Никита Культин

bhv®

Delphi
в задачах и примерах

Готовые решения и тексты программ

Использование базовых компонентов

Работа с графикой, звуком и базами данных

Программирование игр

Справочник по компонентам и функциям

+ CD-ROM

Никита Культин

Delphi
в задачах и примерах

Санкт-Петербург
«БХВ-Петербург»
2003

УДК 681.3.068+800.92Delphi
ББК 32.973.26-018.1
К90

Культин Н. Б.

К90 Delphi в задачах и примерах. — СПб.: БХВ-Петербург, 2003. — 288 с.: ил.

ISBN 5-94157-353-7

Книга представляет собой сборник программ и задач для самостоятельного решения в среде разработки Delphi. Примеры различной степени сложности — от простейших до приложений работы с графикой, звуком и базами данных — демонстрируют возможности среды разработки Delphi, назначение основных компонентов. Книга также содержит краткий справочник наиболее часто используемых компонентов и функций. На прилагаемом компакт-диске находятся тексты программ.

Для начинающих программистов

УДК 681.3.068+800.92Delphi
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Анатолий Адаменко*
Зав. редакцией	*Григорий Добин*
Редактор	*Анатолий Хрипов*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 20.06.03.
Формат 60×90 1/16. Печать офсетная. Усл. печ. л. 18.
Тираж 5000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-353-7

Содержание

Предисловие

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

Бурное развитие вычислительной техники, потребность в эффективных средствах разработки программного обеспечения привели к появлению систем программирования, ориентированных на так называемую "быструю разработку". В основе идеологии систем быстрой разработки (RAD-систем, Rapid Application Development — среда быстрой разработки приложений) лежат технологии визуального проектирования и событийного объектно-ориентированного программирования, суть которых заключается в том, что среда разработки берет на себя большую часть рутины, оставляя программисту работу по конструированию диалоговых окон и созданию функций обработки событий. Производительность программиста при использовании RAD-систем — фантастическая!

Среди RAD-систем особо выделяется среда Borland Delphi, которая позволяет создавать различные программы: от простейших однооконных приложений до программ управления распределенными базами данных. В качестве языка программирования в среде Borland Delphi используется язык Delphi (Delphi language), являющийся прямым потомком хорошо известного всем программистам языка Pascal.

Чтобы научиться программировать, надо программировать — писать программы, решать конкретные задачи. Для этого необходимо изучить язык программирования и среду разработки. Освоить язык программирования Delphi не очень сложно. Труднее

изучить среду программирования, точнее научиться использовать компоненты. И здесь хорошим подспорьем могут быть программы, которые демонстрируют назначение компонентов и особенности их применения.

В книге, которую вы держите в руках, собраны разнообразные примеры, которые не только демонстрируют возможности среды разработки Delphi, но и знакомят с принципами работы с графикой, звуком, базами данных. Следует обратить внимание, что большинство примеров не являются учебными в чистом смысле, это — вполне работоспособные программы.

Книга состоит из двух частей и приложений.

Первая часть содержит примеры и задачи для самостоятельного решения. Примеры представлены в виде краткого описания, сформулированного в форме задания для самостоятельного решения, диалоговых окон и хорошо документированных текстов программ. Для простых задач рассмотрены только функции обработки событий. Текст остальных программ приведен полностью.

Вторая часть книги — это краткий справочник по языку программирования Delphi. В нем можно найти описание свойств компонентов, использованных в приведенных примерах.

Научиться программировать можно, только программируя, решая конкретные задачи. При этом достигнутые в программировании успехи в значительной степени зависят от опыта. Поэтому, чтобы получить максимальную пользу от книги, вы должны работать с ней активно. Изучайте листинги, старайтесь понять, как работают программы. Не бойтесь экспериментировать — вносите изменения в программы.

Если что-то не понятно, обратитесь к справочнику *(часть 2)*, к справочной системе Delphi или к литературе, например: **Культин Н. Б. Основы программирования в Delphi 7. — СПб.: БХВ-Петербург, 2003**. В ней, помимо описания языка программирования и среды разработки Delphi, компонентов, процессов создания и отладки программ, вы найдете ответы на многие вопросы, в том числе: как при помощи Microsoft Help Workshop сформировать файл справки или, используя Install-Shield Express, создать установочный CD-ROM.

Часть 1

Примеры и задачи

Базовые компоненты

В этом разделе приведены простые примеры и задачи, основное назначение которых — научить работать с базовыми компонентами.

Общие замечания

Приступая к решению задач этого раздела, необходимо вспомнить:

- Процесс создания программы в Delphi состоит из двух шагов: сначала нужно создать форму программы (диалоговое окно), затем — написать процедуры обработки *событий*. Форма *приложения* (так принято называть прикладные программы, работающие в Windows) создается путем добавления в форму *компонентов* и последующей их настройки.
- В форме практически любого приложения есть компоненты, которые обеспечивают интерфейс (взаимодействие) между программой и пользователем. Такие компоненты называют *базовыми*. К базовым компонентам можно отнести:

  Label — поле вывода текста;

  Edit — поле ввода/редактирования текста;

  Button — командную кнопку;

  CheckBox — независимую кнопку выбора;

  RadioButton — зависимую кнопку выбора;

  ListBox — список выбора;

  ComboBox — комбинированный список выбора.

- Вид компонента, его размер и поведение определяются значениями *свойств* (характеристик) компонента (описание свойств базовых компонентов можно найти в справочнике, во второй части книги).
- Основную работу в программе выполняют процедуры обработки *событий* (описание основных событий можно найти в справочнике, во второй части книги).
- Исходную информацию программа может получить из полей ввода/редактирования (компонент `Edit`), списка выбора (компонент `ListBox`) или комбинированного списка (компонент `ComboBox`). Для ввода значений логического типа можно использовать компоненты `CheckBox` и `RadioButton`.
- Результат программа может вывести в поле вывода текста (компонент `Label`) или в окно сообщения (функция `MessageDlg`).
- Для преобразования текста, например находящегося в поле ввода/редактирования, в целое число нужно использовать функцию `StrToInt`, а в дробное — функцию `StrToFloat`. Для преобразования целого, например значения переменной, в строку нужно использовать функцию `IntToStr`, а для преобразования дробного — функцию `FloatToStr` или `FloatToStrF`.

1. Написать программу, которая пересчитывает скорость ветра из "метров в секунду" в "километров в час". Рекомендуемый вид формы приведен на рис. 1.1.

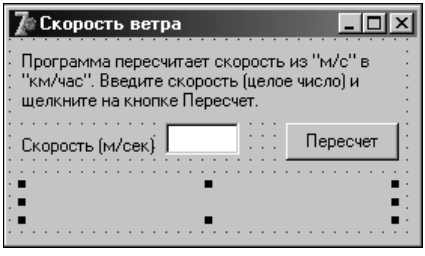


Рис. 1.1. Форма программы **Скорость ветра**

```
// щелчок на кнопке Пересчет
procedure TForm1.Button1Click(Sender: TObject);
var
    ms:  integer; // скорость м/с
    kmh: real;    // скорость км/час
begin
    ms  := StrToInt(Edit1.Text); // ввести исходные данные
    kmh := ms * 3.6;             // пересчитать
    // вывести результат
    Label3.Caption :=
       IntToStr(ms) + ' м/с — это ' +
       FloatToStr(kmh) + ' км/час'
end;
```

2. Написать программу, которая пересчитывает скорость ветра из "метров в секунду" в "километров в час". Рекомендуемый вид формы приведен на рис. 1.1. Программа должна быть спроектирована таким образом, чтобы пользователь мог ввести в поле **Скорость** только целое положительное число.

```
// щелчок на кнопке Пересчет
procedure TForm1.Button1Click(Sender: TObject);
var
    ms:  integer; // скорость м/с
    kmh: real;    // скорость км/час
begin
    // Если поле Edit1 пустое, то при выполнении функции
    // StrToInt возникает ошибка.
    // Проверим, ввел ли пользователь скорость в поле Edit1

    if Length(Edit1.Text) = 0 then
    begin
        ShowMessage('Надо ввести скорость');
        exit; // завершить обработку события
    end;

    ms  := StrToInt(Edit1.Text); // ввести исходные данные
    kmh := ms * 3.6;             // пересчитать
    // вывести результат
    Label3.Caption :=
       IntToStr(ms) + ' м/с — это ' + FloatToStr(kmh) + ' км/час'
end;
```

```
// нажатие клавиши в поле Edit1
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    // В поле Скорость (Edit1) можно ввести только
    // цифры. Процедура проверяет, является ли символ
    // допустимым. Если нет, то она заменяет
    // введенный символ нулевым. В результате символ
    // в поле редактирования не отображается.

    // Key — символ, соответствующий нажатой клавише
    if not ((Key >= '0') and (Key <='9') or (Key = #8))
        then Key := Chr(0);

end;
```

3. Написать программу, которая пересчитывает скорость ветра из "метров в секунду" в "километров в час". Рекомендуемый вид формы приведен на рис. 1.1. Программа должна быть спроектирована таким образом, чтобы пользователь мог ввести в поле **Скорость** только целое положительное число. Вычисление должно выполняться как в результате щелчка на кнопке **Пересчет**, так и при нажатии клавиши <Enter> после ввода последней цифры в поле **Скорость**.

```
// пересчитывает скорость из м/сек в км/час
procedure WindSpeed;
var
    ms:  integer; // скорость м/с
    kmh: real;    // скорость км/час
begin
    // Если поле Edit1 пустое, то при выполнении
    // функции StrToInt возникает ошибка.
    // Проверим, ввел ли пользователь скорость в поле Edit1

    if Length(Form1.Edit1.Text) = 0 then
    begin
        ShowMessage('Надо ввести скорость');
        exit; // завершить обработку события
    end;

    ms  := StrToInt(Form1.Edit1.Text); // ввести исходные данные
    kmh := ms * 3.6;                   // пересчитать
```

```
    // вывести результат
    Form1.Label3.Caption :=
       IntToStr(ms) + ' м/с — это ' + FloatToStr(kmh) + ' км/час'
end;

// щелчок на кнопке Пересчет
procedure TForm1.Button1Click(Sender: TObject);
begin
    WindSpeed; // пересчитать скорость
end;

// нажатие клавиши в поле Edit1
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    // Key — символ нажатой клавиши
    // #8  — клавиша <Backspace>
    // #13 — клавиша <Enter>

    case Key of
        '0'..'9',#8:;           // цифры и клавиша <Backspace>
        #13:         WindSpeed; // пересчитать скорость
         else  Key := Chr(0);   // остальные символы не отображать
    end;

end;
```

4. Написать программу, которая пересчитывает массу из фунтов в килограммы (1 фунт = 409,5 грамм). Рекомендуемый вид формы приведен на рис. 1.2. Программа должна быть спроектирована таким образом, чтобы кнопка **Пересчет** была доступна только в том случае, если пользователь ввел исходные данные.

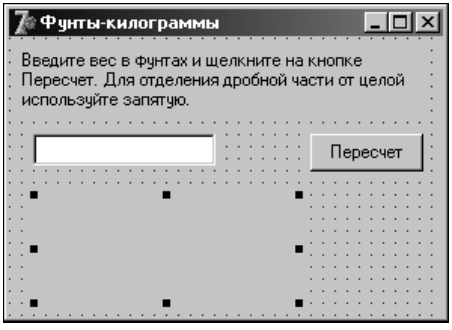


Рис. 1.2. Форма программы **Фунты-килограммы**

```
// щелчок на кнопке Пересчет
procedure TForm1.Button1Click(Sender: TObject);
var
   funt: real; // масса  в фунтах
   kg: real;   // масса  в килограммах
begin
   // Кнопка Пересчет доступна только в том случае,
   // если в поле Edit1 есть данные.
   // Поэтому наличие в поле информации можно не проверять.
   funt := StrToFloat(Edit1.Text);
   kg := funt * 0.4095;
   Label2.Caption := FloatToStrF(funt,ffGeneral,5,2) +
                     ' ф — это ' +
                     FloatToStrF(kg,ffGeneral,5,2) + ' кг';

end;

// нажатие клавиши в поле Edit1
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    case Key of
        '0'..'9', #8:;  // цифры и клавиша <Backspace>

{ Обработку десятичного разделителя
  сделаем "интеллектуальной". Заменим точку и
  запятую на символ DecimalSeparator — символ,
  который используется при записи дробных чисел.
}
        '.',',':
             begin
                Key := DecimalSeparator;
                // Проверим, введен ли уже в поле
                // Edit десятичный разделитель
                if pos(DecimalSeparator,Edit1.Text) <> 0
                   then Key := Char(0);
             end;
         else Key := Char(0); // остальные символы запрещены
     end;
end;

// содержимое поля Edit1 изменилось
procedure TForm1.Edit1Change(Sender: TObject);
```

```
begin
        // проверим, есть ли в поле Edit1 исходные данные
    if Length(Edit1.Text) = 0
       then Button1.Enabled := False // кнопка Пересчет недоступна
       else Button1.Enabled := True; // кнопка Пересчет доступна
end;

// Событие onCreate происходит в момент создания формы,
// до того, как форма появится на экране
procedure TForm1.FormCreate(Sender: TObject);
begin
    { т. к. поле Edit1 пустое (пользователь
    еще не ввел исходные данные), то
    сделаем кнопку Пересчет недоступной }
    Button1.Enabled := False;
end;
```

5. Написать программу, которая пересчитывает массу из фунтов в килограммы (1 фунт = 409,5 грамм). Рекомендуемый вид формы приведен на рис. 1.2. Программа должна быть спроектирована таким образом, чтобы пользователь мог ввести в поле **Масса** только положительное число (целое или дробное).

```
// щелчок на кнопке Пересчет
procedure TForm1.Button1Click(Sender: TObject);
var
   funt: real; // масса  в фунтах
   kg: real;   // масса  в килограммах
begin
   if Length(Edit1.Text) = 0 then
   begin
        // в поле Edit1 нет исходной информации
        ShowMessage('Надо ввести массу.');
        exit;
   end;

   funt := StrToFloat(Edit1.Text);
   kg := funt * 0.4095;
   Label2.Caption := FloatToStrF(funt,ffGeneral,5,2) + ' ф — это ' +
                     FloatToStrF(kg,ffGeneral,5,2) + ' кг';

end;
```

```
// нажатие клавиши в поле Edit
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    case Key of
        '0'..'9', #8:;  // цифры и клавиша <Backspace>

{ Обработку десятичного разделителя
  сделаем "интеллектуальной". Заменим точку и
  запятую на символ DecimalSeparator — символ,
  который используется при записи дробных чисел.
  Этот символ задается в настройках Windows.
}
        '.',',':
             begin
                Key := DecimalSeparator;
                // проверим, введен ли уже в поле
                // Edit десятичный разделитель
                if pos(DecimalSeparator,Edit1.Text) <> 0
                   then Key := Char(0);
             end;
         else Key := Char(0); // остальные символы запрещены
     end;
end;
```

6. Написать программу, которая вычисляет скорость (км/час), с которой бегун пробежал дистанцию. Рекомендуемый вид формы приведен на рис. 1.3. Количество минут задается целым числом, секунд — дробным.

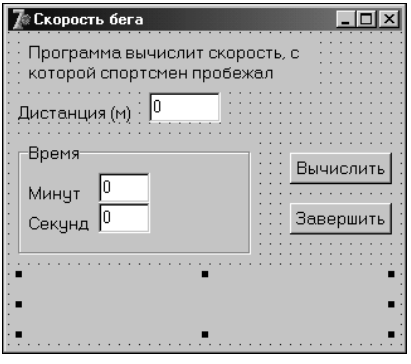


Рис. 1.3. Форма программы **Скорость бега**

```
// нажатие клавиши в поле Дистанция
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  // Key — символ, соответствующий нажатой клавише.
  // Если символ недопустимый, то процедура заменяет его
  // на символ с кодом 0. В результате этого символ в поле
  // редактирования не появляется и у пользователя создается
  // впечатление, что программа не реагирует на нажатие
  // некоторых клавиш.
  case Key of
    '0'..'9':              ; // цифры
    #8      :              ; // клавиша <Backspace>
    #13     : Edit2.SetFocus; // при нажатии <Enter>  курсор
                              // переводится в поле Время:минут

    // остальные символы запрещены
    else Key :=Chr(0);  // символ не отображать
  end;
end;


// нажатие клавиши в поле Время:минут
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
  case Key of
    '0'..'9':              ;
    #8      :              ;  // клавиша <Backspace>
    #13     : Edit3.SetFocus; // при нажатии <Enter> курсор
                              // переводится в поле Время:секунд

    // остальные символы запрещены
    else Key :=Chr(0);  // символ не отображать
  end;
end;

// нажатие клавиши в поле Время:секунд
procedure TForm1.Edit3KeyPress(Sender: TObject; var Key: Char);
begin
   case Key of
    '0'..'9':;
    ',','.' : // десятичный разделитель
             begin
                Key := DecimalSeparator;
```

```
            if Pos(DecimalSeparator,Edit3.Text) <> 0
               then Key := Char(0);
          end;
    #8  :               ;  // клавиша <Backspace>
    #13 : Button1.SetFocus; // при нажатии клавиши <Enter>
                            // активируется кнопка Вычислить

    // остальные символы — запрещены
    else Key :=Chr(0);  // символ не отображать
  end;
end;

// щелчок на кнопке Вычислить
procedure TForm1.Button1Click(Sender: TObject);
var
  dist : integer; // дистанция, метров
  min : integer;  // время, минуты
  sek : real;     // время, секунды

  v: real;        // скорость
begin
  // получить исходные данные из полей ввода
  dist := StrToInt(Edit1.Text);
  min  := StrToInt(Edit2.Text);
  sek  := StrToFloat(Edit3.Text);

  // дистанция и время не должны быть равны нулю
  if (dist = 0) or ((min = 0) and (sek = 0)) then
  begin
      ShowMessage('Надо задать дистанцию и время.');
      exit;
  end;

  // вычисление
  v := (dist/1000) / ((min*60 + sek)/3600);

  // вывод результата
  label5.Caption := 'Дистанция: '+ Edit1.Text + ' м' + #13 +
                    'Время: ' + IntToStr(min) + ' мин ' +
                    FloatToStrF(sek, ffGeneral,4,2) + ' сек ' +
                    #13 + 'Скорость: ' +
                    FloatToStrF(v,ffFixed,4,2) + ' км/час';

end;
```

```
// щелчок на кнопке Завершить
procedure TForm1.Button2Click(Sender: TObject);
begin
      Form1.Close; // закрыть главную форму — завершить работу
                   // программы
end;
```

7. Написать программу, которая вычисляет силу тока в электрической цепи. Рекомендуемый вид формы приведен на рис. 1.4. Программа должна быть спроектирована таким образом, чтобы кнопка **Вычислить** была доступна только в том случае, если пользователь ввел величину сопротивления.

Рис. 1.4. Форма программы **Сила тока**

8. Написать программу, которая вычисляет силу тока в электрической цепи. Цепь состоит из двух параллельно соединенных сопротивлений. Рекомендуемый вид формы приведен на рис. 1.5.

9. Написать программу, которая вычисляет стоимость покупки с учетом скидки. Скидка 1 % предоставляется, если сумма покупки больше 300 рублей, 2 % — если сумма больше 500 рублей, 3 % — если сумма больше 1 000. Информация о предоставленной скидке (процент и величина) должна быть выведена в диалоговом окне. Рекомендуемый вид формы программы приведен на рис. 1.6.

10. Напишите программу, которая вычисляет стоимость покупки. Пользователь должен вводить код товара и количество единиц. Программа должна формировать список товаров. Рекомен-

дуемый вид формы приведен на рис. 1.7. (Развитие. Список кодов и наименований и цен товаров вводить из файла.)

Рис. 1.5. Форма программы **Сила тока**

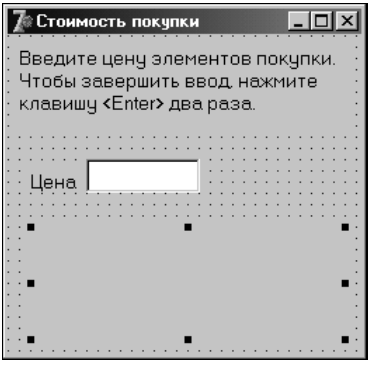


Рис. 1.6. Форма программы **Стоимость покупки**

Рис. 1.7. Форма программы **Супер маркет**

11. Напишите программу, которая вычисляет доход по вкладу. Программа должна обеспечивать расчет простых и сложных процентов. Простые проценты начисляются в конце срока вклада, сложные — ежемесячно и прибавляются к первоначальной

сумме вклада и в следующем месяце проценты начисляются на новую сумму. Рекомендуемый вид формы программы приведен на рис. 1.8.

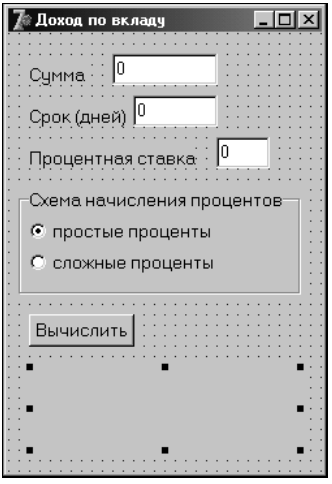


Рис. 1.8. Форма программы **Доход по вкладу**

```
// щелчок на кнопке Вычислить
procedure TForm1.Button1Click(Sender: TObject);
var
    sum : real;    // сумма вклада
    pr:   real;    // процентная ставка
    srok: integer; // срок вклада
    dohod: real;   // доход по вкладу

    buf: real;
    i: integer;
begin
    // получить исходные данные
    sum := StrToFloat(Edit1.Text);
    pr := StrToFloat(Edit2.Text);
    srok := StrToInt(Edit3.Text);

    if RadioButton1.Checked then
        // выбран переключатель Простые проценты
        dohod := sum * (pr/100) * (srok/360)
```

```
    else
        // т. к. в группе два переключателя, то если
        // не выбран RadioButton1, то выбран
        // RadioButton2 — Сложные проценты
        begin
            buf:= sum;
            for i:=1 to srok do
                buf:= buf + buf * (pr/100);
            // здесь buf — сумма в конце срока вклада
            dohod := buf — sum;
        end;

    sum := sum + dohod;
    Label4.Caption := 'Доход: ' + Float-
ToStrF(dohod,ffGeneral,9,2) + #13 +
              'Сумма в конце срока вклада: ' +
              FloatToStrF(sum,ffGeneral,9,2);
end;

// выбор переключателя Простые проценты
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
    Label2.Caption := 'Срок (дней)';
    Label4.Caption := '';

end;

// выбор переключателя Сложные проценты
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
    Label2.Caption := 'Срок (мес.)';
    Label4.Caption := '';
end;
```

12. Написать программу, которая вычисляет сопротивление электрической цепи, состоящей из двух сопротивлений. Сопротивления могут быть соединены последовательно или параллельно. Рекомендуемый вид формы приведен на рис. 1.9. Если величина сопротивления цепи превышает 1 000 Ом, то результат должен быть выведен в килоомах.

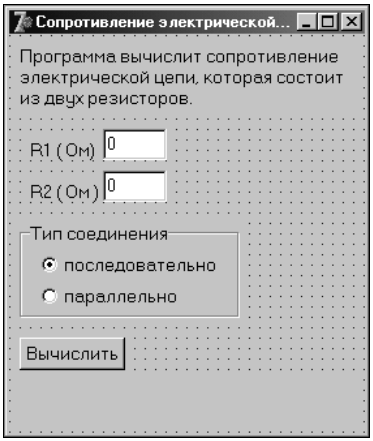


Рис. 1.9. Форма программы **Сопротивление электрической цепи**

```
// щелчок на кнопке Вычислить
procedure TForm1.Button1Click(Sender: TObject);
var
    r1,r2: real; // величины сопротивлений
    r: real;     // сопротивление цепи
begin
    // получить исходные данные
    r1 := StrToFloat(Edit1.Text);
    r2 := StrToFloat(Edit2.Text);

    if (r1 = 0) and (r2 = 0) then
    begin
        ShowMessage('Надо задать величину хотя бы одного
                     сопротивления');
        exit;
    end;

    // переключатели RadioButton1 и RadioButton2
    // зависимые, поэтому о типе соединения можно
    // судить по состоянию одного из этих
    // переключателей
    if RadioButton1.Checked
        then // выбран переключатель Последовательно
            r:= r1+r2
        else //  выбран переключатель Параллельно
            r:= (r1*r2)/(r1+r2);
```

```
    Label4.Caption := 'Сопротивление цепи: ';
    if r < 1000 then
       Label4.Caption := Label4.Caption +
                   FloatToStrF(r,ffGeneral,3,2) + ' Ом'
    else
        begin
            r:=r/1000;
            Label4.Caption := Form1.Label4.Caption +
                   FloatToStrF(r,ffGeneral,3,2) + ' кОм';
        end
end;

// щелчок на переключателе Последовательно
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
   // пользователь изменил тип соединения
   Label4.Caption := '';
end;

// щелчок на переключателе Параллельно
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
   // пользователь изменил тип соединения
   Label4.Caption := '';
end;
```

13. Написать программу, которая вычисляет силу тока в электрической цепи. Цепь состоит из двух сопротивлений. Сопротивления могут быть соединены последовательно или параллельно. Рекомендуемый вид формы приведен на рис. 1.10.

14. Написать программу, которая, используя закон Ома, вычисляет силу тока, напряжение или сопротивление электрической цепи. Рекомендуемый вид формы приведен на рис. 1.11. Во время работы программы, в результате выбора переключателя **Ток**, **Напряжение** или **Сопротивление**, текст, поясняющий назначение полей ввода, должен меняться.

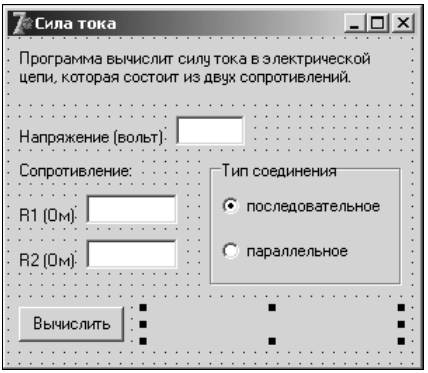


Рис. 1.10. Форма программы **Сила тока**

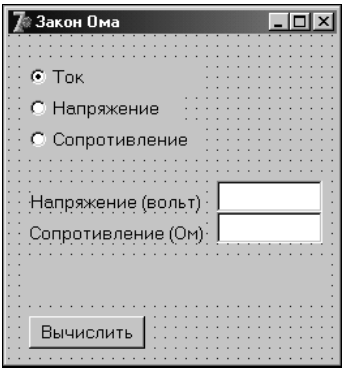


Рис. 1.11. Форма программы **Закон Ома**

```
// вычисление тока, напряжения или сопротивления
procedure Calculate;
var
  I,U,R:  real; // ток, напряжение, сопротивление
begin
  if Form1.RadioButton1.Checked then
  // ток
  begin
    U := StrToFloat(Form1.Edit1.Text);
    R := StrToFloat(Form1.Edit2.Text);
    if (R <> 0) then
    begin
      I := U/R;
      Form1.Label3.Caption := 'Ток: ' + Float-
ToStrF(I,ffFixed,4,2) + ' A';
    end
    else ShowMessage('Сопротивление не должно быть равно нулю.');
    exit;
  end;

  if Form1.RadioButton2.Checked then
  // напряжение
  begin
    I := StrToFloat(Form1.Edit1.Text);
    R := StrToFloat(Form1.Edit2.Text);
    U := I*R;
```

```
    Form1.Label3.Caption := 'Напряжение: ' +
                            FloatToStrF(U,ffFixed,4,2) + ' В';
    exit;
  end;

  if Form1.RadioButton3.Checked then
  // сопротивление
  begin
    U := StrToFloat(Form1.Edit1.Text);
    I := StrToFloat(Form1.Edit2.Text);
    if (I <> 0) then
    begin
      R := U/I;
      Form1.Label3.Caption := 'Сопротивление: ' + Float-
ToStrF(R,ffFixed,4,2) + ' Ом';
    end
    else ShowMessage('Ток не должен быть равен нулю.');
  end;

end;

// Выбор переключателя Ток
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  Label1.Caption := 'Напряжение (вольт)';
  Label2.Caption := 'Сопротивление (Ом)';
  Label3.Caption := '';
end;

// Выбор переключателя Напряжение
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
  Label1.Caption := 'Ток (ампер)';
  Label2.Caption := 'Сопротивление (Ом)';
  Label3.Caption := '';
end;

// Выбор переключателя Сопротивление
procedure TForm1.RadioButton3Click(Sender: TObject);
begin
  Label1.Caption := 'Напряжение (вольт)';
  Label2.Caption := 'Ток (ампер)';
  Label3.Caption := '';
end;
```

```
// Нажатие клавиши в поле Edit1
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key:Char);
begin
  case Key of
    '0'..'9',#8: ; // цифры и клавиша <Backspase>
    #13: Edit2.SetFocus; // клавиша <Enter>
    '.',',':
      begin
        if Key = '.'
          then Key := ',';
        // не позволяет вводить знак запятой повторно
        if Pos(',',Edit1.Text) <> 0
          then Key := Chr(0);
      end;
    else Key := Chr(0);
  end;
end;

// Нажатие клавиши в поле Edit2
procedure TForm1.Edit2KeyPress(Sender: TObject; var Key:Char);
begin
  case Key of
    '0'..'9',#8: ;
    #13: Calculate; // клавиша <Enter> — вычислить
    '.',',':
      begin
        if Key = '.'
          then Key := ',';
        // не позволяет вводить знак запятой повторно
        if Pos(',',Edit2.Text) <> 0
          then Key := Chr(0);
      end;
    else Key := Chr(0);
  end;
end;

// Щелчок на кнопке Вычислить
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (Edit1.Text <> '') and (Edit2.Text <> '')
    then Calculate // вычислить ток, напряжение или сопротивление
  else ShowMessage('Надо ввести исходные данные в оба поля');
end;
```

15. Написать программу, которая вычисляет стоимость поездки на автомобиле, например, на дачу. Рекомендуемый вид формы приведен на рис. 1.12.

Рис. 1.12. Форма программы **Поездка на дачу**

```
implementation

{$R *.dfm}

{ Процедура EditKeyPress обрабатывает нажатие клавиш
  в поле Расстояние, Цена и Потребление.
  Сначала надо обычным образом создать процедуру
  обработки события OnKeyPress для поля Edit1,
  затем назначить эту процедуру событию OnKeyPres
  полей Edit2 и Edit3. Кроме того, свойству Tag
  компонентов Edit1, Edit2 и Edit3 надо присвоить
  соответственно значения 1, 2 и 3. Свойство Tag
  используется в процедуре EditKeyPress
  для идентификации компонента. }

procedure TForm1.EditKeyPress(Sender: TObject; var Key: Char);
var
    Edit: TEdit;
begin
    Edit := Sender as TEdit;
    // в поле Edit можно ввести только дробное число
    case Key of
    '0'..'9':; // цифры
    #8:      ; // клавиша <Backspace>
```

```
    '.',',': begin
                Key := DecimalSeparator;
                if Pos(DecimalSeparator,Edit.Text) <> 0
                    then Key := #0;
             end;
    #13: // клавиша <Enter>
        case Edit.Tag of
        1:  // клавиша нажата в поле Edit1
            Edit2.SetFocus; // фокус в поле Edit2
        2:  // клавиша нажата в поле Edit1
            Edit3.SetFocus; // фокус в поле Edit3
        3:  // клавиша нажата в поле Edit3
            Button1.SetFocus; // фокус на кнопку Button1
        end;
    end;
end;

// щелчок на кнопке Вычислить
procedure TForm1.Button1Click(Sender: TObject);
var
  rast : real; // расстояние
  cena : real; // цена
  potr : real; // потребление на 100 км
  summ : real; // сумма
  mes: string;
begin
  // здесь возможно исключение типа EConvertError
  // в случае, если пользователь оставит
  // одно из полей ввода незаполненным
  try
        rast := StrToFloat(Edit1.Text);
        cena := StrToFloat(Edit2.Text);
        potr := StrToFloat(Edit3.Text);
    except
    on EConvertError do
    begin
        ShowMessage('Даные надо ввести во все поля!');
        // попытаемся найти пустое поле
        if Length(Edit1.Text) = 0
            then Edit1.SetFocus
        else if Length(Edit2.Text) = 0
                then Edit2.SetFocus
```

```
                else Edit3.SetFocus;
          exit;
      end;
    end;

    summ := (rast / 100) * potr * cena;
    mes := 'Поездка на дачу';

    if  CheckBox1.Checked  then
    begin
        summ := summ * 2;
        mes := mes + ' и обратно';
    end;

    mes := mes + 'обойдется в ' + FloatToStrF(summ,ffGeneral,4,2)
               + ' руб.';
    Label4.Caption := mes;
end;

end.
```

16. Напишите программу-калькулятор, выполняющий сложение и вычитание. Рекомендуемый вид формы приведен на рис. 1.13.

К этой задаче даны два варианта решения. В первом варианте для каждой цифровой кнопки создана отдельная процедура обработки события `OnClick`. Во втором варианте событие `OnClick` всех цифровых кнопок обрабатывает одна процедура, что позволило значительно сократить текст программы.

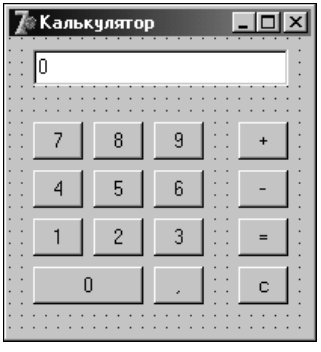


Рис. 1.13. Форма программы **Калькулятор**

```
// ************************************************
// Вариант 1. Событие OnClick на каждой цифровой кнопке
// обрабатывает отдельная процедура.

implementation

{$R *.dfm}

var
  accum:  real;     // аккумулятор
  oper:   integer;  // операция: 1 — '+'; 2 — '-';
                    // 0 — "выполнить" (кнопка "=")

  f:      integer;
  { f = 0 ждем первую цифру нового числа, например,
          после выполнения операции, когда на
          индикаторе результат.
    f = 1 ждем остальные цифры. }

// кнопка 0
procedure TForm1.Button0Click(Sender: TObject);
begin
  if f = 0 // первая цифра числа
  then begin
         Edit1.Text := '0';
         f := 1; // ждем остальные цифры
       end
  else
      if Edit1.Text <> '0'
        // чтобы на индикаторе не было
        // нескольких нулей в начале числа
        then  Edit1.Text := Edit1.Text + '0';
end;

// кнопка 1
procedure TForm1.Button1Click(Sender: TObject);
begin
  if f = 0  // первая цифра числа
  then  begin
      Edit1.Text := '1';
      f := 1;  // ждем остальные цифры
    end
```

```
  else Edit1.Text := Edit1.Text + '1';
end;

// кнопка 2
procedure TForm1.Button2Click(Sender: TObject);
begin
  if (f = 0) // первая цифра числа
  then begin
         Edit1.Text := '2';
         f := 1;   // ждем остальные цифры
    end
  else Edit1.Text := Edit1.Text + '2';
end;

// кнопка 3
procedure TForm1.Button3Click(Sender: TObject);
begin
  if f = 0
  then begin
         Edit1.Text := '3';
         f := 1;
       end
  else Edit1.Text := Edit1.Text + '3';
end;


// кнопка 4
procedure TForm1.Button4Click(Sender: TObject);
begin
  if f = 0 then
    begin
      Edit1.Text := '4';
      f := 1;
    end
  else Edit1.Text := Edit1.Text + '4';
end;

// кнопка 5
procedure TForm1.Button5Click(Sender: TObject);
begin
  if (f = 0)
  then begin
         Edit1.Text := '5';
```

```
        f := 1;
      end
  else Edit1.Text := Edit1.Text + '5';
end;

// кнопка 6
procedure TForm1.Button6Click(Sender: TObject);
begin
  if f = 0
  then begin
        Edit1.Text := '6';
        f := 1;
      end
  else Edit1.Text := Edit1.Text + '6';
end;

// кнопка 7
procedure TForm1.Button7Click(Sender: TObject);
begin
  if f = 0
  then begin
      // первая цифра числа
      Edit1.Text := '7';
      f := 1;
    end
  else Edit1.Text := Edit1.Text + '7';
end;

// кнопка 8
procedure TForm1.Button8Click(Sender: TObject);
begin
  if f = 0
  then begin
        Edit1.Text := '8';
        f := 1;
      end
  else Edit1.Text := Edit1.Text + '8';
end;

// кнопка 9
procedure TForm1.Button9Click(Sender: TObject);
begin
```

```
  if f = 0  // первая цифра числа
  then begin
         Edit1.Text := '9';
         f := 1;  // ждем остальные цифры
       end
  else Edit1.Text := Edit1.Text + '9';
end;

// десятичная точка
procedure TForm1.ButtonZClick(Sender: TObject);
begin
  if Edit1.Text = '0' then
    begin
      Edit1.Text := '0,';
      f := 1;
    end;
  if Pos(',',Edit1.Text) = 0 then
      Edit1.Text := Edit1.Text + ',';
end;

// выполнение операции
procedure DoOper;
var
    numb: real; // число на индикаторе
begin
    // accum содержит результат предыдущей
    // операции, oper — код операции, которую
    // надо выполнить. Операнд находится
    // на индикаторе.
    numb := StrToFloat(Form1.Edit1.Text);
    case oper of
    0: accum := numb;
    1: accum := accum + numb;
    2: accum := accum — numb;
    end;
    Form1.Edit1.Text := FloatToStr(accum);
end;

// кнопка "+".
procedure TForm1.ButtonPlusClick(Sender: TObject);
{  Надо выполнить предыдущую операцию,
  вывести результат на индикатор,
```

```
  запомнить текущую операцию и
  установить режим ожидания первой
  цифры нового числа. }
begin
  if f = 0 // ждем первую цифру, но пользователь
           // щелкнул на кнопке операции
    then oper := 1 // запомним операцию
  else begin
    // на индикаторе есть число
    DoOper;   // выполнить предыдущую операцию
    oper :=1; // запомнить текущую операцию
    f:=0;     // ждем первую цифру нового числа
  end;
end;

// кнопка "-"
procedure TForm1.ButtonMinusClick(Sender: TObject);
// см. комментарий к процедуре обработки OnClick на "+"
begin
  if f = 0 // ждем первую цифру
    then oper := 2
  else begin
    DoOper;   // выполнить предыдущую операцию
    oper :=2; // запомнить текущую операцию
    f:=0;     // ждем первую цифру нового числа
  end;
end;

// кнопка "="
procedure TForm1.ButtonEnterClick(Sender: TObject);
begin
  if f = 0   // ждем первую цифру
    then oper := 0
  else begin
    DoOper;   // выполнить предыдущую операцию
    oper :=0; // запомнить текущую операцию
    f:=0;     // ждем первую цифру нового числа
  end;
end;

// кнопка "c" — очистка
procedure TForm1.ButtonCClick(Sender: TObject);
```

```
begin
  Edit1.Text := '0';
  accum := 0;
  oper := 0;
  f := 0; // ждем первую цифру числа
end;

// нажатие клавиши в поле Edit1
procedure TForm1.Edit1Change(Sender: TObject; var Key:Char);
begin
  Key := Chr(0);  // не отображать символы
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    oper := 0;
end;

end.


// *********************************
// Вариант 2. События OnClick на всех
// цифровых кнопках обрабатывает одна
// процедура

implementation

{$R *.dfm}

{ 1. Во время создания формы свойству Tag каждой
     цифровой кнопки надо присвоить значение,
     равное цифре, которая должна появиться
     на индикаторе калькулятора.
  2. После того как обычным образом будет создана
     процедура обработки события OnClick для одной
     из цифровых кнопок, например для кнопки "1",
     надо задать, что событие OnClick для остальных
     кнопок обрабатывает эта же процедура. Делается
     это выбором  процедуры в списке, который
     появляется в результате щелчка на значке
     раскрывающегося списка в соответствующей
     строке вкладки Events.
  }
```

```
var
  accum:  real;     // аккумулятор
  oper:   integer;  // операция: 1 — '+'; 2 — '-';
                    // 0 — "выполнить" (кнопка "=")

  f:      integer;
  { f = 0 ждем первую цифру нового числа, например,
          после выполнения операции, когда
          на индикаторе результат.
    f = 1 ждем остальные цифры. }

// Щелчок на кнопках "0" — "9"
procedure TForm1.DigitBtnClick(Sender: TObject);
var
    Btn: TButton;
    ch: Char;
begin
  Btn := Sender as TButton;
  ch  := Chr(48+Btn.Tag);
  // chr(48) = '0'; chr(49) = '1' и т. д.
  // можно и так: ch := Btn.Caption;

  case Btn.Tag of
  1..9: // кнопки "1" — "9"
    if f = 0 // первая цифра числа
        then begin
            Edit1.Text := ch;
            f := 1; // ждем остальные цифры
        end
    else Edit1.Text := Edit1.Text + ch;

  0: // кнопка "0"
      if Edit1.Text <> '0' // на индикаторе 0
        // чтобы на индикаторе не было
        // нескольких нулей в начале числа
        then  Edit1.Text := Edit1.Text + '0';
  end;
end;

// десятичная точка
procedure TForm1.ButtonZClick(Sender: TObject);
begin
  if Edit1.Text = '0' then
```

```
    begin
      Edit1.Text := '0,';
      f := 1;
    end;
  if Pos(',',Edit1.Text) = 0 then
      Edit1.Text := Edit1.Text + ',';
end;

// выполнение операции
procedure DoOper;
var
    numb: real; // число на индикаторе
begin
    // accum содержит результат предыдущей
    // операции, oper — код операции, которую
    // надо выполнить. Операнд находится
    // на индикаторе.
    numb := StrToFloat(Form1.Edit1.Text);
    case oper of
    0: accum := numb;
    1: accum := accum + numb;
    2: accum := accum — numb;
    end;
    Form1.Edit1.Text := FloatToStr(accum);
end;

// Обрабатывает щелчок на кнопках "+", "-" и "="
procedure TForm1.OpBtnClick(Sender: TObject);
{  Надо выполнить предыдущую операцию,
  вывести результат на индикатор,
  запомнить текущую операцию и
  установить режим ожидания первой
  цифры нового числа. }
var
  Btn: TButton;
begin
  Btn := Sender as TButton;
  if f = 0 // ждем первую цифру, но пользователь
           // щелкнул на кнопке операции
    then
      // свойство Tag кнопки хранит код операции
      oper :=  Btn.Tag // запомним операцию
```

```
  else begin
    DoOper;          // выполнить предыдущую операцию
    oper := Btn.Tag; // запомнить текущую операцию
    f:=0;            // ждем первую цифру нового числа
  end;
end;

// кнопка "c" — очистка
procedure TForm1.ButtonCClick(Sender: TObject);
begin
  Edit1.Text := '0';
  accum := 0;
  oper := 0;
  f := 0; // ждем первую цифру числа
end;

// нажатие клавиши в поле Edit1
procedure TForm1.Edit1Change(Sender: TObject; var Key:Char);
begin
  Key := Chr(0);  // не отображать символы
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    oper := 0;
end;

end.
```

17. Напишите программу "Электронные часы", на поверхности формы которой отображается текущее время (рис. 1.14).

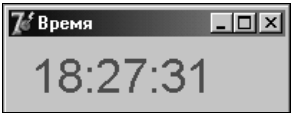


Рис. 1.14. Окно программы **Электронные часы**

```
unit eclock_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,   Dialogs, ExtCtrls, StdCtrls;
```

```
type
  TForm1 = class(TForm)
    Timer1: TTimer;
    Label1: TLabel;
    procedure FormPaint(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);

    // ********************************
    // объявление процедуры ShowTime
    // помещено сюда вручную
    // это сделано, чтобы функция имела
    // доступ к компонентам формы напрямую
    procedure ShowTime;

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// отображает текущее время
procedure TForm1.ShowTime;
var
  Time: TDateTime;  // текущее время
begin
  Time := Now();   // получить системное время
  Label1.Caption := FormatDateTime('hh:mm:ss',Time);
end;

// обработка события Paint
procedure TForm1.FormPaint(Sender: TObject);
begin
  ShowTime;   // отобразить часы
end;

// обработка сигнала таймера
procedure TForm1.Timer1Timer(Sender: TObject);
```

```
begin
  ShowTime;   // отобразить время
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  // настроить и запустить таймер
  Timer1.Interval := 1000;   // период сигналов таймера  1 с
  Timer1.Enabled := True;    // пуск таймера
end;

end.
```

18. Напишите программу "Электронные часы", в окне которой отображается текущее время и дата (рис. 1.15).

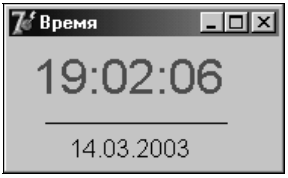


Рис. 1.15. В окне программы **Электронные часы** отображается текущее время и дата

```
unit eclock_2_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, ExtCtrls, StdCtrls;
type
  TForm1 = class(TForm)
    Timer1: TTimer;
    Label1: TLabel;
    Label2: TLabel;
    Shape1: TShape;
    procedure FormPaint(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
```

```
    // *********************************
    // Объявление процедуры ShowTime
    // помещено сюда вручную.
    // Это сделано, чтобы функция имела
    // доступ к компонентам формы напрямую
    procedure ShowTime;

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// отображает текущее время
procedure TForm1.ShowTime;
var
  Time: TDateTime;  // текущее время
begin
  Time := Now();   // получить системное время
  Label1.Caption := FormatDateTime('hh:mm:ss',Time);
end;

// обработка события Paint
procedure TForm1.FormPaint(Sender: TObject);
begin
  ShowTime;   // отобразить часы
end;

// обработка сигнала таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  ShowTime;   // отобразить время
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Label2.Caption := FormatDateTime('dd.mm.yyyy',Now());
```

```
  // настроить и запустить таймер
  Timer1.Interval := 1000;   // период сигналов таймера  1 с
  Timer1.Enabled := True;    // пуск таймера
end;

end.
```

19. Напишите программу "Электронные часы", в окне которой отображается текущее время, дата и день недели (рис. 1.16).

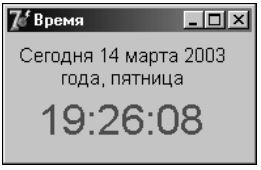


Рис. 1.16. В окне программы **Электронные часы** отображается текущее время, дата и день недели

```
unit eclock_3_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    Label1: TLabel;    // время
    Label2: TLabel;   // дата и день недели
    procedure FormPaint(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);

    // *********************************
    // Объявление процедуры ShowTime
    // помещено сюда вручную.
    // Это сделано, чтобы функция имела
    // доступ к компонентам формы напрямую
    procedure ShowTime;
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;


implementation

const
    stDay : array[1..7] of string[11] =
       ('воскресенье','понедельник','вторник',
        'среда','четверг','пятница','суббота');

    stMonth : array[1..12] of string[8] =
       ('января','февраля','марта',
        'апреля','мая','июня','июля',
        'августа','сентября','октября',
        'ноября','декабря');

{$R *.dfm}

// отображает текущее время
procedure TForm1.ShowTime;
var
  Time: TDateTime;  // текущее время
begin
  Time := Now();   // получить системное время
  Label1.Caption := FormatDateTime('hh:mm:ss',Time);
end;

// обработка события Paint
procedure TForm1.FormPaint(Sender: TObject);
begin
  ShowTime;   // отобразить часы
end;

// обработка сигнала таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  ShowTime;   // отобразить время
end;
```

```
// обработка события OnCreate
procedure TForm1.FormCreate(Sender: TObject);
var
  Present: TDateTime;       // текущая дата и время
  Year, Month, Day : Word; // год, месяц и число, как
                           // отдельные числа
begin

  Present:= Now; // получить текущую дату
  DecodeDate(Present, Year, Month, Day);
  Label2.Caption := 'Сегодня '+IntToStr(Day)+ ' ' +
    stMonth[Month] + ' '+ IntToStr(Year)+
    ' года, '+ stDay[DayOfWeek(Present)];

  // настроить и запустить таймер
  Timer1.Interval := 1000;   // период сигналов таймера  1 с
  Timer1.Enabled := True;    // пуск таймера
end;

end.
```

20. Напишите программу "Таймер". На рис. 1.17 приведена форма и окна программы во время установки интервала и в процессе отсчета времени.

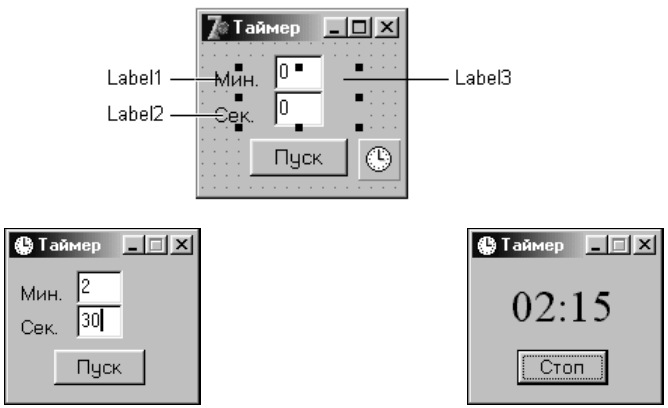


Рис. 1.17. Форма и окна программы **Таймер**

```
implementation

{$R *.dfm}

{ Во время создания формы свойству Visible компонента
  Label3 надо присвоить значение False }
var
  // интервал
  min: integer;  // минут
  sec: integer;  // секунд

// в заголовок окна программы
// выводится, сколько времени осталось
procedure ShowTime;
var
    buf: string[20];
begin
  // минуты и секунды выводим двумя цифрами
  if min < 10 then
     buf := '0' + IntToStr(min) + ':'
  else
     buf := IntToStr(min) + ':';

  if sec < 10 then
    buf := buf + '0' + IntToStr(sec)
  else
    buf := buf + IntToStr(sec);

  Form1.Label3.Caption := buf;
end;

// сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  sec := sec - 1;
  ShowTime; // показать, сколько времени осталось
  if (min = 0) and (sec = 0) then
    // заданный интервал истек
    begin
      Timer1.Enabled := False;   // стоп
      ShowMessage('Заданный интервал истек!');
      Button1.Caption := 'Пуск';
      Label3.Visible := False;   // скрыть индикацию времени
```

```
      // сделать видимыми поля ввода интервала
      Label1.Visible := True;
      Edit1.Visible  := True;
      Label2.Visible := True;
      Edit2.Visible  := True;
      exit;
    end;

  if (sec = 0) and (min > 0) then
     begin
         sec := 60;
         min := min — 1;
     end;

end;

// щелчок на кнопке Пуск/Стоп
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Form1.Timer1.Enabled then
    // таймер работает, надо остановить
    begin
      Timer1.Enabled := False;   // стоп
      Button1.Caption := 'Пуск';
      Label3.Visible := False;   // скрыть индикацию времени
      // сделать видимыми поля ввода интервала
      Label1.Visible := True;
      Edit1.Visible  := True;
      Label2.Visible := True;
      Edit2.Visible  := True;
    end

  else
    // таймер стоит, надо запустить
    begin
      min := StrToInt(Edit1.Text);
      sec := StrToInt(Edit2.Text);
      if (sec = 0) and (min = 0) then
      begin
        ShowMessage('Надо задать интервал!');
        exit;
      end;
```

```
      Timer1.Enabled := True;    // запустить таймер
      // скрыть поля ввода интервала
      Label1.Visible := False;
      Edit1.Visible  := False;
      Label2.Visible := False;
      Edit2.Visible  := False;
      Label3.Visible := True;
      Button1.Caption := 'Стоп';
      ShowTime;
    end;
end;

end.
```

21. Усовершенствуйте программу "Таймер" так, чтобы по истечении установленного интервала программа привлекала внимание пользователя звуковым сигналом, например, одним из стандартных звуков Windows.

```
implementation

{$R *.dfm}

const
    SOUND = 'tada.wav';
var
  MediaPlayer : TMediaPlayer;  // обеспечивает воспроизведение
                               // звукового фрагмента
  // интервал
  min: integer;  // минут
  sec: integer;  // секунд

// в заголовок окна программы
// выводит, сколько времени осталось
procedure ShowTime;
var
    buf: string[20];
begin
  // минуты и секунды выводим двумя цифрами
  if min < 10 then
     buf := '0' + IntToStr(min) + ':'
  else
     buf := IntToStr(min) + ':';
```

```
  if sec < 10 then
    buf := buf + '0' + IntToStr(sec)
  else
    buf := buf + IntToStr(sec);

  Form1.Label3.Caption := buf;
end;

// сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  sec := sec - 1;
  ShowTime; // показать, сколько времени осталось
  if (min = 0) and (sec = 0) then
    // заданный интервал истек
    begin
      Timer1.Enabled := False;   // стоп

      // *** звуковой сигнал ****
      // т. к. возможно, что WAV-файл не был
      // загружен (см. FormCreate),
      // то может возникнуть исключение
      try
        MediaPlayer.Play;  // воспроизвести звуковой фрагмент
      except
        on EMCIDeviceError do;
      end;

      ShowMessage('Заданный интервал истек!');
      Button1.Caption := 'Пуск';
      Label3.Visible := False;   // скрыть индикацию времени
      // сделать видимыми поля ввода интервала
      Label1.Visible := True;
      Edit1.Visible  := True;
      Label2.Visible := True;
      Edit2.Visible  := True;
      exit;
    end;

  if (sec = 0) and (min > 0) then
     begin
         sec := 60;
```

```
        min := min – 1;
     end;
end;

// щелчок на кнопке Пуск/Стоп
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Form1.Timer1.Enabled then
    // таймер работает, надо остановить
    begin
      Timer1.Enabled := False;   // стоп
      Button1.Caption := 'Пуск';
      Label3.Visible := False;   // скрыть индикацию времени
      // сделать видимыми поля ввода интервала
      Label1.Visible := True;
      Edit1.Visible  := True;
      Label2.Visible := True;
      Edit2.Visible  := True;
    end

  else
    // таймер стоит, надо запустить
    begin
      min := StrToInt(Edit1.Text);
      sec := StrToInt(Edit2.Text);
      if (sec = 0) and (min = 0) then
      begin
        ShowMessage('Надо задать интервал!');
        exit;
      end;

      Timer1.Enabled := True;    // запустить таймер
      // скрыть поля ввода интервала
      Label1.Visible := False;
      Edit1.Visible  := False;
      Label2.Visible := False;
      Edit2.Visible  := False;
      Label3.Visible := True;
      Button1.Caption := 'Стоп';
      ShowTime;
    end;
end;
```

```
// создает компонент MediaPlayer и
// загружает WAV-файл
procedure TForm1.FormCreate(Sender: TObject);
var
    pWinDir: PChar;      // указатель на nul-terminated-строку
    sWinDir: String[80];
begin
    // создадим компонент MediaPlayer
    MediaPlayer := TMediaPlayer.Create(self);
    MediaPlayer.ParentWindow := Form1.Handle;
    MediaPlayer.Visible := False;

    // Стандартные WAV-файлы находятся в каталоге Media,
    // но где находится и как называется каталог, в который
    // установлен Windows? Выясним это.
    // Чтобы получить имя каталога Windows,
    // воспользуемся API-функцией GetWindowsDirectory.
    // Строка, которая передается в API-функцию,
    // должна быть nul-terminated-строкой.

    // Получить имя каталога Windows
    GetMem(pWinDir,80);              // выделить память
                                     // для строки
    GetWindowsDirectory(pWinDir,80); // получить каталог Windows
    sWinDir := pWindir;

     // открыть WAV-файл
    MediaPlayer.FileName := sWinDir + '\media\' + SOUND;
    try
        MediaPlayer.Open;
    except
        on EMCIDeviceError do;
    end;
end;

end.
```

22. Напишите программу "Таймер". Для ввода интервала используйте компоненты `UpDown`. Рекомендуемый вид формы приведен на рис. 1.18.

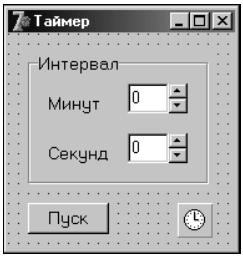


Рис. 1.18. Форма программы **Таймер**

```
{ Чтобы обеспечить синхронизацию компонентов UpDown
  и Edit, нужно в свойство Associate компонента UpDown
  записать имя соответствующего компонента Edit.
  Это надо сделать во время создания формы. }

var
  // интервал
  min: integer;  // минут
  sec: integer;  // секунд

// в заголовок окна программа
// выводит, сколько времени осталось
procedure ShowTime;
var
    buf: string[20];
begin
  buf := 'Таймер ';
  // минуты и секунды выводим двумя цифрами
  if min < 10 then
     buf := buf + '0' + IntToStr(min) + ' : '
  else
     buf := buf + IntToStr(min) + ' : ';

  if sec < 10 then
    buf := buf + '0' + IntToStr(sec)
  else
    buf := buf + IntToStr(sec);

  Form1.Caption := buf;
end;
```

```
// сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  sec := sec — 1;
  if (min = 0) and (sec = 0) then
    // заданный интервал истек
    begin
      Timer1.Enabled := False; // остановить таймер
      UpDown1.Enabled := True;
      UpDown2.Enabled := True;
      Edit1.Enabled := True;
      Edit2.Enabled := True;
      Button1.Caption := 'Пуск';
      ShowMessage('Заданный интервал истек!');
      exit;
    end;

  if (sec = 0) and (min > 0) then
     begin
         sec := 60;
         min := min — 1;
     end;

  ShowTime; // показать, сколько времени осталось
end;

// щелчок на кнопке Пуск/Стоп
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Form1.Timer1.Enabled then
    // таймер работает, надо остановить
    begin
      Timer1.Enabled := False;   // стоп
      Form1.Caption := 'Таймер';
      Button1.Caption := 'Пуск';
      // разрешить ввод интервала
      UpDown1.Enabled:= True;
      UpDown2.Enabled:= True;
      Edit1.Enabled := True;
      Edit2.Enabled := True;
    end
```

```
  else
    // таймер стоит, надо запустить
    begin
      min := UpDown1.Position;
      sec := UpDown2.Position;
      if (sec = 0) and (min = 0) then
      begin
        ShowMessage('Надо задать интервал!');
        exit;
      end;

      Edit1.Enabled := False;
      Edit2.Enabled := False;
      UpDown1.Enabled:= False;
      UpDown2.Enabled:= False;
      Button1.Caption := 'Стоп';
      Timer1.Enabled := True;    // пуск таймера
      ShowTime;
    end;
end;

end.
```

Графика

Общие замечания

Приступая к решению задач этого раздела, необходимо вспомнить:

- Программа может выводить графику на *поверхность* объекта (формы или компонента `Image`), которой соответствует свойство `Canvas`.
- Для того чтобы на поверхности объекта появился графический элемент (линия, окружность, прямоугольник и т. д.) или картинка, необходимо применить к свойству `Canvas` этого объекта соответствующий метод.
- Цвет, стиль и толщину линий, вычерчиваемых методами `Line`, `Ellipse`, `Rectangle` и т. д., определяет свойство `Pen` объекта `Canvas`.

- Цвет закраски внутренних областей геометрических фигур, вычерчиваемых методами `Line`, `Ellipse`, `Rectangle` и т. д., определяет свойство `Brush` объекта `Canvas`.
- Характеристики шрифта текста, выводимого методом `TextOut`, определяет свойство `Font` объекта `Canvas`.
- Основную работу по выводу графики на поверхность формы должна выполнять функция обработки события `OnPaint`.

23. Напишите программу, которая на поверхности формы рисует олимпийский флаг (рис. 1.19).

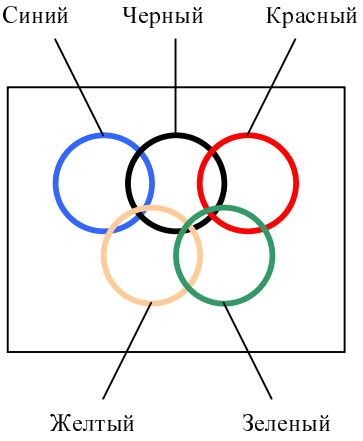


Рис. 1.19. Олимпийский флаг

```
// рисует олимпийский флаг
procedure TForm1.FormPaint(Sender: TObject);
begin
    with Canvas do
    begin
        // полотнище
        Canvas.Pen.Width := 1;
        Canvas.Pen.Color := clBlack;
        Canvas.Brush.Color := clCream;
        Rectangle(30,30,150,115);
```

```
        // кольца
        Pen.Width := 2;
        Brush.Style := bsClear; // область внутри круга
                                // не закрашивать
        Pen.Color := clBlue;
        Ellipse(40,40,80,80);
        Pen.Color := clBlack;
        Ellipse(70,40,110,80);
        Pen.Color := clRed;
        Ellipse(100,40,140,80);
        Pen.Color := clYellow;
        Ellipse(55,65,95,105);
        Pen.Color := clGreen;
        Ellipse(85,65,125,105);
    end;
end;
```

24. Напишите программу, которая на поверхности формы рисует флаг Российской Федерации.

```
// обработка события OnPaint
// процедура рисует флаг Российской Федерации
procedure TForm1.FormPaint(Sender: TObject);
const
    L = 200; // ширина флага (полосы)
    H = 40;  // высота полосы
var
    x,y: integer; // левый верхний угол
begin
    x := 30;
    y := 50;

    with Canvas do
    begin
        // Чтобы у прямоугольников не было
        // границы, цвет границы должен
        // совпадать с цветом закраски

        Brush.Color := clWhite; // цвет закраски
        Pen.Color   := clWhite; // цвет границы
        Rectangle(x,y,x+L,y+H);
```

```
        Brush.Color := clBlue;
        Pen.Color   := clBlue;
        Rectangle(x,y+H,x+L,y+2*H);

        Brush.Color := clRed;
        Pen.Color   := clRed;
        Rectangle(x,y+2*H,x+L,y+3*H);

        // контур
        Pen.Color := clBlack;
        Brush.Style := bsClear; // "прозрачная" кисть
        Rectangle(x,y,x+l,y+h*3);

        Font.Size := 24;
        Font.Name := 'Times New Roman';
        Font.Color := clWhite;
        TextOut(50,200,'Р о с с и я');
        Font.Color := clBlack;
        TextOut(51,201,'Р о с с и я');
    end;
end;

end.
```

25. Напишите программу, в диалоговом окне которой, в точке щелчка кнопкой мыши, вычерчивается контур пятиконечной звезды (рис. 1.20).

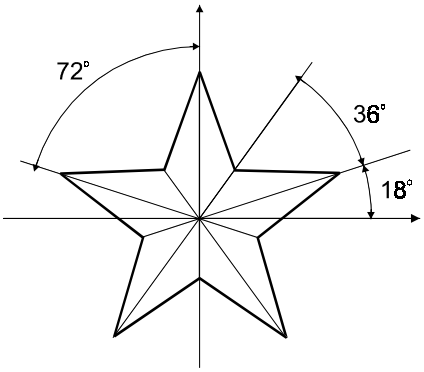


Рис. 1.20. Звезда

```
unit Stars_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,   Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure StarLine(x0,y0,r: integer); // рисует звезду
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// рисует звезду
procedure TForm1.StarLine(x0,y0,r: integer);
    // x0,y0 — координаты центра звезды
    // r — радиус звезды
var
    p : array[1..11] of TPoint; // массив координат лучей и впадин
    a: integer;   // угол между осью OX и прямой, соединяющей
                  // центр звезды и конец луча или впадину
    i: integer;
begin
    a := 18; // строим от правого гор. луча
    for i:=1 to 10 do
       begin
          if (i mod 2 = 0) then
             begin // впадина
               p[i].x := x0+Round(r/3*cos(a*2*pi/360));
               p[i].y:=y0-Round(r/3*sin(a*2*pi/360));
             end
```

```
          else
             begin // луч
               p[i].x:=x0+Round(r*cos(a*2*pi/360));
               p[i].y:=y0-Round(r*sin(a*2*pi/360));
             end;
          a := a+36;
       end;

    p[11].X := p[1].X; // чтобы замкнуть контур звезды
    p[11].Y := p[1].Y;

    Canvas.Polyline(p); // начертить контур звезды
end;

// нажатие кнопки мыши
procedure TForm1.FormMouseDown(Sender: TObject;
                               Button: TMouseButton;
                               Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft  // нажата левая кнопка?
     then Canvas.Pen.Color := clBlack
     else Canvas.Pen.Color := clRed;
  StarLine(x, y, 30);
end;

end.
```

26. Напишите программу, по поверхности окна которой перемещается случайным образом (прыгает) изображение веселой рожицы, на котором пользователь может сделать щелчок кнопкой мыши. Программа должна завершить работу после того, как пользователь сделает 10 щелчков кнопкой мыши. Рекомендуемый вид окна в начале работы программы приведен на рис. 1.21.

```
unit tir_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
```

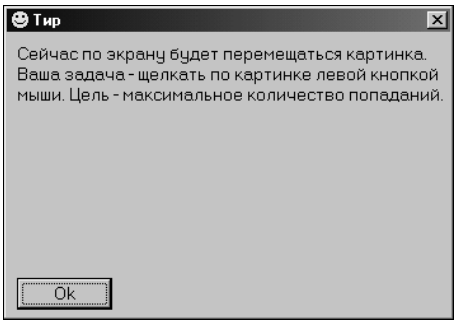


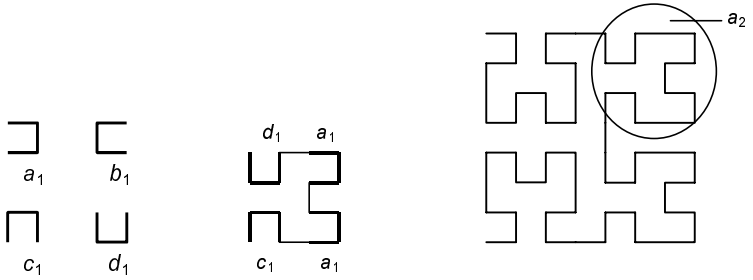
Рис. 1.21. Окно программы **Тир**

```
type
  TForm1 = class(TForm)
    Timer: TTimer;
    Label1: TLabel;
    Button1: TButton;
    procedure TimerTimer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }

    { ****************************
      объявление процедур помещено сюда,
      чтобы процедуры имели прямой доступ
      к форме, на которой они рисуют
      **************************** }

    procedure PaintFace(x,y: integer); // рисует рожицу
    procedure EraseFace(x,y: integer); // стирает рожицу
  end;

var
  Form1: TForm1;
  fx,fy: integer;  // координаты рожицы
  n: integer;      // количество щелчков кнопкой мыши
  p: integer;      // количество попаданий
```

```
implementation

// рисует рожицу
procedure TForm1.PaintFace(x,y: integer);
begin
    Canvas.Pen.Color := clBlack;     // цвет линий
    Canvas.Brush.Color := clYellow;  // цвет закраски
    // рисуем рожицу
    Canvas.Ellipse(x,y,x+30,Y+30);       // лицо
    Canvas.Ellipse(x+9,y+10,x+11,y+13);  // левый глаз
    Canvas.Ellipse(x+19,y+10,x+21,y+13); // правый глаз
    Canvas.Arc(x+4,y+4,x+26,y+26,x,y+20,x+30,y+20); // улыбка
end;

// стирает рожицу
procedure TForm1.EraseFace(x,y: integer);
begin
   // зададим цвет границы и цвет закраски,
   // совпадающий с цветом формы. По умолчанию
   // цвет формы — clBtnFace (см. в Object Inspector)
   Canvas.Pen.Color := clBtnFace;   // цвет окружности
   Canvas.Brush.Color := clBtnFace; // цвет закраски
   Canvas.Ellipse(x,y,x+30,y+30);
end;

{$R *.dfm}

procedure TForm1.TimerTimer(Sender: TObject);
begin
    EraseFace(fx,fy);
    // новое положение рожицы
    fx:= Random(ClientWidth-30);  // 30 — это диаметр рожицы
    fy:= Random(ClientHeight-30);
    PaintFace(fx,fy);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    // исходное положение рожицы
    fx:=100;
    fy:=100;
    Randomize;  // инициализация генератора
                // случайных чисел
end;
```

```
// нажатие клавиши мыши
procedure TForm1.FormMouseDown(Sender: TObject; Button:
TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  inc(n);  // кол-во щелчков

  if (x > fx) and (x < fx+30) and
     (y > fy) and (y < fy+30)
  then begin
       // щелчок по рожице
       inc(p);
       end;
  if n = 10 then
    begin
        // игра закончена
        Timer.Enabled := False; // остановить таймер
        ShowMessage('Выстрелов: 10. Попаданий: ' +
                 IntToStr(p)+'.');

        EraseFace(fx,fy);
        Label1.Visible := True;
        Button1.Visible := True;
        // теперь кнопка и сообщение снова видны
    end;
end;

// щелчок на кнопке Ok
procedure TForm1.Button1Click(Sender: TObject);
begin
    Label1.Visible := False;  // скрыть сообщение
    Button1.Visible := False; // скрыть кнопку
    Timer.Enabled := True;    // пуск таймера
end;

end.
```

27. Напишите программу, которая на поверхности формы вычерчивает кривую Гильберта (рис. 1.22). Пример кривой Гильберта пятого порядка приведен на рис. 1.23.



Кривые первого порядка получаются путем соединения кривых нулевого порядка (точек)

Кривая второго порядка (a_2) образуется путем соединения четырех кривых первого порядка: d_1, a_1, a_1 и c_1

Кривая третьего порядка (a_3) получается путем соединения четырех кривых второго порядка: d_2, a_2, a_2 и c_2

Рис. 1.22. Кривые Гильберта

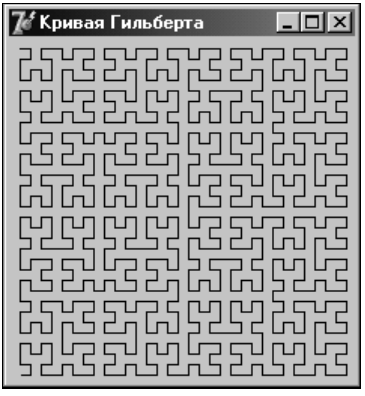


Рис. 1.23. Кривая Гильберта пятого порядка

```
var
  p: integer = 5;  // порядок кривой
  u: integer = 7;  // длина штриха

{ Кривая Гильберта состоит из четырех соединенных
  прямыми элементов: a, b, c и d.
  Каждый элемент строит соответствующая процедура. }
procedure a(i:integer; canvas: TCanvas); forward;
procedure b(i:integer; canvas: TCanvas); forward;
```

```
procedure c(i:integer; canvas: TCanvas); forward;
procedure d(i:integer; canvas: TCanvas); forward;

// Элементы кривой
procedure a(i: integer; canvas: TCanvas);
  begin
    if i > 0 then begin
      d(i-1, canvas);
      canvas.LineTo(canvas.PenPos.X+u, canvas.PenPos.Y);
      a(i-1, canvas);
canvas.LineTo(canvas.PenPos.X,canvas.PenPos.Y+u);
      a(i-1, canvas);
      canvas.LineTo(canvas.PenPos.X-u, canvas.PenPos.Y);
      c(i-1, canvas);
    end;
  end;

procedure b(i: integer; canvas: TCanvas);
  begin
   if i > 0 then
   begin
      c(i-1, canvas);
      canvas.LineTo(canvas.PenPos.X-u, canvas.PenPos.Y);
      b(i-1, canvas);
      canvas.LineTo(canvas.PenPos.X,canvas.PenPos.Y-u);
      b(i-1, canvas);
      canvas.LineTo(canvas.PenPos.X+u,canvas.PenPos.Y);
      d(i-1, canvas);
   end;
  end;

procedure c(i: integer; canvas: TCanvas);
  begin
   if i > 0 then
   begin
      b(i-1, canvas);
      canvas.LineTo(canvas.PenPos.X, canvas.PenPos.Y-u);
      c(i-1, canvas);
      canvas.LineTo(canvas.PenPos.X-u, canvas.PenPos.Y);
      c(i-1, canvas);
      canvas.LineTo(canvas.PenPos.X,canvas.PenPos.Y+u);
      a(i-1, canvas);
   end;
  end;
```

```
procedure d(i: integer; canvas: TCanvas);
  begin
   if i > 0 then
   begin
      a(i-1, canvas);
      canvas.LineTo(canvas.PenPos.X,canvas.PenPos.Y+u);
      d(i-1, canvas);
      canvas.LineTo(canvas.PenPos.X+u,canvas.PenPos.Y);
      d(i-1, canvas);
      canvas.LineTo(canvas.PenPos.X,canvas.PenPos.Y-u);
      b(i-1, canvas);
   end;
  end;

// обработка события OnPaint
procedure TForm1.FormPaint(Sender: TObject);
  begin
    Form1.Canvas.MoveTo(u,u);
    a(5,Form1.Canvas); // вычертить кривую Гильберта
  end;
```

28. Напишите программу, которая на поверхность формы выводит изображение оцифрованной координатной сетки (рис. 1.24).

Рис. 1.24. Координатная сетка

```
// обработка события OnPaint
procedure TForm1.FormPaint(Sender: TObject);
var
 x0,y0:integer; // координаты начала координатных осей
 dx,dy:integer; // шаг координатной сетки (в пикселах)
 h,w:integer;   // высота и ширина области вывода координатной
                // сетки
 x,y:integer;

 lx,ly:real;     // метки (оцифровка) линий сетки по X и Y
 dlx,dly:real;   // шаг меток (оцифровки) линий сетки по X и Y
 cross:integer;  // счетчик не оцифрованных линий сетки
 dcross:integer; // количество не оцифрованных линий
                 // между оцифрованными
begin
 x0:=30; y0:=220; // оси начинаются в точке (40,250)
 dx:=40; dy:=40;  // шаг координатной сетки 40 пикселов
 dcross:=1;       // помечать линии сетки X: 1 — каждую;
                  //                         2 — через одну;
                  //                         3 — через две;
 dlx:=0.5;        // шаг меток оси X
 dly:=1.0;        // шаг меток оси Y, метками будут: 1, 2, 3
                  // и т. д.

 h:=200;
 w:=300;

 with form1.Canvas do
 begin
   cross:=dcross;
   MoveTo(x0,y0); LineTo(x0,y0-h);  // ось X
   MoveTo(x0,y0); LineTo(x0+w,y0);  // ось Y

   // засечки, сетка и оцифровка по оси X
   x:=x0+dx;
   lx:=dlx;
   repeat
     MoveTo(x,y0-3);LineTo(x,y0+3);  // засечка
     cross:=cross-1;
     if cross = 0 then   //оцифровка
     begin
       TextOut(x-8,y0+5,FloatToStr(lx));
       cross:=dcross;
     end;
```

```
      Pen.Style:=psDot;
      MoveTo(x,y0-3);LineTo(x,y0-h);  // линия сетки
      Pen.Style:=psSolid;
      lx:=lx+dlx;
      x:=x+dx;
    until (x>x0+w);

    // засечки, сетка и оцифровка по оси Y
    y:=y0-dy;
    ly:=dly;
    repeat
      MoveTo(x0-3,y);LineTo(x0+3,y);    // засечка
      TextOut(x0-20,y,FloatToStr(ly));  // оцифровка
      Pen.Style:=psDot;
      MoveTo(x0+3,y); LineTo(x0+w,y);    // линия сетки
      Pen.Style:=psSolid;
      y:=y-dy;
      ly:=ly+dly;
    until (y<y0-h);
  end;
end;
```

29. Напишите программу, которая на поверхности формы вычерчивает график функции, например $2\sin(x)\,e^{x/5}$. Вид окна во время работы программы приведен на рис. 1.25.

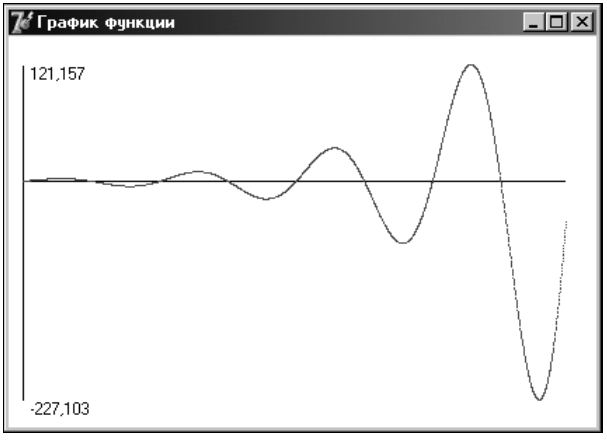


Рис. 1.25. Окно программы **График функции**

```
// Функция, график которой надо построить
 Function f(x:real):real;
 begin
    f:=2*Sin(x)*exp(x/5);
 end;

// строит график функции
procedure GrOfFunc;
var
 x1,x2:real;    // границы изменения аргумента функции
 y1,y2:real;    // границы изменения значения функции
 x:real;        // аргумент функции
 y:real;        // значение функции в точке x
 dx:real;       // приращение аргумента
 l,b:integer;   // левый нижний угол области вывода графика
 w,h:integer;   // ширина и высота области вывода графика
 mx,my:real;    // масштаб по осям X и Y
 x0,y0:integer; // точка — начало координат

begin
 // область вывода графика
 l:=10;                      // X — координата левого верхнего
                             // угла
 b:=Form1.ClientHeight-20;   // Y — координата левого верхнего
                             // угла
 h:=Form1.ClientHeight-40;   // высота
 w:=Form1.Width-40;          // шфрина

 x1:=0;     // нижняя граница диапазона аргумента
 x2:=25;    // верхняя граница диапазона аргумента
 dx:=0.01;  // шаг аргумента

 // найдем максимальное и минимальное значения
 // функции на отрезке [x1,x2]
 y1:=f(x1); // минимум
 y2:=f(x1); // максимум
 x:=x1;
 repeat
   y := f(x);
   if y < y1 then y1:=y;
   if y > y2 then y2:=y;
   x:=x+dx;
 until (x>=x2);
```

```
 // вычислим масштаб
 my:=h/abs(y2-y1);  // масштаб по оси Y
 mx:=w/abs(x2-x1);  // масштаб по оси X

 // оси
 x0:=l;
 y0:=b-Abs(Round(y1*my));

 with form1.Canvas do
 begin
   // оси
   MoveTo(l,b);LineTo(l,b-h);
   MoveTo(x0,y0);LineTo(x0+w,y0);
   TextOut(l+5,b-h,FloatToStrF(y2,ffGeneral,6,3));
   TextOut(l+5,b,FloatToStrF(y1,ffGeneral,6,3));
   // построение графика
   x:=x1;
   repeat
     y:=f(x);
     Pixels[x0+Round(x*mx),y0-Round(y*my)]:=clRed;
     x:=x+dx;
   until (x>=x2);
 end;
end;


procedure TForm1.FormPaint(Sender: TObject);
begin
  GrOfFunc;
end;

// изменился размер окна программы
procedure TForm1.FormResize(Sender: TObject);
begin
  // очистить форму
  form1.Canvas.FillRect(Rect(0,0,ClientWidth,ClientHeight));
  // построить график
  GrOfFunc;
end;
```

30. Напишите программу, которая выводит на экран гистограмму, например, результатов контрольной работы. Пример окна программы во время ее работы приведен на рис. 1.26.

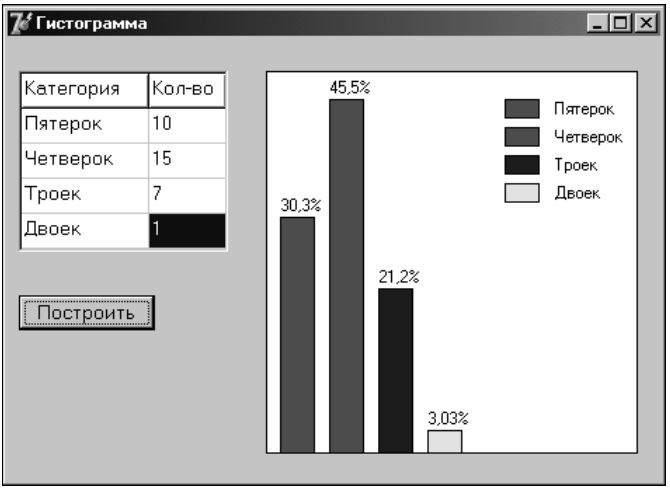


Рис. 1.26. Окно программы **Гистограмма**

```
implementation

{$R *.dfm}

const
  NR = 4; // кол-во строк в таблице
var
  n: array[1..NR] of real;     // значения категорий
  p: array[1..NR] of real;     // процент категории в общей сумме
  h: array[1..NR] of integer; // высота столбиков диаграммы

  // цвет столбиков диаграммы
  BarColor: array[1..4] of TColor = (clRed,clGreen,clBlue,clYellow);

// ввод и обработка
// если исходные данные введены, то Obr = TRUE
function Obr : boolean;
var
  sum: real;  // сумма категорий
  m: integer; // номер категории, имеющей максимальное значение
  i: integer;
begin
  obr := FALSE; // пусть исх. данные не введены
```

```
  // скопируем содержимое второго столбца
  // в массив исходных данных
  for i:=1 to NR do
      // здесь возможно исключение (ошибка) преобразования,
      // если пользователь не ввел данные
  begin
      try
          n[i] := StrToFloat(Form1.StringGrid1.Cells[1,i]);
       except
          on EConvertError do
          begin
              ShowMessage('Надо ввести данные во все' + #13 +
              'ячейки второй колонки.');
              exit;
          end;
      end;
  end;
  // вычислим сумму категорий (эл-тов второго столбца)
  sum := 0;
  for i:=1 to NR do
      sum := sum + n[i];

  // вычислим процент каждой категории
  for i:=1 to NR do
      p[i] := n[i] / sum;

   // определим категорию с максимальным значением
   m := 1;
   for i := 2 to NR do
      if n[i] > n[m] then m:=i;

  // пусть максимальному значению соответствует
  // столбик высотой в Image1.Height-20 пикселов
  // вычислим высоту остальных столбиков
  for i :=1 to NR do
      h[i] := Round((Form1.Image1.Height — 20) *
              n[i]/n[m]);

  // все готово
  // можно строить диаграмму
  obr := TRUE;
end;
```

```
// диаграмма
procedure diagr;
const
  WR = 25; // ширина столбика
  DR = 10; // расстояние между столбиками
var
  x,y: integer; // левый нижний угол столбика

  i: integer;
begin
  with Form1.Image1 do
  begin
    x:=10;
    y:=Height;
    Canvas.Brush.Color := clWindow;
    Canvas.Rectangle(0,0,Width,Height);
    // *** рисуем столбики ***
    for i:=1 to 4 do
    begin
      Canvas.Brush.Color := BarColor[i];  // цвет столбика
      Canvas.Rectangle(x,y,x+WR,y-h[i]);  // столбик
      Canvas.Brush.Color := clWindow;     // чтобы область
                                          // за текстом
                                          // не была окрашена
      // подпись данных (над столбиком)
      Canvas.TextOut(x,y-h[i]-15,
             FloatToStrF(p[i]*100,ffGeneral,3,2)+'%');
      x := x + WR + DR;
    end;

    // *** легенда ***
    // здесь x — координата левой границы
    // последнего столбика
    x := x + 20;
    y:=20;        // 20 пикселов от верхнего края Image1
    for i:=1 to 4 do
    begin
      Canvas.Brush.Color := BarColor[i]; // цвет прямоугольника
                                         // легенды
      Canvas.Rectangle(x,y,x+25,y+14);  // прямоугольник легенды
      Canvas.Brush.Color := clWindow;
      Canvas.TextOut(x+WR+10,y,
             Form1.StringGrid1.Cells[0,i]);
```

```
      y := y + 20;
    end;
  end; // with Form1.Image1
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  // определим заголовки колонок
  StringGrid1.Cells[0,0] := 'Категория';
  StringGrid1.Cells[1,0] := 'Кол-во';
  StringGrid1.Width :=
        StringGrid1.ColWidths[0] + StringGrid1.ColWidths[1]+5;

end;

// нажатие клавиши в ячейке таблицы (компонента StringGrid)
// в результате нажатия клавиши <Enter> курсор переходит
// в следующую ячейку
procedure TForm1.StringGrid1KeyPress(Sender: TObject;
                                     var Key: Char);
begin
   // Col, Row — номер колонки и строки,
   // в которой находится курсор (нумерация с нуля).
   // ColCount и RowCount — кол-во колонок и строк

   if Key = #13 then
   begin
  // нажата клавиша <Enter>
  if StringGrid1.Col < StringGrid1.ColCount — 1
     then
        // ячейка не в последнем столбце
        StringGrid1.Col := StringGrid1.Col + 1 // к след. столбцу
     else
        // ячейка в последнем столбце
        if ( StringGrid1.Row < StringGrid1.RowCount — 1) then
          begin
            // в первый столбец следующей строки
            StringGrid1.Col :=0;
            StringGrid1.Row := StringGrid1.Row +1;
          end
        else Button1.SetFocus;
    exit;
  end;
```

```
  // во вторую колонку разрешается вводить
  // только числа
  if StringGrid1.Col = 1 then
      // клавиша нажата в ячейке
      // второй колонки
      case Key of
      '0'..'9',#8:;
      '.',',':
         begin
              Key := DecimalSeparator;
              if Pos(DecimalSeparator,
                 StringGrid1.Cells[StringGrid1.Row,
                 StringGrid1.Col]) <> 0
              then Key := Char(0);
         end;
      else Key := Char(0);
      end;
end;

// щелчок на кнопке Построить
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Obr  // исходные данные введены
      then diagr; // строим диаграмму
end;

end.
```

31. Напишите программу, которая выводит на поверхность формы круговую диаграмму, отражающую, например, товарооборот книжного магазина (рис. 1.27).

32. Напишите программу, которая в отдельном окне строит график функции. Функция задается таблицей, первая строка которой содержит значения аргументов, вторая — соответствующие значения функции. Рекомендуемый вид главной формы приведен на рис. 1.28. Вид формы, на поверхности которой вычерчивается график, представлен на рис. 1.29.

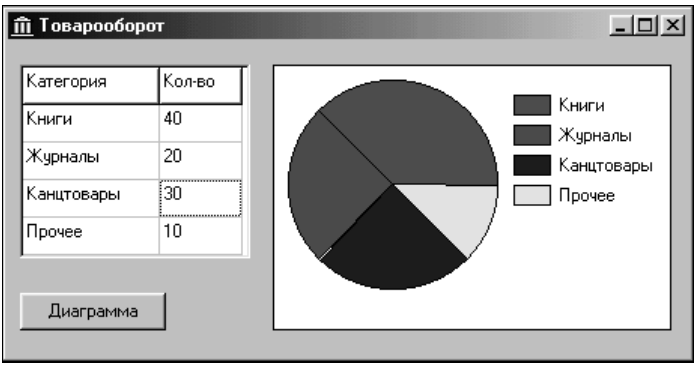


Рис. 1.27. Круговая диаграмма, отражающая товарооборот книжного магазина

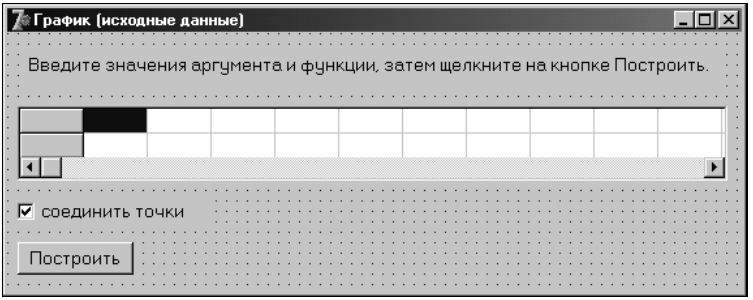


Рис. 1.28. Главная форма приложения **График**

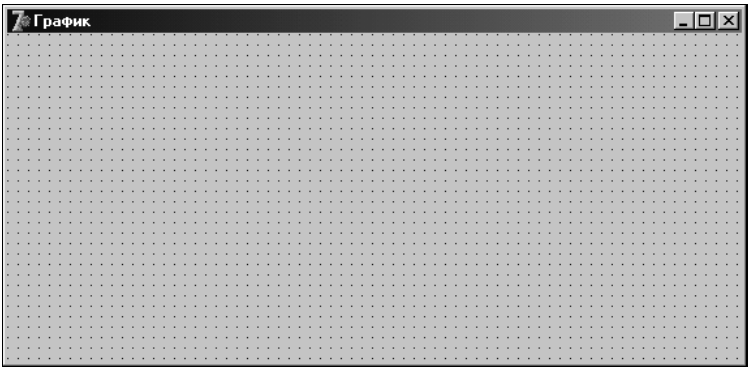


Рис. 1.29. Форма, на поверхности которой будет построен график

```
// ******************************
// Модуль главной формы приложения

unit grafik01;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Dialogs, StdCtrls, Grids;

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    Label1: TLabel;
    Button1: TButton;
    CheckBox1: TCheckBox;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

// Процедуру gr вызывает процедура обработки
// события onPaint для формы Form2, на поверхности
// которой вычерчивается график и которая
// находится в другом модуле. Поэтому объявление
// функции надо поместить в раздел Interface.
procedure gr; // чтобы процедуру можно было вызвать
              // из другого модуля
implementation

uses grafik02;

{$R *.dfm}

{ Во время создания формы установить
  свойства компонента StringGrid:
  FixedRows := 0;
  RowCount := 2;
```

```
  Options.goEditing := True;
  Options.goTab := True;
}

const
    COLCOUNT = 15;
var
    // аргументы и значения функции
    x: array[1..COLCOUNT] of real;
    y: array[1..COLCOUNT] of real;

// строит график по содержимому массивов x,y
procedure gr;
var
    i: integer;

    x1,x2:real;     // границы изменения аргумента функции
    y1,y2:real;     // границы изменения значения функции
    l,b:integer;    // левый нижний угол области вывода графика
    w,h:integer;    // ширина и высота области вывода графика
    mx,my:real;     // масштаб по осям X и Y
    x0,y0:integer;  // точка пересечения координатных осей
    px,py: integer; // координаты точки графика на поверхности
                    // формы
    np: integer;    // кол-во точек графика

begin
 Form2.Canvas.Rectangle(0,0,Form2.ClientWidth,Form2.ClientHeight);
 // область вывода графика
 l:=10;                    // X — координата левого верхнего угла
 b:=Form2.ClientHeight-20; // Y — координата левого нижнего угла
 h:=Form2.ClientHeight-40;  // высота
 w:=Form2.Width-40;         // ширина

 // определим границы изменения аргумента
 // и количество точек (элементы массива X
 // должны образовывать возрастающую последовательность)
 x1:=x[1];
 i:=1;
 while (x[i+1] > x[i]) and (i < COLCOUNT) do
          i:= i+1;

 x2:= x[i]; //x[COLCOUNT-1];
 np:=i; // количество точек
```

```
if np < 2 then begin
   ShowMessage('Количество точек графика не может быть меньше
                двух');
   exit;
end;

// найдем максимальное и минимальное значения функции
y1:=0;  //y[1]; // минимум
y2:=0;  //y[1]; // максимум
for i:=1 to np do
begin
  if y[i] < y1 then y1:=y[i];
  if y[i] > y2 then y2:=y[i];
end;

// вычислим масштаб
my:=h/abs(y2-y1);  // масштаб по оси Y
mx:=w/abs(x2-x1);  // масштаб по оси X

// точка пересечения координатных осей
x0:=l+Abs(Round(x1*mx));
y0:=b-Abs(Round(y1*my));

with form2.Canvas do
begin
  MoveTo(x0,b);LineTo(x0,b-h); // ось Y
  MoveTo(l,y0);LineTo(l+w,y0); // ось X

  TextOut(l+5,b-h,FloatToStrF(y2,ffGeneral,6,3));
  TextOut(l+2,b+2,FloatToStrF(y1,ffGeneral,6,3));

  // построить график
  for i:=1 to np do
  begin
     px := x0+Round(x[i]*mx);
     py := y0-Round(y[i]*my);

     Form2.Canvas.Pen.Color := clRed;
     Rectangle(px-2,py-2,px+2,py+2); // маркер

     if Form1.CheckBox1.Checked then
       // соединительная линия
       if i = 1
         then MoveTo(px,py)
         else LineTo(px,py);
```

```
      Form2.Canvas.Pen.Color := clBlack;

      // значение функции и аргумент
      TextOut(px-5,py-15,FloatToStr(y[i]));
      TextOut(px-5,y0+2, FloatToStr(x[i]));
      MoveTo(px,py); // вернуть перо в точку px, py,
                     // т. к. TextOut меняет положение пера
   end;
 end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
    i: integer;
begin
    StringGrid1.ColCount := COLCOUNT;
    StringGrid1.Cells[0,0] := '  X';
    StringGrid1.Cells[0,1] := '  Y';
    // отладка: заполним таблицу
    for i:=1 to COLCOUNT do
    begin
        StringGrid1.Cells[i,0] := IntToStr(i-1);
        StringGrid1.Cells[i,1] := IntToStr(i-1);
    end;
end;

// щелчок на кнопке Построить
procedure TForm1.Button1Click(Sender: TObject);
var
    i: integer;
begin
    // перепишем содержимое таблицы в массивы
    // аргументов и значений функции
    for i:=1 to COLCOUNT-1 do
    begin
        x[i] := StrToFloat(Form1.StringGrid1.Cells[i,0]);
        y[i] := StrToFloat(Form1.StringGrid1.Cells[i,1]);
    end;

    // функцию вывода графика вызывает процедура
    // обработки события OnPaint для Form2
    if not Form2.Showing
        then Form2.Show     // отобразить окно Form2
```

```
        else Form2.Repaint;
end;

end.

// ***********************************
// Модуль формы, на поверхности которой
// вычерчивается график

 unit grafik02;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs,

  grafik01;  // *** эта ссылка вставлена вручную!

type
  TForm2 = class(TForm)
    procedure FormPaint(Sender: TObject);
    procedure FormResize(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

// событиеOnPaint происходит при первом появлении
// окна на экране и тогда, когда окно
// или его часть надо перерисовать
procedure TForm2.FormPaint(Sender: TObject);
begin
    gr;  // вывести график
end;

// событие OnResize происходит, если
// пользователь изменил размер окна
```

```
procedure TForm2.FormResize(Sender: TObject);
begin
    gr; // вывести график
end;

end.
```

33. Напишите программу, по поверхности окна которой перемещается графический объект, например кораблик (рис. 1.30). Вид окна программы приведен на рис. 1.31.

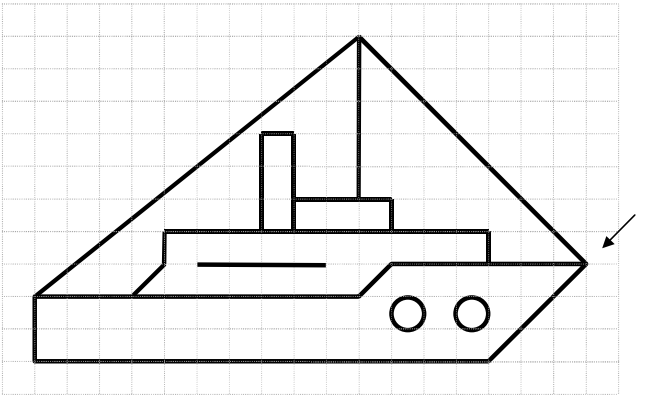


Рис. 1.30. Кораблик

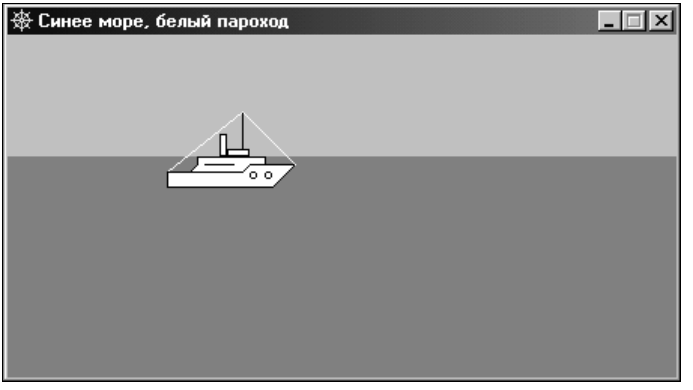


Рис. 1.31. Простая мультипликация — кораблик, плывущий по морю

```
unit ship_;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    procedure Timer1Timer(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

var
    x,y: integer; // координаты корабля (базовой точки)

// вычерчивает на поверхности формы кораблик
// или стирает его
procedure Parohod(x,y: integer; mode: boolean);
     // x,y — координаты базовой точки кораблика
     // mode: True — нарисовать, False — стереть
const
  { Базовые точки кораблика находятся в узлах сетки,
    шаг которой определяет размер кораблика }
  dx = 5; // шаг сетки по X
  dy = 5; // шаг сетки по Y
var
  // корпус и надстройку будем рисовать
  // при помощи метода Polygon
  p1: array[1..7]of TPoint; // координаты точек корпуса
  p2: array[1..8]of TPoint; // координаты точек надстройки
```

```
  pc,bc: TColor; // текущий цвет карандаша и кисти

begin
  if not mode then
  begin
    // стереть
    Form1.Canvas.Brush.Color := clNavy;
    Form1.Canvas.Pen.Color := clNavy;
    Form1.Canvas.Rectangle(x,y+1,x+17*dx,y-10*dy);
    Form1.Canvas.Brush.Color := clNavy;
    // небо
    if y-10*dy < 80 then
    begin
        // конец мачты на фоне неба
        Form1.Canvas.Pen.Color := clSkyBlue;
        Form1.Canvas.Brush.Color := clSkyBlue;
        Form1.Canvas.Rectangle(x,y-10*dy,x+17*dx,80);
    end;
    exit;
  end;

  // рисуем
  with Form1.Canvas do
  begin
    pc := Pen.Color;    // сохраним текущий цвет карандаша
    bc := Brush.Color; // и кисти

    Pen.Color := clBlack;    // установим нужный цвет
    Brush.Color := clWhite;
    // рисуем ...
    // корпус
    p1[1].X := x;      p1[1].y := y;
    p1[2].X := x;      p1[2].Y := y-2*dy;
    p1[3].X :=x+10*dx; p1[3].Y := y-2*dy;
    p1[4].X :=x+11*dx; p1[4].Y := y-3*dy;
    p1[5].X :=x+17*dx; p1[5].Y := y-3*dy;
    p1[6].X :=x+14*dx; p1[6].Y := y;
    p1[7].X :=x;       p1[7].Y := y;
    Polygon(p1);
    // надстройка
    p2[1].X := x+3*dx;  p2[1].Y := y-2*dy;
    p2[2].X := x+4*dx;  p2[2].Y := y-3*dy;
```

```
    p2[3].X := x+4*dx;  p2[3].Y := y-4*dy;
    p2[4].X := x+13*dx; p2[4].Y := y-4*dy;
    p2[5].X := x+13*dx; p2[5].Y := y-3*dy;
    p2[6].X := x+11*dx; p2[6].Y := y-3*dy;
    p2[7].X := x+10*dx; p2[7].Y := y-2*dy;
    p2[8].X := x+3*dx;  p2[8].Y := y-2*dy;
    Polygon(p2);
    MoveTo(x+5*dx,y-3*dy);
    LineTo(x+9*dx,y-3*dy);
    // капитанский мостик
    Rectangle(x+8*dx,y-4*dy,x+11*dx,y-5*dy);
    // труба
    Rectangle(x+7*dx,y-4*dy,x+8*dx,y-7*dy);
    // иллюминаторы
    Ellipse(x+11*dx,y-2*dy,x+12*dx,y-1*dy);
    Ellipse(x+13*dx,y-2*dy,x+14*dx,y-1*dy);
    // мачта
    MoveTo(x+10*dx,y-5*dy);
    LineTo(x+10*dx,y-10*dy);
    // оснастка
    Pen.Color := clWhite;
    MoveTo(x+17*dx,y-3*dy);
    LineTo(x+10*dx,y-10*dy);
    LineTo(x,y-2*dy);
    Pen.Color := pc; // восстановим старый цвет карандаша
  end;
end;

// обработка сигнала таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
   Parohod(x,y, False); // стереть рисунок
   if x < Form1.ClientWidth
      then x := x+2
      else begin // новый рейс
        x := 0;
        y := Random(50) + 100;
      end;
   Parohod(x,y,True); // нарисовать в новой точке
end;
```

```
procedure TForm1.FormPaint(Sender: TObject);
begin
    // небо
    Canvas.Brush.Color := clSkyBlue;
    Canvas.Pen.Color   := clSkyBlue;
    Canvas.Rectangle(0,0,ClientWidth,80);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    x:=0; y:=80;           // исходное положение парохода
    Form1.Color:=clNavy;   // цвет моря
    Timer1.Interval := 50; // сигнал таймера каждые 50 мсек
end;

end.
```

34. Напишите программу, которая на поверхность формы выводит изображение идущих часов с часовой, минутной и секундной стрелками. Окно и форма приложения приведены на рис. 1.32.

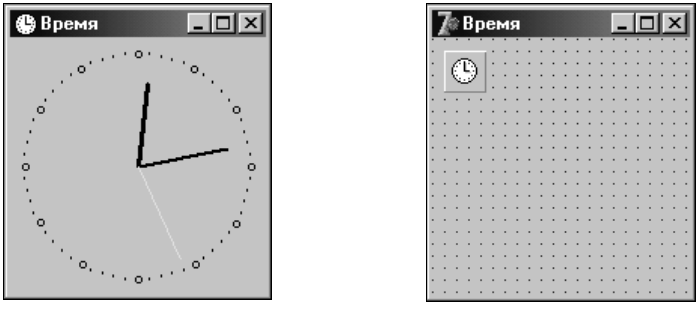


Рис. 1.32. Окно и форма программы **Часы**

```
unit clock_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, ExtCtrls;
```

```
type
  TForm1 = class(TForm)
    Timer1: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);

    // эти объявления вставлены вручную
    procedure Vector(x0,y0,a,l: integer);
    procedure DrawClock;

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.dfm}

uses
   DateUtils; // для доступа к SecondOf,
              // MinuteOf и HourOf

const
  R = 75;    // радиус циферблата часов

var
 x0,y0: integer;          // центр циферблата
 ahr,amin,asec: integer; // положение стрелок (угол)

// инициализация формы
procedure TForm1.FormCreate(Sender: TObject);
var
    t: TDateTime;
begin
  // зададим размер формы
  // в соответствии с размером циферблата
  ClientHeight := (R+10)*2;
  ClientWidth  := (R+10)*2;
```

```
  x0 := R+10;
  y0 := R+10;

  t := Now();

  // положение стрелок
  ahr :=  90 — HourOf(t)*30-(MinuteOf(Today)div 12)*6;
  amin := 90 — MinuteOf(t)*6;
  asec := 90 — SecondOf(Today)*6;

  Timer1.Interval := 1000; // период сигнала от таймера 1 сек
  Timer1.Enabled := True;  // пуск таймера
end;

// вычерчивает вектор заданной длины из точки (x0,y0)
procedure TForm1.Vector(x0,y0: integer; a, l: integer);
    // x0,y0 — начало вектора
    // a — угол между осью x и вектором
    // l — длина вектора
const
  GRAD = 0.0174532;   // коэффициент пересчета угла из градусов
                      // в радианы
var
  x,y: integer;       // координаты конца вектора
begin
  Canvas.MoveTo(x0,y0);
  x := Round(x0 + l*cos(a*GRAD));
  y := Round(y0 — l*sin(a*GRAD));
  Canvas.LineTo(x,y);
end;

// рисует стрелки
procedure TForm1.DrawClock;
var
    t: TDateTime;
begin
  // шаг секундной и минутной стрелок 6 градусов,
  // часовой — 30.

  // стереть изображение стрелок
  Canvas.Pen.Color := clBtnFace;
  Canvas.Pen.Width :=3;
  // часовую
  Vector(x0,y0, ahr, R-20);
```

```
  // минутную
  Vector(x0,y0, amin, R-15);
  // секундную
  Vector(x0,y0, asec, R-7);

  t := Now();

  // новое положение стрелок
  ahr :=  90 — HourOf(t)*30-(MinuteOf(t)div 12)*6;
  amin := 90 — MinuteOf(t)*6;
  asec := 90 — SecondOf(t)*6;

  // нарисовать стрелки
  // часовая стрелка
  Canvas.Pen.Width := 3;
  Canvas.Pen.Color := clBlack;
  Vector(x0,y0, ahr, R-20);

  // минутная стрелка
  Canvas.Pen.Width := 2;
  Canvas.Pen.Color := clBlack;
  Vector(x0,y0, amin, R-15);

  // секундная стрелка
  Canvas.Pen.Width := 1;
  Canvas.Pen.Color := clYellow;
  Vector(x0,y0, asec, R-7);
end;

// прорисовка циферблата и начальных стрелок
procedure TForm1.FormPaint(Sender: TObject);
var
    x,y: integer;    // координаты маркера на циферблате
    a: integer;      // угол между OX и прямой (x0,y0) (x,y)

    pc: TColor;      // цвет карандаша
    pw: integer;     // ширина карандаша
begin
  pc := Canvas.Pen.Color;
  pw := Canvas.Pen.Width;

  Canvas.Pen.Width := 1;
  Canvas.Pen.Color := clBlack;

  a:=0; // метки ставим от 3-х часов, против часовой стрелки
```

```
  // циферблат
  while a < 360 do
  begin
    x:=x0+Round( R * cos(a*2*pi/360));
    y:=x0-Round( R * sin(a*2*pi/360));
    Form1.Canvas.MoveTo(x,y);
    if (a mod 30) = 0
        then Canvas.Ellipse(x-2,y-2,x+3,y+3)
        else Canvas.Ellipse(x-1,y-1,x+1,y+1);
    a:=a+6; // 1 минута — 6 градусов
  end;
  // восстановить карандаш-кисть
  Canvas.Pen.Width := pw;
  Canvas.Pen.Color := pc;

  DrawClock;
end;

// прорисовка текущих положений стрелок часов
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    DrawClock;
end;

end.
```

35. Напишите программу, которая в диалоговом окне выводит изображение идущих часов с часовой, минутной и секундной стрелками (рис. 1.33).

Рис. 1.33. Окно программы **Часы**

```
unit clock2_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, ExtCtrls;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);

    // эти объявления вставлены вручную
    procedure Vector(x0,y0,a,l: integer);
    procedure DrawClock;

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.dfm}

uses
   DateUtils; // для доступа к SecondOf,
              // MinuteOf и HourOf

const
  R = 75;    // радиус циферблата часов

var
 x0,y0: integer;          // центр циферблата
 ahr,amin,asec: integer; // положение стрелок (угол)

// инициализация формы
procedure TForm1.FormCreate(Sender: TObject);
var
    t: TDateTime;
```

```
begin
  // зададим размер формы
  // в соответствии с размером циферблата
  ClientHeight := (R+30)*2;
  ClientWidth  := (R+30)*2;
  x0 := R+30;
  y0 := R+30;

  t := Now();

  // положение стрелок
  ahr :=  90 — HourOf(t)*30-(MinuteOf(Today)div 12)*6;
  amin := 90 — MinuteOf(t)*6;
  asec := 90 — SecondOf(Today)*6;

  Timer1.Interval := 1000; // период сигналов от таймера 1 сек
  Timer1.Enabled := True;  // пуск таймера
end;

// вычерчивает вектор заданной длины из точки (x0,y0)
procedure TForm1.Vector(x0,y0: integer; a, l: integer);
    // x0,y0 — начало вектора
    // a — угол между осью x и вектором
    // l — длина вектора
const
  GRAD = 0.0174532;   // коэффициент пересчета угла из градусов
                      // в радианы
var
  x,y: integer;       // координаты конца вектора
begin
  Canvas.MoveTo(x0,y0);
  x := Round(x0 + l*cos(a*GRAD));
  y := Round(y0 — l*sin(a*GRAD));
  Canvas.LineTo(x,y);
end;

// рисует стрелки
procedure TForm1.DrawClock;
var
    t: TDateTime;
begin
  // шаг секундной и минутной стрелок 6 градусов,
  // часовой — 30.
```

```
  // стереть изображение стрелок
  Canvas.Pen.Color := clBtnFace;
  Canvas.Pen.Width :=3;
  // часовую
  Vector(x0,y0, ahr, R-20);
  // минутную
  Vector(x0,y0, amin, R-15);
  // секундную
  Vector(x0,y0, asec, R-7);

  t := Now();

  // новое положение стрелок
  ahr :=  90 — HourOf(t)*30-(MinuteOf(t)div 12)*6;
  amin := 90 — MinuteOf(t)*6;
  asec := 90 — SecondOf(t)*6;

  // нарисовать стрелки
  // часовая стрелка
  Canvas.Pen.Width := 3;
  Canvas.Pen.Color := clBlack;
  Vector(x0,y0, ahr, R-20);

  // минутная стрелка
  Canvas.Pen.Width := 2;
  Canvas.Pen.Color := clBlack;
  Vector(x0,y0, amin, R-15);

  // секундная стрелка
  Canvas.Pen.Width := 1;
  Canvas.Pen.Color := clYellow;
  Vector(x0,y0, asec, R-7);
end;

// прорисовка циферблата и начальных стрелок
procedure TForm1.FormPaint(Sender: TObject);
var
    x,y: integer;    // координаты маркера на циферблате
    a: integer;      // угол между OX и прямой (x0,y0) (x,y)
    h: integer;      // метка часовой риски

    bs: TBrushStyle; // стиль кисти
    pc: TColor;      // цвет карандаша
    pw: integer;     // ширина карандаша
```

```
begin
  bs := Canvas.Brush.Style;
  pc := Canvas.Pen.Color;
  pw := Canvas.Pen.Width;

  Canvas.Brush.Style := bsClear;
  Canvas.Pen.Width := 1;
  Canvas.Pen.Color := clBlack;

  a:=0; // метки ставим от 3-х часов, против
        // часовой стрелки
  h:=3; // угол 0 градусов — это 3 часа

  // циферблат
  while a < 360 do
  begin
    x:=x0+Round( R * cos(a*2*pi/360));
    y:=x0-Round( R * sin(a*2*pi/360));
    Form1.Canvas.MoveTo(x,y);
    if (a mod 30) = 0 then
        begin
            Canvas.Ellipse(x-2,y-2,x+3,y+3);
            // цифры по большему радиусу
            x:=x0+Round( (R+15) * cos(a*2*pi/360));
            y:=x0-Round( (R+15) * sin(a*2*pi/360));
            Canvas.TextOut(x-5,y-7,IntToStr(h));
            dec(h);
            if h = 0 then h:=12;
        end
        else Canvas.Ellipse(x-1,y-1,x+1,y+1);
    a:=a+6; // 1 минута — 6 градусов
  end;
  // восстановить карандаш-кисть
  Canvas.Brush.Style := bs;
  Canvas.Pen.Width := pw;
  Canvas.Pen.Color := pc;

  DrawClock;
end;

// прорисовка текущих положений стрелок часов
procedure TForm1.Timer1Timer(Sender: TObject);
```

```
begin
    DrawClock;
end;

end.
```

36. Напишите программу, по поверхности формы которой движется изображение (рис. 1.34). Изображение перемещающегося объекта и фоновый рисунок (рис. 1.35) должны загружаться из файла.

Рис. 1.34. Летящий над городом самолет

Рис. 1.35. Объект и фоновый рисунок

```
unit aplane_;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Buttons;


type
  TForm1 = class(TForm)
    Timer1: TTimer;
    Image1: TImage;
    procedure Timer1Timer(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
var
  Back, Picture: TBitMap; // фон и картинка
  BackRct : TRect;        // положение и размер области фона,
                          // которая должна быть восстановлена

  x,y:integer;  // текущее положение картинки
  W,H: integer; // размеры картинки


procedure TForm1.FormCreate(Sender: TObject);
begin
  { Свойству AutoSize обязательно надо
    присвоить значение False. Это можно
    сделать во время создания формы.
    }
  Image1.AutoSize := False;
```

```
  // создать два объекта — битовых образа
  Back := TBitmap.Create;     // фон
  Picture := TBitmap.Create;  // картинка

  // загрузить и вывести фон
  Back.LoadFromFile('factory.bmp');
  Image1.Width  := Back.Width;
  Image1.Height := Back.Height;
  Image1.Canvas.Draw(0,0,Back);

  // загрузить картинку, которая будет двигаться
  Picture.LoadFromFile('aplane.bmp');
  W := Picture.Width;
  H := Picture.Height;

  // определим "прозрачный" цвет
  Picture.Transparent := True;
  // прозрачный цвет картинки определяет
  // левый верхний пиксел картинки
  Picture.TransParentColor := Picture.Canvas.Pixels[1,1];

  // начальное положение картинки
  x := -W;
  y := 20;

  // определим сохраняемую область фона
  BackRct:=Bounds(x,y,W,H);

end;

// обработка сигнала таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
     // восстановлением фона удалим рисунок
     Image1.Canvas.CopyRect(BackRct,Back.Canvas,BackRct);

     x:=x+2;
     if x > Image1.Width then x:=-W;

     // определим сохраняемую область фона
     BackRct:=Bounds(x,y,W,H);

     // выведем рисунок
    Image1.Canvas.Draw(x,y,Picture);
end;
```

```
// завершение работы программы
procedure TForm1.FormClose(Sender: TObject;
                           var Action: TCloseAction);
begin
 // освободим память, выделенную
 // для хранения битовых образов
 Back.Free;
 Picture.Free;
end;
end.
```

37. Напишите программу, в окне которой отображается "мультик", загруженный из BMP-файла. Вид формы приложения приведен на рис. 1.36, пример мультика (картинка, которая находится в BMP-файле) — на рис. 1.37.

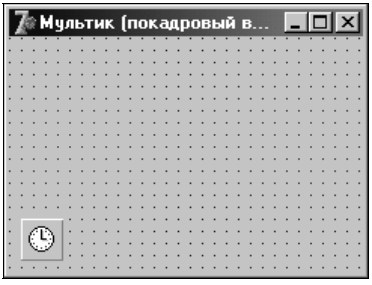


Рис. 1.36. Форма программы **Мультипликация**

Рис. 1.37. Пример "мультика" — содержимое BMP-файла

```
implementation

{$R *.DFM}

const
  FILMFILE = 'delphi.bmp'; // фильм — BMP-файл
  N_KADR=12; // кадров в фильме (для данного файла)
```

```
var
  Film: TBitMap;         // фильм — все кадры
  WKadr,HKadr: integer; // ширина и высота кадра
  CKadr: integer;        // номер текущего кадра
  RectKadr: TRect;       // положение и размер кадра в фильме
  RectScr :  Trect;      // координаты и размер области
                         // отображения фильма


procedure TForm1.FormCreate(Sender: TObject);
begin
    Film := TBitMap.Create;       // создать объект типа TBitMap
    Film.LoadFromFile(FILMFILE);  // загрузить "фильм" из файла
    WKadr := Round(Film.Width/N_Kadr);
    HKadr := Film.Height;

    RectScr := Bounds(10,10,WKadr,HKadr);

    Ckadr:=0;

    Timer1.Interval := 150; // период обновления кадров — 0.15 сек
    Timer1.Enabled:=True;  // запустить таймер
end;

// обработка сигнала от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // определим положение текущего кадра в фильме
  RectKadr:=Bounds(WKadr*CKadr,0,WKadr,HKadr);

  // вывод кадра из фильма
  Form1.Canvas.CopyRect(RectScr,Film.Canvas,RectKadr);

  // подготовимся к выводу следующего кадра
  CKadr := CKadr+1;
  if CKadr = N_KADR
     then CKadr:=0;
end;

end.
```

38. Напишите программу, в окне которой прокручивается текст, подобный титрам в конце фильма. Титры могут быть на фоне иллюстрации, которая должна прокручиваться вместе с текстом. Рекомендуемый вид формы приведен на рис. 1.38.

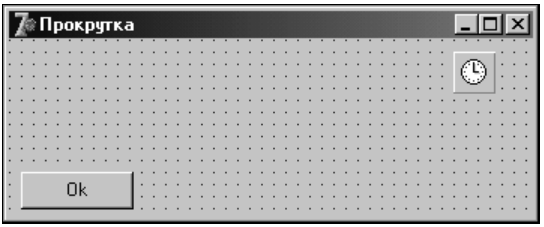


Рис. 1.38. Форма программы **Прокрутка**

```
implementation

{$R *.dfm}

const
  HB = 58;  // высота области вывода
            // картинки на форме
  HR = 274; // высота плаката
  { В простейшем случае плакат в файле
    должен быть продублирован по вертикали
    два раза.
    Высота прокручиваемой картинки
    (битового образа в файле)
    должна быть больше или равна
    HB+HR.
  }

var
  pic :TBitMap;       // прокручиваемая картинка
  sRect,dRect: TRect; // область-источник
  t: integer;

procedure TForm1.FormCreate(Sender: TObject);
begin
  pic := TBitMap.Create;
  pic.LoadFromFile('baner.bmp');       // загрузить картинку
  dRect := Bounds(10,10,pic.Width,HB); // положение и размер
                                       // области, в которой
                                       // прокручивается картинка
  sRect := Rect(0,0,pic.Width,HB);     // отображаемая область
  t:=0;
end;
```

```
// сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Canvas.CopyRect(dRect,pic.Canvas,sRect); // отобразить часть
                                           // картинки
  inc(t);
  if t = HR // длина ролика
      then t:=0;
  sRect := Bounds(0,t,pic.Width,HB);  // следующий кадр
end;

// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
begin
  Form1.Close;
end;

end.
```

39. Напишите программу, в окне которой в стиле бегущей строки прокручивается битовый образ (рис. 1.39). Битовый образ должен загружаться из ресурса программы (подготовить файл ресурса можно при помощи утилиты Image Editor).

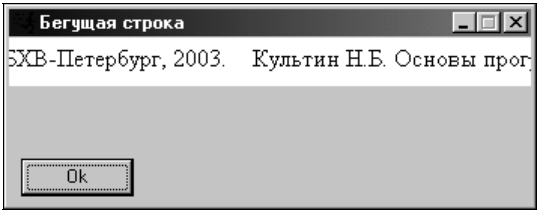


Рис. 1.39. Окно программы **Бегущая строка**

```
{ Бегущая строка.
  Битовый образ загружается из ресурса. }
unit hscroll_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,  Dialogs, StdCtrls, ExtCtrls;
```

```
type
  TForm1 = class(TForm)
    Timer: TTimer;
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure TimerTimer(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

{$R hbaner.res} // файл ресурсов, в котором
                // находится битовый образ, можно создать
                // при помощи Image Editor

{ Высота битового образа в файле ресурсов
  не может быть меньше 32 пикселов }

const
    WB = 350;  // ширина области вывода
               // картинки на форме
    TP = 573;  // период рисунка бегущей строки

var
    pic :TBitMap;       // картинка — бегущая строка
    sRect,dRect: TRect; // область-источник и область-приемник
    t: integer;

procedure TForm1.FormCreate(Sender: TObject);
begin
    pic := TBitMap.Create;
    pic.LoadFromResourceName(HInstance,'BANER2'); // загрузить
                                                  // картинку
    dRect := Bounds(0,0,WB,pic.Height); // область, в которой
                                        // бежит строка,
```

```
    sRect := Rect(0,0,TP,pic.Height);   // отображаемая в данный
                                        // момент область рисунка
    t:=0;
end;

// сигнал от таймера
procedure TForm1.TimerTimer(Sender: TObject);
begin
    Canvas.CopyRect(dRect,pic.Canvas,sRect); // отобразить часть
                                             // картинки
    inc(t);
    if t = TP // длина ролика
        then t:=0;
    sRect := Bounds(t,0,WB,pic.Height);  // следующий кадр
end;

// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
begin
    Form1.Close;
end;

end.
```

40. Напишите программу, используя которую можно просмотреть иллюстрации, находящиеся в одном из каталогов компьютера. Вид окна программы приведен на рис. 1.40.

```
{ Просмотр иллюстраций }
unit shpic_;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls, Menus;
type
  TForm1 = class(TForm)
    Image1: TImage;     // поле вывода иллюстрации
    Button1: TButton;   // кнопка Дальше
    Label1: TLabel;
    Edit1: TEdit;
    RadioButton1: TRadioButton;  // выбор: BMP — формат
    RadioButton2: TRadioButton;  // выбор: JPG — формат
```

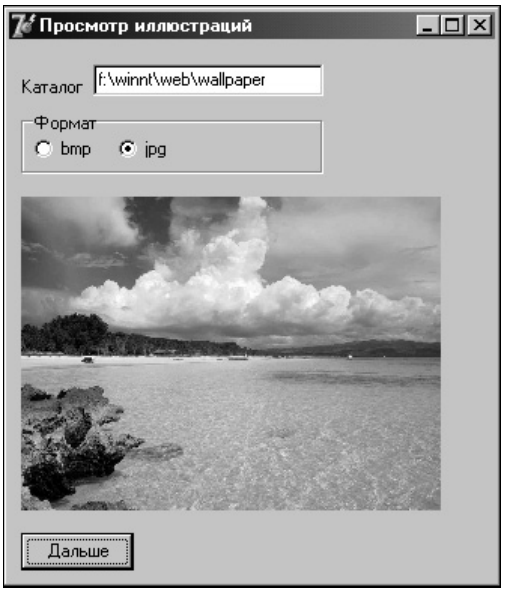


Рис. 1.40. Окно программы **Просмотр иллюстраций**

```
    GroupBox1: TGroupBox;
    procedure Button1Click(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure FormCreate(Sender: TObject);
    procedure RadioButton1Click(Sender: TObject);
    procedure RadioButton2Click(Sender: TObject);

    // эти объявления вставлены сюда вручную
    procedure FirstPicture; // выводит первую иллюстрацию
    procedure NextPicture;  // выводит следующую иллюстрацию
    procedure ScaleImage;   // масштабирует картинку

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  iw,ih: integer; // первоначальный размер компонента Image
```

```
implementation

{$R *.DFM}
uses
     jpeg; // чтобы иметь возможность просмотра
           // jpg — иллюстраций

var
  aSearchRec : TSearchRec;

  aPath: String[128]; // каталог, в котором находятся иллюстрации
  aFile: String[128]; // файл иллюстрации
  aMask: String[5];   // расширение файла иллюстрации
  n: integer = 0;

procedure TForm1.FormCreate(Sender: TObject);
begin
    Image1.AutoSize := False;
    Image1.Stretch := True;     // разрешим масштабирование

    // запомним первоначальный размер области вывода иллюстрации
    iw := Image1.Width;
    ih := image1.Height;

    Button1.Enabled := False; // сделаем недоступной
                              // кнопку Дальше
    FirstPicture; // вывести первую иллюстрацию
end;

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    if Key = #13 then
        if DirectoryExists(Edit1.Text)
            then FirstPicture
            else ShowMessage('Каталог '+
            Edit1.Text +' указан неверно.');
end;

// вывести первую иллюстрацию
procedure TForm1.FirstPicture;
var
   r : integer; // результат поиска файла
begin
   aPath := Edit1.Text;
```

```
   if aPath[Length(aPath)] <> '\'
        then aPath := aPath + '\';
   if RadioButton1.Checked
        then aMask := '*.bmp'
        else aMask := '*.jpg';
   r := FindFirst(aPath + aMask, faAnyFile, aSearchRec);
   if r = 0 then
      begin
          aFile := aPath + aSearchRec.Name;
          Image1.Picture.LoadFromFile(aFile); // загрузить
                                              // иллюстрацию
          ScaleImage;
          r := FindNext(aSearchRec); // найти следующий файл
          if r = 0 then  // еще есть файлы иллюстраций
             Button1.Enabled := True;
      end;
end;

// вывести следующую иллюстрацию
Procedure TForm1.NextPicture();
var
   r : integer;
begin
   aFile := aPath + aSearchRec.Name;
   Image1.Picture.LoadFromFile(aFile);
   ScaleImage;

   // подготовим вывод след. иллюстрации
   r := FindNext(aSearchRec); // найти следующий файл
   if r <> 0
      then // больше нет иллюстраций
         Button1.Enabled := False;
end;

// щелчок на кнопке Дальше
procedure TForm1.Button1Click(Sender: TObject);
begin
    NextPicture;
end;

// изменение размера области вывода иллюстрации пропорционально
// размеру иллюстрации
Procedure Tform1.ScaleImage;
```

```
var
    pw, ph : integer;      // размер иллюстрации
    scaleX, scaleY : real; // масштаб по X и Y
    scale : real;          // масштаб
begin
      // иллюстрация уже загружена
      // получим ее размеры
      pw := Image1.Picture.Width;
      ph := Image1.Picture.Height;
      if pw > iw // ширина иллюстрации больше ширины компонента
                 // Image
          then scaleX := iw/pw  // нужно масштабировать
          else scaleX := 1;
      if ph > ih // высота иллюстр. больше высоты компонента
          then scaleY := ih/ph  // нужно масштабировать
          else scaleY := 1;

      // выберем наименьший коэффициент
      if  scaleX < scaleY
          then scale := scaleX
          else scale := scaleY;

      // изменим размер области вывода иллюстрации
      Image1.Height := Round(Image1.Picture.Height*scale);
      Image1.Width := Round(Image1.Picture.Width*scale);
      // т. к. Strech = True и размер области пропорционален
      // размеру картинки, то картинка масштабируется
      // без искажений
end;

// выбор BMP-формата
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
    FirstPicture;
end;

// выбор JPG-формата
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
    FirstPicture;
end;

end.
```

41. Напишите программу, диалоговое окно которой имеет фоновый рисунок, загружаемый из файла (рис. 1.41). Если размер рисунка меньше размера окна, то фоновый рисунок должен быть составлен по принципу кафельной плитки. В качестве фонового использовать один из стандартных рисунков Windows, например, находящийся в файле Колечки.bmp или Пузырьки.bmp.

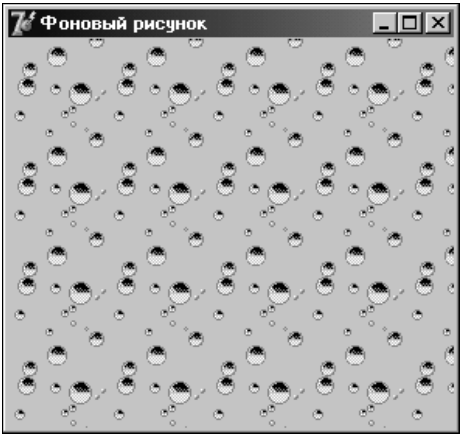


Рис. 1.41. Окно с фоновым рисунком

```
implementation

var
    Back : TBitmap; // фоновая картинка

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
   Back := TBitmap.Create;
   // Back.LoadFromResourceName(HInstance,'PUZ');
   Back.LoadFromFile('пузырьки1.bmp'); // загрузить картинку

end;

procedure TForm1.FormPaint(Sender: TObject);
var
    x,y: integer; // левый верхний угол картинки
```

```
begin
    x:=0; y:=0;
    while y < Form1.Height do begin
        while x < Form1.Width do begin
            form1.Canvas.Draw(x,y,Back);
            x:=x+Back.Width;
        end;
        x:=0;
        y:=y+Back.Height;
    end;
end;

end.
```

Мультимедиа

Общие замечания

Приступая к решению задач этого раздела, необходимо вспомнить:

- Работу с анимацией и звуком обеспечивают компоненты `Animate` и `MediaPlayer`.
- Компонент `Animate` позволяет воспроизвести только простую, не сопровождаемую звуком анимацию.
- Компонент `MediaPlayer` позволяет воспроизводить звук, анимацию и видео.

42. Напишите программу, в главном окне которой, сразу после появления окна, воспроизводится не сопровождаемая звуком анимация, например рекламный ролик. Анимация должна воспроизводиться один раз. На рис. 1.42 приведена форма программы.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    try
        Animate1.FileName := 'Delphi.avi';
        Animate1.Play(0,Animate1.FrameCount,1);
```

```
    except
       on Exception do begin
            Label1.Caption :=
                'Ошибка доступа к файлу ' +
                Animate1.FileName +
                '. Файл не найден или анимация' +
                'сопровождается звуком.'
               ;
            exit;
       end;
    end;
end;

// щелчок на кнопке Ok
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    Form1.Close;
end;
```

Рис. 1.42. Форма программы **Анимация**

43. Напишите программу, в диалоговом окне которой в результате щелчка на командной кнопке выводится сопровождаемая звуком анимация — содержимое AVI-файла. Рекомендуемый вид формы приведен на рис. 1.43.

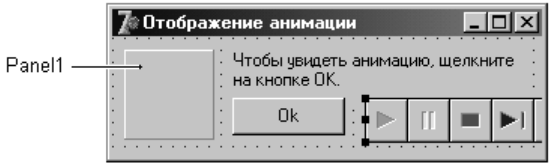


Рис. 1.43. Компонент `Panel1` используется в качестве экрана для отображения анимации

```
{ Отображение сопровождаемой звуком анимации
  при помощи компонента MediaPlayer }

unit UsMP_;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, MPlayer, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    MediaPlayer1: TMediaPlayer; // универсальный проигрыватель
    Panel1: TPanel;    // панель, на которую выводится анимация
    Button1: TButton;  // кнопка Ok

    Label1: TLabel;

    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

{
  Во время создания формы надо настроить
  компонент MediaPlayer1:
  MediaPlayer1.Display = Panel1
  MediaPlayer1.FileName = delphi_s.avi — имя файла
  MediaPlayer1.AutoOpen = True
  MediaPlayer1.Visible = False
}

// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
     MediaPlayer1.Play; // воспроизведение анимации
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
     { Размер области отображения анимации
       должен соответствовать размеру анимации.
       Размер анимации в файле delphi_s.avi
       60x60 }

     // зададим размер области вывода анимации
     MediaPlayer1.DisplayRect:=Rect(0,0,60,60);
end;

end.
```

44. Напишите программу, используя которую можно просматривать анимацию (содержимое AVI-файла) в реальном масштабе времени или по кадрам (рис. 1.44). Для отображения анимации используйте компонент `Animate`. Обратите внимание, что этот компонент работает только с AVI-файлами, в которых нет звука (для отображения анимации, сопровождаемой звуком, используется компонент `MediaPlayer`). Для выбора файла анимации используйте стандартное диалоговое окно **Открытие файла**. Рекомендуемый вид окна программы приведен на рис. 1.45.

```
{ Просмотр анимации — содержимого AVI-файла.
  (c) Культин Н.Б., 2003.

  **** ВНИМАНИЕ! ****

  Компонент Animate отображает AVI-файлы,
  в которых нет звука! При попытке открыть
  файл, в котором находится сопровождаемая
  звуком анимация, возникает ошибка. }

unit ShowAVI_;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ComCtrls, ExtCtrls;
```

Рис. 1.44. Окно программы **Просмотр анимации**

Рис. 1.45. Форма программы **Просмотр анимации**

```
type
  TForm1 = class(TForm)
    Animate1: TAnimate;  // компонент Animate
    Button1: TButton;    // Пуск-Стоп
    Button2: TButton;    // следующий кадр
    Button3: TButton;    // предыдущий кадр
    // режим просмотра
    RadioButton1: TRadioButton; // просмотр всей анимации
    RadioButton2: TRadioButton; // покадровый

    OpenDialog1: TOpenDialog; // диалоговое окно Открытие файла
    Button4: TButton; // активизация окна Открытие файла

    GroupBox1: TGroupBox;
    Label1: TLabel;
    Edit1: TEdit;

    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure RadioButton1Click(Sender: TObject);
    procedure RadioButton2Click(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure Button4Click(Sender: TObject);

    // это объявление вставлено сюда вручную
    procedure OpenAVI;

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;   // форма
  CFrame: integer; // номер кадра, отображаемого
                   // в режиме покадрового просмотра
implementation

{$R *.DFM}

// нажатие клавиши в поле Файл
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
```

```
begin
    if (Key = #13) and (Length(Edit1.Text) <> 0) then
        OpenAVI;
end;


// Щелчок на кнопке Выбрать
// открывает стандартное диалоговое
// окно Открытие файла
procedure TForm1.Button4Click(Sender: TObject);
begin
    OpenDialog1.Title := 'Выбрать AVI-файл';
    if OpenDialog1.Execute then
    begin
        Edit1.Text := OpenDialog1.FileName;
        OpenAVI;
    end;
end;

// пуск и остановка просмотра анимации
procedure TForm1.Button1Click(Sender: TObject);
begin
     if Animate1.Active = False // в данный момент анимация
                                // не выводится
      then begin
      Animate1.StartFrame:=1; // вывод с первого кадра
      Animate1.StopFrame:=Animate1.FrameCount; // по последний
                                               // кадр
        Animate1.Active:=True;
        Button1.caption:='Стоп';
        RadioButton2.Enabled:=False;
      end
    else  // анимация отображается
      begin
        Animate1.Active:=False; // остановить отображение
        Button1.caption:='Пуск';
        RadioButton2.Enabled:=True;
      end;
end;

// активизация режима просмотра всей анимации
procedure TForm1.RadioButton1Click(Sender: TObject);
```

```
begin
     Button1.Enabled:=True; // доступна кнопка Пуск
     // сделать недоступными кнопки покадрового просмотра
     Form1.Button3.Enabled:=False;
     Form1.Button2.Enabled:=False;
end;

// активизация режима покадрового просмотра
procedure TForm1.RadioButton2Click(Sender: TObject);
begin
     Button2.Enabled:=True;  // кнопка Следующий кадр доступна
     Button3.Enabled:=False; // кнопка Предыдущий кадр
недоступна

     // сделать недоступной кнопку Пуск — вывод всей анимации
     Button1.Enabled:=False;
     Animate1.StartFrame:=1;

     CFrame:=1;
end;

// переход к следующему кадру
procedure TForm1.Button2Click(Sender: TObject);
begin
     if CFrame < Animate1.FrameCount then
     begin
        CFrame := CFrame + 1;
        // вывести кадр
        Animate1.StartFrame := CFrame;
        Animate1.StopFrame := CFrame;
        Animate1.Active := True;

        if CFrame = Animate1.FrameCount // текущий кадр —
                                        // последний
               then Button2.Enabled:=False;
     end;
      if CFrame > 1 then Button3.Enabled := True;
end;

// переход к предыдущему кадру
procedure TForm1.Button3Click(Sender: TObject);
begin
     if CFrame > 1 then
```

```
     begin
        CFrame := CFrame — 1;
        // вывести кадр
        Animate1.StartFrame := CFrame;
        Animate1.StopFrame := CFrame;
        Animate1.Active := True;

        if CFrame = 1 // текущий кадр — первый
            then Form1.Button3.Enabled := False;
     end;
     if CFrame < Animate1.FrameCount
         then Button2.Enabled := True;
end;

// открывает AVI-файл
procedure TForm1.OpenAVI;

begin
    Button1.Enabled := False;
    Button2.Enabled := False;
    Button3.Enabled := False;
    RadioButton1.Enabled := False;
    RadioButton2.Enabled := False;
    try
        Animate1.FileName := Edit1.Text;
    except
      on Exception do
      begin
        MessageDlg('Ошибка формата AVI-файла.'+
           #13+'(Анимация не должна сопровождаться звуком.)',
           mtError,[mbOk],0);
        exit;
      end;
    end;
    Button1.Enabled := True;
    // Button2.Enabled := True;
    // Button3.Enabled := True;
    RadioButton1.Enabled := True;
    RadioButton2.Enabled := True;
end;

end.
```

45. Напишите программу, используя которую можно прослушать звуковые файлы Windows. Рекомендуемый вид диалогового окна программы приведен на рис. 1.46.

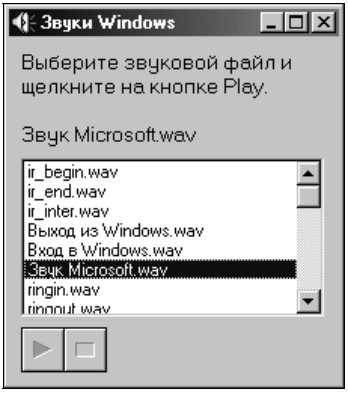


Рис. 1.46. Окно программы **Звуки Windows**

```
unit WinSound_;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, MPlayer, ExtCtrls;

type
  TForm1 = class(TForm)
    MediaPlayer1: TMediaPlayer;
    ListBox1: TListBox;          // список WAV-файлов
    Label2: TLabel;
    Label1: TLabel;              // выбранный из списка файл
    procedure ListBox1Click(Sender: TObject);
    procedure MediaPlayer1Click(Sender: TObject; Button: TMPBtnType;
      var DoDefault: Boolean);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.DFM}

var
    SOUNDPATH: string[80];

{ Определяет каталог Windows и формирует
  список WAV-файлов, находящихся в
  подкаталоге Media }
procedure TForm1.FormCreate(Sender: TObject);
var
  lpBuf: PChar;          // указатель на nul-terminated-строку
  sWinDir: string[128];  // обычная Паскаль-строка

  SearchRec: TSearchRec; // структура SearchRec содержит
           // информацию о файле, удовлетворяющем условию поиска
begin
  // определить положение каталога Media
  GetMem(lpBuf,MAX_PATH);        // выделить память для строки
  GetWindowsDirectory(lpBuf,MAX_PATH); // получить каталог Windows
  sWinDir:=lpbuf;          // скопировать nt-строку в p-строку
  SOUNDPATH:=sWinDir+'\Media\';       // вывести результат
  FreeMem(lpBuf);                     // освободить память

  // сформировать список WAV-файлов
  if FindFirst(SOUNDPATH+'*.wav', faAnyFile, SearchRec) =0 then
     begin
          // в каталоге есть файл с расширением wav
          // добавим имя этого файла в список
          Form1.ListBox1.Items.Add(SearchRec.Name);
          // пока в каталоге есть другие файлы с расширением wav
          while (FindNext(SearchRec) = 0) do
              Form1.ListBox1.Items.Add(SearchRec.Name);
     end;
end;

// щелчок на элементе списка
procedure TForm1.ListBox1Click(Sender: TObject);
```

```
begin
 // вывести в поле метки Label2 имя выбранного файла
 Label2.Caption:=ListBox1.Items[ListBox1.itemIndex];
end;

// щелчок на кнопке компонента MediaPlayer
procedure TForm1.MediaPlayer1Click(Sender: TObject;
                                   Button: TMPBtnType;
  var DoDefault: Boolean);
begin
     if (Button = btPlay) and (Label2.Caption <> '') then
     begin
        // нажата кнопка Play
        with MediaPlayer1 do
          begin
             FileName:=SOUNDPATH+Label2.Caption; // имя  выбранного
                                                 // файла
             Open;
             // Wait:= True;
          end;
     end;
end;

end.
```

46. Напишите программу, используя которую можно просмотреть видеоклип. Клип должен воспроизводиться в диалоговом окне программы. Для выбора клипа (AVI-файла) используйте стандартное диалоговое окно **Открытие файла**. Рекомендуемый вид формы программы приведен на рис. 1.47.

```
{ Видео (AVI) Плеер
  (c) Культин Н.Б., 2003}
unit Vp_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,  Dialogs, ExtCtrls, MPlayer, Buttons,
  StdCtrls;
```

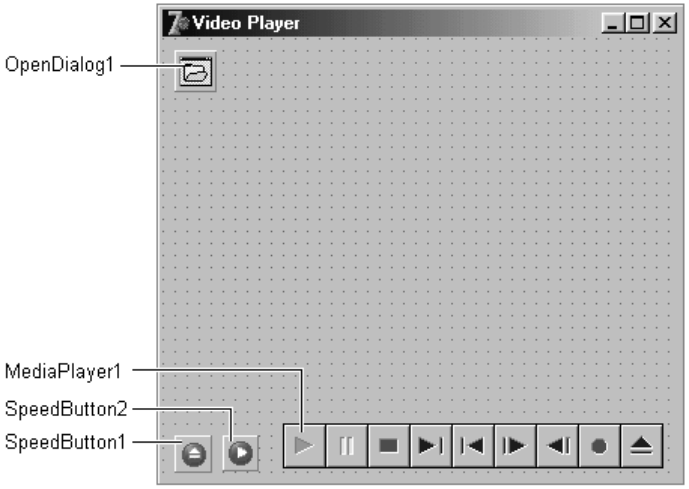


Рис. 1.47. Форма программы **Video Player**

```
type
  TForm1 = class(TForm)
    OpenDialog: TOpenDialog;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    MediaPlayer: TMediaPlayer;
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure MediaPlayerNotify(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
   MediaPlayer.Display := Form1;

   // это можно сделать во время создания
   // формы. Но на всякий случай ...
   SpeedButton1.GroupIndex := 1;
   SpeedButton1.AllowAllUp := True;
end;

// возвращает размер изображения AVI-файла
procedure DimAvi(f: string; var w,h: integer);
var
    fst: TFileStream;
    // структуру заголовка AVI-файла можно
    // найти, например, в  ..\CBuilder\Include\aviriff.h
    header: record
        RIFF: array[1..4] of char;    // 'RIFF'
        nu1:  array[1..5] of LongInt; // не используется
        AVIH: array[1..4] of char;    // 'avih'
        nu2:  array[1..9] of LongInt; // не используется
        Width:  LongInt;
        Height: LongInt;
    end;

begin
    fst := TFileStream.Create(f,fmOpenRead);
    fst.Read(header,sizeof(header));
    w := header.Width;
    h := header.Height;
    fst.Destroy;
end;


// щелчок на кнопке Eject — выбор файла
procedure TForm1.SpeedButton2Click(Sender: TObject);
var
    top,left: integer; // левый верхний угол "экрана"
    width,height: integer; // размер экрана
    mw,mh: integer; // максимально возможный размер экрана
    kh,kw: real;    // коэф-ты масштабирования по h и w
    k: real;        // коэф-т масштабирования
```

```
begin
    OpenDialog.Title := 'Выбор клипа';
    if not OpenDialog.Execute
      then exit;

    // пользователь выбрал файл
    // определим размер и положение
    // "экрана" (области на поверхности формы),
    // на котором будет выведен клип
    DimAvi(OpenDialog.FileName,width,height);

    mh:=SpeedButton1.Top — 10;
    mw:=Form1.ClientWidth;

    if mh > height
        then kh :=1
        else kh := mh/height;

    if mw > width
        then kw :=1
        else kw := mw/width;

    if kw < kh
        then k := kw
        else k := kh;

    // здесь масштаб определен

    width  := Round(width * k);
    height := Round(height * k);

    left := (Form1.ClientWidth — width) div 2;
    top := 10;

    MediaPlayer.FileName := OpenDialog.FileName;
    MediaPlayer.Open;
    MediaPlayer.DisplayRect := Rect(left,top,width,height);
    SpeedButton1.Enabled := True;
end;

// щелчок на кнопке Play/Stop
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
    if SpeedButton1.Down  then
```

```
    begin
      // начать воспроизведение
      MediaPlayer.Play;
      SpeedButton1.Hint := 'Stop';
    end
    else begin
      // остановить воспроизведение
      MediaPlayer.Stop;
      SpeedButton1.Hint := 'Play';
    end;
end;

// сигнал от плеера
procedure TForm1.MediaPlayerNotify(Sender: TObject);
begin
  if (MediaPlayer.Mode = mpStopped)
     and SpeedButton1.Down
  then
     SpeedButton1.Down := False; // "отжать" кнопку Play
end;

end.
```

47. Напишите программу, используя которую можно прослушать компакт-диск. Во время воспроизведения в диалоговом окне должен отображаться номер воспроизводимого трека и время воспроизведения. Рекомендуемый вид формы и окно программы приведены на рис. 1.48.

```
{ Проигрыватель Audio CD.
  (c) Культин Н.Б., 2003 }
unit CDp_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, Buttons, MPlayer,
  ExtCtrls, MMSYSTEM;
type
  TForm1 = class(TForm)
    MediaPlayer: TMediaPlayer;
```

Рис. 1.48. Форма и окно программы **CD Player**

```
    Timer1: TTimer;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Image1: TImage;
    procedure FormActivate(Sender: TObject);
    procedure MediaPlayerNotify(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure BitBtn3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
var
  Track: integer; // воспроизводимый трек

// начало работы программы
procedure TForm1.FormActivate(Sender: TObject);
begin
  MediaPlayer.Open;
  { Проверить, есть в CD-ROM Audio CD.
    Если в дисководе есть диск, то свойство
    Mode = mpStopped. Если диска нет
    или дисковод открыт, то Mode = mpOpened.
    Тип диска можно определить по количеству
    треков. Если диск — CD-ROM, то на нем один
    трек, на аудиодиске — кол-во треков больше 1 }
  if (MediaPlayer.Mode = mpStopped) and
     (MediaPlayer.Tracks > 1) then
  begin
    BitBtn1.Enabled := True;
    MediaPlayer.Notify := True;
  end
  else Timer1.Enabled := True;
end;

// сигнал от MediaPlayer
procedure TForm1.MediaPlayerNotify(Sender: TObject);
begin
  case MediaPlayer.Mode of

  mpOpened: // пользователь открыл дисковод
    begin
        BitBtn1.Enabled := False;
        BitBtn1.Caption := 'Play';
        BitBtn1.Tag := 0;
        BitBtn2.Enabled := False;
        BitBtn3.Enabled := False;
        Timer1.Enabled := True;
        Label2.Caption := '00:00';
    end;

  end;
  MediaPlayer.Notify := True;
end;
```

```
// щелчок на кнопке Play/Stop
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  if BitBtn1.Tag = 0
  then // щелчок на кнопке Play
  begin
    BitBtn1.Caption := 'Stop';
    BitBtn3.Enabled := True;
    MediaPlayer.Notify := False;
    MediaPlayer.Play;
    Timer1.Enabled :=True;
    BitBtn1.Tag := 1;
    Track := MCI_TMSF_TRACK(MediaPlayer.Position);
  end
  else
  begin // щелчок на кнопке Stop
    BitBtn1.Caption := 'Play';
    BitBtn3.Enabled := True;
    MediaPlayer.Notify := True;
    MediaPlayer.Stop;
    Timer1.Enabled := False;
    BitBtn1.Tag := 0;
   end;
end;

// сигнал от таймера: вывести номер трека
// и время воспроизведения
procedure TForm1.Timer1Timer(Sender: TObject);
var
  trk,            // трек
  min, sec: byte; // время
begin

  if MediaPlayer.Mode = mpPlaying then
  // режим воспроизведения
  begin
    trk := MCI_TMSF_TRACK(MediaPlayer.Position);
    min := MCI_TMSF_MINUTE(MediaPlayer.Position);
    sec := MCI_TMSF_SECOND(MediaPlayer.Position);
    Label1.Caption :=
        Format('Track %d.',[trk]);
    Label2.Caption :=
        Format('%d:%.2d',[min,sec]);
```

```
    if trk <> Track then
    begin
      Track := trk;
      if Track = 2
        then BitBtn2.Enabled := True;
      if Track = MediaPlayer.Tracks
        then BitBtn3.Enabled := False;
    end;
    exit;
  end;

  // если дисковод открыт или в нем нет
  // Audio CD, то Mode = mpOpen
  // ждем диск ( mpStopped + кол-во треков > 1)
  if (MediaPlayer.Mode = mpStopped) and
     (MediaPlayer.Tracks > 1) then
  begin
    Timer1.Enabled := False;
    MediaPlayer.Open;
    BitBtn1.Enabled := True;
    MediaPlayer.Notify := True;
    Label1.Caption := 'Audio CD';
    Label1.Visible := True;
  end;

  // дисковод открыт или в дисководе нет диска
  if MediaPlayer.Mode = mpOpen then
  begin
    Label1.Caption := 'Вставьте Audio CD';
    if Label1.Visible then
      Label1.Visible := False
    else
      Label1.Visible := True;
  end;
end;

// кнопка <<
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
    MediaPlayer.Previous; // в начало текущего трека
    MediaPlayer.Previous; // в начало предыдущего трека
    if MCI_TMSF_TRACK(MediaPlayer.Position) = 1 then
      BitBtn2.Enabled := False;
```

```
    if not BitBtn3.Enabled
      then BitBtn3.Enabled := True;
end;

// кнопка >>
procedure TForm1.BitBtn3Click(Sender: TObject);
begin
  MediaPlayer.Next;
  // если перешли к последнему треку, то кнопку
  // Next сделать недоступной
  if MCI_TMSF_TRACK(MediaPlayer.Position) = MediaPlayer.Tracks
    then BitBtn3.Enabled := False;
end;

// пользователь закрыл окно программы
procedure TForm1.FormClose(Sender: TObject;
                           var Action: TCloseAction);
begin
  MediaPlayer.Stop;
end;

end.
```

48. Напишите программу, используя которую можно прослушать компакт-диск. Для выбора воспроизводимого трека (мелодии) используйте компонент `TrackBar` (рис. 1.49).

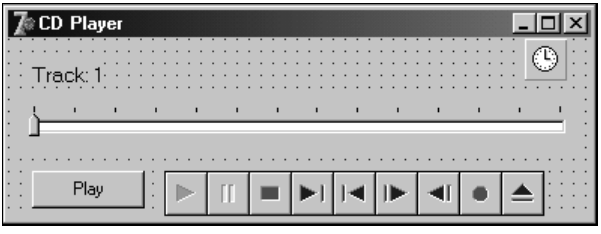


Рис. 1.49. Форма программы **CD Player**

```
{ CD Player. (c) Культин Н.Б., 2003}
unit CDp2_;

interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, ComCtrls, StdCtrls, MPlayer,
  ExtCtrls, MMSYSTEM;
type
  TForm1 = class(TForm)
    MediaPlayer: TMediaPlayer;
    TrackBar: TTrackBar;
    Button1: TButton;
    Timer1: TTimer;
    Label1: TLabel;
    Label2: TLabel;
    procedure TrackBarChange(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action:
TCloseAction);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

var
  Track: integer;

// пользователь изменил положение бегунка
procedure TForm1.TrackBarChange(Sender: TObject);
var
  trk: integer; // трек, который надо воспроизвести
begin
  trk :=TrackBar.Position;
  if trk > MCI_TMSF_TRACK(MediaPlayer.Position) then
    while trk > MCI_TMSF_TRACK(MediaPlayer.Position) do
      MediaPlayer.Next
```

```
  else
     while trk < MCI_TMSF_TRACK(MediaPlayer.Position) do
      MediaPlayer.Previous;
end;

// кнопка Play/Stop
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Button1.Tag = 0 then
    begin
      Button1.Caption := 'Stop';
      Button1.Tag :=1;
      MediaPlayer.Play;
    end
  else
    begin
      Button1.Caption := 'Play';
      Button1.Tag :=0;
      MediaPlayer.Stop;
    end;
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  if (MediaPlayer.Mode = mpStopped) and
     (MediaPlayer.Tracks > 1)
  then begin
        Button1.Enabled := True;
        TrackBar.Max := MediaPlayer.Tracks;
       end
  else begin
        label1.Caption := 'Вставьте в дисковод Audio CD';
       end;
end;

// сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
var
  trk,            // трек
  min, sec: byte; // время
begin
  case MediaPlayer.Mode of
```

```
mpPlaying: // режим воспроизведения
begin
  min := MCI_TMSF_MINUTE(MediaPlayer.Position);
  sec := MCI_TMSF_SECOND(MediaPlayer.Position);
  Label2.Caption :=
    Format('%d:%.2d',[min,sec]);

 // определим, надо ли перемещать бегунок
 trk := MCI_TMSF_TRACK(MediaPlayer.Position);
 if trk <> Track then
 begin
   Track:= trk;
   TrackBar.Position := Track;
   Label1.Caption :=
    Format('Track %d.',[trk]);
 end;
end;

mpOpen: // дисковод открыт или в нем нет диска
begin
  if Button1.Enabled then
  begin
    Label1.Caption :='Вставьте в дисковод Audio CD';
    Label2.Caption :='';
    Button1.Enabled := False; // режим ожидания диска
    Button1.Caption := 'Play';
  end;

  // мигание сообщения
  if Label1.Visible
    then Label1.Visible := False
    else Label1.Visible := True;

end;

mpStopped: // плеер остановлен
begin
  if not Button1.Enabled
    then  begin
      // плеер был в режиме ожидания диска
      // теперь диск вставлен
      MediaPlayer.Open;
      Button1.Enabled := True;
```

```
          Label1.Caption := 'Track: 1';
          Label1.Visible := True;
          Label2.Caption:= '';
          TrackBar.Position := 1;
          TrackBar.Max := MediaPlayer.Tracks;
        end;

      if Button1.Tag = 1 then
      begin
        // завершено воспроизведение последнего трека
        Button1.Caption := 'Play';
        Button1.Tag :=0;
        TrackBar.Position := 1;
        Label1.Caption := 'Track 1';
        Label2.Caption := '';
      end;
    end;
   end; // case

end;

procedure TForm1.FormClose(Sender: TObject;
                           var Action: TCloseAction);
begin
  MediaPlayer.Stop;
end;

end.
```

49. Напишите программу MP3 Player. Программа должна обеспечивать возможность выбора каталога, в котором находятся MP3-файлы, а также регулировку громкости звука непосредственно в диалоговом окне программы. Для отображения списка MP3-файлов используйте компонент `ListBox`, а для управления медиаплеером — кнопки `SpeedButton`. Рекомендуемый вид формы программы приведен на рис. 1.50.

```
{ MP3 Player с регулятором громкости.
  (c) Культин Н.Б., 2003. }
unit mp3p_;

interface
```

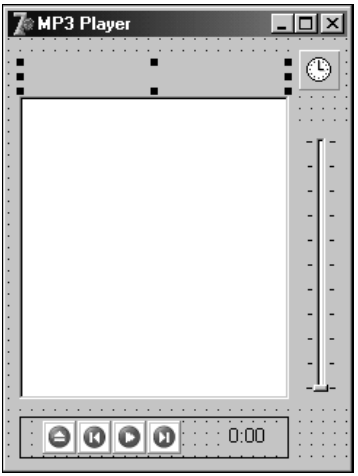


Рис. 1.50. Форма программы **MP3 Player**

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, Buttons,
  ExtCtrls, MPlayer, ComCtrls,
  MMSYSTEM, FileCtrl; // эти ссылки вставлены вручную
type
  TForm1 = class(TForm)
    // кнопки
    SpeedButton1: TSpeedButton; // Предыдущая композиция
    SpeedButton2: TSpeedButton; // Воспроизведение/Стоп
    SpeedButton3: TSpeedButton; // Следующая композиция
    SpeedButton4: TSpeedButton; // Выбор папки

    ListBox1: TListBox; // Список композиций (MP3-файлов)

    MediaPlayer1: TMediaPlayer; // медиаплеер

    TrackBar1: TTrackBar; // регулятор громкости

    Timer1: TTimer;
    Label1: TLabel;
    Label2: TLabel;

    Shape1: TShape; // рамка вокруг кнопок

    procedure FormCreate(Sender: TObject);
    procedure ListBox1Click(Sender: TObject);
```

```
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure TrackBar1Change(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);

    // эти объявления вставлены сюда вручную
    procedure Play; // воспроизведение
    procedure PlayList(Path: string); // формирует список
                                      // MP3-файлов

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
var
  SoundPath: string[255];
  min,sec: integer; // время воспроизведения
  volume: LongWord; // старшее слово — правый канал,
                    // младшее — левый.

// формирует список MP3-файлов
procedure TForm1.PlayList(Path: string);
var
  lpBuf: PChar;          // указатель на nul-terminated-строку
  sWinDir: string[128];  // обычная Паскаль-строка

SearchRec: TSearchRec; // структура SearchRec содержит
                       // информацию о файле, удовлетворяющем
                       // условию поиска
begin
  ListBox1.Clear;
  // сформировать список MP3-файлов
  if FindFirst(Path + '*.mp3', faAnyFile, SearchRec) = 0 then
```

```
     begin
          // в каталоге есть файл с расширением wav
          // добавим имя этого файла в список
          ListBox1.Items.Add(SearchRec.Name);
          // пока в каталоге есть другие файлы с расширением wav
          while (FindNext(SearchRec) = 0) do
              ListBox1.Items.Add(SearchRec.Name);
     end;
   ListBox1.ItemIndex := 0;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
   PlayList('');
   ListBox1.ItemIndex := 0;
   Label1.Caption:=ListBox1.Items[ListBox1.itemIndex];

   TrackBar1.Position := 7;

   // старшее слово переменной volume — правый канал,
   // младшее — левый
   volume := (TrackBar1.Position — TrackBar1.Max+1)* 6500;
   volume := volume +   (volume shl 16);
   waveOutSetVolume(WAVE_MAPPER,volume); // уровень сигнала
end;

// щелчок на названии произведения
procedure TForm1.ListBox1Click(Sender: TObject);
begin
  // вывести в поле метки Label1 имя выбранного файла
 if not SpeedButton2.Down
    then  SpeedButton2.Down := True;
 Label1.Caption:=ListBox1.Items[ListBox1.itemIndex];
 Play;
end;

// щелчок на кнопке Воспроизведение
procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
  if SpeedButton2.Down then
    // пользователь нажал кнопку
    // начать воспроизведение
    Play
```

```
  else
    // если кнопка Воспроизведение нажата,
    // то повторное нажатие останавливает
    // воспроизведение
    begin
      MediaPlayer1.Stop;
      Timer1.Enabled := False;
      SpeedButton2.Down := False;
      SPeedButton2.Hint := 'Play';
    end;
end;

// кнопка К предыдущей
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  if ListBox1.ItemIndex > 0 then
        ListBox1.ItemIndex := ListBox1.ItemIndex — 1;
  Play;
end;

// кнопка К следующей
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
  if ListBox1.ItemIndex < ListBox1.Count then
        ListBox1.ItemIndex := ListBox1.ItemIndex + 1;
  Play;
end;

// пользователь изменил положение
// регулятора громкости
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  volume := 6500* (TrackBar1.Max — TrackBar1.Position);
  volume := volume +   (volume shl 16);
  waveOutSetVolume(WAVE_MAPPER,volume);
end;

// воспроизвести композицию,
// название которой выделено
// в списке ListBox1
procedure TForm1.Play;
```

```
begin
  Timer1.Enabled := False;
  Label1.Caption:=ListBox1.Items[ListBox1.itemIndex];
  MediaPlayer1.FileName := SoundPath +
                           ListBox1.Items[ListBox1.itemIndex];

  try
   Mediaplayer1.Open;
  except
    on EMCIDeviceError do
      begin
        ShowMessage('Ошибка обращения к файлу '+
                    ListBox1.Items[ListBox1.itemIndex]);
        SpeedButton2.Down := False;
        exit;
      end;
  end;
  MediaPlayer1.Play;
  min :=0;
  sec :=0;
  Timer1.Enabled := True;
  SpeedButton2.Hint := 'Stop';
end;

// сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // изменить счетчик времени
  if sec < 59
    then inc(sec)
    else begin
      sec :=0;
      inc(min);
    end;

  // вывести время воспроизведения
  Label2.Caption := IntToStr(min)+':';
  if sec < 10
    then Label2.Caption :=
           Label2.Caption +'0'+ IntToStr(sec)
    else Label2.Caption :=
           Label2.Caption + IntToStr(sec);
```

```
  // если воспроизведение текущей композиции
  // не завершено
  if MediaPlayer1.Position < MediaPlayer1.Length
    then exit;

  // воспроизведение текущей композиции
  // закончено
  Timer1.Enabled := False; // остановить таймер
  MediaPlayer1.Stop;       // остановить плеер

  if ListBox1.ItemIndex < ListBox1.Count // список не исчерпан
  then begin
        ListBox1.ItemIndex := ListBox1.ItemIndex + 1;
        Play;
       end
end;

// щелчок на кнопке Папка
// выбрать папку, в которой находятся MP3-файлы
procedure TForm1.SpeedButton4Click(Sender: TObject);
var
  Root: string;      // корневой каталог
  pwRoot : PWideChar;
  Dir: string;
begin
  Root := ''; // корневой каталог — папка Рабочий стол
  GetMem(pwRoot, (Length(Root)+1) * 2);
  pwRoot := StringToWideChar(Root,pwRoot,MAX_PATH*2);
  if not SelectDirectory('Выберите папку', pwRoot, Dir)
     then Dir :=''
     else Dir := Dir+'\';

  // каталог, в котором находятся MP3-файлы выбран
  SoundPath := Dir;
  PlayList(SoundPath);
end;

end.
```

Файлы

Общие замечания

Приступая к решению задач этого раздела, необходимо вспомнить:

- При выполнении файловых операций возможны ошибки.
- Для обработки ошибок выполнения файловых операций нужно использовать инструкцию `try ... except`.

50. Напишите программу, которая в поле `Memo` выводит содержимое текстового файла. Рекомендуемый вид формы приведен на рис. 1.51.

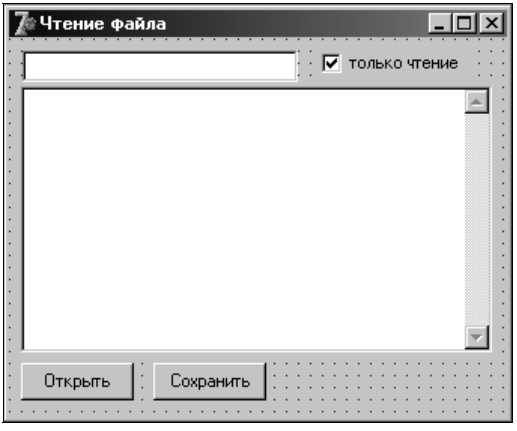


Рис. 1.51. Форма программы **Чтение файла**

```
implementation

{$R *.dfm}

// нажатие клавиши в поле редактирования
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    if Key = #13 // клавиша <Enter>
        then Button1.SetFocus;
end;
```

```
// щелчок на кнопке Открыть
procedure TForm1.Button1Click(Sender: TObject);
var
   f: TextFile;       // файл
   fName: String[80]; // имя файла
   buf: String[80];   // буфер для чтения строк

begin
   fName := Edit1.Text;
   AssignFile(f, fName);

   try
       Reset(f);  // открыть для чтения
   except
        on EInOutError do
        begin
            ShowMessage('Ошибка доступа к файлу '+
                        fName);
            exit;
        end;
   end;

   // чтение из файла
   while not EOF(f) do
     begin
        readln(f, buf);       // прочитать строку из файла
        Memo1.Lines.Add(buf); // добавить строку в поле Memo1
     end;

   CloseFile(f); // закрыть файл
end;

// щелчок на кнопке Сохранить
procedure TForm1.Button2Click(Sender: TObject);
var
   f: TextFile;       // файл
   fName: String[80]; // имя файла
   i: integer;
begin
   fName := Edit1.Text;
   AssignFile(f, fName);

   try
       Rewrite(f);  // открыть для перезаписи
```

```
   except
        on EInOutError do
        begin
            ShowMessage('Ошибка доступа к файлу '+ fName);
            exit;
        end;
   end;

   // запись в файл
   for i:=0 to Memo1.Lines.Count do // строки в поле Memo
                                    // пронумерованы с нуля
          writeln(f, Memo1.Lines[i]);

   CloseFile(f); // закрыть файл

   MessageDlg('Данные записаны в файл',
              mtInformation,[mbOk],0);
end;

// щелчок на переключателе Только чтение
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
    if not CheckBox1.Checked then
       begin
         Memo1.ReadOnly  := False; // разрешить редактирование
         Button2.Visible := True  // кнопка Сохранить доступна
       end
    else
        begin
          Memo1.ReadOnly := True; // запретить редактирование
          Button2.Visible := False; // скрыть кнопку Сохранить
        end;
end;

end.
```

51. Напишите программу, которая в поле `Memo` выводит содержимое текстового файла. Для получения от пользователя имени файла используйте стандартное диалоговое окно **Открытие файла**. Рекомендуемый вид диалогового окна приведен на рис. 1.52.

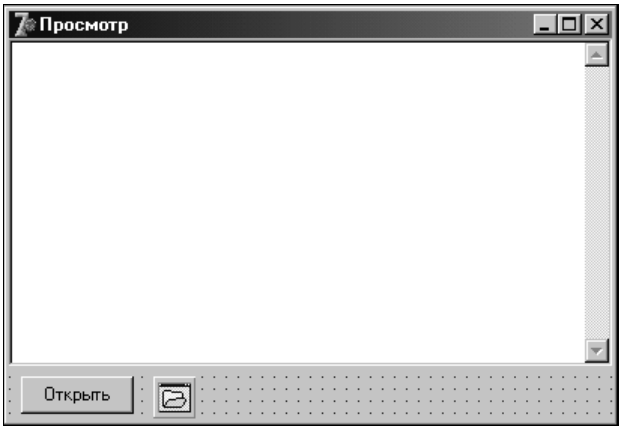


Рис. 1.52. Форма программы **Просмотр**

```
implementation

{$R *.dfm}

// щелчок на кнопке Открыть
procedure TForm1.Button1Click(Sender: TObject);
var
   f: TextFile;        // файл
   fName: String[80]; // имя файла
   buf: String[80];    // буфер для чтения строк

begin
   if not OpenDialog1.Execute
    then { пользователь закрыл диалог
           щелчком на кнопке Отмена }
         exit;

   // пользователь выбрал файл
   fName := OpenDialog1.FileName;
   Form1.Caption := fName;
   AssignFile(f, fName);

   try
       Reset(f);  // открыть для чтения
   except
        on EInOutError do
```

```
        begin
            ShowMessage('Ошибка доступа к файлу '+
                            fName);
            exit;
        end;
    end;

    // чтение из файла
    while not EOF(f) do
      begin
        readln(f, buf);        // прочитать строку из файла
        Memo1.Lines.Add(buf); // добавить строку в поле Memo1
      end;

    CloseFile(f); // закрыть файл
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    // определим фильтр
    OpenDialog1.Filter := 'Текст|*.txt';
end;

end.
```

52. Напишите программу, которая добавляет в базу данных "Погода", реализованную в виде текстового файла, информацию о дневной температуре. Для ввода даты используйте компонент `MonthCalendar`. Если файл данных отсутствует, то программа должна его создать. Рекомендуемый вид формы программы приведен на рис. 1.53.

```
implementation

{$R *.dfm}
const
    DBNAME = 'pogoda.txt';
var
    db: TextFile; // файл — база данных

// Начало работы. Откроем или создадим
// файл данных.
procedure TForm1.FormCreate(Sender: TObject);
```

Рис. 1.53. Форма программы базы данных "Погода"

```
var
    r: integer; // ответ пользователя
begin
    AssignFile(db,DBNAME);
    try
        Append(db); // возможна ошибка
    except
        on E: EInOutError do
        begin
            r := MessageDlg('Файл базы данных (pogoda.txt) +
            'не найден.' +
            #13+'Создать файл?',mtWarning,[mbOk,mbCancel],0);
            if  r = mrOK
                then begin
                    Rewrite(db);  // создадим файл
                    ShowMessage('Файл базы данных создан!');
                end
            else Application.Terminate; // завершить работу
        end;
    end;
    Edit1.Enabled := True;
    Button1.Enabled := True;
end;
```

```
// нажатие клавиши в поле Температура
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    case Key of
        '0'..'9',#8:;  // цифры и клавиша <Backspace>

        #13:    // клавиша <Enter>
                if Length(Edit1.Text) <> 0
                    then Button1.SetFocus;

        ',','.': begin // десятичная точка
                    Key:= DecimalSeparator;
                    if Pos(DecimalSeparator,Edit1.Text) <> 0
                        then Key :=Char(0);
                 end;

        '-':     if Length(Edit1.Text) <> 0
                    then Key:=Char(0);

        else Key:= Char(0); // остальные символы запрещены
    end;

end;

// щелчок на кнопке Добавить
procedure TForm1.Button1Click(Sender: TObject);
begin
    if Length(Edit1.Text) = 0
      then ShowMessage('Надо ввести температуру.')
      else begin
           writeln(db,
                   FormatDateTime('dd/mm/yy',
                                  MonthCalendar1.Date),
                  ' ',Edit1.Text);
           Edit1.Text := '';
        end;
end;

// завершение работы программы
procedure TForm1.FormClose(Sender: TObject;
                           var Action: TCloseAction);
begin
        CloseFile(db); // закрыть файл
end;

end.
```

53. Напишите программу, которая в указанном пользователем каталоге и его подкаталогах выполняет поиск файла. Рекомендуемый вид диалогового окна приведен на рис. 1.54. Для ввода имени каталога во время работы программы используйте стандартное диалоговое окно **Обзор папок** (рис. 1.55).

```
// Поиск файла в указанном каталоге и его подкаталогах
// используется рекурсивная процедура Find.
unit FindFile_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,  Dialogs, StdCtrls, FileCtrl;
type
  TForm1 = class(TForm)
    Edit1: TEdit;       // что искать
    Edit2: TEdit;       // где искать
    Memo1: TMemo;       // результат поиска
    Button1: TButton;   // кнопка Поиск
    Button2: TButton;   // кнопка Папка
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

var
   FileName: string;  // имя или маска искомого файла
   cDir: string;
   n: integer;       // кол-во файлов, удовлетворяющих запросу
```

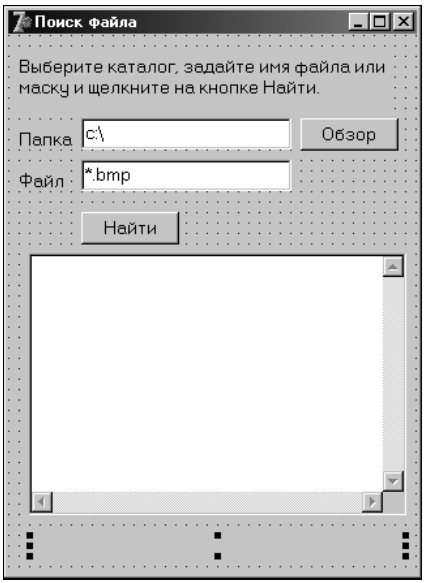


Рис. 1.54. Форма программы **Поиск файла**

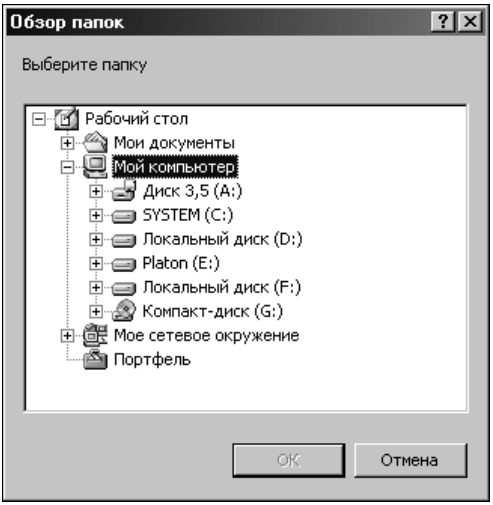


Рис. 1.55. Стандартное окно **Обзор папок**

```
// поиск файла в текущем каталоге
procedure Find;
var
   SearchRec: TSearchRec; // информация о файле или каталоге
begin
   GetDir(0,cDir); // получить имя текущего каталога
   if cDir[length(cDir)] <> '\' then cDir := cDir+'\';

   if FindFirst(FileName, faAnyFile,SearchRec) = 0 then
       repeat
          if (SearchRec.Attr and faAnyFile) = SearchRec.Attr
          then begin
                Form1.Memo1.Lines.Add(cDir +  SearchRec.Name);
                n := n + 1;
               end;
       until FindNext(SearchRec) <> 0;

    // обработка подкаталогов текущего каталога

    { Если не обрабатывать системные каталоги, то
      вместо faAnyFile можно задать faDirectory.
      Это объясняется тем, что значение SearchRec.Attr
      для обычного каталога равно faDirectory (16),
      для Program Files — faDirectory+faReadOnly (17),
     а для каталога Windows — faDirectory+faSysFile (20) }

    if FindFirst('*', faAnyFile, SearchRec) = 0 then
       repeat
         if (SearchRec.Attr and faDirectory) = faDirectory
            then
               // каталоги '..' и '.' тоже каталоги,
               // но в них входить не надо !!!
               if SearchRec.Name[1] <> '.' then
                 begin
                   ChDir(SearchRec.Name); // войти в каталог
                   Find; // выполнить поиск в подкаталоге
                   ChDir('..');// выйти из каталога
                 end;
       until FindNext(SearchRec) <> 0;
end;

// щелчок на кнопке Поиск
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
    if not DirectoryExists(Edit2.Text) then
       begin
            ShowMessage('Каталог указан неверно.');
            Edit2.SetFocus;
            exit;
       end;

    Button1.Enabled := False;
    Label4.Caption := '';
    Label4.Repaint;
    Memo1.Clear;             // очистить поле Memo1
    Label4.Caption := '';
    FileName := Edit1.Text;  // что искать
    cDir := Edit2.Text;      // где искать
    n:=0;                    // кол-во найденных файлов
    ChDir(cDir);             // войти в каталог начала поиска
    Find;                    // начать поиск
    if n = 0 then
       ShowMessage('Файлов, удовлетворяющих критерию ' +
                   'поиска нет.')
       else Label4.Caption := 'Найдено файлов:' + IntToStr(n);
    Button1.Enabled := True;
end;

// возвращает каталог, выбранный пользователем
function GetPath(mes: string):string;
var
  Root: string;      // корневой каталог
  pwRoot : PWideChar;
  Dir: string;
begin
  Root := ''; // корневой каталог — папка Рабочий стол
  GetMem(pwRoot, (Length(Root)+1) * 2);
  pwRoot := StringToWideChar(Root,pwRoot,MAX_PATH*2);
  if SelectDirectory(mes, pwRoot, Dir)
     then
          if length(Dir) = 2  // выбран корневой каталог
              then GetPath := Dir+'\'
              else GetPath := Dir
     else
          GetPath := '';
end;
```

```
// щелчок на кнопке Папка
procedure TForm1.Button2Click(Sender: TObject);
var
  Path: string;
begin
  Path := GetPath('Выберите папку');
  if Path <> ''
     then Edit2.Text := Path;
end;

end.
```

Игры и полезные программы

54. Всем известна игра "15". Вот ее правила. В прямоугольной коробочке находятся 15 фишек, на которых написаны числа от 1 до 15. Размер коробочки — 4×4, таким образом в коробочке есть одна пустая ячейка. В начале игры фишки перемешаны (рис. 1.56). Задача игрока состоит в том, чтобы, не вынимая фишки из коробочки, выстроить фишки в правильном порядке (рис. 1.57).

5	2	1	4
9	6	8	14
3	15	11	17
12		13	10

Рис. 1.56. В начале игры фишки перемешаны

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Рис. 1.57. Правильный порядок фишек

Напишите программу "Игра “15”". На рис. 1.58 приведена форма и окна программы.

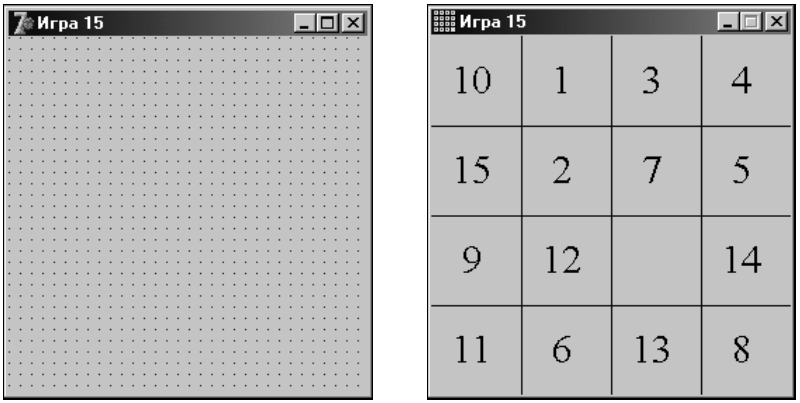


Рис. 1.58. Форма и окна программы **Игра 15**

```
{ Игра "15"}
unit game15_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormMouseDown(Sender: TObject;
                            Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormPaint(Sender: TObject);

    // эти объявления вставлены сюда вручную
    procedure ShowPole;
    procedure Mixer;

  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}
const
    H = 4;   W = 4;   // размер поля — 4x4
    CH = 64; CW = 64; // размер клеток — 16x16

var
    // правильное расположение фишек
    stp : array[1..H, 1..W] of byte =
    (( 1, 2, 3, 4),
     ( 5, 6, 7, 8),
     ( 9,10,11,12),
     (13,14,15, 0));

    // игровое поле
    pole: array[1..H, 1..W] of byte;

    ex,ey: integer; // координаты пустой клетки

// новая игра
procedure NewGame;
var
    i,j: integer;
begin
    // исходное (правильное) положение
    for i:=0 to H+1 do
        for j:=0 to W+1 do
            pole[i,j] := stp[i,j];
    Form1.Mixer;    // перемешать фишки
    Form1.ShowPole; // отобразить поле
end;

// проверяет, расположены ли
// фишки в нужном порядке
function Finish: boolean;
var
    row,col: integer;
    i: integer;
begin
    row :=1; col :=1;
    Finish := True; // пусть фишки в нужном порядке
```

```
    for i:=1 to 15 do
    begin
       if pole[row,col] <> i then
       begin
            Finish:= False;
            break;
       end;
       // к следующей клетке
       if col < 4
          then inc(col)
       else begin
          col :=1;
          inc(row);
       end;
    end;
end;

// "перемещает" фишку в соседнюю пустую клетку,
// если она есть, конечно
procedure Move(cx,cy: integer);
// cx,cy — клетка, в которой игрок сделал щелчок
var
    r: integer;       // выбор игрока
begin
    // проверим, возможен ли обмен
    if not (( abs(cx-ex) = 1) and (cy-ey = 0) or
            ( abs(cy-ey) = 1) and (cx-ex = 0))
    then exit;
    // Обмен. Переместим фишку из x,y в ex,ey
    Pole[ey,ex] := Pole[cy,cx];
    Pole[cy,cx] := 0;
    ex:=cx;
    ey:=cy;
    // отрисовать поле
    Form1.ShowPole;
    if Finish then
    begin
       r := MessageDlg('Цель достигнута!'+ #13+
                    'Еще раз?',mtInformation,[mbYes,mbNo],0);
       if r = mrNo then Form1.Close; // завершить работу программы
    end;

end;
```

```
// щелчок в клетке
procedure TForm1.FormMouseDown(Sender: TObject;
                               Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
    cx,cy: integer; // координаты клетки
begin
    // преобразуем координаты мыши в координаты клетки
    cx := Trunc(X / CW) + 1;
    cy := Trunc(Y / CH) + 1;
    Move(cx,cy);
end;

// выводит игровое поле
procedure TForm1.ShowPole;
var
    i,j: integer;
    x,y: integer; // x,y — координаты вывода
                  //  текста в клетке
begin
    // сетка: вертикальные линии
    for i:= 1 to W — 1 do
    begin
        Canvas.MoveTo(i*CW,0);
        Canvas.LineTo(i*CW,ClientHeight);
    end;
    // сетка: горизонтальные линии
    for i:= 1 to H — 1 do
    begin
        Canvas.MoveTo(0,i*CH);
        Canvas.LineTo(ClientWidth,i*CH);
    end;

    // содержимое клеток
    // x,y — координаты вывода текста
    for i:= 1 to H do
    begin
       y:=(i-1)*CH + 15;
       for j:=1 to W do
       begin
         x:= (j-1)*CW + 15;
```

```
        case Pole[i,j] of
          0:    Canvas.TextOut(x,y,'     ');
          1..9: Canvas.TextOut(x,y,' '+
                              IntToStr(Pole[i,j])+'  ');
          10..15: Canvas.TextOut(x,y,IntToStr(Pole[i,j]));
        end;
      end;
    end;
end;

// "перемешивает" фишки
procedure TForm1.Mixer;
var
    x1,y1: integer; // пустая клетка
    x2,y2: integer; // эту переместить в пустую
    d: integer;     // направление, относительно пустой
    i: integer;
begin
    x1:=4;
    y1:=4;
    randomize;
    for i:= 1 to 150 do
    begin
        repeat
            x2:=x1;
            y2:=y1;
            d:=random(4)+1;
            case d of
                1: dec(x2);
                2: inc(x2);
                3: dec(y2);
                4: inc(y2);
            end;
        until (x2>=1) and (x2<=4) and (y2>=1) and (y2<=4);
        // здесь определили фишку, которую
        // надо переместить в пустую клетку
        Pole[y1,x1] := Pole[y2,x2];
        Pole[y2,x2] := 0;
        x1:=x2;
        y1:=y2;
    end;
```

```
    // запомним координаты пустой клетки
    ex:= x1;
    ey:= y1;
end;

// обработка события OnCreate
procedure TForm1.FormCreate(Sender: TObject);
begin
    ClientWidth := CW * W;
    ClientHeight := CH * H;
    Canvas.Font.Name := 'Times New Roman';
    Canvas.Font.Size := 22;
    NewGame;
end;

// обработка события OnPaint
procedure TForm1.FormPaint(Sender: TObject);
begin
    Form1.ShowPole;
end;

end.
```

55. Напишите программу "Собери картинку" — аналог игры "15", в которой игрок будет перемещать не цифры, а фрагменты картинки (рис. 1.59).

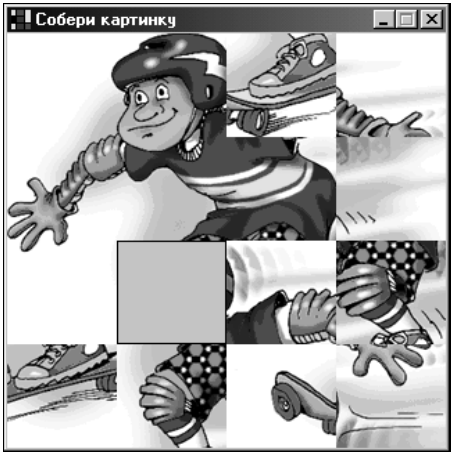


Рис. 1.59. Окно программы **Собери картинку**

```
{
  Игра "Собери картинку" — игра "15" с графическим
  интерфейсом (вместо цифр — фрагменты картинки).
  Картинка загружается из файла pic_1.bmp,
  который находится в том же каталоге
  что и выполняемый файл программы.
  (c) Культин Н.Б., 2003.
}
unit soberi_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormPaint(Sender: TObject);

    // эти объявления вставлены сюда вручную
    procedure ShowPole;
    procedure Mixer;
    procedure NewGame;

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

const
    H = 4;   W = 4;   // размер поля — 4x4
```

```
var

    wc,hc: integer; // ширина и высота клетки

    // игровое поле
    pole: array[1..H, 1..W] of byte;
    ex,ey: integer; // координаты пустой клетки

    // правильное расположение клеток
    stp : array[1..H, 1..W] of byte =
    (( 1, 2, 3, 4),
     ( 5, 6, 7, 8),
     ( 9,10,11,12),
     (13,14,15, 0));

     pic: TBitmap;   // картинка

// новая игра
procedure TForm1.NewGame;
var
    fname: string[20]; // файл картинки
    i,j: integer;
begin
    { сюда можно вставить инструкции,
    обеспечивающие, например, случайный
    выбор загружаемой картинки }

    fname := 'pic_1.bmp';
    try
      pic.LoadFromFile(fname);
      except
       on EFopenError do
           begin
             ShowMessage('Ошибка обращения к файлу ' + fname);
                Form1.Close;
           end;
    end;

    // установить размер формы,
    // равный размеру картинки

    // размер клетки
    hc := Pic.Height div H;
    wc := Pic.Width div W;
```

```
    // размер формы
    ClientWidth  := wc * W;
    ClientHeight := hc * H;

    // исходное (правильное) положение
    for i:=1 to H do
        for j:=1 to W do
            pole[i,j] := stp[i,j];
    Form1.Mixer;    // перемешать фишки
    Form1.ShowPole; // отобразить поле
end;

// проверяет, расположены ли
// клетки (фрагменты картинки) в нужном порядке
function Finish: boolean;
var
    row,col: integer;
    i: integer;
begin
    row :=1; col :=1;
    Finish := True; // пусть фишки в нужном порядке
    for i:=1 to 15 do
    begin
       if pole[row,col] <> i then
       begin
            Finish:= False;
            break;
       end;
       // к следующей клетке
       if col < 4
          then inc(col)
       else begin
          col :=1;
          inc(row);
       end;
    end;
end;


// "перемещает" фишку в соседнюю пустую клетку,
// если она есть, конечно
procedure Move(cx,cy: integer);
```

```
// cx,cy — клетка, в которой игрок сделал щелчок
var
    r: integer;       // выбор игрока
begin
    // проверим, возможен ли обмен
    if not (( abs(cx-ex) = 1) and (cy-ey = 0) or
            ( abs(cy-ey) = 1) and (cx-ex = 0))
    then exit;
    // Обмен. Переместим фишку из x,y в ex,ey
    Pole[ey,ex] := Pole[cy,cx];
    Pole[cy,cx] := 0;
    ex:=cx;
    ey:=cy;
    // отрисовать поле
    Form1.ShowPole;
    if Finish then
    begin
        pole[4,4] := 16;
        Form1.ShowPole;
        r := MessageDlg('Цель достигнута!'+ #13+
        'Еще раз?',mtInformation,[mbYes,mbNo],0);
        if r = mrNo then Form1.Close; // завершить работу
                                      // программы
        Form1.NewGame;
    end;

end;

// щелчок в клетке
procedure TForm1.FormMouseDown(Sender: TObject;
                               Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
    cx,cy: integer; // координаты клетки
begin
    // преобразуем координаты мыши в координаты клетки
    cx := Trunc(X / wc) + 1;
    cy := Trunc(Y / hc) + 1;
    Move(cx,cy);
end;

// выводит игровое поле
procedure TForm1.ShowPole;
```

```
var
    Source, Dest: TRect; // области: источник и приемник
    sx,sy: integer;      // левый верхний угол области-источника
    i,j: integer;

begin

    // содержимое клеток
    for i := 1 to W do
        for j := 1 to H do
        begin
            // преобразуем номер картинки
            // в координаты левого верхнего
            // угла области-источника
            sy := ((pole[i,j] -1) div W) * hc;
            sx := ((pole[i,j] -1) mod W) * wc;

            Source := Bounds(sx,sy,wc,hc);
            Dest := Bounds((j-1)*wc,(i-1)*hc,wc,hc);
            if pole[i,j] <> 0
                then Canvas.CopyRect(Dest,pic.Canvas,Source)
                else Canvas.Rectangle((j-1)*wc, (i-1)*hc,
                                      j*wc, i*hc);
        end;
end;

// "перемешивает" фишки
procedure TForm1.Mixer;
var
    x1,y1: integer; // пустая клетка
    x2,y2: integer; // эту переместить в пустую
    d: integer;     // направление, относительно пустой
    i: integer;
begin
    x1:=4; y1:=4; // см. описание массива stp
    randomize;
    for i:= 1 to 150 do  // кол-во перестановок
    begin
        repeat
            x2:=x1;
            y2:=y1;
            d:=random(4)+1;
```

```
            case d of
                1: dec(x2);
                2: inc(x2);
                3: dec(y2);
                4: inc(y2);
            end;
        until (x2>=1) and (x2<=4) and (y2>=1) and (y2<=4);
        // здесь определили фишку, которую
        // надо переместить в пустую клетку
        Pole[y1,x1] := Pole[y2,x2];
        Pole[y2,x2] := 0;
        x1:=x2;
        y1:=y2;
    end;
    // запомним координаты пустой клетки
    ex:= x1;
    ey:= y1;
end;

// обработка события OnCreate
procedure TForm1.FormCreate(Sender: TObject);
begin
    pic := TBitMap.Create;
    NewGame;
end;

// обработка события OnPaint
procedure TForm1.FormPaint(Sender: TObject);
begin
    Form1.ShowPole;
end;

end.
```

56. Напишите программу, используя которую можно оценить способность игрока (испытуемого) запоминать числа. Программа должна выводить числа, а испытуемый — вводить эти числа с клавиатуры. Время, в течение которого игрок будет видеть число, ограничьте, например, одной секундой. По окончании теста программа должна вывести результат: количество показанных чисел и количество чисел, которые испытуемый запомнил и ввел правильно. Вид формы приведен на рис. 1.60.

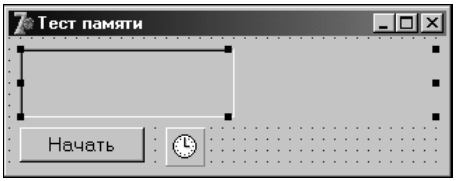


Рис. 1.60. Форма программы **Тест памяти**

```
implementation

const
  KC = 5;   // разрядность числа (кол-во цифр)
  LT = 10;  // количество чисел (длина теста)

var
  numb: integer;   // число, которое должен запомнить испытуемый
  right: integer;  // количество правильных чисел
  n: integer;      // счетчик чисел

{$R *.dfm}

// генерирует k — разрядное число
function GetNumb(k: integer) : integer;
var
  n: integer; // генерируемое число
  i: integer;
begin
  // процедура генерирует число по разрядам
  // начиная со старшего

  n:= Random(9)+1; // старший разряд не может быть нулем
  // остальные разряды
  for i := 1 to (k-1) do
    n := n*10 + Random(10);

  GetNumb := n;
end;

// создание формы
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit1.Visible := False;  // скрыть поле ввода Edit1
  Edit1.MaxLength := KC;   // кол-во символов, которое
                           // можно ввести
```

```
  Label1.WordWrap := True; // разрешить перенос слов на
                           // следующую строку

  Label1.Caption :=
  'Сейчас на экране будут появляться числа. ' +
  'Вы должны запомнить число, набрать его на клавиатуре '+
  'и нажать <Enter>';
  Button1.Caption := 'Начать';
  Timer1.Enabled := False;  // таймер остановлен
  Timer1.Interval := 1000;  // время показа числа — 1 секунда

  right := 0;     // кол-во правильных
  n := 0;     // счетчик чисел
  Randomize;      // инициализация ГСЧ
end;

// щелчок на кнопке "Начать/Завершить"
procedure TForm1.Button1Click(Sender: TObject);
begin
  if Button1.Caption = 'Завершить' then
     Form1.Close; // закрыть окно программы

  if Button1.Caption = 'Начать' then
  begin
      Button1.Caption := 'Завершить';
      Button1.Visible := False;  // скрыть кнопку
      // кнопка Button1 станет доступной после того,
      // как испытание закончится
      Label1.Caption := '';

      Label1.Font.Size := 24;  // размер шрифта поля Label1
      Edit1.Font.Size := 24;   // размер шрифта поля Edit1

      // сгенерировать и вывести число
      numb := GetNumb(KC);
      Label1.Caption := IntToStr(numb);

      Timer1.Enabled := True;       // запуск таймера
      // процедура обработки сигнала от таймера
      // "сотрет" число
    end;
end;

// обработка события таймера
procedure TForm1.Timer1Timer(Sender: TObject);
```

```
begin
  Timer1.Enabled := False; // остановить таймер
  Label1.Visible := False; // скрыть число
  Edit1.Visible := True;   // сделать доступным поле Edit1
  Edit1.SetFocus;          // установить курсор в поле Edit1
end;

// нажатие клавиш в поле Edit1
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
var
  igrok: integer;  // число, которое ввел испытуемый
begin
    case Key of
        '0'..'9',#8: ; // клавиши "0"-"9", клавиша <Backspace>

        #13:           // клавиша <Enter>
    begin
       igrok := StrToInt(Edit1.Text);
       if (igrok = numb)
          then right := right + 1;

       n := n + 1; // счетчик чисел

       Edit1.Text := '';
       Edit1.Visible := False;   // скрыть поле Edit1

       if n < LT then
       begin
         numb := GetNumb(KC); // сгенерировать следующее
                              // число
         Label1.Caption := IntToStr(numb); // отобразить
                                 // число
         Label1.Visible := True;
         Timer1.Enabled := True;           // пуск таймера
       end
       else begin
          // испытание закончено
          // вывести результат

          Label1.Font.Size := 10;
          Label1.Caption := 'Результат:' + #13 +
            'Показано чисел:  ' + IntToStr(LT) + #13 +
            'Правильных:  ' + IntToStr(right);
```

```
            Label1.Visible := True;
            Button1.Visible := True; // показать кнопку Завершить
          end;
      end;
      else Key := Chr(0);
      end;
end;

end.
```

57. Напишите программу, используя которую можно оценить способность игрока (испытуемого) запоминать числа. Программа должна выводить числа, а испытуемый — вводить эти числа с клавиатуры. Время, в течение которого игрок будет видеть число, ограничьте, например, одной секундой. Программа должна быть "интеллектуальной". Сначала она должна предлагать запоминать четырехразрядные числа, затем пяти, шести и т. д. Количество чисел одной разрядности — 10. Переход на следующий уровень сложности (увеличение разрядности числа) должен выполняться, если количество правильных чисел больше, например, восьми, или количество подряд правильно введенных чисел больше шести. По окончании теста программа должна вывести результат по каждой подгруппе: количество показанных чисел и количество чисел, которые испытуемый запомнил и ввел правильно.

58. Игра "Парные картинки" развивает внимание. Вот ее правила. Игровое поле разделено на клетки, за каждой из которых скрыта картинка. Картинки — парные, т. е. на игровом поле есть две клетки, в которых находятся одинаковые картинки. В начале игры все клетки "закрыты". Щелчок левой кнопкой мыши "открывает" клетку, в клетке появляется картинка. Теперь надо найти клетку, в которой находится такая же картинка, как и в открытой клетке. Щелчок к другой клетке открывает вторую картинку (рис. 1.61). Если картинки в открытых клетках одинаковые, то эти клетки "исчезают". Если разные — то клетки остаются открытыми. Очередной щелчок закрывает открытые клетки и открывает следующую. Следует обратить внимание, что две открытые клетки закрываются даже в том случае, если открытая картинка такая же, как и одна из двух открытых. Игра заканчивается, когда игрок откроет ("найдет") все пары картинок.

Рис. 1.61. Игровое поле программы **Парные картинки**

Разработайте программу, реализующую игру "Парные картинки". Картинки должны загружаться из файла.

В приведенной реализации игры все картинки квадратные и находятся в одном файле (рис. 1.62). Это позволило сделать программу "интеллектуальной" — размер игрового поля (количество клеток по горизонтали и вертикали) определяется количеством картинок в файле: зная высоту и ширину картинки в файле, программа вычисляет размер картинок, их количество и устанавливает соответствующий размер игрового поля.

Вид формы программы "Парные картинки" приведен на рис. 1.63.

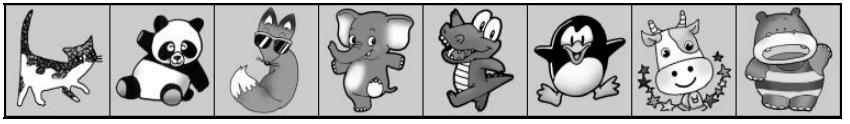


Рис. 1.62. Все картинки находятся в одном файле

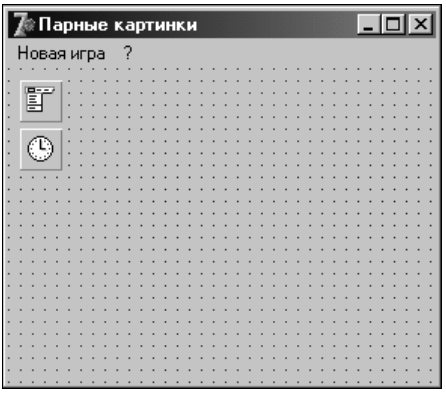


Рис. 1.63. Форма программы **Парные картинки**

```
unit dblpic_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,  Dialogs, StdCtrls, jpeg, ExtCtrls, Menus;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure Timer1Timer(Sender: TObject);
    procedure N1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
  // объявление нового типа col_row
  col_row = record
    col: integer;
    row: integer;
  end;

const


  MAX_SIZE = 32; // максимальное кол-во парных картинок
  MAX_H = 8;     // максимальный размер поля — 8x8
  MAX_W = 8;

var
  Form1: TForm1;

  Pole: array [1..MAX_H,1..MAX_W] of integer;
  { Pole[i,j] < 100 — код картинки, клетка закрыта;
    Pole[i,j] > 100 и < 200 — клетка открыта,
                              игрок видит картинку;
    Pole[i,j] > 200 — игрок нашел пару для этой картинки }

  Pictures: TBitmap;    // картинки, загруженные из файла

  n : integer;     // кол-во открытых пар картинок
  count: integer;  // кол-во открытых в данный момент клеток
  open1: col_row;  // координаты 1-й открытой клетки
  open2: col_row;  // координаты 2-й открытой клетки

  W: integer;    // кол-во клеток по горизонтали
  H: integer;    // кол-во клеток по вертикали
            // произведение W и H должно быть кратно двум
  WK: integer;  // ширина клетки
  HK: integer;  // высота клетки


implementation

{$R *.dfm}

// рисует клетку поля
procedure Kletka(col,row: integer);
var
  x,y: integer;     // левый верхний угол клетки (координаты)
  src, dst : Trect; // источник и получатель битового образа
```

```
begin
  // преобразуем координаты клетки
  // в координаты на поверхности формы
  x := (col-1)*WK;
  y := (row-1)*HK;


  if Pole[col,row] > 200 then
   // для этой клетки найдена пара
   // клетку надо убрать с поля
   begin
      // установить цвет границы, закраски и текста
      Form1.Canvas.Brush.Color := clBtnFace;
      Form1.Canvas.Pen.Color := clBtnFace;
      Form1.Canvas.Font.Color := clBtnFace;
    end;


  if (Pole[col,row] > 100) and (Pole[col,row] < 200)
   then
       // клетка открыта — вывести картинку
       begin

      // Pole[col,row] = номер картинки + 100,
      // где 100 — признак того, что клетка открыта
      // определим положение картинки в Pictures
      src := Bounds((Pole[col,row]-100 -1)*WK,0,WK,HK);

      // координаты картинки (клетки) на форме
      dst := Bounds(x,y,HK-2,WK-2);

      // вывести картинку в клетку
      Form1.Canvas.CopyRect(dst,Pictures.Canvas,src);

      // установить цвет границы и цифры
      Form1.Canvas.Pen.Color := clBlack;
      Form1.Canvas.Font.Color := clBlack;
      Form1.Canvas.Brush.Style := bsClear;
  end;


  if (Pole[col,row] > 0) and (Pole[col,row] < 100) then
    // клетка закрыта, рисуем только контур
    begin
      Form1.Canvas.Brush.Color := clBtnFace;
```

```
      Form1.Canvas.Pen.Color := clBlack;
      Form1.Canvas.Font.Color := clBtnFace;
    end;

  // отрисовать клетку
  Form1.Canvas.Rectangle(x,y,x+WK-2,y+HK-2);
  //Form1.Canvas.TextOut(x+15,y+15, IntToStr(Pole[col,row]));
  Form1.Canvas.Brush.Color := clBtnFace;

end;

// отрисовывает поле
procedure ShowPole;
var
   row,col: integer;
begin
   for row:=1 to H do
      for col:=1 to W do
           Kletka(row,col);
end;

// новая игра
Procedure NewGame;
var

  k: integer;       // кол-во парных картинок
  r: integer;       // случайное число
  buf: array[1..MAX_SIZE] of integer;
  // в buf[i] записываем, сколько чисел i
  // записали в массив Pole
  i,j: integer; // индексы массивов
begin
  Randomize;
  k := Trunc(H*W/2);

  for i:=1 to k do
      buf[i] := 0;

  // запишем в массив Pole случайные числа
  // от 1 до 2
  // каждое число должно быть записано два раза
  for i:=1 to H do
    for j:=1 to W do
```

```
      begin
        repeat
          r := random (k) + 1;
        until buf[r] < 2;
        Pole[i,j] := r;   // код картинки
        inc(buf[r]);
      end;
   // здесь поле сгенерировано
   n:=0;
   ShowPole;
end;

// создание формы
procedure TForm1.FormCreate(Sender: TObject);
var
  np: integer; // кол-во парных картинок
begin
  Pictures := TBitmap.Create;
  // загрузить картинки из файла
  Pictures.LoadFromFile('pictures.bmp');

  HK := Pictures.Height-1; // высота картинки
  WK := HK;                // ширина картинки

  np:= Round(Pictures.Width / WK);
  if np <= 15
    then H := 4
    else H :=5;
  W := Round(np*2/H);

  // установить размеры поля
  Form1.ClientHeight := H * HK;
  Form1.ClientWidth := W * WK;


  Form1.Timer1.Enabled := False;
  Form1.Timer1.Interval := 200;

  n := 0;
  NewGame;
end;

// прорисовка клеток на поле
procedure TForm1.FormPaint(Sender: TObject);
```

```
begin
    ShowPole;
end;

// щелчок в клетке
procedure TForm1.FormMouseDown(Sender: TObject;
                               Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
var
  col_: integer;   // номер клетки по горизонтали
  row_: integer;   // номер клетки по вертикали

begin
  col_ := Trunc(X/WK) + 1;
  row_ := Trunc(Y/HK) + 1;


  if Pole[col_,row_] > 200 then
     // щелчок в месте одной из двух
     // уже найденных парных картинок
  exit;

  // открытых клеток нет
  if count = 0 then
  begin
    count := 1;
    open1.col := col_;
    open1.row := row_;

    // клетка помечается как открытая
    Pole[open1.col,open1.row] := Pole[open1.col,open1.row] + 100;
    Kletka(open1.col,open1.row);
    exit;
  end;

  // открыта одна клетка, надо открыть вторую
  if count = 1 then begin
    open2.col := col_;
    open2.row := row_;

    // если открыта одна клетка и щелчок сделан
    // в этой клетке, то ничего не происходит
    if (open1.col = open2.col) and (open1.row = open2.row)
       then exit
```

```
    else begin
      count := 2; // теперь открыты две клетки
      Pole[open2.col,open2.row] :=
           Pole[open2.col,open2.row] + 100;
      Kletka(open2.col,open2.row); // отрисуем вторую клетку

      // проверим, открытые картинки одинаковые?
      if Pole[open1.col,open1.row] = Pole[open2.col,open2.row] then
        // открыты две одинаковые картинки
        begin
          n := n+1;
          Form1.Timer1.Enabled := True; // запустить таймер
          // процедура обработки события OnTimer
          // "сотрет" две одинаковые картинки
        end;
    end;
    exit;
  end;

  if count = 2 then
  begin
    // открыты 2 клетки с разными картинками
    // закроем их и откроем новую, в которой
    // сделан щелчок

    // закрыть открытые клетки
    Pole[open1.col,open1.row] := Pole[open1.col,open1.row] — 100;
    Pole[open2.col,open2.row] := Pole[open2.col,open2.row] —
100;
    Kletka(open1.col,open1.row);
    Kletka(open2.col,open2.row);

    // запись в open1 номера текущей клетки
    open1.col := col_;
    open1.row := row_;
    count := 1;   // счетчик открытых клеток

    // открыть текущую клетку
    Pole[open1.col,open1.row] := Pole[open1.col,open1.row] + 100;
    Kletka(open1.col,open1.row);
  end;
end;
```

```
// обработка события таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  // в массиве Pole клетки помечаются как совпавшие
  Pole[open1.col,open1.row] := Pole[open1.col,open1.row] + 100;
  Pole[open2.col,open2.row] := Pole[open2.col,open2.row] + 100;
  count := 0;

  // отрисовать клетки
  Kletka(open2.col,open2.row);
  Kletka(open1.col,open1.row);

  // остановка таймера
  Form1.Timer1.Enabled := False;

  if n = Trunc(W*H/2)
  then // открыты все пары
  begin
    Form1.Canvas.Font.Name := 'Times New Roman';
    Form1.Canvas.Font.Size := 36;
    Form1.Canvas.Font.Color := clBlack;
    Form1.Canvas.TextOut(70,160,'Game Over!');
    Form1.Canvas.Font.Size := 10;
    Form1.Canvas.TextOut(120,210,'(c) Культин П.Н., 2003');
  end;
end;

// выбор в меню команды Новая игра
procedure TForm1.N1Click(Sender: TObject);
begin
    Canvas.Rectangle(0,0,ClientWidth,ClientHeight);
    NewGame;
end;

end.
```

59. Хорошо знакомая всем пользователям Windows игра "Сапер" развивает логическое мышление. Вот правила игры. Игровое поле состоит из клеток, в каждой из которых может быть мина. Задача игрока — найти все мины и пометить их флажками. Используя кнопки мыши, игрок может открыть клетку или поставить в нее флажок, указав тем самым, что в клетке находится

мина. Клетка открывается щелчком левой кнопки мыши, флажок ставится щелчком правой. Если в клетке, которую открыл игрок, есть мина, то происходит взрыв (сапер ошибся, а он, как известно, ошибается только один раз) и игра заканчивается (рис. 1.64). Если в клетке мины нет, то в этой клетке появляется число, соответствующее количеству мин, находящихся в соседних клетках. Анализируя информацию о количестве мин в клетках, соседних с уже открытыми, игрок может обнаружить и пометить флажками все мины. Ограничений на количество клеток, помеченных флажками, нет. Однако для завершения игры (выигрыша) флажки должны быть установлены только в тех клетках, в которых есть мины. Ошибочно установленный флажок можно убрать, щелкнув правой кнопкой мыши в клетке, в которой он находится.

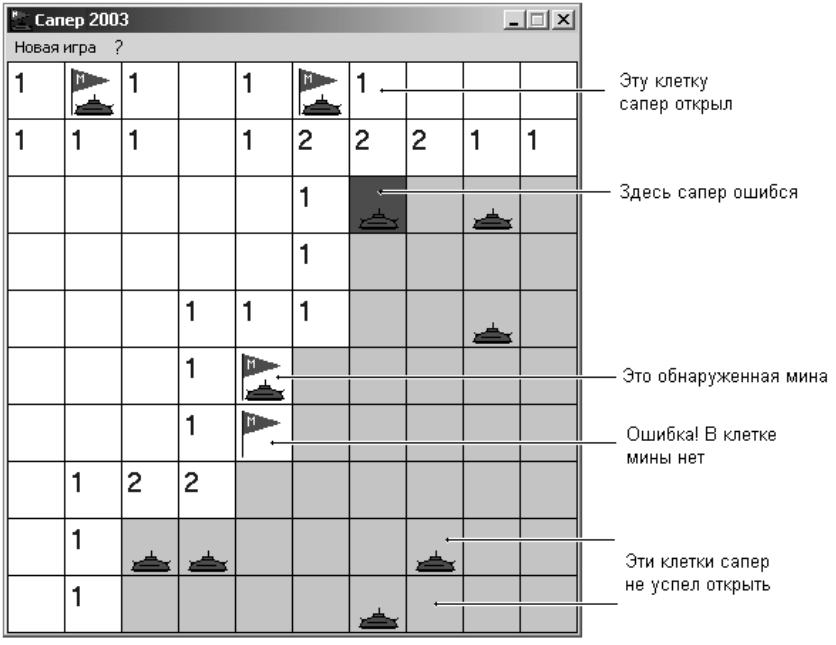


Рис. 1.64. Вид окна во время (в конце) игры

Разработайте программу, реализующую игру "Сапер". Вид главной формы приведен на рис. 1.65.

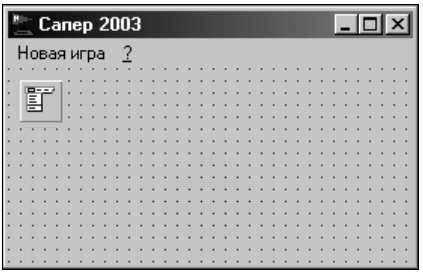


Рис. 1.65. Главная форма программы **Сапер**

```
// модуль главной формы
unit saper_1;
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs,  Menus, StdCtrls, OleCtrls;

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;

    procedure Form1Create(Sender: TObject);
    procedure Form1Paint(Sender: TObject);
    procedure Form1MouseDown(Sender: TObject; Button:
TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure N1Click(Sender: TObject);

    procedure N4Click(Sender: TObject);
    procedure N3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation

uses saper_2;

{$R *.DFM}

const
   MR = 10; // кол-во клеток по вертикали
   MC = 10; // кол-во клеток по горизонтали
   NM = 10; // кол-во мин

   W    = 40;  // ширина клетки поля
   H    = 40;  // высота клетки поля

var
   Pole: array[0..MR+1, 0.. MC+1] of integer; // минное поле
   // значение элемента массива:
   // 0..8 — количество мин в соседних клетках
   // 9 — в клетке мина
   // 100..109 — клетка открыта
   // 200..209 — в клетку поставлен флаг

   nMin  : integer;  // кол-во найденных мин
   nFlag : integer;  // кол-во поставленных флагов

   status : integer; // 0 — начало игры; 1 — игра; 2 — результат

// генерирует новое поле
Procedure NewGame(); forward;

// показывает поле
Procedure ShowPole(Canvas : TCanvas; status:integer); forward;

// выводит содержимое клетки
Procedure Kletka(Canvas: TCanvas; row,col,status :integer);
          forward;

// открывает текущую и все соседние клетки, в которых нет мин
Procedure Open( row, col : integer); forward;

// рисует мину
Procedure Mina(Canvas : TCanvas; x, y : integer); forward;

// рисует флаг
Procedure Flag( Canvas : TCanvas; x, y : integer); forward;
```

```
// выводит на экран содержимое клетки
Procedure Kletka(Canvas : TCanvas; row,col,status : integer);
    var
         x,y : integer; // координаты области вывода
    begin
       x := (col-1)* W + 1;
       y := (row-1)* H + 1;

       if status = 0  then
          begin
            Canvas.Brush.Color := clLtGray;
            Canvas.Rectangle(x-1,y-1,x+W,y+H);
            exit;
          end;

       if Pole[row,col] < 100  then
          begin
           // не открытые клетки — серые
            Canvas.Brush.Color := clLtGray;
            Canvas.Rectangle(x-1,y-1,x+W,y+H);
            // если игра завершена (status = 2),
            // то показать мины
             if (status = 2) and (Pole[row,col] = 9)
                 then Mina(Canvas, x, y);
             exit;
           end;

    // открываем клетку
    Canvas.Brush.Color := clWhite;      // открытые белые
    Canvas.Rectangle(x-1,y-1,x+W,y+H);
    if  ( Pole[row,col] = 100)
         then exit; // клетка открыта, но она пустая

    if (Pole[row,col] >= 101) and (Pole[row,col] <= 108) then
       begin
           Canvas.Font.Size := 14;
           Canvas.Font.Color := clBlue;
           Canvas.TextOut(x+3,y+2,
                          IntToStr(Pole[row,col] — 100));
           exit;
        end;

    if ( Pole[row,col] >= 200) then
            Flag(Canvas, x, y);
```

```
    if (Pole[row,col] = 109) then
       // на этой мине подорвались!
       begin
         Canvas.Brush.Color := clRed;
         Canvas.Rectangle(x-1,y-1,x+W,y+H);
       end;

    if ( (Pole[row,col] mod 10) = 9)  and  (status = 2) then
              Mina(Canvas, x, y);
end;

// показывает поле
Procedure ShowPole(Canvas : TCanvas; status : integer);
    var
      row,col : integer;
    begin
       for row := 1 to MR do
         for col := 1 to MC do
              Kletka(Canvas, row, col, status);
    end;

// рекурсивная функция открывает текущую и все соседние
// клетки, в которых нет мин
Procedure Open( row, col : integer);
    begin
    if Pole[row,col] = 0 then
           begin
              Pole[row,col] := 100;
              Kletka(Form1.Canvas, row,col, 1);
              Open(row,col-1);
              Open(row-1,col);
              Open(row,col+1);
              Open(row+1,col);
              //примыкающие диагонально
              Open(row-1,col-1);
              Open(row-1,col+1);
              Open(row+1,col-1);
              Open(row+1,col+1);
         end
    else
         if (Pole[row,col]<100)and(Pole[row,col]<>-3) then
                  begin
                     Pole[row,col] := Pole[row,col] + 100;
```

```
                    Kletka(Form1.Canvas, row, col, 1);
                end;
    end;

// новая игра — генерирует новое поле
procedure NewGame();

   var
     row,col : integer; // координаты клетки
     n : integer;       // кол-во поставленных мин
     k : integer;       // кол-во мин в соседних клетках
   begin
     // Очистим эл-ты массива, соответствующие клеткам
     // игрового поля.
     for row :=1 to MR do
        for col :=1 to MC do
            Pole[row,col] := 0;

     // расставим мины
     Randomize();  // инициализация ГСЧ
     n := 0; // кол-во мин
     repeat
       row := Random(MR) + 1;
       col := Random(MC) + 1;
       if ( Pole[row,col] <> 9) then
         begin
            Pole[row,col] := 9;
            n := n+1;
         end;
   until ( n = NM);

   // для каждой клетки вычислим
   // кол-во мин в соседних клетках
   for row := 1 to MR do
       for col := 1 to MC do
       if ( Pole[row,col] <> 9) then
             begin
           k :=0;
        if Pole[row-1,col-1] = 9 then k := k + 1;
        if Pole[row-1,col]   = 9 then k := k + 1;
        if Pole[row-1,col+1] = 9 then k := k + 1;
        if Pole[row,col-1]   = 9 then k := k + 1;
        if Pole[row,col+1]   = 9 then k := k + 1;
```

```
        if Pole[row+1,col-1] = 9 then k := k + 1;
        if Pole[row+1,col]   = 9 then k := k + 1;
        if Pole[row+1,col+1] = 9 then k := k + 1;
        Pole[row,col] := k;
         end;
   status := 0;  // начало игры
   nMin   := 0;  // нет обнаруженных мин
   nFlag  := 0;  // нет флагов

end;

// рисует мину
Procedure Mina(Canvas : TCanvas; x, y : integer);
begin
    with Canvas do
        begin
           Brush.Color := clGreen;
           Pen.Color := clBlack;
           Rectangle(x+16,y+26,x+24,y+30);
           Rectangle(x+8,y+30,x+16,y+34);
           Rectangle(x+24,y+30,x+32,y+34);
           Pie(x+6,y+28,x+34,y+44,x+34,y+36,x+6,y+36);

           MoveTo(x+12,y+32); LineTo(x+26,y+32);
           MoveTo(x+8,y+36);  LineTo(x+32,y+36);
           MoveTo(x+20,y+22); LineTo(x+20,y+26);
           MoveTo(x+8, y+30); LineTo(x+6,y+28);
           MoveTo(x+32,y+30); LineTo(x+34,y+28);
         end;
end;

// рисует флаг
Procedure Flag( Canvas : TCanvas; x, y : integer);
    var
       p : array [0..3] of TPoint;  // координаты флажка и
                                    // нижней точки древка
       m : array [0..4] of TPoint;  // буква М
    begin
        // зададим координаты точек флажка
        p[0].x:=x+4;   p[0].y:=y+4;
        p[1].x:=x+30;  p[1].y:=y+12;
        p[2].x:=x+4;   p[2].y:=y+20;
        p[3].x:=x+4;     p[3].y:=y+36;  // нижняя точка древка
```

```
        m[0].x:=x+8; m[0].y:=y+14;
        m[1].x:=x+8; m[1].y:=y+8;
        m[2].x:=x+10; m[2].y:=y+10;
        m[3].x:=x+12; m[3].y:=y+8;
        m[4].x:=x+12; m[4].y:=y+14;

        with Canvas do
          begin
            // установим цвет кисти и карандаша
            Brush.Color := clRed;
            Pen.Color := clRed;

            Polygon(p);  // флажок

            // древко
            Pen.Color := clBlack;
            MoveTo(p[0].x, p[0].y);
            LineTo(p[3].x, p[3].y);

            // буква M
            Pen.Color := clWhite;
            Polyline(m);

            Pen.Color := clBlack;
          end;
end;

// выбор из меню ? команды О программе
procedure TForm1.N4Click(Sender: TObject);
    begin
        AboutForm.Top := Trunc(Form1.Top + Form1.Height/2 -
                               AboutForm.Height/2);
        AboutForm.Left := Trunc(Form1.Left +Form1.Width/2 -
                                AboutForm.Width/2);
        AboutForm.ShowModal;
    end;

procedure TForm1.Form1Create(Sender: TObject);
var
  row,col : integer;
begin
   // В неотображаемые эл-ты массива, которые соответствуют
   // клеткам по границе игрового поля, запишем число -3.
   // Это значение используется функцией Open для завершения
   // рекурсивного процесса открытия соседних пустых клеток.
```

```
   for row :=0 to MR+1 do
        for col :=0 to MC+1 do
           Pole[row,col] := -3;

   NewGame();  // "разбросать" мины
   Form1.ClientHeight := H*MR + 1;
   Form1.ClientWidth := W*MC + 1;
end;

// нажатие кнопки мыши на игровом поле
procedure TForm1.Form1MouseDown(Sender: TObject;
                      Button: TMouseButton;
                      Shift: TShiftState; X, Y: Integer);
  var
      row, col : integer;
  begin
     if status = 2 // игра завершена
         then exit;

     if status = 0 then // первый щелчок
           status := 1;

     // преобразуем координаты мыши в индексы
     // клетки поля
     row := Trunc(y/H) + 1;
     col := Trunc(x/W) + 1;

     if Button = mbLeft then
     begin
        if Pole[row,col] = 9 then
            begin // открыта клетка, в которой есть мина
              Pole[row,col] := Pole[row,col] + 100;
              status := 2;  // игра закончена
              ShowPole(Form1.Canvas, status);
            end
            else if  Pole[row,col] < 9 then
                      Open(row,col);
     end
     else
        if Button = mbRight then
            if Pole[row,col] > 200 then
            begin
               // уберем флаг и закроем клетку
               nFlag := nFlag — 1;
```

```
                Pole[row,col] := Pole[row,col] — 200;
                x := (col-1)* W + 1;
                y := (row-1)* H + 1;
                Canvas.Brush.Color := clLtGray;
                Canvas.Rectangle(x-1,y-1,x+W,y+H);
              end
              else
                 begin // поставить в клетку флаг
                    nFlag := nFlag + 1;
                    if  Pole[row,col] = 9
                       then nMin := nMin + 1;
                    Pole[row,col] := Pole[row,col]+ 200;
                    if (nMin = NM) and (nFlag = NM) then
                    begin
                        status := 2;  // игра закончена
                        ShowPole(Form1.Canvas, status);
                    end
                    else Kletka(Form1.Canvas, row, col,
                                status);
                 end;
  end;

// выбор меню Новая игра
procedure TForm1.N1Click(Sender: TObject);
begin
  NewGame();
  ShowPole(Form1.Canvas,status);
end;

// выбор из меню ? команды Справка
procedure TForm1.N3Click(Sender: TObject);
begin
  // вывести справочную информацию
  Winhelp(Form1.Handle,'saper.hlp',HELP_CONTEXT,1);

end;

// обработка события OnPaint
procedure TForm1.Form1Paint(Sender: TObject);
begin
        // отобразить игровое поле
        ShowPole(Form1.Canvas, status);
end;
end.
```

```
// модуль формы О программе
unit saper_2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls,   saper_1;

type
  TAboutForm = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  AboutForm: TAboutForm;

implementation

{$R *.DFM}

procedure TAboutForm.Button1Click(Sender: TObject);
begin
        ModalResult := mrOk;
end;

end.
```

60. Напишите программу "Будильник". Форма программы приведена на рис. 1.66. После того как пользователь установит время сигнала, задаст текст напоминания и щелкнет на кнопке **OK**, окно программы должно исчезнуть с экрана. В установленное время на экране должно появиться окно с напоминанием. Появление окна должно сопровождаться звуковым сигналом.

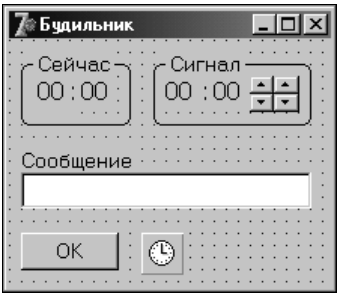


Рис. 1.66. Форма программы **Будильник**

```
{ Будильник. Модуль главной формы. }
unit Alarm2_1_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Dialogs, StdCtrls,
  ExtCtrls, ComCtrls, DateUtils;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    UpDown1: TUpDown;
    Label5: TLabel;
    UpDown2: TUpDown;
    Shape1: TShape;
    Label7: TLabel;
    Button1: TButton;
    Shape2: TShape;
    Label8: TLabel;
    Edit1: TEdit;
    Label6: TLabel;
    Label9: TLabel;

    procedure FormCreate(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
```

```
    procedure UpDown1Click(Sender: TObject;
                           Button: TUDBtnType);
    procedure UpDown2Click(Sender: TObject;
                           Button: TUDBtnType);
    procedure Button1Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses Alarm2_2_;

{$R *.dfm}

var
    Hour,Min: word;       // время на индикаторе
    AlHour, AlMin: word; // будильник установлен на AlHour:AlMin

// начало работы программы
procedure TForm1.FormCreate(Sender: TObject);
begin
    Hour := HourOf(Now);
    Min := MinuteOf(Now);
    Label1.Caption := IntToStr(Hour);
    if Min < 10
       then Label2.Caption := '0'+IntToStr(Min)
       else Label2.Caption := IntToStr(Min);
end;

// сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
var
    cHour,cMin: word;
begin
    // получить текущее время
    cHour := HourOf(Now);
    cMin :=  MinuteOf(Now);
```

```
    if Timer1.Tag = 0 // окно программы на экране
    then begin
          { проверим, совпадает ли текущее время
            с отображаемым на индикаторе }
          if cHour <> Hour then
          begin
             Hour := cHour;
             Label1.Caption := IntToStr(Hour);
          end;

          if cMin <> Min then
          begin
             Min := cMin;
             if min <10
                  then Label2.Caption := '0' + IntToStr(Min)
                  else Label2.Caption := IntToStr(Min);
          end;

          // обеспечим мигание двоеточия
         if Label3.Visible
           then Label3.Visible := False
           else label3.Visible := True;
     end

     else // окно программы скрыто, контролируем
          // наступление момента подачи сигнала
          if (cHour = AlHour) and (cMin = AlMin)
              // сигнал !
             then begin
                 Form2.Show;
                 Timer1.Tag := 0;
                 Timer1.Interval := 1000;
             end;
end;

// щелчок на UpDown1 изменяет
// время сигнала будильника — часы
procedure TForm1.UpDown1Click(Sender: TObject; Button: TUDBtnType);
begin
    if UpDown1.Position < 10
        then Label4.Caption := '0' +
                               IntToStr(UpDown1.Position)
```

```
        else Label4.Caption := IntToStr(UpDown1.Position);
end;

// щелчок на UpDown2 изменяет
// время сигнала будильника — минуты
procedure TForm1.UpDown2Click(Sender: TObject;
                              Button: TUDBtnType);
begin
    if UpDown2.Position < 10
        then Label5.Caption := '0' +
                               IntToStr(UpDown2.Position)
        else Label5.Caption := IntToStr(UpDown2.Position);
end;

// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
begin
    // установить будильник
    AlHour := UpDown1.Position;
    AlMin  := UpDown2.Position;
    Timer1.Tag := 1;
    Form1.Hide; //
    Timer1.Interval := 3000; // проверять каждые 3 секунды
end;

end.

{ Будильник. Модуль окна сообщения. }
unit Alarm2_2_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,  Dialogs, StdCtrls,
  MMSYSTEM,  // для доступа к PlaySound
  Alarm2_1_; // для доступа к стартовой форме программы

type
  TForm2 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;

implementation

{$R *.dfm}

procedure TForm2.Button1Click(Sender: TObject);
begin
    Form2.Hide; // скрыть окно с сообщением
    Form1.Show; // сделать доступным главное окно
end;

procedure TForm2.FormActivate(Sender: TObject);
begin
     Label1.Caption := Form1.Edit1.Text; // текст сообщения
     PlaySound('tada.wav',0,SND_ASYNC); // звуковой сигнал
end;

end.
```

61. Напишите программу "Будильник". Форма программы приведена на рис. 1.67. После того как пользователь установит время сигнала и щелкнет на кнопке **OK**, окно программы должно исчезнуть с экрана, а значок программы — появиться на системной панели (рис. 1.68). Окно программы должно появиться на экране в установленное время. Появление окна должно сопровождаться звуковым сигналом.

```
unit Alarm_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Dialogs, StdCtrls,
   ExtCtrls, ComCtrls,

   ShellAPI,  // для доступа к Shell_NotifyIcon
   DateUtils, MPlayer;
```

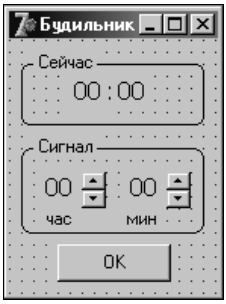


Рис. 1.67. Форма программы **Будильник**

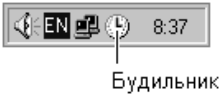


Рис. 1.68. Значок программы **Будильник** во время ее работы должен находиться на системной панели

```
type
  TForm1 = class(TForm)
    Timer1: TTimer;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    UpDown1: TUpDown;
    Label5: TLabel;
    UpDown2: TUpDown;
    Label6: TLabel;
    Shape1: TShape;
    Label7: TLabel;
    Button1: TButton;
    Shape2: TShape;
    Label8: TLabel;

    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure UpDown1Click(Sender: TObject;
                           Button: TUDBtnType);
    procedure UpDown2Click(Sender: TObject;
                           Button: TUDBtnType);
    procedure Button1Click(Sender: TObject);

    // **** эти объявления вставлены сюда вручную
    procedure CreateTrayIcon(n: integer; Tip: String);
```

```
    procedure DeleteTrayIcon(n: integer);
    procedure SetSound;
    // ****************************************

  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{ Имя WAV-файла, который должен находиться
  в системном каталоге Media, программа
  берет из командной строки. Если параметр
  не указан, будильник воспроизводит звук,
  который располагается в файле sound.wav,
  который должен находиться в том же
  каталоге, что и выполняемый файл программы.
  }

{$R *.dfm}

var
    Hour,Min: word;       // время на индикаторе
    AlHour, AlMin: word; // будильник установлен на AlHour:AlMin

    MediaPlayer : TMediaPlayer;
   // компонент MediaPlayer обеспечивает воспроизведение
   // звукового сигнала

// начало работы программы
procedure TForm1.FormCreate(Sender: TObject);
begin
    Hour := HourOf(Now);
    Min := MinuteOf(Now);
    Label1.Caption := IntToStr(Hour);
    if Min < 10
       then Label2.Caption := '0'+IntToStr(Min)
       else Label2.Caption := IntToStr(Min);
```

```
    // CreateTrayIcon(1); // поместить картинку на System Tray
    SetSound;             // загрузить WAV-файл
end;

// завершение работы программы
procedure TForm1.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
    // удалить картинку с System Tray
    DeleteTrayIcon(1);
end;

const
    WM_MYTRAYNOTIFY = WM_USER + 123;

// помещает картинку на System Tray
procedure TForm1.CreateTrayIcon(n: integer; Tip: string);
var
    nidata: TNotifyIconData;
begin
    // заполним структуру nidata,
    // поля которой определяют значок
    // на System Tray
    with nidata do
    begin
        cbSize := SizeOf(TNotifyIconData);
        Wnd := Self.Handle; // окно (приложение), которое
                            // представляет значок
        uId := n; // номер значка (одно приложение может
                  // разместить несколько значков)
        uFlags := NIF_ICON or NIF_MESSAGE or NIF_TIP; // что
                                                // надо сделать
        uCallBackMessage := WM_MYTRAYNOTIFY;
        hIcon := Application.Icon.Handle;
        //szTip := 'Будильник';
        StrPCopy(szTip,Tip);  // копируем Ansi-строку
                              // в nul-terminated-строку
    end;
    Shell_NotifyIcon(NIM_ADD, @nidata);
end;

// удаляет картинку с System Tray
procedure TForm1.DeleteTrayIcon(n: integer);
```

```
var
    nidata: TNotifyIconData;
begin
    // заполним структуру nidata,
    // поля которой определяют значок
    // на System Tray
    with nidata do
    begin
        cbSize := SizeOf(TNotifyIconData);
        Wnd := Self.Handle; // окно (приложение), которое
                            // представляет значок
        uId := n; // номер значка (одно приложение может
                  // разместить несколько значков)
    end;
    Shell_NotifyIcon(NIM_DELETE, @nidata);
end;

// сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
var
    cHour,cMin: word;
begin
    // получить текущее время
    cHour := HourOf(Now);
    cMin :=  MinuteOf(Now);

    if Timer1.Tag = 0 // окно программы на экране
    then begin
          { проверим, совпадает ли текущее время
            с отображаемым на индикаторе }
          if cHour <> Hour then

          begin
             Hour := cHour;
             Label1.Caption := IntToStr(Hour);
          end;

          if cMin <> Min then
          begin
             Min := cMin;
             if min <10
                 then Label2.Caption := '0' + IntToStr(Min)
                 else Label2.Caption := IntToStr(Min);
        end;
```

```
        // обеспечим мигание двоеточия
        if Label3.Visible
           then Label3.Visible := False
           else label3.Visible := True;
     end

     else // окно программы скрыто, контролируем
          // наступление момента подачи сигнала
          if (cHour = AlHour) and (cMin = AlMin)
            // сигнал !
            then begin
              Form1.Show;
              Timer1.Tag := 0;
              Timer1.Interval := 1000;
              try
                 MediaPlayer.Play;  // воспроизвести
                                    // звуковой фрагмент
                 except
                    on EMCIDeviceError do;
                 end;
             end;
end;

// щелчок на UpDown1 изменяет
// время сигнала будильника — часы
procedure TForm1.UpDown1Click(Sender: TObject;
                              Button: TUDBtnType);
begin
    if UpDown1.Position < 10
        then Label4.Caption := '0' +
                               IntToStr(UpDown1.Position)
        else Label4.Caption := IntToStr(UpDown1.Position);
end;

// щелчок на UpDown2 изменяет
// время сигнала будильника — минуты
procedure TForm1.UpDown2Click(Sender: TObject;
                              Button: TUDBtnType);
begin
    if UpDown2.Position < 10
        then Label5.Caption := '0' + IntToStr(UpDown2.Position)
        else Label5.Caption := IntToStr(UpDown2.Position);
end;
```

```
// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
begin
    // установить будильник
    AlHour := UpDown1.Position;
    AlMin  := UpDown2.Position;
    Timer1.Tag := 1;
    // поместить картинку на System Tray
    CreateTrayIcon(1,'Будильник '+
       Label4.Caption+':'+Label5.Caption);

    Form1.Hide;
    Timer1.Interval := 3000; // проверять каждые 3 секунды
end;

// определяет звук будильника
procedure TForm1.SetSound;
var
    pWinDir: PChar;      // указатель на nul-terminated-строку
    sWinDir: String[80];
begin
    // создадим компонент MediaPlayer
    MediaPlayer := TMediaPlayer.Create(Form1);
    MediaPlayer.ParentWindow := Form1.Handle;
    MediaPlayer.Visible := False;

    // Стандартные WAV-файлы находятся в каталоге Media,
    // но где находится и как называется каталог, в который
    // установлен Windows? Выясним это.
    // Чтобы получить имя каталога Windows,
    // воспользуемся API-функцией GetWindowsDirectory.
    // Строка, которая передается в API-функцию,
    // должна быть nul-terminated-строкой.

    // Получить имя каталога Windows
    GetMem(pWinDir,80);    // выделить память для строки
    GetWindowsDirectory(pWinDir,80); // получить каталог
                                     // Windows
    sWinDir := pWindir;

     // открыть WAV-файл
     if ParamStr(1) = ''
        then MediaPlayer.FileName := 'Sound.wav'
```

```
        else MediaPlayer.FileName := sWinDir + '\media\' +
                                     ParamStr(1);

     try
        MediaPlayer.Open;
        except
          on EMCIDeviceError do;
     end;
end;

end.
```

62. Напишите программу тестирования, в которой выбор правильного ответа осуществляется при помощи переключателя (рис. 1.69).

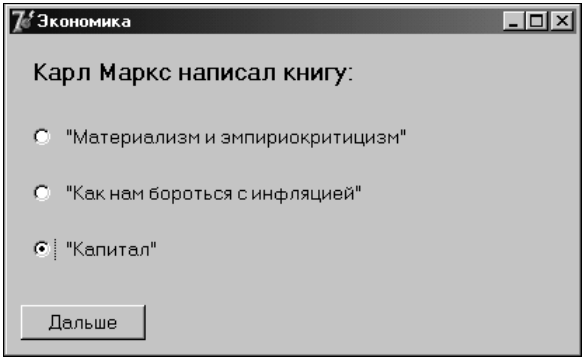


Рис. 1.69. Окно программы

Тест — последовательность вопросов и альтернативных ответов, должен находиться в файле. Количество вопросов теста не ограничено. Программа тестирования должна получать имя файла теста из командной строки.

Ниже приведен пример файла теста. Первая строка — это заголовок теста. Потом следует описание 4-х уровней оценок. Для каждого уровня задается сообщение и количество правильных ответов, необходимых для достижения уровня. Далее следуют вопросы и варианты ответов. После каждого альтернативного ответа стоит 1 или 0. Единица показывает, что данный вариант

ответа — правильный. Следует обратить внимание, что каждое сообщение, каждый вопрос и ответ в файле теста представлены одной строкой.

```
Экономика
Вы прекрасно справились с вопросами. Оценка — ОТЛИЧНО!
6
На один или несколько вопросов Вы ответили неправильно. Оценка — ХОРОШО.
5
На некоторые вопросы Вы ответили неправильно. Оценка —
                                                    УДОВЛЕТВОРИТЕЛЬНО.
4
Вы плохо подготовились к испытанию. Оценка — ПЛОХО!
3
Карл Маркс написал книгу:
"Материализм и эмпириокритицизм"
0
"Как нам бороться с инфляцией"
0
"Капитал"
1
Что означает выражение "Делать бизнес"?
обманывать и хитрить
0
учиться в школе бизнесменов
0
заниматься конкретным делом, приносящим доход
1
Когда впервые появились бартерные сделки?
при первобытнообщинном строе
1
в период общественного разделения труда
0
в наше время
0
Слово "бухгалтер" переводится с немецкого как:
человек, держащий книгу
1
человек, считающий на счетах
0
человек, работающий с большой кипой бумаг
0
Как переводится с английского "ноу-хау" и что оно обозначает?
секрет
0
новое предприятие
0
новая идея (знаю, как)
1
```

```
Конкуренция в переводе с латинского:
столкновение
1
соревнование
0
конкурс
0
```

```
implementation

{$R *.dfm}

var
  f: TextFile;       // файл теста (вопросы и варианты ответов)
  nq: integer;       // количество вопросов в тесте
  right: integer;    // количество правильных ответов
  level: array[1..4] of integer;  // критерии оценок
  mes: array[1..4] of string;     // комментарии
  buf: string;

// читает вопрос из файла и выводит его
// в поля формы
function NextQw : boolean;
begin
  if not EOF(f) then
  begin
    // прочитать и вывести вопрос
    Readln(f,buf);
    Form1.Label1.Caption := buf;

    // прочитать и вывести варианты ответов
    // 1-й вариант
    Readln(f,buf);  // прочитать 1-й вариант ответа
    Form1.Label2.Caption := buf;
    Readln(f,buf); // оценка за выбор этого ответа:
                   // 1 — правильно, 0 — нет
    Form1.RadioButton1.Tag := StrToInt(buf);

    // 2-й вариант
    Readln(f,buf);
    Form1.Label3.Caption := buf;
    Readln(f,buf);
    Form1.RadioButton2.Tag := StrToInt(buf);
```

```
    // 3-й вариант
    Readln(f,buf);
    Form1.Label4.Caption := buf;
    Readln(f,buf);
    Form1.RadioButton3.Tag := StrToInt(buf);

    // счетчик общего количества вопросов
    nq:= nq + 1;

    // кнопка Дальше недоступна,
    // пока не выбран один из вариантов ответа
    Form1.Button1.Enabled := False;

    // ни один из переключателей не выбран
    Form1.RadioButton1.Checked := False;
    Form1.RadioButton2.Checked := False;
    Form1.RadioButton3.Checked := False;
    NextQw := TRUE;
  end
  else NextQw := FALSE;
end;

// событие FormCreate возникает в момент
// создания формы
procedure TForm1.FormCreate(Sender: TObject);
var
  i: integer;
  fname : string;
begin
  { Если программа запускается из Delphi,
    то имя файла теста надо ввести в
    поле Parameters диалогового окна
    Run Parameters, которое становится
    доступным в результате выбора в меню
    Run команды Parameters.}
  fname := ParamStr(1); // взять имя файла теста
                        // из командной строки
  if fname = '' then
  begin
    ShowMessage('В командной строке запуска программы' +#13+
                'надо указать имя файла теста.');
    Application.Terminate; // завершить программу
  end;
  AssignFile(f,fname);
```

```
  // в процессе открытия файла возможны
  // ошибки, поэтому ...
  try
    Reset(f); // эта инструкция может вызвать ошибку
  except
    on EInOutError do
        begin
            ShowMessage('Ошибка обращения к файлу теста: ' +
                        fname);
            Application.Terminate; // завершить программу
        end;
  end;

  // здесь файл теста успешно открыт
  // прочитать название теста — первая строка файла
  Readln(f,buf);
  Form1.Caption := buf;

  // прочитать оценки и комментарии
  for i:=1 to 4 do
    begin
      Readln(f,buf);
      mes[i] := buf;
      Readln(f,buf);
      level[i] := StrToInt(buf);
    end;

   right := 0; // правильных ответов
   nq := 0;    // всего вопросов
   NextQW;     // прочитать и вывести первый вопрос
end;

// щелчок на кнопке Дальше
procedure TForm1.Button1Click(Sender: TObject);
var
    buf: string;
    i: integer;
begin
    if Button1.Caption = 'Завершить' then Close;

    // добавим оценку за выбранный вариант ответа
    // оценка находится в свойстве Button.Tag
    // Button.Tag = 1 — ответ правильный, 0 — нет
```

```
    if RadioButton1.Checked  then
        right := right + RadioButton1.Tag;
    if RadioButton2.Checked  then
       right := right + RadioButton2.Tag;
    if RadioButton3.Checked  then
       right := right + RadioButton3.Tag;

  // вывести следующий вопрос
  // NextQW читает и выводит вопрос
  // NextQW = FALSE, если в файле теста
  // вопросов больше нет
  if not NextQW then
  begin
      // здесь значение NextQW = FALSE
      Button1.Caption := 'Завершить';

      // скрыть переключатели и поля меток
      RadioButton1.Visible := False;
      RadioButton2.Visible := False;
      RadioButton3.Visible := False;
      Label2.Visible := False;
      Label3.Visible := False;
      Label4.Visible := False;

      buf := 'Тестирование завершено.' + #13 +
             'Правильных ответов: ' + IntToStr(right) +
             ' из ' + IntToStr(nq) + '.' + #13;

      // выставить оценку
      // right — кол-во правильных ответов
      i:=1; // номер уровня
      while (right < level[i]) and (i < 4) do
            inc(i);
      buf := buf + mes[i];

      Label1.AutoSize := TRUE;
      Label1.Caption := buf;

    end;
end;

// щелчок на переключателе выбора первого варианта ответа
procedure TForm1.RadioButton1Click(Sender: TObject);
```

```
begin
  Button1.Enabled := True; // кнопка Дальше теперь доступна
end;

procedure TForm1.RadioButton2Click(Sender: TObject);
begin
  Button1.Enabled := True;
end;

procedure TForm1.RadioButton3Click(Sender: TObject);
begin
  Button1.Enabled := True;
end;

end.
```

63. Напишите программу тестирования, в которой вопрос может сопровождаться иллюстрацией, а количество вариантов ответа к каждому вопросу может быть от двух до 4. Пример окна программы приведен на рис. 1.70.

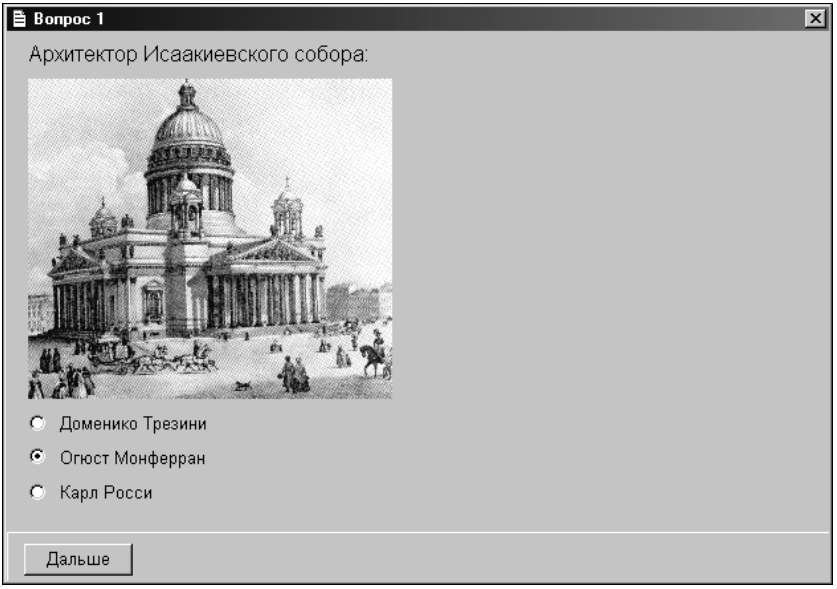


Рис. 1.70. Окно программы

Вопросы теста должны находиться в текстовом файле, имя которого должно передаваться программе тестирования как параметр командной строки.

Предлагается следующая структура файла теста.

Первая строка файла теста — название теста, за которым следуют разделы:

- ❒ заголовка;
- ❒ оценок;
- ❒ вопросов.

Название выводится в заголовке стартового окна программы тестирования.

Раздел заголовка содержит общую информацию о тесте, например, о его назначении. Заголовок может состоять из нескольких строк. Признаком конца раздела заголовка является точка, стоящая в начале строки.

Вот пример заголовка файла теста:

```
Сейчас Вам будут предложены вопросы о знаменитых памятниках
и архитектурных сооружениях Санкт-Петербурга. Вы должны
из предложенных нескольких вариантов ответа выбрать правильный.
.
```

За заголовком следует раздел оценок, в котором приводятся названия оценочных уровней и количество баллов, необходимое для достижения этих уровней. Название уровня должно располагаться в одной строке. Вот пример раздела оценок:

```
Отлично
100
Хорошо
85
Удовлетворительно
60
Плохо
50
```

За разделом оценок следует раздел вопросов теста.

Каждый вопрос начинается текстом вопроса, за которым может следовать имя файла иллюстрации, размещаемое на отдельной

строке и начинающееся символом \. Имя файла иллюстрации является признаком конца текста вопроса. Если к вопросу нет иллюстрации, то вместо имени файла ставится точка.

После вопроса следуют альтернативные ответы. Текст альтернативного ответа может занимать несколько строк. В строке, следующей за текстом ответа, располагается оценка (количество баллов) за выбор этого ответа. Если альтернативный ответ не является последним для текущего ответа, то перед оценкой ставится запятая, если последний — то точка.

Вот пример вопроса:

```
Архитектор Исаакиевского собора:
\isaak.bmp
Доменико Трезини
,0
Огюст Монферран
,1
Карл Росси
.0
```

В приведенном вопросе первый и третий варианты ответа помечены как не правильные (оценка за их выбор равна нулю).

Вид формы программы тестирования приведен на рис. 1.71.

```
{ Универсальная программа тестирования.
  (c) Культин Н.Б., 2003 }
unit tester_;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls,
  jpeg;

type
  TForm1 = class(TForm)
    // вопрос
    Label5: TLabel;

    // альтернативные ответы
    Label1: TLabel;
```

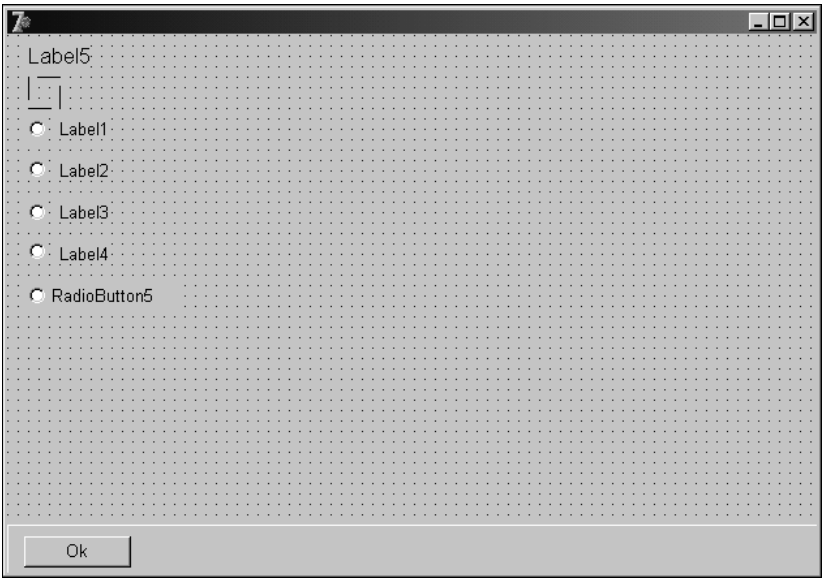


Рис. 1.71. Форма программы тестирования

```
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;

// переключатели выбора ответа
RadioButton1: TRadioButton;
RadioButton2: TRadioButton;
RadioButton3: TRadioButton;
RadioButton4: TRadioButton;

Image1: TImage;   // область вывода иллюстрации
Button1: TButton;
Panel1: TPanel;
RadioButton5: TRadioButton;

procedure FormActivate(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure RadioButtonClick(Sender: TObject);
procedure FormCreate(Sender: TObject);

// Эти объявления вставлены сюда вручную
procedure Info;
```

```
    procedure VoprosToScr;
    procedure ShowPicture; // выводит иллюстрацию
    procedure ResetForm;   // "очистка" формы перед выводом
                           // очередного вопроса
    procedure Itog; // результат тестирования
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;     // форма

implementation
{$R *.DFM}
const
   N_LEV=4; // четыре уровня оценки
   N_ANS=4; // четыре варианта ответов

var
  f:TextFile;
  fn:string; // имя файла вопросов

  // сумма, соответствующая уровню
  level:array[1..N_LEV] of integer;

// сообщение, соответствующее уровню
  mes:array[1..N_LEV] of string;

  score:array[1..N_ANS] of integer; // оценка за выбор ответа
  summa:integer;  // набрано очков
  vopros:integer; // номер текущего вопроса
  otv:integer;    // номер выбранного ответа

// вывод информации о тесте
procedure Tform1.Info;
var
   s,buf:string;
begin
   readln(f,s);
   Form1.Caption := s;
   buf:='';
```

```
   repeat
         readln(f,s);
         if s[1] <> '.'
            then buf := buf +s + #13;
   until s[1] ='.';
   Label5.caption:=buf;
end;

// прочитать информацию об оценках за тест
Procedure GetLevel;
var
   i:integer;
   buf:string;
begin
     i:=1;
     repeat
         readln(f,buf);
         if buf[1] <> '.' then begin
            mes[i]:=buf;        // сообщение
            readln(f,level[i]); // оценка
            i:=i+1;
         end;
     until buf[1]='.';
end;

// масштабирование иллюстрации
Procedure TForm1.ShowPicture;
var
  w,h: integer; // максимально возможные размеры картинки
begin
  // вычислить допустимые размеры картинки
  w:=ClientWidth-10;
  h:=ClientHeight
         - Panel1.Height -10
         - Label5.Top
         - Label5.Height - 10;

  // вопросы
  if Label1.Caption <> ''
     then h:=h-Label1.Height-10;
  if Label2.Caption <> ''
     then h:=h-Label2.Height-10;
```

```
  if Label3.Caption <> ''
     then h:=h-Label3.Height-10;
  if Label4.Caption <> ''
     then h:=h-Label4.Height-10;

  // если размер картинки меньше w на h,
  // то она не масштабируется
  Image1.Top:=Form1.Label5.Top+Label5.Height+10;
  if Image1.Picture.Bitmap.Height > h
        then Image1.Height:=h
        else Image1.Height:= Image1.Picture.Height;
  if Image1.Picture.Bitmap.Width > w
        then Image1.Width:=w
        else Image1.Width:=Image1.Picture.Width;

  Image1.Visible := True;
end;

// вывести вопрос
Procedure TForm1.VoprosToScr;
  var
   i:integer;
   s,buf:string;
   ifn:string; // файл иллюстрации
  begin
     vopros:=vopros+1;
     caption:='Вопрос ' + IntToStr(vopros);
     // прочитать вопрос
     buf:='';
     repeat
           readln(f,s);
           if (s[1] <> '.') and (s[1] <> '\')
              then buf:=buf+s+' ';
     until (s[1] ='.') or (s[1] ='\');
     Label5.caption:=buf;  // вывести вопрос

     { Иллюстрацию прочитаем, но выведем только после
       того, как прочитаем альтернативные ответы
       и определим максимально возможный размер
       области формы, который можно использовать
       для ее вывода.}
     if s[1] <> '\'
       then Image1.Tag:=0 // к вопросу нет иллюстрации
```

```
  else // к вопросу есть иллюстрация
   begin
     Image1.Tag:=1;
     ifn:=copy(s,2,length(s));
     try
       Image1.Picture.LoadFromFile(ifn);
       except
         on E:EFOpenError do
            Image1.Tag:=0;
     end;
   end;

// читаем варианты ответов
i:=1;
repeat
   buf:='';
   repeat // читаем текст варианта ответа
     readln(f,s);
     if (s[1]<>'.') and (s[1] <> ',')
           then buf:=buf+s+' ';
   until (s[1]=',')or(s[1]='.');
   // прочитан альтернативный ответ
   score[i]:= StrToInt(s[2]);
   case i of
       1: Label1.caption:=buf;
       2: Label2.caption:=buf;
       3: Label3.caption:=buf;
       4: Label4.caption:=buf;
   end;
   i:=i+1;
until s[1]='.';
// здесь прочитана иллюстрация и альтернативные ответы

// текст вопроса уже выведен
if Image1.Tag =1  // есть иллюстрация к вопросу
   then ShowPicture;

// вывод альтернативных ответов
if Form1.Label1.Caption <> ''
then begin
   if Form1.Image1.Tag =1
       then Label1.Top:=Image1.Top+Image1.Height+10
       else Label1.Top:=Label5.Top+Label5.Height+10;
```

```
        RadioButton1.Top:=Label1.Top;
        Label1.visible:=TRUE;
        RadioButton1.visible:=TRUE;
      end;

      if Form1.Label2.Caption <> ''
      then begin
        Label2.top:=Label1.top+ Label1.height+10;
        RadioButton2.top:=Label2.top;
        Label2.visible:=TRUE;
        RadioButton2.visible:=TRUE;
      end;

      if Form1.Label3.Caption <> ''
      then begin
        Label3.top:=Label2.top+ Label2.height+10;
        RadioButton3.top:=Label3.top;
        Label3.visible:=TRUE;
        RadioButton3.visible:=TRUE;
      end;

      if Form1.Label4.Caption <> ''
      then begin
        Label4.top:=Label3.top+ Label3.height+10;
        RadioButton4.top:=Label4.top;
        Label4.visible:=TRUE;
        RadioButton4.visible:=TRUE;
      end;
  end;

Procedure TForm1.ResetForm;
begin // сделать невидимыми все метки и переключатели

      Label1.Visible:=FALSE;
      Label1.Caption:='';
      Label1.Width:=ClientWidth-Label1.left-5;
      RadioButton1.Visible:=FALSE;

      Label2.Visible:=FALSE;
      Label2.Caption:='';
      Label2.Width:=ClientWidth-Label2.left-5;
      RadioButton2.Visible:=FALSE;

      Label3.Visible:=FALSE;
      Label3.Caption:='';
```

```
     Label3.Width:=ClientWidth-Label3.left-5;
     RadioButton3.Visible:=FALSE;

     Label4.Visible:=FALSE;
     Label4.Caption:='';
     Label4.Width:=ClientWidth-Label4.left-5;
     RadioButton4.Visible:=FALSE;

     Label5.Width:=ClientWidth-Label5.left-5;

     Image1.Visible:=FALSE;
end;

// определение достигнутого уровня
procedure TForm1.Itog;
  var
   i:integer;
   buf:string;
  begin
   buf:='';
   buf:='Результаты тестирования'+ #13
        +'Всего баллов: '+ IntToStr(summa);
   i:=1;
   while (summa < level[i]) and (i<N_LEV) do
         i:=i+1;
   buf:=buf+ #13+mes[i];
   Label5.Top:=20;
   Label5.Caption:=buf;
  end;

procedure TForm1.FormActivate(Sender: TObject);
  begin
     ResetForm;
     if ParamCount = 0
     then begin
             Label5.Caption:= 'Не задан файл вопросов теста.';
             Button1.Caption:='Ok';
             Button1.Tag:=2;
             Button1.Enabled:=TRUE
          end
     else begin
          fn := ParamStr(1);
          assignfile(f,fn);
```

```
          try
            reset(f);
          except
            on EFOpenError do
               begin
                    ShowMessage('Файл теста '+fn+' не найден.');
                    Button1.Caption:='Ok';
                    Button1.Tag:=2;
                    Button1.Enabled:=TRUE;
                    exit;
               end;
          end;
          Info; // прочитать и вывести информацию о тесте
          GetLevel;    // прочитать информацию об уровнях оценок
     end;
end;

// щелчок на кнопке Button1
procedure TForm1.Button1Click(Sender: TObject);
  begin
     case Button1.tag of
       0: begin
               Button1.caption:='Дальше';
               Button1.tag:=1;
               RadioButton5.Checked:=TRUE;
               // вывод первого вопроса
               Button1.Enabled:=False;
               ResetForm;
               VoprosToScr;
          end;
       1: begin // вывод остальных вопросов
               summa:=summa+score[otv];
               RadioButton5.Checked:=TRUE;
               Button1.Enabled:=False;
               ResetForm;
               if not eof(f)
                  then VoprosToScr
                  else
                     begin
                        summa:=summa+score[otv];
                        closefile(f);
```

```
                        Button1.caption:='Ok';
                        Form1.caption:='Результат';
                        Button1.tag:=2;
                        Button1.Enabled:=TRUE;
                        Itog; // вывести результат
                      end;
          end;
       2: begin // завершение работы
            Form1.Close;
          end;
     end;
end;

// Процедура обработки события OnClick
// для компонентов RadioButton1-RadioButton4
procedure TForm1.RadioButtonClick(Sender: TObject);
begin
  if sender = RadioButton1
  then otv:=1
  else if sender = RadioButton2
          then otv:=2
          else if sender = RadioButton3
                  then otv:=3
                  else otv:=4;
  Button1.enabled:=TRUE;
end;

// обеспечивает настройку компонентов
procedure TForm1.FormCreate(Sender: TObject);
begin
    Image1.AutoSize := False;
    Image1.Proportional := True;
    RadioButton1.Visible := False;
end;

end.
```

Базы данных

Общие замечания

Приступая к решению задач этого раздела, необходимо вспомнить:

- Для того чтобы программа могла работать с базой данных, на компьютере должен быть установлен процессор баз данных — Borland Database Engine (BDE). На компьютер программиста BDE устанавливается в процессе установки Delphi.
- Создать базу данных (таблицу данных) и наполнить ее информацией можно при помощи утилиты Database Desktop, которая входит в состав Delphi.
- Перед тем как приступить к созданию таблицы данных, надо создать псевдоним (`Alias`) базы данных. Сделать это можно при помощи утилиты BDE Administrator или SQL Explorer. Обе эти утилиты входят в состав Delphi.
- Для того чтобы перенести программу работы с базой данных на другой компьютер, надо создать установочный CD. Для решения этой задачи Borland рекомендует использовать утилиту InstallShield Express, которая поставляется вместе с Delphi.

64. Напишите программу работы с локальной базой данных "Архитектурные памятники Санкт-Петербурга" (рис. 1.72).

Для создания базы данных используйте утилиту Database Desktop, для создания псевдонима — SQL Explorer или BDE Administrator. Характеристики полей записей базы данных приведены в табл. 1.1, форма программы — на рис. 1.73.

Таблица 1.1. *Поля записей базы данных "Архитектурные памятники Санкт-Петербурга"*

Поле	Тип	Размер	Информация
Monument	A	60	Название памятника
Architect	A	40	Архитектор

Таблица 1.1 *(окончание)*

Поле	Тип	Размер	Информация
Note	A	255	Краткая историческая справка
Photo	A	12	Файл иллюстрации

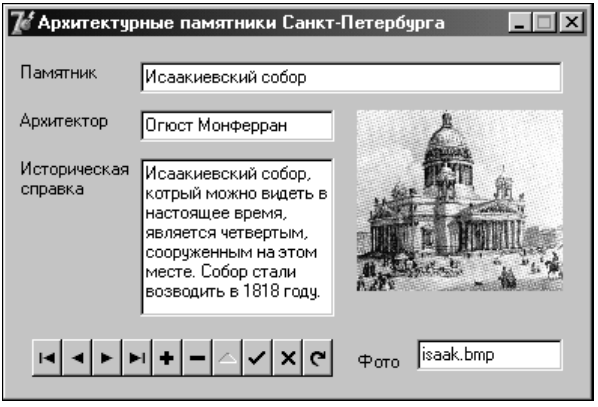


Рис. 1.72. Окно программы работы с базой данных "Архитектурные памятники Санкт-Петербурга"

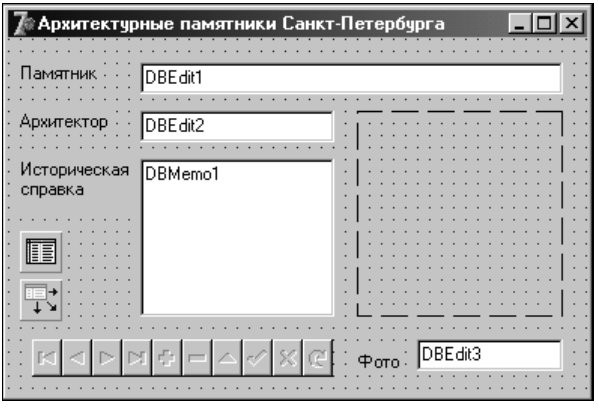


Рис. 1.73. Форма программы работы с базой данных "Архитектурные памятники Санкт-Петербурга"

```
{ Программа работы с базой данных "Архитектурные
  памятники Санкт-Петербурга".
  (c) Культин Н.Б., 2003
  Сначала надо создать базу данных типа STANDARD
  (таблицу monuments.db), затем — псевдоним
  Peterburg. Псевдоним можно создать при помощи
  SQL Explorer. }

unit peter_;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls, DBCtrls,
  Mask, Db, DBTables,
  jpeg; // чтобы можно было выводить JPG-иллюстрации

type
  TForm1 = class(TForm)
    Table1: TTable;             // база данных — таблица
    DataSource1: TDataSource;   // источник данных для полей
                                // редактирования-просмотра
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBMemo1: TDBMemo;
    DBEdit3: TDBEdit;
    Image1: TImage;
    DBNavigator1: TDBNavigator;
    Label4: TLabel;
    procedure Table1AfterScroll(DataSet: TDataSet);
    procedure DBEdit3KeyPress(Sender: TObject; var Key: Char);
    procedure DBNavigator1Click(Sender: TObject;
                                Button: TNavigateBtn);
    procedure Table1BeforeOpen(DataSet: TDataSet);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;
  BmpPath: string;  // Путь к файлам иллюстраций. Иллюстрации
                    // находятся в подкаталоге Data каталога
                   // программы.

implementation
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
    Table1.Active := True; // открыть (таблицу) базу данных
end;

// выводит фотографию в поле Image1
procedure ShowFoto(foto: string);
  begin
    try
       Form1.Image1.Picture.LoadFromFile(BmpPath+foto);
       Form1.Image1.Visible:=True;
     except
       on EFOpenError do
       begin
         MessageDlg('Файл иллюстрации '+foto+' не найден.',
                    mtInformation, [mbOk], 0);
       end;
    end;
end;

// переход к другой записи (следующей, предыдущей,
// первой или последней)
procedure TForm1.Table1AfterScroll(DataSet: TDataSet);
begin
     if form1.DBEdit3.Visible
        then
          begin
             form1.DBEdit3.Visible := False;
             form1.Label4.Visible:=False;
          end;
     if Form1.DBEdit3.Text <> ''
        then ShowFoto(Form1.DBEdit3.Text)
        else form1.Image1.Visible:=False;
end;
```

```
// нажатие клавиши в поле Фото
procedure TForm1.DBEdit3KeyPress(Sender: TObject; var Key: Char);
begin
     if (key = #13) then
         if Form1.DBEdit3.Text <> ''
             then ShowFoto(Form1.DBEdit3.Text) // показать
                                               // иллюстрацию
             else form1.Image1.Visible:=False;
end;

// щелчок на компоненте Навигатор
procedure TForm1.DBNavigator1Click(Sender: TObject;
Button: TNavigateBtn);
begin
     case Button of
     nbInsert: begin  // добавить запись
                Image1.Visible:=False;  // скрыть область
                                        // вывода иллюстрации
                DBEdit3.Visible:=True;  // показать поле Фото
                Label4.Visible:=True;   // показать метку Фото
               end;
     nbEdit:   begin  // редактирование записи
                 DBEdit3.Visible:=True; // показать поле Фото
                 Label4.Visible:=True;  // показать метку Фото
               end;
     end;
end;

// перед тем, как открыть базу данных (таблицу)
procedure TForm1.Table1BeforeOpen(DataSet: TDataSet);
begin
   // определить каталог, в котором находятся
   // иллюстрации
   BmpPath:=ExtractFilePath(ParamStr(0))+'data\';
end;

end.
```

65. Напишите программу, при помощи которой можно создать базу данных "Архитектурные памятники Санкт-Петербурга". Вид формы программы приведен на рис. 1.74.

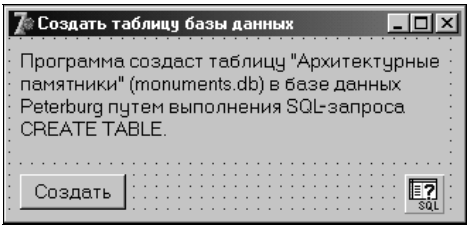


Рис. 1.74. Форма программы **Создать таблицу базы данных**

```
// щелчок на кнопке Создать
procedure TForm1.Button1Click(Sender: TObject);
begin
    Query1.DatabaseName := 'Peterburg';
    { Peterburg — псевдоним базы данных,
      в которой будет создана таблица. }
    with Query1 do begin
        SQL.Clear;
        SQL.Add('CREATE TABLE pam (');
        SQL.Add('Monument  CHAR(60),');
        SQL.Add('Architect CHAR(40),');
        SQL.Add('Note      CHAR(255),');
        SQL.Add('Photo     CHAR(12));');

        // в процессе выполнения запроса возможны ошибки,
        // например нельзя создать таблицу, которая
        // уже есть в базе данных
        try
            ExecSQL; // выполнить запрос
        except
            on E: EDBEngineError do
            begin
                ShowMessage('Ошибка создания таблицы БД.'+
                            #13 + E.Message);
                Button1.Enabled := False;
            end;
    end;
end;
end;
```

66. Напишите программу работы с базой данных "Записная книжка". Для создания базы данных используйте утилиту Database Desktop, для создания псевдонима — SQL Explorer или

BDE Administrator. Характеристики полей записей базы данных приведены в табл. 1.2, форма программы — на рис. 1.75.

Таблица 1.2. *Поля записей базы данных "Записная книжка"*

Поле	Тип	Размер	Информация
Fam	A	15	Фамилия
Name	A	20	Имя
Tel	A	15	Телефон
Email	A	20	Адрес электронной почты

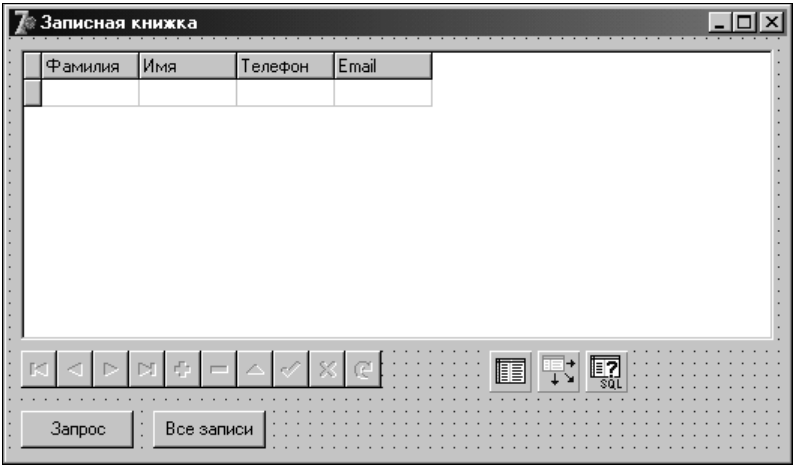


Рис. 1.75. Форма программы **Записная книжка**

```
{ Программа работы с базой данных
  "Записная книжка".}
unit adrbook_;
{ Сначала надо создать таблицу adrbook.db (в формате Paradox)
  и псевдоним adrbook. После этого нужно добавить
  в форму компоненты Table, DataSource, DBGrid и выполнить
  их настройку в соответствии с приведенной ниже таблицей.
```

```
  Свойство              Значение
  -------------------------------
  Table1.DataBaseName   adrbook
  Table1.TableName      adrbook.db
  Table1.Active         True
  DataSource1.DataSet   Table1
  DBGrid.DataSource     DataSource1
  -------------------------------
}

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,  Dialogs, ExtCtrls, DBCtrls, Grids, DBGrids,
  DB, DBTables, StdCtrls;

type
  TForm1 = class(TForm)
    Table1: TTable;              // база данных
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;          // компонент отображения
                               // базы данных
    DBNavigator1: TDBNavigator;
    Button1: TButton;
    Button2: TButton;
    Query1: TQuery;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
        Table1.Active := True; // открыть базу данных
end;

// щелчок на кнопке Запрос
procedure TForm1.Button1Click(Sender: TObject);
var
    buf: string[30]; // критерий запроса
begin
    buf := InputBox('Выборка информации из БД',
                    'Введите фамилию и щелкните на кнопке OK','');
    if buf = '' then exit;

    // пользователь ввел критерий запроса
    with Query1 do begin
        Close;      // закрыть результат выполнения
                    // предыдущего запроса
        SQL.Clear; // удалить текст предыдущего запроса

        // формируем новый запрос
        SQL.Add('SELECT Fam, Name, Tel, Email');
        SQL.Add('FROM ":adrbook:adrbook.db" ');
        SQL.Add('WHERE');
        SQL.Add('(Fam ="' + buf + '")');
        SQL.Add('ORDER BY Fam, Name');

        Open; // выполнить запрос

        if RecordCount <> 0
        then  // отобразить результат выполнения запроса
             DataSource1.DataSet := Query1
        else
             ShowMessage('В БД нет записей, удовлетворяющих' +
                         #13 + 'критерию запроса.');
    end;
end;

// щелчок на кнопке Все записи
procedure TForm1.Button2Click(Sender: TObject);
begin
    DataSource1.DataSet := Table1; // источник данных — таблица
end;

end.
```

67. Напишите программу работы с базой данных "Ежедневник", каждая запись которой содержит информацию о запланированном мероприятии. В начале работы программа должна вывести список дел, запланированных на дату запуска программы и ближайшие дни. Псевдоним базы данных должна создавать программа. Таблица данных должна находиться в том же каталоге, что и выполняемый файл программы. Если таблицы данных нет, то программа должна ее создать. Рекомендуемый вид формы программы приведен на рис. 1.76, структура записей БД — в табл. 1.3.

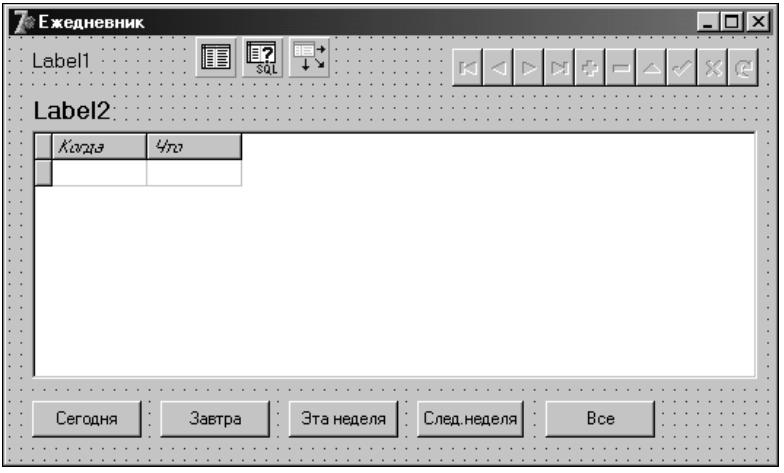


Рис. 1.76. Форма программы **Ежедневник**

Таблица 1.3. *Поля записей базы данных "Ежедневник"*

Поле	Тип	Размер	Информация
What	A	80	Мероприятие
Data	D		Дата

```
// Главный модуль программы "Организатор".
// Чтобы его увидеть, выберите в меню
// Project команду View Source
program org;
```

```
uses
  Forms,
  org_ in 'org_.pas' {Form1},
  SysUtils,Dialogs,
  DateUtils,  // для доступа к IncDay
  DBTables;   // для доступа к Session, AddStandardAlias

{$R *.res}

var
  Present: TDateTime;       // сегодня
  NextDay: TDateTime;       // следующий день
  Year, Month, Day : Word;  // год, месяц, день

  Query: TQuery; // запрос, обеспечивающий
                 // создание БД (таблицы)
begin
  Application.Initialize;
  Application.Title := 'Ежедневник';
  Application.CreateForm(TForm1, Form1);

 { Псевдоним для доступа к базе данных создается
   во время запуска программы и существует только
   во время работы программы.
   База данных находится в том же каталоге, что
   и выполняемый файл программы. Имя каталога,
   в котором находится выполняемый файл, можно получить
   обратившись к функции ParamStr. }

 // создадим псевдоним
 with Session do
 begin
    ConfigMode := cmSession;
    AddStandardAlias('diary',   // псевдоним БД
                     ExtractFilePath(ParamStr(0)), // каталог
                     'PARADOX');
 end;

 // определим текущую дату
  Present:= Now; // Now — функция, возвращает текущую дату
                 // и время
  DecodeDate(Present, Year, Month, Day);
```

```
  case {dof} DayOfWeek(Present) of
    6:   NextDay := IncDay(Present,3);   // пятница
    7:   NextDay := IncDay(Present,2);   // суббота
    else NextDay := IncDay(Present,1)
  end;

  // запрос к базе данных: есть ли дела, запланированные
  // на сегодня и ближайшие дни
  Form1.Query1.SQL[3] :=
         '(Data >= '''+
FormatDateTime('dd/mm/yyyy',Present)+''')' +  'and'+
         '(Data <= '''+ FormatDateTime('dd/mm/yyyy',NextDay)+''')';

  try
    Form1.Query1.Open;  // выполнить запрос

  except
    on E:EDBEngineError do
        // Ошибка при выполнении запроса может
        // быть вызвана тем, что файла базы данных нет.
        // Предложить пользователю создать
        // файл базы данных.
       begin
           MessageDlg('Файл таблицы базы данных не найден .'+#13+
           'Таблица будет создана.',mtWarning,[mbYes],0);

           Query := TQuery.Create(Form1);
           with Query do
           begin
               // сформируем запрос, обеспечивающий
               // создание БД
               SQL.Add('CREATE TABLE diary (');
               SQL.Add('What CHAR(80),');
               SQL.Add('Data DATE);');

               ExecSQL;
           end;
           // таблица создана
           Form1.Query1.Open; // выполним запрос
       end;
  end;

  if Form1.Query1.RecordCount <> 0  // есть дела,
                              // запланированные на ближайшие дни
```

```
    then
        Form1.DataSource1.DataSet := Form1.Query1
    else
        begin
            Form1.DataSource1.DataSet := Form1.Table1;
            Form1.Table1.Open;
            ShowMessage('На сегодня и ближайшие дни' +
                        'никаких дел не запланировано.');
        end;

  Application.Run;
end.

unit org_; // модуль формы
{
 Сразу после запуска программа посылает запрос к базе
 данных, для того чтобы выяснить, есть ли дела, запланированные
на сегодня. Если что-то намечено, то на экране появляется  окно
Ежедневник.
 Запрос к базе данных в начале работы программы формирует
главная процедура приложения. Чтобы ее увидеть,
 выберите из меню Project команду View Source.
}

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,  Dialogs, DB, Grids, DBGrids, DBTables,
  StdCtrls, DBCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Table1: TTable;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Query1: TQuery;
    Label1: TLabel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Label2: TLabel;
    DBNavigator1: TDBNavigator;
```

```
    procedure FormActivate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

  stDay : array[1..7] of string[11] =
                    ('воскресенье','понедельник','вторник',
                     'среда','четверг','пятница','суббота');

  stMonth : array[1..12] of string[8] =
                        ('января','февраля','марта',
                         'апреля','мая','июня','июля',
                         'августа','сентября','октября',
                         'ноября','декабря');

implementation

{$R *.dfm}

uses DateUtils;

// сегодняшняя дата и день недели
procedure TForm1.FormActivate(Sender: TObject);
var
  Present: TDateTime;
  Year, Month, Day : Word;
 begin
  Present:= Now; // Now — функция, возвращает текущую дату
                 // и время
  DecodeDate(Present, Year, Month, Day);
  Label1.Caption := 'Сегодня ' + IntToStr(Day)+' '
    + StMonth[Month] + ' '+ IntToStr(Year)+' года, '
    + stDay[DayOfWeek(Present)];
  Form1.Label2.Caption := 'Сегодня и ближайшие дни';

end;
```

```
// щелчок на кнопке Сегодня
procedure TForm1.Button2Click(Sender: TObject);
var
    st : string; // критерий запроса
begin
    Form1.Label2.Caption := 'Сегодня';
    st:= FormatDateTime('dd/mm/yyyy',Now);
    Form1.Query1.SQL[3] := '(Data = '''+st+''')';
    Form1.Query1.Open;
    if form1.Query1.RecordCount <> 0 then
        form1.DataSource1.DataSet := Form1.Query1
    else begin
        ShowMessage('На сегодня никаких дел не запланировано.');
        //form1.DataSource1.DataSet := Table1;
    end;
end;

// завтра
procedure TForm1.Button3Click(Sender: TObject);
var
  Present,              // сегодня
  Tomorrow: TDateTime;  // завтра

 begin
  Form1.Label2.Caption := 'Завтра';
  Present:= Now; // Now — функция, возвращает текущую дату
                 // и время
  Tomorrow := IncDay(Present);  // завтра

  Form1.Query1.SQL[3] :=
        '(Data = '''+ FormatDateTime('dd/mm/yyyy',Tomorrow)+''')';
  Form1.Query1.Open;
  if form1.Query1.RecordCount <> 0 then
        form1.DataSource1.DataSet := Form1.Query1
    else
        ShowMessage('На завтра никаких дел не запланировано.');
end;

// на этой неделе
procedure TForm1.Button4Click(Sender: TObject);
var
  Present: TDateTime;
  EndOfWeek: TDateTime;
```

```
 begin
  Form1.Label2.Caption := 'На этой неделе';
  Present:= Now; // Now — функция, возвращает текущую дату
                 // и время
  EndOfWeek := StartOfAWeek(YearOf(Present),WeekOf(Present)+1);

  Form1.Query1.SQL[3] :=
        '(Data >= '''+
FormatDateTime('dd/mm/yyyy',Present)+''')' +  'and'+
        '(Data <  '''+
FormatDateTime('dd/mm/yyyy',EndOfWeek)+''')';
  Form1.Query1.Open;
  if form1.Query1.RecordCount <> 0 then
       form1.DataSource1.DataSet := Form1.Query1
    else
       ShowMessage('На эту неделю никаких дел не запланировано.');
end;

// на следующей неделе
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  d1, d2: TDateTime;

 begin
  Form1.Label2.Caption := 'На следующей неделе';
  Present:= Now; // Now — функция, возвращает текущую дату
                 // и время
  d1 := StartOfAWeek(YearOf(Present),WeekOf(Present)+1);
  d2 := StartOfAWeek(YearOf(Present),WeekOf(Present)+2);
  Form1.Query1.SQL[3] :=
        '(Data >= '''+ FormatDateTime('dd/mm/yyyy',d1)+''')' +
        'and'+
        '(Data <  '''+ FormatDateTime('dd/mm/yyyy',d2)+''')';
  Form1.Query1.Open;
  if form1.Query1.RecordCount <> 0 then
       form1.DataSource1.DataSet := Form1.Query1
    else
       ShowMessage('На следующую неделю никаких дел
                    не запланировано.');
end;

// показать все записи
procedure TForm1.Button5Click(Sender: TObject);
```

```
begin
        Form1.Label2.Caption := 'Все, что намечено сделать';
        DataSource1.DataSet := Table1;
        Table1.Active := True;

end;

end.
```

Печать

68. Напишите программу, используя которую можно подготовить и распечатать накладную. Рекомендуемый вид формы программы приведен на рис. 1.77.

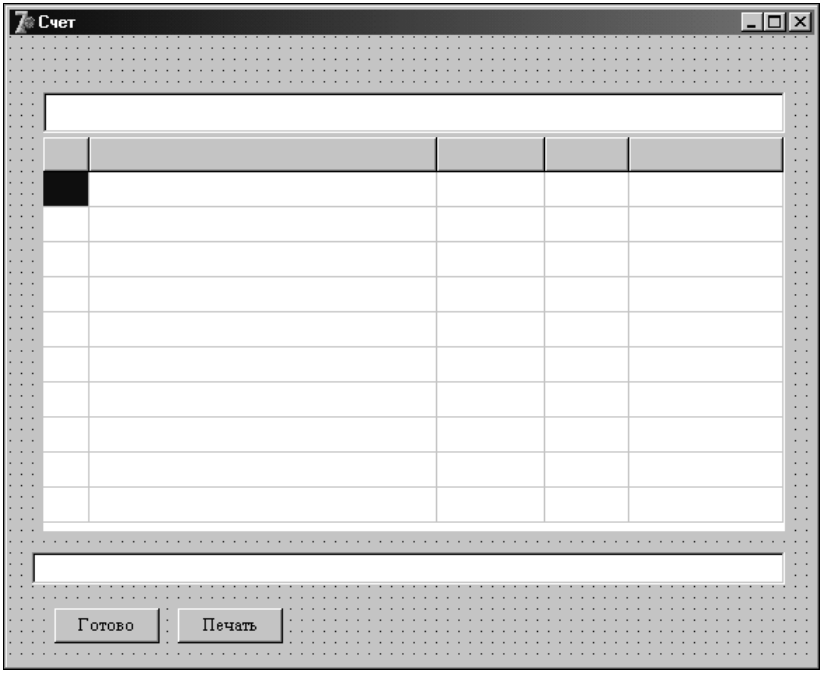


Рис. 1.77. Форма программы **Счет**

```
{ Программа демонстрирует только вывод на печать. }
unit schet_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,  Dialogs, StdCtrls, Grids, ExtCtrls;

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses Printers;

procedure TForm1.FormCreate(Sender: TObject);
var
    i: integer;
begin
    with StringGrid1 do
    begin
       Cells[0,0] := '   №';
       Cells[1,0] := '   Наименование';
       Cells[2,0] := '   Цена';
       Cells[3,0] := '   Кол-во';
       Cells[4,0] := '   Сумма';
```

```
      for i:=1 to 10 do
          Cells[0,i] := '   '+IntToStr(i);
    end;
end;

// щелчок на кнопке Печать
procedure TForm1.Button2Click(Sender: TObject);
const
    LEFT_MARGIN = 2; // отступ слева 2 см
    TOP_MARGIN  = 2; // отступ сверху 2 см
var
    dpiX, dpiY : integer; // разрешение принтера по X и Y
    kx,ky: real; // коэф. пересчета координат экрана
                 // в координаты принтера по X и Y
    // таблица
    p: array[0..4] of integer; // позиции колонок
    x1,y1,x2,y2: integer; // границы таблицы

    px,py: integer; // указатель точки вывода
    i,j: integer;

begin
    { Разрешение экрана и принтера разное,
      поэтому, чтобы добиться соответствия
      размеров изображения на экране и принтере,
      координаты точек экрана надо преобразовать
      в координаты принтера, домножить на коэф.,
      значение которого зависит от разрешения принтера.
      Например, если разрешение принтера 300 dpi,
      то значение коэффициента равно 3.125, т. к.
      разрешение экрана — 96 dpi.
    }

    // функция GetDeviceCaps позволяет получить характеристики
    // устройства. LOGPIXELSX — кол-во пикселов на дюйм по X
    dpiX := GetDeviceCaps(Printer.handle,LOGPIXELSX);
    dpiY := GetDeviceCaps(Printer.handle,LOGPIXELSY);
    kx := dpiX / Screen.PixelsPerInch;
    ky := dpiY / Screen.PixelsPerInch;

    px := Round(LEFT_MARGIN / 2.54 * dpiX);
    py := Round(TOP_MARGIN  / 2.54 * dpiY);
```

```
    // вычислим "принтерные" координаты колонок таблицы
    p[0] := px;
    for i:=1 to 4 do
    begin
        p[i] := p[i-1] + Round(StringGrid1.ColWidths[i-1]* kx);
    end;

    with Printer do
    begin
        BeginDoc; // открыть печать

        // заголовок таблицы
        Canvas.Font.Name := Edit1.Font.Name;
        Canvas.Font.Size := Edit1.Font.Size;
        Canvas.TextOut(px,py,Edit1.Text);

        // таблица — содержимое StringGrid1
        py := py+ Round(Edit1.Font.Size * 2 * ky);

        x1 := px; y1 := py; // левый верхний угол таблицы

        Canvas.Font.Name := StringGrid1.Font.Name;
        Canvas.Font.Size := StringGrid1.Font.Size;

        x2 := p[4] + Round(StringGrid1.ColWidths[4]* kx);
        y2 := py +
              Round(StringGrid1.RowCount *
StringGrid1.RowHeights[1] * ky);

        for j:=0 to StringGrid1.RowCount do
        begin
            // строки таблицы
            for i:=0 to StringGrid1.ColCount do
            begin
                Canvas.TextOut(P[i],py,StringGrid1.Cells[i,j]);
                // горизонтальная линия
                Canvas.MoveTo(p[0],py);
                Canvas.LineTo(x2,py);
            end;
            py:=py+ Round(StringGrid1.RowHeights[j]* ky);
        end;

        // вертикальные линии
        for i:=0 to StringGrid1.ColCount -1 do
```

```
        begin
           Canvas.MoveTo(p[i],y1);
           Canvas.LineTo(p[i],y2);
        end;
        Canvas.MoveTo(x2,y1);
        Canvas.LineTo(x2,y2);

        EndDoc;  // закрыть печать
    end;
end;

end.
```

Часть 2

DELPHI — КРАТКИЙ СПРАВОЧНИК

Вторая часть книги — краткий справочник по компонентам и функциям, которые использовались для решения задач, представленных в первой части.

Компоненты

В этом разделе приведено краткое описание базовых компонентов Delphi. Подробное описание этих и других компонентов можно найти в справочной системе.

Форма

Форма (объект тип `TForm`) является основой программы. Свойства формы (табл. 2.1) определяют вид окна программы.

Таблица 2.1. *Свойства формы (объекта* `TForm`*)*

Свойство	Описание
`Name`	Имя формы. В программе имя формы используется для управления формой и доступа к компонентам формы
`Caption`	Текст заголовка
`Top`	Расстояние от верхней границы формы до верхней границы экрана
`Left`	Расстояние от левой границы формы до левой границы экрана
`Width`	Ширина формы

Таблица 2.1 *(продолжение)*

Свойство	Описание
`Height`	Высота формы
`ClientWidth`	Ширина рабочей (клиентской) области формы, т. е. без учета ширины левой и правой границ
`ClientHeight`	Высота рабочей (клиентской) области формы, т. е. без учета высоты заголовка и ширины нижней границы формы
`BorderStyle`	Вид границы. Граница может быть обычной (`bsSizeable`), тонкой (`bsSingle`) или отсутствовать (`bsNone`). Если у окна обычная граница, то во время работы программы пользователь может при помощи мыши изменить размер окна. Изменить размер окна с тонкой границей нельзя. Если граница отсутствует, то на экран во время работы программы будет выведено окно без заголовка. Положение и размер такого окна во время работы программы изменить нельзя
`BorderIcons`	Кнопки управления окном. Значение свойства определяет, какие кнопки управления окном будут доступны пользователю во время работы программы. Значение свойства задается путем присвоения значений уточняющим свойствам `biSystemMenu`, `biMinimize`, `biMaximize` и `biHelp`. Свойство `biSystemMenu` определяет доступность кнопки **Свернуть** и кнопки системного меню, `biMinimize` — кнопки **Свернуть**, `biMaximize` — кнопки **Развернуть**, `biHelp` — кнопки вывода справочной информации
`Icon`	Значок в заголовке диалогового окна, обозначающий кнопку вывода системного меню
`Color`	Цвет фона. Цвет можно задать, указав название цвета или привязку к текущей цветовой схеме операционной системы. Во втором случае цвет определяется текущей цветовой схемой, выбранным компонентом привязки и меняется при изменении цветовой схемы операционной системы
`Font`	Шрифт. Шрифт, используемый "по умолчанию" компонентами, находящимися на поверхности формы. Изменение свойства `Font` формы приводит к автоматическому изменению свойства `Font` компонента,

Таблица 2.1 *(окончание)*

Свойство	Описание
(прод.)	располагающегося на поверхности формы. То есть компоненты наследуют свойство `Font` от формы (имеется возможность запретить наследование)
`Canvas`	Поверхность, на которую можно вывести графику

Label

Компонент `Label` (рис. 2.1) предназначен для вывода текста на поверхность формы. Свойства компонента (табл. 2.2) определяют вид и расположение текста.

Рис. 2.1. Компонент `Label` — поле вывода текста

Таблица 2.2. *Свойства компонента* `Label` *(поле вывода текста)*

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`Caption`	Отображаемый текст
`Left`	Расстояние от левой границы поля вывода до левой границы формы
`Top`	Расстояние от верхней границы поля вывода до верхней границы формы
`Height`	Высота поля вывода
`Width`	Ширина поля вывода
`AutoSize`	Признак того, что размер поля определяется его содержимым

***Таблица 2.2** (окончание)*

Свойство	Описание
`WordWrap`	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (значение свойства `AutoSize` должно быть `False`)
`Alignment`	Задает способ выравнивания текста внутри поля. Текст может быть выравнен по левому краю (`taLeftJustify`), по центру (`taCenter`) или по правому краю (`taRightJustify`)
`Font`	Шрифт, используемый для отображения текста. Уточняющие свойства определяют способ начертания символов (`Font.Name`), размер (`Font.Size`) и цвет символов (`Font.Color`)
`ParentFont`	Признак наследования компонентом характеристик шрифта формы, на которой находится компонент. Если значение свойства равно `True`, то текст выводится шрифтом, установленным для формы
`Color`	Цвет фона области вывода текста
`Transparent`	Управляет отображением фона области вывода текста. Значение `True` делает область вывода текста прозрачной (область вывода не закрашивается цветом, заданным свойством `Color`)
`Visible`	Позволяет скрыть текст (`False`) или сделать его видимым (`True`)

Edit

Компонент `Edit` (рис. 2.2) представляет собой поле ввода-редактирования строки символов. Свойства компонента приведены в табл. 2.3.

Рис. 2.2. Компонент `Edit` — поле ввода-редактирования строки символов

Таблица 2.3. *Свойства компонента* `Edit` *(поле редактирования)*

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам, в частности — для доступа к тексту, введенному в поле редактирования
`Text`	Текст, находящийся в поле ввода и редактирования
`Left`	Расстояние от левой границы компонента до левой границы формы
`Top`	Расстояние от верхней границы компонента до верхней границы формы
`Height`	Высота поля
`Width`	Ширина поля
`Font`	Шрифт, используемый для отображения вводимого текста
`ParentFont`	Признак наследования компонентом характеристик шрифта формы, на которой находится компонент. Если значение свойства равно `True`, то при изменении свойства `Font` формы автоматически меняется значение свойства `Font` компонента
`Enabled`	Используется для ограничения возможности изменить текст в поле редактирования. Если значение свойства равно `False`, то текст в поле редактирования изменить нельзя
`Visible`	Позволяет скрыть компонент (`False`) или сделать его видимым (`True`)

Button

Компонент `Button` (рис. 2.3) представляет собой командную кнопку. Свойства компонента приведены в табл. 2.4.

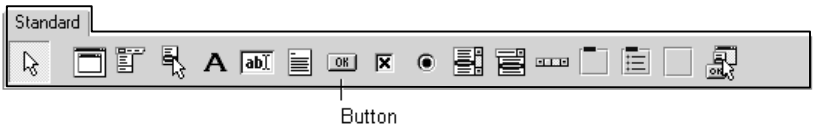


Рис. 2.3. Компонент `Button` — командная кнопка

Таблица 2.4. *Свойства компонента* `Button` *(командная кнопка)*

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`Caption`	Текст на кнопке
`Left`	Расстояние от левой границы кнопки до левой границы формы
`Top`	Расстояние от верхней границы кнопки до верхней границы формы
`Height`	Высота кнопки
`Width`	Ширина кнопки
`Enabled`	Признак доступности кнопки. Если значение свойства равно `True`, то кнопка доступна. Если значение свойства равно `False`, то кнопка не доступна, например, в результате щелчка на кнопке событие `Click` не возникает
`Visible`	Позволяет скрыть кнопку (`False`) или сделать ее видимой (`True`)
`Hint`	Подсказка — текст, который появляется рядом с указателем мыши при позиционировании указателя на командной кнопке (для того чтобы текст появился, надо чтобы значение свойства `ShowHint` было `True`)
`ShowHint`	Разрешает (`True`) или запрещает (`False`) отображение подсказки при позиционировании указателя на кнопке

Memo

Компонент `Memo` (рис. 2.4) представляет собой элемент редактирования текста, который может состоять из нескольких строк. Свойства компонента приведены в табл. 2.5.

Рис. 2.4. Компонент `Memo`

Таблица 2.5. Свойства компонента Memo

Свойство	Описание
Name	Имя компонента. Используется в для доступа к свойствам компонента
Text	Текст, находящийся в поле Memo. Рассматривается как единое целое
Lines	Массив строк, соответствующий содержимому поля. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля
Lines.Count	Количество строк текста в поле Memo
Left	Расстояние от левой границы поля до левой границы формы
Top	Расстояние от верхней границы поля до верхней границы формы
Height	Высота поля
Width	Ширина поля
Font	Шрифт, используемый для отображения вводимого текста
ParentFont	Признак наследования свойств шрифта родительской формы

RadioButton

Компонент RadioButton (рис. 2.5) представляет зависимую кнопку, состояние которой определяется состоянием других кнопок группы. Свойства компонента приведены в табл. 2.6.

Если в диалоговом окне надо организовать несколько групп переключателей, то каждую группу следует представить компонентом RadioGroup.

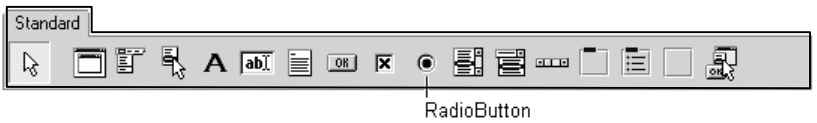


Рис. 2.5. Компонент RadioButton

Таблица 2.6. *Свойства компонента* `RadioButton`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента
`Caption`	Текст, который находится справа от кнопки
`Checked`	Состояние, внешний вид кнопки: если кнопка выбрана, то `Checked = True`; если кнопка не выбрана, то `Checked = False`
`Left`	Расстояние от левой границы флажка до левой границы формы
`Top`	Расстояние от верхней границы флажка до верхней границы формы
`Height`	Высота поля вывода поясняющего текста
`Width`	Ширина поля вывода поясняющего текста
`Font`	Шрифт, используемый для отображения поясняющего текста
`ParentFont`	Признак наследования характеристик шрифта родительской формы

CheckBox

Компонент `CheckBox` (рис. 2.6) представляет собой независимую кнопку (переключатель). Свойства компонента приведены в табл. 2.7.

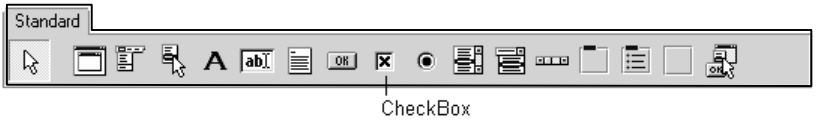


Рис. 2.6. Компонент `CheckBox`

Таблица 2.7. *Свойства компонента* `CheckBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента
`Caption`	Текст, который находится справа от флажка
`Checked`	Состояние, внешний вид флажка: если флажок установлен (в квадратике есть "галочка"), то `Checked = True`; если флажок сброшен (нет "галочки"), то `Checked =False`
`State`	Состояние флажка. В отличие от свойства `Checked`, позволяет различать установленное, сброшенное и промежуточное состояния. Состояние флажка определяет одна из констант: `cbChecked` (установлен); `cbGrayed` (серый, неопределенное состояние); `cbUnChecked` (сброшен)
`AllowGrayed`	Свойство определяет, может ли флажок быть в промежуточном состоянии: если `AllowGrayed = False`, то флажок может быть только установленным или сброшенным; если `AllowGrayed = True`, то допустимо промежуточное состояние
`Left`	Расстояние от левой границы флажка до левой границы формы
`Top`	Расстояние от верхней границы флажка до верхней границы формы
`Height`	Высота поля вывода поясняющего текста
`Width`	Ширина поля вывода поясняющего текста
`Font`	Шрифт, используемый для отображения поясняющего текста
`ParentFont`	Признак наследования характеристик шрифта родительской формы

ListBox

Компонент ListBox (рис. 2.7) представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. 2.8.

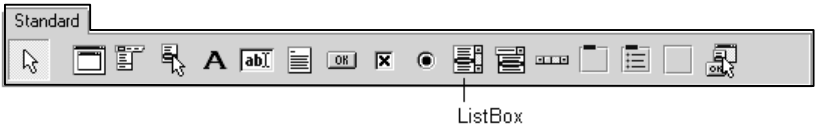


Рис. 2.7. Компонент ListBox

Таблица 2.8. *Свойства компонента* ListBox

Свойство	Описание
Name	Имя компонента. В программе используется для доступа к компоненту и его свойствам
Items	Элементы списка — массив строк
Count	Количество элементов списка
Sorted	Признак необходимости автоматической сортировки (True) списка после добавления очередного элемента
ItemIndex	Номер выбранного элемента (элементы списка нумеруются с нуля). Если в списке ни один из элементов не выбран, то значение свойства равно минус 1
Left	Расстояние от левой границы списка до левой границы формы
Top	Расстояние от верхней границы списка до верхней границы формы
Height	Высота поля списка
Width	Ширина поля списка
Font	Шрифт, используемый для отображения элементов списка
ParentFont	Признак наследования свойств шрифта родительской формы

ComboBox

Компонент ComboBox (рис. 2.8) дает возможность ввести данные в поле редактирования путем набора на клавиатуре или выбором из списка. Свойства компонента приведены в табл. 2.9.

Рис. 2.8. Компонент ComboBox

Таблица 2.9. *Свойства компонента* ComboBox

Свойство	Описание
Name	Имя компонента. Используется для доступа к свойствам компонента
Text	Текст, находящийся в поле ввода-редактирования
Items	Элементы списка — массив строк
Count	Количество элементов списка
ItemIndex	Номер элемента, выбранного в списке. Если ни один из элементов списка не был выбран, то значение свойства равно минус 1
Sorted	Признак необходимости автоматической сортировки (True) списка после добавления очередного элемента
DropDownCount	Количество отображаемых элементов в раскрытом списке. Если количество элементов списка больше чем DropDownCount, то появляется вертикальная полоса прокрутки
Left	Расстояние от левой границы компонента до левой границы формы
Top	Расстояние от верхней границы компонента до верхней границы формы
Height	Высота компонента (поля ввода-редактирования)

Таблица 2.9 *(окончание)*

Свойство	Описание
`Width`	Ширина компонента
`Font`	Шрифт, используемый для отображения элементов списка
`ParentFont`	Признак наследования свойств шрифта родительской формы

StringGrid

Компонент `StringGrid` (рис. 2.9) представляет собой таблицу, ячейки которой содержат строки символов. Свойства компонента приведены в табл. 2.10.

Рис. 2.9. Компонент `StringGrid`

Таблица 2.10. *Свойства компонента* `StringGrid`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к компоненту и его свойствам
`ColCount`	Количество колонок таблицы
`RowCount`	Количество строк таблицы
`DefaultColWidth`	Ширина колонок таблицы
`DefaultRowHeight`	Высота строк таблицы
`FixedCols`	Количество зафиксированных слева колонок таблицы. Зафиксированные колонки выделяются цветом и при горизонтальной прокрутке таблицы остаются на месте

Таблица 2.10 *(окончание)*

Свойство	Описание
FixedRows	Количество зафиксированных сверху строк таблицы. Зафиксированные строки выделяются цветом и при вертикальной прокрутке таблицы остаются на месте
Cells	Соответствующий таблице двумерный массив. Ячейке таблицы, находящейся на пересечении столбца с номером col и строки с номером row, соответствует элемент cells[col,row]
GridLineWidth	Ширина линий, ограничивающих ячейки таблицы
Left	Расстояние от левой границы поля таблицы до левой границы формы
Top	Расстояние от верхней границы поля таблицы до верхней границы формы
Height	Высота поля таблицы
Width	Ширина поля таблицы
Options.goEditing	Признак допустимости редактирования содержимого ячеек таблицы. True — редактирование разрешено, False — запрещено
Options.goTab	Разрешает (True) или запрещает (False) использование клавиши <Tab> для перемещения курсора в следующую ячейку таблицы
Options. goAlwaysShowEditor	Признак нахождения компонента в режиме редактирования. Если значение свойства False, то для того, чтобы в ячейке появился курсор, надо начать набирать текст, нажать клавишу <F2> или сделать щелчок мышью
Font	Шрифт, используемый для отображения содержимого ячеек таблицы
ParentFont	Признак наследования характеристик шрифта формы

Image

Компонент Image (рис. 2.10) обеспечивает вывод на поверхность формы иллюстраций, представленных в bmp-формате (чтобы компонент можно было использовать для отображения иллюстраций в формате JPG, надо подключить модуль JPEG — указать имя модуля в директиве uses). Свойства компонента Image приведены в табл. 2.11.

Рис. 2.10. Компонент Image

Таблица 2.11. *Свойства компонента* Image

Свойство	Описание
Picture	Иллюстрация, которая отображается в поле компонента
Width, Height	Размер компонента. Если размер компонента меньше размера иллюстрации, и значение свойств AutoSize, Strech и Proportional равно False, то отображается часть иллюстрации
Proportional	Признак автоматического масштабирования картинки без искажения. Чтобы масштабирование было выполнено, значение свойства AutoSize должно быть False
Strech	Признак автоматического масштабирования (сжатия или растяжения) иллюстрации в соответствии с реальным размером компонента. Если размер компонента не пропорционален размеру иллюстрации, то иллюстрация будет искажена
AutoSize	Признак автоматического изменения размера компонента в соответствии с реальным размером иллюстрации

Таблица 2.11 (окончание)

Свойство	Описание
Center	Признак определяет расположение картинки в поле компонента по горизонтали, если ширина картинки меньше ширины поля компонента. Если значение свойства равно False, то картинка прижата к правой границе компонента, если True — то картинка располагается по центру
Visible	Отображается ли компонент и, соответственно, иллюстрация на поверхности формы
Canvas	Поверхность, на которую можно вывести графику

Timer

Компонент Timer (рис. 2.11) обеспечивает генерацию последовательности событий OnTimer. Свойства компонента приведены в табл. 2.12.

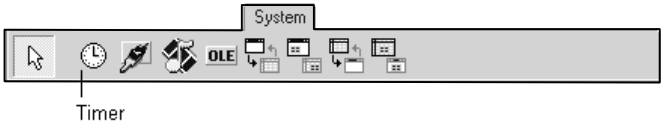


Рис. 2.11. Компонент Timer

Таблица 2.12. Свойства компонента Timer

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту
Interval	Период генерации события OnTimer. Задается в миллисекундах
Enabled	Разрешение работы. Разрешает (значение True) или запрещает (значение False) генерацию события OnTimer

Animate

Компонент Animate (рис. 2.12) позволяет воспроизводить простую, не сопровождаемую звуком анимацию, кадры которой находятся в AVI-файле. Свойства компонента приведены в табл. 2.13.

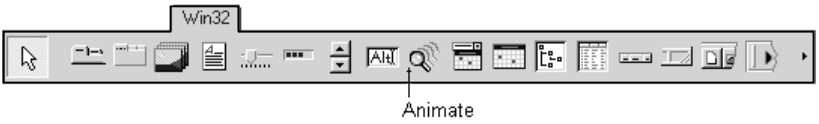


Рис. 2.12. Компонент Animate

Таблица 2.13. *Свойства компонента* Animate

Свойство	Описание
Name	Имя компонента. Используется для доступа к свойствам компонента и управлением его поведением
FileName	Имя AVI-файла, в котором находится анимация, отображаемая при помощи компонента
StartFrame	Номер кадра, с которого начинается отображение анимации
StopFrame	Номер кадра, на котором заканчивается отображение анимации
Activate	Признак активизации процесса отображения кадров анимации
Color	Цвет фона компонента (цвет "экрана"), на котором воспроизводится анимация
Transparent	Режим использования "прозрачного" цвета при отображении анимации
Repetitions	Количество повторов отображения анимации

MediaPlayer

Компонент MediaPlayer (рис. 2.13) позволяет воспроизвести видеоролик, звук и сопровождаемую звуком анимацию. Свойства компонента приведены в табл. 2.14.

Рис. 2.13. Компонент `MediaPlayer`

Таблица 2.14. *Свойства компонента* `MediaPlayer`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента и управлением работой плеера
`DeviceType`	Тип устройства. Определяет конкретное устройство, которое представляет собой компонент `MediaPlayer`. Тип устройства задается именованной константой: `dtAutoSelect` — тип устройства определяется автоматически; `dtVaweAudio` — проигрыватель звука; `dtAVIVideo` — видеопроигрыватель; `dtCDAudio` — CD-проигрыватель
`FileName`	Имя файла, в котором находится воспроизводимый звуковой фрагмент или видеоролик
`AutoOpen`	Признак автоматического открытия сразу после запуска программы, файла видеоролика или звукового фрагмента
`Display`	Определяет компонент, на поверхности которого воспроизводится видеоролик (обычно в качестве экрана для отображения видео используют компонент `Panel`)
`VisibleButtons`	Составное свойство. Определяет видимые кнопки компонента. Позволяет сделать невидимыми некоторые кнопки

SpeedButton

Компонент `SpeedButton` (рис. 2.14) представляет собой кнопку, на поверхности которой находится картинка. Свойства компонента приведены в табл. 2.15.

Рис. 2.14. Компонент `SpeedButton`

Таблица 2.15. *Свойства компонента* `SpeedButton`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Glyph`	Битовый образ, в котором находятся картинки для каждого из состояний кнопки. В битовом образе может быть до четырех изображений кнопки (рис. 2.15)
`NumGlyphs`	Количество картинок в битовом образе `Glyph`
`Flat`	Свойство `Flat` определяет вид кнопки (наличие границы). Если значение свойства равно `True`, то граница кнопки появляется только при позиционировании указателя мыши на кнопке
`GroupIndex`	Идентификатор группы кнопок. Кнопки, имеющие одинаковый идентификатор группы, работают подобно переключателям: нажатие одной из кнопок группы вызывает срабатывание других кнопок этой группы
`Down`	Идентификатор состояния кнопки. Изменить значение свойства можно, если значение свойства `GroupIndex` не равно 0
`Left`	Расстояние от левой границы кнопки до левой границы формы
`Top`	Расстояние от верхней границы кнопки до верхней границы формы
`Height`	Высота кнопки
`Width`	Ширина кнопки
`Enabled`	Признак доступности кнопки. Если значение свойства равно `True`, то кнопка доступна. Если значение свойства равно `False`, то кнопка не доступна

Таблица 2.15 *(окончание)*

Свойство	Описание
`Visible`	Позволяет скрыть кнопку (`False`) или сделать ее видимой (`True`)
`Hint`	Подсказка — текст, который появляется рядом с указателем мыши при позиционировании указателя на командной кнопке (для того чтобы текст появился, надо чтобы значение свойства `ShowHint` было `True`)
`ShowHint`	Разрешает (`True`) или запрещает (`False`) отображение подсказки при позиционировании указателя на кнопке

Рис. 2.15. Структура и пример битового образа `Glyph`: картинки, соответствующие состоянию кнопки

UpDown

Компонент `UpDown` (рис. 2.16) представляет собой две кнопки, используя которые можно изменить значение внутренней переменной-счетчика на определенную величину. Увеличение или уменьшение значения происходит при каждом щелчке на одной из кнопок. Свойства компонента приведены в табл. 2.16.

Рис. 2.16. Компонент `UpDown`

Таблица 2.16. *Свойства компонента* `UpDown`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Position`	Счетчик. Значение свойства изменяется в результате щелчка на кнопке `Up` (увеличивается) или `Down` (уменьшается). Диапазон изменения определяют свойства `Min` и `Max`, величину изменения — свойство `Increment`
`Min`	Нижняя граница диапазона изменения свойства `Position`
`Max`	Верхняя граница диапазона изменения свойства `Position`
`Increment`	Величина, на которую изменяется значение свойства `Position` в результате щелчка на одной из кнопок компонента
`Associate`	Определяет компонент (`Edit` — поле ввода-редактирования), используемый в качестве индикатора значения свойства `Position`. Если значение свойства задано, то при изменении содержимого поля редактирования автоматически меняется значение свойства `Position`
`Orientation`	Задает ориентацию кнопок компонента. Кнопки могут быть ориентированы вертикально (`udVertical`) или горизонтально (`udHorizontal`)

Table

Компонент `Table` (рис. 2.17) представляет всю таблицу базы данных. Свойства компонента приведены в табл. 2.17.

Рис. 2.17. Компонент `Table` — таблица базы данных

Таблица 2.17. *Свойства компонента* `Table`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента
`DatabaseName`	Имя базы данных, частью которой является таблица (файл данных), для доступа к которой используется компонент. В качестве значения свойства следует применять псевдоним базы данных
`TableName`	Имя файла данных (таблицы данных), для доступа к которому используется компонент
`TableType`	Тип таблицы. Таблица может быть набором данных в формате Paradox (`ttParadox`), dBase (`ttDBase`), FoxPro (`ttFoxPro`) или представлять собой форматированный текстовый файл (`ttASCII`)
`Active`	Признак того, что таблица активна (файл данных открыт). В результате присваивания свойству значения `True` происходит открытие файла таблицы

Query

Компонент `Query` (рис. 2.18) представляет часть базы данных — записи, содержимое которых удовлетворяют критерию SQL-запроса к таблице. Свойства компонента приведены в табл. 2.18.

Рис. 2.18. Компонент `Query` обеспечивает выбор информации из базы данных

Таблица 2.18. *Свойства компонента* `Query`

Свойство	Описание
`Name`	Имя компонента. Используется компонентом `DataSource` для связи результата выполнения запроса (набора записей) с компонентом, обеспечивающим просмотр записей, например `DBGrid`

Таблица 2.18 (окончание)

Свойство	Описание
SQL	Записанный на языке SQL-запрос к базе данных (к таблице)
Active	При присвоении свойству значения True активизирует выполнение запроса

DataSource

Компонент DataSource (рис. 2.19) обеспечивает связь между данными, представленными компонентом Table или Query и компонентами отображения данных (DBEdit, DBMemo, DBGrid). Свойства компонента приведены в табл. 2.19.

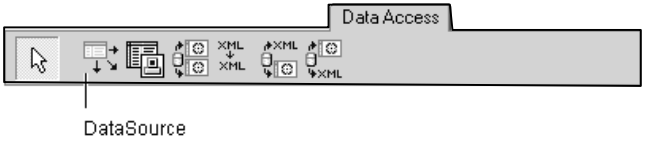


Рис. 2.19. Компонент DataSource обеспечиват связь между данными и компонентом просмотра-редактирования

Таблица 2.19. Свойства компонента DataSource

Свойство	Описание
Name	Имя компонента. Используется компонентом отображения данных для доступа к компоненту и, следовательно, к данным, связь с которыми обеспечивает компонент
DataSet	Компонент, представляющий собой входные данные (Table или Query)

DBEdit, DBMemo, DBText

Компоненты DBEdit и DBMemo (рис. 2.20) обеспечивают просмотр и редактирование полей записи базы данных, компонент

DBText — только просмотр. Свойства компонентов приведены в табл. 2.20.

Рис. 2.20. Компоненты просмотра и редактирования полей БД

Таблица 2.20. *Свойства компонентов* DBText, DBEdit *и* DBMemo

Свойство	Описание
Name	Имя компонента. Используется для доступа к свойствам компонента
DataSource	Компонент-источник данных
DataField	Поле базы данных, для отображения или редактирования которого используется компонент

DBGrid

Компонент DBGrid (рис. 2.21) используется для просмотра и редактирования базы данных в режиме таблицы. Свойства компонента приведены в табл. 2.21.

Рис. 2.21. Компонент DBGrid обеспечивает работу с базой данных в режиме таблицы

Таблица 2.21. *Свойства компонента* DBGrid

Свойство	Описание
Name	Имя компонента

Таблица 2.21 *(окончание)*

Свойство	Описание
DataSource	Источник отображаемых в таблице данных (компонент DataSource)
Columns	Свойство Columns представляет собой массив компонентов типа Column, каждый из которых определяет колонку таблицы и отображаемую в ней информацию (табл. 2.22)
Options.dgTitles	Разрешает вывод строки заголовка столбцов
Options.dgIndicator	Разрешает вывод колонки индикатора. Во время работы с базой данных текущая запись помечается в колонке индикатора треугольником, новая запись — звездочкой, редактируемая — специальным значком
Options.dgColumnResize	Разрешает менять во время работы программы ширину колонок таблицы
Options.dgColLines	Разрешает выводить линии, разделяющие колонки таблицы
Options.dgRowLines	Разрешает выводить линии, разделяющие строки таблицы

Column

Таблица 2.22. *Свойства компонента* Column

Свойство	Описание
FieldName	Поле записи, содержимое которого выводится в колонке
Width	Ширина колонки в пикселах
Font	Шрифт, используемый для вывода текста в ячейках колонки
Color	Цвет фона колонки

Таблица 2.22 *(окончание)*

Свойство	Описание
`Alignment`	Способ выравнивания текста в ячейках колонки. Текст может быть выравнен по левому краю (`taLeftJustify`), по центру (`taCenter`) или по правому краю (`taRightJustify`)
`Title.Caption`	Заголовок колонки. Значением по умолчанию является имя поля записи
`Title.Alignment`	Способ выравнивания заголовка колонки. Заголовок может быть выравнен по левому краю (`taLeftJustify`), по центру (`taCenter`) или по правому краю (`taRightJustify`)
`Title.Color`	Цвет фона заголовка колонки
`Title.Font`	Шрифт заголовка колонки

DBNavigator

Компонент `DBNavigator` (рис. 2.22 и 2.23) обеспечивает перемещение указателя текущей записи, активизацию режима редактирования, добавление и удаление записей. Компонент представляет собой совокупность командных кнопок (табл. 2.23). Свойства компонента приведены в табл. 2.24.

Рис. 2.22. Значок компонента `DBNavigator`

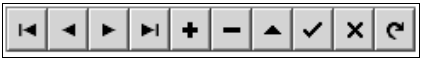


Рис. 2.23. Компонент `DBNavigator`

Таблица 2.23. *Кнопки компонента* `DBNavigator`

Кнопка	Обозначение	Действие
 К первой	`nbFirst`	Указатель текущей записи перемещается к первой записи файла данных
 К предыдущей	`nbPrior`	Указатель текущей записи перемещается к предыдущей записи файла данных
 К следующей	`nbNext`	Указатель текущей записи перемещается к следующей записи файла данных
 К последней	`nbLast`	Указатель текущей записи перемещается к последней записи файла данных
 Добавить	`nbInsert`	В файл данных добавляется новая запись
 Удалить	`nbDelete`	Удаляется текущая запись файла данных
 Редактирование	`nbEdit`	Устанавливает режим редактирования текущей записи
 Сохранить	`nbPost`	Изменения, внесенные в текущую запись, записываются в файл данных
 Отменить	`Cancel`	Отменяет внесенные в текущую запись изменения
 Обновить	`nbRefresh`	Записывает внесенные изменения в файл

Таблица 2.24. *Свойства компонента* `DBNavigator`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента

Таблица 2.24 *(окончание)*

Свойство	Описание
DataSource	Имя компонента, являющегося источником данных. В качестве источника данных может выступать база данных (компонент Database), таблица (компонент Table) или результат выполнения запроса (компонент Query)
VisibleButtons	Видимые командные кнопки

Графика

Canvas

Canvas — это поверхность (формы или компонента Image), на которой соответствующие методы (табл. 2.25) могут вычерчивать графические примитивы. Вид графических элементов определяют свойства поверхности, на которой эти элементы вычерчиваются (табл. 2.26).

Таблица 2.25. *Методы объекта* Canvas

Метод	Описание
TextOut(x,y,s)	Выводит строку s от точки с координатами (x, y). Шрифт определяет свойство Font поверхности (Canvas), на которую выводится тест, цвет закраски области вывода текста — свойство Brush этой же поверхности
Draw(x,y,b)	Выводит от точки с координатами (x, y) битовый образ b. Если значение свойства Transparent поверхности, на которую выполняется вывод, равно True, то точки, цвет которых совпадают с цветом левой нижней точки битового образа, не отображаются

Таблица 2.25 *(продолжение)*

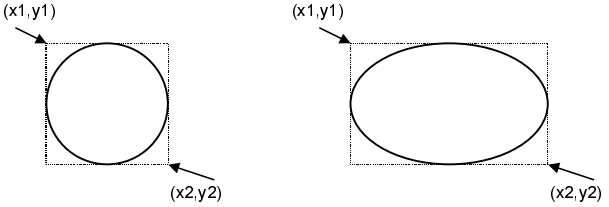
Метод	Описание
`LineTo(x,y)`	Вычерчивает линию из текущей точки в точку с указанными координатами. Вид линии определяет свойство `Pen`
`MoveTo(x,y)`	Перемещает указатель текущей точки в точку с указанными координатами
`PolyLine(pl)`	Вычерчивает ломаную линии. Координаты точек перегиба задает параметр `pl` — массив структур типа `TPoint`. Если первый и последний элементы массива одинаковые, то будет вычерчен замкнутый контур. Вид линии определяет свойство `Pen`
`Polygon(pl)`	Вычерчивает и закрашивает многоугольник. Координаты углов задает параметр `pl` — массив структур типа `TPoint`. Первый и последний элементы массива должны быть одинаковые. Вид границы определяет свойство `Pen`, цвет и стиль закраски внутренней области — свойство `Brush`
`Ellipse(x1,y, x2,y2)`	Вычерчивает эллипс, окружность или круг. Параметры `x1`, `y1`, `x2` и `y2` задают размер прямоугольника, в который вписывается эллипс. Вид линии определяет свойство `Pen` 
`Arc(x1,y1,x2,y2, x3,y3,x4,y4)`	Вычерчивает дугу. Параметры `x1`, `y1`, `x2`, `y2` определяют эллипс, из которого вырезается дуга, параметры `x2`, `y2`, `x3` и `y4` — координаты концов дуги. Дуга вычерчивается против часовой стрелки от точки (x3,y3) к точке (x4,y4).

Таблица 2.25 *(окончание)*

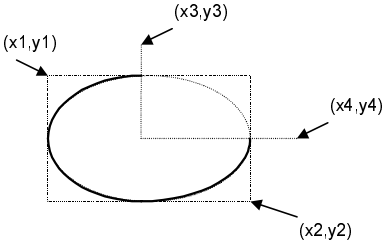
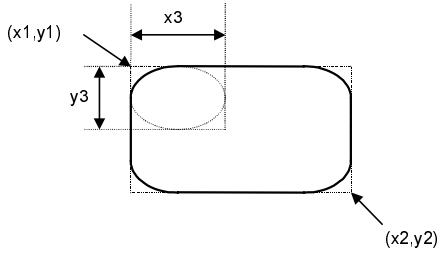
Метод	Описание
(прод.)	Вид линии (границы) определяет свойство `Pen`, цвет и способ закраски внутренней области — свойство `Brush` 
`Rectangle(x1,y, x2,y2)`	Вычерчивает прямоугольник. Параметры `x1`, `y1`, `x2` и `y2` задают координаты левого верхнего и правого нижнего углов. Вид линии определяет свойство `Pen`, цвет и способ закраски внутренней области — свойство `Brush`
`RoundRec(x1,y1, x2,y2, x3,y3)`	Вычерчивает прямоугольник со скругленными углами. Параметры `x1`, `y1`, `x2` и `y2` задают координаты левого верхнего и правого нижнего углов, `x3` и `y3` — радиус скругления. Вид линии определяет свойство `Pen`, цвет и способ закраски внутренней области — свойство `Brush` 

Таблица 2.26. *Свойства объекта* `Canvas`

Свойство	Описание
`Transparent`	Признак использования "прозрачного" цвета при выводе битового образа методом `Draw`. Если значение свойства равно `True`, то точки, цвет которых совпадают с цветом левой нижней точки битового образа, не отображаются
`Pen`	Свойство `Pen` представляет собой объект (см. табл. 2.27), уточняющие свойства которого определяют цвет, толщину и стиль линий, вычерчиваемых методами вывода графических примитивов
`Brush`	Свойство `Brush` представляет собой объект (см. табл. 2.28), уточняющие свойства которого определяют цвет и стиль закраски областей, вычерчиваемых методами вывода графических примитивов
`Font`	Свойство `Font` представляет собой объект, уточняющие свойства которого определяют шрифт (название, размер, цвет, способ оформления), используемый для вывода на поверхность холста текста

Pen

Таблица 2.27. *Свойства объекта* `Pen`

Свойство	Описание
`Color`	Цвет линии (`clBlack` — черный; `clMaroon` — каштановый; `clGreen` — зеленый; `clOlive` — оливковый; `clNavy` — темно-синий; `clPurple` — розовый; `clTeal` — зелено-голубой; `clGray` — серый; `clSilver` — серебристый; `clRed` — красный; `clLime` — салатный; `clBlue` — синий; `clFuchsia` — ярко-розовый; `clAqua` — бирюзовый; `clWhite` – белый)
`Style`	Вид линии. Линия может быть: `psSolid` — сплошная; `psDash` — пунктирная (длинные штрихи); `psDot` — пунктирная (короткие штрихи); `psDashDot` — пунктирная (чередование длинного и короткого штрихов);

Таблица 2.27 (окончание)

Свойство	Описание
(прод.)	`psDashDotDot` — пунктирная (чередование одного длинного и двух коротких штрихов); `psClear` — не отображается (используется, если не надо изображать границу, например, прямоугольника)
`Width`	Толщина линии задается в пикселах. Толщина пунктирной линии не может быть больше 1

Brush

Таблица 2.28. Свойства объекта `TBrush` (кисть)

Свойство	Описание
`Color`	Цвет закрашивания замкнутой области
`Style`	Стиль (тип) заполнения области (`bsSolid` — сплошная заливка; `bsClear` — область не закрашивается; `bsHorizontal` — горизонтальная штриховка; `bsVertical` — вертикальная штриховка; `bsFDiagonal` — диагональная штриховка с наклоном линий вперед; `bsBDiagonal` — диагональная штриховка с наклоном линий назад; `bsCross` — горизонтально-вертикальная штриховка, в клетку; `bsDiagCross` — диагональная штриховка, в клетку)

Функции

В этом разделе приведено краткое описание наиболее часто используемых функций. Подробнее о них можно прочитать в справочной системе.

Ввода и вывода

Таблица 2.29. *Функции ввода и вывода*

Функция	Описание
InputBox(*Заголовок*, *Подсказка*, *Значение*)	В результате выполнения функции на экране появляется диалоговое окно, в поле которого пользователь может ввести строку символов. Значением функции является введенная строка. Параметр *Значение* задает значение функции "по умолчанию", т. е. строку, которая будет в поле редактирования в момент появления окна
ShowMessage(s)	Процедура ShowMessage выводит окно, в котором находится сообщение s и командная кнопка **OK**
MessageDlg(s,t,b,h)	Выводит на экран диалоговое окно с сообщением s и возвращает код кнопки, щелчком на которой пользователь закрыл окно. Параметр t определяет тип окна: mtWarning — Внимание; mtError — ошибка; mtInformation — информация; mtConfirmation — запрос; mtCustom — пользовательское (без значка). Параметр b (множество — заключенный в квадратные скобки список констант) задает командные кнопки диалогового окна (mbYes, mbNo, mbOK, mbCancel, mbHelp, mbAbort, mbRetry, mbIgnore и mbAll). Параметр h задает раздел справочной системы программы, который появится в результате нажатия кнопки **Help** или клавиши <F1>. Если справочная система не используется, значение параметра должно быть 0. Значением функции может быть одна из констант: mrAbort, mrYes, mrOk, mrRetry, mrNo, mrCancel, mrIgnore или mrAll, обозначающая соответствующую командную кнопку

Математические

Таблица 2.30. *Математические функции*

Функция	Значение
`Abs(n)`	Абсолютное значение `n`
`Sqrt(n)`	Квадратный корень из `n`
`Sqr(n)`	Квадрат `n`
`Exp(n)`	Экспонента `n`
`Ln(n)`	Натуральный логарифм `n`
`Rardom(n)`	Случайное целое число в диапазоне от 0 до `n-1` (перед первым обращением к функции необходимо вызвать функцию `Randomize`, которая выполнит инициализацию программного генератора случайных чисел)
`Sin(α)`	Синус выраженного в радианах угла `α`
`Cos(α)`	Косинус выраженного в радианах угла `α`
`Arctan(α)`	Арктангенс выраженного в радианах угла `α`

Величина угла тригонометрических функций должна быть выражена в радианах. Для преобразования величины угла из градусов в радианы используется формула `(a*3.1415256)/180`, где: `a` — величина угла в градусах; `3.1415926` — число "ПИ". Вместо константы `3.1415926` можно использовать стандартную именованную константу `PI`.

Преобразования

Таблица 2.31. *Функции преобразования*

Функция	Значение
`Chr(n)`	Символ, код которого равен `n`

Таблица 2.31 *(окончание)*

Функция	Значение
`IntToStr(k)`	Строка, являющаяся изображением целого `k`
`FloatToStr(n)`	Строка, являющаяся изображением вещественного `n`
`FloatToStrF(n,f,k,m)`	Строка, являющаяся изображением вещественного `n`. При вызове функции указывают: `f` — формат; `k` — точность; `m` — количество цифр после десятичной точки. Формат определяет способ изображения числа: `ffGeneral` — универсальный; `ffExponent` — научный; `ffFixed` — с фиксированной точкой; `ffNumber` — с разделителями групп разрядов; `ffCurrency` — финансовый. Точность — нужное общее количество цифр: 7 или меньше для значения типа `Single`, 15 или меньше для значения типа `Double` и 18 или меньше для значения типа `Extended`
`Format(s,[n1,n2, ..])`	Строка, являющаяся изображением значений `n1`, `n2` и т. д. Способ преобразования значений в строку символов определяют управляющие символы, которые находятся в строке форматирования `s`
`StrToInt(s)`	Целое, изображением которого является строка `s`
`StrToFloat(s)`	Вещественное, изображением которого является строка `s`
`Round(n)`	Целое, полученное путем округления `n` по известным правилам
`Trunc(n)`	Целое, полученное путем отбрасывания дробной части `n`
`Frac(n)`	Дробное, представляющее собой дробную часть вещественного `n`
`Int(n)`	Дробное, представляющее собой целую часть вещественного `n`

Манипулирования датами и временем

Большинству функций манипулирования датами в качестве параметра передается переменная типа `TDateTime`, в качестве которой, как правило, используется значение функции `Now`.

Таблица 2.32. *Функции манипулирования датами и временем*

Функция	Значение
`Now`	Системная дата и время — переменная типа `TDateTime`
`DateToStr(dt)`	Строка символов, изображающая дату в формате dd.mm.yyy
`TimeToStr(dt)`	Строка символов, изображающая время в формате hh:mm:ss
`DayOf(dt)`	День (номер дня в месяце), соответствующий дате, указанной в качестве параметра функции
`MonthOf(dt)`	Номер месяца, соответствующий дате, указанной в качестве параметра функции
`WeekOf(dt)`	Номер недели, соответствующий дате, указанной в качестве параметра функции
`YearOf(dt)`	Год, соответствующий указанной дате
`DayOfWeek(dt)`	Номер дня недели, соответствующий указанной дате: 1 — воскресенье, 2 — понедельник, 3 — вторник и т. д.
`StartOfWeek(w)`	Дата первого дня указанной недели
`HourOf(dt)`	Количество часов
`MinuteOf(dt)`	Количество минут
`SecondOf(dt)`	Количество секунд
`DecodeDate(dt,y,m,d)`	Возвращает год, месяц и день, представленные отдельными числами

Таблица 2.32 *(окончание)*

Функция	Значение
`DecodeTime(dt,h,m,s,ms)`	Возвращает время (часы, минуты, секунды и миллисекунды), представленное отдельными числами
`FormatDateTime(s,dt)`	Строка символов, представляющая собой дату или время. Способ представления задает строка формата s, например, строка `dd/mm/yyyy` задает, что значением функции является дата, а строка `hh:mm` — время

События

Таблица 2.33. *События*

Событие	Происходит
`OnClick`	При щелчке кнопкой мыши
`OnDblClick`	При двойном щелчке кнопкой мыши
`OnMouseDown`	При нажатии кнопки мыши
`OnMouseUp`	При отпускании кнопки мыши
`OnMouseMove`	При перемещении мыши
`OnKeyPress`	При нажатии клавиши клавиатуры
`OnKeyDown`	При нажатии клавиши клавиатуры. События `OnKeyDown` и `OnKeyPress` — это чередующиеся, повторяющиеся события, которые происходят до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие `OnKeyUp`)
`OnKeyUp`	При отпускании нажатой клавиши клавиатуры
`OnCreate`	При создании объекта (формы, элемента управления). Процедура обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий

Таблица 2.33 *(окончание)*

Событие	Происходит
`OnPaint`	При появлении окна на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном и в других случаях. Событие сообщает о необходимости обновить (перерисовать) окно
`OnEnter`	При получении элементом управления фокуса
`OnExit`	При потере элементом управления фокуса

Исключения

Во время работы программы могут возникать ошибки, которые называют *исключениями* (табл. 2.34).

Таблица 2.34. *Типичные исключения*

Тип исключения	Возникает
`EConvertError`	При выполнении преобразования, если преобразуемая величина не может быть приведена к требуемому виду. Наиболее часто возникает при преобразовании строки символов в число
`EZeroDivide`	При выполнении операции деления, если делитель равен нулю
`EFOpenError`	При обращении к файлу, например при попытке загрузить файл иллюстрации при помощи метода `LoadFromFile`. Наиболее частой причиной является отсутствие требуемого файла или, в случае использования сменного диска, отсутствие диска в накопителе
`EInOutError`	При обращении к файлу, например при попытке открыть для чтения (инструкция `reset`) несуществующий файл
`EDBEngineError`	При выполнении операций с базой данных, например при попытке выполнить SQL-запрос к несуществующей таблице

Приложения

Приложение 1. Как создать анимацию

Анимацию можно создать, например, при помощи программы Macromedia Flash.

В Macromedia Flash анимация, которую так же довольно часто называют роликом (Movie), состоит из слоев. *Слой* — это последовательность кадров (рис. П1.1), которые в процессе воспроизведения анимации выводятся последовательно, один за другим.

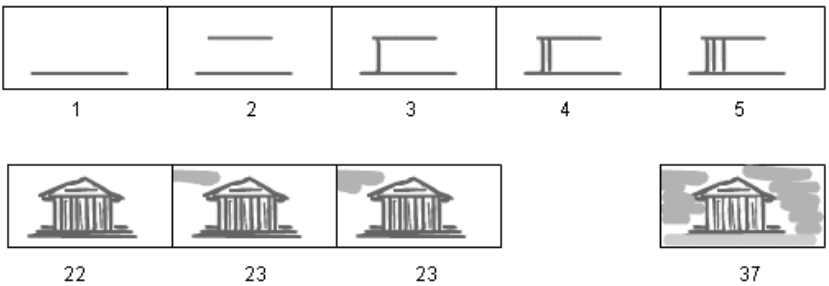


Рис. П1.1. Кадры анимации процесса рисования Дельфийского храма

Если ролик состоит из нескольких слоев, то кадры анимации получаются путем наложения кадров одного слоя на кадры другого. Например, один слой может содержать изображение фона, на котором разворачивается действие, а другой — изображение персонажей. Возможность формирования изображения путем наложения слоев существенно облегчает процесс создания анимации. В простейшем случае ролик представляет собой один-единственный слой. Таким образом, чтобы создать анимацию, нужно распределить изображение по слоям и для каждого слоя создать кадры.

После запуска Macromedia Flash на фоне главного окна программы появляется окно **Movie1** (рис. П1.2), используемое для создания анимации. В верхней части окна, которая называется Timeline, отражена структура анимации, в нижней части, которая именуется рабочей областью, находится изображение текущего кадра выбранного слоя. После запуска Macromedia Flash анимация состоит из одного слоя (`Layer 1`), который в свою очередь представляет один пустой (чистый) кадр.

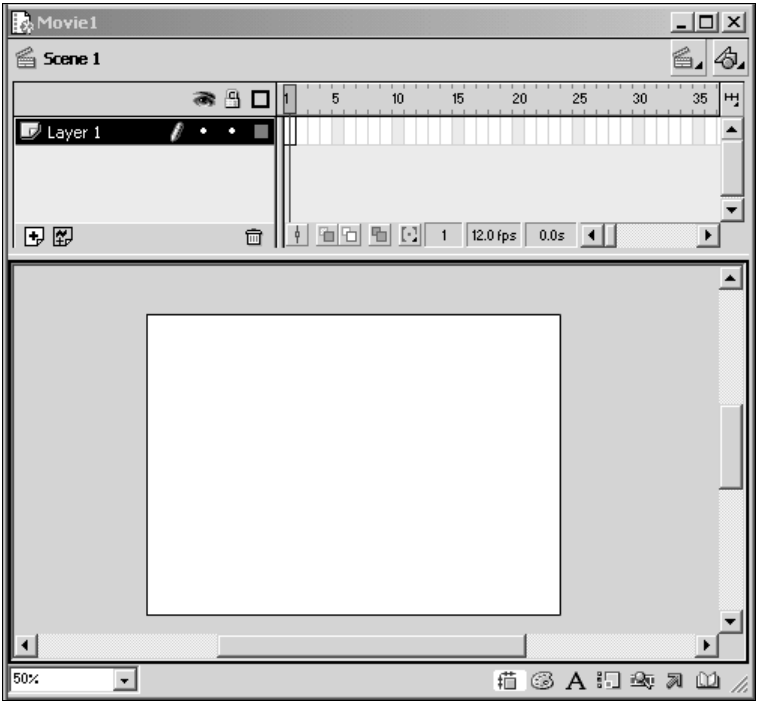


Рис. П1.2. Окно **Movie** в начале работы над новой анимацией

Перед тем как приступить непосредственно к созданию кадров анимации, нужно задать общие характеристики анимации (ролика): размер кадров и скорость их воспроизведения. Характеристики вводятся в поля диалогового окна **Movie Properties** (рис. П1.3), которое появляется в результате выбора из меню **Modify** команды **Movie**. В поле **Frame Rate** нужно ввести ско-

рость воспроизведения ролика, которая измеряется в кадрах в секунду (fps — frame per second, кадров в секунду), в поля **Width** и **Height** — ширину и высоту кадров. В этом же окне можно выбрать фон кадров (список **Background Color**).

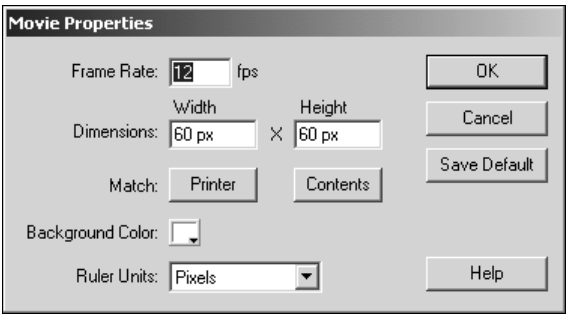


Рис. П1.3. Характеристики ролика отображаются в окне **Movie Properties**

После того как установлены характеристики ролика, можно приступить к созданию кадров анимации.

Первый кадр нужно просто нарисовать. Технология создания изображений Macromedia Flash обычная, используется стандартный набор инструментов: кисть, карандаш, пульверизатор, резинка и др.

Чтобы создать следующий кадр, необходимо из меню **Insert** выбрать команду **Keyframe**. В результате в текущий слой будет добавлен кадр, в который будет скопировано содержимое предыдущего кадра (т. к. в большинстве случаев следующий кадр создается путем изменения предыдущего). Теперь можно нарисовать второй кадр. Аналогичным образом создаются остальные кадры анимации.

Иногда не нужно, чтобы новый кадр содержал изображение предыдущего, в этом случае вместо команды **Keyframe** следует воспользоваться командой **Blank Keyframe**.

Если некоторое изображение должно оставаться статичным в течение времени, кратного выводу нескольких кадров, то вместо того, чтобы вставлять в слой несколько одинаковых кадров (команда **Keyframe**), нужно сделать кадр статичным. Если кадр, изображение которого должно быть статичным, является по-

следним кадром ролика, то в окне Timeline нужно выделить кадр, после которого изображение должно оставаться статичным, и из меню **Insert** выбрать команду **Frame**. Если кадр, изображение которого должно быть статичным, не является последним, то необходимо выделить этот кадр и несколько раз из меню **Insert** выбрать команду **Frame**.

Можно значительно облегчить работу по созданию анимации, если разделить изображение на основное и фоновое, поместив каждое в отдельный слой (именно так поступают при создании мультфильмов). Сначала нужно создать кадры слоя фона так, как было описано выше. Затем, выбрав из меню **Insert** команду **Layer**, необходимо добавить слой основного действия.

Следует обратить внимание, что все действия по редактированию изображения направлены на текущий кадр выбранного слоя. В списке слоев выбранный слой выделен цветом, номер текущего кадра помечен маркером — красным квадратиком.

Чтобы выводимая анимация сопровождалась звуком, нужно сначала сделать доступным соответствующий звуковой файл. Для этого надо из меню **File** выбрать команду **Import** и добавить в проект звуковой файл (рис. П1.4).

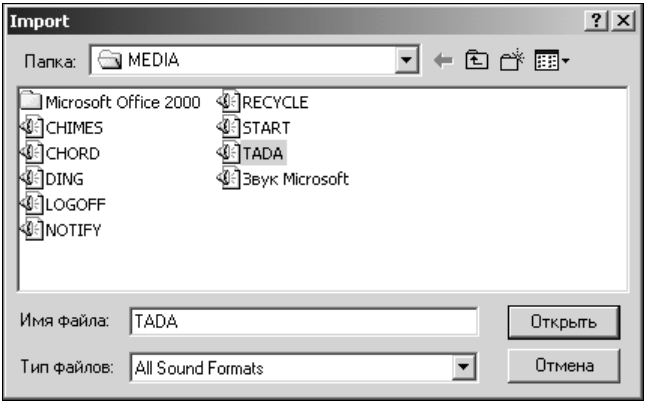


Рис. П1.4. Импорт звукового файла

Затем в окне Timeline необходимо выделить кадр, при отображении которого должно начаться воспроизведение звукового фрагмента, используя диалоговое окно **Sound** (рис. П1.5), вы-

брать звуковой фрагмент и задать, если нужно, параметры его воспроизведения. Количество повторов необходимо ввести в поле **Loops**, эффект, используемый при воспроизведении, можно выбрать из списка **Effect**.

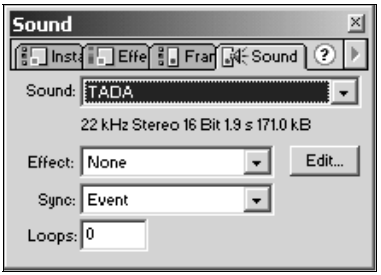


Рис. П1.5. Диалоговое окно **Sound**

В качестве примера на рис. П1.6 приведен вид окна Timeline в конце работы над анимацией. Анимация состоит из двух слоев. Слой `Layer 2` содержит фон. Детали фона появляются постепенно, в течение 9 кадров. После этого фон не меняется, поэтому 9-й кадр является статичным. Слой `Layer 1` — слой основного действия, которое начинается после того, как будет выведен фон. Вывод анимации заканчивается стандартным звуком TADA (его длительность равна одной секунде). Начало воспроизведения звука совпадает с выводом последнего (49-го, если считать от начала ролика) кадра основного действия, поэтому этот кадр сделан статичным в течение вывода следующих 12 кадров (скорость вывода анимации — 12 кадров в секунду). Сделано это для того, чтобы процесс вывода анимации завершился одновременно с окончанием звукового сигнала.

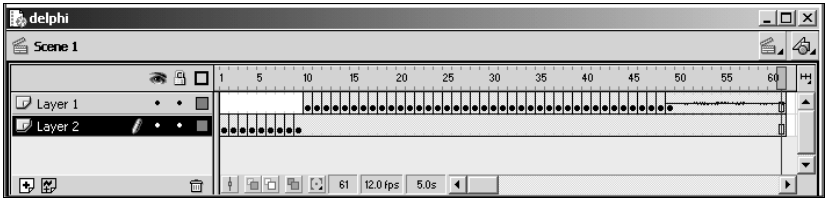


Рис. П1.6. Пример анимации

После того как ролик будет готов, его надо сохранить. Делается это обычным образом, т. е. выбором из меню **File** команды **Save**.

Для преобразования файла из формата Macromedia Flash в AVI-формат нужно из меню **File** выбрать команду **Export Movie** и задать имя файла. Затем в появившемся диалоговом окне **Export Windows AVI** (рис. П1.7) необходимо задать размер кадра (поля **Width** и **Height**), из списка **Video Format** выбрать формат, в котором будет записана видеочасть ролика, а из поля **Sound Format** — формат звука.

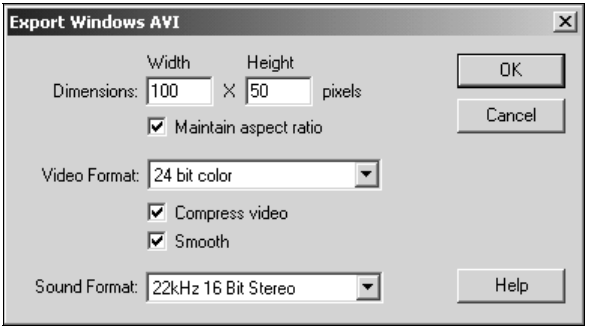


Рис. П1.7. Окно **Export Windows AVI**

Если установлен переключатель **Compress video**, то после щелчка на кнопке **OK** появится диалоговое окно, в котором можно будет выбрать один из стандартных методов сжатия видео. При выборе видео и звукового формата нужно учитывать, что чем более высокие требования будут предъявлены к качеству записи звука и изображения, тем больше места на диске займет AVI-файл. Здесь следует иметь в виду, что завышенные требования не всегда оправданы.

Приложение 2. Содержание компакт-диска

Прилагаемый к книге компакт-диск содержит исходные тексты программ, выполняемые файлы и необходимые для работы программ файлы данных. Каждая программа находится в отдельном каталоге.

Большинство программ не требуют для своей работы никаких дополнительных программных компонентов (библиотек) и могут быть запущены непосредственно с компакт-диска.

Некоторые программы, например программы работы с базами данных, требуют, чтобы на компьютере был установлен процессор баз данных Borland Database Engine (BDE). Кроме того, программа работы с базой данных может требовать, чтобы в системе был зарегистрирован псевдоним базы данных. Создать псевдоним можно при помощи утилиты SQL Explorer или BDE Administrator. Информацию о работе с утилитами SQL Explorer, BDE Administrator можно найти в справочной системе Delphi или, например, в книге: Культин Н. Б. Основы программирования в Delphi 7. — СПб.: БХВ-Петербург, 2003.

Для активной работы, чтобы иметь возможность вносить изменения в программы, скопируйте каталоги проектов на жесткий диск компьютера.

Предметный указатель

А

Б

В

Г

Д

З

И

К

Л

М

О

П

Р

С

Т

Ф

Ц

Ч

Э