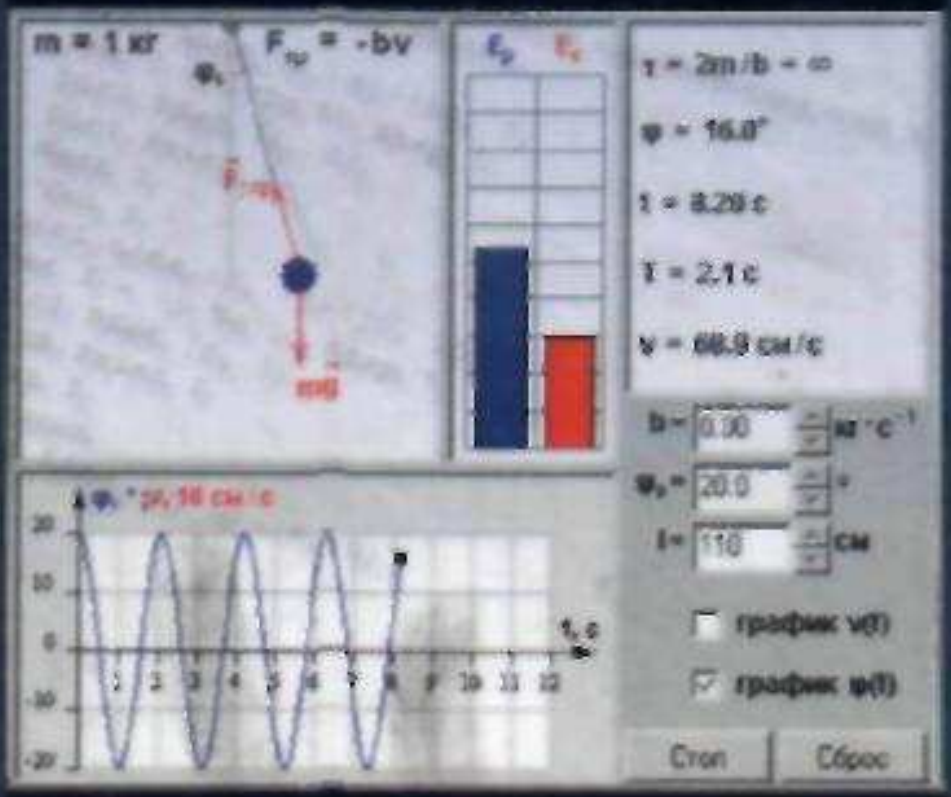
Н. Д. Угринович

ЭЛЕКТИВНЫЙ КУРС

ИССЛЕДОВАНИЕ ИНФОРМАЦИОННЫХ МОДЕЛЕЙ

Учебное пособие



БИНОМ

Оглавление

Рекомендации по использованию учебно-методического комплекта

1. В состав учебно-методического комплекта по элективному курсу входят:
   - учебное пособие по элективному курсу «Исследование информационных моделей»;
   - CD-ROM, содержащий:
     - свободно распространяемые дистрибутивы систем объектно-ориентированного программирования Visual Basic и Delphi, а также интегрированные офисные приложения StarOffice и OpenOffice, содержащие электронные таблицы Calc;
     - компьютерный практикум, содержащий указания по выполнению практических заданий;
     - ответы на задания, т. е. готовые проекты на языках программирования и файлы электронных таблиц.
2. Комплект представляет собой единую образовательную среду, связанную между собой гиперссылками:

   Установить систему программирования VB5.0 CCE CD-ROM

   Проект «Обычный калькулятор»

3. В тексте пособия приняты следующие обозначения и шрифтовые выделения:
   - Шрифтом Arial выделены имена программ, файлов, папок, дисков и URL-адреса в Интернете.
   - *Курсивом* выделены названия диалоговых панелей, пунктов меню и управляющих элементов (текстовых полей, кнопок и т. д.) графического интерфейса.
   - **Полужирным шрифтом** выделены в тексте важные термины и понятия.
   - Шрифтом `Courier` выделены тексты программ на языках программирования Visual Basic и Delphi.
4. Начало выполнения практических заданий и разработки проектов обозначается значком  .

Введение в объектно-ориентированное программирование

Объекты (Objects). Основной единицей в объектно-ориентированном программировании является программный объект, который объединяет в себе как описывающие его данные (свойства), так и средства обработки этих данных (методы). Если говорить образно, то объекты — это существительные, свойства объекта — прилагательные, а методы объекта — глаголы.

Программные объекты обладают **свойствами**, могут использовать **методы** и реагируют на **события**.

Классы объектов являются «шаблонами», определяющими наборы свойств, методов и событий, по которым создаются объекты. Основными классами объектов являются классы, реализующие графический интерфейс проектов.

Объект, созданный по «шаблону» класса объектов, является **экземпляром класса** и **наследует** весь набор свойств, методов и событий данного класса. Каждый экземпляр класса имеет уникальное для данного класса имя. Различные экземпляры класса обладают одинаковым набором свойств, однако значения свойств у них могут отличаться.

Свойства объектов (Properties). Каждый объект обладает определенным набором свойств, первоначальные значения которых можно установить с использованием диалогового окна системы программирования.

Значения свойств объектов можно изменять в программном коде. Для присваивания свойству объекта нового значения слева от знака равенства в строке программного кода необходимо указать имя объекта и затем — название свойства, которые в соответствии с правилами точечной нотации

разделяются между собой точкой. Справа от знака равенства необходимо записать конкретное значение свойства:

```
Объект.Свойство = ЗначениеСвойства
```

Методы объектов (Methods). Для того чтобы объект выполнил какую-либо операцию, необходимо применить метод, которым он обладает. Многие методы имеют аргументы, которые позволяют задать параметры выполняемых действий. Для присваивания аргументам конкретных значений используется двоеточие и знак равенства, а между собой аргументы разделяются запятой.

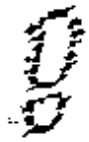
Обратиться к методу объекта можно также с использованием **точечной нотации**. Чтобы определить, для какого объекта вызывается метод, перед именем метода указывается имя объекта, отделенное точкой:

```
Объект.Метод арг1:=значение, арг2:=значение
```

События (Events). Событие представляет собой действие, распознаваемое объектом. Событие может создаваться пользователем (например, щелчок мышью или нажатие клавиши) или быть результатом воздействия других программных объектов. В качестве реакции на события вызывается определенная процедура, которая может изменять свойства объекта, вызывать его методы и т. д.

Графический интерфейс. Система объектно-ориентированного программирования позволяет визуализировать процесс создания графического интерфейса разрабатываемого приложения, т. е. дает возможность создавать объекты и задавать значения их свойств с помощью диалоговых окон системы программирования.

Графический интерфейс необходим для реализации интерактивного диалога пользователя с работающим приложением. Основой для создания графического интерфейса разрабатываемого приложения является форма, представляющая собой окно, в котором размещаются управляющие элементы. Необходимо отметить, что графический интерфейс проекта может включать в себя несколько форм.

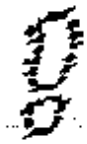


Форма — это объект, представляющий собой окно на экране, в котором размещаются управляющие элементы.

Визуальное конструирование графического интерфейса приложения состоит в том, что на форму с помощью мыши помещаются и «рисуются» те или иные управляющие элементы.

Классы управляющих элементов имеют различное назначение в графическом интерфейсе приложения. **Текстовые поля, метки и списки** обычно используются для ввода и вывода данных, **графические окна** — для вывода графики, **командные кнопки, переключатели и флажки** — для организации диалога и т. д.

На форму могут быть помещены несколько экземпляров одного класса управляющих элементов. Например, несколько кнопок, каждая из которых обладает индивидуальными значениями свойств (надпись, размеры и др.).



Управляющие элементы — это объекты, являющиеся элементами графического интерфейса приложения и реагирующие на события, производимые пользователем или другими программными объектами.

Форма и управляющие элементы обладают определенными наборами свойств, методов и событий.

Событийные процедуры. Для каждого события можно запрограммировать отклик, т. е. реакцию объекта на произошедшее событие. Если пользователь производит какое-либо воздействие на элемент графического интерфейса (например, щелчок), в качестве отклика выполняется некоторая последовательность действий (событийная процедура).

Каждая событийная процедура представляет собой отдельный программный модуль, который реализует определенный алгоритм. Создание программного кода событийной процедуры производится с использованием алгоритмических структур различных типов (**линейная, ветвление, выбор** и **цикл**).

Имя процедуры включает в себя имя объекта и имя события:

```
Объект_Событие()
```

Событийная процедура представляет собой подпрограмму, которая начинает выполняться после реализации определенного события.

Вопросы для размышления

1. В чем состоит различие между классом объектов и экземпляром класса?
2. Можно ли для выбранного программного объекта изменить набор свойств? Набор методов? Набор событий? Значения свойств?
3. Какие объекты обычно используются при конструировании графического интерфейса проекта?
4. В каком случае начинает выполняться событийная процедура?

Глава 1

Основы объектно-ориентированного программирования на языке Visual Basic

1.1. Графический интерфейс системы программирования Visual Basic

Установить систему программирования Visual Basic 5.0 CCE CD-ROM ☞

Окно системы программирования Visual Basic. Система программирования Visual Basic предоставляет пользователю удобный графический интерфейс в процессе разработки приложения. После запуска Visual Basic для начала работы над новым проектом необходимо ввести команду [*Файл-Новый-Standard.exe*].

Появится окно системы программирования Visual Basic (рис. 1.1). Оно включает в себя:

- строку заголовка *Project1-Microsoft Visual Basic [design]*, которая состоит из имени проекта, после которого через тире указана программная среда, затем — текущий режим работы *[design]* — **проектирование**. В режиме **выполнения проекта** текст в квадратных скобках заменяется на *[run]*;
- строку *Главного меню* — под строкой заголовка;
- кнопки с пиктограммами наиболее часто используемых команд — под строкой *Главного меню*.

Окно *Конструктор форм*. Это окно является основным рабочим окном и располагается в центре окна системы программирования Visual Basic. По умолчанию окну проекта присваивается имя *Project1*. Именно в этом окне происходит визуальное конструирование графического интерфейса разрабатываемого приложения.

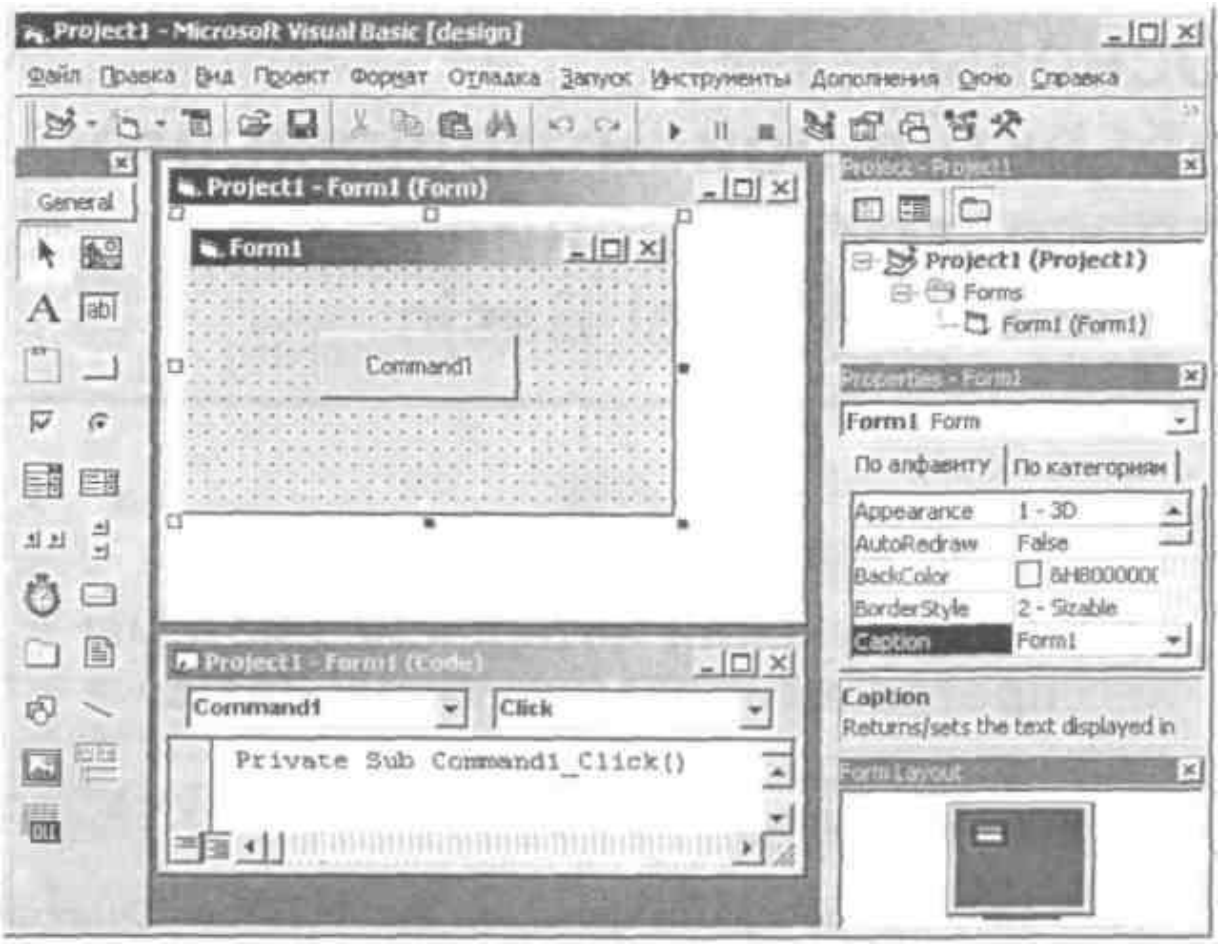


Рис. 1.1. Система программирования Visual Basic

В окне *Конструктор форм* (рис. 1.2) располагается сама форма (в данном случае `Form1`), которая также является объектом и принадлежит классу объектов `Form`. Размеры формы можно менять, перетаскивая мышью правую или нижнюю границу формы.

Форма вызывается командой [*Вид-Объект*].

Первоначально форма пуста, в дальнейшем, в процессе создания графического интерфейса приложения, на ней размещаются управляющие элементы (`Controls`).

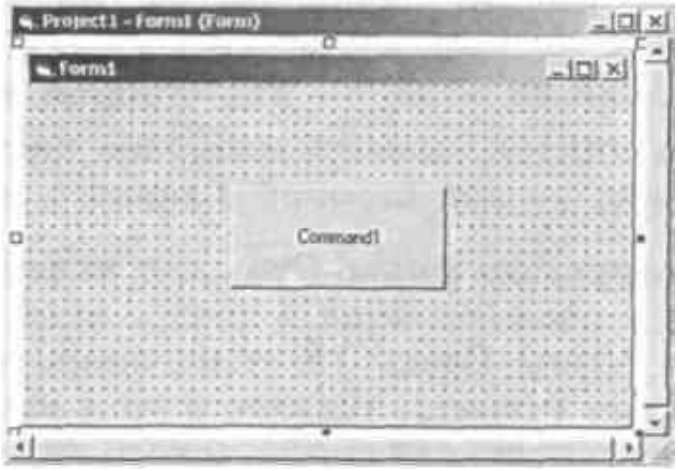


Рис. 1.2. Окно *Конструктор форм*

Окно *Программный код*. С формой связан программный модуль, содержащий программные коды процедур. Для ввода и редактирования текста программы служит окно *Программный код* (рис. 1.3) (в данном случае *Project1-Form1(Code)*), которое вызывается командой [*Вид-Код*].

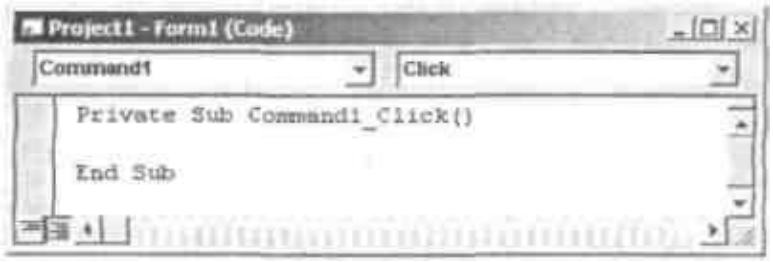


Рис. 1.3. Окно *Программный код*

Сразу под строкой заголовка окна *Программный код* размещаются два раскрывающихся списка. Левый список содержит перечень объектов проекта (объектов, размещенных на форме), а правый — перечень событий, доступных для выбранного объекта.

Панель инструментов. В левой части окна интегрированной среды разработки Visual Basic располагается *Панель инструментов* (рис. 1.4), содержащая пиктограммы управляющих элементов. Стандартный набор управляющих элементов *General* включает в себя 21 класс объектов: командная кнопка (`CommandButton`), текстовое поле (`TextBox`), надпись (`Label`) и т. д.

Выбрав щелчком мышью нужный элемент на *Панели инструментов*, мы можем поместить его на форму проектируемого приложения. Процесс размещения на форме управляющих элементов аналогичен рисованию графических примитивов с использованием графического редактора.

Фактически мы размещаем на форме экземпляры определенных классов объектов. Например, выбрав класс `CommandButton`, мы можем разместить на форме неограниченное количество экземпляров этого класса, т. е. командных кнопок `Command1`, `Command2`, `Command3` и т. д.

Рис. 1.4. Окно *Панель инструментов*

Окно *Свойства объекта*. Справа посередине располагается окно *Свойства объекта* (*Properties*) (рис. 1.5). Окно со-

держит список объектов и список свойств, относящихся к выбранному объекту (форме или управляющему элементу на форме).

Список свойств разделен на две колонки. В левой находятся имена свойств, а в правой — их значения. Установленные по умолчанию значения могут быть изменены. Свойством объекта является количественная или качественная характеристика этого объекта (размеры, цвет, шрифт и др.).

Для некоторых свойств предусмотрена возможность выбора значений из раскрывающегося списка.

Рис. 1.5. Окно *Свойства объекта*

Окно *Обозреватель объектов*. Еще одно важнейшее окно — окно *Обозреватель объектов* (рис. 1.6) может быть вызвано командой [*Вид·Обозреватель объектов*].

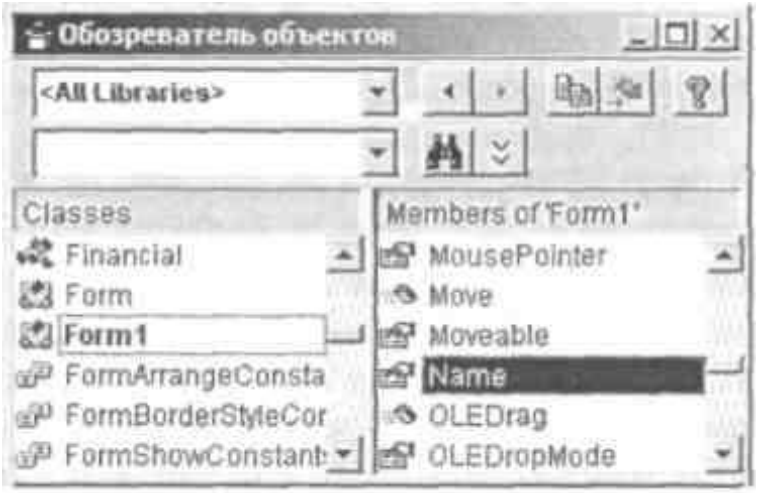


Рис. 1.6. Окно *Обозреватель объектов*

В левой колонке окна производится выбор объекта или класса объектов. В данном случае выбран объект `Form1`.

В правой колонке появляется перечень свойств, методов и событий выбранного объекта или класса объектов.

Окно *Проводник проекта*. Это окно (рис. 1.7) располагается в правом верхнем углу окна системы программирования Visual Basic.

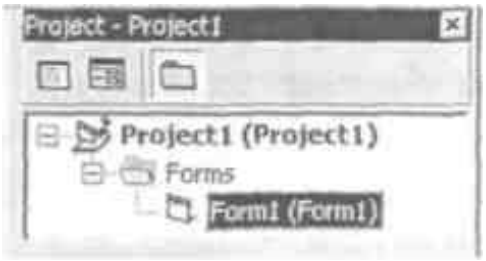


Рис. 1.7. Окно *Проводник проекта*

Оно отображает в виде иерархического каталога все составные части текущего проекта (в данном случае `Project1`) и позволяет переключаться между ними.

Окно *Расположение формы*. В правом нижнем углу находится окно *Расположение формы* (*Form Layout*) (рис. 1.8). Оно показывает, где будет располагаться окно формы на экране монитора в период выполнения программы. Положение формы можно изменять перетаскиванием мышью.

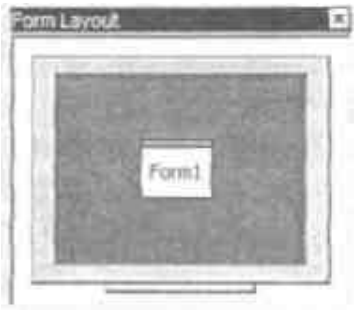


Рис. 1.8. Окно *Расположение формы*

Местоположение и размеры формы. Точное местоположение и размеры формы отображаются двумя парами чисел в правой части линейки инструментов окна приложения (рис. 1.9).

Рис. 1.9. Линейка инструментов

Первая пара чисел показывает расстояние от левого верхнего угла монитора до левого верхнего угла формы, вторая пара — это размеры формы (ширина и высота). Размеры отображаются в особых единицах — твипах (один твип равен примерно 0,018 мм).

Если необходимо установить точные значения местоположения и размеров формы, то это можно сделать, установив значения свойств формы:

- `Left` — расстояние по горизонтали от левого верхнего угла монитора до левого верхнего угла формы;
- `Top` — расстояние по вертикали от левого верхнего угла монитора до левого верхнего угла формы;
- `Width` — ширина формы;
- `Height` — высота формы.

Расположение окон. Расположение вышеперечисленных окон в окне интегрированной среды разработки, а также их размеры можно изменять с помощью мыши.

Вопросы для размышления

1. Какие основные окна системы прогаммирования Visual Basic вы можете назвать? Каково их назначение?

Практические задания для самостоятельного выполнения CD-ROM

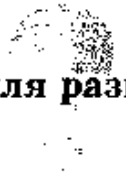
1.1. Запустить систему объектно-ориентированного программирования Visual Basic и выполнить работу «Знакомство с графическим интерфейсом системы прогаммирования Visual Basic».

1.2. Этапы разработки проектов на языке Visual Basic

Создание проектов и приложений в среде Visual Basic можно условно разделить на несколько этапов:

1. **Создание графического интерфейса будущего приложения.** В окне *Конструктор форм* с помощью *Панели инструментов* на форму помещаются управляющие элементы, которые должны обеспечить взаимодействие приложения с пользователем.
2. **Задание значений свойств объектов графического интерфейса.** С помощью окна *Свойства объекта* задаются значения свойств управляющих элементов, помещенных ранее на форму.
3. **Создание и редактирование программного кода.** Для создания заготовки событийной процедуры необходимо осуществить двойной щелчок мышью по управляющему элементу. В окне *Редактор кода* появится заготовка событийной процедуры, имя которой состоит из двух частей: имени объекта и имени события, разделенных знаком подчеркивания (например, `Command1_Click`). Затем в окне *Редактор кода* производится ввод и редактирование программного кода процедуры.
4. **Сохранение проекта.** Так как проект включает в себя несколько файлов (в том числе может быть несколько файлов форм), рекомендуется для каждого проекта создать отдельную папку на диске. Сохранение проекта производится с помощью меню *Файл*:
   - сначала необходимо сохранить форму и связанный с ней программный модуль с помощью команды *Сохранить Form1 Как...*. По умолчанию для файла формы предлагается имя Form1.frm;
   - далее необходимо сохранить файл проекта с помощью команды *Сохранить Проект Как...* По умолчанию для файла проекта предлагается имя Project.vbp.

5. **Компиляция проекта в приложение.** Сохраненный проект может выполняться только в самой системе программирования Visual Basic. Для того чтобы преобразовать проект в приложение, которое может выполняться непосредственно в среде операционной системы, необходимо сохранить проект в исполнимом файле, который имеет расширение exe. Для компиляции проекта в исполнимый файл используется команда [*Файл-Компилировать...*] (в свободно распространяемой версии VB5.0 CCE такая возможность, к сожалению, отсутствует).



Вопросы для размышления

1. Какие основные этапы разработки проекта на языке Visual Basic вы можете назвать? Каков порядок сохранения проекта?

1.3. Создание первого проекта «Обычный калькулятор» на языке Visual Basic

Проект «Обычный калькулятор». Разработаем проект «Обычный калькулятор», который будет производить четыре арифметических действия над числами (сложение, вычитание, умножение и деление).

Проект «Обычный калькулятор»

1. Запустить систему программирования Visual Basic. Создать новый проект командой [*Файл-Новый-Standard.exe*].

Работа над проектом начинается с создания графического интерфейса, для этого в окне *Конструктор форм* на форму помещаются управляющие элементы. Для создания графического интерфейса нашего проекта разместим на форме два текстовых поля для ввода числовых данных и одну метку для вывода результата, а также четыре кнопки для реализации событийных процедур: сложения, вычитания, умножения, деления.

2. С помощью *Панели инструментов* поместить на форму Form1 текстовые поля Text1 и Text2, метку Label1 и командные кнопки Command1, Command2, Command3 и Command4.

Объектам проекта целесообразно присваивать имена, которые дают возможность распознать их тип и на-

значение. Принято, что имя начинается с префикса, который определяет тип объекта. Для форм принят префикс frm, для командных кнопок — cmd, текстовых полей — txt, для надписей — lbl и т. д. После префикса идет информативная часть имени, которая пишется с прописной буквы (например: frmFirst, lblText, cmdExit) или содержит число (например: txt1, txt2, txt3).

Далее необходимо задать новые значения свойств управляющих элементов.

3. Последовательно выделить все объекты и с помощью окна *Свойства объекта* изменить значения свойств формы и управляющих элементов, согласно таблице:

Объект	Свойство	Значение по умолчанию	Новое значение
Form1	Name	Form1	frm1
	Caption	Form1	Обычный калькулятор
Text1	Name	Text1	txt1
	Text	Text1	
Text2	Name	Text2	txt2
	Text	Text2	
Label1	Name	Label1	lbl1
	Caption	Label1	
Command1	Name	Command1	cmdPlus
	Caption	Command1	+
Command2	Name	Command2	cmdMinus
	Caption	Command2	-
Command3	Name	Command3	cmdUmn
	Caption	Command3	*
Command4	Name	Command4	cmdDelen
	Caption	Command4	/

Следующим шагом является создание программного кода событийных процедур. Двойной щелчок мышью по кнопке, для которой нужно создать программный код, вызывает окно *Программный код* с пустой заготовкой событийной процедуры.

4. Осуществить двойной щелчок по кнопке cmdPlus. Появится заготовка событийной процедуры cmdPlus_Click:

```
Private Sub cmdPlus_Click()
End Sub
```

Событийная процедура сложения cmdPlus_Click должна изменить значение свойства Caption метки lbl1 так, чтобы оно являлось суммой числовых значений свойства Text текстовых полей txt1 и txt2. Для преобразования

строковых значений, вводимых в текстовые поля, в десятичное число воспользуемся функцией `Val()`.

5. Код событийной процедуры, реализующей сложение чисел, будет следующим:

```
Private Sub cmdPlus_Click()
lbl1.Caption = Val(txt1.Text) + Val(txt2.Text)
End Sub
```

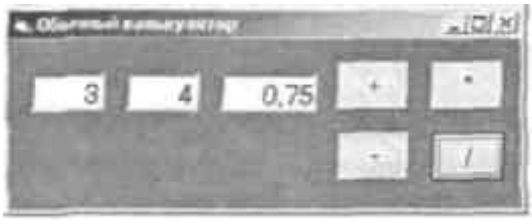
Событийные процедуры вычитания, умножения и деления создаются аналогично.

6. Для каждой из кнопок ввести программный код событийной процедуры.

Сделаем внешний вид проекта более привлекательным. Для этого изменим свойства объектов, определяющие их внешний вид (цвет фона формы, цвет, размер и выравнивание шрифта на метке и в текстовых полях).

7. Активизировать форму `frm1`. В окне *Свойства объекта* выбрать свойство `BackColor` (цвет фона) и двойным щелчком открыть диалоговое окно с цветовой палитрой. Выбрать цвет, например *синий*.
8. Активизировать метку `lbl1`. В окне *Свойства объекта* установить значения свойств:
   - `BackColor` — *белый*,
   - `Font` — *размер шрифта 14*,
   - `Alignment` (выравнивание) — *Right Justify* (по правому краю).
9. Активизировать текстовые поля `txt1` и `txt2`. В окне *Свойства объекта* установить значения свойств:
   - `Font` — *размер шрифта 14*,
   - `Alignment` (выравнивание) — *Right Justify* (по правому краю).

   Сохранить проект.
10. Сохранить файл формы командой *Сохранить Form1 Как...* как frmCalc.frm и файл проекта командой *Сохранить Проект Как...* как prjCalc.vbp.
11. Запустить проект на выполнение. Ввести числа в два текстовых поля и щелкнуть по кнопке арифметической операции. На метку будет выведен результат.



Проект «Обычный калькулятор» хранится в папке \VB\Calc\ CD-ROM

Практические задания для самостоятельного выполнения

CD-ROM

1.2. Создать проект «Расположение формы и управляющих элементов», в котором после запуска форма располагается в центре экрана монитора, а четыре кнопки — в центре формы. После щелчка на кнопках они должны перемещаться в углы формы.

1.4. Переменные в языке программирования Visual Basic

Тип переменной. Тип переменной определяется набором допустимых значений (данных) переменной и допустимыми операциями над этими зрначениями. Значениями переменных числовых типов (**Byte**, **Integer**, **Long**, **Single**, **Double**) являются числа, логических (**Boolean**) — **True** и **False**, строковых (**String**) — последовательности символов и т. д. Обозначения типов переменных являются ключевыми словами языка и поэтому выделяются.

Различные типы требуют для хранения данных в оперативной памяти компьютера различного количества ячеек (байтов) (табл. 1.1).

Таблица 1.1. Типы переменных

Тип переменной	Возможные значения	Объем занимаемой памяти	Приставка к имени
`Byte`	Целые неотрицательные числа от 0 до 255	1 байт	`byt`
`Integer`	Целые числа от −32 768 до 32 767	2 байта	`int`
`Long`	Целые числа от − 2 147 483 648 до 2 147 483 647	4 байта	`lng`
`Single`	Десятичные числа одинарной точности (7–8 значащих цифр) от $-1.4 \cdot 10^{-45}$ до $3.4 \cdot 10^{38}$	4 байта	`sng`
`Double`	Десятичные числа двойной точности (15–16 значащих цифр) от $-5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	8 байтов	`dbl`
`Boolean`	Логическое значение `True` или `False`	2 байта	`bln`
`String`	Строка символов	1 байт на каждый символ	`str`
`Variant`	Любые значения	≥ 16 байтов	`vnt`

Имя переменной. Имя каждой переменной (идентификатор) *уникально* и *не может* меняться в процессе выполнения программы. Имя переменной может состоять из различных символов (латинские и русские буквы, цифры и т. д.), но *должно обязательно* начинаться с буквы и не должно включать знак «.» (точка). Количество символов в имени не может быть более 255.

Разработчик языка Visual Basic фирма Microsoft рекомендует для большей понятности текстов программ для программиста в имена переменных включать особую приставку, которая обозначает тип переменных (например, `intA`, `strA` и т. д).

Объявление переменной. Важно указать исполнителю программы (компьютеру), переменные какого типа используются в программе, в противном случае он будет считать, что переменная имеет универсальный тип **`Variant`** и отведет для ее хранения в памяти 16 или более байтов. Это будет приводить к неэффективному использованию памяти и замедлению работы программы.

Для объявления переменной используется оператор определения переменной. Синтаксис (правило записи) этого оператора следующий:

```
Dim ИмяПеременной As ТипПеременной
```

С помощью одного оператора можно объявить сразу несколько переменных, например:

```
Dim intЧисло As Integer, strСтрока As String
```

Переменные, значения которых не меняются в процессе выполнения программы, называются **константами**. Синтаксис объявления констант следующий:

```
Const ИмяКонстанты As Тип =
                        ЗначениеКонстанты
```

Присваивание переменным значений. Переменная может получить или изменить значение с помощью ***оператора присваивания.*** *Синтаксис этого оператора следующий:*

```
ИмяПеременной = Выражение
```

При выполнении оператора присваивания переменная, имя которой указано слева от знака равенства, получает значение, равное значению выражения (арифметического, строкового или логического), которое находится справа от знака равенства.

Проект «Переменные». Создадим проект, который позволит продемонстрировать использование переменных различных типов, арифметических, строковых и логических выражений и операции присваивания.

Будем производить деление двух целых чисел, а для хранения результата использовать различные типы числовых переменных, т. е. результаты будет вычисляться с различной точностью.

Проект «Переменные»

1. Создать новый проект. Для создания графического интерфейса разместить на форме `frm1` управляющий элемент командную кнопку `cmd1`.

Аргументами программы пусть являются две неотрицательные целочисленные переменные `bytA` и `bytB`, а результатами — целочисленная переменная `intC`, переменная одинарной точности `sngD` и переменная двойной точности `dblE`.

Печать результатов алгоритма будем осуществлять с помощью метода **Print**, которым обладает объект `frm1`.

Метод **Print** используется для печати на форме чисел и строк, а также значений числовых и строковых переменных или выражений, которые образуют **список печати.** В качестве разделителей списка печати используется либо запятая, либо точка с запятой. В первом случае элементы списка печатаются каждый в своей зоне (по 14 символов в каждой зоне), во втором — вплотную друг к другу. В случае отсутствия списка печати на форму выводится пустая строка.

Синтаксис метода **Print** следующий:

```
Объект.Print [СписокПечати]
```

Создадим заготовку событийной процедуры, в которой в качестве объекта будет использоваться кнопка `cmd1`, а в качестве события — щелчок `Click()`.

2. В окне *Программный код* ввести событийную процедуру:

```
Dim bytA, bytB As Byte, intC As Integer,
sngD As Single, dblE As Double
Sub cmd1_Click()
bytA = 2
bytB = 3
intC = bytA / bytB
sngD = bytA / bytB
dblE = bytA / bytB
frm1.Print intC, sngD, dblE
End Sub
```

Создадим вторую событийную процедуру, реализующую операцию конкатенации со строками и строковыми переменными.

3. Разместить на форме frm1 командную кнопку cmd2.
4. В окне *Программный код* ввести вторую событийную процедуру:

```
Dim strA, strB As String
Sub cmd2_Click()
strA = "форма"
strB = "ин" + strA + "тика"
frm1.Print strB
End Sub
```

Создадим третью событийную процедуру, реализующую логические операции с логическими переменными. Объявим логические переменные, присвоим им значения логических выражений, в которые входят операции сравнения, и произведем операцию логического умножения над двумя логическими переменными.

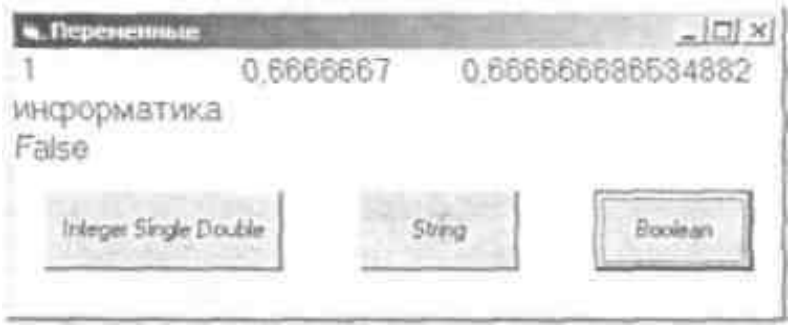
5. Разместить на форме frm1 командную кнопку cmd3.
6. В окне *Программный код* ввести третью событийную процедуру:

```
Dim blnA, blnB, blnC As Boolean
Sub cmd3_Click()
blnA = 5 > 3
blnB = 2*2 = 5
blnC = blnA And blnB
frm1.Print blnC
End Sub
```

7. После запуска проекта на экране появится его графический интерфейс (форма с размещенными на ней командными кнопками).

Последовательные щелчки по кнопкам вызовут выпол-

нение событийных процедур, и на форму будут выведены результаты выполнения проекта.



В первой событийной процедуре производится деление двух чисел с различной точностью, которая зависит от типа переменной, используемой для хранения результатов вычислений.

После выполнения второй событийной процедуры строковая переменная `strC` примет значение `"информатика"`.

После выполнения третьей событийной процедуры логическая переменная `blnC` примет значение **`False`** («Ложь»).

Проект «Переменные» хранится в папке \VB\Variant **CD-ROM**

Вопросы для размышления

1. В чем разница между типом, именем и значением переменной?
2. Можно ли использовать переменную в программе без предварительного объявления?
3. Какие типы переменных используются в языке программирования Visual Basic и какую функцию выполняют приставки в именах переменных?
4. Могут ли входить в состав одного выражения переменные различных типов?
5. В чем состоит разница между операцией арифметического сложения и операцией конкатенации?

Практические задания для самостоятельного выполнения

CD-ROM

1.3. Создать проект «Переменные-2» на основе проекта «Переменные» с выводом результатов в текстовые поля.

1.5. Функции в языке программирования Visual Basic

1.5.1. Функции преобразования типов данных

Функции преобразования реализуют преобразование данных из одного типа в другой.

Функция `Val()`. Часто необходимо бывает преобразовать строковое значение в числовое. Это можно сделать с помощью функции `Val()`, аргументом которой является строка, а значением — число:

```
Val(Строка$)
```

Например, функция `Val()` применяется при вводе чисел в текстовые поля для преобразования строкового значения свойства `Text` в число, которое затем используется в вычислениях.

Проект «Обычный калькулятор»

Строковое выражение, являющееся аргументом функции `Val()`, может быть задано не только в десятичной, но также в восьмеричной (приставка `"&O"`) и шестнадцатеричной (приставка `"&H"`) системах счисления. Например, значением функций `Val("&O3720")` и `Val("&H7D0")` является десятичное число 2000.

Таким образом, появляется возможность перевода чисел, выраженных в строковой форме, из восьмеричной и шестнадцатеричной систем счисления в десятичную систему счисления.

Функции `Str()`, `Oct()` и `Hex()`. Функции `Str()`, `Oct()` и `Hex()` позволяют производить преобразование десятичных чисел соответственно в десятичные, восьмеричные и шестнадцатеричные числа в строковой форме. Аргументом функции является десятичное число, а значением — строка:

```
Str(Число)
Oct(Число)
Hex(Число)
```

Например, значениями функций Str(2000), Oct(2000), Hex(2000) являются десятичное число 2000, восьмеричное число 3720 и шестнадцатеричное число 7D0 в строковой форме.

Проект «Перевод чисел». Создадим проект, который позволит переводить целые числа из десятичной системы счисления в восьмеричную и шестнадцатеричную и обратно — из восьмеричной и шестнадцатеричной в десятичную.

Проект «Перевод чисел»

1. Создать новый проект. Разместить на форме:
   - три текстовых поля txtDec, txtOct, txtHex для ввода и вывода чисел;
   - четыре кнопки cmdDecOct, cmdDecHex, cmdOctDec, cmdHexDec для создания событийных процедур, реализующих перевод чисел;
   - три метки lblDec, lblOct, lblHex для вывода поясняющих надписей над текстовыми полями.

Вводимые в текстовые поля txtOct и txtHex в строковой форме числа переведем сначала в восьмеричное или шестнадцатеричное представление добавлением приставок "&O" или "&H" с помощью операции конкатенации строк. Полученное восьмеричное или шестнадцатеричное число в строковой форме переведем в десятичную числовую форму с помощью функции Val().

2. Ввести событийную процедуру cmdOctDec_Click(), реализующую перевод чисел из восьмеричной системы в десятичную:

```
Sub cmdOctDec_Click()
txtDec.Text = Val("&O" + txtOct.Text)
End Sub
```

3. Создать событийную процедуру cmdHexDec_Click(), реализующую перевод чисел из шестнадцатеричной системы в десятичную.

Введенные в текстовое поле txtDec числа будем переводить сначала из строковой формы в числовую с помощью функции Val(), а затем — из десятичной числовой в строковую восьмеричную или шестнадцатеричную с помощью функций Oct() или Hex().

4. Создать событийную процедуру cmdDecHex_Click(), реализующую перевод чисел из шестнадцатеричной системы в десятичную:

```
Sub cmdDecHex_Click()
txtHex.Text = Hex(Val(txtDec.Text))
End Sub
```

5. Создать событийную процедуру `cmdDecOct_Click()`, реализующую перевод чисел из восьмеричной системы в десятичную.
6. Запустить проект. Для перевода десятичного числа в восьмеричную и шестнадцатеричную системы счисления ввести в левое текстовое поле десятичное число и последовательно щелкнуть по кнопкам *Dec-Oct* и *Dec-Hex*. Для перевода восьмеричного и шестнадцатеричного чисел в десятичную систему счисления ввести в центральное и правое текстовые поля восьмеричное и шестнадцатеричное числа и последовательно щелкнуть по кнопкам *Oct-Dec* и *Hex-Dec*.

Проект «Перевод чисел» хранится в папке \VB\HEX-DEC **CD-ROM**

Практические задания для самостоятельного выполнения **CD-ROM**

1.4. Создать проект «Мультисистемный калькулятор», который позволяет производить арифметические операции над целыми числами в десятичной и восьмеричной системах счисления.

1.5.2. Математические функции

В математических функциях значениями как аргументов, так и функций являются числа. В языке программирования Visual Basic имеется 12 математических функций (табл. 1.2).

Проект «Инженерный калькулятор». Воспользуемся математическими функциями для расширения возможностей проекта «Обычный калькулятор» и превращения его в проект *«Инженерный калькулятор»*.

Таблица 1.2. **Математические функции в Visual Basic**

Функция	Аргумент функции *x*	Возвращаемое функцией значение
`Sin(x)`	Число (в радианах)	Синус числа
`Cos(x)`	Число (в радианах)	Косинус числа
`Tan(x)`	Число (в радианах)	Тангенс числа
`Atn(x)`	Число	Арктангенс в радианах
`Sqr(x)`	Неотрицательное число	Квадратный корень из числа
`Log(x)`	Число	Натуральный логарифм числа
`Exp(x)`	Число	Экспонента числа
`Rnd()`	Нет аргумента	Псевдослучайное число N ($0<N<1$)
`Int(x)`	Число	Наибольшее целое, не превышающее значение аргумента
`Fix(x)`	Число	Число без дробной части
`Abs(x)`	Число	Модуль числа
`Sgn(x)`	Число	Знак числа

Проект «Инженерный калькулятор»

1. Открыть проект «Обычный калькулятор». Добавить на форму семь кнопок `cmdSin`, `cmdCos`, `cmdTan`, `cmdSqr`, `cmdSt`, `cmdLog` и `cmdExit`.

Для этих кнопок создадим событийные процедуры, реализующие вычисление соответствующих функций: синуса, косинуса, тангенса, квадратного корня, возведения в степень, натурального логарифма, а также событийную процедуру окончания работы с проектом.

Для преобразования строкового значения числа, вводимого в текстовые поля, в числовую форму воспользуемся функцией `Val()`.

2. Например, для возведения в степень событийная процедура `cmdSt_Click()` примет вид:

```
Sub cmdSt_Click()
txt3Dec.Text = Val(txt1Dec.Text)^Val(txt2Dec.Text)
End Sub
```

3. Создать событийную процедуру выхода `cmdExit_Click()`:

```
Private Sub cmdExit_Click()
End
End Sub
```

4. Ввести самостоятельно программный код других событийных процедур с использованием встроенных функций языка Visual Basic: `Sin()`, `Cos()`, `Tan()`, `Sqr()` и `Log()`.

5. Запустить проект на выполнение.
Произвести вычисление, например, 2^{32}. Ввести числа 2 и 32 и щелкнуть по кнопке X^Y.

Проект «Инженерный калькулятор» хранится в папке \VB\Calc\ CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

1.5. Создать проект «Треугольник», позволяющий вычислить гипотенузу и площадь прямоугольного треугольника, если известны его катеты.

1.5.3. Строковые функции

В строковых функциях либо аргументы, либо возвращаемые функциями значения являются строками.

Конкатенация строк. Над переменными и строками может производиться операция конкатенации. Эта операция состоит в объединении строк или значений строковых переменных в единую строку. Операция конкатенации обозначается знаком «+», который не следует путать со знаком сложения чисел в арифметических выражениях.

Функция определения длины строки. В функции определения длины строки `Len(Строка$)` аргументом является строка `Строка$`, а возвращает функция числовое значение длины строки (количество символов в строке). Синтаксис функции:

```
Len(Строка$)
```

Пусть аргументом функции `Len` будет строка `"информатика"`, тогда в результате выполнения оператора присваивания

```
intДлинаСтроки = Len("информатика")
```

значением целочисленной переменной `intДлинаСтроки` будет число 11.

Функции вырезания подстроки. В функциях вырезания подстроки (части строки) `Left(Строка$, Длина%)`, `Right(Строка$, Длина%)` и `Mid(Строка$, Позиция%, Длина%)` аргументами являются строка `Строка$`, и числа или целочисленные переменные `Длина%` и `Позиция%`. Функции возвращают строковое значение, равное вырезанной подстроке. Синтаксис функций:

```
Left(Строка$, Длина%)
Right(Строка$, Длина%)
Mid(Строка$, Позиция%, Длина%)
```

Значением функции `Left` является левая подстрока, которая начинается с крайнего левого символа строки и имеет количество символов, равное значению числового аргумента `Длина%`.

Пусть аргументом функции `Left` будет строка "информатика", тогда в результате выполнения оператора присваивания

```
strЛеваяПодстрока = Left("информатика",2)
```

значением строковой переменной `strЛеваяПодстрока` будет строка "ин".

Значением функции `Right` является правая подстрока, которая начинается с крайнего правого символа строки и имеет количество символов, равное значению числового аргумента `Длина%`.

Пусть аргументом функции `Right` будет строка "информатика", тогда в результате выполнения оператора присваивания

```
strПраваяПодстрока = Right("информатика",4)
```

значением строковой переменной `strПраваяПодстрока` будет строка "тика".

Значением функции `Mid` является подстрока, которая начинается с символа, позиция которого в строке задается числовым аргументом `Позиция%`, и имеет длину, равную значению числового аргумента `Длина%`.

Пусть аргументом функции `Mid` будет строка "информатика", тогда в результате выполнения оператора присваивания

```
strПодстрока = Mid("информатика",3,5)
```

значением строковой переменной `strПодстрока` будет строка "форма".

Функция Asc. Функция `Asc` осуществляет преобразование строки в числовой код первого символа. Аргументом функции является строка, а значением — число. Синтаксис функции:

```
Asc(Строка$)
```

Функция Chr. Функция `Chr` осуществляет преобразование числового кода символа в символ. Аргументом функции является число, а значением — символ. Синтаксис функции:

```
Chr(Число)
```

Проект «Строковый калькулятор». Создадим строковый калькулятор, который позволит производить различные преобразования строк.

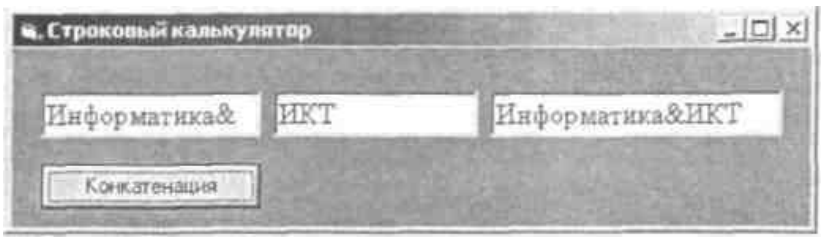
Проект «Строковый калькулятор»

Создадим событийную процедуру, реализующую сложение (конкатенацию) двух строк.

1. Разместить на форме:
   - два текстовых поля `txt1`, `txt2` для ввода строк;
   - текстовое поле `txt3` для вывода результата;
   - кнопку `cmdCon`.
2. Для кнопки ввести программный код событийной процедуры `cmdCon_Click()`, реализующий операцию конкатенации:

```
Sub cmdCon_Click()
txt3.Text = txt1.Text + txt2.Text
End Sub
```

3. Запустить проект, в два первых поля ввести строки и щелкнуть по кнопке *Конкатенация*. В третьем поле появится результат сложения двух строк.



Воспользуемся теперь для преобразования строк строковой функцией Mid$(строка$,bytM,bytN). Функция вырезает из строки строка$ подстроку, начиная с символа, позиция которого в строке задается целочисленной переменной bytM; длина подстроки задается целочисленной переменной bytN.

4. Разместить на форме два текстовых поля txt1Mid и txt2Mid для ввода значений переменных bytM и bytN и кнопку cmdMid.
5. Для кнопки ввести программный код событийной процедуры cmdMid_Click(), реализующий операцию вырезания подстроки. Для преобразования строковых значений свойства Text текстовых полей в числа использовать функцию Val:

```
Sub cmdMid_Click()
txt3.Text = Mid$(txt1.Text, Val(txt1Mid.Text),
Val(txt2Mid.Text))
End Sub
```

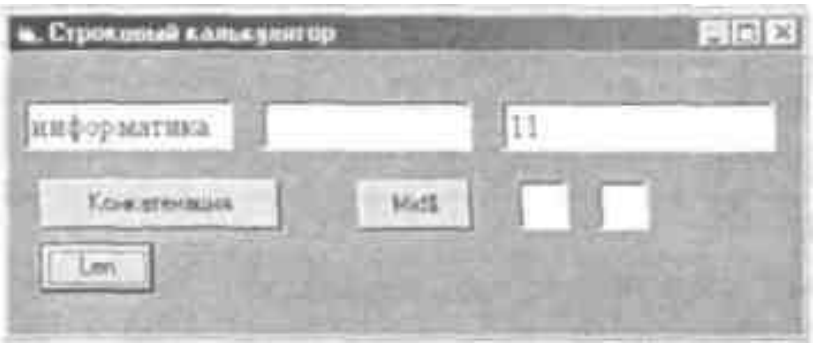
6. Запустить проект, в первое поле ввести строку, в поля функции вырезания подстроки ввести числа и щелкнуть по кнопке *Mid$*. В третьем поле появится вырезанная подстрока.

Для определения количества символов в строке воспользуемся функцией определения длины строки Len(строка$), аргументом которой является строка, а возвращает функция число, равное количеству символов в строке.

7. Разместить на форме кнопку cmdLen и ввести программный код событийной процедуры cmdLen_Click(), реализующий операцию определения количества символов в строке:

```
Sub cmdLen_Click()
txt3.Text = Len(txt1.Text)
End Sub
```

8. Запустить проект, в первое поле ввести строку и щелкнуть по кнопке *Len*. В третьем поле появится число символов в строке.

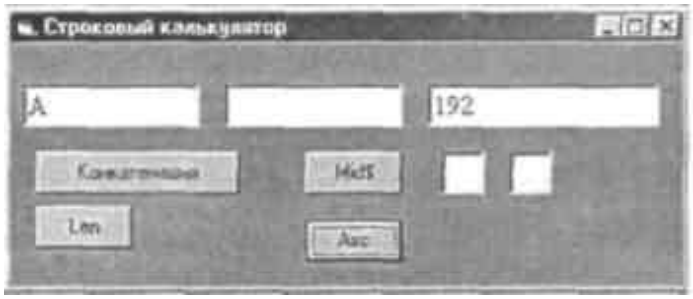


Для преобразования символов в соответствующий им числовой код воспользуемся функцией `Asc(строка$)`, при этом необходимо иметь в виду, что функция возвращает числовой код первого символа строки.

9. Разместить на форме кнопку `cmdAsc` и ввести программный код событийной процедуры `cmdAsc_Click()`, реализующий операцию определения числового кода символа:

```
Sub cmdAsc_Click()
txt3.Text = Asc(txt1.Text)
End Sub
```

10. Запустить проект, в первое поле ввести символ и щелкнуть по кнопке *Asc*. В третьем поле появится числовой код символа.



Проект «Строковый калькулятор» хранится в папке \VB\CalcString **CD-ROM**

Практические задания для самостоятельного выполнения **CD-ROM**

1.6. Создать проект «Строковый калькулятор-2» на основе проекта «Строковый калькулятор». Добавить возможности вырезания из строки левой и правой подстрок и преобразования числового кода символа в символ.

1.5.4. Функции ввода и вывода данных

Функция `InputBox()`. Функция `InputBox()` позволяет вводить данные с помощью диалоговой панели ввода. Аргументами этой функции являются три строки, а значением функции — строка по умолчанию или строка, введенная пользователем. Синтаксис функции следующий:

```
Переменная$ = InputBox("Подсказка",
     "Заголовок", ["ЗначениеПоУмолчанию"])
```

Квадратные скобки здесь и далее означают, что данный параметр необязателен.

В результате выполнения этой функции появляется диалоговая панель с текстовым полем (рис. 1.10). В строке заголовка панели будет выведено значение второго аргумента "Заголовок", на саму панель — значение аргумента "Подсказка", в текстовое поле — значение аргумента "ЗначениеПоУмолчанию" (если это значение отсутствует, содержимое текстового окна также отсутствует).

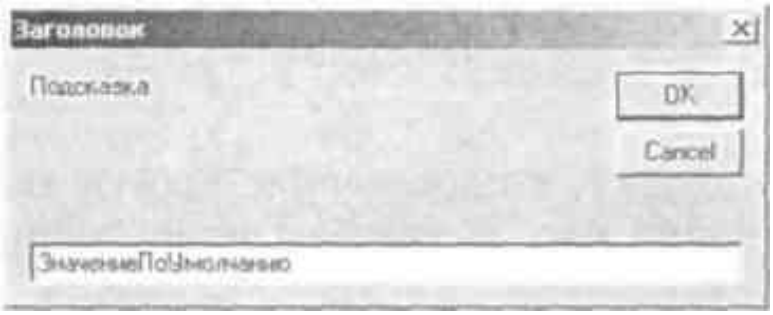


Рис. 1.10. Диалоговая панель ввода функции `InputBox()`

Если пользователь щелкнет по кнопке *OK*, то значением функции станет строка, введенная пользователем в текстовое поле. Если пользователь щелкнет по кнопке *Cancel*, то значением функции станет строка "ЗначениеПоУмолчанию".

Функция MsgBox(). Функция `MsgBox()` позволяет выводить сообщения не на форму, а на специальную панель сообщений, на которой можно разместить определенный набор кнопок и информационный значок о типе сообщения. Кроме того, функция `MsgBox()` получает определенное значение, которое может быть присвоено числовой переменной. Синтаксис функции следующий:

```
Переменная = MsgBox("Сообщение"
       [,ЧисКод1+ ЧисКод2][,"Заголовок"])
```

Аргумент "Сообщение" выводится на панель сообщений, аргумент ЧисКод1+ЧисКод2 определяет внешний вид панели, а строка "Заголовок" выводится в строку заголовка панели. Последние два аргумента не являются обязательными.

Внешний вид панели сообщений можно менять, используя различные значения ЧисКод1 и ЧисКод2. Значение ЧисКод1 определяет вид пиктограммы, которая помещается на панель сообщений, а значение ЧисКод2 — набор кнопок, размещаемых на панели (табл. 1.3).

Таблица 1.3. **Значения ЧисКод1 и ЧисКод2, определяющие вид панели сообщений**

ЧисКод1	Пиктограмма
16	
32	
48	
64	

ЧисКод2	Набор кнопок
0	*OK*
1	*OK, Отмена*
2	*Стоп, Повтор, Пропустить*
3	*Да, Нет, Отмена*
4	*Да, Нет*
5	*Повтор, Отмена*

Определенную пиктограмму и определенную комбинацию кнопок, размещаемых на панели сообщений, можно одновременно установить с помощью одного числа, являющегося суммой чисел ЧисКод1 и ЧисКод2. Например, число 36 можно рассматривать как сумму чисел 32 (код пиктограммы «Вопрос») и 4 (код комбинации кнопок *Да*, *Нет*). В этом случае функция MsgBox() будет выводить панель сообщений с текстом, пиктограммой-знаком вопроса и кнопками *Да*, *Нет*.

Можно задать также два числа, разделенных знаком «+».

Например, если задать функцию с аргументами MsgBox("Сообщение", 48 + 3, "Заголовок"), то будет выведена панель сообщений, показанная на рис. 1.11.

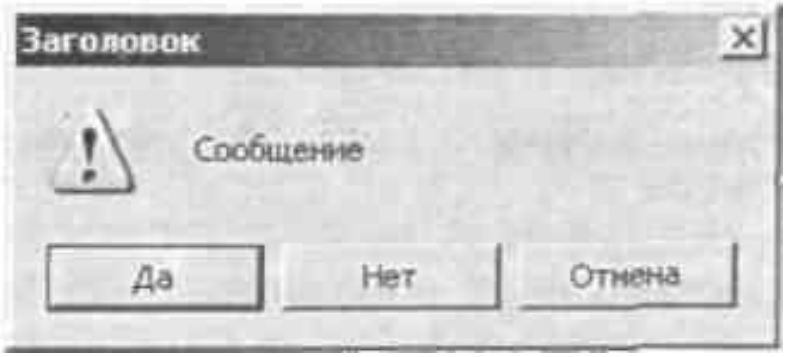


Рис. 1.11. Диалоговая панель сообщений функции MsgBox()

Щелчок по той или иной кнопке приводит к вычислению значения функции, которое зависит от этой кнопки. То есть значение, возвращаемое функцией MsgBox(), позволяет определить, по какой из кнопок был осуществлен щелчок (какая из кнопок была «нажата») (табл. 1.4).

Таблица 1.4. **Значения функции MsgBox()**

«Нажатая» кнопка	Значение функции
ОК	1
Отмена	2
Стоп	3
Повтор	4
Пропустить	5
Да	6
Нет	7

Проект «Проверка знаний». Разработаем проект, который позволит контролировать знания. Алгоритм контроля должен последовательно реализовывать следующие операции:

- задать (ввести) вопрос;
- запросить ответ и запомнить введенное с клавиатуры значение;
- полученный ответ сравнить с правильным и, в зависимости от их совпадения или несовпадения, реализовать те или иные действия.

Проект «Проверка знаний»

1. Разместить на форме кнопку cmd1, задать значение Начать проверку свойства Caption. Создать событийную процедуру cmd1_Click().

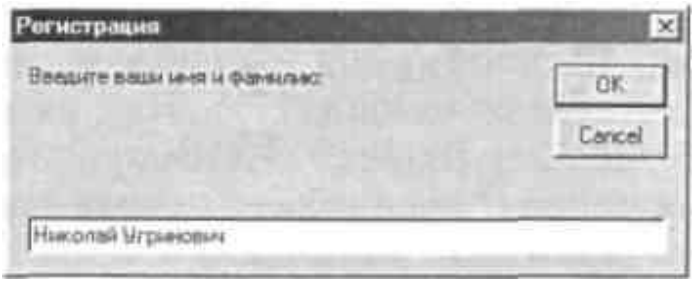
Сначала реализуем регистрацию проверяемого с использованием функций InputBox() и MsgBox().

2. С помощью функции InputBox() запросить имя и фамилию и присвоить ее значение строковой переменной strA, а с помощью функции MsgBox() вывести результаты регистрации:

```
Dim strA As String, bytB As Byte
Sub cmd1_Click()
strA = InputBox("Введите ваши имя и фамилию:",
"Регистрация")
bytB = MsgBox("Уважаемый " + strA +
", Вы готовы к проверке знаний?", 36,
"Конец регистрации")
```

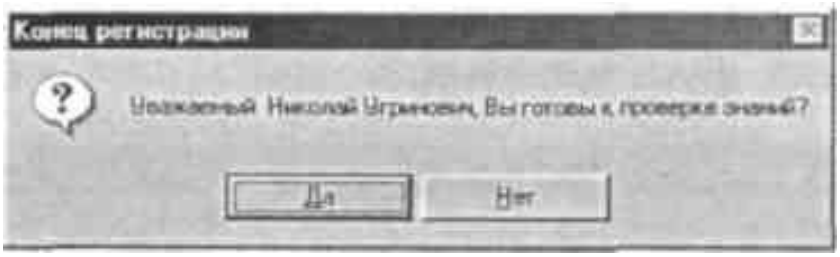
```
If bytB = 7 Then End
End Sub
```

3. Запустить проект и щелкнуть по кнопке *Начать проверку*.
   На появившейся диалоговой панели *Регистрация* ввести в текстовое поле имя и фамилию.



В функции MsgBox() второй аргумент представляет собой числовое значение, которое одновременно задает тип выводимого информационного окна и набор размещенных на нем кнопок.

4. Число 36 обеспечивает вывод информационного окна типа «Вопрос», которое имеет две кнопки *Да* и *Нет*.



Щелчок по одной из кнопок приводит к возвращению функцией определенного числового значения (*Да* — 6, *Нет* — 7), которое присваивается числовой переменной bytB.

5. Условный оператор реализует либо выход из программы (щелчок по кнопке *Нет*), либо продолжение работы и переход к проверке знаний (щелчок по кнопке *Да*).

Вопрос задается с помощью функции InputBox(), проверка правильности ответа производится с помощью оператора условного перехода **If-Then-Else**.

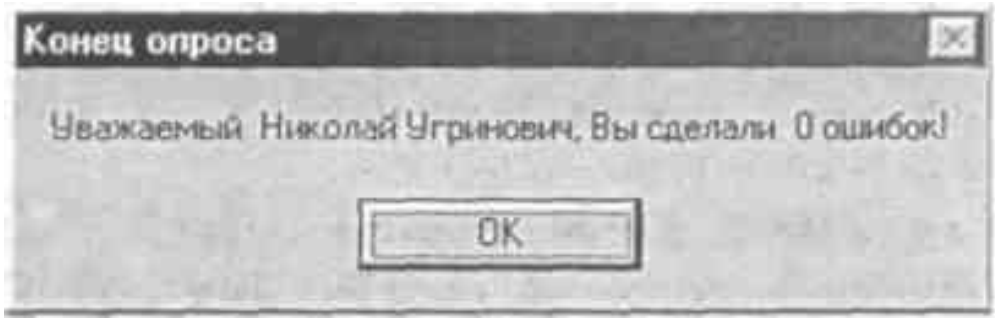
1.6.2. Алгоритмическая структура «ветвление»

Вывод информации о правильности или неправильности ответа производится с помощью функции MsgBox() в форме оператора (бесскобочная запись), с числовым значением второго аргумента 0, что обеспечивает вывод информационного окна с одной кнопкой *OK*.

6. Ввести в событийную процедуру программный код, реализующий проверку знаний в виде последовательности вопросов. В переменной bytN будем накапливать количество неправильных ответов:

```
strC = InputBox("Чему равен 1 байт?:",
"Первый вопрос")
If strC = "8 бит" Then MsgBox "Правильно!",
0, "Первый вопрос"
Else MsgBox "Неправильно!", 0,
"Первый вопрос": bytN = bytN + 1
strC = InputBox("Переведите десятичное число 5
в двоичную систему счисления:",
"Второй вопрос")
If strC = "101" Then MsgBox "Правильно!", 0,
"Второй вопрос"
Else MsgBox "Неправильно!", 0,
"Второй вопрос": bytN = bytN + 1
MsgBox "Уважаемый "+ strA + ", Вы сделали "
+ Str(bytN) + " ошибок!", 0, "Конец опроса"
```

7. Запустить проект, пройти регистрацию и ответить на вопросы. Результат будет выведен с помощью информационного окна функции MsgBox().



Проект «Проверка знаний» хранится в папке \VB\InputOutput **CD-ROM**

Практические задания для самостоятельного выполнения

CD-ROM

1.7. Создать проект «Игра Баше», в котором ввод данных осуществляется с помощью панели ввода, вывод — с помощью панели сообщений. Суть игры состоит в следующем: имеется *N* предметов; два игрока по очереди берут 1, 2 или 3 предмета; проигрывает тот игрок, который забирает последний предмет.

1.6. Основные типы алгоритмических структур и их кодирование на языке Visual Basic

1.6.1. Линейный алгоритм

В предыдущих параграфах были рассмотрены событийные процедуры, в которых команды выполняются последовательно одна за другой. Такие последовательности команд будем называть **сериями**, а алгоритмы, состоящие из таких серий, — **линейными**.

Алгоритм, в котором команды выполняются последовательно одна за другой, называется **линейным алгоритмом**.

Для того чтобы сделать алгоритм более наглядным, часто используют **блок-схемы**.

На блок-схеме различные элементы алгоритма изображаются с помощью разных геометрических фигур: для обозначения начала и конца алгоритма используются прямоугольники с закругленными углами, а для обозначения последовательностей команд — обычные прямоугольники.

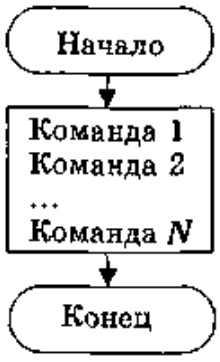


Рис. 1.12. Линейный алгоритм

На блок-схеме, изображенной на рис. 1.12, хорошо видна структура линейного алгоритма, по которой исполнителю (человеку) удобно отслеживать процесс его выполнения.

1.6.2. Алгоритмическая структура «ветвление»

В отличие от линейных алгоритмов, в которых команды выполняются последовательно одна за другой, в алгоритмическую структуру «ветвление» входит условие, в зависимости от выполнения или невыполнения которого реализуется та или иная последовательность команд (серий).

В алгоритмической структуре «**ветвление**» в зависимости от истинности **условия** выполняется та или иная серия команд.

Условие. Будем называть условием высказывание, которое может быть либо истинным, либо ложным. Условие, записанное на формальном языке, называется **условным или логическим выражением.**

Условные выражения могут быть **простыми и сложными.** Простое условное выражение включает в себя два числа, две **переменных** или два **арифметических выражения,** которые сравниваются между собой с использованием операций сравнения (равно, больше, меньше и т. д.). Например: `X > 3`, `Y = 4*4` и т. д.

Сложное условие — это последовательность простых условий, объединенных между собой символами логических операций. Например: `X > 3` **And** `Y = 4*4`.

Алгоритмическая структура «ветвление» может быть зафиксирована различными способами:

- графически, с помощью блок-схемы;
- на языке программирования Visual Basic с использованием специальной инструкции ветвления (оператора условного перехода).

Инструкция `If-Then-Else`. Алгоритмическая структура «ветвление» кодируется на языке Visual Basic с помощью инструкции **`If-Then-Else`.** После первого ключевого слова **`If`** должно быть размещено условие. После второго ключевого слова **`Then`** — последовательность команд `серия 1`, которая должна выполняться, если условие принимает значение «истина». После третьего ключевого слова **`Else`** размещается последовательность команд `серия 2`, которая должна выполняться, если условие принимает значение «ложь» (рис. 1.13).

Блок-схема	Язык программирования Visual Basic

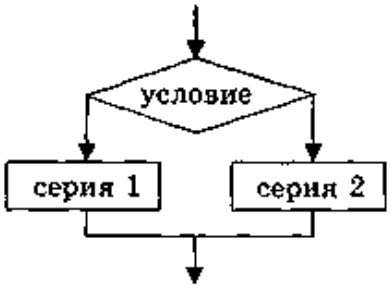
	`If условие Then`
	`  серия 1`
	`[Else`
	`  серия 2]`
	`End If`
	
	`If условие Then серия 1 [Else серия 2]`

Рис. 1.13. Алгоритмическая структура «ветвление»

Оператор условного перехода может быть записан в **многострочной форме** или в **однострочной форме.**

В многострочной форме он записывается с помощью инструкции **If-Then-Else-End If** (Если-То-Иначе-Конец Если). В этом случае второе ключевое слово **Then** размещается в той же строке, что и условие, а последовательность команд (серия 1), должна размещаться в следующей строке. Третье ключевое слова **Else** размещается в третьей строке, а последовательность команд (серия 2) — в четвертой. Конец инструкции ветвления **End If** размещается в пятой строке.

В однострочной форме он записывается с помощью инструкции **If-Then-Else** (Если-То-Иначе). Если инструкция не помещается на одной строке, она может быть разбита на несколько строк. Такое представление инструкций более наглядно для человека. Компьютер же должен знать, что разбитая на строки инструкция представляет единое целое. Это обеспечивает знак «переноса», который задается символом подчеркивания «_» после пробела.

Третье ключевое слово **Else** в сокращенной форме инструкции может отсутствовать. (Необязательные части оператора записываются в квадратных скобках.) Тогда, в случае если условие ложно, выполнение оператора условного перехода заканчивается и выполняется следующая строка программы.

Проект «Проверка знаний»

Практические задания для самостоятельного выполнения **CD-ROM**

1.8. Создать проект «Поиск большего из двух чисел» на языке программирования Visual Basic.

1.6.3. Алгоритмическая структура «выбор»

Алгоритмическая структура «выбор» применяется для реализации ветвления со многими вариантами серий команд. Вструктуру выбора входят несколько условий, проверка которых осуществляется в строгой последовательности их записи в команде выбора. При истинности одного из условий `условие 1`, `условие 2` и т. д. выполняется соответствующая последовательность команд `серия 1`, `серия 2` и т. д. Если ни одно из условий не будет истинно, то будет выполнена последовательность команд `серия`.

В алгоритмической структуре **«выбор»** выполняется одна из нескольких последовательностей команд при истинности соответствующего **условия**.

Алгоритмическая структура «выбор» может быть зафиксирована различными способами:

- графически, с помощью блок-схемы;
- на языке программирования Visual Basic с использованием специальной инструкции выбора.

Инструкция Select Case. Алгоритмическая структура «выбор» кодируется на языке Visual Basic с помощью инструкции **Select Case**. Инструкция выбора начинается с ключевых слов **Select Case**, после которых записывается выражение (переменная или арифметическое выражение). После ключевых слов **Case** записываются значения, с которыми сравнивается заданное выражение (проверяется выполнение условий). При выполнении одного из условий начинает выполняться соответствующая серия команд. Если ни одно из условий не выполняется, то будет выполнена серия команд после ключевых слов **Case Else**. Заканчивается инструкция ключевыми словами **End Select** (рис. 1.14).

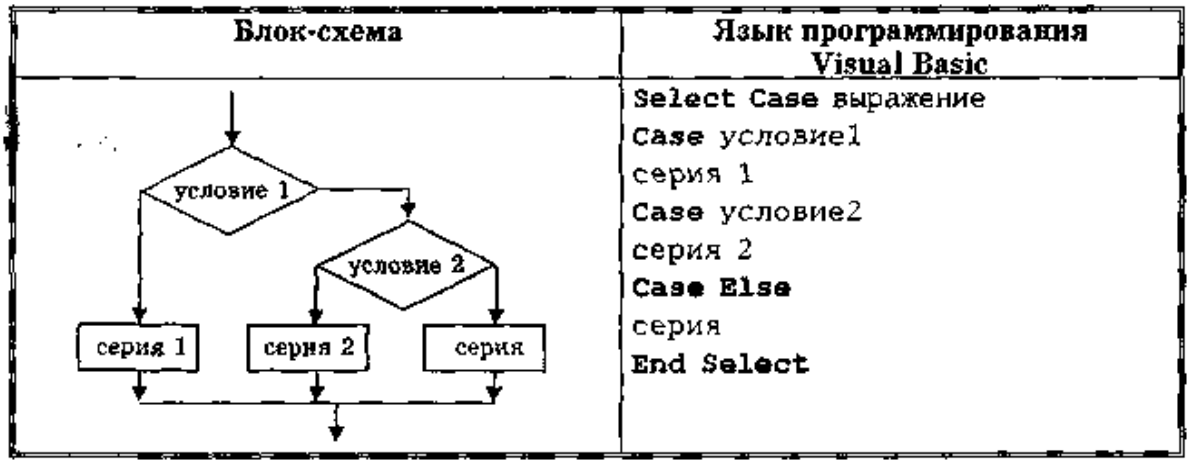


Рис. 1.14. Алгоритмическая структура «выбор»

Проект «Отметка». В качестве примера использования инструкции выбора разработаем проект «Отметка», который позволяет выставлять отметку за работу в зависимости от количества сделанных ошибок.

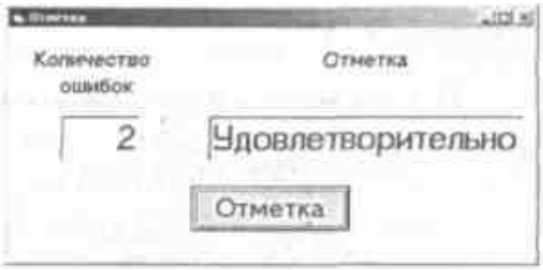
Проект «Отметка»

1. Создать новый проект. Разместить на форме:
   - текстовое поле `txtN` для ввода количества ошибок;
   - текстовое поле `txtBall` для вывода отметки;
   - две метки для вывода поясняющих надписей;
   - кнопку `cmdBall` для запуска событийной процедуры.

2. Ввести код событийной процедуры cmdBall_Click(), которая обеспечивает ввод количества ошибок в поле txtN и осуществляет вывод отметки в текстовое поле txtBall:

```
Dim bytN As Byte
Private Sub cmdBall_Click()
bytN = Val(txtN.Text)
Select Case bytN
Case 0
txtBall.Text = "Отлично"
Case 1
txtBall.Text = "Хорошо"
Case 2
txtBall.Text = "Удовлетворительно"
Case Else
txtBall.Text = "Плохо"
End Select
End Sub
```

3. Запустить проект. Ввести в левое текстовое поле количество ошибок и щелкнуть по кнопке *Отметка*. В правое текстовое поле будет выведена отметка.



Проект «Отметка» хранится в папке \VB\Ball\ CD-ROM

Практические задания для самостоятельного выполнения

CD-ROM

1.9. Создать проект «Тест с выборочным ответом» на языке программирования Visual Basic.

1.6.4. Алгоритмическая структура «цикл»

В алгоритмическую структуру «цикл» входит серия команд, выполняемая многократно. Такая последовательность команд называется телом цикла.

Циклические алгоритмические структуры бывают двух типов:

- **циклы со счетчиком**, в которых тело цикла выполняется определенное количество раз;
- **циклы по условию**, в которых тело цикла выполняется, пока выполняется условие.

В алгоритмической структуре «цикл» серия команд (**тело цикла**) выполняется многократно.

Алгоритмическая структура «цикл» может быть зафиксирована различными способами:

- графически, с помощью блок-схемы;
- на языке программирования Visual Basic с использованием специальных инструкций, реализующих циклы различного типа.

Цикл со счетчиком. Когда заранее известно, какое число повторений тела цикла необходимо выполнить, можно воспользоваться циклической инструкцией **For-Next** (оператором цикла со счетчиком).

Синтаксис оператора **For-Next** следующий: строка, начинающаяся с ключевого слова **For**, является заголовком цикла, а строка с ключевым словом **Next** — концом цикла, между ними располагаются операторы, являющиеся телом цикла.

В начале выполнения цикла значение переменной Счетчик устанавливается равным НачЗнач. При каждом проходе цикла (выполнении тела цикла) переменная Счетчик изменяется (увеличивается или уменьшается) на величину шага. Если она достигает величины КонЗнач, то цикл завершается и выполняются следующие за ним операторы (*рис. 1.15*).

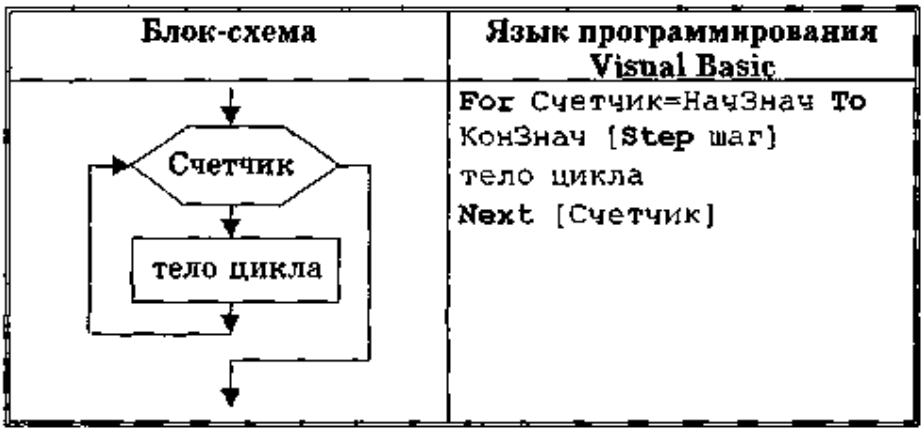


Рис. 1.15. Цикл со счетчиком

Проект «Коды символов». В качестве примера использования цикла со счетчиком создадим проект «Коды символов», который выводит символы по числовым кодам (кодировочную таблицу символов).

Проект «Коды символов»

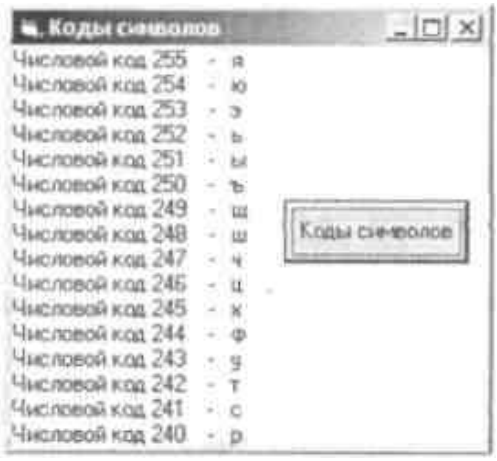
1. Создать новый проект. Разместить на форме кнопку cmdT.

Воспользуемся циклом со счетчиком с шагом −1, для того чтобы выводить на форму символы, начиная с наибольшего числового кода 255. Такая последовательность вывода символов связана с тем, что первые 33 числовых кода (от 0 до 32) соответствуют не символам, а клавишам клавиатуры (клавиши управления курсором, {Пробел}, {Ввод} и др).

2. Ввести код событийной процедуры cmdT_Click(), которая осуществляет вывод кодировочной таблицы символов на форму:

```
Dim strA As String, N As Integer
Sub cmdT_Click()
For N = 255 To 33 Step -1
strA = Chr(N)
Print "Числовой код"; N; "    -    "; strA
Next N
End Sub
```

3. Запустить проект.
   Для получения кодировочной таблицы символов щелкнуть по кнопке *Кодировочная таблица*.



Проект «Коды символов» хранится в папке \VB\FOR-NEXT CD-ROM

Цикл с предусловием. Часто бывает так, что необходимо повторить тело цикла, но заранее неизвестно, какое количество раз это надо сделать. В таких случаях количество повторений зависит от некоторого условия.

Цикл называется **циклом с предусловием**, если условие выхода из цикла стоит в начале, перед телом цикла.

Цикл с предусловием никогда не выполняется в случае невыполнения условия.

В языке программирования Visual Basic цикл с предусловием реализуется с помощью инструкций **Do While-Loop** и **Do Until-Loop**.

Проверка условия выхода из цикла проводится с помощью ключевых слов **While** или **Until**. Эти слова придают одному и тому же условию противоположный смысл. Ключевое слово **While** обеспечивает выполнение цикла, пока выполняется условие, т. е. пока условие имеет значение «истина». Как только условие примет значение «ложь», выполнение цикла закончится. В этом случае условие является **условием продолжения цикла**.

Ключевое слово **Until** обеспечивает выполнение цикла до тех пор, пока не выполнится условие, т. е. пока условие имеет значение «ложь». Как только условие примет значение «истина», выполнение цикла закончится. В этом случае условие является **условием завершения цикла** (рис. 1.16).

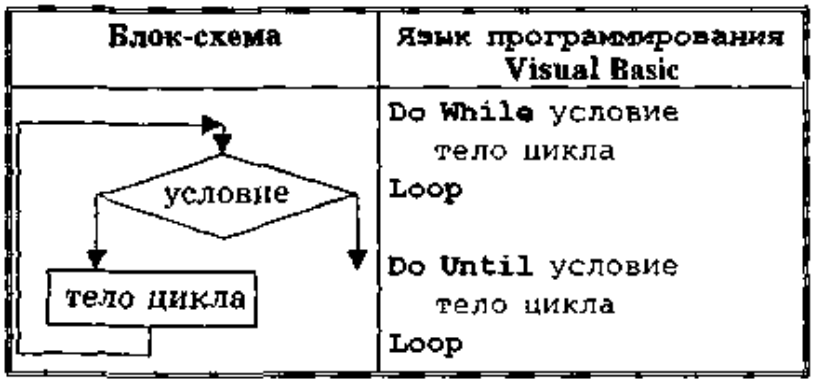


Рис. 1.16. Цикл с предусловием

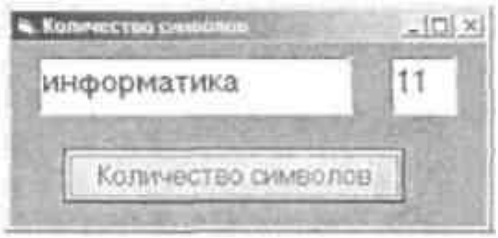
Проект «Количество символов». В качестве примера использования цикла с предусловием разработаем проект, позволяющий подсчитывать количество символов в заданном тексте.

Проект «Количество символов»

1. Создать новый проект. Разместить на форме:
   - текстовое поле `txtText` для ввода текста;
   - текстовое поле `txtNum` для вывода количества символов в тексте;
   - кнопку `cmd1` для запуска событийной процедуры.
2. Ввести код событийной процедуры `cmd1_Click()`, которая осуществляет в цикле с предусловием подсчет количества символов:

```
Dim bytN As Byte
Private Sub cmd1_Click()
bytN = 0
Do While bytN < Len(txtText.Text)
bytN = bytN + 1
Loop
txtNum.Text = bytN
End Sub
```

3. Запустить проект. Ввести текст в левое поле и щелкнуть по кнопке *Количество символов*.
В правом поле появится число, равное количеству символов.



Проект «Количество символов» хранится в папке \VB\While **CD-ROM**

Цикл с постусловием. Цикл с постусловием выполняется обязательно как минимум один раз, независимо от того, выполняется условие или нет.

Цикл называется **циклом с постусловием**, если условие выхода из цикла стоит в конце, после тела цикла.

В языке программирования Visual Basic цикл с постусловием реализуется с помощью инструкций **Do-Loop While** и **Do-Loop Until**. Проверка условия выхода из цикла проводится с помощью ключевых слов **While** или **Until** (рис. 1.17).

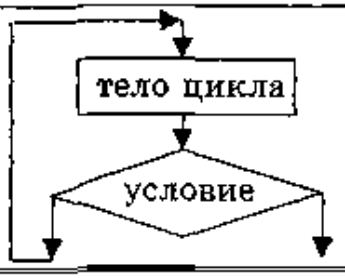
Блок-схема	Язык программирования Visual Basic
	Do тело цикла **Loop While** условие Do тело цикла **Loop Until** условие

Рис. 1.17. Цикл с постусловием

Вопросы для размышления

1. Алгоритмическую структуру какого типа необходимо применить, если:
   - последовательность команд должна быть выполнена определенное количество раз;
   - последовательность команд выполняется или не выполняется в зависимости от условия;
   - последовательность команд должна быть обязательно выполнена хотя бы один раз и должна повторяться, пока условие справедливо?

Практические задания для самостоятельного выполнения

CD-ROM

1.10. Создать проект «Факториал числа», реализующий вычисление факториала числа на языке программирования Visual Basic.

1.11. Создать проект «Слово-перевертыш», который изменяет прямую последовательность символов в слове (слева направо) на обратную последовательность (справа налево), на языке программирования Visual Basic.

1.6.5. Общие процедуры

Определение общей процедуры. При разработке сложного алгоритма целесообразно стараться выделить в нем последовательности действий (алгоритмы), которые выполняют решение каких-либо подзадач и могут многократно вызываться из основного алгоритма. Такие алгоритмы называются **вспомогательными** и в алгоритмических языках программирования реализуются в форме **подпрограмм**, которые вызываются из основной программы.

В объектно-ориентированных языках программирования вспомогательные алгоритмы реализуются с помощью общих процедур. Общие процедуры создаются в тех случаях, когда в программном модуле можно выделить многократно встречающиеся последовательности действий (алгоритмы).

Запуск общих процедур не связывается с какими-либо событиями, а реализуется путем вызова из других процедур.

Общая процедура представляет собой подпрограмму, которая начинает выполняться после ее вызова из другой процедуры.

Каждой общей процедуре дается уникальное название — **имя процедуры** и устанавливается **список входных и выходных параметров** процедуры.

Список входных параметров представляет собой набор переменных, значение которых должно быть установлено до начала выполнения процедуры.

Список выходных параметров представляет собой набор переменных, значение которых должно быть установлено после окончания выполнения процедуры.

Синтаксис общей процедуры в языке Visual Basic:

```
Sub ИмяПроцедуры(СписокПараметров)
программный код
End Sub
```

Общая процедура вызывается на выполнение либо с помощью оператора **Call**, либо по имени. В случае вызова процедуры с использованием оператора **Call** список параметров заключается в скобки:

```
Call ИмяПроцедуры(СписокПараметров)
```

В случае вызова процедуры по имени список параметров приводится без скобок:

```
ИмяПроцедуры СписокПараметров
```

1.7. Графические возможности языка программирования Visual Basic

Графические методы. На формах (`Form`) или в графических окнах (`PictureBox`) можно рисовать различные графические примитивы с использованием графических методов.

Метод **Scale** позволяет задать систему координат и масштаб для формы или графического окна:

```
object.Scale (X1,Y1) - (X2,Y2)
```

Аргументами метода являются X1,Y1 — координаты левого верхнего угла объекта и X2,Y2 — координаты правого нижнего угла объекта.

Метод **Pset** — установка точки с заданными координатами и цветом:

```
object.Pset (X,Y) [,color]
```

Аргументами метода являются X, Y — координаты точки и color — цвет точки. Значение аргумента color можно задать различными способами:

- с помощью одной из восьми констант, определяющих цвет (vbBlack — черный, vbBlue — синий, vbGreen — зеленый, vbCyan — голубой, vbRed — красный, vbMagenta — пурпурный, vbYellow — желтый, vbWhite — белый);
- с помощью функции QBColor(number), аргументом которой являются числа от 0 до 15, а результатом — один из основных 16 цветов;
- с помощью функции RGB(bytRed,bytGreen,bytBlue), аргументами которой являются три числа из диапазона 0 до 255 (интенсивности базовых цветов), а значением — число типа **Long** в диапазоне от 0 до $256^3 - 1$ (16 777 215). Таким образом, определяется цветовая палитра с более чем 16 миллионами цветов, а каждый цвет задается числом, которое вычисляется по формуле bytRed + 256·bytGreen + 256^2·bytBlue.

В случае отсутствия аргумента color рисование будет производиться цветом, принятым по умолчанию (черным).

Метод **Line** — рисование линии, прямоугольника или закрашенного прямоугольника заданного цвета:

```
object.Line (X1,Y1) - (X2,Y2) [,color][,B][F]
```

Аргументами метода являются X1, Y1 и X2, Y2 — координаты концов линии (или левого верхнего и правого нижнего угла прямоугольника), color — цвет линии. Флаг **B** задает рисование прямоугольника, а флаг **F** его закрашивает.

Метод **Circle** — рисование окружности, овала или дуги:

```
object.Circle (X,Y),radius [,color, start,
                    end, aspect]
```

Аргументами метода являются X, Y — координаты центра окружности, radius — радиус окружности, color — цвет окружности, start и end — начальный и конечный угол дуги, aspect — коэффициент сжатия.

Если графический метод применяется к объекту форма (Form), то при его записи имя объекта object можно опускать.

Проект «Построение графика функции». Разработаем проект построения в графическом окне графика функции с использованием графических методов. В качестве примера рассмотрим построение графика функции $y = \sin x$.

Проект «Построение графика функции»

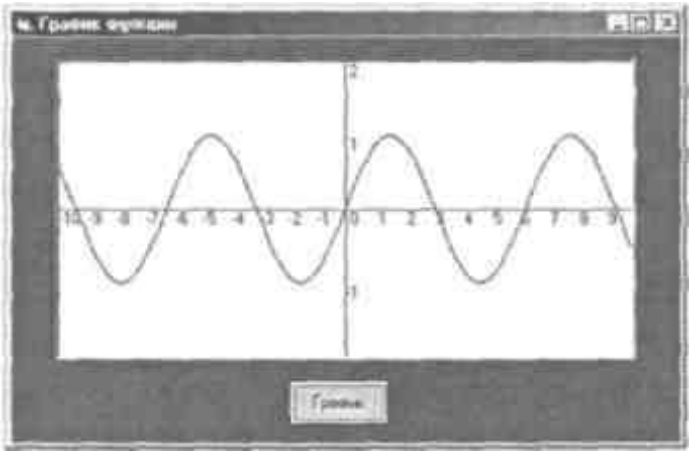
1. Разместить на форме графическое поле picGraph, в котором будет производиться построение графика.

Для большей понятности программного кода будем вводить в него комментарии, которые начинаются с символа апостроф «'»:

2. Разместить на форме кнопку cmd1 и создать событийную процедуру построения графика, в которой устанавливается масштаб, в цикле осуществляется построения графика функции, рисуются оси координат и выводятся на них числовые шкалы.

```
Dim sngX As Single, intI As Integer
Sub cmd1_Click()
'Задание масштаба
picGraph.Scale (-10, 2)-(10, -2)
'Построение графика
For sngX = -10 To 10 Step 0.01
picGraph.PSet (sngX, Sin(sngX))
Next sngX
'Ось X
picGraph.Line (-10, 0)-(10, 0)
For intI = -10 To 10
picGraph.PSet (intI, 0)
picGraph.Print intI
Next intI
'Ось Y
picGraph.Line (0, 2)-(0, -2)
For intI = -2 To 2
picGraph.PSet (0, intI)
picGraph.Print intI
Next intI
End Sub
```

3. Запустить проект и щелкнуть по кнопке *График*.



Проект « График функции» хранится в папке \VB\Graph CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

1.12. Создать проект «График функции-2» построения графиков функций tgx и x^3 и вертикальных линий координатной сетки.

1.13. Создать проект «Графический редактор», который позволяет нарисовать в графическом поле все графические примитивы (точку, линию, прямоугольник, закрашенный прямоугольник, окружность).

1.14. Создать проект «Установка цвета», позволяющий устанавливать цвета различными способами (с помощью цветовых констант, функции `QBColor()` и функции `RGB()`) и демонстрирующий заданный цвет.

1.8. Массивы в языке программирования Visual Basic

1.8.1. Числовые массивы: заполнение и поиск

Типы массивов и объявление массива. Массив является набором однотипных переменных, объединенных одним именем. Массивы бывают **одномерные**, которые можно представить в форме одномерной таблицы, и **двумерные**, которые можно представить в форме двумерной таблицы.

Массив состоит из пронумерованной последовательности элементов. Номера в этой последовательности определяются **индексом**, который может принимать целочисленные значения. Каждый из этих элементов является переменной, т. е. обладает именем и значением, и поэтому массив можно назвать **переменной с индексом**.

Массивы могут быть различных типов: числовые, строковые и т. д.

Объявление массива производится аналогично объявлению переменных, необходимо только дополнительно указать диапазон изменения индекса. Например, объявление одномерного целочисленного массива, содержащего 100 элементов, производится следующим образом:

```
Dim intA(1 To 100) As Integer
```

Обращение к элементу массива производится по его имени, состоящему из имени массива и значения индекса, например: `intA(5)`.

Заполнение массива случайными числами. Для начала работы с массивом необходимо его предварительно заполнить, т. е. присвоить элементам массива определенные значения. Заполним числовой массив `bytA(bytI)` целыми случайными числами в интервале от 1 до 100.

Для генерации последовательности случайных чисел используем функцию `Rnd()`. При запуске программы функция `Rnd()` дает равномерно распределенную псевдослучайную (т. е. каждый раз повторяющуюся) последовательность чисел в интервале $0 \le X < 1$.

Для получения последовательности случайных чисел в заданном интервале $A \le X < B$ необходимо использовать следующую формулу:

```
(B-A)*Rnd + A.
```

Тогда, получение целочисленной последовательности случайных чисел на интервале $0 \le X < 100$ достигается использованием функции выделения целой части числа:

```
Int(Rnd * 100).
```

Для генерации различающихся между собой последовательностей случайных чисел рекомендуется использовать оператор `Randomize`.

Проект «Поиск минимального элемента в числовом массиве»

1. Поместить на форму `frm1` кнопку `cmd1` и создать для нее событийную процедуру `cmd1_Click()`, реализующую заполнение массива случайными числами:

```
Dim bytA(1 To 100), bytI As Byte
'Заполнение массива
Randomize
```

```
Sub cmd1_Click()
For bytI = 1 To 100
bytA(bytI) = Int(Rnd * 100)
Next bytI
End Sub
```

Поиск в числовом массиве. Часто бывает необходимо произвести поиск наименьшего или наибольшего элемента в числовом массиве.

Осуществим поиск наименьшего элемента в числовом массиве, состоящем из 100 элементов.

Будем хранить значение минимального элемента в переменной bytMin, а его индекс — в переменной bytN. Будем считать, что сначала минимальный элемент равен первому элементу массива bytA(1), поэтому присвоим переменной bytMin его значение.

Затем в цикле сравним последовательно значения элементов массива со значением переменной bytMin, если какой-либо элемент окажется меньше, присвоим его значение переменной bytMin, а его индекс присвоим переменной bytN. Результат поиска выведем на форму.

2. Поместить на форму кнопку cmd2 и создать для нее событийную процедуру cmd2_Click(), реализующую поиск:

```
Dim bytA(1 To 100), bytI, bytN, bytMin As Byte
'Поиск минимального элемента
Sub cmd2_Click()
bytMin = bytA(1)
bytN = 1
For bytI = 2 To 100
If bytA(bytI) < bytMin Then bytMin =
bytA(bytI): bytN = bytI
Next bytI
frm1.Print "Минимальный элемент   "; bytMin;
"его индекс"; bytN
End Sub
```

3. Запустить проект. Щелкнуть по кнопкам *Заполнить массив* и *Найти Min элемент* несколько раз.

На форму будут выведены результаты поиска минимального элемента для различных вариантов заполнения массива.

Проект «Поиск минимального элемента в числовом массиве» хранится в папке \VB\ArrayS CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

1.15. Создать проект «Поиск максимального элемента в числовом массиве», предусматривающий заполнение массива случайными числами и возможность просмотра элементов массива в текстовом поле.

1.8.2. Сортировка числовых массивов

Под сортировкой массива понимается процесс перестановки элементов массива, целью которого является размещение элементов массива по возрастанию (или по убыванию) их значений.

Пусть у нас имеется целочисленный числовой массив `bytA(bytI)`, состоящий из десяти элементов и заполненный случайными числами. Представим массив в виде таблицы (табл. 1.5).

Таблица 1.5. **Алгоритм сортировки массива по возрастанию**

bytI	1	2	3	4	5	6	7	8	9	10	Значение счетчика цикла
bytA(bytI)	70	53	57	28	30	77	1	76	81	70	До сортировки
	70 ←——————→ 1										1
		53 ←——→ 28									2
			57 ←——→ 30								3
				53							4
					57						5
						77 ↔ 70					6
							77 ←——→ 70				7
								76			8
									81 ↔ 77		9
bytA(bytI)	1	28	30	53	57	70	70	76	77	81	После сортировки

Идея алгоритма сортировки состоит в следующем:

1. Проводим поиск минимального элемента во всем массиве — с 1-го по 10-й элемент. Далее меняем найденный минимальный элемент местами с 1-м элементом (выполняем перестановку).

2. Проводим поиск минимального элемента среди элементов со 2-го по 10-й. Выполняем перестановку: меняем местами найденный элемент со 2-м элементом.

3. Повторяем процедуру поиска минимального элемента среди оставшихся неупорядоченных элементов многократно и каждый раз делаем перестановку.

Повторение поиска элементов реализуем с помощью цикла со счетчиком, максимальное значение которого составляет $N-1$, где N — количество элементов массива (в рассматриваемом случае цикл повторяется 9 раз). В результате массив упорядочивается.

Поиск минимального элемента массива проводится многократно, поэтому реализуем его как общую процедуру МинЭлемент (bytI, bytN **As Byte**), где:

bytI — входной параметр (номер минимального элемента массива после предыдущего шага);

bytN — выходной параметр (номер минимального элемента массива после данного шага).

Таким образом, программный модуль формы должен содержать:

- событийную процедуру заполнения массива случайными числами;
- общую процедуру поиска минимального элемента;
- событийную процедуру сортировки.

Проект «Сортировка числового массива по возрастанию»

1. Поместить на форму frm1 кнопку cmd1 и создать для нее событийную процедуру cmd1_Click(), реализующую заполнение массива случайными числами.
2. Определить переменные для всего программного модуля. Преобразовать событийную процедуру из проекта «Поиск в числовом массиве» в общую процедуру МинЭлемент(bytI, bytN **As Byte**):

```
Dim bytA(1 To 10), bytMin, bytI, bytJ, bytK,
bytR, bytN As Byte
'Общая процедура поиска минимального элемента
Sub МинЭлемент(bytI, bytN As Byte)
bytMin = bytA(bytI)
bytN = bytI
For bytJ = bytI + 1 To 10
If bytA(bytJ) < bytMin Then bytMin =
bytA(bytJ): bytN = bytJ
Next bytJ
End Sub
```

3. Создать событийную процедуру сортировки. Для осуществления перестановки использовать промежуточную переменную bytR. Для визуализации процесса сортировки для каждого цикла перестановки элементов (цикл по переменной bytI) в цикле по переменной bytK выводить в текстовое поле txtSort значения элементов массива.

```
'Событийная процедура сортировки
Private Sub cmd2_Click()
txtSort.Text = ""
For bytI = 1 To 9
'Вызов общей процедуры поиска минимального
элемента
Call МинЭлемент(bytI, bytN)
'Перестановка
bytR = bytA(bytI)
bytA(bytI) = bytA(bytN)
bytA(bytN) = bytR
'Печать массива для каждого цикла перестановки
For bytK = 1 To 10
txtSort.Text = txtSort.Text + Str(bytA(bytK))
Next bytK
Next bytI
End Sub
```

4. Запустить проект. Щелкнуть по кнопкам *Заполнить массив* и *Сортировать*.
В текстовом поле будет реализована визуализация процесса сортировки числового массива по шагам.

Проект «Сортировка числового массива по возрастанию» хранится в папке \VB\ArraySort — CD-ROM

Практические задания для самостоятельного выполнения — CD-ROM

1.16. Создать проект «Сортировка числового массива по убыванию», в котором реализуется сортировка с использованием общей процедуры поиска максимального элемента, предусмотреть возможность просмотра начального состояния массива и процесса сортировки в текстовых полях.

Глава 2

Основы объектно-ориентированного программирования на языке Delphi

2.1. Графический интерфейс системы объектно-ориентированного программирования Delphi

Установить систему программирования Delphi 6.0 — **CD-ROM** 

Окно системы программирования Delphi. Система программирования Delphi предоставляет пользователю удобный графический интерфейс в процессе разработки приложения. После запуска Delphi появится окно системы программирования Delphi (рис. 2.1), которое включает в себя:

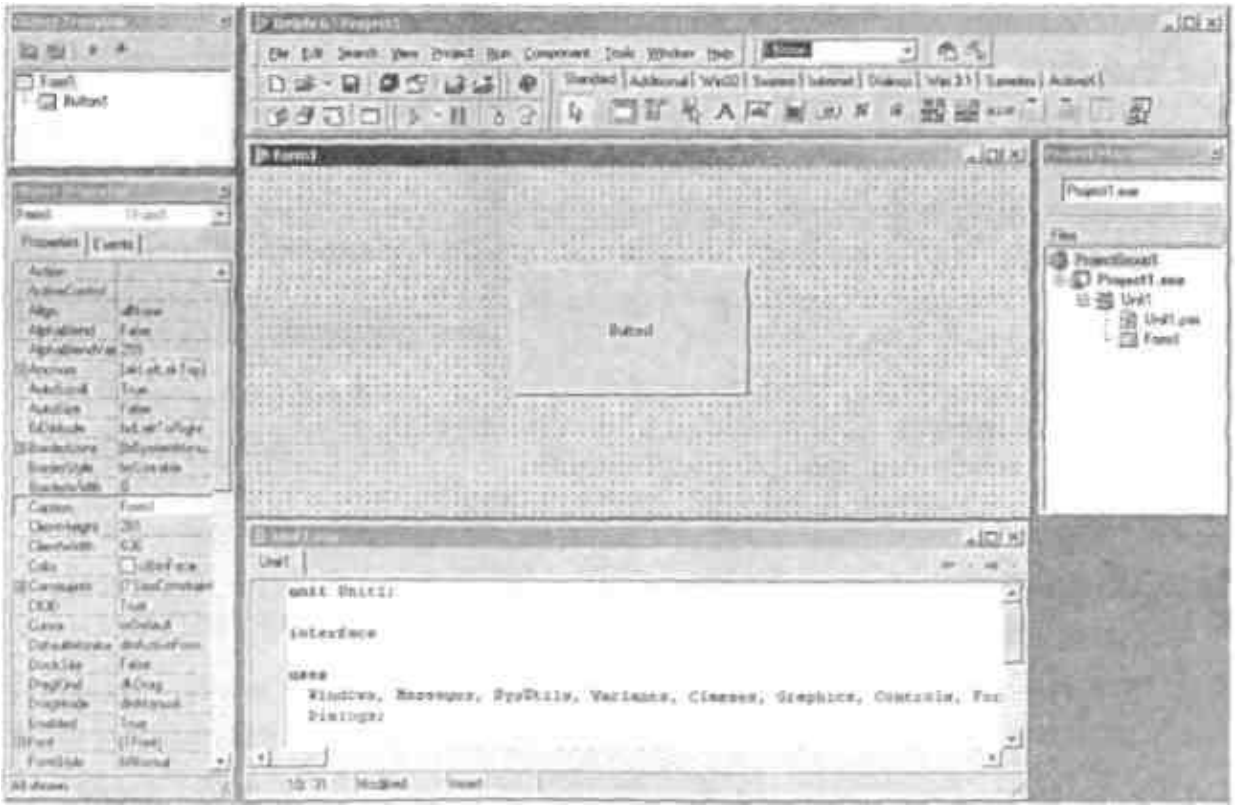


Рис. 2.1. Система программирования Delphi

- строку заголовка *Delphi 6 – Project1*, в которой после имени среды программирования *Delphi 6* указано имя проекта *Project1*.
- Под строкой заголовка располагается строка *Главного меню*.
- Под строкой *Главного меню* слева располагаются кнопки с пиктограммами наиболее часто используемых команд.

Окно *Конструктор форм*. В центре располагается окно *Конструктор форм*, в котором происходит визуальное конструирование графического интерфейса разрабатываемого проекта (рис. 2.2). Размеры формы можно менять, перетаскивая мышью правую или нижнюю границу формы.

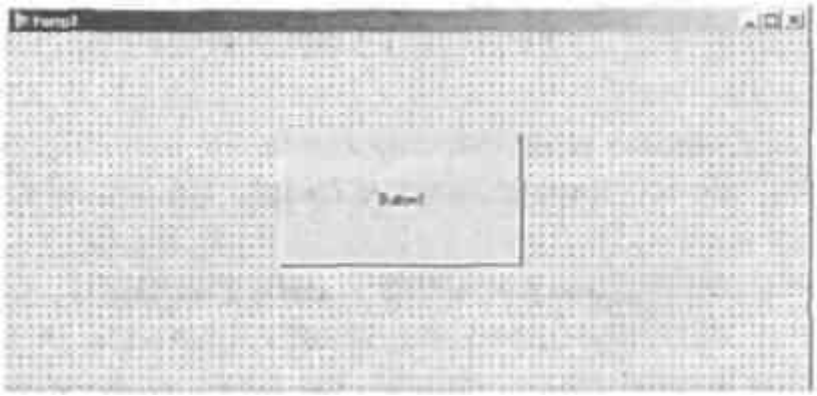


Рис. 2.2. Окно *Конструктор форм*

Окно *Конструктор форм* вызывается командой [*View-Forms...*].

Первоначально форма пуста, в дальнейшем в процессе создания графического интерфейса приложения на ней размещаются управляющие элементы.

Окно *Программный код*. С формой связан программный модуль, содержащий программные коды процедур. Для ввода и редактирования текста программы служит окно *Программный код* (в данном случае *Unit1.pas*) (рис. 2.3), которое вызывается командой [*View-Units...*].

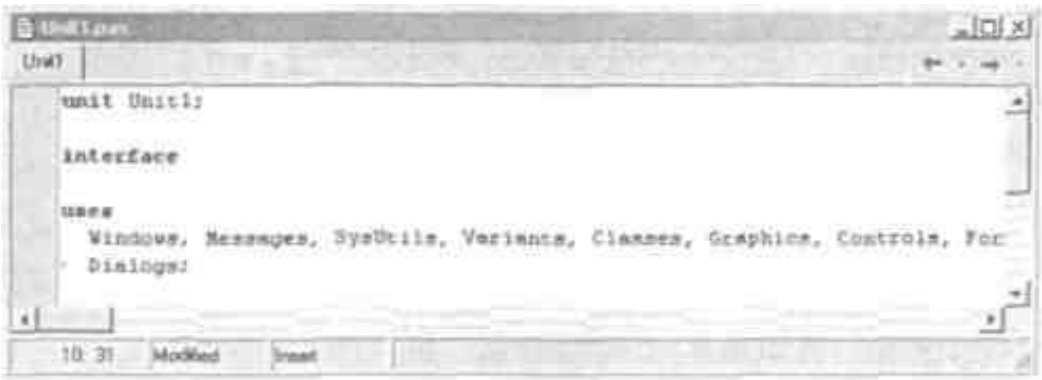


Рис. 2.3. Окно *Программный код*

Панель инструментов. Под строкой *Главного меню* справа располагается *Панель инструментов* (рис. 2.4), содержащая пиктограммы управляющих элементов. Стандартный набор управляющих элементов размещается на вкладке *Standard* и включает в себя 16 классов объектов: командная кнопка (`Button`), текстовое поле (`Edit`), надпись (`Label`) и т. д.

Рис. 2.4. *Панель инструментов*

На вкладках *Additional*, *Win32* и других располагаются дополнительные управляющие элементы: графическое поле (`Image`), список изображений (`ImageList`) и др.

Выбрав щелчком мышью по *Панели инструментов* нужный элемент, мы можем поместить его на форму проектируемого приложения. Процесс размещения на форме управляющих элементов аналогичен рисованию графических примитивов с использованием графического редактора.

Фактически мы размещаем на форме экземпляры определенных классов объектов. Например, выбрав класс объектов `Button`, мы можем разместить на форме неограниченное количество экземпляров этого класса, т. е. командных кнопок `Button1`, `Button2`, `Button3` и т. д.

Рис. 2.5. Окно *Свойства объекта*

Окно *Свойства объекта*. Слева посередине располагается окно *Свойства объекта* (*Object Inspectors*) (рис. 2.5). В верхней части окна размещается список объектов, на вкладках содержатся список свойств (*Properties*) и список событий (*Events*), относящихся к выбранному объекту (форме или управляющему элементу на форме). На рисунке выбран объект `Form1` из класса `TForm`.

На вкладке *Properties* список свойств разделен на две колонки. В левой находятся имена свойств, в правой — их значения. Установленные по

умолчанию значения могут быть изменены. Свойством объекта является количественная или качественная характеристика этого объекта (размеры, цвет, шрифт и др.).

Для некоторых свойств предусмотрена возможность выбора значений из раскрывающегося списка значений, например, из списка можно выбрать значение цвета фона формы (свойство `Color`).

На вкладке *Events* список событий разделен на две колонки. В левой находятся имена событий, в правой можно ввести их значения.

Окно *Свойства объекта* вызывается командой [*View-Object Inspector*].

Окно *Проводник проекта*. Окно *Проводник проекта (Project Manager)* (рис. 2.6) располагается в правом верхнем углу окна системы программирования Delphi.

В нем отображаются в виде иерархического каталога все составные части текущего проекта (в данном случае *Project1*). Между ними можно переключаться.

Окно *Проводник проекта* вызывается командой [*View-Project Manager*].

Рис. 2.6. Окно *Проводник проекта*

Окно *Дерево объектов*. В левом верхнем углу располагается окно *Дерево объектов (Object TreeView)* (рис. 2.7), отображающее перечень объектов, размещенных на форме.

Окно *Дерево объектов* вызывается командой [*View-Object TreeView*].

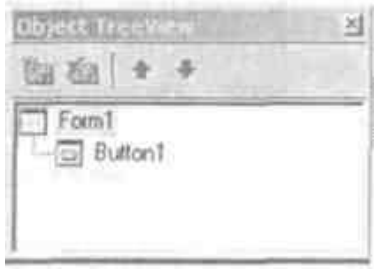


Рис. 2.7. Окно *Дерево объектов*

Вопросы для размышления

кие основные окна интегрированной среды разработки Delphi вы можете назвать? Каково их назначение?

Практические задания для самостоятельного выполнения CD-ROM

2.1. Запустить систему объектно-ориентированного программирования Delphi и выполнить работу «Знакомство с графическим интерфейсом системы программирования Delphi».

2.2. Этапы разработки приложения на языке Delphi

Создание приложения в среде Delphi можно условно разделить на несколько этапов:

1. **Создание графического интерфейса будущего приложения.** С помощью *Панели инструментов* на форму помещаются управляющие элементы, которые должны обеспечить взаимодействие приложения с пользователем.
2. **Задание значений свойств объектов графического интерфейса.** С помощью окна *Свойства объекта* задаются значения свойств управляющих элементов, помещенных ранее на форму.
3. **Создание и редактирование программного кода.** Для создания заготовки событийной процедуры необходимо осуществить двойной щелчок мышью по управляющему элементу. В окне *Редактор кода* появится заготовка событийной процедуры, имя которой состоит из двух частей: имени формы, содержащей управляющий элемент, и имени объекта и имени события (например: `TForm1.Button1Click`). Затем в окне *Редактор кода* производится ввод и редактирование программного кода процедуры.
4. **Сохранение проекта.** Так как проект включает в себя несколько файлов, рекомендуется для каждого проекта создать отдельную папку на диске. Сохранение проекта производится с помощью меню *File*:
   - сначала необходимо сохранить форму и связанный с ней программный модуль (файл с расширением pas) с помощью команды *Save As...* По умолчанию для файла формы предлагается имя Unit1.pas;
   - далее необходимо сохранить файл главного модуля, который содержит описание проекта (файл с расширением dpr) с помощью команды *Save Project As...*;

- в процессе сохранения в папку проекта записываются вспомогательные файлы: файл с расширением res, описывающий ресурсы; файл с расширением dfm, описывающий форму, и некоторые другие файлы.

5. **Компиляция проекта в приложение.** Сохраненный проект может выполняться только в самой системе программирования Delphi. Для того чтобы преобразовать проект в приложение, которое может выполняться непосредственно в среде операционной системы, необходимо сохранить проект в исполнимом файле (типа exe). Для компиляции проекта в исполнимый файл используется команда [*Project-Compile*].

Вопросы для размышления

1. Какие основные этапы разработки проекта на языке Delphi вы можете назвать? Каков порядок сохранения проекта?

2.3. Создание первого проекта «Обычный калькулятор» на языке Delphi

Проект «Обычный калькулятор». Разработаем проект «Обычный калькулятор», который будет производить четыре арифметических действия над числами (сложение, вычитание, умножение и деление).

Проект «Обычный калькулятор»

1. Запустить систему программирования Delphi.

Работа над проектом начинается с создания графического интерфейса, для этого в окне *Конструктор форм* на форму помещаются управляющие элементы. Для создания графического интерфейса проекта разместим на форме два текстовых поля для ввода числовых данных и одну метку для вывода результата, а также четыре кнопки для реализации событийных процедур: сложения, вычитания, умножения, деления.

2. С помощью *Панели инструментов* поместить на форму Form1 текстовые поля Edit1 и Edit2, метку Label1 и командные кнопки Button1, Button2, Button3 и Button4.

Далее необходимо задать новые значения свойств управляющих элементов.

3. Последовательно выделить все объекты и с помощью окна *Свойства объекта* изменить значения некоторых свойств формы и управляющих элементов согласно таблице:

Объект	Свойство	Значение по умолчанию	Новое значение
Form1	Caption	Form1	Обычный калькулятор
Edit1	Text	Edit1	
Edit2	Text	Edit2	
Label1	Caption	Label1	
Button1	Caption	Button1	+
Button2	Caption	Button2	-
Button3	Caption	Button3	*
Button4	Caption	Button4	/

Следующим шагом является создание программного кода событийных процедур. Двойной щелчок мышью по кнопке, для которой надо создать пограммный код, вызывает окно *Программный код* с пустой заготовкой событийной процедуры.

4. Осуществить двойной щелчок по кнопке Button1, появится заготовка событийной процедуры TForm1.Button1Click:

```
procedure TForm1.Button1Click(Sender: Tobject);
begin

end;
```

Событийная процедура сложения TForm1.Button1Click должна изменить значение свойства Caption метки label1 так, чтобы оно являлось суммой числовых значений свойства Text текстовых полей Edit1 и Edit2.

Сначала для преобразования строковых значений, вводимых в текстовые поля, в десятичные числа, воспользуемся функцией StrToFloat().

Затем для преобразования полученной суммы в числовой форме в строковую воспользуемся функцией FloatToStr().

5. Код событийной процедуры, реализующей сложение чисел, будет следующим:

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
Label1.Caption := FloatToStr(StrToFloat
(Edit1.Text) + StrToFloat(Edit2.Text));
end;
```

Событийные процедуры вычитания, умножения и деления создаются аналогично.

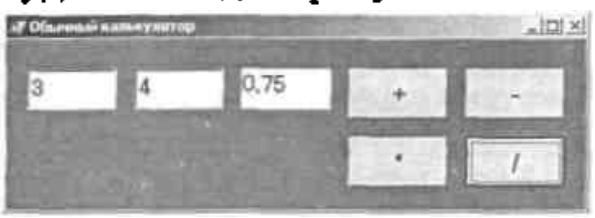
6. Для каждой из кнопок ввести программные коды событийных процедур.

Сделаем внешний вид проекта более привлекательным. Для этого изменим свойства объектов, определяющие их внешний вид (цвет фона формы, цвет, размер и выравнивание шрифта на метке и в текстовых полях).

7. Активизировать форму Form1. В окне *Свойства объекта* выбрать свойство BackColor (цвет фона) и двойным щелчком открыть диалоговое окно с цветовой палитрой. Выбрать цвет, например, *синий*.
8. Активизировать метку Label1. В окне *Свойства объекта* для свойств установить значения:
   - BackColor — *белый*,
   - Font — *размер шрифта 14*,
   - Alignment — *Right Justify*.
9. Активизировать текстовые поля Edit1 и Edit2. В окне *Свойства объекта* для свойств установить значения свойства:
   - Font — *размер шрифта 14*.

   Сохранить проект.
10. Сначала необходимо сохранить форму и связанный с ней программный модуль с помощью пункта меню *Save As...* в файле с именем Unit1.pas.
11. Далее необходимо сохранить файл главного модуля, который содержит описание проекта, с помощью пункта меню *Save Project As...* в файле с именем Calc.dpr.
12. Запустить проект на выполнение. Ввести числа в два текстовых поля и щелкнуть по кнопке арифметической операции.

На метку будет выведен результат.



Проект «Обычный калькулятор» хранится в папке \Delphi\Calc CD-ROM

2.4. Переменные в языке программирования Delphi

Тип переменной. Тип переменной определяется набором допустимых значений (данных) переменных и допустимыми операциями над этими значениями. Значениями переменных числовых типов (**Byte**, **Integer**, **Real**, **Single**, **Double**) являются числа, логических (**Boolean**) — **True** и **False**, строковых (**Char**) — последовательности символов и т. д.

Различные типы требуют для хранения данных в оперативной памяти компьютера различное количество ячеек (байтов) (табл. 2.1).

Таблица 2.1. **Типы переменных**

Тип переменной	Возможные значения	Объем занимаемой памяти
`Byte`	Целые неотрицательные числа от 0 до 255	1 байт
`Smallint`	Целые числа от −32 768 до 32 767	2 байта
`Integer`	Целые числа от − 2 147 483 648 до 2 147 483 647	4 байта
`Int64`	Целые числа -2^{63} до $2^{63}-1$	8 байтов
`Single`	Десятичные числа одинарной точности (7–8 значащих цифр) от $-1.5 \cdot 10^{-45}$ до $3.4 \cdot 10^{38}$	4 байта
`Real`	Десятичные числа (11–12 значащих цифр) от $-2.9 \cdot 10^{-39}$ до $1.7 \cdot 10^{38}$	6 байтов
`Double`	Десятичные числа двойной точности (15–16 значащих цифр) от $-5.0 \cdot 10^{-324}$ до $1.7 \cdot 10^{308}$	8 байтов
`Extended`	Десятичные числа расширенной точности (19–20 значащих цифр) от $-3.6 \cdot 10^{-4951}$ до $1.1 \cdot 10^{4932}$	10 байтов
`Boolean`	Логическое значение `True` или `False`	2 байта

Тип переменной	Возможные значения	Объем занимаемой памяти
Char	Символ в кодировке Windows	1 байт на каждый символ
WideChar	Символ в кодировке Unicode	2 байта на каждый символ
String	Строка длиной до 256 символов	256 байтов

Имя переменной. Имя каждой переменной (идентификатор) уникально и не может меняться в процессе выполнения программы. Имя переменной может состоять из различных символов (латинские буквы, цифры и т. д.), но должно обязательно начинаться с буквы и не должно включать знак «.» (точка).

Объявление переменной. В языке программирования Delphi каждая переменная должна быть *обязательно объявлена*! Для объявления переменной используется инструкция следующего вида:

```
ИмяПеременной : ТипПеременной;
```

Переменные, значения которых не меняются в процессе выполнения программы, называются **константами.** Инструкция объявления констант имеет следующий вид:

```
ИмяКонстанты = ЗначениеКонстанты;
```

Присваивание переменным значений. Переменная может получить или изменить значение с помощью **инструкции присваивания.** Инструкция присваивания имеет следующий вид:

```
ИмяПеременной := Выражение;
```

При выполнении инструкции присваивания переменная, имя которой указано слева от знака присваивания, получает значение, равное значению выражения (арифметического, строкового или логического), которое находится справа от знака присваивания.

Проект «Переменные». Создадим проект, который позволит продемонстрировать использование переменных различных типов, арифметических, строковых и логических выражений и операции присваивания.

Проект «Переменные»

1. Создать новый проект. Для создания графического интерфейса разместить на форме `Form1` командную кнопку `Button1` и метки `Label1`, `Label2`, `Label3` и `Label4`.

Во-первых, произведем деление двух целых чисел. Аргументами программы являются две неотрицательные целочисленные переменные `bytA` и `bytB`. Сначала произведем деление нацело с использованием операции **div**, результат присвоим целочисленнаой переменной `intC`, а затем деление с заданной точностью, результат присвоим переменной двойной точности `dblD`. Для вывода результатов на метки `Label1` и `Label2` необходимо использовать функции преобразования типов данных `IntToStr(intC)` и `FloatToStr(dblD)`.

Во-вторых, пусть значением строковой переменной `strK` будет результат сложения (конкатенации) двух строк "Кило" и "байт", которые являются значениями строковых переменных `strE` и `strF`. Результат выведем в метку `Label3`.

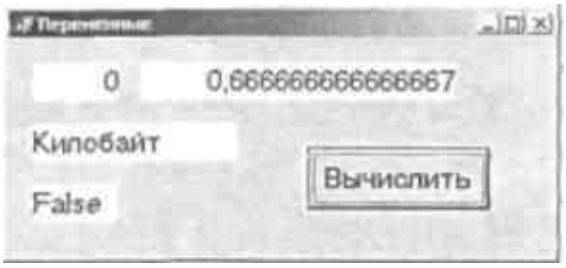
В-третьих, объявим логические переменные `blnA` и `blnB`, присвоим им значения логических выражений, в которые входят операции сравнения, и произведем операцию логического умножения над двумя логическими переменными. Результат присвоим логической переменной `blnC` и выведем в метку `Label4` с использованием функции преобразования типов данных `BoolToStr(blnC,True)`.

2. В окне *Программный код* ввести событийную процедуру:

```
var   //начало раздела объявления переменных
bytA: byte;
bytB: byte;
intC: integer;
dblD: double;
strE: string;
strF: string;
strK: string;
```

```
blnA: boolean;
blnB: boolean;
blnC: boolean;
procedure TForm1.Button1Click(Sender: TObject);
begin
bytA:=2;
bytB:=3;
intC:=bytA div bytB;
dblD:=bytA/bytB;
strE:='Кило';
strF:='байт';
strK:= strE + strF;
blnA := 5 > 3;
blnB := 2 * 2 = 5;
blnC := blnA and blnB;
Label1.Caption := IntToStr(intC);
Label2.Caption := FloatToStr(dblD);
Label3.Caption := strK;
Label4.Caption := BoolToStr(blnC,True);
end;
end.
```

3. После запуска проекта на экране появится его графический интерфейс.



Щелчок по кнопке *Button1* вызовет выполнение событийной процедуры, и на форму будут выведены результаты выполнения проекта.

Выведенные в две верхние метки результаты деления двух чисел различны, так как деление производится с разной точностью, которая зависит от типа переменной.

В третью метку будет выведен результат сложения строк — "Килобайт".

В четвертую будет выведено значение логической переменной `blnC` — `False` («Ложь»).

Проект «Переменные» хранится в папке \Delphi\Variant CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

2.2. Создать проект «Переменные-2» на основе проекта «Переменные», с выводом результатов в текстовые поля.

2.5. Функции в языке программирования Delphi

2.5.1. Функции преобразования типов данных

Функции преобразования реализуют преобразование данных из одного типа в другой.

Функции StrToInt() и StrToFloat(). Часто необходимо преобразовать строковое значение в числовое значение. Это можно сделать с помощью функций `StrToInt()` и `StrToFloat()`, аргументом которых является строка, а значением — целое или вещественное число:

```
StrToInt(Строка)
StrToFloat(Строка)
```

Функции `StrToInt()` и `StrToFloat()` применяются, например, при вводе чисел в текстовые поля для преобразования строкового значения свойства `Text` в число, которое затем используется в вычислениях.

Проект «Обычный калькулятор»

Функции IntToStr(), FloatToStr(), IntToHex(). Функции `IntToStr()` и `FloatToStr()` позволяют производить преобразование десятичных чисел в десятичные числа в строковой форме, а функция `IntToHex()` — переводить десятичные числа в шестнадцатеричные в строковой форме. Аргументом функции является десятичное число, а значением — строка:

```
IntToStr(Число)
FloatToStr(Число)
IntToHex(Число)
```

Проект «Перевод чисел». Создадим проект, который позволит переводить целые числа из десятичной системы счисления в шестнадцатеричную и обратно — из шестнадцатеричной в десятичную.

Проект «Перевод чисел»

1. Создать новый проект. Разместить на форме:

- два текстовых поля Edit1 и Edit2 для ввода чисел;
- две метки Label1 и Label2;
- четыре метки для вывода поясняющих надписей над текстовыми полями и метками;
- кнопку Button1 для запуска событийной процедуры.

Перевод десятичного числа в шестнадцатеричное. Для перевода десятичного числа, введенного в текстовое поле Edit1, в шестнадцатеричное число будем переводить сначала его из десятичной строковой формы в числовую с помощью функции StrToInt(), а затем из числовой в шестнадцатеричную строковую с помощью функции IntToHex().

Перевод шестнадцатеричного числа в десятичное. Для перевода шестнадцатеричного числа, введенного в текстовое поле Edit2, в десятичное число будем переводить сначала его из шестнадцатеричной строковой формы в числовую с помощью функции StrToInt(), а затем из числовой в десятичную строковую с помощью функции IntToStr().

2. В окне *Программный код* ввести событийную процедуру TForm1.Button1Click():

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
Label1.Caption :=
IntToHex(StrToInt(Edit1.Text),1);
Label2.Caption :=
IntToStr(StrToInt(Edit2.Text));
end;
```

3. Запустить проект. Ввести в первое текстовое поле десятичное число (например, 100), во второе — шестнадцатеричное число (например, $FF). Щелкнуть по кнопке *Перевести*.

Проект «Перевод чисел» хранится в папке \Delphi\DEC-HEX\ CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

2.3. Создать проект «Мультисистемный калькулятор», который позволяет производить арифметические операции над числами в десятичной и шестнадцатеричной системах счисления.

2.5.2. Математические функции

В математических функциях значениями как аргументов, так и функций являются числа. В языке программирования Delphi имеется 13 математических функций (табл. 2.2).

Таблица 2.2. **Математические функции в Delphi**

Функция	Аргумент функции x	Возвращаемое функцией значение
`Sin(x)`	Число (в радианах)	Синус числа
`Cos(x)`	Число (в радианах)	Косинус числа
`Arctan(x)`	Число	Арктангенс в радианах
`Sqr(x)`	Неотрицательное число	Квадратный корень из числа
`Sqrt(x)`	Число	Квадрат числа
`Ln(x)`	Число	Натуральный логарифм числа
`Exp(x)`	Число	Экспонента числа
`Random(x)`	Число	Псевдослучайное число N ($0 \le N < x$)
`Int(x)`	Число	Наибольшее целое, не превышающее значение аргумента
`Round(x)`	Число	Целое, ближайшее к значению аргумента
`Trunc(x)`	Число	Целая часть аргумента
`Frac(x)`	Число	Дробная часть аргумента
`Abs(x)`	Число	Модуль числа

Проект «Инженерный калькулятор». Воспользуемся математическими функциями для расширения возможностей проекта «Обычный калькулятор» и превращения его в проект «Инженерный калькулятор».

Проект «Инженерный калькулятор»

1. Открыть проект «Обычный калькулятор». Добавить на форму пять кнопок `Button1` — `Button5`.

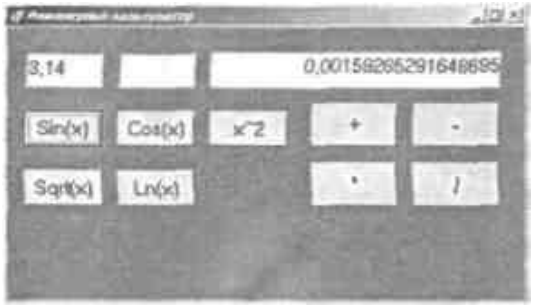
Для этих кнопок создадим событийные процедуры, реализующие вычисление соответствующих функций: синуса, косинуса, квадрата, квадратного корня и натурального логарифма.

Для преобразования строкового значения числа, вводимого в текстовые поля, в числовую форму воспользуемся функцией `StrToFloat()`, а затем для преобразования числа в строковую форму для вывода на метку используем функцию `FloatToStr()`.

2. Например, для вычисления синуса числа событийная процедура `TForm1.Button5Click()` примет вид:

```
procedure TForm1.Button5Click(Sender: Tobject);
begin
Label1.Caption :=
FloatToStr(Sin(StrToFloat(Edit1.Text)));
end;
```

3. Ввести самостоятельно программный код других событийных процедур с использованием встроенных функций языка Delphi: `Cos()`, `Sqr()`, `Sqrt()` и `Ln()`.
4. Запустить проект на выполнение. Произвести вычисление, например, синуса числа. Ввести число 3,14 и щелкнуть по кнопке *Sin(x)*.



Проект «Инженерный калькулятор» хранится в папке *\Delphi\Calc* **CD-ROM**

Практические задания для самостоятельного выполнения **CD-ROM**

2.4. Создать проект «Треугольник», позволяющий вычислить гипотенузу и площадь прямоугольного треугольника, если известны его катеты.

2.5.3. Строковые функции

В строковых функциях либо аргументы, либо возвращаемые функциями значения являются строками.

Конкатенация строк. Операция конкатенации состоит в объединении строк или значений строковых переменных в единую строку. Операция конкатенации обозначается знаком «+» или выполняется с помощью функции `Concat()`. Синтаксис функции:

```
Concat (Строка1, ..., СтрокаN)
```

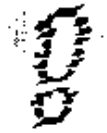
Функция определения длины строки. В функции определения длины строки `Length(Строка)` аргументом является строка `Строка`, а возвращает функция числовое значение длины строки (количество символов в строке). Синтаксис функции:

```
Length(Строка)
```

Функция вырезания подстроки. В функции `Copy(Строка, НомерСимвола, Длина)` аргументами являются строка `Строка` и числа или целочисленные переменные `НомерСимвола` и `Длина`. Функция возвращает строковое значение, равное вырезанной подстроке. Синтаксис функции:

```
Copy(Строка, НомерСимвола, Длина)
```

Функция Chr. Функция `Chr` осуществляет преобразование числового кода символа в символ. Аргументом функции является целое число, а значением — символ. Синтаксис функции:



```
Chr(Число)
```

Проект «Строковый калькулятор». Создадим строковый калькулятор, который позволит производить различные преобразования строк.

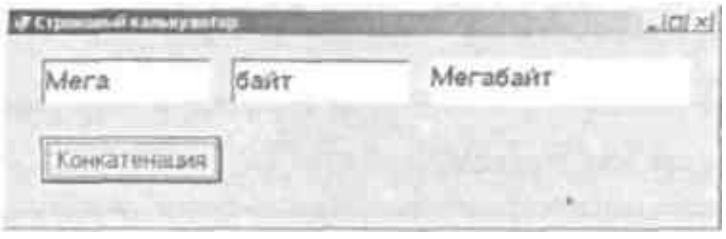
Проект «Строковый калькулятор»

Создадим событийную процедуру, реализующую сложение (конкатенацию) двух строк.

1. Разместить на форме:
   - два текстовых поля `Edit1` и `Edit2` для ввода строк;
   - метку `Label1` для вывода результата;
   - кнопку `Button1`.
2. Для кнопки ввести программный код событийной процедуры `TForm1.Button1Click()`, реализующий операцию конкатенации:

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
Label1.Caption := Edit1.Text + Edit2.Text;
end;
```

3. Запустить проект, в два первых поля ввести строки и щелкнуть по кнопке *Конкатенация*. В третьем поле появится результат сложения двух строк.

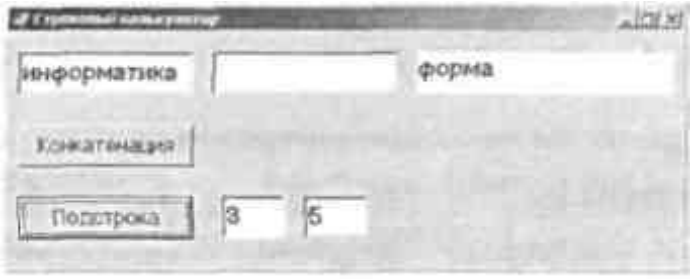


Воспользуемся теперь для преобразования строк строковой функцией Copy(Строка, НомерСимвола, Длина). Функция вырезает из строки Строка подстроку, начиная с символа, позиция которого в строке задается целочисленной целочисленным значением НомерСимвола, длина подстроки задается целочисленным значением Длина.

4. Разместить на форме два текстовых поля Edit3 и Edit4 для ввода значений НомерСимвола и Длина и кнопку Button2.
5. Для кнопки ввести программный код событийной процедуры TForm1.Button2Click(), реализующий операцию вырезания подстроки. Для преобразования строковых значений свойства Text текстовых полей в числа использовать функцию StrToInt():

```
procedure TForm1.Button2Click(Sender: Tobject);
begin
Label1.Caption := Copy(Edit1.Text,
StrToInt(Edit3.Text),
StrToInt(Edit4.Text));
end;
```

6. Запустить проект, в первое поле ввести строку, в поля функции вырезания подстроки ввести числа и щелкнуть по кнопке *Подстрока*. На метке появится вырезанная подстрока.

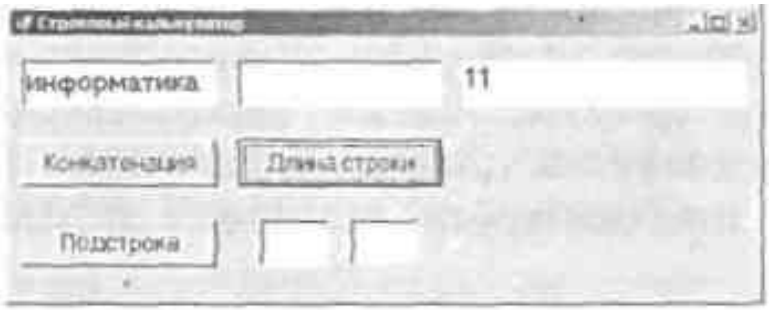


Для определения количества символов в строке воспользуемся функцией определения длины строки Length(), аргументом которой является строка, а возвращает функция число, равное количеству символов в строке.

7. Разместить на форме кнопку Button3 и ввести программный код событийной процедуры TForm1.Button3Click(), реализующий операцию определения количества символов в строке:

```
procedure TForm1.Button3Click(Sender: Tobject);
begin
Label1.Caption := IntToStr(Length(Edit1.Text));
end;
```

8. Запустить проект, в первое поле ввести строку и щелкнуть по кнопке *Длина строки*. На метке появится число символов в строке.



Проект «Строковый калькулятор» хранится в папке \Delphi\CalcString **CD-ROM**

Практические задания для самостоятельного выполнения **CD-ROM**

2.5. Создать проект «Строковый калькулятор-2» на основе проекта «Строковый калькулятор». Добавить возможность определения числового кода символа.

2.5.4. Функции ввода и вывода данных

Функция `InputBox()`. Функция `InputBox()` позволяет вводить данные с помощью диалоговой панели ввода. Аргументами этой функции являются три строки, а значением функции — строка по умолчанию или строка, введенная пользователем. Синтаксис функции следующий:

```
Переменная := InputBox('Заголовок',
'Подсказка',['ЗначениеПоУмолчанию'])
```

В результате выполнения этой функции появляется диалоговая панель с текстовым полем (рис. 2.8). В строку заголовка панели будет выведено значение первого аргумента

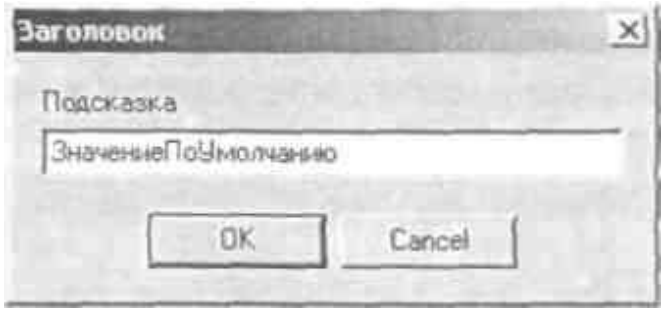


Рис. 2.8. Диалоговая панель ввода функции `InputBox()`

'Заголовок', на саму панель — значение второго аргумента 'Подсказка', в текстовое поле — значение аргумента 'ЗначениеПоУмолчанию' (если это значение отсутствует, содержимое текстового окна также отсутствует).

Если пользователь щелкнет по кнопке *OK*, то значением функции станет строка, введенная пользователем в текстовое поле. Если пользователь щелкнет по кнопке *Cancel*, то значением функции станет строка 'ЗначениеПоУмолчанию'.

Функция MessageDlg(). Функция MessageDlg() позволяет выводить сообщения не на форму, а на специальную **панель сообщений**, на которой можно разместить определенный набор кнопок и информационный значок о типе сообщения. Кроме того, функция MessageDlg() получает определенное значение, которое может быть присвоено целочисленной переменной. Синтаксис функции следующий:

```
Переменная := MessageDlg('Сообщение',Тип,
                [Кнопки],Справка)
```

Аргумент 'Сообщение' выводится на панель сообщений.

Аргумент Тип задается именованной константой и определяет вид информационного значка, который помещается на панель сообщений (табл. 2.3).

Таблица 2.3. **Константы, задающие тип панели сообщений**

Константа	Значок	Тип сообщения
MtError		Ошибка
MtConfirmation		Вопрос
MtWarning		Внимание
MtInformation		Информация

Аргументы Кнопки представляют собой набор именованных констант, разделенных запятыми, заключенный в квадратные кнопки, и определяют набор кнопок, размещаемых на панели (табл. 2.4).

Таблица 2.4. **Константы, определяющие набор кнопок на панели сообщений**

Константа	Набор кнопок
mbOk	*OK*
mbYes	*Yes* (Да)
mbNo	*No* (Нет)
mbOkCancel	*OK*, *Cancel* (Отмена)
mbAbortRetryIgnore	*Abort* (Стоп), *Rertry* (Повтор), *Ignore* (Пропустить)
mbYesNoCancel	*Yes* (Да), *No* (Нет), *Cancel* (Отмена)

Аргумент `Справка` — параметр, определяющий раздел справочной системы, который вызывается нажатием клавиши {F1}. Если вывод справки не предусмотрен, то значение аргумента `Справка` должно быть равно 0.

Например, если задать функцию с аргументами `MessageDlg('Сообщение',MtWarning,mbYesNoCancel,0)`, то будет выведена панель сообщений, показанная на рис. 2.9.

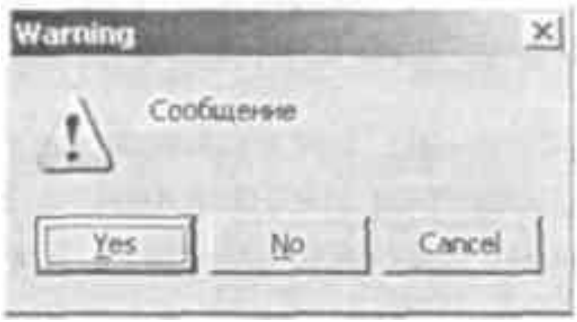


Рис. 2.9. *Диалоговая панель сообщений функции* `MessageDlg()`

Щелчок по кнопке приводит к вычислению значения функции, которое зависит от этой кнопки. Значение, возвращаемое функцией `MessageDlg()`, позволяет определить, по какой из кнопок был произведен щелчок (какая кнопка была «нажата») (табл. 2.5).

Таблица 2.5. **Значения функции `MessageDlg()`**

«Нажатая» кнопка	Значения функции
OK	idOk
Yes	idYes
No	idNo
Cancel	idCancel
Abort	idAbort
Retry	idRetry
Ignore	idIgnore

Проект «Проверка знаний». Разработаем проект, который позволит контролировать знания. Сначала реализуем регистрацию проверяемого с использованием функций `InputBox()` и `MessageDlg()`.

Проект «Проверка знаний»

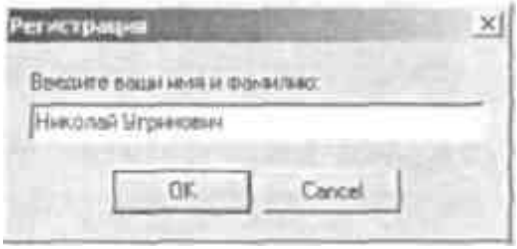
1. Разместить на форме кнопку `Button1`, задать значение `Начать проверку` свойства `Caption`. Создать событийную процедуру `TForm1.Button1Click()`.

2. С помощью функции `InputBox()` запросить имя и фамилию и присвоить ее значение строковой переменной `A`, а с помощью функции `MessageDlg()` вывести результаты регистрации:

```
var
A: string;
B: integer;
C: string;
N: integer;
procedure TForm1.Button1Click(Sender: Tobject);
begin
A := InputBox('Регистрация',
'Введите ваши имя и фамилию:','');
B := MessageDlg('Уважаемый ' + A +
', Вы готовы к проверке знаний?',
MtConfirmation, [mbYes,mbNo],0);
If B = idNo Then Exit;
end;
```

3. Запустить проект и щелкнуть по кнопке *Начать проверку*.

4. В появившемся диалоговом окне *Регистрация* ввести в текстовое поле имя и фамилию.



5. Аргумент `MtConfirmation` обеспечивает вывод информационного окна типа «Вопрос», а аргумент `[mbYes, mbNo]` — две кнопки *Yes* и *No*.

Щелчок по одной из кнопок приводит к возвращени функцией определенного значения (*Yes* — idYes, *No* — idNo), которое присваивается числовой переменной B.

6. С помощью условного оператора в краткой форме реали зуется *либо* выход из программы (щелчок по кнопк *No*), *либо* продолжение работы и переход к проверк знаний (щелчок по кнопке *Yes*).

Вопрос задается с помощью функции InputBox, провер ка правильности ответа производится с помощью операто условного перехода **If-Then-Else**.

2.6.1. Алгоритмическая структура «ветвление»

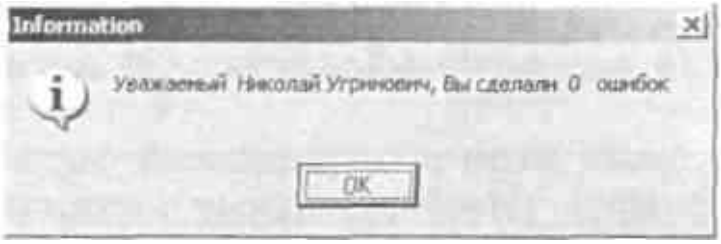
Вывод информации о правильности или неправильност ответа производится с помощью функции MessageDlg(с числовым значением третьего аргумента [mbOK], что обес печивает вывод информационной панели сообщений с одно *кнопкой OK*.

7. Ввести в событийную процедуру программный код, реа лизующий проверку знаний в виде последовательност вопросов. В целочисленной переменной N будем накап ливать количество неправильных ответов:

```
C := InputBox('Чему равен 1 байт?:',
'Первый вопрос','');
If C = '8 бит'
Then
begin
MessageDlg('Правильно!', MtWarning, [mbOK],0);
end
Else
begin
MessageDlg('Неправильно!', MtWarning,[mbOK],0);
N := N + 1;
end;
C := InputBox('Переведите десятичное число 5
в двоичную систему счисления:',
'Второй вопрос','');
If C = '101'
Then
begin
MessageDlg('Правильно!', MtWarning,[mbOK],0);
end
Else
begin
MessageDlg('Неправильно!', MtWarning,[mbOK],0);
```

```
N := N + 1;
end;
MessageDlg('Уважаемый ' + A + ', Вы сделали ',
+ IntToStr(N) + 'ошибок!',
MtInformation, [mbOK],0);
end;
```

8. Запустить проект, пройти регистрацию и ответить на вопросы. Результат будет выведен с помощью информационного окна функции `MessageDlg()`.



Проект «Проверка знаний» хранится в папке \Delphi\InputOutput **CD-ROM**

Практические задания для самостоятельного выполнения **CD-ROM**

2.6. Создать проект «Игра Баше», в котором ввод данных осуществляется с помощью панели ввода, вывод — с помощью панели сообщений. Суть игры состоит в следующем: имеется *N* предметов; два игрока по очереди берут 1, 2 или 3 предмета; проигрывает тот игрок, который забирает последний предмет.

2.6. Кодирование алгоритмических структур на языке Delphi

2.6.1. Алгоритмическая структура «ветвление»

Инструкция `If-Then-Else`. Алгоритмическая структура «ветвоение» кодируется на языке программирования Delphi с помощью инструкции **`If-Then-Else`**. После ключевого слова **`If`** должно быть размещено условие. После ключевого слова **`Then`** между служебными словами **`begin`** и **`end`** — последовательность команд `серия 1`, которая должна выполняться, если условие принимает значение «истина». После ключевого слова **`Else`** между служебными словами **`begin`** и **`end`** размещается последовательность команд `серия 2`, которая должна выполняться, если условие принимает значение «ложь» (рис. 2.10).

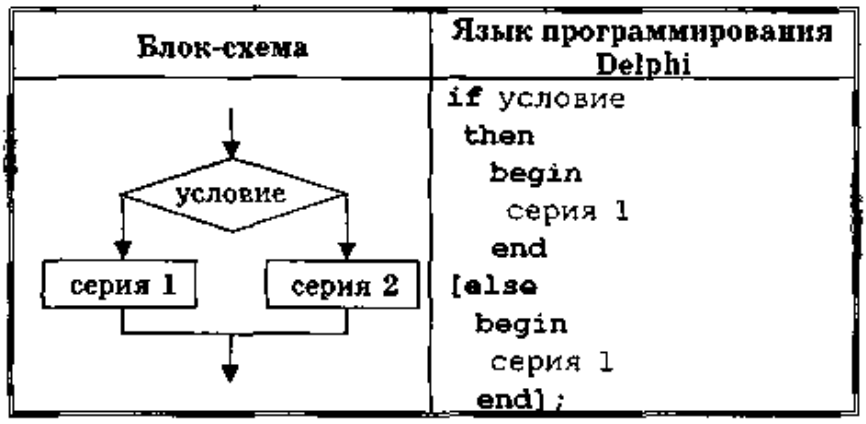


Рис. 2.10. Алгоритмическая структура «ветвление»

Ключевое слово **Else** в сокращенной форме инструкции может отсутствовать. (Необязательные части оператора записываются в квадратных скобках.) Тогда, в случае если условие ложно, выполнение оператора условного перехода заканчивается и выполняется следующая строка программы.

Если между служебными словами **begin** и **end** находится только одна команда, то эти слова можно опустить.

Проект «Проверка знаний»

Практические задания для самостоятельного выполнения CD-ROM

2.7. Создать проект «Поиск большего из двух чисел» на языке программирования Delphi.

2.6.2. Алгоритмическая структура «выбор»

Инструкция case of. Алгоритмическая структура «выбор» кодируется на языке Delphi с помощью инструкции **case of**. Инструкция выбора начинается с ключевого слова **case**, после которого записывается выражение (переменная или арифметическое выражение). Далее записываются списки констант. Если значение выражения, записанного после **case**, совпадает с константой из некоторого списка, то выполняется соответствующая серия команд между служебными словами **begin** и **end**. Если значение выражения не совпадает ни с одной из констант ни одного из списков констант, то выполняется серия команд после ключевого слова **else** (рис. 2.11).

Если между служебными словами **begin** и **end** находится только одна команда, то эти слова опустить.

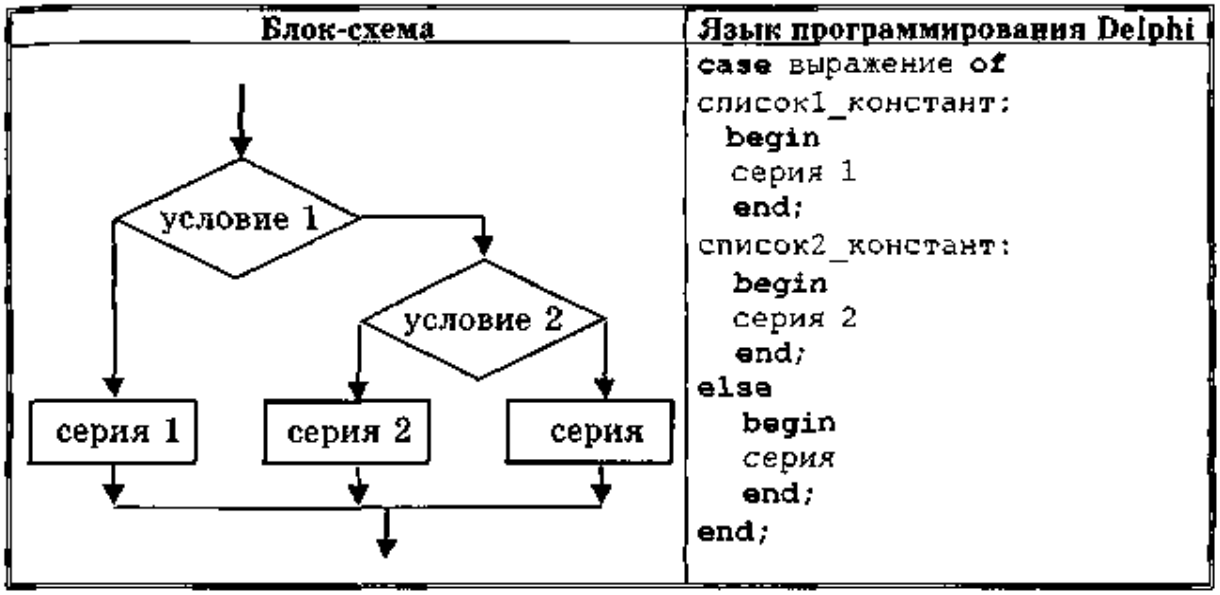


Рис. 2.11. Алгоритмическая структура «выбор»

Проект «Отметка». В качестве примера использования инструкции выбора разработаем проект «Отметка», который позволяет выставлять отметку за работу в зависимости от количества сделанных ошибок.

Проект «Отметка»

1. Создать новый проект. Разместить на форме:
   - текстовое поле EditN для ввода количества ошибок;
   - текстовое поле EditBall для вывода отметки;
   - две метки для вывода поясняющих надписей;
   - кнопку Button1 для запуска событийной процедуры.

2. Ввести код событийной процедуры TForm1.Button1Click(), которая обеспечивает ввод количества ошибок в поле EditN и осуществляет вывод отметки в текстовое поле EditBall:

```
var
N:integer;
procedure TForm1.Button1Click(Sender: Tobject);
begin
N := StrToInt(EditN.Text);
case N of
     0: EditBall.Text := 'Отлично';
     1: EditBall.Text := 'Хорошо';
     2: EditBall.Text := 'Удовлетворительно';
     else EditBall.Text := 'Плохо';
end;
end;
```

3. Запустить проект. Ввести в левое текстовое поле количество ошибок и щелкнуть по кнопке *Отметка*. В правое текстовое поле будет выведена отметка.

Проект «Отметка» хранится в папке \Delphi\Ball\ CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

2.8. Создать проект «Тест с выборочным ответом» на языке программирования Delphi.

2.6.3. Алгоритмическая структура «цикл»

Цикл со счетчиком. Алгоритмическая структура «цикл со счетчиком» кодируется на языке Delphi с помощью инструкции **For-to-Do**. После ключевого слова **For** переменной Счетчик присваивается значение равное НачЗнач. При каждом выполнении тела цикла, находящегося между служебными словами **begin** и **end**, переменная Счетчик увеличивается или уменьшается на единицу. Если она достигает величины КонЗнач, то выполнение цикла завершается и выполняются следующие за ним операторы (рис. 2.12).

Если между служебными словами **begin** и **end** находится только одна команда, то эти слова можно опустить.

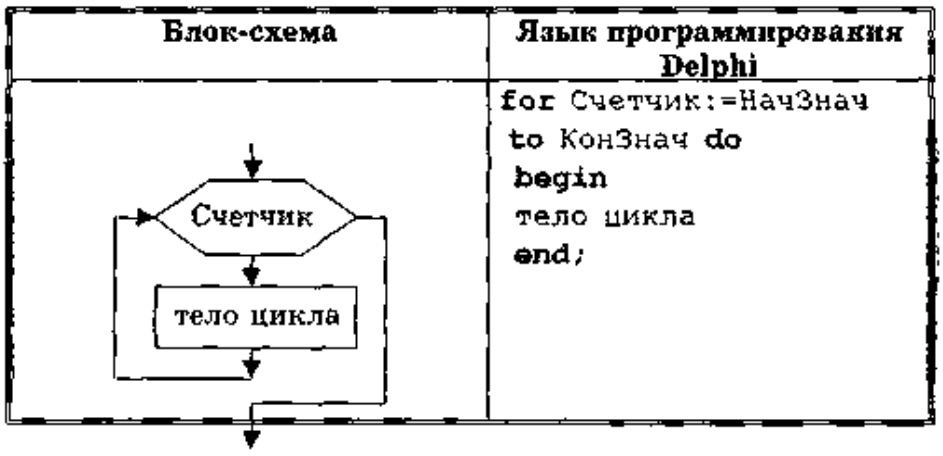


Рис. 2.12. Цикл со счетчиком

Проект «Коды символов». В качестве примера использования цикла со счетчиком создадим проект «Коды символов», который выводит символы по числовым кодам (кодировочную таблицу символов).

Проект «Коды символов»

1. Создать новый проект. Разместить на форме метку Label1 и кнопку Button1.

Воспользуемся циклом со счетчиком с шагом −1, для того чтобы выводить на метку символы, начиная с наибольшего числового кода 255.

2. Ввести код событийной процедуры TForm1.Button1Click(), которая осуществляет вывод кодировочной таблицы символов на метку:

```
var
N:integer;
procedure TForm1.Button1Click(Sender: TObject);
begin
for N := 255 downto 33 Do
begin
Label1.Caption := Label1.Caption
+ IntToStr(N) + '-'+ Chr(N)+ ' ';
end;
end;
```

3. Запустить проект.

Для получения кодировочной таблицы символов щелкнуть по кнопке *Коды символов*.

Проект «Коды символов» хранится в папке \Delphi\FOR-NEXT\ CD-ROM

Цикл с предусловием. В языке программирования Delphi цикл с предусловием реализуется с помощью инструкции **while-do**. Сначала проверяется условие, если оно истинно, то выполняется тело цикла, находящееся между служебными словами **begin** и **end**, и так до тех пор, пока условие не станет ложным (рис. 2.13).

Если между служебными словами **begin** и **end** находится только одна команда, то эти слова можно опустить.

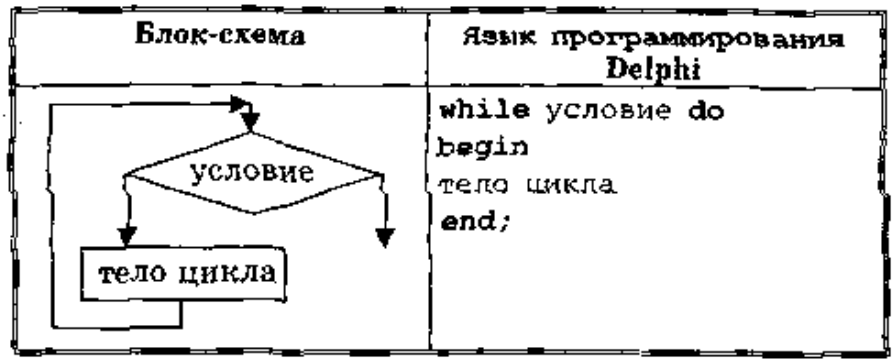


Рис. 2.13. Цикл с предусловием

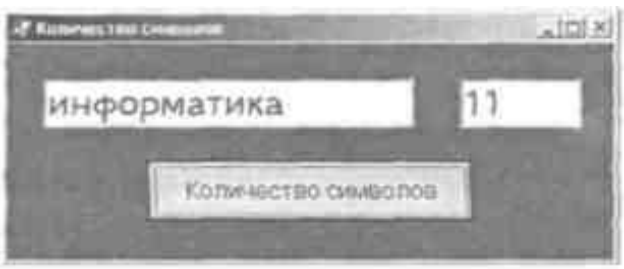
Проект «Количество символов». В качестве примера использования цикла с предусловием разработаем проект, позволяющий подсчитывать количество символов в заданном тексте.

Проект «Количество символов»

1. Создать новый проект. Разместить на форме:
   - текстовое поле EditText для ввода текста;
   - текстовое поле EditNum для вывода количества символов в тексте;
   - кнопку Button1 для запуска событийной процедуры.
2. Ввести код событийной процедуры TForm1.Button1Click(), которая осуществляет в цикле с предусловием подсчет количества символов:

```
var
N:integer;
procedure TForm1.Button1Click(Sender: TObject);
begin
N := 0;
while N < Length(EditText.Text) Do
begin
N := N + 1;
end;
EditNum.Text := IntToStr(N);
end;
```

3. Запустить проект. Ввести текст в левое поле и щелкнуть по кнопке *Количество символов*.
   В правом поле появится число, равное количеству символов.



Проект «Количество символов» хранится в папке \Delphi\While\ CD-ROM

Цикл с постусловием. В языке программирования Delphi цикл с постусловием реализуется с помощью инструкции **repeat-until**. Сначала выполняется тело цикла, затем проверяется условие, если оно ложно, то тело цикла повторяется до тех пор, пока условие не станет истинным (рис. 2.14).

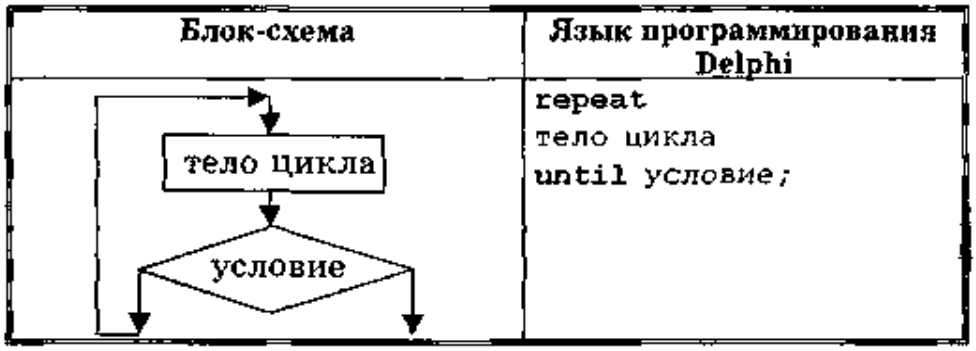


Рис. 2.14. Цикл с постусловием

Практические задания для самостоятельного выполнения CD-ROM

2.9. Создать проект «Факториал», реализующий вычисление факториала числа на языке программирования Delphi.

2.10. Создать проект «Слово-перевертыш», который изменяет прямую последовательность символов в слове (слева направо) на обратную последовательность (справа налево), на языке программирования Delphi.

2.6.4. Общие процедуры

В объектно-ориентированном языке программирования Delphi вспомогательные алгоритмы реализуются с помощью общих процедур. Общие процедуры создаются в тех случаях, когда в програмном модуле можно выделить многократно встречающиеся последовательности действий (алгоритмы).

Запуск общих процедур не связывается с какими-либо событиями, а реализуется путем вызова из других процедур.

Каждой общей процедуре дается уникальное название — **имя процедуры** и устанавливается **список входных и выходных параметров** процедуры.

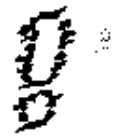
Список входных параметров представляет собой набор переменных, значения которых должно быть установлено до начала выполнения процедуры.

Список выходных параметров представляет собой набор переменных, значения которых должно быть установлено после окончания выполнения процедуры.

Синтаксис общей процедуры:

```
procedure ИмяПроцедуры(СписокПараметров)
const     //начало раздела объявления констант
var       //начало раздела объявления переменных
begin
программный код
end;
```

Общая процедура вызывается на выполнение по имени:



```
ИмяПроцедуры(СписокПараметров)
```

Общая процедура может входить в состав программного модуля формы, например в модуль Unit1.pas.

2.7. Графические возможности языка программирования Delphi

Холст. На формах `Form` или в графических окнах `Image` можно рисовать различные графические примитивы с использованием свойства `Canvas` (холст).

Цвет, толщина и стиль линий, которым рисуются графические примитивы на холсте, задаются как значения свойства `Pen` (карандаш).

Цвет и стиль заливки внутри геометрических примитивов задаются как значения свойства `Brush` (кисть).

Точка. Метод `Pixels` — установка точки с заданными координатами и цветом:

```
object.Canvas.Pixels[X,Y] := color
```

Аргументами метода являются `X, Y` — целочисленные координаты точки и `color` — цвет линии.

Значение аргумента `color` можно задать различными способами:

- с помощью одной из нескольких десятков констант, определяющих цвет (clBlack — черный, clBlue — синий, clGreen — зеленый, clRed — красный, clYellow — желтый, clWhite — белый и т. д.);
- с помощью цветовой модели RGB (красный, зеленый, синий) в шестнадцатеричном представлении, в котором для задания интенсивностей базовых цветов используются по два шестнадцатеричных разряда (например, $00FF0000 — синий, $0000FF00 — зеленый, $000000FF — красный; $00000000 — черный и $00FFFFFF — белый).

В случае отсутствия аргумента color рисование будет производиться цветом, принятым по умолчанию (черным).

Линия. Метод LineTo — рисование линии:

```
object.Canvas.LineTo(X1,Y1)
```

Метод LineTo рисует прямую линию из точки с текущими координатами в точку с координатами X1,Y1.

Переход в точку с требуемыми координатами реализуется с помощью метода MoveTo:

```
object.Canvas.MoveTo(X0,Y0)
```

Прямоугольник. Метод Rectangle — рисование прямоугольника:

```
object.Canvas.Rectangle(X1,Y1,X2,Y2)
```

Метод Rectangle рисует прямоугольник с координатами X1, Y1 левого верхнего угла прямоугольника и координатами X2, Y2 правого нижнего угла.

Дуга. Метод Ellipse — рисование окружностей и эллипсов:

```
object.Canvas.Ellipse(X1,Y1,X2,Y2)
```

Здесь X1,Y1,X2,Y2 — соответственно координаты левого верхнего и правого нижнего углов прямоугольника, в который вписана окружность (эллипс).

Вывод текста на холст. Для вывода текста на холст используется метод TextOut:

```
object.Canvas.TextOut(X,Y,Text)
```

Здесь: X, Y — координаты точки холста, начиная с которой выполняется вывод текста; Text — строковая переменная или строка, которая выводится на холст.

Проект «Построение графика функции». Разработаем проект построения в графическом окне графика функции с использованием графических методов. В качестве примера рассмотрим построение графика функции $y = \sin x$.

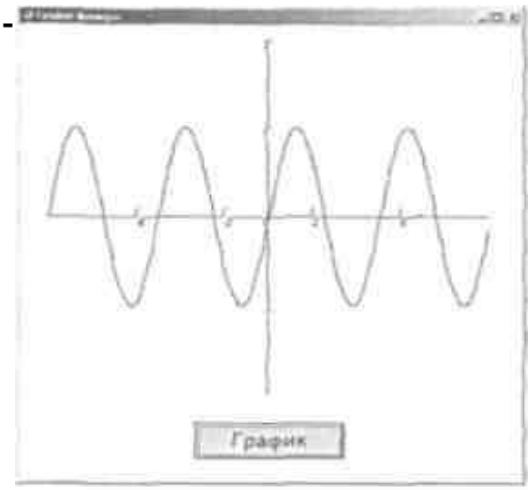
Проект «Построение графика функции»

1. Разместить на форме графическое поле Image1, в котором будет производиться построение графика. Для большей понятности программного кода будем вводить в него комментарии, которые начинаются с символов «//».
2. Разместить на форме кнопку Button1 и создать событийную процедуру построения графика, которая в цикле осуществляет построение графика функции, рисует оси координат и выводит на них числовые шкалы.

```
var
X:real;
Y:real;
N:integer;
procedure TForm1.Button1Click(Sender: Tobject);
begin
with Image1.Canvas do
begin
      X:=0;
      while X<500 Do
      begin
      X:=X+0.1;
      Y := 200-100*(Sin(X/20));
      Pixels[Round(X),Round(Y)]:=clBlack;
      end;
      MoveTo(0,200); LineTo(500,200);   //Ось X
      MoveTo(250,0); LineTo(250,500);   //Ось Y
                           //шкала оси X
      N:=0;
      while N<500 do
      begin
      N:=N+100;
      MoveTo(N,190); LineTo(N,210);
      TextOut(N,200,IntToStr(Round((N-250)/20)));
      end;
                           //шкала оси Y
      N:=0;
      while N<400 do
      begin
      N:=N+100;
```

```
        MoveTo(245,400-N); LineTo(255,400-N);
        TextOut(245,400-N,
                IntToStr(Round((N-200)/100)));
        end;
    end;
    end;
```

3. Запустить проект и щелкнуть по кнопке *График*.



Проект «График функции» хранится в папке \Delphi\Graph CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

2.11. Создать проект «График функции-2» построения графиков функций x^2 и x^3 и вертикальных линий координатной сетки.

2.12. Создать проект «Графический редактор», который позволяет нарисовать в графическом поле графические примитивы (точку, линию, прямоугольник, окружность).

2.8. Массивы в языке программирования Delphi

2.8.1. Числовые массивы: заполнение и поиск

Объявление массива. Объявление массива производится аналогично объявлению переменных, необходимо только дополнительно указать диапазон изменения индекса. Например, объявление одномерного целочисленного массива, содержащего 100 элементов, производится следующим образом:

```
A:array[1..100]:of integer;
```

Заполнение массива случайными числами. Для начала работы с массивом необходимо его предварительно заполнить, т. е. присвоить элементам массива определенные значения. Заполним числовой массив $A[I]$ целыми случайными числами в интервале от 1 до 100.

Для генерации последовательности случайных чисел используем функцию `Random(100)`. При запуске программы функция `Random(100)` дает псевдослучайную последовательность целых чисел в интервале от 0 до 100.

Для генерации различающихся между собой последовательностей случайных чисел рекомендуется использовать оператор `Randomize`.

Проект «Поиск в числовом массиве»

1. Поместить на форму `Form1` кнопку `Button1` и создать для нее событийную процедуру `TForm1.Button1Click()`, реализующую заполнение массива случайными числами:

```
var
A: array[1..100] of integer;
I: integer;
Min: integer;
N: integer;
procedure TForm1.Button1Click(Sender: TObject);
begin
Randomize;
For I := 1 To 100 Do
begin
A[I] := Random(100);
end;
end;
```

Поиск в числовом массиве. Осуществим поиск наименьшего элемента в числовом массиве, состоящем из 100 элементов.

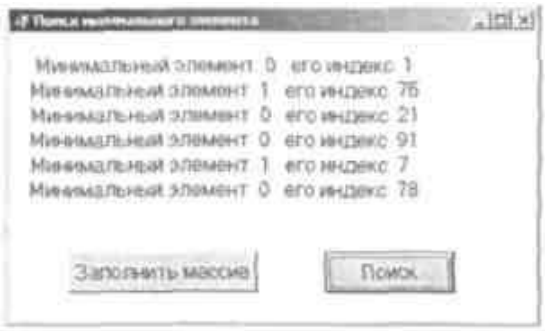
Будем хранить значение минимального элемента в переменной `Min`, а его индекс — в переменной `N`. Будем считать, что сначала минимальный элемент равен первому элементу массива `A[1]`, поэтому присвоим переменной `Min` его значение.

Затем в цикле сравним последовательно значения элементов массива со значением переменной `Min`, если какой-либо элемент окажется меньше, присвоим его значение переменной `Min`, а его индекс присвоим переменной `N`. Результат поиска выведем на метку.

2. Поместить на форму кнопку Button2 и создать для нее событийную процедуру TForm1.Button2Click(), реализующую поиск:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Min := A[1];
  N := 1;
  For I := 2 To 100 Do
   begin
     If A[I] < Min Then
        begin
           Min := A[I];
           N := I;
        end;
    end;
  Label1.Caption := Label1.Caption +
  ' Минимальный элемент ' + IntToStr(Min) +
  ' его индекс ' + IntToStr(N);
end;
```

3. Запустить проект. Щелкнуть по кнопкам *Заполнить массив* и *Поиск* несколько раз.
На метку будут выведены результаты поиска минимального элемента для различных вариантов заполнения массива.



Проект «Поиск минимального элемента в числовом массиве» хранится в папке \Delphi\Arrays — CD-ROM

Практические задания для самостоятельного выполнения — CD-ROM

2.13. Создать проект «Поиск максимального элемента в числовом массиве», предусматривающий заполнение массива случайными числами и возможность вывода найденного максимального элемента и его индекса на метку.

2.8.2. Сортировка числового массива

Пусть у нас имеется целочисленный числовой массив A[I], состоящий из десяти элементов и заполненный случайными числами. Отсортируем этот массив по возрастанию.

Повторение поиска номера минимального элемента, перестановки элементов массива и вывода массива на метку реализуем с помощью цикла со счетчиком I, максимальное значение которого равно 9.

Поиск номера минимального элемента реализуем с помощью вложенного цикла со счетчиком J.

Вывод массива на метку на каждом шаге сортировки реализуем с помощью вложенного цикла со счетчиком K.

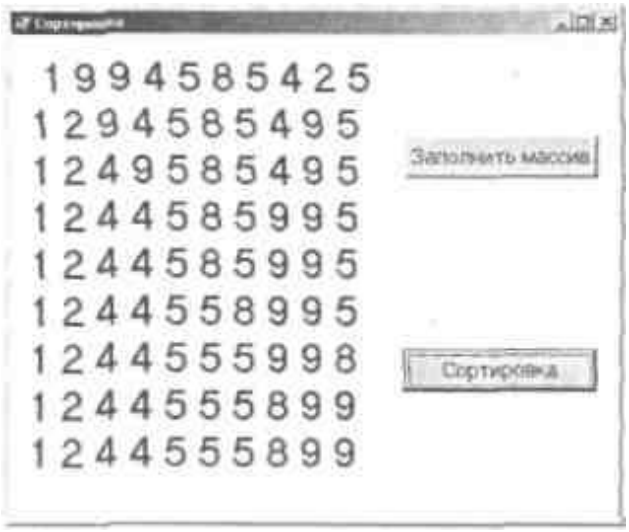
Проект «Сортировка числового массива»

1. Поместить на форму Form1 кнопку Button1 и создать для нее событийную процедуру TForm1.Button1Click(), реализующую заполнение массива случайными числами.
2. Поместить на форму Form1 кнопку Button2 и создать для нее событийную процедуру TForm1.Button2Click(), реализующую сортировку массива.

```
var
A: array[1..10] of integer;
I: integer;    // счетчик цикла
J: integer;    // счетчик цикла
K: integer;    // счетчик цикла
R: integer;    // буфер для обмена значений
               // элементов массива
Min:integer;   // номер минимального элемента
procedure TForm1.Button2Click(Sender: TObject);
begin
For I := 1 To 9 Do
begin
    Min := I;
    'поиск номера минимального элемента
    For J := I + 1 To 10 Do
       If A[J] < A[Min]
        Then Min := J;
    'Перестановка
    R := A[I];
    A[I] := A[Min];
    A[Min] := R;
    'Вывод массива
    'для каждого цикла перестановки
     For K := 1 To 10 Do
     begin
     Label1.Caption := Label1.Caption +
                       ' ' + IntToStr(A[K]);
```

```
        end;
    end;
    end;
```

3. Запустить проект. Щелкнуть по кнопкам *Заполнить массив* и *Сортировка*.
   На метку будет выведен процесс сортировки числового массива по шагам.



Проект «Сортировка числового массива по возрастанию» хранится в папке \Delphi\ArraySort — CD-ROM

Практические задания для самостоятельного выполнения — CD-ROM

2.14. Создать проект «Сортировка числового массива по убыванию», в котором реализуется сортировка с использованием поиска максимального элемента во вложенном цикле.

Глава 3

Построение и исследование информационных моделей

3.1. Моделирование как метод познания

3.1.1. Системный подход в моделировании

Понятие о системе. Окружающий нас мир состоит из множества различных объектов, каждый из которых имеет разнообразные свойства, и при этом объекты взаимодействуют между собой. Например, такие объекты, как планеты нашей Солнечной системы, имеют различные свойства (массу, геометрические размеры и т. д.) и по закону всемирного тяготения взаимодействуют с Солнцем и друг с другом.

Планеты входят в состав более крупного объекта — Солнечной системы, а Солнечная система — в состав нашей галактики Млечный путь. С другой стороны, планеты состоят из атомов различных химических элементов, а атомы — из элементарных частиц. Можно сделать вывод, что практически каждый объект состоит из других объектов, т. е. представляет собой систему.

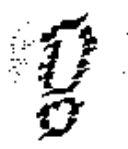
Система состоит из объектов, которые называются **элементами системы**.

Важным признаком системы является ее **целостное функционирование**. Система является не набором отдельных элементов, а совокупностью взаимосвязанных элементов.

Например, компьютер является системой, состоящей из различных устройств, при этом устройства связаны между собой и аппаратно (физически подключены друг к другу) и функционально (между устройствами происходит обмен информацией).

Состояние системы характеризуется ее **структурой**, т. е. составом элементов, их отношениями и связями между собой. Система сохраняет свою целостность под воздействием различных внешних воздействий и внутренних изменений, пока она сохраняет неизменной свою структуру. Если струк-

тура системы меняется (например, удаляется один из элементов), то система может перестать функционировать как целое. Так, если удалить одно из устройств компьютера (например, процессор), компьютер выйдет из строя, т. е. прекратит свое существование как система.



Система — это совокупность взаимосвязанных элементов, структура которой сохраняет свою целостность под действием внешних воздействий и внутренних изменений.

Статические информационные модели. Любая система существует в пространстве и времени. В каждый момент времени система находится в определенном состоянии, которое характеризуется составом элементов, значениями их свойств, величиной и характером взаимодействия между элементами и т. д.

Так, состояние Солнечной системы в любой момент времени характеризуется составом входящих в нее объектов (Солнце, планеты и др.), их свойствами (размерами, положением в пространстве и т. д.), величиной и характером взаимодействия между собой (силами тяготения, с помощью электромагнитных волн и т. д.).

Модели, описывающие состояние системы в определенный момент времени, называются статическими информационными моделями.

В физике примером статических информационных моделей являются модели, описывающие простые механизмы, в биологии — модели строения растений и животных, в химии — модели строения молекул и кристаллических решеток и т. д.

Динамические информационные модели. Состояние систем изменяется во времени, т. е. происходят процессы **изменения и развития систем.** Так, планеты движутся, изменяется их положение относительно Солнца и друг друга, Солнце, как и любая другая звезда, развивается, меняются ее химический состав, излучение и т. д.

Модели, описывающие процессы изменения и развития систем, называются динамическими информационными моделями.

В физике динамические информационные модели описывают движение тел, в биологии — развитие организмов или популяций животных, в химии — процессы прохождения химических реакций и т. д.

Вопросы для размышления

1. Образуют ли систему комплектующие компьютера до сборки? После сборки? После включения компьютера?
2. В чем разница между статическими и динамическими информационными моделями? Приведите примеры статических и динамических информационных моделей.

3.1.2. Материальные модели и информационные модели

Все модели можно разбить на два больших класса: **предметные (материальные) модели** и **информационные модели**. Предметные модели воспроизводят геометрические, физические и другие свойства объектов в материальной форме (глобус, анатомические муляжи, модели кристаллических решеток, макеты зданий и сооружений и др.).

Информационные модели представляют объекты и процессы в **образной** или в **знаковой** форме.

Образные модели (рисунки, фотографии и др.) представляют собой зрительные образы объектов, зафиксированные на каком-либо носителе информации (бумаге, фото- и кинопленке и др.). Широко используются образные информационные модели в образовании (вспомните учебные плакаты по различным предметам) и науках, где требуется классификация объектов по их внешним признакам (ботанике, биологии, палеонтологии и др.).

Знаковые информационные модели строятся с использованием различных языков (знаковых систем). Знаковая информационная модель может быть представлена в форме текста (например, описания на русском языке, программы на языке программирования), формулы (например, второго закона Ньютона $\vec{F} = m\vec{a}$), таблицы (например, периодической таблицы элементов Д. И. Менделеева) и т. д.

Иногда при построении знаковых информационных моделей используются одновременно несколько различных языков. Примерами таких моделей могут служить географические карты, графики, диаграммы и др. Во всех этих моделях используются одновременно как язык графических элементов, так и символьный язык.

На протяжении своей истории человечество использовало различные способы и инструменты для создания информационных моделей. Эти способы постоянно совершенствовались, так, первые информационные модели создавались в форме наскальных рисунков. В настоящее время информационные модели обычно строятся и исследуются с использованием современных компьютерных технологий.

Формализация. Естественные языки используются для создания **описательных информационных моделей.** В истории науки известны многочисленные описательные информационные модели, например, гелиоцентрическая модель мира, которую предложил Коперник, формулировалась следующим образом:

- Земля вращается вокруг своей оси и вокруг Солнца;
- орбиты всех планет проходят вокруг Солнца.

С помощью формальных языков строятся **формальные информационные модели** (математические, логические и др.). Одним из наиболее широко используемых формальных языков является математика. Модели, построенные с использованием математических понятий и формул, называются **математическими моделями.** Язык математики является совокупностью формальных языков, с некоторыми из них (алгебра, геометрия, тригонометрия) вы знакомитесь в школе, с другими (теория множеств, теория вероятностей и др.) сможете ознакомиться в процессе дальнейшего обучения.

Язык алгебры позволяет формализовать функциональные зависимости между величинами. Так, Ньютон формализовал гелиоцентрическую систему мира, открыв законы механики и закон всемирного тяготения и записав их в виде алгебраических функциональных зависимостей. В школьном курсе физики рассматривается много разнообразных функциональных зависимостей, выраженных на языке алгебры, которые представляют собой математические модели изучаемых явлений или процессов.

Язык алгебры логики (алгебры высказываний) позволяет строить **формальные логические модели.** С помощью алгебры высказываний можно формализовать (записать в виде логических выражений) простые и сложные высказывания, выраженные на естественном языке. Построение логических моделей позволяет решать логические задачи, строить логические модели устройств компьютера (сумматора, триггера) и т. д.

Процесс построения информационных моделей с помощью формальных языков называется **формализацией.**

В процессе познания окружающего мира человечество постоянно использует моделирование и формализацию. При изучении нового объекта сначала обычно строится его описательная информационная модель на естественном языке, затем она формализуется, т. е. выражается с использованием формальных языков (математики, логики и др.).

Визуализация формальных моделей. В процессе исследования формальных моделей часто производится их визуализация. Для визуализации алгоритмов используются блок-схемы, пространственных соотношений между объектами — чертежи, моделей электрических цепей — электрические схемы, логических моделей устройств — логические схемы и т. д.

При визуализации формальных физических моделей на компьютере с помощью анимации может отображаться динамика процесса, производиться построение графиков изменения физических величин и т. д. Визуальные компьютерные модели обычно являются интерактивными, т. е. исследователь может менять начальные условия и параметры протекания процессов и наблюдать изменения в поведении модели.

http://www.college.ru/physics/applets/11a.htm Интернет

В качестве примера можно рассмотреть модель, которая демонстрирует свободные колебания математического маятника (рис. 3.1). С помощью анимации показывается движение тела и действующие силы, строятся графики зависимости от времени угловой координаты или скорости, диаграммы потенциальной и кинетической энергий. Исследователь может изменять длину нити l, угол начального отклонения маятника φ_0 и коэффициент вязкого трения b.

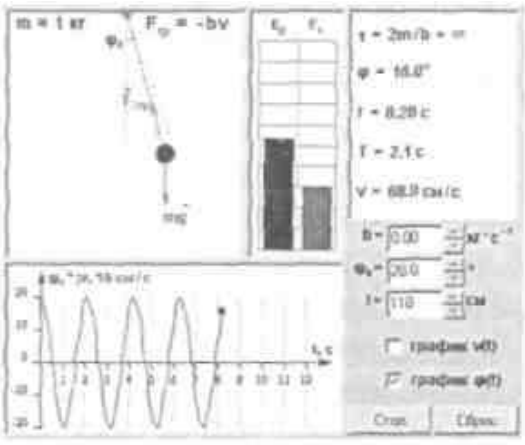


Рис. 3.1. Модель математического маятника

Вопросы для размышления

1. Какие бывают модели? Приведите примеры материальных и информационных моделей.
2. Что такое формализация? Приведите примеры формальных моделей.

Практические задания для самостоятельного выполнения

Интернет

3.1. В Интернете по адресу http://www.college.ru ознакомиться с визуализированными формальными моделями из различных предметных областей.

3.1.3. Основные этапы разработки и исследования моделей на компьютере

Использование компьютера для исследования информационных моделей различных объектов и систем позволяет изучить их изменения в зависимости от значения тех или иных параметров. Процесс разработки моделей и их исследование на компьютере можно разделить на несколько основных этапов.

Описательная информационная модель. На первом этапе исследования объекта или процесса обычно строится описательная информационная модель. Такая модель выделяет существенные с точки зрения целей проводимого исследования параметры объекта, а несущественными параметрами пренебрегает.

Формализованная модель. На втором этапе создается формализованная модель, т. е. описательная информационная модель записывается с помощью какого-либо формального языка. В такой модели с помощью формул, уравнений, неравенств и т. д. фиксируются формальные соотношения между начальными и конечными значениями свойств объектов, а также накладываются ограничения на допустимые значения этих свойств.

Однако далеко не всегда удается найти формулы, явно выражающие искомые величины через исходные данные. В таких случаях используются приближенные математические методы, позволяющие получать результаты с заданной точностью.

Компьютерная модель. На третьем этапе необходимо формализованную информационную модель преобразовать в компьютерную модель, т. е. выразить ее на понятном для компьютера языке. Существуют два принципиально различных пути построения компьютерной модели:

- создание проекта на одном из языков программирования;
- построение компьютерной модели с использованием некоторого приложения, например электронных таблиц.

В процессе создания компьютерной модели полезно разработать удобный графический интерфейс, который позволит визуализировать формальную модель, а также реализовать интерактивный диалог человека с компьютером на этапе исследования модели.

Компьютерный эксперимент. Четвертый этап исследования информационной модели состоит в проведении компьютерного эксперимента. Если компьютерная модель существует в виде программы на одном из языков программирования, ее нужно запустить на выполнение и получить результаты.

Если компьютерная модель исследуется в приложении, например в электронных таблицах, можно провести сортировку или поиск данных, построить диаграмму или график и т. д.

Анализ полученных результатов и корректировка исследуемой модели. Пятый этап состоит в анализе полученных результатов и корректировке исследуемой модели. В случае различия результатов, полученных при исследовании информационной модели, с измеряемыми параметрами реальных объектов, можно сделать вывод, что на предыдущих этапах построения модели были допущены ошибки или неточности.

Например, при построении описательной качественной модели могут быть неправильно отобраны существенные свойства объектов, в процессе формализации могут быть допущены ошибки в формулах и т. д. В этих случаях необходимо провести корректировку модели, причем уточнение модели может проводиться многократно, пока анализ результатов не покажет их соответствие изучаемому объекту.

Вопросы для размышления

1. В каких случаях могут быть опущены отдельные этапы построения и исследования модели? Приведите известные вам примеры создания моделей в процессе изучения физики, химии, биологии, математики, географии и других предметов.

3.2. Исследование физических моделей

3.2.1. Построение информационной модели движения тела, брошенного под углом к горизонту

Рассмотрим процесс построения и исследования модели на конкретном примере движения тела, брошенного под углом к горизонту.

Физика-9

Содержательная постановка задачи «Попадание в стенку тела, брошенного под углом к горизонту». В процессе тренировок теннисистов используются автоматы по бросанию мячика в мишень. Необходимо задать автомату необходимую скорость и угол бросания мячика для попадания в мишень определенной высоты, находящуюся на известном расстоянии.

Качественная описательная модель. Сначала построим качественную описательную модель процесса движения тела с использованием физических объектов, понятий и законов, т. е. в данном случае идеализированную модель движения объекта. Из условия задачи можно сформулировать следующие основные предположения:

- мячик мал по сравнению с Землей, поэтому его можно считать материальной точкой;
- изменение высоты мячика мало, поэтому ускорение свободного падения можно считать постоянной величиной $g = 9{,}8$ м/с^2, и движение по оси Y можно считать равноускоренным;
- скорость бросания тела мала, поэтому сопротивлением воздуха можно пренебречь, и движение по оси X можно считать равномерным.

Формальная модель. Для формализации модели используем известные из курса физики формулы равномерного и равноускоренного движения. При заданных начальной скорости v_0 и угле бросания α значения координат дальности полета x и высоты y от времени можно описать следующими формулами:

$$x = v_0 \cdot \cos\alpha \cdot t;$$
$$y = v_0 \cdot \sin\alpha \cdot t - g \cdot t^2/2. \qquad (3.1)$$

Пусть мишень высотой h будет размещаться на расстоянии s. Из первой формулы выражаем время, которое понадобится мячику, чтобы преодолеть расстояние s:

$$t = s/v_0 \cdot \cos\alpha.$$

Подставляем это значение для t в формулу для y. Получаем l — высоту мячика над землей на расстоянии s:

$$l = s \cdot tg\alpha - g \cdot s^2/2 \cdot v_0^2 \cdot \cos 2\alpha. \qquad (3.2)$$

Формализуем теперь условие попадания мячика в мишень. Попадание произойдет, если значение высоты l мячика будет удовлетворять условию в форме неравенства:

$$0 \le l \le h.$$

Если $l < 0$, то это означает «недолет», а если $l > h$, то это означает «перелет».

Практические задания для самостоятельного выполнения CD-ROM

3.2. Построить формальную модель решения задачи «Попадание в площадку тела, брошенного под углом к горизонту». В процессе тренировок теннисистов используются автоматы по бросанию мячика в определенное место площадки. Необходимо задать автомату необходимую скорость и угол бросания мячика для попадания в площадку определенной длины, находящуюся на известном расстоянии.

3.2.2. Компьютерная модель движения тела на языке Visual Basic

На основе формальной модели, описывающей движение тела, брошенного под углом к горизонту, создадим компьютерную модель с использованием системы программирования Visual Basic.

Проект «Попадание в стенку тела, брошенного под углом к горизонту»

Создадим сначала графический интерфейс проекта.

1. Поместить на форму:
   - четыре текстовых поля (объекты TextBox) для ввода значений начальной скорости, угла бросания мячика, расстояния до мишени и ее высоты;
   - две метки (объекты Label) для вывода высоты мячика на заданном расстоянии и текстового сообщения о результатах броска.
2. Поместить на форму десять меток (объекты Label) для обозначения назначения текстовых полей (имен переменных и единиц измерения).

Создадим программный код событийной процедуры, определяющей попадание мячика в мишень.

3. Поместить на форму кнопку cmdCalc и создать для нее событийную процедуру cmdCalc_Click(), в которой:
   - объявить вещественные константы одинарной точности G и Pi;
   - объявить вещественные переменные двойной точности V0, A, S, H, L и T;
   - объявить целую переменную I (счетчик цикла);
   - присвоить переменным V0, A, S, H значения, введенные в текстовые поля, с использованием функции преобразования строки в вещественное число Val();
   - вычислить высоту мячика L на заданном расстоянии;
   - вывести высоту мячика L на метку lblL;
   - вывести текстовое сообщение о результатах броска на метку lblM с использованием инструкции Select Case:

```
Const G As Single = 9.81
Const Pi As Single = 3.14
Dim V0, A, S, L, T As Double, I As Integer
Private Sub cmdCalc_Click()
'Ввод начальных значений
V0 = Val(txtV0.Text)
A  = Val(txtA.Text)
S  = Val(txtS.Text)
H  = Val(txtH.Text)
'Попадание в мишень
L = S * Tan(A * Pi / 180) - (G * S ^ 2) /
(2 * V0 ^ 2 * Cos(A * Pi / 180) ^ 2)
lblL.Caption = L
```

```
Select Case L
Case Is < 0
lblM.Caption = "Недолет"
Case Is > H
lblM.Caption = "Перелет"
Case Else
lblM.Caption = "Попадание"
End Select
End Sub
```

Для визуализации формальной модели построим траекторию движения тела (график зависимости высоты мячика над поверхностью земли от дальности полета). Снабдим график осями координат и выведем положение мишени.

4. Поместить на форму графическое поле pic1, в котором будет осуществляться построение графика.

   В событийную процедуру ввести код установки масштаба графического поля:

```
'Установка масштаба
pic1.Scale (0, 15)-(S + 5, -5)
```

   В событийную процедуру ввести код построения траектории движения мячика:

```
'Построение траектории движения мячика
For T = 0 To 10 Step 0.1
Y = V0 * Sin(A * Pi/180) * T - G * T * T/2
X = V0 * Cos(A * Pi/180) * T
pic1.PSet (X, Y)
Next T
```

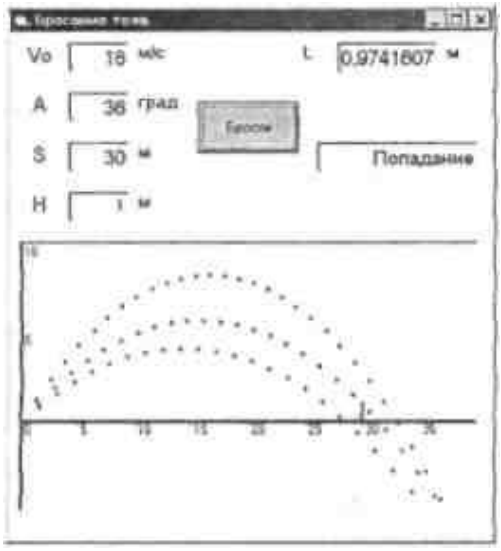
   В событийную процедуру ввести код построения осей *X* и *Y* со шкалами и рисования мишени:

```
'Ось X
pic1.Line (0, 0)-(50, 0)
For I = 0 To 50 Step 5
pic1.PSet (I, 0)
pic1.Print I
Next I
'Ось Y
pic1.Line (0, -5)-(0, 15)
For I = -5 To 20 Step 5
pic1.PSet (0, I)
pic1.Print I
Next I
'Мишень
pic1.Line (S, 0)-(S, H)
```

Проект «Попадание в стенку тела, брошенного под углом к горизонту» хранится в папке \VB\Phys1 CD-ROM

Компьютерный эксперимент. Введем произвольные значения начальной скорости и угла бросания мячика; скорее всего, попадания в мишень не будет. Меняя один из параметров, например, угол, произведем «пристрелку», используя известный артиллерийский прием «взятие в вилку», в котором применяется эффективный метод «деление пополам». Сначала найдем угол, при котором мячик перелетит мишень, затем угол, при котором мячик не долетит до стены. Вычислим среднее значение углов, составляющих «вилку», и проверим, попадет ли мячик в мишень. Если он попадет в мишень, то задача выполнена, если не попадет, то рассматривается новая «вилка» и т. д.

5. Запустить проект и ввести значения начальной скорости, угла, расстояния до мишени и ее высоты.
   Щелкнуть по кнопке *Бросок*.
   На метки будут выведены результаты, а в графическом поле появится траектория движения тела.



6. Подобрать значения начальной скорости и угла бросания мячика, обеспечивающие его попадание в мишень.
   Например, при скорости бросания мячика $v_0 = 18$ м/с и угле бросания $\alpha = 36°$ мячик попадет в мишень высотой $h = 1$ м, находящуюся на расстоянии $S = 30$ м на высоте $l = 0{,}9741607$ м.

Анализ результатов и корректировка модели. Модернизируем проект так, чтобы можно было для каждого значения скорости бросания мячика получить с заданной точностью диапазон значений углов, обеспечивающих попадание мячика в мишень.

Проект «Диапазон углов, обеспечивающий попадание в стенку»

1. Удалить с формы текстовое поле `txtA` для ввода значения угла, метку `lblL` для вывода результатов бросания и графическое поле `pic1`.

2. Поместить на форму текстовое поле txtP для ввода точности определения диапазона углов и метку lblA для вывода значений диапазона углов.
3. Внести изменения в объявление переменных и программный код событийной процедуры:

```
Const G As Single = 9.81
Const Pi As Single = 3.14
Dim V0, S, H, L As Double, A, P As Integer
Private Sub cmdCalc_Click()
'Ввод начальных значений
V0 = Val(txtV0.Text)
S  = Val(txtS.Text)
H  = Val(txtH.Text)
P  = Val(txtP.Text)
For A = 0 To 90 Step P
'Попадание в мишень
L = S * Tan(A * Pi / 180) - (G * S ^ 2) /
(2 * V0 ^ 2 * Cos(A * Pi / 180) ^ 2)
'Вывод значений диапазона углов
If 0 < L And L < H Then
lblA.Caption = lblA.Caption + Str(A)
End If
Next A
End Sub
```

4. Запустить проект и ввести скорость бросания мячика, расстояние до мишени и ее высоту, а также точность определения значений углов.

Щелкнуть по кнопке *Диапазон углов*.

Получен не очевидный результат, оказывается существуют два диапазона углов от 33 до 36° и от 56 до 57°, которые обеспечивают попадание мячика при скорости бросания $v_0 = 18$ м/с в мишень высотой $h = 1$ м, находящуюся на расстоянии $S = 30$ м.

Проект «Диапазон углов, обеспечивающий попадание в стенку» хранится в папке \VB\Phys2 **CD-ROM**

Вопросы для размышления

1. Как можно реализовать получение результатов с заданной точностью в языке программирования Visual Basic?
2. Имеет ли физический смысл вычисление значения высоты попадания мячика в мишень с точностью до 7 знаков после запятой? До какой точности целесообразно округлить полученное значение?

Практические задания для самостоятельного выполнения

CD-ROM

3.3. На основе формальной модели «Попадание в площадку тела, брошенного под углом к горизонту» (см. задание 3.2), построить компьютерную модель на языке программирования Visual Basic.

3.4. На языке программирования Visual Basic создать проект «Диапазон углов, обеспечивающий попадание в площадку», который позволяет определить для любой скорости бросания диапазон углов, обеспечивающих попадание в площадку.

3.2.3. Компьютерная модель движения тела на языке Delphi

На основе формальной модели, описывающей движение тела, брошенного под углом к горизонту, создадим компьютерную модель с использованием системы программирования Delphi.

Проект «Попадание в стенку тела, брошенного под углом к горизонту»

Создадим сначала графический интерфейс проекта.

1. Поместить на форму:
   - четыре текстовых поля (объекты `Edit`) для ввода значений начальной скорости, угла бросания мячика, расстояния до мишени и ее высоты;
   - две метки (объекты `Label`) для вывода высоты мячика на заданном расстоянии и текстового сообщения о результатах броска.
2. Поместить на форму десять меток (объекты `Label`) для обозначения назначения текстовых полей (имен переменных и единиц измерения).

Создадим программный код событийной процедуры, определяющей попадание мячика в мишень.

3. Поместить на форму кнопку Button1 и создать для нее событийную процедуру TForm1.Button1Click, в которой:
   - объявить константы G и Pi;
   - объявить вещественные переменные V0, A, S, H и L;
   - присвоить переменным V0, A, S, H значения, введенные в текстовые поля, с использованием функции преобразования строки в вещественное число StrToFloat();
   - вычислить высоту мячика L на заданном расстоянии;
   - вывести высоту мячика L на метку Label1 с использованием функции преобразования типа данных FloatToStr(L);
   - вывести текстовое сообщение о результатах броска на метку Label2 с использованием инструкции **if-then-else**:

```
procedure TForm1.Button1Click(Sender: Tobject);
const    //начало раздела объявления констант
G  = 9.81;
Pi = 3.14;
var      //начало раздела объявления переменных
V0: real; //начальная скорость
A : real; //угол бросания
S : real; //расстояние до мишели
H : real; //высота мишени
L : real;//высота мячика на заданном расстоянии
begin
              //Ввод начальных значений
V0:= StrToFloat(EditV0.Text);
A := StrToFloat(EditA.Text);
S := StrToFloat(EditS.Text);
H := StrToFloat(EditH.Text);
              //Попадание в мишень
L:=S*Sin(Pi*A/180)/Cos(Pi*A/180)-G*Sqr(S)
/(2*Sqr(V0)*Sqr(Cos(Pi*A/180)));
Label1.Caption := FloatToStr(L);
if L<0 then
            Label2.Caption := 'Недолет'
          else if L<1 then
            Label2.Caption := 'Попадание'
          else
            Label2.Caption := 'Перелет';
end;
```

Для визуализации формальной модели построим траекторию движения тела (график зависимости высоты мячика над поверхностью земли от дальности полета). Снабдим график осями координат со шкалами и вывести положение мишени.

4. Поместить на форму графическое поле Image1, в котором будет осуществляться построение графика.
   С помощью диалоговой панели *Object Inspector* установить размеры графического поля, например, свойству Height присвоить значение 400, а Width – 500.
   Поместить на форму кнопку Button2.
5. Создать событийной процедуру TForm1.Button2Click, в которой:
   - объявить константы G и Pi;
   - объявить вещественные переменные V0, A, S, H, L и T;
   - объявить целочисленные переменные X, Y и N;
   - присвоить переменным V0, A, S, H значения, введенные в текстовые поля, с использованием функции преобразования строки в вещественное число StrToFloat();
   - построить траекторию движения мячика с помощью объекта Image1.Canvas;
   - построить оси *X* и *Y* со шкалами и мишень.

```
procedure TForm1.Button2Click(Sender: Tobject);
const          //начало раздела объявления кон-
стант
G = 9.81;
Pi = 3.14;
var    //начало раздела объявления переменных
V0: real; //начальная скорость
A : real;  //угол бросания
S : real;  //расстояние до мишели
H : real;  //высота мишени
L : real;//высота мячика на заданном расстоянии
X : integer;   //координата X
Y : integer;   //координата Y
T : real;      //время
N : integer;   //счетчик
begin
                //Ввод начальных значений
V0:= StrToFloat(EditV0.Text);
A := StrToFloat(EditA.Text);
S := StrToFloat(EditS.Text);
H := StrToFloat(EditH.Text);
```

```
                    //рисование траектории
with Image1.Canvas do
begin
   while T<5 Do
   begin
   T:=T+0.005;
   Y:=380-Round(30*(V0*Sin(A*Pi/180)*T-G*T*T/2));
   X:= 5+Round(10*(V0*Cos(A*Pi/180)*T));
   Pixels[X,Y]:=clBlack;
   end;
   MoveTo(0,380); LineTo(500,380);    //ось X
   MoveTo(5,0); LineTo(5,500);        //ось Y
   MoveTo(5+Round(10*S),380);
   LineTo(5+Round(10*S),380-Round(30*H));//мишень
                    //шкала оси X
   N:=0;
   while N<500 do
   begin
   N:=N+100;
   MoveTo(5+N,380); LineTo(5+N,360);
   TextOut(5+N,380,IntToStr(Round(N/10)));
   end;
//шкала оси Y
   N:=0;
while N<400 do
   begin
   N:=N+100;
   MoveTo(0,380-N); LineTo(10,380-N);
   TextOut(0,380-N,IntToStr(Round(N/10)));
   end;
end;
end;
end.
```

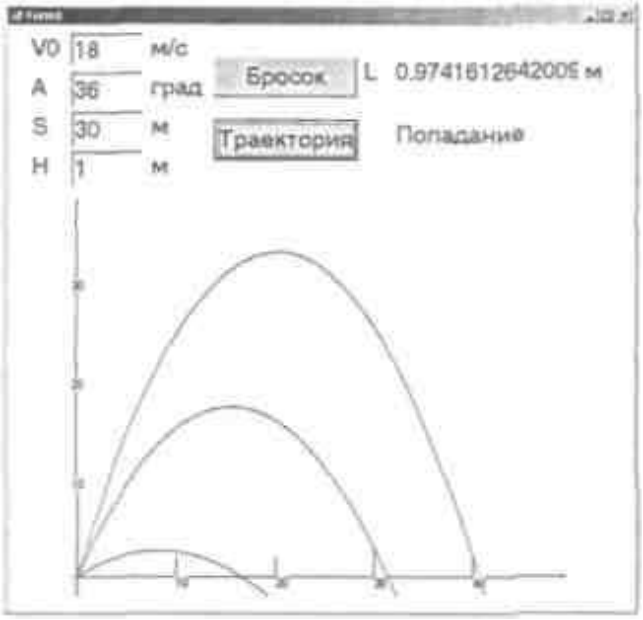
Проект «Попадание в стенку тела, брошенного под углом к горизонту» хранится в папке \Delphi\Phys1 **CD-ROM** 

Компьютерный эксперимент

6. Запустить проект и ввести значения начальной скорости, угла бросания, расстояния до мишени и ее высоты. Щелкнуть по кнопке *Бросок*. На метки будут выведены значение высоты мячика и результат броска.

В графическом поле появится траектория движения тела.

Подобрать значения начальной скорости и угла бросания мячика, обеспечивающие его попадание в мишень.



7. Например, при скорости бросания мячика $v_0 = 18$ м/с и угле бросания $\alpha = 36°$ мячик попадет в мишень высотой $h = 1$ м, находящуюся на расстоянии $S = 30$ м на высоте $l = 0{,}9741612642009$ м.

Анализ результатов и корректировка модели. Модернизируем проект так, чтобы для каждого значения скорости бросания получить диапазон значений углов, обеспечивающий попадание мячика в мишень.

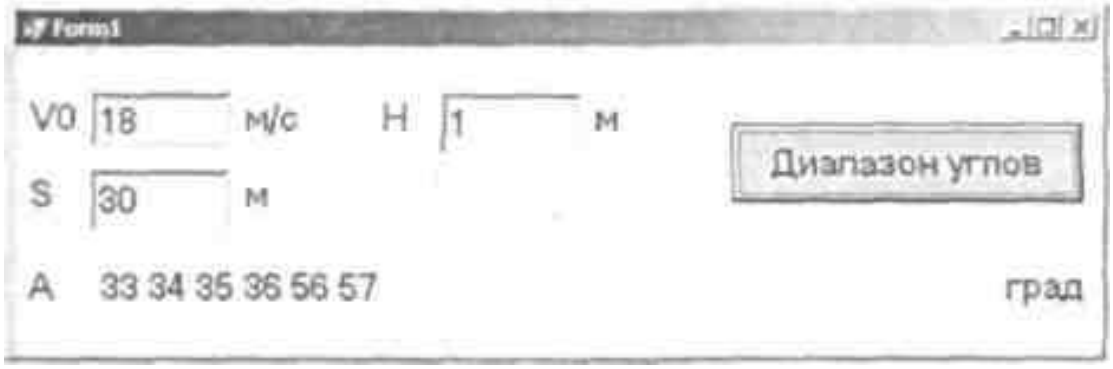
Проект «Диапазон углов, обеспечивающий попадание в стенку»

1. Удалить с формы текстовое поле EditA для ввода значения угла, метку Label2 для вывода результатов бросания и графическое поле Image1.
2. Использовать метку Label1 для вывода значений диапазона углов.
3. Внести изменения в программный код событийной процедуры:

```
procedure TForm1.Button1Click(Sender: Tobject);
const   //начало раздела объявления констант
G  = 9.81;
Pi = 3.14;
var     //начало раздела объявления переменных
V0: real;  //начальная скорость
```

```
A: integer;//угол бросания
S: real;   //расстояние до мишени
H: real;   //высота мишени
L: real;//высота мячика на заданном расстоянии
begin
//Ввод начальных значений
V0 := StrToFloat(EditV0.Text);
S  := StrToFloat(EditS.Text);
H  := StrToFloat(EditH.Text);
//Попадание в мишень
for A := 0 to 90 do
begin
L := S*Sin(Pi*A/180)/Cos(Pi*A/180)-G*Sqr(S)/
(2*Sqr(V0)*Sqr(Cos(Pi*A/180)));
if (0<L) And (L<H) then
Label1.Caption := Label1.Caption+' '
                  + IntToStr(A);
end;
end;
end.
```

4. Запустить проект и ввести скорость бросания мячика, расстояние до мишени и ее высоту.
Щелкнуть по кнопке *Диапазон углов*.



Получен не очевидный результат, оказывается существуют два диапазона углов от 33 до 36° и от 56 до 57°, которые обеспечивают попадание мячика при скорости бросания $v_0 = 18$ м/с в мишень высотой $h = 1$ м, находящуюся на расстоянии $S = 30$ м.

Проект «Диапазон углов, обеспечивающий попадание в стенку» хранится в папке \Delphi\Phys2 CD-ROM

Вопросы для размышления

1. Как можно реализовать получение результатов с заданной точность в языке программирования Delphi?
2. Имеет ли физический смысл вычисление значения высоты попадания мячика в мишень с точностью до тринадцати знаков после запятой? До какой точности целесообразно округлить полученное значение?

Практические задания для самостоятельного выполнения

CD-ROM

3.5. На основе формальной модели «Попадание в площадку тела, брошенного под углом к горизонту» (см. задание 3.2), построить компьютерную модель на языке программирования Delphi.

3.6. На языке программирования Delphi создать проект «Диапазон углов, обеспечивающий попадание в площадку», который позволяет определить для любой скорости бросания диапазон углов, обеспечивающих попадание в площадку.

3.2.4. Компьютерная модель движения тела в электронных таблицах

На основе формальной модели «Попадание в стенку тела, брошенного под углом к горизонту» создадим компьютерную модель с использованием электронных таблиц Microsoft Excel или StarOffice Calc (OpenOffice Calc).

Компьютерная модель «Попадание в стенку тела, брошенного под углом к горизонту»

Выделим в таблице определенные ячейки для ввода значений начальной скорости v_0 и угла α и вычислим по формулам (3.1) значения координат x и y тела для определенных значений времени t с заданным интервалом.

Для преобразования значений углов из градусов в радианы используем функцию *РАДИАНЫ( )*.

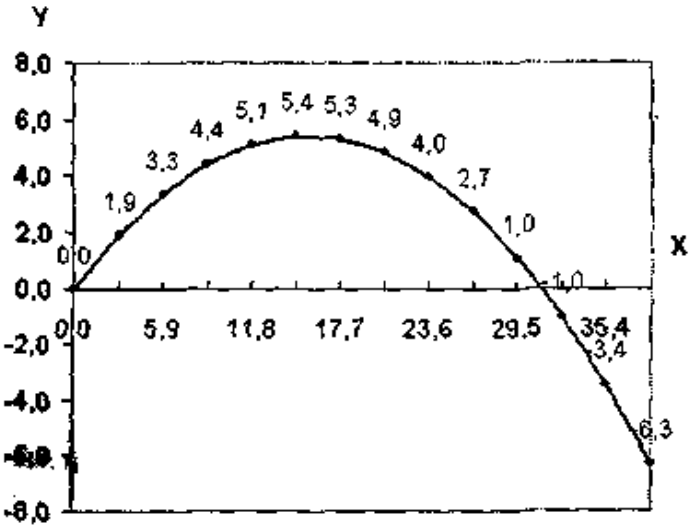
1. Для ввода начальной скорости будем использовать ячейку B1, а для ввода угла — ячейку B2.
2. Введем в ячейки A5:A18 значения времени с интервалом в 0,2 с.

3. В ячейки B5 и C5 введем формулы:
   =$B$1*COS(РАДИАНЫ($B$2))*A5
   =$B$1*SIN(РАДИАНЫ($B$2))*A5-4,9*A5*A5
4. Скопируем формулы в ячейки B6:B18 и C6:C18 соответственно.

	A	B	C
1	V_0 =	18,0 м/с	
2	α =	35,0 град	
3			
4	t	$x = v_0 \cdot \cos\alpha \cdot t$	$y = v_0 \cdot \sin\alpha \cdot t - g \cdot t^2/2$
5	0,0	0,0	0,0
6	0,2	2,9	1,9
7	0,4	5,9	3,3
8	0,6	8,8	4,4
9	0,8	11,8	5,1
10	1,0	14,7	5,4
11	1,2	17,7	5,3
12	1,4	20,6	4,8
13	1,6	23,6	4,0
14	1,8	26,5	2,7
15	2,0	29,5	1,0
16	2,2	32,4	-1,0
17	2,4	35,4	-3,5
18	2,6	38,3	-6,3

Визуализируем модель, построив график зависимости координаты *y* от координаты *x* (траекторию движения тела).

5. Построить диаграмму типа *график*, в которой используется в качестве категории диапазон ячеек B5:B18, а в качестве значений — диапазон ячеек C5:C18.



Исследование модели. Исследуем модель и определим с заданной точностью 0,1° значения диапазона углов бросания, которые обеспечивают попадание мячика в мишень

(например, при скорости бросания v_0 = 18 м/с в мишень высотой h = 1 м, находящуюся на расстоянии S = 30 м).

Воспользуемся для этого методом *Подбор параметра*, который позволяет задать значение функции и найти значение аргумента функции, который обеспечивает требуемое значение функции. При подборе параметра изменяется значение в ячейке аргумента функции до тех пор, пока не получится требуемое значение в ячейке самой функции.

6. Установить для ячеек точность один знак после запятой.
7. Ввести в ячейку B21 значение расстояния до мишени, в ячейку B22 — значение начальной скорости, в ячейку B23 — значение угла, а в ячейку B25 — формулу для вычисления высоты мячика над поверхностью для заданных начальных условий (см. формулу 3.2):

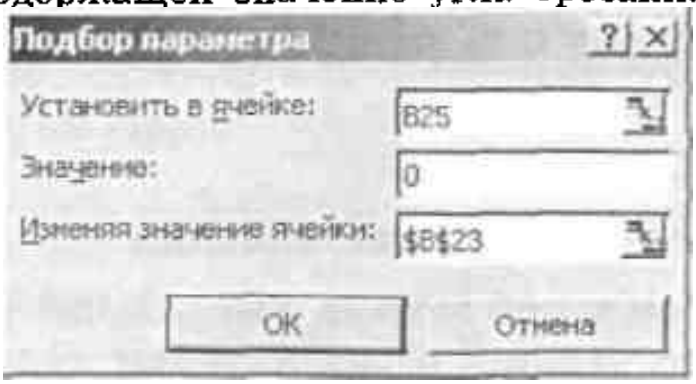
21	S =	30,0 м
22	V_0 =	18,0 м/с
23	α =	35,0 град
24		
25	L =	0,7 м

=B21*TAN(РАДИАНЫ(B23))-(9,81*B21^2)/(2*B22^2*COS(РАДИАНЫ(B23))^2)

Для заданных начальных условий (скорости бросания и расстояния до мишени) проведем поиск углов, которые дают попадание в мишень на высотах 0 м и 1 м.

Методом *Подбор параметра* будем сначала искать значение угла бросания, которое обеспечит попадание мячика в мишень на минимальной высоте 0 м.

8. Выделить ячейку B25, содержащую значение высоты мячика, и ввести команду [*Сервис· Подбор параметра...*]. На появившейся диалоговой панели ввести в поле *Значение:* наименьшую высоту попадания в мишень (т. е. 0). В поле *Изменяя значение ячейки:* ввести адрес ячейки B$23, содержащей значение угла бросания.



9. В ячейке B23 появится значение 32,6, т. е. значение минимального угла бросания мячика, которое обеспечивает попадание в мишень при заданных начальных условиях.

Методом *Подбор параметра* найдем теперь угол бросания, который обеспечит попадание мячика в мишень на максимальной высоте 1 м.

10. Выделить ячейку B25, содержащую значение высоты мячика, и ввести команду [*Сервис-Подбор параметра...*]. На появившейся диалоговой панели ввести в поле *Значение:* наибольшую высоту попадания в мишень (т. е. 1). В поле *Изменяя значение ячейки:* ввести адрес ячейки B$23, содержащей значение угла бросания.
11. В ячейке B23 появится значение 36,1, т. е. значение максимального угла бросания мячика, которое обеспечивает попадание в мишень при заданных начальных условиях.

Таким образом, исследование компьютерной модели в электронных таблицах показало, что существует диапазон значений угла бросания мячика от 32,6° до 36,1°, в котором обеспечивается попадание в мишень высотой 1 м, находящуюся на расстоянии 30 м, мячиком, брошенным со скоростью 18 м/с.

Если повторить процедуру определения диапазона углов при начальном значении угла в ячейке B23, равном 55°, то получим значения предельных углов 55,8° и 57,4°, т. е. второй диапазон углов: от 55,8° до 57,4°.

С учетом точности вычислений в электронных таблицах оба диапазона углов, обеспечивающие попадание в мишень при заданных начальных условиях, совпадают с результатами, полученными при исследовании компьютерных моделей на языках программирования Visual Basic и Delphi.

Модель «Попадание в стенку тела, брошенного под углом к горизонту» хранится в папке \calc\ в файле Model.xls на листе *Бросание тела под углом*

CD-ROM ☞

Практические задания для самостоятельного выполнения

CD-ROM ☞

3.7. На основе формальной модели «Попадание в площадку тела, брошенного под углом к горизонту» (см. задание 3.2), построить и исследовать компьютерную модель в электронных таблицах Microsoft Excel или StarOffice Calc (OpenOffice Calc).

3.3. Приближенное решение уравнений

Алгебра-9

На языке алгебры формальные модели записываются с помощью уравнений, точное решение которых основывается на поиске равносильных преобразований алгебраических выражений, позволяющих выразить переменную величину с помощью формулы.

Точные решения существуют только для некоторых уравнений определенного вида (линейные, квадратные, тригонометрические и др.), поэтому для большинства уравнений приходится использовать методы приближенного решения с заданной точностью (графические или численные).

3.3.1. Приближенное решение уравнений на языке Visual Basic

Задача. Найти корень уравнения $x^3 - \cos x = 0$ приближенными методами (графическим и численным методом деления числового отрезка аргумента пополам).

Графический метод. Построение графиков функций может использоваться для грубо приближенного решения уравнений. Для уравнений вида $f(x) = 0$, где $f(x)$ — некоторая непрерывная функция, корень (или корни) этого уравнения являются точкой (или точками) пересечения графика функции с осью X.

Формальная модель задана уравнением, для нахождения корня уравнения разработаем компьютерную модель на языке Visual Basic.

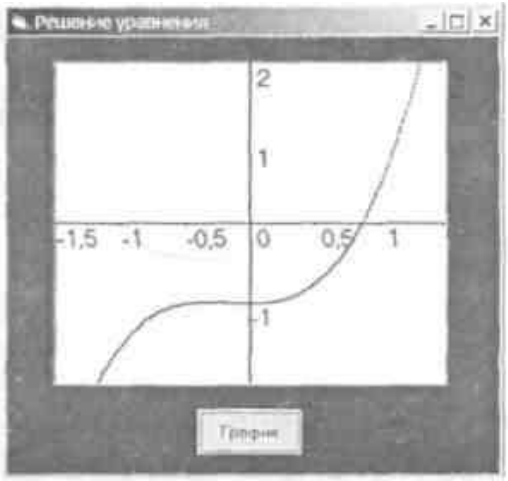
Проект «Построение графика функции»

Проект «Приближенное решение уравнения-1» на языке Visual Basic

```
1. Private Sub cmd1_Click()
   'Задание масштаба
   picGraph.Scale (-1.5, 2)-(1.5, -2)
   'Построение графика
   For sngX = -2 To 2 Step 0.01
   picGraph.PSet (sngX, sngX ^ 3 - Cos(sngX))
   Next sngX
   'Ось X
   picGraph.Line (-1.5, 0)-(1.5, 0)
```

```
For bytI = -1.5 To 1.5 Step 0.5
picGraph.PSet (bytI, 0)
picGraph.Print bytI
Next bytI
'Ось Y
picGraph.Line (0, 2)-(0, -2)
For bytI = -2 To 2
picGraph.PSet (0, bytI)
picGraph.Print bytI
Next bytI
End Sub
```

2. График функции пересекает ось X один раз и, следовательно, уравнение имеет один корень. По графику грубо приближенно можно определить, что $x \approx 0{,}8$.



Численный метод половинного деления. Для решения уравнений с заданной точностью можно применить разработанные в вычислительной математике численные методы решения уравнений путем последовательных приближений. Если мы знаем числовой отрезок аргумента x, на котором существует корень, и функция на краях этого отрезка принимает значения разных знаков, то можно использовать метод половинного деления.

Идея метода состоит в выборе точности решения и сведении первоначального отрезка $[A; B]$, на котором существует корень уравнения, к отрезку заданной точности. Процесс сводится к последовательному делению отрезков пополам точкой $C = (A+B)/2$ и отбрасыванию той половины отрезка ($[A; C]$ или $[C; B]$), на которой корня нет.

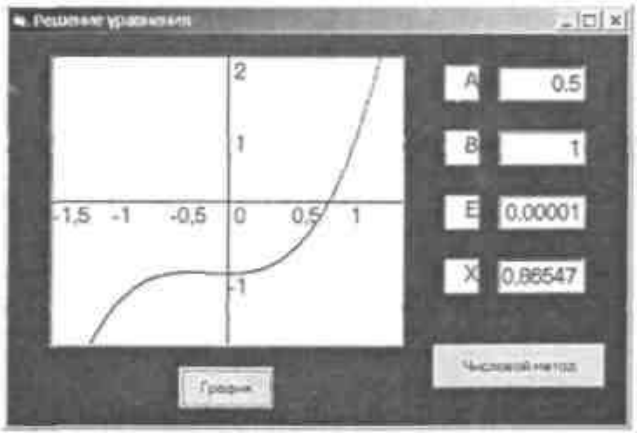
Выбор нужной половины отрезка основывается на проверке знаков значений функции на его краях. Выбирается та половина, на которой произведение значений функции на краях отрицательно, т. е. функция имеет разные знаки и пересекает ось абсцисс.

Процесс продолжается до тех пор, пока длина отрезка не станет меньше удвоенной точности. Деление этого отрезка пополам дает значение корня с заданной точностью $x \approx (A+B)/2$.

3. Поместить на форму текстовые поля для ввода числовых значений концов отрезка A и B, для ввода точности вычислений и поле для вывода значений корня.
4. Поместить на форму кнопку cmdNum и создать событийную процедуру cmdNum_Click(). Ввести программный код, позволяющий вычислить корень уравнения методом половинного деления с использованием цикла с постусловием, который будет выполняться, пока выполняется условие (dblB-dblA)/2>dblE:

```
Dim dblA, dblB, dblE, dblC As Double Private
Sub cmdNum_Click()
dblA = Val(txtA.Text)
dblB = Val(txtB.Text)
dblE = Val(txtE.Text)
Do
dblC = (dblA + dblB) / 2
If (dblA ^ 3 - Cos(dblA)) * (dblC ^ 3 -
  Cos(dblC)) < 0 Then
dblB = dblC
Else
dblA = dblC
End If
Loop While (dblB - dblA) / 2 > dblE
txtX.Text = (dblA + dblB) / 2
End Sub
```

5. Из графика функции видно, что корень находится на отрезке [0,5; 1]. Введем в текстовые поля значения концов числового отрезка, а также точность вычислений (например, 0,00001).
В текстовое поле будет выведено значения корня с заданной точностью:
$x \approx 0{,}86547$.



Проект «Приближенное решение уравнения-1» хранится в папке \VB\Math1\ CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

3.8. Создать проект на языке программирования Visual Basic «Приближенное решение уравнения-2», который позволяет найти корень уравнения $x^3 = \sin x$ приближенными методами (графическим и численным методом деления числового отрезка аргумента пополам).

3.3.2. Приближенное решение уравнений на языке Delphi

Задача. Найти корень уравнения $x^3 - \cos x = 0$ приближенными методами (графическим и численным методом деления числового отрезка аргумента пополам).

Графический метод. Построение графиков функций может использоваться для грубо приближенного решения уравнений. Для уравнений вида $f(x) = 0$, где $f(x)$ — некоторая непрерывная функция, корень (или корни) этого уравнения являются точкой (или точками) пересечения графика функции с осью X.

Формальная модель задана уравнением, для нахождения корня уравнения разработаем компьютерную модель на языке Delphi.

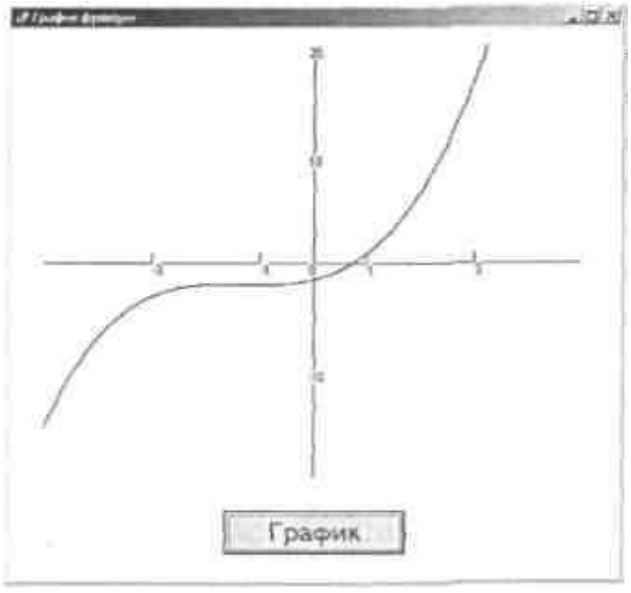
Проект «Построение графика функции»

Проект «Приближенное решение уравнения-1» на языке Delphi

1.
```
var
  X: real;
  Y: real;
  N: integer;
procedure TForm1.Button1Click(Sender: Tobject);
begin
with Image1.Canvas do
begin
   //график функции
   X:=-3;
   while X<3 Do
```

```
    begin
    X:=X+0.001;
    Y  :=  X*X*X-Cos(X);
    Pixels[Round(100*X)+200,200 -
    Round(20*Y)]:=clBlack;
    end;
    MoveTo(0,200);  LineTo(500,200);    //Ось  X
    MoveTo(250,0);  LineTo(250,500);    //Ось  Y
                    //Шкала  оси  X
    N:=0;
    while  N<500  do
    begin
    N:=N+100;
    MoveTo(N,190);  LineTo(N,210);
    TextOut(N,200,FloatToStr(Round(N-250)/50));
    end;
                    //Шкала  оси  Y
    N:=0;
    while  N<400  do
    begin
    N:=N+100;
    MoveTo(245,400-N);  LineTo(255,400-N);
    TextOut(245,400-N,
    FloatToStr(Round((N-200)/10)));
    end;
end;
end;
```

2. График функции пересекает ось X один раз и, следовательно, уравнение имеет один корень. По графику грубо приближенно можно определить, что $x \approx 0{,}8$.

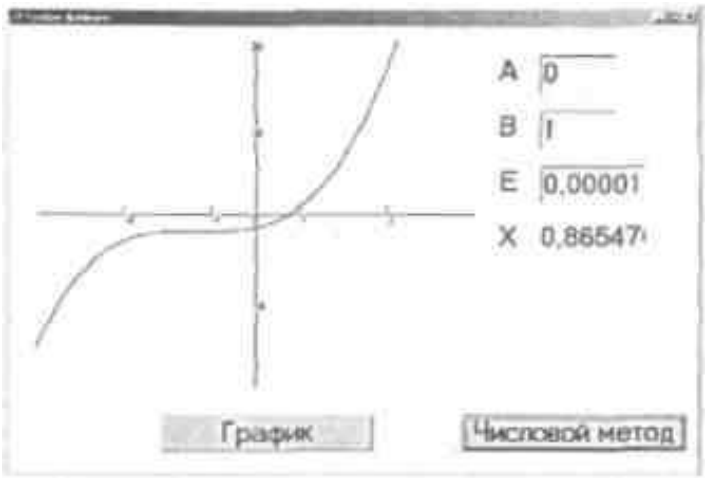


Численный метод половинного деления. Для решения уравнений с заданной точностью применим численный метод решения уравнений путем последовательных приближений методом половинного деления.

3. Поместить на форму:
   - два текстовых поля Edt1 и Edit2 для ввода числовых значений концов числового отрезка *A* и *B*;
   - текстовое поле Edit3 для ввода точности вычислений;
   - метку Label4 для вывода значений корня;
   - метки Label1, Label2, Label3 и Label4 для надписей к полям и метке.
4. Поместить на форму кнопку Button2 и создать событийную процедуру TForm1.Button2Click(). Ввести программный код, позволяющий вычислить корень уравнения методом половинного деления с использованием цикла с постусловием, который будет выполняться, пока выполняется условие (B-A)/2>E:

```
var
A: real;
B: real;
C: real;
E: real;
procedure TForm1.Button2Click(Sender: TObject);
begin
A:=StrToFloat(Edit1.Text);
B:=StrToFloat(Edit2.Text);
E:=StrToFloat(Edit3.Text);
Repeat
C:= (A+B)/2;
If (A*A*A - Cos(A)) * (C*C*C - Cos(C)) < 0
   Then B := C
   Else A := C
Until (B-A)/2<E;
Label4.Caption :=FloatToStr((A+B)/2);
end;
```

5. Из графика функции видно, что корень находится на отрезке [0; 1]. Введем в текстовые поля значения концов числового отрезка, а также точность вычислений (например, 0,00001).
   На метку будет выведено значения корня с заданной точностью: $x \approx 0{,}86547$.



Проект «Приближенное решение уравнения-1» хранится в папке \Delphi\Math1 CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

3.9. Создать проект на языке программирования Delphi «Приближенное решение уравнения-2», который позволяет найти корень уравнения $x^3 = \sin x$ приближенными методами (графическим и численным методом деления числового отрезка аргумента пополам).

3.3.3. Приближенное решение уравнений в электронных таблицах

Возможности электронных таблиц не ограничиваются вычислениями по формулам и построением диаграмм и графиков. Надстройка *Подбор параметра* в электронных таблицах Microsoft Excel и StarOffice Calc (OpenOffice Calc) устанавливается по умолчанию. Надстройка *Подбор параметра* позволяет задать значение функции и найти значение аргумента функции, который обеспечивает требуемое значение функции. При подборе параметра изменяется значение в ячейке аргумента функции до тех пор, пока не получится требуемое значение в ячейке самой функции.

С помощью надстройки *Подбор параметра* можно приближенно с заданной точностью решать уравнения.

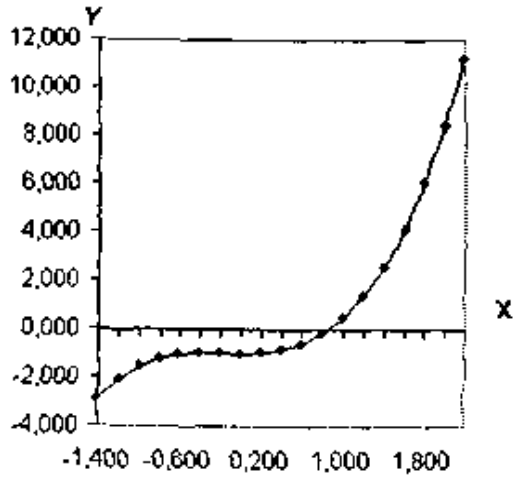
Задача. Найти корень уравнения $x^3 - \cos x = 0$ приближенными методами (графическим и с помощью метода *Подбор параметра*).

Представим функцию в табличной форме, построим ее график, который позволит определить корень уравнения грубо приближенно.

1. Представить заданное уравнение в табличной форме.

	A	B
1	x	$y = x^3 - \cos x$
2	-1,400	-2,914
3	-1,200	-2,090
4	-1,000	-1,540
5	-0,800	-1,209
6	-0,600	-1,041
7	-0,400	-0,985
8	-0,200	-0,988
9	0,000	-1,000
10	0,200	-0,972
11	0,400	-0,857
12	0,600	-0,609
13	0,800	-0,185
14	0,866	0,000
15	1,066	0,726
16	1,266	1,726
17	1,466	3,043
18	1,666	4,715
19	1,866	6,783
20	2,066	9,287

2. Для грубо приближенного определения корня построить диаграмму типа график.
   По графику грубо приближенно можно определить, что $x \approx 0{,}8$.

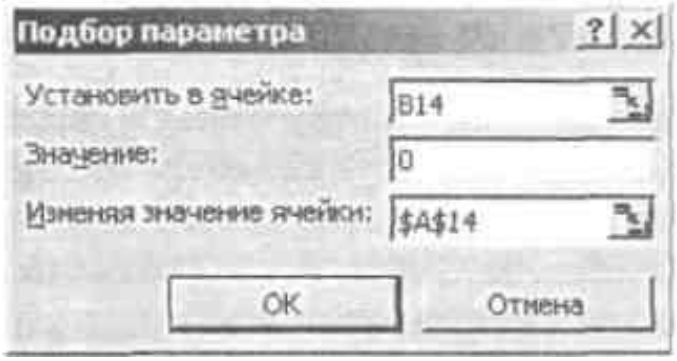


Для поиска решения с заданной точностью используем метод *Подбор параметра*. Точность подбора зависит от заданной точности представления чисел в ячейках таблицы (например, до трех знаков после запятой).

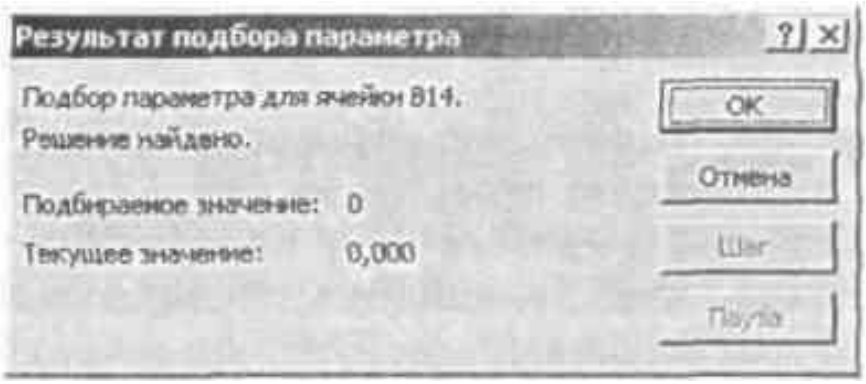
Методом подбора параметра необходимо определить значение аргумента x (ячейка A14), при котором значение функции y (ячейка B14) равно нулю.

3. Выделить ячейку со значением функции B14 и ввести команду [*Сервис-Подбор параметра...*].
4. На панели *Подбор параметра* в поле *Значение:* ввести требуемое значение функции (в данном случае 0).

В поле *Изменяя значение ячейки:* ввести адрес ячейки $A$14, в которой будет производиться подбор значения аргумента, и щелкнуть по кнопке *ОК*.



5. На панели *Результат подбора параметра* будет выведена информация о величине подбираемого и подобранного значений.



6. В ячейке аргумента A14 появится подобранное значение 0,855. Таким образом, корень уравнения $x \approx 0{,}855$ найден с заданной точностью.

Модель «Приближенное решение уравнений-1» хранится в папке \calc\ в файле Model.xls на листе *Решение уравнений*

CD-ROM

Практические задания для самостоятельного выполнения

CD-ROM

3.10. Создать компьютерную модель «Приближенное решение уравнений-2» с использованием электронных таблиц Microsoft Excel или StarOffice Calc (OpenOffice Calc), которая позволяет найти корень уравнения $x^3 = \sin x$ приближенными методами (графическим и с помощью метода *Подбор параметра*).

3.4. Вероятностные модели

3.4.1. Построение информационной модели с использованием метода Монте-Карло

Вероятностные модели базируются на использовании больших серий испытаний со случайными параметрами, причем точность полученных результатов зависит от количества проведенных опытов. Воспользуемся методом Монте-Карло для приближенного вычисления площадей геометрических фигур.

Качественная модель вычисления площадей геометрических фигур с использованием метода Монте-Карло. Сначала построим качественную вероятностную модель данного метода:

1) поместим геометрическую фигуру полностью внутрь квадрата;
2) будем случайным образом «бросать» точки в этот квадрат, т. е. с помощью генератора случайных чисел задавать координаты точек внутри квадрата;
3) будем считать, что отношение числа точек, попавших внутрь фигуры к общему числу точек в квадрате приблизительно равно отношению площади фигуры к площади квадрата, причем это отношение тем точнее, чем больше количество точек.

Формальная модель «Определение площади круга методом Монте-Карло». Построим формальную модель для вычисления площади круга радиуса r, центр которого совпадает с началом координат. Круг вписан в квадрат со стороной $2r$, площадь которого можно вычислить по формуле $S = 4r^2$ (рис. 3.2).

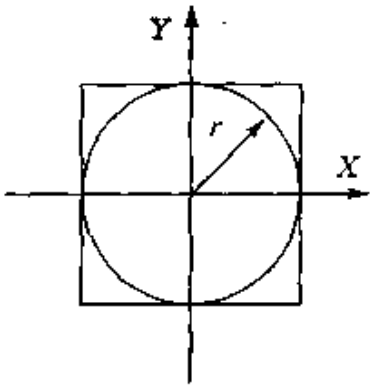


Рис. 3.2. Круг, вписанный в квадрат

Пусть N — количество точек, которые случайным образом генерируются внутри квадрата. Случайный выбор координат точек, которые попадают внутрь квадрата (N точек), должен производиться так, чтобы координаты точек x и y удовлетворяли условиям:

$$-r < x < r, \qquad -r < y < r.$$

Пусть M — количество точек, попавших внутрь круга, т. е. их координаты удовлетворяют условию:

$$x^2 + y^2 < r.$$

Тогда площадь круга можно вычислить по формуле:

$$S = 4r^2 \cdot M/N.$$

Практические задания для самостоятельного выполнения CD-ROM

3.11. Построить с использованием метода Монте-Карло формальную вероятностную модель «Бросание монеты».

3.4.2. Компьютерные модели, построенные с использованием метода Монте-Карло на языке Visual Basic

Разработаем на языке Visual Basic компьютерную модель, позволяющую определять площадь круга методом Монте-Карло.

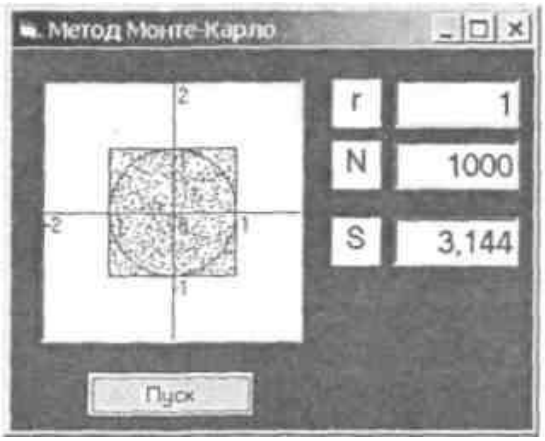
Проект «Определение площади круга с использованием метода Монте-Карло» на языке Visual Basic

1. Поместить на форму графическое поле `pic1`, в котором будет отображаться процесс случайной генерации точек. Нарисовать в графическом поле квадрат, круг и оси координат.
2. Поместить на форму:
   - два текстовых поля `txtR` для ввода радиуса окружности и `txtN` для ввода количества генерируемых точек;
   - текстовое поле `txtS` для вывода значения площади круга.
3. Поместить на форму три текстовых поля для обозначения назначения текстовых полей.
4. Поместить на форму кнопку `cmd1` и создать для нее событийную процедуру `cmd1_Click()`, которая обеспечивает:
   - ввод значения радиуса окружности — присваивание его переменной `R`;
   - ввод количества генерируемых точек — присваивание его переменной `N`;
   - генерацию случайных точек;
   - подсчет в переменной `M` количества точек, попавших внутрь круга; вычисление и вывод значения площади круга в текстовое поле:

```
Dim X, Y, S As Double, I, N, M, R As Long
Private Sub cmd1_Click()
M = 0
```

```
pic1.Cls
'Ввод значений
R = Val(txtR)
N = Val(txtN)
pic1.Scale (-(R + 1), R + 1)-(R + 1, -(R+1))
pic1.Line (-R, R)-(R, -R), , B
pic1.Circle (0, 0), R
'Генерация точек
For I = 1 To N
X = 2 * R * Rnd - R
Y = 2 * R * Rnd - R
pic1.PSet (X, Y)
If X ^ 2 + Y ^ 2 <= R Then M = M + 1
Next I
'Площадь
txtS.Text = 4 * R ^ 2 * (M / N)
'Ось X
pic1.Line (-(R + 1), 0)-(R + 1, 0)
For I = -(R + 1) To R + 1
pic1.PSet (I, 0)
pic1.Print I
Next I
'Ось Y
pic1.Line (0, -(R + 1))-(0, R + 1)
For I = -(R + 1) To R + 1
pic1.PSet (0, I)
pic1.Print I
Next I
End Sub
```

5. Ввести радиус окружности и количество генерируемых точек. После щелчка по кнопке *Пуск* в графическом поле будет отображен процесс генерации случайных точек, а в текстовое поле выведено значение площади круга.



Исследование модели. Существует геометрическая формула, позволяющая вычислить площадь круга $S = \pi r^2$. Если в процессе исследования модели в качестве радиуса окружности выбрать 1, то числовое значение площади круга будет соответствовать числу π.

Таким образом, с помощью метода Монте-Карло можно определить с необходимой точностью значение числа π (при увеличении количества генерируемых точек можно наблюдать все большее приближение значения площади к значению числа π).

Проект «Определение площади круга с использованием метода Монте-Карло» **CD-ROM** *хранится в папке \VB\Math2*

Практические задания для самостоятельного выполнения **CD-ROM**

3.12. Построить на основе формальной вероятностной модели (см. задание 3.11) проект «Бросание монеты» на языке программирования Visual Basic.

3.4.3. Компьютерные модели, построенные с использованием метода Монте-Карло на языке Delphi

Разработаем на языке Delphi компьютерную модель, позволяющую определять площадь круга методом Монте-Карло.

Проект «Определение площади круга с использованием метода Монте-Карло» на языке Delphi

1. Поместить на форму графическое поле `Image1`, в котором будет отображаться процесс случайной генерации точек.
   Нарисовать в графическом поле круг.
2. Поместить на форму:
   - два текстовых поля: `Edit1` для ввода радиуса окружности и `Edit2` для ввода количества генерируемых точек;
   - метку `Label1` для вывода значения площади круга.
3. Поместить на форму кнопку `Button1` и создать для нее событийную процедуру `TForm1.Button1Click()`, которая обеспечивает:

- ввод значения радиуса окружности — присваивание его переменной R;
- ввод количества генерируемых точек — присваивание его переменной N;
- генерацию случайных точек;
- подсчет в переменной M количества точек, попавших внутрь круга; вычисление и вывод значения площади круга на метку:

```
var
R: integer; //радиус
N: integer; //количество "брошенных" точек
X: real;    //координата точек X
Y: real;    //координата точек Y
M: integer;//количество точек, попавших в круг
I: integer; //счетчик цикла
procedure TForm1.Button1Click(Sender: Tobject);
begin
M := 0;
'Ввод значений
R := StrToInt(Edit1.Text);
N := StrToInt(Edit2.Text);
Image1.Canvas.Ellipse(0,0,200,200);
'Генерация точек
Randomize;
For I := 1 To N Do
begin
X := Random(100*R)/100;
Y := Random(100*R)/100;
Image1.Canvas.Pixels[Round(400*X)-200,200 -
Round(400*Y)]:=clBlack;
If X*X + Y*Y <= R
     Then M := M + 1
end;
'Площадь
Label1.Caption := FloatToStr(4*R*R*M/N);
end;
```

4. Ввести радиус окружности и количество генерируемых точек. После щелчка по кнопке *Пуск* в графическом поле будет отображен процесс генерации случайных точек, а на метку выведено значение площади круга.

Исследование модели. Существует геометрическая формула, позволяющая вычислить площадь круга $S = \pi r^2$. Если в процессе исследования модели в качестве радиуса окружности выбрать 1, то числовое значение площади круга будет соответствовать числу π.

Таким образом, с помощью метода Монте-Карло можно определить с необходимой точностью значение числа π (при увеличении количества генерируемых точек можно наблюдать все большее приближение значения площади к значению числа π).

Проект «Определение площади круга с использованием метода Монте-Карло» CD-ROM хранится в папке \Delphi\Math2\

Практические задания для самостоятельного выполнения CD-ROM

3.13. Построить на основе формальной вероятностной модели (см. задание 3.11) проект «Бросание монеты» на языке программирования Delphi.

3.5. Биологические модели развития популяций

Общая биология 10-11

3.5.1. Информационные модели развития популяций

В биологии при исследовании развития биосистем строятся *динамические модели изменения численности популяций* различных живых существ (бактерий, рыб, животных и т. д.) с учетом различных факторов. Взаимовлияние популяций рассматривается в моделях типа «жертва—хищник».

Формальная модель «Численность популяций». Изучение динамики популяций естественно начать с простейшей модели **неограниченного роста**, в которой численность популяции ежегодно увеличивается на определенный процент. Математическую модель можно записать с помощью рекуррентной формулы, связывающей численность популяции следующего года с численностью популяции текущего года, с использованием коэффициента роста a:

$$x_{n+1} = a \cdot x_n.$$

Например, если ежегодный прирост численности популяции составляет 5%, то $a = 1{,}05$.

В модели **ограниченного роста** учитывается эффект перенаселенности, связанный с нехваткой питания, болезнями и т. д., который замедляет рост популяции с увеличением ее численности. Введем коэффициент перенаселенности b, значение которого обычно существенно меньше a ($b \ll a$). Тогда, коэффициент ежегодного увеличения численности равен $(a - b \cdot x_n)$ и формула принимает вид:

$$x_{n+1} = (a - b \cdot x_n) \cdot x_n.$$

В модели **ограниченного роста с отловом** учитывается, что на численность популяций промысловых животных и рыб оказывает влияние величина ежегодного отлова. Если величина ежегодного отлова равна c, то формула принимает вид:

$$x_{n+1} = (a - b \cdot x_n) \cdot x_n - c.$$

Популяции обычно существуют не изолированно, а во взаимодействии с другими популяциями. Наиболее важным типом взаимодействия является взаимодействие между жертвами и хищниками (например, караси—щуки, зайцы—волки и т. д.). В модели **«жертва—хищник»** количество жертв x_n и количество хищников y_n связаны между собой. Количество встреч жертв с хищниками можно считать пропорциональной произведению количеств жертв и хищников, а коэффициент f характеризует возможность гибели жертвы при встрече с хищниками. В этом случае численность популяции жертв уменьшается на величину $f \cdot x_n \cdot y_n$ и формула для расчета численности жертв принимает вид:

$$x_{n+1} = (a - b \cdot x_n) \cdot x_n - c - f \cdot x_n \cdot y_n.$$

Численность популяции хищников в отсутствие жертв (в связи с отсутствием пищи) уменьшается, что можно описать рекуррентной формулой

$$y_{n+1} = d \cdot y_n,$$

где значение коэффициента $d < 1$ характеризует скорость уменьшения численности популяции хищников.

Увеличение популяции хищников можно считать пропорциональной произведению количеств жертв и хищников, а коэффициент e характеризует величину роста численности хищников за счет жертв. Тогда для численности хищников можно использовать формулу:

$$y_{n+1} = d \cdot y_n + e \cdot x_n \cdot y_n.$$

Практические задания для самостоятельного выполнения

CD-ROM

3.14. Построить формальную модель, описывающую численность популяций в модели «жертва—хищник с отловом», в которой производится отлов не только жертв, но и хищников.

3.5.2. Компьютерные модели развития популяций на языке Visual Basic

Построим на языке Visual Basic компьютерную модель, позволяющую исследовать изменение со временем численности популяций с использованием различных моделей: неограниченного роста, ограниченного роста, ограниченного роста с отловом и «жертва—хищник».

Проект «Численность популяций» на языке Visual Basic

1. Поместить на форму текстовые поля для ввода:
   - значений коэффициентов a, b, c и f, влияющих на изменение численности жертв: `txtA`, `txtB`, `txtC` и `txtF`;
   - значений коэффициентов d и e, влияющих на изменение численности хищников: `txtD` и `txtE`;
   - начальной численности популяций жертв и хищников: `txtX1` и `txtY1`;
   - количества рассматриваемых жизненных циклов (лет) `txtN`.
2. Поместить на форму текстовые поля для вывода численности популяции через заданное количество лет:
   - при неограниченном росте `txtXnn`;
   - при ограниченном росте `txtXno`;
   - при ограниченном росте с отловом `txtXnoo`;
   - в модели «жертва—хищник» `txtX_Y` и `txtY_X`.
3. Поместить на форму текстовые поля для имен соответствующих переменных.
4. Поместить на форму кнопку `cmd1` и начать создание событийной процедуры. Прежде всего, объявить переменные и обеспечить ввод начальных данных в соответствующие переменные:

```
Dim bytN, I As Byte, dblA, dblB, dblC, dblD,
dblE, dblF, dblX, dblY As Double
Private Sub cmd1_Click()
'Ввод данных
bytN = txtN.Text
```

```
dblA = txtA.Text
dblB = txtB.Text
dblC = txtC.Text
dblD = txtD.Text
dblE = txtE.Text
dblF = txtF.Text
dblX = txtX1.Text
dblY = txtY1.Text
End Sub
```

5. Поместить на форму графическое поле, в котором будут строиться графики зависимости численности популяций от количества прошедших лет. В событийную процедуру ввести:

```
'Масштаб и координаты
pic1.Scale (-1, 10)-(bytN + 1, -1)
pic1.Line (0, 0)-(bytN + 1, 0)
For I = 0 To bytN
pic1.PSet (I, 0)
pic1.Print I
Next I
pic1.Line (0, -1)-(0, 10)
For I = 0 To 10
pic1.PSet (0, I)
pic1.Print I
Next I
```

Для различных моделей динамики изменения численности популяций создать фрагменты кода событийной процедуры, где в цикле вычисляется численность популяции и строится график, а затем результат (численность популяций) выводится в текстовое поле.

6. Ввести код модели неограниченного роста:

```
dblX = txtX1.Text
For I = 1 To bytN
pic1.PSet (I, dblX)
dblX = dblA * dblX
Next I
txtXnn = dblX
```

7. Ввести код модели ограниченного роста:

```
dblX = txtX1.Text
For I = 1 To bytN
pic1.PSet (I, dblX), vbMagenta
dblX = (dblA - dblB * dblX) * dblX
```

```
Next I
txtXno = dblX
```

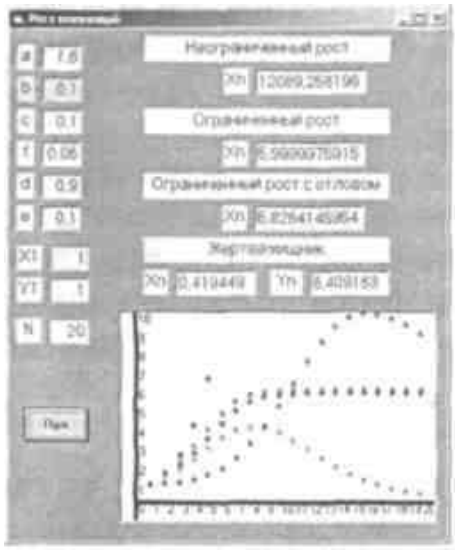
8. Ввести код модели ограниченного роста с отловом:

```
dblX = txtX1.Text
For I = 1 To bytN
pic1.PSet (I, dblX), vbBlue
dblX = (dblA - dblB * dblX) * dblX - dblC
Next I
txtXnoo = dblX
```

9. Ввести код модели «жертва—хищник»:

```
dblX = txtX1.Text
For I = 1 To bytN
pic1.PSet (I, dblX), vbGreen
pic1.PSet (I, dblY), vbRed
dblX = (dblA - dblB * dblX) * dblX - dblC -
dblF * dblX * dblY
dblY = dblD * dblY + dblE * dblX * dblY
Next I
txtX_Y = dblX
txtY_X = dblY
```

10. Запустить проект и ввести значения коэффициентов, начальное количество жертв и хищников и количество жизненных циклов (лет). (Для простоты примем начальные количества жертв и хищников равными 1.) Щелкнуть по кнопке *Пуск*. Графики будут показывать динамику развития популяций, а в текстовые поля будут выведены значения численности популяций.



Анализ полученных результатов. Из графиков видно, что модель неограниченного роста приводит к катастрофическому возрастанию численности популяции. В природе существуют отдельные периоды, когда создаются наиболее благоприятные условия для жизни какого-либо вида, и тогда мы являемся свидетелями появления несметного количества комаров, саранчи и т. д.

В моделях ограниченного роста и ограниченного роста с отловом динамика изменения численности популяции практически одинакова при определенных значениях коэффициента

отлова. Это означает, что могут существовать научно обоснованные нормы лова рыбы или охоты, которые практически не влияют на численность популяции с течением времени.

Исследование динамики численности популяций в модели «жертва—хищник» показывает, что численности популяций жертв и хищников тесно связаны. Так первоначальный рост численности жертв стимулирует резкий рост численности хищников, что с течением времени приводит к резкому уменьшению численности жертв и, спустя некоторое время, — к уменьшению численности самих хищников.

Проект «Численность популяций» хранится в папке \VB\Biol\ CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

3.15. На основе формальной модели развития популяций «жертва—хищник с отловом», в которой производится отлов не только жертв, но и хищников (задание 3.14), создать проект «Жертва—хищник с отловом» на языке программирования Visual Basic.

3.5.3. Компьютерные модели развития популяций на языке Delphi

Построим на языке Delphi компьютерную модель позволяющую исследовать изменение со временем численности популяций с использованием различных моделей: неограниченного роста, ограниченного роста, ограниченного роста с отловом и «жертва—хищник».

Проект «Численность популяций» на языке Delphi

1. Поместить на форму текстовые поля для ввода:
   - значений коэффициентов a, b, c и f, влияющих на изменение численности жертв: `EditA`, `EditB`, `EditC` и `EditF`;
   - значений коэффициентов d и e, влияющих на изменение численности хищников: `EditD` и `EditE`;
   - начальной численности популяций жертв и хищников: `EditX` и `EditY`;
   - количества рассматриваемых жизненных циклов (лет) `EditN`.
2. Поместить на форму метки для вывода численности популяции через заданное количество лет:

- при неограниченном росте Label1NR;
- при ограниченном росте Label1OR;
- при ограниченном росте с отловом Label1ORO;
- в модели «жертва—хищник» Label1X_Y и Label1Y_X.

3. Поместить на форму текстовые поля для имен соответствующих переменных.
4. Прежде всего, необходимо объявить переменные:

```
var
A: real;    //коэффициент роста популяции
B: real;    //коэффициент уменьшения популяции
C: real;    //коэффициент отлова
D: real;    //коэффициент уменьшения численности
            // хищников в отсутствие жертв
E: real;    //коэффициент увеличения численности
            // хищников при наличии жертв
F: real;    //коэффициент уменьшения численности
            // жертв при наличии хищников
X: real;    //первоначальное количество жертв
Y: real;    //первоначальное количество хищников
N: integer; //количество циклов (лет)
I: integer; //счетчик цикла
```

5. Поместить на форму кнопку Button1 и начать создание событийной процедуры TForm1.Button1Click(). Присвоить переменным значения, вводимые в текстовые поля, с использованием функций преобразования типов данных StrToFloat()и StrToInt():

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
//Ввод данных
A := StrToFloat(EditA.Text);
B := StrToFloat(EditB.Text);
C := StrToFloat(EditC.Text);
D := StrToFloat(EditD.Text);
E := StrToFloat(EditE.Text);
F := StrToFloat(EditF.Text);
X := StrToFloat(EditX.Text);
Y := StrToFloat(EditY.Text);
N := StrToInt(EditN.Text);
end;
```

6. Поместить на форму графическое поле Image1, в котором будут строиться графики зависимости численности популяций от количества прошедших лет. В событийной процедуре установить ширину линий рисования на холсте равную, например, 3 пикселям:

```
//Установка ширины линии рисования
Image1.Canvas.Pen.Width := 3;
```

7. Ввести код модели неограниченного роста, где:
   - задается начальная точка графика с использованием метода MoveTo();
   - задается цвет графика путем задания значения свойства Color;
   - в цикле вычисляется численность популяции и строится график с использованием метода LineTo();
   - конечная численность населения выводится на метку LabelNR с использованием функции преобразования типов данных FloatToStr(X):

```
//Неограниченный рост
Image1.Canvas.MoveTo(0,250);
Image1.Canvas.Pen.Color := clBlack;
For I:=1 to N  Do
begin
Image1.Canvas.LineTo(25*I-25,250-Round(25*X)+25);
X := A*X;
end;
LabelNR.Caption := FloatToStr(X);
```

8. Ввести код модели ограниченного роста:

```
//Ограниченный рост
X := StrToFloat(EditX.Text);
Image1.Canvas.MoveTo(0,250);
Image1.Canvas.Pen.Color := clDkGray;
For I:=1 to N Do
begin
Image1.Canvas.LineTo(25*I-25,250-
Round(25*X)+25);
X := (A-B*X)*X;
end;
LabelOR.Caption := FloatToStr(X);
```

9. Ввести код модели ограниченного роста с отловом:

```
//Ограниченный рост с отловом
X:= StrToFloat(EditX.Text);
Image1.Canvas.MoveTo(0,250);
Image1.Canvas.Pen.Color :=clBlue;
For I:=1 to N Do
begin
Image1.Canvas.LineTo(25*I-25,250-
Round(25*X)+25);
X := (A-B*X)*X-C;
```

```
Label1ORO.Caption := FloatToStr(X);
end;
Label1ORO.Caption := FloatToStr(X);
```

10. Ввести код модели «жертва—хищник» для вычисления численности жертв:

```
//Жертвы
X := StrToFloat(EditX.Text);
Y := StrToFloat(EditY.Text);
Image1.Canvas.MoveTo(0,250);
Image1.Canvas.Pen.Color := clGreen;
For I:=1 to N Do
begin
Image1.Canvas.LineTo(25*I-25,250-Round(25*X)+25);
X := (A-B*X)*X-C-F*X*Y;
Y := D*Y+E*X*Y;
end;
Label1X_Y.Caption := FloatToStr(X);
```

11. Ввести код модели «жертва—хищник» для вычисления численности хищников:

```
//Хищники
X := StrToFloat(EditX.Text);
Y := StrToFloat(EditY.Text);
Image1.Canvas.MoveTo(0,250);
Image1.Canvas.Pen.Color :=clRed;
For I:=1 to N Do
begin
Image1.Canvas.LineTo(25*I-25,250-Round(25*Y)+25);
X := (A-B*X)*X-C-F*X*Y;
Y := D*Y+E*X*Y;
end;
Label1Y_X.Caption := FloatToStr(Y);
```

12. Запустить проект. Ввести значения коэффициентов, начальное количество жертв и хищников и количество жизненных циклов (лет). (Для простоты примем начальные количества жертв и хищников равными 1.) Щелкнуть по кнопке *Пуск*. Графики будут показывать динамику развития популяций, а в текстовые поля будут выведены значения численности популяций.

Проект «Численность популяций» хранится в папке \Delphi\Biol\ CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

3.16. На основе формальной модели развития популяций «жертва—хищник с отловом», в которой производится отлов не только жертв, но и хищников (см. задание 3.14), создать проект «Жертва—хищник с отловом» на языке программирования Delphi.

3.5.4. Компьютерные модели развития популяций в электронных таблицах

Построим в электронных таблицах компьютерную модель позволяющую исследовать численность популяций с использованием различных моделей: неограниченного роста, ограниченного роста, ограниченного роста с отловом и «жертва—хищник».

Компьютерная модель «Численность популяций»

1. В ячейки B1 и B6 внести начальные значения численности популяций жертв и хищников. (Для простоты установим начальные количества жертв X1 и хищников Y1 равными 1.)
   В ячейки B2:B5 внести значения коэффициентов a, b, c и f, влияющих на изменение численности жертв.
   В ячейки B7 и B8 внести значения коэффициентов d и e, влияющих на изменение численности хищников.

	A	B
1		
2	X1=	1,00
3	a=	1,60
4	b=	0,10
5	c=	0,10
6	f=	0,06
7	Y1=	1,00
8	d=	0,90
9	e=	0,10

В столбце D будем вычислять численность популяции в соответствие с моделью неограниченного роста, в столбце E — ограниченного роста, в столбце F — ограниченного роста с отловом, в столбцах G и H — численность популяций жертв и хищников.

2. В ячейки D2, E2, F2, G2 и H2 ввести значения начальной численности популяций.
   В ячейку D3 внести рекуррентную формулу неограниченного роста =$B$2*D2.
   В ячейку E3 внести рекуррентную формулу ограниченного роста =($B$2–$B$3*E2)*E2.
   В ячейку F3 внести рекуррентную формулу ограниченного роста с отловом =($B$2–$B$3*F2)*F2–$B$4.

В ячейку G3 внести рекуррентную формулу роста жертв =($B$2–$B$3*G2)*G2–$B$4–$B$5*G2*H2.

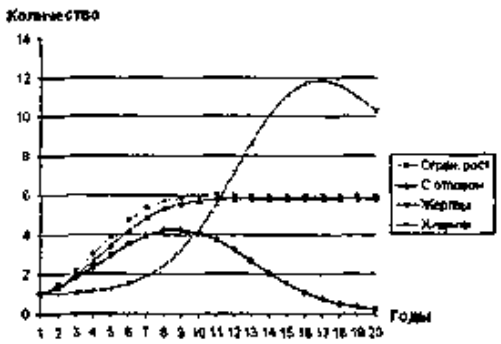
В ячейку H3 внести рекуррентную формулу роста хищников =$B$7*H2+$B$8*G2*H2.

3. Скопировать внесенные формулы в ячейки столбцов командами [*Правка-Заполнить-Вниз*].
В ячейках столбцов ознакомиться с динамикой изменения численности популяций.

D	E	F	G	H
Неогр. рост	Огран. рост	С отловом	Жертвы	Хищники
1,0	1,0	1,0	1,0	1,0
1,6	1,5	1,4	1,3	1,0
2,6	2,2	1,9	1,8	1,0
4,1	3,0	2,6	2,3	1,1
6,6	3,9	3,4	2,9	1,3
10,5	4,7	4,2	3,5	1,5
16,8	5,3	4,9	4,0	1,9
26,8	5,7	5,3	4,2	2,4
42,9	5,9	5,6	4,3	3,2
68,7	5,9	5,7	4,1	4,3
110,0	6,0	5,8	3,7	5,6
175,9	6,0	5,8	3,2	7,1
281,5	6,0	5,8	2,6	8,7
450,4	6,0	5,8	2,1	10,1
720,6	6,0	5,8	1,5	11,2
1152,9	6,0	5,8	1,1	11,7
1844,7	6,0	5,8	0,8	11,8
2951,5	6,0	5,8	0,5	11,5
4722,4	6,0	5,8	0,3	11,0
7555,8	6,0	5,8	0,2	10,3

Для визуализации компьютерной модели построим графики изменения популяций с течением времени. Численность популяции в модели неограниченного роста превосходит численность популяций в других моделях на три порядка, поэтому этот график строить не будем.

4. Выделить столбцы данных и построить диаграмму типа график.
Появятся графики изменения численности популяций в соответствии с моделями неограниченного роста, ограниченного роста, ограниченного роста с отловом, роста жертв и хищников.



Исследование модели. Изменяя значения начальной численности популяций, а также коэффициенты, можно получать различные варианты изменения численности популяций в зависимости от времени. При заданных значениях коэффициентов по графикам видно, что, начиная примерно с 10 года:

- отлов на численность популяции не влияет;
- количество хищников резко возрастает, а количество жертв уменьшается практически до нуля.

Модель «Численность популяций» хранится в папке \calc\ в файле Model.xls на листе *Численность популяций* CD-ROM

Практические задания для самостоятельного выполнения *CD-ROM*

3.17. На основе формальной модели развития популяций «жертва—хищник с отловом», в которой производится отлов не только жертв, но и хищников (задание 3.14), создать проект «Жертва—хищник с отловом» в электронных таблицах.

3.6. Оптимизационное моделирование в экономике

3.6.1. Информационные оптимизационные модели

В сфере управления сложными системами (например, в экономике) применяется оптимизационное моделирование, в процессе которого осуществляется поиск наиболее оптимального пути развития системы.

Критериями оптимальности могут быть различные параметры, например в экономике можно стремиться к максимальному количеству выпускаемой продукции, а можно — к ее низкой себестоимости. Оптимальное развитие соответствует экстремальному (максимальному или минимальному) значению выбранного целевого параметра.

Развитие сложных систем зависит от множества факторов (параметров), следовательно, значение целевого параметра зависит от множества параметров. Выражением такой зависимости является целевая функция

$$K = F(X_1, X_2, \ldots, X_n),$$

где K — значение целевого параметра; $X_1, X_2, \ldots, X_n$ — параметры, влияющие на развитие системы.

Цель исследования состоит в нахождении экстремума этой функции и определении значений параметров, при которых этот экстремум достигается. Если целевая функция нелинейная, то она имеет экстремумы, которые находятся определенными методами.

Однако часто целевая функция линейна и, соответственно, экстремумов не имеет. Задача поиска оптимального режима при линейной зависимости приобретает смысл только при на-

личии определенных ограничений на параметры. Если ограничения на параметры (система неравенств) также имеют линейный характер, то такие задачи являются задачами **линейного программирования**. (Термин «линейное программирование» в моделировании понимается как поиск экстремумов линейной функции, на которую наложены ограничения.)

Рассмотрим в качестве примера экономического моделирования поиск вариантов оптимального раскроя листов материала на заготовки определенного размера.

Содержательная постановка проблемы. В ходе производственного процесса из листов материала получают заготовки деталей двух типов А и Б тремя различными способами, при этом количество заготовок, получаемых из листа материала при каждом методе различается (табл. 3.1).

Таблица 3.1. **Способы раскроя заготовок**

Тип заготовки	Способы раскроя		
	1	2	3
А	10	3	8
Б	3	6	4

Необходимо выбрать оптимальное сочетание способов раскроя, для того чтобы получить 500 заготовок первого типа и 300 заготовок второго типа при расходовании наименьшего количества листов материала.

Формальная модель «Оптимизация раскроя». Параметрами, значения которых требуется определить, являются количества листов материала, которые будут раскроены различными способами:

X_1 – количество листов, раскроенное способом 1;

X_2 – количество листов, раскроенное способом 2;

X_3 – количество листов, раскроенное способом 3.

Тогда целевая функция, равная количеству листов материала, примет вид:

$$F = X_1 + X_2 + X_3.$$

Ограничения накладываются значениями требуемых количеств заготовок типов А и Б, тогда с учетом количества заготовок, получаемых различными способами, должны выполняться два равенства:

$$10 \cdot X_1 + 3 \cdot X_2 + 8 \cdot X_3 = 500,$$
$$3 \cdot X_1 + 6 \cdot X_2 + 4 \cdot X_3 = 300.$$

Кроме того, количества листов не могут быть отрицательными, поэтому должны выполняться неравенства:

$$X_1 \geq 0, \quad X_2 \geq 0, \quad X_3 \geq 0.$$

Таким образом, необходимо найти удовлетворяющие ограничениям значения параметров, при которых целевая функция принимает минимальное значение.

Практические задания для самостоятельного выполнения CD-ROM

3.18. Построить формальную модель «Оптимизация перевозки» перевозки компьютерного класса, состоящего из 15 компьютеров, с использованием единственного легкового автомобиля. Каждый компьютер упакован в две коробки (монитор и системный блок) и существует три варианта погрузки коробок в автомобиль (количество коробок двух типов):

Тип коробки	Вариант погрузки		
	1	2	3
Монитор	3	2	1
Системный блок	1	2	4

Необходимо выбрать оптимальное сочетание вариантов погрузки, для того чтобы перевести 15 коробок с мониторами и 15 коробок с системными блоками за минимальное количество рейсов автомобиля.

3.6.2. Построение и исследование оптимизационной модели на языке Visual Basic

Построим модель оптимизации раскроя листов материала на заготовки деталей на языке Visual Basic.

Набор параметров X_1, X_2 и X_3 (количества листов материала, которые должны быть раскроены различными способами) должен удовлетворять одновременно двум условиям, что на языке Visual Basic запишется следующим образом:

```
10*X1+3*X2+8*X3 = 500 And 3*X1+6*X2+4*X3 = 300
```

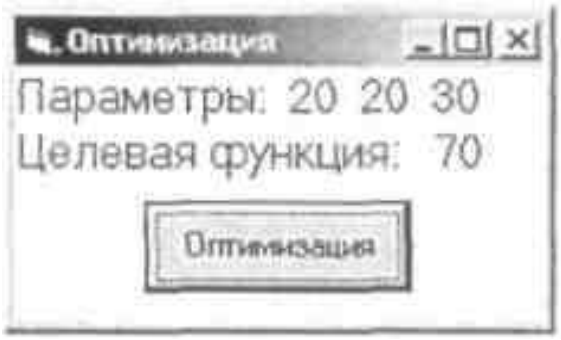
Для того чтобы найти наборы значений параметров, удовлетворяющих условию, необходимо произвести перебор всех возможных вариантов с помощью трех вложенных циклов. С помощью оператора условного перехода надо вывести значения набора параметров и минимальное значение целевой функции на форму.

Проект «Оптимизация раскроя» на языке Visual Basic

1. Поместить на форму frm1 кнопку cmd1 и создать для нее событийную процедуру cmd1_Click():

```
Dim X1, X2, X3 As Byte
Private Sub cmd1_Click()
For X1 = 0 To 100
For X2 = 0 To 100
For X3 = 0 To 100
If 10 * X1 + 3 * X2 + 8 * X3 = 500 And
3 * X1 + 6 * X2 + 4 * X3 = 300 Then
Print "Параметры: "; X1; X2; X3: Print
"Целевая функция:  "; X1 + X2 + X3
Next X3
Next X2
Next X1
End Sub
```

2. Запустить проект и щелкнуть по кнопке *Оптимизация*. На форму будут выведены наборы параметров и значение целевой функции.



3. Если наборов параметров будет несколько, то необходимо будет выполнить дополнительный поиск того набора, который соответствует минимальному значению целевой функции.

Проект «Оптимизация раскроя» хранится в папке \VB\Optim\ CD-ROM

Практические задания для самостоятельного выполнения

CD-ROM

3.19. На языке программирования Visual Basic создать проект «Оптимизация перевозки» перевозки компьютерного класса, состоящего из 15 компьютеров, с использованием единственного легкового автомобиля (см. задание 3.18).

3.6.3. Построение и исследование оптимизационной модели на языке Delphi

Набор параметров X_1, X_2 и X_3 (количества листов материала, которые должны быть раскроены различными способами) должен удовлетворять одновременно двум условиям, что на языке Delphi запишется следующим образом:

```
(10*X1+3*X2+8*X3=500) And (3*X1+6*X2+4*X3=300)
```

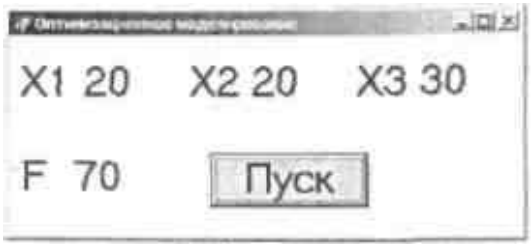
Для того чтобы найти наборы значений параметров, удовлетворяющих условию, необходимо произвести перебор всех возможных вариантов с помощью трех вложенных циклов. С помощью оператора условного перехода надо вывести значения набора параметров и минимальное значение целевой функции на форму.

Проект «Оптимизация раскроя» на языке Delphi

1. Поместить на форму Form1 кнопку Button1 и создать для нее событийную процедуру TForm1.Button1Click():

```
var
X1:integer;
X2:integer;
X3:integer;
F:integer;
procedure TForm1.Button1Click(Sender: Tobject);
begin
For X1 := 0 To 100 Do
begin
For X2 := 0 To 100 Do
begin
For X3 := 0 To 100 Do
begin
If (10*X1+3*X2+8*X3=500) And (3*X1+6*X2+4*X3=300)
      Then
      begin
      F:= X1+X2+X3;
      Label1.Caption := IntToStr(X1);
      Label2.Caption := IntToStr(X2);
      Label3.Caption := IntToStr(X3);
      Label4.Caption := IntToStr(F);
      end;
end;
end;
end;
end;
```

2. Запустить проект и щелкнуть по кнопке *Пуск*.
На метки будут выведены наборы параметров и значение целевой функции.



3. Если наборов параметров будет несколько, то необходимо будет выполнить дополнительный поиск того набора, который соответствует минимальному значению целевой функции.

Проект «Оптимизация раскроя» хранится в папке \Delphi\Optim\ CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

3.20. На языке программирования Delphi создать проект «Оптимизация перевозки» перевозки компьютерного класса, состоящего из 15 компьютеров, с использованием единственного легкового автомобиля (см. задание 3.18).

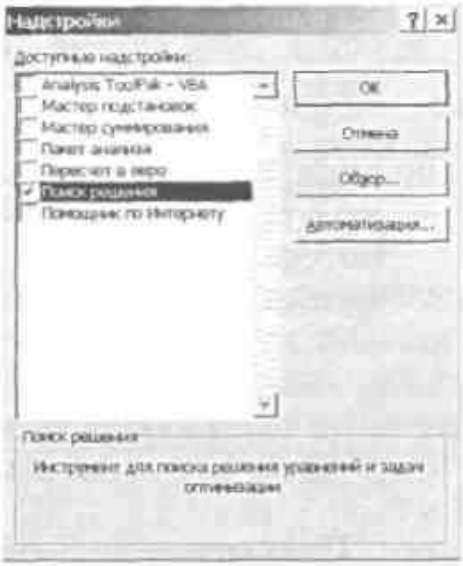
3.6.4. Построение и исследование оптимизационной модели в электронных таблицах

Надстройка *Поиск решения*. Возможности электронных таблиц не ограничиваются вычислениями по формулам и построением диаграмм и графиков. Задачи оптимизационного моделирования можно решать с помощью надстройки электронных таблиц *Поиск решения*, которая требует дополнительной установки.

Процедура поиска решения позволяет найти оптимальное значение формулы, содержащейся в ячейке, которая называется целевой. Эта процедура работает с группой ячеек, прямо или косвенно связанных с формулой в целевой ячейке. Чтобы получить по формуле, содержащейся в целевой ячейке, заданный результат, процедура изменяет значения во влияющих ячейках. Чтобы сузить множество значений, используемых в модели, применяются ограничения. Эти ограничения могут ссылаться на другие влияющие ячейки.

Компьютерная модель «Оптимизация раскроя»

1. Ввести команду [*Сервис-Надстройки*].
На панели *Надстройки* в списке *Доступные надстройки* выбрать нужные путем установки флажков, в нашем случае — флажка *Поиск решения*.
Щелкнуть по кнопке *ОК*.

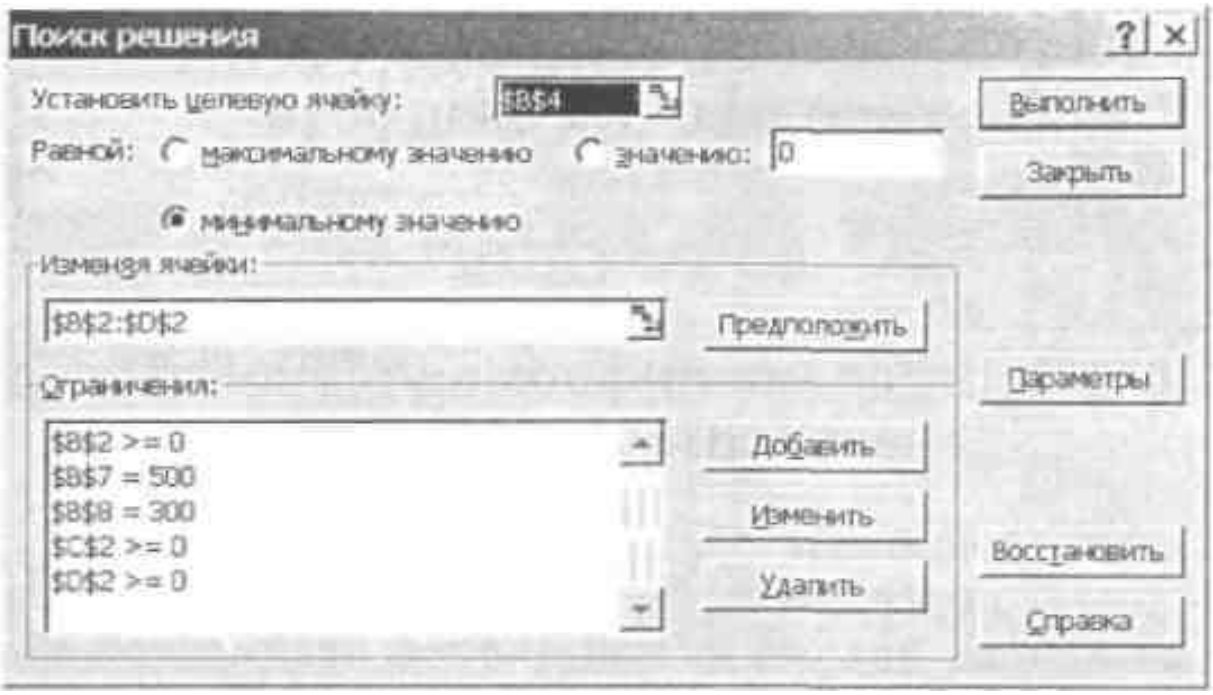


2. Ячейки B2, C2 и D2 выделить для хранения значений параметров X1, X2 и X3.
В ячейку B4 ввести формулу вычисления целевой функции: =B2+C2+D2.
В ячейку B7 ввести формулу вычисления количества заготовок типа А:
=10*B2+3*C2+8*D2.
В ячейку B8 ввести формулу вычисления количества заготовок типа Б: =3*B2+6*C2+4*D2.

	A	B	C	D
1		X1	X2	X3
2	Параметры:	0	0	0
3				
4	Целевая функция:	0		
5				
6	Ограничения			
7	Кол-во заготовок А:	0		
8	Кол-во заготовок Б:	0		

Исследование модели. Для поиска оптимального набора значений параметров, который соответствует минимальному значению целевой функции, воспользоваться надстройкой электронных таблиц *Поиск решения*.

3. Ввести команду [*Сервис-Поиск решения...*]. На появившейся диалоговой панели *Поиск решения* установить:
   - адрес целевой ячейки;
   - вариант оптимизации значения целевой ячейки (максимизация, минимизация или подбор значения);
   - адреса ячеек, значения которых изменяются в процессе поиска решения (в которых хранятся значения параметров);
   - ограничения (типа «равно» для ячеек, хранящих количество деталей, и типа «больше или равно» для параметров):



4. *Щелкнуть по кнопке Выполнить.* В ячейке целевой функции появится значение 70, а в ячейках параметров — значения 20, 20, 30.

	A	B	C	D
1		X1	X2	X3
2	Параметры:	20	20	30
3				
4	Целевая функция:	70		
5				
6	Ограничения			
7	Кол-во заготовок А:	500		
8	Кол-во заготовок Б:	300		

Таким образом, для изготовления 500 деталей А и 300 деталей Б требуется 70 листов материала, при этом 20 листов необходимо раскроить по первому, 20 листов — по второму и 30 листов — по третьему варианту.

Модель «Оптимизация раскроя» хранится в папке \calc\ в файле Model.xls на листе *Оптимизационное моделирование*

CD-ROM

Практические задания для самостоятельного выполнения

CD-ROM

3.21. В электронных таблицах построить компьютерную модель «Оптимизация перевозки» перевозки компьютерного класса, состоящего из 15 компьютеров, с использованием единственного легкового автомобиля (см. задание 3.18).

3.7. Экспертные системы распознавания химических веществ

Химия-9 ☞

3.7.1. Построение информационной модели экспертной системы

Профессиональные экспертные системы — это интеллектуальные программы, способные делать логические выводы на основе знаний из конкретной предметной области, обеспечивающие решение диагностических задач и способные заменить специалиста (эксперта).

Простейшие модели экспертных систем — учебные экспертные системы — способны «решать» гораздо более простые задачи и, естественно, имеют намного более простую структуру. В школьном курсе встречается достаточно много учебных ситуаций, когда ученик выступает в роли эксперта и должен распознать (идентифицировать) тот или иной объект. Обычно такие задачи выполняются учеником методом проб и ошибок, без осознания и фиксации стратегии поиска.

Создание учебной экспертной системы как раз и является осознанием и фиксацией последовательности рассуждений (действий), которая приводит к распознаванию того или иного объекта среди некоторой совокупности.

В качестве примера можно рассмотреть лабораторную работу по химии «Распознавание химических удобрений». Учащемуся даются удобрения, химические реактивы и справочная таблица по взаимодействию шести различных удобрений с некоторыми реактивами (табл. 3.2). Учащемуся предлагается распознать каждое из удобрений.

Стратегия поиска может быть представлена в виде дерева поиска на основе структуры «если-то-иначе», причем может быть множество различных деревьев с различным количеством шагов. Выбор оптимальной стратегии распознавания (достижения цели за минимальное число шагов) и будет являться созданием учебной экспертной системы. Такая стратегия будет реализована, если каждый шаг будет максимально уменьшать неопределенность (нести максимальное количество информации).

Таблица. 3.2. **Свойства удобрений**

№	Внешний вид	Взаимодействие раствора удобрения			Удобрение (результат распознавания)
		с H_2SO_4	с BaCl	с раствором щелочи	
1	Белая кристаллическая масса или гранулы	Выделяется бурый газ	—	Ощущается запах аммиака	Аммиачная селитра
2	Крупные бесцветные кристаллы	Выделяется бурый газ	Небольшое помутнение раствора	—	Натриевая селитра
3	Мелкие светло-серые кристаллы	—	Выпадает белый осадок	Ощущается запах аммиака	Сульфат аммония
4	Светло-серый порошок или гранулы	—	Выпадает белый осадок	—	Суперфосфат
5	Розовые кристаллы	—	—	—	Сильвинит
6	Бесцветные кристаллы	—	—	—	Калийная соль

Формальная модель экспертной системы «Распознавание удобрений». На первом шаге разделим шесть веществ на две группы по условию «при взаимодействии с H_2SO_4 выделяется бурый газ», если условие выполняется, то это вещества первой группы под номерами 1 и 2, если не выполняется, то это вещества второй группы под номерами 3, 4, 5 и 6.

Для идентификации веществ первой группы достаточно проверить справедливость условия «при взаимодействии с раствором щелочи ощущается запах аммиака». Если условие выполняется, то это вещество 1 — аммиачная селитра, если не выполняется, то это вещество 2 — натриевая селитра.

Для идентификации веществ второй группы сначала необходимо проверить справедливость условия «при взаимодействии с BaCl выпадает белый осадок». Если условие выполняется, то это вещества 3 и 4, если не выполняется, то это вещества 5 и 6.

Для идентификации веществ 3 и 4 достаточно проверить справедливость условия «при взаимодействии с раствором щелочи ощущается запах аммиака». Если условие выполняется, то это вещество 3 — сульфат аммония, если не выполняется, то это вещество 4 — суперфосфат.

Для идентификации веществ 5 и 6 достаточно проверить справедливость условия «внешний вид — розовые кристал-

лы». Если условие выполняется, то это вещество 5 — сильвинит, если не выполняется, то это вещество 6 — калийная соль.

Целесообразно представить иерархическую модель экспертной системы в виде блок-схемы (рис. 3.3).

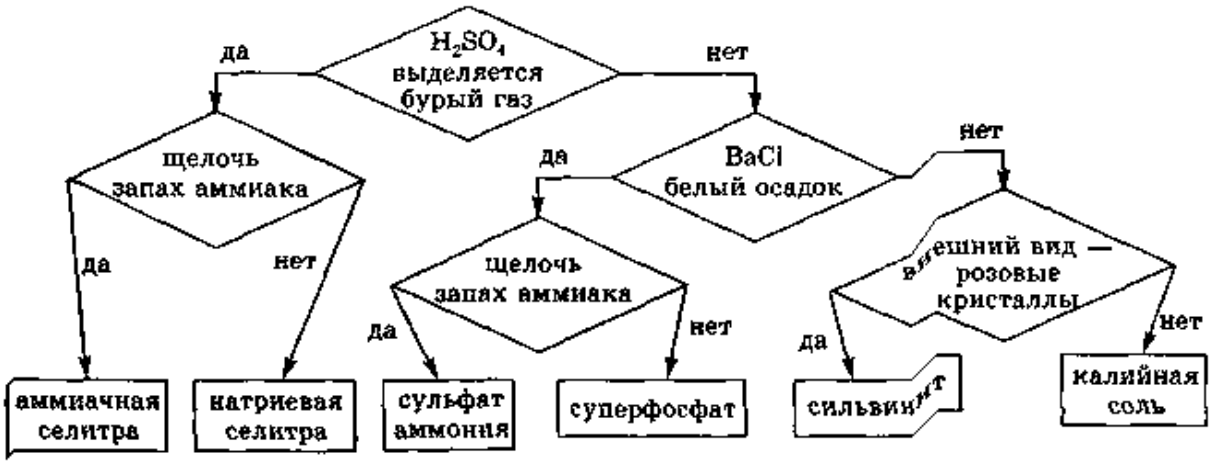


Рис. 3.3. Блок-схема экспертной системы «Распознавание удобрений»

Практические задания для самостоятельного выполнения

CD-ROM

3.22. Построить формальную модель экспертной системы «Распознавание пластмасс» для лабораторной работы по химии «Распознавание пластмасс».

3.7.2. Модель экспертной системы на языке Visual Basic

Реализуем экспертную систему распознавания удобрений с использованием языка Visual Basic. Экспертная система задает пользователю серию вопросов, анализирует ответы и сравнивает с имеющимися в ней фактами. При этом производится логический вывод и формируется ответ на интересующий пользователя вопрос, т. е. определяется название удобрения.

Результат распознавания (названия удобрений) будем помещать в управляющий элемент класса `ListBox` (список), который удобен для ввода элементов списка с помощью метода `AddItem`.

Проект «Распознавание удобрений» на языке Visual Basic

1. Поместить на форму кнопку `cmd1` и список `lst1`.

Первую развилку (условие «при взаимодействии с H_2SO_4 выделяется бурый газ») реализуем в форме событийной процедуры, а остальные — в форме общих процедур:

- для идентификации удобрений первой группы (1-ого и 2-го) создадим процедуру Щелочь1 (условие «при взаимодействии с раствором щелочи ощущается запах аммиака»);
- для идентификации удобрений второй группы сначала необходимо создать процедуру Соль (условие «при взаимодействии с BaCl выпадает белый осадок»);
- для идентификации 3-го и 4-го удобрений создадим процедуру Щелочь2 (условие «при взаимодействии с раствором щелочи ощущается запах аммиака»);
- для идентификации 5-го и 6-го удобрений создадим процедуру Внешний_вид (условие «внешний вид — розовые кристаллы»).

2. Создать событийную процедуру cmd1_Click(), которая содержит вызовы общих процедур Щелочь1 и Соль:

```
Dim bytA As Byte
Private Sub cmd1_Click()
bytA = MsgBox("При взаимодействии с серной
кислотой выделяется бурый газ?", 36,
"Первый вопрос")
If bytA = 6 Then Щелочь1 Else Соль
End Sub
```

3. Создать общую процедуру Щелочь1, которая позволяет распознать 1-е и 2-е удобрения:

```
Sub Щелочь1()
bytA = MsgBox("При взаимодействии со щелочью
ощущается запах аммиака?", 36,
"Второй вопрос")
If bytA = 6 Then lst1.AddItem
"1.Аммиачная селитра" Else
lst1.AddItem "2.Натриевая селитра"
End Sub
```

4. Создать общую процедуру Соль, которая содержит вызовы общих процедур Щелочь2 и Внешний_вид:

```
Sub Соль()
bytA = MsgBox("При взаимодействии с солью
выпадает белый осадок?", 36, "Второй вопрос")
If bytA = 6 Then Щелочь2 Else Внешний_вид
End Sub
```

5. Создать общую процедуру Щелочь2, которая позволяет распознать 3-е и 4-е удобрение:

```
Sub Щелочь2()
bytA = MsgBox("При взаимодействии со щелочью
ощущается запах аммиака?", 36,
"Третий вопрос")
If bytA = 6 Then lst1.AddItem "3.Сульфат
    аммония" Else lst1.AddItem "4.Суперфосфат"
End Sub
```

6. Создать общую процедуру Внешний_вид, которая позволяет распознать 5-е и 6-е удобрения:

```
Sub Внешний_вид()
bytA = MsgBox("Розовые кристаллы?", 36,
"Третий вопрос")
If bytA = 6 Then lst1.AddItem "5.Сильвинит"
Else lst1.AddItem "6.Калийная соль"
End Sub
```

Компьютерный эксперимент. Работа с экспертной системой позволит более эффективно спланировать и провести распознавание удобрений в процессе выполнения лабораторной работы по химии.

7. Запустить экспертную систему и проводить химические опыты в соответствии с задаваемыми вопросами.
   Выполнить процедуру распознавания для каждого вещества.

Проект «Распознавание удобрений» хранится в папке \VB\HIM CD-ROM

Практические задания для самостоятельного выполнения

CD-ROM

3.23. На языке программирования Visual Basic создать проект «Распознавание пластмасс» для лабораторной работы по химии «Распознавание пластмасс» (см. задание 3.22).

3.7.3. Модель экспертной системы на языке Delphi

Реализуем экспертную систему распознавания удобрений с использованием языка Delphi. Экспертная система задает пользователю серию вопросов, анализирует ответы и сравнивает с имеющимися в ней фактами. При этом производится логический вывод и формируется ответ на интересующий пользователя вопрос, т. е. определяется название удобрения.

Результат распознавания (названия удобрений) будем помещать в управляющий элемент `ListBox` (список), который удобен для ввода элементов списка с помощью метода `Items.Add()`.

Проект «Распознавание удобрений» на языке Delphi

1. Поместить на форму кнопку `Button1` и список `ListBox1`.

Первую развилку (условие «при взаимодействии с H_2SO_4 выделяется бурый газ») реализуем в форме событийной процедуры, а остальные — в форме общих процедур:

- для идентификации удобрений первой группы (1-го и 2-го) создадим процедуру `Sheloch1` (условие «при взаимодействии с раствором щелочи ощущается запах аммиака»);
- для идентификации удобрений второй группы сначала необходимо создать процедуру `Sol` (условие «при взаимодействии с BaCl выпадает белый осадок»);
- для идентификации 3-го и 4-го удобрений создадим процедуру `Sheloch2` (условие «при взаимодействии с раствором щелочи ощущается запах аммиака»);
- для идентификации 5-го и 6-го удобрений создадим процедуру `Vn_Vid` (условие «внешний вид — розовые кристаллы).

2. Определить переменные и процедуры. Создать событийную процедуру `TForm1.Button1Click()`, которая содержит вызовы общих процедур `Sheloch1` и `Sol`:

```
var
A:   integer;
Form1: TForm1;
procedure Sheloch1;
procedure Sol;
```

```
procedure Sheloch2;
procedure Vn_Vid;
procedure TForm1.Button1Click(Sender: Tobject);
begin
A := MessageDlg('При взаимодействии с серной
кислотой выделяется бурый газ?',
MtConfirmation, [mbYes,mbNo],0);
if A = idYes then Sheloch1 else Sol;
end;
```

3. **Создать общую процедуру** Sheloch1, **которая позволяет распознать 1-е и 2-е удобрения:**

```
procedure Sheloch1;
begin
A := MessageDlg('При взаимодействии со щелочью
ощущается запах аммиака?',
MtConfirmation, [mbYes,mbNo],0);
If A = idYes
Then Form1.ListBox1.Items.Add('1.Аммиачная
селитра')
Else Form1.ListBox1.Items.Add('2.Натриевая
селитра');
end;
```

4. **Создать общую процедуру** Sol, **которая содержит вызовы общих процедур** Sheloch2 **и** Vn_Vid:

```
procedure Sol;
begin
A := MessageDlg('При взаимодействии с солью
выпадает белый осадок?', MtConfirmation,
[mbYes,mbNo],0);
If A = idYes Then Sheloch2 Else Vn_Vid;
end;
```

5. **Создать общую процедуру** Sheloch2, **которая позволяет распознать 3-е и 4-е удобрение:**

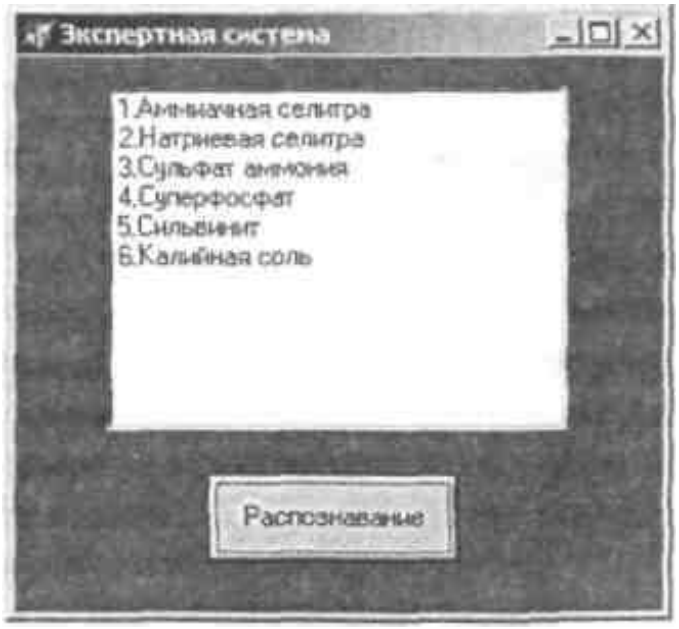
```
procedure Sheloch2;
begin
A := MessageDlg('При взаимодействии со щелочью
ощущается запах аммиака?',
MtConfirmation, [mbYes,mbNo],0);
If A = idYes
Then Form1.ListBox1.Items.Add('3.Сульфат
аммония')
Else Form1.ListBox1.Items.Add('4.Суперфосфат');
end;
```

6. Создать общую процедуру Vn_Vid, которая позволяет распознать 5-ое и 6-ое удобрения:

```
procedure Vn_Vid;
begin
A := MessageDlg('Розовые кристаллы?',
MtConfirmation, [mbYes,mbNo],0);
If A = idYes
Then Form1.ListBox1.Items.Add('5.Сильвинит')
Else Form1.ListBox1.Items.Add('6.Калийная
     соль');
end;
```

Компьютерный эксперимент. Работа с экспертной системой позволит более эффективно спланировать и провести распознавание удобрений в процессе выполнения лабораторной работы по химии.

7. Запустить экспертную систему и проводить химические опыты в соответствии с задаваемыми вопросами. Выполнить процедуру распознавания для каждого вещества.



Проект «Распознавание удобрений» хранится в папке \Delphi\Him\ CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

3.24. На языке программирования Delphi создать проект «Распознавание пластмасс» для лабораторной работы по химии «Распознавание пластмасс» (см. задание 3.22).

3.8. Геоинформационные модели в электронных таблицах Microsoft Excel

Надстройка электронных таблиц *Microsoft Map*. Надстройка электронных таблиц *Microsoft Map* позволяет создавать геоинформационные модели, которые представляют собой географические карты с отображенными на них статистическими данными. (Надстройка *Microsoft Map* входит в состав электронных таблиц Microsoft Excel 97.)

Отображение информации на географических картах может производиться различными способами: закрашиванием стран различными цветами, построением диаграмм и т. д. На географическую карту могут быть дополнительно выведены различные слои объектов: города, дороги, аэропорты и др.

Надстройка электронных таблиц *Microsoft Map* содержит набор электронных географических карт (карты *Страны мира*, *Европа*, *Россия* и др.) и книгу электронных таблиц (файл mapstats.xls), содержащую статистические сведения о количестве населения в различных странах мира.

В приведенном фрагменте электронной таблицы (рис. 3.4) названия стран отображаются в столбце A, а количество населения — в столбце B.

	A	B
1	Название	Население
2	АВСТРАЛИЯ	17661468
3	АВСТРИЯ	7914127
4	АЗЕРБАЙДЖАН	7021178
5	АЗОРСКИЕ О-ВА (ПОРТ.)	236000
6	АЛБАНИЯ	1626315
7	АЛЖИР	22600957
8	АНГИЛЬЯ	9200
9	АНГОЛА	4830449
10	АНДОРРА	61599
11	АНТИГУА И БАРБУДА	64794
12	АРГЕНТИНА	32712930
13	АРМЕНИЯ	3611700
14	АРУБА (НИДЕР.)	66687
15	АФГАНИСТАН	15513267
16	БАГАМСКИЕ О-ВА	264176
17	БАНГЛАДЕШ	109291000
18	БАРБАДОС	255200
19	БАХРЕЙН	520653
20	БЕЛАРУСЬ	10222649

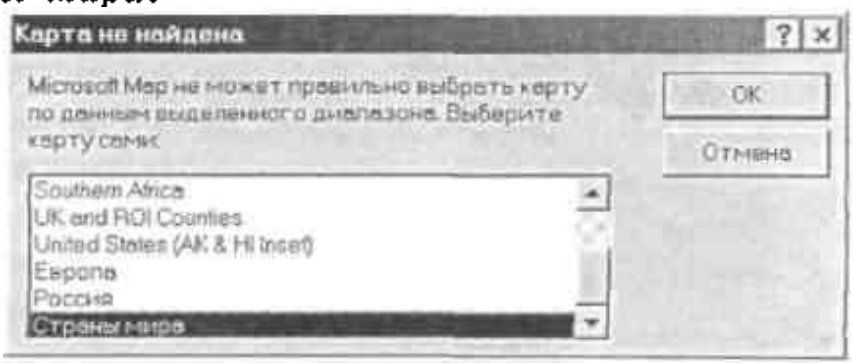
Рис. 3.4. Фрагмент электронной таблицы с данными о количестве населения

Задание. Отобразить на географической карте *Страны мира* статистическую информацию о количестве населения стран мира.

Построение геоинформационной модели «Население стран мира» с использованием надстройки электронных таблиц *Microsoft Map*

1. Запустить электронные таблицы Microsoft Excel командой [*Программы-Microsoft Excel*].

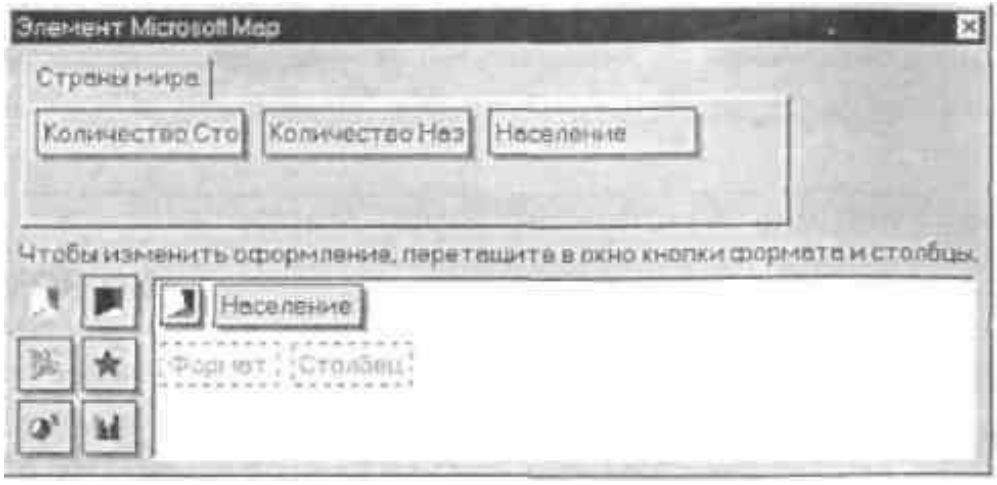
2. Открыть файл mapstats.xls и перейти на лист *Страны мира*.
3. Для вызова карты щелкнуть по кнопке *Карта*. На появившейся диалоговой панели выбрать требуемую карту *Страны мира*.



4. Для вывода на карту статистических данных ввести команду [*Вставка-Данные*].
   Выделить на листе электронных таблиц столбцы, содержащие названия стран и численность населения. Созданная карта отобразится на листе и откроется диалоговая панель *Оформление карты*.

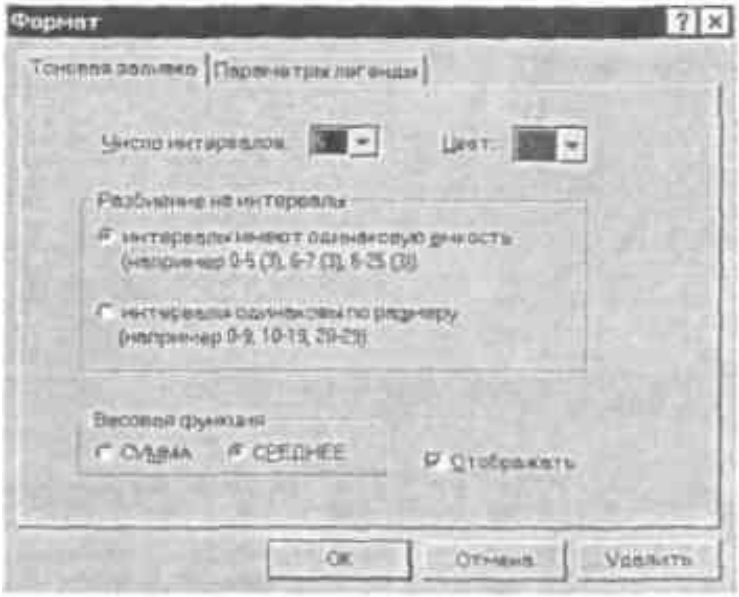
Статистические данные на карте могут отображаться различными способами. Форматы *Тоновая заливка*, *Цветовая заливка* и *Плотность точек* позволяют отобразить один ряд данных, а форматы *Круговая диаграмма* и *Гистограмма* могут отображать несколько рядов данных.

5. На появившейся диалоговой панели выбрать формат *Тоновая заливка*.



В выбранном формате *Тоновая заливка* значения численности населения разбиваются на интервалы, и страны, входящие в каждый из интервалов, закрашиваются своим оттенком цвета. Количество интервалов и способ разбиения на интервалы можно изменить.

6. Щелкнуть по кнопке *Тоновая заливка*.
На появившейся диалоговой модели *Формат* установить количество интервалов (например, 5), способ разбиения по интервалам и базовый цвет.



7. На карте мира каждая из стран будет закрашена одним из пяти оттенков выбранного цвета.
Легенда карты содержит информацию о числовых значениях интервалов и количестве стран, попавших в каждый из интервалов.

Модель «Население стран мира» хранится в папке \calc\ в файле Model.xls на листе *Геоинформационная модель*

CD-ROM

Практические задания для самостоятельного выполнения

CD-ROM

3.25. Построить геоинформационную модель «Численность населения в странах Европы», отображающую статистические данные о численности населения стран Европы.

3.9. Модели логических устройств

3.9.1. Логические схемы сумматора и триггера

При изучении базовых логических устройств компьютера (сумматор, триггер) целесообразно использовать компьютерные модели. Такие модели позволяют визуализировать процесс преобразования логических значений входных сигналов в значения выходных сигналов.

В целях максимального упрощения работы компьютера все многообразие математических операций в процессоре сводится к сложению двоичных чисел. Поэтому главной частью процессора является сумматор, который как раз и обеспечивает такое сложение.

Полусумматор. Вспомним, что при сложении двоичных чисел образуется сумма в данном разряде, при этом возможен перенос в старший разряд. Обозначим слагаемые как A, B, перенос как P и сумму как S. Таблица сложения одноразрядных двоичных чисел с учетом переноса в старший разряд выглядит так, как показано в табл. 3.3.

Таблица 3.3. **Таблица сложения полусумматора**

Слагаемые		Перенос	Сумма
A	*B*	*P*	*S*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Из этой таблицы сразу видно, что перенос можно реализовать с помощью операции логического умножения:

$$P = A \ \& \ B.$$

Получим теперь формулу для вычисления суммы. Значения суммы более всего совпадают с результатом операции логического сложения (кроме случая, когда на входы подаются две единицы, а на выходе должен получиться нуль).

Нужный результат достигается, если результат логического сложения умножить на инвертированный перенос. Таким образом, для определения суммы можно применить следующее логическое выражение:

$$S = (A \vee B) \ \& \ \overline{(A \ \& \ B)}.$$

Построим таблицу истинности для данного логического выражения и убедимся в правильности нашего предположения (табл. 3.4).

Таблица 3.4. **Таблица истинности логической функции $F = (A \vee B)\&(\overline{A\&B})$**

A	B	$A \vee B$	$A \& B$	$\overline{A \& B}$	$(A \vee B)\&(\overline{A \& B})$
0	0	0	0	1	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	1	1	0	0

Теперь на основе полученных логических выражений можно построить из базовых логических элементов схему полусумматора.

По логической формуле переноса легко определить, что для получения переноса необходимо использовать логический элемент «И».

Анализ логической формулы для суммы показывает, что на выходе должен стоять элемент логического умножения «И», который имеет два входа. На один из входов подается результат логического сложения исходных величин $A \vee B$, т. е. на него должен подаваться сигнал с элемента логического сложения «ИЛИ».

На второй вход требуется подать результат инвертированного логического умножения исходных сигналов $A \& B$, т. е. на второй вход подается сигнал с элемента «НЕ», на вход которого поступает сигнал с элемента логического умножения «И» (рис. 3.5).

Данная схема называется полусумматором, так как реализует суммирование одноразрядных двоичных чисел без учета переноса из младшего разряда.

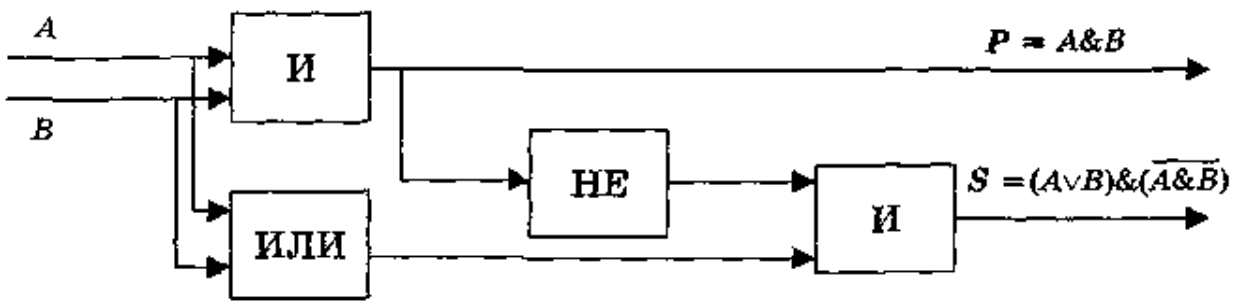


Рис. 3.5. Логическая схема полусумматора

Триггер. Оперативная память компьютера, а также внутренних регистров процессора состоит из ячеек, которые технически реализуются с помощью триггеров. Триггер — это устройство, которое позволяет запоминать, хранить и считывать информацию (каждый триггер может хранить 1 бит информации).

Триггер можно построить из двух логических элементов «ИЛИ» и двух элементов «НЕ» (рис. 3.6).

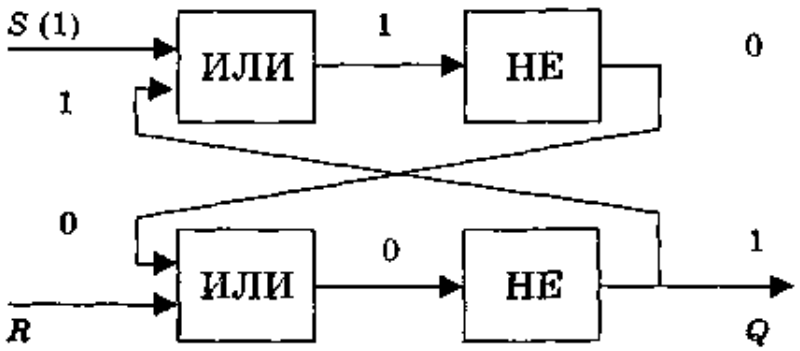


Рис. 3.6. Логическая схема триггера

В обычном состоянии на входы триггера подан сигнал 0, и триггер хранит 0. Для записи 1 на вход S (установочный) подается сигнал 1. Последовательно рассмотрев прохождение сигнала по схеме, мы видим, что триггер переходит в это состояние и будет устойчиво находиться в нем и после того, как сигнал на входе S исчезнет. Триггер запомнил 1, т. е. с выхода триггера Q можно считать 1.

Для того чтобы «сбросить» информацию и подготовиться к приему новой, подается сигнал 1 на вход R (сброс), после чего триггер возвратится к исходному «нулевому» состоянию.

Практические задания для самостоятельного выполнения CD-ROM

3.26. Построить логическую схему полного одноразрядного сумматора двоичных чисел с учетом переноса из младшего разряда.

3.9.2. Модели логических устройств компьютера на языке Visual Basic

В языке программирования Visual Basic основные логические операции могут быть реализованы с помощью логических операторов:

- **And** (логическое умножение);
- **Or** (логическое сложение);
- **Not** (логическое отрицание);
- **Xor** (исключающее **Or**, которое принимает логическое значение **True** тогда и только тогда, когда лишь один из аргументов имеет значение **True**);
- **Eqv** (операция эквалентности, которая принимает логическое значение **True**, когда оба аргумента имеют значение **True** или оба аргумента имеют значение **False**).

Логические операторы могут оперировать с логическими аргументами **True** (логическая единица) и **False** (логический нуль), а также с логическими переменными типа **Boolean**.

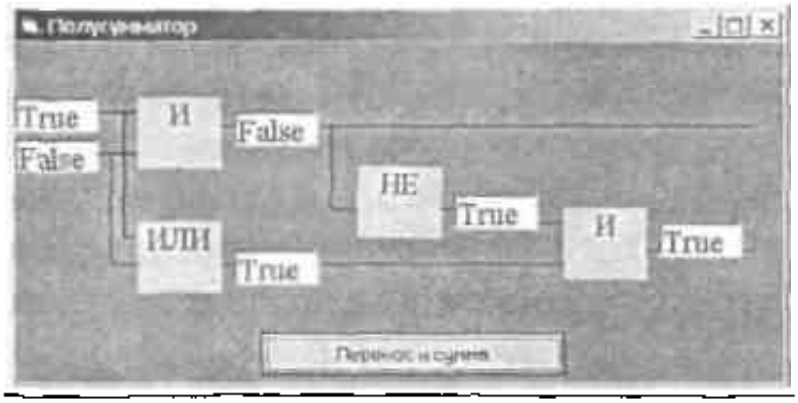
Построим компьютерную модель полусумматора с использованием языка программирования Visual Basic.

Проект «Модель полусумматора» на языке Visual Basic

1. Поместить на форму:
   - кнопку cmd1 для создания событийной процедуры;
   - четыре метки lblAnd1, lblAnd2, lblOr и lblNot для изображения базовых логических элементов;
   - два текстовых поля txtA и txtB для ввода логических значений на входы сумматора;
   - четыре текстовых поля: txtOr и txtNot — для вывода промежуточных логических значений, txtS — для вывода итоговых значений суммы, txtP — для вывода итогового значения переноса.
2. Создать событийную процедуру, реализующую определение логических значений на выходе каждого базового логического элемента и их вывод в текстовые поля:

```
Dim blnA, blnB, blnP, blnS As Boolean
Sub cmd1_Click()
blnA = txtA.Text
blnB = txtB.Text
blnP = blnA And blnB
blnS = (blnA Or blnB) And Not (blnA And blnB)
txtP.Text = blnP
txtOtr.Text = Not blnP
txtOr.Text = blnA Or blnB
txtS.Text = blnS
End Sub
```

3. Запустить проект, ввести логические значения аргументов и щелкнуть по кнопке *Перенос и сумма*.
   В текстовые поля будут выведены логические значения *на выходах логических элементов*.



Проект «Модель полусумматора» хранится в папке \VB\Summ\ CD-ROM

Проект «Модель триггера» на языке Visual Basic

1. Поместить на форму шесть текстовых полей:
   - txtSet и txtReset — для ввода начальных логических значений на входы триггера;
   - txtOr1Out и txtOr2Out — для визуального контроля промежуточных логических значений;
   - txtQ2 и txtQ1 — для вывода состояния выходов триггера.
2. Поместить на форму восемь меток:
   - lblOr1, lblOr2, lblNot1, lblNot2 — для обозначения составляющих триггер логических элементов;
   - lblSet, lblReset, lblQ1, lblQ2 — для обозначения входов и выходов триггера.
3. Определить логические переменные:

```
Dim blnS, blnR, BlnOr1Out, BlnOr2Out,
blnNot1Out, blnNot2Out As Boolean
```

4. Поместить на форму кнопку cmdSet и создать для нее событийную процедуру установки значения триггера:

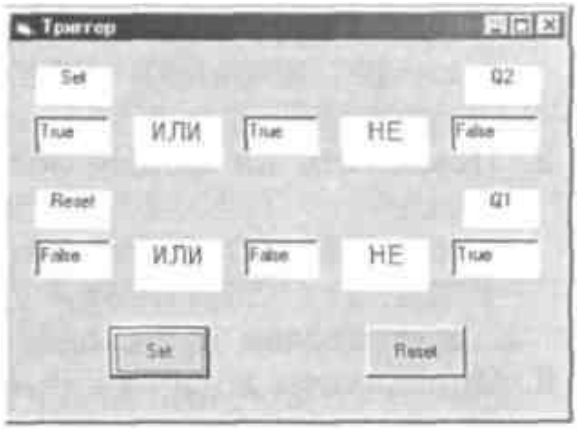
```
Private Sub cmdSet_Click()
txtSet.Text = True
txtReset.Text = False
blnS = txtSet.Text
blnR = txtReset.Text
BlnOr1Out = blnS Or blnNot2Out
blnNot1Out = Not BlnOr1Out
BlnOr2Out = blnNot1Out Or blnR
blnNot2Out = Not BlnOr2Out
```

```
txtOr1Out = BlnOr1Out
txtOr2Out = BlnOr2Out
txtQ2 = blnNot1Out
txtQ1 = blnNot2Out
End Sub
```

5. Поместить на форму кнопку cmdReset и создать для нее событийную процедуру сброса значения триггера:

```
Private Sub cmdReset_Click()
txtSet.Text = False
txtReset.Text = True
blnS = txtSet.Text
blnR = txtReset.Text
BlnOr1Out = blnS Or blnNot2Out
blnNot1Out = Not BlnOr1Out
BlnOr2Out = blnNot1Out Or blnR
blnNot2Out = Not BlnOr2Out
txtOr1Out = BlnOr1Out
txtOr2Out = BlnOr2Out
txtQ2 = blnNot1Out
txtQ1 = blnNot2Out
End Sub
```

6. Запустить проект.
Установить триггер, ввести в поле Set значение **True**, проследить за установкой значений на элементах триггера.
Сбросить триггер, ввести в поле Reset значение **True**.



Проект «Модель триггера» хранится в папке \VB\Trigger CD-ROM

Практические задания для самостоятельного выполнения

CD-ROM

3.27. На языке программирования Visual Basic разработать проект «Сумматор», строящий таблицу истинности логических выражений, реализующих перенос и сумму в полном одноразрядном сумматоре двоичных чисел с учетом переноса из младшего разряда.

3.9.3. Модели логических устройств компьютера на языке Delphi

В языке программирования Delphi основные логические операции могут быть реализованы с помощью логических операторов:

- **And** (логическое умножение);
- **Or** (логическое сложение);
- **Not** (логическое отрицание);
- **Xor** (исключающее **Or**, которое принимает логическое значение **True**, тогда и только тогда, когда лишь один из аргументов имеет значение **True**).

Логические операторы могут оперировать с логическими аргументами **True** (логическая единица) и **False** (логический нуль), а также с логическими переменными типа Boolean.

Построим компьютерную модель полусумматора с использованием языка программирования Delphi.

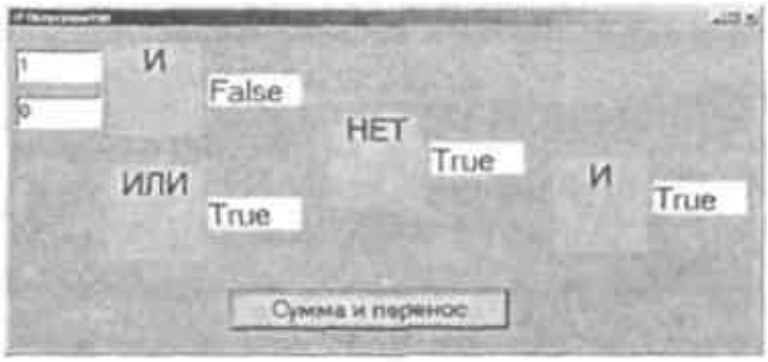
Проект «Модель полусумматора» на языке Delphi

1. Поместить на форму:
   - кнопку Button1 для создания событийной процедуры;
   - четыре метки Label1, Label2, Label3 и Label4 для изображения базовых логических элементов;
   - два текстовых поля EditA и EditB для ввода логических значений на входы сумматора;
   - четыре метки: LabelOr и LabelNot — для вывода промежуточных логических значений, LabelS — для вывода итогового значения суммы, LabelP — для вывода итогового значения переноса.
2. Создать событийную процедуру, реализующую:
   - ввод логических значений на входы и преобразование их логических значений в строковые с использованием функции BoolToStr();
   - определение логических значений на выходе каждого базового логического элемента;
   - вывод полученных логических значений на метки с использованием функции преобразования логического типа данных в строковые BoolToStr():

```
var
A : Boolean;
B : Boolean;
P : Boolean;
S : Boolean;
```

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
A := StrToBool(EditA.Text);
B := StrToBool(EditB.Text);
P := (A And B);
S := (A Or B) And (Not (A And B));
LabelP.Caption := BoolToStr(P,True);
LabelNo.Caption := BoolToStr(Not(P),True);
LabelOr.Caption := BoolToStr((A Or B),True);
LabelS.Caption := BoolToStr(S,True);
end;
```

3. Запустить проект, ввести логические значения аргументов и щелкнуть по кнопке *Сумма и Перенос*.
В текстовые поля будут выведены логические значения на выходах логических элементов.



Проект «Модель полусумматора» хранится в папке \Delphi\Summ CD-ROM

Проект «Модель триггера» на языке Delphi

1. Поместить на форму шесть текстовых полей:
   - EditSet и EditReset — для ввода начальных логических значений на входы триггера;
   - EditOr1Out и EditOr2Out — для визуального контроля промежуточных логических значений;
   - EditQ2 и EditQ1 — для вывода состояния выходов триггера.
2. Поместить на форму метки:
   - четыре для обозначения составляющих триггер логических элементов;
   - четыре для обозначения входов и выходов триггера.
3. Определить логические переменные:

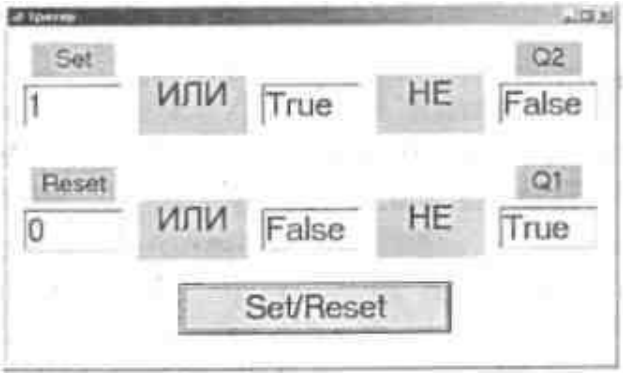
```
var
S: boolean;
R: boolean;
```

```
Or1Out  : boolean;
Not1Out : boolean;
Or2Out  : boolean;
Not2Out : boolean;
```

4. Поместить на форму кнопку Button1 и создать для нее событийную процедуру установки значения триггера:

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
S := StrToBool(EditSet.Text);
R := StrToBool(EditReset.Text);
Or1Out := S Or Not2Out;
Not1Out := Not Or1Out;
Or2Out := Not1Out Or R;
Not2Out := Not Or2Out;
EditOr1Out.Text := BoolToStr(Or1Out,True);
EditOr2Out.Text := BoolToStr(Or2Out,True);
EditQ2.Text := BoolToStr(Not1Out,True);
EditQ1.Text := BoolToStr(Not2Out,True);
end;
```

5. Запустить проект.
Установить триггер, ввести в поле *Set* значение 1, а в поле *Reset* значение 0. Щелкнуть по кнопке *Set/Reset*. Проследить за установкой значений на элементах триггера.
Сбросить триггер.



Проект «Модель триггера» хранится в папке \Delphi\Trigger CD-ROM

Практические задания для самостоятельного выполнения

CD-ROM

3.28. На языке программирования Delphi разработать проект «Сумматор», строящий таблицу истинности логических выражений, реализующих перенос и сумму в полном одноразрядном сумматоре двоичных чисел с учетом переноса из младшего разряда.

3.9.4. Модели логических устройств компьютера в электронных таблицах

В электронных таблицах имеются логические функции, реализующие базовые логические операции. Аргументами и значениями логических функций являются логические значения ИСТИНА или ЛОЖЬ.

Логические значения могут быть получены также как результат вычисления логических выражений. Например, для логического выражения 10>5 результатом будет логическое значение ИСТИНА, а для логического выражения A1<A2 (где в ячейке A1 хранится число 10, а в ячейке A2 — число 5) — значение ЛОЖЬ.

Логическая функция «И» имеет в качестве аргументов логические значения, которые могут быть истинными или ложными, и задается формулой =И(лог_знач1; лог_знач2;...). Принимает значение ИСТИНА только тогда, когда все аргументы имеют значение ИСТИНА.

Например, значение функции =И(10>5;10<5) — ЛОЖЬ.

Логическая функция «ИЛИ» имеет в качестве аргументов логические значения и задается формулой =ИЛИ(лог_знач1; лог_знач2; ...). Принимает значение ИСТИНА, если хотя бы один из аргументов имеет значение ИСТИНА.

Например, значение функции =ИЛИ(10>5;10<5) — ИСТИНА.

Логическая функция «НЕ» меняет на противоположное значение своего аргумента и задается формулой =НЕ(лог_знач). Принимает значение ИСТИНА, если аргумент имеет значение ЛОЖЬ, и наоборот.

Например, значение функции =НЕ(10>5) — ЛОЖЬ.

Построим с помощью электронных таблиц таблицу истинности операции логического умножения, используя логическую функцию «И».

Проект «Таблица истинности операции логического умножения» в электронных таблицах

1. В пары ячеек (A1, B1), (A2, B2), (A3, B3), (A4, B4) ввести пары значений аргументов логической операции (ЛОЖЬ, ЛОЖЬ), (ИСТИНА, ЛОЖЬ), (ЛОЖЬ, ИСТИНА) и (ИСТИНА, ЛОЖЬ).

2. В ячейку C1 ввести формулу логической функции «И»: =И(A1;B1).
3. Скопировать формулу в ячейки C2, C3 и C4.
4. Значением этой функции в трех случаях является ЛОЖЬ *и только в последнем* — ИСТИНА.
   Мы получили таблицу истинности операции логического умножения.

	A	B	C
1	ЛОЖЬ	ЛОЖЬ	ЛОЖЬ
2	ИСТИНА	ЛОЖЬ	ЛОЖЬ
3	ЛОЖЬ	ИСТИНА	ЛОЖЬ
4	ИСТИНА	ИСТИНА	ИСТИНА

Таблица истинности хранится в папке \calc\ в файле Model.xls на листе *Логические функции* CD-ROM

Проект «Полусумматор» в электронных таблицах

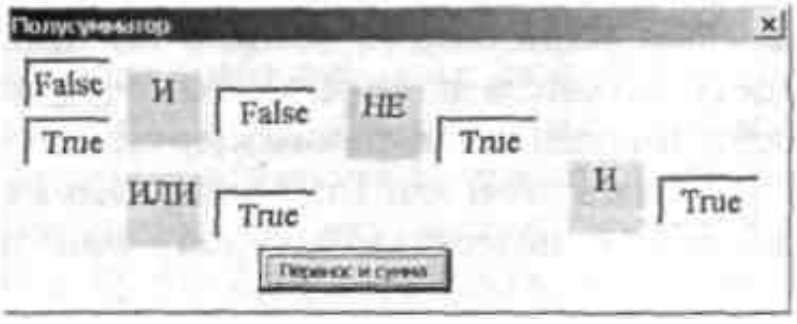
1. *На листе Полусумматор предусмотреть* ввод логических значений аргументов в ячейки B1 и B2.
   В ячейку B3 ввести формулу =И(B1;B2).
   В ячейку B4 ввести формулу =НЕ(B3).
   В ячейку B5 ввести формулу =ИЛИ(B1;B2).
   В ячейку B6 ввести формулу =И(B5;B4).

	A	B
1	A=	ЛОЖЬ
2	B=	ИСТИНА
3	P=	=И(B1;B2)
4	Not P =	=НЕ(B3)
5	Or =	=ИЛИ(B1;B2)
6	S=	=И(B5;B4)

2. Добавить форму и поместить на нее четыре метки и шесть текстовых полей. Создать событийную процедуру:

```
Private Sub cmd1_Click()
Cells(1, 2) = txtA.Text
Cells(2, 2) = txtB.Text
txtP.Text = Cells(3, 2)
txtNot.Text = Cells(4, 2)
txtOr.Text = Cells(5, 2)
txtS.Text = Cells(6, 2)
End Sub
```

3. Запустить проект, ввести значения аргументов и щелкнуть по кнопке *Перенос и сумма*.



Проект хранится в папке \calc\
в файле Model.xls на листе *Полусумматор* CD-ROM

Практические задания для самостоятельного выполнения CD-ROM

3.29. В электронных таблицах получить таблицы истинности операций логического сложения и логического отрицания.

3.10. Информационные модели управления объектами

3.10.1. Информационные модели систем управления

В процессе функционирования сложных систем (биологических, технических и т. д.), входящие в них объекты постоянно обмениваются информацией. Так, для поддержания своей жизнедеятельности любой живой организм постоянно получает информацию из внешнего мира с помощью органов чувств, обрабатывает ее и управляет своим поведением (например, перемещаясь в пространстве, избегает опасности).

В процессе управления полетом самолета в режиме автопилота бортовой компьютер получает информацию от датчиков (скорости, высоты и т. д.), обрабатывает ее и передает команды на исполнительные механизмы, изменяющие режим полета (закрылки, клапаны, регулирующие работу двигателей, и т. д.).

В любом процессе управления всегда происходит взаимодействие двух объектов — **управляющего** и **управляемого**, которые соединены каналами ***прямой*** и ***обратной связи***. По каналу прямой связи передаются управляющие сигналы, а по каналу обратной связи — информация о состоянии управляемого объекта.

Разомкнутые системы управления. Если в процессе управления не учитывается состояние управляемого объекта и обеспечивается управление только по прямому каналу (от управляющего объекта к управляемому), то такие системы управления называются разомкнутыми. Информационную модель разомкнутой системы управления можно наглядно представить с помощью схемы, изображенной на рис. 3.7.

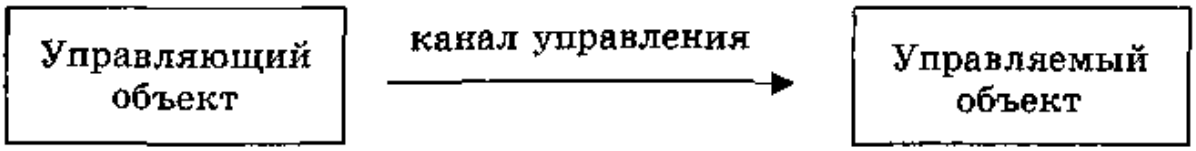


Рис. 3.7. Разомкнутая система управления

Замкнутые системы управления с обратной связью. В замкнутых системах управления управляющий объект по прямому каналу управления производит необходимые действия над объектом управления, а по каналу обратной связи получает информацию о реальных параметрах объекта управления. Это позволяет осуществлять управление с гораздо большей точностью.

Информационную модель замкнутой системы управления можно наглядно представить с помощью схемы, изображенной на рис. 3.8.

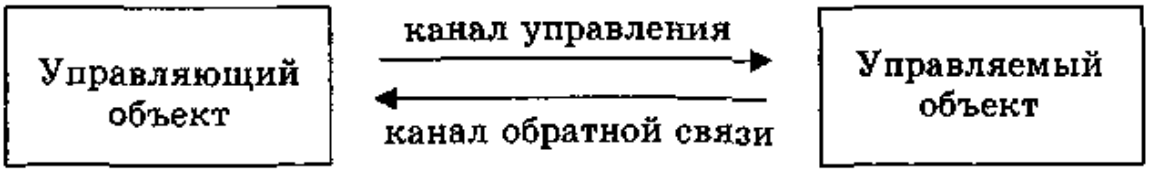


Рис. 3.8. Замкнутая система с обратной связью

Вопросы для размышления

1. В чем состоит различие разомкнутых и замкнутых систем управления? Приведите примеры.

3.10.2. Модели систем управления на языке Visual Basic

Разомкнутые системы управления. Для демонстрации принципа работы разомкнутых систем управления разработаем компьютерную модель на языке программирования Visual Basic. Пусть управляемым объектом будет точка, которую управляющий объект (пользователь) должен переместить в центр мишени (круга). Прямое управление положением точки будем производить путем нажатия на кнопки, которые перемещают объект вверх, вниз, влево и вправо. Обратная связь будет отсутствовать, так как текущее поло-

жение управляемого объекта (точки) будет невидимым (точка будет рисоваться цветом фона).

Проект «Модель разомкнутой системы управления» на языке Visual Basic

1. Поместить на форму:
   - графическое поле pic1, по которому будет перемещаться точка;
   - кнопку cmd1 для вывода мишени и первоначального положения управляемого объекта (точки);
   - четыре кнопки cmdUp, cmdD, cmdL и cmdR для управления движением точки;
   - кнопку cmd2 для вывода положения управляемого объекта (точки).

2. Событийная процедура вывода первоначального положения управляемого объекта (точки) должна включать задание масштаба и случайную генерацию координат точки:

```
Dim bytX1, bytY1, bytX2, bytY2 As Byte
Private Sub cmdP_Click()
pic1.Scale (0, 20)-(20, 0)
Randomize
bytX1 = Int(Rnd * 20)
bytY1 = Int(Rnd * 20)
pic1.PSet (bytX1, bytY1), vbRed
End Sub
```

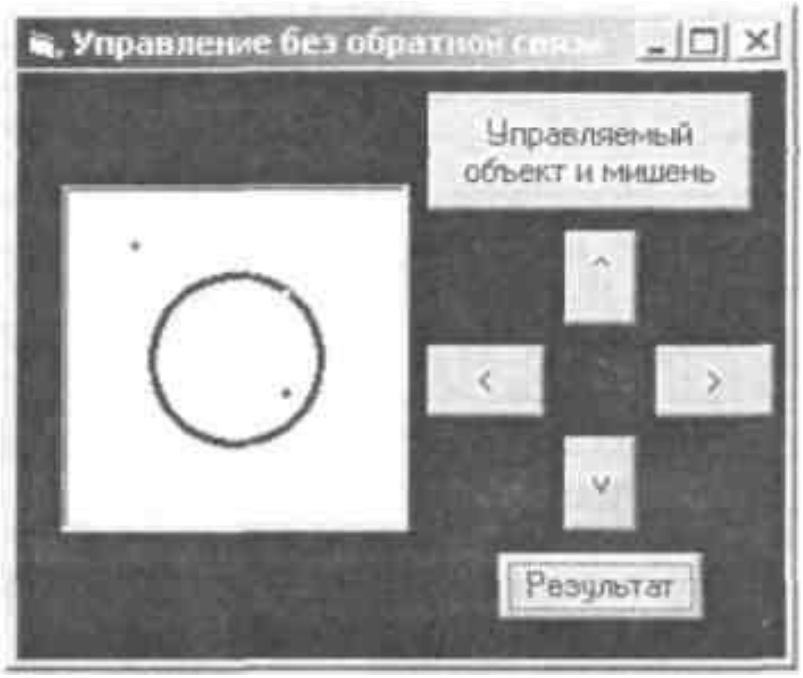
3. Четыре событийные процедуры перемещения точки должны обеспечивать изменение координат точки. Для перемещения влево предназначена событийная процедура:

```
Private Sub cmdL_Click()
pic1.Scale (0, 20)-(20, 0)
bytX1 = bytX1 - 1
pic1.PSet (bytX1, bytY1), vbWhite
End Sub
```

4. Событийная процедура вывода конечного положения управляемого объекта (точки):

```
Private Sub cmd2_Click()
pic1.Scale (0, 20)-(20, 0)
pic1.Circle (10, 10), 5
pic1.PSet (bytX1, bytY1), vbBlack
End Sub
```

5. Запустить проект.
   Щелкнуть по кнопке *Управляемый объект и мишень*. Переместить управляемый объект (точку) в центр мишени щелчками по кнопкам со стрелками.
   Щелкнуть по кнопке *Результат*. Скорее всего, управляемый объект (точка) не попадет в центр мишени.



Проект «Модель разомкнутой системы управления» хранится в папке \VB\Upr1 CD-ROM

Замкнутые системы управления с обратной связью. Для демонстрации принципа работы замкнутых систем управления разработаем компьютерную модель. Для осуществления обратной связи сделаем текущие положения управляемого объекта (точки) видимыми (точку будем рисовать красным цветом), а также будем выводить текущие значения координат точки в текстовые поля.

Проект «Модель замкнутой системы управления» на языке Visual Basic

Усовершенствуем проект «Модель разомкнутой системы управления».

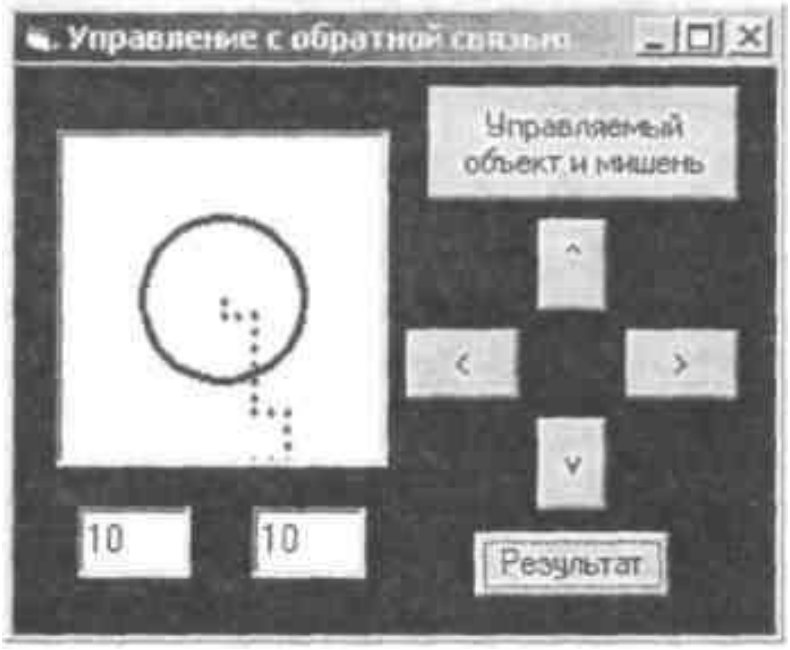
1. Поместить на форму два текстовых поля `txtX` и `txtY` для вывода текущих координат точки.
2. В коды процедур добавить строки:

```
txtX.Text = bytX1
txtY.Text = bytY1
```

3. В четырех событийных процедурах перемещения управляемого объекта сделать текущие положения точки видимыми (изменить цвет точки на красный). Для перемещения влево служит событийная процедура:

```
Private Sub cmdL_Click()
pic1.Scale (0, 20)-(20, 0)
bytX1 = bytX1 - 1
pic1.PSet (bytX1, bytY1), vbRed
txtX.Text = bytX1
txtY.Text = bytY1
End Sub
```

4. Запустить проект и осуществить попадание управляемого объекта (точки) в мишень, координаты центра которой (10, 10). Легко убедиться, что использование обратной связи обеспечивает гарантированное попадание управляемого объекта (точки) в мишень.



Проект «Модель замкнутой системы управления» хранится в папке \VB\Upr2 **CD-ROM**

Практические задания для самостоятельного выполнения **CD-ROM**

3.30. На языке Visual Basic создать проект «Модель системы управления с автоматической обратной связью», в которой реализуется пошаговое автоматическое попадание в мишень.

3.10.3. Модели систем управления на языке Delphi

Разомкнутые системы управления. Для демонстрации принципа работы разомкнутых систем управления разработаем компьютерную модель на языке программирования Delphi.

Пусть управляемым объектом будет круг, который управляющий объект (пользователь) должен переместить в центр мишени (окружности). Прямое управление положением круга будем производить путем нажатия на кнопки, которые перемещают объект (круг) вверх, вниз, влево и вправо. Обратная связь будет отсутствовать, так как текущее положение управляемого объекта (круга) будет невидимым (круг рисоваться не будет).

Проект «Модель разомкнутой системы управления» на языке Delphi

1. Поместить на форму:
   - графическое поле `Image1`, по которому будет перемещаться точка;
   - кнопку `Button1` для вывода первоначального положения управляемого объекта (закрашенного круга) и мишени (незакрашенного круга);
   - четыре кнопки `ButtonUp`, `ButtonD`, `ButtonL` и `ButtonR` для управления движением точки;
   - кнопку `Button2` для вывода конечного положения управляемого объекта (закрашенного круга).

2. Событийная процедура вывода первоначального положения управляемого объекта (закрашенного круга) должна обеспечивать случайную генерацию координат центра закрашенного круга, выбор цвета и типа закрашивания, а также рисовать мишень в центре графического поля:

```
var
X1:integer;
Y1:integer;
procedure TForm1.Button1Click(Sender: Tobject);
begin
Randomize;
X1 := Random(200);
Y1 := Random(200);
```

```
Form1.Image1.Canvas.Brush.Color := clRed;
Form1.Image1.Canvas.Brush.Style := bsSolid;
Form1.Image1.Canvas.Ellipse (X1-3,Y1-3,X1+3,Y1+3);
Form1.Image1.Canvas.Brush.Color := clBlack;
Form1.Image1.Canvas.Brush.Style := bsClear;
Form1.Image1.Canvas.Ellipse(80,80,120,120);
end;
```

3. Четыре событийные процедуры перемещения круга должны обеспечивать изменение координат центра круга:

```
procedure TForm1.ButtonLClick(Sender: TObject);
begin
X1 := X1 - 1;
end;

procedure TForm1.ButtonRClick(Sender: TObject);
begin
X1 := X1 + 1;
end;


procedure TForm1.ButtonUpClick(Sender: TObject);
begin
Y1 := Y1 - 1;
end;


procedure TForm1.ButtonDClick(Sender: TObject);
begin
Y1 := Y1 + 1;
end;
```

4. Событийная процедура вывода конечного положения управляемого объекта (круга):

```
procedure TForm1.Button2Click(Sender: TObject);
begin
Form1.Image1.Canvas.Brush.Color := clBlack;
Form1.Image1.Canvas.Brush.Style := bsSolid;
Form1.Image1.Canvas.Ellipse(X1-3,Y1-3,X1+3,Y1+3);
end;
```

5. Щелкнуть по кнопке *Управляемый объект и мишень*. Переместить управляемый объект (точку) в центр мишени щелчками по кнопкам со стрелками.

Щелкнуть по кнопке *Результат*. Скорее всего, управляемый объект (круг) не попадет в центр мишени.

Проект «Модель разомкнутой системы управления» хранится в папке \Delphi\Upr1\ CD-ROM

Замкнутые системы управления с обратной связью. Для демонстрации принципа работы замкнутых систем управления разработаем компьютерную модель. Для осуществления обратной связи текущие положения управляемого объекта (круга) сделаем видимыми (круг будем рисовать красным цветом), а также будем выводить текущие значения координат центра круга в текстовые поля.

Проект «Модель замкнутой системы управления» на языке Delphi

Усовершенствуем проект «Модель разомкнутой системы управления».

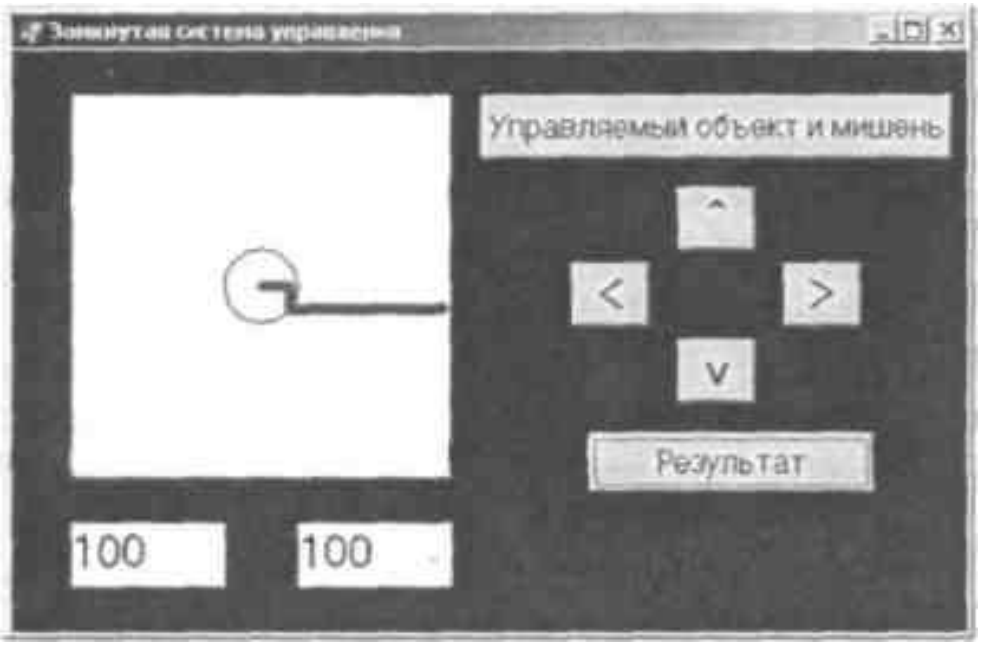
1. Поместить на форму две метки `LabelX` и `LabelY` для вывода текущих координат центра круга.
2. В коды процедур добавить строки:

```
LabelX.Caption := IntToStr(X1);
LabelY.Caption := IntToStr(Y1);
```

3. Четыре событийные процедуры перемещения круга должны обеспечивать изменение координат центра круга, а также рисовать его текущие положения. Для перемещения влево событийная процедура примет вид:

```
procedure TForm1.ButtonLClick(Sender: TObject);
begin
X1 := X1 - 1;
Form1.Image1.Canvas.Pen.Color := clRed;
Form1.Image1.Canvas.Brush.Color := clRed;
Form1.Image1.Canvas.Brush.Style := bsSolid;
Form1.Image1.Canvas.Ellipse(X1-3,Y1-3,X1+3,Y1+3);
LabelX.Caption := IntToStr(X1);
LabelY.Caption := IntToStr(Y1);
end;
```

4. Запустить проект и осуществить попадание в мишень (окружность), имеющую координаты (100, 100) и размещенную в графическом поле с размерами (200, 200). Легко убедиться, что использование обратной связи обеспечивает гарантированное попадание точки в мишень.



Проект «Модель замкнутой системы управления» хранится в папке \Delphi\Upr2 **CD-ROM**

Практические задания для самостоятельного выполнения CD-ROM

3.31. На языке Delphi создать проект «Модель системы управления с автоматической обратной связью».

Список рекомендуемой литературы

1. *Волченков Н.Г.* Программирование на Visual Basic 6: Учебное пособие. Часть 1. М.: ИНФРА-М, 2000.
2. *Волченков Н.Г.* Программирование на Visual Basic 6: Учебное пособие. Часть 2. М.: ИНФРА-М, 2000.
3. *Волченков Н.Г.* Программирование на Visual Basic 6: Учебное пособие. Часть 3. М.: ИНФРА-М, 2000.
4. *Культин Н.Б.* Delphi 6. Программирование на Object Pascal. СПб.: БХВ-Петербург, 2001.

Учебное издание

Угринович Николай Дмитриевич

ИССЛЕДОВАНИЕ ИНФОРМАЦИОННЫХ МОДЕЛЕЙ
Элективный курс

Редактор *О. Полежаева*
Художник *Ф. Инфанте*
Компьютерная верстка *В. Носенко, С. Яшкова*

Подписано в печать 05.10.04. Формат 60×90 $^1/_{16}$.
Бумага офсетная. Гарнитура Школьная. Печать офсетная.
Усл. печ. л. 11,5. Тираж 5000 экз. Заказ 2728

Издательство «БИНОМ. Лаборатория знаний»
Телефон: (095)955-0398, E-mail: lbz@aha.ru

Отпечатано с готовых диапозитивов
в полиграфической фирме «Полиграфист»,
160001, г. Вологда, ул. Челюскинцев, 3.

Победитель конкурса по созданию учебной литературы
нового поколения для средней школы,
проводимого НФПК - Национальным фондом подготовки кадров
и Министерством образования Российской Федерации

Это учебное пособие поможет вам научиться

- работать в системах объектно-ориентированного программирования Visual Basic и Delphi
- создавать и исследовать информационные модели из предметных областей
  - физики
  - математики
  - химии
  - биологии
  - географии
  - экономики
  - информатики
- проектной деятельности

Для информационно-технологического,
физико-математического, естественно-научного профилей

ISBN 5-94774-154-7