две командные кнопки **CommandButton**. Первая будет скрывать Лист3, а другая — делать этот лист видимым. Перепишите следующий программный код:

```
Private Sub CommandButton1_Click()
Лист3.Visible = xlSheetHidden
End Sub

Private Sub CommandButton2_Click()
Лист3.Visible = xlSheetVisible
End Sub
```

Columns

Возвращает объект **Range**, который представляет все столбцы на определенном рабочем листе. Только для чтения.

Синтаксис этого свойства:

`expression.Columns`

Где **expression** — обязательный аргумент, выражение, которое возвращает объект в **Applies**.

Для информации о возврате единственного элемента набора смотри **Returning an Object from a Collection** (Возвращение Объектов Коллекции).

Использование этого свойства без объектного классификатора — эквивалент использования записи `ActiveSheet.Columns`.

Когда используется для объекта **Range**, который представляет множественную область выбора, это свойство возвращает

столбцы только из первой области диапазона. Например, если объект **Range** имеет две области (A1:B2 и C3:D4 — Selection.Columns.Count), возвращается значение 2, а не 4. Чтобы использовать это свойство в диапазоне, который может содержать множественную область выбора Areas.Count, необходимо определить содержание диапазона более чем в одной области.

Этот пример форматирует шрифт первого столбца (столбец A) на Лист1 как жирный шрифт:

```
Worksheets("Лист1").Columns(1).Font.Bold = True
```

Этот пример устанавливает величину каждой ячейки в первом столбце в именованной области, названной *Zona* в 0 (нуль):

```
Range("Zona").Columns(1).Value = 0
```

Этот пример отображает число выбранных столбцов на Лист1:

```
Private Sub Worksheet_Activate()
Worksheets("Лист1").Activate
areaCount = Selection.Areas.Count
If areaCount <= 1 Then
    MsgBox "Выбор содержит " & _
    Selection.Columns.Count & " столбцов"
Else
    For i = 1 To areaCount
        MsgBox "Область " & i & " выбора содержит" & _
        Selection.Areas(i).Columns.Count & " столбцов"
    Next i
End If
End Sub
```

Rows

Для объекта **Application** возвращает объект **Range**, который представляет все строки в активном рабочем листе. Если активный документ не рабочий лист, свойство **Rows** генерирует ошибку. Для объекта **Range** возвращает объект **Range**, который представляет строки в определенном диапазоне. Для объекта **Worksheet** возвращает объект **Range**, который представляет все строки в определенном рабочем листе. Объект **Range** только для чтения.

Для информации о возврате единственного элемента набора смотри [Returning an Object from a Collection](#) (Возвращение Объектов Коллекции).

Использование этого свойства без объектного классификатора — эквивалент использования записи ActiveSheet.Rows.

Когда это используется для объекта **Range**, который представляет множественную область выбора, свойство возвращает строки только из первой области диапазона. Например, если объект **Range** имеет две области — A1:B2 и C3:D4 — Selection.Rows.Count, то возвращается значение 2, а не 4. Для того чтобы использовать это свойство в диапазоне, который может содержать множественную область выбора Areas.Count, необходимо определить содержание диапазона более чем в одной области.

Этот пример удаляет третью строку на Лист1:

```
Worksheets("Лист1").Rows(3).Delete
```

Следующий пример удаляет строки в текущей области в первом рабочем листе, где величина ячейки в первой строке такая же, как и величина в первой ячейке в предшествующей строке:

```
For Each rw In Worksheets(1).Cells(1, 1).CurrentRegion.Rows
    this = rw.Cells(1, 1).Value
    If this = last Then rw.Delete
    last = this
Next
```

Этот пример отображает число выделенных строк в Лист1:

```
Private Sub Worksheet_Activate()
    Worksheets("Лист1").Activate
    areaCount = Selection.Areas.Count
    If areaCount <= 1 Then
        MsgBox "Выбор содержит " & _
            Selection.Rows.Count & " строк"
    Else
```

```
i = 1
For Each a In Selection.Areas
MsgBox "Область " & i & " выбора содержит " & _
a.Rows.Count & " строк"
i = i + 1
Next a
End If
End Sub
```

PageSetup

Возвращает объект **PageSetup**, который содержит все страницные параметры установки для определенного объекта. Только для чтения.

Этот пример устанавливает текст заголовка в центр для Диаграмма1:

```
Charts("Диаграмма1").PageSetup.CenterHeader = "Отчет
по продажам отдела 13"
```

Свойства диаграммы

Свойство ChartTitle

Возвращает объект **ChartTitle**, который представляет название указанной диаграммы. Только для чтения.

Следующий пример устанавливает текст для названия Диаграмма1:

```
With Charts("Диаграмма1")
.HasTitle = True
.ChartTitle.Text = "Отчет о продажах в I квартале
2023 года"
End With
```

Как видно из этого примера, при создании заголовка очень удобно применять операторные скобки **With-End With**.

Свойство ChartArea

Возвращает объект **ChartArea**, который представляет полную область диаграммы для диаграммы на отдельном листе. Только для чтения.

Следующий пример устанавливает область диаграммы внутреннего цвета Диаграмма1 (красный) и устанавливает граничный цвет (синий):

```
With Charts("Диаграмма1").ChartArea  
.Interior.Color = RGB(255, 0, 0)  
.Border.Color = RGB(0, 0, 255)  
End With
```

Свойство ChartType

Возвращает или устанавливает тип диаграммы. Читает/пишет **xlChartType**

Свойство имеет следующий синтаксис:

```
expression.ChartType
```

Где **expression** — обязательный аргумент, выражение, которое возвращает один из объектов в **Applies**, чтобы идентифицировать его.

Значения свойства **ChartType** можно посмотреть в окне **Properties**.

Свойства формы

Список свойств **UserForm** можно просмотреть в справочной системе по теме **UserForm Object**.

Кроме стандартных свойств, таких как **Caption**, **BackColor**, **Font** и т. д., формы имеют и собственные свойства. Для просмотра свойств формы в окне свойств **Properties** нужно или щелкнуть в пустом месте формы (но не в системной строке), или выбрать форму из списка объектов в окне свойств **Properties**.

Свойство **ActiveControl** идентифицирует и разрешает обработку элементов управления, которые имеют фокус. Свойство **ActiveControl** имеет синтаксис:

```
object.ActiveControl
```

Где **object** — обязательный аргумент, любой объект.

Свойство **ActiveControl** только для чтения и устанавливается в том случае, если выбирается элемент управления в интерфейсе. **ActiveControl** можно использовать как замену управляющего имени при установке методов свойств или вызове.

Свойство **CanUndo** указывает последнее действие пользователя, которое может быть отменено. Свойство **CanUndo** имеет синтаксис:

object. CanUndo

Где **object** — обязательный аргумент, любой объект.

Возвращаемые значения свойства **CanUndo**:

- **True** — последнее действие пользователя может быть отменено.
- **False** — последнее действие пользователя не может быть отменено.

Командой **Undo** можно отменить сразу несколько действий пользователя. Свойство **CanUndo** указывает, что последнее действие может быть отменено.

Свойство **CanRedo** указывает последнюю **Undo** для повторения. Свойство **CanRedo** имеет синтаксис:

object. CanRedo

Где **object** — обязательный аргумент, любой объект.

Возвращаемые значения свойства **CanRedo**:

- **True** — последняя **Undo** может быть повторена;
- **False** — последняя **Undo** не реверсивна, то есть повторить ее нельзя.

Свойство **CanRedo** только для чтения. Следующие действия пользователя иллюстрируют использование свойств **Undo** и **Redo**.

Измените параметры кнопки выбора.

Введите текст в текстовое окно.

Нажмите **Undo**. Текст исчезает из текстового окна.

Нажмите **Undo**. Кнопка выбора возвращается в свою предшествующую установку.

Нажмите **Redo**. Размер кнопки выбора изменяется.

Нажмите **Redo**. Текст появляется снова в текстовом окне.

Свойство **Tag** загружает дополнительную информацию об объекте. Имеет следующий синтаксис:

object.Tag [= String]

Где:

- **object** — обязательный аргумент, любой объект;
- **String** — необязательный аргумент, выражение строки, опознающее объект. Невыполнение является строкой нулевой длины («»).

Свойство **Tag** можно использовать, чтобы назначать строке идентификацию объекта, не влияя на другие значения свойств или атрибутов. Например, можно использовать **Tag**, чтобы проверять тождество формы или элементов управления, используемых в процедуре как переменные.

Свойства элементов управления

Рассмотрим несколько свойств элементов управления, которым мы не уделили особого внимания.

Свойство **ControlSource** идентифицирует использованную позицию данных, чтобы устанавливать или загружать свойство **Value** элемента управления. Свойство **ControlSource** устанавливает области рабочего листа из Microsoft Excel. Имеет следующий синтаксис:

```
object.ControlSource [= String]
```

Где:

- **object** — обязательный аргумент, любой объект;
- **String** — необязательный аргумент, определяет ячейку рабочего листа, связанную со свойством **Value** элемента управления.

Свойство **ControlSource** идентифицирует ячейку или область. Если изменяется значение **Value** элемента управления, изменение автоматически отражается в связанной ячейке или области. Аналогично, если изменяется величина связанной ячейки или области, изменение автоматически отражается в значении **Value** элемента управления. Нельзя определить другой элемент управления для **ControlSource**. Это вызывает ошибку.

Значение по умолчанию для **ControlSource** — пустая строка. Если **ControlSource** содержит значение кроме пустой строки, оно идентифицирует связанную ячейку или область. Содержание этой ячейки или области автоматически копируется в свойство **Value**, когда элемент управления загружен.

Если свойство **Value** — **Null**, ни одно значение не появляется в позиции, идентифицированной **ControlSource**.

Например:

```
Private Sub UserForm_Initialize()  
ListBox1.ColumnCount = 5  
ListBox1.RowSource = "a1:e4"  
ListBox1.ControlSource = "a6"  
ListBox1.BoundColumn = 0  
End Sub
```

Свойство **ControlTipText** определяет текст подсказки, который появляется, если пользователь держит указатель мыши над элементом управления, не совершая никаких действий. Имеет следующий синтаксис:

```
object.ControlTipText [= String]
```

Где:

- **object** — обязательный аргумент, любой объект;
- **String** — необязательный аргумент, текст, который появляется, когда пользователь задерживает указатель мыши над элементом управления.

Свойство **ControlTipText** позволяет предоставлять пользователю информацию об элементе управления на форме во время работы. Может быть установлено в ходе проектирования, но появиться может над элементом управления только во время выполнения. Значение по умолчанию **ControlTipText** является пустой строкой. Когда значение **ControlTipText** установлено в пустую строку, сообщение не выводится.

Свойства объекта Application

Наиболее важными и применяемыми свойствами приложения **Application** являются (таблица 19).

Таблица 19. Свойства объекта Application

Свойство	Описание
ActiveCell	Активная ячейка
ActiveSheet	Активный лист
ActiveWindow	Активное окно
ActiveWorkbook	Активная рабочая книга
RangeSelection	Выделенный диапазон ячеек
Selection	Выделенный объект
ThisWorkbook	Рабочая книга, которая содержит исполняемую процедуру

Свойство ActiveCell

Возвращает объект **Range**, который представляет активную ячейку в активном окне или в определенном окне. Если окно не отображает рабочий лист, это свойство не работает. Только для чтения.

Если вы не определяете объектный классификатор, это свойство возвращает активную ячейку в активное окно. Будьте внимательны: активная ячейка — единственная в текущем выборе. Выбор может содержать более чем одну ячейку, но только одна из них активная. Следующие выражения возвращают активную ячейку:

```
ActiveCell  
Application.ActiveCell  
ActiveWindow.ActiveCell  
Application.ActiveWindow.ActiveCell
```

Этот пример использует окно сообщения, чтобы отображать значение в активной ячейке. Поскольку свойство **ActiveCell** не работает, если активный лист — не рабочий лист, активизируйте Лист1 перед использованием этого примера при изучении свойства **ActiveCell**:

```
Worksheets("Лист1").Activate  
MsgBox ActiveCell.Value
```

Этот пример изменяет параметры шрифта, используемого при форматировании активной ячейки:

```
Worksheets("Лист1").Activate  
With ActiveCell.Font  
.Bold = True  
.Italic = True  
.Color = RGB(255, 0, 0)  
End With
```

Свойство ActiveSheet

Возвращает объект, который представляет активный лист в активной рабочей книге. Возвращает **Nothing**, если ни один лист не активен. Только для чтения.

Если определяется объектный классификатор, это свойство возвращает активный лист в активной рабочей книге. Если рабочая книга появляется более чем в одном окне, свойство **ActiveSheet** может быть другим в другом окне. Этот пример отображает имя активного листа:

```
MsgBox "Имя активного листа: " & ActiveSheet.Name
```

Свойство ActiveWindow

Возвращает объект **Window**, который представляет активное окно. Только для чтения. Возвращает **Nothing**, если нет открытого окна.

Этот пример отображает имя (свойство **Caption**) активного окна:

```
MsgBox "Имя активного окна: " & ActiveWindow.Caption
```

Свойство ActiveWorkbook

Возвращает объект **Workbook**, который представляет рабочую книгу в активном окне. Только для чтения. Возвращает **Nothing**, если нет открытого окна или если окна **Info** или **Clipboard** являются активными.

Этот пример отображает имя активной рабочей книги:

```
MsgBox "Имя активной рабочей книги: " &  
ActiveWorkbook.Name
```

Свойство RangeSelection

Возвращает объект **Range**, который представляет выбранные ячейки на рабочем листе в определенном окне, даже если выбран графический объект. Только для чтения.

Когда графический объект выбран на рабочем листе, свойство **Selection** возвращает графический объект вместо объекта **Range**. Свойство **RangeSelection** возвращает диапазон ячеек, которые были выбраны прежде, чем был выбран графический объект.

Это свойство и свойство **Selection** возвращают идентичные величины, если диапазон ячеек (а не графический объект) выбран на рабочем листе.

Пример отображает адреса выбранных ячеек на рабочем листе в активном окне:

```
MsgBox ActiveWindow.RangeSelection.Address
```

Свойство Selection

Свойство **Selection** объекта Application возвращает выбранный в настоящее время объект на активном листе в активном окне приложения Excel. Если объект не выбран, возвращается значение **Nothing**.

Возвращаемый объектный тип зависит от текущего выбора (например, если ячейка выбрана, это свойство возвращает объект **Range**).

Следующий пример очищает выбранные ячейки на Лист1 (предполагается, что выбор является диапазоном ячеек):

```
Worksheets("Лист1").Activate  
Selection.Clear
```

Этот пример отображает выбор объектного типа Visual Basic:

```
Worksheets("Лист1").Activate  
MsgBox "Объектный тип выбора: " &  
TypeName(Selection)
```

Свойство ThisWorkbook

Возвращает объект **Workbook**, который представляет рабочую книгу, где работает текущий код макроса. Только для чтения.

Используйте это свойство, чтобы ссылаться на рабочую книгу, содержащую код вашего макроса. **ThisWorkbook** — единственный способ сослаться на нее. Свойство **ActiveWorkbook** не возвращает дополнительную рабочую книгу; это возвращает рабочая книга, которая вызывает расширение. Свойство **Workbooks** может получить ошибку, если имя рабочей книги, возможно, изменилось, когда создавалось расширение. **ThisWorkbook** всегда возвращает рабочую книгу, в которой работает код.

Например, чтобы активизировать диалоговый лист, сохраненный в дополнительной рабочей книге, можно использовать следующий программный код:

```
ThisWorkbook.DialogSheets(1).Show
```

Пример закрывает рабочую книгу, которая содержит его программный код. Изменения в рабочей книге, если они есть, не сохраняются:

```
ThisWorkbook.Close SaveChanges:=False
```

Примеры создания экранных форм

Создание формы с панелями, расположенными друг на друге

В этом примере при щелчке одной кнопкой должна быть видима одна панель, вторая — невидима. При щелчке второй кнопкой должна быть видима вторая панель, а первая делается невидимой. Это вполне реальная ситуация. Она может встретиться, например, при анализе какого-то значения, когда при одном значении выбирается одна рамка с набором управляющих элементов, а при других — другая панель с другими элементами управления.

Перенесите на форму две **CommandButton**. В кнопке **CommandButton1** в свойстве **Accelerator** напишите английскую букву *C*. В **CommandButton2** в свойстве **Accelerator** напишите английскую букву *o*. Теперь видно, что на кнопках в имени кнопки один из символов подчеркнут. Это означает, что нажимать на кнопку можно не только мышью. Достаточно нажать на сочетание клавиш Alt и подчеркнутой буквы. Причем совершенно не важно, какой язык реально включен на клавиатуре — английский или русский. Здесь важна сама клавиша, где находятся английские буквы. Например, английская буква *o* находится на клавише с русской буквой *щ*. Поэтому не обязательно для выполнения команды переходить на английский язык.

Перенесите на форму элемент управления **Label** (Надпись), мы будем выводить сообщение о номере открытой панели (или имени этой панели). На рисунках этого не видно, так как в моем приложении эта надпись оказалась в нижней части

формы, я обрезал ее при уменьшении размера формы для экономии места в книге.

Надписи на кнопках и остальных элементах управления вы можете установить свои, так как свойство **Caption** пользователь выбирает по его усмотрению.

Перенесите на форму элемент управления **Frame** (Рамку). Сделайте его размер примерно $1,5 \times 3$ см. Перенесите на рамку элемент управления **Label** (Надпись). В элементе управления **Frame1** в свойстве **Visible** (Видимость) установите значение **False**. При открытии приложения эта панель будет невидима.

Перенесите на форму **Frame** (Рамку). Сделайте его размер примерно $1,5 \times 3$ см. Перенесите на рамку **TextBox** (Текстовое поле). В элементе управления **Frame2** в свойстве **Visible** (Видимость) не меняйте значение, так как при загрузке приложения одна из панелей должна быть открыта.

Выделите **Frame2** и перенесите его на **Frame1** так, чтобы **Frame2** полностью закрыл рамку **Frame1**.

Общий вид проекта будет выглядеть примерно так, как показано на рисунке 18.

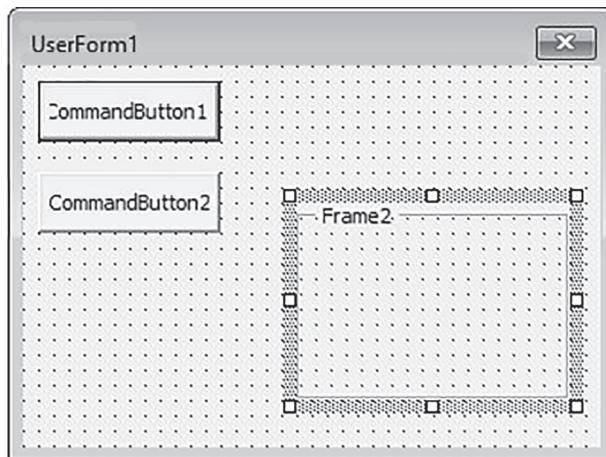


Рисунок 18. Проект многопанельного приложения

Обработчик первой кнопки будет иметь программный код:

```
Private Sub CommandButton1_Click()
Label1.Caption = "Первая панель!"
Frame2.Visible = False
Frame1.Visible = True
End Sub
```

Обработчик второй кнопки будет иметь программный код:

```
Private Sub CommandButton2_Click()
Label1.Caption = "Вторая панель!"
Frame1.Visible = False
Frame2.Visible = True
End Sub
```

Одна из панелей невидима, но это вовсе не означает, что она удаляется из программы или из формы. Она невидима только глазу пользователя. Например, если на невидимой панели происходят таймерные операции, они будут происходить независимо от того, видима эта панель или нет.

Создание приложения с использованием раскрывающегося списка

Элементы управления **ComboBox** и **ListBox** очень часто используются в различных приложениях. Так как они во многом похожи, рассмотрим только один из них — **ComboBox**.

Перенесите на форму элемент управления **ComboBox**. Каждое действие будем выполнять кнопкой **CommandButton**, поэтому вы должны для каждой процедуры добавлять новую кнопку на форму. Для некоторых процедур будем добавлять текстовое поле **TextBox**, но об этом я предупрежу дополнительно.

Первоначальные значения удобно загружать при загрузке самой формы. Для этого удобно использовать событие формы **Initialize**. Программный код можно представить так:

```
Private Sub UserForm_Initialize()
ComboBox1.AddItem "Первая строчка"
```

```
ComboBox1.AddItem "Вторая строчка"  
ComboBox1.AddItem "Третья строчка"  
ComboBox1.AddItem "Четвертая строчка"  
End Sub
```

В списках **ComboBox** и **ListBox** очень широко применяются специально предусмотренные для этого методы. Для добавления новых элементов используется метод **AddItem**.

Синтаксис его следующий:

```
expression.AddItem(Text, Index)
```

Где:

- **expression** — обязательный аргумент, выражение, которое возвращает объект **ControlFormat**;
- **Text** — обязательный аргумент, добавляемый текст в виде **String**;
- **Index** — необязательный аргумент, указывает место вставки в список нового элемента. Если этот аргумент опущен, пункт добавляется в конец существующего списка.

Как правило, первоначальное заполнение списка происходит при загрузке формы, но мы сделаем это нажатием кнопки.

В первой процедуре при щелчке кнопкой в элемент управления **ComboBox1** будут загружаться записи, указанные методом **AddItem**.

```
Private Sub CommandButton1_Click()  
On Error Resume Next  
ComboBox1.AddItem "Первая строка"  
ComboBox1.AddItem "Вторая строка"  
ComboBox1.AddItem "Третья строка"  
ComboBox1.AddItem "Четвертая строка"  
End Sub
```

Для игнорирования ошибок мы в каждую процедуру записываем команду On Error Resume Next, которая заставляет программу работать дальше.

При каждом нажатии на **CommandButton1** в список **ComboBox1** будут добавляться по четыре указанные строки.

Как правило, в список попадают записи, которые вводятся в какое-то окно, например в элемент управления **TextBox**. Чтобы изучить такую возможность, перенесите на форму **TextBox1**. Метод добавления записи будем использовать тот же самый, что и в предыдущем примере, — **AddItem**. Напишите обработчик для **CommandButton2**:

```
Private Sub CommandButton2_Click()
On Error Resume Next
ComboBox1.AddItem TextBox1.Text
End Sub
```

При каждом нажатии на **CommandButton2** в список **ComboBox1** будет добавляться новая запись. Даже если вы случайно нажали на запись дважды, в списке будут созданы две одинаковые записи.

Для удаления всех элементов списка используется метод **Clear**:

```
Private Sub CommandButton3_Click()
On Error Resume Next
ComboBox1.Clear
End Sub
```

Для удаления конкретной записи можно применять **RemoveItem**, которому необходимо задать индекс удаляемой записи из списка. У этого метода следующий синтаксис:

`expression.RemoveItem(Index)`

Где:

- **expression** — обязательный аргумент, выражение, которое возвращает объект **CommandBarComboBox**;

- **Index** — обязательный аргумент, тип **Long**, индекс записи, которую нужно удалить из списка.

Следует помнить, что нумерация индексов начинается с 0. Поэтому первая запись будет иметь индекс 0. Так как постоянно помнить об этом и делать поправки на смещение индекса трудно, можно в нашу формулу добавить -1 . В этом случае задаваемый индекс будет соответствовать номеру записи в списке.

Для следующего примера нам потребуется элемент управления **TextBox2**, в котором мы будем задавать номер индекса удаляемой записи.

Перед тем как удалять какую-то запись из списка, убедитесь, что она существует. Если такой записи не окажется, будет сгенерирована ошибка.

```
Private Sub CommandButton4_Click()
On Error Resume Next
ComboBox1.RemoveItem CInt(TextBox2.Text) - 1
End Sub
```

В поле **TextBox2** задается индекс удаляемой строки. Индекс можно задать только в виде строки, поэтому **String** необходимо преобразовать в другой тип, например в **Integer**. Это мы и делаем с помощью функции **CInt**. Очень многие пользователи никак не могут запомнить, что такое преобразование необходимо.

От заданного в окне **TextBox2** значения отнимаем 1 и получаем реальный индекс. Например, у первой строки в списке индекс = 0. Если мы зададим в поле **TextBox2** единицу, согласно нашей формуле, получим: $1 - 1 = 0$. Это и будет индекс первой строки.

Этот пример заполняет список целыми от 1 до 10.

```
Private Sub CommandButton5_Click()
On Error Resume Next
For x = 1 To 10
    ComboBox1.AddItem x
Next
End Sub
```

Часто в списке необходимо отражать те или иные значения в зависимости от запросов. Это можно делать с помощью переключателей **OptionButton** или индикаторов с флагками **CheckBox**. В нашем примере используем два переключателя **OptionButton**. В зависимости от того, какой переключатель включен, будет загружаться тот или иной список. Перепишите обработчик для командной кнопки:

```
Private Sub CommandButton6_Click()
If OptionButton1 = True Then
    ComboBox1.Clear
    ComboBox1.AddItem "Уголок"
    ComboBox1.AddItem "Швейлер"
    ComboBox1.AddItem "Лист"
    ComboBox1.AddItem "Балка"
Else
    ComboBox1.Clear
    ComboBox1.AddItem "Валенки"
    ComboBox1.AddItem "Галоши"
    ComboBox1.AddItem "Сапоги"
    ComboBox1.AddItem "Лапти"
End If
End Sub
```

Рассмотрим примеры с перемещением строк вверх или вниз. Каждое нажатие на кнопку **Вверх** перемещает выделенную строку на одну строку вверх. Соответственно, каждое нажатие на кнопку **Вниз** перемещает выделенную строку на одну строку вниз.

Обработчик кнопки, перемещающей строку вверх:

```
Private Sub CommandButton7_Click()
With ComboBox1
    NN = .ListIndex
    If NN < .ListCount - 1 And NN <> -1 Then
        TNN = .List(NN + 1)
        .List(NN + 1) = .List(NN)
        .List(NN) = TNN
        .ListIndex = .ListIndex + 1
    End If
End With
End Sub
```

Обработчик кнопки, перемещающей строку вниз:

```
Private Sub CommandButton8_Click()
With ComboBox1
NN = .ListIndex
If NN > 0 Then
TNN = .List(NN - 1)
.List(NN - 1) = .List(NN)
.List(NN) = TNN
.ListIndex = .ListIndex - 1
End If
End With
End Sub
```

Многие программисты дополнительно предусматривают возможность не только перемещаться вверх или вниз щелчком левой клавиши, но и двойным щелчком. Это очень предусмотрительно: многие пользователи торопятся и щелкают левой клавишей очень быстро. Но на любом компьютере предусмотрено время, в течение которого щелчки левой клавишей считаются двойными, поэтому необходимо сделать защиту от торопливых пользователей. Для этого нужно просто написать обработчики для двойных щелчков по кнопкам **Вверх** и **Вниз**:

```
Private Sub CommandButton7_DblClick(ByVal Cancel As
MSForms.ReturnBoolean)
CommandButton7_Click
End Sub
```

и

```
Private Sub CommandButton8_DblClick(ByVal Cancel As
MSForms.ReturnBoolean)
CommandButton8_Click
End Sub
```

Если пользователь желает вводить значения непосредственно в **ComboBox**, он может это делать в специальном поле этого элемента. Программный код обработчика можно представить так:

```
Private Sub CommandButton9_Click()
ComboBox1.AddItem ComboBox1.Value
End Sub
```

Аналогично можем удалять записи из списка, не только указывая индекс этой строки, но и выбирая значение в списке. После выбора значения в списке оно попадает в текстовое поле элемента управления. Обработчик кнопки будет такой:

```
Private Sub CommandButton10_Click()
ComboBox1.RemoveItem ComboBox1.ListIndex
End Sub
```

Очень часто требуется на основе списка создавать другой список. Например, когда пользователь начинает составлять запрос в Microsoft Excel (**Данные ⇒ Импорт внешних данных ⇒ Создать запрос**), после подсоединения к базе данных открывается похожее окно. Рассмотрим такую возможность и мы.

В нашем примере будем переносить выбранные записи из списка **ComboBox** в список **ListBox**. Мы должны предусмотреть и обратный процесс: удаление записей из списка **ListBox**. При переносе выделенных записей из одного списка существует несколько возможностей сохранять записи в разных списках. При переносе из одного списка в другой возможны следующие варианты:

- перенести запись из первого списка во второй и не удалять перенесенную запись из первого списка;
- перенести запись из первого списка во второй и удалить перенесенную запись из первого списка;
- перенести запись из первого списка во второй и сделать запись в первом списке недоступной.

Рассмотрим только первый вариант, потому что все варианты — очень долго. К тому же, научившись удалять запись из списка, вы можете сделать это самостоятельно.

Программный код переноса выделенной записи из одного списка в другой будет такой:

```
Private Sub CommandButton11_Click()
On Error Resume Next
If ComboBox1.ListIndex = -1 Then Exit Sub
For i = 0 To ListBox1.ListCount - 1
If ComboBox1.Value = ListBox1.List(i) Then
Exit Sub
End If
Next i
ListBox1.AddItem ComboBox1.Value
End Sub
```

Программный код удаления выделенной записи из списка будет такой:

```
Private Sub CommandButton12_Click()
On Error Resume Next
If ListBox1.ListIndex = -1 Then Exit Sub
ListBox1.RemoveItem ListBox1.ListIndex
End Sub
```

Значение `ListIndex = -1` бывает в том случае, если ни одна запись в списке не выделена. В этом случае удалять нечего, и процедура завершает работу.

В этом разделе были показаны лишь незначительные способы работы с элементом управления **ComboBox**.

Создание градиентной заливки рабочего листа
Создание градиентной заливки — очень привлекательный способ обратить на приложение дополнительное внимание.

Рассмотрим метод **TwoColorGradient** — двухцветной градиентной заливки.

Синтаксис этого метода следующий:

```
expression.TwoColorGradient(Style, Variant)
```

Где:

- **expression** — обязательный аргумент, возвращаемый объект;
- **Style** — обязательный аргумент, одна из констант **MsoGradientStyle** (таблица 20);
- **Variant** — обязательный аргумент, вариант градиента. Может быть величиной с 1 до 4, соответствующей одному из четырех вариантов на вкладке **Gradient** диалогового окна **Fill Effects**. Если **Style** — **msoGradientFromCenter**, аргумент **Variant** может только быть 1 или 2.

Таблица 20. Константы градиентной заливки MsoGradientStyle

Константа	Описание
msoGradientDiagonalDown	Диагональная заливка из верхнего правого угла в нижний левый угол
msoGradientDiagonalUp	Диагональная заливка из верхнего левого угла в нижний правый угол
msoGradientFromCenter	Заливка от центра к краям
msoGradientFromCorner	Лучевая заливка из верхнего левого угла в нижний правый угол
msoGradientFromTitle	Лучевая заливка от центра
msoGradientHorizontal	Горизонтальная заливка сверху вниз
msoGradientMixed	Равномерная однотонная заливка ForeColor
msoGradientVertical	Вертикальная заливка слева направо

В зависимости от значения аргумента **Variant** виды заливок меняются. Для примера будем использовать значения **Variant = 1**, но в своих примерах вы можете подставлять другие значения **Variant**, чтобы иметь представление об этом аргументе. Перенесите на форму три командные кнопки. При щелчке по первой будем заливать автофигуру горизонтальной градиентной заливкой, а при щелчке по второй — вертикальной градиентной заливкой.

Горизонтальная градиентная заливка:

```
Private Sub CommandButton1_Click()
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle,
0, 0, 300, 350).Fill
```

```
.ForeColor.RGB = RGB(255, 0, 0)
.BackColor.RGB = RGB(0, 0, 255)
.TwoColorGradient msoGradientHorizontal, 1
End With
End Sub
```

Вертикальная градиентная заливка:

```
Private Sub CommandButton2_Click()
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle,
0, 0, 290, 350).Fill
    .ForeColor.RGB = RGB(255, 255, 0)
    .BackColor.RGB = RGB(0, 255, 255)
    .TwoColorGradient msoGradientVertical, 1
End With
End Sub
```

В этих примерах **msoShapeRectangle** — константа, определяющая автофигуру. Полную коллекцию фигур (в том числе и для градиентной заливки) можно просмотреть в приложениях.

Этот пример устанавливает цвет переднего плана, цвет фона, и градиентную заливку для заполнения области диаграммы на **Диаграмма1**. Так как у нас пока нет листа **Диаграмма1**, нужно создать его. Создайте любую диаграмму и поместите ее на отдельном листе с именем **Диаграмма1**. Для третьей кнопки напишите программный код:

```
Private Sub CommandButton3_Click()
With Charts(1).ChartArea.Fill
    .Visible = True
    .ForeColor.SchemeColor = 10
    .BackColor.SchemeColor = 18
    .TwoColorGradient msoGradientHorizontal, 1
End With
End Sub
```

Более простым методом градиентной заливки является метод **OneColorGradient** — одноцветная градиентная заливка, у которой синтаксис:

expression.OneColorGradient(Style, Variant, Degree)

В этом методе, в отличие от **TwoColorGradient**, есть аргумент **Degree** — обязательный аргумент, степень градиента. Может быть величиной от 0.0 (темный) до 1.0 (светлый).

Перенесите на форму четвертую кнопку и напишите программный код для нее:

```
Private Sub CommandButton4_Click()
Set myDocument = Worksheets(1)
With myDocument.Shapes.AddShape(msoShapeRectangle,
0, 0, 300, 350).Fill
    .ForeColor.RGB = RGB(255, 0, 255)
    .OneColorGradient msoGradientFromCorner, 1, 0
End With
End Sub
```

Если **Degree** = 0.5, получается однородная заливка.

Режим конструктора

В режиме конструктора элементы управления можно вставлять не на форму, а непосредственно на рабочий лист книги. Сказать что-либо о достоинствах или недостатках такого размещения элементов не могу, так как это не имеет принципиального значения.

Для входа в режим конструктора необходимо нажать на кнопку Режим конструктора () , которая находится на рибоне Разработчик.

Для выхода из режима конструктора отожмите эту кнопку.

Вставка элементов управления в рабочий лист
Вставка элементов управления на рабочий лист ничем не отличается от их вставки на форму. Единственное требование — элементы управления можно поместить только на самом рабочем листе.

На риббоне **Разработчик** нажмите на треугольную кнопочку на кнопке **Вставить** (). Откроется список элементов управления, расположенных на двух панелях. Мы будем рассматривать только расположенные на панели **Элементы Active X**.

При работе с элементами управления на рабочем листе есть некоторые отличия от элементов управления, расположенных на форме. Если при вставке на форму окно свойств открывается при выделении элемента управления на форме, при вставке на рабочий лист для вызова окна свойств необходимо щелкнуть правой клавишей по элементу управления, свойства которого вы хотите открыть. В контекстном меню выберите команду **Свойства** и щелкните по ней левой клавишей.

Второе отличие в том, что на форме элементы управления всегда доступны для их изменения. На рабочем листе для этого нужно включить режим конструктора.

При выделении элемента управления на рабочем листе вокруг него образуются восемь круглых белых маркеров. Каждый дает возможность изменить размеры элемента управления. Появляющиеся двунаправленные стрелки показывают направления изменений. При перемещении указателя мыши внутрь объекта он принимает вид крестообразной стрелки. Это означает, что элемент управления можно перемещать по поверхности рабочего листа.

При выделении элемента управления на рабочем листе в списке **Имя**, где до этого выводилось имя выделенной ячейки, выводится имя выделенного элемента управления. В строке формул выводится значение:

```
=ВНЕДРИТЬ ("Forms.TextBox.1"; "")
```

Где **TextBox** — тип выделенного элемента управления.

Рассмотрим команды контекстного меню. Команды **Вырезать**, **Копировать**, **Вставить** не будем обговаривать, они вам хорошо известны.

При выборе команды **Свойства** открывается окно **Properties**, ничем не отличающееся от такого же окна при работе с формами.

Команда **Просмотреть код** открывает окно **Code** на том рабочем листе, где создаются элементы управления. Отличие: когда открывается обычный рабочий лист без элементов управления, в списке **Object** окна **Code** имеются два значения — **(General)** и **Worksheet**. Если на этом рабочем листе есть элементы управления, в список попадают и имена элементов управления. При выделении конкретного элемента управления в списке **Object** окна **Code** в другом списке этого же окна — **Procedure** — появляются события выбранного элемента управления. Это логично.

Команда **Объект (Имя_Элемента_Управления)** имеет субкоманду **Edit**, с помощью которой можно изменить текст в элементе управления (**свойства Caption, Text, Value**).

Команды **Группировка** и **Порядок** определяют наборы команд по группировке нескольких элементов и порядку расположения их относительно друг друга. Набор команд **Группировка** доступен только при выделении более чем одного элемента управления.

При выборе команды **Формат объекта** открывается диалоговое окно **Формат элемента управления**, у которого четыре вкладки: **Размер**, **Защита**, **Свойства** и **Веб**.

На вкладке **Размер** можно изменить размеры с помощью полей Высота, Ширина, изменить масштаб отображения элемента управления и т. д. Установка флажка **Сохранить пропорции** автоматически изменяет ширину или высоту при изменении другого параметра.

На вкладке **Защита** есть только один параметр — индикатор с флажком **Защищаемый объект**. Если флажок установлен, при установке защиты рабочего листа становятся защищенными и элементы управления, у которых флажок установлен. Если флажок сброшен, при установленной защите рабочего листа этот элемент управления остается незащищенным. До тех пор, пока рабочий лист не станет защищенным, невозможно установить защиту отдельного элемента управления.

На вкладке **Свойства** есть три переключателя связывания элемента управления с ячейками:

- перемещать и изменять объект вместе с ячейками — элемент управления связывается с расположенными под ним ячейками и перемещается вместе с ними. Если ячейки изменяют размер, и он изменит свой размер;
- перемещать, но не изменять размеры — элемент управления связывается с расположенными под ним ячейками, но не изменяет свои размеры при изменениях размеров ячеек;
- не перемещать и не изменять размеры — элемент управления и ячейки, расположенные под ним, совершенно не зависят друг от друга.

Флажок в индикаторе **Выводить объект на печать** определяет видимость элементов управления при печати рабочего листа. Если флажок установлен, элементы управления печатаются вместе с данными рабочего листа. Если сброшен, элементы управления на печать не выводятся.

На инструментальной панели **Элементы управления** находятся немного меньше элементов управления, чем на **Toolbox** при работе с формой. Здесь нет всех элементов управления, которые являются родителями и контейнерами: **Frame**, **TabStrip**, **MultiPage** и элемента диапазонов **RefEdit**. На инструментальной панели **Элементы управления** есть кнопка **Другие элементы**, с помощью которой можно вставить на рабочий лист дополнительные элементы управления. Внешний вид списка дополнительных элементов немного другой, чем при работе с формами, это не диалоговое окно, а раскрывающийся список.

Для отключения режима конструктора нужно отжать кнопку **Режим конструктора**.

После выхода из режима конструктора ни создавать программный код, ни изменять размеры, ни перемещать элементы управления по поверхности рабочего листа невозможно, так как это рабочий режим, в котором можно пользоваться готовым приложением. Чтобы изменить что-либо (программный код, размеры, местоположение элемента управления), нужно опять нажать на кнопку **Режим конструктора**.

Создание программного кода для элементов управления на рабочем листе

Для создания программного кода на форме достаточно дважды щелкнуть по этому элементу левой клавишей или в контекстном меню элемента управления выбрать команду **Исходный текст**. При создании программного кода для элемента управления, вставленного на рабочий лист, щелкните правой клавишей мыши по этому элементу управления. В контекстном меню выберите команду **Исходный текст**. Будет сгенерирован текст, между которым необходимо вписать программный код по обработке щелчка по данному элементу.

```
Private Sub CheckBox1_Click()
```

```
End Sub
```

Дальнейшие действия ничем не отличаются от действий для создания программного кода при работе с формами.

Пример создания листа с элементами управления

Для примера создадим простейший калькулятор, который будет считывать значения из текстовых окон **TextBox1** и **TextBox2** и выводить готовое число в окно **TextBox3** с полужирным начертанием и красного цвета. Такие программы мы создавали при работе с формами. Ничего нового здесь нет.

```
Private Sub CommandButton1_Click()
    TextBox3.Font.Bold = True
    TextBox3.ForeColor = RGB(255, 0, 0)
    TextBox3.Text = CStr(CDbl(TextBox1.Text) +
        CDbl(TextBox2.Text))
End Sub
```

Как видно из этого примера, программа создается так же, как и при работе с формами. Не забывайте только указывать функции преобразования типов — в противном случае программа может не понять ваших действий и провести операцию конкатенации, а не сложения двух чисел.

Функция диалога MsgBox

Функция **MsgBox** предназначена для вывода самых разнообразных сообщений. **MsgBox** входит в состав Windows, поэтому, какую бы программу вы ни взяли, она наверняка работает с **MsgBox**.

У функции **MsgBox** пять аргументов, из которых четыре необязательные, в формуле они заключены в квадратные скобки.

Синтаксис **MsgBox** следующий:

```
Возвращаемое_значение = MsgBox (Prompt [,Buttons ]  
[,Title ] __[ ,Helpfile, Context ])
```

Где:

- **Prompt** — сообщение, обязательный аргумент; это текст, который отображается в окне сообщения;
- **Buttons** — коды кнопок, которые необходимо отразить в окне сообщения. Значение параметра формируется из нескольких частей, их можно складывать с помощью символа плюс (+);
- **Title** — заголовок, текст, помещаемый в строке заголовка окна сообщения (в системной полосе);
- **Helpfile** — файл справки, имя справочного файла;
- **Context** — контекст, контекстный индекс темы справочной системы.

Коды кнопок представлены в таблице 21.

Таблица 21. Константы функции MsgBox

Константа	Значение	Категория	Описание
vbOKOnly	0	Button	Только кнопка OK
vbOKCancel	1	Button	Кнопки OK и Отмена
vbAbortRetryIgnore	2	Button	Кнопки Стоп, Повторить и Пропустить
vbYesNoCancel	3	Button	Кнопки Да, Нет и Отмена
vbYesNo	4	Button	Кнопки Да и Нет
vbRetryCancel	5	Button	Кнопки Повторить и Отмена
vbCritical	16	Icon	Отображает пиктограмму Critical Message — белый крест в красном кружке
vbQuestion	32	Icon	Отображает пиктограмму Warning Query — синий вопросительный знак в белом кружке
vbExclamation	48	Icon	Отображает пиктограмму Warning Message — черный восклицательный знак в желтом треугольнике
vblnformation	64	Icon	Отображает пиктограмму Information Message — синяя буква i в белом кружке
vbDefaultButton1	0	Default	По умолчанию активна первая кнопка
vbDefaultButton2	256	Default	По умолчанию активна вторая кнопка
vbDefaultButton3	512	Default	По умолчанию активна третья кнопка
vbDefaultButton4	768	Default	По умолчанию активна четвертая кнопка
vbApplicationModal	0	Modal	Модальное диалоговое окно приложения

vbSystemModal	4096	Modal	Модальное диалоговое окно системы: все программы приостанавливаются до тех пор, пока пользователь не ответит на сообщение в окне MsgBox
vbMsgBoxHelpButton	16384	Extras	Дополнительная кнопка для справки
VbMsgBoxSetForeground	65536	Extras	Отображение диалогового окна в новом режиме
vbMsgBoxRight	524288	Extras	Текст выровнен по правому краю
vbMsgBoxRtlReading	1048576	Extras	Текст отображается справа налево (еврейский, арабский)

Аргумент **Buttons** определяет внешний вид функции **MsgBox**. Значение аргумента формируется из нескольких частей, которые можно складывать:

$$\text{Buttons} = \text{Button} + \text{Icon} + \text{Default} + \text{Modal} + \text{Extras}$$

Для категорий аргумента **Button**, **Icon**, **Default** и **Modal** можно использовать только одну из допустимых констант. Для категории **Extras** разрешается применять комбинации значений. В каком порядке вводить константы — безразлично. Категория **Icon** может находиться не только после **Button**, но и перед ним. То же относится и к другим категориям.

MsgBox возвращает по умолчанию некоторые значения. Возвращаемое значение позволяет определить, какую кнопку нажал пользователь. Возвращаемые значения можно просмотреть в таблице 22.

Таблица 22. Возвращаемые значения функции **MsgBox**

Константа	Значение	Описание
vbOK	1	Нажата кнопка OK
vbCancel	2	Нажата кнопка Cancel
vbAbort	3	Нажата кнопка Abort

Таблица 22. Возвращаемые значения функции MsgBox.
Окончание

Константа	Значение	Описание
vbRetry	4	Нажата кнопка Retry
vbIgnore	5	Нажата кнопка Ignore
vbYes	6	Нажата кнопка Yes
vbNo	7	Нажата кнопка No

Рассмотрим выдачу сообщений с помощью функции **MsgBox**. С помощью констант можно создавать конструкции, которые нужны конкретному пользователю. Соединение констант производится с помощью символа плюс (+).

Сообщение не появляется само собой. Это результат какого-либо действия: поиска, подтверждения о закрытии чего-либо (приложения, книги, листа и т. д.), выполнения неразрешенных операций, например деления на нуль и т. д.

Поэтому мы должны создать форму UserForm, перенести на нее CommandButton, при нажатии на которую должно выдаваться сообщение. При выполнении какого-то действия (нажатие кнопки) можно программировать выдачу как одного сообщения, так и нескольких сразу. В последнем случае сообщения будут выведены на экран не все сразу, а по одному — в том порядке, как они записаны в программе. После выбора варианта кнопки в первом сообщении будет выведено второе сообщение и т. д. Правда, таких множественных сообщений следует избегать, так как может получиться ошибка выполнения относительно времени.

Мы будем изучать создание сообщений, подробно комментируя.

Внимание! Учебник имеет границы по ширине, поэтому длинная программная строка в нем переносится на следующую строку. В тексте программы это делается с помощью оператора _ (символ подчеркивания). Программная строка переносится на следующую только с этим оператором.

Прежде всего, давайте вспомним синтаксис функции **MsgBox**: перед ее именем должно находиться возвращаемое значение. В качестве него можно использовать любое слово. В таких

случаях я всегда говорю своим ученикам: «Посмотрите на потолок, там все написано». В данном случае это действительно так. Вы можете взять любое слово или бессвязный набор символов. Сообщение можно написать английскими символами и русскими. После возвращаемого значения находится знак равенства (=) и имя функции **MsgBox**. Все аргументы **MsgBox** — в круглых скобках. Каждый отделяется от другого запятой. Если аргумент один, его разрешается использовать без круглых скобок, но обязательно в кавычках, если это текст.

Мы должны написать некоторое сообщение. Например, «Действительно хотите выйти?» Так как это строка, мы заключаем ее в двойные кавычки.

В первом сообщении создадим две кнопки: **Да** и **Нет**. Для этого необходимо выбрать константу **vbYesNo**. Так как в сообщении вопрос, можно вывести пиктограмму с вопросительным знаком. Среди констант есть единственное значение с вопросительным знаком: это **vbQuestion**. Константы соединяются между собой символом плюс (+).

Наконец, в заголовок сообщения помещаем текст «Закрытие файла». Текст в заголовке, как и текст сообщения, заключается в двойные кавычки. Остальные два аргумента, поскольку они не обязательные, рассматривать здесь не будем.

Наша программная строка готова, ее можно компилировать (рис. 19).

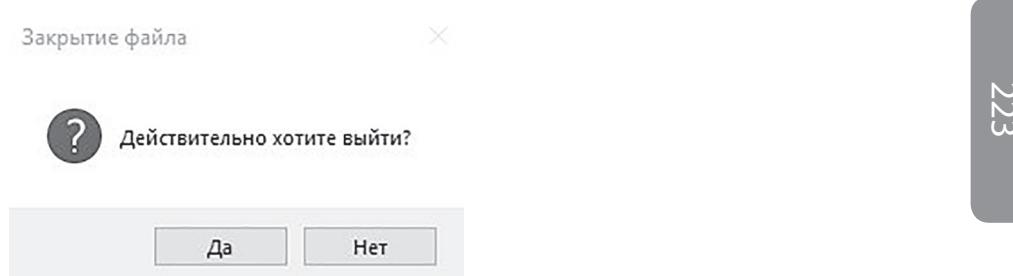


Рисунок 19. Окно выхода из приложения

```
ret = MsgBox("Действительно хотите выйти?", vbYesNo  
+ vbQuestion, "Закрытие файла")
```

Обратите внимание на несколько маленьких особенностей: по умолчанию активна всегда первая кнопка слева; невозможно закрыть окно с сообщением при щелчке по кнопке **Закрыть**, расположенной на системной полосе. Закрыть окно с сообщением можно только при выборе кнопки в окне **Да** или **Нет**. Кнопка **Закрыть** недоступна, что видно даже по ее цвету: она затенена. В следующих примерах мы рассмотрим активизацию не первых кнопок слева, а других, и рассмотрим порядок включения кнопки **Закрыть**.

В следующем примере закрепим наши знания. Тексты сообщения и заголовка не будем даже рассматривать: они создаются аналогично. В наборе кнопок замена всего одной константы: **vbQuestion** (вопросительный знак) — на **vbInformation** (восклицательный знак). Пиктограммы **vbQuestion**, **vbInformation** и другие не обязательны, но они несут дополнительную информацию для пользователя. В этом примере кнопка **Закрыть** по-прежнему недоступна. Возвращаемое значение в этом примере представляет набор символов. Если в первом примере символы **ret** еще можно принять за какое-то очень нужное слово, то символы **fff** показывают, что это может быть просто бессмысленный набор символов (рис. 20).

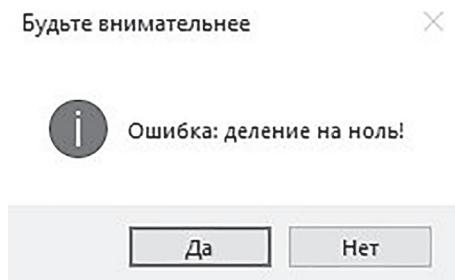


Рисунок 20. Окно с предупреждением

```
fff = MsgBox ("Ошибка: деление на ноль!", vbYesNo +  
vbInformation, "Будьте внимательнее")
```

В этом примере мы обозначили возвращаемое значение русскими символами (сообщение) (рис. 21). Константа **vbAbortRetryIgnore** ничем не отличается принципиально

от **vbYesNo**. В этой константе три кнопки, и у них другие названия. С константой **vbInformation** мы знакомы. Константа **vbDefaultButton3** делает активной (то есть выделенной) третью кнопку слева. Это кнопка **Пропустить**. Кнопка **Закрыть** по-прежнему недоступна.

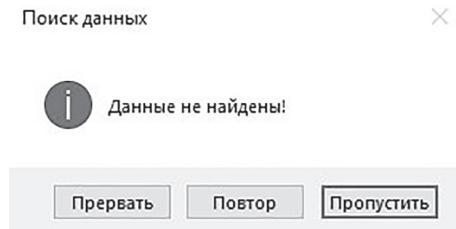


Рисунок 21. Информационное сообщение с третьей активной кнопкой

```
сообщение = MsgBox("Данные не найдены!",
vbAbortRetryIgnore + vbInformation +
vbDefaultButton3, "Поиск данных")
```

В этом примере возвращаемое значение мы назвали *вопрос* (рис. 22), что показывает: это непринципиально. С константами **vbOKCancel** и **vbDefaultButton2** мы знакомы — они выводят кнопки **OK** и **Отмена** и делают активной вторую кнопку слева, то есть кнопку **Отмена**.

Константа **vbMsgBoxHelpButton** создает на панели сообщения кнопку **Справка**. Константа **vbMsgBoxSetForeground** делает доступной кнопку **Закрыть** на системной полосе сообщения.

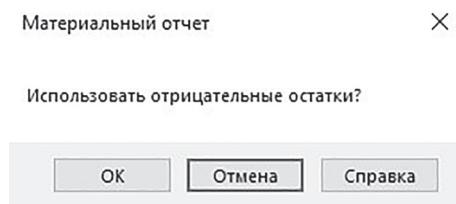


Рисунок 22. Сообщение в фоновом режиме со второй выделенной кнопкой

```
вопрос = MsgBox("Использовать отрицательные
остатки?", vbOKCancel + vbDefaultButton2 +
vbMsgBoxHelpButton + vbMsgBoxSetForeground,
"Материальный отчет")
```

В рассмотренных примерах мы использовали имена констант. Вместо них можно использовать числовое значение (рис. 23). Например:

```
ззз = MsgBox("Вы выполнили недопустимое действие", 2
+ 64 + 65536 + 512, "Будьте внимательнее")
```

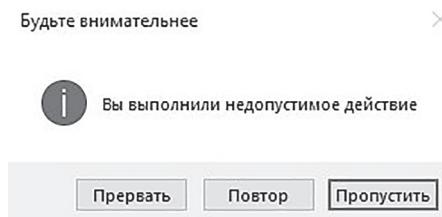


Рисунок 23. Сообщение, созданное числовыми значениями

Сообщения, полученные константами и числовыми значениями, не отличаются друг от друга.

В примерах мы суммировали набор констант (или числовых значений) с помощью символа плюс (+). Вместо плюса можно использовать ключевое слово **Or**. Например:

```
vbOKCancel Or vbDefaultButton2 Or
vbMsgBoxSetForeground
```

Бывают сообщения, текст которых занимает много места. Поэтому логично иметь возможность разбивать одну длинную строку на несколько коротких. Для этого нужно воспользоваться константой **vbCrLf**. Разбитые строки соединяются между собой с помощью оператора конкатенации & (или +). Разработаем следующий пример:

```
ФФФ = "Использовать" & vbCrLf
ФФФ = ФФФ & "отрицательные" & vbCrLf
ФФФ = ФФФ & "остатки?"
```

```
ллл = MsgBox(ффф, vbOKCancel + vbCritical  
+ vbDefaultButton2 + vbMsgBoxHelpButton +  
vbMsgBoxSetForeground, "Материальный отчет")
```

В этом примере длинное сообщение получило имя *ффф*. Оно разбивается на три короткие строки. В первой — начало сообщения. Во второй строке к *ффф* приклеиваются предыдущая *ффф* и продолжение сообщения. Все части соединяются оператором &. В третьей строке происходит то же самое, только константа **vbCrLf** не нужна. В третьей строке в левой части выражения находится полный текст сообщения. Этот текст и помещается в функцию **MsgBox** в качестве первого аргумента.

Функция InputBox

Функция **InputBox** принадлежит языку VBA, метод принадлежит Excel.

InputBox предназначена для ввода одного аргумента, например даты, или чтобы указать ячейку, с которой необходимо начать расчет, или имя книги, которую необходимо открыть, и т. д. На форме две кнопки: **OK** и **Cancel**.

Функция вызова окна **InputBox** имеет следующий синтаксис:

```
Возвращаемое_значение = InputBox((prompt [,title ] [,default ]  
[_[ ,pos_x ] [,pos_y ] [,helpfile, context ]])
```

Где:

- **prompt** — сообщение, обязательный аргумент, текст, отображаемый в окне. Максимальная длина подсказки — приблизительно 1024 символа, в зависимости от ширины используемых символов;
- **title** — заголовок, необязательный аргумент, текст, который появляется в строке заголовка окна;
- **default** — значение по умолчанию, необязательный аргумент, отображается в окне ввода по умолчанию;

- **pos_x** и **pos_y** — необязательные аргументы, координаты верхнего левого угла окна ввода на экране;
- **helpfile** и **context** — файл справки и контекст, необязательные аргументы.

Функция **InputBox** всегда возвращает текстовую строку (рис. 24). Если вводите число или дату, необходимо преобразовать строку в требуемые типы данных.



Рисунок 24. Форма окна InputBox

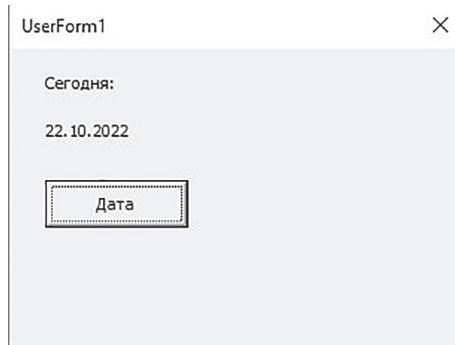


Рисунок 25. Дата выводится на форму

Для примера возьмем ситуацию, когда требуется задать дату и полученный результат поместить в метку **Label** на форме. Создайте новое приложение. Создайте новую форму и перенесите на нее **CommandButton** и одну метку **Label**. При нажатии на **CommandButton1** должна вызываться функция **InputBox**, в поле ее формы вводится дата, затем нажимаем

OK — и введенная data заносится в метку **Label1** после текста: «Сегодня: ». Так как на странице учебника не выводятся визуальные пробелы, сообщаю: в тексте после двоеточия находятся один или лучше два пробела, иначе текст «Сегодня: » и заданная data сольются. С помощью маркеров элемента управления **Label1** раздвиньте ее горизонтальные границы, т. к. получаемый текст может не уместиться в границах по умолчанию (рис. 25).

Назовите командную кнопку **OK**, а в свойстве **Caption** у метки **Label1** введите текст **Сегодня:**

Напишите следующий программный код:

```
Private Sub CommandButton1_Click()
    aaa = InputBox("Введите дату", "Создание даты",
#12/10/2022#)
    Label1.Caption = "Сегодня: " + aaa
End Sub
```

Разберем этот код. Во второй строке мы создаем возвращаемое значение функции **InputBox** под именем **aaa**, так как имя не имеет никакого значения. Возвращаемое значение **aaa** мы объявляем неявно, ему присваивается тип **Variant**. В функции **InputBox** создаем три аргумента. Первые два должны быть понятны, а вот написание третьего требует объяснений. В этом аргументе мы создаем дату, а дата должна быть заключена между символами **#**.

Третья запись в программе проста: в элементе управления **Label1** в свойство **Caption** записывается текстовая константа «Сегодня: » (после двоеточия находится пробел) и приклеивается возвращаемое значение функции **InputBox** под именем **aaa**. Возвращаемое значение **InputBox** всегда имеет тип строки **String**. В этом примере мы не преобразовывали тип строки в другой тип, например **Date**.

Усложним задание: получим возвращаемое значение функции **InputBox** в виде строки и преобразуем ее в дату, чтобы с датой производить какие-либо арифметические или логические действия. После арифметических действий над датой мы

преобразуем ее обратно в строку, чтобы сообщить пользователю результат.

На форму, которую использовали в предыдущем примере, перенесите еще одну метку **Label** под именем **Label2**. С помощью маркеров раздвиньте ее горизонтальные границы.

Задача — преобразовать строковое значение даты, полученной с помощью функции **InputBox**, в тип **Date**. К полученной дате прибавить 10 дней (научимся прибавлять 10 дней, сможем прибавить и любое другое число дней), затем полученный результат преобразовать в тип строки **String**, чтобы продемонстрировать пользователю, что получена некая дата, и выдать ее в метку **Label2**. Мы можем сравнить: что было (Label1) — что стало (Label2).

Допишите программный код, чтобы он принял следующий вид:

```
Private Sub CommandButton1_Click()
    aaa = InputBox("Введите дату", "Создание даты",
#12/10/2022#)
    Label1.Caption = "Сегодня: " + aaa
    aaa = CDate(aaa) + 10
    Label2.Caption = CStr(aaa)
End Sub
```

Разберем изменения в коде. В четвертой строке преобразуем возвращаемое значение **aaa** в тип **Date** с помощью функции **CDate** и прибавим заданное число дней — в нашем примере это 10.

В результате значение **aaa** изменится, как изменится и тип. Если бы мы сейчас попытались вывести значение **aaa** в метку **Label1** (после превращения **aaa** в строку), увидели бы, что дата совершенно другая, увеличенная на 10 дней. Возвращаемое значение **aaa** в ходе выполнения программного кода постоянно меняется: и значение, и тип. Обратите на это особое внимание — **aaa** упоминается в нескольких программных строках и постоянно изменяется.

В пятой строке мы преобразовываем **aaa** в строку и выдаем ее пользователю.

Попробуйте в этом примере в условии задачи изменить число 10 на 40. В этом случае у нас должен измениться не только номер дня, но и номер месяца. Вы увидите, что программа работает безошибочно и совершенно правильно ориентируется в расчете даты.

Метод **InputBox**

Метод **InputBox** позволяет задавать не только значение, но и тип этого значения. Это может быть в некоторых случаях удобнее, чем функция **InputBox**, которая возвращает значение только с типом **String**.

У метода вызова окна **InputBox** следующий синтаксис:

```
expression.InputBox(Prompt, Title, Default, Left,  
Top, HelpFile, HelpContextId, Type)
```

Где:

- **expression** — объект, обязательный аргумент, выражение, которое возвращает объект **Application**;
- **Prompt** — сообщение, обязательный аргумент, текст, отображаемый в окне;
- **Title** — заголовок, необязательный аргумент, текст, помещаемый в строку заголовка (системную строку);
- **Default** — значение по умолчанию, необязательный аргумент;
- **Left, Top** — координаты верхнего левого угла окна ввода на экране, необязательные аргументы;
- **HelpFile, HelpContextId** — имя справочного файла и контекстный индекс темы справочной системы, необязательные аргументы;
- **Type** — код типа данных возвращаемого значения **expression**, необязательный аргумент. Определяет

возвращаемый тип данных. Если этот аргумент опущен, диалоговый блок возвращает текст. Может быть один или суммой величин кодов.

Код типов может принимать следующие значения (таблица 23).

Таблица 23. Коды типов значений

Код	Тип данных
0	Формула
1	Число
2	Текстовая строка
4	Логическое значение (True или False)
8	Адрес ячейки или объекта диапазона Range
16	Значение ошибки #N/A
64	Массив значений

Вы можете использовать сумму допустимых величин для **Type**. Скажем, можно получаемые значения принять и как текст, и как числа, например устанавливать **Type** как 1 + 2.

Используйте **InputBox**, чтобы отображать простой диалог, чтобы вы могли ввести информацию, которой нужно пользоваться в макросе. Диалоговый блок имеет кнопки **OK** и **Отмена**. Если выбираете **OK**, **InputBox** возвращает величину, введенную в диалоговый блок. Если щелкаете **Отмена**, **InputBox** возвращает **False**.

Если Тип — 0, **InputBox** возвращает формулу в форме текста, например «=2*PI()/360». Если есть любые ссылки в формуле, они возвращаются как ссылки стиля A1.

Если Тип — 8, **InputBox** возвращает объект диапазона (**Range**). Вы должны использовать утверждение набора (**Set**), чтобы назначать результат на объект диапазона (**Range**), как показано в следующем примере.

```
Set myRange = Application.InputBox(prompt :=  
"Образец", type := 8)
```

Если вы не используете утверждение набора (Set), переменная устанавливается в величину в диапазоне, а не в самом объекте диапазона (Range).

Следующий пример подсказывает пользователю необходимость ввода числа:

```
myNum = Application.InputBox("Введите число")
```

В результате будет выведено окно **InputBox** (рис. 26):

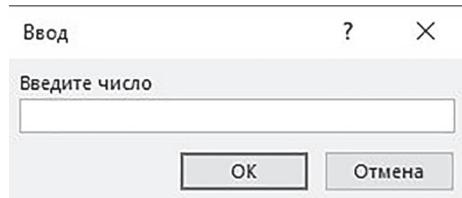


Рисунок 26. Форма метода InputBox

Разумеется, вывод окна **InputBox** — следствие какого-то действия, например нажатия на кнопку **CommandButton**.

Чтобы составить приложение, необходимо в Excel на Лист1 составить следующий пример (таблица 24).

Таблица 24. Исходные данные для примера

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	22
25	25	25	25
25	25	25	25
25	25	25	25
25	25	25	25
25	25	25	25

Во время демонстрации приложения Книга, где хранится Лист1 с данными, должна быть открыта.

Назначение приложения будет следующее: после нажатия на **CommandButton1** открывается окно **InputBox**, в котором задаем номер ячейки, где хранится очень важная и нужная информация. Значение, которое хранится в этой ячейке, должно после закрытия **InputBox** отразиться в надписи **Label1** на нашей форме.

Создайте форму **UserForm1**. Перенесите на нее одну кнопку **CommandButton** и одну метку **Label**. Щелкните дважды по **CommandButton1**, чтобы сгенерировать первоначальный код. Программный код для этого приложения должен иметь следующий вид:

```
Private Sub CommandButton1_Click()
Worksheets("Лист1").Activate
Set myCell = Application.InputBox( _
    prompt:="Выберите ячейку", Type:=8)
Label1.Caption = myCell
End Sub
```

Скомпилируйте проект. Нажмите на **CommandButton**. В окне **InputBox** задайте ячейку, например B4. В ней хранится число 14. Оно и будет выведено в метку **Label1**.

Если мы скопируем наши данные с Лист1 и вставим их на Лист3, а в тексте программы изменим Лист1 на Лист3, в Excel откроем Лист1, после компилирования нашего проекта, как только нажмем на **CommandButton1**, заданный Лист3 будет открыт автоматически.

Это показывает, что мы создали программу, которую нельзя обмануть, какой бы Лист ни был открыт в Excel, нужный нам Лист всегда будет найден (если, конечно, он есть) и открыт.

Выбор ячейки вовсе означает, что ее адрес нужно задавать вручную, достаточно просто выделить ее на рабочем листе. Адрес выделенной ячейки оказывается в поле ввода окна **InputBox**.

Выравнивание элементов управления на форме

После того как элементы управления перенесены на форму, возникает проблема размещения их на форме и относительно друг друга.

Самый простой способ выравнивания элементов относительно друг друга — их выравнивание по сетке, которая есть на форме. Сетка после компилирования становится невидимой. У нее единственное назначение — помочь пользователю более или менее точно выровнять элементы управления.

Наличие сетки на форме и ее параметры определяются в диалоговом окне **Options** (Параметры) на вкладке **General** (Общие параметры) на панели **Form Grid Setting** (Установки линий сетки на форме).

Обратите внимание: при перемещении элементов управления по форме создается ощущение, что оно происходит не плавно, а небольшими рывками. Расстояние рывка равно расстоянию ячейки на сетке. Левый верхний угол элемента управления автоматически по умолчанию привязывается к левому верхнему углу ячейки сетки.

Выравнивание элементов управления относительно друг друга происходит в разделе меню **Format**. Если на форме выделен один элемент управления или не выделен ни один, практически ни одна команда в этом разделе меню недоступна. Выравнивание элементов управления в большинстве случаев происходит относительно друг друга, поэтому для выравнивания нужно выделить несколько элементов, как минимум два. При выделении нескольких элементов управления все команды раздела меню **Format** доступны.

Элемент управления, с которого началось выделение, имеет после выделения группы элементов белые маркеры. Остальные (выделенные потом) получают черные маркеры. Вся группа — как с белыми, так и с черными маркерами — подчиняется

общим правилам при изменении, например, их размеров. Достаточно начать изменять размеры в одном элементе управления (потянув за маркер), как автоматически изменяются размеры всех выделенных элементов управления. Изменение происходит пропорционально имеющимся размерам.

Выравнивание происходит по элементу управления с белыми маркерами.

Рассмотрим команды раздела меню **Format**. Здесь находятся четыре отдельные команды:

- **Size to Fit** — привязать левый верхний угол элемента управления к верхнему левому углу сетки;
- **Size to Grid** — привязать границы элемента управления к границам сетки;
- **Group** — сгруппировать элементы управления;
- **Ungroup** — разгруппировать элементы управления.

Кроме отдельных команд, в разделе меню **Format** находятся субменю, включающие следующие наборы команд.

- **Align** (Выравнивание) имеет команды:
 - **Lefts** (По левому краю) — выравнивание происходит по левому краю элемента управления с белыми маркерами;
 - **Centers** (По центру по вертикальной оси) — выравнивание происходит по центру, находящемуся на вертикальной оси элемента управления с белыми маркерами;
 - **Rights** (По правому краю) — выравнивание происходит по правому краю элемента управления с белыми маркерами;
 - **Tops** (По верхнему краю) — выравнивание происходит по верхнему краю элемента управления с белыми маркерами;

- **Middles** (По центру по горизонтальной оси) — выравнивание происходит по центру, находящемуся на горизонтальной оси элемента управления с белыми маркерами;
 - **Bottoms** (По нижнему краю) — выравнивание происходит по нижнему краю элемента управления с белыми маркерами;
 - **to Grid** (По границам ячеек сетки) — выравнивание происходит по всем четырем углам ячеек сетки на форме.
- **Make Same Size** (Выравнивание по размеру) имеет команды:
- **Width** (Выравнивание по ширине) — все выделенные элементы управления выравниваются по ширине элемента управления с белыми маркерами;
 - **Height** (Выравнивание по высоте) — все выделенные элементы управления выравниваются по высоте элемента управления с белыми маркерами;
 - **Both** (Выравнивание по ширине и по высоте) — все выделенные элементы управления выравниваются по ширине и по высоте элемента управления с белыми маркерами.
- **Horizontal Spacing** (Горизонтальное расстояние) имеет команды:
- **Make Equal** (Установление равенства) — устанавливает одинаковые расстояния между элементами управления по горизонтали;
 - **Increase** (Увеличение) — увеличивает расстояния между элементами управления по горизонтали;
 - **Decrease** (Уменьшение) — уменьшает расстояния между элементами управления по горизонтали;

- **Remove** (Удаление разрывов) — удаляет все зазоры между элементами управления, устанавливая их встык по горизонтали.
- **Vertical Spacing** (Вертикальное расстояние) имеет команды:
 - **Make Equal** (Установление равенства) — устанавливает одинаковые расстояния между элементами управления по вертикали;
 - **Increase** (Увеличение) — увеличивает расстояния между элементами управления по вертикали;
 - **Decrease** (Уменьшение) — уменьшает расстояния между элементами управления по вертикали;
 - **Remove** (Удаление разрывов) — удаляет все зазоры между элементами управления, устанавливая их встык по вертикали.
- **Center in Form** (Выравнивание по центру формы) имеет команды:
 - **Horizontally** (По горизонтали) — выравнивание всех элементов управления по горизонтали, то есть в один столбец;
 - **Vertically** (По вертикали) — выравнивание всех элементов управления по вертикали, то есть в одну строку.
- **Arrange Buttons** (Кнопки привязки) имеет команды:
 - **Bottom** (Вниз) — перемещает **CommandButton** в нижний левый угол по горизонтали;
 - **Right** (Вправо) — перемещает командные кнопки **CommandButton** в верхний правый угол в столбец.

- **Order** (Порядок) имеет команды:
 - **Bring to Front** (На передний план) — помещает выделенный элемент управления поверх всех остальных, если они наложены друг на друга;
 - **Send to Back** (На задний план) — помещает выделенный элемент управления в самый низ всех остальных, если они наложены друг на друга;
 - **Bring Forward** (Выдвинуть) — перемещает выделенный элемент управления на один уровень вверх, если они наложены друг на друга;
 - **Send Backward** (Задвинуть) — перемещает выделенный элемент управления на один уровень вниз, если они наложены друг на друга.

Практическая работа № 1. Преобразование типов данных

Необходимо складывать два числа, задаваемых пользователем, а полученный результат будет выводиться на экран.

Создайте форму. Перенесите на форму три текстовых поля **TextBox** и одну командную кнопку **CommandButton**. В текстовые поля **TextBox1** и **TextBox2** будут вводиться исходные числа, а в текстовом поле **TextBox3** при нажатии на **CommandButton1** будет выводиться результат.

Пример кажется очень простым, но таит в себе проблему с преобразованием типов данных. Казалось бы, ничего сложного здесь нет:

```
TextBox3.Text = TextBox1.Text + TextBox2.Text
```

и результат готов. На самом деле это совсем не так. Программа не найдет здесь ошибки. Проект будет скомпилирован. Но вместо суммы двух чисел будет производиться складывание текста

двух строк. Такое складывание, или склеивание строк, называется конкатенация. Предположим, необходимо сложить два числа, которые вводятся в поля **TextBox1** и **TextBox2**: 20 и 25.

Результат — 45. Но ответ будет 2025, то есть цифры не складываются, а склеиваются между собой. Это происходит потому, что в элементах управления **TextBox** в свойстве **Text** имеется тип данных — *строка*. Чтобы можно было складывать числа, которые задаем в этих полях, нужно строковое значение преобразовать в число. Это делается с помощью функции **CDbl**. После преобразования строки в число с числами можно производить математические вычисления. После арифметического вычисления (в данном случае, сложения) необходимо число обратно преобразовать в строку, чтобы его мог увидеть пользователь. Это производится с помощью функции **CStr**. Теперь можно записать следующий программный код:

```
Private Sub CommandButton1_Click()
Dim A As Double
A = CDbl(TextBox1.Text) + CDbl(TextBox2.Text)
TextBox3.Text = CStr(A)
End Sub
```

В примере использовалась переменная **A** с типом **Double**, чтобы последовательность действий была понятна: сначала в третьей строке преобразовываем строки в числа и производим с ними арифметические действия, а в четвертой преобразовываем полученное число в строку. Но можно написать и такой программный код:

```
Private Sub CommandButton1_Click()
TextBox3.Text = CStr(CDbl(TextBox1.Text) +
CDbl(TextBox2.Text))
End Sub
```

Здесь с помощью круглых скобок разделяем преобразования типов. Действие начинается с самых внутренних круглых скобок и затем продолжается по мере приближения к выходу из выражения.

Чтобы производить другие арифметические действия, нужно заменить знак суммирования (+) на другие знаки: вычитание (-), умножение (*), деление (/), возведение в степень (^) и т. д.

Формально можно работать без преобразования типов данных, так как есть возможность работать с типом данных **Variant**. Но все же этого лучше не делать, а работать с конкретными типами данных. Надежда, что **Variant** сам выберет подходящий тип, как правило, чреват ошибками в будущем. Сообщения о таких ошибках не выдаются, так как программа просто не может их обнаружить.

Практическая работа № 2. Программирование линейных и разветвляющихся алгоритмов

**Задание 1. Создание линейного алгоритма. Дизайн
формы. Управление параметрами фона**

Создайте пользовательскую форму в любом приложении MS Office, например Excel.

Разместите на ней четыре кнопки.

Создайте процедуру инициализации формы.

Создайте процедуры обработки событий от кнопок, изменяющие фон формы.

На модуль, связанный с формой, введите код:

```
Option Explicit
```

```
Private Sub UserForm_Initialize()
Me. Caption = "Графический интерфейс"
Me. BackColor = vbCyan

Label1.Caption = "Времена года"
Label1.Font.Size = 20
CommandButton1.Caption = "Зима"
CommandButton2.Caption = "Весна"
CommandButton3.Caption = "Лето"
CommandButton4.Caption = "Осень"
End Sub
```

```
Private Sub CommandButton1_Click()
Me. BackColor = vbCyan
Label1.Caption = "Зима"
Label1.ForeColor = vbCyan
End Sub

Private Sub CommandButton2_Click()
Me. BackColor = vbGreen
Label1.Caption = "Весна"
Label1.ForeColor = vbGreen
End Sub

Private Sub CommandButton3_Click()
Me. BackColor = vbRed
Label1.Caption = "Лето"
Label1.ForeColor = vbRed
End Sub

Private Sub CommandButton4_Click()
Me. BackColor = vbYellow
Label1.Caption = "Осень"
Label1.ForeColor = vbYellow
End Sub
```

Для фона метки установите свойство Прозрачный. Измените размер шрифта всех элементов, имеющих надпись, на 20 пунктов. Установите для всех надписей свойство, обеспечивающее изменение размера элемента управления в зависимости от размера введенной надписи. Измените цвет шрифта метки, подобрав его к цвету фона. Вставьте в качестве фона рисунки, выбранные из любых файлов. Изучите режимы выравнивания рисунков фона:

- по центру;
- слева;
- справа.

Режимы отображения рисунков фона растяните по форме, впишите в форму (для изображений больших по размеру, чем

форма), расположите мозаикой (для изображений меньших по размеру, чем форма).

Задание 2. Создание разветвляющегося алгоритма

Создайте новое приложение, перенесите в него форму **UserForm**. На форму перенесите текстовое окно **TextBox**, где будем задавать какие-то значения, и командную кнопку **CommandButton**, при щелчке по которой будет происходить анализ введённого числа. Программный код можно представить так:

```
Private Sub CommandButton1_Click()
If TextBox1.Text < 100 Then MsgBox "Значение меньше
100"
End Sub
```

В данной процедуре логический оператор имеет первую форму синтаксиса. Все выражение записано в одну строку. Здесь не нужен оператор завершения логического блока **End If**.

Если бы записали логический блок как:

```
If TextBox1.Text < 100 Then
MsgBox "Значение меньше 100"
```

это была бы запись не в одну, а в две строки. В этом случае должны были указать оператор завершения логического блока **End If**.

В следующем выражении операторы записаны в несколько строк. Поэтому необходимо записать оператор завершения логического блока **End If**. Программный код можно представить так:

```
Private Sub CommandButton1_Click()
If TextBox1.Text > 100 Then
Label1.Caption = "Значение больше 100"
End If
TextBox1.SetFocus
TextBox1 = ""
End Sub
```

```
Private Sub UserForm_Initialize()  
    TextBox1.SetFocus  
End Sub
```

Здесь сначала создается событие формы **Initialize**, которое генерируется при загрузке формы **UserForm**. В нем передается фокус в элемент управления **TextBox1**, то есть при загрузке формы в этом элементе управления будет находиться мигающий курсор, и не нужно устанавливать его вручную.

5

Организация циклов

Оператор цикла For-Next

Циклы предназначены для многократного выполнения одного или нескольких операторов.

Оператор цикла For-Next предназначен для прохождения стольких шагов, сколько задано в условии оператора.

Синтаксис оператора цикла For-Next такой:

```
For counter = start To end [Step step]
[statements]
[Exit For]
[statements]
Next [counter]
```

Где:

- **counter** — обязательный аргумент, числовая переменная, использованная как счетчик цикла. Переменная не может быть типом Boolean или элементом массива;
- **start** — обязательный аргумент, начальное значение **counter**;
- **end** — обязательный аргумент, конечное значение **counter**;
- **step** — необязательный аргумент, счетчик **counter** изменяется всякий раз, когда проходит через цикл. Если не определено, **step** устанавливается по умолчанию в значение 1;
- **statements** — необязательный аргумент, одно или более утверждений между **For** и **Next**, которые выполняются определенное число раз.

Аргумент **step** может быть или положительным, или отрицательным (таблица 25). Величина аргумента **step** определяет цикл обработки следующим образом.

Таблица 25. Влияние аргумента Step на цикл обработки

Значение	Цикл выполняется, если:
Положительное или 0	counter <= end
Отрицательное	counter >= end

Когда утверждения в цикле выполнены, шаг (step) будет добавлен к счетчику (counter). Начиная с этой точки или утверждения, цикл выполняется снова, или цикл выходит из выполнения и продолжается утверждение, следующее за **Next** утверждением.

Любой номер **Exit For** утверждения может устанавливаться в любом месте цикла как альтернативный путь выхода. **Exit For** часто используется после анализа некоторого условия, например **If...Then**, и управление в утверждении немедленно передается в следующее утверждение, следующее за **Next**.

Можно вкладывать циклы **For...Next**, устанавливая один **For...Next** цикл в пределах другого цикла. Задайте каждому циклу уникальное переменное имя как счетчик. Следующая конструкция показывает, как это нужно делать правильно:

```
For I = 1 To 10
    For J = 1 To 10
        For K = 1 To 10
            ...
        Next K
    Next J
Next I
```

Если счетчик опускается в **Next** утверждении, выполнение организуется так, будто счетчик включен со значением, равным 1. Если **Next** утверждение столкнулось с соответствующим **For** утверждением, генерируется ошибка.

Составим пример для практического изучения цикла **For-Next**. Создайте новое приложение. Перенесите на него форму **UserForm**. Перенесите на форму надпись **Label**, в которую будем выводить готовый результат, и кнопку, щелчком

по которой будет запускаться цикл **For-Next**. Программный код можно представить так:

```
Private Sub CommandButton1_Click()
Dim Itogo As Double
Itogo = 0
For Начало = 1 To 100
Itogo = Itogo + Начало
Next Начало
Label1.Caption = Itogo
End Sub
```

В примере задавался шаг цикла. В этом случае по умолчанию шаг принимает значение 1. Цикл был задан от 1 до 100. В реальной жизни значения цикла и шаг цикла задаются пользователем. Доработаем пример.

Перенесите на форму еще одну кнопку **CommandButton** для запуска цикла и три **TextBox**, в которых будем задавать начальное, конечное значение цикла и шаг цикла. Программный код можно представить так:

```
Private Sub CommandButton2_Click()
Dim Itogo As Double
Itogo = 0
For Начало = TextBox1.Text To TextBox2.Text Step
TextBox3.Text
Itogo = Itogo + Начало
Next Начало
Label1.Caption = Itogo
End Sub
```

Оператор цикла Do-While

Если количество проходов должно зависеть от какого-то условия, используют цикл **Do...Loop**. В зависимости от позиции условия различают два варианта цикла **Do...Loop**: **Do-While** или **Do-Until**.

Цикл **Do-While** проверяется в начале условия. Если условие проверяется в начале цикла, он никогда не выполняется в случае невыполнения заданного условия.

Синтаксис оператора цикла **Do-While** следующий:

```
Do [{While} condition]
[statements]
[Exit Do]
[statements]
Loop
```

Или можно использовать следующий синтаксис:

```
Do
[statements]
[Exit Do]
[statements]
Loop [{While} condition]
```

Где:

condition — необязательный аргумент, числовое выражение или строковое выражение, которые могут быть **True** или **False**. Если условие **Null**, условие рассматривается как **False**;

statements — одно или более утверждений, которые повторяются до тех пор, пока **condition** является **True**.

Любой номер утверждения **Exit Do** может устанавливаться в любом месте в **Do...Loop** как альтернативный путь выхода из **Do...Loop**. **Exit Do** часто может быть использован после анализа некоторого условия, например **If...Then**.

Рассмотрим примеры с использованием обоих синтаксисов написания оператора цикла **Do-While**. Создайте новое приложение. Перенесите в него форму **UserForm**. На форму перенесите две кнопки **CommandButton**, с помощью которых будем запускать циклы, и одно текстовое поле **TextBox**, в которое будем выводить результат. Программный код можно представить так:

```
Private Sub CommandButton1_Click()
Do
X = X + 1
Loop While X < 100
```

```
TextBox1.Text = X  
End Sub
```

Для второй кнопки этот же пример переписан с другим синтаксисом. Программный код можно представить так:

```
Private Sub CommandButton2_Click()  
Do While X < 100  
    X = X + 1  
Loop  
TextBox1.Text = X  
End Sub
```

Результат действия в обеих процедурах одинаков.

Оператор цикла Do-Until

Цикл **Do-Until** проверяется в начале условия. Так как проверка заданного условия происходит в конце цикла, он выполняется как минимум один раз — независимо от того, выполнено заданное условие или нет.

Синтаксис оператора цикла **Do-Until** следующий:

```
Do [{Until} condition]  
[statements]  
[Exit Do]  
[statements]  
Loop
```

Или можно использовать следующий синтаксис:

```
Do  
[statements]  
[Exit Do]  
[statements]  
Loop [{Until} condition]
```

Где аргументы точно такие же, как и в операторе цикла **Do-While**.

Этот оператор очень похож на оператор **Do-While** — с той лишь разницей, что фактически это антипод. Оператор **Do-While** выполняется до тех пор, пока условие соответствует **True**, а оператор **Do-Until** выполняется, пока условие соответствует **False**.

Конструкция For Each-Next

Для работы со всеми элементами коллекции можно воспользоваться специальной формой цикла **For Each-Next**. В этом цикле используется объектная переменная цикла (счетчик).

Конструкция **For Each-Next** имеет следующий синтаксис:

```
For Each element In group  
[statements]  
[Exit For]  
[statements]  
Next [element]
```

Где:

- **element** — обязательный аргумент, переменная, используемая для повторения элементов коллекции или массива. Для коллекций элемент может быть только переменной типа **Variant**, общей объектной переменной или любой специфической объектной переменной. Для массивов элемент может быть только переменной типа **Variant**;
- **group** — обязательный аргумент, имя объекта коллекции или массива;
- **statements** — необязательный аргумент, одно или более утверждений, которые выполняются в каждом пункте группы.

Каждый элемент в коллекции один раз присваивается объектной переменной. Если обработаны все элементы коллекции, выполняется автоматический выход из цикла **For Each-Next**.

В следующем примере активации рабочего листа Лист1 будет происходить перебор всех элементов (листов и отдельных диаграмм) в рабочей книге. То есть если покинуть Лист1, ничего не произойдет, а если перейти на Лист1, последовательно будет показана вся коллекция рабочих листов и отдельных диаграмм. Программный код будет иметь следующий вид:

```
Private Sub Worksheet_Activate()
Dim AAA As Worksheet
For Each AAA In ActiveWorkbook.Sheets
MsgBox AAA.Name
Next AAA
End Sub
```

В этом примере запускается цикл перебора всех элементов коллекции. В принципе все равно, что в выбранной коллекции перебирать (в следующем примере — заданные ячейки (A1:D10) на рабочем листе Лист1). Если значение в ячейке (Value) меньше 40, то значение в этой ячейке меняется на 25. При создании процедуры использовалось событие **Deactivate**. Это событие происходит при переходе с Лист1 на любой другой рабочий лист, отдельную диаграмму, в другую рабочую книгу. Поэтому анализ и пересчет ячеек будет происходить только после закрытия указанного листа. В дальнейшем можно убедиться, что это более экономно, чем если бы проводили такой анализ для каждой ячейки отдельно.

Программный код этой процедуры будет выглядеть следующим образом:

```
Private Sub Worksheet_Deactivate()
For Each yy In Worksheets("Лист1").Range("A1:D10")
If yy.Value <= 40 Then
yy.Value = 25
End If
Next yy
End Sub
```

Практическая работа № 3. Программирование циклических вычислительных процессов

В данном примере необходимо указать два числа, от меньшего до большего, в диапазоне которых будет выполняться суммирование.

```
Sub Calc()
    Dim x As Integer
    Dim Pervoe
    Dim Vtoroe
    Dim Itogo As Long
    Pervoe = InputBox("Введите первое число:")
    Vtoroe = InputBox("Введите второе число:")
    Itogo = 0
    For x = CInt(Pervoe) To CInt(Vtoroe)
        Itogo = Itogo + x
    Next x
    MsgBox "Сумма " & Pervoe & " è " & Vtoroe & " равна "
    & Itogo
End Sub
```

Есть одно ограничение: первое число должно быть обязательно меньше второго. Чтобы избавиться от этого недостатка, немного изменим текст программы:

```
Sub Calc()
    Dim x As Integer
    Dim Pervoe
    Dim Vtoroe
    Dim Tretye
    Dim Shet
    Dim Itogo As Long
    Tretye = InputBox("Введите первое число:")
    Shet = InputBox("Введите второе число:")
    If Tretye > Shet Then
        Pervoe = Tretye
        Vtoroe = Shet
    Else
        Pervoe = Shet
        Vtoroe = Tretye
    End If
    For x = CInt(Pervoe) To CInt(Vtoroe)
        Itogo = Itogo + x
    Next x
    MsgBox "Сумма " & Pervoe & " è " & Vtoroe & " равна "
    & Itogo
End Sub
```

```
Else
Pervoe = Shet
Vtoroe = Tretye
End If
Itogo = 0
For x = CInt(Pervoe) To CInt(Vtoroe)
Itogo = Itogo + x
Next x
MsgBox "Сумма " & Pervoe & " è " & Vtoroe & " равна "
& Itogo
End Sub
```

В последнем примере создаются промежуточные переменные *Tretye* и *Shet*. Затем происходит сравнение этих переменных: в каком меньшее значение. Меньшее значение присваивается переменной *Pervoe*, а большее — переменной *Vtoroe*.

6

Пользовательские подпрограммы в VBA

Функция

При работе с языком VBA работать можно только со встроенными функциями VBA.

Для примера суммируем значения, находящиеся в ячейках с A1 по A15. Командную кнопку создадим прямо на поверхности рабочего листа. Для этого войдите в режим конструктора и с панели **Элементы управления** перенесите кнопку **CommandButton1** на рабочий лист. Заполните ячейки с A1 по A15 любыми числами.

Вспомним, как нужно написать функцию суммирования в Excel:

=СУММ(A1:A15)

Использовать функцию суммирования с именем СУММ в VBA нельзя: на языке VBA имеется встроенная функция с именем SUM.

В VBA нет строки формул, поэтому запись формулы с функцией необходимо записывать в кавычках. Все остальные правила записи формулы действуют так же, как и в Excel.

Щелкните дважды по **CommandButton1** и запишите программный код:

```
Private Sub CommandButton1_Click()  
Range("A17").Formula = "=SUM(A1:A15)"  
End Sub
```

Вначале указывается ячейка, в которую собираемся выводить результат вычисления. Далее указывается свойство **Formula**, и после знака равенства указываем формулу с использованием функции.

После того как создадите программу, скомпилируете и опробуйте ее, выделите ячейку с формулой A17. Обратите внимание, что в строке формул указано не имя функции, которую использовали в программе (SUM), а имя функции, используемой

в Excel, то есть СУММ. При использовании программы происходит автоматическое преобразование имени функции из VBA в имя функции в Excel.

Если попробовать вместо имени функции SUM в программном коде использовать имя из Excel, то есть СУММ, ни при компиляции программы, ни при ее выполнении не будет выдано сообщение об ошибке. Но в результате формула не будет рассчитана, а будет выдано сообщение #ИМЯ?, то есть ошибочное имя функции.

Для примера возьмем функцию СРЗНАЧ — расчет среднего значения диапазона ячеек. Перенесите на рабочий лист еще одну кнопку — CommandButton2. Функции СРЗНАЧ (расчет среднего значения в Excel) в VBA соответствуетстроенная функция AVERAGE. Программа для расчета этой формулы такая:

```
Private Sub CommandButton2_Click()
Range("A20").Formula = "=AVERAGE(A1:A10)"
End Sub
```

Принципиально нового в этой программе нет. Имя одной встроенной функции заменяется на другое, а одни аргументы заменяются на другие.

Рассмотрим более сложный пример — с использованием нескольких аргументов. Перенесите на рабочий лист Лист1 **CommandButton3**. Важное отличие синтаксиса встроенных функций VBA от синтаксиса функций Excel: в Excel аргументы отделяются друг от друга точкой с запятой, а в VBA — запятой. Рассмотрим функцию Excel СУММЕСЛИ, которая имеет следующий синтаксис:

СУММЕСЛИ (диапазон; критерий; диапазон_суммирования)

Где:

- диапазон — диапазон вычисляемых ячеек;
- критерий — критерий в форме числа, выражения или текста, определяющего суммируемые ячейки. Например,

критерий может быть выражен как 41, «41», «>41», «валенки»;

- диапазон_суммирования — фактические ячейки для суммирования.

В VBA этой функции соответствует функция SumIf. Разумеется, и синтаксис этой функции такой же, за исключением символа разделения аргументов. Здесь применяется запятая.

Для примера на рабочем листе Лист1 заполните ячейки в диапазоне B1:B15 и D1:D15. Критерий занесите в ячейку C4 (запишите в нее, например >10).

Если бы такая формула создавалась в Excel, нужно было бы записать:

=СУММЕСЛИ(B1:B15; C4; D1:D15)

В VBA программа будет такая:

```
Private Sub CommandButton3_Click()
Range("A21").Formula = "=SumIf(B1:B15, C4, D1:D15)"
End Sub
```

Процедуры-функции

Синтаксис процедур-функций очень похож на объявление процедур и объявляет имя, аргументы и код, чтобы формировать тело процедуры Function (Функция):

```
[Public | Private | Friend] [Static] Function name
[(arglist)] [As type]
[statements]
[name = expression]
[Exit Function]
[statements]
[name = expression]
End Function
```

Где:

- **Public** — необязательный аргумент, указывает, что процедура функции **Function** доступна во всех других процедурах во всех модулях. Если использована в модуле, который содержит опцию **Private**, процедура недоступна за пределами проекта;
- **Private** — необязательный аргумент, указывает, что процедура **Function** доступна только в других процедурах, в том модуле, где она объявлена;
- **Friend** — необязательный аргумент, используется только в модуле класса. Указывает, что **Function** процедура видима для всего проекта, но не видима диспетчеру объекта;
- **Static** — необязательный аргумент, указывает, что **Function** процедура для локальных переменных будет сохранена между вызовами. Атрибут **Static** не влияет на переменные, которые объявлены за пределами **Function**, даже если бы они были использованы в процедуре;
- **name** — обязательный аргумент, имя функции **Function**; следует за стандартным переменным присваиванием имен соглашений;
- **arglist** — необязательный аргумент, список переменных, представляющих аргументы, которые передаются в процедуру функции **Function** после объявления. Каждая переменная отделена запятыми;
- **type** — необязательный аргумент, тип данных аргумента в функции; может быть **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (к настоящему времени не поддерживается), **Date**, **String** (переменная заданной длины), **Object**, **Variant** или специфический объектный тип. Если параметр не дополнительный, он может быть определен как пользовательский тип;
- **statements** — необязательный аргумент, любой набор команд, которые должны выполняться в пределах процедуры функции **Function**;

- **expression** — необязательный аргумент, возвращаемое значение функции **Function**.

Аргумент **arglist** имеет следующий синтаксис:

```
[Optional] [ByVal | ByRef] [ParamArray] varname[( )]  
[As type] [= defaultValue]
```

Где:

- **Optional** — необязательный аргумент, ключевое слово, указывающее, что аргумент не потребовался. Если слово было использовано, все последующие аргументы в **arglist** должны также быть дополнительными и объявленными, используя ключевое слово **Optional**. **Optional** не может быть использовано для любого аргумента, если использован **ParamArray**;
- **ByVal** — необязательный аргумент, указывает, что аргумент имеет величину;
- **ByRef** — необязательный аргумент, указывает, что аргумент имеет ссылку. **ByRef** устанавливается в Visual Basic по умолчанию;
- **ParamArray** — необязательный аргумент, используется только как последний аргумент в **arglist**, чтобы указывать, что конечный аргумент является **Optional** массивом элементов **Variant**. Ключевое слово **ParamArray** позволяет обеспечивать произвольное число аргументов. **ParamArray** не может быть использован вместе с **ByVal**, **ByRef** или **Optional**;
- **varname** — обязательный аргумент, имя переменной, представляющей аргумент; следует за стандартным переменным присваиванием имен соглашений;
- **type** — необязательный аргумент, тип данных аргумента в процедуре; может быть **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (к настоящему времени не поддерживается), **Date**, **String** (переменная заданной

длины), **Object**, **Variant** или специфический объектный тип. Если параметр не дополнительный, может быть определен как пользовательский тип;

- **defaultvalue** — необязательный аргумент, любое постоянное выражение. Используется только для дополнительных параметров. Если объявляется тип **Object**, явное значение по умолчанию может быть только **Nothing**.

Синтаксис похож на описание синтаксиса процедуры. Многое из того, что изучалось в процедурах, применимо и здесь, в том числе ключевые слова **ByVal**, **ByRef** или **Optional**.

Разберем пример по созданию новой функции. Она будет складывать два числа.

Создайте новое приложение. Перенесите новую форму **UserForm**. Перенесите на форму одну **CommandButton** и три **TextBox**. В первых двух текстовых полях будем задавать два суммируемых числа, а в третье — выводить результат.

Щелкните дважды по кнопке **CommandButton1** для создания первоначального кода. Оставим пока этот код без изменения. Задача сейчас в том, чтобы объявить функцию.

Под самой нижней строкой в окне **Code** (а это строка `End Sub`) нажмите несколько раз на клавишу **Enter**, чтобы создать несколько пустых строк. Примерно через одну строку после строки `End Sub` напишите:

```
Function MyFunction(Число1 As Double, Число2 As  
Double) As Double
```

Нажмите на клавишу **Enter**. Будет сгенерирована строка (через одну пустую строку):

```
End Function
```

Рассмотрим первую строку объявления функции-процедуры. После слова **Function** объявляется имя функции — **MyFunction**. Оно может быть любым, его придумывает сам программист.

Хотя нет ограничений на использование русских имен функций, но все-таки лучше брать английские символы, чтобы затем не было вопросов. В круглых скобках указывают аргументы, разделенные запятыми. После последнего аргумента запятой нет. В функции объявляются два аргумента: Число1 и Число2. У каждого аргумента объявляется тип данных. Объявляем тип **Double**. После круглых скобок опять объявляется тип данных, опять как **Double**. Но это не тип аргументов, а тип самой функции MyFunction.

Внутри тела функции-процедуры пишем порядок работы функции:

```
Имя_Функции = (Действие_Функции)
```

Имя функции используется одновременно как переменная. Переменная с именем функции содержит возвращаемое значение. Теперь все связалось воедино: получено возвращаемое значение, совпадающее с именем функции, и тип возвращаемого значения у него в данном примере **Double**. Закончена работа с функцией-процедурой. Переходим к процедуре **CommandButton1_Click**. Здесь необходимо вызвать функцию-процедуру. Вызов функции происходит по следующему принципу:

```
Объект = Имя_Функции(Аргументы_Через_Запятую)
```

Запишите программный код:

```
Private Sub CommandButton1_Click()
    Dim FFF As Double
    FFF = MyFunction(TextBox1.Text, TextBox2.Text)
    TextBox3.Text = FFF
End Sub

Function MyFunction(Число1 As Double, Число2 As
    Double) As Double
    MyFunction = Число1 + Число2
End Function
```

Рассмотрим этот же пример, но не с формой, а с рабочим листом. Командную кнопку можно разместить и на рабочем листе в режиме конструктора. Только вместо текстовых полей

можно задавать ячейки, из которыхчитываются значения, а результат помещаем в заданную ячейку.

Программный код будет иметь следующий вид.

```
Private Sub CommandButton1_Click()
Dim FFF As Double
FFF = MyFunction(Range("B1"), Range("B2"))
Range("B20") = FFF
End Sub
Function MyFunction(Число1 As Double, Число2 As
Double) As Double
MyFunction = Число1 + Число2
End Function
```

Вернемся к примеру на форме. Так как в функции были заданы два аргумента и не обговорено, что это необязательные аргументы, они будут считаться обязательными и передаваться также 2. Так как в функции могут быть обязательные и необязательные аргументы, рассмотрим пример объявления необязательных аргументов. На форме потребуется дополнительное текстовое окно **TextBox**. Вообще-то одно текстовое поле не потребуется, но необходимо следовать строжайшей самодисциплине, введем его.

Объявим в функции третий аргумент — *Число3* — как **Double**. Чтобы объявить этот аргумент как необязательный, нужно перед его именем поставить ключевое слово **Optional**.

```
Private Sub CommandButton1_Click()
Dim FFF As Double
FFF = MyFunction(TextBox1.Text, TextBox2.Text)
TextBox4.Text = FFF
End Sub
Function MyFunction(Число1 As Double, Число2 As
Double, _
Optional Число3 As Double) As Double
MyFunction = Число1 + Число2 + Число3
End Function
```

При вызове функции в процедуре **CommandButton1_Click** не записываем аргумент *Число3*. Так как это необязательный

аргумент, никакого сообщения об ошибке не будет. Если бы решили передать этот аргумент, должны были бы записать:

```
FFF = MyFunction(TextBox1.Text, TextBox2.Text,  
TextBox3.Text)
```

Функции очень похожи на процедуры. Самое существенное отличие — каждая функция всегда возвращает только одно значение, а процедура может возвращать несколько.

Аргументы функций

При работе с аргументами следует знать, что:

- 1) в функциях может вообще не быть аргументов;
- 2) в качестве аргументов можно использовать переменные, константы, текстовые строки, выражения, массивы;
- 3) число аргументов не может быть более 60;
- 4) аргументы могут быть как обязательными, так и необязательными.

Рассмотрим функцию, в которой вообще нет аргументов.

Примером такой функции может служить получение системной даты с некоторым опережением. Такую функцию можно использовать для отслеживания некоторых дат. Так, за 10 дней до некоторых событий можно проанализировать их подготовленность — подготовку к отгрузке товаров: выпуск этих товаров, готовность тары, наличие договоров на перевозку и т. д.

Для создания функции без аргументов потребуется форма **UserForm**, командная кнопка **CommandButton** и текстовое поле **TextBox**.

Функция должна находить текущую дату (с помощью функции **Date**) и прибавлять к ней 10 дней. Как видно из условия создания функции, аргументы не нужны вообще.

Создание функции BezArgum начинаем с объявления процедуры-функции. В круглых скобках ничего не задаем, так как нет аргументов. Тем не менее можем определить тип нашей функции как **Date**. В теле функции задаем:

```
BezArgum = Date + 10
```

Затем переходим к процедуре CommandButton2_Click. Объявим переменную с типом **Date** и присвоим ей значение функции BezArgum без аргументов. Полученное значение выводим в текстовое поле **TextBox**.

Программный код будет иметь такой вид:

```
Private Sub CommandButton2_Click()
Dim AAA As Date
AAA = BezArgum
TextBox1.Text = AAA
End Sub
Function BezArgum() As Date
BezArgum = Date + 10
End Function
```

Если переменным или константам задаем значения с помощью оператора равно (=), то аргументам функций (методам) задаем значения с помощью оператора двоеточие-равно (:=). Например:

```
Set ProbaName = CommandBars(1).Controls _
.Add(Type:="msoControlPopup, Temporary:=True),
```

или

```
Set ProbaPunkt = CommandBars(1).Controls _
.Add(Type:="msoControlPopup, Before:=Punkt.Index,
Temporary:=True)
```

Процедуры VBA

Процедура — это подпрограмма. Она начинается оператором **Sub** и заканчивается оператором **End**. Между этими двумя операторами помещается наш программный код. Такие процедуры могут вызываться или самим VBA (процедуры обработки

событий), или другими процедурами. При этом обработчики событий реализуются как процедуры. Имя процедуры обработки события состоит из имени объекта и имени события:

```
Private Sub CommandButton1_Click()  
End Sub
```

Перед **Sub** указывается метод доступа к процедуре, например **Private**.

Под процедурами подразумевается последовательность команд, объявлений и инструкций, объединенных вместе для выполнения. В зависимости от назначения можно выделить процедуры обработки событий и процедуры общего назначения. В этом случае существенно, кому принадлежит процедура. В зависимости от области определения процедуры бывают закрытые (**Private**) и общие (**Public**). При этом важно, из какого места кода вызывается процедура.

В одной процедуре может находиться неограниченное число программных строк. Вместе с тем принято создавать процедуры небольшого размера, так как процедуры большого объема считаются неудобными для восприятия. К тому же сложный и длинный программный код — это источник ошибок.

Определить границы процедур визуально очень просто — все процедуры (кроме нижней) отделены друг от друга сверху и снизу разделительными процедурными линиями. У нижней процедуры ограничительная процедурная линия только сверху, так как снизу ее граница видна и так, без линии. Это же относится и к первой процедуре, если не объявлены модульные переменные, константы или массивы. Если объявлены, они также отделяются процедурной линией снизу. Если не объявлены, верхняя процедура не имеет верхней разделительной процедурной линии, сверху ее граница видна и так, без линии. То есть разделительные линии создаются по вполне понятным логическим принципам.

Объявление процедуры начинается со слова **Sub**, далее — имя, аргументы и код, чтобы можно было сформировать тело **Sub**-процедуры. Объявление процедуры имеет следующий синтаксис:

```
[Private | Public | Friend] [Static] Sub name  
[(arglist)]  
[statements]  
[Exit Sub]  
[statements]  
End Sub
```

Где:

- **Private** — указывает, что **Sub**-процедура будет доступна только в других процедурах в модуле, в котором она объявлена;
- **Public** — необязательный аргумент, указывает, что **Sub**-процедура доступна во всех других процедурах во всех модулях. Если использована в модуле, который содержит опцию **Private** утверждения, процедура недоступна за пределами проекта;
- **Friend** — необязательный аргумент, используется только в модуле класса. Указывает, что **Sub**-процедура видима для всего проекта, но невидима диспетчеру объекта;
- **Static** — необязательный аргумент, указывает, что **Sub**-процедура для локальных переменных будет сохранена между вызовами. Атрибут **Static** не влияет на переменные, которые объявлены за пределами **Sub**, даже если бы они были использованы в процедуре;
- **name** — имя процедуры **Sub**, обязательный аргумент, любое допустимое имя процедуры, следует за стандартным переменным присваиванием имен соглашений;
- **arglist** — список аргументов, необязательный аргумент, список переменных, представляющих аргументы, которые необходимы в **Sub**-процедуре после объявления. Переменные между собой разделены запятыми;
- **statements** — операторы, необязательный аргумент, любая группа утверждений, которые должны выполняться в пределах **Sub**-процедуры;

- **Exit Sub** — необязательный аргумент, прекращает выполнение процедуры;
- **End Sub** — обязательный аргумент, заканчивает процедуру.

Аргумент **arglist** имеет следующий синтаксис:

```
[Optional] [ByVal | ByRef] [ParamArray] varname[ ( ) ]  
[As type] [= defaultvalue]
```

Где:

- **Optional** — необязательный аргумент, ключевое слово, указывающее, что аргумент не потребовался. Если слово было использовано, все последующие аргументы в **arglist** должны также быть дополнительными и объявленными, используя ключевое слово **Optional**. **Optional** не может быть использовано для любого аргумента, если использован **ParamArray**;
- **ByVal** — необязательный аргумент, указывает, что аргумент имеет величину;
- **ByRef** — необязательный аргумент, указывает, что аргумент имеет ссылку. **ByRef** устанавливается в Visual Basic по умолчанию;
- **ParamArray** — необязательный аргумент, используется только как последний аргумент в **arglist**, чтобы указывать, что конечный аргумент является **Optional** массивом элементов **Variant**. Ключевое слово **ParamArray** позволяет обеспечивать произвольное число аргументов. **ParamArray** не может быть использован вместе с **ByVal**, **ByRef** или **Optional**;
- **varname** — обязательный аргумент, имя переменной, представляющей аргумент; следует за стандартным переменным присваиванием имен соглашений;
- **type** — необязательный аргумент, тип данных аргумента в процедуре; может быть **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (к настоящему времени не поддерживается), **Date**, **String** (переменная заданной

длины), **Object**, **Variant** или специфический объектный тип. Если параметр не дополнительный, может быть определен как пользовательский тип;

- **defaultvalue** — необязательный аргумент, любое постоянное выражение. Используется только для дополнительных параметров. Если объявляется тип **Object**, явное значение по умолчанию может быть только **Nothing**.

Если явно не определено использование, **Public**, **Private** или **Friend**, **Sub**-процедуры являются открытыми по умолчанию. Если слово **Static** не использовано, величина локальных переменных не сохраняется между вызовами. Ключевое слово **Friend** может быть использовано только в модулях класса. Тем не менее процедуры **Friend** могут быть доступны в любом модуле проекта.

Sub-процедуры могут быть рекурсивными. Рекурсия может пройти к переполнению стека. Ключевое слово **Static** обычно не используется рекурсивными **Sub**-процедурами.

Весь выполняемый код должен находиться в процедурах. Нельзя определять одну **Sub**-процедуру в другой **Sub**, **Function** (Функции) или свойстве.

Ключевые слова **Exit Sub** вызывают безотлагательный выход из **Sub**-процедуры. Программное выполнение продолжает утверждение, следующее за утверждением, которое было названо в **Sub**-процедуре. Команда **Exit Sub** может появиться в любом месте в **Sub**-процедуре.

Подобно процедуре **Function** (Функции), **Sub**-процедура является отдельной процедурой, которая может иметь аргументы, выполнять набор утверждений (команд) и изменять величины своих аргументов. В отличие от процедуры **Function** (Функции), которая возвращает величину, **Sub**-процедура не может быть использована в выражении.

Переменные, использованные в **Sub**-процедурах, входят в две категории: которые явно объявлены в пределах процедуры и которые объявлены неявно. Явно объявленные (с использованием

Dim или эквивалентов) всегда бывают локальными. Переменные, которые использованы, но явно не объявлены в процедурах, также локальные, если они явно не объявлены на более высоком уровне за пределами процедуры.

Нельзя использовать **GoSub**, **GoTo** или **Return**, чтобы входить или выходить из **Sub**-процедуры.

Следующий пример использует утверждение, определяющее имя, аргументы и код, которые формируют тело **Sub**-процедуры.

```
' Определение Sub-процедуры.  
' Sub-процедура с двумя аргументами.  
Sub SubComputeArea(Length, TheWidth)  
    Dim Area As Double      ' Объявление локальной  
    переменной.  
    If Length = 0 Or TheWidth = 0 Then  
        ' Если любой аргумент = 0.  
        Exit Sub      ' Немедленное прерывание Sub.  
    End If  
    Area = Length * TheWidth      ' Вычисление площади  
    прямоугольника.  
    Debug.Print Area      ' Вывод Area в окно.  
End Sub
```

У процедур, как и у переменных, строго определенные области видимости. В зависимости от ключевого слова (**Private** или **Public**), заданного перед ключевым словом **Sub**, определяется область видимости процедуры.

Если перед словом **Sub** слово **Private**, процедура видима только в этом модуле. Если перед словом **Sub** слово **Public**, процедура видима во всем приложении.

Если перед словом **Sub** нет этих слов, процедура по умолчанию считается открытой для всего приложения — будто там стоит слово **Public**. Специально отметим: процедуры, в которых перед словом **Sub** нет ключевых слов, считаются открытыми. Но многие программисты все равно стараются в этих случаях писать слово **Public** для лучшего понимания текста программы в дальнейшем: если слово **Public** написано явно, текст программы более понятен.

Процедуру можно вызвать для исполнения различными способами:

- в редакторе Visual Basic выполнить команду **Run** ⇒ **Run Sub/UserForm**, или нажать на кнопку **Run** на инструментальной панели **Standard**, или нажать на клавишу F5;
- в Excel выполнить команду **Сервис** ⇒ **Макрос** ⇒ **Макросы** и в открывшемся диалоговом окне **Макрос** выбрать имя выполняемого макроса и нажать на **Выполнить** или на Alt + F8;
- с помощью *горячих* клавиш, назначенных процедуре;
- с помощью командных кнопок, назначенных процедуре;
- с помощью команды меню, назначенной процедуре;
- с помощью объектов на рабочем листе (об этом в конце следующего раздела);
- при генерировании некоторых событий (щелчок левой клавишей по элементу управления, двойной щелчок, щелчок правой клавишей, проведение мышью по элементу управления, активация и деактивация чего-либо и т. д.);
- из другой процедуры этого же приложения;
- из процедуры другого приложения.

Вызов процедуры

Чтобы вызвать процедуру из другой процедуры, необходимо указать ее в явном виде. Есть три способа:

- 1) указать имя процедуры с ее аргументами, разделенными запятыми (если аргументов нет, указывать их не нужно);
- 2) использовать ключевое слово **Call**, после которого находится имя вызываемой процедуры с аргументами, заключенными в круглые скобки и разделенными запятыми;
- 3) с помощью метода **Run** объекта **Application**.

Рассмотрим первый способ. Создайте новое приложение. Перенесите на него форму **UserForm1** и модуль **Module1**. Смысл в том, что в модуле **Module1** создается процедура, а вызов ее осуществляется щелчком по **CommandButton1**. Чтобы усложнить задание, при щелчке по **CommandButton1** будем не только вызывать процедуру из **Module1**, но и выполнять какие-то другие действия, например выводить текст в надпись **Label1**. Становится понятным, что вызов процедуры ничем не отличается от любого действия в процедуре. Например, в **CommandButton1_Click** можно, кроме вызова процедуры, сделать логический анализ, подсчитать значение в цикле, определить какие-то параметры в операторных скобках **With-End With** и т. д.

Перенесите на форму **CommandButton1** надпись **Label1** и текстовое поле **TextBox1**. Назначение этих элементов управления следующее: щелчок по кнопке вызывает процедуру из модуля **Module1**, результат расчета в **Module1** попадает в текстовое поле **TextBox1**, а в надпись **Label1** выводится текст из процедуры щелчка командной кнопки. В модуле **Module1** напишите следующий программный код:

```
Public Sub MySub()
    Dim AAA As Double
    AAA = 100
    Dim BBB As Double
    BBB = AAA / 25
    UserForm1.TextBox1.Text = BBB
End Sub
```

Щелкните дважды по **CommandButton1** для создания первоначального кода и впишите программный код:

```
Private Sub CommandButton1_Click()
    Module1.MySub
    Label1.Caption = "Задание выполнено!"
End Sub
```

В **Module1** перед словом **Sub** находится слово **Public**, то есть процедура объявлена открытой. Если бы указали слово **Private**, не смогли бы вызвать процедуру **MySub** из другого модуля.

В процедуре **Module1** необходимо явно указать местонахождение элемента управления **TextBox1** — на форме **UserForm1**.

В вызывающей процедуре вызов другой процедуры состоит из двух частей:

- 1) адрес вызываемой процедуры — **Module1**;
- 2) имя вызываемой процедуры — **MySub**.

Итого получилось: **Module1.MySub**.

В следующем примере будем вызывать не одну процедуру, а две. Для этого перенесите на форму еще одно текстовое поле **TextBox2**.

В **Module1** сделайте переменные **AAA** и **BBB** не процедурными, а модульными: вырежьте их объявления и переместите в начало модуляя:

```
Dim AAA As Double  
Dim BBB As Double
```

Первая процедура будет иметь такой программный код:

```
Public Sub MySub()  
AAA = 100  
BBB = AAA / 25  
UserForm1.TextBox1.Text = BBB  
End Sub
```

Вторая процедура будет иметь такой программный код:

```
Public Sub MySub2()  
AAA = 100  
BBB = AAA * 4  
UserForm1.TextBox2.Text = BBB  
End Sub
```

Процедура для обработки щелчка **CommandButton1** будет иметь программный код:

```
Private Sub CommandButton1_Click()  
Module1.MySub  
Module1.MySub2
```

```
Label1.Caption = "Задание выполнено!"  
End Sub
```

Предположим, что вторая процедура (MySub2) находится не в **Module1**, а в **Module2**. Тогда вызов был бы такой:

```
Module2.MySub2
```

В примере указывается имя модуля, где находится вызываемая процедура.

Рассмотрим вызов процедуры с помощью ключевого слова **Call**.

Ключевое слово **Call** управляет передачей процедуры в **Sub**-процедуру, процедуру функции (Function). Имеет следующий синтаксис:

```
[Call] name [argumentlist]
```

Где:

- **Call** — необязательный аргумент, ключевое слово. Если оно определено, необходимо задать **argumentlist** в круглых скобках. Например:

```
Call MyProc(0)
```

- **name** — обязательный аргумент, имя вызываемой процедуры;
- **argumentlist** — необязательный аргумент, Список переменных, массивов или выражений, входящих в процедуру. Компоненты **argumentlist** могут включать ключевые слова **ByVal** или **ByRef**. **ByVal** и **ByRef** могут быть использованы **Call** только при процедуре вызова DLL.

Не обязательно использовать ключевое слово **Call** при вызове процедуры. Но если оно используется для вызова процедуры, которая требует аргументы, **argumentlist** должен быть приложен в круглых скобках. Если **Call** опускается, необходимо также опустить круглые скобки вокруг **argumentlist**. Чтобы передавать целый массив в процедуру, используйте имя массива с пустыми круглыми скобками.

Этот пример иллюстрирует, как используется утверждение **Call**, чтобы передавать процедуру в **Sub**-процедуру, функцию и динамическую библиотеку (DLL).

```
' Вызов Sub-процедуры.  
Call PrintToDebugWindow ("Здравствуй, мир!")  
' Вышеуказанное утверждение предписывает передать  
в следующую Sub-процедуру.  
Sub PrintToDebugWindow(AnyString)  
    Debug.Print AnyString      ' Вывод в окно  
Immediate.  
End Sub
```

Рассмотрим следующий пример. Создайте новое приложение. Вставьте в него **UserForm1** и **Module1**. Перенесите на форму надпись **Label** и кнопку **CommandButton**. Смысл примера в том, что в **Module1** создается процедура, ну, скажем, подсчета чего-нибудь, а при нажатии на **CommandButton1** подсчитанное значение выдается в надпись **Label1**.

В модуле **Module1** запишите следующий программный код:

```
Public Sub ZZZ()  
Dim AAA As Currency  
Dim BBB As Currency  
AAA = 50  
BBB = AAA + 100  
UserForm1.Label1.Caption = BBB  
End Sub
```

Щелкните дважды по **CommandButton1**, чтобы создать первоначальный код:

```
Public Sub CommandButton1_Click()  
' UserForm1.Image1.Visible = True  
Call Module1.ZZZ  
End Sub
```

В первой строке **Module1** создаем процедуру **Sub** с именем **ZZZ**.

Она не привязана ни к одному событию, поэтому необходимо записывать объявление **Sub** вручную.

Чтобы у программы не было вопросов, ей необходимо явно сообщить, что создается открытая процедура, добавляя перед словом **Sub** ключевое слово **Public**.

Теперь созданная процедура будет открыта для всех модулей.

Далее в теле процедуры происходит объявление переменных, инициализация одной из них и само выражение.

Создайте команду, которая выводит полученное значение переменной **VBV** в надпись на **UserForm1**. В приложении может быть несколько форм, и чтобы устранить неясности толкования адреса надписи, нужно прямо сообщить, что необходимо выводить в надпись **Label1**, которая расположена на **UserForm1**. Завершаем процедуру строкой **End Sub**.

Рассмотрим процедуру **CommandButton1_Click**.

Комментарии оформлены строки процедуры, которые могли бы находиться в ней помимо вызова процедуры **ZZZ**. Кроме прямого вызова, здесь могут находиться любые программные строки: вывод надписей, вывод картинки (как в данном случае) и т. д.

Далее следует вызов процедуры **ZZZ**. Сначала записывается ключевое слово **Call**, затем через пробел — имя модуля, в котором находится процедура, и через точку — имя вызываемой процедуры. Затем могут следовать другие программные строки (могут использовать полученное в процедуре **ZZZ** значение).

Наконец процедура **CommandButton1_Click** закрывается.

До этого при вызове ее из другой рабочей книги пользовались адресом модуля нахождения процедуры и именем процедуры. При вызове из другой рабочей книги необходимо указать еще и имя рабочей книги. Например:

Имя_Проекта.Имя_Модуля.Имя_Процедуры

Вызывать процедуры можно и так:

- 1) нажать на инструментальную кнопку или строку меню;
- 2) при наступлении событий (например, щелчок по кнопке, щелчок правой клавишей, при проведении указателя мыши над элементом управления и т. д.);
- 3) нажимая на *горячие* клавиши.

Аргументы процедур

Процедуры могут использовать аргументы, список которых (при необходимости с указанием типа) размещают в скобках после ее имени. В качестве аргументов могут выступать переменные, текстовые строки, константы, массивы, объекты.

Процедура может вообще не содержать аргументов или содержать их неопределенное число, или часть обязательных аргументов, а часть — необязательных.

При создании процедур обработки событий очень часто используем аргументы процедур.

В процедурах событий набор аргументов зависит от события и не может быть изменен разработчиком. Рассмотрим процедуру обработки события **MouseDown** на форме:

```
Private Sub UserForm_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, _  
ByVal X As Single, ByVal Y As Single)  
End Sub
```

В общих процедурах количество и порядок используемых аргументов определяется разработчиком.

В заголовке процедуры можно указывать тип данных для аргумента. В рассмотренной выше процедуре аргументы **Button** и **Shift** имеют тип **Integer**, а **X** и **Y** — тип **Single**. В VBA аргументы могут передаваться двумя способами: либо как

значение (ByVal), либо как ссылки (ByRef). В данном случае **ByVal** — необязательный аргумент, указывающий, что аргумент имеет величину.

Если аргумент передается как ссылка, вызванная процедура получает физический адрес памяти передаваемой переменной. Различие между двумя видами передачи в том, что при передаче аргумента как ссылки можно изменять его значение. Вызывающая и вызывающая процедуры обращаются к одной и той же области памяти, поэтому значение переменной для них идентично. Чтобы передать аргумент как ссылку, следует перед ним указать ключевое слово **ByRef**. Однако поскольку по умолчанию аргументы в Visual Basic именно так и передаются, **ByRef** можно и опустить.

Рассмотрим пример с использованием аргумента, передаваемого по ссылке. Создайте новое приложение. Создайте форму **UserForm**. Перенесите на нее **CommandButton** и **TextBox**. Щелкните дважды по кнопке для создания программного кода и перепишите следующий программный код:

```
Private Sub CommandButton1_Click()
A = 100
SomeProcedure A, B
End Sub
Sub SomeProcedure(ByRef Первый, Второй)
Второй = Первый * 10
TextBox1.Text = Второй
End Sub
```

В этом примере создадим две процедуры: **CommandButton1_Click** и **SomeProcedure**. Из первой процедуры происходит запуск второй. Во вторую передаются два аргумента: А и В. Аргумент А инициализирован, аргумент В — нет. В качестве аргументов здесь выступают переменные. В процедуре **SomeProcedure** нет указаний, что эти аргументы являются дополнительными, поэтому они оба считаются обязательными. Если передать только один аргумент А, будет выдано сообщение об ошибке. Во второй процедуре аргумент А передается аргументу *Первый*, а аргумент В передается аргументу *Второй*. Поэтому, когда во второй процедуре начинается расчет, аргумент *Первый* получает значение 100.

Для передачи аргументов в качестве значений перед именем аргумента в заголовке процедуры указываем ключевое слово **ByVal** — и процедуре передается копия этого значения. При передаче аргументов в качестве значений ключевое слово **ByVal** должно указываться обязательно.

По умолчанию применяется передача аргументов в виде ссылок.

Если в процедуре используются аргументы, передаваемые как ссылки и как значения, указывается только ключевое слово **ByVal**, а слово **ByRef** опускается. Аргументы, перед которыми нет **ByRef**, принимают его автоматически.

Если при вызове процедуры указать не все аргументы, последует сообщение об ошибке. Однако в процессе описания процедуры можно определить, что не все аргументы указываются при вызове. Такие аргументы называются необязательными. Чтобы аргумент стал необязательным, перед его именем ставится ключевое слово **Optional**. После первого необязательного аргумента все последующие должны быть также необязательными. Не разрешается создавать процедуры, в которых задан сначала хотя бы один необязательный аргумент, а затем хотя бы один обязательный. Есть строгая иерархия: сначала перечисляются обязательные аргументы, а потом необязательные.

Рассмотрим предыдущий пример. В первой процедуре объявляем переменную A, а вынуждены передавать в качестве аргументов во вторую процедуру две переменные, в том числе и B, которую даже не объявили. Поэтому можно сделать необязательным аргумент *Второй* в процедуре **SomeProcedure**. При объявлении процедуры перед необязательным аргументом ставится слово **Optional**.

```
Private Sub CommandButton1_Click()
A = 100
B = 0
SomeProcedure A      ', B
End Sub
Sub SomeProcedure(ByRef Первый, Optional Второй)
Второй = Первый * 10
TextBox1.Text = Второй
End Sub
```

В процедуре **CommandButton1_Click** оформлена комментарием переменная *В*. Теперь, когда аргумент *Второй* стал необязательным, и передача в него ссылки также стала необязательной.

Для необязательных аргументов наряду с типом **Variant** можно задавать и другие типы данных, за исключением пользовательских.

Процедуры свойств

Кроме известных процедур **Sub** и **Function**, в VBA есть еще один вид процедур — **Property**, с помощью которых можно определять свойства класса. Чтобы процедуры были видимыми вне собственного контейнера, их следует объявлять как **Public**. Поскольку значения свойств можно как считывать, так и устанавливать, для одного свойства могут потребоваться две процедуры с одним и тем же именем: одна для чтения, другая для присвоения значения свойства. Чтобы различить их, в заголовке процедуры используется дополнительное ключевое слово (**Let** или **Get**).

Операция присваивания значения переменной имеет следующий синтаксис:

```
[Public | Private | Friend] [Static] Property Let  
name ([arglist,] value)  
[statements]  
[Exit Property]  
[statements]  
End Property
```

Где:

- **Public** — необязательный аргумент, указывает, что **Let** процедура доступна во всех других процедурах во всех модулях. Если использована в модуле, который содержит опцию **Private** утверждения, процедура недоступна за пределами проекта;
- **Private** — указывает, что **Let** процедура будет доступна только в других процедурах в этом же модуле, в котором она объявлена;

- **Friend** — необязательный аргумент, используется только в модуле класса. Указывает, что **Let** процедура видима для всего проекта, но не видима диспетчеру объекта;
- **Static** — необязательный аргумент, указывает, что **Let** процедура для локальных переменных будет сохранена между вызовами. Атрибут **Static** не влияет на переменные, которые объявлены за пределами **Let**, даже если они были использованы в процедуре;
- **name** — имя процедуры **Let**, обязательный аргумент, любое допустимое имя процедуры, следует за стандартным переменным присваиванием имен соглашений;
- **arglist** — список аргументов, необязательный аргумент, список переменных, представляющих аргументы, которые необходимы в **Let** процедуре после объявления. Переменные между собой разделены запятыми;
- **value** — обязательный аргумент, переменная, содержит величину, которую нужно передать в свойство;
- **statements** — операторы, необязательный аргумент, любая группа утверждений, которые должны выполняться в пределах **Let** процедуры;
- **Exit Property** — необязательный аргумент, прекращает выполнение процедуры;
- **End Property** — обязательный аргумент, заканчивает процедуру.

Аргумент **arglist** имеет следующий синтаксис и разделы:

```
[Optional] [ByVal | ByRef] [ParamArray] varname[( )]
[As type] [= defaultvalue]
```

Где:

- **Optional** — необязательный аргумент, ключевое слово, указывающее, что аргумент не потребовался. Если слово

использовано, все последующие аргументы в **arglist** должны быть также дополнительными и объявленными, используя ключевое слово **Optional**. **Optional** не может быть использовано для любого аргумента, если использован **ParamArray**;

- **ByVal** — необязательный аргумент, указывает, что аргумент имеет величину;
- **ByRef** — необязательный аргумент, указывает, что аргумент имеет ссылку. **ByRef** устанавливается в Visual Basic по умолчанию;
- **ParamArray** — необязательный аргумент, используется только как последний аргумент в **arglist**, чтобы указывать, что конечный аргумент является **Optional** массивом элементов **Variant**. Ключевое слово **ParamArray** позволяет обеспечивать произвольное число аргументов. **ParamArray** не может быть использован вместе с **ByVal**, **ByRef** или **Optional**;
- **varname** — обязательный аргумент, имя переменной, представляющей аргумент; следует за стандартным переменным присваиванием имен соглашений;
- **type** — необязательный аргумент, тип данных аргумента в процедуре; может быть **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (к настоящему времени не поддерживается), **Date**, **String** (переменная заданной длины), **Object**, **Variant** или специфический объектный тип. Если параметр не дополнительный, может быть определен как пользовательский тип;
- **defaultvalue** — необязательный аргумент, любое постоянное выражение. Используется только для дополнительных параметров. Если объявляется тип **Object**, явное значение по умолчанию может быть только **Nothing**.

Операция считывания значения свойства объявляется с использованием ключевого слова **Get**. Имеет следующий синтаксис:

```
[Public | Private | Friend] [Static] Property Get  
name [(arglist)] [As type]
```

```
[statements]
[name = expression]
[Exit Property]
[statements]
[name = expression]
End Property
```

Где:

- **Public, Private, Friend, Static, name, arglist, statements** — имеют то же описание, что и в синтаксисе Property Let;
- **type** — необязательный аргумент, тип возвращаемых данных процедуры; может быть **Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal** (к настоящему времени не поддерживается), **Date, String** (переменная заданной длины), **Object, Variant** или специфический объектный тип. Если параметр не дополнительный, может быть определен как пользовательский тип;
- **expression** — необязательный аргумент, значение свойства, возвращаемого процедурой определенного **Property Get** утверждения.

Практическая работа № 4. Программирование с использованием функций

По умолчанию в Microsoft Excel нет инструментов для расчетов по цвету. Предлагается пример для анализа цвета фона ячейки. Пример создавался в версии приложения Microsoft Excel 2021. Версия ОС Windows 10. Создайте список студентов по примеру (рис. 29). Каждый предмет разделен на два столбца: на белом фоне число пропусков, на красном фоне — число двоек.

Выполните команду **Разработчик** ⇒ **Microsoft Visual Basic**. В открывшемся окне **Microsoft Visual Basic for Applications** выполните команду **Insert** ⇒ **Module**. Вставьте в окне модуля следующий программный код:

```
Public Function Суммаячейка(DataRange As Range,  
ColorSample As Range) As Double  
    Dim Sum As Double  
    Application.Volatile True  
    For Each Cell In DataRange  
        If Cell.Interior.Color = ColorSample.  
Interior.Color Then  
            Sum = Sum + Cell.Value  
        End If  
    Next Cell  
    Суммаячейка = Sum  
End Function
```

Сохраните рабочую книгу, нажав **Save** (Сохранить). В данном примере суммирование будет выполняться по цвету заливки ячейки.

В функции **Суммаячейка** два параметра:

- **DataRange** — диапазон ячеек, в которых необходимо выполнить расчет. Диапазон необходимо указывать по всем ячейкам, независимо от цвета;
- **ColorSample** — образец цвета. Как правило, ячейка с цветом должна быть в виде абсолютной ссылки (нажмите **F4**).

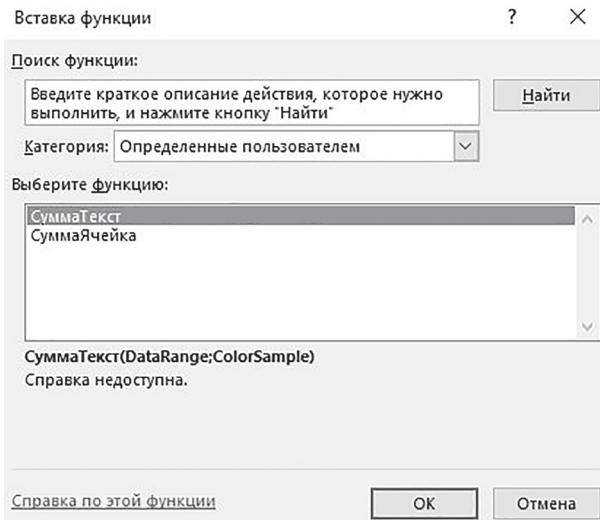


Рисунок 27. Выбор функции

После возвращения на рабочий лист с данными перейдите в раздел **Формулы** и выполните команду **Вставить функцию**. В открывшемся окне **Вставка функции** откройте список **Категория** и выберите категорию **Определенные пользователем** (рис. 27). В данном примере две пользовательские функции: **СуммаЯчейка** суммирует ячейки с одинаковой цветовой заливкой, а функция **СуммаТекст** суммирует ячейки со значениями одного цвета. Щелкните мышью по функции **СуммаЯчейка**, после чего откроется окно **Аргументы функции** (рис. 28).

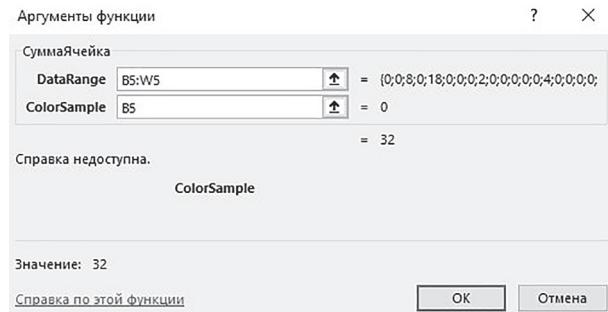


Рисунок 28. Выбор аргументов функции

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC			
Отчет классного руководителя по пропускам и неудовлетворительным оценкам группы 211 специальности Сестринское дело Пропуски/неучтываемость за март 2021 г.																																
ФИО																																
1		Философ	ИНО	ТОМУ	Инфор-	Основы	Основы	Фирмах																				Пропуск	Неу-			
2		и	и	и	техн	латинс-	анатомии	микроби	вза-ра	и	и	и	и	и	и	и	и	и	и	и	и	и	и	и	и	и	и	и	и	и	и	
3		н	2	н	2	н	2	н	2	н	2	н	2	н	2	н	2	н	2	н	2	н	2	н	2	н	2	н	2	н	2	
4																																
5	Альмагузова Марина Фетулаховна	8	15		2					4																			32	0		
6	Амитса Лидда Азимовна	36	4	1	8	4				4																			66	1		
7	Амирова Даира Руслановна																												0	0		
8	Багаевская Надежда Геннадьевна																												32	0		
9	Гавриловская Алена Борисовна	2	4																										16	4		
10	Гаметукова Алина Сергеевна	2	15																										58	1		
11	Городенко Елена Петровна																												0	2		
12	Джабирова Альбина Исмагиловна																												4	0		
13	Джабирова Альбина Исмагиловна																												0	2		
14	Ильинов Олег Муратзалиевич																												16	1		
15	Исаакова Гульмиля Ерхеновна																												16	0		
16	Касимова Христина Алексеевна																												12	3		
17	Козлова Кира Сергеевна	6	14																										78	1		
18	Котлякова Екатерина Дмитриевна																												42	3		
19	Кошелева Валентина Дмитриевна																												4	0		
20	Кропоткина София Борисовна																												4	0		
21	Лукьянова Екатерина Ивановна																												52	1		
22	Лукьяненко Елена Тимуровна																												18	2		
23	Лукьяненко Елена Тимуровна																												4	1		
24	Рашитова Диана Магомедовна																												0	0		
25	Умутоглу Гульмиля Султановна	6	24	8	8	18																						82	2			
26	Фарзалиева Сабрина Алимурадовна																												14	0		
27	Шильбазова Мадина Айнурдановна																												0	1		
																												544	23			

Рисунок 29. Готовый результат

Выделите ячейки со значениями и пустые, независимо от цвета по одному студенту. В нашем примере это ячейки с B4 по M4. Для выбора образца цвета необходимо щелкнуть по ячейке,

в которой находится этот образец. В нашем примере это ячейка B4, в которой нет заливки. Ссылка на ячейку с образцом цвета фона должна быть абсолютной (F4). Нажмите на клавишу F4 для создания абсолютной ссылки, чтобы образец цвета не изменил своего значения. Нажмите на кнопку **OK**.

Полученную формулу необходимо копировать по остальным строкам. Результат представлен на рис. 29.

В следующем примере необходимо суммировать значения ячейкам, если цвет ячеек один и тот же, но сами значения разных цветов, например, каждый участок в цехе оформляет свои данные значениями разных цветов. Снова выполните команду **Разработчик** ⇒ **Microsoft Visual Basic**. В открывшемся окне **Microsoft Visual Basic for Applications** выполните команду **Insert** ⇒ **Module**. Вставьте в окне модуля программный код:

```
Public Function СуммаТекст(DataRange As Range,
ColorSample As Range) As Double
    Dim Sum As Double
    Application.Volatile True
    For Each Cell In DataRange
        If Cell.Font.Color = ColorSample.Font.Color
Then
            Sum = Sum + Cell.Value
        End If
    Next Cell
    СуммаТекст = Sum
End Function
```

В ячейках C1 и E1 текст красного цвета, а в ячейках D1 и F1 текст черного цвета. Ссылка на ячейку с образцом цвета текста должна быть абсолютной (F4). Результат представлен на рис. 30.

	<input checked="" type="checkbox"/>	<input type="button" value="fx"/>	=СуммаТекст(C1:F1;\$C\$1)
C	D	E	F
25	12	39	19
			G
			64
			H
			31

Рисунок 30. Суммирование значений разных цветов

7

Модули VBA

Типы модулей: модули форм, стандартные модули, модули классов

Редактор VBA предлагает множество модулей различного назначения для создания в них программного кода.

Модуль формы предназначен для ее обслуживания и элементов управления на ней.

Стандартные модули предназначены для создания отдельных процедур, как связанных с формами, так и не связанных.

Модуль класса предназначен для создания собственных объектов.

Кроме этого программный код можно создавать для объектов, связанных с приложениями, например Документ в Microsoft Word, Рабочая книга и Рабочий лист в Microsoft Excel, так как элементы управления могут быть расположены в Документе, Рабочих листах.

И модулей, и форм в проекте может быть любое количество: для каждого такого объекта создается отдельный модуль для программного кода.

Для вставки нового модуля можно использовать кнопку **Insert UserForm**. Пиктограмма кнопки зависит от того, какой модуль выбран последним. Так как по умолчанию предлагается модуль UserForm, на кнопке находится пиктограмма этого модуля.

Вставить модули можно также из раздела меню **Insert** (Вставка).

Весь текст программы во всех модулях подсвечивается цветом для подсказок пользователю:

- черным — текст программы;
- синим — ключевые слова;
- красным — ошибки;
- зеленым — комментарии.

Области видимости переменных

Переменные бывают локальными, модульными, глобальными и статическими.

Модульные переменные

Если требуется создать приложение, где одна и та же переменная используется в нескольких процедурах одного и то же модуля, необходимо объявить не локальную переменную, а переменную модульного уровня.

Переменная модульного уровня объявляется в самых первых строках модуля, до записей первой процедуры в этом модуле. Объявление модульных переменных происходит в разделе модуля (General) (Declarations).

Инициализация модульной переменной в разделе (General) (Declarations) не производится. Инициализация такой переменной производится в конкретной процедуре, использующей такую процедуру.

При объявлении модульных переменных объявление можно декларировать не только с помощью ключевого слова **Dim**, но и с помощью ключевых слов **Private** и **Public**. Например:

```
Private DDD As Currency  
Public WWW As Currency  
Private Sub CommandButton1_Click()  
    DDD = 100  
    WWW = DDD * 2  
    TextBox1.Text = WWW  
End Sub
```

При объявлении модульной переменной с ключевым словом **Public** она становится глобальной.

Глобальные переменные

Если переменная используется во всех процедурах всех модулей, ее необходимо объявить глобальной. Глобальная переменная объявляется только с ключевым словом **Public**.

Рассмотрим следующий пример. Создайте форму **UserForm1**. Перенесите на нее одну кнопку **CommandButton** и два текстовых поля **TextBox**. Создайте новый **Module1**. Щелкните дважды по имени модуля в окне **Project — VBA Project**. Объявите переменные в **Module1**:

```
Public HHH As Currency  
Public JJJ As Currency
```

Щелкните дважды левой клавишей по имени **UserForm1**. Щелкните дважды по **CommandButton** для создания исходного программного кода. Впишите между строк текст, чтобы программа стала следующей:

```
Private Sub CommandButton1_Click()  
    TextBox1.Text = TypeName(HHH)  
    TextBox2.Text = TypeName(JJJ)  
End Sub
```

Эта программа выводит тип переменных **HHH** и **JJJ**. Этот тип будет **Currency**.

Усложним пример. В **Module1** запишите новую строку:

```
Private PPP As Currency ' Денежный формат
```

Так как эта переменная объявляется не с **Public**, а с **Private**, она не будет доступна вне границ **Module1**. Следовательно, если попробовать использовать переменную **PPP** в **UserForm1**, тип **Currency** не будет получен. Вместо него будет какой-то другой тип данных или не будет ничего. Ничего — означает **Empty** (Пустой).

Если перенести на форму третье текстовое поле **TextBox** и дописать в обработчике кнопки **CommandButton** следующую строку:

```
TextBox3.Text = TypeName(PPP)
```

то убедимся, что объявленная переменная в **Module1** — переменная PPP — действительно в **UserForm1** не имеет типа **Currency**. В поле **TextBox3** появляется значение **Empty** (Пустой). У переменной PPP в **UserForm1** нет вообще никакого типа.

Может возникнуть вопрос: а почему переменная PPP сообщает, что у нее нет вообще никакого типа? Может, у нее все-таки **Variant**, так как до этого утверждалось, что если тип не задан явно, он автоматически (если не используется оператор вида **Deftype**) получает тип **Variant**, а затем подстраивается под наиболее подходящий тип данных?

Все-таки нет. В переменной PPP действительно нет даже типа **Variant**, так как *нигде* не объявляется (то есть нигде не упоминается об этой переменной) о переменной PPP в этой процедуре и в этом модуле. В примере только выяснялся ее тип с помощью функции **TypeName**. Если бы, например, в процедуре **CommandButton1_Click()** записали:

```
PPP = ННН / ЈЈЈ
```

или

```
PPP = PPP + 0
```

то, действительно, переменная PPP получила бы тип **Variant**, а затем этот тип был бы подстроен под наиболее подходящий, скорее всего, под **Integer**.

Если в модуле произвести какой-нибудь расчет с участием переменной PPP, тогда и тип, и полученное значение переменной PPP останутся только в пределах видимости **Module1**. А переменные ННН и ЈЈЈ видны в **UserForm1**.

Рассмотрим такое понятие, как затенение переменных. Продолжим наш разработанный чуть ранее пример. Объявите в **Module1** переменную:

```
Public МММ As Currency
```

Перейдите в окно **Code** формы **UserForm1**. Объявите модульную переменную (в самом верху этого модуля):

```
Public МММ As Double
```

Полностью перепишите обработчик щелчка кнопки **CommandButton1**:

```
Private Sub CommandButton1_Click()
Dim MMM As Integer
MMM = 10
TextBox1.Text = TypeName(MMM)
End Sub
```

Таким образом, была объявлена переменная **MMM** три раза, но не получено сообщение о том, что обнаружена ошибка. Вспомните, совсем недавно рассматривалась очень похожая ситуация: сначала инициализировали переменную, а затем объявляли ее. При этом было выдано сообщение: (**Duplicate declaration in current scope**). В этом примере переменная объявляется не два раза, а три. Казалось бы, программа должна не просто выдать сообщение об ошибке, но и выдать его не менее двух раз.

На самом деле в последнем примере нет ни одной ошибки. Просто переменную **MMM** объявили на разных уровнях, а в примере, где сначала инициализировали, а потом объявляли переменную, все это делали на одном уровне — уровне процедуры. Поэтому на одном уровне (например, на уровне процедуры) двойное объявление будет ошибкой. А если оно будет сделано на разных уровнях, ошибкой это считаться не будет: речь идет о трех переменных с одним именем, но разными границами видимости.

Теперь необходимо выяснить: если переменная с одним именем объявлена три раза, но разными границами видимости — какая именно переменная будет действовать в процедуре и с каким типом данных? Во всех языках, как правило, всегда действует один принцип: локальная переменная затеняет все остальные (и модульную, и глобальную); модульная переменная затеняет глобальную.

8

Структурные типы данных

Объявление массивов

Массив — это набор элементов определенного типа, у каждого свой порядковый номер, который называется индексом. Различают статические и динамические массивы. Границы статического массива устанавливаются на этапе разработки и могут изменяться только в новой версии программы.

Динамические массивы изменяют свои границы в ходе выполнения программы. С их помощью можно динамически задавать размер массива в соответствии с конкретными условиями. Необходимо учитывать, что работа с динамическими массивами требует специальных затрат на программирование.

Для объявления массива используется оператор **Dim**. В круглых скобках после имени массива указывается его максимальный индекс.

При объявлении массивов используются те же ключевые слова, что и при объявлении переменных: **Dim**, **Static**, **Private** или **Public**. Различие между скалярными переменными и переменными массива состоит в том, что обычно необходимо определить размер массива. Массив, чей размер определен, имеет фиксированный размер и называется статическим. Массив, чей размер может быть изменен в ходе работы программы, называется динамическим.

Индексация массива, если это специально не оговорено, начинается с 0.

На следующей линии кода фиксированный размер массива объявлен как массив **Integer**, имеющий 11 строк и 11 столбцов (так как индексация начинается с 0):

```
Dim MyArray(10, 10) As Integer
```

Первый аргумент представляет строки; второй — столбцы.

Как и любая другая переменная декларации, если не определен тип данных для массива, тип данных элементов в объявленном массиве **Variant**. Каждый числовой элемент **Variant** массива использует 16 байтов. Каждый элемент **Variant** строки

использует 22 байта. Чтобы записывать код по возможности более компактно, необходимо объявить массив явно, чтобы тип данных был явным, а не **Variant**.

Следующие строки кода сравнивают размер массива.

Массив с типом **Integer** использует 22 байта (11 элементов × 2 байта):

```
ReDim MyIntegerArray(10) As Integer
```

Массив с типом двойной точности **Double** использует 88 байтов (11 элементов × 8 байтов):

```
ReDim MyDoubleArray(10) As Double
```

Массив с типом **Variant** использует по крайней мере 176 байтов (11 элементов × 16 байтов):

```
ReDim MyVariantArray(10)
```

Массив с типом **Integer** использует $100 \times 100 \times 2$ байта (20 000 байтов):

```
ReDim MyIntegerArray(99, 99) As Integer
```

Массив с типом двойной точности **Double** использует $100 \times 100 \times 8$ байтов (80 000 байтов):

```
ReDim MyDoubleArray(99, 99) As Double
```

Массив с типом **Variant** использует по крайней мере 160 000 байтов ($100 \times 100 \times 16$ байтов):

```
ReDim MyVariantArray(99, 99)
```

Максимальный размер массива меняется в зависимости от вашей операционной системы и доступной памяти. Использование массива, который превышает размер RAM, доступного в вашей системе, медленнее считывает данные и записывает на диск. В этом случае подключается виртуальная память, а она в любом случае работает медленнее оперативной.

Динамический массив создается в два этапа. Сначала массив определяют в секции (General) (Declarations) контейнера (формы, модуля, класса) без указания размера:

```
' (General) (Declarations)  
Dim MyArray() As Variant
```

Затем с помощью оператора **ReDim** устанавливают фактический размер массива:

```
' (General) (Declarations)  
Dim MyArray () As Variant  
Private Sub Command1_Click()  
ReDim MyArray (50, 10)  
'Код  
End Sub
```

Объявляя динамический массив, можно определить размер массива во время работы программы. Используйте **Static**, **Dim**, **Private** или **Public**, чтобы объявлять динамический массив, оставляя круглые скобки пустыми, как показано в следующем примере.

```
Dim sngArray() As Single
```

Можно использовать утверждение **ReDim**, чтобы объявлять массив в пределах процедуры. Необходимо быть очень осторожным, чтобы не указать имя массива с орфографическими ошибками, если используете утверждение **ReDim**. Даже если **Option Explicit** (явное утверждение выбора) включено в модуль, массив будет создан повторно.

В процедуре в пределах области массива используйте утверждение **ReDim**, чтобы изменить число измерений, определяющее число элементов, и определить верхний и нижний пределы каждого направления. Можно использовать утверждение **ReDim**, чтобы изменять динамический массив в случае необходимости. Всякий раз, когда это делается, существующие значения в массиве будут потеряны. Использование **ReDim** сохраняется, чтобы расширить массив, сохранив возможность его оценки. Например, следующее утверждение расширяет массив **varArray** 10 элементами, не теряя их текущих значений:

```
ReDim Preserve varArray(UBound(varArray) + 10)
```

Когда используется ключевое слово **Preserve** с динамическим массивом, можно изменить только верхнее связанное последнее значение, но нельзя изменять число значений.

Можно объявить, что массив будет работать с установкой величин тех же самых типов данных. Массив — единственная переменная со многими значениями, чтобы загружать величины, тогда как у переменной типа одно значение памяти, в которое можно загрузить только одну величину. Посмотрите на массив в целом, когда необходимо сослаться на все величины, которые он содержит, или можно сослаться на свои индивидуальные элементы.

Например, чтобы хранить ежедневные затраты в течение каждого дня года, можно объявить одну переменную массива с 365 элементами, а не объявлять 365 переменных. Каждый элемент в массиве содержит одну величину. Следующее утверждение объявляет переменную массива curExpense с 365 элементами. По умолчанию массив является индексным и начинается с нуля, так что верхний связанный массив — 364, а не 365:

```
Dim curExpense(364) As Currency
```

Чтобы установить величину индивидуального элемента, необходимо определить элементный индекс. Следующий пример назначает начальную величину 20 в каждый элемент в массиве.

```
Sub FillArray()
    Dim curExpense(364) As Currency
    Dim intI As Integer
    For intI = 0 to 364
        curExpense(intI) = 20
    Next
End Sub
```

Можно использовать утверждение **Option Base** в секции (General) (Declarations) контейнера (формы, модуля, класса), чтобы изменить встроенный индекс первого элемента от 0 до 1. Допустимыми значениями для **Option Base** являются только 0 и 1.

В следующем примере утверждение **Option Base** (База Выбора) изменяет индекс для первого элемента, и **Dim** утверждение объявляет переменную массива **curExpense** с 365 элементами:

```
Option Base 1  
Dim curExpense(365) As Currency
```

Можно также явно установить более низкие значения массива, используя оператор **To**, как показано в следующем примере:

```
Dim curExpense(1 To 365) As Currency  
Dim strWeekday(7 To 13) As String
```

Хранение значения **Variant** в массиве. Есть два варианта создания в массиве значений **Variant**. Первый путь — объявить массив типа данных **Variant**, как показано в следующем примере:

```
Dim varData(3) As Variant  
varData(0) = "Иванов Иван"  
varData(1) = "3-я улица Строителей"  
varData(2) = 38  
varData(3) = Format("06-04-1958", "Текущая дата")
```

Другой путь состоит в том, чтобы назначить массив возвращаемой функцией **Array** с переменной **Variant**, как показано в следующем примере:

```
Dim varData As Variant  
varData = Array(" Кузнецов Олег", "4242 Сиреневый  
бульвар", 38, _  
Format("06-09-1952", "Текущая дата"))
```

Необходимо идентифицировать элементы в массиве с величинами **Variant** индексом независимо от того, какая техника используется для создания массива. Например, следующее утверждение может быть добавлено к любому из предыдущих примеров.

```
MsgBox "Данные для " & varData(0) & " записаны"
```

Использование многомерных массивов. В VBA можно объявлять массивы вплоть до 60 измерений. Например, следующее утверждение объявляет двухмерный, 5 на 10 массив.

```
Dim sngMulti(1 To 5, 1 To 10) As Single
```

Если массив создается как матрица, первый аргумент представляет строки, а второй — колонки. **For...Next** используется в утверждении для обработки многомерного массива. Следующая процедура заполняет двумерный массив **Single** величинами.

```
Sub FillArrayMulti()
    Dim intI As Integer, intJ As Integer
    Dim sngMulti(1 To 5, 1 To 10) As Single
    ' Заполнение массива значениями.
    For intI = 1 To 5
        For intJ = 1 To 10
            sngMulti(intI, intJ) = intI * intJ
            Debug.Print sngMulti(intI, intJ)
        Next intJ
    Next intI
End Sub
```

Свойство **FormulaArray** возвращает или устанавливает формулу массива диапазона. Возвращает единственную формулу или массив Visual Basic. Если определенный диапазон не содержит формулу массива, то возвращает **Null**. Читает/записывает **Variant**.

Если используется это свойство, чтобы вводить формулу массива, формула должна использовать стиль ссылки R1C1, а не стиль ссылки A1 (смотри второй пример).

Этот пример вводит число 3 как константу в массив в ячейки A1:C5 в Книга1.

```
Worksheets ("Книга1") .Range ("A1:C5") .FormulaArray = "=3"
```

Этот пример вводит формулу в массив =SUM(R1C1:R3C3) в ячейки E1:E3 в Книга1.

```
Worksheets ("Лист1") .Range ("E1:E3") .FormulaArray =
"=Sum(R1C1:R3C3)"
```

Свойство **CurrentArray**. Если определенная ячейка является частью массива, возвращает объект **Range**, который представляет целый массив. Только для чтения.

Этот пример считает, что ячейка A1 в Книга1 — активная ячейка и что активная ячейка является частью массива, который включает ячейки A1:A10. Пример выбирает ячейки A1:A10 в Книга1.

```
ActiveCell.CurrentArray.Select
```

Значение True, если определенная ячейка является частью формулы массива. Только для чтения, тип **Variant**. Этот пример отображает сообщение, если активная ячейка на Книга1 — часть массива.

```
Worksheets("Книга1").Activate  
If ActiveCell.HasArray =True Then  
    MsgBox "Активная ячейка является частью массива"  
End If
```

Переопределение динамического массива **ReDim**. Используется в процедуре уровня, чтобы перераспределять пространство хранения для динамических переменных массива.

Синтаксис переопределения **ReDim**:

```
ReDim [Preserve] varname(subscripts) [As type] [,  
varname(subscripts) [As type]] . . .
```

Где:

- **Preserve** — необязательный аргумент, ключевое слово, используемое, чтобы сохранять данные в существующем массиве, когда изменяется размер последнего измерения;
- **varname** — обязательный аргумент, имя переменной, следующей за стандартным переменным присваиванием имен соглашений;
- **subscripts** — обязательный аргумент, измерение переменной массива; может быть объявлено вплоть до 60 значений измерений. Аргумент присваивания использует следующий синтаксис:

```
[lower To] upper [, [lower To] upper] . . .
```

- **type** — необязательный аргумент, тип данных переменной; может быть **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (к настоящему времени не поддерживается), **Date**, **String** (для переменной длины строк), **String** * длины (для фиксированной длины строк), **Object**, **Variant**, определенный тип пользователя или объектный тип. Используйте разделитель **As** в типе для каждой определяемой переменной. Для **Variant**, содержащегося в массиве, тип описывает тип каждого элемента массива, но не преобразует **Variant** в другой тип.

Утверждение **ReDim** используется в определении размера или изменении размеров динамического массива, объявляется с использованием **Private**, **Public** или **Dim** утверждениями с пустыми круглыми скобками.

Утверждение **ReDim** можно использовать многократно, чтобы изменять количество элементов и измерений в массиве. Тем не менее нельзя объявлять массив одного типа данных и позже использовать **ReDim**, чтобы преобразовать массив в другой тип данных, если массив не содержался в **Variant**. Если он содержался в **Variant**, тип элементов может быть изменен, используя в **As** тип, если используется ключевое слово **Preserve** — в этом случае изменения типа данных не разрешены.

Если используется ключевое слово **Preserve**, можно изменить размеры только последнего измерения массива и нельзя изменить число измерений. Например, если массив имеет только одно измерение, можно изменить размеры, поскольку это последнее измерение. Если у массива два или больше измерений, можно изменить размер только последнего измерения и все еще сохранять содержание массива. Следующий пример показывает, как можно увеличить размер последнего измерения динамического массива, не удаляя любые существующие данные, содержащиеся в массиве.

```
ReDim X(10, 10, 10)  
. . .  
ReDim Preserve X(10, 10, 15)
```

Аналогично, когда используется **Preserve**, можно изменить размер массива, изменяя верхние связи; изменение более низких связей — причины ошибок.

При создании массива меньше, чем он был, данные в устраниенных элементах будут потеряны. Если массив передается в процедуру ссылкой, нельзя повторно измерить его в пределах процедуры.

Когда переменные не инициализированы, числовые переменные инициализированы на 0, переменные длины строки инициализированы на нулевую длину строки (" "), и фиксированные длины строк заполнены нулями. Различные переменные инициализированы на **Empty**. Каждый элемент переменной типа, определенного потребителем, инициализирован как будто он был отдельной переменной. Переменная, имеющая отношение к объекту, использовавшая утверждение **Set** прежде, чем она может быть использована, должна быть назначена как существующий объект. Пока не будет назначен объект, объявленная объектная переменная имеет специальную величину **Nothing**. Эта переменная указывает, что она не ссылается на конкретный объект.

Утверждение **ReDim** действует как декларативное утверждение, если переменная не объявлена на модульном уровне или уровне процедуры. Если другая переменная с тем же именем будет создана в дальнейшем, **ReDim** сошлеется на последующую переменную и не вызовет ошибку компиляции, даже если **Option Explicit** остается в силе. Чтобы избегать таких конфликтов, **ReDim** не должен быть использован как декларативное утверждение, но он должен подготавливаться для измерения.

Чтобы изменить массив в **Variant**, нужно явно объявить переменную **Variant** перед попыткой изменения размеров массива.

Метод **Keys**. Возвращает массив существующих ключей в объекте **Dictionary**.

Синтаксис этого метода следующий:

```
object.Keys
```

Где **object** — всегда имя объекта **Dictionary**. Следующий код иллюстрирует использование метода **Keys**:

```

Dim a, d, i           'Создание списка переменных
Set d = CreateObject("Описание.Словарь ")
d.Add "и", "Иванов"   'Добавление списка ключей и строк
d.Add "п", "Петров"
d.Add "с", "Сидорчук"
a = d.keys            'Получение ключей
For i = 0 To d.Count -1 'Повторение массива
    Print a(i)         'Ключ печати
Next
...

```

Метод **Items**. Возвращает массив, содержащий все пункты в объекте **Dictionary**.

Синтаксис этого метода следующий:

object.Items

Где **object** — всегда имя объекта **Dictionary**. Следующий код иллюстрирует использование метода **Items**:

```

Dim a, d, i           'Создание списка переменных
Set d = CreateObject("Описание.Словарь ")
d.Add "и", "Иванов"   'Добавление списка ключей
и пунктов
d.Add "п", "Петров"
d.Add "с", "Сидоров"
a = d.Items            'Получение пунктов
For i = 0 To d.Count -1 'Повторение массива
    Print a(i)         'Печать записи
Next
...

```

Параметр массива может быть использован, чтобы передавать массив аргументов в процедуру. Нельзя знать число элементов в массиве, когда определяется процедура. Применяется при использовании ключевого слова **ParamArray**, чтобы обозначать параметр массива. Массив должен быть объявлен как массив типа **Variant** и быть последним аргументом в процедуре определения. Следующий пример показывает, как можно определить процедуру с параметром массива.

```
Sub AnyNumberArgs(strName As String, ParamArray  
intScores() As Variant)  
    Dim intI As Integer  
    Debug.Print strName; "      Анализ"  
    ' Функция использования UBound, для определения  
    верхнего предела массива.  
    For intI = 0 To UBound(intScores())  
        Debug.Print "      "; intScores(intI)  
    Next intI  
End Sub
```

Практическая работа № 5. Программирование циклических вычислительных процессов с использованием массивов

В следующем примере генерируется массив, заполняемый случайными числами с помощью генератора случайных чисел в диапазоне от -15 до $+15$. Из положительных чисел выбирается минимальное значение, адрес этой ячейки и ее значение выводятся в окно MsgBox (рис. 31).

Создайте Module1. Введите следующий программный код:

```
Sub Poisk_1()  
    Dim x(1 To 20) As Long  
    Dim y As Long  
    Dim min_Item As Long, min_Value As Long  
    For y = 1 To 20 Step 1  
        x(y) = Int((15 - (-15) + 1) * Rnd + (-15))  
    Next y  
    For y = 1 To 20 Step 1  
        ActiveSheet.Cells(y, "A").Value = x(y)  
    Next y  
    For y = 1 To 20 Step 1  
        If x(y) > 0 Then  
            min_Item = y  
            min_Value = x(y)  
            Exit For  
        End If  
    Next y
```

```

Next y

For y = 1 To 20 Step 1
    If x(y) > 0 And x(y) < min_Value Then
        min_Item = y
        min_Value = x(y)
    End If
Next y

MsgBox "Порядковый номер: " & min_Item & vbCrLf &
"Значение: " & min_Value

End Sub

```

	A	B	C	D
1	6			
2	1			
3	2			
4	-7			
5	-6			
6	9			
7	-15			
8	8			
9	10			
10	6			
11	-14			
12	-3			
13	11			
14	9			
15	-4			
16	14			
17	12			
18	-14			
19	14			
20	-4			

Microsoft Excel X
Порядковый номер: 2
Значение: 1

OK

Рисунок 31. Готовый результат

Практическая работа № 6. Программирование с использованием составных пользовательских типов данных

Создание пользовательского типа данных начинается с объявления имени нового пользовательского типа данных. В разделе модуля (General) (Declarations) записывается следующий код:

```
Private Type Rushka
    Наполнитель As String * 15
    Длина As Integer
    Толщина As Integer
    Вес As Double
    Надпись As String * 25
End Type
```

Где **Rushka** — имя нового пользовательского типа данных. Далее следуют параметры этого типа. Каждый параметр имеет имя и тип данных VBA. Завершает объявление пользовательского типа запись End Type.

Под последней строкой объявления пользовательского типа данных следует объявление переменной этого типа.

```
Dim NNM As Rushka
```

После оператора **As** идет не один из встроенных типов данных VBA, а пользовательский тип. Следует отметить, что, как правило, для пользовательских типов данных создается не переменная, а массив. Создавать из-за отдельных переменных новый тип нецелесообразно. Зато для массивов очень выгодно.

Для создания примера перенесите на форму одну командную кнопку и пять текстовых полей **TextBox**. Обработчик щелчка кнопки будет следующий:

```
Private Sub CommandButton1_Click()
    NNM.Наполнитель = "Гелевый"
```

```
NNM.Длина = 250
NNM.Толщина = 10
NNM.Вес = 25
NNM.Надпись = "Stabilo"
TextBox1.Text = NNM.Наполнитель
TextBox2.Text = NNM.Длина
TextBox3.Text = NNM.Толщина
TextBox4.Text = NNM.Вес
TextBox5.Text = NNM.Надпись
End Sub
```

В данной процедуре сначала инициализируется переменная NNM, а затем выводится в поля **TextBox**. При инициализации и при выводе на текстовые поля используется точка между именем переменной и именем инициализируемого параметра.

Пользовательские типы данных в дальнейшем можно использовать для объявления других пользовательских типов данных. Например, срочно потребовался тип ручки с заданным корпусом (например, пластмассовым). О типе корпуса в типе Rushka не объявляли. Поэтому можем объявить новый тип так:

```
Private Type PlastRushka
Параметры As Rushka
Корпус As String * 15
```

Здесь наследуется тип Rushka при создании типа PlastRushka.

9

Объектная модель компонентов Microsoft Office

Коллекции объектов в VBA

Коллекции (**Collection**) в VBA состоят из однотипных объектов. **Collection** — это упорядоченный набор элементов, на который можно ссылаться как на единое целое. Благодаря этому возможно объединение собственных объектов в легко управляемые логические единицы. Некоторые задачи, решаемые с помощью семейств, можно решить также с помощью массивов или переменных, определяемых пользователем, однако коллекции имеют некоторые преимущества:

- коллекциями можно управлять, и они более гибко индексируются;
- методы коллекции позволяют добавлять и удалять объекты;
- коллекции требуют меньше памяти;
- размер коллекций регулируется автоматически (без явного декларирования оператора **ReDim**).

Существуют два типа коллекций — встроенные и коллекции пользователя.

Встроенные коллекции созданы в строгой иерархии, наверху которой расположен объект **Application**, то есть Excel.

Объект **Application** содержит следующие коллекции:

- **AddIns** (Коллекция надстроек);
- **CommandBars** (Коллекция командных панелей);
- **Dialogs** (Коллекция диалогов);
- **Names** (Коллекция имен);
- **Windows** (Коллекция объектов Окно);
- **WorksheetFunction** (Коллекция функций рабочего листа);
- **Workbooks** (Коллекция объектов рабочей книги).

Объект рабочей книги содержит другие коллекции:

- **Chart** (Коллекция диаграмм);
- **Names** (Коллекция имен);
- **Sheets** (Коллекция рабочих листов);
- **Styles** (Коллекция стилей);
- **Windows** (Коллекция объектов Окно).

Объект **Chart** содержит коллекции:

- **Borders** (Коллекция рамок);
- **Font** (Коллекция шрифтов);
- **Interior** (Коллекция фонов).

Полный набор иерархии объектов приведен в справочной системе в разделах: Microsoft Excel Objects, Microsoft Excel Objects (Worksheet), Microsoft Excel Objects (Charts), Microsoft Excel Objects (Shapes), Microsoft Excel Objects (ChartGroups).

Свойства объектов

Свойства объектов находятся в окне **Properties** — *Имя элемента управления*, или объекта.

Так как в ходе программирования будем постоянно пользоваться свойствами объекта, необходимо подробно остановиться на этом понятии. Свойства объекта — это основные характеристики, которыми обладает объект.

Чтобы просмотреть или редактировать свойства объекта, необходимо прежде всего выделить этот объект. Выделение производится либо щелчком левой клавиши по этому объекту в окне **Project**, либо выбирается из списка, который можно просмотреть в окне **Properties** (Свойства), если работаем с элементами управления. В верхней части этого окна сразу под системной полосой есть раскрывающийся список, где находятся все элементы управления, расположенные на форме, и сама форма. Второй способ кажется очень сложным и долгим, но иногда это единственный способ найти нужный элемент управления. Дело в том, что в сложном приложении одна панель с элементами управления может располагаться на другой. Например, нажата Кнопка1 — видна одна панель, а вторая не видна. Если нажата Кнопка2, первая панель не видна, а вторая — видна. В этом случае выделить нижнюю панель (расположенную на панели раньше второй) со всеми элементами управления, расположенными на этой нижней панели, не представляется возможным.

В раскрывающемся списке элементов управления и формы, который расположен под системной полосой, находятся только те элементы управления, которые помещены на данную панель. Покажу на примере.

Форма — основная родительская панель для всех элементов управления, расположенных как на самой форме, так и на других панелях. В нашем примере и Панель1, и Панель2 (со всеми элементами управления, расположенными на них) будут видны, если предварительно щелкнуть по пустому месту формы, а затем открыть раскрывающийся список окна **Properties** (Свойства). Если предварительно выделить одну из панелей, в раскрывающемся списке можно видеть не все элементы управления, расположенные на форме, а только те, которые непосредственно расположены на выделенной форме, включая и саму выделенную форму. Элементы управления, помещенные на каждую из панелей **Frame** (Рамка), в списке формы не видны: они помещены на форму не прямо, а косвенно, то есть находятся на элементе управления **Frame** (Рамка). Здесь используется такое понятие, как **Parent** (Родитель или владелец). Форма — владелец всех элементов управления, помещенных непосредственно на ней, но не прямой владелец элементов, расположенных на других владельцах, например на **Frame** (Рамка). Поэтому элементы управления, расположенные на других владельцах, и не видны в списке элементов управления формы.

Окно **Properties** (Свойства) включает выбранный к настоящему времени объект. Видимы только объекты из активной формы.

Окно свойств состоит из двух столбцов: в правом перечислены названия свойств, в левом — их значения. Редактирование свойства осуществляется либо вручную (например, ввод имени элемента), либо выбором соответствующего поля из списка, либо при помощи диалогового окна настройки свойства.

Окно свойств состоит из двух вкладок:

- **Alphabetic Tab** — в алфавитном порядке включает все свойства для выбранного объекта, который может быть изменен в проекте, а также их текущее значение. Можно

изменить установку свойств, выбирая имя свойства или новое значение.

- **Categorized Tab** — включает все свойства для выбранного объекта категорий. Например, **BackColor**, **Caption** и **ForeColor** — в категории **Appearance** (Внешний вид). Можно сократить список, чтобы видеть категории, можно расширить категорию, чтобы видеть свойства. Когда список расширяется или уменьшается, появляется значок плюс (+) или минус (-) слева от имени категории.

Окно **Properties** (Свойства) включает свойства выделенного объекта. Если выделяется сразу несколько элементов управления, в окне **Properties** (Свойства) выводятся только их общие свойства. Такими свойствами, как правило, являются **Enable** (Доступность), **Visible** (Видимость), **MousePointer** (Вид указателя мыши) и т. д.

10

Создание форм. Обработка событий

События VBA

Программа VBA является событийно ориентированной. Это означает, что каждое действие вызывает какое-то событие, которое в виде сообщения передается в приложение. Приложение анализирует сообщение и выполняет соответствующие действия.

Если посмотрим на первую строку каждой процедуры, обратим внимание, что в ней обязательно присутствуют две части: имя процедуры и через символ подчеркивания — событие, которое запускает эту процедуру.

Самое распространенное событие, которое мы использовали до этого — **Click**, щелчок по какому-то элементу управления. Как правило, щелчок производится по командной кнопке **CommandButton**. Поэтому в первой строке имя процедуры и событие указываются как:

```
Sub CommandButton1_Click()
```

Все события можно разделить на следующие группы:

- события приложения;
- события рабочей книги;
- события рабочего листа;
- события диаграмм;
- события экранных форм;
- события, не связанные с объектами;
- события, связанные с отдельными элементами управления.

В верхней части окна **Code** (Код) находятся два раскрывающихся списка: слева — **Object**, справа — **Procedure**. При работе с событиями мы будем постоянно обращаться к этим спискам, поэтому сразу обратите внимание, где они находятся.

В списке **Object** по умолчанию находятся следующие наименования процедур:

- **General** — во всех окнах **Code** (Код);
- **Workbook** — во всех рабочих книгах (ЭтаКнига);

- **Worksheet** — во всех рабочих листах;
- **Class** — в классе модуля.

В ходе работы в одной процедуре можно создавать указание на другие процедуры, после чего они попадают в список **Object**.

В списке **Procedure** находятся значения событий.

Сборка имени процедуры с событием состоит из двух частей: из списка **Object** выбирается имя процедуры, затем из списка **Procedure** выбирается наименование события.

События приложения

Чтобы пользователь мог работать с событиями уровня приложения, сначала необходимо создать новый модуль класса **Class Module**. Вставка модуля класса осуществляется либо командой **Insert ⇒ Class Module**, либо выбором **Class Module** из списка кнопки **Insert** (Объект) на инструментальной панели **Standard**.

В окне **Project — VBA Project** найдите вставленный класс модуля и выделите его. В окне свойств **Properties** измените имя класса модуля (**Name**) на необходимое. По умолчанию класс модуля получает имя **Class** с очередным номером. В окне **Project — VBA Project** найдите вставленный класс модуля и дважды щелкните по нему левой клавишей. Откроется новое окно **Code**. В списке **Project** найдите значение **Class** и щелкните по нему левой клавишей. Будет сгенерирован код:

```
Private Sub Class_Initialize()  
  
End Sub
```

Установите указатель мыши в крайнее левое положение на первой строке, то есть перед **Private Sub Class_Initialize()**, и нажмите **Enter**, чтобы создать одну пустую строку. Напишите в ней следующий программный код:

```
Public WithEvents App As Application
```

Эта строка должна быть перед строками:

```
Private Sub Class_Initialize()  
End
```

Как только вы переведете указатель мыши на другую строку, немедленно появится разделительная линия, отделяющая одну процедуру от другой. В этой строке App — это имя нового приложения. Вместо App можно написать любое имя (как на русском, так и на английском языке), но мы будем пользоваться именем приложения App.

Установите указатель мыши в конец первой строки и нажмите на Enter. Несмотря на то что мы создали новую пустую строку, казалось бы, в одной процедуре (в той, где объявляем приложение App), эта новая пустая строка уйдет в нижерасположенную процедуру, не обращайте на это внимания. Нам нужно создать объектную переменную, которой будет присвоен объект, объявленный нами в модуле класса. Введите в новой пустой строке следующий программный код:

```
Dim X As New EventClassModule
```

Строка вернется в вышерасположенную процедуру, то есть к Public WithEvents App As Application.

После всех процедур запишите следующие строки:

```
Sub InitializeApp()  
    Set X. App = Application  
End Sub
```

Откройте список **Project** и выберите значение App. В списке **Procedure** появится список событий, относящихся к уровню приложения.

К событиям уровня приложения **Application** относятся (таблица 26).

Таблица 26. События приложения Application

Событие	Описание события
NewWorkbook	Создание новой рабочей книги
SheetActivate	Активизация рабочего листа в открытой рабочей книге
SheetBeforeDoubleClick	Двойной щелчок на одном из рабочих листов открытой рабочей книги
SheetBeforeRightClick	Щелчок правой клавишей мыши на одном из рабочих листов открытой рабочей книги
SheetCalculate	Вычисление или пересчет формул, расположенных на рабочем листе открытой рабочей книги
SheetChange	Происходит при изменении ячеек на любом рабочем листе потребителем или внешней связью
SheetDeactivate	Происходит приdezактивации любого рабочего листа
SheetFollowHyperlink	Выполнение гиперссылки при щелчке по объекту, имеющему гиперссылку
SheetSelectionChange	Изменение выделенного на рабочем листе объекта
SheetPivotTableUpdate	Происходит после того, как будет скорректирован лист сообщения PivotTable
WindowActivate	Активация окна открытой рабочей книги
WindowDeactivate	Деактивация окна открытой рабочей книги
WindowResize	Изменение размеров окна открытой рабочей книги
WorkbookActivate	Активация открытой рабочей книги
WorkbookAddinInstall	Инсталляция рабочей книги в виде надстройки
WorkbookAddinUninstall	Удаление рабочей надстройки
WorkbookBeforeClose	Происходит в процессе закрытия открытой рабочей книги
WorkbookBeforePrint	Происходит в процессе печати открытой рабочей книги
WorkbookBeforeSave	Происходит в процессе сохранения открытой рабочей книги

Таблица 26. События приложения Application. Окончание

Событие	Описание события
WorkbookDeactivate	Деактивация открытой рабочей книги
WorkbookNewSheet	Вставка нового листа в открытую рабочую книгу
WorkbookOpen	Происходит в процессе открытия рабочей книги
WorkbookPivotTableCloseConnection	Закрытие связи с источником данных сводной таблицы
WorkbookPivotTableOpenConnection	Открытие связи с источником данных сводной таблицы

Событие NewWorkbook

Синтаксис:

```
Private Sub object_NewWorkbook(ByVal Wb As Workbook)
```

Где:

- **object** — объект типа Application объявляется событиями в модуле класса. Более подробно смотри в справке по VBA **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Wb** — новая рабочая книга.

Этот пример размещает открытое окно, когда создается новая книга.

Событие SheetActivate

Синтаксис:

```
Private Sub object_SheetActivate(ByVal Sh As Object)
```

Где:

- **object** — Application или Workbook;
- **Sh** — активируемый лист. Может быть объектом Chart или Worksheet.

Событие SheetBeforeDoubleClick

Синтаксис:

```
Private Sub object_SheetBeforeDoubleClick(ByVal Sh As Object, ByVal Target As Range, ByVal Cancel As Boolean)
```

Где:

- **object** — Application или Workbook;
- **Sh** — объект Worksheet, который представляет рабочий лист;
- **Target** — ближайшая к указателю мыши ячейка, когда произошел двойной щелчок;
- **Cancel** — **False**, когда происходит событие. Если процедура события устанавливает этот аргумент в True, двойной щелчок встроенного действия при завершении процедуры не выполняется.

Это событие не происходит на листах диаграммы.

Событие SheetBeforeRightClick

Синтаксис:

```
Private Sub object_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, ByVal Cancel As Boolean)
```

Где:

- **object** — Application или Workbook;
- **Sh** — объект Worksheet, который представляет рабочий лист;
- **Target** — ближайшая к указателю мыши ячейка, когда произошел двойной щелчок;
- **Cancel** — **False**, когда происходит событие. Если процедура события устанавливает этот аргумент в True, двойной щелчок встроенного действия при завершении процедуры не выполняется.

Это событие не происходит на листах диаграммы.

Это событие не происходит, если щелчок правой клавишей мыши производится в то время, когда указатель мыши находится в форме или командной зоне (инструментальной панели или области меню), полосе состояния, именах листов и диаграмм, на полосах прокрутки.

Событие SheetCalculate

Синтаксис:

```
Private Sub object_SheetCalculate(ByVal Sh As Object)
```

Где:

- **object** — Application или Workbook;
- **Sh** — номер рабочего листа. Может быть объектом Chart или Worksheet.

Событие SheetChange

Синтаксис:

```
Private Sub object_SheetChange(ByVal Sh As Object,  
ByVal Source As Range)
```

Где:

- **object** — Application или Workbook;
- **Sh** — объект Worksheet, который представляет рабочий лист;
- **Source** — измененный диапазон.

Это событие не происходит на листах диаграммы.

Событие SheetDeactivate

Синтаксис:

```
Private Sub object_SheetDeactivate(ByVal Sh As Object)
```

Где:

- **object** — Application или Workbook;
- **Sh** — объект Chart или Worksheet, которые представляют диаграмму или рабочий лист.

Событие WorkbookActivate

Синтаксис:

```
Private Sub app_WorkbookActivate(ByVal Wb As Workbook)
```

Где:

- **app** — объект типа Application объявляется событиями в модуле класса. Более подробно смотри в справке по VBA **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Wb** — активируемая книга Workbook.

Событие WorkbookBeforeClose

Синтаксис:

```
Private Sub object_WorkbookBeforeClose(ByVal Wb As Workbook, ByVal Cancel As Boolean)
```

Где:

- **object** — объект типа Application объявляется событиями в модуле класса. Более подробно смотри в справке по VBA **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Wb** — рабочая книга, которая закрывается;
- **Cancel** — **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, рабочая книга не закрывается, даже если процедура завершена.

Событие WorkbookBeforePrint

Синтаксис:

```
Private Sub object_WorkbookBeforePrint(ByVal Wb As  
Workbook, ByVal Cancel As Boolean)
```

Где:

- **object** — объект типа Application объявляется событиями в модуле класса. Более подробно смотри в справке по VBA **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Wb** — рабочая книга;
- **Cancel** — **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, рабочая книга не будет напечатана, даже когда процедура завершена.

Событие WorkbookBeforeSave

Синтаксис:

```
Private Sub object_WorkbookBeforeSave(ByVal Wb As  
Workbook, ByVal SaveAsUi As Boolean, ByVal Cancel As  
Boolean)
```

Где:

- **object** — объект типа Application объявляется событиями в модуле класса. Более подробно смотри в справке по VBA **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Wb** — рабочая книга;
- **SaveAsUi** — **True**, если будет отображено диалоговое окно **Save As**;
- **Cancel** — **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, рабочая книга не будет сохранена, даже когда процедура завершена.

Событие WorkbookDeactivate

Синтаксис:

```
Private Sub object_WorkbookDeactivate(ByVal Wb As  
Workbook)
```

Где:

- объект типа Application объявляется событиями в модуле класса. Более подробно смотри в справке по VBA **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Wb** — рабочая книга.

Событие WorkbookNewSheet

Синтаксис:

```
Private Sub object_WorkbookNewSheet(ByVal Wb As  
Workbook, ByVal Sh As Object)
```

Где:

- объект типа Application объявляется событиями в модуле класса. Более подробно смотри в справке по VBA **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Wb** — рабочая книга;
- **Sh** — новый рабочий лист.

Событие WorkbookOpen

Синтаксис:

```
Private Sub object_WorkbookOpen(ByVal Wb As Workbook)
```

Где:

- объект типа Application объявляется событиями в модуле класса. Более подробно смотри в справке по VBA **Using**

Events with the Application Object (Использование Событий с Прикладным Объектом);

- Wb — рабочая книга.

События рабочей книги

Чтобы работать с событиями рабочей книги, их сначала нужно получить. Для этого в окне Project — VBA Project найдите и выберите имя книги — ЭтаКнига. В окне свойств Properties измените имя рабочей книги (Name) на необходимое. По умолчанию рабочая книга получает имя ЭтаКнига. Имя рабочей книги можно задавать как русскими, так и английскими буквами. В имя можно вводить цифры и символ подчеркивания (_). Но, как правило, имя ЭтаКнига не меняется, так как многие воспринимают его как стандарт обозначения книги.

Дважды щелкните левой клавишей по имени рабочей книги в окне Project — VBA Project. В Procedure появится список событий, относящихся к уровню рабочей книги.

К событиям рабочей книги относятся (таблица 27).

Таблица 27. События рабочей книги

Событие	Описание события
Activate	Активация рабочей книги
AddinInstall	Происходит при инсталляции рабочей книги в виде надстройки
AddinUninstall	Юнинсталляция (удаление) рабочей надстройки
BeforeClose	Происходит в процессе закрытия рабочей книги
BeforePrint	Происходит в процессе печати рабочей книги
BeforeSave	Происходит в процессе сохранения рабочей книги
Deactivate	Деактивация рабочей книги
NewSheet	Происходит при вставке нового листа в рабочую книгу
Open	Происходит в процессе открытия рабочей книги
PivotTableCloseConnection	Происходит при закрытии связи с источником данных сводной таблицы

PivotTableOpenConnection	Происходит при открытии связи с источником данных сводной таблицы
SheetActivate	Происходит при активации рабочего листа рабочей книги
SheetBeforeDoubleClick	Двойной щелчок по листу рабочей книги
SheetBeforeRightClick	Щелчок правой клавишей по рабочему листу рабочей книги
SheetCalculate	Происходит при вычислении или пересчете формул рабочего листа
SheetChange	Происходит при изменении данных рабочего листа пользователем или внешними связями
SheetDeactivate	Деактивация рабочего листа рабочей книги
SheetFollowHyperlink	Происходит при реализации гиперссылки, назначенной данному объекту
SheetPivotTableUpdate	Изменение данных на рабочем листе, содержащем сводную таблицу
SheetSelectionChange	Происходит при изменении объекта, выделенного на рабочем листе
WindowActivate	Происходит при активации окна рабочей книги
WindowDeactivate	Происходит при деактивации окна рабочей книги
WindowResize	Происходит при изменении размеров окна рабочей книги

Событие Activate

Синтаксис:

```
Private Sub object_Activate()
```

Где **object** — **Chart**, **Workbook** или **Worksheet**. Для информации об использовании событий с объектом **Chart** смотри раздел справки VBA **Using Events with the Chart Object** (Использование Событий с Объектом Диаграммы).

Если пользователь переключается между двумя окнами, показывающими ту же рабочую книгу, происходит событие **WindowActivate**, но событие **Activate** для рабочей книги не происходит. Оно не происходит, если пользователь создает новое окно.

Рассмотрим следующий пример. При активации рабочей книги (например, при ее открытии или переключении с одной

рабочей книги на другую) необходимо провести сортировку данных в определенном интервале, например в диапазоне a1:a15 и b1:b15. Сгенерируйте начальный программный код, для чего перейдите в окно **Project-VBAPrj** и выберите рабочую книгу *Эта книга*. В списке **Procedure** выберите событие **Activate** и щелкните по нему. Список событий в окне **Procedure** становится доступным только после создания первого события **Open** (См. описание события **Open**).

Перепишите следующий программный код:

```
Private Sub Workbook_Activate()
Range("a1:a15").Sort Key1:=Range("a1"),
Order1:=xlAscending
Range("b1:b15").Sort Key1:=Range("b1"),
Order1:=xlAscending
End Sub
```

При активизации книги будет происходить сортировка данных в указанном диапазоне, что очень удобно.

Рассмотрим следующий пример. При открытии рабочей книги она сворачивается до значка и ожидает, пока пользователь не развернет ее. Программный код этой процедуры будет такой:

```
Private Sub Workbook_Activate()
ActiveWindow.WindowState = xlMinimized
End Sub
```

Событие BeforeClose

Синтаксис:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
```

Где **Cancel** — **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, закрытие действия прекращается, и рабочая книга остается открытой.

Событие **BeforeClose** происходит перед самым закрытием рабочей книги и следует после события **Deactivate**, если рабочая книга закрывается. Рассмотрим следующий пример.

Очень часто при закрытии рабочей книги требуется сохранять внесенные изменения. Обычный путь — сохранять изменения с помощью встроенной в Excel программы. Она определяет, были ли внесены в рабочую книгу изменения. Если несохраненные изменения есть, программа выдает сообщение о необходимости сохранить эти изменения. Но может наступить событие **BeforeClose**, то есть ваша процедура **Workbook_BeforeClose** может быть игнорирована. Чтобы перехватить управление ситуацией в свои руки, необходимо в процедуре **Workbook_BeforeClose** прямо предусмотреть сохранение изменений в рабочей книге. Пользователь обычно привык к тому, что перед закрытием любой программы при наличии несохраненных данных ему выдается сообщение, что необходимо сохранить изменения. Если изменения в рабочей книге сохраним молча, возможны сомнения, что изменения сохранены. Мы должны сообщить пользователю о сохранении данных, например, с помощью функции **MsgBox**. Вызовите событие **BeforeClose** и перепишите следующий программный код:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    If Me. Saved = False Then
        MsgBox "Рабочая книга сохранена!"
        Me. Save
    End If
End Sub
```

Эта процедура перехватывает управление по сохранению изменений и выдает пользователю сообщение, что изменения сохранены. Пользователь закроет окно **MsgBox** — будет закрыта и рабочая книга. Если в рабочей книге нет изменений, не будет ни сохранения, ни сообщения пользователю о том, что в рабочей книге были изменения.

Событие **BeforePrint**

Синтаксис:

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
```

Где **Cancel** — **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, рабочая книга не будет напечатана, даже когда процедура завершена.

В процедуре можно переопределить единственный аргумент — **Cancel**. Книга будет распечатана, если **Cancel** установлен в **False** (по умолчанию). Если **Cancel** установить в **True**, печать рабочей книги будет отменена.

Разработаем небольшой пример. Его смысл в том, что после запуска печати (например, кнопкой **Печать** на панели **Стандартная** в Excel) на экран выводится диалоговое окно на подтверждение печати рабочей книги. Если пользователь нажимает **Да**, печать будет продолжена, если **Нет**, печать будет сброшена и произойдет возврат в обычный режим редактирования.

Перепишите следующий программный код:

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
Печать = MsgBox("Вы действительно хотите напечатать
рабочую книгу?", _
vbYesNo + vbQuestion)
If Печать = vbNo Then
Cancel = True
End If
End Sub
```

Рассмотрим пример. Перед печатью часто необходимо убедиться в пересчете всех формул в рабочей книге. Поэтому необходимо принудительно произвести пересчет с помощью метода **Calculate**, который вычисляет все открытые рабочие книги, специфические листы в рабочей книге или определенном диапазоне ячеек в рабочей книге.

Перепишите следующий программный код:

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
For Each wk In Worksheets
wk.Calculate
Next
MsgBox "Расчет произведен", vbInformation
End Sub
```

Сначала производится пересчет всех формул и только потом начинается печать.

Событие BeforeSave

Синтаксис:

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUi As Boolean, Cancel As Boolean)
```

Где:

- **SaveAsUi** — **True**, если будет отображено диалоговое окно **Save As**;
- **Cancel** — **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, рабочая книга не будет сохранена, даже когда процедура завершена.

Событие **BeforeSave** наступает перед сохранением рабочей книги. Сохранение рабочей книги может происходить, например, в следующих ситуациях:

- первое сохранение рабочей книги;
- выполнение команды **Файл ⇒ Сохранить**;
- выполнение команды **Файл ⇒ Сохранить как...**;
- закрытие рабочей книги, если в ней имеются несохраненные данные.

В данных случаях открывается диалоговое окно **Сохранение документа**. При этом значение переменной **SaveAsUI** равно **True**. В данной процедуре проверяется значение этой переменной, и если **SaveAsUI** действительно равно **True**, выводится сообщение.

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)
    MsgBox "Нажмите OK"
    If SaveAsUI Then
        End If
    End Sub
```

Рассмотрим следующий пример, а в нем — участие аргумента **Cancel**. После нажатия кнопки **Сохранить** на инструментальной панели **Стандартная** в Excel (или выполнения других аналогичных команд) выводим диалоговое окно, в котором запрашиваем пользователя о подтверждении сохранения данных. Если он нажимает **Да**, рабочая книга сохраняется, если **Нет**, рабочая книга не сохраняется. Если не сохраняется, необходимо установить аргумент **Cancel** в **True**. Если сохраняется, не нужно переопределять аргумент **Cancel**, так как он по умолчанию и так установлен в **False**. Текст процедуры будет следующий:

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)
Сохранение = MsgBox("Вы действительно хотите
сохранить рабочую книгу?", _
vbYesNo + vbQuestion)
If Сохранение = vbNo Then
Cancel = True
End If
End Sub
```

Такое сложное сохранение данных может потребоваться, например, в книгах, имеющих особо важное значение, так как удаленную книгу еще можно восстановить, но ошибочно измененную — никогда.

Событие Deactivate

Синтаксис:

```
Private Sub object_Deactivate()
```

Где: **object** — **Chart**, **Workbook** или **Worksheet**. Для информации об использовании событий с объектом **Chart** смотри раздел справки VBA **Using Events with the Chart Object** (Использование Событий с Объектом Диаграммы).

Составим пример, где обрабатывались бы какие-нибудь действия при деактивации рабочей книги. Деактивацией может быть закрытие этой книги или переход в другую. Перепишите следующий программный код:

```
Private Sub Workbook_Deactivate()
MsgBox "Караул! Закрывают! Спасите!", vbInformation
End Sub
```

Пример хотя и шуточный, но показывает возможные операторы при деактивации рабочей книги.

Событие NewSheet

Синтаксис:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
```

- Где **Sh** — новый рабочий лист. Может быть объектом **Worksheet** или **Chart**.

Рассмотрим следующий пример. При вставке нового рабочего листа выдается соответствующее сообщение. В ячейки C1 и D1 вставляются текущие дата и время, затем столбцы C и D выравниваются по ширине под конкретную дату и время.

Ширина столбцов зависит от выбранного формата представления даты и времени, поэтому угадать вручную нужную ширину столбцов нельзя. Если не установить автоширину столбцов, данные будут представлены в виде решеток (или диезов) — #####. Программный код этой процедуры такой:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
MsgBox "Вы вставили новый лист!"
Range("C1") = Date
Range("D1") = Time
Columns("C:C").EntireColumn.AutoFit
Columns("D:D").EntireColumn.AutoFit
End Sub
```

Событие Open

Синтаксис:

```
Private Sub Workbook_Open()
```

Это событие настолько распространено в рабочей книге, что оно предлагается по умолчанию, если в списке **Object** (в окне **Code**) вы выбираете значение **Workbook**, поэтому заготовка создается сразу.

Рассмотрим пример. В окне **Project-VBAProject** выберите рабочую книгу *Эта книга*. В списке **Object** выберите значение **Workbook**. Первоначально в списке два значения: **Workbook** и **General**, выбор несложный. Немедленно будет сгенерирована программная заготовка для события **Open**. Впишите программный код:

```
Private Sub Workbook_Open()
    MsgBox "Добрый день. Не забудьте сделать отчет",
    vbInformation
End Sub
```

Сохраните приложение, скомпилируйте его и закройте рабочую книгу, в которой создан данный макрос. Снова откройте эту рабочую книгу — при открытии будет выдаваться сообщение.

В этом же событии можно написать и так:

```
Private Sub Workbook_Open()
    MsgBox "Рабочая книга по учету черного нала. —
    Для представления в налоговую инспекцию",
    vbInformation
End Sub
```

Событие **Open** может использоваться для решения многих задач: открытие рабочих книг и рабочих листов, всевозможные настройки меню и т. д.

При нажатой во время загрузки клавиши Shift процедура **Workbook_Open** не выполняется.

Событие **PivotTableCloseConnection**

Синтаксис:

```
Private Sub expression_PivotTableCloseConnection(ByVal
    Target As PivotTable)
```

Где:

- **expression** — переменная, на которую ссылается объект **Workbook**, объявленный событиями в модуле класса;

- **Target** — обязательный аргумент, выбранное сообщение PivotTable.

Событие PivotTableOpenConnection

Синтаксис:

```
Private Sub expression_PivotTableOpenConnection(ByVal  
Target As PivotTable)
```

Где:

- **expression** — переменная, на которую ссылается объект **Workbook**, объявленный событиями в модуле класса;
- **Target** — обязательный аргумент, выбранное сообщение PivotTable.

Событие SheetActivate

Синтаксис:

```
Private Sub object_SheetActivate(ByVal Sh As Object)
```

Где:

- **object** — **Application** или **Workbook**;
- **Sh** — номер рабочего листа. Может быть объектом **Chart** или **Worksheet**.

Рассмотрим пример. При переходе на новый лист пользователь получает сообщение о переходе. Одновременно происходит сортировка данных на этом листе. Если пользователь сделал изменения на другом листе, при возврате на тот лист неизбежно произойдет активация этого листа и также будет организована сортировка.

```
Private Sub Workbook_SheetActivate(ByVal Sh As  
Object)  
On Error Resume Next  
MsgBox "Вы перешли на новый лист!"  
Range("a1:a15").Sort Key1:=Range("a1"),
```

```
Order1:=xlAscending  
Range("b1:b15").Sort Key1:=Range("b1"),  
Order1:=xlAscending  
End Sub
```

Событие SheetBeforeDoubleClick

Синтаксис:

```
Private Sub object_SheetBeforeDoubleClick(ByVal Sh As  
Object, ByVal Target As Range, ByVal Cancel As Boolean)
```

Где:

- **object** — **Application** или **Workbook**. Более подробно об использовании событий с объектом **Application** смотри **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Sh** — объект **Worksheet**, который представляет лист;
- **Target** — ячейка, ближайшая к указателю мыши при двойном щелчке;
- **Cancel** — **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, двойной щелчок встроенного действия не выполняется, даже когда процедура завершена.

Рассмотрим небольшой пример. При двойном щелчке левой клавишей выдается сообщение и анализируется, какая кнопка нажата в диалоговом окне **MsgBox**. Если нажата **Да**, выделяется ближайшая ячейка, рядом с которой происходил двойной щелчок. Двойной щелчок не обязательно происходит в конкретной ячейке. Он может быть и на границе сразу нескольких ячеек. Поэтому происходит определение ячейки, которая ближе всего к точке щелчка. Если двойной щелчок производится по границе ячеек, событие **SheetBeforeDoubleClick** не генерируется, и процедура по обработке этого события не вызывается.

Если пользователь в окне **MsgBox** нажимает **Нет**, выделения ячейки не происходит.

Перепишите следующий программный код:

```
Private Sub Workbook_SheetBeforeDoubleClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)
Двойной_щелчок = MsgBox("Двойной щелчок произведен",
vbYesNo + vbInformation)
If Двойной_щелчок = vbNo Then
Cancel = True
End If
End Sub
```

Событие SheetBeforeRightClick

Синтаксис:

```
Private Sub object_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, ByVal Cancel As Boolean)
```

Где:

- **object** — **Application** или **Workbook**. Более подробно об использовании событий с объектом **Application** смотри **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Sh** — объект **Worksheet**, который представляет рабочий лист;
- **Target** — ячейка, ближайшая к указателю мыши при щелчке правой клавишей;
- **Cancel** — **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, щелчок правой клавишей мыши встроенного действия не выполняется, даже когда процедура завершена.

Подобно другим событиям рабочей книги, это событие не происходит, если щелчок правой клавишей мыши производится, когда указатель мыши находится в форме или командной зоне (инструментальной панели или области меню), полосе состояния, именах листов и диаграмм, на полосах прокрутки.

Свойство **SheetBeforeRightClick** очень похоже на свойство **SheetBeforeDoubleClick**. Это и не удивительно, так как они выполняют одинаковую работу.

Рассмотрим следующий небольшой пример. При щелчке правой клавишей мыши выдается сообщение и анализируется, какая кнопка нажата в диалоговом окне **MsgBox**. Если **Да**, открывается встроенное в Excel контекстное меню. Если **Нет**, контекстное меню не открывается.

Перепишите следующий программный код:

```
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)
    Правый = MsgBox("Правый щелчок произведен", vbYesNo + vbInformation)
    If Правый = vbNo Then
        Cancel = True
    End If
End Sub
```

Событие SheetCalculate

Синтаксис:

```
Private Sub object_SheetCalculate(ByVal Sh As Object)
```

Где:

- **object** — **Application** или **Workbook**. Более подробно об использовании событий с объектом **Application** смотрите **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Sh** — лист. Может быть объектом **Chart** или **Worksheet**.

Рассмотрим пример. Пересчет формул на рабочем листе требуется при большом наборе различных ситуаций. В этом примере мы произведем сортировку данных в диапазоне a1:a10. Заполните числами ячейки a1:a10 и b1:b10. В столбце С в диапазоне c1:c10 напишите любую формулу. Затем создайте заготовку под событие **SheetCalculate** и допишите программный код:

```
Private Sub Workbook_SheetCalculate(ByVal Sh As Object)
With Worksheets(1)
Range("a1:a10").Sort Key1:=Range("a1"),
Order1:=xlAscending
End With
End Sub
```

При изменении любого значения в столбце А производится сортировка по возрастанию числовых значений.

Событие SheetChange

Синтаксис:

```
Private Sub object_SheetChange(ByVal Sh As Object,
ByVal Source As Range)
```

Где:

- **object** — **Application** или **Workbook**. Более подробно об использовании событий с объектом **Application** смотри **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Sh** — объект **Worksheet**, который представляет рабочий лист;
- **Source** — измененный диапазон.

Это событие не происходит на отдельных листах диаграммы. Оно происходит при изменении объекта на рабочем листе. В нашем примере выведем пользователю сообщение о том, что на рабочем листе произведено изменение, и отсортируем измененный столбец.

Создайте заготовку программного кода для этого события и перепишите программный код:

```
Private Sub Workbook_SheetChange(ByVal Sh As Object,
ByVal Target As Range)
MsgBox "Вы произвели изменение"
With Worksheets(1)
```

```
.Range("a1:a10").Sort Key1:=.Range("a1"),  
Order1:=xlAscending  
End With  
End Sub
```

Сначала выводится сообщение об изменении объекта, а затем производится сортировка.

Обратите внимание: сортировка диапазона (по сравнению с примером, рассмотренным в событии **SheetCalculate**) внешне выглядит быстрее, так как мы тратим некоторое время на то, чтобы нажать **OK** в окне **MsgBox**.

Событие SheetDeactivate

Синтаксис:

```
Private Sub object_SheetDeactivate(ByVal Sh As  
Object)
```

Где:

- **object** — **Application** или **Workbook**;
- **Sh** — лист. Может быть объектом **Chart** или **Worksheet**.

Происходит при дезактивации листов. Здесь необходимо специально немного остановиться. Дезактивация рабочих листов (или отдельных листов диаграмм) происходит только при переходе с листа на лист, а не при закрытии книги. Некоторые пользователи путают эти события и не получают запуска процедуры с данным событием при закрытии рабочей книги. Создадим пример с использованием этого события. При переходе с одного рабочего листа на другой пользователь получает сообщение, что он покинул Лист№ (с указанием этого листа), его за это пожурят. Создайте заготовку программного кода для этого события и допишите процедуру:

```
Private Sub Workbook_SheetDeactivate(ByVal Sh As  
Object)  
MsgBox "Вы покинули " & Sh. Name & " И ты, Брут!"  
End Sub
```

В этом примере мы использовали в качестве оператора конката-
нации символ &, хотя можем использовать и символ +. Причем
в одной строке мы можем использовать и тот и другой символы
конкатенации. Например:

MsgBox «Вы покинули « + Sh. Name & « И ты, Брут!»

Событие SheetFollowHyperlink

Синтаксис:

```
Private Sub Workbook_SheetFollowHyperlink(ByVal Sh  
As Object, ByVal Target As Hyperlink)
```

Где:

- **Sh** — обязательный объект, объект рабочей книги (**Worksheet**), который содержит гиперссылку;
- **Target** — обязательная гиперссылка, объект **Hyperlink**, которая представляет расположение гиперссылки.

Событие происходит, если в ячейке находится гиперссылка.
Рассмотрим пример, где событие генерируется только в том
случае, если в ячейке находится гиперссылка. Необходимо
предварительно вставить гиперссылку в одну из ячеек. Выдели-
те эту ячейку. В Excel выполните команду **Вставка** ⇒ **Гипер-
ссылка**. В открывшемся диалоговом окне выберите файл,
который должен открыться при щелчке по ячейке, содержащей
гиперссылку, и нажмите **OK**.

Перепишите программный код:

```
Private Sub Workbook_SheetFollowHyperlink(ByVal Sh  
As Object, ByVal Target As Hyperlink)  
    MsgBox "Гиперссылка"  
End Sub
```

Если в практических примерах необходимо провести какие-то
дополнительные операции, их также можно записать в этой
процедуре. Последовательность событий следующая: сначала
выполняется гиперссылка, затем деактивация окна рабочей

книги, открытие книги, указанной в гиперссылке и, наконец, выдается наше сообщение.

Событие SheetPivotTableUpdate

Синтаксис:

```
Private Sub expression_SheetPivotTableUpdate(ByVal  
Sh As Object, Target As PivotTable)
```

Где:

- **expression** — переменная, на которую ссылается объект типа **Application** или **Workbook**, объявленный в модуле класса;
- **Sh** — обязательный аргумент, выбранный лист;
- **Target** — обязательный аргумент, выбранное сообщение **PivotTable**.

Это событие происходит при изменении значений сводной таблицы. Сводная таблица создается на основании данных какого-то диапазона ячеек, баз данных, других сводных таблиц и т. д. Поэтому необходимо сообщать пользователю об изменении таких данных как в сводной таблице, так и в диапазоне — источнике данных. Изменения в сводной таблице на основании изменений в диапазоне данных происходят по команде **Обновить данные**. Если в исходном диапазоне произошли изменения, пользователь должен узнать о них. Мы не будем создавать излишне сложное задание — просто сообщим пользователю о наличии таких изменений. В реальных приложениях можно сообщать и адреса измененных ячеек, и изменение значений. Итак, программный код в нашем примере будет такой:

```
Private Sub Workbook_SheetPivotTableUpdate(ByVal Sh  
As Object, ByVal Target As PivotTable)  
    MsgBox "Изменение листа"  
End Sub
```

Событие WindowActivate

Синтаксис:

```
Private Sub object_WindowActivate(ByVal Wb As Excel.  
Workbook, ByVal Wn As Excel.Window)
```

Где:

- **object** — **Application** или **Workbook**. Более подробно об использовании событий с объектом **Application** смотри **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Wb** — используется только с объектом **Application**. Рабочая книга отображается в активном окне;
- **Wn** — активируемое окно.

Если аргумента **Application** нет, используется только аргумент **Wn**. Аргумент **Wn** определяет вариант окна для рабочей книги и раскрытие окна (**xlNormal**, **xlMaximized** или **xlMinimized**), то есть положения **Свернуть**, **Развернуть**, **Свернуть до значка** на системной полосе рабочего окна. Аргумент **Wb** определяет эти же положения, но для всего приложения.

Создайте заготовку программного кода для события **WindowActivate**. Запишите программный код:

```
Private Sub Workbook_WindowActivate(ByVal Wn As  
Window)  
Wn.WindowState = xlNormal  
End Sub
```

Событие WindowDeactivate

Синтаксис:

```
Private Sub object_WindowDeactivate(ByVal Wb As  
Excel.Workbook, ByVal Wn As Excel.Window)
```

Где:

- **object** — **Application** или **Workbook**. Более подробно об использовании событий с объектом **Application** смотрите **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Wb** — использование только с объектом **Application**. Рабочая книга отображается в деактивированном окне;
- **Wn** — деактивируемое окно.

Если аргумента **Application** нет, используется только аргумент **Wn**. Он определяет вариант окна для рабочей книги и вариант хранения деактивированного окна (**xlNormal**, **xlMaximized** или **xlMinimized**), есть положения **Свернуть**, **Развернуть**, **Свернуть до значка** на системной полосе рабочего окна. Аргумент **Wb** определяет эти же положения, но для всего приложения.

Создайте заготовку программного кода для события **WindowDeactivate**. Запишите программный код:

```
Private Sub Workbook_WindowDeactivate(ByVal Wn As  
Window)  
Wn.WindowState = xlMinimized  
End Sub
```

В этом примере при переходе от данной рабочей книги к другой она будет храниться в виде значка до тех пор, пока не будет активирована. Событие активации описывается в процедуре **Workbook_WindowActivate**. Эти события — **WindowActivate** и **WindowDeactivate**, как правило, описываются вместе, так как являются двумя сторонами одного и того же процесса.

Событие **WindowResize**

Синтаксис:

```
Private Sub object_WindowResize(ByVal Wb As Excel.  
Workbook, ByVal Wn As Excel.Window)
```

Где:

- **object** — **Application** или **Workbook**. Более подробно об использовании событий с объектом **Application** смотри **Using Events with the Application Object** (Использование Событий с Прикладным Объектом);
- **Wb** — использованное только с объектом **Application**. Рабочая книга отображается в измененном окне приложения;
- **Wn** — изменяемое окно.

Рассмотрим небольшой пример. В этом примере при попытке изменения рабочего окна рабочей книги выдается сообщение о том, что это запрещено. После этого окно раскрывается во весь экран. Так как окно раскрывается во весь экран, это опять приводит к возникновению события **WindowResize**, и сообщение выдается второй раз.

```
Private Sub Workbook_WindowResize(ByVal Wn As  
Window)  
MsgBox "Размер окна изменился!"  
Wn.WindowState = xlMaximized  
End Sub
```

Чтобы сообщение второй раз не выдавалось, необходимо блокировать второе событие **WindowResize**.

События рабочего листа

Чтобы работать с событиями рабочего листа, их сначала нужно получить. Для этого в окне **Project — VBA Project** выберите имя рабочего листа — Лист (Номер листа). В окне свойств **Properties** измените имя рабочего листа (**Name**), если это необходимо. По умолчанию рабочие листы получают имя Лист (Номер листа). Имя можно задавать как русскими, так и английскими буквами, вводить цифры и символ подчеркивания (_). В круглых скобках отражается имя этого же листа в Microsoft Excel. Если в Microsoft Excel вы его переименуете, в VBA имя листа в круглых скобках также изменится.

Теперь дважды щелкните левой клавишей по имени рабочего листа в **Project — VBA Project**. В **Procedure** появится список событий, относящихся к уровню рабочего листа.

По умолчанию будем создавать процедуры к **Лист1**. Если потребуется создавать процедуры для других листов, это будет оговорено особо.

К событиям рабочего листа относятся (таблица 28).

Таблица 28. События рабочего листа

Событие	Описание события
Activate	Происходит при активации рабочего листа
BeforeDoubleClick	Происходит при двойном щелчке по рабочему листу
BeforeRightClick	Происходит при щелчке правой клавишей по рабочему листу
Calculate	Происходит при вычислении и пересчете формул рабочего листа
Change	Изменения, внесенные в ячейки рабочего листа пользователем или внешними связями
Deactivate	Происходит при деактивации рабочего листа
FollowHyperlink	Происходит при реализации гиперссылки, назначенной данному объекту
PivotTableUpdate	Происходит при изменении сводной таблицы, содержащейся на данном рабочем листе
SelectionChange	Происходит при изменении объекта, выделенного на рабочем листе

Событие **Activate**

Синтаксис:

```
Private Sub object_Activate()
```

Где **object** — **Chart**, **Workbook** или **Worksheet**. Об использовании событий с объектом **Chart** смотри раздел справки VBA **Using Events with the Chart Object** (Использование Событий с Объектом Диаграммы).

Событие **Activate** генерируется, когда рабочий лист, для которого написана процедура, получает активность. Не следует

путать это событие с **Activate** для рабочей книги. Если **Activate** для рабочей книги генерируется при активации любого рабочего листа, включая диаграммы на отдельных листах для данной рабочей книги, то событие **Activate** для рабочего листа генерируется только в том случае, если активным становится именно этот рабочий лист, для которого написана процедура обработки события **Activate**, а не любой другой.

Рассмотрим следующую ситуацию. Предположим, что в рабочей книге три рабочих листа. Если напишем процедуру для события **Activate** для рабочей книги, при переходе с Лист1 на Лист2 будет сгенерировано событие **Activate** для Лист2, так как именно этот лист получит активность. Если после этого перейдем на Лист3, также будет сгенерировано событие **Activate** для Лист3: теперь уже этот лист получит активность и будет запущена процедура, обрабатывающая это событие.

Если обрабатывать событие **Activate** для Лист1, при переходе с Лист1 на Лист2 произойдет активация Лист2, но генерация события **Activate** для Лист2 будет, а вот запуска процедуры для этого события не будет: ведь мы написали процедуру только для Лист1. При переходе с Лист2 на Лист3 генерация события **Activate** производиться будет, но запуска процедуры обработки этого события не случится: процедура обработки события **Activate** написана только для Лист1. Если вернемся на Лист1, будет сгенерировано событие **Activate** и запущена процедура обработки этого события. Напишем пример обработки такого события для Лист1. Перепишите следующий программный код:

```
Private Sub Worksheet_Activate()
    MsgBox Name & " получил активность"
    With Worksheets(1)
        .Range("a1:a10").Sort Key1:=.Range("a1"),
        Order1:=xlAscending
    End With
    Columns("A:A").EntireColumn.AutoFit
    Columns("B:B").EntireColumn.AutoFit
End Sub
```

Для события **Activate** для листов, включая диаграммы на отдельных листах для данной рабочей книги, можно записать

процедуру для каждого рабочего листа и листа диаграмм. Для рабочей книги для события **Activate** такое невозможно.

Событие **BeforeDoubleClick**

Синтаксис:

```
Private Sub expression_BeforeDoubleClick(ByVal  
Target As Range, Cancel As Boolean)
```

Где:

- **expression** — переменная, на которую ссылается объект типа **Worksheet**, объявленный в модуле класса;
- **Target** — обязательный аргумент, ячейка, ближайшая к указателю мыши при двойном щелчке;
- **Cancel** — необязательный аргумент, **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, при двойном щелчке встроенное действие не выполнится, даже когда процедура завершена.

Не следует путать событие **BeforeDoubleClick** с другими, в которых производится двойной щелчок правой клавишей. Событие **BeforeDoubleClick** для рабочего листа действует только для рабочих листов. Это нужно понимать буквально — событие **BeforeDoubleClick** генерируется для рабочих листов, а не для листов, на которых расположены отдельные диаграммы. Событие **BeforeDoubleClick** генерируется только для рабочего листа, для которого написана процедура.

Рассмотрим небольшой пример. При двойном щелчке левой клавишей выдается сообщение и анализируется, какая кнопка нажата в диалоговом окне **MsgBox**. Если **Да**, выделяется ближайшая ячейка, рядом с которой происходил двойной щелчок. Двойной щелчок не обязательно происходит в конкретной ячейке. Он может быть и на границе сразу нескольких ячеек. Происходит определение ячейки, которая ближе всего к точке щелчка. Если двойной щелчок производится по границе ячеек, событие **BeforeDoubleClick** не генерируется, и процедура по обработке этого события не вызывается.

Если пользователь в окне **MsgBox** нажимает **Нет**, выделения ячейки не происходит.

Перепишите следующий программный код:

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
Двойной щелчок = MsgBox("Выделить ячейку?", vbYesNo + vbQuestion)
If Двойной щелчок = vbNo Then
Cancel = True
End If
End Sub
```

Если созданы процедуры обработки двойного щелчка и для рабочей книги, и для рабочего листа, сначала генерируется событие для рабочего листа, и только потом — для рабочей книги.

При нажатой клавише Shift и двойном щелчке левой клавишей событие **BeforeDoubleClick** не генерируется, процедура для этого события не вызывается.

По умолчанию аргумент **Cancel** установлен в **False**. При установке этого аргумента в **True** событие **BeforeDoubleClick** блокируется, и процедура не вызывается.

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
Cancel = True
Двойной = MsgBox("Двойной щелчок отключен",
vbInformation)
End Sub
```

Некоторым этот парадокс покажется странным. С одной стороны, совершенно очевидно, что событие все-таки генерируется, сообщение **MsgBox** появляется. С другой — мое утверждение о том, что событие блокируется. На самом деле никакого противоречия здесь нет. При двойном щелчке левой клавишей генерируется событие **BeforeDoubleClick**, и начинает выполняться процедура для этого события. Затем, установив аргумент

Cancel в **True**, встроенное действие блокируется. Встроенное действие — предусмотренное действие при двойном щелчке в Excel. Если в процедуре создать еще несколько окон **MsgBox**, все они будут выполнены, но это выполнение ни о чем не говорит: они-то будут выполнены, а вот встроенное действие — нет. Если у вас встроенное действие для двойного щелчка не заложено, это может выглядеть незаметно и не совсем понятно.

Событие BeforeRightClick

Синтаксис:

```
Private Sub expression_BeforeRightClick (ByVal Target  
As Range, Cancel As Boolean)
```

Где:

- **expression** — переменная, на которую ссылается объект типа **Worksheet**, объявленный в модуле класса;
- **Target** — обязательный аргумент, ячейка, ближайшая к указателю мыши при щелчке правой клавишей;
- **Cancel** — необязательный аргумент, **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, при щелчке правой клавишей встроенное действие не выполнится, даже когда процедура завершена.

Это событие не происходит, если щелчок правой клавишей производится, когда указатель мыши находится в форме или командной зоне (инструментальной панели или области меню), полосе состояния, именах листов и диаграмм, на полосах прокрутки.

Событие **BeforeRightClick** очень похоже на **BeforeDoubleClick**. Эти два события определяют возможность перегрузки мыши. Как правило, правой клавишей мыши вызываются контекстные меню.

Рассмотрим следующий пример. При щелчке правой клавишей выдается сообщение и анализируется, какая кнопка нажата в диалоговом окне **MsgBox**. Если **Да**, открывается встроенное в Excel контекстное меню. Если **Нет**, контекстное меню не открывается.

Перепишите следующий программный код:

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
    MsgBox " Нажата правая клавиша мыши"
End Sub
```

Принципиальное отличие: при двойном щелчке левой клавишей и нажатой клавише Shift событие **BeforeDoubleClick** не генерируется, а **BeforeRightClick** будет сгенерировано и при нажатии всех трех служебных клавиш — Shift, Ctrl и Alt.

Как правило, щелчок правой клавишей вызывает контекстное меню. Но его появление можно и отключить. Рассмотрим пример:

```
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
    Cancel = True
    Правая = MsgBox ("Контекстное меню отключено",
    vbInformation)
End Sub
```

Здесь аргумент **Cancel**, который по умолчанию имеет значение **False**, устанавливается в **True**. В этом случае событие **BeforeRightClick** блокируется, и процедура для него не выполняется.

Событие Calculate

Синтаксис:

```
Private Sub object_Calculate()
```

Где **object** — **Chart** или **Worksheet**. Об использовании событий с объектом **Chart** смотри [Using Events with the Chart Object](#) ([Использование Событий с Объектом Диаграммы](#)).

Рассмотрим следующий пример. Пересчет формул на рабочем листе требуется во многих случаях. В этом примере будем производить сортировку данных в диапазоне a1:a100. Заполните числами ячейки a1:a100 и b1:b100. В столбце C в диапазоне

c1:c100 напишите любую формулу. Затем создайте заготовку под событие **Calculate** и допишите следующий программный код:

```
Private Sub Workbook_Calculate(ByVal Sh As Object)
With Worksheets(1)
Range("a1:a100").Sort Key1:=Range("a1"),
Order1:=xlAscending
End With
Columns("A:A").EntireColumn.AutoFit
Columns("B:B").EntireColumn.AutoFit
Columns("C:C").EntireColumn.AutoFit
End Sub
```

При изменении любого значения в столбце А производится сортировка по возрастанию числовых значений.

Событие Change

Синтаксис:

```
Private Sub Worksheet_Change(ByVal Target As Range)
```

Где **Target** — измененный диапазон. Может быть более чем одной ячейкой.

Это событие не происходит, если ячейки изменяются в течение пересчета. Используйте событие **Calculate**, чтобы перехватывать операцию пересчета листа.

Это событие генерируется, если значение хотя бы одной ячейки рабочего листа изменяется любым способом. Если на рабочий лист вставляется новый объект или происходит пересчет формул, событие **Change** не наступает.

```
Private Sub Worksheet_Change(ByVal Target As Range)
MsgBox "Ячейка " & Target.Address & " изменена"
End Sub
```

Причем процедура будет находить на рабочем листе изменения, произведенные любым способом. Например, если вы в ячейку A1 ввели число и нажали Enter, это будет первое сообщение. Если вернулись в эту же ячейку и ввели (или исправили)

другое число, это второе сообщение. Во всех этих случаях в окне **MsgBox** будет указываться только одна ячейка. Если в другой книге (листе, диапазоне) вы скопировали диапазон ячеек, например с A1 по A 100, и вставляете на рабочий лист, для которого мы написали процедуру обработки события **Change**, в этот же диапазон A1:A100, в окне **MsgBox** будет сообщение именно об этом интервале A1:A100, а не 100 отдельных сообщений об изменении каждой ячейки отдельно.

Событие **Change** наступает только при изменении содержания ячеек. Объясню на этом же примере с использованием написанной процедуры. Если мы введем в ячейку A1 число 25 и нажмем Enter, будет сгенерировано событие **Change** и выведено окно **MsgBox**. Если изменим содержимое ячейки A1 любым способом, опять будет сгенерировано событие **Change**. Если скопируем диапазон ячеек из любого места книги, листа и вставим этот диапазон, начиная с ячейки A1, опять будет сгенерировано событие **Change**. Все эти действия приводят к событию **Change**, так как изменяется содержимое ячейки. Даже если мы дважды щелкнем по ячейке A1 с целью откорректировать содержимое, но саму корректировку делать не будем, а опять нажмем Enter, то и в этом случае будет генерироваться событие **Change**. Но если в ячейке A1 будем делать форматирование — изменим начертание шрифта, его цвет или изменим формат ячейки, например на денежный, — генерирования события **Change** не будет, так как мы не изменяем содержания этой ячейки и, следовательно, не изменяем внутреннего содержания ячейки.

Правда, и здесь имеются некоторые накладки: при очистке форматов командой **Правка ⇒ Очистить ⇒ Форматы** почему-то генерируется событие **Change**.

Существует еще несколько мелких подобных накладок при работе с ячейками.

Необходимо также отметить, что событие **Change** касается ячеек, а не объектов. Если мы попробуем на листе Excel поместить геометрическую фигуру (прямоугольник, овал) или формулу с помощью Microsoft Equation, генерирования события **Change** не будет, так как мы не изменяем содержимого ячеек.

Событие Deactivate

Синтаксис:

```
Private Sub object_Deactivate()
```

Где **object** — **Chart**, **Workbook** или **Worksheet**. Для информации об использовании событий с объектом **Chart** смотри раздел справки VBA **Using Events with the Chart Object** (Использование Событий с Объектом Диаграммы).

Событие **Deactivate** не нужно путать с событием **SheetDeactivate**. Событие **Deactivate** является событием конкретного листа, а событие **SheetDeactivate** является событием листа в рабочей книге. Рассмотрим это более подробно. Если создать процедуру для события **SheetDeactivate**, это будет относиться к каждому листу рабочей книги. А событие **Deactivate** относится только к тому листу, для которого написана процедура. Например, предположим, что в нашей рабочей книге три листа. Процедура **object_SheetDeactivate** будет генерироваться при деактивации каждого из этих трех листов. Процедура **object_Deactivate**, написанная для рабочего листа Лист1, будет генерироваться только при деактивации Лист1, а при деактивации Лист2 и Лист3 генерироваться не будет. Вот пример создания такой процедуры. Сгенерируйте заготовку программного кода для события **Deactivate**. Программный код можно написать такой:

```
Private Sub Worksheet_Deactivate()
MsgBox Name & " потерял активность"
With Worksheets(1)
    .Range("a1:a10").Sort Key1:=.Range("a1"),
    Order1:=xlAscending
End With
End Sub
```

В данном примере при деактивации рабочего листа Лист1 происходит сортировка заданного интервала. Это очень удобно, так как пользователь освобождается от необходимости помнить о сортировке листа и при повторной его активации получает уже отсортированные данные.

Событие SelectionChange

Синтаксис:

```
Private Sub Worksheet_SelectionChange(ByVal Target  
As Excel.Range)
```

Где **Target** — новый выбранный диапазон.

Данное событие генерируется при выделении пользователем ячейки или диапазона ячеек. В нашем примере при выделении ячейки она становится самой верхней левой ячейкой на рабочем листе. Сгенерируйте программный код для события **SelectionChange** и впишите программный код:

```
Private Sub Worksheet_SelectionChange(ByVal Target  
As Range)  
    With ActiveWindow  
        .ScrollRow = Target.Row  
        .ScrollColumn = Target.Column  
    End With  
End Sub
```

В следующем примере содержимое активной ячейки визуально выделяется. Для визуального выделения содержимое активной ячейки форматируется полужирным и курсивным начертанием, шрифт красный. При написании процедуры обработки события **SelectionChange** необходимо понимать, что при изменении форматирования ячейки не будет автоматического возврата предыдущих форматов в этой же ячейке. Например, представьте, что на листе заполнены ячейки в диапазоне с A1 по C10.

Если выделить A1, ее содержимое будет отформатировано полужирным и курсивным начертанием и красным цветом. Если после этого выделить ячейку C10, то и она будет отформатирована так же, но ячейка A1 так и останется с этими же форматами. Становится понятным: мало форматировать выделенную ячейку, потерявшей выделение, прежние форматы. Поэтому процедура должна состоять из двух частей: в первой ячейки оформляются общими форматами, а во второй части выделенная ячейка форматируется специальными форматами.

Перепишите следующий программный код:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
With Worksheets("Лист1").Cells.Font
    .Bold = False
    .Italic = False
    .Color = RGB(0, 0, 0)
End With
With ActiveCell.Font
    .Bold = True
    .Italic = True
    .Color = RGB(255, 0, 0)
End With
End Sub
```

Можно продолжить выделение активной ячейки. Чтобы облегчить пользователю задачу по определению адреса выделенной ячейки, можно создать перекрестье, указывающее на номер столбца и номер строки. Принцип его создания принципиально ничем не отличается от примера, который мы только что рассматривали. Он также будет состоять из двух частей — в первой мы отменяем параметры форматирования строки и столбца, выделенных до этого, во второй части определяем новые параметры строки и столбца, которые находятся на соединении выделенной ячейки. Если вы сравните эти программы, заметите различия:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
With Worksheets("Лист1").Cells.Font
    .Bold = False
    .Italic = False
    .Color = RGB(0, 0, 0)
Cells.Interior.ColorIndex = xlNone
End With
With ActiveCell.Font
    .Bold = True
    .Italic = True
    .Color = RGB(255, 0, 0)
End With
With ActiveCell
```

```
.EntireRow.Interior.Color = RGB(215, 215, 215)
.EntireColumn.Interior.Color = RGB(215, 215, 215)
End With
End Sub
```

События диаграмм

События диаграммы происходят, когда потребитель активизирует или изменяет диаграмму. События отслеживаются по умолчанию на отдельных листах диаграммы. Чтобы рассматривать процедуры события для листа, необходимо выделить наименование листа с диаграммой в окне **Project** и выбрать значение **View Code** (Код Вида) из сокращенного меню. Выберите имя объекта в списке **Object** окна **Code**. После создания имени процедуры в окне **Code** выберите имя события из списка **Procedure** (Процедуры) в окне **Code**.

Так как выбор события для диаграммы вызывает некоторые затруднения, рассмотрим создание списка событий. Переайдите в Microsoft Excel. На одном из рабочих листов создайте данные, по которым можно создать диаграмму, и постройте ее, но не в этом же рабочем листе, а на отдельном. События, описанные в этой группе событий, относятся к диаграммам, созданным на отдельных листах.

Перейдите в редактор VBA и в окне **Project** найдите лист, на котором была создана диаграмма. Его имя состоит из двух частей: **Имя_Диаграммы_Номер_Листа_Включая_Все_Листы** и в круглых скобках — имя отдельного листа, на котором была создана диаграмма. Дважды щелкните по имени листа с отдельной диаграммой. Откройте список **Object** в окне **Code**. Там два значения: **General** и **Chart**. Выберите **Chart**. Будет создана процедура (по умолчанию это **Chart_Activate()**). В списке **Procedure** (Процедуры) будут доступны события диаграмм.

Если диаграмма создана на обычном рабочем листе, а не на отдельном, необходимо создать модуль класса.

К событиям диаграмм относятся (таблица 29).

Таблица 29. События диаграмм

Событие	Описание события
Activate	Активация отдельного листа диаграмм или внедренной диаграммы на рабочем листе
BeforeDoubleClick	Двойной щелчок по диаграмме или рабочему листу с диаграммой
BeforeRightClick	Щелчок правой клавишей мыши по диаграмме или рабочему листу с диаграммой
Calculate	Происходит после того, как диаграмма вычерчивает новые или измененные данные для объекта Диаграммы (Chart). Происходит после того, как рабочий лист будет пересчитан для объекта Worksheet (Рабочий лист)
Deactivate	Происходит, когда деактивируется диаграмма, рабочий лист или рабочая книга
DragOver	Происходит при перетаскивании диапазона ячеек на диаграмму
DragPlot	Происходит, когда диапазон ячеек переташен и брошен на диаграмму
MouseDown	Происходит, когда кнопка мыши нажимается при проведении указателя над диаграммой
MouseMove	Происходит в момент изменения указателя мыши, если он в это время находился над диаграммой
MouseUp	Происходит в момент отпускания нажатой кнопки мыши, если указатель в это время находился над диаграммой
Resize	Происходит, когда диаграмма изменяет свои размеры
Select	Происходит, когда выбирается элемент диаграммы
SeriesChange	Происходит, когда пользователь изменяет значение точки ряда данных диаграммы

Событие Activate

Синтаксис:

```
Private Sub object_Activate()
```

Где **object** — **Chart**, **Workbook** или **Worksheet**. Об использовании событий с объектом **Chart** смотри раздел справки VBA

Using Events with the Chart Object (Использование Событий с Объектом Диаграммы).

Событие наступает при активации листа диаграммы.

Событие очень похоже на аналогичные события для рабочего листа и книги. Перепишите программный код:

```
Private Sub Chart_Activate()
    MsgBox "Лист Диаграмм1 активирован"
    ActiveWindow.WindowState = xlMaximized
End Sub
```

Данная процедура максимизирует окно диаграммы и выводит сообщение MsgBox.

Событие BeforeDoubleClick

Синтаксис:

```
Private Sub expression_BeforeDoubleClick(ByVal
ElementID As Long, ByVal Arg1 As Long, ByVal Arg2 As
Long, Cancel As Boolean)
```

Где:

- **expression** — переменная, на которую ссылается объект типа **Chart**, объявленный в модуле класса;
- **Cancel** — необязательный аргумент, **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, двойной щелчок встроенного действия не выполняется, даже когда процедура завершена;
- **ElementID** — обязательный аргумент, значения **Arg1** и **Arg2** зависят от величины **ElementID**, как показано в следующей таблице 30.

Таблица 30. Зависимость ElementID, Arg1 и Arg2

ElementID	Arg1	Arg2
xlAxis	AxisIndex	AxisType
xlAxisTitle	AxisIndex	AxisType
xlDisplayUnitLabel	AxisIndex	AxisType
xlMajorGridlines	AxisIndex	AxisType
xlMinorGridlines	AxisIndex	AxisType
xlPivotChartDropZone	DropZoneType	None
xlPivotChartFieldButton	DropZoneType	PivotFieldIndex
xlDownBars	GroupIndex	None
xlDropLines	GroupIndex	None
xlHiLoLines	GroupIndex	None
xlRadarAxisLabels	GroupIndex	None
xlSeriesLines	GroupIndex	None
xlUpBars	GroupIndex	None
xlChartArea	None	None
xlChartTitle	None	None
xlCorners	None	None
xlDataTable	None	None
xlFloor	None	None
xlLegend	None	None
xlNothing	None	None
xlPlotArea	None	None
xlWalls	None	None
xlDataLabel	SeriesIndex	PointIndex
xlErrorBars	SeriesIndex	None
xlLegendEntry	SeriesIndex	None
xlLegendKey	SeriesIndex	None
xlSeries	SeriesIndex	PointIndex
xlTrendline	SeriesIndex	TrendLineIndex
xlXErrorBars	SeriesIndex	None
xlYErrorBars	SeriesIndex	None
xlShape	ShapeIndex	None

Следующая таблица описывает значения аргументов (таблица 31).

Таблица 31. Значения аргументов

Argument	Description
AxisIndex	Определяет независимую ось — первичную или второстепенную. Может быть одной из следующих констант XlAxisGroup: xlPrimary или xlSecondary
AxisType	Определяет тип оси. Может быть одной из следующих констант XlAxisType: xlCategory, xlSeriesAxis или xlValue
DropZoneType	Определяет тип области переноса: столбец, данные, страница или область колонки. Может быть одной из следующих констант XlPivotFieldOrientation: xlColumnField, xlDataField, xlPageField или xlRowField. Столбец и колонка области констант определяют соответственно последовательность и область категории
GroupIndex	Определяет компенсацию в пределах набора ChartGroups для специфического набора диаграмм
PivotFieldIndex	Определяет смещение в пределах набора PivotFields для специфического столбца (серии), данных, страницы или области колонки (категории)
PointIndex	Определяет компенсацию в пределах набора Points для специфической точки в пределах серии. Величина –1 указывает, что выбраны все точки данных
SeriesIndex	Определяет компенсацию в пределах набора Series для специфической серии
ShapeIndex	Определяет компенсацию в пределах набора Shapes для специфической формы
TrendlineIndex	Определяет компенсацию в пределах набора Trendlines для специфической линии тренда в пределах серии

При двойном щелчке левой клавиши по одному из элементов диаграммы открывается диалоговое окно форматирования этого элемента диаграммы. Для отключения такой возможности (чтобы заставить пользователя форматировать элементы диаграммы через контекстное меню) необходимо установить значение **Cancel** в **True**.

Для этого перепишите программный код:

```
Private Sub Chart_BeforeDoubleClick(ByVal ElementID  
As Long, ByVal Arg1 As Long, ByVal Arg2 As Long,  
Cancel As Boolean)  
MsgBox "Форматирование диаграммы отключено"  
Cancel = True  
End Sub
```

Событие BeforeRightClick

Синтаксис:

```
Private Sub expression_BeforeRightClick (Cancel As  
Boolean)
```

Где:

- **expression** — переменная, на которую ссылается объект типа **Chart**, объявленный в модуле класса;
- **Cancel** — обязательный аргумент, **False**, когда событие происходит. Если процедура события устанавливает этот аргумент в **True**, встроенное действие щелчка правой клавишей не выполняется, даже когда процедура завершена.

Подобно другим событиям рабочей книги, это событие не происходит, если щелчок правой клавишей производится, когда указатель мыши находится в форме или командной зоне (инструментальной панели или области меню), полосе состояния, ярлычках листов и диаграмм, на полосах прокрутки.

Прежде всего давайте разберемся с аргументом **expression**. В модуле класса диаграммы мы в самом начале изучения группы событий договорились выбирать объект с именем **Chart**. Этот объект и есть переменная **expression**. Если посмотрим пример создания процедуры для события **BeforeRightClick**, увидим, что это действительно так.

В предыдущем разделе при изучении события двойного щелчка левой клавишей мы отключали возможность вызывать окна для форматирования элементов диаграммы. В этом примере мы

создадим пример по отключению контекстного меню. Чтобы отменить встроенное действие для щелчка правой клавишей, нужно установить аргумент **Cancel** в **True**. Перепишите следующий программный код:

```
Private Sub Chart_BeforeRightClick(Cancel As Boolean)
    MsgBox "Контекстное форматирование отключено"
    Cancel = True
End Sub
```

Чтобы пользователь все-таки имел возможность форматирования элементов диаграммы, необходимо предварительно отключить процедуру **BeforeDoubleClick**.

Событие Calculate

Синтаксис:

```
Private Sub object_Calculate()
```

Где **object** — **Chart** или **Worksheet**. Об использовании событий с объектом **Chart** смотри **Using Events with the Chart Object** (Использование Событий с Объектом Диаграммы).

Событие **Calculate** завершает цепь событий, связанных с изменением данных на диаграмме. Перепишите программный код, иллюстрирующий работу данного события.

```
Private Sub Chart_Calculate()
    MsgBox "Изменения на диаграмме"
End Sub
```

В диаграммах сначала генерируется событие **SeriesChange**, а затем **Calculate**.

Событие Deactivate

Синтаксис:

```
Private Sub object_Deactivate()
```

Где **object** — **Chart**, **Workbook** или **Worksheet**. Об использовании событий с объектом **Chart** смотри раздел справки VBA

Using Events with the Chart Object (Использование Событий с Объектом Диаграммы).

Событие **Deactivate** генерируется при потере активности листом диаграммы. Так как аналогичные события мы уже изучали для рабочих листов и для рабочих книг, это событие не должно показаться новым. Вот пример создания процедуры по обработке этого события:

```
Private Sub Chart_Deactivate()
MsgBox "Лист Диаграммал потерял активность"
ActiveWindow.WindowState = xlMinimized
End Sub
```

Событие DragOver

Синтаксис:

```
Private Sub object_DragOver()
```

Где **object** — объект типа **Chart**, объявленный в модуле класса. Об использовании событий с объектом **Chart** смотри раздел справки VBA **Using Events with the Chart Object** (Использование Событий с Объектом Диаграммы).

Событие DragPlot

Синтаксис:

```
Private Sub object_DragPlot()
```

Где **object** — объект типа **Chart**, объявленный в модуле класса. Об использовании событий с объектом **Chart** смотри раздел справки VBA **Using Events with the Chart Object** (Использование Событий с Объектом Диаграммы).

Событие MouseDown

Синтаксис:

```
Private Sub object_MouseDown(ByVal Button As Long,
ByVal Shift As Long, ByVal X As Long, ByVal Y As
Long)
```

Где:

- **object** — объект типа **Chart**, объявленный в модуле класса. Об использовании событий с объектом **Chart** смотри раздел справки VBA **Using Events with the Chart Object** (Использование Событий с Объектом Диаграммы);
- **Button** — кнопка мыши, которая была нажата. Может принимать значение одной из следующих констант **XlMouseButton**: **xlNoButton**, **xlPrimaryButton**, **xlSecondaryButton** или **xlMiddleButton**;

X и **Y** — координаты X и Y указателя мыши в координатах объектной области диаграммы;

Shift — состояние клавиш SHIFT, CTRL и ALT, когда произошло событие. Может быть одним значением или суммой следующих величин (таблица 32).

Таблица 32. Значение аргумента Shift

Значение	Величина
0 (zero)	Без ключей
1	Нажата клавиша SHIFT
2	Нажата клавиша CTRL
4	Нажата клавиша ALT

Событие **MouseDown** наступает после щелчка левой клавиши по диаграмме, расположенной на отдельном листе.

При использовании событий **MouseDown**, **MouseMove** и **MouseUp** необходимо помнить, что эти события генерируются только на самом листе диаграммы, но не генерируются на элементах меню, инструментальных панелях, полосе состояния, линейках прокрутки. Например, если на площади диаграммы расположена плавающая инструментальная панель, при перемещении мыши по ней эти три события не генерируются.

Рассмотрим следующий пример. Создадим процедуру, которая будет вводить заголовок диаграммы и наименования ее осей.

Чтобы ввести заголовок диаграммы, прежде всего установим свойство **HasTitle** в **True**. Этим разрешим использовать в диаграмме заголовок. Свойство **ChartTitle** определяет его параметры. Свойство **Font** определяет параметры шрифта. Шрифт сам имеет свойства. Если вы посмотрите на программный код, увидите, что ничего нового здесь нет — используются те же хорошо нам знакомые **Bold**, **Bold**, **Color**, **Size** и т. д.

Свойство **ChartTitle** включает также свойство **Text**, в котором задается текст заголовка диаграммы.

На Лист1 введите данные:

Деталь	Дефицит в шт.
Деталь А	2
Деталь Б	3
Деталь В	5
Деталь Г	1
Деталь Е	6
Деталь И	7
Деталь К	2

Выделите ячейки с A2 по B8. Нажмите клавишу F11 для создания диаграммы типа Гистограмма на отдельном листе. Щелкните по диаграмме правой клавишей, из открывшегося контекстного меню выполните команду **Изменить тип диаграммы для ряда**. Выберите тип **График**, вид **График**. Нажмите **OK**.

Для объекта **Charts** используются метод **ChartWizard** и свойство **ChartArea**.

```
Private Sub Chart_MouseDown(ByVal Button As Long,  
    ByVal Shift As Long, ByVal x As Long, ByVal y As  
    Long)  
    ActiveChart.HasTitle = True  
    ActiveChart.ChartTitle.Font.Bold = True  
    ActiveChart.ChartTitle.Font.Italic = True  
    ActiveChart.ChartTitle.Font.Color = RGB(255, 0, 0)  
    ActiveChart.ChartTitle.Font.Size = 24
```

```

ActiveChart.ChartTitle.Text = "Отчет о дефиците
деталей в цехе 1 в феврале 2023 года"
Charts("Диаграмма1").ChartWizard =
    Gallery:=xlLine, _
    HasLegend:=True, CategoryTitle:="Название
деталей", ValueTitle:="В штуках"
Charts("Диаграмма1").ChartArea.Border.ColorIndex = 10
End Sub

```

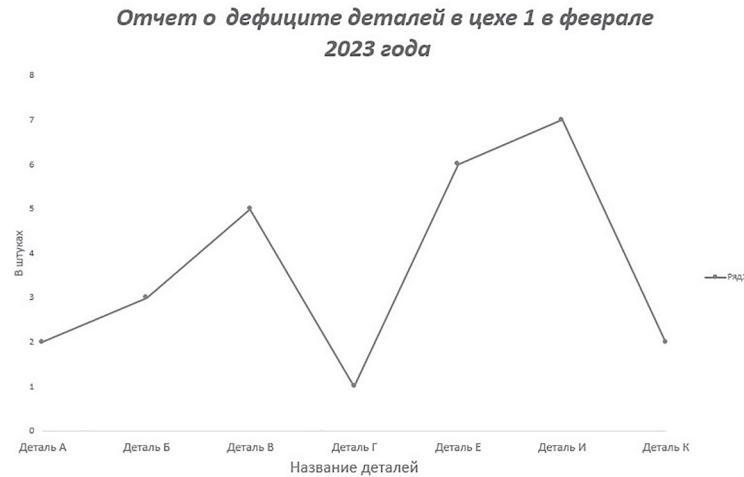


Рисунок 32. Преобразование диаграммы с помощью события MouseDown

Опробуйте этот небольшой пример, характеризующий событие **MouseDown** (рис. 32). После щелчка левой клавишей по диаграмме внешний вид диаграммы изменится: появятся заголовок, названия осей.

Событие MouseMove

Синтаксис:

```

Private Sub object_MouseMove(ByVal Button As Long,
ByVal Shift As Long, ByVal X As Long, ByVal Y As
Long)

```

Где:

- **object** — объект типа **Chart**, объявленный в модуле класса. Об использовании событий с объектом **Chart** смотри раздел справки VBA **Using Events with the Chart Object** (Использование Событий с Объектом Диаграммы);
- **Button** — кнопка мыши, которая была нажата. Может принимать значение одной из следующих констант **XlMouseButton**: **xlNoButton**, **xlPrimaryButton**, **xlSecondaryButton** или **xlMiddleButton**;
- **X** и **Y** — координаты X и Y указателя мыши в координатах объектной области диаграммы;
- **Shift** — состояние клавиш SHIFT, CTRL и ALT, когда произошло событие. Может быть одним значением или суммой следующих величин (таблица 33).

Таблица 33. Значение аргумента Shift

Значение	Величина
0 (zero)	Без ключей
1	Нажата клавиша SHIFT
2	Нажата клавиша CTRL
4	Нажата клавиша ALT

Описание события **MouseMove** отличается от аналогичного события на форме **UserForm**, которое мы будем изучать в следующих разделах.

Прежде всего рассмотрим несколько примеров, в которых я покажу некоторые неточные и неудобные процедуры по обработке этого события. Договоримся сразу: единственное, что нам нужно от этого события, — получить координаты чего-либо (в пикселях), например указателя мыши. Текст этого сообщения должен быть такой:

```
"X = " & x & " Y = " & y
```

Где X и Y — заголовки координат, а x и у — сами координаты указателя мыши (см. синтаксис описания события). Большини

и маленькими буквами они обозначены только для того, чтобы было яснее, что является чем. Программе все равно — большими или маленькими буквами указаны аргументы.

В первом примере пойдем испытанным способом — используем окно **MsgBox**, куда и будем выводить текст точки, на которой находится указатель мыши. Программный код этой процедуры такой:

```
Private Sub Chart_MouseMove(ByVal Button As Long,  
ByVal Shift As Long, ByVal x As Long, ByVal y As  
Long)  
MsgBox "X = " & x & " Y = " & y  
End Sub
```

После компилирования убеждаемся, что работать с такой процедурой сложно, если не сказать невозможно: при перемещении хотя бы на одну точку будем получать сообщение, на которое будем вынуждены отвечать.

Во втором примере воспользуемся аргументами **Button** и **Shift**. В качестве значения **Button** возьмем вторую кнопку на мыши (**xlSecondaryButton**), то есть правую. Но правая клавиша обычно привязана к контекстному меню, мы просто обязаны связать ее с одной из служебных клавиш (**Shift**, **Ctrl**, и **Alt**). В качестве аргумента **Shift** возьмем клавишу **Ctrl**, которая имеет значение 2 (таблица 33). Связывание этих двух условий будем осуществлять с помощью логического оператора **And**.

Программный код этой процедуры следующий:

```
Private Sub Chart_MouseMove(ByVal Button As Long,  
ByVal Shift As Long, ByVal x As Long, ByVal y As  
Long)  
If Shift = 2 And Button = xlSecondaryButton Then  
MsgBox "X = " & x & " Y = " & y  
End If  
End Sub
```

После компилирования становится понятным, что работать намного легче, так как окно **MsgBox** появляется только по запросу, то есть при одновременном нажатии правой клавиши

и Ctrl и перемещении мыши. Наше приложение выглядит хотя и неплохо, но имеет один недостаток: координата статична, а не динамична. Чтобы сделать получаемые данные о перемещении указателя мыши более динамичными, возьмем в качестве панели для вывода такой информации ярлычок с именем диаграммы. Имя диаграммы имеет свойство **Name**.

Если откроете окно свойств листа с диаграммой, увидите два внешне похожих свойства: **Name** и **(Name)**. Второе (в скобках) — имя этой диаграммы, а первое (без скобок) — имя диаграммы на ярлычке листа. В этом примере используем левую клавишу без нажатия. Создайте диаграмму 2 с любыми данными.

Программный код этой процедуры можно описать так:

```
Private Sub Chart_MouseMove(ByVal Button As Long,  
ByVal Shift As Long, ByVal x As Long, ByVal y As  
Long)  
If Shift = 2 Then  
Диаграмма2.Name = "X = " & x & " Y = " & y  
End If  
End Sub
```

Процедура позволяет отслеживать перемещение мыши при нажатой клавише Ctrl, но обладает одним небольшим недостатком: имя на ярлычке меняется на адрес координаты, на которой пользователь отпустил клавишу Ctrl.

Событие MouseUp

Синтаксис:

```
Private Sub object_MouseUp(ByVal Button As Long,  
ByVal Shift As Long, ByVal X As Long, ByVal Y As  
Long)
```

Где:

- **object** — объект типа **Chart**, объявленный в модуле класса.
Об использовании событий с объектом **Chart** смотри раздел справки VBA **Using Events with the Chart Object** (Использование Событий с Объектом Диаграммы);

- **Button** — кнопка мыши, которая была нажата. Может принимать значение одной из следующих констант **XlMouseButton**: **xlNoButton**, **xlPrimaryButton**, **xlSecondaryButton** или **xlMiddleButton**;
- **X** и **Y** — координаты X и Y указателя мыши в координатах объектной области диаграммы;
- **Shift** — состояние клавиш SHIFT, CTRL и ALT, когда произошло событие. Может быть одним значением или суммой следующих величин (таблица 34).

Таблица 34. Значение аргумента Shift

Значение	Величина
0 (zero)	Без ключей
1	Нажата клавиша SHIFT
2	Нажата клавиша CTRL
4	Нажата клавиша ALT

Продолжим создание процедуры, начатой в предыдущем разделе. Там оставался недостаток — имя диаграммы менялось. Так как имя диаграммы может использоваться в других процедурах, нужно вернуть его на место.

Имя листа с диаграммой мы должны где-то запомнить и хранить. Хранить можем или в переменной, или (в данном примере) лучше всего в константе. Объявим модульную константу:

```
Const Имя As String = "Диаграмма1"
```

Закончив определять координаты положения указателя мыши, возвращаем имя диаграммы на место. Для этого свойству **Name** объекта **Диаграмма1** нужно присвоить значение, хранящееся в константе (в нашем примере константа имеет значение **Диаграмма1**). Программный код будет следующий:

```
Private Sub Chart_MouseUp(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)
Диаграмма1.Name = Имя
End Sub
```

Рассмотрим другую процедуру обработки события **MouseUp**. Как вы, наверное, уже поняли, для этого можем использовать любые темы и направления обработки диаграмм. Создадим процедуру, изменяющую оформление некоторых элементов диаграммы: рамки, фон, цвет и т. д. Перепишите следующий программный код:

```
Private Sub Chart_MouseUp(ByVal Button As Long,  
ByVal Shift As Long, ByVal x As Long, ByVal y As  
Long)  
Charts("Диаграмма1").Activate  
With ActiveChart  
    .ChartArea.Border.LineStyle = xlDash  
    .PlotArea.Border.LineStyle = xlDot  
    .ChartTitle.Border.LineStyle = xlDash  
    .ChartTitle.Interior.Color = RGB(50, 255, 255)  
End With  
End Sub
```

Для обработки события (не только **MouseUp**) можно использовать много направлений программирования. Границы здесь ограничены только фантазией и возможностями программиста.

Событие **MouseUp** наступает после отпускания левой клавиши. Чтобы сгенерировать событие **MouseUp**, нужно сначала щелкнуть по диаграмме, чтобы вызвать событие **MouseDown**, затем — после отпускания левой клавиши — будет сгенерировано событие **MouseUp**.

Событие **Resize**

Синтаксис:

```
Private Sub object_Resize()
```

Где **object** — **Chart** или объект типа **Chart**, объявленный в модуле класса. Более подробно смотри **Using Events with Embedded Charts** (Использование Событий с Вложенными Диаграммами).

Событие Select

Синтаксис:

```
Private Sub object_Select(ByVal ElementID As Long,  
 ByVal Arg1 As Long, ByVal Arg2 As Long)
```

Где:

- **object** — **Chart** или объект типа **Chart**, объявленный в модуле класса. Более подробно смотри **Using Events with Embedded Charts** (Использование Событий с Вложенными Диаграммами);
- **ElementID, Arg1 и Arg2** — выбранный элемент диаграммы. Более подробно об этих аргументах смотри событие **BeforeDoubleClick**.

Событие генерируется при выборе любого элемента диаграммы. Если событие не очень сложное, пример для него составим простой. Перепишите следующий программный код:

```
Private Sub Chart_Select(ByVal ElementID As Long,  
 ByVal Arg1 As Long, ByVal Arg2 As Long)  
 MsgBox "Выбран элемент диаграммы"  
 End Sub
```

Событие SeriesChange

Синтаксис:

```
Private Sub object_SeriesChange(ByVal SeriesIndex As Long,  
 ByVal PointIndex As Long)
```

Где:

- **object** — объект типа **Chart**, объявленный в модуле класса. Об использовании событий с **Chart** смотри раздел справки **VBA Using Events with the Chart Object** (Использование Событий с Объектом Диаграммы);
- **SeriesIndex** — смещение в пределах набора **Series** для измененной последовательности;

- **PointIndex** — компенсация в пределах набора **Points** для измененной точки.

Событие **SeriesChange** генерируется, если пользователь пытается изменить точки ряда данных. Например, если потянуть за маркер на графике, значение ряда будет меняться в виде пунктирной линии, соединяющей две соседние точки. Отпустите мышь — будет сгенерировано событие **SeriesChange**. Перепишите следующий программный код:

```
Private Sub Chart_SeriesChange(ByVal SeriesIndex As Long, ByVal PointIndex As Long)
    MsgBox "Значение исходного ряда изменилось"
End Sub
```

Отметим достаточную интеллектуальность этого события. Если пользователь пытается изменить ряд данных на диаграмме, а затем, не отпуская мыши, возвращает этот ряд в исходную точку (всплывающая подсказка с числом позволяет сделать это легко), событие **SeriesChange** не генерируется. При изучении события рабочего листа **Change**, как вы помните, мы специально останавливались на нелогичности поведения этого противоречивого события. Здесь как раз наоборот. Событие **SeriesChange** логично и понятно.

В диаграммах сначала генерируется событие **SeriesChange**, а затем **Calculate**.

События экранных форм

Одновременно с созданием формы создаются события, которые могут происходить на форме.

События экранной формы доступны при выделении формы или любого элемента управления. Чтобы увидеть события формы, выделите форму, перейдите в окно кода. Если окно кода не видно, то выполните двойной щелчок по форме.

К событиям экранных форм относятся (таблица 35).

Таблица 35. События экранных форм

Событие	Описание события
Activate	Происходит при активации экранной формы
AddControl	Происходит при добавлении элемента управления при выполнении экранной формы
BeforeDragOver	Происходит при методе «перетащить и бросить», если указатель мыши проходит над формой и кнопка мыши не отпущена
BeforeDropOrPaste	Происходит при вставке данных или при методе «перетащить и бросить», если указатель мыши проходит над формой и кнопка мыши не отпущена
Click	Происходит при щелчке левой клавиши, если указатель мыши проходит над формой
DblClick	Происходит при двойном щелчке левой клавиши, если указатель мыши проходит над формой
Deactivate	Происходит при деактивации экранной формы
Error	Происходит при обнаружении ошибки при выполнении экранной формы
Initialize	Происходит при отображении экранной формы
KeyDown	Происходит при нажатии любой клавиши на клавиатуре
KeyPress	Происходит при нажатии клавиши с кодами ANSI, то есть с печатаемыми символами
KeyUp	Происходит при отпускании ранее нажатой клавиши
Layout	Происходит при изменении размера или местоположения экранной формы или элемента управления
MouseDown	Происходит при нажатии кнопки мыши, если указатель мыши проходит над формой
MouseMove	Происходит при изменении местоположения указателя мыши, если указатель мыши проходит над формой
MouseUp	Происходит при отпускании кнопки мыши, если указатель мыши проходит над формой
QueryClose	Происходит при закрытии экранной формы
RemoveControl	Происходит при удалении элемента управления во время выполнения экранной формы
Resize	Происходит при изменении размеров экранной формы
Scroll	Происходит при прокрутке экранной формы

Таблица 35. События экранных форм. Окончание

Событие	Описание события
Terminate	Происходит при завершении выполнения экранной формы
Zoom	Происходит при изменении масштаба отображения экранной формы

Событие **Activate**

Синтаксис:

```
Private Sub object_Activate()
```

Где **object** — объектная метка, представляющая объектное выражение.

Событие **Activate** происходит при активизации формы.

Объект может стать активным, используя метод **Show** в коде. Событие **Activate** может произойти только в том случае, если объект видим. **UserForm** при загрузке **Load** не видим, если вы не используете метод **Show**.

Если в редакторе VBA у нас активен объект форма **UserForm1** и мы запускаем компиляцию, форма становится видна, и происходит событие **Activate**. Но так как запускать приложение мы будем из-под Excel, скорее всего, будем открывать форму, нажимая на кнопку, связанную с макросом, или щелкая по команде меню, также связанную с макросами.

Создайте новое приложение. Включите в него **UserForm1** и **Module1**. В **Module1** напишите процедуру, в которой **UserForm1** становится видимой:

```
Public Sub WWW()
UserForm1.Show
End Sub
```

Теперь, когда форма стала видимой, можно записывать процедуру ее активизации. В ней можно задавать различные первоначальные данные. Например, загружать какие-то значения в список

ListBox или **ComboBox**, делать невидимыми или недоступными некоторые элементы управления, считывать данные и т. д.

В нашем примере будем загружать в раскрывающийся список **ComboBox1** начальный список значений:

```
Private Sub UserForm_Activate()
'UserForm_Resize
ComboBox1.AddItem "Первая строка"
ComboBox1.AddItem "Вторая строка"
ComboBox1.AddItem "Третья строка"
ComboBox1.AddItem "Четвертая строка"
End Sub
```

Строка, оформленная здесь в виде комментария, запускает процедуру **UserForm_Resize**, в которой обрабатываются размеры формы.

В данном примере форма загружается только после вызова ее методом **Show**, после этого происходит событие **Activate**.

Событие AddControl

Происходит, когда управление включено в форму, рамку или страницу **Page** в **MultiPage**.

Синтаксис для рамки **Frame**:

```
Private Sub object_AddControl( )
```

Синтаксис для **MultiPage**:

```
Private Sub object_AddControl( index As Long, ctrl
As Control)
```

Где:

- **object** — обязательный аргумент, любой объект;
- **index** — обязательный аргумент, индекс страницы (**Page**), которая будет содержать новый элемент управления;

- **ctrl** — обязательный аргумент, элемент управления, который нужно добавить.

Для примера воспользуемся методом **Add**, который добавляет элемент управления на форму, а затем передает управление в событие **AddControl**. Смысл примера в следующем: при нажатии на **CommandButton1** на **UserForm1** будет добавляться элемент управления **Label**, который получит номер 1 и станет **Label1**. Добавить элемент управления на форму во время работы программы означает не только создать видимый объект. Добавление объекта связано с конкретными координатами этого нового объекта и прочими параметрами, которые можно добавить только программным путем. В данном примере мы создадим надпись **Label1**. Это означает, что нужно позаботиться и о первоначальной надписи в этом объекте.

Сначала объявляем модульную переменную, это обязательный аргумент:

```
Dim MyLabel As Control ' Объявление модульной переменной
```

Затем пишем обработчик щелчка кнопки **CommandButton1**:

```
Private Sub CommandButton1_Click()
    Set MyLabel = Controls.Add("Forms.Label.1")
    MyLabel.Left = 20
    MyLabel.Top = 50
    MyLabel.Width = 200
    MyLabel.Height = 20
    MyLabel.Caption = "Мы получили: " & MyLabel.Name
    UserForm1_AddControl
End Sub
```

Определяем координаты и первоначальный текст. Далее управление передается процедуре **UserForm1_AddControl**. Наконец, пишем процедуру указанного события:

```
Private Sub UserForm1_AddControl()
    CommandButton1.Enabled = False
    CommandButton2.Caption = "Выключить запись"
End Sub
```

CommandButton1 стала не нужна, ее нужно отключить. Если не отключить, вторичное (случайное) нажатие на нее создаст на форме дубликат надписи **Label**, но уже с номером 2. Далее следуют прочие возможные команды этой процедуры.

Для создания элементов управления во время работы программы можно воспользоваться значениями, представленными в следующей таблице 36.

Таблица 36. Аргументы метода Add

CheckBox	Forms.CheckBox.1
ComboBox	Forms.ComboBox.1
CommandButton	Forms.CommandButton.1
Frame	Forms.Frame.1
Image	Forms.Image.1
Label	Forms.Label.1
ListBox	Forms.ListBox.1
MultiPage	Forms.MultiPage.1
OptionButton	Forms.OptionButton.1
ScrollBar	Forms.ScrollBar.1
SpinButton	Forms.SpinButton.1
TabStrip	Forms.TabStrip.1
TextBox	Forms.TextBox.1
ToggleButton	Forms.ToggleButton.1

Событие BeforeDragOver

Синтаксис:

Для Frame:

```
Private Sub object_BeforeDragOver( ByVal Cancel As  
MSForms.ReturnBoolean, ctrl As Control, ByVal Data  
As DataObject, ByVal X As Single, ByVal Y As Single,  
ByVal DragState As fmDragState, ByVal Effect As  
MSForms.ReturnEffect, ByVal Shift As fmShiftState)
```

Для MultiPage:

```
Private Sub object_BeforeDragOver( index As Long, ByVal  
Cancel As MSForms.ReturnBoolean, ctrl As Control,  
ByVal Data As DataObject, ByVal X As Single, ByVal Y As  
Single, ByVal DragState As fmDragState, ByVal Effect As  
MSForms.ReturnEffect, ByVal Shift As fmShiftState)
```

Для TabStrip:

```
Private Sub object_BeforeDragOver( index As Long,  
ByVal Cancel As MSForms.ReturnBoolean, ByVal Data As  
DataObject, ByVal X As Single, ByVal Y As Single,  
ByVal DragState As fmDragState, ByVal Effect As  
MSForms.ReturnEffect, ByVal Shift As fmShiftState)
```

Для остальных элементов управления:

```
Private Sub object_BeforeDragOver( ByVal Cancel As  
MSForms.ReturnBoolean, ByVal Data As DataObject,  
ByVal X As Single, ByVal Y As Single, ByVal  
DragState As fmDragState, ByVal Effect As MSForms.  
ReturnEffect, ByVal Shift As fmShiftState)
```

Где:

- **object** — обязательный аргумент, любой объект;
- **index** — обязательный аргумент, индекс страницы (Page) в **MultiPage**, влияющий на операцию drag-and-drop (хватай и тащи);
- **Cancel** — обязательный аргумент, статус события. **False** указывает, что управление проигнорирует событие. **True** предписывает приложению выполнить инструкцию события;
- **ctrl** — обязательный аргумент, элемент управления, над которым происходит протаскивание;
- **Data** — обязательный аргумент, данные, которые перетаскиваются в операции drag-and-drop. Данные упакованы в **DataObject**;
- **X, Y** — обязательный аргумент, горизонтальная и вертикальная координаты управляющей позиции. Обе координаты измеряются в пикселях. X измеряется с левого края элемента управления; Y измеряется с верхнего края элемента управления;
- **DragState** — обязательный аргумент, состояние перехода перемещаемых данных;

- **Effect** — обязательный аргумент, действия, поддерживаемые источником переноса;
- **Shift** — обязательный аргумент, определяет состояние SHIFT, CTRL и ALT.

Значения для аргумента **DragState** (таблица 37).

Таблица 37. Значения для DragState

Константа	Значение	Описание
fmDragStateEnter	0	Указатель мыши — в пределах диапазона цели
fmDragStateLeave	1	Указатель мыши — за пределами диапазона цели
fmDragStateOver	2	Указатель мыши — в новой позиции, но остаток находится в пределах диапазона той же самой цели

Значения для аргумента **Effect** (таблица 38).

Таблица 38. Значения для Effect

Константа	Значение	Описание
fmDropEffectNone	0	Не копирует или перемещает источник переноса на объект перемещения
fmDropEffectCopy	1	Копирует источник переноса в цель переноса
fmDropEffectMove	2	Перемещает источник перемещения на цель перемещения
fmDropEffectCopyOrMove	3	Копирует или перемещает источник переноса на объект перемещения

Значения для аргумента **Shift** (таблица 39).

Таблица 39. Значения аргумента Shift

Константа	Значение	Описание
fmShiftMask	1	Нажата клавиша SHIFT
fmCtrlMask	2	Нажата клавиша CTRL
fmAltMask	4	Нажата клавиша ALT

Используйте это событие для проверки указателя мыши при вводе, отпускании или паузе непосредственно над выбранным объектом. Когда происходит перемещение drag-and-drop, система использует это событие, если пользователь перемещает мышь или нажимает или отпускает кнопку мыши. Позиция указателя мыши определяет целевой объект, который получает это событие. Вы можете определить состояние указателя мыши, изучая аргумент **DragState**.

Когда управление генерирует это событие, можно использовать аргумент **Effect**, чтобы идентифицировать вид объекта. Когда **Effect** установлен на **fmDropEffectCopyOrMove**, источник переноса поддерживает копию (**fmDropEffectCopy**), перемещение (**fmDropEffectMove**) или отмену (**fmDropEffectNone**) действия.

Когда **Effect** установлен в **fmDropEffectCopy**, источник переноса поддерживает копию или отмену (**fmDropEffectNone**) действия.

Когда **Effect** установлен в **fmDropEffectMove**, источник переноса поддерживает перемещение или отмену (**fmDropEffectNone**) действия.

Когда **Effect** установлен в **fmDropEffectNone**, источник переноса поддерживает действие отмены.

Большинство элементов управления не поддерживает drag-and-drop, пока **Cancel** установлен в **False**, которое выбрано по умолчанию. Это является источником отказа, чтобы перетаскивать или бросать что-либо в элементах управления, и управление не вводит событие **BeforeDropOrPaste**. Элементы управления

TextBox и **ComboBox** являются исключениями из этого правила; эти элементы управления поддерживают drag-and-drop переноса, даже когда **Cancel** установлен в **False**.

Событие **BeforeDropOrPaste**

Синтаксис:

Для **Frame**:

```
Private Sub object_BeforeDropOrPaste( ByVal Cancel  
As MSForms.ReturnBoolean, ctrl As Control, ByVal  
Action As fmAction, ByVal Data As DataObject, ByVal  
X As Single, ByVal Y As Single, ByVal Effect As  
MSForms.ReturnEffect, ByVal Shift As fmShiftState)
```

Для **MultiPage**:

```
Private Sub object_BeforeDropOrPaste( index As Long,  
ByVal Cancel As MSForms.ReturnBoolean, ctrl As Control,  
ByVal Action As fmAction, ByVal Data As DataObject,  
ByVal X As Single, ByVal Y As Single, ByVal Effect As  
MSForms.ReturnEffect, ByVal Shift As fmShiftState)
```

Для **TabStrip**:

```
Private Sub object_BeforeDropOrPaste( index As Long,  
ByVal Cancel As MSForms.ReturnBoolean, ByVal Action  
As fmAction, ByVal Data As DataObject, ByVal X As  
Single, ByVal Y As Single, ByVal Effect As MSForms.  
ReturnEffect, ByVal Shift As fmShiftState)
```

Для остальных элементов управления:

```
Private Sub object_BeforeDropOrPaste( ByVal Cancel  
As MSForms.ReturnBoolean, ByVal Action As fmAction,  
ByVal Data As DataObject, ByVal X As Single, ByVal Y  
As Single, ByVal Effect As MSForms.ReturnEffect, ByVal  
Shift As fmShiftState)
```

Где:

- **object** — обязательный аргумент, любой объект;
- **index** — обязательный аргумент, индекс страницы (Page) в **MultiPage**, влияющий на операцию drag-and-drop;

- **Cancel** — обязательный аргумент, статус события. `False` указывает, что управление проигнорирует событие. `True` предписывает приложению выполнить инструкцию события;
- **ctrl** — обязательный аргумент, элемент управления, над которым происходит протаскивание;
- **Action** — показывает результат, основанный на текущих клавишных параметрах, незаконченной операции drag-and-drop;
- **Data** — обязательный аргумент, данные, которые перетаскиваются в операции drag-and-drop. Данные упакованы в `DataObject`;
- **X, Y** — обязательный аргумент, горизонтальная и вертикальная координаты управляющей позиции. Обе координаты измеряются в пикселях. X измеряется с левого края элемента управления; Y измеряется с верхнего края элемента управления;
- **Effect** — обязательный аргумент, вид drag-and-drop переноса на объекте управления;
- **Shift** — обязательный аргумент, определяет состояние SHIFT, CTRL и ALT.

Значения для аргумента **Action** (таблица 40).

Таблица 40. Значения для Action

Константа	Значение	Описание
<code>fmActionPaste</code>	2	Вставляемый выбранный объект переносится в объект
<code>fmActionDragDrop</code>	3	Указывает, что пользователь перетащил объект от своего источника до объекта перемещения и бросил это на объект

Значения для аргумента **Effect** (таблица 41).

Таблица 41. Значения для Effect

Константа	Значение	Описание
fmDropEffectNone	0	Не копирует или перемещает источник переноса на объект перемещения
fmDropEffectCopy	1	Копирует источник переноса в цель переноса
fmDropEffectMove	2	Перемещает источник перемещения на цель перемещения
fmDropEffectCopyOrMove	3	Копирует или перемещает источник перемещения на объект перемещения

Значения для аргумента **Shift** (таблица 42).

Таблица 42. Значения аргумента Shift

Константа	Значение	Описание
fmShiftMask	1	Нажата клавиша SHIFT
fmCtrlMask	2	Нажата клавиша CTRL
fmAltMask	4	Нажата клавиша ALT

Для **MultiPage** или **TabStrip**, Visual Basic для Applications (приложений) вводит это событие, когда объект данных передается для управления. Для других элементов управления система вводит это событие немедленно до операции перемещения или вставки.

Если управление генерирует это событие, можно скорректировать аргумент **Action**, чтобы идентифицировать drag-and-drop переноса. Если **Effect** установлен в **fmDropEffectCopyOrMove**, можно назначить **Action** в **fmDropEffectNone**, **fmDropEffectCopy** или **fmDropEffectMove**. Если **Effect** установлен в **fmDropEffectCopy** или **fmDropEffectMove**, можно переназначить **Action** на **fmDropEffectNone**. Вы не можете переназначить **Action**, если **Effect** установлен в **fmDropEffectNone**.

Событие Click

Синтаксис:

Для MultiPage, TabStrip:

```
Private Sub object_Click( index As Long)
```

Для остальных элементов управления:

```
Private Sub object_Click()
```

Где:

- **object** — обязательный аргумент, любой объект;
- **index** — обязательный аргумент, позиция или номер **Page** или **Tab** в пределах набора Страниц или Таб.

Самое распространенное событие на форме. Его мы рассматривали уже десятки раз. Происходит при одиночном щелчке по какому-либо объекту — элементу управления, форме. (Отсюда произошло новое русское слово — по-Click-ать, то есть постучать по клавишам, поработать на компьютере). Рассмотрим наиболее распространенные случаи применения события **Click**:

Щелчок по элементу управления, например по **CommandButton1**:

```
Private Sub CommandButton1_Click()  
End Sub
```

Щелчок по форме **UserForm1**:

```
Private Sub UserForm_Click()  
End Sub
```

В этих программных заготовках я не вписываю конкретные программные строки только потому, что мы уже вписывали их много раз для данного события.

Событие DblClick

Синтаксис:

Для MultiPage, TabStrip:

```
Private Sub object_DblClick( index As Long, ByVal  
Cancel As MSForms.ReturnBoolean)
```

Для остальных элементов управления:

```
Private Sub object_DblClick( ByVal Cancel As  
MSForms.ReturnBoolean)
```

Где:

- **object** — обязательный аргумент, любой объект;
- **index** — обязательный аргумент, позиция или номер **Page** или **Tab** в пределах набора Страниц или Таб;
- **Cancel** — обязательный аргумент, статус события. Может быть **True** или **False**. По умолчанию установлен **False**.

Двойной щелчок левой клавишей по объекту — элементу управления или форме. Для объекта можно одновременно создавать события одного щелчка левой клавишей и двойного щелчка левой клавишей. В других программах, в частности, в системе визуального программирования Borland C++ Builder, такое невозможно, так как там одиночный щелчок левой клавишей всегда перехватывает событие и поэтому невозможно одновременно задать и одиночный, и двойной щелчки.

Создание процедур с событием двойного щелчка левой клавишей абсолютно идентично созданию процедур с одиночным щелчком. Но так как я должен описывать все события подряд, создадим один пример с использованием события **DblClick**.

Событие **DblClick** вызывается, если выполняются два щелчка за промежуток времени (определяется значением параметра **Скорость выполнения двойного щелчка**), установленный в операционной системе компьютера. Скорость выполнения двойного щелчка можно изменять в любую сторону. Если в установленном промежутке времени не регистрируются два

щелчка, генерируется не событие **DblClick**, а два последовательных события **Click**.

Так как наш пример должен быть в сравнении с одиночным щелчком события **Click**, сначала напишем обработчик для одиночного щелчка:

```
Private Sub UserForm_Click()
    TextBox1.Text = "Одиночный щелчок!"
End Sub
```

Обработчик двойного щелчка будет такой:

```
Private Sub UserForm_DblClick(ByVal Cancel As
    MSForms.ReturnBoolean)
    TextBox1.Text = "Двойной щелчок!"
End Sub
```

Событие Deactivate

Синтаксис:

```
Private Sub object_Deactivate()
```

Событие **Deactivate** происходит, когда объект больше не является активным окном.

Рассмотрим следующий пример. Перенесите на форму одну командную кнопку **CommandButton**. Единственное ее назначение — передать дальнейшее управление в процедуру **UserForm_Deactivate**. Создайте вторую форму без всяких элементов управления, так как единственное назначение этой формы — показать, как деактивируется форма **UserForm1** и выполняются команды деактивации.

Перепишите следующий программный код:

```
Private Sub CommandButton1_Click()
    Call UserForm_Deactivate
End Sub
```

```

Private Sub UserForm_Activate()
UserForm1.Caption = "Пример с событием Deactivate"
CommandButton1.Caption = "Нажми меня!"
End Sub

Private Sub UserForm_Deactivate()
Dim Счетчик As Integer
For Счетчик = 1 To 3
    Beep
    MsgBox "Форма закрывается"
Next
Unload Me
UserForm2.Show
End Sub

```

При перехвате управления в процедуру **UserForm_Deactivate** трижды открывается диалоговое окно **MsgBox** с сообщением. После третьего сообщения **UserForm1** закрывается и открывается **UserForm2**. Конечно, в реальной жизни запускать диалоговое окно **MsgBox** три раза не нужно. Единственное назначение тройного появления — показать, что программа действительно выполняется.

Событие Error

Синтаксис:

Для MultiPage:

```

Private Sub object_Error( index As Long, ByVal
Number As Integer, ByVal Description As MSForms.
ReturnString, ByVal SCode As SCode, ByVal Source
As String, ByVal HelpFile As String, ByVal
HelpContext As Long, ByVal CancelDisplay As MSForms.
ReturnBoolean)

```

Для остальных элементов управления:

```

Private Sub object_Error( ByVal Number As Integer,
ByVal Description As MSForms.ReturnString, ByVal
SCode As SCode, ByVal Source As String, ByVal
HelpFile As String, ByVal HelpContext As Long, ByVal
CancelDisplay As MSForms.ReturnBoolean)

```

Где:

- **object** — обязательный аргумент, любое объектное имя;
- **index** — обязательный аргумент, индекс страницы в **MultiPage**, связанной с этим событием;
- **Number** — обязательный аргумент, определяет уникальную величину для используемого элемента управления, для идентификации ошибки;
- **Description** — обязательный аргумент, текстовое описание ошибки;
- **SCode** — обязательный аргумент, определяет код статуса OLE для ошибки. Младшие 16 битов определяют величину, которая идентична аргументу **Number**;
- **Source** — обязательный аргумент, строка, которая идентифицирует элемент управления, создающий событие;
- **HelpFile** — обязательный аргумент, определяет путь до файла Help, который описывает данную ошибку;
- **HelpContext** — обязательный аргумент, определяет контекст ID файловой темы Help, которая содержит описание ошибки;
- **CancelDisplay** — обязательный аргумент, определяет текст ошибки в окне сообщения.

Происходит, когда управление обнаруживает ошибку и не может возвратить информацию об ошибке.

Событие Initialize

Синтаксис:

```
Private Sub object_Initialize()
```

Происходит после того, как объект будет загружен, но прежде, чем он будет показан.

- **object** — метка-заполнитель, представляет объектное выражение.

Событие **Initialize** обычно используется, чтобы подготавливать приложение или **UserForm** для использования. Переменные назначают начальные величины, и элементы управления могут перемещаться или изменять размеры, чтобы размещать данные инициализации.

Для примера перенесите на форму **CommandButton1** надпись **Label1** и текстовое поле **TextBox1**. Щелкните по **UserForm1** два раза левой клавишей и, когда будет сформирован первоначальный код, выберите в списке **Procedure** событие **Initialize**. Впишите программный код:

```
Private Sub UserForm_Initialize()
    CommandButton1.Caption = "Рассчитать остатки"
    CommandButton1.AutoSize = True
    CommandButton1.TakeFocusOnClick = True
    TextBox1.Text = "0.00"
    Label1.AutoSize = True
    Label1.WordWrap = False
    Label1.Caption = "Начиная с ячейки C5"
End Sub
```

После компилирования на форме будет создана начальная инициализация элементов управления.

Событие **KeyDown**

Синтаксис:

```
Private Sub object_KeyDown( ByVal KeyCode As
MSForms.ReturnInteger, ByVal Shift As fmShiftState)
```

Где:

- **object** — обязательный аргумент, любое объектное имя;
- **KeyCode** — обязательный аргумент, целое, представляет код ключа, который был нажат;

- **Shift** — обязательный аргумент, состояние SHIFT, CTRL и ALT.

Значения аргумента **Shift** представлены в таблице 43.

Таблица 43. Значения аргумента Shift

Константа	Значение	Описание
fmShiftMask	1	Нажата клавиша SHIFT
fmCtrlMask	2	Нажата клавиша CTRL
fmAltMask	4	Нажата клавиша ALT

Событие **KeyDown** используется для различия:

- ключей навигации — как, например HOME, END, PAGEUP, PAGEDOWN, UP ARROW, DOWN ARROW, RIGHT ARROW, LEFT ARROW или TAB;
- комбинации клавиш и стандартных клавищных модификаторов (SHIFT, CTRL или ALT).

Для различия правой числовой клавиатуры от верхней (правая клавиатура имеет больше возможностей, например, вводить символы, которых нет на клавиатуре, с помощью кодов).

Событие **KeyDown** не происходит при следующих обстоятельствах:

- пользователь нажимает Enter в форме с командной кнопкой, чье свойство **Default** установлено в **True**;
- пользователь нажимает Esc в форме с командной кнопкой, чье свойство **Cancel** установлено в **True**.

Параметр **KeyCode** содержит клавиатурный код (а не Ascii) нажатой клавиши. Поэтому он предназначен для определения клавиш, а не конкретных символов. Например, на клавише A (на английской раскладке) находятся четыре символа: английские A и a, русские Ф и ф. Параметр **KeyCode** понимает только английскую букву A, остальные не понимает. Поэтому если мы попробуем с помощью параметра **KeyCode** определить какие-то

конкретные символы из набора этих четырех, воспринята будет только английская А.

Рассмотрим следующий пример. Перенесите на форму два текстовых поля **TextBox**. В первое задаем символы английского алфавита в верхнем регистре, которые будут преобразовываться в числовые коды от 0 до 255, а во втором — получать введенный символ. Если введем А, откроется окно **MsgBox** с сообщением, что нажата буква А. При нажатии остальных букв окно **MsgBox** открываться не будет. Так как в данном примере используются символы только с 0 до 255, русские символы в этот диапазон не входят.

Перепишите следующий программный код:

```
Private Sub TextBox1_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As Integer)
    TextBox5.Text = KeyCode
    If KeyCode = 65 Then
        MsgBox "Код 65 или А"
    End If
    TextBox2.Text = Chr(KeyCode)
End Sub

Private Sub UserForm_Initialize()
    UserForm1.Caption = "Ищем английскую букву А"
End Sub
```

Сравните этот пример с использованием события **KeyUp**. Преобразование символов в коды происходит в разное время, поэтому эффект выглядит по-разному.

При работе с клавиатурой события происходят в такой последовательности: **KeyDown**, **KeyPress**, **KeyUp**.

Несмотря на то что событие **KeyDown** не различает регистра и работает только в первых 128 (0-127) кодах символов, оно активно используется при задании горячих клавиш: привязка происходит не к символам, а к клавишам на клавиатуре. Например, вырезать или скопировать выделенный фрагмент можно сочетанием клавиш Ctrl + X и Ctrl + C. При этом совершенно

не важно, какая раскладка включена на клавиатуре на самом деле — английская, русская или другая национальная.

Событие KeyPress

Синтаксис:

```
Private Sub object_KeyPress( ByVal KeyAscii As  
MSForms.ReturnInteger)
```

Где:

- **object** — обязательный аргумент, любое место;
- **KeyAscii** — обязательный аргумент, целое число, которое представляет стандартный числовой ключевой код ANSI.

Событие **KeyPress** происходит, когда нажат любой из следующих клавиш.

- Любой выводимый клавишный символ.
- CTRL, объединенный с символом из стандартного алфавита.
- CTRL, объединенный с любым специальным символом.
- BACKSPACE.
- ESC.

Событие **KeyPress** не происходит при следующих условиях.

- Нажата клавиша TAB.
- Нажата клавиша ENTER.
- Нажата клавиша одной из стрелок.
- Когда нажатие клавиши заставляет фокус перемещаться с одного элемента управления на другой.

Событие **KeyPress** возвращает код Ascii нажатой клавиши. При этом не перехватываются специальные клавиши, такие как PrintScreen или Alt, а Enter, Esc и Backspace перехватываются. Процедура передает параметр **KeyAscii**, содержащий код Ascii нажатой клавиши.

Рассмотрим следующий пример. Перенесите на форму два текстовых поля **TextBox**. В первое задаем любые символы, в том числе и буквы русского алфавита. Мы можем отличать с помощью **KeyAscii** символы разных регистров. Если введем A, откроется окно **MsgBox** с сообщением, что нажата буква A. При нажатии остальных букв окно **MsgBox** открываться не будет. Обращаю ваше внимание, что при использовании **KeyAscii** различается регистр, поэтому мы будем искать *A*, а не *a*.

Перепишите следующий программный код:

```
Private Sub UserForm_Initialize()
UserForm1.Caption = "Ищем английскую букву А"
End Sub

Private Sub TextBox1_KeyPress(ByVal KeyAscii As
MSForms.ReturnInteger)
TextBox1.Text = KeyAscii
If KeyAscii = 65 Then
MsgBox "Код 65 или А"
End If
End Sub
```

Сравните событие **KeyPress** с событиями **KeyDown** и **KeyUp** — они работают по-разному.

При работе с клавиатурой события происходят в такой последовательности: **KeyDown**, **KeyPress**, **KeyUp**.

Событие **KeyUp**

Синтаксис:

```
Private Sub object_KeyUp(ByVal KeyCode As MSForms.
ReturnInteger, ByVal Shift As fmShiftState)
```

Где:

- **object** — обязательный аргумент, любое объектное имя;
- **KeyCode** — обязательный аргумент, целое, представляет код ключа, который был отпущен;
- **Shift** — обязательный аргумент, состояние SHIFT, CTRL и ALT.

Значения аргумента **Shift** представлены в таблице 44.

Таблица 44. Значения аргумента Shift

Константа	Значение	Описание
fmShiftMask	1	Нажата клавиша SHIFT
fmCtrlMask	2	Нажата клавиша CTRL
fmAltMask	4	Нажата клавиша ALT

Событие **KeyUp** используется для различия:

- ключей навигации, например HOME, END, PAGEUP, PAGEDOWN, UP ARROW, DOWN ARROW, RIGHT ARROW, LEFT ARROW или TAB;
- комбинации клавиш и стандартных клавищных модификаторов (SHIFT, CTRL или ALT);
- числовой вспомогательной клавиатуры и основной числовой клавиатуры.

Событие **KeyUp** не происходит при следующих обстоятельствах:

- пользователь нажимает Enter в форме с командной кнопкой, чье свойство **Default** установлено в **True**;
- пользователь нажимает Esc в форме с командной кнопкой, чье свойство **Cancel** установлено в **True**.

Рассмотрим пример. Перенесите на форму два текстовых поля **TextBox**. В первое мы задаем символы английского алфавита в верхнем регистре, они будут преобразовываться в числовые коды от 0 до 255, а во втором будем получать введенный символ. Если введем A, откроется окно **MsgBox** с сообщением, что нажата буква A. При нажатии остальных букв окно **MsgBox** открываться не будет. В данном примере используются символы только с 0 до 255, русские символы в этот диапазон не входят.

Перепишите следующий программный код:

```
Private Sub TextBox1_KeyPress(ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As Integer)
    TextBox1.Text = Chr(KeyCode)
    If KeyCode = 65 Then
        MsgBox "Код 65 или A"
    End If
    TextBox2.Text = Chr(KeyCode)
End Sub

Private Sub UserForm_Initialize()
    UserForm1.Caption = "Ищем английскую букву А"
End Sub
```

Сравните этот пример с использованием события **KeyDown**. Преобразование символов в коды происходит в разное время, поэтому результат выглядит по-разному.

При работе с клавиатурой события происходят в такой последовательности: **KeyDown**, **KeyPress**, **KeyUp**.

Событие **Layout**

Синтаксис:

Для **MultiPage**:

```
Private Sub object_Layout( index As Long)
```

Для остальных элементов управления:

```
Private Sub object_Layout( )
```

Где:

- **object** — обязательный аргумент, любой объект;
- **index** — обязательный аргумент, индекс страницы в **MultiPage**, связанной с этим событием.

Встроенное действие события должно вычислить новые координаты элементов управления и перерисовать экран. Пользователь может создать событие **Layout**, изменяя размер элемента управления.

Для элементов управления, которые поддерживают свойство **AutoSize**, событие **Layout** применяется, когда **AutoSize** изменяет размер элемента управления. Это происходит, если пользователь изменяет величину свойства, на которую влияет размер элемента управления. Например, увеличивая размер шрифта (**Font**) в **TextBox** или **Label**, можно значительно изменить координаты элемента управления и применить событие **Layout**.

Пример с использованием события **Layout**. Создайте новое приложение. Перенесите на форму одну **CommandButton1** и одну двухходовую кнопку **ToggleButton1**. Установите в обоих элементах управления свойства **AutoSize = True**, а **WordWrap** в **False**. В примере мы собираемся изменять надписи на кнопках. Поэтому размеры кнопок будут меняться. Перепишите следующие процедуры:

```
Private Sub UserForm_Initialize()
    CommandButton1.Caption = "Перемещение элемента
управления"
    CommandButton1.AutoSize = True
    CommandButton1.TakeFocusOnClick = True
    ToggleButton1.Caption = " Изучаем событие
Layout"
End Sub

Private Sub CommandButton1_Click()
    If ActiveControl.Name = "ToggleButton1" Then
        Else
            ActiveControl.Move 10, 10, , ,
ToggleButton1.Value
```

```

CommandButton1.Caption = "Смена наименования
кнопки"
    ToggleButton1.Move 100, 100, , ,
ToggleButton1.Value
End If
End Sub

Private Sub UserForm_Layout()
    Dim MyControl As Control
    For Each MyControl In Controls
        If MyControl.LayoutEffect = fmLayoutEffectInitiate
Then
    Exit For
End If
Next
End Sub

Private Sub ToggleButton1_Click()
    If ToggleButton1.Value = True Then
        ToggleButton1.Caption = "Событие Layout"
        ActiveControl.Move 10, 50, , ,
ToggleButton1.Value
        CommandButton1.Caption = "Новая смена
наименования кнопки"
    Else
        ToggleButton1.Caption = "Нет события Layout"
        ActiveControl.Move 10, 50, , ,
ToggleButton1.Value
        CommandButton1.Caption = "Наименование
кнопки меняется"
        ToggleButton1.Caption = "Меняем наименование
в Layout"
    End If
End Sub

```

После инициализации диалогового окна в зависимости от того, какая кнопка нажимается, происходит перемещение кнопок и надписей на них. Размеры кнопок при этом также меняются и подстраиваются под требуемые размеры. Изображение диалогового окна перерисовывается.

Событие MouseDown

Синтаксис:

Для MultiPage, TabStrip:

```
Private Sub object_MouseDown( index As Long, ByVal
Button As fmButton, ByVal Shift As fmShiftState,
ByVal X As Single, ByVal Y As Single)
```

Для остальных элементов управления:

```
Private Sub object_MouseDown( ByVal Button As
fmButton, ByVal Shift As fmShiftState, ByVal X As
Single, ByVal Y As Single)
```

Где:

- **object** — обязательный аргумент, любой объект;
- **index** — обязательный аргумент, индекс страницы или закладки в **MultiPage** или **TabStrip** с определенным событием;
- **Button** — обязательный аргумент, целая величина, которая идентифицирует кнопку мыши, вызвавшую событие;
- **Shift** — обязательный аргумент, состояние SHIFT, CTRL и ALT;
- **X** и **Y** — обязательный аргумент, горизонтальная или вертикальная позиция, в пунктах, с левого или верхнего края формы, рамки (Frame) или страницы (Page).

Значения для **Button** (таблица 45).

Таблица 45. Значения для аргумента **Button**

Константа	Значение	Описание
fmButtonLeft	1	Нажата левая клавиша мыши
fmButtonRight	2	Нажата правая клавиша мыши
fmButtonMiddle	4	Нажата средняя клавиша мыши

Значения для **Shift** (таблица 46).

Таблица 46. Значения для аргумента Shift

Значение	Описание
1	Нажата клавиша SHIFT
2	Нажата клавиша CTRL
3	Нажаты клавиши SHIFT и CTRL
4	Нажата клавиша ALT
5	Нажаты клавиши ALT и SHIFT
6	Нажаты клавиши ALT и CTRL
7	Нажаты клавиши ALT, SHIFT и CTRL

Вы можете идентифицировать индивидуальные клавишные модификаторы, используя следующие константы (таблица 47).

Таблица 47. Значения констант клавиш

Константа	Значение	Описание
fmShiftMask	1	Нажата клавиша SHIFT
fmCtrlMask	2	Нажата клавиша CTRL
fmAltMask	4	Нажата клавиша ALT

Последовательность связанных событий мыши:

- 1** MouseDown;
- 2** MouseUp;
- 3** Click;
- 4** DblClick;
- 5** MouseUp.

Информацию о состоянии указателя мыши в большинстве приложений выводят на полосу состояния **StatusBar**. Так как у некоторых пользователей может не быть этого дополнительного элемента управления, построим наш пример с использованием двух элементов управления: **StatusBar** и **TextBox**. Как увидите из примера, никакой принципиальной разницы здесь нет.

Перенесите на форму **CommandButton1**, **TextBox1** и полосу состояния **StatusBar1**.

Для элемента управления **TextBox** никакой дополнительной настройки не нужно. Нужно только немного растянуть по горизонтали его размеры. В элементе управления **StatusBar** установите свойство **Style** в значение **sbrSimple**, чтобы занять всю площадь полосы под выводимый текст. Помните, как расположена полоса состояния в приложениях? Она находится внизу по всей длине диалогового окна. Растяните эту полосу. В этом примере мы будем учитывать перемещение мыши по кнопке **CommandButton1**, хотя на практике обычно его учитывают по форме или по изображению **Image**. Создайте первоначальный код, для чего дважды щелкните по **CommandButton1**. Создается заготовка под событие **Click**, оно нам не нужно. В списке **Procedure** найдите событие **MouseDown** и щелкните по нему. Напишите программный код:

```
Private Sub CommandButton1_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
StatusBar1.SimpleText = "Button = " & Button &
Chr$(13) & _
"Shift = " & Shift & Chr$(13) & _
"X = " & X & " Y = " & Y
End Sub
```

Как видно из этого примера, никакой принципиальной разницы в способах вывода информации о перемещении мыши нет.

Примечание: после каждого выдаваемого параметра (Button, Shift, X и Y) поставьте по три пробела, чтобы текст был более удобочитаемым.

Событие MouseMove

Синтаксис:

Для **MultiPage**, **TabStrip**:

```
Private Sub object_MouseMove( index As Long, ByVal
Button As fmButton, ByVal Shift As fmShiftState,
ByVal X As Single, ByVal Y As Single)
```

Для остальных элементов управления:

```
Private Sub object_MouseMove( ByVal Button As  
fmButton, ByVal Shift As fmShiftState, ByVal X As  
Single, ByVal Y As Single)
```

Где:

- **object** — обязательный аргумент, любой объект;
- **index** — обязательный аргумент, индекс страницы или закладки в **MultiPage** или **TabStrip** с определенным событием;
- **Button** — обязательный аргумент, целая величина, которая идентифицирует кнопку мыши, вызвавшую событие;
- **Shift** — обязательный аргумент, состояние SHIFT, CTRL и ALT;
- **X** и **Y** — обязательный аргумент, горизонтальная или вертикальная позиция, в пунктах, с левого или верхнего края формы, рамки (Frame) или страницы (Page).

Значения для **Button** (таблица 48).

Таблица 48. Значения для аргумента Button

Значение	Описание
0	Клавиши не нажаты
1	Нажата левая клавиша мыши
2	Нажата правая клавиша мыши
3	Нажаты правая и левая клавиши
4	Нажата средняя клавиша
5	Нажаты средняя и левая клавиши
6	Нажаты средняя и правая клавиши
7	Нажаты все три кнопки

Значения для **Shift** (таблица 49).

Таблица 49. Значения для аргумента Shift

Значение	Описание
1	Нажата клавиша SHIFT
2	Нажата клавиша CTRL
3	Нажаты клавиши SHIFT и CTRL
4	Нажата клавиша ALT
5	Нажаты клавиши ALT и SHIFT
6	Нажаты клавиши ALT и CTRL
7	Нажаты клавиши ALT, SHIFT и CTRL

Вы можете идентифицировать индивидуальные клавишные модификаторы клавиатуры, используя следующие константы (таблица 50).

Таблица 50. Значения констант клавиш

Константа	Значение	Описание
fmShiftMask	1	Нажата клавиша SHIFT
fmCtrlMask	2	Нажата клавиша CTRL
fmAltMask	4	Нажата клавиша ALT

Информацию о перемещении указателя мыши в большинстве приложений выводят на полосу состояния **StatusBar**. Так как у некоторых пользователей может не быть этого дополнительного элемента управления, построим наш пример с использованием двух элементов управления: **StatusBar** и **TextBox**. Как видите, никакой принципиальной разницы здесь нет.

Перенесите на форму **CommandButton1**, **TextBox1** и **StatusBar1**.

Для элемента управления **TextBox** никакой дополнительной настройки не нужно, только немного растянуть по горизонтали его размеры. В элементе управления **StatusBar** установите свойство **Style** в значение **sbrSimple**, чтобы занять всю площадь полосы под выводимый текст. Полоса состояния в приложениях находится внизу по всей длине диалогового окна. Растяните эту полосу. В этом примере будем учитывать перемещение мыши по кнопке **CommandButton1**, хотя на практике обычно его

учитывают по форме или по изображению **Image**. Создайте первоначальный код, для чего дважды щелкните по кнопке **CommandButton1**. При этом создастся заготовка под событие **Click**, оно нам не нужно. В списке **Procedure** найдите событие **MouseMove** и щелкните по нему. Напишите программный код:

```
Private Sub CommandButton1_MouseMove(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    TextBox1.Text = "X = " & X & " Y = " & Y
    StatusBar1.SimpleText = "X = " & X & " Y = " & Y
End Sub
```

Видите, никакой принципиальной разницы в способах вывода информации о перемещении мыши нет.

Примечание: после второго X поставьте три пробела, чтобы текст был более удобочитаемым.

Событие **MouseUp**

Синтаксис:

Для **MultiPage**, **TabStrip**:

```
Private Sub object_MouseUp( index As Long, ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)
```

Для остальных элементов управления:

```
Private Sub object_MouseUp( ByVal Button As fmButton, ByVal Shift As fmShiftState, ByVal X As Single, ByVal Y As Single)
```

Где:

- **object** — обязательный аргумент, любой объект;
- **index** — обязательный аргумент, индекс страницы или закладки в **MultiPage** или **TabStrip** с определенным событием;
- **Button** — обязательный аргумент, целая величина, которая идентифицирует кнопку мыши, вызвавшую событие;

- **Shift** — обязательный аргумент, состояние SHIFT, CTRL и ALT;
- **X и Y** — обязательный аргумент, горизонтальная или вертикальная позиция, в пунктах, с левого или верхнего края формы, рамки (Frame) или страницы (Page).

Значения для **Button** (таблица 51).

Таблица 51. Значения для аргумента Button

Константа	Значение	Описание
fmButtonLeft	1	Нажата левая клавиша мыши
fmButtonRight	2	Нажата правая клавиша мыши
fmButtonMiddle	4	Нажата средняя клавиша мыши

Значения для **Shift** (таблица 52).

Таблица 52. Значения для аргумента Shift

Значение	Описание
1	Нажата клавиша SHIFT
2	Нажата клавиша CTRL
3	Нажаты клавиши SHIFT и CTRL
4	Нажата клавиша ALT
5	Нажаты клавиши ALT и SHIFT
6	Нажаты клавиши ALT и CTRL
7	Нажаты клавиши ALT, SHIFT и CTRL

Вы можете идентифицировать индивидуальные клавишные модификаторы, используя следующие константы (таблица 53).

Таблица 53. Значения констант клавиш

Константа	Значение	Описание
fmShiftMask	1	Нажата клавиша SHIFT
fmCtrlMask	2	Нажата клавиша CTRL
fmAltMask	4	Нажата клавиша ALT

Последовательность связанных событий мыши:

- 1 MouseDown;
- 2 MouseUp;
- 3 Click;
- 4 DblClick;
- 5 MouseUp.

Информацию о состоянии указателя мыши в большинстве приложений выводят на полосу состояния **StatusBar**. Так как у некоторых пользователей может не быть этого дополнительного элемента управления, построим наш пример с использованием **StatusBar** и **TextBox**. Пример покажет, что никакой принципиальной разницы здесь нет.

Перенесите на форму **CommandButton1**, **TextBox1** и **StatusBar1**.

Для элемента управления **TextBox** никакой дополнительной настройки не нужно, немного растяните по горизонтали его размеры. В **StatusBar** установите свойство **Style** в значение **sbrSimple**, чтобы занять всю площадь полосы под выводимый текст. Она находится внизу по всей длине диалогового окна. Растяните эту полосу. Мы будем учитывать перемещение мыши по **CommandButton1**, хотя на практике обычно его учитывают по форме или по изображению **Image**. Создайте первоначальный код (дважды щелкните по **CommandButton1**). При этом создается заготовка под событие **Click**, оно нам не нужно. В списке **Procedure** найдите событие **MouseUp** и щелкните по нему. Напишите программный код:

```
Private Sub CommandButton1_MouseUp(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    StatusBar1.SimpleText = "Button = " & Button &
    Chr$(13) & _
    "Shift = " & Shift & Chr$(13) & _
```

```
"X = " & X & " Y = " & Y
End Sub
```

Никакой принципиальной разницы в способах вывода информации о перемещении мыши нет.

Примечание: после каждого выдаваемого параметра (Button, Shift, X и Y) поставьте по три пробела, чтобы текст был более удобочитаемым.

Событие QueryClose

Синтаксис:

```
Private Sub UserForm_QueryClose(cancel As Integer,
closemode As Integer)
```

Где:

- **cancel** — целое значение. При установке этого аргумента в любую величину, кроме 0 (False), происходит блокировка события **QueryClose**, и **UserForm** не закрывается;
- **closemode** — значение или константа, указывающие причину события **QueryClose**.

Значения **closemode** могут быть следующими (таблица 54).

Таблица 54. Значения аргумента closemode

Константа	Значение	Описание
vbFormControlMenu	0	Пользователь выбрал команду Close (Закрыть) из меню управления на UserForm
vbFormCode	1	Утверждение Unload (Закрытие) введено из кода
vbAppWindows	2	Текущее окно, обслуживающее сеанс среды, закрывается
vbAppTaskManager	3	Окно Task Manager закрывает приложение

Эти константы указаны в Visual Basic для объектной библиотеки Applications в Object Browser. Отметьте, что константа

vbFormMDIForm также определена в Object Browser, но еще не поддерживается.

Это событие обычно используется, чтобы убедиться: нет незаконченных задач в формах пользователя, включенных в приложение прежде, чем это приложение закроется. Например, если пользователь не сохранит новые данные в любом **UserForm**, приложение может подсказать, что необходимо сохранять данные.

Когда приложение закрывается, можно использовать процедуру события **QueryClose**, чтобы устанавливать свойство **Cancel** в **True**, и остановить тем самым процесс закрытия.

Рассмотрим следующий пример. Во всех приложениях можно закрыть приложения с помощью кнопки **Закрыть (x)**, она расположена на системной полосе приложения. В любом приложении есть и другие варианты выйти из приложения — с помощью команды меню, кнопки и т. д. Причем всегда проверяется сохранение введенной информации. Поэтому мы создадим такие условия, при которых при нажатии на **Закрыть (X)**, расположенной на системной полосе, форма закрыта не будет. Закрыть форму можно будет только при нажатии на командную кнопку. Перенесите на форму кнопку **CommandButton**, которая будет закрывать форму. Измените ей заголовок на **OK**.

Перепишите следующий программный код:

```
Private Sub CommandButton1_Click()
Unload Me
End Sub

Private Sub UserForm_QueryClose(Cancel As Integer,
CloseMode As Integer)
If CloseMode = vbFormControlMenu Then
MsgBox "Для закрытия нажмите на OK"
Cancel = True
End If
End Sub
```

В данном примере аргумент **CloseMode** проверяет наступление события **UserForm_QueryClose**, а **Cancel** определяет, нужно ли закрывать форму. Если установлен 0 (**False**), форму закрывать нужно, если установлен не 0 (**True**), форму закрывать не нужно.

Событие RemoveControl

Синтаксис:

Для MultiPage:

```
Private Sub object_RemoveControl( index As Long,  
ctrl As Control)
```

Для остальных элементов управления:

```
Private Sub object_RemoveControl( ctrl As Control)
```

Где:

- **object** — обязательный аргумент, любое объектное имя;
- **index** — обязательный аргумент, индекс страницы в **MultiPage**, который указывает на удаляемый элемент управления;
- **ctrl** — обязательный аргумент, удаляемый элемент управления.

Это событие происходит, когда элемент управления удаляется из формы — до закрытия экранной формы.

Событие Resize

Синтаксис:

```
Private Sub UserForm_Resize()
```

Используйте процедуру события **Resize**, чтобы перемещать или менять размеры элементов управления, когда родитель **UserForm** изменяет размеры. Вы можете также использовать эту процедуру события, чтобы пересчитывать переменные или свойства.

Рассмотрим следующий пример. Перенесите на форму несколько элементов управления. Они нужны только для украшения нашего примера, никакой другой нагрузки нести не будут.

Одиночный щелчок левой клавишей по форме будет изменять в 1,5 раза высоту формы, а двойной щелчок левой клавишей — изменять ширину формы в 1,7 раза. Вторичные щелчки один и два раза будут возвращать размеры формы, соответственно, по высоте и по ширине. Данные по высоте и по ширине будем выводить в заголовок формы.

Перепишите следующий программный код:

```
Dim AAA As Integer
Dim BBB As Integer

Private Sub UserForm_Activate()
    UserForm1.Caption = "Изучение свойства Resize!"
    AAA = Height
    BBB = Width
End Sub

Private Sub UserForm_Click()
    Dim MyHeight As Single
    MyHeight = Height
    If MyHeight = Val(AAA) Then
        Height = Val(AAA) * 1.5
    Else
        Height = Val(AAA)
    End If
End Sub

Private Sub UserForm_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
    Dim MyWidth As Single
    MyWidth = Width
    If MyWidth = Val(BBB) Then
        Width = Val(BBB) * 1.7
    Else
        Width = Val(BBB)
    End If
End Sub

Private Sub UserForm_Resize()
    UserForm1.Caption = "Новая высота: " & Height _
```

```
& " Новая ширина: " & Width  
End Sub
```

В этом примере мы создаем две модульные переменные, в которых храним начальные значения высоты и ширины. Далее пересчитываем новые значения высоты и ширины и храним их в локальных переменных. Результаты размеров формы выдаются пользователю для сведения.

Событие Scroll

Синтаксис:

Для ScrollBar:

```
Private Sub object_Scroll( )
```

Для MultiPage:

```
Private Sub object_Scroll( index As Long, ActionX  
As fmScrollAction, ActionY As fmScrollAction, ByVal  
RequestDx As Single, ByVal RequestDy As Single,  
ByVal ActualDx As MSForms.ReturnSingle, ByVal  
ActualDy As MSForms.ReturnSingle)
```

Для Frame:

```
Private Sub object_Scroll( ActionX As  
fmScrollAction, ActionY As fmScrollAction, ByVal  
RequestDx As Single, ByVal RequestDy As Single,  
ByVal ActualDx As MSForms.ReturnSingle, ByVal  
ActualDy As MSForms.ReturnSingle)
```

Где:

- **object** — обязательный аргумент, любое объектное имя;
- **index** — обязательный аргумент, индекс страницы в **MultiPage**, связываемый с этим событием;
- **ActionX** — обязательный аргумент, действие, которое происходит в горизонтальном направлении;
- **ActionY** — обязательный аргумент, действие, которое происходит в вертикальном направлении;

- **RequestDx** — обязательный аргумент, расстояние в пикселях, на которое необходимо переместить бегунок, чтобы переместиться в горизонтальном направлении;
- **RequestDy** — обязательный аргумент, расстояние в пикселях, на которое необходимо переместить бегунок, чтобы переместиться в вертикальном направлении;
- **ActualDx** — обязательный аргумент, расстояние в пикселях, пройденное бегунком в горизонтальном направлении;
- **ActualDy** — обязательный аргумент, расстояние в пикселях, пройденное бегунком в вертикальном направлении.

Значения для **ActionX** и **ActionY** могут выбираться следующие (таблица 55).

Таблица 55. Значения для ActionX и ActionY

Константа	Значение	Описание
fmScrollActionNoChange	0	Изменений нет
fmScrollActionLineUp	1	Небольшое расстояние в вертикальном перемещении бегунка; небольшое расстояние влево при горизонтальном перемещении бегунка. Перемещение эквивалентно нажатию Up или стрелки на клавиатуре при перемещении бегунка
fmScrollActionLineDown	2	Небольшое расстояние вниз при вертикальном перемещении бегунка; небольшое расстояние вправо при горизонтальном перемещении бегунка. Перемещение является эквивалентом нажатия вниз или стрелки вправо на клавиатуре, чтобы перемещать бегунок

Таблица 55. Значения для ActionX и ActionY. Окончание

Константа	Значение	Описание
fmScrollActionPageUp	3	Перемещение бегунка на страницу по вертикали; бегунок перемещается на одну страницу влево по горизонтали. Перемещение является эквивалентом нажатия PAGE UP на клавиатуре, чтобы перемещать бегунок
fmScrollActionPageDown	4	Перемещение бегунка на одну страницу вниз по вертикали; перемещение бегунка на одну страницу вправо по горизонтали. Перемещение является эквивалентом нажатия PAGE DOWN на клавиатуре для перемещения бегунка
fmScrollActionBegin	5	Перемещение бегунка вверх по вертикали; бегунок перемещается влевую сторону по горизонтали
fmScrollActionEnd	6	Перемещение бегунка вниз по вертикали; бегунок перемещается вправую сторону по горизонтали
fmScrollActionPropertyChanged	8	Значение свойств ScrollTop или ScrollLeft изменяется. Направление и величина перемещения зависят от свойства, установленного в новом значении свойства
fmScrollActionControlRequest	9	Элемент управления сообщает о перемещении контейнера. Величина перемещения зависит от элемента управления и включенного контейнера
fmScrollActionFocusRequest	10	Передать фокус в другой элемент управления. Значение перемещения зависит от размера выбранного элемента управления, обычно имеет эффект перемещения выбранного элемента управления, так что он становится полностью видим пользователю

Создадим пример с участием события. Перенесите на форму одну линейку прокрутки **ScrollBar**, три надписи **Label**. Установите в свойстве **Orientation** значение **fmOrientationHorizontal**. Установите размеры линейки в соответствии с размерами формы.

Перепишите следующий программный код:

```
Dim MyScroll As Integer

Private Sub UserForm_Initialize()
    ScrollBar1.Min = -100
    ScrollBar1.Max = 100
    ScrollBar1.Value = 0
    Label1.Caption = "100 -----Интервал----- -100"
    Label1.AutoSize = True
    Label2.Caption = ""
End Sub

Private Sub ScrollBar1_Change()
    Label2.Caption = "Изменение интервала " & (MyScroll
    - ScrollBar1.Value)
End Sub

Private Sub ScrollBar1_Exit(ByVal Cancel As MSForms.ReturnBoolean)
    Label2.Caption = " Изменение интервала " & (MyScroll
    - ScrollBar1.Value)
    MyScroll = ScrollBar1.Value
End Sub

Private Sub ScrollBar1_Scroll()
    Label3.Caption = " Изменение интервала " & (MyScroll
    - ScrollBar1.Value)
End Sub
```

В данном примере сначала происходит инициализация параметров линейки прокрутки, затем, в зависимости от того, каким образом пользователь перемещает значения бегунка (с помощью самого бегунка, кнопок на концах линейки, клавишами PageUp или PageDown), изменяются значения в надписях **Label2** и **Label3**.

Событие Terminate

Синтаксис:

```
Private Sub object_Terminate()
```

Где **object** — объектная метка, представляющая объектное выражение.

Создадим пример с событием **Terminate**. Перенесите на форму одну **CommandButton**. Смысл примера заключается в следующем: при загрузке формы устанавливаются первоначальные параметры; при щелчке по кнопке форма закрывается и передает управление в событие **Terminate**; при вызове события **Terminate** организуется цикл вызова диалогового окна **MsgBox**, чтобы убедиться, что после закрытия формы процедура **UserForm_Terminate** выполняется полностью, достаточно трех вызовов этого окна.

Перепишите следующий программный код:

```
Private Sub CommandButton1_Click()
Unload Me
End Sub

Private Sub UserForm_Activate()
UserForm1.Caption = "Пример с событием Terminate"
CommandButton1.Caption = "Нажми меня!"
End Sub

Private Sub UserForm_Terminate()
Dim Счетчик As Integer
For Счетчик = 1 To 3
    Beep
    MsgBox "Форма закрывается!"
Next
End Sub
```

Окно **MsgBox** вызывается три раза, после чего форма окончательно выгружается.

Событие Zoom

Синтаксис:

Для Frame:

```
Private Sub object_Zoom( Percent As Integer)
```

Для MultiPage:

```
Private Sub object_Zoom( index As Long, Percent As Integer)
```

Где:

- **object** — обязательный аргумент, любое объектное имя;
- **index** — обязательный аргумент, индекс страницы в MultiPage, связанный этим событием;
- **Percent** — обязательный аргумент, масштаб изменения формы должен быть в соответствии с допустимым масштабом. Допустимые значения колеблются от 10 до 400%.

Величина свойства масштаба **Zoom** идентифицируется как размер изменения формы или **Page**. Величина свойства указывает, что размер управления должен изменяться относительно своего текущего размера. Величины менее 100 уменьшают отображаемый размер формы; величины более 100 увеличивают отображаемый размер формы. Вы можете установить это свойство в любое целое от 10 до 400.

Создадим пример для свойства **Zoom**. Перенесите на форму одну **CommandButton**, один счетчик **SpinButton** и одно текстовое поле **TextBox**. Смысл примера в следующем: мы соединяя счетчик **SpinButton** с текстовым полем — при изменении значения счетчика его значение **Value** попадает в поле **TextBox**. Такое соединение мы уже делали, оно не должно быть новым. При нажатии на **CommandButton1** масштаб, заданный в **TextBox1**, пересчитывается. При этом необходимо помнить, что диапазон масштаба может находиться между 10 и 400. Начальное значение масштаба равно 100.

Перепишите следующий программный код:

```
Private Sub CommandButton1_Click()
    Zoom = SpinButton1.Value
End Sub

Private Sub SpinButton1_SpinDown()
    TextBox1.Text = SpinButton1.Value
End Sub

Private Sub SpinButton1_SpinUp()
    TextBox1.Text = SpinButton1.Value
End Sub

Private Sub UserForm_Initialize()
    SpinButton1.Min = 10
    SpinButton1.Max = 400
    SpinButton1.Value = 100
    TextBox1.Text = SpinButton1.Value
    CommandButton1.Caption = "Измени масштаб"
End Sub

Private Sub UserForm_Zoom(Percent As Integer)
    Dim MyZoom As Double
    If Percent > 99 Then
        ScrollBars = fmScrollBarsBoth
        ScrollLeft = 0
        ScrollTop = 0
        MyZoom = Width * Percent / 100
        ScrollWidth = MyResult
        MyZoom = Height * Percent / 100
        ScrollHeight = MyResult
    Else
        ScrollBars = fmScrollBarsNone
        ScrollLeft = 0
        ScrollTop = 0
    End If
End Sub
```

При значениях меньше 100 масштаб формы и элементов управления пропорционально уменьшается. При значениях больше 100 масштаб формы пропорционально увеличивается, но ее границы остаются без изменения. Для просмотра невидимой части формы появляются линейки прокрутки.

События, не связанные с объектами

Событие **OnTime**

Событие **OnTime** происходит в заданное время или через заданный интервал времени. При этом используется метод **OnTime**, который имеет следующий синтаксис:

```
expression.OnTime(EarliestTime, Procedure,  
LatestTime, Schedule)
```

Где:

- **expression** — обязательный аргумент, выражение, которое возвращает объект **Application**;
- **EarliestTime** — обязательный аргумент, время, когда необходимо выполнить эту процедуру;
- **Procedure** — обязательный аргумент, имя процедуры, которую необходимо выполнить;
- **LatestTime** — необязательный аргумент, время ожидания выполнения другой процедуры;
- **Schedule** — необязательный аргумент, **True**, если нужно выполнить новую процедуру **OnTime**. **False**, если нужно очистить прежде установленную процедуру. Значением по умолчанию является **True**.

Аргумент **EarliestTime** задается в долях суток. Например, 0,5 означает половину суток или, проще говоря, 12:00 дня. Задавать так время пользователю очень неудобно, но можно пользоваться функцией VBA **TimeValue**, которая переводит время в привычный для пользователя формат — доля суток.

Функция **TimeValue** имеет следующий синтаксис:

TimeValue(*time*)

Где **time** — время, заданное в любом удобном пользователю формате.

Рассмотрим пример: в заданное время, например в 12:00 дня, труба позвонит нас на заслуженный обед. Как вы понимаете, пропустить такое важное событие невозможно. Чтобы другие дела не отвлекли от этого мероприятия, напишите следующий программный код:

```
Sub Время_обедать()
Application.OnTime TimeValue("12:00:00"),
"ЗвонокТукТук"
End Sub
```

В этой программе **Время_обедать()** — это имя процедуры, которая задает программные команды, в данном случае — время звонка. С помощью функции **TimeValue** мы задаем время в удобной для нас форме. **ЗвонокТукТук** в кавычках — это имя процедуры, которая будет запущена при наступлении события **OnTime**. Сразу же под написанным кодом продолжайте программный код:

```
Sub ЗвонокТукТук ()
Beep
MsgBox "Бросай все важные дела! Хватай большую ложку!"
End Sub
```

Программа сама поймет, что вы пишете процедуру, имя которой только что оговорено. Поэтому она создаст разделительные процедурные линии. Уберите курсор из этой процедуры — хотя бы в процедуру **Время_обедать()**. Скомпилируйте приложение. Ровно в 12:00 будет выведено сообщение.

Рассмотрим следующий пример. Очень часто нам требуется задать не какое-то конкретное время, а время от текущей точки. В этом случае необходимо после имени метода **OnTime** указывать слово **Now**, затем плюс (+) и дальше — функцию

TimeValue. В нашем примере через 20 секунд после компиляции будет выведено сообщение со звуковым сигналом.

Перепишите следующий программный код:

```
Sub Время_пришло ()  
Application.OnTime Now + TimeValue("00:00:20"), "Звонок"  
End Sub  
  
Sub Звонок ()  
Beep  
MsgBox "Время пришло!"  
End Sub
```

После записи программного кода не забудьте убрать курсор из последней процедуры.

Если все процедуры мы записали в одном приложении, они николько не мешают друг другу, так как процедуры, обрабатывающие события **OnTime**, разные: **ЗвонокТукТук()** и **Звонок()**.

Событие **OnKey**

Событие **OnKey** выполняет действия при нажатии на заданное сочетание клавиш. Каждое нажатие клавиш анализируется, и если нажата заранее предусмотренная комбинация клавиш, происходит данное событие. При этом выполняется определенная процедура, когда нажимается или конкретный ключ, или клавишная комбинация.

Модификация комбинаций клавиш происходит с использованием метода **OnKey**. Этот метод имеет следующий синтаксис:

```
expression.OnKey (Key, Procedure)
```

Где:

- **expression** — обязательный аргумент. Выражение, которое возвращает объект **Application**;
- **Key** — обязательный аргумент, Стока (String). Стока, указывающая ключ, который нужно нажимать;

- **Procedure** — необязательный аргумент, тип **Variant**. Стока, указывающая имя процедуры, которая должна работать. Если **Procedure** — «» (пустой текст), ничего не происходит, когда **Key** (Ключ) нажат. **OnKey** изменяет нормальный результат нажатий клавиши в Microsoft Excel. Если **Procedure** опущена, **Key** возвращается в свой нормальный результат, и любые специальные ключевые назначения, созданные перед использованием **OnKey**, очищаются.

Замечания

Аргумент **Key** может определить любую единственную клавишу, объединенную с ALT, CTRL или SHIFT, или любую комбинацию этих клавиш. Каждая клавиша представлена одним или более символами, как, например, *a* для символа a или {ENTER} для клавиши ENTER.

Чтобы определять символы, которые не отображены, когда вы нажимаете соответствующий ключ (например, ENTER или TAB), используйте кодовые названия в следующей таблице. Каждый код в таблице 56 представляет один код клавиатуры.

Таблица 56. Соответствие кодов ключам

Ключ	Код
BACKSPACE	{BACKSPACE} или {BS}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
CLEAR	{CLEAR}
DELETE или DEL (Удалить)	{DELETE} или {DEL}
DOWN ARROW (Стрелка вниз)	{DOWN}
END	{END}
ENTER (На цифровой клавиатуре)	{ENTER}
ENTER	~ (Тильда)
ESC	{ESCAPE} или {ESC}
HELP	{HELP}
HOME	{HOME}
INS	{INSERT}
LEFT ARROW (Стрелка влево)	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN (Страница вниз)	{PGDN}
PAGE UP (Страница вверх)	{PGUP}

RETURN	{RETURN}
RIGHT ARROW (Стрелка вправо)	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW (Стрелка вверх)	{UP}
C F1 по F15	C {F1} по {F15}

Вы можете также определить клавиши в сочетании с SHIFT и/или CTRL и/или ALT. Чтобы определить клавишу, объединенную с другой клавишей или клавишами, используйте таблицу 57.

Таблица 57. Символы, заменяющие клавиши

Чтобы объединять ключи с:	Предшествует клавевому коду:
SHIFT	+ (плюс)
CTRL	^ (символ ^)
ALT	% (знак процента)

Этот пример назначает "InsertProc" в ключевой последовательности CTRL + PLUS (плюс) и правопреемника "SpecialPrintProc" в ключевую последовательность SHIFT + CTRL + RIGHT ARROW (правая стрелка):

```
Application.OnKey "+ {+}", "InsertProc"
```

```
Application.OnKey "+ ^ {RIGHT}", "SpecialPrintProc"
```

Этот пример возвращает SHIFT + CTRL + RIGHT ARROW (правая стрелка) в свое нормальное значение:

```
Application.OnKey "+ ^ {RIGHT}"
```

Этот пример отключает SHIFT + CTRL + RIGHT ARROW (правая стрелка) клавишу последовательность:

```
Application.OnKey "+ ^ {RIGHT}", ""
```

В качестве примера назначим комбинацию Ctrl и PgDn. В нашем примере при нажатии этих клавиш должен быть активизирован Лист3 и перемещен в начало перечня всех рабочих листов.

Прежде всего, в этом приложении нам совершенно не нужно создавать форму, она не нужна. В окне кода напишите следующий программный код:

```
Sub Новая_комбинация()
Application.OnKey "^{PgDn}", "PgDn_Sub"
End Sub
```

Здесь мы создаем процедуру под именем `Новая_комбинация()`. Метод `OnKey` имеет два аргумента, каждый из которых заключен в кавычки. В первом задаваемая клавиша заключена в фигурные скобки. Слева от задаваемой клавиши PgDn, вне фигурных скобок, задан символ `^`. Он означает, что в сочетании с клавишей PgDn должна быть нажата клавиша CTRL, так как именно эту клавишу заменяет символ `^` (см. таблицу выше). Никаких соединительных символов для комбинации клавиш не предусмотрено. Например, если бы мы захотели создать комбинацию клавиш Shift + Ctrl + PageDn, написали бы в первом аргументе метода `OnKey`: `"+ ^ {PgDn}"`. (Здесь плюс (+) означает клавишу Shift, верхняя крышка (^) означает клавишу Ctrl, а PageDn — клавишу PageDn, так как символьные сокращения предусмотрены только для трех клавиши: Shift, Ctrl и Alt). Символьные сокращения находятся в кавычках, но вне фигурных скобок. Второй аргумент объявляет имя процедуры, которая будет выполняться, если в ходе работы пользователя нажал на заданную комбинацию клавиш, в данном примере это Ctrl и PageDn.

Написав процедуру объявления комбинации клавиш, начинаем записывать выполняемую процедуру. Перепишите сразу же под только что записанными строками следующий программный код:

```
Sub PgDn_Sub()
Sheets ("Лист3").Select
Sheets ("Лист3").Move Before:=Sheets (1)
End Sub
```

Во время записи первой программной строки ничего особенного не происходит. Но как только мы, закончив запись первой строки, нажмем на Enter, немедленно все это будет оформлено как процедура, то есть будут выведены разделительные процедурные линии (во всяком случае, сверху, для отделения от процедуры `Новая_комбинация()`). Допишите оставшиеся

программные строки. Скомпилируйте приложение и откройте Excel. При нажатии на комбинацию клавиш Ctrl и PageDn будет открыт Лист3, и он же станет первым в списке листов. Если вручную перенести Лист3 в любое место (кроме начала списка) и открыть другой лист, при нажатии заданной комбинации клавиш будет выполняться предусмотренная нами процедура.

Рассмотрим пример, в котором мы отменяем заданную комбинацию клавиш. В этом случае во втором аргументе указывается пустая строка, то есть "":

```
Sub Новая_комбинация()
Application.OnKey "^{PgDn}", ""
End Sub
```

Событие **OnKey** происходит все равно. Но во втором аргументе не указано имя выполняющей процедуры, и поэтому комбинация клавиш игнорируется, так как не известно, как обработать такое событие.

События, связанные с отдельными элементами управления

В некоторых элементах управления есть события, присущие только им. Они выполняют действия, которые не выполняют другие элементы. Таких событий очень много, рассмотрим некоторые для примера.

AfterUpdate

Используется в элементах управления **CheckBox**, **ComboBox**, **ListBox**, **OptionButton**, **ScrollBar**, **SpinButton**, **TextBox**, **TogleButton**.

Происходит после того, как данные на элементе управления будут изменены через интерфейс пользователя. Имеет следующий синтаксис:

```
Private Sub object_AfterUpdate( )
```

Где **object** — обязательный параметр, имя объекта.

Событие **AfterUpdate** происходит независимо от того, что элемент управления связан (то есть когда свойство **RowSource** определяет источник данных для элемента управления). Это событие не может быть отменено. Если хотите отменить изменение (чтобы восстановить предшествующее значение элемента управления), используйте событие **BeforeUpdate** и устанавливайте аргумент **Cancel** в **True**.

Событие **AfterUpdate** происходит после события **BeforeUpdate** и перед событием **Exit** для текущего элемента управления, а также перед событием **Enter** для следующего управления в порядке перебора **Tab**.

События **SpinDown**, **SpinUp**

Используется в элементе управления **SpinButton**. Имеет следующий синтаксис:

```
Private Sub object_SpinDown( )
```

```
Private Sub object_SpinUp( )
```

Где **object** – обязательный параметр, имя объекта.

Событие **SpinDown** уменьшает значение **Value**. Событие **SpinUp** увеличивает **Value**.

Блокировка и разблокировка событий

По умолчанию все события разблокированы, то есть разрешены. Но иногда для предотвращения зацикливания событий необходимо их блокировать.

Для изучения такой возможности нужно изучить свойство **EnableEvents**. Оно может принимать только булевые выражения.

Следующий пример отключает события прежде, чем файл будет сохранен, чтобы событие **BeforeSave** действительно происходило.

```
Application.EnableEvents = False  
ActiveWorkbook.Save  
Application.EnableEvents = True
```

Чтобы показать, о чём, собственно, речь, создадим пример. Откройте книгу, перейдите на Лист3 (номер листа не имеет значения, но я хочу показать это на примере третьего листа) и в ячейки столбцов с А по F введите данные. Задайте каждому столбцу заголовок, чтобы наши данные не выглядели безликими и были более реальными. В столбце G создайте формулы для введенных данных. Какие конкретно формулы здесь создавать — безразлично, например, я использовал функцию расчета среднего значения СРЗНАЧ.

Создайте форму, перенесите на неё несколько элементов управления для придания правдоподобности нашему проекту.

В окне Project — VBA Project выберите Лист3 и дважды щелкните по этой записи, чтобы открыть новое окно Code. В созданном окне не будет ни одной записи. В списке Project выберите значение Worksheet. После этого будут автоматически созданы программные строки:

```
Private Sub Worksheet_SelectionChange(ByVal Target  
As Range)
```

```
End Sub
```

Нам эти строки не нужны, так как собираемся работать с событием Calculate, которое пересчитывает формулы при изменении значений в ячейках. Найдите это событие в списке Procedure и щелкните по нему. В окне Code будет создан программный код, между строчками которого допишите наш программный код. В итоге программа будет выглядеть так:

```
Private Sub Worksheet_Calculate()  
Worksheets("Лист3").Activate  
Columns("A:G").AutoFit  
End Sub
```

В этой программе мы попутно выравниваем ширину столбцов. Щелкните дважды по имени формы в окне Project — VBA Project, иначе нам предложат задать имя макросу.

Скомпилируйте проект. Закройте форму. На Листе3 в одной из ячеек измените значение и перейдите в другую ячейку. Как только вы попробуете это сделать, ваша программа немного зациклится (время зацикливания зависит от компьютера). Но все когда-то заканчивается, закончится и этот цикл.

Мы убедились: вроде бы сделали все правильно, но организуется зацикливание. Естественно, что использовать в дальнейшем такую программу явно нецелесообразно, она занимает слишком много времени.

Допишем несколько новых программных строк в эту же процедуру для ликвидации зацикливания:

```
Private Sub Worksheet_Calculate()  
Application.EnableEvents = False  
Worksheets("Лист3").Activate  
Columns("A:G").AutoFit  
Application.EnableEvents = True  
End Sub
```

В этой программе сначала блокируем события, а после завершения всех действий опять их включаем — и зацикливания не происходит. Скомпилируйте проект и опробуйте его. Убедитесь, что все работает правильно. Сохраните этот проект. В будущем в ваших программах иногда могут возникать зацикливания, а по этому примеру можно вспомнить действия по их устранению и решить проблему.

Практическая работа № 7. Разработка пользовательских диалоговых окон (форм)

В этом примере будет разработана форма для расчета подоходного налога работника и сумма к выдаче (рис. 33, 34).

```
Dim a As Double      ' начисленная сумма  
Dim b As Double      ' число детей у работника  
Dim c As Double      ' сумма вычета
```

```
Dim d As Double          ' облагаемая налогом сумма
начислений
Dim e As Double          ' сумма к выдаче

Private Sub CommandButton1_Click()
On Error Resume Next
a = TextBox1.Text
If TextBox2.Text = Empty Then
TextBox2.Text = 0
Else
b = TextBox2.Text
End If
c = 400 + 1400 * b
d = (a - c) * 0.13
If d < 0 Then
d = 0
Label7.ForeColor = RGB(255, 0, 0)
Label6.ForeColor = RGB(255, 0, 0)
Else
Label7.ForeColor = RGB(0, 0, 0)
Label6.ForeColor = RGB(0, 0, 0)
End If
e = a - d
Label6.Visible = True
Label7.Visible = True
Label6.Caption = "Подоходный налог " & d
Label7.Caption = "Необлагаемое значение " & c
TextBox3.Text = e
End Sub

Private Sub CommandButton2_Click()
On Error Resume Next
TextBox1.Text = ""
TextBox2.Text = ""
TextBox3.Text = ""
TextBox1.SetFocus
End Sub

Private Sub UserForm_Initialize()
Label1.Caption = "Начислено"
Label2.Caption = "Число детей"
```

```
Label3.Caption = "К выдаче"  
Label4.Caption = "Льгота на 1 работника = 400 руб"  
Label5.Caption = "Льгота на 1 ребенка = 1400 руб"  
Label4.Enabled = False  
Label5.Enabled = False  
CommandButton1.Caption = "Вычислить"  
CommandButton2.Caption = "Очистить"  
Me. Caption = "Расчет зарплаты"  
Label3.ForeColor = RGB(255, 0, 0)  
Label3.Font.Italic = True  
Label3.Font.Bold = True  
Label3.Font.Size = 14  
Label6.Visible = False  
TextBox1.SetFocus  
Me. BackColor = RGB(100, 255, 100)  
Label1.BackColor = RGB(100, 255, 100)  
Label2.BackColor = RGB(100, 255, 100)  
Label3.BackColor = RGB(100, 255, 100)  
Label4.BackColor = RGB(100, 255, 100)  
Label5.BackColor = RGB(100, 255, 100)  
Label6.BackColor = RGB(100, 255, 100)  
Label7.BackColor = RGB(100, 255, 100)  
Label7.Visible = False  
  
End Sub
```

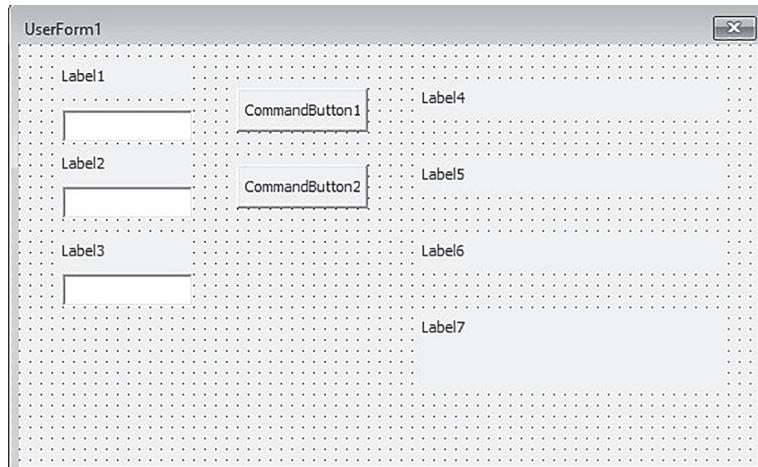


Рисунок 33. Форма с элементами в редакторе VBA

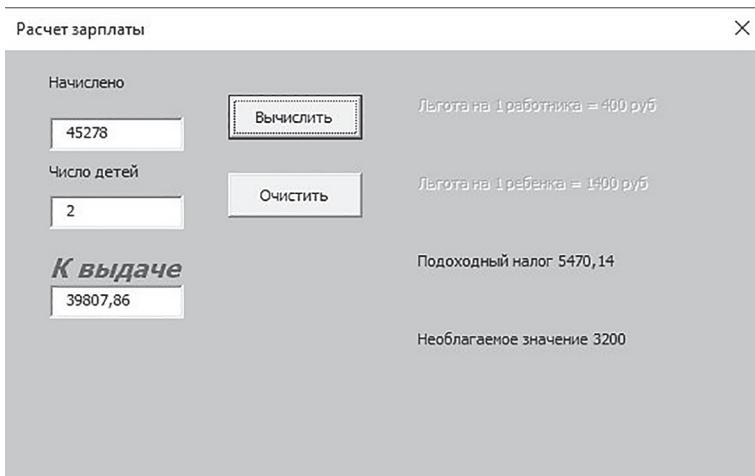


Рисунок 34. Пример расчета подоходного налога и суммы к выдаче

Ежегодно правительство устанавливает сумму вычетов, не облагаемых подоходным налогом. На момент создания примера могут быть другие суммы и правила. Мы использовали для детей единую льготу в 1400 рублей (для упрощения примера).

Обратите внимание на команды:

```
If TextBox2.Text = Empty Then  
    TextBox2.Text = 0  
Else
```

Оператор, выполняющий вычисления, может забыть ввести число детей, если оно равно 0. В этом случае будет сгенерировано значение *Empty*, то есть *Пусто*. Вследствие этого будет сгенерирована ошибка, так как *Empty* — не число. Чтобы защититься от таких естественных и часто повторяющихся ошибок, значения *Empty* автоматически заменяются на число 0.

Сумма подоходного налога может быть меньше 0. Например, если работник был в отпуске или на больничном, или в отпуске без содержания и в текущем месяце не работал или отработал всего несколько дней. В этом случае подоходный налог принудительно обнуляется (подоходный налог рассчитывается

от накопительного дохода с начала года, и отрицательные значения учитываются при расчете следующего месяца):

```
d = (a - c) * 0.13  
If d < 0 Then  
d = 0
```

После очистки полей фокус передается в поле TextBox1 для подготовки следующего вычисления:

```
TextBox1.SetFocus
```

Команда:

```
On Error Resume Next
```

запрещает закрывать форму и выводить код ошибки, если оператор в процессе работы введет недопустимое значение в одно из полей (например, буквы вместо цифр или точку вместо запятой в качестве десятичного знака).

11

Интеграция с внешними приложениями

Основы автоматизации

Все приложения офисного пакета можно связывать между собой. Например, можно вставить в документ Word таблицу из Excel, и наоборот. Можно из Word загрузить презентацию и т. д. Автоматизацией называется процесс управления одним приложением из другого. Большинство современных приложений поддерживает технологию OLE, то есть возможность использовать потенциал конкретного приложения через другое с помощью программного кода. Приложения, не применяющие технологии OLE, также можно использовать при автоматизации с помощью стандарта DDE (Dynamic Data Exchange — динамический обмен данными) и метода SendKeys.

Приложение, из которого выполняется связывание, называется приложение-клиент. Приложение, с которым связывается клиент, называется приложение-сервер. Приложение-клиент управляет приложением-сервером.

Ссылка на библиотеку объектов приложения-сервера

Существует несколько способов связывать одно приложение с другим. Например, если макрос выполняется из-под Excel, а связать нужно Excel с Word, можно выполнить команду **Tools ⇒ References**. В открывшемся окне **References ⇒ VBAProject** (рис. 35) установите флажок в опцию **Microsoft Word № Object Library**. Где № — номер версии Word. Нажмите **OK**. Если открыть это же окно еще раз, можно убедиться, что опции с флажками находятся в начальных строках окна. Для связанного приложения создается переменная, например, типа Word, если подключено приложение Word:

```
Dim wrd As Word.Application
```

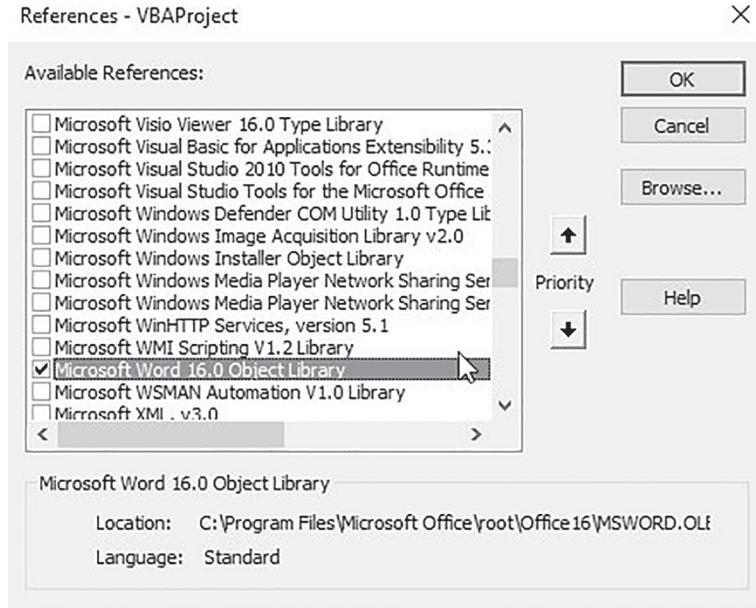


Рисунок 35. Ссылка на объект Word

Второй способ связаться с библиотекой подключаемого приложения — объявить о создании объекта Word:

```
Dim Word As Object
Set Word = CreateObject("Word.Application")
```

Первый способ более удобен: во-первых, его программный код выполняется быстрее (не менее чем в два раза) и, во-вторых, при создании программы объекты другого приложения уже присоединены, при обращении к ним выводятся подсказки. При использовании второго способа объекты другого приложения еще не присоединены, они будут присоединены только после выполнения макроса, когда подсказки уже не нужны.

В окне **References – VBAProject** установите флагок в опцию **Microsoft Word № Object Library**. В данном примере приложение Excel является клиентом, а приложение Word — сервером. Создайте Module1. Введите следующий текст:

```
Sub WordAppl()
    Dim wrd As Word.Application
    Dim new1 As Word.Document

    ' Создается новый экземпляр приложения Word
    Set wrd = New Word.Application
    ' Документ Word видим
    wrd.Visible = True

    ' Создается новый документ
    Set new1 = wrd.Documents.Add
    ' Сохранить новый документ в папке под именем
    new1.SaveAs "C:\Users\Имя_Пользователя\Desktop\
Имя_Папки\Knopka01.docx"

    ' Объекты освобождаются после использования
    Set new1 = Nothing
    Set wrd = Nothing
End Sub
```

Если не указать имя файла для сохранения («C:\Users\Имя_Пользователя\Desktop\Имя_Папки»), оно будет взято как имя после последнего разделителя, то есть Имя_Папки с форматом по умолчанию, то есть DOCX.

Отладка

Инструменты отладки

Переход в режим отладки выполняется нажатием клавиш **Ctrl + Break** или щелчком на кнопке **Break**, расположенной на инструментальной панели **Standard**.

В режим выполнения можно перейти, нажав повторно клавишу **F5** или щелкнув на кнопку **Continue**, расположенную на инструментальной панели **Standard**. Обратите внимание: в режиме отладки кнопка **Run Sub/UserForm** имеет название **Continue**.

Название текущего режима отображается в квадратных скобках в строке заголовка, расположенного на системной полосе.

При возникновении ошибки в режиме отладки выполнение программы приостанавливается в месте, где возникла ошибка. При этом все значения переменных сохраняются. Это дает дополнительную возможность для анализа ошибки.

В VBA инструменты отладки находятся в разделе меню **Debug** и на инструментальной панели с тем же наименованием.

Точка останова

Почти все инструменты отладки можно использовать только в режиме отладки.

Visual Basic содержит указатели поля в виде специальных значков. Если в строке происходит более чем одно действие, подходящие указатели также появляются.

Указатели поля появляются в полосе **Margin Indicator** (Указатель Поля) в левой стороне окна **Code**.

Вы можете включить полосу **Margin Indicator Bar** (Указатель Поля) на вкладке **Editor Format** диалогового окна **Options**.

Полоса индикатора находится слева в окне кода после установки опции **Margin Indicator Bar**. После щелчка (левой кнопкой по этой полосе) в этом месте появляется красная точка, и находящаяся рядом строка закрашивается красным. Установить или удалить точки останова можно с помощью контекстного меню или кнопки **Toggle Breakpoint** инструментальной панели **Debug**. Точки останова можно поместить в любой строке кода, включая заголовок процедуры (**Sub/Function/Property**) и строку **End**. Их нельзя установить в отдельных строках с комментариями или в пустых строках. Если на строках имеются и код, и комментарий, на них точку останова поместить можно.

Удалить ее можно или щелчком левой клавиши по точке останова, или повторным выполнением команды **Toggle Breakpoint**.

Удалить все точки останова во всем проекте можно с помощью команды **Clear All Breakpoint** раздела меню **Debug**.

Точки останова предназначены для принудительной остановки программы в нужном месте и перехода в режим отладки. В результате становятся доступными все средства отладки.

Таблица 58. Указатели полей на полосе Margin Indicator

Указатель Поля	Имя Указателя поля	Описание
	Точечный разрыв	Указывает, что установлен точечный разрыв, использовавший команду Toggle Breakpoint (Точечный разрыв переключателя) в меню Debug (Отладка). Вы можете переключить точечный разрыв, устанавливая указатель мыши в области указателя поля и щелчка
	Текущая линия выполнения	Указывает строку кода, которая выполнится следующей. Вы можете перетащить этот указатель поля в новую позицию в пределах любого кодового модуля прогона. Если перетащите линии Current (Течение) указателя поля выполнения в любую ошибочную область или строку, ничего не происходит, указатель возвращается в установленную позицию
	Закладка	Указывает позицию закладки, установленной при использовании команды Toggle Bookmark (Закладки Переключателя) в меню Edit (Редактирования)
	Маркер стека вызова	Указывает строки, которые в настоящем времени находятся в стеке вызова. Указатель Call Stack Marker (Маркер стека вызова) появляется только в точке прерывания

Пошаговое выполнение

Команды пошагового выполнения можно вызывать из раздела меню **Debug** или из инструментальной панели **Debug**.

При пошаговом выполнении строки программы выполняются последовательно друг за другом. После выполнения одной строки маркер следующей перемещается на одну строку. Пошаговое выполнение происходит:

- С заходом (Step Into);
- С выходом (Step Out);
- С обходом (Step Over).

Выполнить до текущей позиции (Run To Cursor).

Шаг с заходом позволяет не только выполнить соответствующий оператор. Если это оператор вызова процедуры или функции, он дает возможность перейти в эту процедуру. Для этого необходимо нажать кнопку **Step Into** на инструментальной панели **Debug** или клавишу F8. Это основная команда пошаговой проверки процедуры.

Команда **Step Out** раздела меню **Debug** позволяет выполнить оставшуюся часть текущей процедуры и вернуться в точку вызова. Для выполнения команды **Step Out** можно нажать кнопку инструментальной панели **Debug** или комбинацию *горячих* клавиш Ctrl + Shift + F8. Эта команда и в меню, и любым другим способом доступна только в режиме отладки. Если текущая строка находится в вызванной процедуре, с помощью команды **Step Out** остальная часть процедуры по шагам не выполняется.

Отличие команды **Step Out** от команды **Continue** в том, что после выхода из процедуры переключение в режим выполнения не происходит, если она была вызвана другой. Если не вызвана другой процедурой, происходит переход в режим выполнения, и VBA ожидает возникновения события, выполнение процедуры обработки которого начнется в режиме отладки.

Шаг с обходом похож на шаг с заходом. Различие — только при вызове текущей процедурой других процедур. Если при этом осуществляется переход в вызываемую процедуру, шаг с обходом выполняет вызов процедуры как единичный оператор, то есть без захода.

Шаг с обходом выполняется нажатием кнопки **Step Over** на инструментальной панели **Debug** или комбинации клавиш

Shift + FS. Этот вид пошагового выполнения можно использовать при поиске ошибки в процедурах, содержащих вызовы других процедур. Сначала тестируется текущая процедура без захода в вызываемые процедуры. Если при этом выяснится, что ошибка возникает в вызываемой процедуре, при следующем проходе можно будет войти в эту процедуру.

Команда **Run To Cursor** меню **Debug** выполняет программу от текущей выполняемой строки до строки, где установлен текстовый курсор. Если он находится в выполняемой строке, результат команды будет таким же, что и команды **Continue**. Для вызова команды **Run To Cursor** можно использовать комбинацию клавиш Ctrl + F8. Команда **Run To Cursor** используется, как правило, при отладке программ, содержащих циклы. Она позволяет немедленно перейти к выполнению нужного оператора, в то время как при пошаговом выполнении команда **Step Into** вызывается несколько раз.

Список вызовов

Ошибки обнаруживаются в ходе проверки программы. Для анализа последовательности их возникновения предназначено окно **Call Stack**.

Окно **Call Stack** открывается командой **View ⇒ Call Stack**, которая доступна только в режиме отладки. Чтобы открыть окно, можно воспользоваться комбинацией клавиш Ctrl + L или кнопкой с тем же именем на инструментальной панели **Debug**.

С помощью кнопки **Show** этого окна можно перейти в окно программного кода к выбранной в списке процедуре. На индикаторной полосе процедура выделена желтым фоном. Клавишей F8 выполняется поиск ошибки в тексте программы. Ошибочная строка выделяется голубым фоном.

В окне **Call Stack** отображается список имен всех выполняемых в данный момент процедур. Первым — имя текущей. За ним — список процедур в той последовательности, в которой они были вызваны. Имя процедуры обработки события указывается в конце. В результате последовательного создания записей образуется список всех вызванных процедур **Sub**, **Function** или **Property**. После завершения процедура удаляется из списка.

Заключение

Приведенные выше сведения помогут начать программировать в среде Microsoft Office.

Эти знания можно использовать на практике. Овладение VBA позволит автоматизировать ручную работу, то есть начать устранение рутинных операций. Автоматизация повторяющихся задач сделает работу более интересной, высвободит время для решения важных задач.

Автор не ставил целью дать полное описание всех возможностей программирования в среде Microsoft Excel 2021 — учебное пособие в этом случае составило бы тысячи страниц текста с иллюстрациями.

Оставшиеся возможности применения VBA предлагаю изучить на практике.

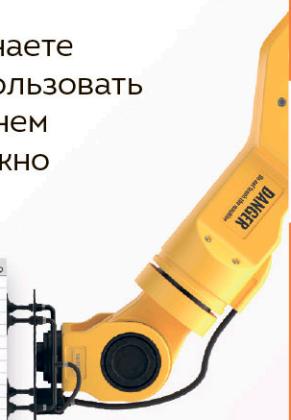
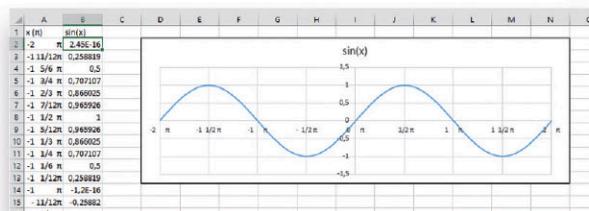
Список литературы

1. Андреева О. В. Основы алгоритмизации и программирования на VBA: учебник / О. В. Андреева, А. И. Широков. — М. : МИСИС, 2021. — 188 с. — ISBN 978-5-907227-44-6. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/178085> (дата обращения: 05.12.2022).
2. Белоусова С. Н. Основные принципы и концепции программирования на языке VBA в Excel (2-е изд.) / С. Н. Белоусова, И. А. Бессонова. — М. : НОУ «Интуит», 2016. — 191 с.
3. Бернхт Ганс-Йоахим. Измерение, управление и регулирование с помощью макросов VBA в Word и Excel (+ CD-ROM) / Ганс-Йоахим Бернхт, Буркард Каинка. — М. : МК-Пресс, Корона-Век, 2017. — 256 с.
4. Гарбер Г. З. Основы программирования на Visual Basic и VBA в Excel 2007 / Г. З. Гарбер. — М. : СОЛООН-ПРЕСС, 2016. — 191 с. — ISBN 978-5-91359-003-9. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROFобразование: [сайт]. — URL: <https://profspo.ru/books/90386> (дата обращения: 05.12.2022).
5. Гарбер Г. З. Начальный курс программирования на VBA Excel / Геннадий Гарбер. — М. : Palmarium Academic Publishing, 2017. — 240 с.
6. Гарнаев А. Ю. Excel, VBA, Internet в экономике и финансах / А. Ю. Гарнаев. — М. : БХВ-Петербург, 2015. — 397 с.
7. Гарнаев А. Использование MS Excel и VBA в экономике и финансах / Андрей Гарнаев. — М. : БХВ-Петербург, 2015. — 902 с.
8. Гарнаев А. Самоучитель VBA / Андрей Гарнаев. — М. : БХВ-Петербург, 2015. — 474 с.
9. Гуриков С. Р. Введение в программирование на языке Visual Basic for Applications (VBA). Учебное пособие / Гуриков Сергей Ростиславович/ — ИНФРА-М, 2021–316 с.
10. Кашаев С. Офисные решения с использованием Microsoft Excel 2007 и VBA / Сергей Кашаев. — М. : Питер, 2017. — 352 с.
11. Киммел. Excel 2003 и VBA. Справочник программиста / Киммел, др. П. И. — М. : Вильямс, 2017. — 262 с.

12. Кузьменко В. Г. VBA. Эффективное использование / В. Г. Кузьменко. — М. : Бином-Пресс, 2017. — 624 с.
13. Макграт М. Excel VBA. Стань продвинутым пользователем за неделю / Майк Макграт; [пер. с английского М. А. Райтмана]. — Москва: Эксмо, 2022–240 с.: ил. — (Excel для всех).
14. Основы информационных технологий: учебное пособие / С. В. Назаров, С. Н. Белоусова, И. А. Бессонова [и др.]. — 3-е изд. — Москва, Саратов: Интернет-университет информационных технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 530 с. — ISBN 978-5-4497-0339-2. — Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROФобразование: [сайт]. — URL: <https://profspo.ru/books/89454> (дата обращения: 05.12.2022).
15. Основы программирования в VBA: методические указания по курсу «Информатика» для лабораторных и контрольных работ для студентов всех специальностей и направлений подготовки. Составители: Ф. Г. Ахмадиев, И. Г. Бекбулатов, Ф. Г. Габбасов. — Казань: Изд. Казанского государственного архитектурно-строительного университета, 2013. — 36 с.
16. Сдвижков О. А. Дискретная математика и математические методы экономики с применением VBA Excel / О. А. Сдвижков. — М. : ДМК Пресс, 2016. — 212 с.
17. Сдвижков О. А. Непараметрическая статистика в MS Excel и VBA / О. А. Сдвижков. — М. : ДМК Пресс, 2017. — 172 с.
18. Сдвижков О. А. Непараметрическая статистика в MS Excel и VBA: учебное пособие / Сдвижков Олег Александрович. — М. : ДМК Пресс, 2015. — 415 с.
19. Сидорова Е. А. Обработка одномерных массивов на VBA: учебно-методическое пособие / Е. А. Сидорова, С. П. Железняк. — Омск: ОмГУПС, 2022. — 23 с. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/264509> (дата обращения: 05.12.2022).
20. Сидорова Е. А. Основы программирования на языке VBA: учебное пособие / Е. А. Сидорова, С. П. Железняк. — Омск: ОмГУПС, 2021. — 118 с. — ISBN 978-5-949-41276-3. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/190239> (дата обращения: 05.12.2022).
21. Уokenбах Дж. Excel 2013. Профессиональное программирование на VBA / Джон Уokenбах. — М. : Вильямс, 2016. — 960 с.

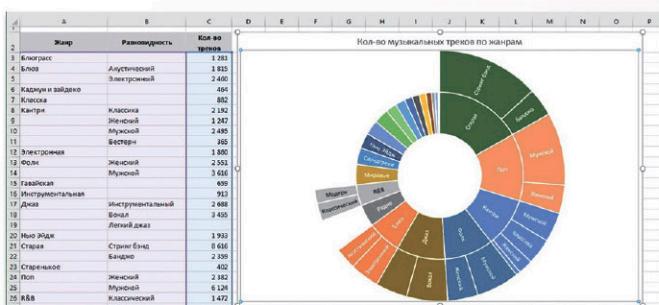
РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ EXCEL VBA, КОТОРОЕ ПОМОЖЕТ ВАМ АВТОМАТИЗИРОВАТЬ РАБОЧУЮ РУТИНУ И ИЗБАВИТЬСЯ ОТ ОДНОТИПНЫХ ЗАДАЧ!

Часто работаете с большим количеством данных? Знаете основные приемы в Excel, но еще не пробовали использовать язык Visual Basic? Тогда самое время разобраться в нем и перестать тратить время на операции, которые можно выполнить одним кликом.



С ПОМОЩЬЮ ЭТОГО РУКОВОДСТВА ВЫ НАУЧИТЕСЬ:

- ПРАВИЛЬНО ИСПОЛЬЗОВАТЬ СИНТАКСИС VBA
- ДЕЛАТЬ ТРИВИАЛЬНЫЕ ДЕЙСТВИЯ В EXCEL АВТОМАТИЧЕСКИМИ
- БЫСТРО ОБРАБАТЫВАТЬ ЛЮБЫЕ ДАННЫЕ И ВИЗУАЛИЗИРОВАТЬ ИХ В ПОДХОДЯЩЕМ ФОРМАТЕ
- ЛЕГКО ПОДСТРАИВАТЬ ФУНКЦИИ VBA ПОД СВОИ НУЖДЫ
- ПИСАТЬ ЛЮБОЙ КОД САМОСТОЯТЕЛЬНО



И, КОНЕЧНО
ЖЕ, РАБОТАТЬ
С ДАННЫМИ
БОЛЕЕ
ЭФФЕКТИВНО!

 **БОМБОРА**
ИЗДАТЕЛЬСТВО

БОМБОРА – лидер на рынке полезных и вдохновляющих книг.
Мы любим книги и создаем их, чтобы вы могли творить, открывать мир, пробовать новое, расти. Быть счастливыми. Быть на волне.

 bombora.ru  [bomborabooks](http://bomborabooks.com)  [bombora](http://bombora.com)

ISBN 978-5-04-180209-7



9 785041 802097 >