Никита Культин
Лариса Цой

Visual Basic
ДЛЯ СТУДЕНТОВ И ШКОЛЬНИКОВ

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.068+800.92Visual Basic
ББК 32.973.26–018.1
К90

Культин, Н. Б.

К90 Visual Basic для студентов и школьников / Н. Б. Культин, Л. Б. Цой. — СПб.: БХВ-Петербург, 2010. — 416 с.: ил. + CD-ROM — (ИиИКТ)

ISBN 978-5-9775-0487-4

Рассматривается процесс создания программ различного назначения на языке программирования Visual Basic — от простейших до программ работы с графикой и базами данных. Последовательность изложения, дозировка материала, а также наличие контрольных вопросов и задач для решения соотносятся с учебным процессом. Демонстрируется среда разработки Visual Basic, приводится описание языка программирования Visual Basic, рассматриваются основные алгоритмические структуры, операции со строками, одномерными и двухмерными массивами и файлами, большое внимание уделено практике программирования, что позволит полноценно подготовиться к ЕГЭ по информатике по разделам, касающимся алгоритмизации и программирования. Приложение содержит справочник по языку программирования Visual Basic и базовым компонентам. На компакт-диске приводятся рассматриваемые в книге примеры программ и программа Экзаменатор, позволяющая автоматизировать процесс контроля и самоконтроля знаний.

Для образовательных учреждений

УДК 681.3.068+800.92Visual Basic
ББК 32.973.26–018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Людмила Еремеевская*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 04.12.09.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 33,54.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0487-4

Оглавление

Предисловие

Visual Basic — что это?

В последнее время возрос интерес к программированию. Это связано с развитием и внедрением в повседневную жизнь информационно-коммуникационных технологий. Если кто-то имеет дело с компьютером, то рано или поздно у него возникает желание, а иногда и необходимость, программировать.

Среди пользователей персональных компьютеров в настоящее время наиболее популярна операционная система Windows, и естественно, что тот, кто собирается программировать, стремится писать программы, которые будут работать в ней.

Раньше начинающему программисту оставалось только мечтать о создании собственных программ, работающих в Windows, т. к. средства разработки были явно ориентированы на профессионалов, обладающих серьезными знаниями и опытом.

Бурное развитие вычислительной техники, потребность в эффективных средствах разработки программного обеспечения привели к появлению систем программирования, ориентированных на так называемую "быструю разработку", пионером среди которых был пакет Microsoft Visual Basic.

В основе систем быстрой разработки (RAD-систем, Rapid Application Development — среда быстрой разработки приложений) лежит технология визуального проектирования и событийного программирования. Суть этой технологии заключается в том, что среда разработки берет на себя большую часть рутинной работы, оставляя программисту работу по созданию диалоговых окон и функций обработки событий. Производительность программиста при работе в RAD-системе — фантастическая!

Microsoft Visual Basic — это среда быстрой разработки, в которой в качестве языка программирования используется Visual Basic.

В настоящее время, несмотря на появление новых версий Visual Basic, широко используется шестая версия пакета — Microsoft Visual Basic 6.0, которая стала "классикой".

В Visual Basic 6.0 можно создавать программы различного назначения: от простейших однооконных приложений, до программ, работающих с графикой, мультимедиа и базами данных.

Microsoft Visual Basic может работать в среде операционных систем от Windows 98 до Windows Vista. Особых требований, по современным меркам, к ресурсам компьютера пакет не предъявляет.

Об этой книге

В книге, которая посвящена программированию в конкретной среде разработки, необходим баланс между тремя линиями: язык программирования, технология программирования (программирование как таковое) и среда разработки. Уже при первом знакомстве со средой разработки, представлении ее возможностей у автора возникает проблема: чтобы описать процесс разработки программы, объяснить, как работает программа, нужно оперировать такими терминами как *объект, событие, свойство,* понимание которых на начальном этапе изучения программирования весьма проблематично. Как поступить? Сначала дать описание языка, а затем приступить к описанию среды разработки и процесса программирования? Очевидно, что это не лучший вариант решения. Поэтому при изложении материала принят подход, который можно назвать "от задачи". Суть его заключается в том что, берется конкретная задача и на ее примере рассматривается определенная технология, возможности среды разработки и особенности языка программирования, необходимые для решения этой, конкретной задачи.

Книга, которую вы держите в руках, — это не описание языка программирования и среды разработки Visual Basic. Это руководство по программированию в Microsoft Visual Basic. В нем рассмотрена вся цепочка, весь процесс создания программы: от разработки алгоритма, диалогового окна и процедур обработки событий до установки на компьютер пользователя.

Цель этой книги — научить программировать в Microsoft Visual Basic 6.0, т. е. создавать законченные программы различного назначения: от простых однооконных приложений до программ работы с базами данных.

Научиться программировать можно, только решая конкретные задачи. Поэтому, чтобы получить максимальную пользу от книги, вы должны работать с ней активно. Не занимайтесь просто чтением примеров, реализуйте их с помощью вашего компьютера. Не бойтесь экспериментировать — вносите изменения в программы. Чем больше вы сделаете самостоятельно, тем большему вы научитесь!

Глава 1

Среда программирования Visual Basic

Запускается Visual Basic обычным образом, т. е. выбором в меню **Пуск** команды **Программы ▸ Microsoft Visual Basic 6.0 ▸ Microsoft Visual Basic 6.0** (рис. 1.1).

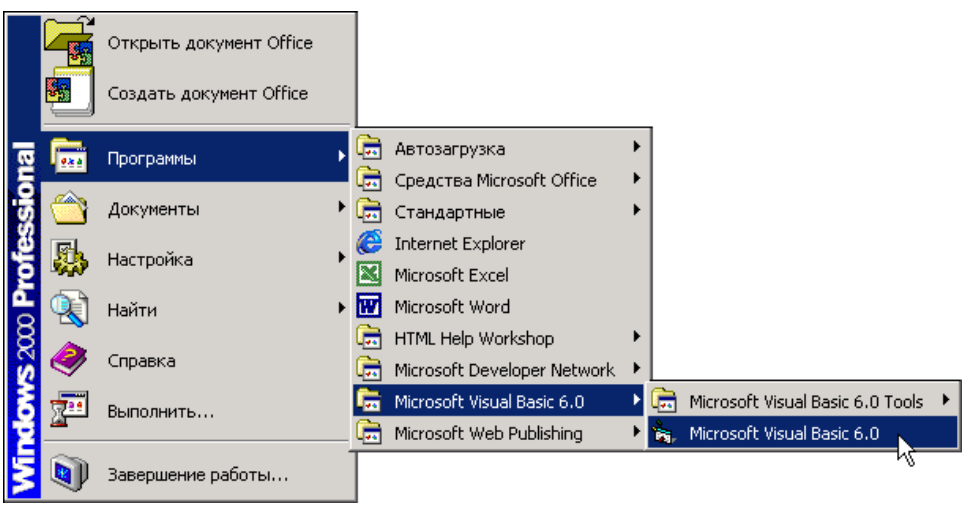


Рис. 1.1. Запуск Visual Basic

Перед тем как запустить Visual Basic первый раз, рекомендуется создать папку для проектов (программ) Visual Basic и указать ее в качестве рабочей. Папка (назвать ее можно, например, VBProjects) создается обычным образом. Чтобы указать, что папка является рабочей, надо раскрыть окно свойств команды запуска Visual Basic (сделать щелчок *правой* кнопкой мыши на команде запуска Visual Basic) и ввести имя в поле **Рабочая папка**.

Если Visual Basic запущен сразу после установки, то на фоне главного окна отображается окно **New Project** (рис. 1.2). В этом окне на вкладке **New** перечислены типы проектов (программ), которые можно создать в Visual Basic.

Чтобы приступить к работе над новой программой, или, как принято говорить, *приложением*, надо выбрать **Standard EXE** и сделать щелчок на кнопке **Открыть**.

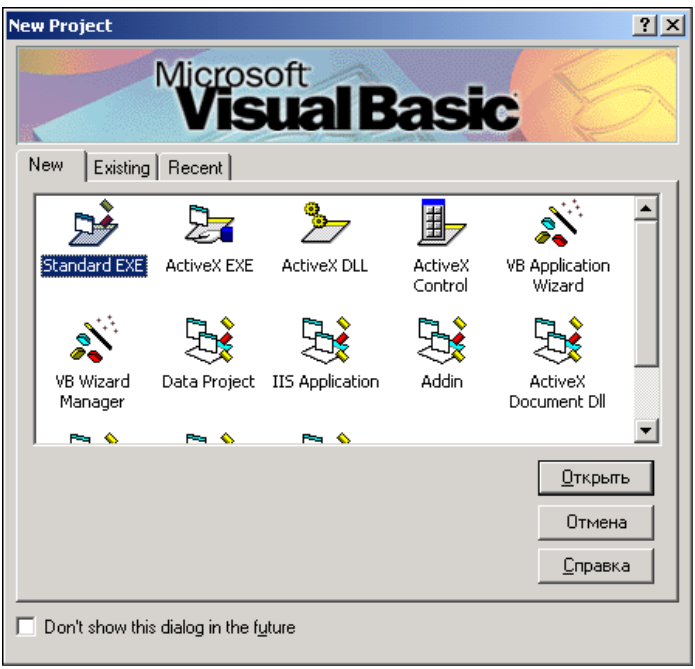


Рис. 1.2. Начало работы над новой программой (приложением)

Замечание

Если после запуска Visual Basic окно **New Project** на экране не отображается, то, для того чтобы начать работу над новой программой, надо в меню **File** выбрать команду **New Project**.

Окно Visual Basic в начале работы над новой программой приведено на рис. 1.3. В верхней части окна находится строка меню и панель инструментов, слева — *палитра компонентов*, в центре — окно *конструктора формы*, справа — окно *проекта*, окно *свойств* и окно *отображения положения формы*.

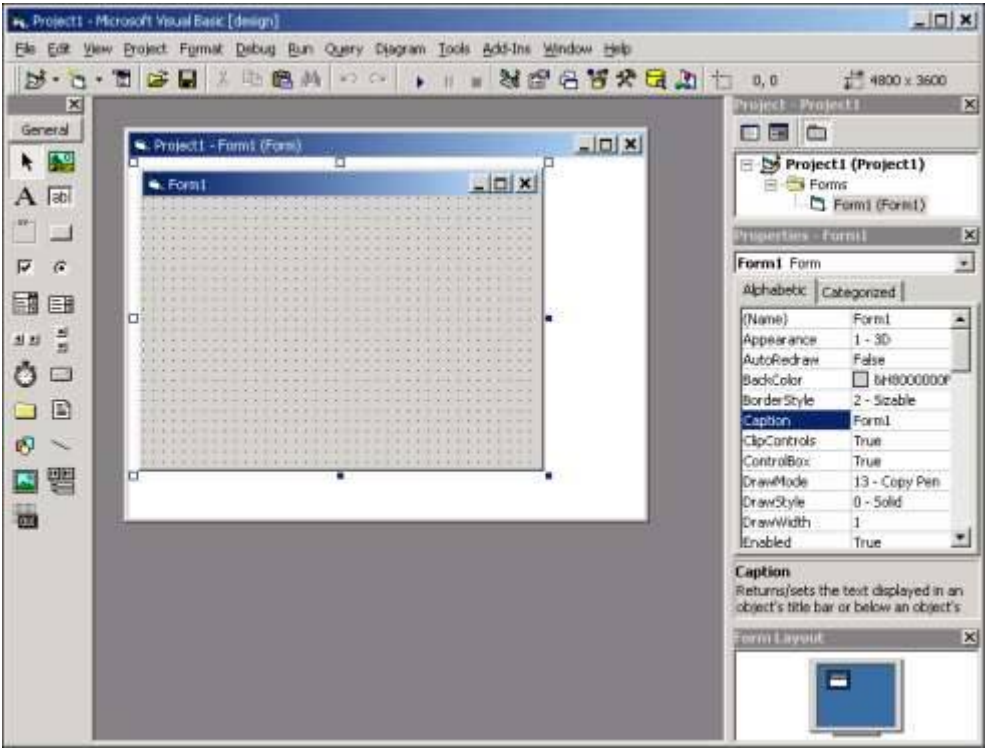


Рис. 1.3. Окно Visual Basic в начале работы над новой программой

На стандартной панели инструментов (рис. 1.4) находятся кнопки активизации наиболее часто используемых команд. Там же находятся кнопки, используя которые можно быстро сделать доступным окно палитры компонентов, менеджера проектов, свойств и др.

Рис. 1.4. Стандартная панель инструментов

Окно программы во время ее разработки принято называть формой.

В окне конструктора формы (рис. 1.5) находится *форма* — заготовка окна разрабатываемого приложения.

В палитре компонентов (рис. 1.6) отображаются значки компонентов, которые программист может поместить на форму.

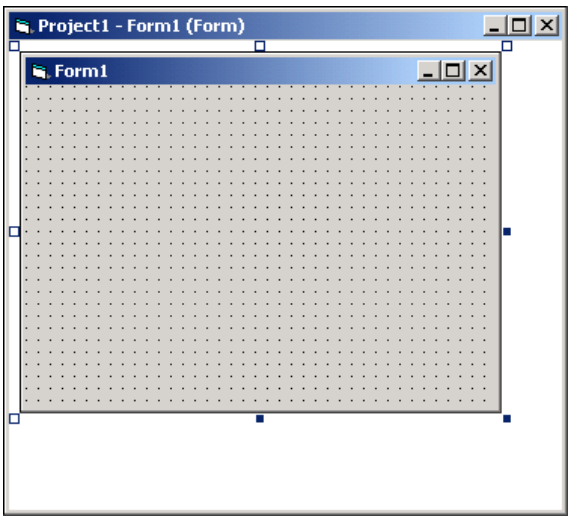


Рис. 1.5. Окно конструктора формы

Рис. 1.6. Палитра компонентов (окно **Toolbox**)

Замечание

Если палитра компонентов не отображается, то, для того чтобы она стала доступной, надо в меню **View** выбрать команду **Toolbox** или сделать щелчок на соответствующей кнопке панели инструментов.

В окне проекта (рис. 1.7) отображается структура (состав) проекта, над которым в данный момент идет работа.

Замечание

Если окно проекта не отображается, то, для того чтобы оно стало доступным, надо в меню **View** выбрать команду **Project Explorer** или сделать щелчок на соответствующей кнопке панели инструментов.

Окно **Properties** (рис. 1.8) предназначено для редактирования значений свойств объектов. В нем отображаются свойства выбранного в данный момент объекта (в начале работы над новой программой — формы). На вкладке **Alphabetic** свойства отображаются в алфавитном порядке,

а на вкладке **Categorized** свойства сгруппированы по функциональному признаку. Например, в группу **Appearance** объединены свойства, которые определяют вид объекта, а в группу **Position** — его размер и положение на экране (для формы) или поверхности формы (другие компоненты).

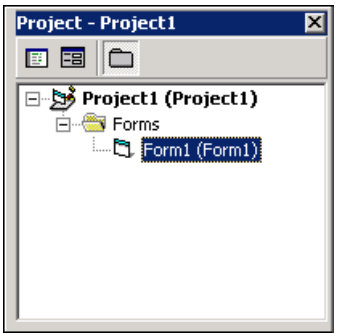


Рис. 1.7. В окне **Project** отображается структура (состав) проекта

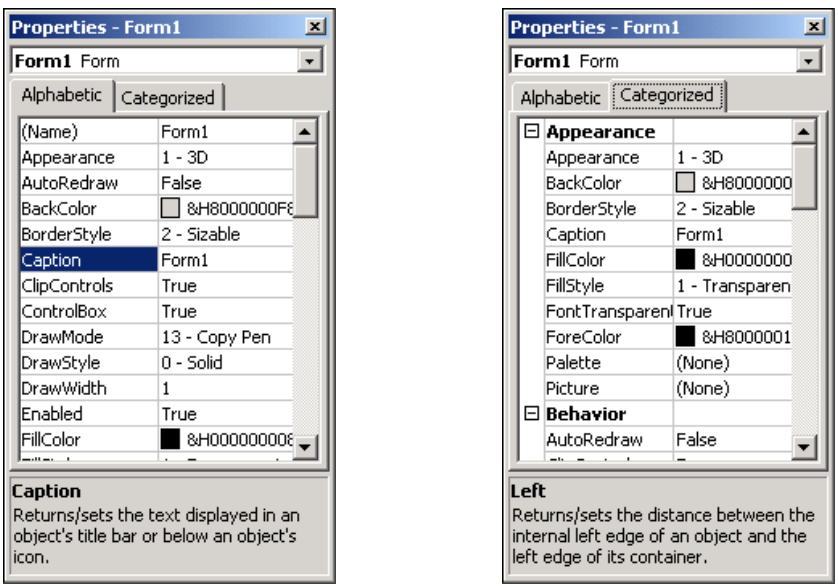


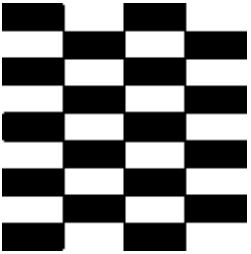
Рис. 1.8. В окне **Properties** перечислены свойства объекта и указаны их значения

В терминологии визуального проектирования *объект* — это диалоговое окно или элемент интерфейса пользователя (поле ввода, командная кнопка, пере-

ключатель и др.). *Свойство* — это характеристика, которая определяет (задает) внешний вид объекта. Например, значение свойства `Caption` задает заголовок формы, а свойств `Width` и `Height` — ее размер.

Контрольные вопросы

1. Что такое приложение?
2. Перечислите основные окна среды разработки Visual Basic.
3. Что надо сделать, если какое-либо из окон, например окно свойств, на экране не отображается?
4. Как начать работу над новой программой?
5. Что такое свойство?



Глава 2

Первый проект

Процесс создания программы в Visual Basic рассмотрим на примере. Создадим приложение, которое позволяет пересчитать цену из долларов в рубли, — Конвертер (рис. 2.1).

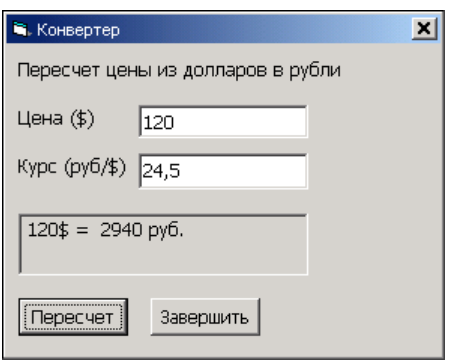


Рис. 2.1. Окно программы пересчета цены из долларов в рубли

Начало работы

Чтобы начать работу над новой программой, или, как принято говорить, проектом, надо сначала в меню **File** выбрать команду **New Project**, затем в открывшемся окне **New Project** указать тип создаваемого приложения (**Standard EXE**) и сделать щелчок на кнопке **OK**.

Форма

Работа над новой программой начинается с создания стартовой (главной) формы.

Стартовая форма создается путем изменения значений ее свойств (табл. 2.1) и добавления к форме необходимых компонентов (полей ввода, отображения текста, командных кнопок и т. д.).

Таблица 2.1. *Свойства формы*

Свойство	Описание
`Name`	Имя формы. Используется для доступа к форме и ее компонентам
`Caption`	Текст заголовка
`Width`	Ширина формы
`Height`	Высота формы
`StartUpPosition`	Положение формы при первом ее появлении на экране. Форма может располагаться в центре экрана (`Center Screen`), в центре родительской формы (`Center Owner`). Положение формы могут определять также значения свойств `Top` и `Left` (в этом случае значение свойства `StartUpPosition` должно быть равно `Manual`)
`Top`	Расстояние от верхней границы формы до верхней границы экрана или до верхней границы родительской формы
`Left`	Расстояние от левой границы формы до левой границы экрана или до левой границы родительской формы
`Icon`	Значок (картинка) в заголовке
`MaxButton`	Признак наличия в заголовке окна кнопки **Развернуть**
`MinButton`	Признак наличия в заголовке окна кнопки **Свернуть**
`BorderStyle`	Стиль (вид) границы. Граница может быть обычной (`Sizable`), тонкой (`Fixed Single`) (в этом случае изменить размер окна путем перемещения границы мышью нельзя) или вообще отсутствовать (`None`). Если значение свойства равно `Fixed Dialog`, то граница окна тонкая и кнопки **Развернуть** и **Свернуть** в заголовке не отображаются
`BackColor`	Цвет формы. Цвет можно задать, выбрав из палитры или указав привязку к элементу цветовой схемы операционной системы. Во втором случае цвет определяется текущей цветовой схемой и выбранным компонентом привязки и меняется при изменении цветовой схемы операционной системы
`ScaleMode`	Единица измерения размеров и координат компонентов, находящихся на форме. Размер и координаты компонентов могут измеряться в твипах (`Twip`), пикселах (`Pixel`), миллиметрах (`Millimeter`) и других единицах

Таблица 2.1 *(окончание)*

Свойство	Описание
`Font`	Шрифт, который по умолчанию используется находящимися на поверхности формы компонентами для отображения текста (например, надпись на командной кнопке, текст в поле редактирования или в поле отображения текста)

Для изменения значений свойств формы и компонентов используется окно **Properties**. В верхней части окна **Properties** указано имя объекта, значения свойств которого отображаются в данный момент. В левой колонке окна перечислены свойства объекта, в правой — указаны значения свойств.

Сначала надо задать заголовок формы — изменить значение свойства `Caption` с `Form1` на `Конвертер`. Чтобы это сделать, нужно в окне **Properties** выбрать свойство `Caption` и щелкнуть мышью в поле значения свойства. В результате этих действий в поле значения свойства (после слова `Form1`) появится курсор и можно будет ввести значение свойства (рис. 2.2).

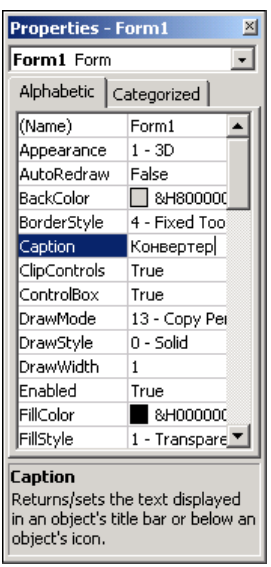


Рис. 2.2. Изменение значения свойства путем ввода строки

Следует обратить внимание, что ширина и высота формы измеряются в специальных единицах — твипах. Задавать значения свойств `Width` и `Height` в твипах неудобно. Проще захватить один из находящихся на границе формы

черных квадратиков и переместить границу (вертикальную, горизонтальную или обе сразу) в нужном направлении (рис. 2.3). По окончании перемещения границы значения свойств `Width` и `Height` автоматически изменятся и будут соответствовать установленному размеру формы.

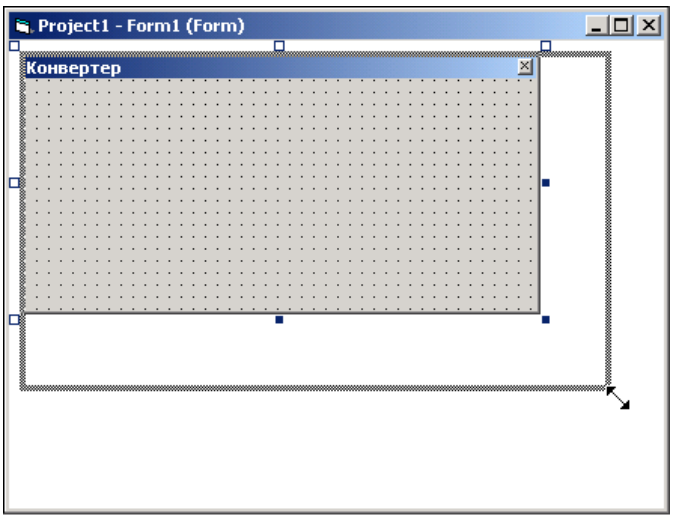


Рис. 2.3. Изменение размера формы путем перемещения границы

При выборе некоторых свойств, например `BorderStyle`, справа от текущего значения свойства появляется значок раскрывающегося списка. Очевидно, что значение таких свойств можно задать путем выбора из списка (рис. 2.4). Здесь следует обратить внимание, что в списке сначала указывается числовое значение константы, а затем — ее название. При этом не следует путать название константы и ее обозначение. Например, численное значение константы `Fixed Single` равно единице, а символьное значение — `vbFixedSingle`.

Рядом со значениями некоторых свойств отображается командная кнопка с тремя точками. Это значит, что для изменения значения свойства используется дополнительное диалоговое окно. Например, в результате щелчка на кнопке с тремя точками в строке свойства `Icon` открывается окно **Load Icon**, в котором можно открыть один из каталогов компьютера и выбрать ICO-файл — картинку, которая будет изображать системное меню в заголовке формы.

В табл. 2.2 приведены значения свойств стартовой формы разрабатываемой программы. Остальные свойства оставлены без изменения и в таблице не приведены.

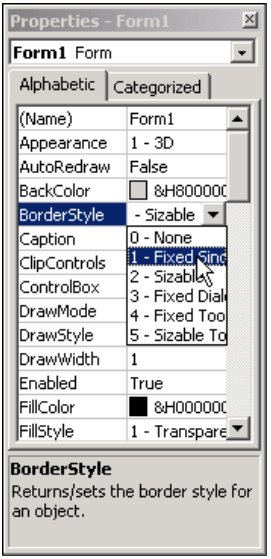


Рис. 2.4. Изменение значения свойства путем выбора из списка

Таблица 2.2. *Значения свойств формы*

Свойство	**Значение**
Caption	Конвертер
Width	4425
Height	3480
BorderStyle	FixedSingle
MaxButton	False
MinButton	False
StartUpPosition	CenterScreen
ScaleMode	Pixel
Font	Tahoma, обычный, 10 pt

Компоненты

Программа пересчета цены из долларов в рубли должна получить от пользователя исходные данные — цену в долларах и курс (соотношение рубля к доллару). Ввод данных с клавиатуры обеспечивает компонент `TextBox` — поле редактирования. Поэтому в форму надо добавить два компонента `TextBox`.

Чтобы добавить в форму компонент TextBox, нужно в палитре компонентов сделать щелчок на значке компонента (рис. 2.5). Затем установить указатель мыши в ту точку формы, в которой должен быть левый верхний угол компонента. Потом нажать левую кнопку мыши и переместить указатель мыши в точку, в которой должен быть правый нижний угол компонента. После того как кнопка мыши будет отпущена, на форме появится компонент.

Рис. 2.5. Значок компонента TextBox

Каждому компоненту, добавленному в форму, автоматически присваивается имя, которое формируется из стандартного имени компонента и его порядкового номера. Например, первый компонент TextBox получает имя Text1, второй — Text2. Программист, путем изменения значения свойства Name, может изменить имя компонента. В простых программах имена компонентов, как правило, не изменяют.

На рис. 2.6 приведен вид формы программы "Конвертер" после добавления двух компонентов TextBox, предназначенных для ввода исходных данных. Один из компонентов выделен. Свойства этого (выделенного) компонента отображаются в окне **Properties**. Чтобы увидеть свойства другого компонента, надо щелкнуть левой кнопкой мыши на его изображении в форме или выбрать имя нужного компонента в раскрывающемся списке, который находится в верхней части окна **Properties**.

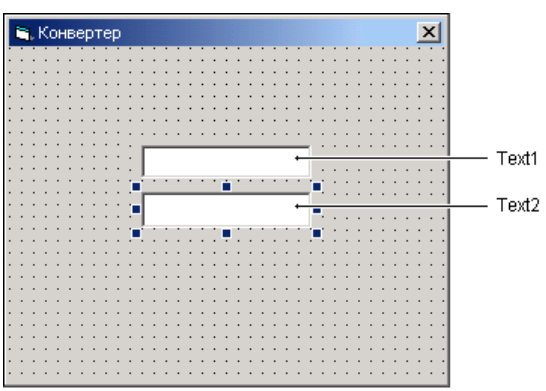


Рис. 2.6. Форма после добавления компонентов TextBox

В табл. 2.3 перечислены основные свойства компонента TextBox.

Таблица 2.3. *Свойства компонента* `TextBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Text`	Текст, находящийся в поле редактирования
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Width`	Ширина поля редактирования
`Height`	Высота поля редактирования
`Font`	Шрифт, используемый для отображения текста в поле компонента
`ForeColor`	Цвет текста
`Locked`	Блокировка. Используется для ограничения возможности изменения текста в поле редактирования. Если значение свойства равно `True`, то текст в поле редактирования изменить нельзя
`MultiLine`	Разрешает (`True`) отображение текста в несколько строк
`ScrollBars`	Управляет отображением полос прокрутки. У компонента может быть полоса вертикальной прокрутки (`Vertical`), горизонтальной (`Horizontal`) или обе полосы прокрутки (`Both`). Если значение свойства равно `None`, то полосы прокрутки не отображаются
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

Изменить размер и положение компонента на форме можно, присвоив нужные значения соответственно свойствам `Width`, `Height`, `Left` и `Top` или при помощи мыши.

Для того чтобы изменить положение компонента, необходимо установить указатель мыши на его изображении, нажать левую кнопку мыши и, удерживая кнопку нажатой, переместить контур компонента в нужную точку формы, затем отпустить кнопку мыши. Во время перемещения компонента (рис. 2.7) отображаются текущие значения координат левого верхнего угла компонента (значения свойств `Left` и `Top`).

Для того чтобы изменить размер компонента, необходимо установить указатель мыши на один из маркеров, помечающих границу компонента, нажать левую кнопку мыши и, удерживая кнопку нажатой, изменить положение гра-

ницы компонента. Затем отпустить кнопку мыши. Во время изменения размера компонента отображаются текущие значения свойств `Height` и `Width` (рис. 2.8).

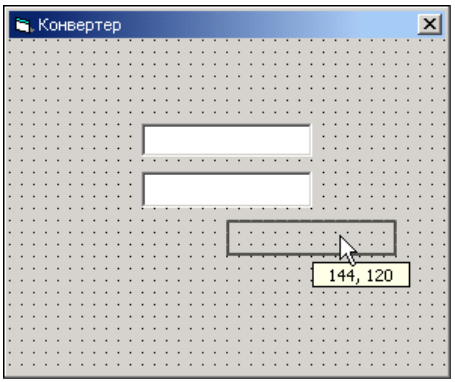


Рис. 2.7. Отображение текущих значений свойств `Left` и `Top` при изменении положения компонента

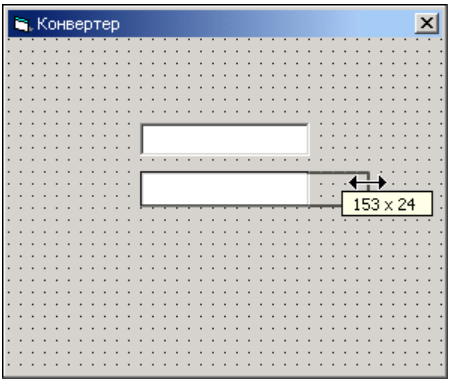


Рис. 2.8. Отображение текущих значений свойств `Width` и `Height` при изменении размера компонента

В табл. 2.4 приведены значения свойств компонентов `Text1` и `Text2`. Компонент `Text1` предназначен для ввода цены, `Text2` — курса доллара. Обратите внимание на то, что значением свойства `Text` обоих компонентов является пустая строка. Также следует обратить внимание, что координаты и размер компонентов указаны в пикселах.

Таблица 2.4. *Значения свойств компонентов `Text1` и `Text2`*

Компонент	Свойство	Значение
Text1	Left	88
	Top	47
	Width	113
	Height	22
	Text	—
Text2	Left	88
	Top	87
	Width	113
	Height	22
	Text	—

Помимо полей редактирования в окне программы должна находиться краткая информация о программе и назначении полей редактирования. Отображение текста на поверхности формы обеспечивает компонент `Label` (рис. 2.9). Добавляется компонент `Label` в форму точно так же, как и поле редактирования.

Свойства компонента `Label` перечислены в табл. 2.5.

Рис. 2.9. Компонент `Label` — поле отображения текста

Таблица 2.5. *Свойства компонента* `Label`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Caption`	Текст, отображаемый в поле компонента
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Width`	Ширина компонента (поля отображения текста)
`Height`	Высота компонента (поля отображения текста)
`AutoSize`	Признак автоматического изменения размера компонента в соответствии с его содержимым
`WordWrap`	Признак необходимости переноса текста, который не помещается в текущей строке, в следующую строку (значение свойства `AutoSize` должно быть `False`)
`Alignment`	Способ выравнивания текста в поле компонента. Текст может быть выровнен по левому краю (`LeftJustify`), по центру (`Center`) или по правому краю (`RightJustify`)
`Font`	Шрифт, используемый для отображения текста. По умолчанию используется шрифт, заданный для формы
`ForeColor`	Цвет текста в поле компонента
`BorderStyle`	Задает вид границы компонента. По умолчанию граница отсутствует
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

В форму разрабатываемого приложения надо добавить четыре компонента Label. Первый компонент Label обеспечивает вывод информационного сообщения, второй и третий — отображение информации о назначении полей редактирования, четвертый — отображение результата расчета.

После того как компоненты Label будут добавлены в форму, надо выполнить их настройку — установить значения свойств в соответствии с табл. 2.6.

Таблица 2.6. *Значения свойств компонентов* Label1, Label2, Label3 *и* Label4

Компонент	Свойство	Значение
Label1	Caption	Пересчет цены из долларов в рубли
	Left	8
	Top	8
	AutoSize	True
Label2	Caption	Цена ($)
	Left	8
	Top	40
	AutoSize	True
Label3	Caption	Курс (руб/$)
	Left	8
	Top	72
	AutoSize	True
Label4	Caption	—
	Left	8
	Top	112
	Width	193
	Height	41
	BorderStyle	Fixed Single

Форма программы после настройки компонентов Label должна выглядеть так, как показано на рис. 2.10.

Последнее, что надо сделать на этапе создания формы, — добавить в форму две командные кнопки: **Пересчет** и **Завершить**. Назначение этих кнопок очевидно.

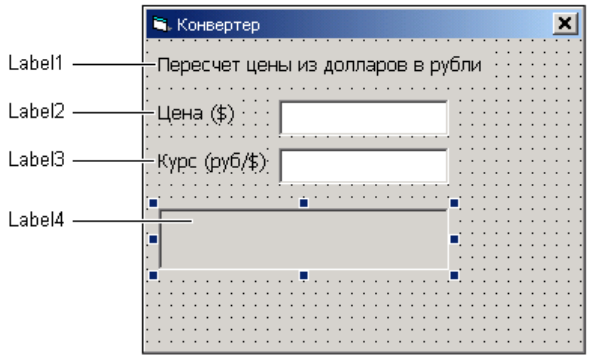


Рис. 2.10. Форма после добавления полей отображения текста

Рис. 2.11. Командная кнопка — компонент `CommandButton`

Командная кнопка — компонент `CommandButton` (рис. 2.11) — добавляется в форму точно так же, как и другие компоненты. Свойства компонента приведены в табл. 2.7.

Таблица 2.7. *Свойства компонента* `CommandButton`

Свойство	**Описание**
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Caption`	Текст на кнопке
`Left`	Расстояние от левой границы кнопки до левой границы формы
`Top`	Расстояние от верхней границы кнопки до верхней границы формы
`Width`	Ширина кнопки
`Height`	Высота кнопки
`Enabled`	Признак доступности кнопки. Если значение свойства равно `True`, то кнопка доступна и пользователь может ее нажать, сделав щелчок на кнопке левой кнопкой мыши или, если фокус находится на кнопке, нажав клавишу <Enter>. Если значение свойства равно `False`, то кнопка не доступна

Таблица 2.7 *(окончание)*

Свойство	Описание
Visible	Позволяет скрыть кнопку (False) или сделать ее видимой (True)
Style	Тип кнопки. Кнопка может быть обычной (Standard) или "графической" (Graphical). На поверхности "графической" кнопки отображается картинка
Picture	Картинка, которая отображается на "графической" кнопке (значение свойства Style должно быть равно Graphical)
DisabledPicture	Для "графической" кнопки задает картинку, которая отображается на кнопке в случае, если она не доступна (значение свойства Enabled равно False)
DownPicture	Для "графической" кнопки (значение свойства Style равно Graphical) задает картинку, которая отображается на кнопке в случае, если она нажата
ToolTipText	Задает текст подсказки, которая появляется при позиционировании указателя мыши на кнопке

После добавления к форме двух командных кнопок (компонентов CommandButton) нужно установить значения их свойств в соответствии с табл. 2.8.

Таблица 2.8. *Значения свойств компонентов Command1 и Command2*

Компонент	Свойство	Значение
Command1	Caption	Пересчет
	Left	8
	Top	168
	Width	73
	Height	25
Command2	Caption	Завершить
	Left	96
	Top	168
	Width	73
	Height	25

Окончательный вид формы разрабатываемого приложения "Конвертер" приведен на рис. 2.12.

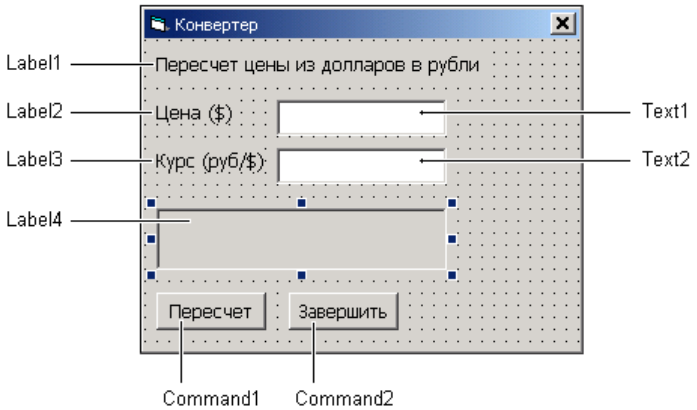


Рис. 2.12. Форма программы "Конвертер"

Завершив работу над формой приложения, можно приступить к программированию — созданию процедур обработки событий.

Событие и процедура обработки события

Вид созданной формы подсказывает, как работает приложение. Очевидно, что пользователь должен ввести в поля редактирования исходные данные и сделать щелчок на кнопке **Пересчет**. Щелчок на изображении командной кнопки — это пример того, что называется *событием*.

Событие (`Event`) — это то, что происходит во время работы программы. У каждого события есть имя. Например, щелчок кнопкой мыши на изображении командной кнопки — это событие `Click`, нажатие клавиши в процессе ввода строки текста в поле компонента `TextBox` — событие `KeyPress`.

В табл. 2.9 приведены некоторые события.

Таблица 2.9. *События*

Событие	Описание
`Click`	Щелчок кнопкой мыши
`DblClick`	Двойной щелчок кнопкой мыши
`MouseDown`	Нажатие кнопки мыши
`MouseUp`	Отпускание кнопки мыши
`MouseMove`	Перемещение мыши
`KeyPress`	Нажатие клавиши

Таблица 2.9 *(окончание)*

Событие	Описание
KeyDown	Нажатие клавиши. События KeyDown и KeyPress — это чередующиеся, повторяющиеся события, которые происходят до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие KeyUp)
KeyUp	Отпускание нажатой (удерживаемой) клавиши
Initialize	Создание объекта (например, формы). Процедура обработки этого события обычно используется для инициализации переменных и выполнения подготовительных действий
Activate	Событие происходит, когда элемент управления (форма) становится активным окном
Paint	Событие происходит при появлении окна на экране в начале работы программы, а также во время работы программы, когда окно вновь становится видимым, например, после того как пользователь развернет свернутое окно или отодвинет другое окно, которое перекрывает окно программы
Resize	Изменение размера формы или элемента управления
GotFocus	Получение элементом управления фокуса, например перемещение курсора в поле редактирования текста
LostFocus	Потеря элементом управления фокуса, например перемещение курсора из одного поля редактирования в другое

Следует обратить внимание, что действия пользователя, как правило, приводят к возникновению последовательности (цепочки) событий. Например, в начале работы программы возникает цепочка событий Initialize — Load — Activate — Resize — Paint, а в конце работы программы, когда пользователь сделает щелчок на кнопке **Закрыть**, — QueryUnload — Unload — Terminate.

Реакцией на событие должно быть какое-либо действие. В Visual Basic реакция на событие реализуется как *процедура обработки события*. Таким образом, для того чтобы программа выполняла некоторую работу в ответ на действия пользователя, программист должен написать процедуру обработки соответствующего события. Следует обратить внимание на то, что значительную часть обработки событий берет на себя компонент. Поэтому программист должен разрабатывать процедуру обработки события только в том случае, если реакция на событие отличается от стандартной или не определена. Например, если по условию задачи ограничений на символы, вводимые в поле редактирования (компонент TextBox), нет, то процедуру обработки

события `KeyPress` для этого компонента писать не надо, т. к. во время работы программы будет использована стандартная (скрытая от программиста) процедура обработки этого события.

Методику создания процедур обработки событий рассмотрим на примере процедуры обработки события `Click` для командной кнопки **Пересчет**. Процедура обработки этого события должна получить исходные данные из полей редактирования `Text1` (цена в долларах) и `Text2` (курс), выполнить расчет (пересчитать цену из долларов в рубли) и вывести результат в поле отображения текста (`Label4`).

Чтобы создать процедуру обработки события `Click` на командной кнопке, надо в окне дизайнера формы сделать двойной щелчок мышью на изображении этой кнопки. В результате этого станет доступным окно редактора кода, в которое будет добавлена сформированная средой разработки процедура обработки события (рис. 2.13). Имя процедуры обработки события формирует среда разработки. Оно состоит из двух частей. Первая часть имени идентифицирует объект, для которого создана процедура обработки события, вторая часть — событие. В рассматриваемом примере объект — это командная кнопка `Command1`, а событие — `Click`.

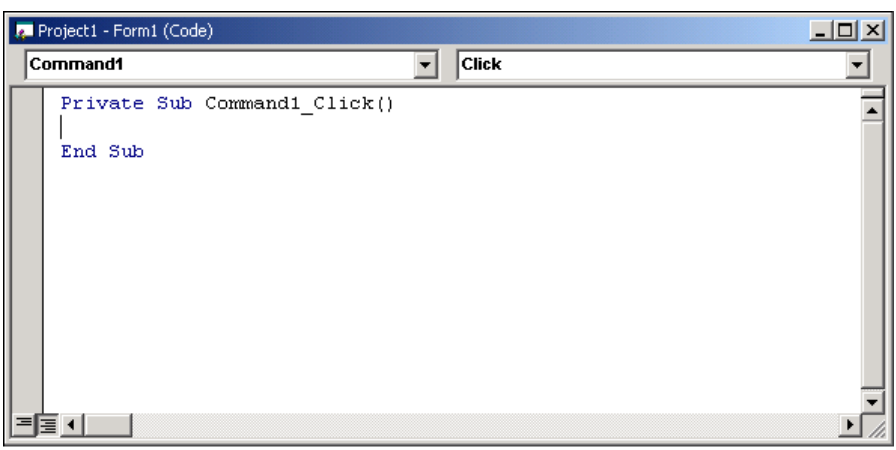


Рис. 2.13. Процедура обработки события

Чтобы создать процедуру обработки другого события, надо активизировать окно редактора кода (выбрать в меню **View** команду **Code**). Затем выбрать объект (рис. 2.14), для которого надо создать процедуру обработки события, и из раскрывающегося списка событий, который находится в верхней части окна редактора кода, справа от списка объектов, выбрать событие.

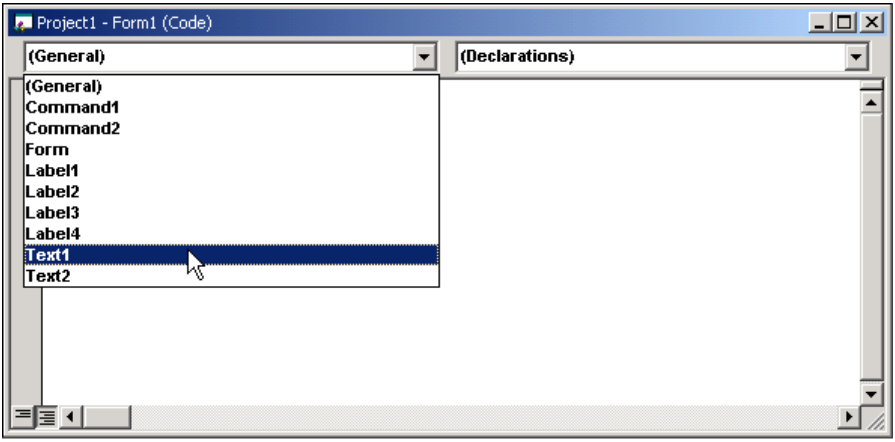


Рис. 2.14. Выбор объекта, для которого надо создать процедуру обработки события

Например, чтобы создать процедуру обработки события `KeyPress` для поля редактирования `Text1`, надо сначала выбрать объект `Text1`, затем — событие `KeyPress` (рис. 2.15).

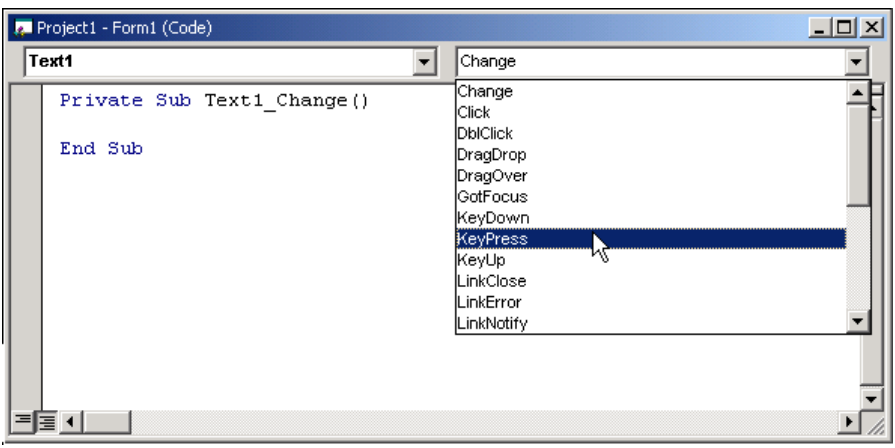


Рис. 2.15. Выбор события, обработка которого должна быть выполнена

В листинге 2.1 приведена процедура обработки события `Click` для командной кнопки **Пересчет**. Обратите внимание на то, как представлена программа. Ее общий вид соответствует тому, как она выглядит в окне редактора кода: ключевые слова выделены полужирным, комментарии — курсивом (выделение выполняет редактор кода).

Листинг 2.1. Обработка события `Click` на кнопке *Пересчет*

```
Private Sub Command1_Click()

    Dim usd As Double   ' цена в долларах
    Dim k As Double     ' курс
    Dim rub As Double   ' цена в рублях

    ' ввод исходных данных
    usd = CDbl(Text1.Text)
    k = CDbl(Text2.Text)

    ' пересчет
    rub = usd * k

    ' вывод результата
    Label4.Caption = Str(usd) + "$ = " + Str(rub) + "руб."

End Sub
```

Процедура `Command1_Click` выполняет пересчет цены из долларов в рубли и выводит результат расчета в поле `Label4`. Исходные данные вводятся из полей редактирования `Text1` и `Text2` путем обращения к свойству `Text`. Доступ к свойству осуществляется путем указания имени объекта (`Text1` или `Text2`) и свойства (`Text`). Имя свойства от имени объекта отделяется точкой. Свойство `Text` (символьного типа) содержит строку символов, которую ввел пользователь. Для правильной работы программы строка должна представлять собой изображение дробного числа, т. е. содержать только цифры и, возможно, запятую (разделитель целой и дробной частей числа). Преобразование строки символов в дробное число выполняет функция `CDbl`, которой в качестве параметра передается значение свойства `Text` — строка символов, находящаяся в поле редактирования. Значение функции `CDbl` — число, изображением которого является строка-параметр.

После того как исходные данные будут помещены в переменные `k` и `usd`, выполняется расчет.

Вычисленное значение цены в рублях выводится в поле `Label4` путем присваивания значения свойству `Caption`. Для преобразования числа в строку символов используется функция `Str`.

В результате нажатия кнопки **Завершить** программа должна завершить работу. Чтобы это произошло, надо закрыть главное окно программы. Делается это путем вызова процедуры `End`. Процедура обработки события `Click` для кнопки **Завершить** приведена в листинге 2.2.

Листинг 2.2. Обработка события `Click` на кнопке *Завершить*

```
Private Sub Command2_Click()
    End ' завершить работу программы
End Sub
```

Редактор кода

Редактор кода выделяет ключевые слова языка программирования (`Sub`, `Function`, `Dim`, `If`, `Else` и др.) полужирным шрифтом, что делает текст программы более выразительным и облегчает восприятие структуры программы.

Помимо ключевых слов редактор кода выделяет цветом комментарии.

Во время работы над программой часто возникает необходимость переключения между окном редактора кода и окном дизайнера формы. Выбрать нужное окно можно при помощи командных кнопок **View Object** и **View Code**, которые находятся в верхней части окна проекта (рис. 2.16).

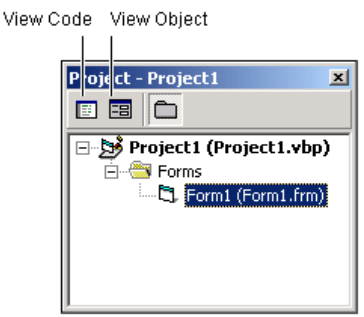


Рис. 2.16. Командные кнопки **View Object** и **View Code**

В процессе набора текста программы редактор кода автоматически выводит справочную информацию о параметрах процедур и функций, о свойствах и методах объектов.

Например, если в окне редактора кода набрать `MsgBox` (имя функции, которая выводит на экран окно сообщения) и открывающую скобку, то на экране

появится окно подсказки, в котором будут перечислены параметры функции `MsgBox` с указанием их типа (рис. 2.17). Один из параметров выделен полужирным шрифтом. Так редактор подсказывает программисту, какой параметр он должен вводить. После набора параметра и запятой в окне подсказки будет выделен следующий параметр. И так до тех пор, пока не будут указаны все параметры.

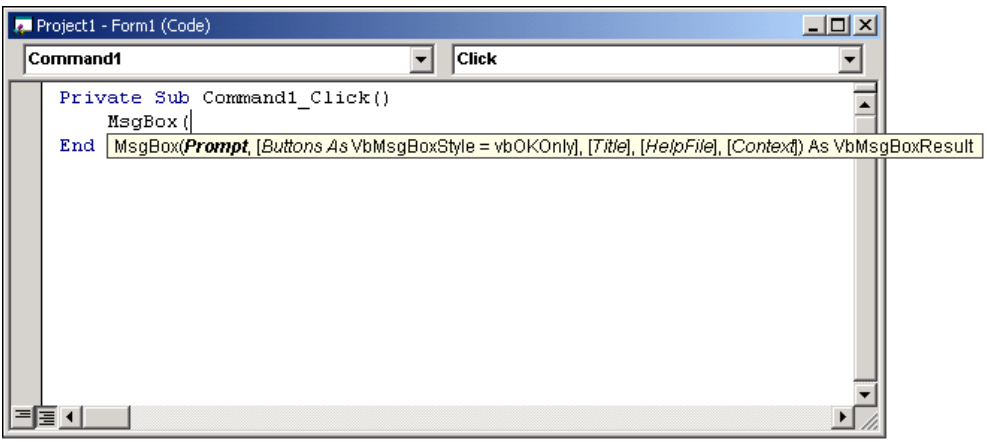


Рис. 2.17. Пример подсказки редактора кода

Для объектов редактор кода выводит список свойств и методов. Как только программист наберет имя объекта (компонента) и точку, на экране появляется список свойств и методов этого объекта (рис. 2.18). Чтобы вставить в текст программы элемент из списка, надо его выбрать (при помощи клавиш перемещения курсора или набрав на клавиатуре несколько первых букв имени нужного свойства или метода) и нажать комбинацию клавиш <Ctrl>+<Enter>. Выбранное свойство или метод будет вставлено в текст программы.

Система подсказок существенно облегчает процесс подготовки текста программы, избавляет от рутины. Следует обратить внимание: если во время набора программы подсказка не появилась, то это значит, что программист допустил ошибку, скорее всего, неверно набрал имя процедуры или функции.

В процессе набора текста программы Visual Basic проверяет программу на наличие *грубых синтаксических* ошибок. Процесс контроля активизируется сразу после набора очередной строки текста программы. Если в набранной строке есть ошибка, то компилятор выделяет цветом эту строку и выводит сообщение. Окончательная проверка программы на отсутствие синтаксических ошибок выполняется во время ее работы.

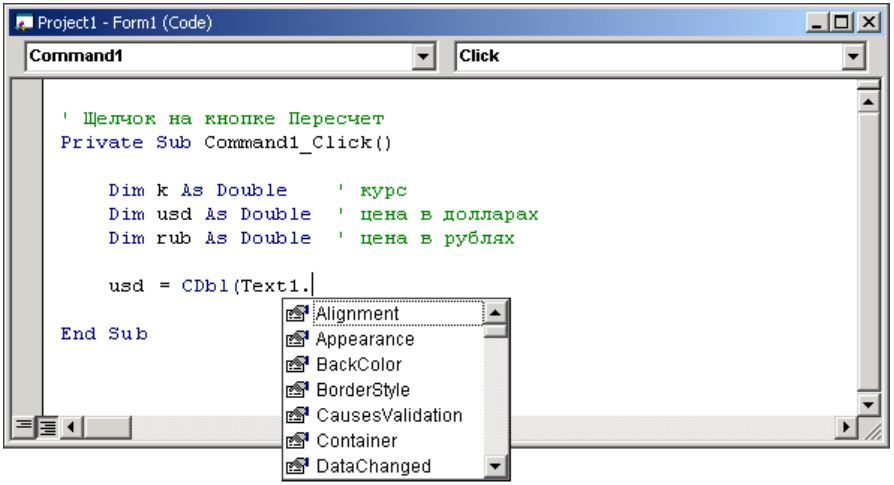


Рис. 2.18. Редактор кода автоматически выводит список свойств и методов объекта (компонента)

Запись инструкций

Программа на языке Visual Basic представляет собой последовательность инструкций, каждую из которых, за исключением инструкций `If`, `Select`, `For`, `While` и некоторых других, обычно записывают в отдельной строке. Например:

```
Dim k As Double
Dim i As Integer
k = CDbl(Text1.Text)
usd = CDbl(Text2.Text)
rub = k * usd
```

Несколько инструкций можно записать в одной строке, отделив одну инструкцию от другой двоеточием:

```
Dim k As Double : Dim usd As Double
k = CDbl(Text1.Text) : usd = Cdbl(Text2.Text) : rub = k * usd
```

Однако лучше так не делать — существенно ухудшается восприятие программы.

Длинные инструкции, например инструкции вызова функций или процедур, у которых много параметров, можно записать в несколько строк. Если инструкцию нужно записать в несколько строк, то перед тем как нажать клавишу <Enter>, для того чтобы *продолжить* набор текста в следующей строке,

в конец текущей строки надо поместить символ "подчеркивание", отделив этот символ от последнего набранного символа пробелом. Например:

```
MsgBox("Надо задать цену и курс.", _
       "Конвертер", vbInformation + vbOk)
```

Инструкции `If`, `Select`, `For` и некоторые другие необходимо записывать в несколько строк, причем именно так этого требует синтаксис Visual Basic. Например, слово `Then` инструкции `If` должно следовать сразу за условием, а не в следующей строке (если в процессе набора программы нажать клавишу <Enter> сразу после того, как будет набрано условие, Visual Basic выведет сообщение об ошибке).

В процессе набора текста программы рекомендуется следовать правилам хорошего стиля программирования, что предполагает использование отступов (при записи инструкций выбора и циклов), комментариев, а также пустых строк.

> ***ЗАМЕЧАНИЕ***
>
> Формально отступы, пустые строки и комментарии в тексте программы не нужны, но их наличие существенно облегчает восприятие программы, делает более понятной ее структуру.

Далее приведен пример записи инструкции `Select` с использованием отступов.

```
Select Case KeyAscii
    Case 8, 48 To 57 ' <Backspace> и цифры

    Case 44 ' запятая
        If InStr(1, Text1.Text, ",") <> 0 Then
            KeyAscii = 0
        End If

    Case 46 ' точка
        KeyAscii = 44 ' заменим точку на запятую
        If InStr(1, Text1.Text, ",") <> 0 Then
            KeyAscii = 0
        End If

    Case 13 ' клавиша <Enter>
        Text2.SetFocus

    Case Else
        KeyAscii = 0
End Select
```

Следует обратить внимание на то, что слова `Case` записаны с отступом относительно слова `Select`. Инструкции, которые должны быть выполнены в случае совпадения значения переменной-селектора `KeyAscii` со значением одной из констант, указанной после слова `Case`, также смещены, но уже относительно слова `Case`. Также обратите внимание, что слово `End`, отмечающее конец инструкции `If`, находится строго под словом `If`, а инструкции, которые должны быть выполнены в случае, если условие истинно или ложно, записаны одна под другой и смещены относительно `If`.

Приведенную выше инструкцию `Select` можно записать и так:

```
Select Case KeyAscii
Case 8, 48 To 57 ' <Backspace> и цифры
Case 44 ' запятая
If InStr(1, Text1.Text, ",") <> 0 Then
KeyAscii = 0
End If
Case 46 ' точка
KeyAscii = 44
If InStr(1, Text1.Text, ",") <> 0 Then
KeyAscii = 0
End If
Case 13 ' клавиша <Enter>
Text2.SetFocus
Case Else
KeyAscii = 0
End Select
```

Однако очевидно, что понять, как она работает (что делает), значительно труднее, чем при первом варианте ее записи.

Для облегчения понимания логики работы программы в текст программы рекомендуется включать поясняющий текст — комментарии. В Visual Basic *комментарий* — это текст, который следует за одинарной кавычкой до конца строки. Комментировать надо назначение основных переменных, действия, выполняемые процедурами и функциями, ключевые операции и точки программы.

Справочная информация

Во время работы над программой можно получить справку, например, о конструкции языка программирования, процедуре или функции. Для этого надо в окне редактора кода набрать слово (инструкцию языка программирования,

имя процедуры или функции и т. д.), о котором надо получить справку, и нажать клавишу <F1>.

Получить доступ к справочной информации можно, выбрав в меню **Help** команду **Contents** или **Index**. Команда **Contents** обеспечивает доступ к вкладке **Содержание**, а команда **Index** — к вкладке **Указатель** окна справочной системы. На вкладке **Содержание** надо выбрать раздел **Visual Basic Documentation** и в этом разделе — нужный подраздел. Вкладка **Указатель** обеспечивает доступ к нужному разделу справочной информации по ключевому слову. Как правило, в качестве ключевого слова используют первые несколько букв имени функции, процедуры, свойства или метода (рис. 2.19).

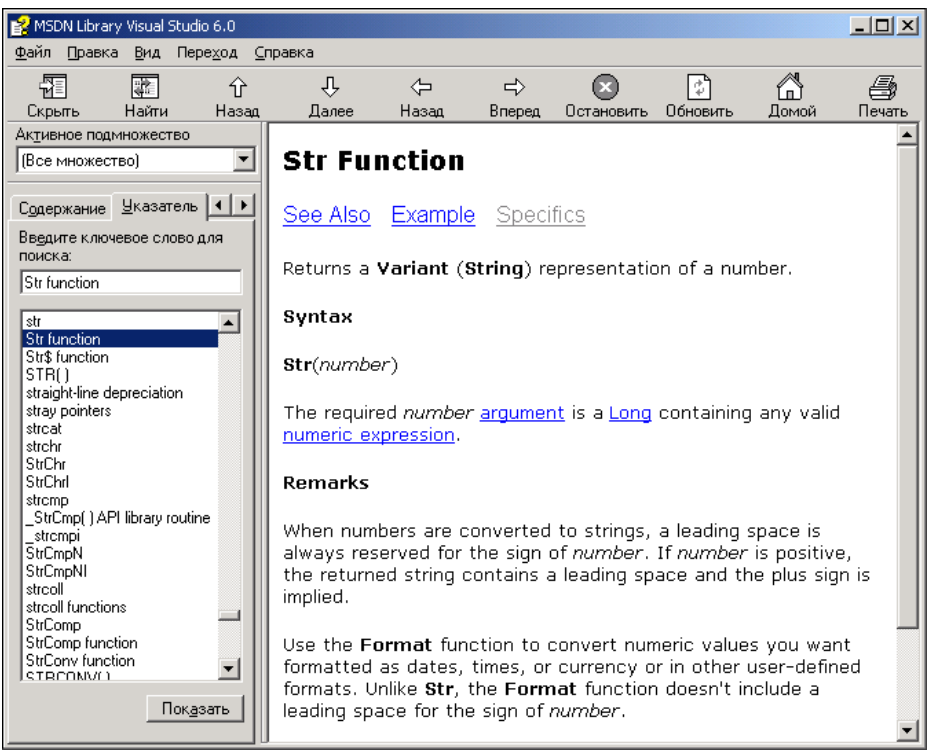


Рис. 2.19. Поиск справочной информации по ключевому слову

Сохранение проекта

Проект Visual Basic — это совокупность файлов (форм, модулей и других компонентов), которые в совокупности и образуют приложение (программу) и которые компилятор использует для создания исполняемого (EXE) файла.

В простейшем случае проект образуют файл описания проекта (VBP) и файл формы (FRM).

Перед тем как сохранить проект, рекомендуется задать имя проекта (это имя отображается в заголовке главного окна, окна дизайнера формы, проекта и др.). Чтобы задать имя проекта, надо в окне **Project** сделать щелчок на названии проекта и в окне **Properties** изменить значение свойства `Name` (рис. 2.20).

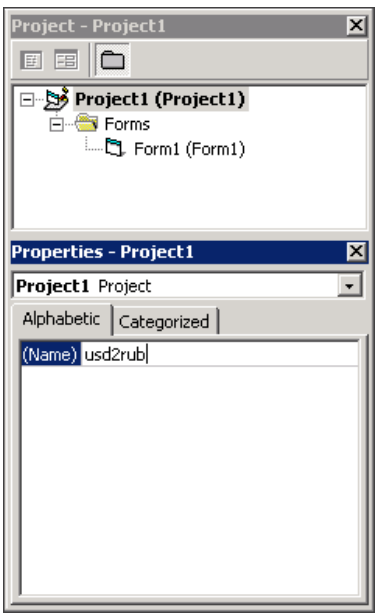


Рис. 2.20. Перед тем как сохранить проект, надо задать его имя

Для того чтобы сохранить проект (программу, над которой в данный момент работает программист), надо в меню **File** выбрать команду **Save Project**. Если проект еще ни разу не был сохранен, то на экране появляется окно **Save File As**, в котором надо выбрать папку, предназначенную для проектов Visual Basic, создать в этой папке новую папку для сохраняемого проекта, открыть ее и в поле **Имя файла** ввести имя файла модуля формы (рис. 2.21). Затем в следующем окне **Save Project As** надо задать имя файла проекта (рис. 2.22). Следует обратить внимание на то, что имя проекта определяет имя EXE-файла, который будет создан во время компиляции программы.

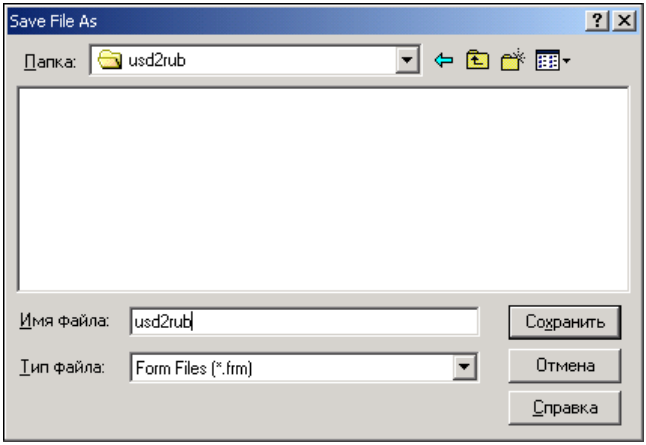


Рис. 2.21. Сохранение модуля формы

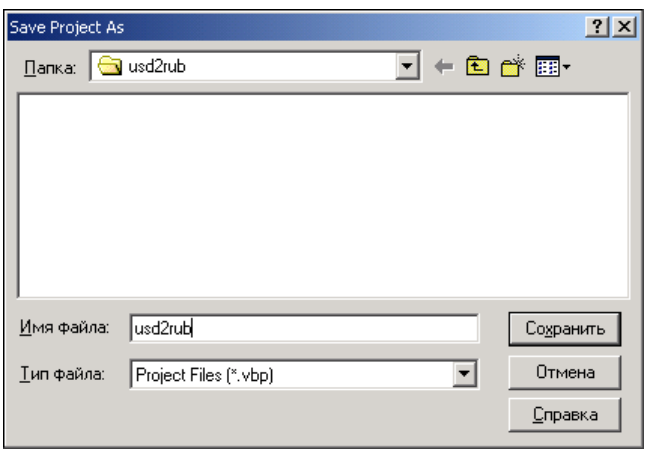


Рис. 2.22. Сохранение файла проекта

Запуск программы

После того как программа будет сохранена, можно выполнить ее запуск. Запустить программу можно несколькими способами. Можно в меню **Run** выбрать команду **Start**, нажать клавишу <F5> или сделать щелчок на командной кнопке **Start** (рис. 2.23), которая находится на стандартной панели инструментов (на этой же панели находится кнопка **End**, щелчок на которой останавливает запущенную программу).

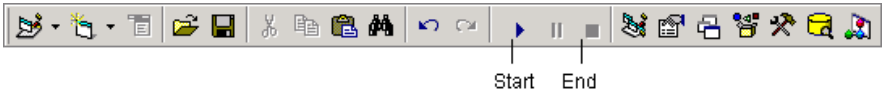


Рис. 2.23. Кнопки управления процессом выполнения программы

Если в процессе выполнения программы, запущенной из среды разработки, обнаруживается ошибка, то на экране появляется окно сообщения об этом (рис. 2.24). Чтобы исправить ошибку, надо завершить выполнение программы: сначала закрыть окно сообщения об ошибке (сделать щелчок на кнопке **OK**), затем — щелкнуть на кнопке **End**.

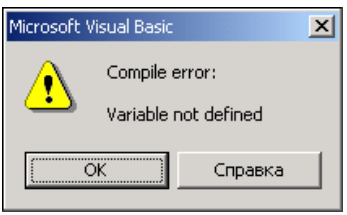


Рис. 2.24. Пример сообщения об ошибке

Следует обратить внимание, что окончательная проверка программы происходит во время ее выполнения и, следовательно, возможна ситуация, когда в программе (в какой-либо ее части, которая еще ни разу не работала) есть ошибки, хотя программа работает (до тех пор, пока не будет активизирован фрагмент, в котором есть ошибка). Чтобы избежать подобных неприятностей, надо перед запуском программы выполнить ее компиляцию — выбрать в меню **Run** команду **Start With Full Compile**. При выборе этой команды программа будет запущена только в том случае, если в ней нет синтаксических ошибок.

Также следует обратить внимание, что по умолчанию в Visual Basic переменные можно не объявлять. Эта особенность языка является источником ошибок, которые иногда довольно трудно обнаружить. Поэтому настоятельно рекомендуется в начало текста программы поместить директиву `Option Explicit`, которая устанавливает, что все переменные программы должны быть объявлены явно в инструкции `Dim`.

Вместо добавления в начало программы директивы `Option Explicit` можно на вкладке **Editor** окна **Options** (рис. 2.25) установить флажок **Require Variable Declaration** (Требовать объявления переменных). Окно **Options** становится доступным в результате выбора команды **Tools ▸ Options**.

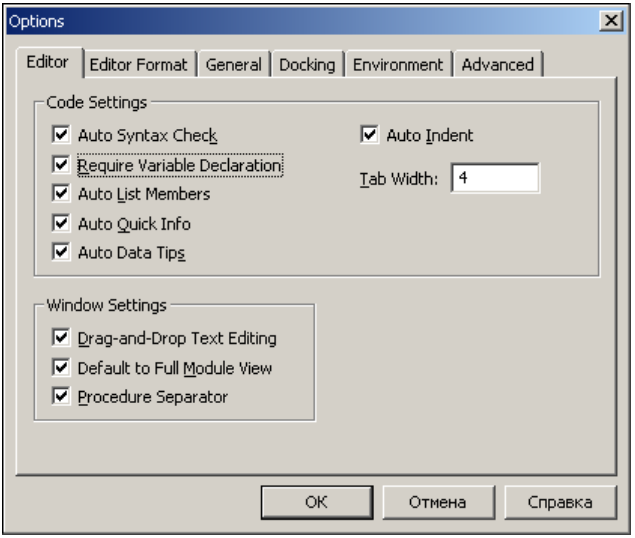


Рис. 2.25. Установите во включенное состояние флажок **Require Variable Declaration**

Исключения

Даже если в программе нет синтаксических ошибок, во время ее *работы* ошибки все-таки возможны (такие ошибки называют *ошибками времени выполнения*, или *исключениями*). Например, если в поле **Курс** программы "Конвертер" ввести строку `24.5` и сделать щелчок на кнопке **Пересчет**, то на экране появится окно с сообщением "Run-time error '13': Type mismatch" (рис. 2.26).

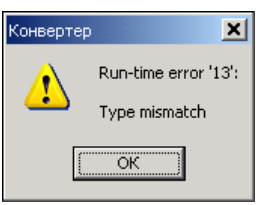


Рис. 2.26. Пример сообщения об ошибке

Причина возникновения описанной ошибки в следующем. Преобразование строки, введенной в поле редактирования, в число выполняет функция `CDbl`. Эта функция работает правильно, если ее параметром является *правильное* строковое представление дробного числа, что предполагает при стандартной

(для России) настройке операционной системы Windows использование в качестве десятичного разделителя запятой. В рассматриваемом примере строка `24.5` не является изображением дробного числа, поэтому и возникает ошибка.

Если программа запущена из среды разработки, то при возникновении ошибки выполнение программы приостанавливается, и на экране также появляется окно с сообщением об ошибке (рис. 2.27). Щелчок на кнопке **End** завершает выполнение программы, а щелчок на кнопке **Debug** переводит ее в режим отладки, и в окне редактора кода выделяется инструкция, при выполнении которой возникла ошибка.

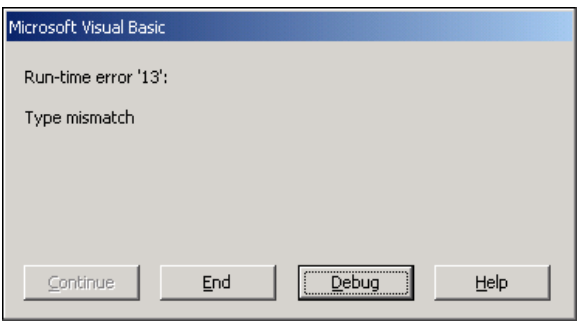


Рис. 2.27. Пример сообщения о возникновении исключения

Обработка исключений

Простейшую обработку ошибок выполняет автоматически добавляемый в программу код, который обеспечивает вывод сообщения об ошибке и завершение выполнения программы. Вместе с тем программист может поместить в программу код, который выполнит обработку исключения.

Инструкция обработки исключения в общем виде выглядит так:

```
On Error Goto M
```

Где *M* — метка (идентификатор) инструкции, которая должна быть выполнена при возникновении ошибки.

В качестве примера использования инструкции `On Error` в листинге 2.3 приведена процедура обработки события `Click` на кнопке **Пересчет** программы "Конвертер", в которую добавлены инструкции, обеспечивающие обработку наиболее вероятной ошибки — ошибки преобразования строки в дробное число. При возникновении ошибки на экране отображается окно с сообщением об ошибке (рис. 2.28).

Листинг 2.3. Щелчок на кнопке *Пересчет* (обработка ошибки)

```
' щелчок на кнопке Пересчет
Private Sub Command1_Click()

    Dim usd As Double   ' цена в долларах
    Dim k As Double     ' курс
    Dim rub As Double   ' цена в рублях

    On Error GoTo e1 ' если ошибка, то перейти к метке e1

   ' ввод исходных данных
    usd = CDbl(Text1.Text)
    k = CDbl(Text2.Text)

    ' вычисление
    rub = usd * k

    ' вывод результата
    Label4.Caption = Str(usd) + "$ = " + Str(rub) + " руб."

    Exit Sub ' завершение работы программы

e1: ' обработка ошибки
    If Len(Text1.Text) = 0 Or Len(Text2.Text) = 0 Then
        MsgBox "Надо ввести данные в оба поля", vbExclamation
    Else
        MsgBox "Ошибка ввода исходных данных." + vbCr + _
               "При вводе дробных чисел используйте запятую", _
               vbExclamation
    End If
End Sub
```

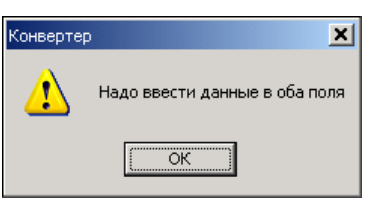


Рис. 2.28. Пример сообщения об ошибке

В приведенной процедуре обработки события `Click` для вывода сообщения об ошибке используется процедура `MsgBox`. Инструкция вызова в общем виде выглядит так:

MsgBox *Сообщение*, *Тип*, *Заголовок*

Здесь:

- *Сообщение* — текст сообщения;
- *Тип* — тип сообщения. Сообщение может быть информационным (`vbInformation`), предупреждающим (`vbWarning`) или сообщением о критической ошибке (`vbCritical`). Каждому типу сообщения соответствует свой значок (табл. 2.10);
- *Заголовок* — текст в заголовке окна сообщения.

Таблица 2.10. *Тип сообщения*

Тип сообщения	Значок
`vbExclamation`	
`vbCritical`	
`vbInformation`	

Обратите внимание, что в приведенном примере в заголовке окна сообщения отображается *название* программы — Конвертер. Чтобы в заголовке окна сообщения отображалось именно название программы, его надо задать — ввести в поле **Application Title** вкладки **Make** окна **Project Properties** (окно становится доступным в результате выбора в меню **Project** команды **Properties**).

Создание EXE-файла

Для того чтобы иметь возможность запустить программу из операционной системы, а не только из среды разработки Visual Basic, надо создать выполняемый файл.

Чтобы создать выполняемый (EXE) файл, надо в меню **File** выбрать команду **Make** и в появившемся окне **Make Project** (рис. 2.29) сделать щелчок на кнопке **OK**.

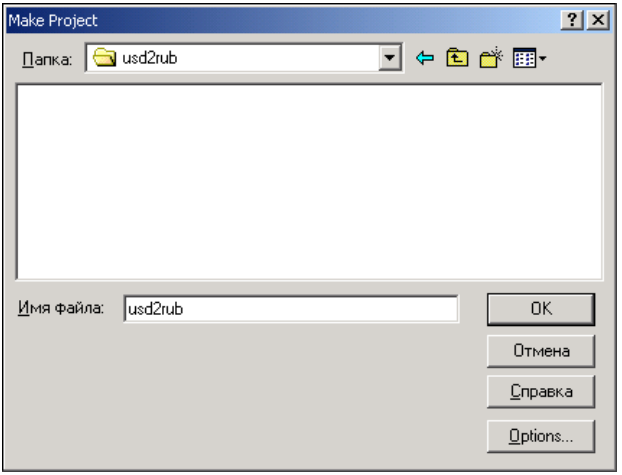


Рис. 2.29. Щелчок на кнопке **OK** активизирует процесс создания EXE-файла

Завершение работы

Для того чтобы завершить работу с Visual Basic, надо в меню **File** выбрать команду **Exit**. Если с момента последнего сохранения проекта в программу были внесены какие-либо изменения, то на экране появится окно с запросом о необходимости их сохранения (рис. 2.30).

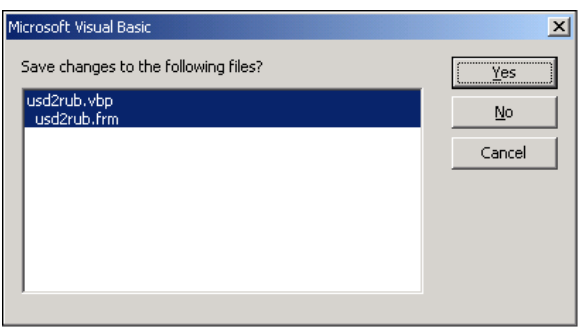


Рис. 2.30. Запрос о необходимости сохранения изменений

Внесение изменений

После нескольких запусков программы "Конвертер" возникает желание ее усовершенствовать. Например, сделать так, чтобы в поля **Цена** и **Курс** можно было вводить только числа, и чтобы после ввода цены (в результате нажатия

клавиши <Enter>) курсор переходил в поле **Курс**, а при нажатии клавиши <Enter> в поле **Курс** фокус переходил бы на кнопку **Пересчет**.

Чтобы внести изменения в программу, нужно запустить Visual Basic и открыть соответствующий проект. Для этого надо в меню **File** выбрать команду **Open Project**. В результате становится доступным окно **Open Project**. Вкладка **Existing** этого окна позволяет открыть проект обычным образом — найти нужную папку, раскрыть ее и открыть файл проекта. На вкладке **Recent** перечислены проекты, над которыми программист работал в последнее время. Чтобы внести изменения в программу "Конвертер", надо на вкладке **Recent** выбрать проект **usd2rub** (рис. 2.31).

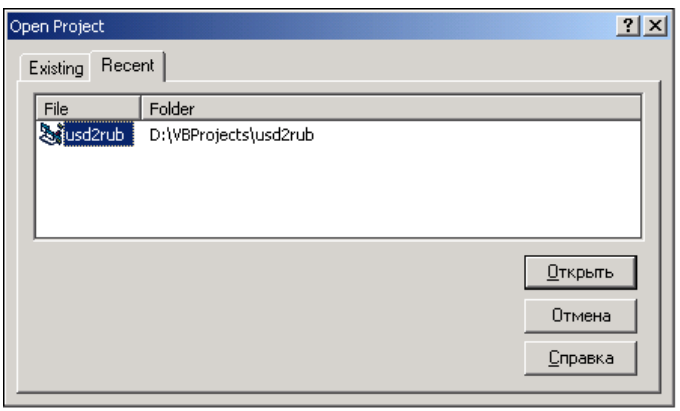


Рис. 2.31. На вкладке **Recent** перечислены проекты, над которыми программист работал в последнее время

В листинге 2.4 приведен модуль главной формы программы "Конвертер", в который добавлены процедуры обработки события `KeyPress` для компонентов `Text1` и `Text2`.

Листинг 2.4. Модуль главной формы программы "Конвертер"

```
Option Explicit

' щелчок на кнопке Пересчет
Private Sub Command1_Click()

    Dim usd As Double   ' цена в долларах
    Dim k As Double     ' курс
```

```
    Dim rub As Double  ' цена в рублях

    ' ввод исходных данных
    usd = CDbl(Text1.Text)
    k = CDbl(Text2.Text)

    ' вычисление
    rub = usd * k

    ' вывод результата
    Label4.Caption = Str(usd) + "$ = " + Str(rub) + " руб."

End Sub

' щелчок на кнопке Завершить
Private Sub Command2_Click()
    End
End Sub

' нажатие клавиши в поле Цена
' (обработка события KeyPress в поле компонента Text1)
Private Sub Text1_KeyPress(KeyAscii As Integer)
    '
    ' Параметр KeyAscii содержит код нажатой клавиши.
    ' Если в процедуре параметру KeyAscii присвоить значение 0, то символ,
    ' соответствующий нажатой клавише, в поле редактирования
    ' не появится, и у пользователя будет впечатление, что программа
    ' не реагирует на нажатие клавиши.
    ' В данном случае допустимы символы:
    ' цифры (код клавиш от 48 до 57),
    ' запятая (код 44),
    ' а также клавиша <Backspace>.
    ' Точку процедура меняет на запятую.

    Select Case KeyAscii
        Case 8, 48 To 57
            ' <Backspace> и цифры

        Case 44 ' запятая
```

```
            If InStr(1, Text1.Text, ",") <> 0 Then
                KeyAscii = 0
            End If

        Case 46 ' точка
            KeyAscii = 44
            If InStr(1, Text1.Text, ",") <> 0 Then
                KeyAscii = 0
             End If

        Case 13 ' клавиша <Enter>
            Text2.SetFocus

        Case Else
            KeyAscii = 0
    End Select
End Sub

' нажатие клавиши в поле Курс
' (обработка события KeyPress в поле компонента Text2)
Private Sub Text2_KeyPress(KeyAscii As Integer)
    Select Case KeyAscii
        Case 8, 48 To 57
            ' <Backspace> и цифры

        Case 44 ' запятая
            If InStr(1, Text2.Text, ",") <> 0 Then
                KeyAscii = 0
            End If

        Case 46 ' точка
            KeyAscii = 44
            If InStr(1, Text2.Text, ",") <> 0 Then
                KeyAscii = 0
            End If

        Case 13 ' клавиша <Enter>
```

```
            Command1.SetFocus

        Case Else
            ' не отображать символ
            KeyAscii = 0

    End Select
End Sub
```

Процедура обработки события `KeyPress` для компонента `Text1` получает в качестве параметра (`KeyAscii`) код клавиши, нажатие которой вызвало событие. Если в процедуре параметру `KeyAscii` присвоить значение `0`, то символ, соответствующий нажатой клавише, в поле редактирования не появится, и у пользователя будет впечатление, что программа не реагирует на нажатие клавиши. Таким образом можно обеспечить фильтрацию символов. В рассматриваемой программе поле `Text1` должно содержать дробное число, т. е. допустимыми символами являются цифры и запятая. Если нажата допустимая клавиша, то процедура обработки события `KeyPress` "ничего не делает" и соответствующий символ появляется в поле редактирования. Если нажата недопустимая клавиша, то параметру `KeyAscii` присваивается значение `0`. В случае нажатия клавиши <Enter> процедура путем вызова метода `SetFocus` компонента `Text2` переводит курсор в поле редактирования `Text2`. Если пользователь нажимает клавишу с запятой, то процедура проверяет, есть ли в поле редактирования запятая (делает это функция `InStr`, которая возвращает номер позиции искомого символа в строке или ноль, если символа в строке нет). Если в поле редактирования запятая уже есть, то параметру `KeyAscii` присваивается значение `0`, и вторая запятая не появляется. Если нажата клавиша с точкой, то программа заменяет точку запятой. Таким образом, пользователь вводит точку, а в поле редактирования появляется запятая (если она еще не введена).

Процедура обработки события `KeyPress` для компонента `Text2` работает аналогичным образом.

После внесения изменений проект следует откомпилировать (команда **Run ▸ Start With Full Compile**), создать выполняемый файл (команда **File ▸ Make**) и сохранить (команда **File ▸ Save Project**).

Значок приложения

Свойство `Icon` стартовой формы определяет значок, который отображается в заголовке окна программы. Этот же значок по умолчанию отображается

в панели задач во время работы программы, а также изображает программу (EXE-файл) в папке. По умолчанию это стандартный значок (рис. 2.32).

Рис. 2.32. Стандартный значок Visual Basic

Программист может изменить значок формы. Для этого надо в строке свойства `Icon` сделать щелчок на кнопке с тремя точками и в появившемся окне **Load Icon** (рис. 2.33) выбрать файл значка (ICO-файл).

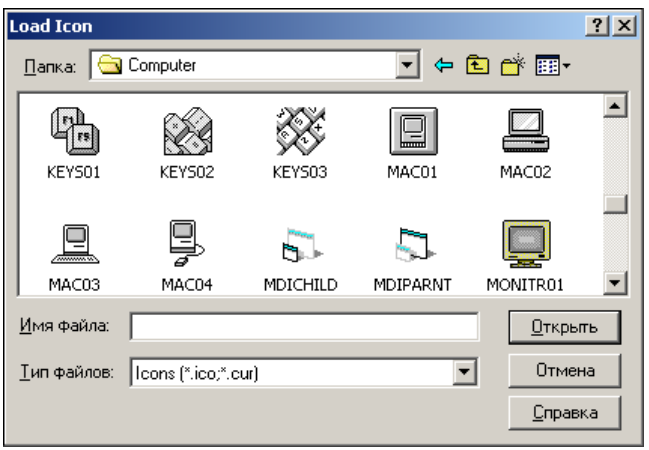


Рис. 2.33. Выбор значка для формы

Программист может создать для своего приложения уникальный значок. Сделать это можно при помощи утилиты Image Editor (Imageedit.exe), которая входит в состав Visual Basic и находится в папке Microsoft Visual Studio\Common\Tools\Vb\Imagedit.

Перед тем как описать процесс создания файла значка, необходимо сказать несколько слов о структуре ICO-файла.

В простейшем случае в ICO-файле может находиться один-единственный значок. Размер значка может быть 16×16 или 32×32. Значок 16×16 отображается в заголовке окна без искажения ("один к одному"). В папке, в режиме отображения **Крупные значки**, значок 16×16 масштабируется до размера 32×32 и поэтому выглядит "размытым". Значок 32×32, наоборот, в папке

выглядит хорошо, а в заголовке окна — нечетко, т. к. масштабируется до размера 16×16. Таким образом, чтобы значок отображался четко и в заголовке окна, и в папке, в ICO-файле должны быть две картинки: 16×16 и 32×32.

Значки могут отличаться и глубиной цвета (количеством цветов палитры). До недавнего времени стандартной считалась 16-цветная палитра. Сейчас используется палитра из 256 цветов. К сожалению, Image Editor работает с 16-цветной палитрой.

Процесс создания ICO-файла рассмотрим на примере — создадим файл, который будет содержать два значка: 16×16 и 32×32.

Сначала надо запустить Image Editor и в меню **File** выбрать команду **New**. Затем в окне **Resource Type** следует выбрать **Icon** и щелкнуть на кнопке **OK** (рис. 2.34). После этого в появившемся окне **New Icon Image** (рис. 2.35) надо выбрать **EGA/VGA 16-Color 32x32**. В результате этих действий в ICO-файл, над которым идет работа, будет добавлен новый *ресурс* — значок, и станет доступным окно графического редактора (рис. 2.36).

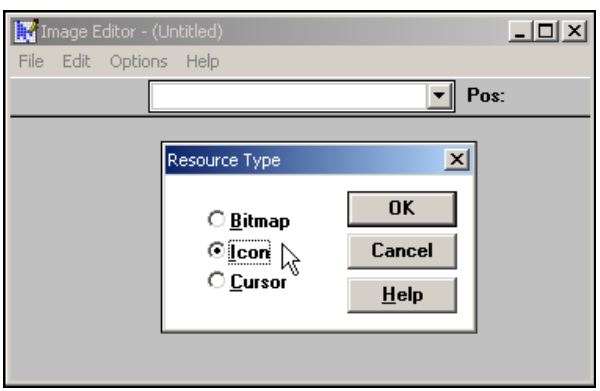


Рис. 2.34. Начало работы над новым значком

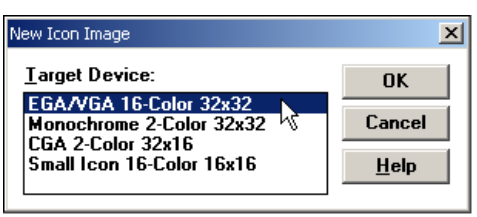


Рис. 2.35. Выбор характеристик значка

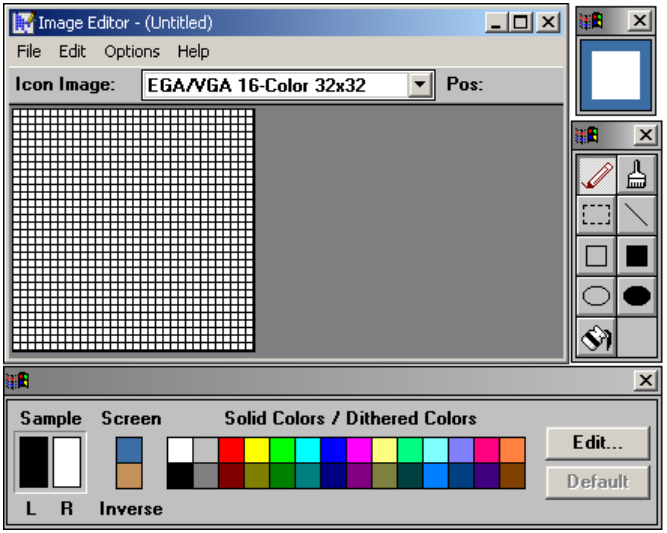


Рис. 2.36. Графический редактор Image Editor

Процесс рисования в Image Editor обычный и практически ничем не отличается от рисования в обычном графическом редакторе. Вместе с тем следует обратить внимание, что помимо обычных цветов в палитре есть цвет "экран" (**Screen**). Точки картинки, закрашенные этим цветом, при отображении значка окрашиваются в цвет точек поверхности, на которой значок отображается (т. е. фактически это "прозрачный" цвет).

После того как картинка 32×32 будет готова, можно приступить к работе над картинкой размера 16×16. Для этого надо сначала в меню **Edit** выбрать команду **New Image**, затем в появившемся окне **New Icon Image** — **Small Icon 16-Color 16x16**.

Чтобы сохранить созданные картинки, надо в меню **File** выбрать команду **Save** и задать имя файла. Следует обратить внимание, что обе картинки будут сохранены в одном файле.

Окончательная настройка приложения

После того как программа будет отлажена, можно выполнить ее окончательную настройку — задать название и значок, который будет изображать программу (исполняемый файл приложения) в папке или на рабочем столе. Также в выполняемый файл можно поместить, например, информацию о разработчике программы (авторских правах) и ее версии. Эта информация отображается на вкладках окна **Свойства** (рис. 2.37), которое появляется

в результате щелчка правой кнопкой мыши на значке EXE-файла и выбора в появившемся контекстном меню команды **Свойства**.

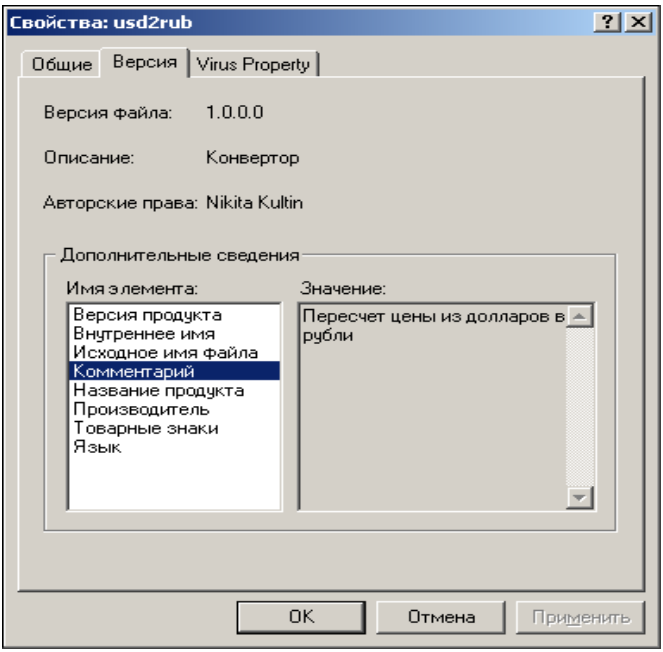


Рис. 2.37. Подробная информация о программе отображается на вкладках окна **Свойства**

Чтобы выполнить настройку приложения, надо в меню **Project** выбрать команду **Properties**. В поле **Application Title** вкладки **Make** надо ввести название программы (рис. 2.38). На этой же вкладке можно задать значок, который будет использоваться для изображения файла программы в папках компьютера (для этого надо раскрыть список **Icon** и выбрать форму, значок которой будет использоваться в качестве значка приложения). Чтобы задать информацию о программе (краткое описание, имя разработчика и т. д.), надо в группе **Version Information**, в списке **Type**, выбрать характеристику (свойство) программы и в поле **Value** ввести текст. Свойства программы приведены в табл. 2.11.

После того как будет выполнена настройка приложения, надо активизировать процесс создания выполняемого файла — выбрать команду **File ▸ Make**.

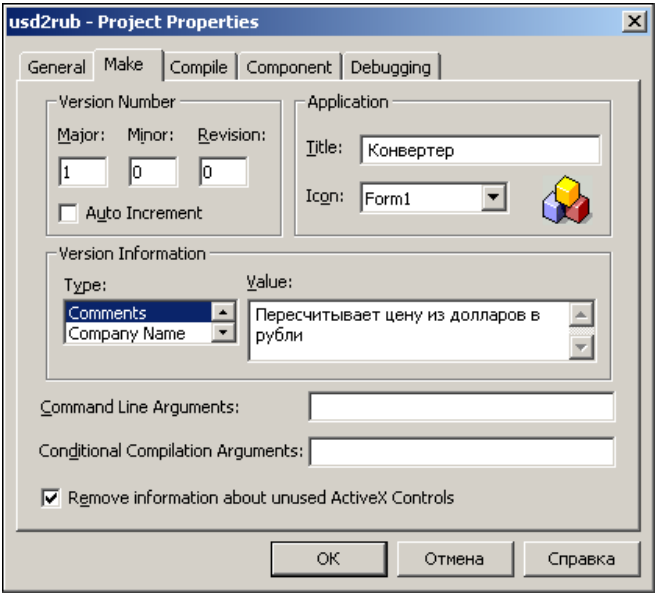


Рис. 2.38. Окончательная настройка приложения

Таблица 2.11. *Свойства программы (EXE-файла)*

Свойство	Содержание
`Comments`	Краткая информация о программе (назначение программы)
`Company Name`	Информация о разработчике
`File Description`	Описание файла (для программы — "Приложение")
`Legal Copyright`	Информация об авторских правах на программу
`Product Name`	Название программы

Установка приложения на другой компьютер

Приложение, созданное в Visual Basic, можно перенести на другой компьютер, например, с помощью флэш-накопителя. Вместе с тем следует понимать, что программе, созданной в Visual Basic, для работы необходимы динамические библиотеки (список библиотек, используемых программой, отображается в окне **References**, которое становится доступным в результате выбора в меню **Project** команды **References**). На компьютер программиста они уста-

навливаются вместе с Visual Basic, а на компьютере пользователя их может и не быть. Поэтому возможно, что у пользователя из-за отсутствия необходимых динамических библиотек программа работать не будет. Однако это не значит, что для успешного запуска программы на компьютере пользователя должен быть установлен Visual Basic — достаточно установить только библиотеки, используемые программой.

Замечание

С **http://microsoft.com/downloads** можно загрузить файл vbrun60sp5.exe, представляющий собой самораспаковывающийся архив библиотек, обеспечивающих работу приложений, созданных в Visual Basic 6.0.

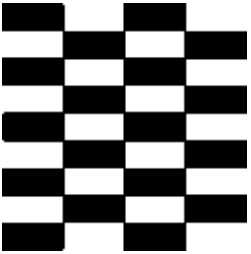
Программу (EXE-файл, файлы данных и справки), а также необходимые для ее работы динамические библиотеки (или файл vbrun60sp5.exe) можно поместить на промежуточный носитель (CD или "флэшку") и в таком виде передать пользователю, чтобы он сам скопировал файлы на диск своего компьютера и установил библиотеки. Но лучше все-таки создать программу-установщик, которая в автоматическом режиме выполнит установку приложения и динамических библиотек на компьютер пользователя.

Создать установщик можно, например, при помощи утилиты Package & Deployment Wizard, которая входит в состав Visual Basic, или IExpress, поставляемой вместе с Microsoft Windows.

Контрольные вопросы

1. Что такое событие? Приведите примеры.
2. Что такое компонент? Приведите примеры.
3. Что такое свойство? Приведите примеры свойств формы.
4. Значение какого свойства определяет текст, отображаемый в заголовке формы?
5. Какой компонент следует использовать для отображения текста на поверхности формы?
6. Какое событие происходит в результате щелчка кнопкой мыши на командной кнопке?
7. Какое свойство определяет текст, отображаемый в поле компонента `Label`?
8. Что надо сделать, чтобы вывести результат работы программы в поле компонента `Label`?
9. Какой компонент предназначен для ввода исходных данных?
10. Какое свойство компонента `TextBox` содержит (хранит) текст, введенный пользователем в поле редактирования?
11. Для чего предназначена функция `Val`?

12. Какой символ следует использовать в качестве десятичного разделителя при вводе дробных чисел, если в программе используется функция `Val`?
13. Какой символ следует использовать в качестве десятичного разделителя при вводе дробных чисел, если в программе используется функция `CDec`?
14. Для чего предназначена функция `Str`?
15. Для чего предназначена функция `CDec`?
16. В чем состоит различие между функциями `Val` и `CDec`?
17. Во время работы программы, после ввода в поле редактирования дробного числа на экране появилось сообщение об ошибке "Run-time error '13': Type mismatch". Назовите наиболее вероятную причину возникновения ошибки.
18. Как создать выполняемый (EXE) файл программы?



Глава 3

Язык программирования Visual Basic

Алгоритм и программа

Программа, работающая на компьютере, нередко отождествляется с самим компьютером, т. к. пользователь (человек, использующий программу) "вводит в компьютер" исходные данные с клавиатуры, а "компьютер выдает результат". На самом деле преобразование исходных данных, введенных с клавиатуры, в результат, отображаемый на экране монитора, выполняет процессор, и делает он это в соответствии с программой — последовательностью команд (инструкций), составленной программистом. Таким образом, чтобы компьютер выполнил некоторую работу, необходимо сначала разработать *алгоритм* решения задачи и затем представить его в виде последовательности команд на языке программирования, т. е. "написать программу". Следует обратить внимание, выражение "написать программу" отражает только один из этапов создания компьютерной программы, когда программист действительно пишет команды на бумаге или в редакторе текста среды разработки.

Этапы разработки программы

Программирование — это процесс создания программы, который может быть представлен как последовательность таких шагов (этапов):

- ❒ определение требований к программе (постановка задачи);
- ❒ разработка (составление) алгоритма решения поставленной задачи;
- ❒ написание команд (кодирование);
- ❒ отладка;
- ❒ тестирование.

Определение требований к программе — один из важнейших этапов. На этом этапе подробно описывается исходная информация и формулируются требования к результату. Например, требование к программе вычисления сопротивления электрической цепи, состоящей из двух сопротивлений, которые могут быть соединены последовательно или параллельно, может быть сформулировано так:

- исходной информацией для программы являются величины сопротивлений, выраженные в омах, и способ соединения сопротивлений;
- если сопротивления соединены последовательно, то величина сопротивления цепи вычисляется по формуле:

$$R = R_1 + R_2;$$

- если сопротивления соединены параллельно, то величина сопротивления цепи вычисляется по формуле:

$$R = \frac{R_1 \times R_2}{R_1 + R_2};$$

- результат расчета (сопротивление цепи) должен быть выведен в омах, если величина сопротивления цепи меньше 1000 Ом, или в килоомах, если величина сопротивления цепи больше 1000 Ом.

Следует обратить внимание также на то, что здесь же, на этапе определения требований к программе, часто формулируются (в виде эскиза) требования к виду диалоговых окон.

На этапе *разработки алгоритма* необходимо определить последовательность элементарных действий (шагов), которые надо выполнить для достижения поставленной цели (получения результата).

После того как будут определены требования к программе и составлен алгоритм решения поставленной задачи, алгоритм *записывается* на языке программирования.

Под *отладкой* понимается процесс устранения ошибок в программе. Различают синтаксические и алгоритмические ошибки. Синтаксические ошибки — это ошибки записи инструкций. Они наиболее легко устранимы, т. к. об их наличии программиста информирует компилятор (среда разработки). Алгоритмические ошибки обнаружить труднее. Действительно, если формула, по которой выполняется расчет, записана неверно (например, вместо знака + поставлен –), то как компилятор может это определить? Этап отладки можно считать законченным, если программа правильно работает на одном-двух наборах входных данных.

Этап *тестирования* особенно важен, если вы предполагаете, что вашей программой будут пользоваться другие люди. На этом этапе следует проверить, как ведет себя программа при обработке различных данных, причем, возможно, и неверных. Например, следует проверить, как будет вести себя программа вычисления тока в электрической цепи, если оставить незаполненным поле, предназначенное для ввода величины сопротивления.

Алгоритм

Алгоритм — точное предписание, определяющее процесс перехода от исходных данных к результату.

Предписание считается алгоритмом, если оно обладает следующими тремя свойствами:

- *определенностью*, т. е. точностью, не оставляющей место произволу при выполнении предписания;
- *универсальностью*, т. е. возможностью обработки различных, находящихся в заранее оговоренных пределах, исходных данных;
- *результативностью*, т. е. направленностью на получение результата.

Далее приведен алгоритм вычисления сопротивления электрической цепи, состоящей из двух сопротивлений (резисторов), которые могут быть соединены последовательно или параллельно. Алгоритм состоит из 3 шагов. Шаги алгоритма выполняются, начиная с первого.

1. Получить исходные данные: величина сопротивления первого резистора (R_1); величина сопротивления второго резистора (R_2); способ соединения резисторов (s = 1, если резисторы соединены последовательно, и s = 2, если резисторы соединены параллельно).
2. Если резисторы соединены последовательно, то по формуле $R = R_1 + R_2$ вычислить сопротивление цепи, иначе вычислить сопротивление цепи по формуле $R = \frac{R_1 \times R_2}{R_1 + R_2}$.
3. Вывести на экран значение сопротивления цепи (R).

Приведенное предписание обладает всеми тремя свойствами алгоритма:

- определенностью: в предписании указаны исходные данные, условия и формулы, по которым надо выполнять расчет; формулы указаны для каждого из двух возможных способов соединения сопротивлений;
- универсальностью: в предписании указаны не конкретные значения сопротивлений, а формулы;

- ❒ результативностью: при выполнении предписания получается конкретный результат — значение сопротивления цепи.

Следует обратить внимание, при описании алгоритма используются обобщенные понятия, например "величина сопротивления первого резистора", "способ соединения резисторов". При решении задачи "по алгоритму" эти понятия конкретизируются, получают конкретные значения.

Алгоритм решения задачи может быть представлен в виде словесного описания или графически — в виде блок-схемы.

В блок-схемах для обозначения логически различающихся фрагментов программы (действий) используются определенные стандартные символы, основные из которых показаны на рис. 3.1.

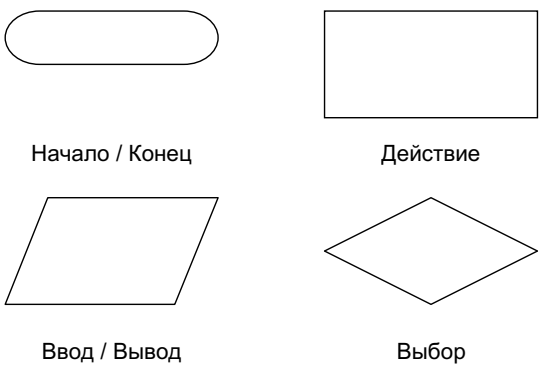


Рис. 3.1. Символы, используемые для графического представления алгоритмов

В качестве примера на рис. 3.2 представлена блок-схема алгоритма вычисления сопротивления электрической цепи, состоящей из двух резисторов, которые могут быть соединены последовательно или параллельно.

Изображение алгоритма в виде блок-схемы позволяет наглядно представить последовательность действий, которые необходимо выполнить для решения поставленной задачи, убедиться, в том числе и самому программисту, в правильности понимания поставленной задачи, процесса ее решения.

Следует обратить внимание, что одну и ту же задачу можно решить поразному, следуя разным алгоритмам. Так, например, алгоритм вычисления сопротивления электрической цепи, состоящий из двух сопротивлений, которые могут быть соединены последовательно или параллельно, можно представить и так, как показано на рис. 3.3.

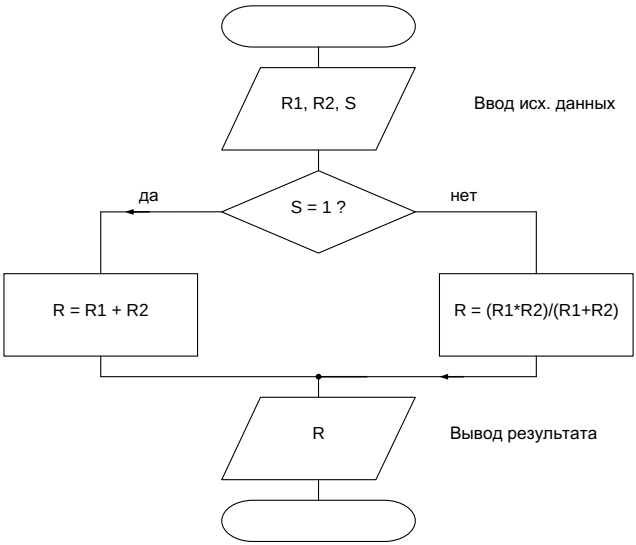


Рис. 3.2. Блок-схема алгоритма вычисления сопротивления электрической цепи

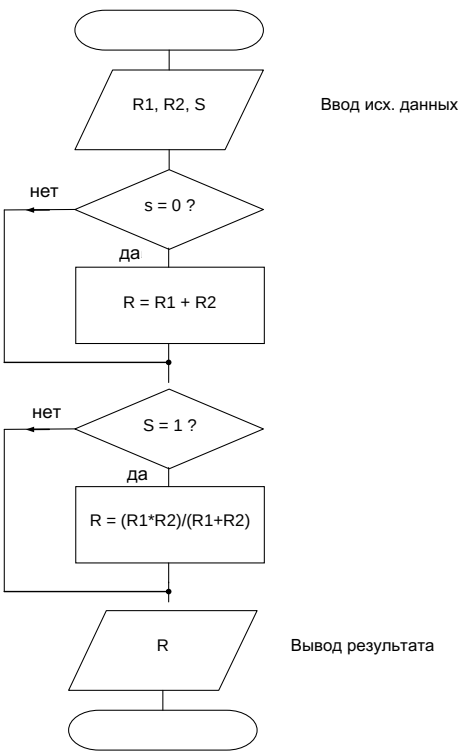


Рис. 3.3. Алгоритм вычисления сопротивления электрической цепи (вариант 2)

После разработки алгоритма решения задачи и представления его в виде блок-схемы можно перейти к составлению программы — записи алгоритма на выбранном языке программирования.

Алгоритмические структуры

Алгоритм решения любой задачи можно представить как совокупность следующих алгоритмических структур:

- следование;
- выбор;
- цикл.

Следование

Действия, которые надо выполнить, чтобы пересчитать расстояние из верст в километры (расчет по формуле), можно представить так: получить исходные данные (ввод), посчитать (расчет), отобразить результат (вывод). Это пример последовательного алгоритма.

В алгоритмической структуре "следование" все действия (шаги алгоритма) выполняются последовательно, одно за другим и всегда в одном и том же порядке (рис. 3.4).

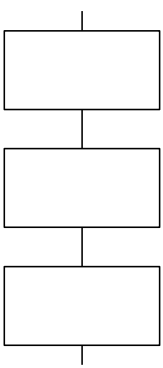


Рис. 3.4. Алгоритмическая структура "следование"

Выбор

На практике редко встречаются задачи, алгоритмы решения которых являются линейным. Действия, которые необходимо выполнить для достижения результата, как правило, зависят от исходных условий и промежуточных (полу-

ченных во время работы программы) данных. Например, в программе расчета сопротивления электрической цепи, состоящей из двух сопротивлений, формула, которую следует использовать для расчета, выбирается в зависимости от способа соединения сопротивлений.

Алгоритмическая структура, соответствующая ситуации, в которой надо выбрать действие (путь дальнейшего развития алгоритма) в зависимости от выполнения (или невыполнения) некоторого *условия*, называется "выбором". Наиболее часто встречаются ситуации, когда надо выбрать один из двух возможных вариантов действия (рис. 3.5), выполнить некоторое действие при условии выполнения определенного условия (рис. 3.6) или выбрать одно действие из нескольких возможных вариантов (рис. 3.7). Эти ситуации действия моделируются, соответственно, алгоритмическими структурами "выбор" и "множественный выбор".

В Visual Basic алгоритмическая структура "выбор" реализуется инструкцией `IF`, алгоритмическая структура "множественный выбор" — инструкцией `Select` (описание этих инструкций приведено в соответствующих разделах этой главы).

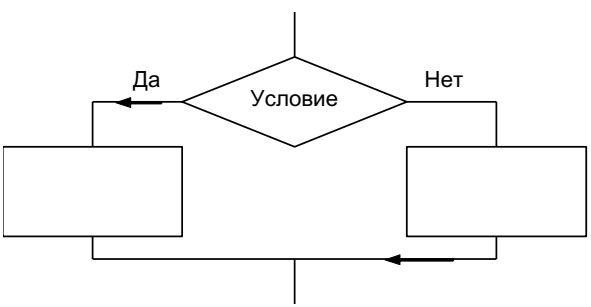


Рис. 3.5. Алгоритмическая структура "выбор"

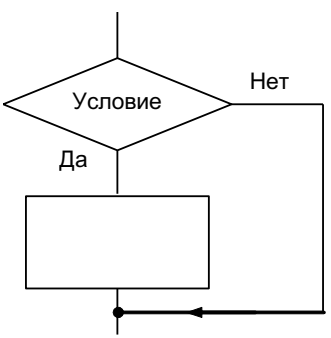


Рис. 3.6. Вариант алгоритмической структуры "выбор"

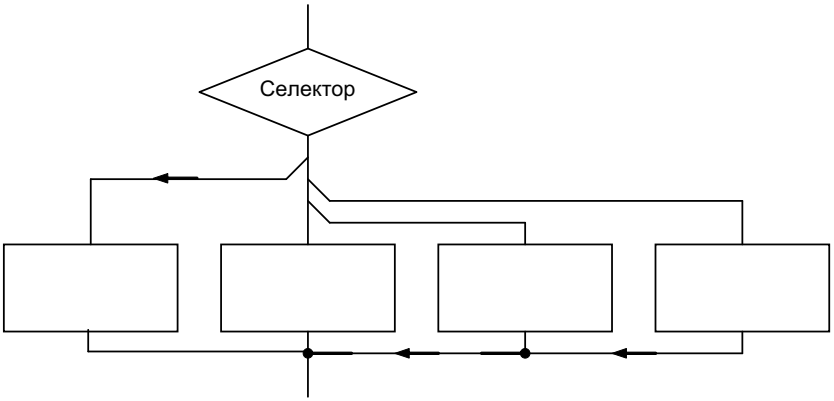


Рис. 3.7. Алгоритмическая структура "множественный выбор"

Цикл

Часто для достижения результата одну и ту же последовательность действий необходимо выполнить несколько раз. Например, чтобы построить таблицу значений функции, надо вычислить значение функции, вывести на экран значение аргумента и функции, изменить значение аргумента, затем повторить описанные выше действия еще раз. Такой алгоритм решения задачи называется *циклическим* или *циклом*.

Различают циклы с предусловием (рис. 3.8), постусловием (рис. 3.9) и циклы с фиксированным количеством повторений (рис. 3.10).

В Visual Basic цикл с предусловием реализуется инструкцией `Do While`, цикл с постусловием — `Do Loop...Until`. Цикл с фиксированным количеством повторений реализует инструкция `For` (описание этих инструкций приведено в соответствующих разделах этой главы).

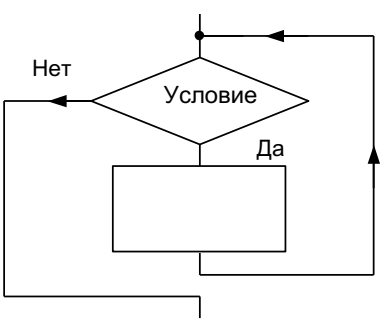


Рис. 3.8. Цикл с предусловием

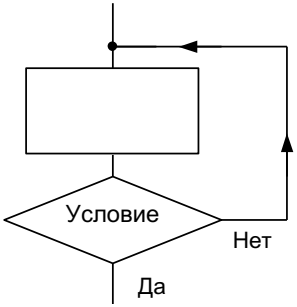


Рис. 3.9. Цикл с постусловием

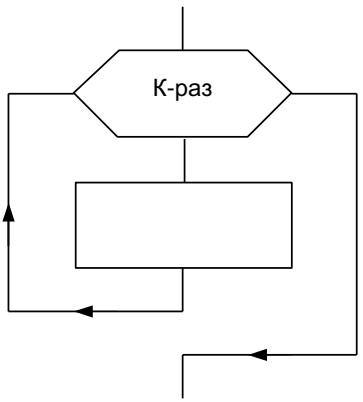


Рис. 3.10. Цикл с фиксированным количеством повторений

Структурное программирование

Разрабатывая программу, следует придерживаться определенных правил, совокупность которых часто называют правилами структурного программирования, а метод программирования, основанный на выполнении этих правил, называют структурным программированием.

Под структурным программированием понимается метод программирования, обеспечивающий создание программы, структура которой ясна и неразрывно связана со структурой решаемой задачи. В основе метода структурного программирования лежит подход, при котором исходная (большая) задача делится (разбивается) на несколько достаточно крупных подзадач (подзадачи, в свою очередь, тоже могут быть разбиты на подзадачи), каждая из которых оформляется в виде подпрограммы (процедура или функция). При этом алгоритм каждой подзадачи представляется как композиция (совокупность) алго-

ритмических конструкций (структур): следование, выбор, множественный выбор, цикл с предусловием, цикл с постусловием, цикл с фиксированным количеством повторений.

В Visual Basic возможность разбиения задачи на подзадачи может быть реализована путем оформления фрагментов программы в виде подпрограмм (процедур или функций), а перечисленные выше алгоритмические структуры могут быть реализованы при помощи соответствующих инструкций. Так структуре "выбор" соответствует инструкция `If`, структуре "множественный выбор" — `Select`, циклу с фиксированным количеством повторений соответствует инструкция `For`, а циклам с пред- и постусловием — инструкции `Do While` и `Do Loop While`.

С понятием структурного программирования тесно связано понятие "стиль программирования". Под хорошим стилем программирования понимаются определенные правила записи текста программы, а именно:

- использование имеющих смысловую нагрузку имен переменных и подпрограмм;
- использование отступов и пустых строк при записи текста программы;
- комментирование назначения переменных и ключевых точек программы.

Использование имеющих смысловую нагрузку имен переменных делает программу более понятной. Переменной можно присвоить любое имя. Однако лучше, чтобы имя переменной было связано с ее назначением. Например, в программе вычисления дохода по вкладу, переменные лучше назвать `Sum`, `Percent` и `Period`, а не `s`, `p` и `n`.

Отступы следует использовать при записи инструкций выбора, варианта и циклов. Они позволяют выделить логическую структуру инструкции. Пустые строки между последовательностями инструкций программы, реализующими некоторую часть задачи подпрограммы, выделяют логическую структуру программы.

Комментарии существенно облегчают понимание текста программы. Программист пишет комментарии, прежде всего, для себя и если предполагается, что с программой будут работать другие люди, то и для них. Когда комментарии пишутся для других, то понятно, для чего они нужны, — чтобы читающий программу мог в ней разобраться. Комментарий "для себя" программист пишет для того, чтобы облегчить процесс отладки, а также для того, чтобы, если через некоторое время потребуется внести изменения в программу, не тратить много времени на вспоминание, что делает программа и как она работает. Обычно комментируют назначение переменных в инструкциях их объявления и основные точки программы, в том числе назначение

подпрограмм. Текст комментариев должен быть кратким, но содержательным.

Контрольные вопросы

1. Перечислите алгоритмические структуры.
2. Перечислите алгоритмические структуры типа "выбор".
3. Перечислите циклические алгоритмические структуры.
4. Что такое условие?
5. В цикле с предусловием задается условие выполнения инструкций тела цикла?
6. В цикле с постусловием задается условие повторного выполнения инструкций тела цикла или условие завершения цикла?

Программа

Программа на Visual Basic представляет собой совокупность подпрограмм — процедур и функций, в простейшем случае — процедур обработки событий.

Начинается процедура заголовком (`Sub`), за которым указывается имя процедуры. Заканчивается процедура инструкцией `End Sub`. Между `Sub` и `End Sub` располагаются выполняемые инструкции процедуры. Обычно в одной строке пишут одну инструкцию. Некоторые инструкции, например `If`, принято записывать в несколько строк.

Комментарии

Для пояснения логики работы программы в ее текст можно включать комментарии. Комментарии обычно размещают в отдельной строке текста программы или после инструкции, действие которой поясняется. Перед поясняющим текстом ставят слово `Rem` (сокращение от remark) или апостроф (одинарную кавычку).

Типы данных и переменные

Visual Basic позволяет обрабатывать числовые и символьные данные, оперировать с датами и логическими величинами.

Для хранения в программе данных (исходных, промежуточных и результата) необходимы переменные. *Переменная* — это область памяти компьютера, в которой находятся данные. Обращение к переменной, например, для получения или сохранения данных, осуществляется по имени.

В отличие от языков Паскаль (Pascal) и Си (C), в программе Visual Basic можно не перечислять все переменные (перечисление переменных, используемых в программе, называется *объявлением переменных*). Тем не менее, все-таки лучше объявить все переменные в начале программы (процедуры). Делается это при помощи инструкции `Dim`, которая в общем виде записывается так:

Dim *Имя* **As** *Тип*

Здесь *Имя* и *Тип* — соответственно, имя и тип объявляемой переменной.

Например:

```
Dim funt As Double
Dim kg As Double
Dim Kol As Integer
Dim I As Integer
Dim Name As String
```

В качестве имени переменной можно использовать последовательность символов из букв латинского алфавита и цифр (первым символом должна быть буква).

Инструкция объявления переменной (`Dim`) не только декларирует факт существования переменной, но и задает диапазон ее допустимых значений. В табл. 3.1 перечислены основные типы данных Visual Basic.

Таблица 3.1. *Типы данных Visual Basic*

Тип данных	Диапазон значений
`Byte` (байт)	От 0 до 255
`Integer` (целое)	От –32 768 до 32 767
`Long` (длинное целое)	От –2 147 483 648 до 2 147 483 647
`Boolean` (логический тип)	`True` или `False`
`Single` (с плавающей точкой, обычной точности)	От –3,402823E38 до –1,401298E–45 для отрицательных значений; от 1,401298E–45 до 3,402823E38 для положительных значений
`Double` (с плавающей точкой, двойной точности)	От –1,79769313486232E308 до –4,94065645841247E–324 для отрицательных значений; от 4,94065645841247E–324 до 1,79769313486232E308 для положительных значений

Таблица 3.1 *(окончание)*

Тип данных	Диапазон значений
`Currency` (денежный)	От –922337203685477,5808 до 922337203685477,5807
`Decimal` (масштабируемое целое)	–79 228 162 514 264 337 593 543 950 335 без дробной части; –7,9228162514264337593543950335 с 28 знаками справа от запятой; минимальное ненулевое значение имеет вид –0,0000000000000000000000000001
`Date` (даты и время)	От 1 января 100 г. до 31 декабря 9999 г.
`String` (строка переменной длины)	Строка переменной длины: от 0 до приблизительно 2 миллиардов символов. Строка постоянной длины: от 1 до приблизительно 65 400 символов
`Variant` (числовые подтипы)	Любое числовое (вплоть до границ диапазона для типа `Double`) или строковое значение

Типы `Byte`, `Integer` и `Long` используются для представления целых значений. Для представления дробных значений служат типы `Single`, `Double`, `Currency`. Тип `Boolean` предназначен для представления логических величин, `Date` — для календарных дат и времени.

Контрольные вопросы

1. Перечислите основные типы данных языка Visual Basic.
2. Перечислите целые типы данных.
3. Перечислите дробные типы данных.
4. Объявите переменные, необходимые в программе вычисления:
   - площади прямоугольника;
   - величины силы тока в электрической цепи;
   - стоимости печати фотографий;
   - дохода по вкладу в банке.

Константы

Константы используются в программе для представления конкретных значений. Различают числовые (целые и дробные), строковые и логические константы.

Числовые константы

В тексте программы числовые константы записываются обычным образом. Следует обратить внимание, *в тексте программы* при записи дробных констант в качестве десятичного разделителя используется *точка*, а при вводе дробных значений во время работы программы — *запятая*.

Примеры числовых констант:

```
0
123
0.05
-1.0
```

Числовые константы можно записать в научной (экспоненциальной) форме. Представление числа в экспоненциальной форме основано на том, что любое число можно записать в алгебраической форме, как произведение числа, меньшего 10, которое называется мантиссой, и степени десятки, именуемой порядком.

В табл. 3.2 приведены примеры чисел, записанных в обычной форме, в алгебраической форме и форме с плавающей точкой.

Таблица 3.2. *Примеры записи дробных чисел*

Число	Алгебраическая форма	Экспоненциальная форма (с плавающей точкой)
0.05	5×10^{-2}	5. 0000000000E–02
1 000 000	1×10^{6}	1.0000000000E+06
–123.452	-1.23452×10^{2}	–1.2345200000E+02
0.0056712	5.6712×10^{-3}	5.6712000000E–03

Строковые константы

Строковые константы, для того чтобы их можно было отличить от имен переменных, в тексте программы заключаются в кавычки. Вот примеры строковых констант:

```
"Visual Basic"
"руб."
""
```

Здесь следует обратить внимание на последнюю из приведенных констант (между двумя кавычками нет ни одного символа). Это так называемая *пустая строка*, т. е. строка, не содержащая ни одного символа. Она часто используется для инициализации строковых переменных.

Именованные константы

В Visual Basic определены именованные константы — идентификаторы, обозначающие конкретные (числовые или символьные) значения. Согласно принятому соглашению, именованные константы начинаются префиксом `vb`. Например, в инструкциях вывода часто используется константа `vbCrLf`, которая обозначает последовательность символов "возврат каретки, перевод строки". Именованные константы также часто используются как параметры функций. Например, тип сообщения, выводимого функцией `MsgBox`, определяет значение ее второго параметра, в качестве которого можно указать одну из констант: `vbInformation`, `vbExclamation`, `vbQuestion` или `vbCritical`.

Использование именованных констант позволяет сделать программу более "читаемой", избавляет программиста от необходимости помнить коды командных кнопок, типов сообщений и т. п.

Инструкция присваивания

Инструкция присваивания является основной вычислительной инструкцией.

В результате выполнения инструкции присваивания значение переменной меняется, ей *присваивается* новое значение.

Пример:

```
Sum = Cena * Kol
Kg = 0.495 * Funt
Total = 0
```

В общем виде инструкция присваивания записывается так:

Переменная = Выражение

Здесь:

- *Переменная* — переменная, значение которой надо изменить (присвоить значение);
- = — символ "присвоить";
- *Выражение* — выражение, значение которого надо записать в переменную.

Выполняется инструкция присваивания следующим образом: сначала вычисляется значение выражения, указанного справа от символа присваивания (=), затем полученное значение записывается в переменную, имя которой указано слева от символа присваивания.

Например, в результате выполнения инструкций:

`i = 0` значение переменной `i` становится равным нулю;

`a = b + c` значение переменной `a` становится равным сумме значений переменных `b` и `c`;

`n = n + 1` значение переменной `n` увеличивается на единицу.

Инструкция присваивания считается верной, если тип выражения соответствует типу переменной. Числовой переменной можно присвоить значение числового типа, символьной — символьного.

Выражение

Выражение состоит из *операндов* и *операторов*. Операнды, в качестве которых могут выступать константы, переменные, а также функции — это объекты, над которыми выполняются действия. Операторы (табл. 3.3) обозначают действия, выполняемые над операндами.

Таблица 3.3. *Операторы*

Оператор	Действие
+	Сложение
–	Вычитание
*	Умножение
/	Деление
\	Целочисленное деление (целая часть результата деления)

Простое выражение состоит из двух операндов и находящегося между ними оператора.

Например:

```
i + 1
Cena * Kol
sum — discount
cena * 0.05
U / R
```

Если выражение состоит из одного-единственного операнда, представляющего собой константу, переменную или функцию, то оно называется *простейшим*.

Например:

```
0.05
K
0
```

При вычислении значения выражения следует учитывать порядок выполнения действий. В общем случае действует известное правило вычислений значений выражений: сначала выполняется умножение и деление, затем — сложение и вычитание. Другими словами, сначала выполняются действия, соответствующие операторам, имеющим более высокий приоритет (`*`, `/`), затем более низкий (`+`, `-`). Если приоритет операторов одинаков, то сначала выполняется действие, соответствующее оператору, находящемуся левее.

Для задания требуемого порядка выполнения операторов в выражении можно использовать скобки. Например:

```
(a + b) / 2
```

Выражение, заключенное в скобки, трактуется как один операнд. Это значит, что операторы выражения, стоящего в скобках, будут выполняться в обычном порядке, но раньше, чем операторы, находящиеся за скобками.

При записи выражений, содержащих скобки, должна соблюдаться парность скобок, т. е. число открывающих скобок должно быть равно числу закрывающих скобок. Нарушение парности скобок — наиболее распространенная ошибка при записи выражений.

Тип выражения

Важной характеристикой выражения является его тип.

Тип выражения определяется типом операндов, входящих в выражение, а также зависит от вида операций, выполняемых над операндами (табл. 3.4). Например, если оба операнда, над которыми выполняется операция сложения, целого типа, то очевидно, что результат тоже будет целого типа. В то же время, при выполнении операции деления, даже если оба операнда целого типа, в общем случае получается дробное значение.

При выполнении инструкции присваивания значение выражения приводится к типу переменной, которая получает значение. Например, если переменные `a` и `b` целого типа (например, `Integer`) и их значения равны, соответственно, 15 и 4, то в результате выполнения инструкции `c = a / b` значение

переменной c будет равно 4, если переменная c объявлена как целая (значение 3.75 округляется до 4), и 3.75, если переменная c дробного (Double) типа.

Таблица 3.4. *Правила определения типа выражения*

Оператор	Тип операндов	Тип выражения
*, +, -	Оба операнда целого типа	Целый
*, +, -	Хотя бы один из операндов дробного типа	Дробный
/	Целый или дробный	Дробный
\	Целый или дробный	Целый

Контрольные вопросы

1. Запишите инструкцию присваивания, обеспечивающую вычисление:
   - — площади прямоугольника;
   - — величины силы тока в электрической цепи;
   - — стоимости печати фотографий;
   - — дохода по вкладу в банке;
   - — увеличение на единицу значения переменной N.
2. Укажите тип выражения, если переменные A и B целого типа, а переменная C — дробная:
   - — A + C
   - — B + 0.25
   - — (A + B) / 2
   - — A / B
3. Значение переменной A равно 75. Чему будет равно значение переменной B, в результате выполнения инструкции B = A * 0.1, если переменная B объявлена:
   - — **Dim** B **As Integer**
   - — **Dim** B **As Double**
4. Переменной целого типа присваивается значение выражения дробного типа. Что при этом происходит?

Функция

Функция — это подпрограмма, обеспечивающая решение некоторой задачи и возвращающая в вызвавшую ее программу результат, который называется *значением функции*. Например, значением стандартной функции Sin является синус угла, указанного в качестве параметра функции, а значением функции Sqr — квадратный корень числа.

Чтобы получить значение функции, ее надо вызвать — указать функцию в качестве операнда в инструкции присваивания.

Например:

```
r = Sqr(x)
y = L * Sin(a)
```

В табл. 3.5 приведены некоторые математические функции Visual Basic.

Таблица 3.5. *Математические функции*

Функция	Значение
`Abs(x)`	Абсолютное значение `x`
`Sqr(x)`	Квадратный корень из `x`
`Exp(x)`	Экспонента `x`
`Log(x)`	Натуральный (по основанию *e*) логарифм `x`
`Sin(a)`	Синус угла `a`
`Cos(a)`	Косинус угла `a`
`Tan(a)`	Тангенс угла `a`
`Atan(x)`	Угол, тангенс которого равен `x`

Следует обратить внимание на то, что величина угла тригонометрических функций должна быть выражена в радианах.

В программах Visual Basic, в инструкциях, обеспечивающих ввод исходных данных и отображение результата, часто используются функции `Val` и `Str`. Функция `Val` преобразует строковое значение в число, а функция `Str` — число в строку.

Ввод данных

Операция ввода данных заключается в получении от пользователя исходных данных, например, значений переменных, используемых в расчетных формулах.

Наиболее просто программа может получить данные из поля редактирования (компонент `TextBox`). В качестве примера на рис. 3.11 приведена форма программы пересчета расстояния из миль в километры. Очевидно, что для выполнения расчета программа должна "взять" число, введенное пользователем в поле редактирования, и записать его в переменную, например, `mile`, которая в общем случае должна быть дробного типа. Обратите внимание,

Text1 — это *имя* компонента. Имя компоненту присваивает среда разработки в момент добавления компонента на форму. По умолчанию имя компонента TextBox состоит из префикса Text и порядкового номера компонента.

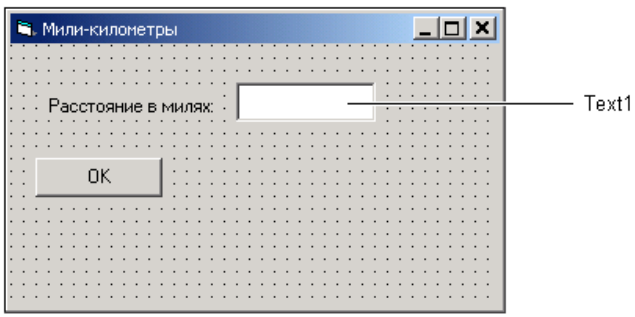


Рис. 3.11. Компонент Text1 предназначен для ввода исходных данных — расстояния в милях

Доступ содержимому поля редактирования обеспечивает свойство Text. Так как свойство Text символьного типа (в поле редактирования можно ввести любую последовательность символов), то напрямую присвоить значение свойства Text переменной числового типа нельзя. Нужно преобразовать символьное значение в числовое. Сделать это можно при помощи функции Val, например, так:

```
mile = Val(Text1.Text)
```

В приведенной инструкции:

- mile — переменная числового (дробного или целого) типа;
- Val — функция преобразования строки в число;
- Text1 — имя компонента (поля редактирования), в котором находятся данные;
- Text — свойство, которое содержит строку, находящуюся в поле редактирования.

Следует обратить внимание, если строка, указанная в качестве параметра функции Val, не является изображением числа, функция возвращает ноль. Строго говоря, функция обрабатывает символы строки-параметра от начала до первого недопустимого символа. Таблица 3.6 иллюстрирует работу функции Val. Обратите внимание, если в качестве десятичного разделителя в строке-параметре функции Val указана запятая, то дробная часть отбрасывается.

Таблица 3.6. *Пример значений функции* `Val`

Строка-параметр функции `Val`	Значение функции `Val`
`0.5`	`0.5`
`0,5`	`0`
`125.45`	`125.45`
`125,45`	`125`
`125 45`	`125`
`двадцать пять`	`0`
Пустая строка	`0`

Необходимо еще раз обратить внимание на то, что во время работы программы при вводе дробных чисел при помощи функции `Val` в качестве десятичного разделителя следует использовать *точку*.

Для преобразования символьного значения в числовое можно использовать также функции `CDbl` и `CInt`. Функция `CDbl` преобразует строку в дробное число, `CInt` — в целое. Например, если переменная `cena` дробного (`Double`) типа, а переменная `kol` целая (`Integer`), то инструкции ввода данных могут выглядеть так:

```
cena = CDbl(Text1.Text)
kol = CInt(Text2.Text)
```

Отличие функций `CDbl` и `CInt` от функции `Val` состоит в том, что функция `Val` всегда возвращает значение (число или ноль, если строка-параметр не является изображением числа), в то время как функции `CDbl` и `CInt` возвращают значение только в том случае, если строка-параметр является *правильным* изображением числа (дробного для `CDbl` и целого для `CInt`). Если строка-параметр не является изображением числа, то возникает *исключение* (ошибка) и программа завершает работу. Также необходимо обратить внимание на то, что во время работы программы при вводе дробных чисел при помощи функции `CDbl` в качестве десятичного разделителя следует использовать *запятую*.

Контрольные вопросы

1. Для ввода значения переменной дробного типа `Cena` используется компонент `TextBox` (его имя `Text1`). Запишите инструкцию, обеспечивающую ввод значения этой переменной.
2. Для ввода значения переменной целого типа `Kol` используется компонент `TextBox` (его имя `Text2`). Запишите инструкцию, обеспечивающую ввод значения этой переменной.

3. Для ввода значения переменной `m` в программе используется инструкция `m = Val(Text1.Text)`. Во время работы программы в поле `Text1` пользователь ввел строку `25,5`. Чему равно значение переменной `m`? Почему?
4. Во время работы программы после ввода исходных данных и активизации процесса вычисления в программе произошла ошибка — на экране появилось сообщение "Run-time error '13': Type mismatch". Укажите наиболее вероятную причину ее возникновения.

Вывод результата

Операция вывода заключается в отображении результата работы программы, например значения переменной, в окне программы.

Отображение текста в окне программы обеспечивает компонент `Label`. В качестве примера на рис. 3.12 приведена форма программы пересчета расстояния из миль в километры. Компонент `Label2` предназначен для отображения результата расчета.

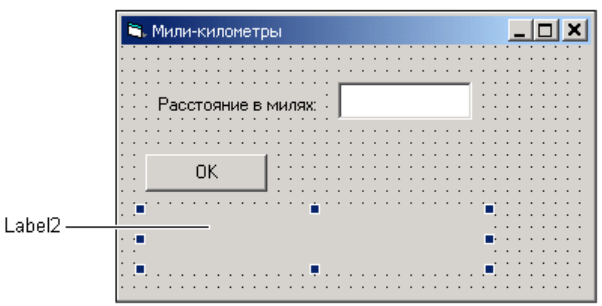


Рис. 3.12. Компонент `Label2` предназначен для отображения результата расчета

Для того чтобы результат расчета появился в поле компонента `Label`, надо свойству `Caption` этого компонента присвоить соответствующее значение. Так как свойство `Caption` символьного типа, то присвоить ему значение числовой переменной напрямую нельзя. Нужно преобразовать числовое значение в символьное. Сделать это можно при помощи функции `CStr`, например, так:

```
Label2.Caption = CStr(km)
```

В приведенной инструкции:

- `km` — переменная числового (дробного) типа;
- `CStr` — функция преобразования числового значения в строковое;

- Label2 — имя компонента (поля отображения текста), в которое надо вывести значение переменной km;
- Caption — свойство компонента Label3, значение которого определяет текст, отображаемый в поле компонента.

Если после значения переменной надо вывести пояснительный текст, то в левой части инструкции присваивания надо записать выражение, обеспечивающее формирование нужной строки. Например, чтобы в рассматриваемом примере после числового значения отображался текст "км", приведенную выше инструкцию надо переписать так:

```
Label2.Caption = CStr(km) + "км"
```

Если требуется, чтобы текст в поле редактирования был выведен в несколько строк, то перед строковой константой (пояснительным текстом) или строковой переменной, значение которой должно быть выведено в следующей строке, надо поместить константу vbCrLf. Например, так:

```
Label2.Caption = CStr(km) + "км" + vbCrLf + "(1 миля = 1.6 км)"
```

Контрольные вопросы

1. Переменная Sum содержит значение суммы покупки. Запишите инструкцию, обеспечивающую вывод значения переменной в поле компонента Label (его имя Label3). После числового значения должен отображаться текст "руб.".
2. Переменные Sum и Profit содержат значения величины дохода и суммы в конце срока вклада. Запишите инструкцию, которая обеспечивает отображение значения этих переменных в поле компонента Label3. Значения должны быть выведены каждое в отдельной строке. Перед значением переменной Profit должен отображаться текст "Доход:", перед значением переменной Sum — "Сумма в конце срока вклада:".

Вывод сообщений

Во время работы программы может возникнуть необходимость вывести сообщение. Для этой цели используют процедуру или функцию MsgBox.

Процедура MsgBox выводит на экран окно сообщения с кнопкой **OK**, щелчок на которой закрывает окно сообщения. Например, инструкция

```
MsgBox "Надо ввести расстояние в милях", vbExclamation, "Мили-километры"
```

выводит окно сообщения, изображенное на рис. 3.13.

В общем виде инструкция вызова процедуры MsgBox выглядит так:

```
MsgBox Сообщение, ТипСообщения, Заголовок
```

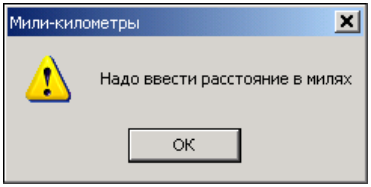


Рис. 3.13. Пример сообщения

Назначение параметров *Сообщение* и *Заголовок* понятно из их названия.

Необязательный параметр *ТипДиалога* (целое число) задает значок, отображаемый в окне сообщения (табл. 3.7). Если параметр не указан, то значок не отображается.

Таблица 3.7. *Возможные значения параметра* *ТипСообщения*

Значение	Значок
vbInformation	— "Информация"
vbExclamation	— "Внимание"
vbQuestion	— "Вопрос"
vbCritical	— "Стоп"

Функция MsgBox не только выводит сообщение, но и позволяет определить, нажатием какой кнопки пользователь закрыл окно диалога.

Кнопки, отображаемые в окне сообщения, задает параметр *ТипДиалога*. Его удобно представить в виде суммы трех чисел: первое определяет картинку, второе — командные кнопки, третье — кнопку, выбранную по умолчанию.

В качестве первого числа можно использовать одну из приведенных в табл. 3.7 констант, в качестве второго — константу vbOkCancel, vbYesNo или vbYesNoCancel, в качестве третьей — vbDefaultButton1 (если по умолчанию выбранной является первая из отображаемых кнопок), vbDefaultButton2 или vbDefaultButton3.

Значением функции MsgBox является число, соответствующее кнопке, нажатой пользователем в окне сообщения. В табл. 3.8 приведены значения функции MsgBox в зависимости от способа завершения диалога (нажатой кнопки).

Таблица 3.8. *Значения функции* `MsgBox`

Нажата кнопка	Значение функции
OK	`vbOk`
Cancel	`vbCancel`
Да	`vbYes`
Нет	`vbNo`

Инструкции управления

Как было сказано ранее в этой главе, алгоритмы решения большинства задач не являются линейными. Действия, которые необходимо выполнить для решения поставленной задачи, как правило, зависят от выполнения определенных условий.

Условие

Условие — это выражение логического (`Boolean`) типа, которое может принимать одно из двух значений: истина (`True`) или ложь (`False`).

Различают простые и сложные условия. Простое условие состоит из двух операндов и оператора сравнения, находящегося между ними. Операндами выражения-условия могут быть константы (числовые, строковые, логические), переменные и функции. Операторы обозначают действия, выполняемые над операндами.

Вот примеры условий:

```
Sum > 1000
Name = "Nikita"
D <> 0
```

Операторы сравнения приведены в табл. 3.9.

Таблица 3.9. *Операторы сравнения*

Оператор	Значение
`>` (больше)	`True`, если операнд, стоящий слева от оператора, больше операнда, стоящего справа от оператора, иначе `False`
`<` (меньше)	`True`, если операнд, стоящий слева от оператора, меньше операнда, стоящего справа от оператора, иначе `False`

Таблица 3.9 *(окончание)*

Оператор	Значение
= (равно)	`True`, если операнд, стоящий слева от оператора, равен операнду, стоящему справа от оператора, иначе `False`
<> (не равно)	`True`, если операнд, стоящий слева от оператора, не равен операнду, стоящему справа от оператора, иначе `False`
>= (больше или равно)	`True`, если операнд, стоящий слева от оператора, больше или равен операнду, стоящему справа от оператора, иначе `False`
<= (меньше или равно)	`True`, если операнд, стоящий слева от оператора, меньше или равен операнду, стоящему справа от оператора, иначе `False`

Применение операторов сравнения к операндам числового типа не вызывает затруднений. Например, если значение переменной `n` равно `10`, то значение выражения `n < 10` равно `False`, а выражения `n = 10` — `True`.

Переменную символьного типа можно сравнить с другой переменной символьного типа или с символьной константой. Операторы сравнения можно применять к символам потому, что каждому символу соответствует число, называемое *кодом символа*, причем код символа "0" меньше, чем код символа "9", код символа "A" меньше, чем код символа "B", код символа "Z" меньше, чем код символа "a". Таким образом:

"0" < "1" < ... < "9" < ... < "A" < "B" < ... < "Z" < "a" < "b" < ... < "z"

Символам русского алфавита соответствуют числа большие, чем символам латинского алфавита, при этом справедливо следующее:

"А" < "Б" < "В" < ... < "Ю" < "Я" < "а" < "б" < "в" < ... < "э" < "ю" < "я"

Строки сравниваются посимвольно, начиная с первого символа. Если количество символов в сравниваемых строках одинаково и все символы совпадают, то такие строки считаются равными. Если в одинаковых позициях строк находятся разные символы, то большей считается та строка, у которой в этой позиции находится символ с большим кодом. В табл. 3.10 приведены примеры сравнения строк.

Таблица 3.10. *Примеры сравнения строк*

Строка 1	Строка 2	Результат сравнения
Visual Basic	Visual Basic	Строки равны
Иванов	Петров	Строка 1 меньше строки 2
Иванова	Иванов	Строка 1 больше строки 2

Операторы сравнения позволяют записывать простые условия, из которых путем применения логических операторов `And` (логическое И), `Or` (логическое ИЛИ) и `Not` (отрицание) можно строить сложные условия.

В табл. 3.11 приведены результаты применения логических операторов к операндам-условиям `У1` и `У2`.

Таблица 3.11. *Действие логических операторов*

У1	**У2**	**У1 And У2**	**У1 Or У2**
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

Из таблицы видно, что выражение `У1 And У2` истинно (его значение равно `True`) только в том случае, если значение обоих операндов истинно, т. е. оба условия выполняются. Выражение `У1 Or У2` истинно, если значение хотя бы одного операнда равно `True`, т. е. в том случае, когда хотя бы одно из условий (выполняется) истинно.

Оператор `Not` изменяет значение логического выражения на противоположное.

Примеры сложных условий:

```
(x >= x1) And (x <= x2)
(x < x1) Or (x > x2)
(Sum >= 1000) And (sum < 10000)
```

Контрольные вопросы

1. Что такое условие?
2. Из чего состоит простое условие?
3. Перечислите операторы сравнения.
4. Скидка предоставляется, если сумма покупки больше 1000 руб. Запишите условие предоставления скидки.
5. При печати фотографий скидка предоставляется, если количество заказанных экземпляров не мене 15. Запишите условие предоставления скидки.
6. Запишите условие, позволяющее определить способ соединения (последовательно или параллельно) сопротивлений электрической цепи.
7. Перечислите логические операторы.
8. Запишите условие принадлежности значения переменной `x` интервалу `x1 .. x2`.
9. Скидка предоставляется при печати не менее 5 фотографий формата 18×24 или не менее 10 формата 10×15. Запишите условие предоставления скидки.

Инструкция *If*

Инструкция `If` используется в том случае, если нужно выбрать одно из двух возможных действий. В общем виде она записывается так:

```
If Условие Then
    Действие 1
Else
    Действие 2
End If
```

Выполняется инструкция `If` следующим образом. Сначала вычисляется значение выражения *Условие*. Затем, если условие выполняется (значение выражения *Условие* равно `True`), выполняются инструкции, находящиеся между `Then` и `Else`. Если условие не выполняется (значение выражения *Условие* равно `False`), то выполняются инструкции, находящиеся между `Else` и `End If`. Алгоритм, реализуемый инструкцией `If`, приведен на рис. 3.14.

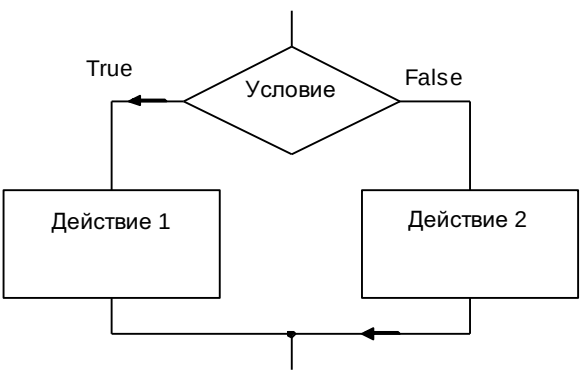


Рис. 3.14. Блок-схема инструкции `If`

Примеры:

```
If t  = 1 Then
    r = r1+r2
Else
    r = r1*r2/(r1+r2)
End If
```

В качестве примера использования инструкции `If` в листинге 3.1 приведена процедура обработки события `Click` на командной кнопке **OK** окна программы вычисления дохода по вкладу в банке (процентная ставка зависит

от суммы вклада: если сумма меньше 10 тыс. руб., то ставка 10%, если больше, то 12%). Форма программы приведена на рис. 3.15.

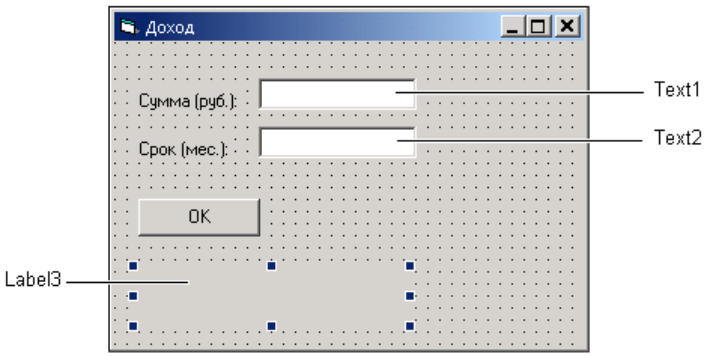


Рис. 3.15. Форма программы "Доход"

Листинг 3.1. Доход (инструкция `If`)

```
Private Sub Command1_Click()
    Dim sum As Double      ' сумма вклада
    Dim period As Integer  ' срок вклада, месяцев
    Dim percent As Double  ' процентная ставка (годовых)
    Dim profit As Double   ' доход

    ' ввод исходных данных
    sum = Val(Text1.Text)
    period = Val(Text2.Text)

    ' определить величину процентной ставки
    If sum < 10000 Then
        percent = 0.1
    Else
        percent = 0.12
    End If

    ' расчет
    profit = sum * (percent / 12) * period

    ' вывод результата
```

```
    Label3 = "Сумма вклада: " + CStr(sum) + " руб." + vbCrLf + _
             "Срок вклада: " + CStr(period) + " мес." + vbCrLf + _
             "Процентная ставка: " + CStr(percent * 100) + "%" + vbCrLf + _
             "Доход: " + CStr(profit) + " руб."
End Sub
```

Если какое-либо действие требуется выполнить только в случае выполнения какого-либо условия, а в случае невыполнения этого условия действие надо пропустить (ничего делать не надо), то инструкция `If` записывается так:

```
If Условие Then
    Действие
End If
```

При помощи нескольких инструкций `If` можно осуществить множественный выбор. Например, если необходимо выбрать один из трех вариантов, то реализовать это можно так:

```
If Условие 1 Then
    Действие 1
Else
    If Условие 2 Then
        Действие 2
    Else
        Действие 3
    End If
End If
```

В качестве примера в листинге 3.2 приведена процедура обработки события `Click` на кнопке **OK** окна программы "Контроль веса".

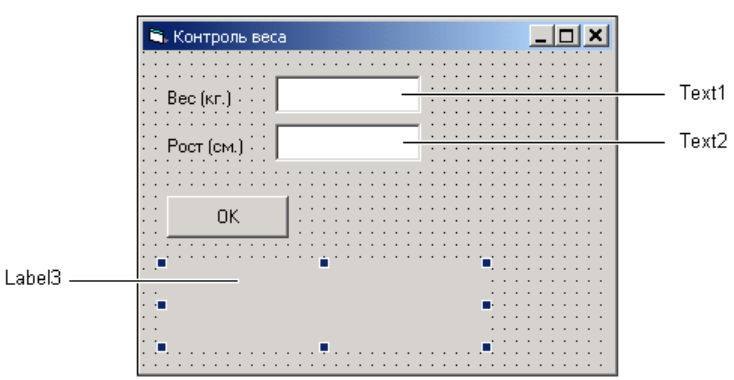


Рис. 3.16. Форма программы "Контроль веса"

Вес человека может быть недостаточным, избыточным или оптимальным. Оптимальным принято считать вес, вычисляемый по формуле `Рост - 100`. Программа "Контроль веса" вычисляет оптимальный вес, сравнивает его с реальным и выводит соответствующее сообщение. Форма программы приведена на рис. 3.16, алгоритм процедуры обработки события `Click` — на рис. 3.17.

Листинг 3.2. Контроль веса (множественный выбор)

```
Private Sub Command1_Click()
    Dim weight As Double ' вес
    Dim height As Double ' рост

    Dim optimal As Double ' оптимальный вес
    Dim dw As Double      ' отклонение веса от оптимального

    Dim msg As String     ' результат - сообщение

    ' ввод исходных данных
    weight = Val(Text1.Text)
    height = Val(Text2.Text)

    ' вычислить оптимальный вес
    optimal = height - 100

    If weight = optimal Then
        msg = "Ваш вес оптимален"
    Else
        If (weight < optimal) Then
            dw = optimal - weight
            msg = "Вам надо поправиться на " + CStr(dw) + " кг."
        Else
            dw = weight - optimal
            msg = "Вам надо похудеть на " + CStr(dw) + " кг."
        End If
    End If

    ' вывод результата
    Label3.Caption = msg

End Sub
```

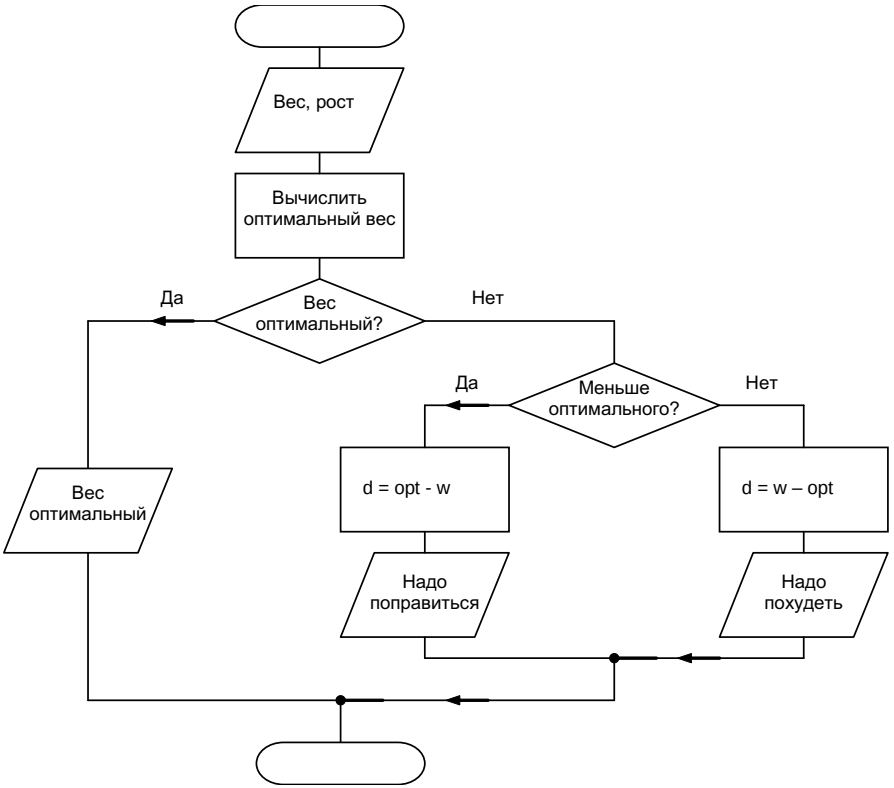


Рис. 3.17. Алгоритм обработки события `Click` на кнопке **OK** (программа "Контроль веса")

Контрольные вопросы

1. Какой алгоритмической структуре соответствует инструкция `If`?
2. Запишите инструкцию `If`, позволяющую вычислить сопротивление электрической цепи, состоящей из двух резисторов, которые могут быть соединены последовательно или параллельно.
3. Запишите инструкцию `If`, позволяющую вычислить величину скидки, предоставляемой покупателю, если сумма покупки больше 1000 руб.
4. Цена минуты разговора по телефону 1 руб. при условии, что длительность разговора не менее 5 минут, и 1,5 руб. в противном случае. Запишите инструкцию `If`, позволяющую вычислить стоимость разговора.
5. В субботу и воскресенье всем покупателям предоставляется скидка в 10%. Запишите инструкцию `If`, позволяющую вычислить величину скидки.
6. Процент дохода зависит от суммы вклада: 10%, если сумма меньше 5 тыс. руб., 11%, если сумма не превышает 20 тыс. руб., и 12%, если сумма вклада больше 20 тыс. руб. Запишите инструкцию `If`, позволяющую определить величину процентной ставки.

Инструкция *Select*

Инструкция `Select` позволяет реализовать множественный выбор. В общем виде она записывается так:

```
Select Case Селектор
     Case Список_1
          Инструкции1
     Case Список_2
          Инструкции2
     Case Список_3
          Инструкции3
     . . .
     Case Else
          Инструкции
End Select
```

Здесь *Селектор* — выражение, значение которого определяет дальнейший ход развития программы; *Список_i* — список констант (разделенные запятыми константы).

Выполняется инструкция `Case` следующим образом. Сначала вычисляется значение выражения *Селектор* (обычно в качестве селектора используется переменная). Затем значение селектора последовательно сравнивается со значениями констант списков *Список_1*, *Список_2* и т. д. Если значение выражения *Селектор* совпадет с каким-либо значением списка, то выполняется соответствующая группа инструкций. Если значение выражения *Селектор* не совпадает ни с одним значением ни одного из списков, то выполняются инструкции, следующие за `Else`. На рис. 3.18 приведена блок-схема инструкции `Select`.

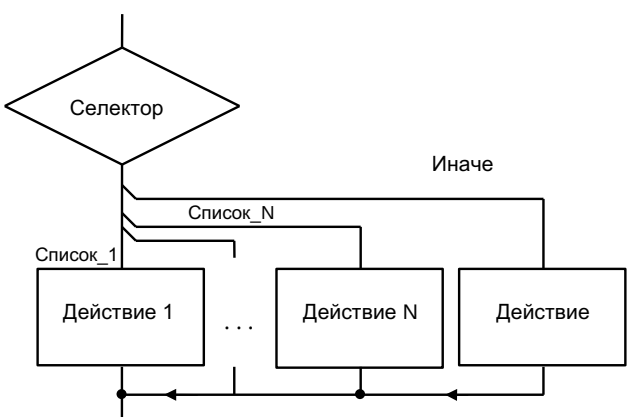


Рис. 3.18. Блок-схема инструкции `Select`

Пример:

```
Select Case period
        Case 1, 2, 3
            percent = 0.9
        Case 4, 5, 6
            percent = 0.115
        Case 7, 8, 9, 10, 11, 12
            percent = 0.135
        Case Else
            percent = 0.15
End Select
```

Если одно и то же действие должно быть выполнено для нескольких последовательных значений (диапазона), то после `Case` можно задать диапазон — указать нижнюю и верхнюю границы диапазона, разделив их словом `To`.

Например, приведенную выше инструкцию `Case` можно переписать так:

```
Select Case period
        Case 1 To 3
            percent = 0.9
        Case 4 To 6
            percent = 0.115
        Case 7 To 12
            percent = 0.135
        Case Else
            percent = 0.15
End Select
```

В качестве примера использования инструкции `Case` в листинге 3.3 приведена процедура обработки события `Click` на кнопке **OK** окна программы "Доход". Форма программы приведена на рис. 3.19.

Листинг 3.3. Доход (инструкция `Select`)

```
Private Sub Command1_Click()
    Dim sum As Double ' сумма вклада
    Dim period As Double ' срок вклада, месяцев
    Dim percent As Double ' процентная ставка (годовых)
    '-------------------------------------------------------------
```

```
    ' Срок      3 мес.  6 мес.  12 мес. 18 мес. 24 мес. 36 мес.
    '-------------------------------------------------------------
    ' Процент    9,0%    11,5%   13,5%   15,0%   18,0%   24,0%
    '-------------------------------------------------------------

    Dim profit As Double ' доход

    ' ввод исходных данных
    sum = Val(Text1.Text)
    period = Val(Text2.Text)

    ' определить величину процентной ставки
    Select Case period
        Case 1 To 3
            percent = 0.9
        Case 4 To 6
            percent = 0.115
        Case 7 To 12
            percent = 0.135
        Case 13 To 18
            percent = 0.15
        Case 19 To 24
            percent = 0.18
        Case 25 To 36
            percent = 0.24
    End Select

    ' вычислить доход
     profit = sum * (percent / 12) * period

    ' вывод результата
    Label3 = "Сумма вклада: " + CStr(sum) + " руб." + vbCrLf + _
           "Срок вклада: " + CStr(period) + " мес." + vbCrLf + _
           "Процентная ставка: " + CStr(percent * 100) + "%" + vbCrLf + _
           "Доход: " + CStr(profit) + " руб."
End Sub
```

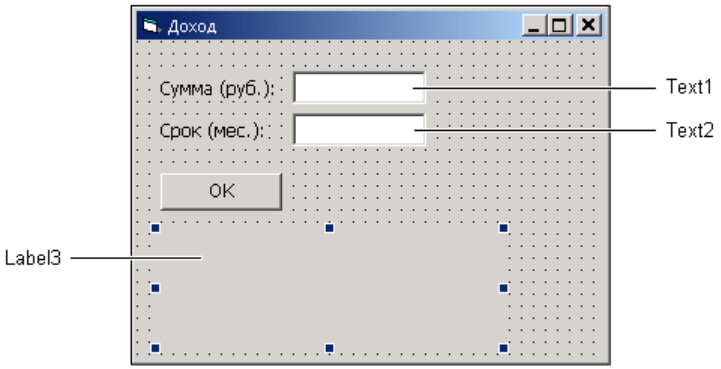


Рис. 3.19. Форма программы "Доход"

Контрольные вопросы

1. Какой алгоритмической структуре соответствует инструкция `Case`?
2. Какого типа должна быть переменная-селектор инструкции `Case`?
3. Цена печати фотографии размером 9×12 равна 2,5 руб., 10×15 — 4 руб., 18×24 — 8 руб. Запишите инструкцию `Case`, позволяющую вычислить стоимость печати фотографий в зависимости от выбранного размера.

Циклы

При решении многих задач одну и ту же последовательность действий приходится выполнять несколько раз. Например, чтобы построить таблицу значений функции в диапазоне `[x1, x2]` с шагом изменения аргумента `dx`, надо вычислить значение функции в точке `x`, вывести результат (значения аргумента и функции), увеличить значение `x` на `dx` и затем повторить описанные выше действия еще раз (столько раз, в скольких точках надо вычислить значение функции). Такие повторяющиеся действия называются *циклами* и реализуются в программе с использованием *инструкций циклов*. В Visual Basic циклические вычисления могут быть реализованы при помощи инструкций `For` (цикл с фиксированным количеством повторений), `Do While` (цикл с предусловием) и `Do Loop...Until` (цикл с постусловием).

Инструкция *For*

Инструкция `For` используется, если некоторую последовательность действий надо выполнить несколько раз, причем число повторений можно задать (вычислить) заранее.

В общем виде инструкция `For` записывается так:

```
For Счетчик = Нач_знач To Кон_знач Step Приращение
    Инструкции
Next Счетчик
```

Здесь:

- *Счетчик* — переменная — счетчик циклов;
- *Нач_знач* — выражение, определяющее начальное значение счетчика циклов;
- *Кон_знач* — выражение, определяющее конечное значение счетчика циклов;
- *Приращение* — выражение, определяющее величину изменения счетчика циклов после каждого выполнения инструкций тела цикла.

> ***ЗАМЕЧАНИЕ***
>
> Если значение приращения равно единице, то слово `Step` и величину приращения можно не указывать.

Выполняется инструкция `For` следующим образом. Сначала переменной *Счетчик* присваивается значение выражения *Нач_знач*. Затем, если значение счетчика меньше значения выражения *Кон_знач*, выполняются инструкции цикла. После этого значение счетчика увеличивается на величину *Приращение*. Если значение счетчика меньше *Кон_знач*, то снова выполняются инструкции цикла. И так до тех пор, пока значение счетчика не превысит значение *Кон_знач* (рис. 3.20).

Инструкции цикла `For` выполняются (*Кон_знач* – *Нач_знач* + 1) раз. Обратите внимание: если начальное значение счетчика превышает указанное конечное значение счетчика, инструкции цикла ни разу не будут выполнены.

В качестве выражений, определяющих начальное и конечное значения счетчика циклов, обычно используют переменные или константы.

В качестве примера использования `For` инструкции в листинге 3.4 приведена процедура обработки события `Click` на кнопке **OK** окна программы "Таблица значений функции". Процедура формирует таблицу значений функции $0.5x^2 + x - 5$ (рис. 3.21).

Обратите внимание, при формировании таблицы вместо функции `Str` используется функция `Format`. Она, как и функция `Str`, преобразует численное значение в строку. У функции два параметра: первый параметр задает значение, которое надо преобразовать в строку, второй — задает формат (вид)

отображения значения. В рассматриваемом примере строка `0.00` указывает, что значение должно быть выведено как минимум с одной цифрой целой части и двумя цифрами дробной.

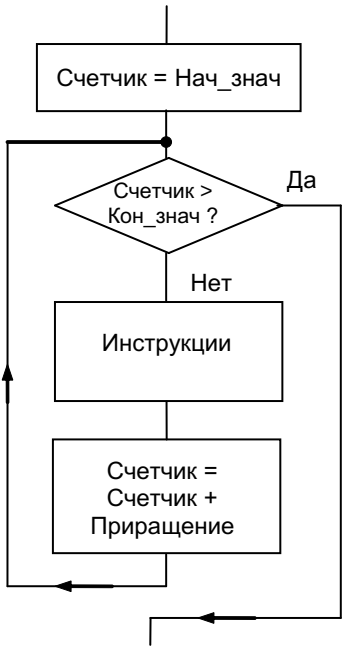


Рис. 3.20. Блок-схема инструкции `For`

Листинг 3.4. Таблица значений функции (цикл `For`)

```
Private Sub Command1_Click()

    Dim x1 As Double ' нижняя граница изменения аргумента
    Dim d2 As Double ' верхняя граница изменения аргумента
    Dim dx As Double ' приращение аргумента

    Dim x As Double  ' значение аргумента
    Dim y As Double  ' значение функции

    Dim st As String ' таблица значений функции
```

```
    Dim n As Integer ' количество точек (строк таблицы)

    Dim i As Integer ' счетчик циклов

    ' ввод исходных данных
    x1 = Val(Text1.Text)
    x2 = Val(Text2.Text)
    dx = Val(Text3.Text)

    n = (x2 - x1) / dx + 1

    st = " X            Y" + vbCrLf ' "шапка таблицы"

    x = x1
    For i = 1 To n
        y = 0.5 * x * x + x - 5
        st = st + Format(x, "0.00") + "    " + Format(y, "0.000") + vbCrLf
        x = x + dx ' увеличить значение аргумента
    Next i

    Label4.Caption = st

End Sub
```

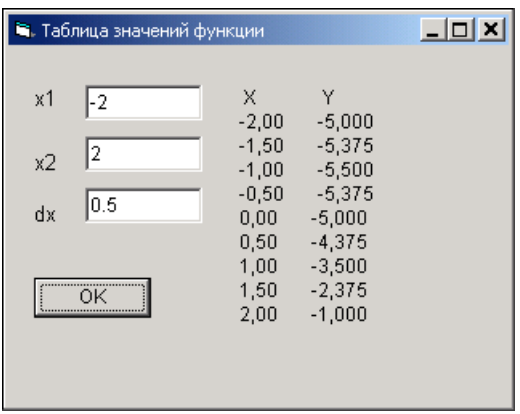


Рис. 3.21. Окно программы "Таблица значений функции"

Контрольные вопросы

1. Какой алгоритмической структуре соответствует инструкция `For`? Изобразите эту структуру.
2. Какого типа должна быть переменная счетчика циклов инструкции `For`?
3. В каком случае величину приращения переменной счетчика циклов можно не указывать?
4. Значение переменной `n` равно 5. Чему будет равно значение переменной `s` после выполнения приведенного далее кода?

```
s = 0
For i=1 To n
    s = s + i
Next i
```

5. Что делает следующий фрагмент кода? Чему будет равно значение переменной `m`, если значение `n` равно нулю?

```
m=1
For i=1 To n
   m = m * 2
Next i
```

Инструкция *Do Loop*

Инструкция `Do Loop` используется для реализации циклов с постусловием. Существуют два варианта этой инструкции: `Do Loop While` и `Do Loop Until`, которые в общем виде записываются так:

Do

Инструкции

Loop While *Условие*

и

Do

Инструкции

Loop Until *Условие*

Здесь:

- *Инструкции* — инструкции, которые надо выполнить несколько раз;
- *Условие* — условие повторения (для цикла `While`) или завершения (для цикла `Until`) цикла.

Выполняется инструкция `Do Loop While` следующим образом. Выполняются инструкции цикла (находящиеся между `Do` и `Loop`). Затем вычисляется значение выражения *Условие*. Если условие истинно (значение выражения равно

`True`), то инструкции цикла выполняются еще раз. И так до тех пор, пока условие выполняется. Если условие не выполняется (значение выражения *Условие* равно `False`), то цикл завершается.

Инструкция `Do Loop Until` выполняется аналогично: инструкции цикла выполняются повторно, если значение выражения *Условие* равно `False`, и цикл завершается, если условие выполняется.

Алгоритмы, реализуемые инструкциями `Do Loop While` и `Do Loop Until`, приведены на рис. 3.22.

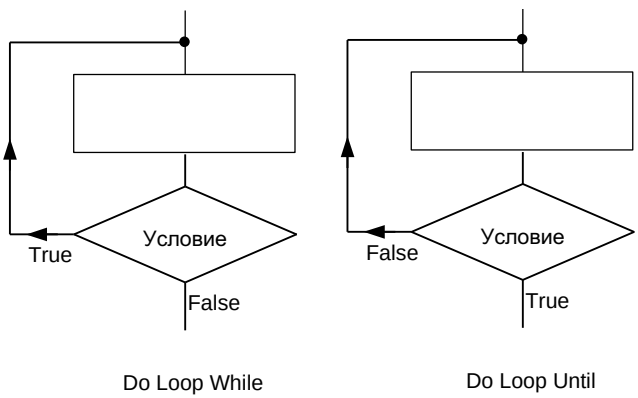


Рис. 3.22. Блок-схема инструкции `Do Loop`

Следует обратить особое внимание на то, что при использовании цикла `Do Loop` возможна ситуация "зацикливания" программы, когда инструкции цикла будут выполняться бесконечно. Обычно это происходит, если инструкции тела цикла не оказывают никакого влияния на условие завершения цикла. В качестве примера в листингах 3.5 и 3.6 приведены варианты процедуры обработки события на кнопке **OK** окна программы "Таблица функции".

Листинг 3.5. Таблица значений функции (цикл `Do Loop While`)

```
Private Sub Command1_Click()

    Dim x1 As Double ' нижняя граница изменения аргумента
    Dim d2 As Double ' верхняя граница изменения аргумента

    Dim dx As Double ' приращение аргумента
```

```
    Dim x As Double  ' значение аргумента
    Dim y As Double  ' значение функции

    Dim st As String ' таблица значений функции

    ' ввод исходных данных
    x1 = Val(Text1.Text)
    x2 = Val(Text2.Text)
    dx = Val(Text3.Text)

    st = " X           Y" + vbCrLf ' "шапка таблицы"

    x = x1
    Do
        y = 0.5 * x * x + x - 5
        st = st + Format(x, "0.00") + "   " + Format(y, "0.000") + vbCrLf
        x = x + dx ' увеличить значение аргумента
    Loop While x <= x2

    Label4.Caption = st
End Sub
```

Листинг 3.6. Таблица значений функции (цикл `Do Loop Until`)

```
Private Sub Command1_Click()

    Dim x1 As Double ' нижняя граница изменения аргумента
    Dim d2 As Double ' верхняя граница изменения аргумента

    Dim dx As Double ' приращение аргумента

    Dim x As Double  ' значение аргумента
    Dim y As Double  ' значение функции

    Dim st As String ' таблица значений функции

    ' ввод исходных данных
    x1 = Val(Text1.Text)
```

```
    x2 = Val(Text2.Text)
    dx = Val(Text3.Text)

    st = " X           Y" + vbCrLf  ' "шапка таблицы"

    x = x1
    Do
        y = 0.5 * x * x + x - 5
        st = st + Format(x, "0.00") + "   " + Format(y, "0.000") + vbCrLf
        x = x + dx  ' увеличить значение аргумента
    Loop Until x > x2

    Label4.Caption = st
End Sub
```

Контрольные вопросы

1. Какой алгоритмической структуре соответствуют инструкции `Do Loop While` и `Do Loop Until`? Изобразите эту структуру.
2. Какие инструкции называют телом цикла?
3. После слова `While` записывается условие завершения или повторения выполнения инструкций тела цикла?
4. После слова `Until` записывается условие завершения или повторения выполнения инструкций тела цикла?
5. Замените приведенный далее цикл `For` на цикл `Do Loop While`.

```
For i=1 To n
     s = s + i
Next i
```

6. Замените приведенный далее цикл `For` на цикл `Do Loop Until`.

```
For i=1 To n
     s = s + i
Next i
```

Инструкция *Do While*

Инструкция `Do While` используется для реализации циклов с предусловием. В общем виде она записывается так:

```
Do While Условие
    Инструкции
Loop
```

Здесь:

- *Инструкции* — инструкции, которые надо выполнить несколько раз;
- *Условие* — условие повторения (для цикла `While`) или завершения (для цикла `Until`) цикла.

Выполняется инструкция `Do While Loop` следующим образом. Сначала вычисляется значение выражения *Условие*. Если условие истинно (`True`), то выполняются инструкции цикла. Затем снова вычисляется значение выражения *Условие*. Если значение равно `True`, то инструкции цикла выполняются еще раз, а если `False`, инструкции цикла не выполняются, и на этом выполнение инструкции `Do While` завершается. Таким образом, инструкции цикла выполняются до тех пор, пока значение выражения *Условие* равно `True` (рис. 3.23).

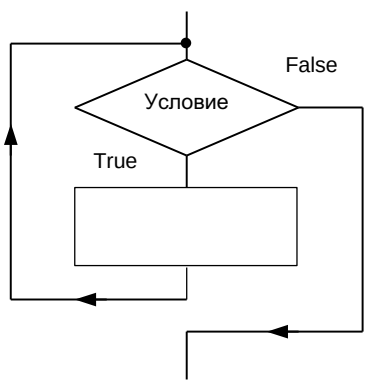


Рис. 3.23. Блок-схема инструкции `Do While`

Следует обратить особое внимание на то, что каждый раз после выполнения инструкций цикла проверяется значение выражения *Условие*. Поэтому, для того чтобы цикл `While` завершился, инструкции цикла обязательно должны влиять на значения переменных, входящих в выражение *Условие*.

В качестве примера в листинге 3.7 приведен вариант процедуры обработки события на кнопке **OK** главного окна программы "Таблица функции".

Листинг 3.7. Таблица значений функции (цикл `Do While`)

```
Private Sub Command1_Click()

    Dim x1 As Double ' нижняя граница изменения аргумента
    Dim d2 As Double ' верхняя граница изменения аргумента
```

```
    Dim dx As Double ' приращение аргумента

    Dim x As Double  ' значение аргумента
    Dim y As Double  ' значение функции

    Dim st As String ' таблица значений функции

    ' ввод исходных данных
    x1 = Val(Text1.Text)
    x2 = Val(Text2.Text)
    dx = Val(Text3.Text)

    st = " X            Y" + vbCrLf ' "шапка таблицы"

    x = x1
    Do While x <= x2
        y = 0.5 * x * x + x - 5
        st = st + Format(x, "0.00") + "   " + Format(y, "0.000") + vbCrLf
        x = x + dx ' увеличить значение аргумента
    Loop

    Label4.Caption = st
End Sub
```

Контрольные вопросы

1. Какой алгоритмической структуре соответствует инструкция `Do While Loop`? Изобразите эту структуру.
2. Возможна ли ситуация, при которой инструкции тела цикла не будут выполнены ни одного раза?
3. Замените приведенный далее цикл `For` на цикл `Do While Loop`.

```
For i=1 To n
     s = s + i
Next i
```

Массивы

Массив — это структура данных, которая представляет собой совокупность переменных одного типа.

Переменные, образующие массив, принято называть элементами массива.

Различают одномерные и двумерные массивы. Графически одномерный массив можно изобразить так, как показано на рис. 3.24.

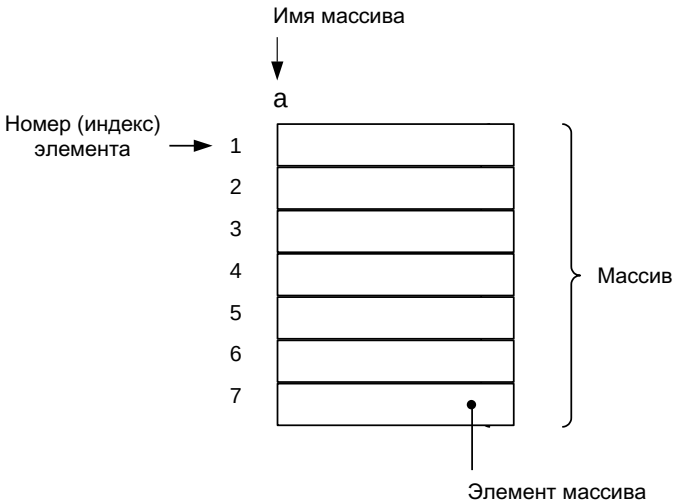


Рис. 3.24. Графическое представление массива

Массивы используют для хранения однородной по своей структуре информации: одномерные — списков, двумерные — таблиц.

Объявление массива

В общем виде объявление одномерного массива выглядит так:

```
Dim Имя(Н To В) As Тип
```

Здесь:

- *Имя* — имя массива;
- *Н* — нижняя граница диапазона изменения индекса;
- *В* — верхняя граница диапазона изменения индекса;
- *Тип* — тип элементов массива.

В инструкции объявления массива в качестве нижней и верхней границ можно использовать только целые константы.

Примеры объявления массивов:

```
Dim A(1 To 10) As Double
Dim Name(1 To 25) As String
Dim B(0 To 15) As Integer
```

В инструкции объявления массива нижнюю границу диапазона изменения индекса можно не указывать. Если нижняя граница не указана, то считается, что она равна нулю. Таким образом, например, объявления

```
Dim B(0 To 10) As Integer
```

и

```
Dim B(10) As Integer
```

эквивалентны.

Доступ к элементу массива

Чтобы получить доступ к элементу массива, надо указать имя массива и номер (индекс) элемента, заключив его в скобки. В качестве индекса следует использовать выражение целого типа (в простейшем случае — константу или переменную). Например:

```
A(1)
A(i)
A(i+)
```

Все операции, которые можно выполнять над переменными, можно выполнять и над элементами массива. Например, элементы массива можно использовать в качестве операндов выражений и условий.

Например:

```
A(3) = 0
Sum = Sum + A(i)
A(i) > A(max)
```

Контрольные вопросы

1. Что такое массив? Дайте определение.
2. Объявите массив, состоящий из 10 элементов целого типа.
3. Объявите массив, позволяющий хранить информацию о дневной температуре воздуха за месяц.
4. Из скольких элементов состоит массив, объявление которого приведено далее?

   ```
   Dim a(10) As Double
   ```

5. Как получить доступ к нужному элементу массива?
6. Выражения какого типа можно использовать в качестве индекса элемента массива?

Ввод массива

Под вводом массива понимается процесс ввода значений всех его элементов. Ввести значения элементов массива можно с клавиатуры (из полей редактирования) или из файла.

В качестве примера, на рис. 3.25 приведена форма программы "Среднее арифметическое", которая вычисляет среднее арифметическое элементов массива. Ввод массива и его обработку выполняет процедура обработки события `Click` на кнопке **OK** (листинг 3.8).

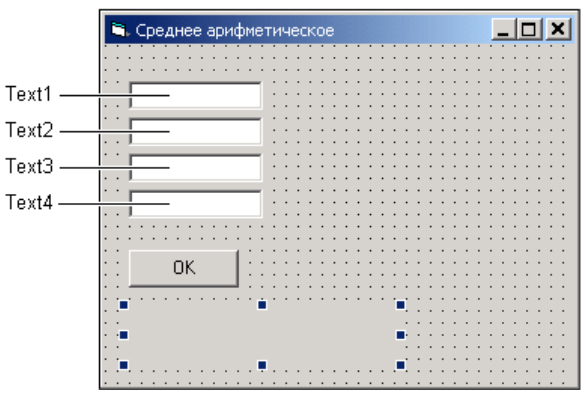


Рис. 3.25. Форма программы "Среднее арифметическое"

Листинг 3.8. Ввод и обработка массива

```
Private Sub Command1_Click()
    Dim A(1 To 4) As Double ' массив
    Dim Sum As Double  ' сумма элементов массива
    Dim M As Double    ' среднее арифметическое элементов

    Dim i As Integer    ' индекс элемента

    ' ввод массива
    A(1) = Val(Text1.Text)
    A(2) = Val(Text2.Text)
    A(3) = Val(Text3.Text)
    A(4) = Val(Text4.Text)
```

```
    ' вычислить сумму элементов
    Sum = 0
    For i = 1 To 4
        Sum = Sum + A(i)
    Next i

    M = Sum / 4 ' среднее арифметическое

    Label1.Caption = "Сумма элементов:" + Str(Sum) + vbCrLf + _
                     "Среднее арифметическое:" + Str(M)

End Sub
```

Процесс ввода элементов массива можно существенно упростить, если объединить в массив поля редактирования формы. Делается это так. Сначала обычным образом надо поместить на форму компонент `TextBox`. Затем в меню **Edit** надо выбрать команду **Copy**. Далее в меню **Edit** следует выбрать команду **Paste** и в появившемся окне в ответ на запрос "You already have a control named "Text1". Do you want to create a control array?" ("Компонент с именем Text1 уже есть. Вы хотите создать массив компонентов?") нажать кнопку **OK**.

В результате описанных действий будет создан массив компонентов `Text1`. Повторив описанные действия несколько раз, можно получить массив требуемого размера. После того как массив компонентов будет создан, с ним можно работать как с обычным массивом. При этом следует обратить внимание на то, что элементы массива компонентов нумеруются с нуля.

На рис. 3.26 приведена форма программы "Среднее арифметическое", поля ввода которой объединены в массив. Листинг 3.9 демонстрирует процесс ввода из полей редактирования, объединенных в массив.

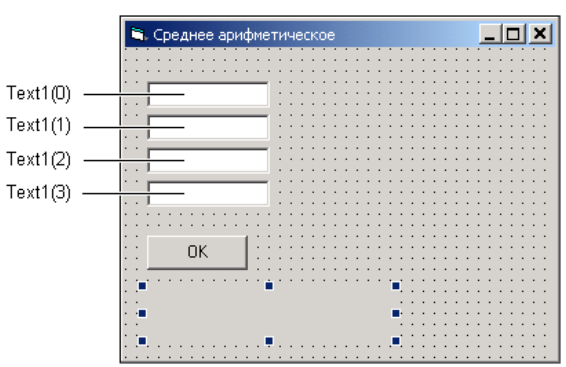


Рис. 3.26. Поля редактирования объединены в массив

Листинг 3.9. Ввод из полей редактирования, объединенных в массив

```
Private Sub Command1_Click()
    ' т. к. элементы массива компонентов нумеруются с нуля,
    ' элементы массива данных тоже будем нумеровать с нуля
    Dim A(0 To 3) As Double ' массив
    Dim Sum As Double       ' сумма элементов массива
    Dim M As Double         ' среднее арифметическое элементов

    Dim i As Integer        ' индекс элемента

    ' ввод массива
    For i = 0 To 3
        A(i) = Val(Text1(i).Text)
    Next i

    ' вычислить сумму элементов
    Sum = 0
    For i = 0 To 3
        Sum = Sum + A(i)
    Next i

    M = Sum / 4 ' среднее арифметическое

    Label1.Caption = "Сумма элементов:" + Str(Sum) + vbCrLf + _
                     "Среднее арифметическое:" + Str(M)
End Sub
```

Вывод массива

Под выводом массива понимается вывод на экран значений всех элементов массива. Наиболее просто вывести массив можно при помощи инструкции `For`, используя счетчик циклов для доступа к элементам массива.

В качестве примера, на рис. 3.27 приведено окно программы "Отклонение от среднего", которая путем обработки массива `A`, введенного пользователем с клавиатуры, формирует массив `B`, элементы которого содержат отклонения от среднего арифметического элементов массива `A` (`B(i)=A(i)-M`, где `M` — среднее арифметическое элементов массива `A`). Ввод массива и его обработку выполняет процедура обработки события `Click` на кнопке **OK** (листинг 3.10).

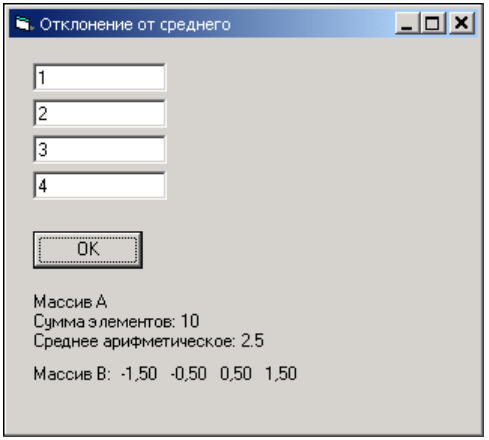


Рис. 3.27. Окно программы "Отклонение от среднего"

Листинг 3.10. Ввод и обработка массива

```
Private Sub Command1_Click()
    ' т. к. элементы массива компонентов нумеруются с нуля,
    ' элементы массива данных тоже будем нумеровать с нуля
    Dim A(0 To 3) As Double ' исходные данные
    Dim Sum As Double       ' сумма элементов массива
    Dim M As Double         ' среднее арифметическое элементов

    Dim b(0 To 3) As Double ' формируемый массив

    Dim i As Integer        ' индекс элемента

    ' ввод исходных данных
    For i = 0 To 3
        A(i) = Val(Text1(i).Text)
    Next i

    ' вычислить сумму элементов
    Sum = 0
    For i = 0 To 3
        Sum = Sum + A(i)
    Next i
```

```
    M = Sum / 4 ' среднее арифметическое

    Label1.Caption = "Массив A" + vbCrLf + _
                     "Сумма элементов:" + Str(Sum) + vbCrLf + _
                     "Среднее арифметическое:" + Str(M)

    ' для каждого элемента массива A вычислить отклонение
    ' от среднего арифметического
    For i = 0 To 3
        b(i) = A(i) - M
    Next i

    Dim st As String ' подстроки этой строки будут
                     ' содержать значения элементов массива B

    ' вывод массива (формирование строки)
    st = "Массив B:  "
    For i = 0 To 3
        st = st + Format(b(i), "0.00") + "   " ' Str(b(i)) + vbCrLf
    Next i

   Label2.Caption = st

End Sub
```

Поиск минимального элемента

Задача поиска минимального элемента массива заключается в определении номера элемента, значение которого не превышает значения остальных элементов массива. Решается задача путем перебора (просмотра) всех элементов массива, т. к. минимальным может быть любой элемент массива.

Алгоритм поиска минимального элемента можно сформулировать так:

- предположим, что первый элемент массива является минимальным;
- будем последовательно сравнивать элементы массива (начиная со второго) с минимальным. Если текущий элемент меньше минимального, то будем считать минимальным этот элемент (запомним его номер).

Программа, реализующая описанный алгоритм, приведена в листинге 3.11, пример ее работы — на рис. 3.28.

Листинг 3.11. Поиск минимального элемента массива

```
Private Sub Command1_Click()
    ' т. к. элементы массива компонентов нумеруются с нуля,
    ' элементы массива данных тоже будем нумеровать с нуля
    Dim A(0 To 4) As Double ' массив

    Dim min As Integer ' номер минимального элемента

    Dim i As Integer        ' индекс элемента

    ' ввод исходных данных
    For i = 0 To 4
        A(i) = Val(Text1(i).Text)
    Next i

    ' поиск минимального элемента
    min = 0 ' предположим, что первый элемент массива минимальный
    For i = 1 To 4
        If A(i) < A(min) Then
            min = i
        End If
    Next i

    Label1.Caption = "Номер минимального элемента: " + Str(min) + _
                     " (нумерация с нуля)" + vbCrLf + _
                     "Минимальный элемент:" + Str(A(min))

End Sub
```

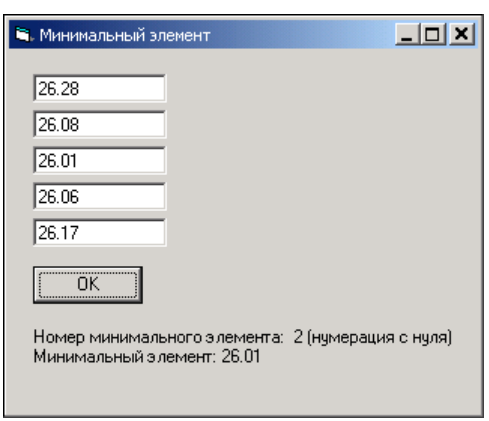


Рис. 3.28. Поиск минимального элемента массива

Сортировка массива

Под сортировкой массива понимается процесс перестановки элементов с целью упорядочивания их в соответствии с каким-либо критерием.

Различают сортировку по возрастанию и убыванию.

Массив, для элементов которого выполняется условие:

```
a(1) ≤ a(2) ≤ ... a(i-1) ≤ a(i) ≤ a(i+1) ... ≤ a(k)
```

называется упорядоченным по возрастанию.

Массив, для элементов которого выполняется условие:

```
a(1) ≥ a(2) ≥ ... a(i-1) ≥ a(i) ≥ a(i+1) ... ≥ a(k)
```

называется упорядоченным по убыванию.

Задача сортировки распространена в информационных системах и часто используется как предварительный этап задачи поиска, т. к. поиск в упорядоченном (отсортированном) массиве можно выполнить значительно быстрее, чем в неупорядоченном.

Существует достаточно много методов сортировки массивов. Здесь мы рассмотрим два:

- ❒ метод прямого выбора;
- ❒ метод обмена.

Сортировка методом прямого выбора

Алгоритм сортировки массива по возрастанию методом прямого выбора может быть представлен так:

1. Просматривая массив от первого элемента, найти минимальный элемент и поместить его на место первого элемента, а первый — на место минимального (поменять элементы местами).
2. Просматривая массив от второго элемента, найти минимальный элемент и поместить его на место второго элемента, а второй — на место минимального.
3. Повторить описанные действия для всех (кроме последнего) оставшихся элементов массива.

Программа сортировки массива по возрастанию представлена в листинге 3.12. Для демонстрации процесса сортировки состояние массива отображается после каждого цикла сортировки (рис. 3.29).

Листинг 3.12. Сортировка массива по возрастанию методом прямого выбора

```
Private Sub Command1_Click()
    ' т. к. элементы массива компонентов нумеруются с нуля,
    ' элементы массива данных тоже будем нумеровать с нуля
    Dim A(0 To 4) As Double ' массив

    Dim i As Integer ' номер элемента, от которого выполняется
                     ' поиск минимального
    Dim j As Integer ' номер элемента, сравниваемого с минимальным

    Dim m As Integer ' номер минимального элемента в части
                     ' массива от i-го элемента

    Dim buf As Double ' буфер, используется для обмена элементов массива

    Dim k As Integer ' счетчик циклов для ввода/вывода

    ' ввод массива
    For k = 0 To 4
        A(k) = Val(Text1(k).Text)
    Next k

    For i = 0 To 3
        ' поиск минимального элемента в части массива
        ' от i + 1 элемента
        min = i ' предположим, что i-й элемент массива минимальный
        For j = i + 1 To 4
            If A(j) < A(min) Then
                min = j
            End If
        Next j
        ' здесь найден минимальный элемент
        ' обменяем местами i-й и минимальный элементы
        buf = A(i)
        A(i) = A(min)
        A(min) = buf

        ' здесь цикл сортировки завершен
```

```
        ' отладочная "печать"
        Dim st As String
        st = ""
        For k = 0 To 4
            st = st + Trim(Str(A(k))) + "   "
        Next k
        Label1.Caption = Label1.Caption + st + vbCrLf

    Next I
    ' здесь массив отсортирован
End Sub
```

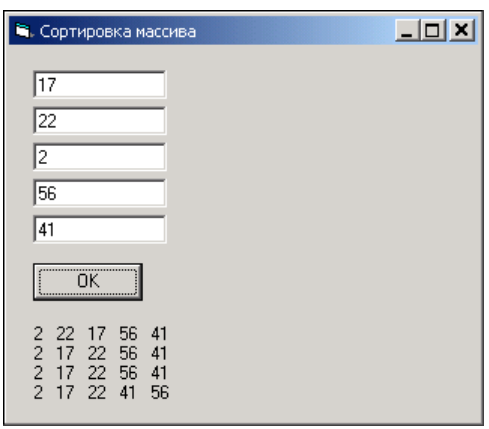


Рис. 3.29. Сортировка массива методом прямого выбора

Сортировка методом "пузырька"

В основе алгоритма лежит идея обмена соседних элементов массива, если следующий элемент меньше предыдущего (при сортировке по возрастанию).

Каждый элемент массива, начиная с первого, сравнивается со следующим, и если он больше следующего, то элементы меняются местами. Таким образом, элементы с меньшим значением продвигаются к началу массива ("всплывают"), а элементы с большим значением — "тонут" (поэтому данный метод сортировки часто называют методом "пузырька"). Чтобы отсортировать массив, описанный выше процесс обменов надо повторить $N - 1$ раз, где N — количество элементов массива.

Рисунок 3.30 показывает процесс сортировки массива методом пузырька. Дуги отмечают элементы, которые следует обменять местами на очередном шаге цикла обменов.

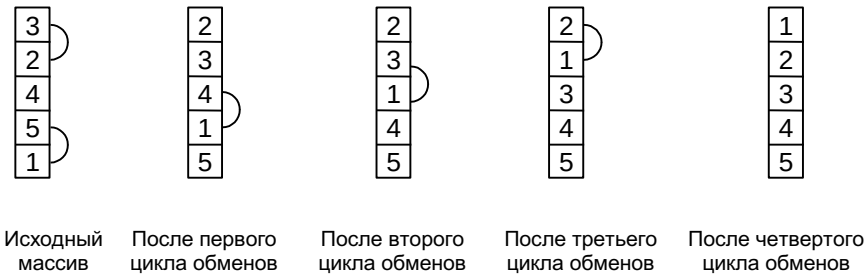


Рис. 3.30. Процесс сортировки массива методом "пузырька"

Программа, реализующая алгоритм сортировки массива по возрастанию методом "пузырька", приведена в листинге 3.13. Для демонстрации процесса сортировки состояние массива отображается после выполнения каждого цикла обменов (рис. 3.31).

Листинг 3.13. Сортировка массива методом "пузырька"

```
Private Sub Command1_Click()
    ' т. к. элементы массива компонентов нумеруются с нуля,
    ' элементы массива данных тоже будем нумеровать с нуля
    Dim A(0 To 4) As Double ' массив

    Dim i As Integer ' счетчик циклов обменов
    Dim j As Integer ' номер сравниваемого элемента:
                     ' j-й сравниваем с j+1

    Dim buf As Double ' буфер (используется для обмена элементов)

    Dim k As Integer ' счетчик циклов для ввода/вывода

    ' ввод массива
    For k = 0 To 4
        A(k) = Val(Text1(k).Text)
    Next k
```

```
    ' *** сортировка по возрастанию***
    ' количество циклов обмена на 1 меньше размера массива
    For i = 1 To 4
        For j = 0 To 3
            If A(j) > A(j + 1) Then
                ' обменять местами A(j) и A(j+1)
                buf = A(j)
                A(j) = A(j + 1)
                A(j + 1) = buf
             End If
        Next j

        ' здесь цикл обменов завершен

        ' отладочная "печать"
        Dim st As String
        st = ""
        For k = 0 To 4
            st = st + Trim(Str(A(k))) + "   "
        Next k

        Label1.Caption = Label1.Caption + st + vbCrLf

    Next i

End Sub
```

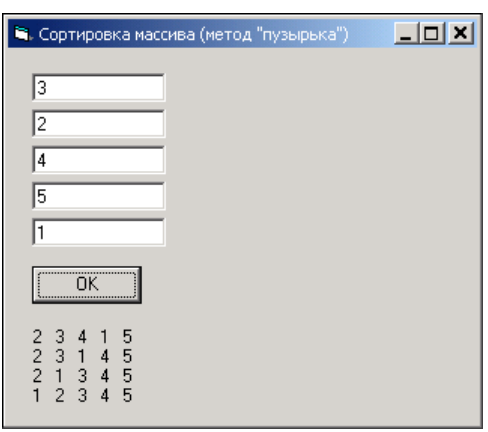


Рис. 3.31. Сортировка массива методом "пузырька"

Поиск в массиве

При решении многих задач часто возникает необходимость установить, содержит ли массив определенную информацию или нет. Например, проверить, есть ли в массиве фамилий фамилия Петров. Задачи такого типа называются поиском в массиве.

Метод перебора

Для организации поиска в массиве могут быть использованы различные алгоритмы. Наиболее простой — это алгоритм простого перебора. Поиск осуществляется последовательным сравнением элементов массива с образцом до тех пор, пока не будет найден элемент, равный образцу, или не будут проверены все элементы. Алгоритм простого перебора применяется, если элементы массива не упорядочены.

В листинге 3.14 приведен текст программы поиска в массиве целых чисел методом перебора элементов. Перебор элементов массива осуществляется в цикле `Do Loop Until`, в теле которого инструкция `If` сравнивает текущий элемент массива с образцом. Если текущий элемент равен образцу, то переменной `found` присваивается значение `True`. Цикл завершается, если в массиве обнаружен нужный элемент (в этом случае значение переменной `found` равно `True`), или если проверены все элементы массива (в этом случае значение счетчика циклов больше количества элементов массива). По завершении цикла, проверив значение переменной `found`, можно определить, успешен поиск или нет. Пример работы программы показан на рис. 3.32.

Листинг 3.14. Поиск в массиве (метод перебора)

```
Private Sub Command1_Click()
    ' т. к. элементы массива компонентов нумеруются с нуля,
    ' элементы массива данных тоже будем нумеровать с нуля
    Dim A(0 To 4) As Double ' массив
    Dim obr As Double       ' образец
    Dim found As Boolean    ' True - нужный элемент найден

    Dim i As Integer

    ' ввод массива
    For i = 0 To 4
        A(i) = Val(Text1(i).Text)
    Next i
```

```
    obr = Val(Text2.Text)

    found = False ' пусть нужного элемента в массиве нет
    i = 0 ' номер проверяемого элемента массива
    Do
        If A(i) = obr Then
            ' нужный элемент найден
            found = True
        Else
            i = i + 1
        End If
    Loop Until (found = True) Or (i > 4)

    If found = True Then
        Label1.Caption = "Совпадение с элементом номер" + Str(i) + _
                         vbCrLf + "(нумерация с нуля)"
    Else
        Label1.Caption = "Нужного элемента в массиве нет"
    End If

End Sub
```

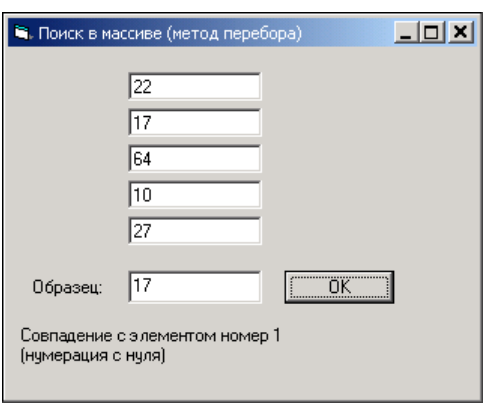


Рис. 3.32. Поиск в массиве методом перебора

Алгоритм перебора на практике применяется редко, т. к. его эффективность невысока: если массив неупорядочен, то нужный элемент может быть как в начале, так и в конце массива. Очевидно, чем больше массив, тем в среднем дольше программа будет искать нужный элемент.

Бинарный поиск

На практике часто приходится иметь дело с информацией, которая упорядочена по некоторому критерию. Например, список фамилий, как правило, упорядочен по алфавиту, массив метеорологических данных — по датам наблюдений. Если информация упорядочена, то можно сделать предположение, в какой части списка (ближе к началу или ближе к концу) она находится, и искать ее именно там.

Для поиска в упорядоченных массивах применяют другие, более эффективные по сравнению с методом простого перебора, алгоритмы, один из которых — метод бинарного поиска.

Суть метода бинарного поиска заключается в следующем. Выбирается средний (по номеру) элемент упорядоченного массива (элемент с номером `m`), и образец сравнивается с этим элементом (рис. 3.33).

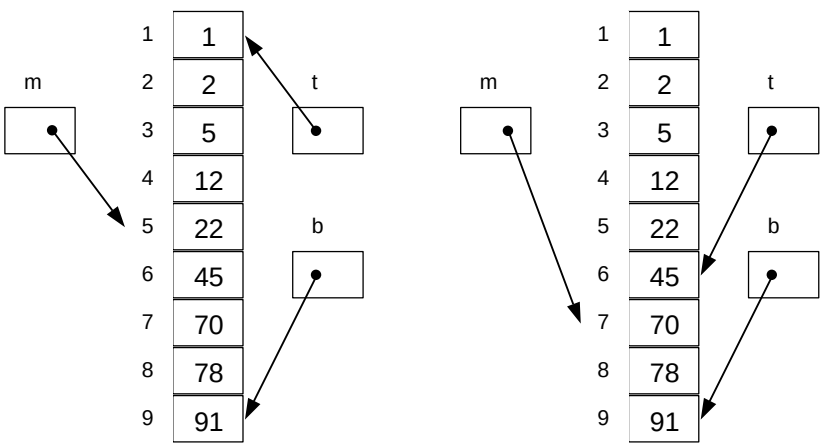


Рис. 3.33. Выбор среднего элемента массива при бинарном поиске

Если средний элемент равен образцу, то задача решена. Если образец меньше среднего элемента (предполагается, что массив упорядочен по возрастанию), то искомый элемент расположен выше (до) среднего элемента (между элементами с номерами `t` и `m-1`). Если образец больше среднего элемента, то искомый элемент расположен ниже (после) среднего (между элементами с номерами `m+1` и `b`). После того как будет определена часть массива, в которой может находиться искомый элемент, поиск проводят в этой части. Номер среднего элемента вычисляется по формуле `(b – t) / 2 + t`.

Алгоритм бинарного поиска в упорядоченном массиве представлен на рис. 3.34, программа — в листинге 3.15.

В программу добавлены инструкции вывода значений переменных `t`, `b` и `m`. Эта информация полезна для понимания сути алгоритма. Пример работы программы приведен на рис. 3.35.

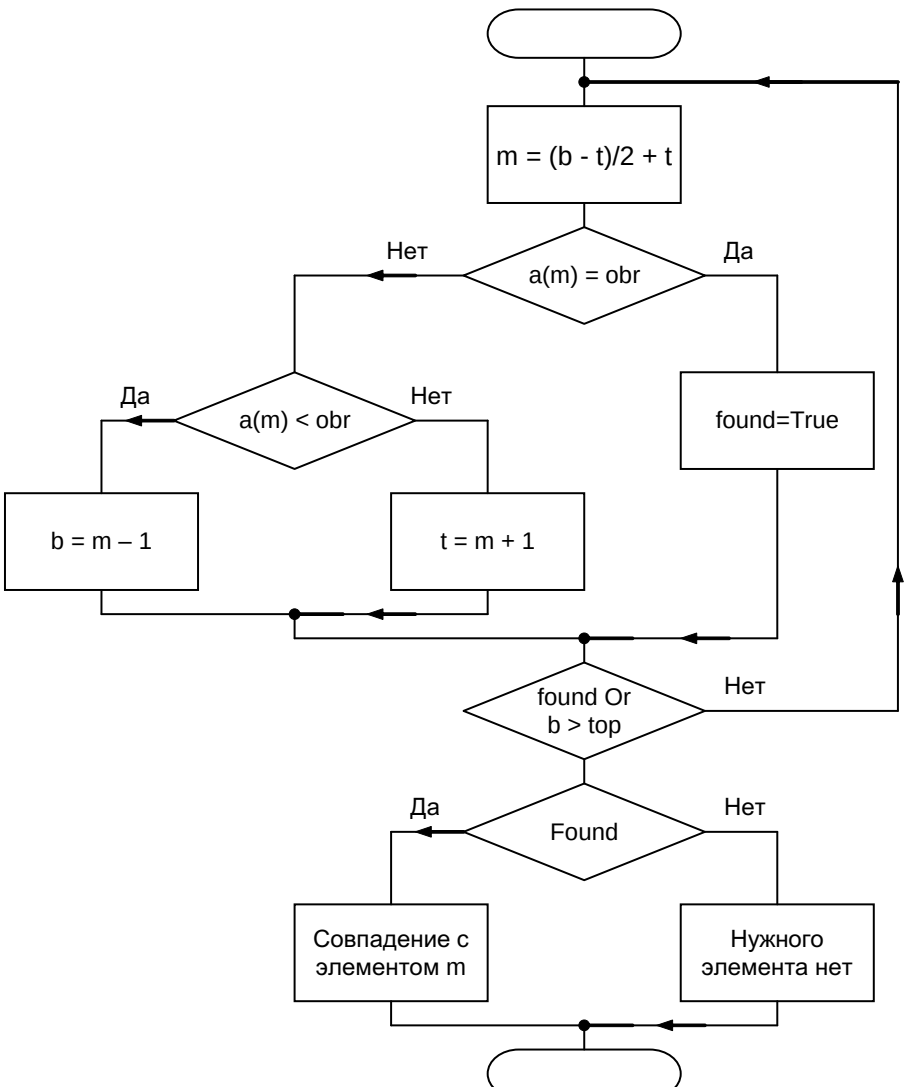


Рис. 3.34. Алгоритм бинарного поиска в упорядоченном по возрастанию массиве

Листинг 3.15. Бинарный поиск в упорядоченном массиве

```
Private Sub Command1_Click()
    Dim A(0 To 8) As Double ' массив
    Dim obr As Double       ' образец
    Dim found As Boolean    ' True - нужный элемент найден

    Dim t, b, m As Integer ' номер верхнего, нижнего и среднего элементов
    Dim k As Integer

    Dim i As Integer

    ' ввод массива
    For i = 0 To 8
        A(i) = Val(Text1(i).Text)
    Next i

    obr = Val(Text2.Text)

    t = 0
    b = 8
    found = False
    k = 0

    Label1.Caption = " t   b   m" + vbCrLf
    Do
       m = (b - t) / 2 + t
       Label1.Caption = Label1.Caption + _
                        Str(t) + "  " + Str(b) + "  " + Str(m) + vbCrLf

        k = k + 1
        If A(m) = obr Then
              found = True
        Else
            If obr < A(m) Then
                b = m - 1
            Else
                t = m + 1
            End If
        End If
    Loop Until (found = True) Or (t > b)
```

```
    If found = True Then
        Label1.Caption = Label1.Caption + vbCrLf + _
                         "Совпадение с элементом " + Str(m) + _
                         " (нумерация с нуля)" + vbCrLf + _
                         "Сравнений:" + Str(k)
    Else
        Label1.Caption = Label1.Caption + vbCrLf + _
                         "Нужного элемента в массиве нет"
    End If

End Sub
```

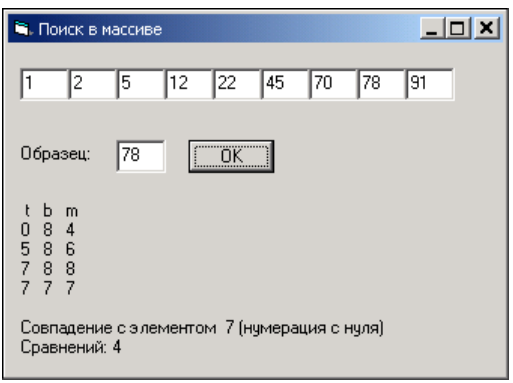


Рис. 3.35. Бинарный поиск в упорядоченном массиве

Контрольные вопросы

1. Что делает следующий фрагмент кода?

```
For i = 1 To 10
    a(i) = 0
Next i
```

2. Что делает следующий фрагмент кода?

```
For i  = 1 To 10
    s = s + a(i)
Next i
m = s / 10
```

3. Какие методы сортировки массивов вы знаете?
4. Какие методы поиска в массиве вы знаете? В какой ситуации следует применять каждый из них?

Многомерные массивы

Исходные данные для решения многих задач часто представляют в табличной форме. Например, результат продаж автомобилей за год можно представить в виде следующей таблицы:

	I кв.	II кв.	III кв.	IV кв.
Модель_1				
Модель_2				
Модель_3				
Модель_4				
Модель_5				

В программе приведенную таблицу можно представить как совокупность массивов, например, так:

```
Dim m1(1 To 4) As Integer
Dim m5(1 To 4) As Integer
Dim m3(1 To 4) As Integer
Dim m4(1 To 4) As Integer
Dim m5(1 To 4) As Integer
```

При такой записи предполагается, что каждый из массивов хранит информацию о количестве проданных автомобилей одной марки.

Возможно и такое представление таблицы:

```
Dim q1(1 To 5) As Integer
Dim q2(1 To 5) As Integer
Dim q3(1 To 5) As Integer
Dim q4(1 To 5) As Integer
```

В этом случае каждый массив хранит информацию о количестве проданных автомобилей всех моделей, соответственно, в первом, втором, третьем и четвертом кварталах.

Если таблица содержит данные одного типа, то ее можно представить как двумерный массив.

В общем виде объявление двумерного массива выглядит так:

```
Dim Имя(H1 To B1, H2 To B2) As Тип
```

Здесь *Имя* — имя массива; *H1*, *B1* и *H2*, *B2* — целые константы, определяющие, соответственно, диапазон изменения индекса строки и столбца; *Тип* — тип элементов массива.

Приведенную ранее таблицу продаж автомобилей можно представить в виде двумерного массива так:

```
Dim A(1 To 5, 1 To 4) As Integer
```

Количество элементов двумерного массива можно вычислить по формуле:

```
(B1 - H1 + 1) * (B2 - H2 + 1)
```

Таким образом, массив `A` состоит из 20 элементов типа `Integer`.

Чтобы получить доступ к элементу двумерного массива, нужно указать имя массива и индексы элемента, заключив их в скобки. Первый индекс соответствует номеру строки таблицы, второй — столбца. Так, например, элемент `A(2,3)` содержит данные о продажах автомобилей модели Модель_2 в третьем квартале.

Значения элементов двумерных массивов выводят на экран (например, в поле компонента `Label`), как правило, по строкам, т. е. сначала все элементы первой строки, затем второй и т. д. Это можно сделать при помощи вложенных инструкций `For`. Следующий фрагмент кода формирует строку, которая содержит изображение двумерного массива:

```
For i = 1 To 5
    For j = 1 To 4
        st = st + Str(A(i,j)) + "   "
    Next j
    st = st + vbCrLf
Next i
```

В приведенном фрагменте каждый раз, когда внутренний цикл завершается, внешний цикл увеличивает `i` на единицу, и внутренний цикл выполняется снова. Таким образом, все компоненты массива `A` выводятся в следующем порядке: `A(1,1)`, `A(1,2)`, `A(1,3)`, `A(1,4)`, `A(2,1)`, `A(2,2)`, `A(2,3)`, `A(2,4)`, `A(3,1)` и т. д.

В качестве примера рассмотрим следующую задачу. Пусть есть данные о продажах автомобилей, и надо посчитать: общее количество проданных автомобилей, количество автомобилей, проданных в каждом квартале, и количество автомобилей каждой марки, проданных за год. Другими словами, надо найти сумму элементов массива, сумму элементов по столбцам и по строкам.

Результат обработки таблицы и исходные данные удобно объединить в одну таблицу:

	I кв.	II кв.	III кв.	IV кв.	Всего
Модель_1					
Модель_2					
Модель_3					
Модель_4					
Модель_5					
Всего					

В программе приведенную таблицу можно представить так:

```
Dim A(1 To 6, 1 To 5) As Integer
```

Текст программы, которая решает поставленную задачу, приведен в листинге 3.16, ее форма и окно — соответственно на рис. 3.36 и 3.37.

Листинг 3.16. Пример работы с двумерным массивом

```
Private Sub Command1_Click()

    ' массив на одну строку и один столбец
    ' больше таблицы
    Dim A(1 To 6, 1 To 5) As Integer

    Dim i As Integer  ' номер строки
    Dim j As Integer  ' номер столбца

    Dim k  As Integer ' номер компонента TextBox

    ' *** Ввод двумерного (5 строк, 4 столбца) массива ***
    ' Исходные данные находятся в одномерном
    ' массиве компонентов TextBox

    k = 0 ' элементы массива компонентов нумеруются с нуля
    For i = 1 To 5
```

```
    ' ввод i-ой строки
    For j = 1 To 4
        A(i, j) = Val(Text1(k))
        k = k + 1 ' следующий TextBox
    Next j
Next i

' *** обработка массива ***

Dim sum As Integer ' сумма элементов строки, столбца или массива

' Сумма элементов по строкам.
' Результат запишем в последний (5-й) столбец
For i = 1 To 5
    sum = 0
    For j = 1 To 4
        sum = sum + A(i, j)
    Next j
    A(i, 5) = sum
Next i

' Сумма элементов по столбцам.
' Результат запишем в последнюю (6-ю) строку
For j = 1 To 4
    sum = 0
    For i = 1 To 5
        sum = sum + A(i, j)
    Next i
    A(6, j) = sum
Next j

' Сумма элементов массива.
' Можно сложить значения элементов 5-го столбца,
' 6-й строки или 1..4 элементы 1..5 строк.
' Реализуем последний способ
sum = 0
For i = 1 To 5
```

```
        For j = 1 To 4
            sum = sum + A(i, j)
        Next j
    Next i
    A(6, 5) = sum

    ' вывод массива
    Dim st As String

    st = vbNewLine
    For i = 1 To 6
        For j = 1 To 5
            st = st + Format(A(i, j), "@@@@@")
       Next j
       st = st + vbNewLine
    Next i

    Label1.Caption = st

End Sub
```

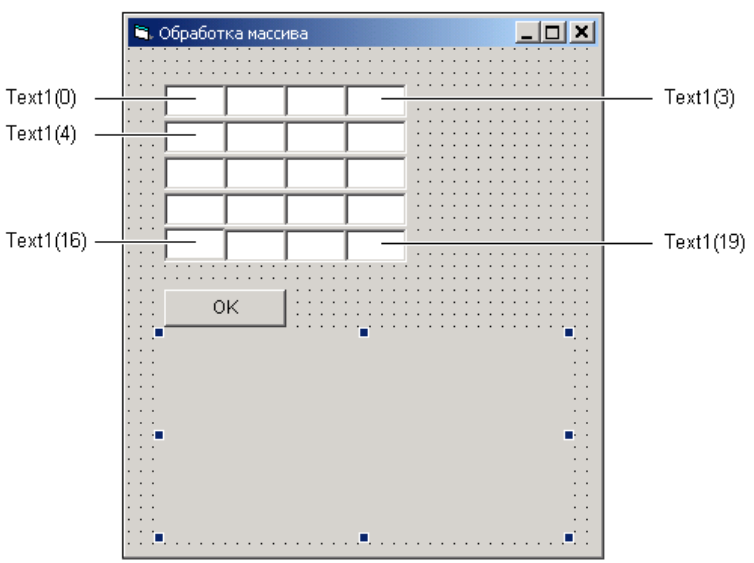


Рис. 3.36. Форма программы "Обработка двумерного массива"

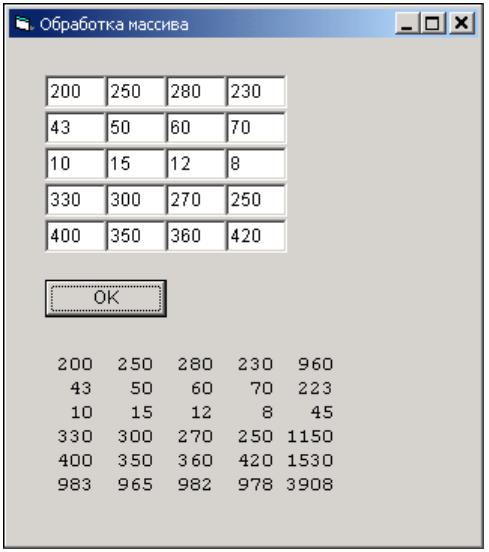


Рис. 3.37. Результат обработки массива

Следующая программа, ее текст приведен в листинге 3.17, после вычисления суммы элементов строк, сортирует массив (строки таблицы) по содержимому 5-го столбца, хранящего суммы элементов строк. Сортировка выполняется методом "пузырька". В качестве буфера обмена используется последняя (дополнительная) строка массива (количество строк и столбцов в массиве на единицу больше, чем количество строк и столбцов исходной таблицы). Пример работы программы приведен на рис. 3.38.

Листинг 3.17. Сортировка двумерного массива

```
Private Sub Command1_Click()

    ' массив на одну строку и один столбец
    ' больше таблицы
    Dim A(1 To 6, 1 To 5) As Integer

    Dim i As Integer ' номер строки
    Dim j As Integer ' номер столбца

    Dim k  As Integer ' номер компонента TextBox

    ' *** Ввод двумерного (5 строк, 4 столбца) массива ***
```

```
' Исходные данные находятся в одномерном
' массиве компонентов TextBox

k = 0 ' элементы массива компонентов нумеруются с нуля
For i = 1 To 5
    ' ввод i-ой строки
    For j = 1 To 4
        A(i, j) = Val(Text1(k))
        k = k + 1 ' следующий TextBox
    Next j
Next i

' *** обработка массива ***

Dim sum As Integer ' сумма элементов строки

' Сумма элементов по строкам.
' Результат запишем в последний (5-й) столбец
For i = 1 To 5
    sum = 0
    For j = 1 To 4
        sum = sum + A(i, j)
    Next j
    A(i, 5) = sum
Next i

' *** сортировка массива по содержимому столбца "Всего"
For i = 1 To 4 ' кол-во циклов обменов на 1 меньше кол-ва строк
    For j = 1 To 4 ' кол-во обменов на 1 меньше кол-ва строк
        If A(j, 5) < A(j + 1, 5) Then
            ' обменять j-ю и j+1 строки
            ' 6-я строка - буфер
                For k = 1 To 5 ' в строке 5 ячеек
                    A(6, k) = A(j, k)
                    A(j, k) = A(j + 1, k)
                    A(j + 1, k) = A(6, k)
```

```
                    Next k
                End If
            Next j
        Next i

        ' вывод массива
        Dim st As String

        st = vbNewLine
        For i = 1 To 5
            For j = 1 To 5
                st = st + Format(A(i, j), "@@@@@")
           Next j
           st = st + vbNewLine
        Next i

        Label1.Caption = st

End Sub
```

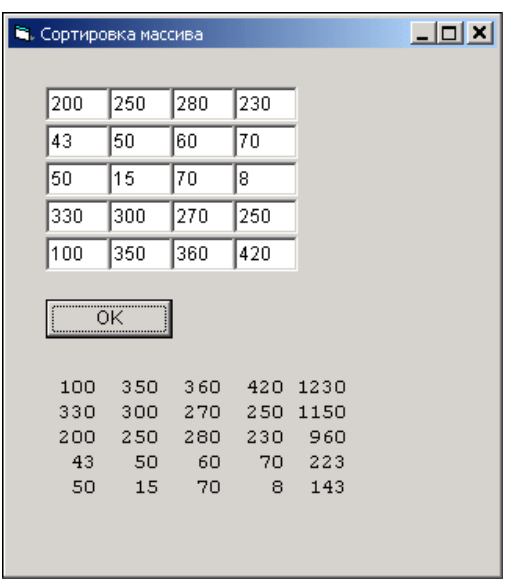


Рис. 3.38. Сортировка двумерного массива

Ошибки при работе с массивами

При работе с массивами наиболее распространенной ошибкой, которая проявляется во время работы программы, является обращение к несуществующему элементу. Причиной этой ошибки является выход значения индексного выражения за допустимые границы, указанные в инструкции объявления массива.

При возникновении этой ошибки программа приостанавливает свою работу и на экране появляется окно с сообщением о возникшей ошибке (рис. 3.39). Щелчок на кнопке **End** завершает выполнение программы, а на кнопке **Debug** — активизирует режим откладки (если программа запущена из среды разработки) и в окне редактора кода цветом выделяется инструкция, во время выполнения которой возникла ошибка (рис. 3.40).

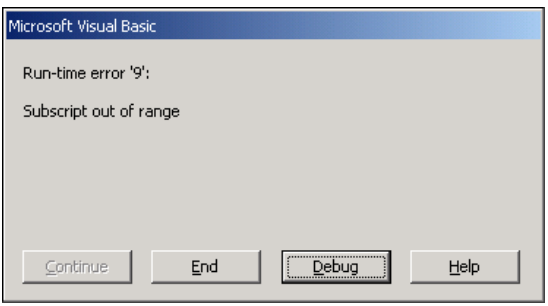


Рис. 3.39. Сообщение об ошибке при попытке обращения к несуществующему элементу массива

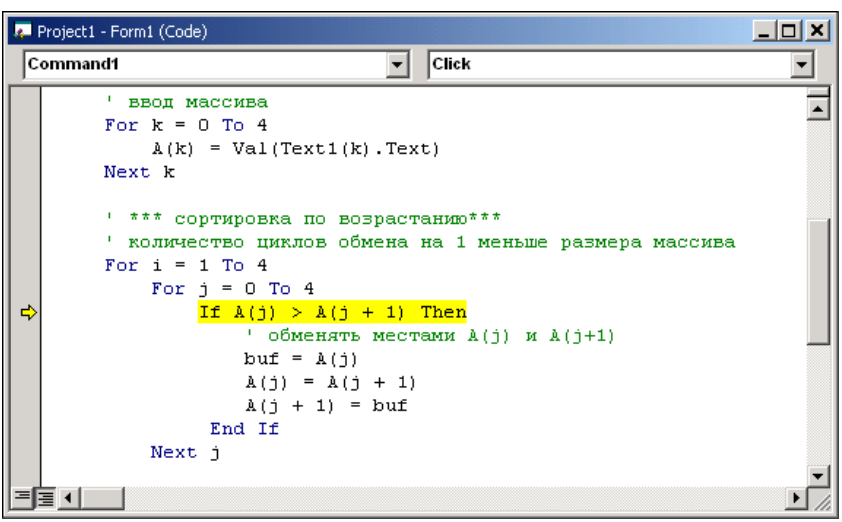


Рис. 3.40. Инструкция, при выполнении которой произошла ошибка, выделяется цветом

В качестве примера рассмотрим программу, приведенную в листинге 3.18. Синтаксических ошибок в ней нет. Однако во время работы программы возникает ошибка. Причина ее в том, что в качестве границы цикла `For` указана константа 4. Если значение переменной `j` равно 4, то в инструкции `If` делается попытка сравнить четвертый и несуществующий пятый элементы массива (см. объявление массива). Чтобы устранить эту ошибку, в качестве верхней границы изменения индекса надо указать 3.

Листинг 3.18. Программа, во время работы которой возникает ошибка

```
' Сортировка массива методом "пузырька"
' !!! В программе есть ошибка!!!
' Демонстрирует возникновение ошибки во время выполнения программы.
Private Sub Command1_Click()
    ' т. к. элементы массива компонентов нумеруются с нуля,
    ' элементы массива данных тоже будем нумеровать с нуля
    Dim A(0 To 4) As Double ' массив

    Dim i As Integer ' счетчик циклов обменов
    Dim j As Integer ' номер сравниваемого элемента:
                     ' j-й сравниваем с j+1

    Dim buf As Double ' буфер (используется для обмена элементов)

    Dim k As Integer ' счетчик циклов для ввода/вывода

    ' ввод массива
    For k = 0 To 4
        A(k) = Val(Text1(k).Text)
    Next k

    ' *** сортировка по возрастанию***
    ' количество циклов обмена на 1 меньше размера массива
    For i = 1 To 4
        For j = 0 To 4
            If A(j) > A(j + 1) Then
                ' обменять местами A(j) и A(j+1)
```

```
                buf = A(j)
                A(j) = A(j + 1)
                A(j + 1) = buf
            End If
        Next j

        ' здесь цикл обменов завершен

        ' отладочная "печать"
        Dim st As String
        st = ""
        For k = 0 To 4
            st = st + Trim(Str(A(k))) + "   "
        Next k

        Label1.Caption = Label1.Caption + st + vbCrLf

    Next i

End Sub
```

Контрольные вопросы

1. Во время работы программы произошла ошибка, и на экране появилось сообщение "Subscript out of range". Укажите наиболее вероятную причину ее возникновения.
2. Объявите массив, соответствующий прямоугольной матрице 5×4 (5 строк, 4 столбца).
3. При обращении к элементу двумерного массива первый индекс задает номер строки или столбца?

Функция программиста

Программист может объявить собственную функцию и в дальнейшем использовать ее точно так же, как и любую стандартную. Например, можно определить (объявить) функцию вычисления факториала, назвав ее `Factorial`. Затем в том месте программы, где нужно вычислить факториал вместо последовательности инструкций, делающих это, написать `k = Factorial(n)`.

Объявление функции

Объявление функции в общем виде выглядит так:

```
Function Имя(Параметры) As Тип
    ' здесь инструкции, реализующие функцию
    Имя = Значение
End Function
```

Здесь:

- `Function` — зарезервированное слово языка Visual Basic, показывающее, что далее следуют инструкции, реализующие функцию;
- *Имя* — имя функции;
- *Параметры* — список переменных, которые используются для передачи в функцию информации, необходимой для вычисления значения функции;
- *Тип* — тип значения функции.

Следует обратить внимание, что функция завершается инструкцией, которая присваивает значение идентификатору *Имя*. Именно эта инструкция и определяет значение функции.

В качестве примера в листинге 3.19 приведена функция `Factorial`, которая вычисляет факториал числа, указанного в качестве ее параметра.

Листинг 3.19. Функция вычисления факториала

```
Function Factorial(x As Integer) As Long
    Dim f As Long
    Dim i As Integer

    f = 1
    For i = 1 To x
        f = f * i
    Next i

    Factorial = f
End Function
```

У функции `Factorial` один параметр — переменная `x` типа `Integer`. Параметр задает число, факториал которого надо вычислить. Конкретное значение `x` получит при вызове функции. Возвращает функция вычисленное значение факториала — значение типа `Long`.

Другой пример. В Visual Basic нет функции вычисления кубического корня. В листинге 3.20 приведен вариант ее реализации.

Листинг 3.20. Функция вычисления кубического корня

```
' функция Cubrt вычисляет кубический корень
Function Cubrt(x As Double) As Double
    Dim pr As Double ' приближенное значение кубического корня

    Dim negative As Boolean ' True - корень из отрицательного числа

    If x < 0 Then
        negative = True
        x = Abs(x)
    Else
        negative = False
    End If

    If (x > 0) Then
        pr = Sqr(x) ' первое приближение - квадратный корень
        Do While Abs(pr - x / (pr * pr)) > 0.001

          ' новое приближение - среднее арифметическое
          ' удвоенного приближения на прошлом шаге и текущего
          pr = (2 * pr + x / (pr * pr)) / 3
        Loop

        ' если аргумент отрицательное число, то
        ' корень - тоже отрицательное число
        If negative Then
            pr = -pr
        End If

        Cubrt = pr
     Else
        Cubrt = 0
     End If
End Function
```

Функция `Cubrt` реализует алгоритм "ручного" метода приближенного вычисления кубического корня, при котором в качестве первого приближения, "на глаз", выбирается наиболее подходящее число (в программе — это значение квадратного корня числа, кубический корень которого надо вычислить). Затем число, из которого надо извлечь корень, делится на выбранное приближение, и полученное таким образом значение еще раз делится на выбранное

приближение. Если полученное число отличается от выбранного приближения на величину, не большую, чем допустимое значение погрешности, то выбранное приближение принимается за значение корня. Если погрешность превышает допустимое значение, то вычисляется новое приближение как среднее арифметическое удвоенного предыдущего приближения и нового вычисленного приближения, и процесс вычисления повторяется.

Использование функции

Для того чтобы функцию, созданную программистом, можно было использовать, ее (в простейшем случае) следует поместить в текст программы, перед функцией или процедурой, которая ее использует.

Функция будет выполнена, если в каком-либо из выражений программы в качестве операнда будет указано ее имя.

Если в объявлении функции указаны параметры (эти параметры называются формальными), то в инструкции вызова функции также должны быть указаны параметры (эти параметры называются фактическими), причем количество и тип фактических параметров должно соответствовать количеству и типу формальных параметров. В качестве фактических параметров обычно используют переменные или константы, реже — выражения.

Следующая программа, модуль ее формы приведен в листинге 3.21, демонстрирует использование функции `Factorial`. Она выводит таблицу факториалов (рис. 3.41). Следует обратить внимание, значение факториала числа 13 больше максимального значения типа `Long`, поэтому при попытке вычислить значение факториала 13 возникает ошибка "Owerflow" ("Переполнение").

Листинг 3.21. Модуль формы программы "Таблица факториалов"

```
' Функция Factorial вычисляет факториал числа x
Function Factorial(x As Integer) As Long
    Dim f As Long
    Dim i As Integer

    f = 1
    For i = 1 To x
        f = f * i
    Next i

    Factorial = f
End Function
```

```
' строит таблицу факториалов
Private Sub Command1_Click()
    Dim i As Integer
    Dim f As Long     ' факториал

    Dim st As String ' таблица факториалов

    st = Trim(" ")

    For i = 1 To 12
        f = Factorial(i)
        st = st + Str(i) + " - " + Str(f) + vbNewLine
    Next i

    Label1.Caption = st
End Sub
```

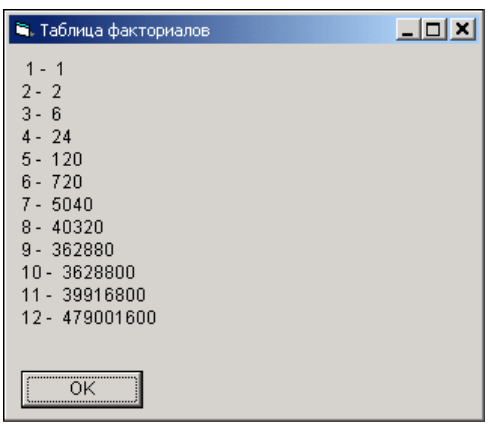
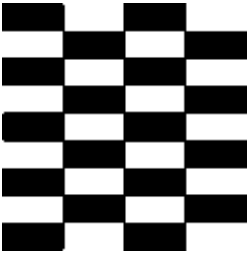


Рис. 3.41. Таблица факториалов

Контрольные вопросы

1. Что такое функция?
2. Для чего используются параметры функции?
3. Как вызвать функцию?
4. Объявите (напишите) функцию `FuntToKg`, выполняющую пересчет веса из фунтов в килограммы.



Глава 4

Базовые компоненты

В этой главе демонстрируется назначение и использование базовых компонентов. Следует обратить внимание, что продемонстрировать назначение и возможности какого-либо компонента практически не возможно, не используя другие компоненты. Поэтому в данной главе принят следующий подход. Основное внимание уделяется компоненту, название которого вынесено в заголовок раздела. Вспомогательные компоненты при этом подробно не рассматриваются. В описании компонентов уделено внимание тем свойствам и методам, которые отражают специфику компонента и представляют для программиста наибольший интерес (описание других свойств можно найти в справочной системе).

К базовым можно отнести компоненты: `Label`, `TextBox`, `CommandButton`, `CheckBox`, `OptionButton`, `ComboBox`, `ListBox`, `PictureBox`, т. е. компоненты, которые используются почти в каждом приложении. Эти компоненты доступны (подключаются) автоматически, и, что немаловажно, программе, которая использует только эти компоненты, не нужны никакие дополнительные библиотеки (кроме стандартных).

Label

Компонент `Label` (рис. 4.1) предназначен для отображения текстовой информации.

Рис. 4.1. Компонент `Label`

Задать текст, отображаемый в поле компонента, можно как во время разработки формы (задав значение свойства `Caption` в окне **Properties**), так и во время работы программы, присвоив значение свойству `Caption`. Свойства компонента `Label` приведены в табл. 4.1.

Таблица 4.1. *Свойства компонента* `Label`

Свойство	Описание
`Caption`	Отображаемый текст
`Width`, `Height`	Размер (ширина, высота) компонента
`Top`, `Left`	Координаты (положение) компонента — расстояние от верхней и левой границы компонента до, соответственно, верхней и левой границы клиентской (внутренней) области формы
`Font`	Шрифт, используемый для отображения текста
`ForeColor`	Цвет символов, отображаемых в поле компонента
`BackColor`	Цвет фона поля компонента
`AutoSize`	Признак автоматического изменения размера поля компонента в соответствии с его содержимым (значением свойства `Caption`)
`WordWrap`	Признак разбиения текста, отображаемого в поле компонента, на строки (значение свойства `AutoSize` должно быть `False`)
`Alignment`	Задает способ выравнивания текста внутри поля. Текст может быть выровнен по левому краю (0), по центру (2) или по правому краю (1)
`BorderStyle`	Стиль границы поля компонента. Граница есть — 1, границы нет — 0. Вид границы определяет значение свойства `Appearance`
`Appearance`	Вид границы компонента (если значение свойства `BorderStyle` равно 1). Граница может представлять собой тонкую (0) или "объемную" (1) линию
`BackStyle`	Управляет отображением фона области вывода текста. Область вывода текста может быть закрашена цветом, заданным свойством `BackColor` (в этом случае значение свойства должно быть равно 1), или быть прозрачной (0)
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

Если в поле компонента `Label` надо ввести числовое значение, например результат расчета, то это значение надо сначала преобразовать в строку (функция `Str` или `Format`), а затем записать полученную строку в свойство `Caption`.

Если некоторую информацию надо вывести в поле компонента Label, причем так, чтобы каждый ее элемент был размещен в отдельной строке, то после каждого элемента надо поместить символ "новая строка" (его код — 13). Например, в результате выполнения инструкции

```
Label.Caption = "Сумма: " + Str(sum) + "руб." + Chr(13) + _
                "Скидка: " + Str(discount) + "руб." + Chr(13) + _
                "К оплате: " + Str(total)
```

в поле компонента Label появятся три строки текста.

Вместо функции Chr(13) можно использовать именованную константу vbCR. Например, приведенную выше инструкцию можно записать так:

```
Label.Caption = "Сумма: " + Str(sum) + "руб." + vbCr + _
                "Скидка: " + Str(discount) + "руб." + vbCr + _
                "К оплате: " + Str(total)
```

Использование компонента Label демонстрирует программа "Вода" (ее форма приведена на рис. 4.2, значения свойств компонентов — в табл. 4.2, а текст — в листинге 4.1).

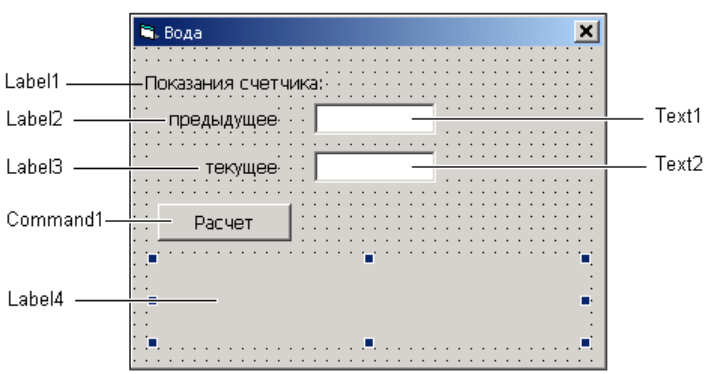


Рис. 4.2. Форма программы "Вода"

Таблица 4.2. *Значения свойств компонентов*

Компонент	Свойство	Значение
Label1	Left	16
	Top	16
	Caption	Показания счетчика:
	AutoSize	True

Таблица 4.2 *(окончание)*

Компонент	Свойство	Значение
Label2	Left	16
	Top	40
	Caption	предыдущее
	AutoSize	False
	Width	80
	Alignment	1 — Right Justify
Label3	Left	16
	Top	72
	Caption	текущее
	AutoSize	False
	Width	80
	Alignment	1 — Right Justify
Label4	Left	16
	Top	144
	AutoSize	False
	Width	280
	Height	50
	Caption	

Листинг 4.1. Вода

```
Option Explicit

Private Sub Command1_Click()

    Const cena = 17.5     ' цена 1 куб. метра воды
    Dim p1 As Double      ' предыдущее значение счетчика
    Dim p2 As Double      ' текущее значение счетчика
    Dim rashod As Double  ' расход
    Dim sum As Double     ' сумма

    p1 = CDbl(Text1.Text)
```

```
    p2 = CDbl(Text2.Text)

    If p2 >= p1 Then
        rashod = p2 - p1
        sum = rashod * cena
        Label4.ForeColor = vbBlack
        Label4.Caption =
             "Расход: " + Format(rashod, "#0.00 м.куб.") + vbCrLf + _
            "Цена за м куб. :" + Format(cena, "#0.00 руб.") + vbCr + _
            "К оплате: " + Format(sum, "#0.00 руб.")

    Else
        Label4.ForeColor = &H80& ' темно-красный
        Label4.Caption = "Ошибка исходных данных." + vbCr + _
          "Текущее показание счетчика не может быть меньше предыдущего."

    End If

End Sub

' изменился текст в поле редактирования Text1
Private Sub Text1_Change()
    If Len(Text1.Text) = 0 Or Len(Text2.Text) = 0 Then
        Command1.Enabled = False
    Else
        Command1.Enabled = True
    End If

End Sub

' нажатие клавиши в поле Text1
Private Sub Text1_KeyPress(KeyAscii As Integer)

    Select Case KeyAscii
        Case 8, 48 To 57 ' <Backspace> и цифры
        Case 44, 46      ' запятая и точка
            If KeyAscii = 46 Then
                ' заменим точку запятой
```

```
            KeyAscii = 44
        End If
        ' проверим, введена ли запятая
        If InStr(1, Text1.Text, ",") <> 0 Then
            ' запятая уже введена
            ' вторая не нужна
            KeyAscii = 0
        End If
    Case 13 ' <Enter>
        Text2.SetFocus ' переместить курсор в поле Text2
    Case Else
        ' прочие символы запрещены
        KeyAscii = 0
End Select

End Sub

' нажатие клавиши в поле Text2
Private Sub Text2_KeyPress(KeyAscii As Integer)

    Select Case KeyAscii
        Case 8, 48 To 57 ' <Backspace> и цифры
        Case 44, 46      ' запятая и точка
            If KeyAscii = 46 Then
                ' заменим точку запятой
                KeyAscii = 44
            End If
            ' проверим, введена ли запятая
            If InStr(1, Text2.Text, ",") <> 0 Then
                ' запятая уже введена
                ' вторая не нужна
                KeyAscii = 0
            End If
        Case 13 ' <Enter>
            Command1.SetFocus ' переместить курсор в поле Text2
        Case Else
            ' прочие символы запрещены
```

```
            KeyAscii = 0
    End Select


End Sub


' изменился текст в поле редактирования
Private Sub Text2_Change()
    If Len(Text1.Text) = 0 Or Len(Text2.Text) = 0 Then
        Command1.Enabled = False
    Else
        Command1.Enabled = True
    End If
End Sub
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какое свойство определяет текст, отображаемый в поле компонента `Label`?
2. Как изменить текст, отображаемый в поле компонента `Label`?
3. Какое свойство определяет цвет текста, отображаемого в поле компонента `Label`?
4. В поле компонента `Label` надо вывести значение переменной дробного типа. Какую функцию следует использовать для преобразования численного значения в строку?

TextBox

Компонент `TextBox` (рис. 4.3) предназначен для ввода данных (строки символов) с клавиатуры. Свойства компонента приведены в табл. 4.3.

Наибольший интерес для программиста представляют события `KeyPress` и `Change`. Событие `KeyPress` возникает в момент нажатия клавиши в поле компонента (до того, как символ появится в поле редактирования), событие `Change` — в момент изменения содержимого поля редактирования.

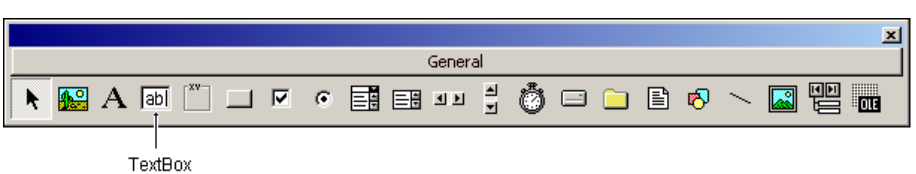


Рис. 4.3. Компонент `TextBox`

Таблица 4.3. *Свойства компонента* `TextBox` *(поле редактирования)*

Свойство	Описание
`Text`	Текст, находящийся в поле компонента
`MaxLength`	Задает максимальное количество символов, которое можно ввести в поле редактирования. Если значение свойства равно 0, то ограничения на количество символов нет
`Font`	Шрифт, используемый для отображения содержимого поля
`MultiLine`	Позволяет (`True`) ввести в поле редактирования несколько строк текста
`ScrollBars`	Управляет отображением полос прокрутки: • 0 — полосы прокрутки не отображать; • 1 — отображать горизонтальную полосу прокрутки; • 2 — отображать вертикальную полосу прокрутки; • 3 — отображать обе полосы прокрутки
`Locked`	Используется для ограничения возможности изменить текст в поле редактирования. Если значение свойства равно `True`, то текст в поле редактирования изменить нельзя
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

Если не предпринимать никаких усилий, то в поле компонента `TexBox` отображаются все символы, которые пользователь набирает на клавиатуре. Что не всегда удобно. Программа, точнее процедура обработки события `KeyPress`, может обеспечить фильтрацию вводимых символов — путем замены кода "запрещенного" символа нулем.

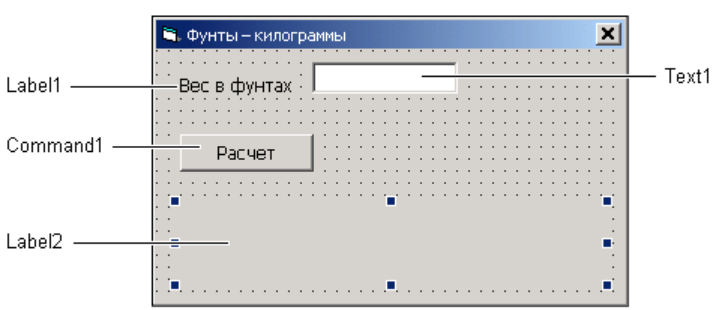


Рис. 4.4. Форма программы "Фунты — килограммы"

Использование компонента `TextBox` демонстрирует программа "Фунты — килограммы" (ее форма приведена на рис. 4.4, а текст — в листинге 4.2).

Программа спроектирована таким образом, что в поле редактирования можно ввести только дробное число (фильтрацию символов осуществляет процедура обработки события `KeyPress`). Кроме того, функция обработки события `Change` управляет доступностью кнопки **Расчет**. Кнопка становится доступной только в том случае, если в поле редактирования есть данные.

Листинг 4.2. Фунты — килограммы

```
Option Explicit

' нажатие клавиши в поле редактирования
Private Sub Text1_KeyPress(KeyAscii As Integer)

    Select Case KeyAscii
        Case 8, 48 To 57 ' <Backspace> и цифры
        Case 44, 46      ' запятая (44) и точка (46)
            If Len(Text1.Text) = 0 Then
                ' запятая не может быть первым символом
                KeyAscii = 0
            Else
                If KeyAscii = 46 Then
                    ' заменим точку запятой
                    KeyAscii = 44
                End If
                ' проверим, введена ли запятая
                If InStr(1, Text1.Text, ",") <> 0 Then
                    ' запятая уже введена
                    ' вторая не нужна
                    KeyAscii = 0
                End If
            End If
        Case 13 ' <Enter>
            If Len(Text1.Text) > 0 Then
                Command1.SetFocus ' переместить фокус на кнопку Command1
            End If
        Case Else
            ' прочие символы запрещены
            KeyAscii = 0
```

```
    End Select
End Sub

' изменилось содержимое поля Text1
Private Sub Text1_Change()
    ' изменился текст в поле Text1 и, следовательно,
    ' значение, отображаемое в поле Label2, уже ему не соответствует

    Label2.Caption = ""

    If Len(Text1.Text) = 0 Then
        ' в поле Text1 нет данных
        ' сделаем кнопку Расчет недоступной
        Command1.Enabled = False
    Else
        Command1.Enabled = True
    End If

End Sub

Private Sub Command1_Click()
    Dim f As Double  ' вес в фунтах
    Dim k As Double  ' вес в килограммах

    f = CDbl(Text1.Text)
    k = f * 0.409

    If k < 1 Then
        k = k * 1000
        Label2.Caption = Format(f, "#0.00 ф.") + " = " + _
                         Format(k, "#0.00 гр.")
    Else

        Label2.Caption = Format(f, "#0.00 ф.") + " = " + _
                         Format(k, "#0.00 кг")
    End If

End Sub
```

Контрольные вопросы

1. Как получить текст, находящийся в поле компонента `TextBox`?
2. Какую функцию следует использовать для преобразования строки, находящейся в поле компонента `TextBox`, в число?
3. Как проверить, есть ли текст в поле редактирования?
4. Какое событие происходит при нажатии клавиши, когда курсор находится в поле редактирования?

CommandButton

Компонент `CommandButton` (рис. 4.5) представляет собой командную кнопку. Обычно на кнопке находится текст, но может быть и картинка (в этом случае кнопку называют графической). Свойства компонента `CommandButton` приведены в табл. 4.4.

Рис. 4.5. Компонент `CommandButton`

Таблица 4.4. *Свойства компонента* `CommandButton`

Свойство	Описание
`Caption`	Текст на кнопке
`Enabled`	Признак доступности кнопки. Если значение свойства равно `True`, то кнопка доступна. Если значение свойства равно `False`, то кнопка не доступна (при щелчке на кнопке событие `Click` не происходит)
`Visible`	Позволяет скрыть кнопку (значение — `False`) или сделать ее видимой (значение — `True`)
`Style`	Вид кнопки: "обычная" (0 — `Standard`) или "графическая" (1 — `Graphical`). Графическая кнопка — это кнопка, на поверхности которой есть картинка
`Picture`	Свойство задает картинку для "графической" кнопки. Картинка отображается на поверхности формы, если значение свойства `Style` равно `1` (`Graphical`)

Таблица 4.4 *(окончание)*

Свойство	Описание
DisabledPicture	Задает картинку для недоступной "графической" кнопки. Картинка отображается, если значение свойства Style равно Graphical, а свойства Enabled — False
DownPicture	Задает картинку для нажатой "графической" кнопки. Картинка отображается в момент нажатия кнопки, если значение свойства Style равно Graphical
MaskColor	Задает "прозрачный" цвет. Точки картинки, окрашенные этим цветом, на поверхности кнопки не отображаются (при условии, что значение свойства UseMaskColor равно True)
UseMaskColor	Устанавливает (True), что точки картинки, окрашенные цветом MaskColor, на поверхности кнопки не должны отображаться
ToolTipText	Задает текст подсказки, которая появляется при позиционировании указателя мыши на кнопке

Использование компонента CommandButton демонстрирует программа "Мили — километры" (ее форма приведена на рис. 4.6, значения свойств компонента Command1 — в табл. 4.5, а текст — в листинге 4.3). Программа спроектирована таким образом, что кнопка **OK** доступна только в том случае, если поле редактирования содержит данные (доступностью кнопки управляет процедура обработки события Change компонента Text1).

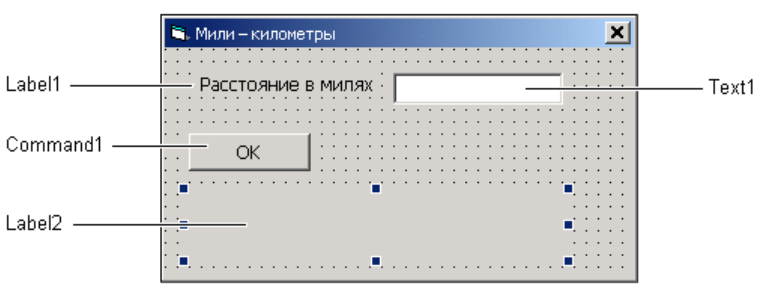


Рис. 4.6. Форма программы "Мили — километры"

Таблица 4.5. *Значения свойств компонента* Command1

Свойство	Значение
Caption	OK
Enabled	False

Листинг 4.3. Мили — километры

```
Option Explicit

' щелчок на кнопке OK
Private Sub Command1_Click()
    Dim m As Double
    Dim k As Double

    m = CDbl(Text1.Text)

    k = m * 1.6093

    Label2.Caption = Format(m, "# ###.00 m") + " = " + _
                     Format(k, "# ###.00 км.")
End Sub

' изменилось содержимое поля редактирования
Private Sub Text1_Change()

    Label2.Caption = ""
    ' проверить, есть ли символы в поле редактирования
    If (Len(Text1.Text) = 0) Or ((Mid(Text1.Text, 1, 1) = ",") And _
           (Len(Text1.Text) = 1)) Then
        ' в поле редактирования нет данных
        ' сделаем кнопку OK недоступной
        Command1.Enabled = False
    Else
        Command1.Enabled = True
    End If
End Sub

' нажатие клавиши в поле редактирования
Private Sub Text1_KeyPress(KeyAscii As Integer)

    Select Case KeyAscii
        Case 8, 48 To 57 ' <Backspace> и цифры
        Case 44, 46      ' запятая (44) и точка (46)
            If KeyAscii = 46 Then
```

```
            ' заменим точку запятой
            KeyAscii = 44
        End If
        ' проверим, введена ли запятая
        If InStr(1, Text1.Text, ",") <> 0 Then
            ' запятая уже введена
            ' вторая не нужна
            KeyAscii = 0
        End If
    Case 13 ' <Enter>
        If Command1.Enabled Then
            Command1.SetFocus ' переместить фокус на кнопку Command1
        End If
    Case Else
        ' прочие символы запрещены
        KeyAscii = 0
End Select
End Sub
```

Контрольные вопросы

1. Какое свойство задает текст, отображаемый на командной кнопке?
2. Какое событие происходит при "нажатии" командной кнопки?
3. Как сделать, чтобы кнопка не реагировала на "нажатие" мышью?
4. Какое свойство задает текст подсказки, отображаемой при позиционировании мыши на командной кнопке?

CheckBox

Компонент `CheckBox` (рис. 4.7) представляет собой флажок, который может находиться в установленном (включенном), сброшенном (выключенном) или в промежуточном состоянии. Когда флажок включен (выбран), в его поле находится галочка, когда выключен — галочки нет. Свойства компонента `CheckBox` приведены в табл. 4.6.

Рис. 4.7. Компонент `CheckBox`

Таблица 4.6. *Свойства компонента* `CheckBox`

Свойство	Описание
`Value`	Состояние флажка: `Checked` — флажок включен (в квадратике есть галочка); `Unchecked` — флажок выключен (нет галочки)
`Caption`	Текст, который находится справа от флажка
`Enabled`	Признак доступности флажка. Если значение свойства равно `True`, то флажок доступен. Если значение свойства равно `False`, то флажок не доступен
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

Состояние флажка изменяется в результате щелчка на его изображении (если значение свойства `Enabled` равно `True`). При этом возникает событие `Click`.

Использование компонента `CheckBox` демонстрирует программа "Стеклопакет" (ее форма приведена на рис. 4.8, значения свойств компонентов `CheckBox` — в табл. 4.7, а текст — в листинге 4.4). Программа вычисляет стоимость стеклопакета.

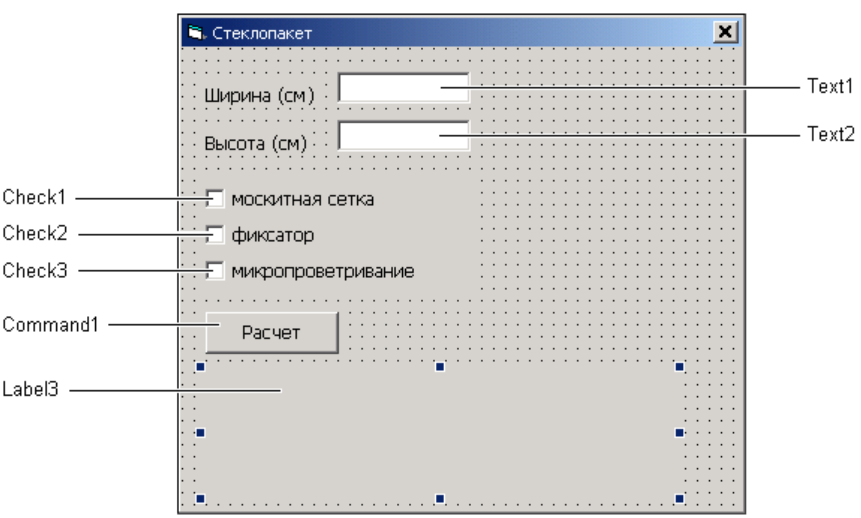


Рис. 4.8. Форма программы "Стеклопакет"

Таблица 4.7. *Значения свойств компонентов*

Компонент	Свойство	Значение
Check1	Caption	москитная сетка
	Value	0 — Unchecked
Check2	Caption	фиксатор
	Value	0 — Unchecked
Check3	Caption	микропроветривание
	Value	0 — Unchecked

Листинг 4.4. Стеклопакет

```
' щелчок на кнопке Расчет
Private Sub Command1_Click()
    Dim w As Double ' ширина (см)
    Dim h As Double ' высота (см)

    Dim s As Double ' площадь

    Dim sum As Double      ' сумма (руб.)
    Dim discount As Double ' скидка
    Dim total As Double    ' к оплате

    s = 0
    sum = 0
    discount = 0

    w = CDbl(Text1.Text)
    h = CDbl(Text2.Text)

    s = w * h / 10000
    sum = s * CENA

    If Check1.Value = 1 Then
        ' установлен флажок "москитная сетка"
        sum = sum + 1800 * s
    End If

    If Check2.Value = 1 Then
```

```
        ' установлен флажок "фиксатор"
        sum = sum + 300
    End If

    If Check3.Value = 1 Then
        ' установлен флажок "микропроветривание"
        sum = sum + 1500
    End If

    If Check1.Value = 1 And Check2.Value = 1 And Check3.Value = 1 Then
        ' скидка 10%
        discount = sum * 0.1
        total = sum - discount
    End If

    If discount = 0 Then
        Label3.Caption = _
           "Размер: " + Str(w) + "x" + LTrim(Str(h)) + "  см" + vbCr + _
           "Площадь: " + Format(s, "#0.00 кв.м") + vbCr + _
           "Сумма: " + Format(sum, "# ##0.00 руб.")
    Else
        Label3.Caption = _
            "Размер: " + Str(w) + "x" + LTrim(Str(h)) + " см" + vbCr + _
            "Площадь: " + Format(s, "#0.00 кв.м") + vbCr + _
            "Сумма: " + Format(sum, "# ##0.00 руб.") + vbCr + _
            "Скидка (10%): "+ Format(discount,"# ##0.00 руб.") + vbCr + _
            "К оплате: " + Format(total, "# ##0.00 руб.")
    End If

End Sub
```

Контрольные вопросы

1. Какое свойство задает текст, отображаемый рядом с флажком (компонентом `CheckBox`)?
2. Как проверить состояние (установлен или сброшен) флажка?
3. Какое свойство компонента `CheckBox` содержит информацию о состоянии флажка?
4. Пусть на форме три флажка (компонента `CheckBox`). Могут ли быть выбранными (установленными) одновременно все флажки?

OptionButton

Компонент `OptionButton` (рис. 4.9) представляет собой переключатель, который может находиться в выбранном (включенном) или в невыбранном (выключенном) состоянии. Следует обратить внимание, что состояние переключателя зависит от состояния других переключателей (компонентов `OptionButton`), которые находятся на форме. В каждый момент времени только один из переключателей может находиться в выбранном состоянии. Вместе с тем возможна ситуация, когда ни один из переключателей не выбран.

Несколько компонентов `OptionButton` можно объединить в группу, разместив их в поле компонента `Frame`. Состояние компонентов, принадлежащих к разным группам, не зависит от состояния других компонентов, принадлежащих другой группе. Свойства компонента приведены в табл. 4.8.

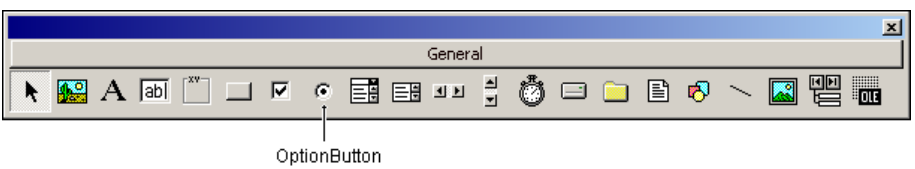


Рис. 4.9. Компонент `OptionButton`

Таблица 4.8. *Свойства компонента* `OptionButton`

Свойство	Описание
`Caption`	Текст, который находится справа от переключателя
`Value`	Состояние переключателя. Если компонент выбран, то значение свойства равно `True`, если компонент не выбран, значение равно `False`
`Font`	Шрифт, используемый для отображения поясняющего текста
`Enabled`	Признак доступности компонента. Если значение свойства равно `True`, то переключатель доступен. Если значение свойства равно `False`, то переключатель не доступен
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

Состояние переключателя изменяется в результате щелчка на его изображении. При этом возникает событие `Click`.

Использование компонента OptionButton демонстрирует программа "Сопротивление" (ее форма приведена на рис. 4.10, значения свойств компонентов — в табл. 4.9, а текст — в листинге 4.5) .

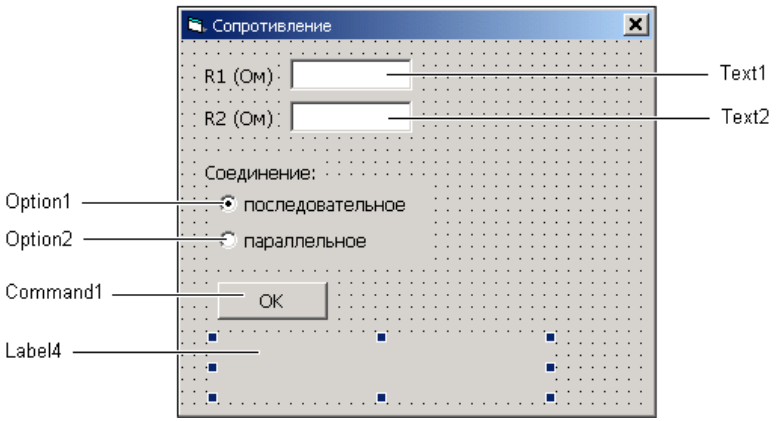


Рис. 4.10. Форма программы "Сопротивление"

Таблица 4.9. *Значения свойств компонентов*

Компонент	Свойство	Значение
Option1	Caption	последовательное
	Value	True
Option2	Caption	параллельное
	Value	False

Листинг 4.5. Сопротивление

```
Private Sub Command1_Click()
    ' величины сопротивлений
    Dim r1 As Double
    Dim r2 As Double

    Dim r As Double ' сопротивление цепи

    r1 = CDbl(Text1.Text)
    r2 = CDbl(Text2.Text)
```

```
    If Option1.Value = True Then
        ' последовательное соединение
        r = r1 + r2
    Else
        ' на форме только два переключателя,
        ' поэтому если не выбран Option1, то выбран Option2
        r = r1 * r2 / (r1 + r2)
    End If

    If r <= 1000 Then
        Label4.Caption = "Сопротивление цепи: " + Format(r, "#0.00 Ом")
    Else
        r = r / 1000
        Label4.Caption = "Сопротивление цепи: " + Format(r, "#0.00 кОм")
    End If

End Sub
```

Контрольные вопросы

1. Какое свойство задает текст, отображаемый рядом с переключателем (компонентом `OptionButton`)?
2. Как проверить состояние (выбран или нет) переключателя?
3. Какое свойство компонента `OptionButton` содержит информацию о состоянии переключателя?
4. Пусть на форме три компонента `OptionButton`. Могут ли быть в установленном состоянии одновременно несколько переключателей?

ComboBox

Компонент `ComboBox` (рис. 4.11) представляет собой комбинацию поля ввода и списка, что позволяет ввести данные в поле редактирования путем набора на клавиатуре или путем выбора из списка. Свойства компонента приведены в табл. 4.10.

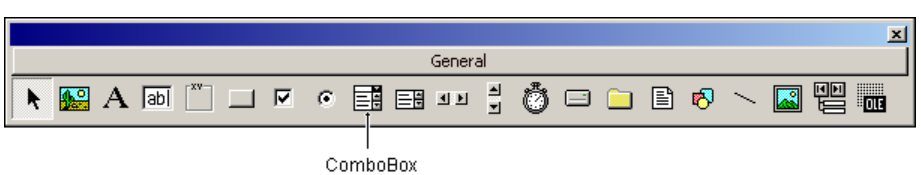


Рис. 4.11. Компонент `ComboBox`

Таблица 4.10. *Свойства компонента* `ComboBox`

Свойство	Описание
`Style`	Вид списка: поле ввода и раскрывающийся список (0 — `Dropdown Combo`); поле ввода со списком (1 — `Simple Combo`); раскрывающийся список (2 — `Dropdown List`)
`Text`	Содержимое поля редактирования (данные, введенные пользователем с клавиатуры или выбранные из списка)
`List`	Элементы списка — массив строк
`ListCount`	Количество элементов списка
`ListIndex`	Номер выбранного элемента списка. Если ни один из элементов списка не был выбран, то значение свойства равно `-1`. Элементы списка нумеруются с нуля
`Sorted`	Признак необходимости выполнить сортировку списка после добавления очередного элемента (значение свойства можно задать только во время разработки формы)
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

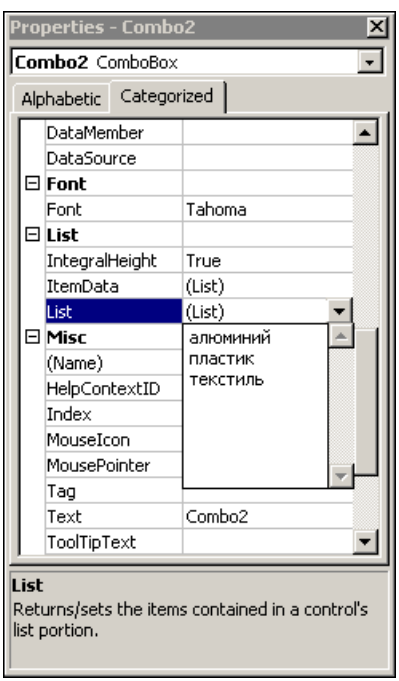


Рис. 4.12. Формирование списка компонента `ComboBox`

Список, отображаемый в поле компонента ComboBox, можно сформировать как во время создания формы, так и во время работы программы. Чтобы сформировать список во время создания формы, надо раскрыть список List и ввести элементы списка (рис. 4.12), нажимая в конце каждой строки комбинацию клавиш <Ctrl>+<Enter>.

Формирование списка во время работы программы обеспечивает метод AddItem. В качестве параметра метода надо указать добавляемый элемент списка и (необязательно) позицию, куда надо этот элемент поместить. Например, следующий фрагмент кода формирует список компонента Combo1.

```
Combo1.AddItem ("алюминий")
Combo1.AddItem ("пластик")
Combo1.AddItem ("текстиль")
```

Использование компонента ComboBox демонстрирует программа "Жалюзи" (ее форма приведена на рис. 4.13, значения свойств компонента Combo1 — в табл. 4.11, а текст — в листинге 4.6).

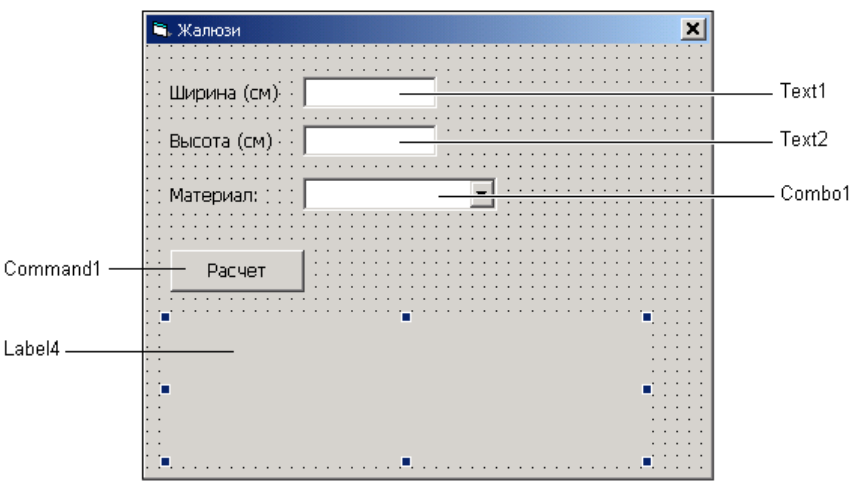


Рис. 4.13. Форма программы "Жалюзи"

Таблица 4.11. *Значения свойств компонента* Combo1

Свойство	Значение
Combo1.Style	0 — Dropdown Combo
Combo1.Sorted	False

Листинг 4.6. Жалюзи

```
Option Explicit

Dim cena(10) As Double ' прайс-лист

' процедура обработки события Load формы
Private Sub Form_Load()

    ' сформировать список компонента Combo1 и
    ' инициализировать массив cena
    Combo1.AddItem ("алюминий")
    cena(0) = 1500

    Combo1.AddItem ("пластик")
    cena(1) = 500

    Combo1.AddItem ("текстиль")
    cena(2) = 1000

    Combo1.AddItem ("бамбуковая соломка")
    cena(3) = 500

    Combo1.AddItem ("дерево")
    cena(4) = 800

End Sub

' щелчок на кнопке Расчет
Private Sub Command1_Click()
    Dim w As Double ' ширина (см)
    Dim h As Double ' высота (см)

    Dim c As Double ' цена 1 кв. м

    Dim s As Double ' площадь
```

```
    Dim sum As Double ' сумма (руб.)

    If Combo1.ListIndex = -1 Then
        MsgBox "Выберите материал", vbExclamation, "Жалюзи"
        Exit Sub ' выход из процедуры
    End If

    s = 0
    sum = 0

    w = CDbl(Text1.Text)
    h = CDbl(Text2.Text)

    s = w * h / 10000 ' перевод в кв. м

    ' элементы списка компонента Combo1
    ' и массива cena (см. его объявление) нумеруются с нуля
    Select Case Combo1.ListIndex
        Case 0
            c = cena(0)
        Case 1
            c = cena(1)
        Case 2
            c = cena(2)
        Case 3
            c = cena(3)
        Case 4
            c = cena(4)
    End Select

    sum = s * c

    Label3.Caption = _
           "Размер: " + Str(w) + "x" + LTrim(Str(h)) + "  см" + vbCr + _
           "Материал: " + Combo1.List(Combo1.ListIndex) + "(" _
           Format(cena(Combo1.ListIndex), "#0.00 руб./м кв.)") + vbCr + _
           "Сумма: " + Format(sum, "# ##0.00 руб.")
End Sub
```

Контрольные вопросы

1. Какое свойство определяет элементы, отображаемые в списке компонента ComboBox?
2. Какое свойство содержит информацию о номере элемента, выбранного в списке компонента ComboBox?
3. Если ни один из элементов списка компонента ComboBox не выбран, чему равно значение свойства ListIndex?
4. Элементы списка компонента ComboBox нумеруются с единицы или с нуля?

Timer

Компонент Timer (рис. 4.14) обеспечивает генерацию последовательности событий Timer. Свойства компонента приведены в табл. 4.12.

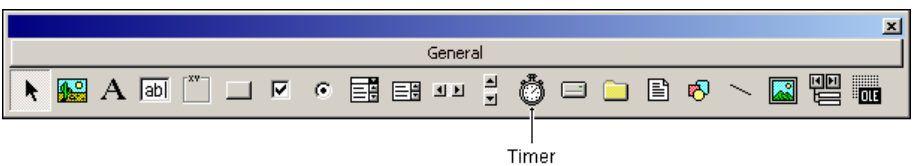


Рис. 4.14. Компонент Timer

Таблица 4.12. *Свойства компонента* Timer

Свойство	Описание
Interval	Период генерации события Timer. Задается в миллисекундах
Enabled	Разрешение работы. Если значение свойства равно True, то таймер генерирует событие Timer с периодом Interval

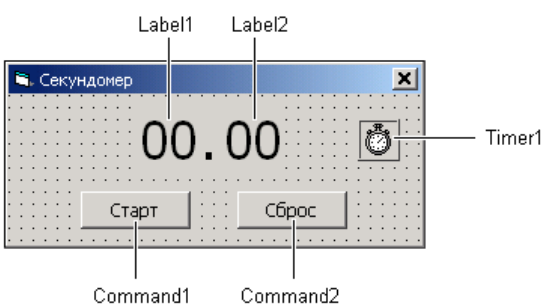


Рис. 4.15. Форма программы "Секундомер"

Использование компонента `Timer` демонстрирует программа "Секундомер" (ее форма приведена на рис. 4.15, значения свойств компонента `Timer` — в табл. 4.13, а текст — в листинге 4.7).

Таблица 4.13. *Значения свойств компонента* `Timer1`

Свойство	Значение
Interval	500
Enabled	True

Листинг 4.7. Секундомер

```
Option Explicit

Dim min As Integer
Dim sec As Integer

' начало работы программы
Private Sub Form_Load()
    min = 0
    sec = 0
End Sub

' щелчок на кнопке Старт/Стоп
Private Sub Command1_Click()
    If Command1.Tag = 0 Then
        ' нажатие кнопки Старт
        Timer1.Enabled = True ' пуск таймера
        Command1.Caption = "Стоп"
        Command1.Tag = 1 ' теперь Command1 — кнопка Стоп
        Command2.Enabled = False ' кнопка Сброс теперь недоступна
    Else
        ' нажатие кнопки Стоп
        Timer1.Enabled = False ' остановить таймер
        Command1.Caption = "Старт"
        Command1.Tag = 0 ' теперь Command1 — кнопка Старт
        Command2.Enabled = True ' кнопка Сброс теперь доступна
```

```
    End If
End Sub

' обработка сигнала таймера (события, сгенерированного таймером)
Private Sub Timer1_Timer()
    If sec < 59 Then
        sec = sec + 1
        Label2.Caption = Format(sec, "00")
    Else
        sec = 0
        Label2.Caption = Format(sec, "00")
        min = min + 1
        Label1.Caption = Format(min, "00")
    End If

End Sub

' щелчок на кнопке Сброс
Private Sub Command2_Click()
    min = 0
    sec = 0
    Label1.Caption = "00"
    Label2.Caption = "00"
End Sub
```

Секундомер запускается щелчком на кнопке **Старт**. Процедура обработки события `Click` на кнопке `Command1` присваивает свойству `Enabled` значение `True`. В результате каждую секунду происходит событие `Timer`, процедура обработки которого изменяет значения счетчиков секунд и минут и выводит их значения в поля `Label1` и `Label2`. Состояние секундомера фиксируется в свойстве `Tag` кнопки `Command1`. Если секундомер работает, то значение свойства равно 1, если остановлен — 0.

Контрольные вопросы

1. Какое свойство определяет период возникновения сигналов (генерации событий) от таймера?
2. Как запустить таймер?
3. Как остановить таймер?
4. В каких единицах задается период генерации событий от таймера?

PictureBox

Компонент PictureBox (рис. 4.16) предназначен для отображения графики, формируемой во время работы программы из графических примитивов (линии, прямоугольники, точки и т. д.), а также иллюстраций. Свойства компонента PictureBox приведены в табл. 4.14.

Рис. 4.16. Компонент PictureBox

Таблица 4.14. *Свойства компонента* PictureBox

Свойство	Описание
Picture	Картинка, отображаемая в поле компонента. Задать картинку можно во время разработки формы или загрузить из файла во время работы программы (функция LoadPicture)
AutoSize	Свойство разрешает (True) или запрещает (False) автоматическое изменение размера компонента (области вывода иллюстрации) в соответствии с размером картинки, загруженной в компонент
BorderStyle	Стиль границы компонента. Если значение свойства равно 1, то граница — тонкая линия, если — 0, то граница не отображается
BackColor	Цвет фона компонента. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
Font	Шрифт, которым метод Print выводит текст
ForeColor	Для метода Print задает цвет символов, для методов вычерчивания графических примитивов (объектов) — цвет линий
FillColor	Задает цвет закраски внутренних областей графических примитивов (объектов), вычерчиваемых в поле (на поверхности) компонента
FillStyle	Стиль закраски графических объектов, вычерчиваемых в поле компонента соответствующими методами: • 0 (Solid) — сплошная заливка; • 1 (Transparent) — закраска "прозрачным" цветом; • 2 (HorizontalLine) — горизонтальная штриховка; • 3 (VerticalLine) — вертикальная штриховка. Цвет линий штриховки определяет свойство FillColor

Таблица 4.14 *(окончание)*

Свойство	Описание
DrawStyle	Вид контура графических объектов, вычерчиваемых в поле компонента соответствующими методами: • 0 (Solid) — сплошная линия; • 1 (Dash) — пунктирная линия; • 2 (Dot) — линия из точек; • 3 (Dash-Dot) — линия вида "точка-тире"; • 4 (Dash-Dot-Dot) — линия вида "тире-точка-точка"; • 5 (Transparent) — "прозрачная" линия
DrawWidth	Толщина линий для графических объектов
ScaleWidth	Ширина рабочей области компонента, т. е. без учета ширины левой и правой границ. Единицу измерения задает свойство ScaleMode
ScaleHeight	Высота рабочей области компонента, т. е. без учета ширины нижней и верхней границ компонента. Единицу измерения задает свойство ScaleMode
ScaleMode	Задает единицу измерения размеров компонента и объектов на его поверхности. Значение этого свойства не влияет на единицы измерения свойств Width и Height (не зависимо от него их значения измеряются в твипах)
Height	Высота компонента
Width	Ширина компонента
Visible	Позволяет скрыть компонент (False) или сделать его видимым (True)

Использование компонента PictureBox демонстрирует программа "График". Программа строит в поле компонента Picture1 график изменения курса доллара (вопросы программирования графики подробно рассмотрены в *главе 5*). Данные загружаются из текстового файла. Форма программы приведена на рис. 4.17, значения свойств компонента Picture1 — в табл. 4.15. Следует обратить внимание на то, что разброс значений ряда незначителен (в пределах единицы), поэтому график строится в отклонениях. Текст программы приведен в листинге 4.8, окно — на рис. 4.18.

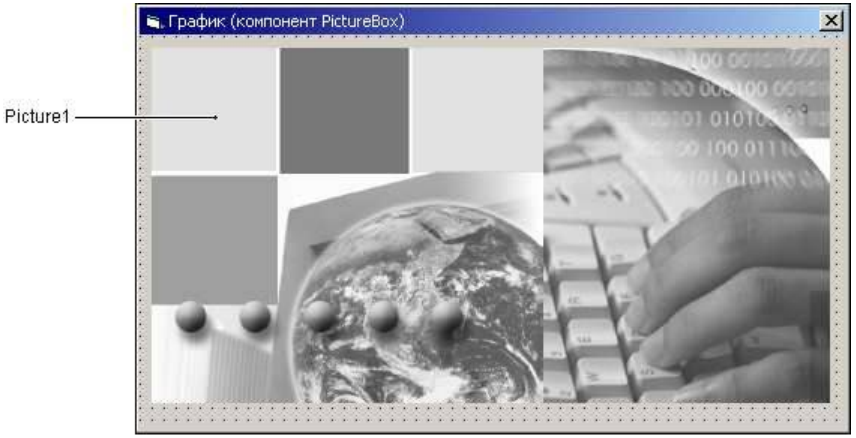


Рис. 4.17. Форма программы "График"

Таблица 4.15. *Значения свойств компонента* `Picture1`

Свойство	Значение
BorderStyle	0 — None
AutoSize	True
Picture	back.bmp
ScaleMode	3 — Pixel

Листинг 4.8. График

```
Option Explicit

Const N = 10                ' количество элементов данных
Dim usd(1 To N) As Double   ' ряд данных
Dim min, max As Double      ' минимальное и максимальное значения ряда данных
Dim loaded As Boolean       ' True — данные загружены, False — нет

Private Sub Form_Activate()

    Dim f As String ' файл данных
    Dim i As Integer
```

```
Dim st As String ' буфер для чтения строк из файла

loaded = False ' данные не загружены

' Загрузить данные из файла
' Внимание! При запуске программы из Visual Basic текущим
' является каталог, заданный как Рабочая папка.
' При запуске программы (EXE-файла) из операционной системы
' текущим является каталог, в котором находится EXE-файл

f = "C:\VBProjects\PictureBox\usd.dat" ' отладка
' f = CurDir + "\usd.dat"

' Открыть файл для чтения данных
On Error GoTo e1
Open f For Input As #1    ' отладка


For i = 1 To N
    Line Input #1, st ' прочитать строку из файла
    usd(i) = Val(st)
Next i
Close #1 ' закрыть файл

loaded = True ' данные загружены

' Домножим значения на 100, чтобы работать с целыми числами
For i = 1 To N
    usd(i) = usd(i) * 100
Next i

 ' Найти минимальное значение
min = usd(1)
For i = 2 To N
    If usd(i) < min Then min = usd(i)
Next i

' Найти максимальное значение
max = usd(1)
```

```
    For i = 2 To N
        If usd(i) > max Then max = usd(i)
    Next i

    Exit Sub

e1: ' обработка ошибки открытия файла
    MsgBox "Ошибка открытия файла данных " + f, vbCritical

End Sub

' график строит процедура обработки события Paint
Private Sub Picture1_Paint()

    Dim k As Double ' количество пикселов на единицу значения функции

    ' График строим в приращениях
    ' Изменение координаты следующей точки относительно предыдущей
    Dim dx As Integer ' по X
    Dim dy As Integer ' по Y

    Dim i As Integer

    Dim cx, cy As Integer ' положение указателя графического вывода

    If Not loaded Then
        ' данные не загружены
        Exit Sub
    End If

    ' заголовок
    Picture1.CurrentX = 100
    Picture1.CurrentY = 5
    Picture1.Font.Size = 14
    Picture1.Font.Bold = False
    Picture1.ForeColor = vbWhite
    Picture1.Print "Изменение курса доллара"
```

```
Picture1.CurrentX = 101
Picture1.CurrentY = 6
Picture1.Font.Size = 14
Picture1.Font.Bold = False
Picture1.ForeColor = vbBlack
Picture1.Print "Изменение курса доллара"

Picture1.Font.Size = 10
Picture1.Font.Bold = False
' Picture1.DrawWidth = 2

dx = (Picture1.ScaleWidth - 20) / N
k = (Picture1.ScaleHeight - 50) / (max - min)

' координаты первой точки графика
Picture1.CurrentX = 20
Picture1.CurrentY = (Picture1.ScaleHeight - 15) - (usd(1) - min) * k



' Так как метод Print изменит положение указателя
' вывода, сохраним значения CurrentX и CurrentY
cx = Picture1.CurrentX
cy = Picture1.CurrentY

Picture1.Print usd(1) / 100

' Указатель вывода переместим в точку, в которой
' он был до выполнения метода Print
Picture1.CurrentX = cx
Picture1.CurrentY = cy

' остальные точки графика
For i = 2 To N
    ' вычислить координату Y относительно
    ' предыдущей точки
    dy = (usd(i - 1) - usd(i)) * k

    ' провести линию из предыдущей точки в текущую
```

```
        Picture1.Line -Step(dx, dy)

        cx = Picture1.CurrentX
        cy = Picture1.CurrentY

        Picture1.Print usd(i) / 100

        Picture1.CurrentX = cx
        Picture1.CurrentY = cy
    Next i
End Sub
```

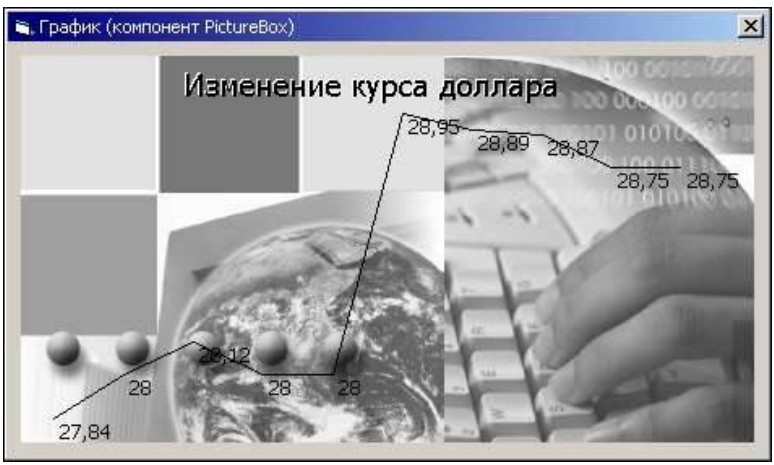


Рис. 4.18. График изменения курса доллара выведен на поверхность компонента `PictureBox`

Контрольные вопросы

1. Какое свойство задает картинку, отображаемую в поле компонента `PictureBox`?
2. Какое свойство задает единицу измерения размеров и координат графических объектов, вычерчиваемых на поверхности компонента `PictureBox`?
3. Какое событие следует использовать для формирования графики на поверхности компонента `PictureBox`?
4. Какой метод обеспечивает отображение (вывод) текста на поверхность компонента `PictureBox`?

Image

Компонент `Image` (рис. 4.19) предназначен для отображения иллюстраций. Основное отличие компонента `Image` от компонента `PictureBox` состоит в том, что он позволяет масштабировать иллюстрации. Еще одно отличие — на поверхности компонента `Image` нельзя рисовать. Иллюстрацию, отображаемую в поле компонента, можно задать во время создания формы (присвоив значению свойству `Picture`) или загрузить из файла. Свойства компонента `Image` приведены в табл. 4.16.

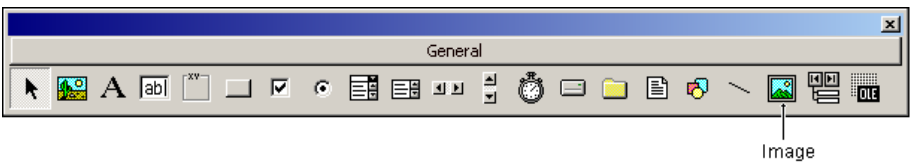


Рис. 4.19. Компонент `Image`

Таблица 4.16. *Свойства компонента* `Image`

Свойство	Описание
`Picture`	Картинка (иллюстрация), отображаемая в поле компонента. Задать иллюстрацию можно во время разработки формы или загрузить из файла во время работы программы (функция `LoadPicture`)
`Stretch`	Задает (`True`), что нужно выполнить масштабирование иллюстрации (сжать или растянуть) в соответствии с размером компонента. Если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена. Если значение свойства равно `False` — масштабирование не выполняется
`Width`, `Height`	Размер компонента. Если значение свойства `Stretch` равно `False`, то после загрузки иллюстрации значение свойства `Width` равно ширине иллюстрации, `Height` — высоте

Использование компонента `Image` демонстрирует программа "Просмотр иллюстраций" (ее форма приведена на рис. 4.20). Программа позволяет просмотреть иллюстрации, которые находятся в папке Мои рисунки. Иллюстрация отображается в поле компонента `Image1`. Если размер иллюстрации больше размера компонента, то она масштабируется. Компонент `Label1` используется для отображения информации об иллюстрации. Кнопка **Дальше** (`Command1`) обеспечивает переход к следующей иллюстрации. Текст программы приведен в листинге 4.9.

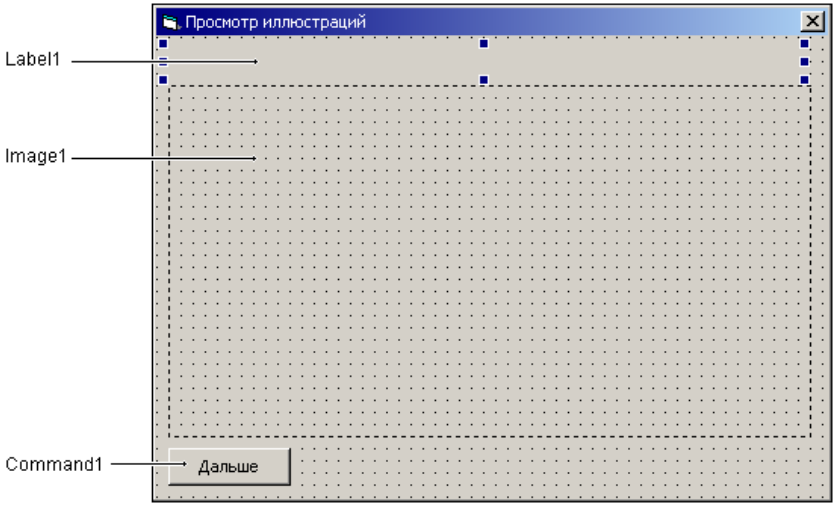


Рис. 4.20. Форма программы "Просмотр иллюстраций"

Листинг 4.9. Просмотр иллюстраций

```
Option Explicit

Dim aPath As String      ' каталог, в кот. находятся JPG-файлы
Dim aPicture As String   ' файл отображаемой иллюстрации
Dim w, h As Double       ' первоначальный размер компонента Image1


' начало работы программы
Private Sub Form_Activate()

    ' запомнить размер компонента Image1
    w = Image1.Width
    h = Image1.Height


    aPath = Environ("HOMEDRIVE") + Environ("HOMEPATH") + _
            "\Мои документы\Мои рисунки\"

    aPicture = Dir(aPath + "*.jpg") ' получить имя первого JPG-файла
```

```
    If aPicture = "" Then
        MsgBox "В каталоге " + aPath + "нет JPG-файлов"
        Exit Sub
    End If

    ShowPicture (aPath + aPicture)

    aPicture = Dir ' получить имя следующего JPG-файла
    If aPicture = "" Then
        Command1.Enabled = False
    End If

End Sub

' щелчок на кнопке Дальше
Private Sub Command1_Click()

    ShowPicture (aPath + aPicture)

    aPicture = Dir ' получить имя следующего JPG-файла
    If aPicture = "" Then
        Command1.Enabled = False
    End If

End Sub

' отображает иллюстрацию в поле компонента Image1
Sub ShowPicture(aPicture As String)

    Dim kw, kh As Double ' коэф. масштабирования иллюстрации
                         ' по ширине и высоте
    Dim k As Double      ' общий коэф. масштабирования иллюстрации

    Image1.Visible = False
    Image1.Stretch = False
    Image1.Picture = LoadPicture(aPicture)
```

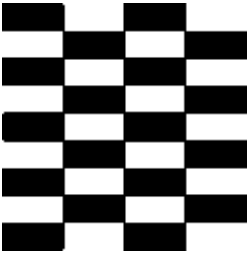
```
    If (Image1.Width > w) Or (Image1.Height > h) Then
        ' размер иллюстрации больше, чем размер
        ' компонента Image1
        kw = w / Image1.Width
        kh = h / Image1.Height
        ' чтобы иллюстрация отображалась без искажения,
        ' коэф. масштабирования по ширине и высоте должен быть одинаковый
        If kw < kh Then
            k = kw
        Else
            k = kh
        End If
    Else
        k = 1

    End If

    ' информация об иллюстрации
    Label1.Caption = aPicture & vbCr & "Размер:" & Image1.Width & "x" & _
                     Image1.Height & "   Масштаб:" & Format(k, "0.00")

    Image1.Width = Image1.Width * k
    Image1.Height = Image1.Height * k
    Image1.Stretch = True ' масштабировать
    Image1.Visible = True

End Sub
```

Непосредственное отображение иллюстрации обеспечивает процедура `showImage`. В начале своей работы она присваивается свойству `Stretch` значение `False`. Делается это для того, чтобы получить информацию о размере иллюстрации. После загрузки иллюстрации, если ее размер превышает размер компонента `Image1`, вычисляются коэффициенты масштабирования: сначала по ширине и высоте, а затем — общий. Далее устанавливаются размеры компонента `Image1` пропорционально размерам загруженной иллюстрации, и свойству `Stretch` присваивается значение `True`. В результате иллюстрация масштабируется и отображается без искажения (рис. 4.21).

Рис. 4.21. Окно программы "Просмотр иллюстраций"

Контрольные вопросы

1. Для чего предназначен компонент `Image`?
2. Какое свойство определяет картинку, отображаемую в поле компонента `Image`?
3. Какая функция обеспечивает во время работы программы загрузку картинки в поле компонента `Image`?
4. Как определить размер картинки, загруженной из файла в компонент `Image`?
5. Если размер иллюстрации больше размера компонента `Image`, то что надо сделать, чтобы иллюстрация отображалась без искажения?



Глава 5

Графика

В этой главе рассказывается, что надо сделать, чтобы на поверхности формы появилась фотография, картинка, созданная в графическом редакторе или сформированная из графических примитивов (линий, прямоугольников, окружностей, точек), диаграмма или график. Также вы познакомитесь с принципами создания анимации, узнаете, как "оживить" картинку.

Программа может вывести графику на поверхность формы или компонента `PictureBox`. Для того чтобы во время работы программы на поверхности объекта появилась, например, линия, необходимо вызвать соответствующий метод. Так в результате выполнения инструкции

```
Form1.Line (10,10) - (50,10)
```

на поверхности формы появится линия.

Графику на поверхности объекта должна формировать процедура обработки события `Paint`, которое происходит всякий раз, когда объект появляется на экране.

Графическая поверхность

Методы, обеспечивающие вычерчивание графических примитивов (линий, прямоугольников, окружностей и т. д.), рассматривают поверхность формы (или компонента `PictureBox`) как *холст*, на котором они могут *рисовать* путем изменения цвета отдельных точек — пикселов. Положение пиксела характеризуется его горизонтальной (`x`) и вертикальной (`y`) координатами. Координаты отсчитываются от левого верхнего угла и возрастают слева направо (`x`) и сверху вниз (`y`). Точка, находящаяся в левом верхнем углу поверхности, имеет координаты (0, 0) (рис. 5.1).

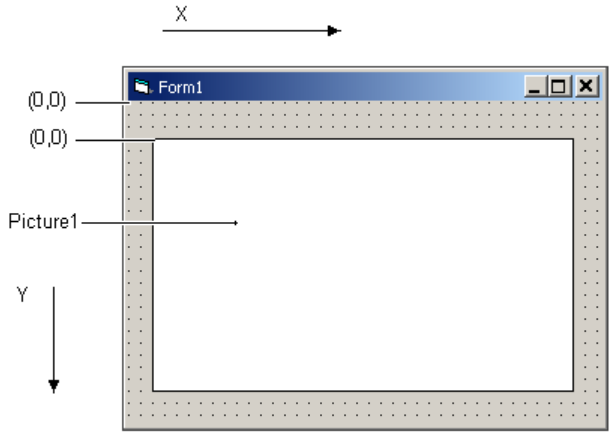


Рис. 5.1. Координаты точек графической поверхности отсчитываются от левого верхнего угла

Следует обратить внимание на то, что координаты точек графической поверхности могут измеряться в твипах (эта единица используется по умолчанию), пикселах, миллиметрах, сантиметрах и др. При программировании графики наиболее удобной единицей измерения является пиксел. Поэтому свойству `ScaleMode` (формы или компонента `PictureBox`) следует присвоить значение `Pixel` (во время создания формы приложения) или `vbPixels` (во время работы программы).

Координаты точек можно отсчитывать от левого верхнего угла графической поверхности (абсолютная адресация) или от текущего положения *указателя графического вывода* (относительная адресация).

Указатель графического вывода — это графический курсор, который, в отличие от обычного, текстового курсора, на экране не отображается. В начале работы программы он находится в точке (0, 0), а после выполнения операции отображения графики — в той точке, в которой она была завершена. Например, после выполнения инструкции

```
Form1.Line (10,10)-(60,10)
```

указатель графического вывода будет находиться в точке (60, 10).

При относительной адресации перед координатами указывается слово `Step`. Например, инструкция

```
Form1.Line (10,10)-Step(50,0)
```

рисует из точки (10, 10) горизонтальную линию длиной в 50 пикселов (координаты конца линии отсчитываются от ее начала).

Информация о текущем положении указателя графического вывода находится в свойствах `CurrentX` и `CurrentY`.

Программа может переместить указатель графического вывода в нужную точку. Для этого надо присвоить соответствующие значения свойствам `CurrentX` и `CuttentY`. Например, в результате выполнения инструкций

```
st = "Visual Basic"
Picture1.CurrentX = _
          (Picture1.ScaleWidth - Picture1.TextWidth(st)) / 2
Picture1.CurrentY = 10
Picture1.Print st
```

в поле компонента `Picture1`, в его центре по горизонтали, появится текст "Visual Basic".

Контрольные вопросы

1. Какой компонент следует использовать для формирования графики в окне программы?
2. От какого угла отсчитываются координаты точек графической поверхности?
3. Как возрастает горизонтальная координата точек графической поверхности?
4. Как возрастает вертикальная координата точек графической поверхности?

Графические примитивы

Картинку, чертеж или схему можно представить как совокупность графических *примитивов*: точек, линий, окружностей, дуг, текста и др.

Рисование графических примитивов обеспечивают соответствующие методы (табл. 5.1).

Таблица 5.1. *Методы рисования графических примитивов*

Метод	Действие
`Line`	Рисует линию или прямоугольник
`Circle`	Рисует окружность, круг, эллипс, дугу или сектор
`PSet`	Рисует точку
`Print`	Выводит текст

Точка

Точку на графической поверхности рисует метод PSet.

Инструкция вызова метода PSet в общем виде выглядит так:

Объект.PSet(*x*,*y*), *Color*

Параметр *Объект* задает графическую поверхность, на которой надо нарисовать точку.

Параметры x и y задают координаты точки графической поверхности, на которой надо нарисовать точку.

Параметр Color задает цвет точки. В качестве параметра Color можно использовать одну из именованных констант (табл. 5.2).

Таблица 5.2. *Константы, используя которые можно задать цвет*

Константа	Цвет
vbBlack	Черный
vbRed	Красный
vbGreen	Зеленый
vbYellow	Желтый
vbBlue	Синий
vbMagenta	Пурпурный
vbCyan	Бирюзовый
vbWhite	Белый

В качестве параметра Color можно использовать также значение функции RGB, которая возвращает код цвета, полученного путем смешивания красной, зеленой и синей красок в указанных пропорциях. У функции RGB три параметра: первый задает долю красной (Red), второй — зеленой (Green), третий — синей (Blue) составляющей. Значение каждого из параметров должно находиться в диапазоне от 0 до 255. Например, значением RGB(205,127,50) является код "золотого" цвета.

Параметр Color необязательный. Если он не указан, то точка будет окрашена в цвет, заданный значением свойства ForeColor графической поверхности, на которой рисует метод.

Размер (диаметр) точки, которую рисует метод PSet, определяет текущее значение свойства DrawWidth поверхности, на которой рисует метод.

Контрольные вопросы

1. Как можно задать цвет точки, рисуемой методом `PSet`?
2. Если в инструкции вызова метода `PSet` цвет не указан явно, точка какого цвета будет нарисована?

Линия

Рисование прямой линии выполняет метод `Line`.

Инструкция вызова метода `Line` в общем виде выглядит так:

Объект.Line Step(*x1*,*y1*,)-Step(*x2*,*y2*), *Color*

Параметр *Объект* задает объект, на поверхности которого надо нарисовать линию. Если линию надо нарисовать на поверхности формы, то параметр *Объект* можно не указывать.

Параметры `x1` и `y1` задают координаты точки начала линии, а параметры `x2`, `y2` — координаты точки конца. Следует обратить внимание на то, что координаты точки начала линии можно не указывать. В этом случае линия будет нарисована из текущей точки графической поверхности в точку с указанными координатами.

Необязательный параметр `Color` задает цвет линии. Если он не указан, то линия будет нарисована цветом, заданным значением свойства `ForeColor` графической поверхности, на которой она рисуется.

Если перед параметрами `x1`, `x2` указано слово `Step`, то координаты точки начала отсчитываются относительно текущего положения указателя вывода. Если слово `Step` указано перед параметрами `x2`, `y2`, то координаты точки конца линии отсчитываются от точки ее начала.

Толщину и вид линии, рисуемой методом `Line`, определяют соответственно свойства `DrawWidth` и `DrawStyle` графической поверхности, на которой рисует метод. В табл. 5.3 приведены константы, используя которые можно задать вид линии. Следует обратить внимание, что линия толщиной более одного пиксела может быть только сплошной.

Таблица 5.3. *Константы, используя которые можно задать вид линии*

Константа	Вид линии
`vbSolid`	Сплошная
`vbDash`	Штриховая (длинные штрихи)
`vbDot`	Пунктирная (короткие штрихи)
`vbDashDot`	Штрихпунктирная
`vbDashDotDot`	Штрих, два пунктира

Использование метода `Line` демонстрирует программа "Сетка" (ее окно приведено на рис. 5.2, а текст — в листинге 5.1).

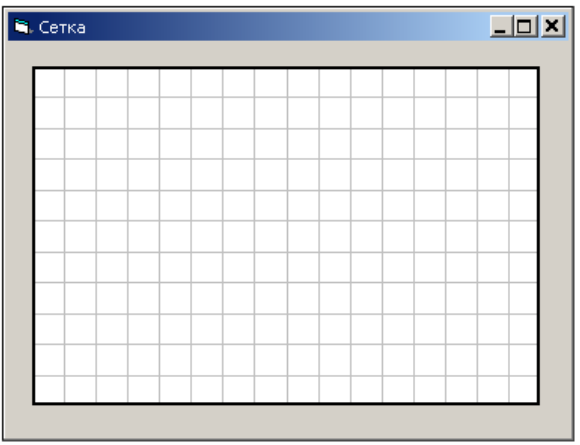


Рис. 5.2. Окно программы "Сетка"

Листинг 5.1. Сетка

```
Option Explicit

Private Sub Picture1_Paint()
 Dim x, y As Integer     ' координаты начала линии
 Dim dx, dy As Integer   ' шаг сетки
 Dim w, h As Integer     ' размер компонента Picture1

 x = 0: y = 0
 dx = 20: dy = 20
 w = Picture1.ScaleWidth - 1
 h = Picture1.ScaleHeight - 1

 ' Picture1.DrawStyle = vbDash
 Picture1.ForeColor = RGB(192, 192, 192) ' серый

 ' вертикальные линии
 Do
     Picture1.Line (x, y)-Step(0, h)
```

```
    x = x + dx
Loop While x < Picture1.ScaleWidth

' горизонтальные линии
x = 0: y = 0
Do
    Picture1.Line (x, y)-Step(w, 0)
    y = y + dy
Loop While y < Picture1.ScaleHeight

' окантовка
Picture1.ForeColor = vbBlack ' черный
Picture1.DrawWidth = 3       ' толщина линии

' переместить указатель графического вывода
' в точку (0, 0)
Picture1.CurrentX = 0
Picture1.CurrentY = 0

Picture1.Line -Step(w, 0)  ' верхняя граница
Picture1.Line -Step(0, h)  ' правая граница
Picture1.Line -Step(-w, 0) ' нижняя граница
Picture1.Line -Step(0, -h) ' левая граница

End Sub
```

Контрольные вопросы

1. Как можно задать цвет линии, рисуемой методом `Line`?
2. Если в инструкции вызова метода `Line` цвет не указан явно, линия какого цвета будет нарисована?
3. Как задать толщину линии, рисуемой методом `Line`?
4. Как задать вид линии, рисуемой методом `Line`?

Прямоугольник

Метод `Line` позволяет нарисовать не только линию, но и прямоугольник (рис. 5.3). Для того чтобы вместо линии был нарисован прямоугольник, после параметра `Color` надо указать константу `B`.

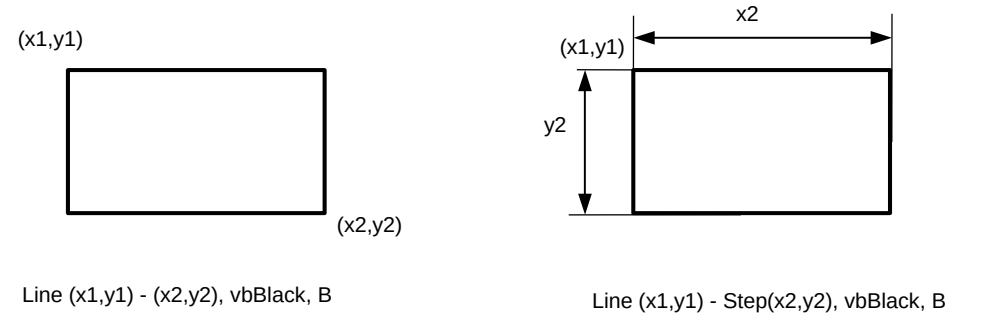


Рис. 5.3. Метод `Line` с параметром `B` рисует прямоугольник

Инструкция вызова метода `Line`, обеспечивающая рисование прямоугольника, в общем виде выглядит так:

Объект`.Line (`*x1*`,`*y1*`,)-(`*x2*`,`*y2*`), `*Color*`, B`

Параметр *Объект* задает графическую поверхность, на которой надо нарисовать прямоугольник (если прямоугольник надо нарисовать на поверхности формы, то параметр *Объект* можно не указывать).

Параметры `x1` и `y1` задают координаты левого (верхнего или нижнего) угла прямоугольника, а параметры `x2`, `y2` — координаты правого (нижнего или верхнего) угла.

Параметр `B` указывает, что надо нарисовать прямоугольник. Цвет границы задает параметр `Color`. Если он не указан, то цвет границы определяет значение свойства `ForeColor` графической поверхности, на которой рисует метод.

Толщину и вид линии границы прямоугольника определяют соответственно свойства `DrawWidth` и `DrawStyle` графической поверхности, на которой рисуется прямоугольник.

Если ширина границы больше, чем один пиксел, то размер прямоугольника (по внешней границе) будет больше, чем размер области, заданной параметрами `x1`, `y1` и `x2`, `y2`. Если необходимо, чтобы размер прямоугольника (по внешней границе) был равен размеру области, свойству `DrawStyle` следует присвоить значение `vbInsideSolid`. В этом случае линия границы смещена внутрь области прямоугольника, заданной точками (`x1`, `y1`) и (`x2`, `y2`).

Цвет и стиль закраски внутренней области прямоугольника определяют соответственно свойства `FillColor` и `FilStyle` той графической поверхности, на которой рисует метод. По умолчанию значение свойства `FillStyle` равно `vbFSTransparent`, поэтому метод рисует только границу прямоугольника. Чтобы внутренняя область прямоугольника была закрашена цветом, задан-

ным свойством FillColor, свойству FillStyle следует присвоить значение, отличное от vbFSTransparent, например vbFSSolid (сплошная заливка).

Константы, с помощью которых можно задать стиль закраски, приведены в табл. 5.4.

Таблица 5.4. *Стили закраски областей*

Константа	Способ (стиль) закраски
vbFSTransparent	Прозрачная. Внутренняя область не закрашивается
vbFSSolid	Сплошная закраска
vbHorizontalLine	Горизонтальная штриховка
vbVerticalLine	Вертикальная штриховка
vbUpwardDiagonal	Диагональная штриховка (наклон линий влево)
vbDownwardDiagonal	Диагональная штриховка (наклон линий вправо)
vbCross	Клетка
vbDiagonalCross	Диагональная клетка

Как и в случае рисования линии, перед параметрами x1, y1 и x2, y2 можно поставить слово Step. Если слово Step указано перед параметрами x1, y1, то координаты левого верхнего (нижнего) угла прямоугольника отсчитываются от текущего положения указателя графического вывода. Если слово Step указано перед параметрами x2, y2, то координаты правого нижнего (верхнего) угла прямоугольника отсчитываются от другого диагонального угла, т. е. фактически они задают размер прямоугольника (x2 — ширина, y2 — высота).

Если вместо параметра B указать параметр BF, то метод Line нарисует закрашенный прямоугольник, цвет границы и цвет закраски которого будут совпадать. Цвет прямоугольника в этом случае определяет параметр Color или, если он не указан, свойство ForeColor поверхности, на которой рисует метод. Следует обратить внимание, что после того, как метод Line нарисует прямоугольник, указатель графического вывода находится в точке (x2, y2).

Использование метода Line для рисования прямоугольника демонстрирует программа "Прямоугольник" (ее текст приведен в листинге 5.2). Она рисует на поверхности формы итальянский флаг (рис. 5.4).

Листинг 5.2. Прямоугольник

```
Option Explicit

Private Sub Form_Paint()
    Dim w, h As Integer ' размер полосы
    Dim x, y As Integer ' координаты левого верхнего угла
    Dim st As String

    w = 25
    h = 45

    x = (Form1.ScaleWidth - w * 3) / 2
    y = 20

    ' полосы
    Form1.Line (x, y)-Step(w, h), RGB(33, 94, 33), BF
    Form1.Line (x + w, y)-Step(w, h), vbWhite, BF
    Form1.Line (x + 2 * w, y)-Step(w, h), vbRed, BF

    ' контур
    Form1.FillStyle = vbFSTransparent
    Form1.Line (x, y)-Step(w * 3, h), vbBlack, B

    st = "Италия"
    CurrentX = (Form1.ScaleWidth - Form1.TextWidth(st)) / 2
    CurrentY = y + h + 10
    Print st
End Sub
```

Рис. 5.4. Прямоугольник нарисован методом `Line` с параметром `B`

Контрольные вопросы

1. Что надо сделать, чтобы метод Line нарисовал не линию, а прямоугольник?
2. Что надо сделать, чтобы метод Line нарисовал не линию, а закрашенный прямоугольник?
3. Как можно задать цвет границы прямоугольника, рисуемого методом Line?
4. Если в инструкции вызова метода Line цвет не указан явно, каким цветом будет нарисована граница прямоугольника?
5. Как задать толщину границы прямоугольника?
6. Как задать вид границы прямоугольника?
7. Как задать цвет и стиль закраски внутренней области прямоугольника?

Окружность и круг

Метод Circle, в зависимости от значения параметров, рисует окружность, эллипс, дугу или сектор. Инструкция вызова метода Circle, обеспечивающая рисование окружности или круга, в общем виде выглядит так:

Объект.Circle Step(*x*,*y*), *r*, *Color*

Параметр *Объект* задает поверхность, на которой надо нарисовать окружность (если окружность надо нарисовать на поверхности формы, то параметр *Объект* можно не указывать).

Параметры x и y задают координаты центра окружности. Если перед ними указано слово Step, то координаты центра отсчитываются от текущего положения указателя графического вывода.

Параметр r задает радиус окружности (круга).

Параметр Color задает цвет окружности или границы круга. Если он не указан, то цвет окружности (границы круга) определяет значение свойства ForeColor графической поверхности, на которой рисует метод.

Толщину и вид линии окружности определяют соответственно свойства DrawWidth и DrawStyle графической поверхности, на которой метод рисует.

Цвет и стиль закраски внутренней области окружности определяют соответственно свойства FillColor и FillStyle той графической поверхности, на которой рисует метод. Чтобы внутренняя область окружности была закрашена, значение свойства FillStyle должно быть отлично от vbFSTransparent.

Использование метода Circle демонстрирует программа "Олимпиада" (ее текст приведен в листинге 5.3). Процедура обработки события Paint рисует на поверхности формы символ Олимпийских игр (рис. 5.5). Следует обратить внимание на то, что явно указаны координаты только первой окружности, координаты каждой следующей отсчитываются от центра предыдущей.

Листинг 5.3. Олимпиада

```
Option Explicit

Private Sub Form_Paint()
    Dim x, y As Integer
    Dim r As Integer ' радиус окружности
    Dim st As String ' сообщение
    Dim w As Integer ' ширина области отображения текста

    r = 20
    x = Form1.ScaleWidth / 2 - 2.25 * r
    y = 40
    DrawWidth = 2
    FillStyle = vbFSTransparent ' область внутри окружностей прозрачная

    ' рисуем от левого кольца первого ряда
    Circle (x, y), r, RGB(50, 50, 205)

    ' второе кольцо сдвинуто относительно первого вправо на 2.25r
    Circle Step(2.25 * r, 0), r, vbBlack

    ' третье кольцо сдвинуто относительно второго вправо на 2.25r
    Circle Step(2.25 * r, 0), r, vbRed

    ' второй ряд колец рисуем справа налево
    ' правое кольцо второго ряда сдвинуто вниз и влево
    ' относительно правого кольца первого ряда
    Circle Step(-2.25 * r, 0), r, RGB(35, 142, 35)
    Circle Step(-1.125 * r, r), r, vbYellow

    st = "Быстрее, выше, сильнее!"
    Form1.Font.Size = 10

    w = Form1.TextWidth(st) ' ширина области отображения текста
    CurrentX = (Form1.ScaleWidth - w) / 2
    CurrentY = CurrentY + 2.25 * r
    Print st
End Sub
```

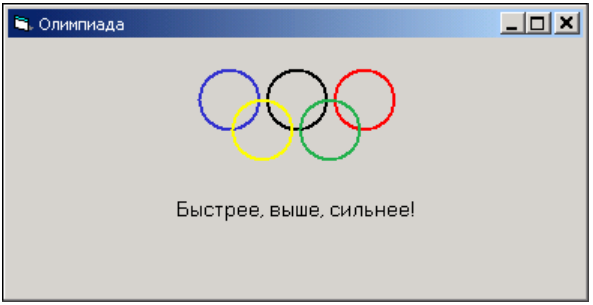


Рис. 5.5. Символ и девиз Олимпийских игр

Контрольные вопросы

1. Как можно задать цвет окружности?
2. Если в инструкции вызова метода `Circle` цвет не указан явно, каким цветом будет нарисована линия окружности?
3. Как задать толщину границы окружности?
4. Как задать вид границы окружности?
5. Как при помощи метода `Circle` нарисовать круг?
6. Как задать цвет и стиль закраски круга (внутренней области окружности)?

Дуга и сектор

Инструкция вызова метода `Circle`, обеспечивающая рисование дуги или сектора, в общем виде выглядит так:

Объект`.Circle Step(`*x*`,`*y*`),` *r*`,` *Color*`,` *start*`,` *fin*

Параметры `x`, `y`, `r` и `Color`, как и при рисовании окружности, определяют соответственно координаты центра окружности (круга), из которой вырезана дуга (сектор), радиус окружности (круга) и цвет линии дуги (границы сектора). Толщину и стиль линии дуги или границы сектора, а также цвет и стиль заливки внутренней области сектора определяют соответственно параметры `DrawWidth`, `DrawStyle`, `FillColor` и `FillStyle` графической поверхности, на которой рисует метод.

Параметр `start` задает начальную точку дуги — точку пересечения линии окружности и прямой, проведенной из центра окружности под углом `start` относительно оси *x*. Параметр `fin` задает конечную точку дуги (рис. 5.6). Дуга вычерчивается от начальной точки к конечной *против* часовой стрелки.

Угловые координаты `start` и `fin` измеряются в радианах и возрастают *против* часовой стрелки.

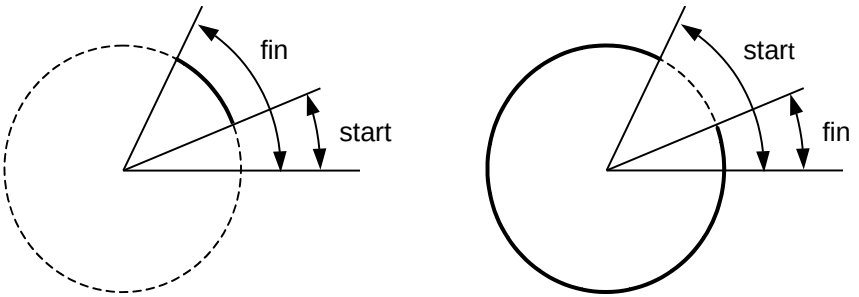


Рис. 5.6. Значения параметров *`start`* и *`fin`* определяют дугу

Если перед параметрами `start` и `fin` поставить минус, то будет нарисован сектор (рис. 5.7). Следует обратить внимание: чтобы нарисовать сектор из точки, соответствующей углу 0°, в качестве значения параметра `start` надо указать угол 360.

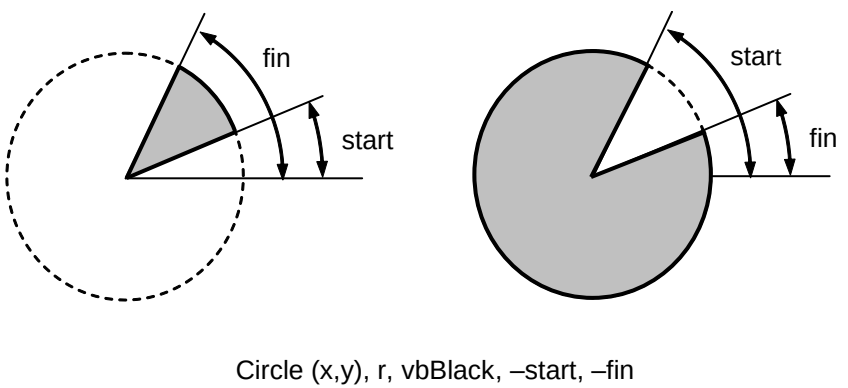


Рис. 5.7. Метод `Circle` позволяет нарисовать сектор

В качестве примера использования метода `Circle` в листинге 5.4 приведена программа, которая рисует на поверхности формы круговую диаграмму (рис. 5.8) — результат социологического опроса. Данные, отображаемые на диаграмме, загружаются из файла.

Рис. 5.8. Круговая диаграмма

Листинг 5.4. Круговая диаграмма

```
Option Explicit

Const N = 4                ' количество категорий
Dim title As String        ' заголовок диаграммы

' данные
Dim d(1 To N) As Double ' значение
Dim s(1 To N) As String ' подпись

Dim p(1 To N) As Double ' доля категории в общей сумме
Dim fc(1 To N) As Long  ' цвет сектора диаграммы


Private Sub Form_Activate()

    Dim f As Integer ' идентификатор файла

    Dim sum As Double
    Dim i As Integer


    ' чтение данных из файла
```

```
    f = FreeFile

    ' Open "d:\temp\result.txt" For Input As f
    Open Environ("APPDATA") + "\result.txt" For Input As f

    Line Input #f, title
    For i = 1 To N
        Line Input #f, s(i) ' название категории
        Input #f, d(i)      ' численное значение
    Next i
    Close #f

    fc(1) = vbGreen
    fc(2) = vbRed
    fc(3) = vbMagenta
    fc(4) = vbBlue

    ' вычислить общую сумму категорий
    sum = 0
    For i = 1 To N
        sum = sum + d(i)
    Next i

    ' вычислить долю каждой категории в общей сумме
    For i = 1 To N
        p(i) = d(i) / sum
    Next i
    Exit Sub

End Sub

Private Sub Form_Initialize()
    Form1.ScaleMode = vbPixels
End Sub

Private Sub Form_Paint()
    Dim r As Integer     ' радиус сектора
    Dim a1, a2 As Double ' углы прямых — границ сектора
```

```
Dim x, y As Integer  ' центр сектора


Dim i As Integer

Const k = 3.1415926 / 180 ' коэффициент пересчета величины угла
                          ' из градусов в радианы

' заголовок
Form1.Font.Size = 11
Form1.FontBold = True
CurrentY = 10
CurrentX = (Form1.ScaleWidth - Form1.TextWidth(title)) / 2
Print title

' диаграмма

Form1.Font.Size = 10
Form1.FontBold = False

r = 80
x = 2 * r ' Form1.ScaleWidth / 2
y = r + 40 ' Form1.ScaleHeight / 2

FillStyle = vbFSSolid

' секторы рисуем против часовой стрелки
' от горизонтали
a1 = 0
For i = 1 To N
    FillColor = fc(i)
    a2 = a1 + 360 * p(i) * k
    If a1 = 0 Then
        ' первый сектор рисуем от 360 до -a2
        Circle (x, y), r, , -6.2831852, -a2
    Else
        Circle (x, y), r, , -a1, -a2
    End If
```

```
        a1 = a2
    Next i

    ' легенда
    x = x + r + 20
    y = y - r + 20

    For i = 1 To N
        FillColor = fc(i)
        Line (x, y)-Step(20, 10), , B ' прямоугольник
        CurrentX = x + 30
        CurrentY = y
        Print s(i) + Format(p(i), " - #0.00%")
        y = y + 20
    Next i

End Sub
```

Контрольные вопросы

1. От какой точки отсчитываются углы, задающие точки начала и конца дуги?
2. Величина угла возрастает по часовой стрелке или против?
3. В каких единицах измеряются углы, указываемые в качестве параметров метода `Circle`, рисующего дугу (сектор)?
4. Что надо сделать, чтобы метод `Circle` нарисовал сектор, а не дугу?
5. Если в инструкции вызова метода `Circle` цвет не указан явно, каким цветом будет нарисована дуга (граница сектора)?
6. Как задать толщину дуги (границы сектора)?
7. Как задать вид дуги (границы сектора)?
8. Как задать цвет и стиль закраски сектора?

Эллипс

Метод `Circle` позволяет нарисовать также эллипс, эллиптическую дугу или эллиптический сектор. Инструкция вызова метода `Circle`, обеспечивающая рисование эллипса, в общем виде выглядит так:

Объект.Circle Step(*x*,*y*), *r*, *Color*, *start*, *fin*, *aspect*

Параметры `x` и `y` определяют координаты центра эллипса, `Color` — цвет границы. Толщину и стиль линии эллипса, дуги или сектора, а также цвет

и стиль заливки внутренней области эллипса (сектора) определяют соответственно параметры `DrawWidth`, `DrawStyle`, `FillColor` и `FillStyle` графической поверхности, на которой рисует метод. Параметры `start` и `fin` задают точки начала и конца эллиптической дуги. Параметр `r` задает больший радиус эллипса, а параметр `aspect` — коэффициент сжатия (трансформации). Если значение параметра `aspect` меньше единицы, то эллипс получается путем сжатия окружности по вертикали, если больше — по горизонтали (рис. 5.9). В случае, если значение параметра `aspect` равно 1, метод рисует окружность.

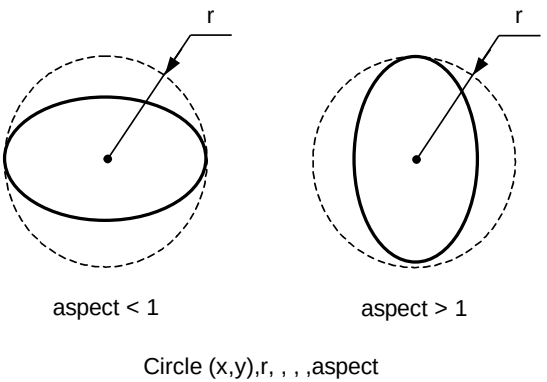


Рис. 5.9. Метод `Circle` позволяет нарисовать эллипс

В качестве примера использования метода `Circle` в листинге 5.5 приведена программа, которая рисует на поверхности формы глобус (рис. 5.10). Следует обратить внимание, что программа спроектирована таким образом, что глобус рисуется в центре окна. Кроме того, при изменении размера окна он также остается в центре окна (автоматически перерисовывается). Достигается это тем, что процедура обработки события `Resize` (оно возникает при изменении размера окна, например, в результате перемещения границы окна) вызывает метод `Refresh`, что в свою очередь приводит к возникновению события `Paint`, процедура обработки которого рисует глобус.

Листинг 5.5. Эллипс

```
Option Explicit

Private Sub Form_Initialize()
    Form1.ScaleMode = vbPixels
```

```
End Sub

Private Sub Form_Paint()

    Dim x, y As Integer ' центр окна

    x = Form1.ScaleWidth / 2
    y = Form1.ScaleHeight / 2

    FillColor = RGB(56, 176, 222) ' цвет — "Осеннее небо"
    FillStyle = vbSolid
    Circle (x, y), 40, , , , 1   ' круг

    FillStyle = vbFSTransparent ' область внутри эллипса не закрашивать
    Circle Step(0, 0), 40, , , , 0.5 ' "горизонтальный" эллипс
    Circle Step(0, 0), 40, , , , 0   ' линия
    Circle Step(0, 0), 40, , , , 1.5 ' "вертикальный" эллипс
    Circle Step(0, 0), 40, , , , 3   ' "вертикальный" эллипс

End Sub

' пользователь изменил размер окна
Private Sub Form_Resize()
    Form1.Refresh ' перерисовать окно программы (инициировать
                  ' событие Paint)
End Sub
```

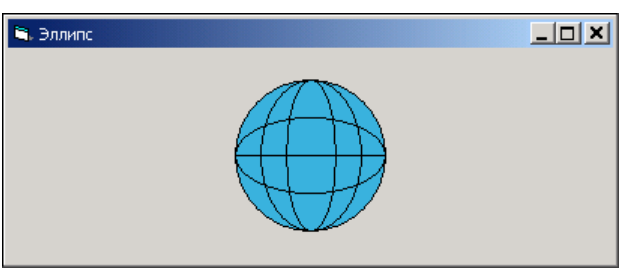


Рис. 5.10. Метод `Circle` позволяет нарисовать эллипс

Контрольные вопросы

1. Что надо сделать, чтобы метод `Circle` нарисовал эллипс (эллиптическую дугу, эллиптический сектор), а не окружность?
2. Как задать цвет границы эллипса?
3. Как задать цвет и стиль закраски внутренней области сектора?

Текст

Вывод текста на графическую поверхность выполняет метод `Print`. Инструкция вызова метода в общем виде выглядит так:

Объект`.Print` *Строка*

Параметр *Строка* задает строку, которую надо вывести. Позицию, в которой появится строка, определяет текущее положение указателя графического вывода. Таким образом, чтобы строка появилась в нужном месте формы, перед тем как вызвать метод `Print`, надо установить указатель графического вывода (присвоить значение свойствам `CurrentX` и `CurrentY`) в ту точку формы, от которой надо вывести текст.

Например:

```
CurrentX = 20
CurrentY = 20
Print "Microsoft Visual Basic"
```

Результат выполнения этих инструкций приведен на рис. 5.11.

После того как строка будет выведена, указатель графического вывода автоматически перемещается в точку (`0`, `y`). Значение координаты `y` зависит от высоты области отображения текста, размер которой, в свою очередь, определяется характеристиками шрифта, используемыми для вывода текста.

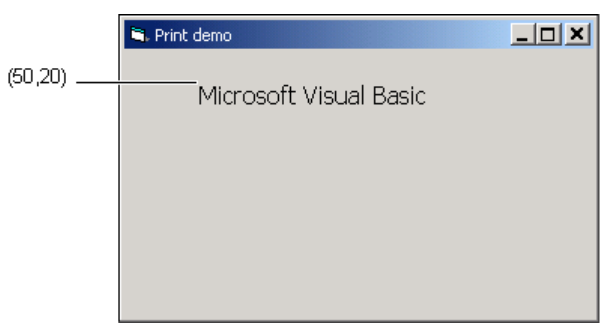


Рис. 5.11. Метод `Print` выводит текст от текущего положения (`CurrentX`, `CurrentY`) указателя графического вывода

Шрифт, который метод Print использует для отображения текста, определяет свойство Font графической поверхности, цвет символов — свойство ForeColor.

Если в качестве параметра метода Print указать переменную или выражение, тип которого отличается от String, то метод автоматически выполнит преобразование значения в строку. Например, если в программе объявлена переменная Today типа Date, то в результате выполнения инструкций

```
Today = Now ' значение функции Now — текущая дата и время
CurrentX = 10
CurrentY = 10
Print Today
```

в форму будет выведена текущая дата и время.

Для преобразования значения выражения в строку нужного формата весьма полезна функция Format. У функции два параметра: выражение, значение которого надо преобразовать в строку, и строка форматирования. Например, значением функции Format(Sum, "### ### ##0.00 руб.") является строковое представление значения переменной Sum в денежном формате — цифры объединены в группы по три, группы разделены пробелами, после десятичного разделителя отображаются две цифры (даже если дробная часть числа равна нулю), и в конец строки добавляется обозначение денежной единицы. Следует обратить внимание, что приведенная строка форматирования обеспечивает автоматическое округление числа до двух значащих цифр дробной части. Например, если значение переменной Sum равно 28580,446, то значением функции Format(Sum, "### ### ##0.00 руб.") будет строка 28 580,45 руб.

Наиболее часто используемые форматы приведены в табл. 5.5.

Таблица 5.5. *Форматы отображения данных*

Формат	Описание	Пример
#.##	Дробное число с двумя знаками после запятой (десятичного разделителя). Если дробная часть числа равна нулю, то цифры дробной части (нули) не отображаются	Format(x, "#.##")
#0.00	Дробное число с двумя знаками после запятой (десятичного разделителя). Выполняется округление	Format(x, "#0.00")

Таблица 5.5 *(окончание)*

Формат	Описание	Пример
`### ### ##0.00`	Дробное число с двумя знаками после десятичного разделителя, цифры целой части объединены в группы по три и разделены пробелами. Выполняется округление. Используется для отображения денежных величин (в этом случае в конец строки форматирования обычно добавляют обозначение денежной единицы)	`Format(sum, _ "### ### ##0.00 руб.")`
`#0.00%`	Процент. Значение автоматически умножается на 100, и в конец строки добавляется символ процента	`Format(discount, "#0.00%")`
`dd/mm/yy`	Дата в формате "день, месяц, год". Символ-разделитель определяет операционная система	`Format(Now,"dd/mm/yy")`
`dddd`	День недели	`Format(Now,"dd/mm, dddd")`
`mmmm`	Месяц в полном формате	`Format(Now,"dd mmmm, dddd")`

Помимо символов формата строка форматирования может содержать обычный текст. Например, если `Discount` = 285, то значением функции

```
Format(Discount, "Скидка: ### ### ##0.00 руб.")
```

является строка

```
Скидка: 285,00 руб.
```

Использование метода `Print` для вывода текста на поверхность формы демонстрирует программа Print demo (текст приведен в листинге 5.6). В окне программы отображается приветствие и информация о текущей дате (рис. 5.12). Следует обратить внимание, что текст расположен в центре окна. Координаты точки, от которой выводится текст, вычисляются на основе информации о размере области, необходимой для отображения текста. Метод `TextWidth`, которому в качестве параметра передается строка для отображения, возвращает ширину области, которую займет текст на поверхности формы; метод `TextHeight`, соответственно, — высоту.

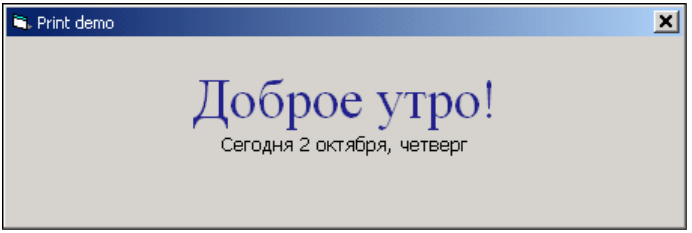


Рис. 5.12. Текст расположен в центре окна

Листинг 5.6. Приветствие (Print demo)

```
Private Sub Form_Load()
       ScaleMode = vbPixels
End Sub

Private Sub Form_Paint()

    Dim st As String ' текст
    Dim w As Integer ' ширина области отображения текста
    Dim h As Integer ' высота области отображения текста

    Dim t As Integer ' время (часы)

    t = Hour(time)

    Select Case t
        Case 0 To 4
            st = "Доброй ночи!"
        Case 5 To 12
            st = "Доброе утро!"
        Case 13 To 16
            st = "Добрый день!"
        Case 17 To 23
            st = "Добрый вечер!"
    End Select


   Font.Name = "Times New Roman"
```

```
    FontSize = 28
    ForeColor = RGB(35, 35, 142)

    ' определить размер области отображения текста
    w = TextWidth(st)
    h = TextHeight(st)

    ' установить указатель графического вывода так,
    ' чтобы текст находился в центре окна
    '
    CurrentX = (ScaleWidth - w) / 2  ' ScaleWidth — ширина формы
                                     '              в пикселах
    CurrentY = ScaleHeight / 2 - h   ' ScaleHeight — высота формы
                                     '               в пикселах

    Print st ' вывести текст от точки (CurrentX,CurrentY)

    Font.Name = "Tahoma"
    FontSize = 10
    ForeColor = vbBlack

    st = Format(Now, "Сегодня d mmmm, dddd")
    w = TextWidth(st)
    CurrentX = (ScaleWidth - w) / 2
    Print st

End Sub
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как задать шрифт, используемый методом `Print` для отображения текста на графической поверхности?
2. Как задать позицию, от которой метод `Print` выведет текст на графическую поверхность?
3. Как определить размер области, которую займет текст, выведенный на графическую поверхность?

Иллюстрации

Отображение иллюстраций обеспечивают компоненты PictureBox и Image (рис. 5.13). Они отображают иллюстрации форматов BMP, GIF, JPEG и PNG, метафайлы (WMF, EMF) и значки (ICO).

Различие компонентов PictureBox и Image состоит в том, что компонент PictureBox обеспечивает отображение иллюстраций, размер которых не больше размера компонента (если размер иллюстрации больше, чем размер компонента, то отображается только часть иллюстрации), а компонент Image, обладая возможностью масштабирования, может отображать иллюстрации любого размера. Кроме того, на поверхности компонента PictureBox можно рисовать.

Основные свойства компонента PictureBox приведены в табл. 5.6.

Рис. 5.13. Компоненты PictureBox и Image обеспечивают отображение иллюстраций

Таблица 5.6. *Свойства компонента* PictureBox

Свойство	Описание
Picture	Иллюстрация, отображаемая в поле компонента (свойства Width и Height объекта Picture содержат информацию о реальном размере иллюстрации, отображаемой в поле компонента)
AutoSize	Признак автоматического изменения размера компонента в соответствии с размером иллюстрации. Если значение свойства равно True, то размер компонента соответствует размеру иллюстрации
Visible	Признак отображения компонента на поверхности формы. Позволяет скрыть компонент (False) или сделать его видимым (True)
Appearance	Стиль компонента. Поле компонента может быть как бы приподнято над поверхностью формы (в этом случае значение свойства равно 3D) или находиться на одном уровне с поверхностью формы (в этом случае значение свойства равно Flat)
BorderStyle	Вид границы компонента. Граница может быть тонкой (FixedSingle) или отсутствовать (None)

Таблица 5.6 *(окончание)*

Свойство	Описание
ScaleMode	Единица измерения размеров компонента и объектов на нем
ScaleWidth	Ширина рабочей области компонента, т. е. без учета ширины левой и правой границ
ScaleHeight	Высота рабочей области компонента, т. е. без учета ширины нижней и верхней границ компонента
Left	Расстояние от левой границы компонента до левой границы формы
Top	Расстояние от верхней границы компонента до верхней границы формы
Width	Ширина компонента с учетом ширины границы
Height	Высота компонента с учетом ширины границы

Задать иллюстрацию, которую надо отобразить в поле компонента PictureBox, можно как во время создания формы приложения, так и во время работы программы.

Чтобы задать иллюстрацию во время разработки формы, надо в строке свойства Picture щелкнуть на кнопке с тремя точками (рис. 5.14) и в диалоговом окне **Load Picture** выбрать файл иллюстрации. Следует обратить внимание, что файл, в котором находится иллюстрация, выбранная таким образом, во время работы программы не нужен (Visual Basic помещает копию иллюстрации в выполняемый файл программы).

Чтобы размер компонента соответствовал размеру иллюстрации, свойству AutoSize надо присвоить значение True.

Загрузку иллюстрации в поле компонента PictureBox во время работы программы обеспечивает функция LoadPicture. В качестве параметра функции надо указать файл иллюстрации, а значение функции — присвоить свойству Picture. Например, в результате выполнения инструкции

```
Picture1.Picture = LoadPicture("d:\images\ufo.bmp")
```

в поле компонента Picture1 появляется иллюстрация, находящаяся в указанном файле.

Для отображения больших иллюстраций, например фотографий, следует использовать компонент Image.

Основные свойства компонента Image приведены в табл. 5.7.

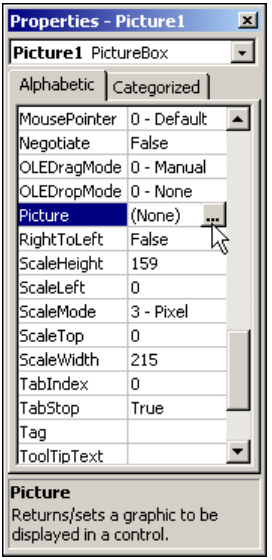


Рис. 5.14. Чтобы выбрать иллюстрацию, щелкните на кнопке с тремя точками в строке свойства `Picture`

Таблица 5.7. *Свойства компонента* `Image`

Свойство	Описание
`Picture`	Иллюстрация, отображаемая в поле компонента
`Stretch`	Признак необходимости выполнить масштабирование иллюстрации так, чтобы она занимала всю область отображения иллюстрации. Масштабирование выполняется, если значение свойства равно `True`
`Width`	Ширина компонента (области отображения иллюстрации) с учетом ширины границы
`Height`	Высота компонента с учетом ширины границы
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Visible`	Признак отображения компонента. Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)
`Appearance`	Стиль компонента. Поле компонента может быть как бы приподнято над поверхностью формы (в этом случае значение свойства равно `3D`) или находиться на одном уровне с поверхностью формы (в этом случае значение свойства равно `Flat`)
`BorderStyle`	Вид границы компонента. Граница может быть тонкой (`FixedSingle`) или отсутствовать (`None`)

Задать иллюстрацию, отображаемую в поле компонента `Image`, можно как во время создания формы (присвоить значение свойству `Picture`), так и во время работы программы (вызвать функцию `LoadPicture`). Следует обратить внимание, что после загрузки в компонент `Image` иллюстрации (если значение свойства `Stretch` равно `False`) размер компонента автоматически изменяется и соответствует размеру иллюстрации.

Использование компонента `Image` демонстрирует программа "Просмотр иллюстраций" (ее форма приведена на рис. 5.15). Программа позволяет выбрать каталог и просмотреть фотографии (JPG-файлы), которые в нем находятся.

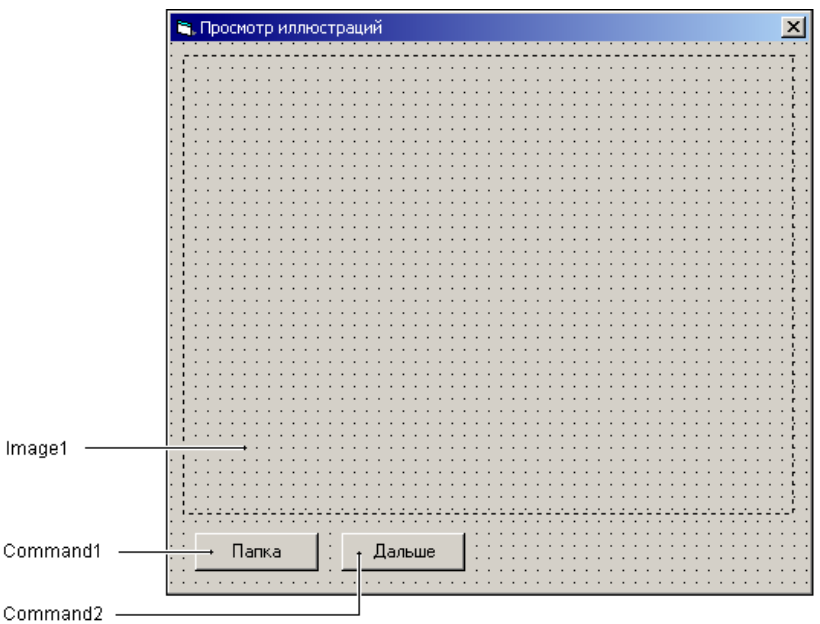


Рис. 5.15. Форма программы "Просмотр иллюстраций"

Выбор каталога осуществляется в окне **Обзор папок** (рис. 5.16), которое появляется на экране в результате щелчка на кнопке **Папка**. Переход к следующей фотографии выполняется щелчком на кнопке **Дальше**.

Значения свойств формы и компонента `Image` приведены в табл. 5.8 и 5.9, текст — в листинге 5.7.

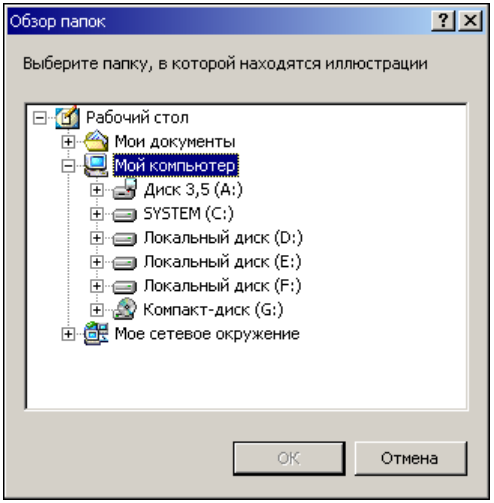


Рис. 5.16. Выбор папки, в которой находятся иллюстрации, выполняется в окне **Обзор папок**

Таблица 5.8. *Значения свойств формы*

Свойство	**Значение**
`Form1.BorderStyle`	`FixedSingle`
`Form1.ScaleMode`	`Pixel`

Таблица 5.9. *Значения свойств компонента* `Image1`

Свойство	**Значение**
`Width`	`400`
`Height`	`300`
`Stretch`	`False`

Листинг 5.7. Просмотр иллюстраций

```
Option Explicit

Private Declare Function SHBrowseForFolder Lib "shell32" _
   Alias "SHBrowseForFolderA" (ByRef b As Any) As Long
Private Declare Function SHGetPathFromIDList Lib "shell32" _
```

```
  (ByVal ResPIDL As Any, ByVal patch As String) As Long

' константы SHELL API
Const CSIDL_DRIVES As Long = 17
Const BIF_RETURNONLYFSDIRS As Long = 1
Const MAX_PATH = 260

' эта структура используется для передачи информации
' в функцию SHBrowseFolder, которая выводит диалоговое
' окно Обзор папок
Private Type bi ' browseinfo
    hwndOwner As Long
    pidlRoot As Long
    pszDisplayName As String ' выбранная папка (без пути)
    lpzTitle As String       ' подсказка
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type

Dim Folder As String  ' папка, в которой находятся иллюстрации
Dim fn As String      ' файл иллюстрации
Dim ImageW, ImageH As Integer   ' исходный размер компонента Image

Private Sub Form_Load()
    ' так как размер иллюстрации измеряется в твипах,
    ' установим, что размер компонентов измеряется
    ' тоже в твипах

    Form1.ScaleMode = 1 ' ScaleMode — Twip

    ' запомнить размер компонента Image1
    ImageW = Image1.Width
    ImageH = Image1.Height

    ' Environ("HOMEDRIVE") — системный диск
    ' Environ("HOMEPATH") — каталог пользователя
```

```
    ' В Windows XP это Documents and Settings\User,
    ' где User — имя пользователя

    Folder = Environ("HOMEDRIVE") + _
             Environ("HOMEPATH") + _
             "\Мои документы\Мои рисунки\"

    fn = Dir(Folder + "*.jpg") ' получить имя JPG-файла

    If fn = "" Then
        MsgBox "В каталоге" & Folder & " JPG-иллюстраций нет", _
               vbExclamation, "Просмотр иллюстраций"
        Exit Sub
    End If

    DisplayPic (Folder + fn) ' вывести иллюстрацию

    fn = Dir ' получить имя файла следующего JPG-файла
    If fn = "" Then
        ' в каталоге Folder больше нет файлов с расширением jpg
        Command2.Enabled = False
    Else
        Command2.Enabled = True
    End If

End Sub

' выводит в поле компонента Image1 иллюстрацию
Private Sub DisplayPic(fn As String)
    ' коэффициенты масштабирования иллюстрации:
    Dim kx As Single ' по X
    Dim ky As Single ' по Y
    Dim k  As Single ' общий

    ' fn — имя файла иллюстрации
    Image1.Visible = False
```

```
    Image1.Stretch = False ' чтобы знать истинный размер иллюстрации
    Image1.Picture = LoadPicture(fn)

    ' здесь Image1.Picture.Width и Image1.Picture.Height — размер
    ' иллюстрации (в твипах)

    If (Image1.Picture.Width <= ImageW) And _
       (Image1.Picture.Height <= ImageH) Then
        ' размер иллюстрации меньше размера компонента
        Image1.Visible = True
     Else
        ' иллюстрация больше, чем компонент Image
        ' надо масштабировать
        Image1.Stretch = True
        ' вычислим коэффициент масштабирования
        kx = ImageW / Image1.Picture.Width
        ky = ImageH / Image1.Picture.Height
        ' чтобы иллюстрация отображалась без искажения,
        ' коэффициенты масштабирования по обеим осям
        ' должны быть равными
        If kx < ky Then k = kx Else k = ky
        Image1.Width = Image1.Picture.Width * k
        Image1.Height = Image1.Picture.Height * k
        Image1.Visible = True
     End If
End Sub

' щелчок на кнопке Дальше
Private Sub Command2_Click()
    DisplayPic (Folder + fn) ' вывести иллюстрацию
    ' получить имя файла следующей иллюстрации
    fn = Dir
    If fn = "" Then
        ' в каталоге Folder больше нет файлов с расширением jpg
        Command2.Enabled = False
    End If
End Sub
```

```
' щелчок на кнопке Папка
Private Sub Command1_Click()

    ' чтобы не потерять имя просматриваемого каталога (в случае, если
    ' пользователь активизирует процесс выбора папки, а затем откажется),
    ' используем буферную переменную. Если пользователь действительно
    ' выберет новый каталог, то запишем имя этого каталога в переменную
    ' Folder

    Dim buf As String
    buf = GetFolder()

    If buf = "" Then Exit Sub

    ' пользователь ввел имя папки
    Folder = buf

    ' получить имя JPG-файла
    fn = Dir(Folder + "*.jpg")

    If fn = "" Then
        MsgBox "В каталоге " & Folder & " нет JPG-иллюстраций", _
                vbExclamation, "Просмотр иллюстраций"
        Exit Sub
    End If

    DisplayPic (Folder + fn) ' вывести иллюстрацию
                             ' в поле компонента Image1

    ' получить имя файла следующей иллюстрации
    fn = Dir
    If fn = "" Then
        ' в каталоге Folder больше нет файлов с расширением jpg
        Command2.Enabled = False
    Else
        Command2.Enabled = True
    End If
```

```
End Sub

' возвращает имя папки, выбранной пользователем в
' окне Обзор папок
Public Function GetFolder() As String

    Dim t As bi
    Dim ResPIDL As Long
    Dim Path As String

    Dim r As Long
    Dim p As String

    t.hwndOwner = Form1.hwnd
    t.lpzTitle = "Выберите папку, в которой находятся иллюстрации"
    t.ulFlags = BIF_RETURNONLYFSDIRS ' кнопка ОК доступна,
                                     ' если пользователь выбрал папку
    t.pszDisplayName = String(MAX_PATH, 0)

    ResPIDL = SHBrowseForFolder(t) ' вывести окно Обзор папок
    If ResPIDL <> 0 Then
        ' пользователь выбрал папку
        ' получить ее полное имя
        Path = String(MAX_PATH, Chr(0))
        r = SHGetPathFromIDList(ResPIDL, Path)
        p = InStr(1, Path, Chr(0))
        Path = Mid(Path, 1, p - 1)
        If Mid(Path, p - 1, 1) <> "\" Then Path = Path + "\"
    End If
    GetFolder = Path
End Function
```

В начале работы программы процедура обработки события `Load` записывает в переменные `ImageW` и `ImageH` размер компонента `Image1`. Эта информация используется для вычисления коэффициента масштабирования. Затем она вызывает функцию `Dir`, которая возвращает имя JPG-файла (если он есть), находящегося в каталоге Мои рисунки. После этого вызывается процедура `DisplayPic`, которая выводит в поле компонента `Image1` иллюстрацию.

Сначала она присваивает значение `True` свойству `Visible` (тем самым делает компонент `Image` невидимым), отменяет режим автоматического изменения размера иллюстрации и загружает иллюстрацию. Загрузку иллюстрации выполняет функция `LoadPicture`. Если размер загруженной иллюстрации меньше, чем размер компонента `Image1`, то свойству `Visible` присваивается значение `True`, компонент `Image1` становится видимым и иллюстрация появляется в окне программы. Если размер иллюстрации больше, чем размер компонента, то выполняется масштабирование. Здесь надо обратить внимание, что для выполнения масштабирования не достаточно присвоить значение `True` свойству `Stretch` (если размер иллюстрации не пропорционален размеру компонента, то иллюстрация будет искажена). Чтобы иллюстрация отображалась без искажения, размер компонента `Image1` должен быть пропорционален размеру иллюстрации. Поэтому после загрузки иллюстрации надо выбрать коэффициент масштабирования. В рассматриваемой программе в качестве коэффициента масштабирования выбирается минимальный из коэффициентов масштабирования по ширине и высоте. Для вычисления коэффициентов масштабирования используется информация о первоначальном размере компонента `Image1` (значение переменных `ImageW` и `ImageH`) и размере загруженной иллюстрации (значение свойств `Picture.Width` и `Picture.Height` компонента `Image1`). После вычисления коэффициента масштабирования процедура `DisplayPic` устанавливает размер компонента пропорциональным размеру загруженной иллюстрации, присваивает значение `True` свойствам `Stretch` и `Visible`. В результате иллюстрация отображается в поле компонента.

Процедура обработки события `Click` на кнопке **Дальше** обращается к функции `Dir`, чтобы получить имя файла следующей иллюстрации. Так как при этом имя папки не указывается, то значением функции является имя следующего JPG-файла, находящегося в каталоге, указанном при первом вызове процедуры `Dir`.

Процедура обработки события `Click` на кнопке **Папка** вызывает функцию `GetFolder`, которая выводит стандартное окно **Обзор папок**. Непосредственный вывод окна выполняет API-функция `SHBrowseForFolder`. После того как пользователь выберет папку, функция `GetFolder` возвращает вызвавшей ее процедуре имя папки. Затем функция `Dir` выполняет поиск файла с расширением JPG. Если в выбранной пользователем папке есть файлы с указанным расширением, то функция `Dir` возвращает имя первого по порядку файла (чтобы получить имя следующего файла, надо вызвать функцию `Dir` еще раз, но уже без параметров).

Контрольные вопросы

1. Какой компонент следует использовать для формирования графики (например, столбчатой диаграммы) во время работы программы?
2. Какой компонент следует использовать для отображения иллюстраций, например, фотографий?
3. Какая функция обеспечивает загрузку иллюстрации из файла в компонент `Image`?
4. Если размер иллюстрации больше размера компонента `Image`, что надо сделать, чтобы иллюстрация отображалась полностью и без искажений?
5. Как узнать истинный размер иллюстрации?

Битовые образы

Для формирования сложных изображений, например при программировании игр, используют *битовые образы*. Битовый образ (bitmap) — это прямоугольная картинка, которая находится в памяти компьютера. Отличительной особенностью битового образа является то, что в памяти он хранится в том виде, в каком отображается на экране. Поэтому его можно очень быстро вывести на экран.

Создать битовый образ можно путем загрузки картинки из файла или *ресурса*. Также битовый образ можно сформировать во время работы программы, скопировав фрагмент другого битового образа.

В Visual Basic битовый образ — это объект типа `StdPicture`. Свойства объекта `StdPicture` (табл. 5.10) содержат информацию о битовом образе.

Таблица 5.10. *Свойства объекта* `StdPicture`

Свойство	Описание
`Type`	Тип битового образа. Битовый образ может представлять собой картинку, значок или метафайл
`Width`	Ширина картинки
`Height`	Высота картинки
`Handle`	Идентификатор объекта. Используется API-функциями для доступа к битовому образу

Загрузку картинки из файла в битовый образ обеспечивает функция `LoadPicture`. Например, следующий фрагмент кода загружает картинку из файла ufo.bmp в битовый образ `ufo`:

```
Dim ufo As StdPicture
Set ufo = LoadPicture("d:\pictures\ufo.bmp")
```

После того как битовый образ сформирован (загружен из файла или из ресурса), его можно вывести на поверхность формы или компонента PictureBox. Сделать это можно с помощью метода PaintPicture.

Инструкция вызова метода PaintPicture в общем виде выглядит так:

Объект.PaintPicture *aBitmap*, *x1*, *y1*, *w1*, *h1*, *x2*, *y2*, *w2*, *h2*, *opcode*

Параметры x2, y2, w2 и h2 задают фрагмент битового образа, который надо вывести на поверхность объекта *Объект*. Параметры x1, y1, w1 и h1 задают положение и размер области на поверхности объекта, в которую надо вывести фрагмент битового образа, заданный параметрами x2, y2, w2 и h2. Таким образом, метод PaintPicture позволяет вывести на графическую поверхность любой фрагмент битового образа. Параметр opcode определяет, каким образом формируется цвет точек результирующего изображения. Например, если значение параметра равно vbSrcCopy, то фрагмент битового образа замещает фрагмент графической поверхности.

В простейшем случае в инструкции вызова метода PaintPicture достаточно указать битовый образ и координаты точки поверхности, от которой надо вывести картинку. Например, инструкция

```
Form1.PaintPicture ufo,10,20
```

выводит на поверхность формы битовый образ ufo — изображение UFO — "неопознанного летающего объекта".

Метод PaintPicture не позволяет задать "прозрачный" цвет, поэтому на графической поверхности битовый образ отображается "как есть" (рис. 5.17).

Рис. 5.17. Битовый образ и его изображение на графической поверхности

Хотя, как было сказано ранее, метод PaintPicture не позволяет задать "прозрачный" цвет, тем не менее, область вокруг объекта все-таки можно сделать

прозрачной. Для этого в дополнение к битовому образу, в котором находится изображение объекта, надо создать маску — битовый образ, в котором точки, соответствующие точкам объекта, должны быть окрашены в черный цвет, а точки фона — в белый. Кроме того, в битовом образе точки фона также должны быть окрашены в черный цвет. В качестве примера на рис. 5.18 приведен битовый образ и соответствующая ему маска.

Рис. 5.18. Битовый образ и маска

Чтобы получить эффект прозрачного фона, надо на графическую поверхность сначала с помощью метода `PaintPicture` вывести маску, затем — битовый образ. При выводе маски в качестве значения параметра *opcode* метода `PainPicture` следует указать константу `vbSrcAnd`, при выводе битового образа — `vbSrcPaint`.

Следующая программа (листинг 5.8) демонстрирует, как с помощью маски можно добиться эффекта прозрачного фона. В программе используются два битовых образа: самолет и маска. Загрузку битовых образов выполняет процедура обработки события `Load`. Самолет рисует процедура обработки события `Paint`. Сначала она выводит на поверхность формы маску, затем — самолет. Командные кнопки **Маска** и **Bitmap** позволяют увидеть процесс формирования картинки. Щелчок на кнопке **Маска** выводит изображение маски, щелчок на кнопке **Bitmap** накладывает на изображение маски битовый образ. Окно программы после щелчка на кнопках **Маска** и **Bitmap** приведено на рис. 5.19.

Листинг 5.8. Прозрачный фон

```
Option Explicit

' битовые образы
Dim plane As StdPicture     ' картинка
Dim plane_m As StdPicture   ' маска

Private Sub Form_Load()
    ' загрузить битовые образы
    Set plane = LoadPicture("d:\pictures\plane.bmp")
    Set plane_m = LoadPicture("d:\pictures\plane_m.bmp")
End Sub
```

```
Private Sub Form_Paint()
    ' вывести маску
    Form1.PaintPicture plane_m, 50, 80, , , , , , , vbSrcAnd
    ' вывести картинку
    Form1.PaintPicture plane, 50, 80, , , , , , , vbSrcPaint
End Sub

' щелчок на кнопке Маска
Private Sub Command1_Click()
    Form1.PaintPicture plane_m, 200, 170, , , , , , , vbSrcAnd
End Sub

' щелчок на кнопке Bitmap
Private Sub Command2_Click()
    Form1.PaintPicture plane, 200, 170, , , , , , , vbSrcPaint
End Sub
```

Рис. 5.19. При использовании маски можно добиться эффекта прозрачного фона

Контрольные вопросы

1. Что такое "битовый образ"?
2. Файлы какого типа используются для хранения битовых образов?
3. Какая функция обеспечивает загрузку битового образа из файла?
4. Какой метод обеспечивает отображение (вывод) битового образа на графическую поверхность?

Мультипликация

Мультипликацией (или анимацией) называют изображение, элементы которого движутся.

Существуют два способа реализации компьютерной анимации. Первый способ (его используют создатели рисованных мультфильмов) предполагает наличие заранее подготовленной серии картинок (кадров), последовательное отображение которых и создает эффект анимации. Второй подход (который используют разработчики компьютерных игр) предполагает создание кадров анимации "на лету". При реализации этого подхода очередной кадр создается путем вывода изображения *объекта* в нужную точку "экрана". При этом изображение объекта может быть сформировано из графических примитивов или загружено из файла.

Программа "Пинг-понг" (ее форма приведена на рис. 5.20, а значения свойств формы — в табл. 5.11) демонстрирует принципы реализации анимации и показывает, как можно заставить объект, в данном случае мячик, двигаться в окне программы.

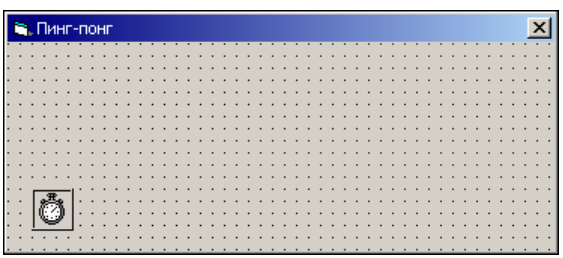


Рис. 5.20. Форма программы "Пинг-понг"

Таблица 5.11. *Значения свойств формы программы "Пинг-понг"*

Свойство	Значение
BorderStyle	FixedSingle
ScaleMode	Pixel

Чтобы у наблюдателя сложилось впечатление, что объект движется, надо вывести изображение объекта, затем, через некоторое время, "стереть" его и снова вывести, но уже на некотором расстоянии от предыдущего положения. Подбором времени между выводом и удалением изображения, а также

расстояния между новым и предыдущим положением (шага перемещения) можно добиться эффекта равномерного движения.

В рассматриваемой программе (ее текст приведен в листинге 5.9) основную работу выполняет процедура обработки события `Timer`. Она стирает мячик (окружность) и рисует его на новом месте. Направление движения мячика задают переменные `dx` и `dy`. Если значение `dx` (`dy`) положительное, то мячик движется слева направо (сверху вниз), если отрицательное — то справа налево (снизу вверх). При соприкосновении мячика с вертикальной или горизонтальной границей окна знак `dx` или `dy` меняется на противоположный, в результате мячик "отскакивает" от границы и движется в противоположном направлении. Скорость движения объекта определяют абсолютные значения `dx` и `dy`, а также период возникновения события `Timer`.

Листинг 5.9. Пинг-понг

```
Option Explicit

Dim x, y As Integer     ' положение мячика
Dim dx, dy As Integer   ' приращение координат
Dim r As Integer        ' радиус мячика
Dim cBall As Long       ' цвет мячика
Dim cBack As Long       ' цвет поля
Dim wp, hp As Integer   ' размер поля (формы)

Private Sub Form_Load()
    r = 2
    x = r
    y = 50
    dx = 1
    dy = 1
    cBall = RGB(217, 217, 25) ' ярко-золотой
    cBack = RGB(33, 94, 33)   ' зеленый "охотничий"

    Form1.BackColor = cBack

    wp = Form1.ScaleWidth
    hp = Form1.ScaleHeight

    ' настройка и запуск таймера
    Timer1.Interval = 10
```

```
    Timer1.Enabled = True

End Sub

' сигнал таймера
Private Sub Timer1_Timer()

    ' стереть изображение мяча
    Form1.Circle (x, y), r, cBack

    ' ** вычислить новое положение мяча **
    If dx > 0 Then
        ' мяч движется вправо
        If x + dx + r > wp Then dx = -dx
    Else
        ' мяч движется влево
        If x + dx - r < 0 Then dx = -dx
    End If

    If dy > 0 Then
        ' мяч движется вниз
        If y + dy + r > Form1.ScaleHeight Then dy = -dy
    Else
        ' мяч движется вверх
        If y + dy - r < 0 Then dy = -dy
    End If
    x = x + dx
    y = y + dy

    ' нарисовать мяч в новой точке
    Form1.Circle (x, y), r, cBall

End Sub
```

Наблюдать за движением мячика в окне программы довольно быстро надоедает. Возникает желание отбить мячик. Следующая программа (листинг 5.10) показывает, как можно сделать графику интерактивной. В окне программы "Пинг-понг-2" кроме мячика есть ракетка (рис. 5.21), которую с помощью

клавиш перемещения можно двигать влево или вправо. Мячик и ракетку рисует процедура обработки события `Timer`. Факт нажатия клавиши перемещения курсора фиксирует процедура обработки события `KeyDown`, которое возникает при нажатии клавиши. Она записывает в переменную `rd` код направления движения ракетки. Значение переменной `rd` контролирует процедура обработки события `Timer` и, в зависимости от ее значения, сдвигает ракетку влево или вправо. Следует обратить внимание на то, как процедура обработки события `Timer` "сдвигает" ракетку. Она, когда надо сдвинуть ракетку, не рисует ее заново целиком, а сдвигает: стирает небольшой кусочек с одной стороны и дорисовывает с другой. Процедура обработки события `KeyUp` фиксирует факт отпускания клавиши — записывает в переменную `rd` ноль и тем самым информирует процедуру обработки события `Timer`, что ракетку перерисовывать не надо.

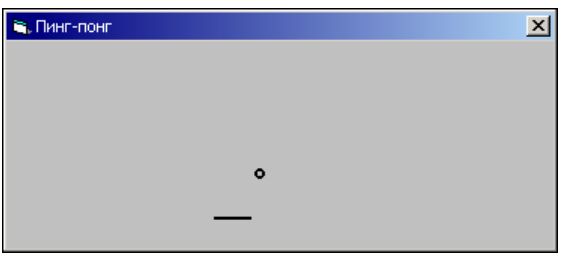


Рис. 5.21. Движением ракетки можно управлять с помощью клавиш перемещения курсора

Листинг 5.10. Пинг-понг-2

```
Option Explicit
Dim x, y As Integer     ' положение мячика
Dim dx, dy As Integer   ' приращение координат
Dim r As Integer        ' радиус мячика
Dim cBall As Long       ' цвет мячика
Dim cBack As Long       ' цвет поля
Dim wp, hp As Integer   ' размер поля (формы)

' это переменные для управления движением ракетки
Dim rd As Integer ' 0 — ракетка не движется;
                  ' 1 — движется влево; 2 — движется вправо
Dim rx1, rx2 As Integer ' координаты X концов ракетки
```

```
Dim ry As Integer        ' координата Y ракетки
Dim rdx As Integer       ' шаг перемещения ракетки

Private Sub Form_Load()
    r = 3
    x = r
    y = 50
    dx = 1
    dy = 1
    cBall = RGB(217, 217, 25)
    cBack = RGB(33, 94, 33)

    Form1.BackColor = cBack

    wp = Form1.ScaleWidth
    hp = Form1.ScaleHeight

    ' ** управление ракеткой **
    rd = 0        ' ракетка на месте
    rx1 = 100
    rx2 = 125
    ry = Form1.ScaleHeight - 20
    rdx = 2 ' шаг движения ракетки

End Sub

Private Sub Form_Paint()
    Form1.Line (rx1, ry)-(rx2, ry), vbRed ' нарисовать ракетку
End Sub

Private Sub Timer1_Timer()

    ' *** мяч ****
    ' стереть изображение мяча
    Form1.Circle (x, y), r, cBack

    ' вычислить новое положение мяча
    If dx > 0 Then
```

```
    ' мяч движется вправо
    If x + dx + r > wp Then dx = -dx
Else
    ' мяч движется влево
    If x + dx - r < 0 Then dx = -dx
End If

If dy > 0 Then
    ' мяч движется вниз
    If (x >= rx1) And (x <= rx2) And (y = ry - r - 1) Then
        ' мячик попал в ракетку
        dy = -dy
    Else
        If y + dy + r > Form1.ScaleHeight Then dy = -dy
    End If
Else
    ' мяч движется вверх
    If (x >= rx1) And (x <= rx2) And _
       (y >= ry - r) And (y <= ry + r) Then
        ' мяч отскочил от нижней стенки и попал в ракетку снизу
        ' чтобы не было дырок в ракетке, перерисуем ее
        Line (rx1, ry)-(rx2, ry), vbRed
    End If
    If y + dy - r < 0 Then dy = -dy
End If
x = x + dx
y = y + dy

' нарисовать мяч в новой точке
Form1.Circle (x, y), r, cBall

'*** ракетка ***
If rd <> 0 Then
    ' игрок нажал и удерживает одну из клавиш
    ' "стрелка вправо" или "стрелка влево"
    ' (см. Form_KeyDown)
    If rd = 1 Then
        ' вправо
```

```
            If rx2 < wp Then
               Line (rx1, ry)-(rx1 + rdx, ry), cBack ' стереть часть слева
               Line (rx2, ry)-(rx2 + rdx, ry), vbRed ' дорисовать справа
               rx1 = rx1 + rdx
               rx2 = rx2 + rdx
            End If
        Else
            ' влево
            If rx1 > 1 Then
              Line (rx2, ry)-(rx2 - rdx, ry), cBack ' стереть часть справа
              Line (rx1 - rdx, ry)-(rx1 + rdx, ry), vbRed ' дорисовать
              rx1 = rx1 - rdx
              rx2 = rx2 - rdx
            End If
        End If
    End If

End Sub

' нажата клавиша
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    If rd <> 0 Then
        ' пользователь удерживает клавишу, ракетка движется
        Exit Sub
    End If

    Select Case KeyCode
        Case 37 ' Шаг (стрелка) право
            rd = 2
        Case 39 ' Шаг (стрелка) влево
            rd = 1
    End Select
End Sub

' отпущена клавиша
Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)
    rd = 0
End Sub
```

В предыдущих примерах изображение объектов формировалось из графических примитивов. Теперь на примере программы "Полет в облаках" (листинг 5.11) рассмотрим, как можно существенно улучшить графику программы за счет использования битовых образов.

Как и в предыдущих программах, эффект перемещения объекта достигается за счет перерисовки изображения объекта с некоторым смещением относительно его прежнего положения. Процесс вывода изображения объекта (битового образа) с использованием маски был подробно описан ранее. Теперь рассмотрим, что нужно сделать, чтобы стереть нарисованный объект.

Удалить объект (восстановить фон) можно путем перерисовки всей фоновой картинки или только той ее части, которая была перекрыта объектом. В рассматриваемой программе используется второй подход. Удаляет изображение объекта метод `PaintPicture`, который копирует фрагмент фона в ту область графической поверхности, в которой находится изображение объекта (рис. 5.22).

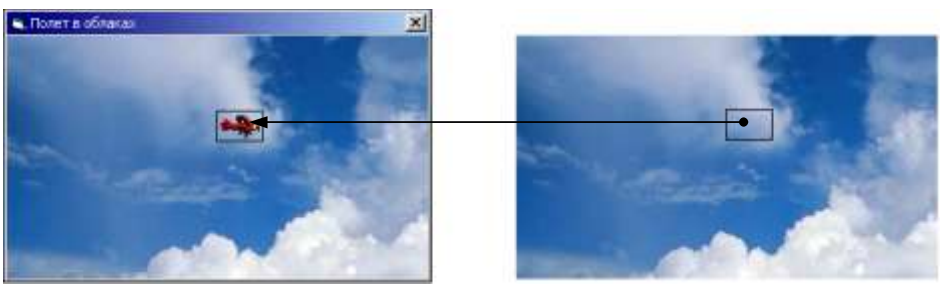


Рис. 5.22. Чтобы стереть изображение объекта, надо вывести фрагмент фона в ту область, где находится объект

Листинг 5.11. Полет в облаках

```
Option Explicit

' битовые образы
Dim back As StdPicture      ' фон
Dim plane As StdPicture     ' самолет
Dim plane_m As StdPicture   ' маска

Dim x, y As Integer    ' координаты объекта (самолета)
```

```
Dim dx, dy As Integer ' приращение координат
Dim w, h As Integer ' ширина и высота битового образа

Private Sub Form_Load()
    Set back = LoadPicture("d:\Pictures\sky.bmp")
    Set plane = LoadPicture("d:\ Pictures \plane.bmp")
    Set plane_m = LoadPicture("d:\ Pictures \plane_m.bmp")

    ' Установить размер формы равным размеру фонового рисунка.
    ' Ширина и высота объекта StdPicture измеряется в единицах,
    ' которые называются HiMETRIC.
    ' Для преобразования величины из HiMETRIC в твипы используются
    ' методы ScaleX и ScaleY.

    Form1.Width = ScaleX(back.Width, vbHimetric, vbTwips)
    Form1.Height = (Form1.Height - Form1.ScaleHeight) + _
                   ScaleY(back.Height, vbHimetric, vbTwips)

    Form1.Picture = back

    w = ScaleX(plane.Width, vbHimetric, vbPixels)
    h = ScaleY(plane.Height, vbHimetric, vbPixels)

    Form1.ScaleMode = vbPixels

    ' начальное положение объекта
    x = 0
    y = 100

    ' скорость движения объекта определяют приращение координаты
    ' и период следования сигналов таймера
    dx = 1
    Timer1.Interval = 10
    Timer1.Enabled = True

End Sub

Private Sub Timer1_Timer()
```

```
    ' стереть объект
    Form1.PaintPicture back, x, y, w, h, x, y, w, h

    ' *** нарисовать объект ***
    ' вывести маску
    Form1.PaintPicture plane_m, x, y, , , , , , , vbSrcAnd
    ' вывести картинку
    Form1.PaintPicture plane, x, y, , , , , , , vbSrcPaint

    If x < Form1.ScaleWidth Then
       x = x + dx
    Else
       x = 0
       dx = 1 + 1 * Rnd

    End If

End Sub
```

Процедура обработки события `Load` загружает необходимые для работы программы битовые образы (изображение объекта, соответствующую изображению объекта маску объекта и фоновый рисунок), устанавливает размер окна в соответствии с размером фонового рисунка, задает начальное положение объекта, выполняет настройку и запуск таймера.

Основную работу по формированию кадров мультипликации выполняет процедура обработки события `Timer`. Она сначала путем копирования фрагмента фонового рисунка на поверхность формы стирает изображение объекта (восстанавливает "испорченный" фон), затем рисует объект на новом месте.

Контрольные вопросы

1. Какие существуют способы формирования анимации?
2. Перечислите шаги, обеспечивающие реализацию эффекта движения графического объекта?
3. Как можно "стереть" графический объект, например линию?
4. Как можно "стереть" изображение графического объекта, выведенного поверх фонового рисунка?
5. Какой метод обеспечивает вывод на графическую поверхность фрагмента битового образа?

Загрузка битового образа из ресурса

Загрузить необходимую программе картинку можно не только из файла, но и из ресурса. Ресурс — это данные, которые находятся в выполняемом файле. Различают следующие виды ресурсов: битовый образ, строка символов и курсор. Очевидным преимуществом использования ресурсов является сокращение количества файлов, образующих приложение (все необходимые программе картинки находятся в выполняемом файле), повышение защищенности программы (пользователь не сможет случайно удалить файлы, нужные программе).

Чтобы программа могла использовать возможность загрузки данных из ресурса, необходимо создать файл ресурсов, поместить в него ресурсы, добавить созданный файл ресурсов в проект, изменить программу так, чтобы она загружала данные из ресурса, и после этого выполнить компиляцию.

Создание файла ресурсов

Создать файл ресурсов можно с помощью надстройки (утилиты) VB Resource Editor, которая входит в Visual Basic. Чтобы запустить VB Resource Editor, надо в меню **Tools** выбрать команду **Resource Editor**.

Окно VB Resource Editor сразу после запуска надстройки приведено на рис. 5.23.

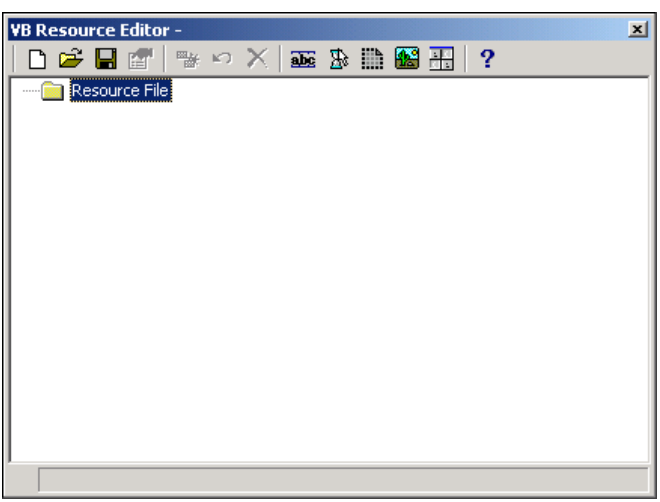


Рис. 5.23. Окно редактора ресурсов VB Resource Editor

Замечание

Если команды **Resource Editor** в меню **Tools** нет, то надо в меню **Add-Ins** выбрать команду **Add-In Manager**. Затем в списке **Available Add-Ins** появившегося окна **Add-In Manager** надо выбрать **VB 6 Resource Editor**, в группе **Load Behavior** установить переключатели **Loaded/Unloaded** и **Load on Startup** и сделать щелчок на кнопке **OK**. После этого надо перезагрузить Visual Basic.

Чтобы добавить в файл ресурсов битовый образ, надо щелкнуть на кнопке **Add Bitmap** (рис. 5.24) и в появившемся окне выбрать BMP-файл, в котором находится картинка, которую надо поместить в ресурс.

Рис. 5.24. Кнопка **Add Bitmap**

В результате в файл ресурсов будет добавлен ресурс — битовый образ, и в окне редактора файла ресурсов появится новый элемент списка **Bitmap**. Число, которое отображается справа от значка битового образа, — это автоматически сформированный идентификатор ресурса. Именно этот идентификатор надо будет использовать в программе, в инструкции загрузки битового образа из ресурса. Идентификатор ресурса можно изменить. Для этого надо сделать щелчок правой кнопкой мыши на значке ресурса, из контекстного меню выбрать команду **Properties** и в поле **ID** окна редактора свойств ввести новый идентификатор ресурса.

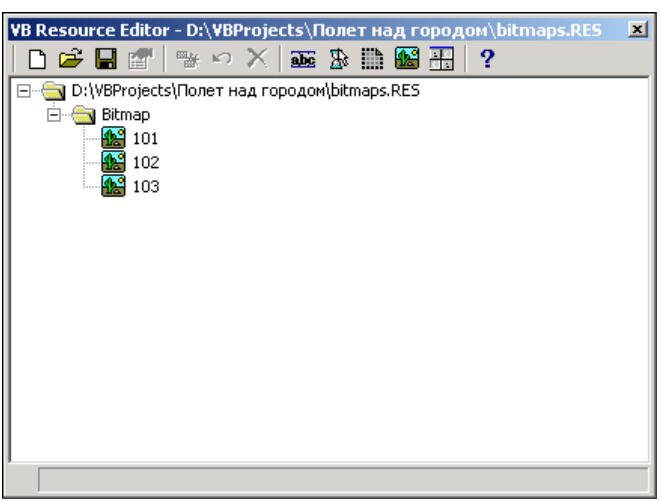


Рис. 5.25. Пример файла ресурсов

Следует обратить внимание, что при записи идентификатора ресурса можно использовать только прописные (заглавные) буквы латинского алфавита и цифры. После того как все необходимые программе иллюстрации будут добавлены, файл ресурсов надо сохранить в каталоге проекта, для которого он предназначен.

На рис. 5.25 приведено окно редактора ресурсов, в котором отображается содержимое файла ресурсов для программы "Полет над городом". Файл называется bitmaps.RES и содержит три битовых образа: самолет (`101`), маску (`102`) и фон (`103`).

Доступ к файлу ресурсов

Для того чтобы в процессе компиляции ресурсы, находящиеся в файле ресурсов, были помещены в выполняемый файл, надо в контекстном меню окна **Project Explorer** выбрать команду **Add ▸ Resource File** и затем указать файл ресурсов. В результате описанных действий в проект будет добавлена ссылка на файл ресурсов (рис. 5.26).

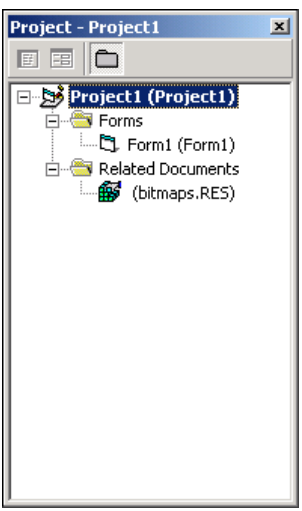


Рис. 5.26. Результат добавления ссылки на файл ресурсов

Загрузка ресурса

Загрузку битового образа из ресурса можно выполнить с помощью функции `LoadResPicture`. В качестве параметров функции надо указать идентификатор ресурса и константу `vbResBitmap`.

Например, инструкция

```
plane = LoadResPicture(101, vbResBitmap)
```

загружает в битовый образ `plane` картинку из ресурса `101`.

В качестве примера в листинге 5.12 приведена программа "Полет над городом", в которой битовые образы загружаются из ресурсов.

Листинг 5.12. Полет над городом (загрузка битовых образов из ресурсов)

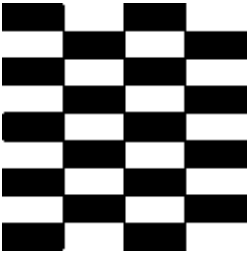
```
Option Explicit

' битовые образы
Dim back As StdPicture      ' фон
Dim plane As StdPicture     ' самолет
Dim plane_m As StdPicture   ' маска

Dim x, y As Integer    ' координаты объекта (самолета)
Dim dx, dy As Integer  ' приращение координат
Dim w, h As Integer ' ширина и высота битового образа

Private Sub Form_Load()

    ' загрузка битовых образов из ресурса
    Set plane = LoadResPicture(101, vbResBitmap)    ' самолет
    Set plane_m = LoadResPicture(102, vbResBitmap) ' маска
    Set back = LoadResPicture(103, vbResBitmap)     ' фон

    ' Установить размер формы в соответствии с размером фонового рисунка.
    ' Размер объекта StdPicture измеряется в единицах, которые
    ' называются HiMETRIC. Для преобразования величины из HiMETRIC
    ' в твипы используются методы ScaleX и ScaleY.

    Form1.Width = ScaleX(back.Width, vbHimetric, vbTwips)
    Form1.Height = (Form1.Height - Form1.ScaleHeight) + _
                   ScaleY(back.Height, vbHimetric, vbTwips)

    Form1.Picture = back

    w = ScaleX(plane.Width, vbHimetric, vbPixels)
```

```
    h = ScaleY(plane.Height, vbHimetric, vbPixels)

    Form1.ScaleMode = vbPixels

    ' начальное положение объекта
    x = 0
    y = 60

    ' скорость движения объекта определяют приращение координаты
    ' и период следования сигналов таймера
    dx = 1
    Timer1.Interval = 10
    Timer1.Enabled = True

End Sub

Private Sub Timer1_Timer()
     ' стереть объект
     Form1.PaintPicture back, x, y, w, h, x, y, w, h

     ' нарисовать объект на новом месте
     ' вывести маску
     Form1.PaintPicture plane_m, x, y, , , , , , , vbSrcAnd
     ' вывести картинку
     Form1.PaintPicture plane, x, y, , , , , , , vbSrcPaint

     If x < Form1.ScaleWidth Then
        x = x + dx
     Else
        x = 0
        dx = 1 + 1 * Rnd

       ' Timer1.Enabled = False
     End If

End Sub
```

Еще раз следует обратить внимание, что при загрузке битовых образов из ресурса, файл ресурсов и файлы иллюстраций, которые использовались в процессе создания файла ресурсов, во время работы программы не нужны. Преимущества загрузки битовых образов из ресурсов очевидны — все необходимые программе картинки находятся в выполняемом файле.

Контрольные вопросы

1. Что такое ресурс?
2. Перечислите основные типы ресурсов.
3. В чем преимущество хранения битовых образов в ресурсе?
4. При помощи какого инструмента (утилиты) можно создать файл ресурсов?
5. Какая функция обеспечивает загрузку битового образа из ресурса?



Глава 6

Мультимедиа

Большинство приложений, работающих в Windows, являются мультимедийными. Такие программы обеспечивают просмотр видеороликов и мультипликации, воспроизведение музыки, речи, звуковых эффектов. Типичными примерами мультимедийных программ являются игры и обучающие программы.

Visual Basic предоставляет в распоряжение программиста компонент `MMControl` (Microsoft Multimedia Control), который обеспечивает воспроизведение звука и видео. Для воспроизведения звука можно также использовать API-функцию `PlaySound`.

Функция *PlaySound*

API-функция `PlaySound` весьма удобна для реализации звуковых эффектов, например фонового музыкального сопровождения в игровой программе или звукового сигнала, сопровождающего появление сообщения. Функция позволяет воспроизвести звук, который находится в WAV-файле. Также функцию `PlaySound` можно использовать для воспроизведения стандартных звуковых сигналов Windows.

Для того чтобы функцию `PlaySound` можно было использовать, в текст программы надо поместить ее объявление:

```
Private Declare Function PlaySound Lib "winmm.dll" Alias "PlaySoundA" _
                  (ByVal lpszName As String, ByVal hModule As Long, _
                   ByVal dwFlags As Long) As Long
```

Инструкция вызова функции `PlaySound` в общем виде выглядит так:

```
r = PlaySound(Файл, 0, Режим)
```

Параметр *Файл* задает имя файла, который надо воспроизвести, параметр *Режим* — режим воспроизведения. Различают два режима воспроизведения

звука: синхронный и асинхронный. При воспроизведении звука в синхронном режиме функция `PlaySound` возвращает управление вызвавшей ее программе только после того, как процесс воспроизведения будет завершен. При воспроизведении звука в асинхронном режиме — сразу, как только процесс воспроизведения будет активизирован. Чтобы задать режим воспроизведения, надо указать в качестве параметра *Режим* константу `&H1` (асинхронный режим) или `&H0` (синхронный режим). Например, инструкция

```
PlaySound ("tada.wav", 0, &H1)
```

обеспечивает воспроизведение звукового файла в асинхронном режиме.

Следует обратить внимание на то, что если имя звукового файла указано не полностью (не задан путь к файлу), то функция `PlaySound` выполнит поиск файла: сначала в текущем каталоге, затем в каталогах Windows. Если указанный звуковой файл все-таки не будет найден, то будет воспроизведен "Стандартный звук" (задается в настройках Windows). Программист может заблокировать воспроизведение "Стандартного звука". Для этого значение параметра *Режим* надо изменить на `&H2`.

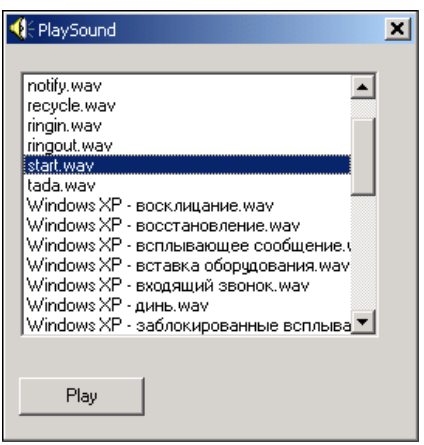


Рис. 6.1. Воспроизведение звука обеспечивает API-функция `PlaySound`

Использование функции `PlaySound` демонстрирует программа PlaySound. Появление ее окна (рис. 6.1) сопровождается звуком notify.wav. Затем пользователь может выбрать звуковой файл и прослушать его. Завершение работы программы сопровождается звуком recycle.wav. Для выбора звукового файла используется компонент `FileListBox`. Список файлов формируется в результате присваивания значений свойствам `Pattern` и `Path`. В свойства `Pattern` записывается маска звуковых файлов — строка `*.wav`, в свойство `Path` — путь

к каталогу Media, в котором находятся звуковые файлы Windows. Настройку компонента File1, а также инициализацию воспроизведения звукового сигнала, сопровождающего появление окна программы, выполняет процедура обработки события Initialize. Следует обратить внимание, что если по какой-либо причине файла notify.wav в каталоге Media не окажется, то появление окна не будет сопровождаться звуком, указанным в настройках операционной системы, т. к. в инструкции вызова функции PlaySound указан параметр SND_NODEFAULT. Текст программы приведен в листинге 6.1.

Листинг 6.1. Использование API-функции PlaySound

```
Option Explicit

' чтобы получить доступ к API-функции, функцию надо объявить
Private Declare Function PlaySound Lib "winmm.dll" Alias "PlaySoundA" _
                (ByVal lpszName As String, ByVal hModule As Long, _
                 ByVal dwFlags As Long) As Long

' API-константы
Const SND_ASYNC = &H1      ' асинхронный режим воспроизведения звука
Const SND_SYNC = &H0       ' синхронный режим воспроизведения звука
Const SND_NODEFAULT = &H2  ' не воспроизводить "Стандартный звук"


Private Sub Form_Initialize()
    ' функция Environ с параметром "windir" возвращает
    ' имя каталога Windows
    File1.Pattern = "*.wav"
    File1.Path = Environ("windir") + "\Media\"
    PlaySound File1.Path + "\notify.wav", 0, SND_ASYNC + SND_NODEFAULT
End Sub

' щелчок на кнопке Play
Private Sub Command1_Click()
    Dim aWAVFile As String ' файл, который надо воспроизвести

    aWAVFile = File1.Path + "\" + File1.FileName
    PlaySound aWAVFile, 0, SND_ASYNC
End Sub
```

```
' завершение работы программы
' сопровождается звуком
Private Sub Form_Unload(Cancel As Integer)
    PlaySound Environ("windir") + "\Media\" + "recycle.wav", 0, _
                    SND_ASYNC + SND_NODEFAULT
End Sub
```

Контрольные вопросы

1. Какого формата звуковые файлы может проигрывать функция (процедура) `PlaySound`?
2. Что надо сделать, чтобы функция `PlaySound` стала доступной?
3. В чем различие синхронного и асинхронного режимов воспроизведения звука?

Компонент *MMControl*

Компонент `MMControl` обеспечивает воспроизведение звуковых файлов (WAV, MP3, MID) и видеофайлов (AVI), а также CD.

Чтобы компонент (рис. 6.2) стал доступен, его надо подключить: в меню **Project** выбрать команду **Components**, на вкладке **Controls** окна **Components** установить флажок **Microsoft Multimedia Control 6.0** (рис. 6.3) и щелкнуть на кнопке **OK**.

Рис. 6.2. Значок компонента `MMControl`

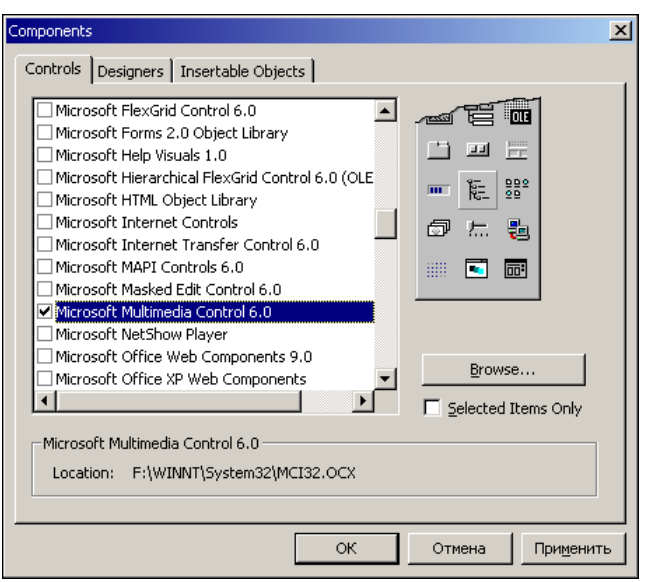


Рис. 6.3. Подключение компонента `MMControl`

Компонент MMControl представляет собой группу кнопок (рис. 6.4), подобных тем, которые можно видеть на обычном аудио- или видеоплеере. Назначение кнопок поясняет табл. 6.1. Свойства компонента MMControl, доступные во время разработки формы, приведены в табл. 6.2. Помимо свойств, перечисленных в таблице, для каждой из кнопок определены свойства Enable и Visible. Свойство Enable позволяет управлять доступом к кнопке, а свойство Visible — скрыть кнопку или сделать ее видимой.

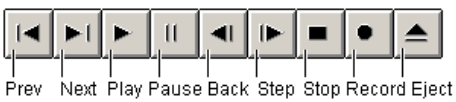


Рис. 6.4. Компонент MMControl

Таблица 6.1. *Кнопки компонента* MMControl

Кнопка	Обозначение	Действие
Воспроизведение	Play	Воспроизведение звука или видео
Пауза	Pause	Приостановка воспроизведения
Стоп	Stop	Остановка воспроизведения
Следующий	Next	Переход к следующему кадру
Предыдущий	Prev	Переход к предыдущему кадру
Шаг	Step	Переход к следующему звуковому фрагменту, например к следующему треку CD
Назад	Back	Переход к предыдущему звуковому фрагменту, например к предыдущему треку CD
Запись	Record	Активизирует процесс записи
Открыть	Eject	Открывает CD-дисковод

Таблица 6.2. *Свойства компонента* MMControl

Свойство	Описание
DeviceType	Тип устройства. Определяет конкретное устройство, которое представляет собой компонент MMControl. Тип устройства задается строковой константой: WaveAudio — проигрыватель звука; AVIVideo — видеопроигрыватель; CDAudio — CD-проигрыватель

Таблица 6.2 *(окончание)*

Свойство	Описание
FileName	Имя файла, в котором находится воспроизводимый звуковой фрагмент или видеоролик
UpdateInterval	Интервал генерации события StatusUpdate, которое обычно используется для организации обновления индикатора состояния устройства воспроизведения. Например, при воспроизведении CD — номера воспроизводимого трека (композиции) и времени воспроизведения

Следует обратить внимание, что некоторые свойства (табл. 6.3) компонента MMControl доступны только во время работы программы и поэтому в окне **Properties** не отображаются.

Таблица 6.3. *Свойства компонента MMControl, доступные во время работы программы*

Свойство	Описание
Length	Время, необходимое для воспроизведения открытого файла (например, WAV или AVI) или всех треков CD
Tracks	Количество треков на открытом устройстве (количество треков на CD)
Track	Номер воспроизводимого трека
Position	Время, прошедшее с момента начала воспроизведения файла или CD
TrackPosition	Положение трека на CD (отсчитывается от начала CD и измеряется в единицах, заданных свойством TimeFormat)
TrackLength	Время, необходимое для воспроизведения трека CD
TimeFormat	Формат представления значений свойств Length, Position, TrackLength и TrackPosition. Наиболее универсальным является формат mciFormatMilliseconds
Mode	Состояние устройства воспроизведения. Устройство может быть в состоянии воспроизведения (msiModePlay). Процесс воспроизведения может быть остановлен (msiModeStop) или приостановлен (msiModePause)
hWndDisplay	Идентификатор объекта (компонента), используемого в качестве экрана, на поверхности которого осуществляется отображение видеоклипа. Если значение свойства не задано, то отображение осуществляется в отдельном, создаваемом во время работы программы окне

Управлять процессом работы устройства воспроизведения (компонентом MMControl) может пользователь (с помощью командных кнопок) или программа, путем записи соответствующих команд (табл. 6.4) в свойство Command. Следует обратить внимание, что в конце работы программы, которая использует компонент MMControl, устройство воспроизведения обязательно надо отключить (закрыть) с помощью команды Close.

Таблица 6.4. *Команды управления компонентом* MMControl

Команда	Действие
Open	Инициализирует устройство мультимедиа
Play	Активизирует процесс воспроизведения. Действие метода аналогично щелчку на кнопке **Play**
Stop	Останавливает процесс воспроизведения
Pause	Приостанавливает процесс воспроизведения
Next	Переход к следующему треку, например к следующей композиции на CD
Prev	Переход к предыдущему треку, например к предыдущей композиции на CD
Step	Переход к следующему кадру
Back	Переход к предыдущему кадру

Контрольные вопросы

1. Что надо сделать, чтобы компонент MMControl стал доступен?
2. Какого формата звуковые файлы может воспроизводить компонент MMControl?

MP3-плеер

В качестве примера применения компонента MMControl для воспроизведения звука рассмотрим программу, используя которую можно прослушать звуковые (MP3 и WAV) файлы. Форма и окно программы "MP3-плеер" представлены на рис. 6.5. Компонент File1 (FileList) используется для выбора файла. Во время работы программы в поле компонента Label1 отображается имя воспроизводимого файла, в поле Label2 — время звучания композиции, а в поле Label3 — время от начала воспроизведения композиции. Кнопка Command1 активизирует отображение стандартного диалога **Обзор папок**, в котором пользователь может выбрать каталог, содержащий звуковые файлы. Текст программы приведен в листинге 6.2.

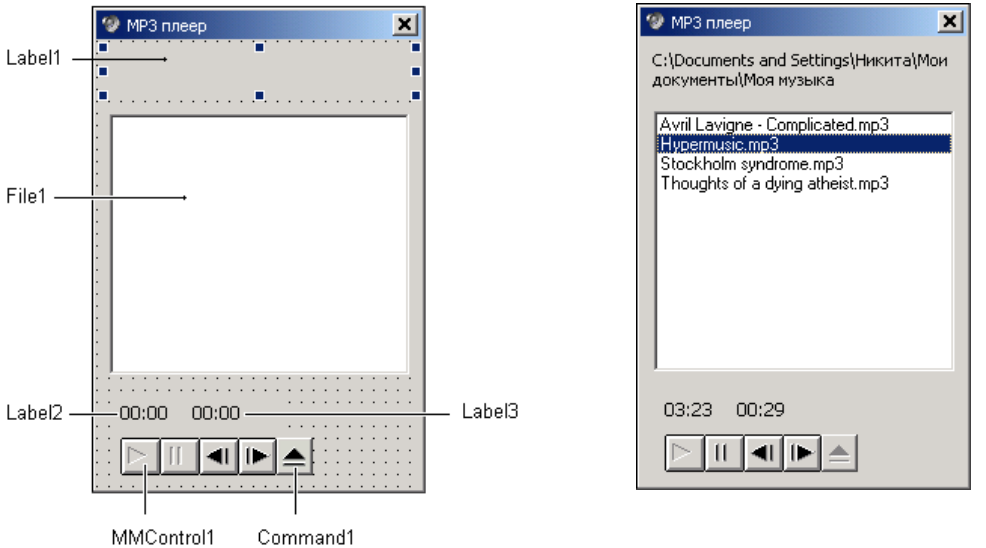


Рис. 6.5. Форма и окно программы "MP3-плеер"

Листинг 6.2. MP3-плеер

```
Option Explicit

' эти API-функции используются для доступа к окну Обзор папок
Private Declare Function SHBrowseForFolder Lib "shell32" Alias _
              "SHBrowseForFolderA" (ByRef b As Any) As Long
Private Declare Function SHGetPathFromIDList Lib "shell32" _
               (ByVal ResPIDL As Any, ByVal patch As String) As Long

' константы SHELL API
Const CSIDL_DRIVES As Long = 17
Const BIF_RETURNONLYFSDIRS As Long = 1
Const MAX_PATH = 260

Const mciTormatMilleseconds = 0

' эта структура используется для передачи информации
' в функцию SHBrowseFolder, которая выводит диалоговое
' окно Обзор папок
```

```
Private Type bi ' browseinfo
    hwndOwner As Long
    pidlRoot As Long
    pszDisplayName As String ' выбранная папка (без пути)
    lpzTitle As String       ' подсказка
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type

Dim status As Integer ' состояние плеера:
                      '    0 — Стоп (Пауза)
                      '    1 — Воспроизведение

' инициализация формы
Private Sub Form_Initialize()
   Dim aPath As String

   On Error Resume Next

   aPath = Environ("homepatch") + File1.Path + _
                  "\Мои документы\Моя музыка"
   File1.Path = aPath
   Label1.Caption = aPath

   ' в поле компонента File1 отображать файлы
   ' с расширением mp3
   File1.Pattern = "*.mp3"

   ' в MMControl1 отображать только кнопки Play, Pause, Step и Back
   ' остальные кнопки скрыть
   MMControl1.PrevVisible = False
   MMControl1.NextVisible = False
   MMControl1.RecordVisible = False
   MMControl1.EjectVisible = False

   Form1.ScaleMode = vbPixels
```

```
   File1.ListIndex = 0

   status = 0 ' плеер в режиме "Стоп"
End Sub

' щелчок на имени файла или программное изменение
' значения свойства ListIndex
Private Sub File1_Click()

   If MMControl1.Command <> "CLOSE" Then MMControl1.Command = "CLOSE"

   If File1.FileName <> "" Then
      MMControl1.FileName = File1.Path + "\" + File1.FileName

      MMControl1.TimeFormat = mciTormatMilleseconds
      MMControl1.Command = "OPEN"

      Label2.Caption = toHMS(MMControl1.Length)
      Label3.Caption = "00:00"
      If status = 1 Then
         ' пользователь сделал щелчок на кнопке Step или Back,
         ' когда плеер находился в режиме воспроизведения,
         ' поэтому после загрузки файла сразу активизируем
         ' воспроизведение
        MMControl1.Command = "Play"
      End If
   End If
End Sub

' щелчок на кнопке Пауза
Private Sub MMControl1_PauseClick(Cancel As Integer)
    ' плеер сам выполнит нужное действие, мы же
    ' только зафиксируем его статус
    If status = 0 Then
        status = 1
        Command1.Enabled = False
    Else
        status = 0
```

```
        Command1.Enabled = True
    End If
End Sub

' щелчок на кнопке Play
Private Sub MMControl1_PlayClick(Cancel As Integer)
    status = 1
    Command1.Enabled = False
End Sub

' щелчок на кнопке Стоп
Private Sub MMControl1_StopClick(Cancel As Integer)
    status = 0
    Command1.Enabled = True
End Sub

' щелчок на кнопке Step — перейти к следующему
' файлу списка
Private Sub MMControl1_StepClick(Cancel As Integer)
    If File1.ListIndex < File1.ListCount - 1 Then
        File1.ListIndex = File1.ListIndex + 1
        If File1.ListIndex = File1.ListCount Then
            MMControl1.StepEnabled = False
        End If
    Else
        File1.ListIndex = 0
    End If
End Sub

' щелчок на кнопке Back — перейти к предыдущему файлу списка
Private Sub MMControl1_BackClick(Cancel As Integer)
    If File1.ListIndex > 0 Then
        File1.ListIndex = File1.ListIndex - 1
    Else
        File1.ListIndex = File1.ListCount - 1
    End If
End Sub
```

```
' обработка сигнала StatusUpdate, генерируемого MMControl
Private Sub MMControl1_StatusUpdate()
    If MMControl1.Position < MMControl1.Length Then
        ' продолжается воспроизведение текущей композиции
        Label3.Caption = toHMS(MMControl1.Position)
    Else
        ' воспроизведение текущей композиции закончено
        If File1.ListIndex < File1.ListCount - 1 Then
            File1.ListIndex = File1.ListIndex + 1
        Else
            MMControl1.Command = "Stop"
            status = 0
            Command1.Enabled = True
        End If
     End If

End Sub

' щелчок на кнопке Eject (Выбор каталога)
Private Sub Command1_Click()
    Dim ResPIDL As Long
    Dim t As bi
    Dim Path As String

    Dim r As Long
    Dim p As String

    t.hwndOwner = Form1.hWnd
    t.lpzTitle = "Выберите папку, в которой находятся MP3-файлы"
    t.ulFlags = BIF_RETURNONLYFSDIRS ' кнопка OK будет доступна,
                                     ' если пользователь выберет папку
    t.pszDisplayName = String(MAX_PATH, 0)

    ResPIDL = SHBrowseForFolder(t) ' вывести окно Обзор папок
    If ResPIDL <> 0 Then
        ' пользователь выбрал папку
        ' получить ее полное имя
```

```
        Path = String(MAX_PATH, Chr(0))
        r = SHGetPathFromIDList(ResPIDL, Path)
        ' строка Path — это Си-строка, содержащая
        ' NULL-символ (код 0). До этого символа находится
        ' полезная информация, после — мусор

        p = InStr(1, Path, Chr(0)) ' положение NULL-символа
        Path = Mid(Path, 1, p - 1)
        If Mid(Path, p - 1, 1) <> "\" Then Path = Path + "\"

        File1.Path = Path   ' отобразить содержимое выбранного каталога
                            ' в поле компонента File1 (FileList)
        File1.ListIndex = 0 ' в результате возникает событие File1.Click
    End If

End Sub

' функция преобразует целое число (время звучания или текущую позицию
' воспроизведения) в формат "чч:мм:сс"
Private Function toHMS(time As Long) As String
    Dim H As Integer    ' часы
    Dim M As Integer    ' минуты
    Dim S As Integer    ' секунды

    H = Int((time / 1000) / 3600)
    M = Int((time / 1000 Mod 3600) / 60)
    S = (time / 1000 Mod 3600) Mod 60

    If H > 0 Then toHMS = Str(H) + ":"
    toHMS = toHMS + Format(M, "0#") + ":" + Format(S, "0#")
End Function

' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
     MMControl1.Command = "STOP"    ' остановить воспроизведение
     MMControl1.Command = "CLOSE"   ' "выключить" плеер
End Sub
```

Работает программа следующим образом. Сразу после запуска функция обработки события `Initialaze` присваивает значения свойствам `Path` и `Pattern` компонента `File1` (`FileList`), в результате в поле компонента отображается список звуковых файлов, находящихся в каталоге `Мои документы\Моя музыка`. Для обработки ошибки, которая возникает, если указанного каталога на диске нет, в программу вставлена инструкция `On Error Resume Next` (При возникновении ошибки выполнить следующую инструкцию). Чтобы активизировать процесс воспроизведения, пользователь должен сделать щелчок на кнопке **Play**. Процедура обработки события `Click` в поле компонента `File1` останавливает воспроизведение текущего файла, открывает выбранный файл и отображает его длину (время, необходимое для воспроизведения) в поле `Label2`. Для преобразования времени воспроизведения (свойство `Length` компонента `MMControl1`), которое измеряется в миллисекундах, в формат `мм:сс` используется специально созданная функция `toHMS`.

Во время воспроизведения компонент `MMControl1` генерирует событие `StatusUpdate` (период возникновения события задает значение свойства `UpdateInterval`). Процедура обработки этого события выводит в поле `Label3` значение свойства `Position` — время, прошедшее от начала воспроизведения файла.

Контрольные вопросы

1. Какое свойство компонента `MMControl` позволяет управлять работой мультимедиаплеера программно?
2. Какое значение надо присвоить свойству `Command` мультимедиаплеера, чтобы активизировать процесс воспроизведения?
3. Какое значение надо присвоить свойству `Command` мультимедиаплеера, чтобы остановить процесс воспроизведения?
4. Какая функция (и с каким параметром) позволяет получить имя папки пользователя?

MIDI

В компьютерных играх, в обучающих и других программах, а также в качестве звуковых сигналов мобильных телефонов широко используется электронная, или MIDI, музыка. Название MIDI электронная музыка получила от названия интерфейса (Musical Instrument Digital Interface) — способа подключения электронных музыкальных инструментов к компьютеру. Находится электронная музыка в MIDI-файлах.

Процесс воспроизведения MIDI-музыки с помощью компонента `MMControl` демонстрирует программа "Угадай число" (ее форма и окно приведены

на рис. 6.6 и 6.7). Музыка (мелодия, находящаяся в MIDI-файле) начинает звучать сразу после запуска программы.

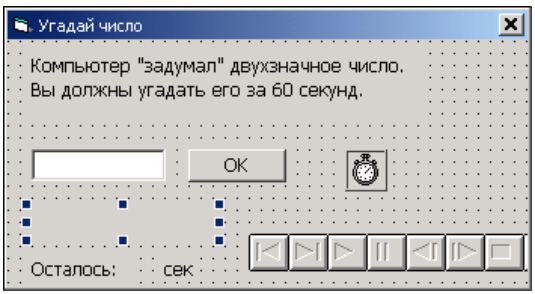


Рис. 6.6. Форма программы "Угадай число"

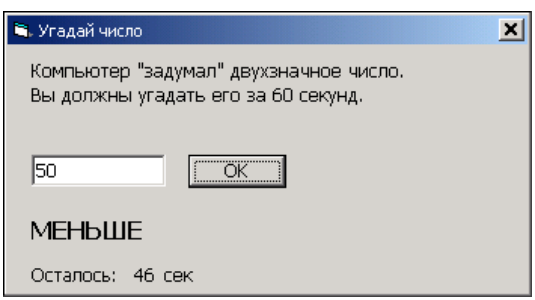


Рис. 6.7. Во время работы программы "Угадай число" звучит MIDI-музыка

Как можно заметить, кнопки управления компонента `MMControl` в окне программы не отображаются (значению свойства `Visible` компонента `MMControl` присвоено значение `False`). Управление воспроизведением мелодии (работой медиаплеера) осуществляется программно — путем записи соответствующих команд в свойство `Command`.

Текст программы "Угадай число" приведен в листинге 6.3.

Листинг 6.3. Угадай число

```
Option Explicit

Dim comp As Integer   ' число, задуманное компьютером
Dim sek As Integer    ' счетчик времени (обратный отсчет)
```

```
' настройка формы и компонентов
Private Sub Form_Load()
    MMControl1.Visible = False
    MMControl1.FileName = Dir("*.mid")
    If MMControl1.FileName = "" Then
        ' в текущем каталоге нет ни одного MIDI-файла
        ' попробуем загрузить из Windows\Media
        MMControl1.FileName = Environ("windir") + "\Media\" + _
                              Dir(Environ("windir") + "\Media\*.mid")
    End If

    ' музыкальное сопровождение - MIDI-мелодия
    MMControl1.Command = "Open"
    MMControl1.Command = "Play"

    Randomize (Time()) ' инициализация генератора случайных чисел
    comp = Int(99 * Rnd() + 1) ' случайное число от 1 до 99

    Command1.Enabled = False

    Text1.MaxLength = 2
    Text1.Text = ""

    sek = 60 ' время на решение задачи — 60 сек
End Sub

Private Sub Form_Activate()
    Text1.SetFocus
End Sub

' сигнал от MMControl
Private Sub MMControl1_StatusUpdate()
    If MMControl1.Position = MMControl1.Length Then
        ' воспроизведение мелодии завершено — повторить
        MMControl1.Command = "Prev" ' переход к началу мелодии
        MMControl1.Command = "Play" ' воспроизведение
    End If
End Sub
```

```
' нажатие клавиши в поле ввода числа
Private Sub Text1_KeyPress(KeyAscii As Integer)
    Select Case KeyAscii
        Case 48 To 57    ' цифры
        Case 8           ' <Backspace> — "забой"
        Case 13          ' <Enter>
            ' проверить, правильное ли число
            Call IsItOk
        Case Else
            ' остальные символы запрещены
            KeyAscii = 0
     End Select
End Sub

' содержимое поля ввода изменилось
Private Sub Text1_Change()
    ' кнопка OK доступна только в том случае,
    ' если в поле редактирования 2 символа
    Label6.Caption = ""
    If Len(Text1.Text) = 0 Then
        Command1.Enabled = False
    Else
        Command1.Enabled = True
    End If
End Sub

' щелчок на кнопке OK
Private Sub Command1_Click()
    Call IsItOk
End Sub

' проверяет, угадал ли игрок число
Sub IsItOk()
    Dim igrok As Integer ' вариант игрока

    igrok = CInt(Text1.Text)

    If igrok = comp Then
        Timer1.Enabled = False ' остановить таймер
```

```
        Command1.Enabled = False
        Text1.Enabled = False
        Label6.Caption = "ПРАВИЛЬНО!"
        MsgBox "Поздравляю! Вы справились с поставленной задачей за " + _
               Str(60 - sek) + "сек.", vbOKOnly, "Угадай число"
    Else
        If igrok < comp Then
            Label6.Caption = "БОЛЬШЕ"
        Else
            Label6.Caption = "МЕНЬШЕ"
        End If
    End If

End Sub

' сигнал таймера
Private Sub Timer1_Timer()
    sek = sek - 1
    Label1.Caption = Str(sek)
    If sek = 0 Then
        ' время, отведенное на решение задачи, истекло
        Timer1.Enabled = False
        Command1.Enabled = False
        Text1.Enabled = False
        MsgBox " Вы не справились с поставленной задачей." + _
               vbCr + "Задуманное число: " + Str(comp), _
               vbOKOnly, "Угадай число"
    End If
End Sub

' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"   ' остановить воспроизведение
    MMControl1.Command = "Close" ' "выключить" плеер
End Sub
```

Работает программа следующим образом. Процедура обработки события `Load` записывает в свойство `FileName` компонента `MMControl` имя одного из

MIDI-файлов, который находится в текущем каталоге. Имя файла возвращает функция `Dir`. Если в текущем каталоге нет ни одного файла с расширением MID, то делается попытка найти MIDI-файл в каталоге Media, который находится в каталоге Windows (здесь Windows — название операционной системы). Так как каталог Windows на разных компьютерах может называться поразному (например, Windows или Winnt), то вместо конкретного имени используется значение функции `Environ` с параметром `windir`. Значением функции `Environ` (от англ. *environment* — окружение) с указанным параметром является значение переменной окружения `windir`, которая содержит имя каталога Windows. После этого медиаплееру направляются команды `Open` и `Play`. В результате начинает звучать музыка, загруженная из MIDI-файла. В процессе воспроизведения медиаплеер генерирует событие `StatusUpdate`. Процедура обработки этого события сравнивает положение указателя текущей позиции воспроизводимого файла со значением свойства `Length` (временем, необходимым для воспроизведения мелодии) и, если значения равны, отправляет медиаплееру команды `Prev` (перевести указатель воспроизведения в начало) и `Play`. В результате мелодия начинает звучать снова. Таким образом, мелодия звучит до тех пор, пока окно программы не будет закрыто. Остальные процедуры реализуют алгоритм игры. Процедура обработки события `KeyPress` компонента `Text` блокирует ввод в поле редактирования запрещенных символов. Процедура обработки события `Change` этого же компонента делает недоступной кнопку **OK**, если в поле редактирования нет ни одного символа. Процедура обработки события таймера обеспечивает обратный отсчет и индикацию времени. Следует обратить внимание, что процедура обработки события `Unload` формы останавливает процесс воспроизведения и закрывает плеер.

Контрольные вопросы

1. Возникновение какого события должна контролировать программа, чтобы обнаружить факт окончания воспроизведения звукового файла?
2. Как определить, что воспроизведение звукового файла завершено?
3. Что надо сделать, чтобы повторно активизировать воспроизведение звукового файла?

CD-плеер

Следующий пример показывает, как на основе компонента `MMControl` можно создать вполне функциональный проигрыватель компакт-дисков. Вид формы программы CD Player приведен на рис. 6.8, значения свойств компонета `MMControl1` — в табл. 6.5. Компонент `Timer1` используется для управления работой "индикатора", обеспечивает мигание сообщения о необходимости вставить в дисковод музыкальный CD.

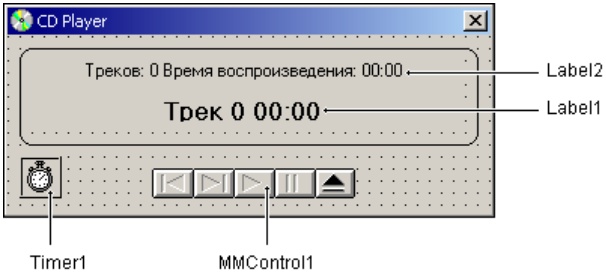


Рис. 6.8. Форма программы CD Player

Таблица 6.5. *Значения свойств компонента* `MMControl1`

Свойство	Значение
`DeviceType`	`CDAudio`
`BackVisible`	`False`
`StepVisible`	`False`
`StopVisible`	`False`
`RecordVisible`	`False`
`BorderStyle`	`msiNone`
`UpdateInterval`	`1000`

Вид окна программы сразу после ее запуска приведен на рис. 6.9. В случае, если диска в дисководе нет или диск не музыкальный, на индикаторе выводится соответствующее сообщение. Щелчок на кнопке **Play** активизирует процесс воспроизведения. Во время воспроизведения на индикаторе отображается номер воспроизводимого трека и время, прошедшее от начала его воспроизведения (рис. 6.10).

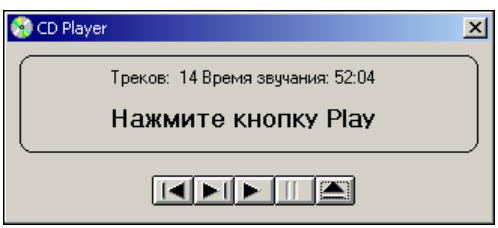


Рис. 6.9. В начале работы программы на индикаторе выводится информация о количестве треков и времени воспроизведения CD

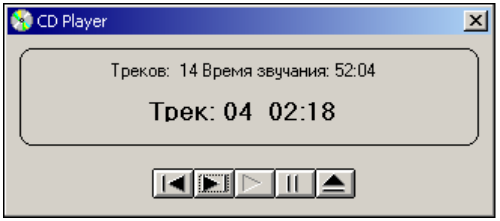


Рис. 6.10. Во время воспроизведения на индикаторе отображается информация о воспроизводимом треке

Текст программы приведен в листинге 6.4. Основную работу в программе выполняет процедура обработки события `StatusUpdate`, которое генерирует компонент `MMControl` (период возникновения события определяет значение свойства `UpdateInterval`). Процедура, в зависимости от состояния плеера, выводит информацию о диске, воспроизводимом треке или сообщение о необходимости вставить в дисковод музыкальный CD. Информация о воспроизводимом треке выводится, если плеер находится в режиме воспроизведения (значение свойства `Mode` равно 526). Здесь следует обратить внимание на то, что свойство `Position` содержит позицию (время) от начала диска, а не от начала трека. Поэтому время, прошедшее от начала воспроизведения трека, вычисляется как разность значений свойств `Position` и `TrackPosition` (положение трека на диске тоже задается от его начала). Также следует обратить внимание, что значение свойства `Track` при переходе к следующему треку автоматически не увеличивается. Поэтому, если значение `Position` выходит за границу текущего трека (`TrackPosition` + `TrackLength`), процедура увеличивает значение свойства `Track` или, если значение `Position` больше, чем `Length`, останавливает процесс воспроизведения. Если диска в дисководе нет (в этом случае плеер находится в состоянии `Ready`) или если диск не музыкальный (плеер находится в состоянии `Stop` и количество треков на диске равно 1), процедура обработки события `StatusUpdate` запускает таймер. Процедура обработки сигнала таймера инвертирует (изменяет на противоположное) значение свойства `Visible` компонента `Label1`, в результате чего текст, находящийся в поле `Label1`, мигает с периодом `Timer1.Interval`. И последнее, на что надо обратить внимание, — процедура обработки события `Unload`, которое возникает в момент завершения работы программы, останавливает процесс воспроизведения и закрывает плеер.

Листинг 6.4. CD-плеер

```
Option Explicit

' начало работы программы
```

```
Private Sub Form_Load()
    MMControl1.DeviceType = "CDAudio"
    MMControl1.Command = "Open"
    MMControl1.TimeFormat = 0 ' формат счета времени — миллисекунды
End Sub

' изменился статус CD, например, в результате щелчка на кнопке управления
Private Sub MMControl1_StatusUpdate()
    Select Case MMControl1.Mode
        Case 525: ' msiModeStop
            If MMControl1.Tracks > 1 Then
                Label1.Caption = "Нажмите кнопку Play"
                Label2.Caption = _
                    "Треков: " + Str(MMControl1.Tracks) + _
                    " Время звучания: " + toHMS(MMControl1.Length)
            Else
                Label1.Caption = "Диск не музыкальный!"
                Timer1.Enabled = True
            End If
        Case 526: ' msiModePlay
            Timer1.Enabled = False
            Label1.Visible = True
            Label2.ForeColor = &H80000012  ' системный цвет Button Text

            If MMControl1.Position < _
                   MMControl1.TrackPosition + MMControl1.TrackLength Then
                ' вывести информацию о воспроизводимом треке
                Label1.Caption = _
                    "Трек: " + Format(MMControl1.Track, "0#") + "  " + _
                    toHMS(MMControl1.Position - MMControl1.TrackPosition)
            Else
                ' воспроизведение трека закончено
                If MMControl1.Position < MMControl1.Length Then
                    ' переход к следующему треку
                    MMControl1.Track = MMControl1.Track + 1
                    Label1.Caption = _
                      "Трек: " + Format(MMControl1.Track, "#") + "  " + _
                    toHMS(MMControl1.Position - MMControl1.TrackPosition)
```

```
                Else
                    MMControl1.Command = "Stop"
                End If
            End If

        Case 529: ' msiModePause
            Label1.Caption = "Pause"
            Timer1.Enabled = True

        Case 530: ' msiModeReady
            Label1.Caption = "В дисководе нет диска!"
            Label2.Caption = "Треков: 0 Время звучания: 0:00"
            Label2.ForeColor = &H80000011  ' системный цвет Disabled Text
            Timer1.Enabled = True
        Case Else
            Label1.Caption = "Mode:" & Str(MMControl1.Mode)
    End Select

End Sub

' таймер обеспечивает мигание индикатора
Private Sub Timer1_Timer()
    DoEvents
    Label1.Visible = Not Label1.Visible
End Sub

' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"  ' остановить воспроизведение CD
    MMControl1.Command = "Close"
End Sub

' функция преобразует целое (время в миллисекундах) в
' строку "чч:мм:сс"
Private Function toHMS(time As Long) As String
    Dim H As Integer     ' часы
    Dim M As Integer     ' минуты
```

```
    Dim S As Integer     ' секунды

    H = Int((time / 1000) / 3600)
    M = Int((time / 1000 Mod 3600) / 60)
    S = (time / 1000 Mod 3600) Mod 60

    If H > 0 Then toHMS = Str(H) + ":"
    toHMS = toHMS + Format(M, "0#") + ":" + Format(S, "0#")
End Function
```

Контрольные вопросы

1. Какие настройки надо задать, чтобы мультимедиаплеер стал CD-проигрывателем?
2. Как определить количество треков CD?
3. Как определить номер воспроизводимого трека?

Регулятор громкости

Проще всего установить требуемую громкость звука можно с помощью стандартного регулятора громкости (рис. 6.11). Но можно регулировать громкость звука и из программы. Рассмотрим, как можно регулировать громкость воспроизведения MP3-файла (громкость воспроизведения WAV-файла регулируется аналогично).

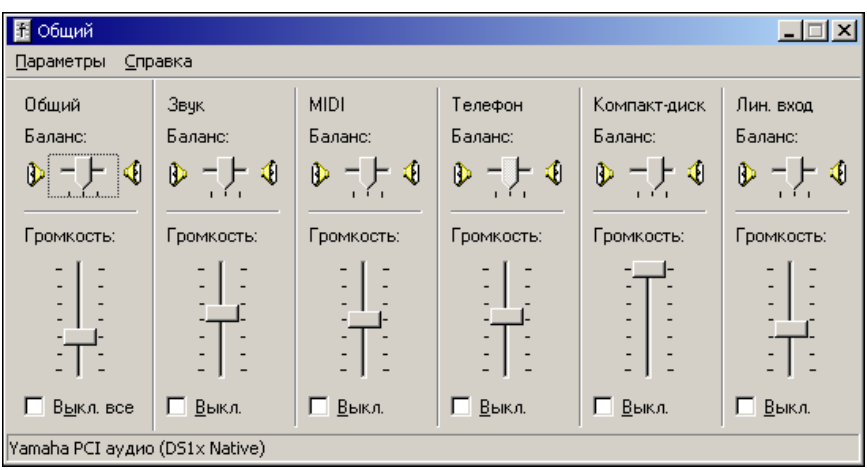


Рис. 6.11. Стандартный регулятор громкости

Задать необходимую громкость воспроизведения WAV-файла можно с помощью API-функции `waveOutSetVolume`. Инструкция вызова функции в общем виде выглядит так:

```
r = waveOutSetVolume(ИдентификаторУстройства, Громкость)
```

Параметр *`ИдентификаторУстройства`* задает устройство воспроизведения, громкость которого надо установить. При регулировке громкости воспроизведения WAV-файла значение этого параметра должно быть равно нулю.

Параметр *`Громкость`* (двойное слово — 4 байта) задает громкость воспроизведения: младшее слово определяет громкость левого канала, старшее — правого. Максимальной громкости звучания канала соответствует шестнадцатеричное значение `FFFF`, минимальной — `0000`. Таким образом, чтобы установить максимальную громкость воспроизведения в обоих каналах, значение параметра *`Громкость`* должно быть `&HFFFFFFFF` (при записи шестнадцатеричных констант в Visual Basic используется префикс `&H`). Уровню громкости 50% соответствует константа `&h7FFF7FFF`.

Необходимо обратить внимание, что функция `waveOutSetVolume` регулирует громкость воспроизведения *звукового канала*, а не общий уровень звука (см. рис. 6.11).

Как можно регулировать громкость воспроизведения MP3-файла, демонстрирует программа "MP3-плеер" (ее окно приведено на рис. 6.12).

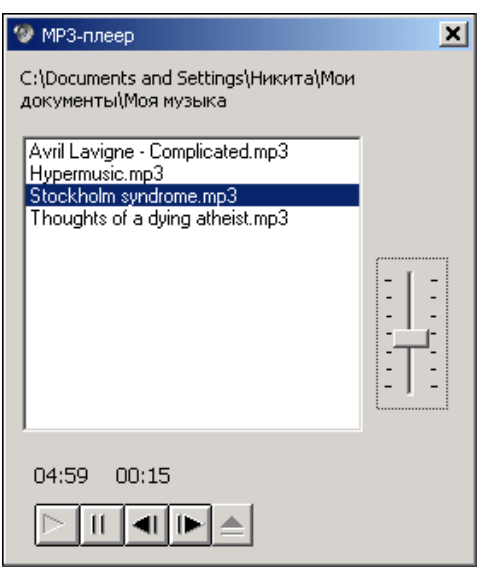


Рис. 6.12. MP3-плеер с регулятором громкости

Изменение громкости осуществляется с помощью компонента `Slider` (чтобы этот компонент стал доступен, надо подключить модуль Microsoft Windows Common Controls-2 6.0 — выбрать в меню **Project** команду **Components** и указать модуль). Значения свойств компонента `Slider` приведены в табл. 6.6. Следует обратить внимание, что при вертикальном расположении компонента минимальному значению свойства `Value` соответствует верхнее положение движка, максимальному — нижнее. Поэтому диапазон изменения значения свойства `Value` задан от –65 535 до 0. И, таким образом, при нулевом значении свойства `Value`, движок регулятора находится в нижнем, а не в верхнем положении.

Текст программы приведен в листинге 6.5.

Таблица 6.6. *Значения свойств компонента* `Slider`

Свойство	Значение
Orientation	ccOrientationVertical
TickStyle	slbBoth
Height	89
Width	42
Min	-65535
Max	0
TickFrequency	10992
SmallChange	256
LargeChange	8192

Листинг 6.5. MP3-плеер с регулятором громкости

```
Option Explicit

' громкость звучания регулируется с помощью API-функций:
' устанавливает значение громкости
Private Declare Function waveOutSetVolume Lib "winmm.dll" _
   (ByVal uDeviceID As Long, ByVal dwVolume As Long) As Long

' возвращает значение громкости
Private Declare Function waveOutGetVolume Lib "winmm.dll" _
```

```
    (ByVal uDeviceID As Long, ByVal dwVolume As Long) As Long

' эти API-функции используются для доступа к окну Обзор папок
Private Declare Function SHBrowseForFolder Lib "shell32" _
    Alias "SHBrowseForFolderA" (ByRef b As Any) As Long
Private Declare Function SHGetPathFromIDList Lib "shell32" _
    (ByVal ResPIDL As Any, ByVal patch As String) As Long

' константы SHELL API
Const CSIDL_DRIVES As Long = 17
Const BIF_RETURNONLYFSDIRS As Long = 1
Const MAX_PATH = 260

Const mciTormatMilleseconds = 0

' эта структура используется для передачи информации
' в функцию SHBrowseFolder, которая выводит диалоговое
' окно Обзор папок
Private Type bi ' browseinfo
    hwndOwner As Long
    pidlRoot As Long
    pszDisplayName As String ' выбранная папка (без пути)
    lpzTitle As String       ' подсказка
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type

Dim status As Integer ' состояние плеера:
                      '    0 — Стоп (Пауза)
                      '    1 — Воспроизведение

' инициализация формы
Private Sub Form_Initialize()
    Dim v As Long ' громкость канала
    Dim st As String * 8
```

```
 Dim aPath As String

 ' настройка компонента Slider
 With Slider1
   .Min = -65535
   .Max = -0
   .SmallChange = 256
   .LargeChange = 8192
   ' чтобы регулятор выглядел как в окне Регулятор громкости
   .Height = 89
   .Width = 42
   .TickFrequency = 10992
 End With

 ' громкость — приблизительно 30% от максимальной
 v = &H2FFF
 st = Hex(v) & Hex(v)
 ' переменная st содержит строковое представление двойного слова
 waveOutSetVolume 0, "&H" & st ' установить уровень громкости
 Slider1.Value = -v ' переместить движок в положение, соответствующее
                    ' установленному уровню громкости

On Error Resume Next
aPath = Environ("homepatch") + File1.Path + _
               "\Мои документы\Моя музыка"
File1.Path = aPath
Label1.Caption = aPath
File1.Pattern = "*.mp3"

' в MMControl1 отображать только кнопки Play, Pause, Step и Back;
' остальные кнопки скрыть
MMControl1.PrevVisible = False
MMControl1.NextVisible = False
MMControl1.RecordVisible = False
MMControl1.EjectVisible = False

Form1.ScaleMode = vbPixels

File1.ListIndex = 0
```

```
   status = 0 ' плеер в режиме Стоп
End Sub

' событие Scroll возникает в процессе перемещения движка мышью,
' и клавишами "курсор вверх", "курсор вниз", <PgUp> и <PgDown>
Private Sub Slider1_Scroll()
    Dim v As Long     ' громкость одного канала
    Dim st As String ' строковое представление
                      ' двойного беззнакового слова

    v = Abs(Slider1.Value)
    st = "000" + Hex(v)
    st = Right(st, 4)
    st = st + st

    waveOutSetVolume 0, "&H" + st
End Sub

' щелчок на имени файла или программное изменение
' значения свойства ListIndex
Private Sub File1_Click()
   If MMControl1.Command <> "CLOSE" Then MMControl1.Command = "CLOSE"

   ' Label1.Caption = File1.Path + "\" + File1.FileName
   If File1.FileName <> "" Then
      MMControl1.FileName = File1.Path + "\" + File1.FileName

      MMControl1.TimeFormat = mciTormatMilleseconds
      MMControl1.Command = "OPEN"

      Label2.Caption = toHMS(MMControl1.Length)
      Label3.Caption = "00:00"
      If status = 1 Then
        MMControl1.Command = "Play"
      End If
   End If
End Sub
```

```
' щелчок на кнопке Пауза
Private Sub MMControl1_PauseClick(Cancel As Integer)
    ' плеер сам выполнит нужное действие, мы же
    ' только зафиксируем его статус
    If status = 0 Then
        status = 1
        Command1.Enabled = False
    Else
        status = 0
        Command1.Enabled = True
    End If
End Sub

' щелчок на кнопке Play
Private Sub MMControl1_PlayClick(Cancel As Integer)
    status = 1
    Command1.Enabled = False
End Sub

' щелчок на кнопке Стоп
Private Sub MMControl1_StopClick(Cancel As Integer)
    status = 0
    Command1.Enabled = True
End Sub

' щелчок на кнопке Step — перейти к следующему файлу списка
Private Sub MMControl1_StepClick(Cancel As Integer)
    If File1.ListIndex < File1.ListCount - 1 Then
        File1.ListIndex = File1.ListIndex + 1
        If File1.ListIndex = File1.ListCount Then
            MMControl1.StepEnabled = False
        End If
    Else
        File1.ListIndex = 0
    End If
End Sub
```

```
' щелчок на кнопке Back — перейти к предыдущему файлу списка
Private Sub MMControl1_BackClick(Cancel As Integer)
    If File1.ListIndex > 0 Then
        File1.ListIndex = File1.ListIndex - 1
    Else
        File1.ListIndex = File1.ListCount - 1
    End If
End Sub

' обработка сигнала StatusUpdate, генерируемого MMControl
Private Sub MMControl1_StatusUpdate()
    If MMControl1.Position < MMControl1.Length Then
        ' продолжается воспроизведение текущей композиции
        Label3.Caption = toHMS(MMControl1.Position)
    Else
        ' воспроизведение текущей композиции закончено
        If File1.ListIndex < File1.ListCount - 1 Then
            File1.ListIndex = File1.ListIndex + 1
        Else
            MMControl1.Command = "Stop"
            status = 0
            Command1.Enabled = True
        End If
     End If
End Sub

' щелчок на кнопке Eject (Выбор каталога)
Private Sub Command1_Click()
    Dim ResPIDL As Long
    Dim t As bi
    Dim Path As String

    Dim r As Long
    Dim p As String

    t.hwndOwner = Form1.hWnd
    t.lpzTitle = "Выберите папку, в которой находятся MP3-файлы"
```

```
    t.ulFlags = BIF_RETURNONLYFSDIRS ' кнопка OK доступна, если
                                     ' пользователь выбрал папку
    t.pszDisplayName = String(MAX_PATH, 0)

    ResPIDL = SHBrowseForFolder(t) ' вывести окно Обзор папок
    If ResPIDL <> 0 Then
        ' пользователь выбрал папку
        ' получить ее полное имя
        Path = String(MAX_PATH, Chr(0))
        r = SHGetPathFromIDList(ResPIDL, Path)
        ' строка Path — это Си-строка, содержащая
        ' NULL-символ (код 0). До этого символа находится
        ' полезная информация, после — мусор

        p = InStr(1, Path, Chr(0)) ' положение NULL-символа
        Path = Mid(Path, 1, p - 1)
        If Mid(Path, p - 1, 1) <> "\" Then Path = Path + "\"

        File1.Path = Path   ' отобразить содержимое выбранного каталога
                            ' в поле компонента File1 (FileList)
        File1.ListIndex = 0 ' в результате возникает событие File1.Click
    End If
End Sub

' функция преобразует целое (время звучания или текущую позицию
' воспроизведения) в формат "чч:мм:сс"
Private Function toHMS(time As Long) As String
    Dim H As Integer    ' часы
    Dim M As Integer    ' минуты
    Dim S As Integer    ' секунды

    H = Int((time / 1000) / 3600)
    M = Int((time / 1000 Mod 3600) / 60)
    S = (time / 1000 Mod 3600) Mod 60

    If H > 0 Then toHMS = Str(H) + ":"
    toHMS = toHMS + Format(M, "0#") + ":" + Format(S, "0#")
End Function
```

```
' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
      MMControl1.Command = "STOP"    ' остановить воспроизведение
      MMControl1.Command = "CLOSE"   ' "выключить" плеер
End Sub
```

Непосредственное изменение громкости осуществляет процедура обработки события `Scroll` регулятора громкости (компонента `Slider`), которое периодически генерируется в процессе перемещения движка регулятора мышью или клавишами <PgUp>, <PgDown> и клавишами перемещения курсора <↑> или <↓>. Процедура обработки этого события сначала преобразует значение свойства `Value` в шестнадцатеричную четырехразрядную строковую константу, затем формирует шестнадцатеричную восьмиразрядную строковую константу, первые четыре разряда которой определяют громкость правого канала, вторые — левого. Сформированная строковая константа передается функции `waveOutSetVolume`. Использование такого "хитрого" формирования параметра функции `waveOutSetVolume` объясняется следующим. Параметр *Громкость* должен быть беззнаковым двойным словом. Если бы в Visual Basic такой тип был, то в программе можно было бы записать так:

```
Dim vl as Word
Dim vl as Word
Dim vl as DoubleWord
vl = &H7777
vr =&H7777
v = vl or vr
waveOutSetVolume 0, v
```

Но в Visual Basic такого типа нет. Наиболее близким (на первый взгляд) является тип `Long` (4 байта), но он позволяет корректно изменять в пределах всего диапазона громкость только левого канала (два младших байта). Громкость же правого канала (два старших байта) при использовании этого типа можно менять только в диапазоне от 00000 до 7FFF, т. к. старший двоичный разряд числа типа `Long` является знаковым. Попытка увеличить громкость правого канала на единицу (прибавить `&H10000` к числу типа `Long`, старшие четыре разряда которого содержат `&H7FFF`) приводит к изменению знака и потере значения переменной, содержащей значение громкости.

Контрольные вопросы

1. Что надо сделать, чтобы функция `waveOutGetVolume`, позволяющая получить текущее значение громкости воспроизведения звука, стала доступной?

2. Что надо сделать, чтобы функция `waveOutSetVolume`, позволяющая регулировать громкость воспроизведения звука, стала доступной?
3. Что представляет собой параметр функции `waveOutSetVolume`, устанавливающий требуемый уровень громкости? Опишите его формат.

Регулировка громкости MIDI

Регулировку громкости воспроизведения MIDI обеспечивает функция `midiOutSetVolume`. Параметры у функции `midiOutSetVolume` такие же, как и у функции `waveOutSetVolume`. Но есть одна особенность. В системе может быть несколько MIDI-устройств. Какое из этих устройств используется для воспроизведения MIDI в данный момент и, следовательно, громкость какого устройства должна регулировать программа, определяют настройки операционной системы. Поэтому использовать константу в качестве идентификатора устройства (первого параметра функции `midiOutSetVolume`) не рекомендуется, лучше определить идентификатор активного устройства во время запуска программы.

Как это сделать, показывает программа "Регулятор громкости MIDI" (ее окно приведено на рис. 6.13, а текст — в листинге 6.6).

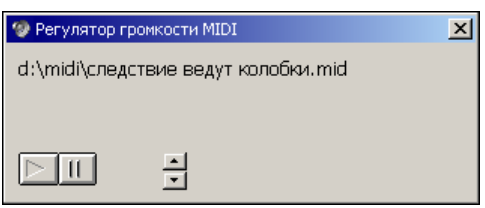


Рис. 6.13. Регулятор громкости MIDI

Листинг 6.6. Регулятор громкости MIDI

```
Option Explicit

' громкость звучания регулируется с помощью API-функции
Private Declare Function midiOutSetVolume Lib "winmm.dll" _
      (ByVal uDeviceID As Long, ByVal dwVolume As Long) As Long

' uDeviceID — идентификатор устройства, громкость которого регулируется
' dwVolume — громкость (старшее слово задает громкость правого
'                       канала, младшее — левого)
```

```
Private Declare Function midiOutGetNumDevs Lib "winmm" () As Integer

Dim id As Long ' идентификатор MIDI-устройства, которое регулируем

Private Sub Form_Load()
    Dim n As Integer ' кол-во MIDI-устройств в системе
    Dim r As Long    ' рез-т выполнения API-функции
    Dim v As Long
    Dim i As Long

    ' в системе может быть несколько MIDI-устройств,
    ' выясним, какое из них активно. Для этого попробуем
    ' задать громкость
    n = midiOutGetNumDevs
    For i = 0 To n - 1
        r = midiOutSetVolume(i, "&H00FF00FF")
        If r = 0 Then
            ' MsgBox "id=" & Str(i) & "r=" & Str(r)
            id = i
            Exit For
        End If
    Next i

    UpDown1.Min = 0
    UpDown1.Max = 65535 ' максимальный уровень громкости — FFFF
    UpDown1.Increment = (UpDown1.Max - UpDown1.Min) / 20
    UpDown1.Value = UpDown1.Increment * 6 ' установить громкость
                                          ' приблизительно 30% от максимума

    MMControl1.FileName =  Dir("*.mid")
    If MMControl1.FileName <> "*.mid" Then
        Label2.Caption = MMControl1.FileName
        MMControl1.Command = "OPEN"
        MMControl1.Command = "Play"
    Else
        Label1.Caption = "В текущем каталоге нет MIDI-файлов"
    End If
```

```
End Sub

' событие Change возникает как реакция на изменение свойства Value,
' которое, в свою очередь, автоматически изменяется в результате щелчка
' на одной из кнопок компонента UpDown
Private Sub UpDown1_Change()
    Dim v As Currency
    Dim s As String
    Dim r As Long

    v = UpDown1.Value
    ' в качестве параметра value функции midiOutSetVolume надо передать
    ' двойное слово, но такого типа в VB нет. Поэтому формируем
    ' строку (шестнадцатеричное восьмиразрядное число) и передаем ее
    ' функции midiOutSetVolume

    s = Right("000" & Hex(v), 4)
    r = midiOutSetVolume(id, "&H" & s & s)
End Sub

' обработка сигнала StatusUpdate от MMControl
Private Sub MMControl1_StatusUpdate()
    If MMControl1.Position >= MMControl1.Length Then
        MMControl1.Command = "Stop"
        MMControl1.Command = "Close"
        MMControl1.FileName = Dir ' получить имя следующего MIDI-файла
        If MMControl1.FileName <> vbNull Then
            Label2.Caption = MMControl1.FileName
            MMControl1.Command = "Open"
            MMControl1.Command = "Play"
        Else
            ' получить имя MIDI-файла из текущего каталога
            MMControl1.FileName = Dir("*.mid")
            Label2.Caption = MMControl1.FileName
            MMControl1.Command = "Open"
            MMControl1.Command = "Play"
        End If
    End If
```

```
End Sub

' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "STOP"
    MMControl1.Command = "CLOSE"
End Sub
```

Работает программа следующим образом. Сразу после запуска процедура обработки события `Load` обращается к API-функции `midiOutGetNumDevs`, которая возвращает количество MIDI-устройств, установленных в системе. Затем делается попытка изменить громкость каждого из устройств и тем самым определить идентификатор активного устройства (если устройство активно, то значение функции `midiOutSetVolume` равно нулю). После этого выполняется настройка компонента `UpDown` и загрузка из текущего каталога MIDI-файла. Громкость регулирует процедура обработки события `Change` компонента `UpDown`. Следует обратить внимание, что программа спроектирована таким образом, что после воспроизведения текущего MIDI-файла делается попытка загрузить следующую мелодию, и, если это удается, автоматически активизируется процесс воспроизведения. Если загрузить следующий файл не удается (в этом случае значение функции `Dir` равно `vbNull`), функция `Dir` вызывается снова, но уже с параметром. В результате снова загружается первый из уже проигранных файлов. Таким образом, MIDI-файлы текущего каталога проигрываются непрерывно.

Контрольные вопросы

1. В чем состоит особенность регулировки громкости MIDI-устройства?
2. Какая функция обеспечивает регулировку громкости MIDI-звука?
3. Что надо сделать, чтобы функция `midiOutSetVolume` стала доступной?

Просмотр видеороликов

Как было сказано ранее, компонент `MMControl` позволяет просматривать видеоролики и сопровождаемую звуком анимацию. В качестве примера использования компонента для решения этой задачи рассмотрим программу Video (рис. 6.14), с помощью которой можно просмотреть клип или анимацию, представленную в формате AVI.

Форма программы Video приведена на рис. 6.15, значения свойств компонента `MMControl1` — в табл. 6.7. Компонент `Picture1` используется в качестве экрана, на поверхности которого отображается клип. Компонент `CommonDialog1` в дан-

ной программе представляет собой стандартное диалоговое окно **Открыть файл**, которое становится доступным в результате щелчка на кнопке **Eject** и используется для выбора AVI-файла. Здесь следует обратить внимание, что управление работой видеоплеера осуществляется не с помощью кнопок компонента `MMControl`, а с помощью командных кнопок **Play** и **Eject**, поэтому свойству `Visible` компонента `MMControl` присвоено значение `False`.

Рис. 6.14. Простой видеоплеер на основе компонента `MMControl`

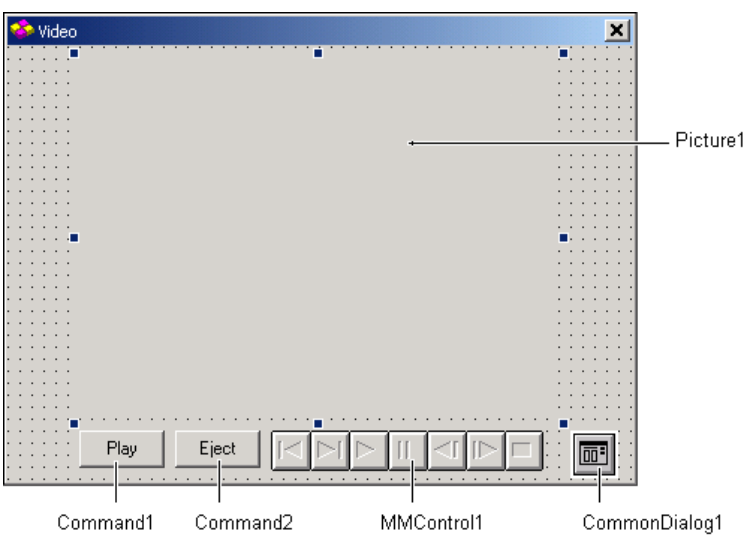


Рис. 6.15. Форма программы Video

Таблица 6.7. *Значения свойств компонента* `MMControl1`

Свойство	Значение	Комментарий
`DeviceType`	`AVIVideo`	
`hWndDisplay`	`Picture1.hWnd`	Свойство доступно только во время работы программы. Значение устанавливает процедура обработки события `Load`
`Visible`	`False`	Кнопки компонента не используются для управления работой плеера

Тест программы Video приведен в листинге 6.7. В начале своей работы программа настраивает компонент `MMControl1`: присваивает значение свойству `hWhdDisplay` и делает попытку загрузить AVI-файл из текущего каталога. Для проверки существования файла используется функция `Dir`, которая возвращает имя файла, соответствующее маске, указанной в качестве параметра функции. Если в текущем каталоге (в каталоге, из которого запущена программа) есть файл с расширением AVI, то компоненту `MMControl1` направляются команды `Open` и `Next`, в результате чего в поле компонента `Picture1` появляется первый кадр видеоролика. Следует обратить внимание, что кнопка `Command1` используется как для активизации, так и для приостановки процесса воспроизведения. В зависимости от режима работы плеера на кнопке находится текст **Play** или **Pause**. Если процесс воспроизведения приостановлен, то щелчок на кнопке `Command1` активизирует процесс воспроизведения. Если плеер находится в режиме воспроизведения, то щелчок на кнопке `Command1` приостанавливает воспроизведение. В процессе воспроизведения компонент `MMControl1` генерирует событие `StatusUpdate`. Процедура обработки этого события контролирует положение указателя текущей позиции и, если указатель достиг конца воспроизводимого ролика, останавливает процесс воспроизведения.

Листинг 6.7. Видеоплеер

```
Option Explicit

' эти константы используются для управления
' диалогом Открыть файл (Выбор AVI-файла)
Const cdlOFNHideReadOnly = &H4  ' не отображать CheckBox "Только чтение"
Const cdlOFNFileMustExist = &H1000 ' в поле Имя файла можно ввести
                                   ' имя только существующего файла

' начало работы программы
```

```
Private Sub Form_Load()

    ' настройка MMControl
     MMControl1.hWndDisplay = Picture1.hWnd ' экран - компонент Picture1

    ' попробуем загрузить AVI-файл из текущего каталога
    ' MMControl1.FileName = Dir("*.avi")
    If MMControl1.FileName <> "" Then
        ' в текущем каталоге есть AVI-файл
        MMControl1.Command = "Open"
        MMControl1.Command = "Next" ' отобразить первый кадр
        Command1.Enabled = True ' кнопка Play доступна
    End If

    ' настройка CommonDialog
    CommonDialog1.DialogTitle = "Выбор AVI-файла"
    CommonDialog1.FileName = "*.avi"
    CommonDialog1.Flags = cdlOFNHideReadOnly + cdlOFNFileMustExist
End Sub

' щелчок на кнопке Play/Pause
Private Sub Command1_Click()
    If (MMControl1.Mode = 525) Or _
       (MMControl1.Mode = 529) Then ' 525 — mciModeStop,
                                    ' 529 — mciModePause
        ' плеер в режиме "Стоп" или "Пауза"
        MMControl1.Command = "Play"
        Command1.Caption = "Pause"
        Command2.Enabled = False
    Else
        ' плеер в режиме "Воспроизведение"
        If MMControl1.Mode = 526 Then ' 526 — mciModePlay
            MMControl1.Command = "Pause"
            Command1.Caption = "Play"
            Command2.Enabled = True
        End If
    End If
End Sub
```

```
' щелчок на кнопке Eject (выбор файла для просмотра)
Private Sub Command2_Click()
    CommonDialog1.FileName = "*.avi"
    CommonDialog1.ShowOpen
    If CommonDialog1.FileName <> "*.avi" Then
        ' пользователь выбрал файл
        MMControl1.Command = "Close"
        MMControl1.FileName = CommonDialog1.FileName
        MMControl1.Command = "Open"
        MMControl1.Command = "Next" ' отобразить первый кадр
        Command1.Enabled = True
    End If

End Sub

' обработка сигнала StatusUpdate от плеера
Private Sub MMControl1_StatusUpdate()
    If MMControl1.Position >= MMControl1.Length Then
        MMControl1.Command = "Stop"
        MMControl1.Command = "Prev"
        Command1.Caption = "Play"
        Command2.Enabled = True
    End If
End Sub

' завершение работы программы
Private Sub Form_Unload(Cancel As Integer)
    MMControl1.Command = "Stop"
    MMControl1.Command = "Close"
End Sub
```

Контрольные вопросы

1. Как задать параметры, чтобы мультимедиаплеер стал видеопроигрывателем?
2. Видеофайлы какого формата может проигрывать (воспроизводить) компонент `MMControl`?
3. Как задать "экран" для воспроизведения видеоролика?

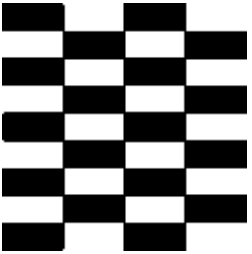
Установка программы на другой компьютер

Мультимедийную программу, созданную в Visual Basic, можно перенести на другой компьютер обычным образом. Однако нужно обратить внимание на следующий важный момент. Компонент `MMControl` — это ActiveX-компонент (динамическая библиотека mci32.ocx). Возможно, что на компьютере пользователя его нет, и поэтому программа работать не будет. Таким образом, чтобы мультимедийная программа, использующая компонент `MMControl`, работала на другом компьютере, пользователю необходимо передать не только выполняемый (EXE) файл, но и файл mci32.ocx. Этот файл надо поместить в системный каталог Windows (C:\Windows\System32). Следует обратить внимание на то, что библиотеку надо не просто поместить в указанный каталог, но и *зарегистрировать* в системе. Для этого в окне **Запуск программы** (которое становится доступным в результате выбора команды **Пуск ▸ Выполнить**) надо набрать команду `regsvr32 mci32.ocx`.

Если вы не уверены, что пользователь сможет самостоятельно выполнить установку приложения, то лучше с помощью специальной утилиты, например, IExpress (она входит в состав Windows).

Контрольные вопросы

1. Какую библиотеку надо установить на компьютер пользователя, чтобы программа, использующая компонент `MMControl`, работала на нем?
2. Достаточно ли просто скопировать файл mci32.ocx в папку System32 компьютера пользователя или надо сделать что-то еще? Что именно?



Глава 7

Базы данных

С точки зрения пользователя, *база данных* — это приложение, которое обеспечивает работу с информацией. При его запуске на экране, как правило, появляется таблица, просматривая которую можно найти нужные сведения. Если система позволяет, то пользователь может внести изменения в базу данных: добавить новую информацию или удалить ненужную.

С точки зрения программиста, *база данных* — это набор файлов, в которых находится информация. Разрабатывая *базу данных для пользователя*, программист создает программу, которая обеспечивает работу с файлами данных.

В состав Visual Basic включены компоненты, используя которые программист может создать программу работы практически с любой из существующих баз данных: от Microsoft Access до Microsoft SQL Server и Oracle.

База данных и СУБД

База данных — это набор файлов, в которых находится информация. Программная система, обеспечивающая работу с базой данных, называется *системой управления базой данных* (СУБД). Microsoft Access — это пример типичной СУБД. Следует обратить внимание, что вместо термина "СУБД" часто используется термин *"база данных"*, при этом файлы данных и СУБД рассматриваются как единое целое.

Локальные и удаленные базы данных

В зависимости от расположения данных и программы, которая манипулирует данными, различают *локальные* и *удаленные* базы данных (БД).

В локальной БД файлы данных находятся на диске того компьютера, на котором работает программа манипулирования данными (СУБД). Локальные

базы данных не обеспечивают одновременный доступ к информации нескольким пользователям. Несомненным достоинством локальной базы данных является высокая скорость доступа к информации. Microsoft Access — это локальные базы данных.

Данные удаленной базы данных размещают на отдельном, удаленном от пользователя и доступном по сети компьютере. Программы работы с удаленными базами данных строят по технологии "клиент-сервер". Клиентская часть СУБД (клиент) работает на компьютере пользователя. Она, взаимодействуя с программой-сервером, обеспечивает отображение данных, прием команд пользователя и передачу команд серверу. Серверная часть СУБД (сервер) работает на удаленном компьютере, принимает запросы (команды) от клиента, выполняет их и пересылает данные клиенту. Программа, работающая на удаленном компьютере, проектируется так, чтобы обеспечить одновременный доступ к базе данных многим пользователям. Microsoft SQL Server 2000, MySQL и Oracle — это удаленные базы данных.

Структура базы данных

В настоящее время на практике наиболее широко используются *реляционные* базы данных.

Реляционная база данных представляет собой совокупность таблиц, в которых находится информация. Обычно каждая таблица содержит информацию об однотипных объектах. Например, базу данных "Проекты" можно представить как совокупность таблиц Projects (Проекты), Tasks (Задачи) и Resources (Ресурсы).

Строки таблиц называют *записями*. Записи содержат информацию об *объектах*, каждая запись — об одном. Например, записи таблицы Projects содержат общую информацию о проектах (идентификатор проекта, название, дата начала, дата завершения), таблица Tasks — о задачах проекта (идентификатор проекта, частью которого является задача, название задачи, даты начала и завершения, идентификатор ресурса, обеспечивающего выполнение задачи), а таблица Resources — о ресурсах (идентификатор, название и тип ресурса).

Записи состоят из *полей*. Поля (ячейки таблицы) хранят информацию о характеристиках объектов. Например, записи таблицы Projects могут образовывать поля ID (Идентификатор проекта), Title (Название проекта), Start (Дата начала), Finish (Дата завершения). При изображении таблицы названия (имена) полей обычно указывают в заголовках столбцов.

Технологии доступа к данным

Существует достаточно много технологий доступа к базам данных. В Microsoft Visual Basic 6.0 основной технологией доступа к данным является технология Microsoft ActiveX Data Objects (ADO — объекты доступа к данным). Поддержка технологии ADO в Visual Basic реализована в виде набора компонентов, обеспечивающих доступ к данным и манипулирование ими (`Adodc`, `DataGrid` и др.).

Контрольные вопросы

1. Что такое база данных?
2. Что такое СУБД?
3. Из чего состоит база данных?
4. Из чего состоят записи?
5. Что такое ADO?

Компоненты доступа и отображения данных

Доступ к базе данных обеспечивает компонент `Adodc`, отображение данных в виде таблицы — компонент `DataGrid` (рис. 7.1). Чтобы эти компоненты стали доступными, их надо подключить: выбрать в меню **Project** команду **Components** и установить во включенное состояние (рис. 7.2) флажки соответствующих этим компонентам модулей (Microsoft ADO Data Control 6.0 и Microsoft DataGrid Control 6.0).

Следует обратить внимание, что компонент `Adodc` обращается к данным не напрямую, а через ядро баз данных Microsoft Jet. Механизм доступа к данным и взаимодействие компонентов, обеспечивающих доступ к данным и их отображение, показан на рис. 7.3.

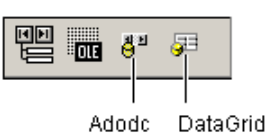


Рис. 7.1. Компонент `Adodc` обеспечивает доступ к данным, `DataGrid` — отображение данных в виде таблицы

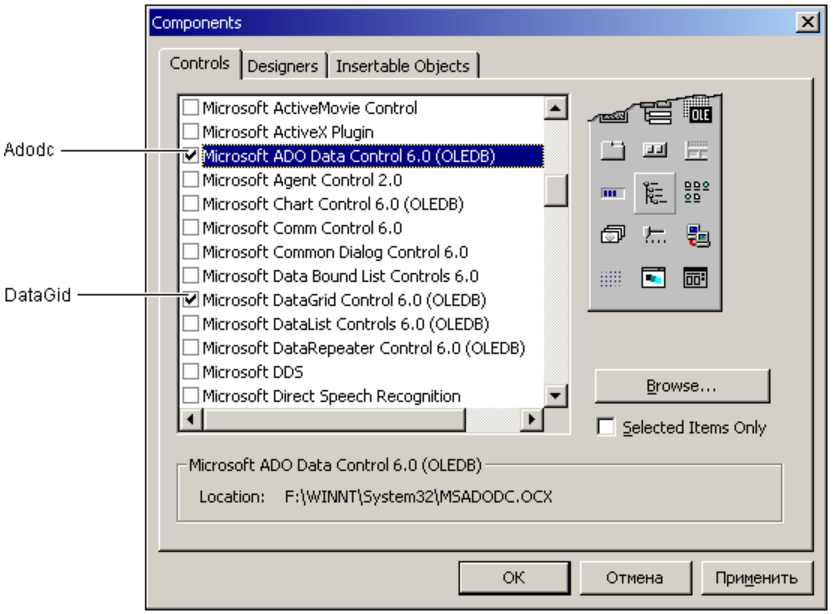


Рис. 7.2. Подключение компонентов `Adodc` и `DataGrid`

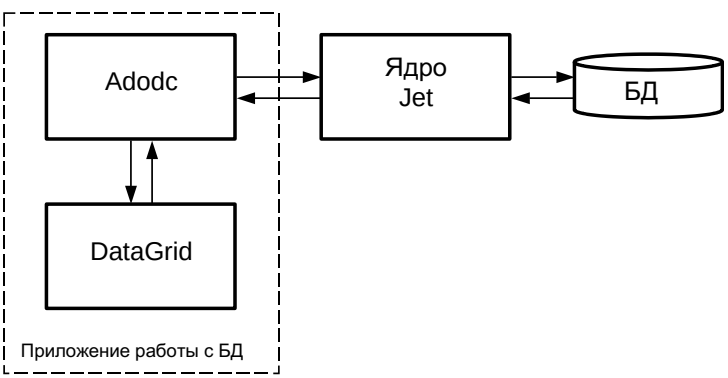


Рис. 7.3. Взаимодействие компонентов, обеспечивающих доступ/отображение данных

Контрольные вопросы

1. Какой компонент обеспечивает соединение с базой данных?
2. Какой компонент обеспечивает отображение базы данных в табличной форме?
3. Что надо сделать, чтобы компоненты `Adodc` и `DataGrid` стали доступны?

Строка соединения

Свойство `ConnectionString` (Строка соединения) компонента `Adodc` определяет базу данных, доступ к которой обеспечивает компонент. Например, строка соединения, обеспечивающая доступ к базе данных Microsoft Access "Адресная книга", файл которой AdrBk.mdb находится в каталоге D:\Database, выглядит следующим образом:

```
Provider=Microsoft.Jet.OLEDB.3.51;Data Source=D:\Database\ADRBK.MDB
```

Приложение работы с базой данных

Процесс создания приложения работы с базой данных рассмотрим на примере программы, обеспечивающей работу с базой данных Microsoft Access "Контакты".

Создание базы данных

Перед тем как приступить к непосредственной работе в Visual Basic, надо с помощью Microsoft Access создать базу данных "Контакты" (contacts.mdb).

База данных "Контакты" содержит одну-единственную таблицу — Contacts (табл. 7.1).

Таблица 7.1. *Поля таблицы Contacts*

Поле	Тип	Размер	Комментарий
Name	Текстовый	50	Имя (имя, фамилия, отчество). Обязательное поле
Phone	Текстовый	50	Телефон
Comment	Текстовый	100	Дополнительная информация

Работа с базой данных в режиме таблицы

После того как база данных будет создана, надо активизировать процесс создания нового приложения и подключить компоненты `Adodc` и `DataGrid` (модули Microsoft ADO Data Control 6.0 и Microsoft DataGrid Control 6.0). Затем на форму приложения надо поместить компоненты `Adodc` и `DataGrid` (рис. 7.4). После этого можно приступить к настройке компонентов.

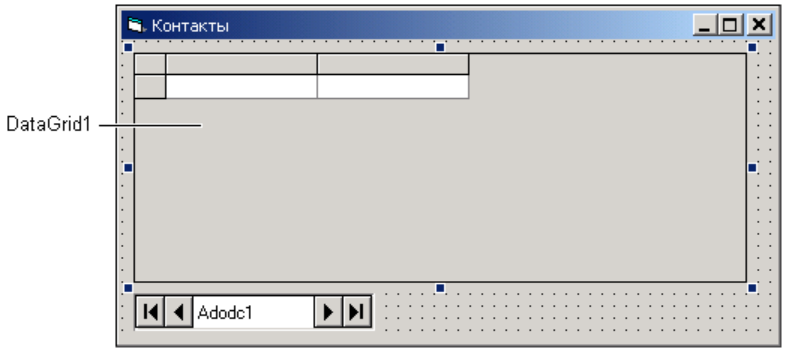


Рис. 7.4. Форма приложения работы с базой данных "Контакты"

Сначала надо настроить компонент `Adodc` — задать значение свойств `ConnectionString` и `RecordSource`. Свойство `ConnectionString` — *строка соединения* — должно содержать информацию, необходимую для подключения к базе данных.

Чтобы настроить соединение (компонент `Adodc`), надо сделать щелчок на кнопке с тремя точками, которая находится в строке свойства `ConnectionString`, в появившемся окне **Property Pages** выбрать **Use Connection String** и сделать щелчок на кнопке **Build** (рис. 7.5). Затем на вкладке **Поставщик данных** окна **Свойства связи с данными** надо выбрать "поставщика" данных (Microsoft Jet 4.0 OLE DB Provider) и нажать кнопку **Далее** (рис. 7.6).

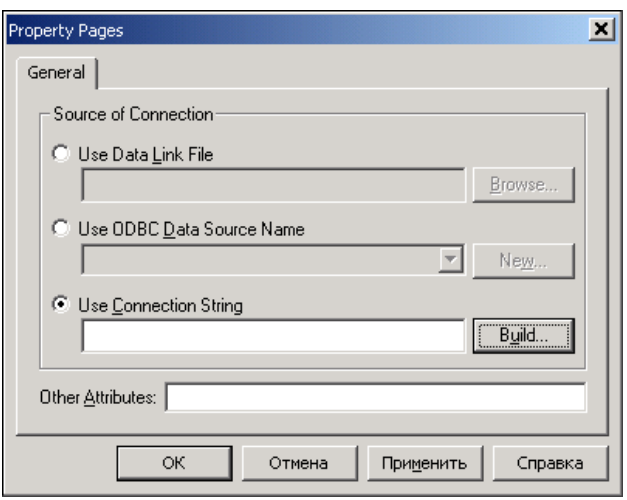


Рис. 7.5. Чтобы создать строку соединения, надо нажать кнопку **Build**

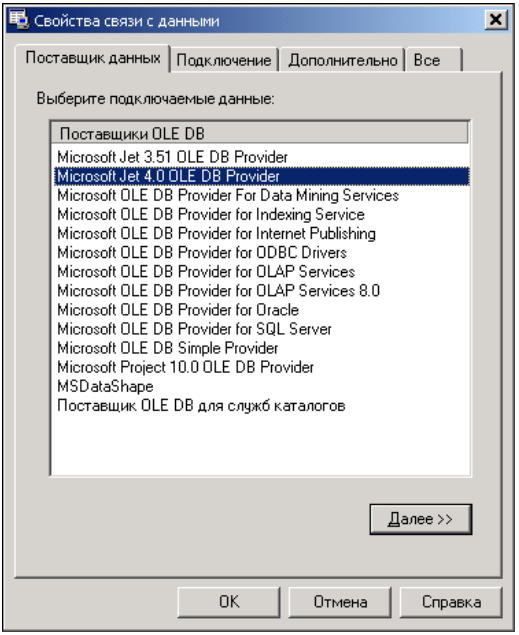


Рис. 7.6. Настройка соединения с базой данных (шаг 1)

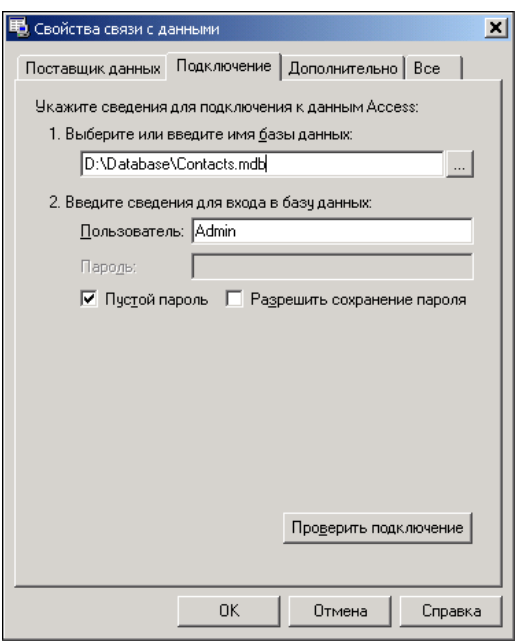


Рис. 7.7. Настройка соединения с базой данных (шаг 2)

Далее на вкладке **Подключение** надо задать файл базы данных (рис. 7.7). После этого можно сделать щелчок на кнопке **Проверить подключение** и убедиться, что соединение настроено, и щелчком на кнопке **OK** закрыть сначала окно **Свойство связи с данными**, затем — **Property Pages**.

В результате описанных действий в поле свойства `ConnectionString` компонента `Adodc` появится строка соединения.

Свойство `RecordSource` компонента `Adodc` определяет данные, которые надо извлечь из базы данных. Это может быть как вся таблица (все записи), так и некоторая ее часть (записи, удовлетворяющие некоторому условию), а также данные, полученные из нескольких таблиц. Чтобы определить, какую информацию необходимо получить из базы данных, надо в свойство `RecordSource` записать SQL-команду, обеспечивающую выбор информации. Например, команда

```
SELECT Name, Phone FROM Contacts
```

обеспечивает выбор из таблицы Contacts информации, которая находится в столбцах Name и Phone.

SQL-команду можно ввести непосредственно в поле значения свойства `RecordSource` или в поле **Command Text** (рис. 7.8) вкладки **RecordSource** окна **Property Pages**, появляющегося в результате щелчка на кнопке с тремя точками, которая находится в строке свойства `RecordSource`.

Значения свойств компонента `Adodc` приведены в табл. 7.2.

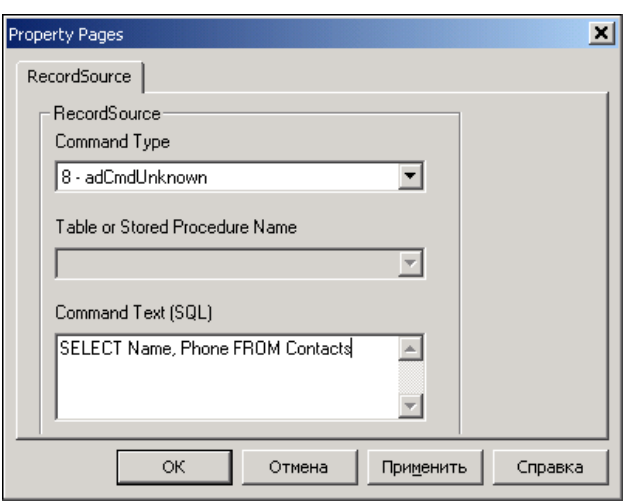


Рис. 7.8. В поле **Command Text** надо ввести SQL-команду, обеспечивающую выбор информации из базы данных

Таблица 7.2. *Значения свойств компонента* `Adodc`

Свойство	Значение

`ConnectionString`	`Provider=Microsoft.Jet.OLEDB.4.0;`
	`Data Source=D:\Database\Contacts.mdb;`
	`Persist Security Info=False`

`RecordSource`	`SELECT Name, Phone FROM Contacts ORDER BY Name`
`Visible`	`False`

Замечание

В строке соединения путь к файлу базы данных можно не указывать. В этом случае файл базы данных будет загружаться из текущего каталога. При запуске программы из Visual Basic текущим каталогом является каталог пользователя. При запуске программы из операционной системы текущим является каталог, в котором находится выполняемый (EXE) файл.

Следующее, что надо сделать, — настроить компонент `DataGrid`, который обеспечивает отображение данных в табличной форме, а также позволяет манипулировать данными (редактировать, добавлять и удалять строки таблицы). Свойства компонента `DataGrid` (табл. 7.3) определяют данные, которые отображаются в таблице.

Таблица 7.3. *Свойства компонента* `DataGrid`

Свойство	Определяет
`DataSource`	Источник данных, отображаемых в поле компонента (компонент `Adodc`)
`AllowAddNew`	Признак возможности добавить в таблицу новую запись
`AllowDelete`	Признак возможности удалить запись из таблицы
`AllowUpdate`	Признак возможности редактировать записи таблицы
`Caption`	Заголовок таблицы. Если заголовок не задан, строка заголовка не отображается
`ColunnHeaders`	Признак отображения заголовков столбцов таблицы. По умолчанию в качестве заголовков столбцов используются имена полей таблицы данных. Если значение свойства равно `False`, то заголовки столбцов не отображаются
`HeadFont`	Шрифт, используемый для отображения заголовка таблицы и заголовков столбцов
`Font`	Шрифт, используемый для отображения данных

Нужно обратить внимание, что порядок следования столбцов в поле компонента DataGrid соответствует порядку следования имен столбцов, указанному в команде SELECT, или порядку следования столбцов в исходной таблице, если после слова SELECT указана звездочка.

По умолчанию в заголовке столбца компонента DataGrid отображается название поля, что не всегда удобно. Задать заголовок столбца компонента DataGrid можно явно на вкладке **Columns** окна **Property Pages**, которое становится доступным в результате выбора в контекстном меню компонента DataGrid команды **Properties**. Текст заголовка надо ввести в поле **Caption** (рис. 7.9), предварительно выбрав в списке **Column** столбец компонента DataGrid, а в списке **DataField** — столбец таблицы данных.

Значения свойств компонента DataGrid1 приведены в табл. 7.4, вид формы после настройки компонента — на рис. 7.10.

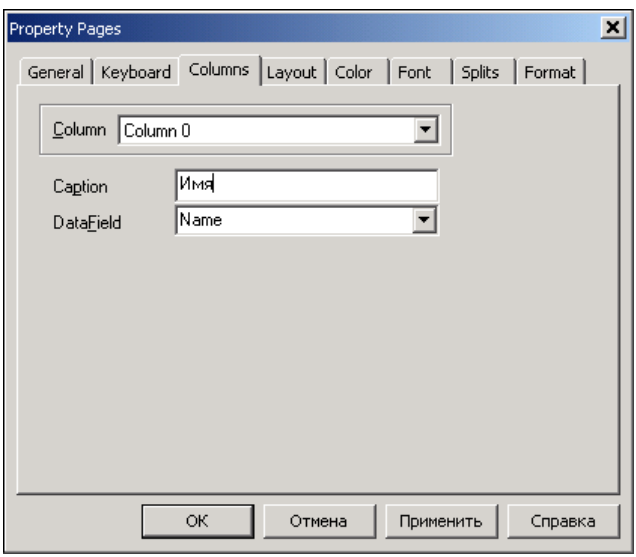


Рис. 7.9. Текст заголовка столбца надо ввести в поле **Caption**

Таблица 7.4. *Значения свойств компонента* DataGrid1

Свойство	**Значение**
DataSource	Adodc1
AllowAddNew	True
AllowDelete	True
AllowUpdate	True

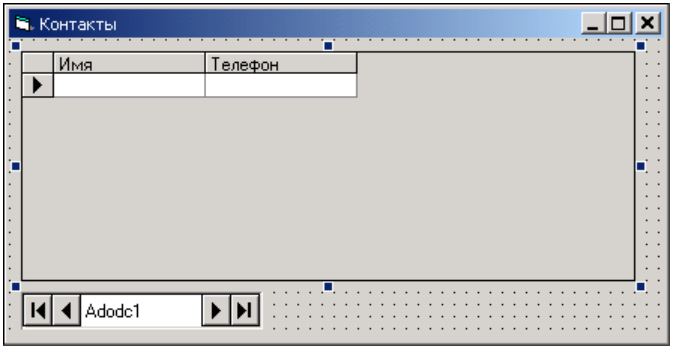


Рис. 7.10. Вид формы после настройки компонента `DataGrid1`

На этом настройка компонентов приложения заканчивается. В принципе, простейшая программа работы с базой данных готова. Программу можно запустить и начать работу с базой данных.

Завершив работу по созданию формы приложения, можно приступить к созданию процедур обработки событий. В простейшем случае надо создать процедуру обработки события `Error` для компонента `Adodc`, а также процедуру обработки события `Initialize` формы (листинг 7.1). Процедура обработки события `Error` обеспечивает вывод информационного сообщения в случае возникновения ошибки при обращении к базе данных, а процедура обработки события `Initialize` устанавливает ширину столбцов таблицы.

Листинг 7.1. Процедуры обработки событий

```
Private Sub Adodc1_Error(ByVal ErrorNumber As Long, _
            Description As String, ByVal Scode As Long, _
            ByVal Source As String, ByVal HelpFile As String, _
            ByVal HelpContext As Long, fCancelDisplay As Boolean)

    Dim msg As String ' сообщение об ошибке

    msg = "Ошибка: " + Str(ErrorNumber) + vbCr + Description
    MsgBox msg, vbCritical, "Контакты"
    fCancelDisplay = True ' не отображать стандартное сообщение об ошибке
End Sub

Private Sub Form_Initialize()
```

```
    Form1.ScaleMode = vbPixels
  ' установить ширину столбцов
    DataGrid1.Columns(0).Width = 270
End Sub
```

Окно программы работы с базой данных "Контакты" приведено на рис. 7.11. Программа позволяет редактировать данные, добавлять и удалять строки. Чтобы изменить содержимое поля, надо переместить курсор в нужную ячейку (сделать это можно с помощью клавиш перемещения курсора или щелкнув в ячейке кнопкой мыши), нажать клавишу <F2> и внести изменения. Чтобы удалить строку, надо щелчком кнопкой мыши в первом столбце выделить строку и нажать клавишу <Del>. Ввести новую информацию можно только в последнюю (пустую) строку таблицы. Следует обратить внимание, что щелчком в заголовке столбца можно выполнить сортировку данных.

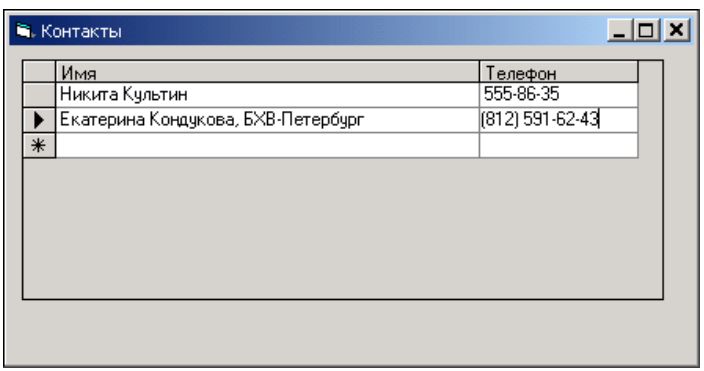


Рис. 7.11. Окно программы работы с базой данных "Контакты"

Контрольные вопросы

1. Какой "поставщик данных" следует выбрать для подключения к базе данных Microsoft Access?
2. В какое свойство компонента `Addc` надо записать SQL-команду, обеспечивающую выбор информации из базы (таблицы) данных?
3. Как связать компонент, обеспечивающий отображение данных в табличной форме (`DataGrid`), с компонентом, обеспечивающим доступ к базе данных (`Adodc`)?
4. По умолчанию в заголовках столбцов компонента `DataGrid` отображаются имена полей записей базы данных. Что надо сделать, чтобы в заголовке столбца отображался пояснительный текст?
5. Как программно установить (изменить) ширину столбца компонента `DataGrid`?

Выбор информации из базы данных

При работе с базой данных пользователя редко интересует все ее содержимое, ему, как правило, нужна некоторая конкретная информация. Найти ее можно последовательным просмотром записей базы. Однако такой способ поиска неудобен и малоэффективен.

Большинство систем управления базами данных позволяют выполнять выборку нужной информации путем выполнения *запросов*. Пользователь формулирует запрос, указывая критерий, которому должна удовлетворять интересующая его информация, а система выводит записи, удовлетворяющие запросу.

Выбрать необходимую информацию из базы данных можно, задав критерий отбора записей в команде `SELECT`.

В общем виде SQL-команда, обеспечивающая выбор информации из базы данных, выглядит так:

SELECT *СписокПолей* **FROM** *Таблица* **WHERE**(*Критерий*) **ORDER BY** *СписокПолей*

Параметр *Таблица* задает таблицу, в которой находится необходимая информация. Параметр *Критерий* задает условие отбора записей из указанной таблицы. Параметр *СписокПолей*, следующий за словом `SELECT`, задает поля, содержимое которых интересует пользователя (если необходимо получить содержимое всех полей, то вместо списка имен полей можно указать "звездочку"). Параметр *СписокПолей*, следующий за `ORDER BY`, задает поля, содержимое которых используется в качестве критерия сортировки записей, полученных из базы данных.

Например, запрос

```
SELECT Author, Title FROM Books WHERE Author = "Н. Культин"
```

обеспечивает выборку из таблицы Books записей, у которых в поле Author находится текст "Н. Культин".

Другой пример. Запрос

```
SELECT Author, Title FROM Books
       WHERE (Author = "Н. Культин") AND (PYEAR = 2009)
```

формирует список книг Н. Культина, вышедших в 2009 году.

В критерии запроса, при сравнении строк, вместо строковой константы можно указать шаблон. Например, шаблон `Культин%` обозначает все строки, которые начинаются словом "Культин", а шаблон `%Культин%` — все строки, в которых есть слово (подстрока) "Культин", например: Н. Культин, Культин Н.,

Культин Н. Б. и т. д. При использовании шаблона вместо оператора "равно" надо использовать оператор LIKE. Например

```
SELECT Author, Title FROM Books
       WHERE (Author LIKE "Культин%") AND (Title LIKE "%Basic%")
```

выводит список книг Культина, посвященных программированию на языке Basic.

Запрос можно сформировать как во время разработки формы, так и во время работы программы.

Следующая программа демонстрирует формирование запроса во время своей работы. Программа позволяет найти в базе данных телефон нужного человека. Форма программы приведена на рис. 7.12.

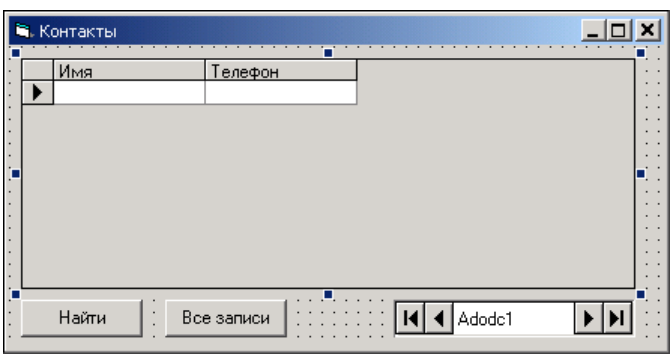


Рис. 7.12. Форма программы "Контакты"

После того как в форму будут добавлены компоненты Adodc и DataGrid, их надо настроить на работу с базой данных "Контакты". Затем надо создать процедуры обработки событий Initialize формы и события Error компонента Adodc (листинг 7.2).

Листинг 7.2. Модуль формы *Контакты*

```
Option Explicit

Public KeyWord As String ' параметр запроса

Private Sub Adodc1_Error(ByVal ErrorNumber As Long, _
            Description As String, ByVal Scode As Long, _
```

```
            ByVal Source As String, ByVal HelpFile As String, _
            ByVal HelpContext As Long, fCancelDisplay As Boolean)

    Dim msg As String ' сообщение об ошибке

    msg = "Ошибка: " + Str(ErrorNumber) + vbCr + Description
    MsgBox msg, vbCritical, "Контакты"
    fCancelDisplay = True ' не отображать стандартное сообщение об ошибке
End Sub

Private Sub Form_Initialize()
    Form1.ScaleMode = vbPixels
   ' установить ширину столбцов
    DataGrid1.Columns(0).Width = 270
End Sub
```

Для ввода параметра запроса — имени человека — в рассматриваемой программе работы с базой данных используется окно **Найти** (рис. 7.13). Это окно становится доступным в результате щелчка на кнопке **Найти**.

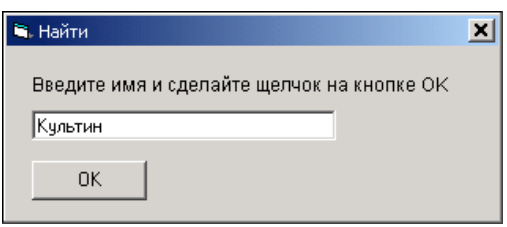


Рис. 7.13. Для ввода параметра запроса используется окно **Найти**

Для того чтобы программа могла вывести окно, отличное от главного, в проект надо добавить форму и выполнить ее настройку. Чтобы добавить в проект форму, надо в меню **Project** выбрать команду **Add Form**, затем (в окне **Add Form**) указать тип формы (в данном случае — **Form**).

После того как форма будет добавлена в проект, надо выполнить ее настройку: задать значения свойств (табл. 7.5) и добавить необходимые компоненты (рис. 7.14).

Таблица 7.5. *Значение свойств формы* ***Найти***

Свойство	Значение	Комментарий
`Name`	`Form2`	
`BorderStyle`	`FixedDialog`	Тонкая граница (нельзя изменить размер окна), нет кнопок **Развернуть** и **Свернуть** в заголовке окна
`StartPosition`	`Center`	Положение окна на экране — по центру относительно главного окна программы

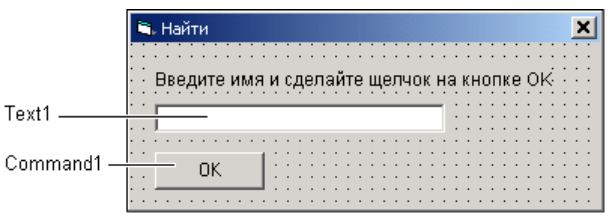


Рис. 7.14. Форма **Найти**

После того как будет создана и настроена форма **Найти** (`Form2`), в модуль главной формы надо добавить процедуры обработки события `Click` для кнопок **Найти** и **Все записи** (листинг 7.3), а в модуль формы `Form2` — процедуру обработки события `Click` на кнопке **OK** (листинг 7.4). Следует обратить внимание, что передача критерия запроса из формы запроса в главную форму осуществляется через глобальную переменную `KeyWord`, которая объявлена в модуле формы `Form1` как `Public`.

Листинг 7.3. Процедуры обработки события `Click` на кнопках *Найти* и *Все записи*

```
' щелчок на кнопке Найти
Private Sub Command1_Click()

    KeyWord = ""
    Form2.Show (vbModal) ' отобразить окно Найти

    If KeyWord <> "" Then
        ' пользователь ввел параметр запроса
        ' сформировать SQL-команду
```

```
        Adodc1.RecordSource = _
                  "SELECT * FROM Contacts WHERE Name LIKE " + _
                  Chr(39) + "%" + KeyWord + "%" + Chr(39) + _
                  "ORDER BY Name"
        Adodc1.Refresh ' выполнить команду
    End If
End Sub


' щелчок на кнопке Все записи
Private Sub Command2_Click()
    Adodc1.RecordSource = _
          "SELECT Name, Phone FROM Contacts ORDER BY Name"
    Adodc1.Refresh ' выполнить запрос
End Sub
```

Листинг 7.4. Процедура обработки события `Click` на кнопке *OK* окна *Найти*

```
Private Sub Command1_Click()
    Form1.KeyWord = Text1.Text ' передать критерий запроса в модуль
                               ' главной формы (Form1)
    Form2.Hide  ' закрыть окно Найти
End Sub
```

Контрольные вопросы

1. Куда надо записать команду, обеспечивающую выбор информации из базы (таблицы) данных?
2. Какой оператор следует использовать в критерии отбора (значение параметра `WHERE`) записей при сравнении строк?
3. Как в критерии отбора указать, что поле, по содержимому которого отбираются записи, должно содержать требуемую подстроку?

Работа с базой данных в режиме формы

Существуют два режима работы с базой данных: режим таблицы и режим формы.

В *режиме таблицы* база данных отображается в виде таблицы, что позволяет видеть несколько записей одновременно.

Если записи базы данных состоят из большого количества полей или информация распределена по нескольким таблицам, то работать с базой в режиме *таблицы* не удобно — размер экрана (ширина области отображения таблицы)

не позволяет видеть всю информацию одновременно (чтобы увидеть содержимое поля, приходится изменять ширину столбца или прокручивать окно по горизонтали). В этом случае удобнее работать с базой данных в режиме формы.

В *режиме формы* на экране отображается только одна запись, что позволяет видеть содержимое всех полей одновременно (рис. 7.15).

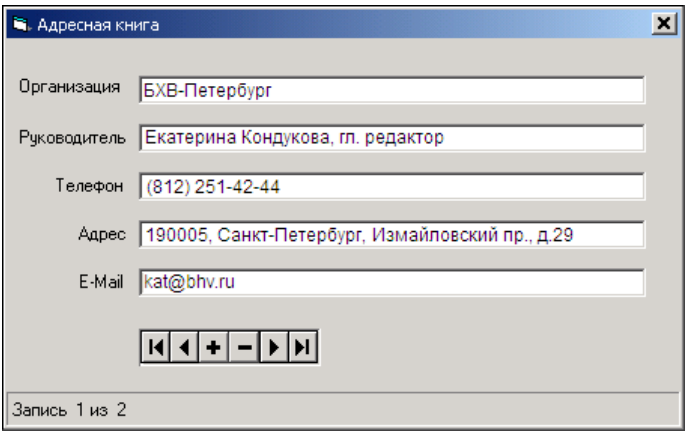


Рис. 7.15. Работа с базой данных в режиме формы

Довольно часто режимы комбинируют. Краткую информацию (содержимое ключевых полей) выводят в виде таблицы, а при необходимости видеть содержимое всех полей выполняют переключение в режим формы. Режим формы также удобен для ввода информации в базу данных.

Работу с базой данных в режимы формы демонстрирует программа "Адресная книга". Программа обеспечивает выполнение основных операций (просмотр, добавление, удаление записей) с базой данных Microsoft Access "Адресная книга" (ArdBk.mdb). База данных "Адресная книга" состоит из одной-единственной таблицы — Contacts (табл. 7.6).

Таблица 7.6. *Поля таблицы Contacts базы данных "Адресная книга"*

Поле	Тип	Размер	Комментарий
`Title`	Строковый	50	Название организации
`Phone`	Строковый	50	Телефон
`Manager`	Строковый	50	Руководитель (контактное лицо)
`Address`	Строковый	50	Адрес
`Email`	Строковый	50	Адрес электронной почты

Форма программы работы с базой данных "Адресная книга" приведена на рис. 7.16. Кнопки, обеспечивающие добавление (`Command1`) и удаление (`Command2`) записей, помещены в поле компонента `Adodc1`. Это графические кнопки `CommandButton`. Рисунок для кнопок создан в редакторе Paint. Компонент `StatusBar` используется для отображения информации о состоянии базы данных, во время работы программы в строке состояния выводится номер отображаемой в данный момент записи. Чтобы компонент был доступен во время разработки формы, надо подключить Microsoft Windows Common Control 6.0 (команда **Project ▸ Components**).

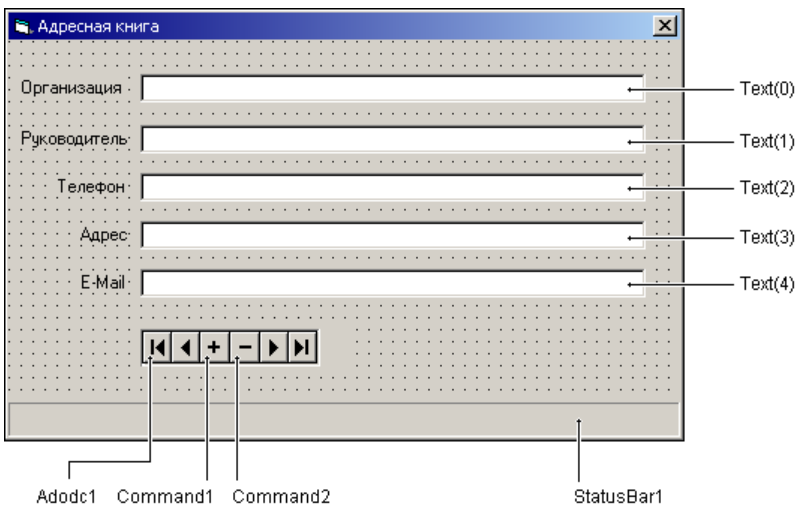


Рис. 7.16. Форма программы "Адресная книга"

Следует обратить внимание, что для отображения содержимого записи (строки таблицы) используются не отдельные компоненты `TextBox`, а массив компонентов. Замена отдельных компонентов массивом компонентов позволила значительно сократить текст программы — десять процедур обработки событий (`KeyPress` и `GotFocus` для каждого из пяти компонентов `TextBox`) заменены двумя, каждая из которых обрабатывает соответствующее событие для всех компонентов массива.

Чтобы создать массив компонентов, надо:

1. Поместить на форму компонент и выполнить его настройку, в том числе изменить имя так, чтобы оно соответствовало имени создаваемого массива компонентов (в рассматриваемом примере имя `Text1` заменено на `Text`).
2. В меню **Edit** выбрать команду **Copy**.

3. В меню **Edit** выбрать команду **Paste**.
4. В окне запроса "You already have a control named ... Do you want to create a control array?" ("Элемент управления с именем ... уже есть. Вы хотите создать массив компонентов?") сделать щелчок на кнопке **Да** (рис. 7.17). В результате в форму будет добавлен компонент.
5. Повторить шаг 3 столько раз, сколько компонентов надо поместить на форму.

Рис. 7.17. Чтобы создать массив компонентов, надо щелкнуть на кнопке **Да**

После того как нужное количество компонентов будет добавлено на форму (сформирован массив компонентов), надо выполнить настройку компонентов и создать процедуры обработки событий. Настройка компонентов выполняется обычным образом.

События, которые возникают для каждого компонента массива компонентов, обрабатываются соответствующей, единой для всех компонентов массива, процедурой. Процесс создания процедуры обработки события для компонентов массива ничем не отличается от процесса создания процедуры обработки события для отдельного компонента. В заголовке процедуру обработки события, созданной для массива компонентов, помимо других параметров есть параметр `Index`. Через этот параметр в процедуру передается номер компонента (элемента массива), на котором произошло событие. В качестве примера в листинге 7.5 приведена процедура обработки события `KeyPress` для массива компонентов `TextBox` формы программы работы с базой данных "Адресная книга". Процедура обрабатывает нажатие клавиши в полях `Text` — при нажатии <Enter> переводит курсор в следующее по порядку поле ввода.

Листинг 7.5. Пример обработки события для массива компонентов

```
' нажатие клавиши в поле редактирования
Private Sub Text_KeyPress(Index As Integer, KeyAscii As Integer)
    If KeyAscii = 13 Then ' пользователь нажал <Enter>
```

```
        ' переместить курсор в следующее поле ввода
        If Index < 4 Then Text(Index + 1).SetFocus
    End If
End Sub
```

Чтобы в поле редактирования (компонента `TextBox`) появилось содержимое поля записи базы данных, надо *связать* поле редактирования и поле записи базы данных: присвоить значения свойствам `DataSource` и `DataField` поля редактирования. В свойство `DataSource` надо записать имя источника данных, а в поле `DataField` — имя поля записи базы данных (рис. 7.18).

Значения свойств компонентов формы программы работы с базой данных "Адресная книга" приведены в табл. 7.7 и 7.8, модуля формы — в листинге 7.6.

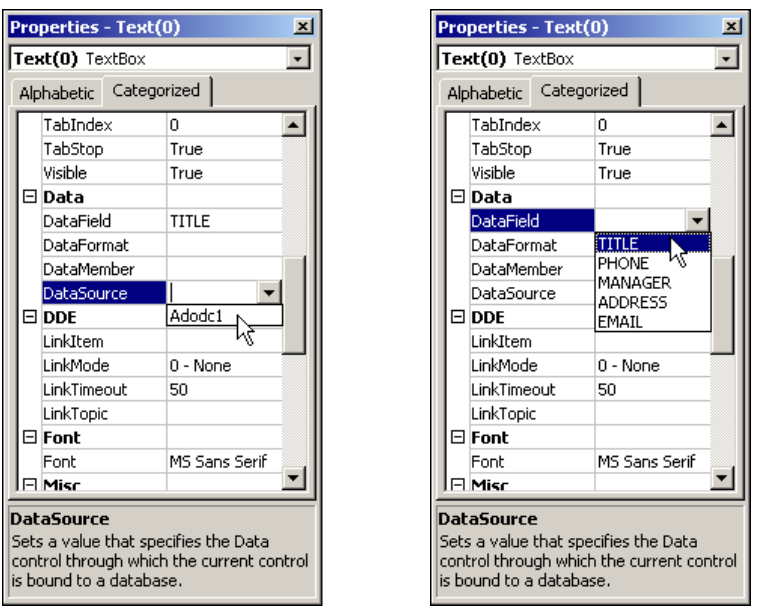


Рис. 7.18. Связывание поля редактирования и поля записи базы данных

Таблица 7.7. *Значения свойств компонента* `Adodc1`

Свойство	Значение
`ConnectionString`	`Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\Database\ADRBK.MDB;Persist Security Info=False`
`RecordSource`	`SELECT * FROM CONTACTS ORDER BY TITLE`

Таблица 7.8. *Значения свойств компонентов* `Text(0)` — `Text(4)`

Свойство	Значение
Text(0).DataSource	Adodc1
Text(0).DataField	TITLE
Text(1).DataSource	Adodc1
Text(1).DataField	MANAGER
Text(2).DataSource	Adodc1
Text(2).DataField	PHONE
Text(3).DataSource	Adodc1
Text(3).DataField	ADDRESS
Text(4).DataSource	Adodc1
Text(4).DataField	EMAIL

Листинг 7.6. Модуль формы программы работы с базой данных "Адресная книга"

```
Option Explicit

' начало работы программы
Private Sub Form_Load()
    ' ограничение количества символов, которое можно
    ' ввести в поле редактирования
    Text(0).MaxLength = 50
    Text(1).MaxLength = 50
    Text(2).MaxLength = 50
    Text(3).MaxLength = 50
    Text(4).MaxLength = 50

    If Adodc1.Recordset.RecordCount = 0 Then

        ' Если в базе данных нет записей и пользователь
        ' попытался сохранить данные, предварительно не нажав
        ' кнопку Добавить запись, то возникнет ошибка
        ' доступа к записи. Чтобы это не произошло,
        ' предварительно добавляется запись. При этом если
```

```
      ' пользователь все-таки нажмет кнопку Добавить запись, то две
      ' новые записи не появятся, т. к. если ни одно из полей
      ' не заполнено, то запись не добавляется

      Adodc1.Recordset.AddNew
   End If

End Sub

' щелчок на кнопке "+" (Добавить запись)
Private Sub Command1_Click()
   Adodc1.Recordset.AddNew
   Text(0).SetFocus
End Sub

' щелчок на кнопке "-" (Удалить запись)
Private Sub Command3_Click()
   Dim r
   r = MsgBox("Удалить запись?", vbQuestion + vbOKCancel, _
              "База данных Адресная книга")
   If r = vbOK Then

      If Adodc1.Recordset.RecordCount <> 0 Then
         Adodc1.Recordset.Delete

         If Adodc1.Recordset.EOF Then
             Adodc1.Recordset.MoveNext
         Else
             Adodc1.Recordset.MoveLast
         End If
      End If

      If Adodc1.Recordset.RecordCount = 0 Then
          Adodc1.Recordset.AddNew
      End If
   End If
End Sub
```

```
' курсор перешел в поле редактирования
Private Sub Text_GotFocus(Index As Integer)
    Text(Index).Text = RTrim(Text(Index).Text)
    ' выделить текст, который находится в поле редактирования
    Text(Index).SelStart = 0
    Text(Index).SelLength = Len(Text(Index).Text)
End Sub

' нажатие клавиши в поле редактирования
Private Sub Text_KeyPress(Index As Integer, KeyAscii As Integer)
    If KeyAscii = 13 Then ' пользователь нажал <Enter>
        ' курсор в следующее поле ввода
        If Index < 4 Then Text(Index + 1).SetFocus
    End If
End Sub

' индикация номера текущей записи
Private Sub Adodc1_MoveComplete _
    (ByVal adReason As ADODB.EventReasonEnum, _
     ByVal pError As ADODB.Error, adStatus As ADODB.EventStatusEnum, _
     ByVal pRecordset As ADODB.Recordset)

StatusBar1.SimpleText =
            "Запись " & Str(Adodc1.Recordset.AbsolutePosition) & _
            " из " & Str(Adodc1.Recordset.RecordCount)
End Sub

' завершение работы с программой
Private Sub Form_Unload(Cancel As Integer)
    If Text(0).DataChanged Or Text(1).DataChanged Or _
       Text(2).DataChanged Or Text(3).DataChanged Or _
       Text(4).DataChanged Then
        Adodc1.Recordset.UpdateBatch (adAffectCurrent)
    End If
End Sub
```

Довольно часто режим формы комбинируют с режимом таблицы. На рис. 7.19 приведена форма приложения работы с базой данных "Адресная книга",

в которую добавлен компонент `DataGrid1` (значения свойств компонента приведены в табл. 7.8), обеспечивающий просмотр базы данных в режиме таблицы. Следует обратить внимание, что работа компонента `DataGrid1` и полей редактирования синхронизирована. Кнопка `Command3` используется для управления отображением таблицы. В начале работы программы таблица не отображается (рис. 7.20). Таблица становится доступной (рис. 7.21), если пользователь сделает щелчок на кнопке **Таблица**. Повторный щелчок на этой кнопке скроет таблицу. Процедура обработки события `Click` на кнопке **Таблица** приведена в листинге 7.7.

Таблица 7.9. *Значения свойств компонента* `DataGrid1`

Свойство	**Значение**
`DataSource`	`Adodc1`
`AllowAddNew`	`True`
`AllowDelete`	`True`
`AllowUpdate`	`True`

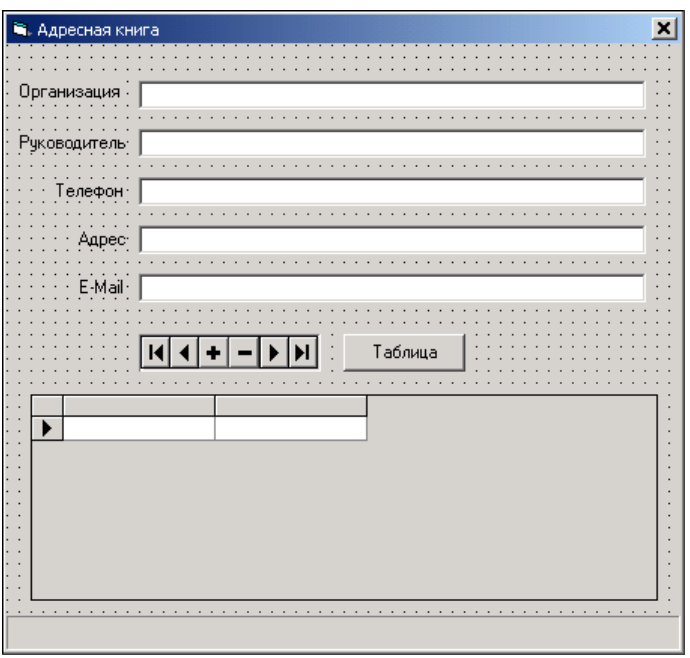


Рис. 7.19. Форма программы "Адресная книга-2"

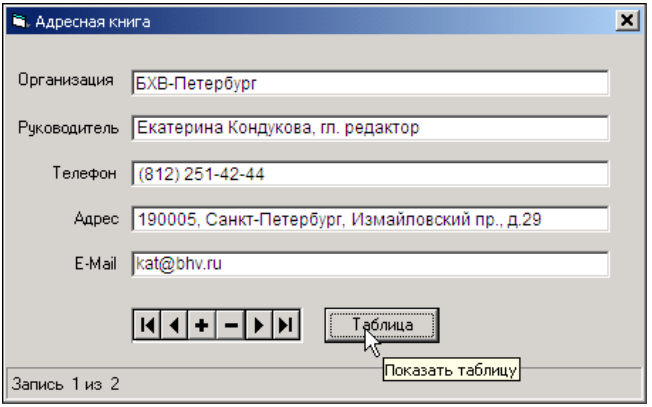


Рис. 7.20. Работа с базой данных в режиме формы

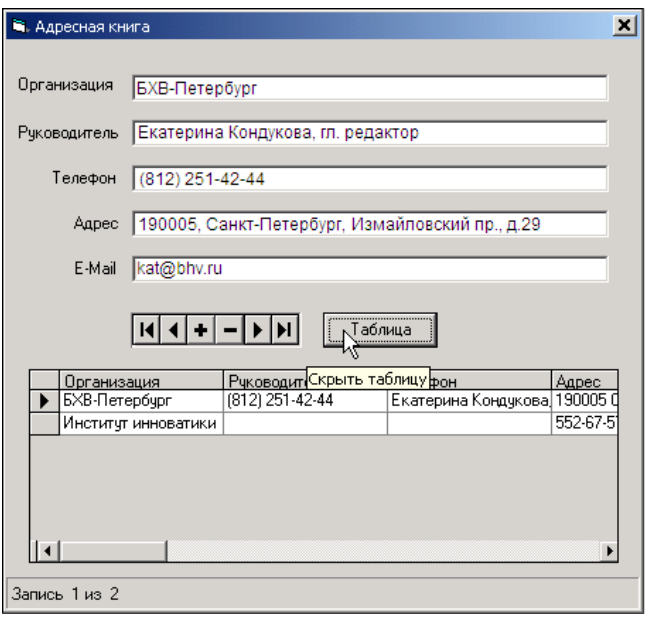


Рис. 7.21. Щелчок на кнопке **Таблица** скроет таблицу

Листинг 7.7. Обработка события `Click` на кнопке *Таблица*

```
' щелчок на кнопке Таблица
Private Sub Command3_Click()
    ' изменить высоту окна
```

```
    If DataGrid1.Visible Then
       ' скрыть таблицу
       DataGrid1.Visible = False
       Form1.Height = Form1.Height - _
                  Form1.ScaleY(DataGrid1.Height + 8, vbPixels, vbTwips)
       Command3.ToolTipText = "Показать таблицу"
    Else
       ' показать таблицу
       DataGrid1.Visible = True
       Form1.Height = Form1.Height + _
                  Form1.ScaleY(DataGrid1.Height + 8, vbPixels, vbTwips)
       Command3.ToolTipText = "Скрыть таблицу"
    End If
End Sub
```

Для того чтобы в начале работы программы таблица не отображалась, в процедуру обработки события `Load` формы добавлены инструкции, которые устанавливают требуемый размер окна (листинг 7.8). Эта же процедура задает заголовки таблицы.

Листинг 7.8. Процедура обработки события `Load`

```
' начало работы программы
Private Sub Form_Load()
    ' ограничение количества символов, которое можно
    ' ввести в поле редактирования
   Text(0).MaxLength = 50
   Text(1).MaxLength = 50
   Text(2).MaxLength = 50
   Text(3).MaxLength = 50
   Text(4).MaxLength = 50

   If Adodc1.Recordset.RecordCount = 0 Then

      ' Если в базе данных нет записей и пользователь
      ' попытался сохранить данные, предварительно не нажав
      ' кнопку Добавить запись, то в программе возникнет
      ' ошибка доступа к записи. Чтобы это не произошло,
      ' предварительно добавляется запись. При этом если
      ' пользователь все-таки нажмет кнопку Добавить запись, то две
```

```
    ' новые записи не появятся, т. к. если ни одно из полей
    ' не заполнено, то запись не добавляется

    Adodc1.Recordset.AddNew
  End If

  Command1.ToolTipText = "Добавить запись"
  Command2.ToolTipText = "Удалить запись"

  ' настройка компонента DataGrid
  DataGrid1.Columns(0).Caption = "Организация"
  DataGrid1.Columns(1).Caption = "Руководитель"
  DataGrid1.Columns(2).Caption = "Телефон"
  DataGrid1.Columns(3).Caption = "Адрес"
  DataGrid1.Columns(4).Caption = "E-Mail"

  ' скрыть таблицу (компонент DataGrid)
  DataGrid1.Visible = False
  Form1.Height = Form1.Height - _
        Form1.ScaleY(DataGrid1.Height + 8, vbPixels, vbTwips)
  Command3.ToolTipText = "Показать таблицу"

End Sub
```

Контрольные вопросы

1. Какой компонент следует использовать для вывода содержимого поля записи базы данных при отображении записи в виде формы?
2. Значения каких свойств компонента `TextBox` надо установить, чтобы связать его с компонентом, обеспечивающим доступ к данным?
3. Значение какого свойства компонента `TextBox` надо установить, чтобы задать поле записи, содержимое которого будет отображаться в поле компонента?

Создание базы данных

При разработке программ работы с базами данных "Контакты" и "Адресная книга" предполагалось, что эти базы данных (файлы contacts.mdb и AdrBk.mdb соответственно) существуют. Теперь рассмотрим, как можно создать базу данных без использования СУБД, в частности базу данных в формате Microsoft Access без Microsoft Access.

Создание файла базы данных

Создание файла базы данных обеспечивает метод `Create` объекта `Catalog`. Объект `Catalog` является объектом ADO, который находится в библиотеке Microsoft ADO Ext. 2.5 for DLL and Security (ADOX). Поэтому, для того чтобы программа могла создать базу данных, в проект надо добавить ссылку на библиотеку ADOX (рис. 7.22).

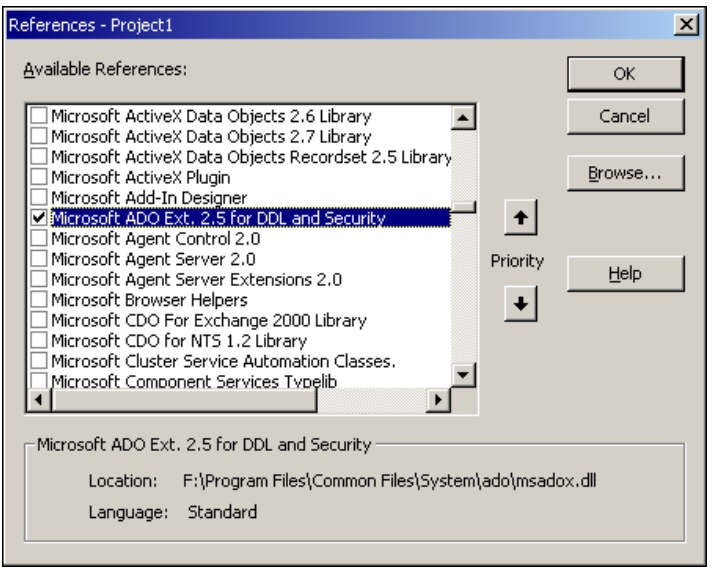


Рис. 7.22. Для доступа к объекту `Catalog`, обеспечивающему создание БД, в проект надо добавить ссылку на библиотеку ADOX

Чтобы создать файл базы данных, надо вызвать метод `Create`, указав в качестве параметра строку соединения. Например, в результате выполнения инструкции

```
aCatalog.Create("Provider=Microsoft.Jet.OLEDB.3.51; _
                Data Source=D:\Database\AdrBk.mdb")
```

в каталоге D:\Database будет создан файл AdrBk.mdb — база данных формата Microsoft Access. Строго говоря, будет создан файл базы данных Microsoft Jet, формат которой использует СУБД Microsoft Access. Следует обратить внимание, что метод `Create` не только создает файл базы данных, но и открывает соединение, доступ к которому можно получить через свойство `ActiveConnection`.

Создание таблицы

После того как будет создана база данных, в ней можно создать таблицу.

Чтобы в базе данных создать таблицу, надо направить серверу SQL-команду CREATE TABLE, которая в общем виде выглядит так:

```
CREATE TABLE Таблица (Поле1 Тип1, Поле2 Тип2, ..., Полеk Типk)
```

Здесь: *Таблица* — имя таблицы, которая будет создана в результате выполнения SQL-команды CREATE; $Поле_i$ — имя *i*-го столбца таблицы, $Тип_i$ — тип *i*-го столбца таблицы. Например, в результате выполнения команды

```
CREATE TABLE Contacts (Title CHAR(50), Phone CHAR(50), Manager CHAR(50),
                       Address CHAR(50), Email CHAR(50))
```

в базе данных, которой адресован запрос, будет создана таблица Contacts. Следует обратить внимание, что таблица будет создана только в том случае, если в базе данных таблицы с указанным именем нет.

Параметр, указанный после имени поля, задает тип данных. Помимо типа CHAR (символьный) можно задать NUMBER (числовой), CURRENCY (денежный), DATE (дата) или TIME (время).

Если поле является обязательным (т. е. обязательно должно содержать информацию), то после идентификатора типа поля надо указать параметр NOT NULL. Например, очевидно, что в таблице Contacts поле Title обязательно должно быть заполнено. Поэтому команду, обеспечивающую создание таблицы, следует переписать так:

```
CREATE TABLE Contacts (Title CHAR(50) NOT NULL, Phone CHAR(50),
                       Manager CHAR(50), Address CHAR(50), Email CHAR(50))
```

Добавление информации

Чтобы добавить в таблицу информацию (запись), надо направить базе данных команду INSERT INTO, указав имя таблицы, имена и значения полей. Например, команда:

```
INSERT INTO Contacts (Title, Phone)
            VALUES ('Никита Культин', '555-86-35')
```

добавляет в таблицу Contacts новую запись.

Следует обратить внимание на то, что если при создании таблицы поле было объявлено как обязательное, его значение обязательно должно быть указано. Если значение обязательного поля не указано, то запись в таблицу добавлена не будет.

Удаление таблицы

Иногда возникает необходимость удалить из базы данных таблицу. Сделать это можно, направив к базе данных SQL-команду `DROP TABLE`. Например, команда

```
DROP TABLE Expenses
```

удаляет из базы данных таблицу Expenses.

Пример программы

Как можно создать базу данных, не прибегая к помощи СУБД, показывает программа "Создать базу данных". Форма программы приведена на рис. 7.23, текст — в листинге 7.9. Следует обратить внимание, что для доступа к объектам ADODB (`Connection` и `Command`) и ADOX (`Catalog`) в проект надо поместить ссылки на библиотеки Microsoft ActiveX Data Objects 2.1 (ADODB) и Microsoft ADO Ext. 2.5 for DLL and Security (ADOX).

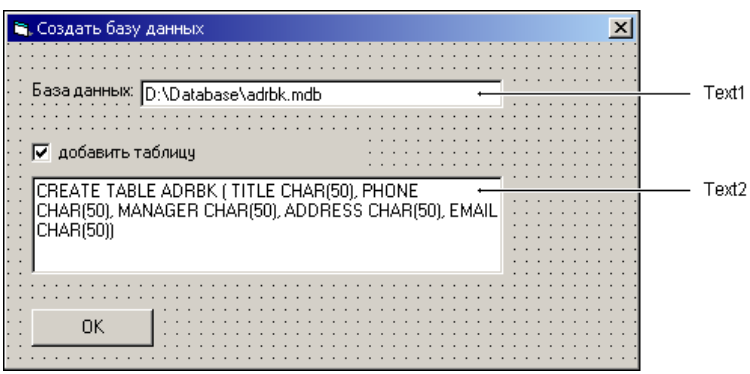


Рис. 7.23. Форма программы "Создать базу данных"

Листинг 7.9. Создание базы данных

```
Option Explicit

' для доступа к объектам ADODB и ADOX в проекте должны быть
' ссылки на:
'  - Microsoft ActiveX Data Objects 2.1 Library (ADODB)
'  - Microsoft ADO Ext. 2.5 for DLL and Security (ADOX)
```

```
Private Sub Command1_Click()

    Dim ConnStr As String '  строка соединения

    Dim aCatalog As New ADOX.Catalog

    Dim conn As New ADODB.Connection
    Dim cmd As New ADODB.Command

    ConnStr = "Provider=Microsoft.Jet.OLEDB.3.51;Data Source=" + _
              Text1.Text

    ' создать базу данных (MDB-файл)
    On Error GoTo er1
    aCatalog.Create (ConnStr)
    MsgBox "База данных " + Text1.Text + " создана.", _
            vbInformation + vbOKOnly, "Создать базу данных"

    Set conn = aCatalog.ActiveConnection

    If Check1.Value = Checked Then
       ' добавить в созданную базу данных таблицу
       ' соединение уже открыто

       Set cmd.ActiveConnection = conn

       cmd.CommandText = Text2.Text
       On Error GoTo er2
       cmd.Execute ' выполнить команду

       MsgBox "SQL-команда: " + vbCr + Text2.Text + "выполнена.", _
               vbInformation, "Выполнить SQL-команду"

       ' закрыть соединение
       conn.Close
    End If

    Exit Sub
```

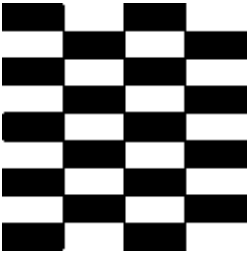
```
er1: ' ошибка создания базы данных
    MsgBox "Ошибка создания базы данных." + vbCr + Error, _
           vbExclamation + vbOKOnly, "Ошибка"
    Exit Sub


er2: ' ошибка выполнения SQL-команды
    MsgBox "Ошибка выполнения SQL-команды." + vbCr + Error, _
           vbExclamation + vbOKOnly, "Ошибка"
    conn.Close
    Exit Sub

End Sub

Private Sub Check1_Click()
    If Check1.Value = Checked Then
        Text2.Enabled = True
        Text2.SetFocus
    Else
        Text2.Enabled = False
    End If
End Sub

Private Sub Text1_GotFocus()
    Text1.SelStart = 0
    Text1.SelLength = Len(Text1.Text)
End Sub

Private Sub Text2_GotFocus()
    Text2.SelStart = 0
    Text2.SelLength = Len(Text2.Text) - 1
End Sub
```

Контрольный вопрос

Что надо сделать, чтобы в программе стали доступны объекты `Connection`, `Command` и `Catalog`, позволяющие программно (не используя СУБД Microsoft Access) создать базу данных формата Microsoft Access?

Установка программы работы с базой данных на другой компьютер

Часто возникает необходимость перенести базу данных на другой компьютер. При установке программы работы с базой данных на компьютер пользователя следует учитывать, что помимо самого приложения и базы данных (файла данных) на компьютер пользователя надо установить библиотеки, обеспечивающие доступ к базе данных. Например, для работы с БД Microsoft Access необходимы компоненты: Microsoft ADO Data Control 6.0 (файл MSADODC.OCX) и Microsoft DataGrid Control 6.0 (файл MSDATGRD.OCX).



Глава 8

Примеры программ

Экзаменатор

Тестирование широко применяется для оценки уровня знаний в учебных заведениях, при приеме на работу, для оценки квалификации персонала учреждений, т. е. практически во всех сферах деятельности человека. Испытуемому предлагается ряд вопросов (тест), на которые он должен ответить. Обычно к каждому вопросу дается несколько вариантов ответа, из которых надо выбрать правильный. После того как испытуемый ответит на все вопросы, подсчитывается количество правильных ответов и на основе этой информации выставляется оценка.

Рассмотрим программу "Экзаменатор", которая позволяет автоматизировать процесс тестирования.

Требования к программе

В результате анализа используемых на практике методик тестирования были сформулированы следующие требования к программе тестирования:

- ❒ программа должна работать с тестом произвольной длины, т. е. не должно быть ограничения на количество вопросов в тесте;
- ❒ для каждого вопроса может быть до четырех возможных вариантов ответа;
- ❒ вопрос может сопровождаться иллюстрацией;
- ❒ результат тестирования должен быть отнесен к одному из четырех уровней, например: "отлично", "хорошо", "удовлетворительно" или "неудовлетворительно";
- ❒ вопросы теста должны находиться в текстовом файле;
- ❒ в программе должна быть заблокирована возможность возврата к предыдущему вопросу. Если вопрос предложен, то на него должен быть дан ответ.

На рис. 8.1 приведен пример окна программы тестирования во время ее работы.

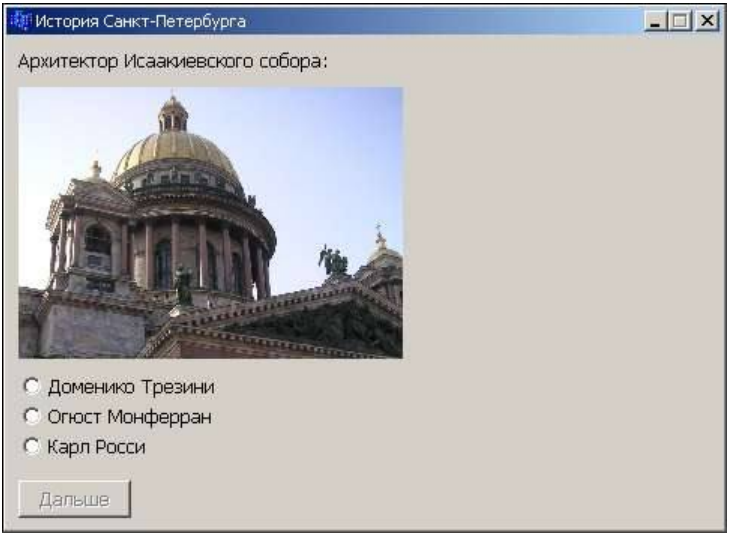


Рис. 8.1. Окно программы "Экзаменатор": испытуемый должен выбрать правильный ответ

Файл теста

Тест представляет собой последовательность вопросов, на которые испытуемый должен ответить путем выбора правильного ответа из нескольких предложенных вариантов.

Файл теста состоит из:

- заголовка;
- раздела оценок;
- раздела вопросов.

Заголовок содержит название теста и общую информацию о тесте, например о его назначении. Состоит заголовок из двух абзацев (строк): первый абзац — название теста, второй — вводная информация.

Вот пример заголовка:

```
История Санкт-Петербурга
Сейчас Вам будут предложены вопросы о знаменитых памятниках и архитектурных сооружениях Санкт-Петербурга. Вы должны из нескольких предложенных вариантов ответа выбрать правильный.
```

Здесь следует обратить внимание, что абзац — это последовательность символов, заканчивающаяся символом "конец абзаца", который добавляется

в текст в результате нажатия клавиши <Enter>. В окне редактора текст абзаца может выглядеть как несколько строк текста. Тем не менее, при чтении текста из файла инструкция `Line Input` считывает абзац.

За заголовком следует раздел оценок, в котором указывается количество баллов, необходимое для достижения уровня, и сообщение, информирующее испытуемого о достижении уровня. В простейшем случае сообщение — это оценка. Для каждого уровня надо указать балл (количество правильных ответов) и в следующей строке сообщение. Вот пример раздела оценок:

```
10
Отлично
8
Хорошо
6
Удовлетворительно
5
Неудовлетворительно
```

За разделом оценок следует раздел вопросов теста.

Каждый вопрос начинается текстом вопроса, за которым (в следующей строке) находятся три целых числа. Перовое число — это количество альтернативных ответов, второе — номер правильного ответа, третье — признак наличия к вопросу иллюстрации. Если вопрос сопровождается иллюстрацией, то значение признака должно быть равно единице, если нет, то нулю. Если к вопросу есть иллюстрация, то в следующей строке должно быть имя файла иллюстрации. Далее следуют альтернативные ответы, каждый из которых должен представлять собой один абзац текста.

Вот пример вопроса:

```
Архитектор Зимнего дворца
3 2 1
herm.jpg
Карл Росси
Бартоломео Растрелли
Огюст Монферран
```

В приведенном примере к вопросу даны три варианта ответа, правильным является второй ответ (архитектор Зимнего дворца — Растрелли). К вопросу есть иллюстрация (третье число во второй строке — единица), которая находится в файле herm.jpg.

Ниже в качестве примера приведен текст файла вопросов для контроля знания истории памятников и архитектурных сооружений Санкт-Петербурга.

```
История Санкт-Петербурга
Сейчас Вам будут предложены вопросы о знаменитых памятниках и архитектурных сооружениях Санкт-Петербурга. Вы должны из нескольких предложенных вариантов ответа выбрать правильный.
7
Вы прекрасно знаете историю Санкт-Петербурга!
6
Вы много знаете о Санкт-Петербурге, но на некоторые вопросы ответили не верно.
5
Вы не достаточно хорошо знаете историю Санкт-Петербурга.
4
Вы, вероятно, только начали знакомиться с историей Санкт-Петербурга?
Архитектор Исаакиевского собора:
3 2 1
isaak.jpg
Доменико Трезини
Огюст Монферран
Карл Росси
Александровская колонна воздвигнута в 1834 году по проекту Огюста Монферрана как памятник, посвященный:
2 1 0
деяниям императора Александра I
подвигу народа в Отечественной войне 1812 года
Архитектор Зимнего дворца
3 2 1
herm.jpg
Карл Росси
Бартоломео Растрелли
Огюст Монферран
Михайловский (Инженерный) замок - жемчужина архитектуры Петербурга построен по проекту
3 3 0
Воронихина Андрея Никифоровича
Старова Ивана Егоровича
Баженова Василия Ивановича
Остров, на котором находится Ботанический сад, основанный императором Петром I, называется:
3 3 1
bot.jpg
```

```
Заячий
Медицинский
Аптекарский
Невский проспект получил свое название
3 2 0
по имени реки, на берегах которой стоит Санкт-Петербург
по имени близко расположенного монастыря, Александро-Невской Лавры
в память о знаменитом полководце Александре Невском
Медный всадник - скульптура знаменитого памятника Петру I выполнена
2 1 0
Фальконе
Клодтом
```

Файл теста можно подготовить в Блокноте или в Microsoft Word. В случае использования Microsoft Word при сохранении текста следует указать, что надо сохранить только текст. Для этого в окне **Сохранить как** в списке **Тип файла** следует выбрать **Обычный текст (*.txt)**.

Форма приложения

Форма программы "Экзаменатор" приведена на рис. 8.2.

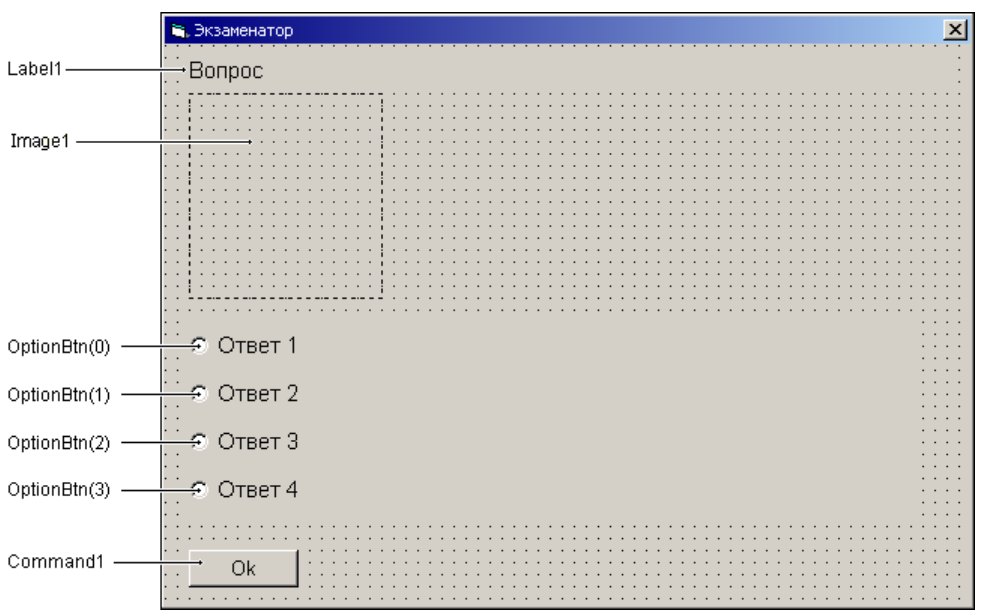


Рис. 8.2. Форма программы "Экзаменатор"

Поле Label1 предназначено для вывода начальной информации, вопроса и результатов тестирования. Компонент Image1 служит для отображения иллюстрации, сопровождающей вопрос. Объединенные в массив компоненты OptionBtn типа Option предназначены для отображения альтернативных ответов и приема ответа испытуемого.

В табл. 8.1 приведены значения свойств формы.

Таблица 8.1. *Значения свойств формы*

Свойство	Значение	Пояснение
BorderStyle	1 — FixedSingle	Тонкая граница. Изменить размер окна путем перетаскивания границы
ScaleMode	1 — Pixel	Единица измерения координат и размеров компонентов — пиксел
StartUpPosition	2 — CenterScreen	В начале работы программы окно разместить в центре экрана

Отображение иллюстрации

Для отображения иллюстраций используется компонент Image1, размер и положение которого проще задать во время разработки формы. В рассматриваемой программе применяется другой подход — положение и размер компонента Image1 задается (вычисляется) во время работы программы.

Очевидно, что размер области формы, которая может быть использована для вывода иллюстрации, зависит от размера поля, предназначенного для отображения вопроса, и количества альтернативных ответов. Чем длиннее вопрос (больше размер поля отображения вопроса) и чем больше альтернативных ответов дано к вопросу (минимальное количество — два, максимальное — четыре), тем меньше места остается для иллюстрации. После того как очередной вопрос прочитан, известно, сколько места необходимо для отображения вопроса, сколько места займут поля отображения альтернативных ответов и, следовательно, сколько места можно выделить для отображения иллюстрации. Если размер иллюстрации превышает размер области, выделенной для ее отображения, то выполняется масштабирование.

Доступ к файлу теста

Имя файла теста можно задать в тексте программы. Очевидно, что такой подход неприемлем, если создается универсальная программа тестирования.

Другой способ задать файл теста — указать имя файла в качестве параметра команды запуска программы (в командной строке). Например, при запуске программы из операционной системы с помощью команды **Пуск ▸ Выполнить** параметры командной строки указывают после имени выполняемого файла программы (рис. 8.3).

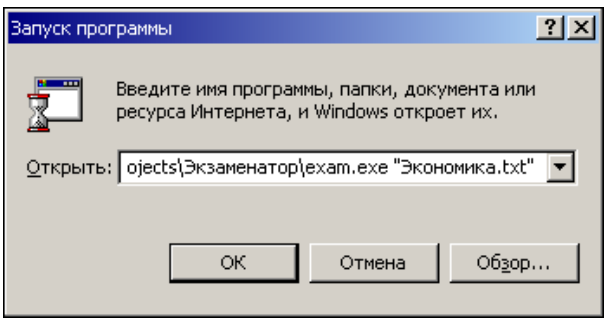


Рис. 8.3. Имя файла теста можно указать в качестве параметра команды запуска программы

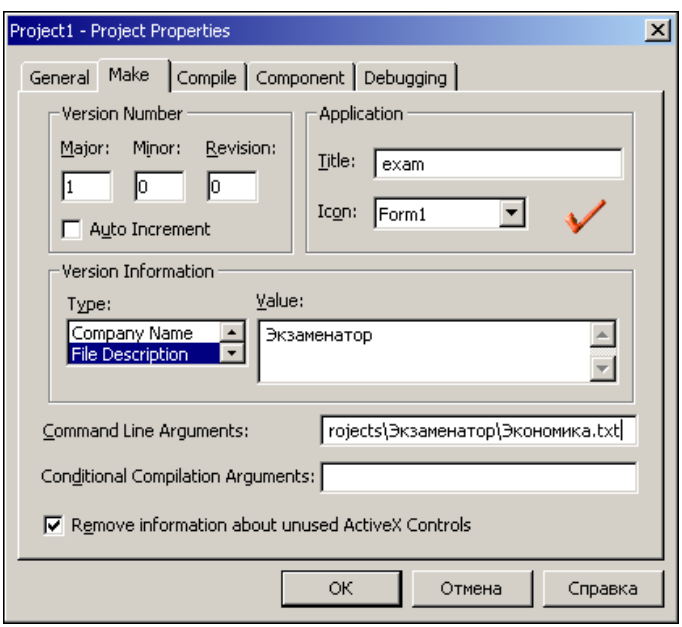


Рис. 8.4. Параметры командной строки надо ввести в поле **Command Line Arguments**

Доступ к параметрам командной строки обеспечивает функция `Command`. Например, фрагмент кода, обеспечивающий прием параметра из командной строки, программы "Экзаменатор" выглядит так:

```
fTest = Command()
   If fTest = "" Then
      ' не задан параметр командной строки
      Label1.Caption = "В командной строке надо указать файл теста." & _
                       vbCr & "Например: exam.exe d:\exam\Экономика.txt"
      Command1.Tag = 2
      Exit Sub
   End If
```

При запуске программы, использующей параметры командной строки, из Visual Basic параметры нужно ввести в поле **Command Line Arguments** вкладки **Make** окна **Project Properties** (рис. 8.4), которое открывается в результате выбора в меню **Project** команды **Properties**.

Текст программы

Текст программы "Экзаменатор" приведен в листинге 8.1.

Листинг 8.1. Экзаменатор

```
Option Explicit

' количество уровней оценки — 4
' возможное количество вариантов ответа — 4

Dim fTest As String  ' файл теста
Dim fn As Integer    ' идентификатор файла
Dim title As String  ' название теста

Dim vopros As Integer     ' номер текущего вопроса
Dim otv As Integer        ' номер выбранного ответа
Dim right As Integer      ' номер правильного ответа
Dim nright As Integer     ' количество правильных ответов
Dim level(1 To 4) As Integer ' кол-во правильных ответов, необходимое
                             ' для достижения уровня
Dim mes(1 To 4) As String    ' уровень (оценка и сообщение)
```

```
' инициализация формы
Private Sub Form_Initialize()

    ' имя файла теста считывается из командной строки
    fTest = Command()
    If fTest = "" Then
       ' не задан параметр командной строки
       Label1.Caption = "В командной строке надо указать файл теста." & _
                        vbCr & "Например: exam.exe d:\exam\Экономика.txt"
       Command1.Tag = 2
       Exit Sub
    End If

    ' параметр командной строки указан
    On Error GoTo e1

    Call InitForm

    fn = FreeFile ' запросить у системы доступный идентификатор файла
    Open fTest For Input As #fn' открыть файл теста для чтения

    Call info          ' вывод информации о тесте
    Call getLevel      ' чтение информации об оценках

    Form1.ScaleMode = vbPixels
    Command1.Tag = 0

    Label1.WordWrap = True
    Label1.AutoSize = True
    Exit Sub

e1:
     ' ошибка доступа к файлу теста
     Label1.WordWrap = True
     Label1.AutoSize = True
     Label1.Caption = "Ошибка доступа к файлу теста " & fTest
     Command1.Tag = 2

End Sub
```

```
' вывод информации о тесте
Sub info()
   Dim buf As String

   Line Input #fn, buf       ' чтение название теста
   Form1.Caption = buf       ' вывод названия теста
   title = buf

   Line Input #fn, buf       ' чтение информации о тесте
   Label1.Caption = buf      ' вывод информации о тесте

End Sub

' чтение информации об уровнях оценки
Sub getLevel()
   Dim buf As String
   Dim i As Integer

   i = 1
   For i = 1 To 4
      Line Input #fn, buf ' количество баллов
      level(i) = buf
      Line Input #fn, buf ' оценка
      mes(i) = buf
   Next i

End Sub

' щелчок на кнопке Ok/Дальше
Private Sub Command1_Click()
   Select Case Command1.Tag
      ' вывод первого вопроса
      Case 0:
         Command1.Enabled = False

         Call InitForm
         Call voprosToScr
```

```
         Command1.Tag = 1
         Command1.Caption = "Дальше"

      ' вывод остальных вопросов
      Case 1:
         If otv = right Then nright = nright + 1

         Command1.Enabled = False

         Call InitForm

         If Not EOF(fn) Then
            Call voprosToScr
         Else
            Close #fn
            Command1.Caption = "Ok"
            Form1.Caption = title
            Command1.Tag = 2
            Command1.Enabled = True
            Call itog   ' вывести результат
         End If

      ' завершение работы
      Case 2:
         Unload Me
   End Select
End Sub

' очистка формы перед выводом очередного вопроса
Sub InitForm()
   Dim i As Integer

   ' "сбросить" переключатели выбора ответа
   For i = 0 To 3
        OptionBtn(i).Visible = False
        OptionBtn(i).Caption = ""
        OptionBtn(i).Value = False
   Next i
```

```
    Image1.Visible = False
End Sub

' вывод вопроса
Sub voprosToScr()
    Dim n As Integer  ' кол-во вариантов ответа
    ' Dim right As Integer  ' номер правильного ответа
    Dim p As Integer  ' признак наличия иллюстрации

    Dim fPicture As String    ' файл иллюстрации
    Dim buf As String         ' буфер чтения

    Dim i As Integer

    vopros = vopros + 1
    Form1.Caption = " Вопрос " + Str(vopros)

    Line Input #fn, buf  ' чтение вопроса
    Label1.Caption = buf ' отображение вопроса

    ' следующая строка имеет вид: N R P,
    ' где: N — количество альтернативных ответов;
    '      R — номер правильного ответа;
    '      P — признак наличия иллюстрации (1 — есть, 0 — нет)

    Line Input #fn, buf ' чтение информации об ответах

    n = Val(Substr(buf, 1))
    right = Val(Substr(buf, 2))  ' right — глобальная переменная
    p = Val(Substr(buf, 3))

    If p = 1 Then
        ' прочитать имя файла иллюстрации
        Line Input #fn, fPicture
        Image1.Tag = 1
        On Error Resume Next
        Image1.Picture = LoadPicture(fPicture)
```

```
      If Err Then
          ' ошибка чтения файла иллюстрации
          ' (файл иллюстрации не найден)
          Image1.Tag = 0
      End If
  Else
      Image1.Tag = 0
  End If

  ' считывание вариантов ответа
   For i = 1 To n
      Line Input #fn, buf
      Select Case i
          Case 1: OptionBtn(0).Caption = buf
          Case 2: OptionBtn(1).Caption = buf
          Case 3: OptionBtn(2).Caption = buf
          Case 4: OptionBtn(3).Caption = buf
     End Select
  Next i

  ' здесь прочитана иллюстрация и альтернативные ответы

  ' вопрос выведен, иллюстрация и альтернативные ответы — нет
  If Image1.Tag = 1 Then    ' есть иллюстрация к вопросу
     Call showPicture
  End If

  ' вывод альтернативных ответов
  ' первый альтернативный ответ
  If OptionBtn(0).Caption <> "" Then
     If Image1.Tag = 1 Then
           OptionBtn(0).Top = Image1.Top + Image1.Height + 5
     Else
           OptionBtn(0).Top = Label1.Top + Label1.Height + 5
     End If
     OptionBtn(0).Visible = True
     OptionBtn(0).Value = False
```

```
    End If

    ' остальные альтернативные ответы
    For i = 1 To 3
     If OptionBtn(i).Caption <> "" Then
       OptionBtn(i).Top = OptionBtn(i - 1).Top + OptionBtn(i - 1).Height
       OptionBtn(i).Visible = True
       OptionBtn(i).Value = False
     End If
    Next i

    Command1.Enabled = False
End Sub

' щелчок на переключателе выбора ответа
Private Sub OptionBtn_Click(Index As Integer)
    otv = Index + 1 ' кнопки пронумерованы с нуля
    Command1.Enabled = True
End Sub

' определение достигнутого уровня
Sub itog()
    Dim i As Integer
    Dim buf As String

    buf = "Результат тестирования" + vbCr + vbCr + _
          "Всего вопросов: " + Str(vopros) + vbCr + _
          "Правильных ответов: " + Str(nright) + vbCr

    i = 0
    ' nright — кол-во правильных ответов
    ' сначала сравниваем с количеством баллов "на пять",
    ' затем — "на четыре" и т. д.
    While (nright <= level(i)) And (i < 3)
       i = i + 1
    Wend

    buf = buf + mes(i)
```

```
   Label1.Caption = buf
End Sub

' вывод иллюстрации
Sub showPicture()
   Dim w As Integer, h As Integer    ' размер области вывода иллюстрации

   Dim k As Single ' коэффициент масштабирования
   Dim i As Integer

   Image1.Stretch = False
   Image1.Top = Label1.Top + Label1.Height + 7

   ' определить размер области вывода иллюстрации
   w = Form1.ScaleWidth - Label1.Left * 2
   h = Command1.Top - (Label1.Top + Label1.Height) - 10 * 2

   ' размер области вывода иллюстрации зависит от количества
   ' альтернативных ответов — чем меньше
   ' вариантов ответа, тем больше область
   For i = 0 To 3
        If OptionBtn(i).Caption <> "" Then h = h - OptionBtn(i).Height
   Next i

   ' если картинка меньше WxH, то она не масштабируется

   ' масштабирование по высоте
   If (Image1.Height > h) Then
      k = Image1.Width / Image1.Height
      Image1.Stretch = True
      Image1.Width = h * k
      Image1.Height = h
   End If

   ' масштабирование по ширине
   If (Image1.Width > w) Then
      Image1.Stretch = True
      Image1.Width = w
```

```
        Image1.Height = w / k
    End If

    Image1.Visible = True
End Sub

' Возвращает подстроку с указанным номером
' (строка разделена на подстроки пробелами)
Function Substr(st As String, n As Integer) As String
    Dim s As Integer ' указатель на первый символ подстроки
    Dim f As Integer ' указатель на последний символ подстроки
    Dim i As Integer

    s = 1
    For i = 1 To n - 1
        s = InStr(s, st, " ")
        s = s + 1
    Next i

    f = InStr(s + 1, st, " ")
    If f <> 0 Then
        f = f - 1
    Else
        f = Len(st)
    End If

    Substr = Mid(st, s, f - s + 1)

End Function
```

В начале работы программы процедура обработки события `Initialize` проверяет, указан ли в командной строке запуска программы параметр — имя файла теста. Если файл теста задан, то она открывает его, считывает из файла название теста и вводную информацию. Название теста отображается в заголовке окна, вводная информация — в поле компонента `Label1`. Следует обратить внимание, что реализация программы предполагает, что программа запускается из того каталога, в котором находятся файл теста и файлы иллюстраций. Такой подход позволяет сгруппировать все файлы одного теста в одном каталоге.

После того как прочитана общая информация о тесте, программа считывает из файла теста информацию об уровнях оценки и фиксирует ее в массивах `level` и `mes`.

После вывода информационного сообщения программа ждет, пока пользователь не нажмет кнопку **Ok** (`Command1`).

Командная кнопка `Command1` используется для активизации процесса тестирования (после вывода информационного сообщения), перехода к следующему вопросу (после выбора варианта ответа) и завершения работы программы (после вывода результата тестирования или вывода сообщения об ошибке, если в командной строке запуска программы не указан файл теста). Действие, выполняемое в результате нажатия кнопки `Command1`, зависит от значения свойства `Tag` кнопки. В начале работы программы значение свойства `Tag` равно нулю. Поэтому процедура обработки события `Click` выводит первый вопрос, заменяет текст на кнопке на **Дальше** и присваивает свойству `Tag` значение 1. В процессе тестирования значение свойства `Tag` кнопки `Command1` равно единице. Поэтому функция обработки события `Click` сравнивает номер выбранного ответа (`otv`) с правильным (`right`), увеличивает на единицу счетчик правильных ответов (если выбран правильный ответ) и активизирует процесс чтения следующего вопроса. Если попытка чтения очередного вопроса завершилась неудачно (это значит, что вопросы исчерпаны), функция выводит результаты тестирования, заменяет текст на командной кнопке на **Ok** и подготавливает операцию завершения работы программы (свойству `Tag` присваивает значение 2).

Чтение из файла очередного вопроса (вопрос, количество альтернативных ответов, номер правильного ответа, признак наличия иллюстрации, имя файла иллюстрации и альтернативные ответы) и его отображение выполняет функция `voprosToScr`. Сначала функция считывает строку из файла теста (вопрос) и выводит ее в поле компонента `Label1`. Затем считывает из файла строку вида `N R P`, где: `N` — количество альтернативных ответов; `R` — номер правильного ответа; `P` — признак наличия к вопросу иллюстрации (`1` — иллюстрация есть, `0` — иллюстрации нет). Номер правильного ответа фиксируется в глобальной переменной `right`. Если к вопросу есть иллюстрация, то на основании информации о количестве альтернативных ответов вычисляется размер области, которую можно использовать для отображения иллюстрации. После этого функция выводит альтернативные ответы. Положение компонента `OptionBtn(0)`, обеспечивающего вывод первого альтернативного ответа отсчитывается от нижней границы компонента `Image1` (если к вопросу есть иллюстрация) или компонента `Label1`, если иллюстрации нет. Положение остальных компонентов `OptionBtn` отсчитывается от предыдущего

компонента `OptionBtn` (`OptionBtn(1)` от `OptionBtn(0)`, `OptionBtn(2)` от `OptionBtn(1)` и т. д.).

Сразу после вывода вопроса кнопка **Дальше** (`Command1`) недоступна. Сделано это для того, чтобы блокировать возможность перехода к следующему вопросу, если не выбран ответ на текущий. Доступной кнопку **Дальше** делает процедура обработки события `Click` на одном из компонентов `OptionBtn`. Эта же процедура фиксирует в глобальной переменной `otv` номер выбранного ответа (увеличенный на единицу номер компонента `OptionBtn`, в поле которого испытуемый сделал щелчок). Следует обратить внимание, что в рассматриваемой программе для обработки события `Click` на всех компонентах `Option` используется одна процедура (это возможно, т. к. все компоненты `Option` объединены в массив).

После того как пользователь ответит на все вопросы, процедура `itog` (ее вызывает процедура обработки события `Click` на кнопке `Command1`, если вопрос, на который ответил пользователь, — последний, т. е. значение функции `EOF` равно `True`) выводит результат тестирования.

Запуск программы

Пользователи Windows привыкли к тому, что в результате щелчка на имени файла автоматически запускается программа, обеспечивающая работу с этим файлом. Рассмотрим, что надо сделать, чтобы программа тестирования автоматически запускалась в результате щелчка на значке файла теста.

Когда пользователь делает щелчок на имени файла, операционная система определяет тип файла (по расширению) и запускает программу, которая *связана* (предназначена для работы) с файлами этого типа. Таким образом, для того чтобы обеспечить автоматический запуск программы тестирования в результате щелчка на имени файла теста, надо *связать* файл теста и программу тестирования. При этом важно, чтобы у файла теста было уникальное расширение, т. к. именно по расширению операционная система определяет тип файла и, соответственно, программу, которую надо запустить. Очевидно, что расширение txt использовать нельзя. Поэтому сначала надо изменить расширение файла теста с txt, например, на exm (от англ. *examiner* — экзаменатор). После этого можно связать файлы типа EXM с программой "Экзаменатор". Для этого надо раскрыть папку, в которой находится файл теста, щелкнуть правой кнопкой мыши на имени файла теста и в появившемся контекстном меню выбрать команду **Открыть с помощью**. На экране появится окно **Выбор программы**. В поле **Описание** этого окна надо ввести краткое описание типа файла, например **Файл теста программы "Экзаменатор"**,

и щелкнуть на кнопке **Другая**. Затем в появившемся окне **Открыть с помощью** надо раскрыть папку, в которой находится программа "Экзаменатор", выбрать файл программы и щелкнуть на кнопке **Открыть**. В результате этих действий файл с расширением exm будет связан с программой тестирования (рис. 8.5).

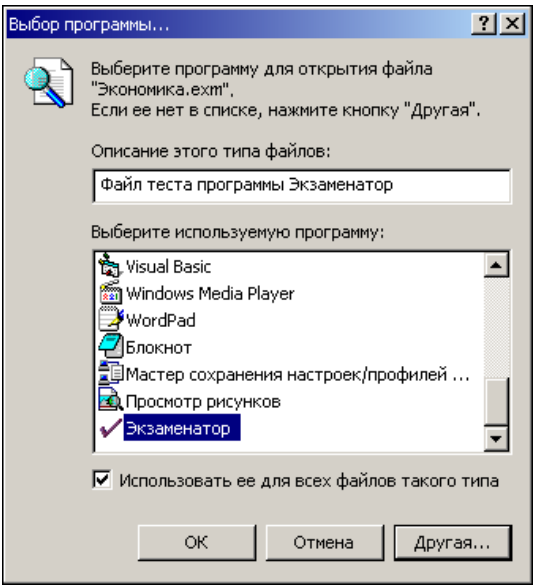


Рис. 8.5. Файлы с расширением exm теперь связаны с программой "Экзаменатор"

Теперь, чтобы запустить программу "Экзаменатор", достаточно сделать щелчок на имени файла теста. Естественно, у вновь создаваемых файлов теста расширение должно быть exm. Следует обратить внимание, что задачу связывания типа файла с конкретным приложением можно возложить на программу установки.

Игра "Сапер"

Всем, кто работает в операционной системе Windows, хорошо знакома игра "Сапер". В этом разделе рассматривается аналогичная программа.

Пример окна программы в конце игры, после того как игрок открыл клетку, в которой находится мина, приведен на рис. 8.6.

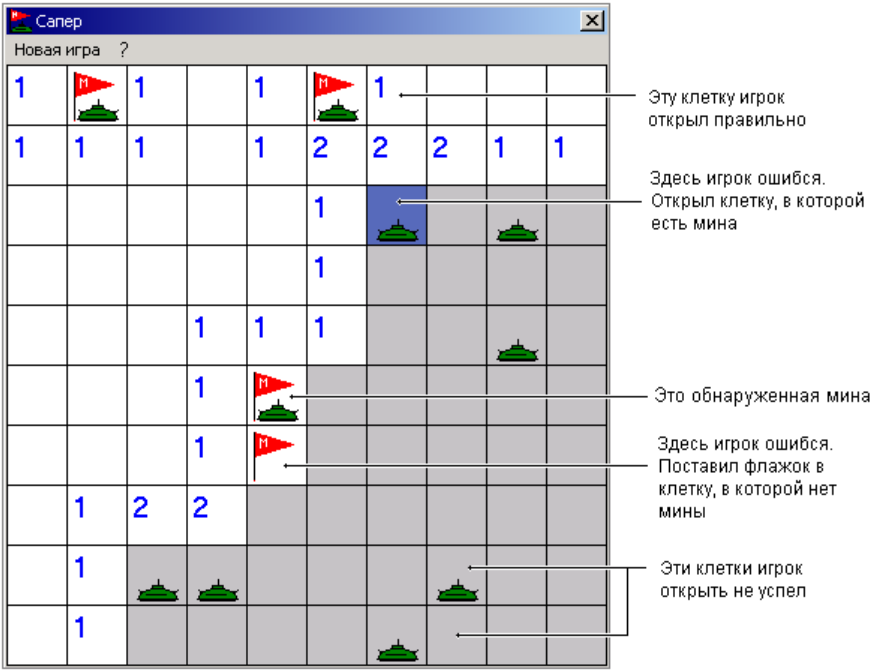


Рис. 8.6. Окно программы "Сапер"

Правила и представление данных

Игровое поле состоит из клеток, в каждой из которых может быть мина. Задача игрока — найти все мины и пометить их флажками.

Используя кнопки мыши, игрок может открыть клетку или поставить в нее флажок, указав тем самым, что в клетке находится мина. Клетка открывается щелчком левой кнопки мыши, флажок ставится щелчком правой. Если в клетке, которую открыл игрок, есть мина, то происходит взрыв (сапер ошибся, а он, как известно, ошибается только один раз) и игра заканчивается. Если в клетке мины нет, то в этой клетке появляется число, соответствующее количеству мин, находящихся в соседних клетках. Анализируя информацию о количестве мин в клетках, соседних с уже открытыми, игрок может обнаружить и пометить флажками все мины. Ограничений на количество клеток, помеченных флажками, нет. Однако для завершения игры (выигрыша) флажки должны быть установлены только в тех клетках, в которых есть мины. Ошибочно установленный флажок можно убрать, щелкнув правой кнопкой мыши в клетке, в которой он находится.

В программе игровое поле представлено массивом N+2 на M+2, где $N \times M$ — размер игрового поля. Элементы массива с номерами строк от 1 до N и номерами столбцов от 1 до M соответствуют клеткам игрового поля (рис. 8.7), первые и последние столбцы и строки соответствуют границе игрового поля.

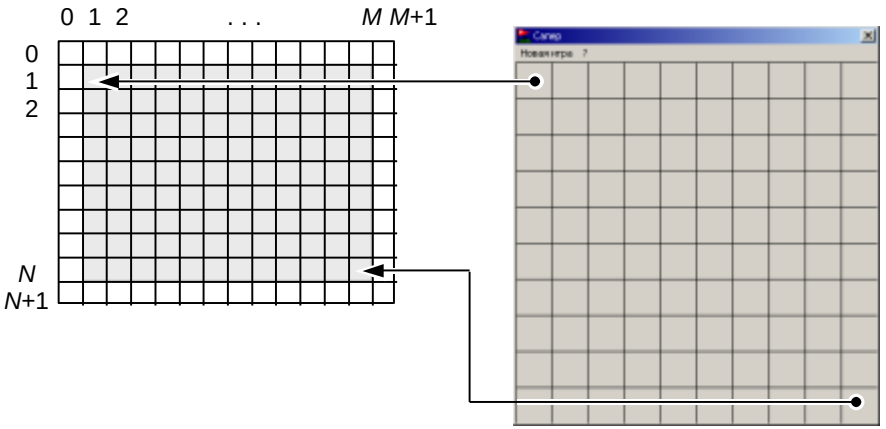


Рис. 8.7. Клетке игрового поля соответствует элемент массива

В начале игры элементы массива, соответствующие клеткам игрового поля, содержат числа от 0 до 9. Если в клетке находится мина, то значение элемента — 9. Если в клетке и в соседней с ней мин нет, то значение элемента — 0. Если в клетке мины нет, но в какой-либо из соседних клеток есть мина, то значением элемента является число, равное количеству мин в соседних клетках (от 1 до 8).

Элементы массива, соответствующие границе поля, содержат –3.

В качестве примера на рис. 8.8 изображен массив, соответствующий состоянию поля в начале игры.

В процессе игры состояние игрового поля меняется (игрок открывает клетки и ставит флажки) и, соответственно, меняются значения элементов массива. Если игрок поставил в клетку флажок, то значение соответствующего элемента массива увеличивается на 100. Например, если флажок поставлен правильно, т. е. в клетку, в которой есть мина, то значение соответствующего элемента массива станет 109. Если флажок поставлен ошибочно, например в пустую клетку, элемент массива будет содержать число 100. Если игрок открыл клетку, то значение элемента массива увеличивается на 200. Такой способ кодирования позволяет сохранить информацию об исходном состоянии клетки.

–3	–3	–3	–3	–3	–3	–3	–3	–3	–3	–3	–3
–3	**9**	1	0	0	0	0	0	0	0	0	–3
–3	1	1	0	0	0	0	0	0	0	0	–3
–3	1	2	2	1	0	0	0	1	1	1	–3
–3	1	**9**	**9**	1	0	0	0	2	**9**	2	–3
–3	1	2	2	1	0	0	0	2	**9**	3	–3
–3	0	0	0	0	0	0	0	2	3	**9**	–3
–3	0	1	2	2	1	0	0	1	**9**	2	–3
–3	0	2	**9**	**9**	1	0	0	1	1	1	–3
–3	0	2	**9**	3	1	0	0	0	0	0	–3
–3	0	1	1	1	0	0	0	0	0	0	–3
–3	–3	–3	–3	–3	–3	–3	–3	–3	–3	–3	–3

Рис. 8.8. Массив в начале игры

Форма

У игры "Сапер" две формы: главная (игровое поле) и **О программе**. Главная (стартовая) форма игры "Сапер" приведена на рис. 8.9.

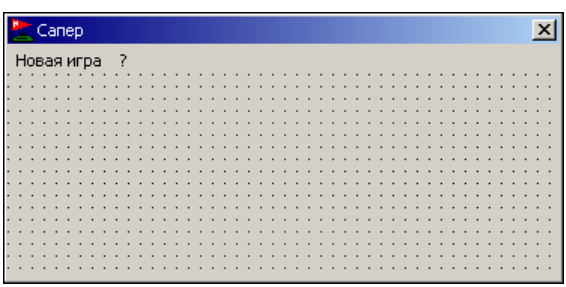


Рис. 8.9. Главная форма программы "Сапер"

Следует обратить внимание, что размер формы не соответствует размеру игрового поля. Нужный размер формы будет установлен во время работы программы. Делает это функция обработки события `Load`, которая на основе информации о размере игрового поля (количестве клеток по вертикали и горизонтали) и размере клеток устанавливает значения свойств `Width` и `Height`.

Управление работой программы осуществляется с помощью команд меню. Меню содержит команды **Новая игра**, **Справка** и **О программе**, назначение которых очевидно. Доступ к командам **Справка** и **О программе** осуществляется через раздел **?**. Меню создается с помощью утилиты Menu Editor, команда запуска которой находится в меню **Tools**. Вид окна **Menu Editor** в конце работы над меню программы "Сапер" приведен на рис. 8.10, характеристики меню — в табл. 8.2.

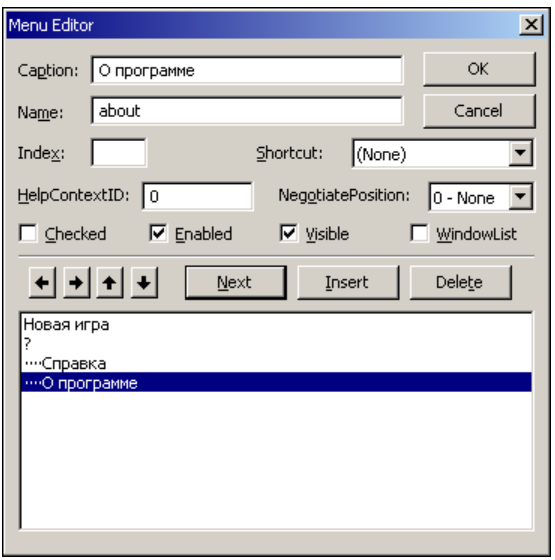


Рис. 8.10. Структура меню программы "Сапер"

Таблица 8.2. *Меню программы "Сапер"*

Команда (Caption)	**Идентификатор (Name)**
Новая игра	`new_game`
?	`question`
Справка	`help`
О программе	`about`

Начало работы программы

В начале работы программы процедура обработки события `Load` (листинг 8.2) сначала загружает из файлов битовые образы (рис. 8.11) — изображения

флажка, мины, помеченной флажком мины и ошибочно открытой мины. Затем на основе информации о размере игрового поля (количестве клеток по горизонтали и вертикали) и размере битового образа (размер всех битовых образов одинаковый) устанавливает размер формы.

Рис. 8.11. Изображения флажка, мины, помеченной флажком мины и ошибочно открытой мины загружаются из файлов

Листинг 8.2. Процедура обработки события `Load`

```
Private Sub Form_Load()
   Dim row As Integer, col As Integer

   ' В неотображаемые элементы массива (клетки по границе
   ' игрового поля) записывается число -3. Это значение
   ' используется процедурой n_open для завершения
   ' рекурсивного процесса открытия соседних пустых клеток
   For row = 0 To MR + 1
      For col = 0 To MC + 1
         Pole(row, col) = -3
      Next col
   Next row

   ' загрузка изображений для клеток
   Set bm1 = LoadPicture("bm1.bmp") ' флажок
   Set bm2 = LoadPicture("bm2.bmp") ' мина
   Set bm3 = LoadPicture("bm3.bmp") ' мина, отмеченная флажком
   Set bm4 = LoadPicture("bm4.bmp") ' мина, на которой сапер подорвался

   W = ScaleX(bm1.Width, vbHimetric, vbPixels)
   H = ScaleX(bm1.Height, vbHimetric, vbPixels)

   ' установка размеров формы
   frmSaper.Width = (frmSaper.Width - frmSaper.ScaleWidth) + _
                    (MC * W) * Screen.TwipsPerPixelX
```

```
    frmSaper.Height = (frmSaper.Height - frmSaper.ScaleHeight) + _
                      (MR * H) * Screen.TwipsPerPixelY

    ' размеры объектов на форме задаются в пикселах
    frmSaper.ScaleMode = vbPixels

    Call NewGame ' новая игра
End Sub
```

Новая игра

В начале игры нужно расставить мины и для каждой клетки поля подсчитать, сколько мин находится в соседних клетках. Процедура `NewGame` (ее текст приведен в листинге 8.3) решает эту задачу.

Листинг 8.3. Процедура NewGame

```
' процедура генерирует новое игровое поле
Sub NewGame()
    Dim row As Integer    ' координаты клетки
    Dim col As Integer

    Dim n As Integer      ' количество поставленных мин
    Dim k As Integer      ' количество мин в соседних клетках

    ' очистка игрового поля
    For row = 1 To MR
        For col = 1 To MC
            Pole(row, col) = 0
        Next col
    Next row

    ' расстановка мин
    Randomize     ' инициализация ГСЧ
    n = 0         ' количество мин

    Do
        row = Int((MR * Rnd) + 1)
        col = Int((MC * Rnd) + 1)
```

```
      If (Pole(row, col) <> 9) Then
         Pole(row, col) = 9
         n = n + 1
      End If
   Loop Until (n = NM)

   ' вычисление количества мин в соседних клетках
   ' для каждой клетки
   For row = 1 To MR
      For col = 1 To MC
         If (Pole(row, col) <> 9) Then
            k = 0
            If Pole(row - 1, col - 1) = 9 Then k = k + 1
            If Pole(row - 1, col) = 9 Then k = k + 1
            If Pole(row - 1, col + 1) = 9 Then k = k + 1
            If Pole(row, col - 1) = 9 Then k = k + 1
            If Pole(row, col + 1) = 9 Then k = k + 1
            If Pole(row + 1, col - 1) = 9 Then k = k + 1
            If Pole(row + 1, col) = 9 Then k = k + 1
            If Pole(row + 1, col + 1) = 9 Then k = k + 1
            Pole(row, col) = k
         End If
      Next col
   Next row

   status = 0        ' начало игры
   nMin = 0          ' нет обнаруженных мин
   nFlag = 0         ' нет поставленных флагов
End Sub
```

После того как процедура `NewGame` расставит мины, процедура `ShowPole` (ее текст приведен в листинге 8.4) выводит изображение игрового поля.

Листинг 8.4. Функция ShowPole

```
' процедура выводит поле
Sub ShowPole(status As Integer)
   Dim row As Integer, col As Integer
```

```
   For row = 1 To MR
      For col = 1 To MC
         Call Kletka(row, col, status)
      Next col
   Next row
End Sub
```

Процедура `ShowPole` выводит изображение поля последовательно, клетка за клеткой. Вывод изображения отдельной клетки выполняет процедура `Kletka`, ее текст приведен в листинге 8.5. Процедура `Kletka` используется для вывода изображения клетки в начале игры, во время игры и в ее конце. В начале игры (значение параметра `status` = 0) функция выводит изображение закрытой клетки, во время игры — количество мин в соседних клетках или флажок, а в конце отображает состояние клетки. Информацию о фазе игры процедура `Kletka` получает через параметр `status`. Непосредственный вывод изображения клетки выполняет метод `PaintPicture`, которому в качестве параметра передается битовый образ, содержащий изображение, соответствующее состоянию клетки.

Листинг 8.5. Процедура `Kletka`

```
Sub Kletka(row As Integer, col As Integer, status As Integer)
   Dim X As Integer, Y As Integer    ' координаты верхнего
                                     ' левого угла клетки

   ' вычислить координаты клетки на поверхности формы
   X = (col - 1) * W
   Y = (row - 1) * H

   If status = 0 Then ' начало игры
      ' клетка закрыта
      Line (X, Y)-Step(W, H), frmSaper.BackColor, BF
      Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
      Exit Sub
   End If

   ' *** неоткрытая клетка ***
   If Pole(row, col) < 100 Then

      Line (X, Y)-Step(W, H), frmSaper.BackColor, BF
```

```
    Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
    ' если игра завершена (status = 2) и клетка с миной
    ' не была открыта, открываем ее (чтобы игрок увидел, где
    ' находятся мины)
    If (status = 2) And (Pole(row, col) = 9) Then _
       frmSaper.PaintPicture bm2, X, Y           ' мина
       Line (X, Y)-Step(W, H), RGB(0, 0, 0), B   ' контур клетки
    Exit Sub
End If

' *** открытая клетка ***
Line (X, Y)-Step(W, H), RGB(255, 255, 255), BF
Line (X, Y)-Step(W, H), RGB(0, 0, 0), B

' клетка открыта, в соседних клетках нет мин
If (Pole(row, col) = 100) Then Exit Sub

' клетка открыта, в соседних клетках есть мины
If (Pole(row, col) >= 101) And (Pole(row, col) <= 108) Then
    frmSaper.CurrentX = X + 3
    frmSaper.CurrentY = Y + 3

    ' вывод количества мин в соседних клетках
    ' шрифт определяют свойства формы Font и ForeColor
    Print Str(Int(Pole(row, col) - 100))
    Exit Sub
End If

' в клетку поставлен флаг
If (Pole(row, col) >= 200) Then
     frmSaper.PaintPicture bm1, X, Y         ' флаг
     Line (X, Y)-Step(W, H), RGB(0, 0, 0), B ' контур клетки
End If

' на этой мине подорвались
If (Pole(row, col) = 109) Then
     frmSaper.PaintPicture bm4, X, Y         ' мина
     Line (X, Y)-Step(W, H), RGB(0, 0, 0), B ' контур клетки
```

```
  End If

   ' правильно поставленный флаг
   If (Pole(row, col) = 209) And (status = 2) Then
      frmSaper.PaintPicture bm3, X, Y     ' мина, помеченная флагом
      Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
   End If
End Sub
```

Игра

Во время игры программа воспринимает нажатия кнопок мыши и в соответствии с правилами игры открывает клетки или ставит в клетки флажки.

Основную работу выполняет процедура обработки события `MouseDown` (ее текст приведен в листинге 8.6). Процедура получает координаты точки формы, в которой игрок щелкнул кнопкой мыши, а также информацию о том, какая кнопка была нажата. Сначала процедура преобразует координаты точки формы, в которой игрок нажал кнопку мыши, в координаты клетки игрового поля. Затем делает необходимые изменения в массиве `Pole` и, если нажата правая кнопка, вызывает функцию `Kletka`, которая отображает содержимое клетки. Если нажата левая кнопка в клетке, в которой мины нет, то вызывается процедура `n_open`, которая открывает клетку (отображает ее содержимое). Если левая кнопка нажата в клетке, в которой есть мина, то фиксируется факт окончания игры и процедура `ShowPole` показывает все мины, в том числе и те, которые игрок не успел найти.

Листинг 8.6. Обработка события `MouseDown` на поверхности игрового поля

```
' нажатие кнопки мыши на игровом поле
Private Sub Form_MouseDown(Button As Integer, _
   Shift As Integer, X As Single, Y As Single)

   Dim row As Integer, col As Integer

   If status = 2 Then Exit Sub       ' игра завершена

   If status = 0 Then status = 1     ' первый щелчок

   ' преобразование координат мыши в индексы клетки поля
```

```
row = Int(Y / H) + 1
col = Int(X / W) + 1

' нажатие левой кнопки мыши
If Button = vbLeftButton Then
   If Pole(row, col) = 9 Then
      ' открыта клетка, в которой есть мина
      Pole(row, col) = Pole(row, col) + 100
      status = 2              ' игра закончена
      Call ShowPole(status)   ' вывод поля
   Else
      ' открытие клетки
      If Pole(row, col) < 9 Then Call n_open(row, col)
   End If
End If

' нажатие правой кнопки мыши
If Button = vbRightButton Then
   ' в клетке стоит флаг, пользователь хочет убрать его
   If Pole(row, col) >= 200 Then
      nFlag = nFlag - 1
      ' уберем флаг из клетки
      Pole(row, col) = Pole(row, col) - 200
      ' закрытие клетки
      Call Kletka(row, col, status)

   ' в клетке нет флага, пользователь хочет его поставить
   Else
      ' если клетка открыта, то флаг нельзя поставить,
      ' если клетка закрыта, то можно
      If Pole(row, col) >= 100 Then Exit Sub

      nFlag = nFlag + 1
      Pole(row, col) = Pole(row, col) + 200    ' установка
                                               ' флага
      Call Kletka(row, col, status)            ' вывод флага

      If Pole(row, col) = 209 Then
```

```
            nMin = nMin + 1

            ' если все мины помечены флажками, игра закончена
            If (nMin = NM) And (nFlag = NM) Then
               status = 2                ' игра закончена
               Call ShowPole(status)     ' вывод поля
            End If
         End If

      End If
   End If

End Sub
```

Справочная информация

В результате выбора в меню **?** команды **Справка** или нажатия клавиши <F1> должна появляться справочная информация — правила игры (рис. 8.12).

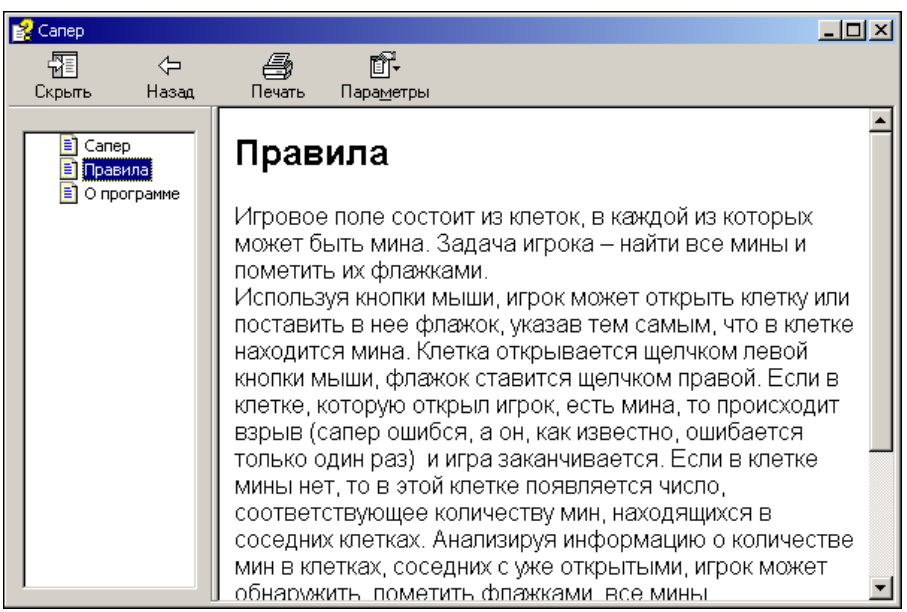


Рис. 8.12. Окно справочной системы программы "Сапер"

Для того чтобы во время работы программы пользователь, нажав клавишу <F1>, мог получить справочную информацию, надо ввести имя файла спра-

вочной информации в поле **Help File Name** окна **Project Properties**, которое становится доступным в результате выбора в меню **Project** команды **Project Properties**.

Для того чтобы справочная информация появилась на экране в результате выбора в меню **?** команды **Справка**, надо создать функцию обработки события `Click` для соответствующей команды меню. Чтобы это сделать, нужно в окне формы раскрыть меню и сделать щелчок мышью в строке команды.

Ниже приведена функция обработки события `Click` для команды **Справка**.

```
' выбор пункта меню Справка
Private Sub help_Click()
    Shell "hh saper.chm", vbNormalFocus
End Sub
```

Информация о программе

При выборе из меню **?** команды **О программе** на экране должно появиться одноименное окно (рис. 8.13).

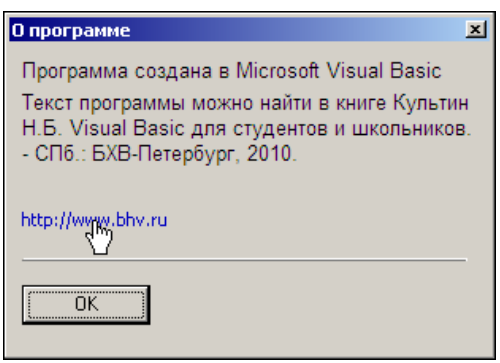


Рис. 8.13. Выбрав ссылку, можно активизировать браузер и перейти на страницу издательства "БХВ-Петербург"

Чтобы программа во время своей работы могла вывести на экран окно, отличное от главного (стартового), нужно добавить в проект форму. Делается это выбором из меню **Project** команды **Add Form**. В результате выполнения команды в проект добавляется новая форма и соответствующий ей модуль.

Вид формы `frmAbout` после добавления необходимых компонентов приведен на рис. 8.14, значения ее свойств — в табл. 8.3.

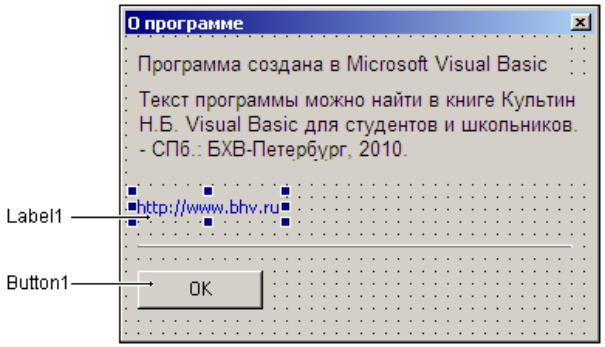


Рис. 8.14. Форма **О программе**

Таблица 8.3. *Значения свойств формы* ***О программе***

Свойство	Значение
`Name`	`frmAbout`
`BorderStyle`	`4 — Fixed ToolWindow`
`StartUpPosition`	`1 — CenterOwner`

Вывод окна **О программе** выполняет функция обработки события `Click`, которое происходит в результате выбора из меню **?** команды **О программе** (листинг 8.7). Непосредственно вывод окна выполняет метод `Show` с параметром `vbModal`, который выводит окно как *модальный* диалог. Модальный диалог перехватывает все события, адресованные другим окнам приложения, в том числе и главному. Таким образом, пока модальный диалог находится на экране, продолжить работу с приложением, которое вывело модальный диалог, нельзя.

Листинг 8.7. Вывод окна *О программе*

```
' выбор команды О программе в меню ?
Private Sub about_Click()
    frmAbout.Show vbModal, Me
End Sub
```

Следует обратить внимание, что окно **О программе** появляется в центре главного окна программы. Это происходит потому, что свойству `StartUpPosition`

присвоено значение `CenterOwner`, предписывающее отображать объект в центре окна, которое активизировало процесс отображения. В данном случае процесс отображения активизировало главное окно программы.

На поверхности формы **О программе** есть ссылка на сайт издательства "БХВ-Петербург". Предполагается, что в результате щелчка на ссылке в окне браузера будет открыта указанная страница. Для того чтобы это произошло, надо создать функцию обработки события `Click` для компонента `Label1`. Кроме того, чтобы при позиционировании указателя мыши на ссылке он принимал форму руки, в проект надо добавить соответствующий *курсор* и установить значения свойств `MousePointer` и `MouseIcon`. Курсор — это битовый образ размером 32×32. Можно использовать один из готовых курсоров (курсоры находятся в файлах с расширением cur) или с помощью утилиты Image Editor создать свой собственный (рис. 8.15). Процесс создания курсора практически ничем не отличается от процесса создания значка приложения, за исключением того, что в окне **Resource Type**, которое появляется в результате выбора в меню **File** команды **New**, надо выбрать **Cursor**, а не **Icon**.

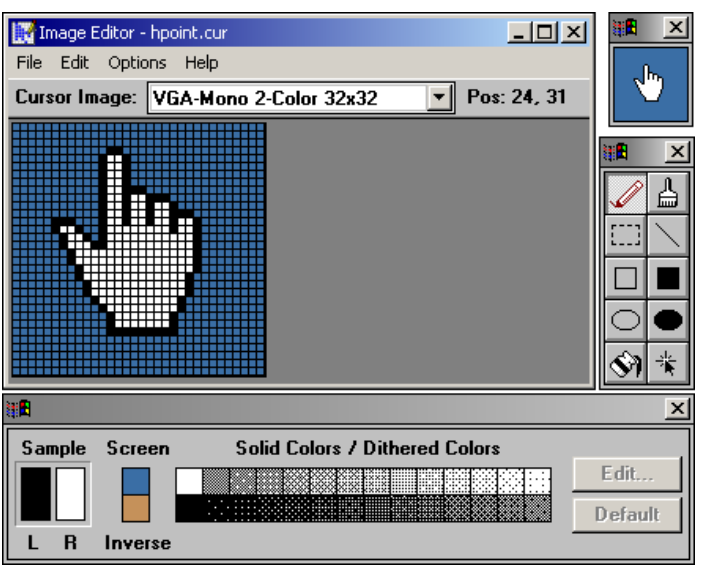


Рис. 8.15. Курсор можно создать с помощью утилиты Image Editor

Значения свойств компонента `Label1` приведены в табл. 8.4, текст процедуры обработки события `Click` в поле компонента — в листинге 8.8.

Таблица 8.4. *Значения свойств компонента* `Label1`

Свойство	Значение
Caption	http://www.bhv.ru
MousePointer	99 — Custom
MouseIcon	hpoint.cur

Листинг 8.8. Щелчок в поле URL

```
Private Declare Function ShellExecute Lib "shell32.dll" Alias _
  "ShellExecuteA" (ByVal hwnd As Long, ByVal lpOperation As String, _
           ByVal lpFile As String, ByVal lpParameters As String, _
           ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long

Private Sub Label1_Click()
    Dim r As Long
    r = ShellExecute(Me.hwnd, "Open", ""http://www.bhv.ru",, "", "", 0)
End Sub
```

Для запуска браузера использована API-функция `ShellExecute`, которая сообщает операционной системе, что необходимо открыть указанный файл. Так как в данном случае документ — URL, то операционная система запускает программу, обеспечивающую работу с документами указанного типа, т. е. браузер.

Окно **О программе** закрывается в результате щелчка на кнопке **OK**. Функция обработки этого события приведена в листинге 8.9.

Листинг 8.9. Щелчок на кнопке *OK*

```
Private Sub Command1_Click()
    Hide ' закрыть (скрыть) окно О программе
End Sub
```

Текст программы

Полный текст программы "Сапер" приведен в листингах 8.10 (модуль главной формы) и 8.11 (модуль формы **О программе**).

Листинг 8.10. Модуль главной формы

```
Const MR = 10    ' количество клеток по вертикали
Const MC = 10    ' количество клеток по горизонтали
Const NM = 10    ' количество мин

' минное поле
Dim Pole(0 To MR + 1, 0 To MC + 1) As Integer
' значение элемента массива:
' 0—8 — количество мин в соседних клетках,
' 9 — в клетке мина,
' 100—109 — клетка открыта,
' 200—209 — в клетку поставлен флаг

Dim nMin As Integer      ' количество найденных мин
Dim nFlag As Integer     ' количество поставленных флагов
Dim status As Integer    ' статус игры: 0 — начало игры,
                         ' 1 — идет игра, 2 — результат игры

' изображения клеток
Dim bm1 As StdPicture ' флажок
Dim bm2 As StdPicture ' мина
Dim bm3 As StdPicture ' мина, отмеченная флажком
Dim bm4 As StdPicture ' мина, на которой сапер подорвался

Dim W, H As Integer ' размер клетки

' начало работы программы
Private Sub Form_Load()
   Dim row As Integer, col As Integer

   ' В неотображаемые элементы массива (клетки по границе
   ' игрового поля) записывается число -3. Это значение
   ' используется процедурой n_open для завершения
   ' рекурсивного процесса открытия соседних пустых клеток
   For row = 0 To MR + 1
      For col = 0 To MC + 1
```

```
            Pole(row, col) = -3
        Next col
    Next row

    ' загрузка изображений для клеток
    Set bm1 = LoadPicture("bm1.bmp") ' флажок
    Set bm2 = LoadPicture("bm2.bmp") ' мина
    Set bm3 = LoadPicture("bm3.bmp") ' мина, отмеченная флажком
    Set bm4 = LoadPicture("bm4.bmp") ' мина, на которой сапер подорвался

    W = ScaleX(bm1.Width, vbHimetric, vbPixels)
    H = ScaleX(bm1.Height, vbHimetric, vbPixels)

    ' установка размеров формы
    frmSaper.Width = (frmSaper.Width - frmSaper.ScaleWidth) + _
                     (MC * W) * Screen.TwipsPerPixelX
    frmSaper.Height = (frmSaper.Height - frmSaper.ScaleHeight) + _
                      (MR * H) * Screen.TwipsPerPixelY

    ' размеры объектов на форме задаются в пикселах
    frmSaper.ScaleMode = vbPixels
    Call NewGame          ' новая игра
End Sub

' процедура генерирует новое игровое поле
Sub NewGame()
    Dim row As Integer    ' координаты клетки
    Dim col As Integer

    Dim n As Integer      ' количество поставленных мин
    Dim k As Integer      ' количество мин в соседних клетках

    ' очистка игрового поля
    For row = 1 To MR
        For col = 1 To MC
            Pole(row, col) = 0
        Next col
```

```
   Next row

   ' расстановка мин
   Randomize     ' инициализация ГСЧ
   n = 0         ' количество мин
   Do
      row = Int((MR * Rnd) + 1)
      col = Int((MC * Rnd) + 1)
      If (Pole(row, col) <> 9) Then
         Pole(row, col) = 9
         n = n + 1
      End If
   Loop Until (n = NM)

   ' вычисление количества мин в соседних клетках
   ' для каждой клетки
   For row = 1 To MR
      For col = 1 To MC
         If (Pole(row, col) <> 9) Then
            k = 0
            If Pole(row - 1, col - 1) = 9 Then k = k + 1
            If Pole(row - 1, col) = 9 Then k = k + 1
            If Pole(row - 1, col + 1) = 9 Then k = k + 1
            If Pole(row, col - 1) = 9 Then k = k + 1
            If Pole(row, col + 1) = 9 Then k = k + 1
            If Pole(row + 1, col - 1) = 9 Then k = k + 1
            If Pole(row + 1, col) = 9 Then k = k + 1
            If Pole(row + 1, col + 1) = 9 Then k = k + 1
            Pole(row, col) = k
         End If
      Next col
   Next row

   status = 0       ' начало игры
   nMin = 0         ' нет обнаруженных мин
   nFlag = 0        ' нет поставленных флагов
End Sub
```

```
' процедура выводит поле
Sub ShowPole(status As Integer)
   Dim row As Integer, col As Integer

   For row = 1 To MR
      For col = 1 To MC
         Call Kletka(row, col, status)
      Next col
   Next row
End Sub

' выводит изображение клетки
Sub Kletka(row As Integer, col As Integer, status As Integer)
   Dim X As Integer, Y As Integer    ' координаты левого
                                     ' верхнего угла клетки
   ' вычислить координаты клетки на поверхности формы
   X = (col - 1) * W
   Y = (row - 1) * H
   If status = 0 Then ' начало игры
      ' клетка закрыта
      Line (X, Y)-Step(W, H), frmSaper.BackColor, BF
      Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
      Exit Sub
   End If

   ' *** неоткрытая клетка ***
   If Pole(row, col) < 100 Then
      Line (X, Y)-Step(W, H), frmSaper.BackColor, BF
      Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
      ' если игра завершена (status = 2) и клетка с миной
      ' не была открыта, открываем ее (чтобы игрок увидел мину)
      If (status = 2) And (Pole(row, col) = 9) Then _
         frmSaper.PaintPicture bm2, X, Y ' мина
         Line (X, Y)-Step(W, H), RGB(0, 0, 0), B ' контур клетки
      Exit Sub
   End If

   ' *** открытая клетка ***
   Line (X, Y)-Step(W, H), RGB(255, 255, 255), BF
```

```
    Line (X, Y)-Step(W, H), RGB(0, 0, 0), B

    ' клетка открыта, в соседних клетках нет мин
    If (Pole(row, col) = 100) Then Exit Sub

  ' клетка открыта, в соседних клетках есть мины
    If (Pole(row, col) >= 101) And (Pole(row, col) <= 108) Then
       frmSaper.CurrentX = X + 3
       frmSaper.CurrentY = Y + 3

       ' вывод количества мин в соседних клетках
       ' шрифт определяют свойства формы Font и ForeColor
       Print Str(Int(Pole(row, col) - 100))
       Exit Sub
    End If

    ' правильно поставленный флаг
    If (Pole(row, col) = 209) And (status = 2) Then
       frmSaper.PaintPicture bm3, X, Y  ' мина, помеченная флагом
       Line (X, Y)-Step(W, H), RGB(0, 0, 0), B
    End If

  ' неправильно поставленный флаг
    If (Pole(row, col) >= 200) Then
       frmSaper.PaintPicture bm1, X, Y          ' флаг
       Line (X, Y)-Step(W, H), RGB(0, 0, 0), B  ' контур клетки
    End If

    ' на этой мине подорвались
    If (Pole(row, col) = 109) Then
       frmSaper.PaintPicture bm4, X, Y          ' мина
       Line (X, Y)-Step(W, H), RGB(0, 0, 0), B  ' контур клетки
   End If
End Sub

' нажатие кнопки мыши на игровом поле
Private Sub Form_MouseDown(Button As Integer, _
```

```
Shift As Integer, X As Single, Y As Single)

Dim row As Integer, col As Integer

If status = 2 Then Exit Sub       ' игра завершена

If status = 0 Then status = 1     ' первый щелчок

' преобразование координат мыши в индексы клетки поля
row = Int(Y / H) + 1
col = Int(X / W) + 1

' нажатие левой кнопки мыши
If Button = vbLeftButton Then
   If Pole(row, col) = 9 Then
      ' открыта клетка, в которой есть мина
      Pole(row, col) = Pole(row, col) + 100
      status = 2              ' игра закончена
      Call ShowPole(status)   ' вывод поля
   Else
      ' открытие клетки
      If Pole(row, col) < 9 Then Call n_open(row, col)
   End If
End If

' нажатие правой кнопки мыши
If Button = vbRightButton Then
   ' в клетке стоит флаг, пользователь хочет убрать его
   If Pole(row, col) >= 200 Then
      nFlag = nFlag - 1
      ' уберем флаг из клетки
      Pole(row, col) = Pole(row, col) - 200
      Call Kletka(row, col, status)

   Else ' в клетке нет флага, пользователь хочет его поставить
        ' если клетка открыта, то флаг нельзя поставить,
        ' если клетка закрыта, то можно
```

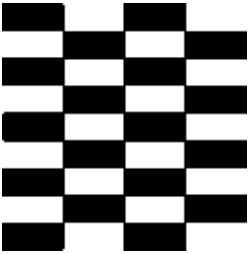
```
        If Pole(row, col) >= 100 Then Exit Sub

        nFlag = nFlag + 1
        Pole(row, col) = Pole(row, col) + 200    ' установка
                                                 ' флага
        Call Kletka(row, col, status)            ' вывод флага

        If Pole(row, col) = 209 Then
           nMin = nMin + 1
           ' если все флаги расставлены на правильных местах
           If (nMin = NM) And (nFlag = NM) Then
              status = 2                ' игра закончена
              Call ShowPole(status)     ' вывод поля
           End If
        End If

     End If
  End If

End Sub

' рекурсивная процедура открывает текущую и все соседние
' клетки, в которых нет мин
Sub n_open(row As Integer, col As Integer)
   If Pole(row, col) = 0 Then
      Pole(row, col) = 100
      Call Kletka(row, col, 1)
      ' примыкающие клетки по вертикали и горизонтали
      Call n_open(row, col - 1)
      Call n_open(row - 1, col)
      Call n_open(row, col + 1)
      Call n_open(row + 1, col)
      ' примыкающие диагонально
      Call n_open(row - 1, col - 1)
      Call n_open(row - 1, col + 1)
      Call n_open(row + 1, col - 1)
      Call n_open(row + 1, col + 1)
```

```
   Else
      If (Pole(row, col) < 100) And (Pole(row, col) <> -3) Then
         Pole(row, col) = Pole(row, col) + 100
         Call Kletka(row, col, 1)
      End If
   End If
End Sub

' обработка события Paint
Private Sub Form_Paint()
   ' вывод игрового поля
   Call ShowPole(status)
End Sub

' выбор команды О программе в меню ?
Private Sub about_Click()
    frmAbout.Show vbModal, Me
End Sub

' выбор пункта меню Справка
Private Sub help_Click()
    Shell "hh saper.chm", vbNormalFocus
End Sub

' выбор пункта меню Новая игра
Private Sub new_game_Click()
   Call NewGame              ' новая игра
   Call ShowPole(status)     ' вывод игрового поля
End Sub
```

Листинг 8.11. Модуль формы *О программе*

```
Private Declare Function ShellExecute Lib "shell32.dll" Alias _
   "ShellExecuteA" (ByVal hwnd As Long, ByVal lpOperation As String, _
              ByVal lpFile As String, ByVal lpParameters As String, _
         ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long

' щелчок на Web-ссылке (в поле компонента Label1)
```

```
Private Sub Label1_Click()
    Dim r As Long
    r = ShellExecute(Me.hwnd, "Open", ""http://www.bhv.ru", ", "", "", 0)
End Sub

' щелчок на кнопке OK
Private Sub Command1_Click()
    Hide ' закрыть (скрыть) окно О программе
End Sub
```



Глава 9

Справочник

Основные типы данных

Таблица 9.1. *Типы данных Visual Basic*

Тип	Диапазон значений
`Byte` (целое без знака)	От 0 до 255
`Integer` (целое)	От –32 768 до 32 767
`Long` (длинное целое)	От –2 147 483 648 до 2 147 483 647
`Single` (с плавающей точкой, обычной точности)	От –3,402823E38 до –1,401298E–45 для отрицательных значений; от 1,401298E–45 до 3,402823E38 для положительных значений
`Double` (с плавающей точкой, двойной точности)	От –1,79769313486232E308 до –4,94065645841247E–324 для отрицательных значений; от 4,94065645841247E–324 до 1,79769313486232E308 для положительных значений
`Currency` (денежный)	От –922 337 203 685 477,5808 до 922 337 203 685 477,5807
`Decimal` (масштабируемое целое)	±79 228 162 514 264 337 593 543 950 335 без дробной части; ±7,9228162514264337593543950335 с 28 знаками справа от запятой; минимальное ненулевое значение имеет вид ±0,0000000000000000000000000001
`Date` (даты и время)	От 1 января 100 г. до 31 декабря 9999 г.
`String` (строка переменной длины)	Строка переменной длины: от 0 до приблизительно 2 миллиардов символов. Строка постоянной длины: от 1 до приблизительно 65 400 символов

Переменная

В общем виде объявление переменной выглядит так:

```
Dim Имя As Тип
```

Здесь *Имя* и *Тип* — соответственно, имя и тип объявляемой переменной.

Массив

Одномерный массив

В общем виде объявление одномерного массива выглядит так:

```
Dim Имя(Н To В) As Тип
```

Здесь:

- *Имя* — имя массива;
- *Н* — нижняя граница диапазона изменения индекса;
- *В* — верхняя граница диапазона изменения индекса;
- *Тип* — тип элементов массива.

Двумерный массив

В общем виде объявление двумерного массива выглядит так:

```
Dim Имя(Н1 To В1, Н2 To В2) As Тип
```

Здесь:

- *Имя* — имя массива; *Н1*, *В1* и *Н2*, *В2* — целые константы, определяющие, соответственно, диапазон изменения индекса строки и столбца;
- *Тип* — тип элементов массива.

Выбор

Инструкция *If*

Инструкция `If` позволяет выбрать один из двух возможных вариантов развития программы. В общем виде она записывается так:

```
If Условие Then
     Действие 1
Else
     Действие 2
End If
```

Инструкция *Select Case*

Инструкция `Select` позволяет реализовать множественный выбор. В общем виде она записывается так:

```
Select Case Селектор
     Case Список_1
          Инструкции1
     Case Список_2
          Инструкции2
     Case Список_3
          Инструкции3
     . . .
     Case Else
          Инструкции
End Select
```

Здесь *Селектор* — выражение, значение которого определяет дальнейший ход развития программы; *Список_i* — список констант (разделенные запятыми константы).

Циклы

Инструкция *For*

Инструкция `For` — цикл с фиксированным количеством повторений. В общем виде она записывается так:

```
For Счетчик = Нач_знач To Кон_знач Step Приращение
     Инструкции
Next Счетчик
```

Здесь:

- *Счетчик* — переменная счетчика циклов;
- *Нач_знач* — выражение, определяющее начальное значение счетчика циклов;
- *Кон_знач* — выражение, определяющее конечное значение счетчика циклов;
- *Приращение* — величина изменения счетчика циклов после каждого выполнения инструкций "тела" цикла (если значение приращения равно единице, то слово `Step` и величину приращения можно не указывать).

Инструкция *Do Loop*

Инструкция `Do Loop` — цикл с постусловием. Существуют два варианта этой инструкции: `Do Loop While` и `Do Loop Until`, которые в общем виде записываются так:

```
Do
    Инструкции
Loop While Условие
```

и

```
Do
    Инструкции
Loop Until Условие
```

Здесь:

- *Инструкции* — инструкции, которые надо выполнить несколько раз;
- *Условие* — условие повторения (для цикла `While`) или завершения (для цикла `Until`) цикла.

Инструкция *Do While*

Инструкция `Do While` — цикл с предусловием. В общем виде она записывается так:

```
Do While Условие
    Инструкции
Loop
```

Здесь:

- *Инструкции* — инструкции, которые надо выполнить несколько раз;
- *Условие* — условие повторения (для цикла `While`) или завершения (для цикла `Until`) цикла.

Функция программиста

Объявление функции в общем виде выглядит так:

```
Function Имя(Параметры) As Тип
    ' здесь инструкции, реализующие функцию
    Имя = Значение
End Function
```

Здесь:

- `Function` — зарезервированное слово языка Visual Basic, показывающее, что далее следуют инструкции, реализующие функцию;
- *Имя* — имя функции;
- *Параметры* — список переменных, которые используется для передачи в функцию информации, необходимой для вычисления значения функции;
- *Тип* — тип значения функции.

Форма

Форма (объект `Form`) является основой программы. Свойства формы приведены в табл. 9.2.

Таблица 9.2. *Свойства формы*

Свойство	Описание
`Name`	Имя формы. Используется для управления формой и доступа к ее компонентам или свойствам
`Caption`	Текст заголовка
`Top`	Расстояние от верхней границы формы до верхней границы экрана
`Left`	Расстояние от левой границы формы до левой границы экрана
`Width`	Ширина формы. Задается в твипах
`Height`	Высота формы. Задается в твипах
`ScaleWidth`	Ширина рабочей области формы, т. е. без учета ширины левой и правой границ. Может задаваться как в твипах, так и в других единицах
`ScaleHeight`	Высота рабочей (клиентской) области формы, т. е. без учета высоты заголовка и ширины нижней и верхней границ формы. Может задаваться как в твипах, так и в других единицах
`ScaleMode`	Определяет единицы измерения размеров формы и объектов на ней. Значение этого свойства не влияет на единицы измерения свойств `Width` и `Height`, независимо от него их значения измеряются в твипах
`BorderStyle`	Стиль (вид) границы формы (окна программы). Граница может быть обычной (`Sizable`), тонкой (`FixedSingle`) или вообще отсутствовать (`None`). Если значение свойства равно `FixedSingle`, то изменить размер окна путем перемещения границы нельзя, но окно можно развернуть на весь экран (сделав щелчок на кнопке **Развернуть**) или свернуть (сделав щелчок на кнопке **Свернуть**). Если значение свойства равно `None`, то граница окна отсутствует. Изменить размер и положение такого окна нельзя. Если значение свойства равно `FixedDialog`, то окно — модальный диалог (нельзя изменить размер окна, нельзя свернуть окно, доступ к другим окнам программы блокируется)

Таблица 9.2 *(окончание)*

Свойство	Описание
Icon	Значок в заголовке окна
BackColor	Цвет фона формы. Цвет фона можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы. Во втором случае цвет определяется текущей цветовой схемой и выбранным компонентом привязки. В этом случае он меняется при изменении цветовой схемы операционной системы
ForeColor	Цвет, используемый при выводе текста и для контуров графических объектов
FillColor	Цвет заполнения внутренней части графических объектов
FillStyle	Стиль (способ) заливки графических объектов: • Solid (0) — сплошная заливка; • Transparent (1) — прозрачный цвет (заливки нет); • HorizontalLine (2) — горизонтальная штриховка; • VerticalLine (3) — вертикальная штриховка. Цвет линий штриховки определяет значение свойства FillColor
DrawMode	Способ вывода графических объектов. Например, если значение свойства равно Blackness (1), то цвет всех контуров объектов и цвет заливки будет черным (значение этого свойства не влияет на цвет текста, выводимого с помощью метода Print)
DrawWidth	Толщина линии для графических объектов
DrawStyle	Стиль контура графических объектов, тип линии: • Solid (0) — сплошная линия; • Dash (1) — пунктирная линия; • Dot (2) — линия из точек; • Dash-Dot (3) — линия "точка-тире"; • Dash-Dot-Dot (4) — линия "тире-точка-точка"; • Transparent (5) — прозрачная линия
Font	Шрифт, заданный в этом свойстве, используется при выводе текста непосредственно на поверхность формы (например, с помощью команды Print)
MaxButton	Признак наличия (True) или отсутствия (False) в заголовке формы кнопки **Развернуть окно на весь экран**
MinButton	Признак наличия (True) или отсутствия (False) в заголовке формы кнопки **Свернуть окно**

Компоненты

CheckBox

Компонент `CheckBox` (рис. 9.1) представляет собой флажок. Свойства компонента приведены в табл. 9.3.

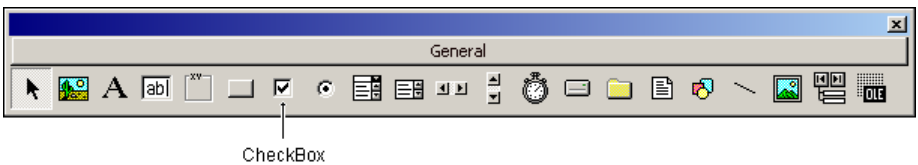


Рис. 9.1. Компонент `CheckBox`

Таблица 9.3. *Свойства компонента* `CheckBox`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`Caption`	Текст, который находится справа от флажка
`Value`	Состояние переключателя: `Checked` — флажок установлен (в квадратике есть галочка); `Unchecked` — флажок сброшен (нет галочки)
`Left`	Расстояние от левой границы флажка до левой границы формы
`Top`	Расстояние от верхней границы флажка до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента (флажок и область для пояснительного текста)
`Font`	Шрифт, используемый для отображения пояснительного текста
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

ComboBox

Компонент `ComboBox` (рис. 9.2) дает возможность ввести данные в поле редактирования путем набора на клавиатуре или путем выбора из списка. Свойства компонента приведены в табл. 9.4.

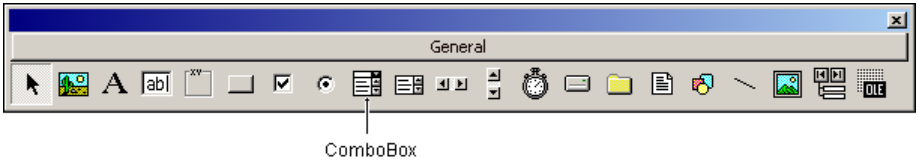


Рис. 9.2. Компонент `ComboBox`

Таблица 9.4. *Свойства компонента* `ComboBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Font`	Шрифт, используемый для отображения элементов списка
`BackColor`	Цвет фона области вывода списка. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`ForeColor`	Цвет, используемый для отображения элементов списка
`List`	Элементы списка — массив строк
`ListCount`	Количество элементов списка
`ListIndex`	Номер элемента, выбранного в списке. Если ни один из элементов списка не был выбран, то значение свойства равно `-1`. Нумерация элементов начинается с нуля
`Style`	Стиль (вид) списка. Если значение свойства равно `DropDownCombo` (`0`), то данные в поле редактирования можно ввести с клавиатуры или выбрать из списка (чтобы получить доступ к списку, его надо раскрыть). Если значение свойства равно `SimpleCombo` (`1`), то данные можно ввести в поле редактирования с клавиатуры или выбрать из списка, причем список доступен всегда. Если значение свойства равно `DropDownList` (`2`), то данные в поле редактирования можно ввести только путем выбора из списка
`Text`	Содержимое поля редактирования (данные, введенные пользователем с клавиатуры или выбранные из списка)
`Sorted`	Признак необходимости автоматической сортировки списка после добавления очередного элемента (значение свойства — `True`)
`Locked`	Блокировка списка. Определяет запрет выбора элемента из списка (строка ввода также блокируется)
`Left`	Расстояние от левой границы компонента до левой границы формы

Таблица 9.4 *(окончание)*

Свойство	Описание
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

CommandButton

Компонент `CommandButton` (рис. 9.3) представляет собой командную кнопку. Свойства компонента приведены в табл. 9.5.

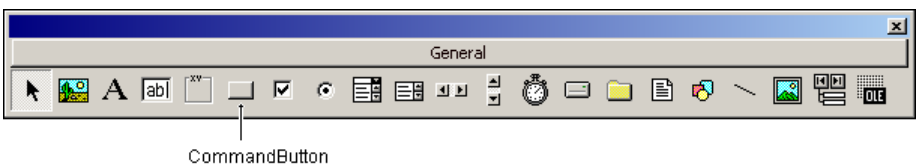


Рис. 9.3. Компонент `CommandButton`

Таблица 9.5. *Свойства компонента* `CommandButton`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`Caption`	Текст на кнопке
`Left`	Расстояние от левой границы кнопки до левой границы формы
`Top`	Расстояние от верхней границы кнопки до верхней границы формы
`Height`	Высота кнопки
`Width`	Ширина кнопки

Таблица 9.5 *(окончание)*

Свойство	Описание
`Enabled`	Признак доступности кнопки. Если значение свойства равно `True`, то кнопка доступна. Если значение свойства равно `False`, то кнопка не доступна, т. е. при щелчке на кнопке никаких событий не возникает
`Visible`	Позволяет скрыть кнопку (значение — `False`) или сделать ее видимой (значение — `True`)
`Style`	Вид кнопки: "обычная" (`Standard`) или "графическая" (`Graphical`). Графическая кнопка — это кнопка, на поверхности которой есть картинка
`Picture`	Свойство задает картинку для "графической" кнопки. Картинка отображается на поверхности формы, если значение свойства `Style` равно `Graphical`
`DisabledPicture`	Задает картинку для недоступной "графической" кнопки. Картинка отображается, если значение свойства `Style` равно `Graphical`, а свойства `Enabled` — `False`
`DownPicture`	Задает картинку для нажатой "графической" кнопки. Картинка отображается в момент нажатия кнопки, если значение свойства `Style` равно `Graphical`
`ToolTipText`	Задает текст подсказки, которая появляется при позиционировании указателя мыши на кнопке

CommonDialog

Компонент `CommonDialog` (рис. 9.4) представляет собой стандартное диалоговое окно **Открыть**, **Сохранить**, **Цвет**, **Шрифт**, **Печать** или **Справка**. Свойства компонента приведены в табл. 9.6. Тип окна определяет метод, обеспечивающий отображение диалога (табл. 9.7).

Для того чтобы компонент был доступен, надо подключить библиотеку Microsoft Common Dialog Control 6.0 (COMDLG32.OCX).

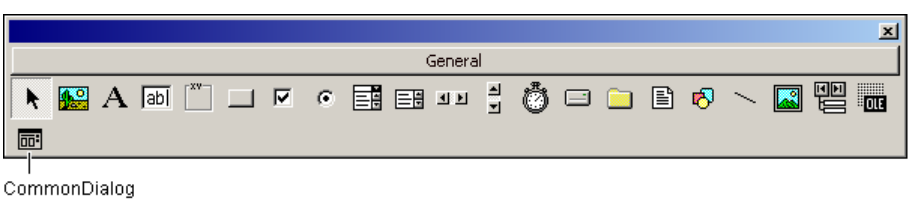


Рис. 9.4. Компонент `CommonDialog`

Таблица 9.6. *Свойства компонента* `CommonDialog`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`DialogTitle`	Заголовок окна
`FileName`	Полное имя файла, выбранного в диалоге **Открыть**. Если файл не был выбран, т. е. диалог завершен нажатием кнопки **Cancel**, значение свойства — пустая строка
`Filter`	Фильтр. Задает файлы, отображаемые в окнах (диалогах) **Открыть** и **Сохранить**. Если фильтр не задан, то отображаются все файлы
`FilterIndex`	Номер выбранного фильтра. Фильтры номеруются с 1
`Flags`	Флаги спецификации для диалоговых окон
`HelpFile`	Путь к help-файлу, который нужно отобразить

Таблица 9.7. *Методы отображения объекта* `CommonDialog`

Метод	Диалог
`ShowOpen`	Открыть
`ShowSave`	Сохранить
`ShowColor`	Цвет
`ShowFont`	Выбор шрифта
`ShowPrinter`	Печать
`ShowHelp`	Справка

DirListBox

Компонент `DirListBox` (рис. 9.5) отображает структуру указанного каталога. Свойства компонента приведены в табл. 9.8.

Рис. 9.5. Компонент `DirListBox`

Таблица 9.8. *Свойства компонента* `DirListBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Font`	Шрифт, используемый для отображения названий каталогов и подкаталогов
`BackColor`	Цвет фона области вывода списка каталогов. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`ForeColor`	Цвет шрифта, используемого для отображения названий каталогов и подкаталогов
`List`	Элементы указанного каталога (подкаталога) — массив строк. В элемент массива записывается полный путь доступа к подкаталогу
`ListCount`	Количество подкаталогов указанного каталога
`ListIndex`	Номер элемента, выбранного в списке каталогов и подкаталогов. Нумерация подкаталогов начинается с нуля. Если в списке выбран каталог, структура которого отображается в компоненте `Dir1`, то значение свойства равно –1. Если в структуре выбрать каталог на один уровень выше (при этом список должен отображаться), то значение будет равно –2. Значение свойства уменьшаться на `1` при каждом переходе вверх по дереву каталогов на один уровень
`Path`	Путь к каталогу
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

DriveListBox

Компонент `DriveListBox` (рис. 9.6) является выпадающим списком, отображающим диски компьютера. Свойства компонента приведены в табл. 9.9.

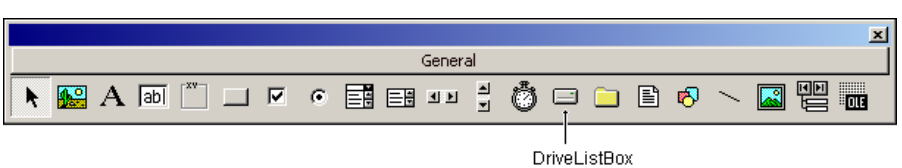


Рис. 9.6. Компонент `DriveListBox`

Таблица 9.9. *Свойства компонента* `DriveListBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Font`	Шрифт, используемый для отображения названий дисков
`BackColor`	Цвет фона области вывода списка дисков. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`ForeColor`	Цвет шрифта, используемого для отображения имен дисков
`List`	Элементы списка — массив строк. В нем находятся названия дисков (например, C:)
`ListCount`	Количество дисков компьютера
`ListIndex`	Номер элемента, выбранного в списке дисков. Если ни один из элементов списка не был выбран, то значение свойства равно –1. Нумерация дисков начинается с нуля
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

FileListBox

Компонент `FileListBox` (рис. 9.7) отображает список файлов указанного каталога. Свойства компонента приведены в табл. 9.10.

Рис. 9.7. Компонент `FileListBox`

Таблица 9.10. *Свойства компонента* `FileListBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Font`	Шрифт, используемый для отображения названий файлов
`BackColor`	Цвет фона области вывода списка файлов. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`ForeColor`	Цвет шрифта, используемого для отображения списка файлов указанного каталога
`List`	Элементы указанного каталога (список файлов) — массив строк. В элемент массива записывается название файла (*имя.расширение*)
`ListCount`	Количество файлов в указанном каталоге
`ListIndex`	Номер элемента, выбранного в списке файлов. Нумерация файлов начинается с нуля
`Path`	Путь к каталогу
`Archive`	Свойство определяет, отображаются ли в списке файлы с установленным атрибутом "архивный"
`Hidden`	Свойство определяет, отображаются ли в списке файлы с установленным атрибутом "скрытый"
`ReadOnly`	Свойство определяет, отображаются ли в списке файлы с установленным атрибутом "только чтение"
`System`	Свойство определяет, отображаются ли в списке файлы с установленным атрибутом "системный"
`Normal`	Свойство определяет, отображаются ли в списке файлы с установленным атрибутом "архивный", "только чтение", без атрибутов или с всевозможными комбинациями этих атрибутов
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

Image

Компонент `Image` (рис. 9.8) обеспечивает отображение иллюстраций. Отличается от компонента `PictureBox` тем, что поверхность компонента недоступна методам вычерчивания графических примитивов (на поверхности компонента рисовать нельзя). Свойства компонента приведены в табл. 9.11.

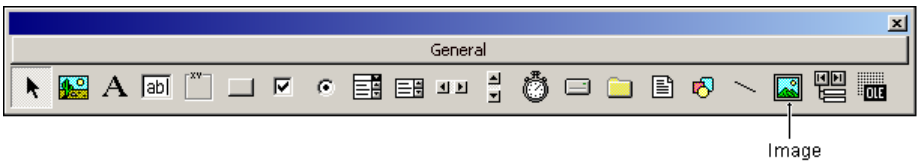


Рис. 9.8. Компонент `Image`

Таблица 9.11. *Свойства компонента* `Image`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Picture`	Картинка (объект `BitMap`), отображаемая в поле компонента. Задать картинку можно во время разработки формы или загрузить из файла во время работы программы (функция `LoadPicture`)
`BorderStyle`	Стиль границы компонента: `FixedSingle` (`1`) — тонкая линия; `None` (`0`) — границы нет
`Stretch`	Признак масштабирования (сжатия или растяжения) иллюстрации в соответствии с реальным размером компонента. Значение `True` — иллюстрация масштабируется в соответствии с размером компонента (если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена). Значение `False` — масштабирование не выполняется
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

Label

Компонент Label (рис. 9.9) предназначен для вывода текста на поверхность формы. Свойства компонента (табл. 9.12) определяют вид и расположение текста на поверхности формы.

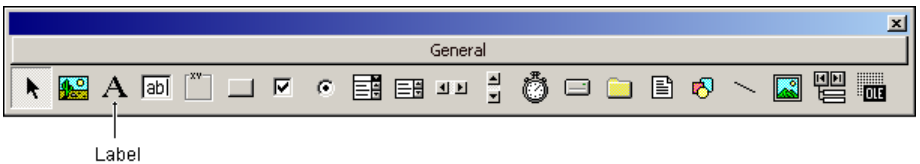


Рис. 9.9. Компонент Label

Таблица 9.12. *Свойства компонента* Label

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
Caption	Отображаемый текст
Left	Расстояние от левой границы поля вывода до левой границы формы
Top	Расстояние от верхней границы поля вывода до верхней границы формы
Height	Высота поля вывода
Width	Ширина поля вывода
AutoSize	Признак того, что размер поля определяется его содержимым
WordWrap	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку
Alignment	Задает способ выравнивания текста внутри поля. Текст может быть выровнен по левому краю — LeftJustify (0), по центру — Center (2) или по правому краю — RightJustify (1)
Font	Шрифт, используемый для отображения текста. Уточняющие свойства Name и Size задают шрифт и размер символов
ForeColor	Цвет символов. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы

Таблица 9.12 *(окончание)*

Свойство	Описание
`BackColor`	Цвет фона области вывода текста. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`BackStyle`	Управляет отображением фона области вывода текста. Область вывода текста может быть закрашена (`Opaque`) или быть "прозрачной" (`Transparent`). Если значение свойства равно `Opaque`, то цвет закраски области определяет значение свойства `BackColor`
`Visible`	Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)

Line

Компонент `Line` (рис. 9.10) — графический объект "линия". Компонент может использоваться только в качестве декоративного элемента, т. к. он не может воспринимать события. Свойства компонента приведены в табл. 9.13.

Рис. 9.10. Компонент `Line`

Таблица 9.13. *Свойства компонента* `Line`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`BorderColor`	Цвет линии
`BorderStyle`	Вид линии: • `Solid` (`0`) — сплошная; • `Dash` (`1`) — пунктирная; • `Dot` (`2`) — линия из точек; • `Dash-Dot` (`3`) — линия "точка-тире"; • `Dash-Dot-Dot` (`4`) — линия "тире-точка-точка"; • `Transparent` (`5`) — прозрачная линия

Таблица 9.13 *(окончание)*

Свойство	Описание
DrawWidth	Толщина линии
X1	Горизонтальная координата точки начала линии
Y1	Вертикальная координата точки начала линии
X2	Горизонтальная координата точки конца линии
Y2	Вертикальная координата точки конца линии
Visible	Позволяет скрыть компонент (значение — False) или сделать его видимым (значение — True)

ListBox

Компонент ListBox (рис. 9.11) представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. 9.14.

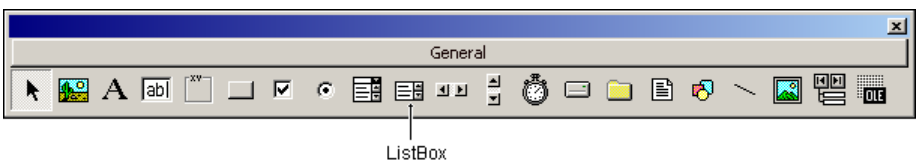


Рис. 9.11. Компонент ListBox

Таблица 9.14. *Свойства компонента* ListBox

Свойство	Описание
Name	Имя компонента. В программе используется для доступа к компоненту и его свойствам
BackColor	Цвет фона области вывода списка. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
ForeColor	Цвет, используемый для отображения элементов списка
List	Элементы списка — массив строк
ListCount	Количество элементов списка
Sorted	Признак необходимости автоматической сортировки (значение свойства — True) списка после добавления очередного элемента
ListIndex	Номер выбранного элемента списка (нумерация элементов списка начинается с нуля)

Таблица 9.14 *(окончание)*

Свойство	Описание
`MultiSelect`	Позволяет сделать возможным множественный выбор из списка. Если значение этого свойства — `None` (`0`), то выбрать несколько элементов из списка нельзя. При значении, равном `Simple` (`1`), каждый элемент, на котором произвелся щелчок, становится выбранным. Отмена выбора производится с помощью повторного щелчка мыши или с помощью клавиши <Пробел>. Если значение свойства равно `Extended` (`2`), то множественный выбор осуществляется с помощью мыши и клавиши <Shift> или <Ctrl> (щелчок кнопкой мыши на элементе списка при нажатой клавише <Shift> помечает элемент как выбранный, повторный щелчок отменяет выбор)
`Style`	Стиль (вид) списка. Если значение свойства равно `CheckBox` (`1`), то перед каждым элементом списка отображается компонент `CheckBox`. При этом для выбора элемента из списка нужно установить соответствующий флажок. При значении свойства равном `Standard` (`0`) список имеет стандартный вид
`Left`	Расстояние от левой границы списка до левой границы формы
`Top`	Расстояние от верхней границы списка до верхней границы формы
`Height`	Высота поля списка
`Width`	Ширина поля списка
`Font`	Шрифт, используемый для отображения элементов списка
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

MMControl

Компонент `MMControl` (рис. 9.12 и 9.13) обеспечивает воспроизведение звуковых файлов и видеофайлов. Свойства компонента приведены в табл. 9.15.

Для того чтобы компонент был доступен, надо подключить библиотеку Microsoft Multimedia Control 6.0 (MCI32.OCX).

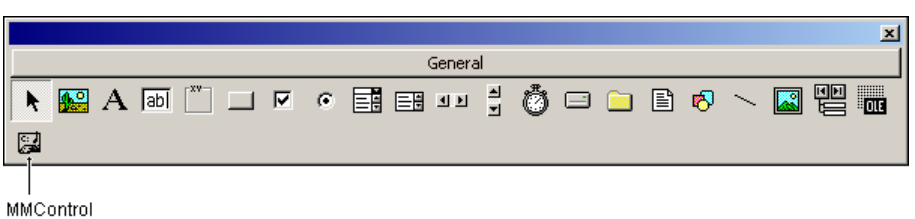


Рис. 9.12. Значок компонента `MMControl`

Рис. 9.13. Кнопки компонента `MMControl`

Таблица 9.15. *Свойства компонента* `MMControl`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`DeviceType`	Тип устройства: `CDAudio` — проигрыватель звуковых CD; `WaveAudio` — проигрыватель WAV-файлов; `AVIVideo` — проигрыватель AVI-файлов и др.
`Command`	Команда
`TimeFormat`	Формат измерения времени
`FileName`	Имя файла, который нужно воспроизвести
`PlayEnabled`	Делает кнопку **Play** данного компонента доступной (значение — `True`) или недоступной (значение — `False`). Аналогичное свойство есть для всех остальных кнопок компонента
`PlayVisible`	Позволяет скрыть кнопку **Play** (значение — `False`) компонента или сделать ее видимой (значение — `True`). Аналогичное свойство есть у остальных кнопок компонента
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)

OptionButton

Компонент `OptionButton` (рис. 9.14) представляет зависимую кнопку — переключатель, состояние которого определяется состоянием других переключателей группы. Свойства компонента приведены в табл. 9.16.

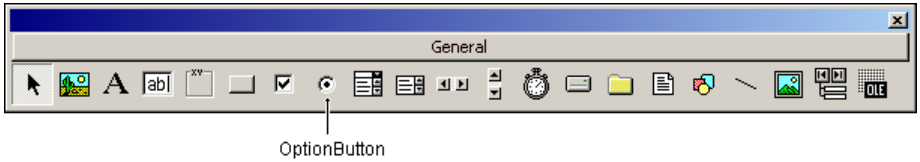


Рис. 9.14. Компонент `OptionButton`

Таблица 9.16. *Свойства компонента* `OptionButton`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Caption`	Текст, который находится справа от кнопки
`Value`	Состояние кнопки. Если кнопка выбрана, то значение свойства — `True`, если кнопка не выбрана, значение свойства — `False`
`Tag`	Visual Basic не использует это свойство. Пользователь может использовать его по своему назначению
`Left`	Расстояние от левой границы флажка до левой границы формы
`Top`	Расстояние от верхней границы флажка до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента (флажок и область для пояснительного текста)
`Font`	Шрифт, используемый для отображения поясняющего текста
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

PictureBox

Компонент `PictureBox` (рис. 9.15) обеспечивает отображение графики. Изображение можно загрузить из файла или сформировать из графических примитивов (нарисовать) во время работы программы. Свойства компонента приведены в табл. 9.17.

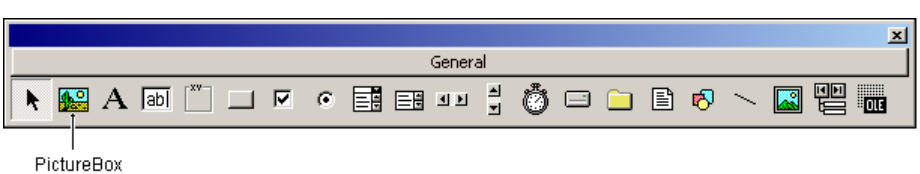


Рис. 9.15. Компонент `PictureBox`

Таблица 9.17. *Свойства компонента* `PictureBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Picture`	Картинка (объект `BitMap`), отображаемая в поле компонента. Задать картинку можно во время разработки формы или загрузить из файла во время работы программы (функция `LoadPicture`)
`AutoSize`	Свойство разрешает (`True`) или запрещает (`False`) автоматическое изменение размера компонента (области вывода иллюстрации) в соответствии с размером картинки, загруженной в компонент
`BorderStyle`	Стиль границы компонента. Если значение свойства — `FixedSingle` (`1`), то граница стандартная (тонкая линия), если — `None` (`0`), то граница не отображается
`BackColor`	Цвет фона компонента. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
`Font`	Шрифт, которым метод `Print` выводит текст
`ForeColor`	Для метода `Print` задает цвет символов, для методов вычерчивания графических примитивов (объектов) — цвет линий
`FillColor`	Задает цвет закраски внутренних областей графических примитивов (объектов) вычерчиваемых в поле (на поверхности) компонента
`FillStyle`	Стиль закраски графических объектов, вычерчиваемых в поле компонента соответствующими методами: • `Solid` (`0`) — сплошная заливка; • `Transparent` (`1`) — закраска "прозрачным" цветом; • `HorizontalLine` (`2`) — горизонтальная штриховка; • `VerticalLine` (`3`) — вертикальная штриховка. Цвет линий штриховки определяет свойство `FillColor`
`DrawStyle`	Вид контура графических объектов, вычерчиваемых в поле компонента соответствующими методами: • `Solid` (`0`) — сплошная линия; • `Dash` (`1`) — пунктирная линия; • `Dot` (`2`) — линия из точек; • `Dash-Dot` (`3`) — линия вида "точка-тире"; • `Dash-Dot-Dot` (`4`) — линия вида "тире-точка-точка"; • `Transparent` (`5`) — прозрачная линия
`DrawWidth`	Толщина линий для графических объектов

Таблица 9.17 *(окончание)*

Свойство	Описание
`ScaleWidth`	Ширина рабочей области компонента, т. е. без учета ширины левой и правой границ. Единицу измерения задает свойство `ScaleMode`
`ScaleHeight`	Высота рабочей области компонента, т. е. без учета ширины нижней и верхней границ компонента. Единицу измерения задает свойство `ScaleMode`
`ScaleMode`	Задает единицу измерения размеров компонента и объектов на его поверхности. Значение этого свойства не влияет на единицы измерения свойств `Width` и `Height` (их значения измеряются в твипах)
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

ProgressBar

Компонент `ProgressBar` (рис. 9.16) представляет собой индикатор, который обычно используется для наглядного представления протекания процесса, например обработки (копирования) файлов, загрузки информации из сети и т. п. Свойства компонента `ProgressBar` приведены в табл. 9.18.

Для того чтобы компонент был доступен, надо подключить библиотеку Microsoft Windows Common Controls 6.0 SP4 (MSCOMCTL.OCX).

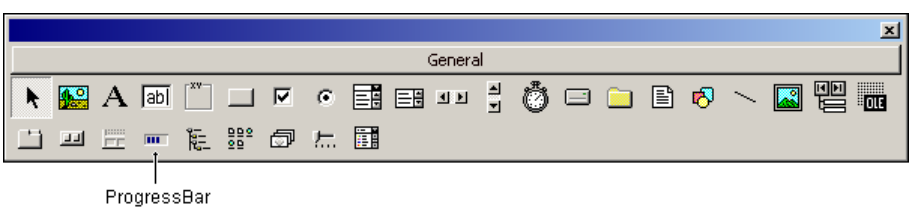


Рис. 9.16. Компонент `ProgressBar`

Таблица 9.18. *Свойства компонента* `ProgressBar`

Свойство	Описание
`Value`	Значение, которое отображается в поле компонента (на индикаторе) в виде полосы или ряда прямоугольников. Длина полосы (количество прямоугольников) пропорционально значению свойства `Value`. Свойство доступно только во время работы программы
`Min`	Минимально допустимое значение свойства `Value`
`Max`	Максимально допустимое значение свойства `Value`
`Scrolling`	Вид индикатора. Индикатор может представлять собой последовательность прямоугольников (`0` — `ccScrollingStandard`) или полосу (`1` — `ccScrollingSmooth`)
`Appearance`	Вид компонента: вровень с поверхностью формы (`0` — `ccFlat`), в стиле 3D (`1` — `cc3D`)
`Border`	Граница компонента: есть (`1` — `ccFixedSingle`), нет (`0` — `ccNone`)

Shape

Компонент `Shape` (рис. 9.17) — графический объект (прямоугольник, овал (круг), прямоугольник со скругленными углами), который можно поместить на поверхность формы. Компонент может использоваться только в качестве декоративного элемента, т. к. он не может воспринимать события. Свойства компонента приведены в табл. 9.19.

Рис. 9.17. Компонент `Shape`

Таблица 9.19. *Свойства компонента* `Shape`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам

Таблица 9.19 *(продолжение)*

Свойство	Описание
Shape	Вид геометрической фигуры: • Rectangle (0) — прямоугольник; • Square (1) — квадрат; • Oval (2) — овал; • Circle (3) — круг; • RoundedRectangle (4) — прямоугольник со скругленными углами; • RoundedSquare(5) — квадрат со скругленными углами
BackColor	Цвет фона компонента. Цвет можно задать, выбрав его из палитры цветов или указав привязку к текущей цветовой схеме операционной системы
BackStyle	Стиль фона компонента: Transparent (0) — прозрачный, Opaque (1) — непрозрачный
BorderColor	Цвет границы объекта (контура геометрической фигуры)
BorderStyle	Вид контура объекта (геометрической фигуры): • Solid (0) — сплошная линия; • Dash (1) — пунктирная линия; • Dot (2) — линия из точек; • Dash-Dot (3) — линия "точка-тире"; • Dash-Dot-Dot (4) — линия "тире-точка-точка"; • Transparent (5) — прозрачная линия
DrawWidth	Толщина линии контура объекта (геометрической фигуры)
FillColor	Цвет закраски внутренней области объекта (геометрической фигуры)
FillStyle	Стиль закраски внутренней области объекта (геометрической фигуры): • Solid (0) — сплошная заливка; • Transparent (1) — "прозрачный" цвет (нет закраски); • HorizontalLine (2) — горизонтальная штриховка; • VerticalLine (3) — вертикальная штриховка. Цвет линий штриховки определяется значением свойства FillColor
Left	Расстояние от левой границы компонента до левой границы формы
Top	Расстояние от верхней границы компонента до верхней границы формы

Таблица 9.19 *(окончание)*

Свойство	Описание
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение — `False`) или сделать его видимым (значение — `True`)

StatusBar

Компонент `StatusBar` (рис. 9.18) представляет собой полосу вывода служебной информации (строку состояния).

Для того чтобы компонент был доступен, надо подключить библиотеку Microsoft Windows Common Controls 6.0 SP4 (MSCOMCTL.OCX).

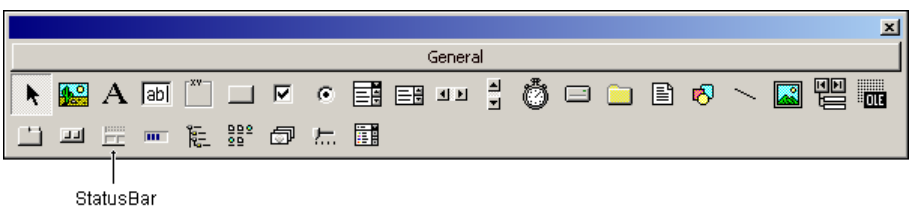


Рис. 9.18. Компонент `StatusBar`

Чтобы полоса состояния была разделена на области, в нее надо добавить соответствующее количество *панелей*. Для этого надо раскрыть окно свойств компонента `StatusBar` (в контекстном меню компонента выбрать команду **Properties**) и на вкладке **Panels** щелкнуть на кнопке **Insert Panel** столько раз, на сколько областей должна быть разделена строка состояния. Свойства компонента `StatusBar` приведены в табл. 9.20, панели компонента `StatusBar` — в табл. 9.21.

Таблица 9.20. *Свойства компонента* `StatusBar`

Свойство	Описание
`Style`	Стиль (вид) области состояния. Панель может быть обычной (`0` — `sbrNormal`) или простой (`1` — `sbrSimple`). Обычная панель разделена на области, простая — представляет собой одну-единственную область
`SimpleText`	Текст, который отображается в панели состояния, если панель простая (значение свойства `Style` равно `sbrSimple`)

Таблица 9.21. *Свойства панели (объекта* `StatusBarPanel`*)*

Свойство	Описание
`Style`	Тип информации, отображаемый в панели. Например: текст (`0` — `sbrText`), время (`5` — `sbrTime`), дата (`6` — `sbrDate`), состояние клавиатуры (`1` — `sbrCaps`, `2` — `sbrNum`, `3` — `sbrIns`)
`Text`	Текст, отображаемый в панели (если значение свойства `Style` равно `sbrText`)
`Alignment`	Размещение текста в панели. Текст может быть прижат к левой границе (`0` — `sbrLeft`), к правой (`2` — `sbrRight`) или расположен по центру (`1` — `sbrCenter`)
`Picture`	Картинка, отображаемая в панели
`AutoSize`	Признак автоматического изменения размера панели. Если значение свойства равно `2` (`sbrContents`), то ширина панели зависит от ее содержания (длины текста). Если значение свойства равно `1` (`sbrSpring`), то ширина панели устанавливается такой, чтобы находящаяся справа другая панель была прижата к правой границе окна. Если справа панели нет, то ширина устанавливается такой, чтобы правая граница панели была прижата к правой границе формы

TextBox

Компонент `TextBox` (рис. 9.19) представляет собой поле ввода/редактирования текста. Свойства компонента приведены в табл. 9.22.

Рис. 9.19. Компонент `TextBox`

Таблица 9.22. *Свойства компонента* `TextBox`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам, в частности — для доступа к тексту, введенному в поле редактирования
`Text`	Текст, находящийся в поле редактирования

Таблица 9.22 *(окончание)*

Свойство	Описание
Left	Расстояние от левой границы компонента до левой границы формы
Top	Расстояние от верхней границы компонента до верхней границы формы
Height	Высота поля
Width	Ширина поля
Font	Шрифт, используемый для отображения содержимого поля
Locked	Используется для ограничения возможности изменить текст в поле редактирования. Если значение свойства равно False, то текст в поле редактирования изменить нельзя
MultiLine	Делает возможным многострочное отображение текста
ScrollBars	Управляет отображением полос прокрутки: • Vertical — только вертикальная полоса прокрутки; • Horizontal — только горизонтальная полоса прокрутки; • Both — вертикальная и горизонтальная полосы прокрутки; • None — без полос прокрутки
Visible	Позволяет скрыть компонент (False) или сделать его видимым (True)

Timer

Компонент Timer (рис. 9.20) обеспечивает генерацию последовательности событий Timer. Свойства компонента приведены в табл. 9.23.

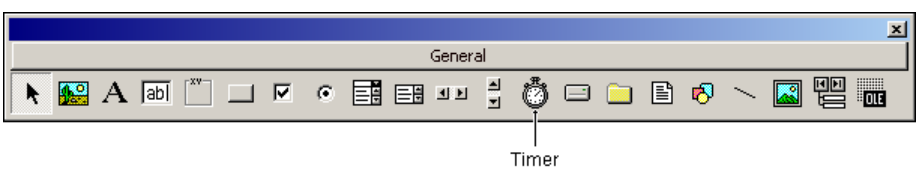


Рис. 9.20. Компонент Timer

Таблица 9.23. *Свойства компонента* Timer

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту и его свойствам

Таблица 9.23 *(окончание)*

Свойство	Описание
`Interval`	Период генерации события `Timer`. Задается в миллисекундах
`Enabled`	Разрешение работы (запуск/остановка). Разрешает (значение свойства — `True`) или запрещает (значение свойства — `False`) генерацию события `Timer`

UpDown

Компонент `UpDown` (рис. 9.21) представляет собой две кнопки, используя которые можно изменить (увеличить или уменьшить) значение внутренней переменной-счетчика. Компонент `UpDown` можно связать с компонентом `Label` или `TextBox` и использовать его в качестве индикатора значения переменной-счетчика. Свойства компонента `UpDown` приведены в табл. 9.24.

Для того чтобы компонент был доступен, надо подключить библиотеку Microsoft Windows Common Controls-2 6.0 (MSCOMCT2.OCX).

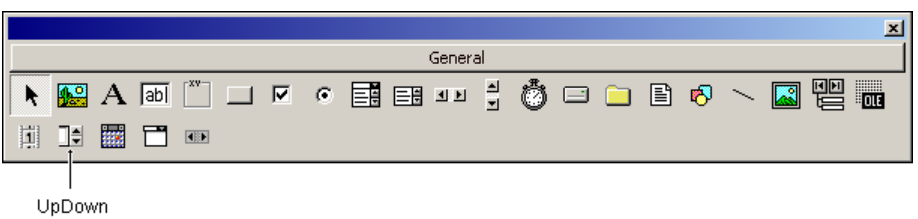


Рис. 9.21. Компонент `UpDown`

Таблица 9.24. *Свойства компонента* `UpDown`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Value`	Счетчик. Значение счетчика изменяется в результате щелчка на кнопке **Up** (Увеличение) или **Down** (Уменьшение)
`Increment`	Величина изменения значения переменной-счетчика
`Max`	Верхняя граница изменения значений переменной-счетчика `Value`
`Min`	Нижняя граница изменения значений переменной-счетчика `Value`

***Таблица 9.24** (окончание)*

Свойство	Описание
`BuddyControl`	Компонент, который используется в качестве индикатора значения переменной-счетчика (например, `Label` или `TextBox`)
`BuddyProperty`	Свойство компонента, указанного в `BuddyControl`, обеспечивающее индикацию значения переменной-счетчика (`Caption`, если индикатор — компонент `Label`)
`AutoBuddy`	Автоматическое определение свойства компонента-индикатора, используемого для индикации состояния переменной-счетчика. Если в качестве индикатора используется компонент `Label`, то автоматически в качестве значения свойства `BuddyProperty` устанавливается `Caption`
`SyncBuddy`	Синхронизация (`True`) изменением значения `Value` и значения свойства компонента-индикатора
`Orientation`	Ориентация кнопок (стрелок на кнопках) компонента: `OrientationVertical` (`0`) — по вертикали (вверх, вниз); `OrientationHorizontal` (`1`) — по горизонтали (влево, вправо)
`Wrap`	Если значение свойства равно `False`, то при достижении максимального значения переменной `Value` ее значение не изменяется при последующих нажатиях кнопки **Up**. Аналогично для кнопки **Down**. Если значение свойства равно `True`, при аналогичных действиях, максимальное значение переменной `Value` изменяется на минимальное и наоборот
`Enabled`	Доступность (значение свойства — `True`) или недоступность компонента (значение свойства — `False`)
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота компонента
`Width`	Ширина компонента
`Visible`	Позволяет скрыть компонент (значение свойства — `False`) или сделать его видимым (значение свойства — `True`)

Графика

В этом разделе приведено описание методов, обеспечивающих вывод графики на поверхность формы или компонента `PictureBox`.

Инструкции вызова метода или обращения к свойству в общем виде выглядят так:

Объект.*Метод*

Объект.*Свойство*

Следует обратить внимание, что объект в инструкции вызова метода или обращения к свойству объекта можно не указывать. Если объект не указан, то используется объект "по умолчанию" — форма.

В описании изменяемые параметры методов выделены курсивом. Необязательные параметры заключены в квадратные скобки.

Circle

[*объект*.]Circle [*Step*] (*x*,*y*), *Radius*, [*Color*, *Start*, *End*, *Aspect*]

Метод Circle позволяет нарисовать окружность, эллипс (если задан параметр *Aspect*) или дугу. Цвет контура определяет свойство ForeColor графической поверхности, на которой метод рисует (если не задан параметр Color).

Вид контура (толщина стиль линии) определяют соответственно свойства DrawWidth и DrawStyle графической поверхности, на которой метод рисует.

Параметр *Step* показывает, что реальные координаты отсчитываются от указателя текущей точки.

Параметр *Radius* задает радиус окружности или, если задан параметр *Aspect*, больший радиус эллипса.

Параметр *Color* задает цвет контура (по умолчанию цвет контура определяет свойство ForeColor графической поверхности).

При вычерчивании дуги параметр *Start* задает угловую координату точки начала дуги, а параметр *End* — угловую координату точки конца дуги. Дуга вычерчивается от точки *Start* против часовой стрелки. Угловые координаты измеряются в радианах. Для пересчета величины угла из градусов в радианы можно воспользоваться формулой: r = 2*p/(a/360), где r — величина угла в радианах, a — величина угла в градусах, p — число π (3,1415926).

Параметр *Aspect* задает вид эллипса. Если значение параметра *Aspect* меньше 1, то эллипс получается путем сжатия окружности по вертикали (если значение параметра 0, то эллипс вырождается в вертикальную линию). Если значение параметра *Aspect* больше 1, то эллипс получается путем сжатия окружности по горизонтали.

Пример:

```
pi = 3.1415926  ' число пи
ScaleMode = 3   ' единица измерения координат — пиксел

Circle (100, 50), 30 ' окружность
Circle (50, 50), 30, RGB(255, 0, 0)
Circle (150, 50), 30, , 0, pi ' дуга

Circle (150, 50), 30, , 0, pi / 2 ' дуга
Circle (150, 50), 30, RGB(255, 0, 0), pi / 2, 0 ' дуга

Circle (100, 120), 30, , , , 0.5 ' эллипс
Circle (100, 120), 30, , , , 2   ' эллипс
```

Line

[*Объект*.]Line [*Step*] (*x1*, *y1*) [*Step*]-(*x2*, *y2*), [*Color*], [B][F]

Метод Line рисует линию или прямоугольник (если указан параметр B).

Параметры *x1*, *y1* и *x2*, *y2* задают координаты точки начала и конца линии или находящихся на одной диагонали углов прямоугольника (если указан параметр B).

Цвет линии (контура прямоугольника) определяет свойство ForeColor графической поверхности (если не указан параметр Color).

Толщину и вид определяют соответственно свойства DrawWidth и DrawStyle объекта, на поверхности которого метод рисует.

Параметр *Step* показывает, что реальные координаты отсчитываются от указателя текущей точки.

Параметр *Color* задает цвет линии или, если задан параметр B, границы прямоугольника (по умолчанию цвет контура определяет свойство ForeColor графической поверхности).

Параметр B задает, что надо нарисовать прямоугольник.

Параметр F задает, что прямоугольник должен быть закрашен тем же цветом, что и граница.

Пример:

```
ScaleMode = 3       ' единица измерения координат — пиксел
Line (10, 20)-(100, 20)              ' линия
Line (10,30)-(100,30), vbRed         ' красная линия
Line (10, 40)-(100, 50), vbBlack, B  ' контур прямоугольника
Line (10, 80)-(100, 90), vbBlack, BF ' прямоугольник
```

LoadPicture

Объект.Picture = LoadPicture(*ФайлИллюстрации*)

Функция LoadPicture позволяет загрузить из файла (BMP, GIF, JPG) и отобразить в поле компонента PictureBox или Image иллюстрацию.

Пример:

```
Picture1.Picture = LoadPicture("D:\Images\image02.gif")
```

LoadResPicture

Объект.Picture = LoadResPicture(*ИдентификаторРесурса*, vbResBitmap)

Функция LoadResPicture позволяет загрузить из ресурса и отобразить в поле компонента PictureBox или Image иллюстрацию. Параметр *ИдентификаторРесурса* задает битовый образ, загружаемый из ресурса.

Пример:

```
Picture1.Picture = LoadResPicture(102, vbResBitmap)
```

PaintPicture

Pic1.PaintPicture *Pic2*,*x1*,*y1*,*[w1]*,*[h1]*,*[x2]*,*[y2]*,*[w2]*,*[h2]*,*[OpCode]*

Метод PaintPicture копирует иллюстрацию (или ее часть) с поверхности объекта *Pic2* (типа Picture, Image или StdPicture) на поверхность объекта *Pic1* (типа Picture). Параметры *x1*, *y1*, *w1*, *h1* задают область, куда копируется иллюстрация, параметры *x2*, *y2*, *w2*, *h2* — откуда.

Пример:

```
Picture1.PaintPicture 0,0 Picture2

' копировать фрагмент рисунка Image1 на поверхность Picture1
Picture1.PaintPicture 0,0 Image1, 15,0,16,16

Dim aPicture As StdPicture
Set aPicture = LoadPicture("D:\Temp\image06.gif")
Picture1.PaintPicture aPicture, 0, 0
```

Print

*[объект.]*Print *String*

Метод Print выводит на поверхность объекта строку *String* от текущей точки (узнать координаты текущей точки можно, обратившись к свойствам

CurrentX и CurrentY). Шрифт определяется свойством Font графической поверхности (например, формы), на которую выводится текст, цвет символов — свойством ForeColor той же поверхности.

Пример:

```
Font.Name = "Arial"
Font.Size = 12
ForeColor = RGB(0, 0, 255)
ScaleMode = 3 ' координаты отсчитывать в пикселах
CurrentX = 10
CurrentY = 20
Print "Microsoft Visual Basic"
```

PSet

Объект.PSet (*x*,*y*)

Метод PSet рисует точку на поверхности графического объекта. Цвет и размер точки определяют соответственно свойства ForeColor и DrawWidth той графической поверхности, на которой рисует метод.

Пример:

```
Form1.Forecolor = RGB(56,176,222) ' цвет "Осеннее небо"
Form1.DrawWidth = 2
Form1.PSet(10,20)
```

RGB

RGB(*r*,*g*,*b*)

Функция RGB возвращает код цвета. Параметры r, g и b задают долю красной (r — red), зеленой (g — green) и синей (b — blue) составляющих. Диапазон изменения параметров *r*, *g*, *b* — от 0 до 255.

В табл. 9.25 приведены значения параметров *r*, *g*, *b* и указан цвет, соответствующий сочетанию значений параметров.

Таблица 9.25. *RGB-кодировка*

Цвет	r	g	b
Аквамарин	112	219	147
Багряный	140	23	23
Васильковый	66	66	111

Таблица 9.25 *(продолжение)*

Цвет	**r**	**g**	**b**
Весенне-зеленый, средний	127	255	0
Бирюзовый	173	234	234
Бирюзовый, средний	112	219	219
Бледно-зеленый	143	188	143
Бронзовый	140	120	83
Бронзовый-2	166	125	61
Весенне-зеленый	0	255	127
Голубой	35	107	142
Голубой кадет	95	159	159
Древесный темный	133	94	66
Древесный, светлый	233	194	166
Древесный, средний	166	128	100
Дымчато-серый	84	84	84
Желтовато-зеленый	153	204	50
Зеленовато-желтый	147	219	112
Зеленовато-медный	82	127	118
Зеленовато-медный, темный	74	118	110
Зеленый морской, средний	66	111	66
Золотарниковый	219	219	112
Золотарниковый, средний	234	234	174
Золотой	205	127	50
Индейский красный	78	47	47
Кварц	217	217	243
Кирпич	142	35	35
Коралловый	255	127	0
Коричневато-зеленый	50	205	50
Коричневый	166	42	42
Латунный	181	166	66
Лесной зеленый	35	142	35

Таблица 9.25 *(продолжение)*

Цвет	**r**	**g**	**b**
Лиловый	153	50	205
Мандариновый (оранжевый)	228	120	51
Медный	184	115	51
Морской волны	35	142	104
Насыщенный синий	89	89	171
Небесно-голубой	50	153	204
Неоновый розовый	255	110	199
Новый полуночно-синий	0	0	156
Оливково-зеленый, темный	79	79	47
Оранжево-красный	255	36	0
Оранжевый	255	127	0
Осеннее небо	56	176	222
Острый розовый	255	28	174
Охотничий зеленый	33	94	33
Охра	142	107	35
Очень светло-серый	205	205	205
Очень темно-коричневый	92	64	51
Перванш	0	127	255
Перванш, средний	127	0	255
Перванш, темный	107	35	142
Полевой шпат	209	146	117
Полуночно-голубой	47	47	79
Прохладный медный	217	135	25
Пшеничный	216	216	191
Розовый	188	143	143
Розовый, с серовато-коричневым оттенком	133	99	99
Рыжевато-коричневый	219	147	112
Рыжевато-коричневый, новый	235	199	158
Рыжевато-коричневый, темный	151	105	79

Таблица 9.25 *(окончание)*

Цвет	**r**	**g**	**b**
Светло-голубой	143	143	189
Светло-лиловый	219	112	219
Светло-лиловый, средний	147	112	219
Светло-серый	168	168	168
Светло-синий	192	217	217
Серебряный	230	232	250
Серый	192	192	192
Сине-фиолетовый	159	95	159
Сливовый	234	173	234
Сомон	111	66	66
Средне-синий	50	50	205
Средний лесной зеленый	107	142	35
Средний аквамарин	50	205	153
Старое золото	207	181	59
Темно-бирюзовый	112	147	219
Темно-бордовый	142	35	107
Темно-зеленый	47	79	47
Темно-коричневый	92	64	51
Темно-пурпурный	135	31	120
Темно-серый	47	79	79
Темно-синий	35	35	142
Фиолетово-красный	204	50	153
Фиолетово-красный, средний	219	112	147
Фиолетовый	79	47	79
Хаки	159	159	95
Чертополох	216	191	216
Шоколадный	80	51	23
Шоколадный светлый	107	66	38
Яркий золотой	217	217	25

Функции

В описании функций необязательные параметры заключены в квадратные скобки.

Ввод и вывод

Ввод исходных данных может быть реализован с помощью функции `InputBox`, а вывод результата — с помощью функции `MsgBox`.

InputBox

`InputBox(`*`Prompt`*`[,`*`Title`*`][,`*`Default`*`][,`*`X`*`][,`*`Y`*`][,`*`HlpFile`*`,`*`Cnt`*`])`

Функция `InputBox` выводит на экран диалоговое окно, в поле редактирования которого пользователь может ввести исходные данные — строку символов. Значением функции является введенная строка.

Параметр *`Prompt`* задает строку-подсказку — сообщение, которое отображается в диалоговом окне.

Параметр *`Title`* задает текст заголовка окна. Если этот параметр не указан, то в заголовке будет имя приложения — программы, которая запрашивает данные.

Параметр *`Default`* (выражение строкового типа) задает текст, отображаемый в поле ввода (пользователь может ввести исходные данные путем редактирования этого текста). Если параметр не указан, то при появлении окна на экране поле ввода будет пустым.

Параметры *`X`* и *`Y`* задают положение окна ввода. Параметры задаются в твипах. Если параметры не указаны, окно будет выведено в центр экрана.

MsgBox

`MsgBox(`*`Prompt`*`[,`*`MsgType`* `[,`*`Title`*`][,`*`HlpFile`*`,`*`Cnt`*`])`

Функция `MsgBox` выводит на экран окно с сообщением. Значение функции — код кнопки, щелчком на которой пользователь закрыл окно.

Параметр *`Prompt`* (выражение строкового типа) задает текст сообщения.

Параметр *`MsgType`* (целого типа) задает тип сообщения и командные кнопки, которые отображаются в окне сообщения. Необходимое значение параметра *`MsgType`* можно вычислить по формуле: `Msg + Btn`, где `Msg` — тип сообщения, `Btn` — код кнопки (кнопок), которую надо отобразить в окне сообщения. Тип сообщения: `vbInformation` (`64`) — информационное, `vbCritical` (`16`) — сообщение об ошибке. Код кнопки (кнопок): `vbOKOnly` (`0`) — отображается кнопка **OK**; `vbOKCancel` (`1`) — отображаются кнопки **OK** и **Cancel**;

vbYesNo (4) — отображаются кнопки **Yes** и **No**. Если параметр не указан, в окне сообщения отображается только кнопка **OK**.

Параметр *Title* задает заголовок окна сообщения. Если этот параметр не указан, то в заголовке отображается имя приложения — программы, которая вывела сообщение.

Параметр *HlpFile* задает файл справочной информации, а параметр *Cnt* — номер раздела справочной информации. Чтобы получить доступ к справочной информации, пользователь должен нажать клавишу <F1>.

Математические функции

В табл. 9.26 приведены наиболее часто используемые математические функции.

Таблица 9.26. *Математические функции*

Функция	Значение
Abs(*n*)	Абсолютное значение (модуль) *n*
Sqr(*n*)	Квадратный корень *n*
Exp(*n*)	Экспонента *n*
Sgn(*n*)	Знак числа *n*. Если значение выражения *n* меньше нуля, то значение функции равно –1, если больше или равно 0, то значение функции равно 0
Rnd()	Случайное число в диапазоне от 0 до 1. Перед первым обращением к функции Rnd необходимо инициализировать генератор случайных чисел — вызвать функцию Randomize
Int(*n*)	Целая часть числа. Значение получается путем "отбрасывания" дробной части. Если *n* отрицательное, то значение функции — ближайшее отрицательное целое число, меньшее либо равное *n*. Например: Int(5.85)=5, Int(-5.85)=-6
Fix(*n*)	Функция отбрасывает дробную часть числа и возвращает целое значение. Для отрицательных чисел функция возвращает ближайшее отрицательное целое число, большее либо равное *n*. Например: Fix(5.85)=5, Fix(-5.85)=-5
IsNumeric(*s*)	Функция проверяет, является ли строка *s* изображением числа. Если *s* (или подстрока от первого символа) является изображением числа, то значение функции — True. Если строка не является изображением числа, то значение функции — False. Например: IsNumeric("5,85") возвращает True, IsNumeric("5,8 5") возвращает True, IsNumeric("hello") возвращает False

Таблица 9.26 *(окончание)*

Функция	Значение
`Log(n)`	Функция вычисляет натуральный логарифм (логарифм по основанию *e*). Десятичный логарифм можно вычислить по формуле: `Log(n)/Log(10)`
`Sin(α)`	Синус угла α
`Cos(α)`	Косинус угла α
`Tan(α)`	Тангенс угла α
`Atn(α)`	Арктангенс угла α

Величина угла тригонометрических функций (`Sin`, `Cos`, `Tan`, `Atn`) должна быть указана в радианах. Для преобразования величины угла из градусов в радианы можно воспользоваться формулой

```
(g * 3.1415926)/180,
```

где `g` — величина угла в градусах, `3.1415926` — число π.

Преобразование данных

В табл. 9.27 приведены наиболее часто используемые функции преобразования.

Таблица 9.27. *Функции преобразования данных*

Функция	Описание
`CDbl(s)`	Преобразует строку *s* в тип `Double`
`CInt(s)`	Преобразует строку *s* в тип `Integer`
`CLng(s)`	Преобразует строку *s* в тип `Long`
`CSng(s)`	Преобразует строку *s* в тип `Single`
`CStr(n)`	Преобразует числовое выражение *n* в строку. В строковом представлении дробных чисел в качестве десятичного разделителя используется символ, заданный в настройках операционной системы
`Str(n)`	Преобразует числовое выражение *n* в строку. В строковом представлении дробных чисел в качестве десятичного разделителя используется точка

Работа со строками

В табл. 9.28 приведены наиболее часто используемые функции, обеспечивающие операции со строками.

Таблица 9.28. *Функции работы со строками*

Функция	Описание
`Chr(Code)`	Функция `Chr` возвращает ANSI-символ, код которого указан в качестве параметра. Значение `Code` должно лежать в промежутке от 0 до 255
`Asc(Ch)`	Функция `Asc` возвращает код символа `Ch`
`InStr([Start,] String1, String2 [,Compare])`	Функция `InStr` выполняет поиск подстроки в строке. Просмотр осуществляется слева направо от первого или от заданного параметром `Start` символа. Значение функции — позиция подстроки в строке. Если искомой подстроки в строке нет, то значение функции — `0`. Необязательный параметр `Start` задает позицию символа в строке, от которого надо выполнить поиск. Если параметр не указан, то поиск начинается от первого символа. Параметр `Sring1` — строка, в которой ведется поиск. Параметр `Sring2` — подстрока, которую надо найти в строке `Sring1`. Параметр `Compare` задает режим сравнения строк: `TextCompare` (`1`) — текстовое сравнение, `BinaryCompare` (`0`) — побитовое сравнение. В режиме сравнения строк прописные и строчные символы считаются одинаковыми, а в режиме побитового сравнения — разными. Например, значение `InStr(1, "Hh", "h", vbBinaryCompare)` равно 2, а `InStr(1, "Hh", "h", vbTextCompare)` равно 1
`InStrRev(String1, String2 [, Start] [, Compare])`	Функция `InStrRev` выполняет поиск подстроки (в частном случае — символа) в строке. Просмотр осуществляется справа налево от последнего или от заданного параметром `Start` символа. Значение функции — позиция подстроки (символа) в строке (положение найденной подстроки отсчитывается от первого (левого) символа). Если искомой подстроки в строке нет, то значение функции равно `0`. Необязательный параметр `Start` задает позицию символа в строке, от которого надо выполнить поиск. Если параметр не указан, то поиск начинается от последнего символа. Параметр `Sring1` — строка, в которой ведется поиск. Параметр `Sring2` — подстрока (символ), который надо найти в строке `Sring1`

Таблица 9.28 *(продолжение)*

Функция	Описание
`InStrRev(String1, String2 [, Start] [, Compare])`	Параметр *Compare* задает режим сравнения строк: `TextCompare` (1) — текстовое сравнение, `BinaryCompare` (0) — побитовое сравнение. В режиме сравнения строк прописные и строчные символы считаются одинаковыми, а в режиме побитового сравнения — разными. Например, значение `InStrRev(1, "HHt", "ht", vbBinaryCompare)` равно 0, а `InStr(1, "Hht", "ht", vbTextCompare)` равно 2
`Len(S)`	Функция `Len` возвращает длину строки *S* (количество символов в строке)
`LCase(S)`	Функция `LCase` преобразует прописные символы строки *S* в строчные. Цифры и строчные буквы остаются без изменения
`UCase(S)`	Функция `UCase` преобразует строчные символы строки *S* в прописные. Цифры и прописные буквы остаются без изменения
`Left(S, L)`	Функция `Left` возвращает первые (отсчет от начала строки, т. е. слева) *L* символов строки *S*. Если значение *L* больше, чем количество символов в строке *S*, то значение функции — строка *S*
`Right(S, L)`	Функция `Right` возвращает последние (отсчет от конца строки, т. е. справа) `L` символов строки `S`. Если значение `L` больше, чем количество символов в строке `S`, то значение функции — строка `S`
`LTrim(S)`	Функция `LTrim` удаляет пробелы в начале строки
`RTrim(S)`	Функция `RTrim` удаляет пробелы в конце строки
`Trim(S)`	Функция `Trim` удаляет пробелы в начале и конце строки
`Mid(Str, Start [,Len])`	Значение функции `Mid` — подстрока, выделенная из строки *Str*. Параметр *Start* задает позицию подстроки, а *Len* — ее длину (число символов). Например, значение `Mid("Ms Visual Basic",4,6)` равно `Visual`
`Space(N)`	Значение функции — строка из *N* пробелов
`String(N, Ch)`	Значение функция — строка, состоящая из *N* символов *Ch*
`StrReverse(S)`	Функция "переворачивает" строку *S*. Например, значение `StrReverse("Hello")` равно `olleH`

***Таблица 9.28** (окончание)*

Функция	Описание
`Val(S)`	Функция `Val` возвращает числовое значение, изображением которого является строка `S`. Если в строке есть недопустимые символы, то будет обработана только часть строки, которую можно преобразовать в число. Пробелы игнорируются. При обработке строки, являющейся изображением дробного числа, правильным символом-разделителем является точка. Если строку преобразовать в число нельзя, то значение функции — `0`. Примеры: `Val(123 45) = 12345; Val(123,45) = 123; Val(123.45) = 123.45; Val("Text") = 0`

Работа с датами и временем

В табл. 9.29 приведены наиболее часто используемые функции манипулирования датами и временем.

***Таблица 9.29.** Функции работы с датами и временем*

Функция	Значение
`Date`	Текущая дата по системному календарю компьютера
`Time`	Текущее время по системным часам компьютера
`Now`	Текущее время и дата по системным часам и календарю компьютера
`Year(Date)`	Год для заданной даты
`Month(Date)`	Номер месяца для заданной даты
`MonthName(Month [,Abbreviate])`	Функция возвращает полное или сокращенное (3 символа) название месяца по его номеру (аргумент `Month` должен быть числом в диапазоне от 1 до 12; если значение аргумента `Abbreviate` равно `True`, то возвращается сокращенное название месяца, если `False` — возвращается полное название)
`Day(Date)`	День месяца (число от 1 до 31) для заданной даты
`Weekday(Date [,FirstDayOfWeek])`	Функция возвращает номер дня недели по дате `Date`. Аргумент `FirstDayOfWeek` определяет первый день недели: `vbUseSystem` (`0`) — используются значения системных установок, `Sunday` (`1`) — воскресенье, `Monday` (`2`) — понедельник и т. д.

Таблица 9.29 *(окончание)*

Функция	Значение

`WeekdayName(`*`Weekday`*`,`
`  `*`Abbreviate`*`,`
`  `*`FirstDayOfWeek`*`)`
	Функция возвращает полное или сокращенное (2 символа) название дня недели по его номеру. Аргумент *`Weekday`* должен быть числом в диапазоне от 1 до 7; если значение аргумента *`Abbreviate`* равно `True`, то возвращается сокращенное название месяца, если `False` — возвращается полное название. Аргумент *`FirstDayOfWeek`* определяет первый день недели: `vbUseSystem` (`0`) — используются значения системных установок, `Sunday` (`1`) — воскресенье, `Monday` (`2`) — понедельник и т. д.

`Hour(`*`Time`*`)`	Количество часов из выражения *`Time`*
`Minute(`*`Time`*`)`	Количество минут из выражения *`Time`*
`Second(`*`Time`*`)`	Количество секунд из выражения *`Time`*
`Timer()`	Количество секунд с точностью до одной сотой, прошедших от полуночи до текущего момента времени

Работа с файлами и каталогами

В табл. 9.30 приведены функции, обеспечивающие операции с файлами и каталогами.

Таблица 9.30. *Функции работы с файлами*

Функция	Описание

`Open` *`PathName`*
`  For` *`Mode`*
`  [Access` *`Am`*`]`
`  [Lock]`
`  As #`*`FileNumber`*
`  [Len =` *`reclen`*`]`
	Функция `Open` открывает файл для выполнения операций чтения/записи.
	Параметр *`PathName`* задает имя файла, к которому надо получить доступ.
	Параметр *`Mode`* задает режим доступа к файлу: `Input` — ввод данных (чтение), `Output` — вывод данных (запись), `Binary` — чтение/запись файла прямого доступа, `Random` — чтение/запись текстового двоичного файла.
	Параметр *`Am`* задает операции, разрешенные для открытого файла: `Read` (чтение), `Write` (запись), `Read Write` (чтение/запись)

Таблица 9.30 *(продолжение)*

Функция	Описание

`Open` *PathName*
`For` *Mode*
`[Access` *Am*`]`
`[Lock]`
`As #`*FileNumber*
`[Len =` *reclen*`]`
	Параметр *FileNumber* — номер файла (число в диапазоне от 1 до 511) используется в файловых операциях в качестве идентификатора файла.
	Параметр *reclen* задает длину записи файла (размер буфера), если файл открывается в режиме прямого доступа (`Binary`)

`Seek #`*FileNumber*`,` *Position*	Функция `Seek` устанавливает указатель текущей позиции для выполнения операции чтения/записи файла, открытого в режиме прямого доступа (`Binary`). Параметр *FileNumber* — идентификатор файла. Параметр *Position* задает позицию (номер байта или записи), которую надо прочитать или перезаписать
`Seek(#`*FileNumber*`)`	Функция `Seek` возвращает текущую позицию указателя чтения/записи для файла
`FreeFile[(`*Range*`)]`	Функция `FreeFile` возвращает число, которое можно использовать в качестве идентификатора файла (параметра *FileNumber*) в функции `Open`

`Get #`*FileNumber*`,`
`[`*RecNumber*`],`
VarName
	Функция `Get` считывает данные из файла:
	FileNumber — номер (идентификатор) файла,
	RecNumber — позиция (номер байта или номер записи, если файл открыт в режиме `Binary`), в которую надо установить указатель чтения перед выполнением операции, *VarName* — переменная, в которую надо поместить данные

`Put #`*FileNumber*`,`
`[RecNumber],`
VarName
	Функция `Put` записывает данные в файл:
	`FileNumber` — номер (идентификатор) файла,
	`RecNumber` — позиция (номер байта или номер записи, если файл открыт в режиме `Binary`), в которую надо установить указатель чтения перед выполнением операции, *VarName* — переменная, в которой находятся данные

`Line Input #`*FileNumber*`,` *VarName*	Функция `Line` с параметром `Input` считывает строку из файла *FileNumber* и записывает ее в переменную *VarName*. Чтение происходит до тех пор, пока не будет обнаружен символ "новая строка" (код 13)
`Input #`*FileNumber*`,` *VarList*	Функция `Input` считывает данные из файла. *FileNumber* — номер файла, *VarList* — список переменных, значение которых надо прочитать из файла. Пример: `Input #1, a,b,c`

Таблица 9.30 *(продолжение)*

Функция	Описание
`Input(`*`Number`*`, #`*`FileNumber`*`)`	Функция `Input` считывает символьные или байтовые данные из файла, открытого в режиме `Input` или `Binary`. *`Number`* — число считываемых символов или байтов, *`FileNumber`* — номер файла. Пример: `IDChar = Input(1, #1)`
`Print #`*`FileNumber`*`, [`*`OutputList`*`]`	Записывает в заданный параметром *`FileNumber`* текст. Параметр *`OutputList`* (список вывода) — список выражений символьного типа. Пример: `Print #1, "a="+Str(a),"b="+Str(b)`
`Write #`*`FileNumber`*`, [`*`OutputList`*`]`	Функция записывает данные в файл. *`OutputList`* — записываемые данные (список переменных). Символьные данные в файле будут заключены в кавычки. Пример: `Write #1, a, b`
`FileLen(`*`PathName`*`)`	Функция `FileLen` возвращает длину файла (в байтах)
`LOF(`*`FileNumber`*`)`	Функция `LOF` возвращает длину файла (в байтах)
`EOF(`*`FileNumber`*`)`	Функция `EOF` проверяет положение указателя чтения/записи. Значение функции равно `True`, если достигнут конец файла (прочитан последний элемент данных)

`Dir[(`*`Path`*`[,`*`Attributes`*`])]`	Функция `Dir` возвращает имя файла или папки, соответствующее критерию, заданному параметрами `Path` и *`Attributes`*. Если файлов (каталогов), удовлетворяющих указанным параметрам нет, то значение функции — "пустая" строка (`""`). Если в качестве параметра `Path` задан шаблон имени файла (например, c:\temp*.bmp), то значение функции — имя файла, соответствующее шаблону. Чтобы получить имена остальных файлов, соответствующих шаблону, надо вызвать функцию `Dir` еще раз, но без параметров. Например:
	`fn = Dir("c:\temp\*.bmp")`
	`fn = fn + Chr(13) + Dir`
	Параметр *`Attributes`* задает (уточняет) тип файла: `Normal` (`0`), `ReadOnly` (`1`), `Hidden` (`2`), `System` (`4`), `Directory` (`16`) — каталог.
	Примеры:
	`Dir("e:\test.txt")` возвращает `"test.txt"`, если файл test.txt есть на диске e:;
	`Dir("e:\t\*.txt")` возвращает имя первого найденного в каталоге e:\t файла с расширением txt;
	`Dir("e:\",vbDirectory)` возвращает имя первой (по порядку) папки корневого каталога диска e:

Таблица 9.30 *(окончание)*

Функция	Описание
`CurDir()`	Функция `CurDir` (без параметров) возвращает полное имя текущей (рабочей) папки. Сразу после запуска программы текущая папка — это папка, из которой запущена программа
`ChDir` *Path*	Функция `ChDir` задает текущий (рабочий) каталог
`MkDir` *Path*	Функция `MkDir` создает новый каталог. Параметр *Path* задает путь к новому каталогу и имя каталога. При попытке создать каталог в несуществующей папке возникнет ошибка
`RmDir` *Path*	Функция `RmDir` удаляет каталог. Параметр *Path* (полное имя каталога) задает каталог, который надо удалить. При попытке удалить каталог, в котором есть файлы, возникнет ошибка. В этом случае нужно сначала из папки удалить файлы (функция `Kill`), а затем — каталог
`Kill` *PathName*	Функция `Kill` удаляет файл. Параметр *PathName* (полное имя файла) задает файл, который надо удалить. Если в качестве имени файла задать шаблон, то будут удалены все файлы, имена которых соответствуют указанному шаблону. Например: `Kill "c:\temp\*.tmp"`
`Environ(`*p*`)`	Возвращает значение переменной окружения. Позволяет получить имя каталога Windows, каталога пользователя, каталога временных файлов. Примеры: `Environ("Windir")` — каталог Windows; `Environ("Homedrive")` — диск, на котором находится каталог пользователя; `Environ("Homepath")` — каталог пользователя; `Environ("Temp")` — каталог временных файлов

Приложение

Описание компакт-диска

Прилагаемый к книге компакт-диск содержит проекты Visual Basic, приведенные в книге. Каждый проект находится в отдельном каталоге.

Для активной работы, чтобы иметь возможность вносить изменения в программы, скопируйте каталоги проектов на жесткий диск компьютера.

Чтобы увидеть, как работает программа, загрузите проект в Visual Basic и запустите его.

Предметный указатель