Никита Культин

Delphi
в задачах и примерах

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681.3.068+800.92Delphi
ББК 32.973.26-018.1
К90

Культин Н. Б.

К90 Delphi в задачах и примерах. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2012. — 288 с.: ил.

ISBN 978-5-9775-0811-7

Книга представляет собой сборник примеров программ и задач для самостоятельного решения в среде программирования Delphi. Примеры различной степени сложности — от простейших до программ работы с графикой, звуком и базами данных — демонстрируют возможности среды разработки Delphi, назначение основных компонентов. Справочник содержит описание наиболее часто используемых компонентов и функций. В третьем издании обновлены старые и добавлены новые примеры. Проекты из книги размещены на сайте издательства.

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	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Марины Дамбиевой*
Зав. производством	*Николай Тверских*

Подписано в печать 29.02.12.
Формат 60×90 1/16. Печать офсетная. Усл. печ. л. 18.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

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

ISBN 978-5-9775-0811-7

Оглавление

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 Delphi, которая позволяет создавать различные программы: от простейших однооконных приложений до программ управления распределенными базами данных.

В качестве языка программирования в Delphi используется язык Delphi (Delphi language), являющийся прямым потомком языка

Pascal хорошо известного многим, в том числе начинающим программистам.

Чтобы научиться программировать, надо программировать — писать программы, решать практические, реальные 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, демонстрирующие назначение компонентов и особенности работы с ними.

В книге, которую вы держите в руках, собраны разнообразные примеры, которые не только демонстрируют назначение компонентов и 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.

- *Часть 1* 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.
- *Часть 2* — это краткий справочник по компонентам и функциям Delphi. В нем можно найти описание базовых компонентов, компонентов доступа к базам данных, описание часто используемых функций.

Научиться программировать можно только программируя, решая реальные 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)* или к справочной системе Delphi.

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

- 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 на форму *компонентов* и последующей их настройки.
- На 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 **Мили-километры**, которая пересчитывает расстояние из миль в километры (1 миля равна 1 км 609,34 м). Рекомендуемый вид формы приведен на рис. 1.1.

```
// щелчок на кнопке Пересчет
procedure TForm1.Button1Click(Sender: TObject);
var
  mile: real;  // расстояние в милях
  km: real;    // расстояние в километрах

begin
  // ввести исходные данные
  mile := StrToFloat(Edit1.Text);
```

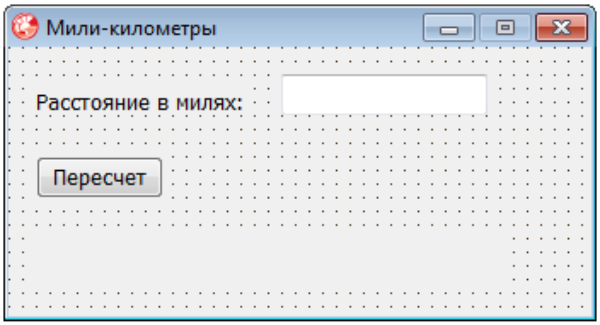


Рис. 1.1. Форма программы **Мили-километры**

```
  // пересчитать
  km := mile * 1.609344; // 1 миля — 1,609344 км

  // вывести результат
  Label2.Caption := FloatToStr(mile) + ' миль — это ' +
                    FloatToStr(km) + ' км.';
end;
```

2. Усовершенствуйте программу **Мили-километры** так, чтобы пользователь мог ввести в поле **Расстояние в милях** только число.

```
// нажатие клавиши в поле компонента Edit1
procedure TForm1.Edit1KeyPress(Sender: TObject;
                               var Key: Char);
begin
  // Процедура проверяет, является ли символ, соответствующий
  // нажатой клавише (Key), допустимым. Если символ неверный,
  // то он заменяется "нуль символом", который в поле
  // редактирования не отображается. В результате
  // у пользователя создается впечатление, что клавиатура
  // не реагирует на нажатие "неверных" клавиш.
  // В данном случае правильными являются
  // цифровые клавиши, запятая и <Backspace>.

  case Key of
    '0' ..'9', #8: ;  // цифра или <Backspace>
              ',':    // запятая
```

```
                if Pos(',',Edit1.Text) <> 0 // запятая уже
                                            // введена
                        then Key := #0;
    else  Key := #0; // остальные символы не отображать
  end;
end;

// щелчок на кнопке Пересчет
procedure TForm1.Button1Click(Sender: TObject);
var
  mile: real;  // расстояние в милях
  km: real;    // расстояние в километрах
begin
  // Если в поле Edit1 нет данных, то при выполнении
  // функции StrToFloat возникает исключение (ошибка).
  // Проверим, введены ли исходные данные.

  if Length(Edit1.Text) = 0 then
  begin
    ShowMessage('Надо ввести исходные данные');
    exit;
  end;

  // пользователь ввел расстояние в милях
  mile := StrToFloat(Edit1.Text);

  km := mile * 1.609344; // 1 миля — 1,609344 км

  Label2.Caption := FloatToStr(mile) + ' миль — это ' +
                    FloatToStrF(km,ffFixed,6,2) + ' км.';
end;
```

3. Написать программу **Конвертор**, которая пересчитывает цену из долларов в рубли. Рекомендуемый вид формы приведен на рис. 1.2. Программа должна быть спроектирована таким образом, чтобы пользователь мог ввести в поля редактирования только правильные данные (дробные числа). При нажатии клавиши <Enter> в поле **Курс** курсор должен переходить в поле **Цена**, а

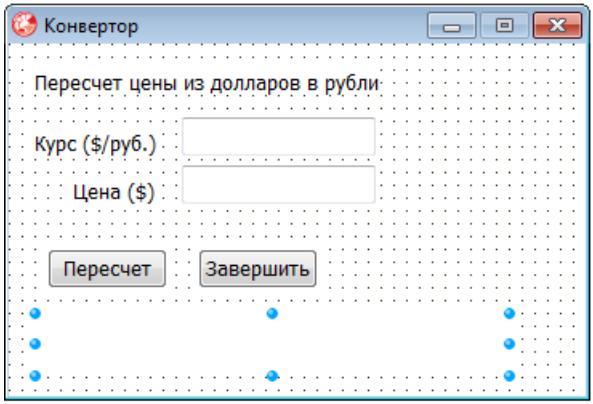


Рис. 1.2. Форма программы **Конвертор**

при нажатии этой же клавиши в поле **Цена** — на кнопку **Пересчет**.

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

        '.',',':
             // Обработку десятичного разделителя
             // сделаем "интеллектуальной". Заменим точку
             // и запятую на символ
             // FormatSettings.DecimalSeparator — символ,
             // который при текущей настройке операционной
             // системы является десятичным разделителем.
             begin
                Key := FormatSettings.DecimalSeparator;
                // Проверим, введен ли уже в поле
                // Edit десятичный разделитель
                if pos(FormatSettings.DecimalSeparator,
                                          Edit1.Text) <> 0
                   then Key := #0;
             end;
```

```
        #13: Edit2.SetFocus; // Нажата клавиша <Enter>
                             // Переместить курсор в поле Edit2
        // остальные символы запрещены
        else Key := #0;
     end;
end;

// нажатие клавиши в поле Цена
procedure TForm1.Edit2KeyPress(Sender: TObject;
                               var Key: Char);
begin
    case Key of
        '0'..'9', #8: ;  // цифры и <Backspace>
        '.',',':
            begin
               Key := FormatSettings.DecimalSeparator;
               // проверим, введен ли уже в поле Edit
               // десятичный разделитель
               if
pos(FormatSettings.DecimalSeparator,Edit1.Text) <> 0
                  then Key := #0;
            end;
        #13: Button1.SetFocus; // Сделать активной
                               // кнопку Пересчет

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

// щелчок на кнопке Пересчет
procedure TForm1.Button1Click(Sender: TObject);
var
  usd: real;   // цена в долларах
  k:   real;   // курс
  rub: real;   // цена в рублях

begin
    k := StrToFloat(Edit1.Text);
    usd := StrToFloat(Edit2.Text);
```

```
    // пересчитать цену из долларов в рубли
    rub := usd * k;

    // вывести результат расчета в поле Label4
    Label4.Caption := FloatToStr(usd) + '$ = ' +
                      FloatToStrF(rub, ffCurrency, 6,2);
end;
```

4. Усовершенствуйте программу **Конвертор** так, чтобы событие `KeyPress` обоих полей редактирования обрабатывала одна процедура, а также чтобы кнопка **Пересчет** становилась доступной только после ввода данных в оба поля редактирования.

```
{
  Процедура EditKeyPress обрабатывает нажатие клавиш
  в полях Курс и Цена. Сначала надо обычным образом
  создать процедуру обработки события KeyPress для поля
  Edit1, назвав ее EditKeyPress. Затем надо выбрать компонент
  Edit2 и указать процедуру EditKeyPress в качестве процедуры
  обработки события KeyePress.
  Чтобы узнать, на каком компоненте произошло
  событие, надо проверить значение свойства Sender.
}
procedure TForm1.EditKeyPress(Sender: TObject;
                              var Key: Char);
begin
    case Key of
        '0'..'9', #8: ;  // цифры и <Backspace>

        '.',',':
             // Обработку десятичного разделителя
             // сделаем "интеллектуальной". Заменим точку
             // и запятую на символ
             // FormatSettings.DecimalSeparator — символ,
             // который при текущей настройке операционной
             // системы должен использоваться при записи
             // дробных чисел.
             begin
                Key := FormatSettings.DecimalSeparator;
```

```
                // проверим, введен ли уже в поле Edit
                // десятичный разделитель
                if
pos(FormatSettings.DecimalSeparator,Edit1.Text) <> 0
                   then Key := #0;
            end;
        #13: // клавиша <Enter>
             // параметр Sender содержит имя компонента,
             // на котором произошло событие
             if Sender = Edit1 then
                Edit2.SetFocus // Переместить курсор
                               // в поле Edit2
            else Button1.SetFocus; // Установить фокус
                                   // на Button1

        else Key := #0; // остальные символы запрещены
     end;
end;

// EditChange - текст, находящийся в поле редактирования,
// изменился. Процедура EditChange обрабатывает изменение
// текста в полях Курс и Цена
procedure TForm1.EditChange(Sender: TObject);
begin
    // проверим, есть ли данные в полях редактирования
    if (Length(Edit1.Text) = 0) or (Length(Edit2.Text) = 0)
       then Button1.Enabled := False // кнопка Пересчет
                                     // недоступна
       else Button1.Enabled := True; // кнопка Пересчет
                                     // доступна
end;

// щелчок на кнопке Пересчет
procedure TForm1.Button1Click(Sender: TObject);
var
  usd: real;   // цена в долларах
  k:   real;   // курс
  rub: real;   // цена в рублях
```

```
begin
    k := StrToFloat(Edit1.Text);
    usd := StrToFloat(Edit2.Text);

    // пересчитать цену из долларов в рубли
    rub := usd * k;

    // вывести результат расчета в поле Label4
    Label4.Caption := FloatToStr(usd) + '$ = ' +
                      FloatToStrF(rub, ffCurrency, 6,2);
end;
```

5. Написать программу, которая пересчитывает вес из фунтов в килограммы (1 фунт = 0,4536 кг). Рекомендуемый вид формы приведен на рис. 1.3. Программа должна быть спроектирована таким образом, чтобы пользователь мог ввести в поле **Вес в фунтах** только положительное число (целое или дробное).

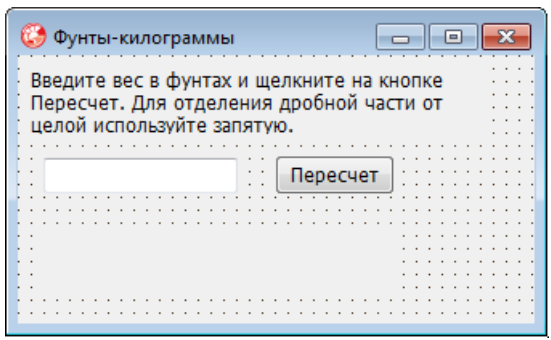


Рис. 1.3. Форма программы **Фунты-килограммы**

6. Написать программу, вычисляющую скорость (км/час), с которой бегун пробежал дистанцию. Рекомендуемый вид формы приведен на рис. 1.4. Программа должна быть спроектирована таким образом, чтобы в поля **Дистанция** и **Минут** можно было ввести только цело число, а в поле **Секунд** — дробное.

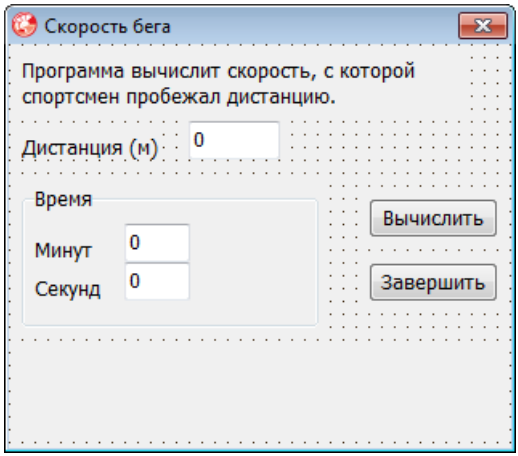


Рис. 1.4. Форма программы **Скорость**

```
// 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':                ; // цифра
      #8      :                ; // <Backspace>
      #13     : Edit2.SetFocus; // <Enter> — курсор
                                // в поле Минут

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

// нажатие клавиши в поле Минут
procedure TForm1.Edit2KeyPress(Sender: TObject;
                               var Key: Char);
```

```
begin
    case Key of
      '0'..'9':               ;
      #8      :               ; // <Backspace>
      #13     : Edit3.SetFocus; // <Enter> — курсор
                                // в поле Секунд

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

// нажатие клавиши в поле Секунд
procedure TForm1.Edit3KeyPress(Sender: TObject;
                               var Key: Char);
begin
   case Key of
     '0'..'9': ;
     ',','.' : // десятичный разделитель
               begin
                  Key := FormatSettings.DecimalSeparator;
                  if
Pos(FormatSettings.DecimalSeparator,Edit3.Text) <> 0
                     then Key := Char(0);
                end;
     #8:  ; // <Backspace>
     #13: Button1.SetFocus; // установить фокус
                            // на кнопку 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
    // Если поле редактирования не содержит данных,
    // то при выполнении преобразования строки в число
    // (функция StrToInt или StrToFloat)
    // возникает исключение EconvertError.
    // Чтобы предотвратить эту ситуацию, проверим,
    // есть ли данные в полях редактирования
    // и, если их там нет, запишем нулевое значение.

    if Length(Edit1.Text) = 0
        then Edit1.Text := '0';

    if Length(Edit2.Text) = 0
        then Edit2.Text := '0';

    if Length(Edit3.Text) = 0
        then Edit3.Text := '0';

    // 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
        MessageDlg('Надо задать дистанцию и время',
                   mtWarning,[mbOk],0);
        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, ffFixed,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. 1.5. Программа должна быть спроектирована таким образом, чтобы кнопка **Вычислить** была доступна только в том случае, если пользователь ввел величину напряжения и сопротивления.

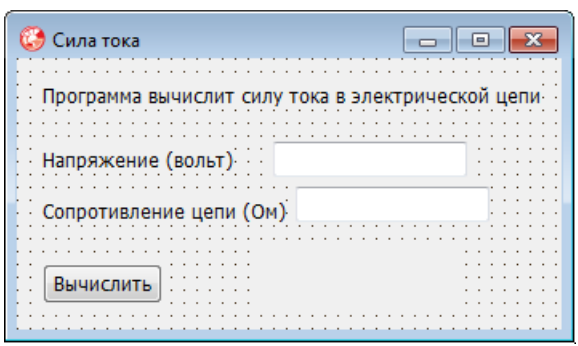


Рис. 1.5. Форма программы **Сила тока**

8. Написать программу, которая вычисляет сопротивление электрической цепи, состоящей из двух параллельно соединенных резисторов. Рекомендуемый вид формы приведен на рис. 1.6.

9. Написать программу, которая вычисляет доход по вкладу методом простых процентов (Доход = Сумма * (Процент / 12) * Срок). Рекомендуемый вид формы программы приведен на рис. 1.7. В результате щелчка на кнопке **Вычислить** в окне про-

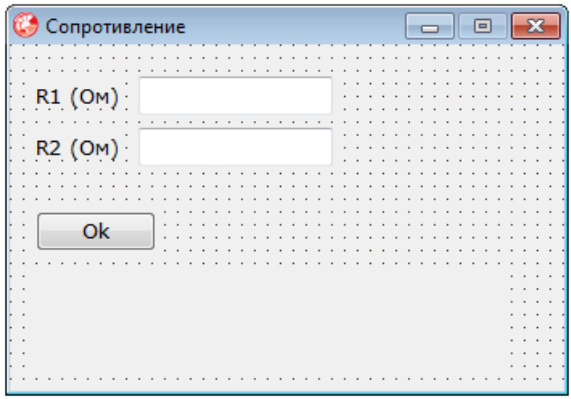


Рис. 1.6. Форма программы **Сопротивление**

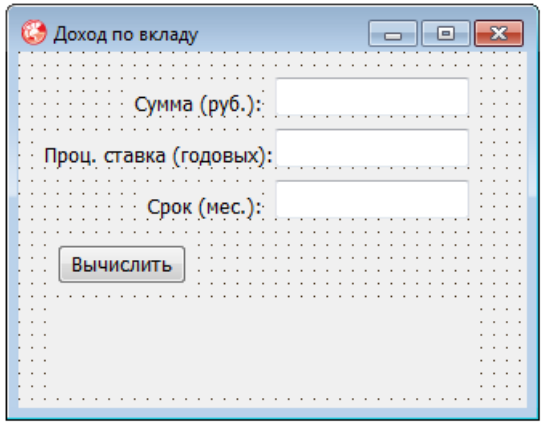


Рис. 1.7. Форма программы **Доход по вкладу**

граммы должна отображаться величина дохода и сумма в конце срока вклада. Программа должна быть спроектирована таким образом, чтобы в поля **Сумма** и **Проц. ставка** можно было ввести дробные числа, а в поле **Срок** — только целое.

10. Написать программу, которая вычисляет сопротивление электрической цепи, состоящей из двух сопротивлений. Сопротивления могут быть соединены последовательно или параллельно. Рекомендуемый вид формы приведен на рис. 1.8. Если величина

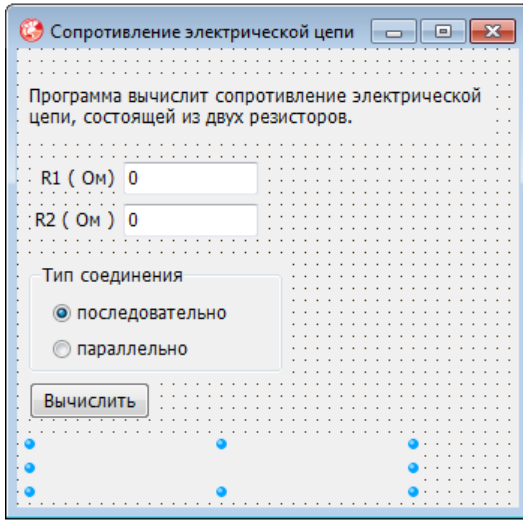


Рис. 1.8. Форма программы **Сопротивление электрической цепи**

сопротивления цепи превышает 1 000 Ом, то результат должен быть выведен в килоомах.

```
// 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 них
```

```
    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,ffFixed,3,2) + ' Ом'
    else
        begin
            r:=r/1000;
            Label4.Caption := Form1.Label4.Caption +
                   FloatToStrF(r,ffFixed,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;
```

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 текущей (накопленной) сумме вклада и в следующем месяце проценты начисляются на новую сумму. Рекомендуемый вид формы программы приведен на рис. 1.9.

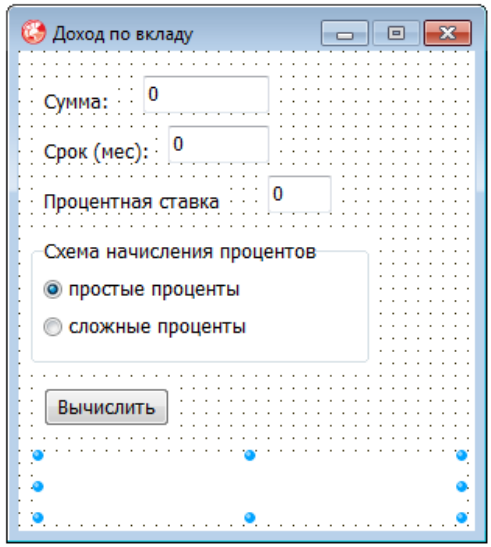


Рис. 1.9. Форма программы **Доход**

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

    sum2: real;       // сумма, при вычислении методом
                      // сложных процентов
    i: integer;
begin
    // получить исходные данные
    sum := StrToFloat(Edit1.Text);
    pr := StrToFloat(Edit2.Text);
    period := StrToInt(Edit3.Text);

    if RadioButton1.Checked then
        // выбран переключатель Простые проценты
        profit := sum * (pr/100/12) * period
```

```
    else
        // т. к. в группе два переключателя, то если
        // не выбран RadioButton1, то выбран
        // RadioButton2 — Сложные проценты
        begin
            sum2:= sum;
            for i:=1 to period do
                sum2 := sum2 + sum2 * (pr/100/12);
            // здесь sum2 — сумма в конце срока вклада
            profit := sum2 — sum;
        end;

    sum := sum + profit;
    Label4.Caption := 'Доход: ' +
                      FloatToStrF(profit,ffCurrency,6,2) +
                      #13 +
                      'Сумма в конце срока вклада: ' +
                      FloatToStrF(sum,ffCurrency,6,2);
end;
```

12. Написать программу, которая вычисляет стоимость жалюзи. Рекомендуемый вид формы приведен на рис. 1.10.

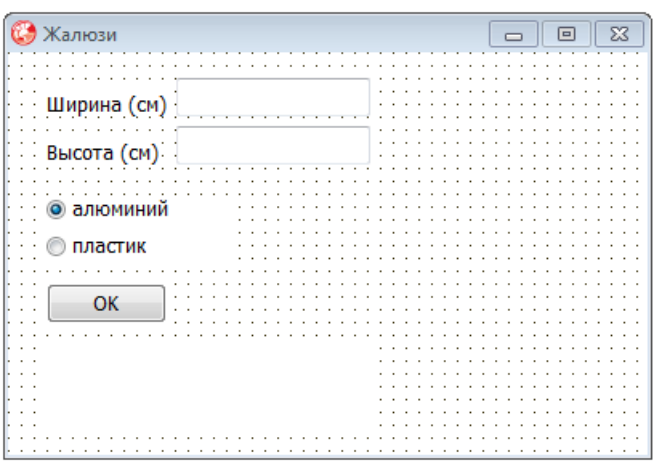


Рис. 1.10. Форма программы **Жалюзи**

13. Написать программу, которая позволяет пересчитать цену из долларов в рубли или из рублей в доллары. Рекомендуемый вид формы приведен на рис. 1.11. Во время работы программы, в результате выбора вида конвертации, соответствующим образом должен меняться заголовок окна и текст, поясняющий назначение полей ввода.

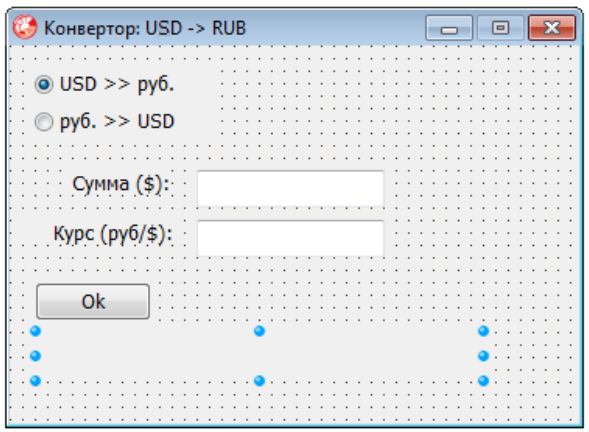


Рис. 1.11. Форма программы **Конвертор**

```
  // щелчок на переключателе  USD >> руб.
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
  // изменить заголовок окна
  Form1.Caption := 'Конвертор: USD -> RUB';

  // изменить текст перед полем Edit1
  Label1.Caption := 'Сумма ($): ';

  // установить курсор в поле Сумма
  Edit1.SetFocus;

  Label3.Caption := '';
end;

// щелчок на переключателе  руб. >> USD
procedure TForm1.RadioButton2Click(Sender: TObject);
```

```
begin
  Form1.Caption := 'Конвертор: RUB -> USD';
  Label1.Caption := 'Сумма (руб.): ';
  Edit1.SetFocus;
  Label3.Caption := '';
end;

// Щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
var
  usd: real; // цена в долларах
  rub: real; // цена в рублях
  k: real;   // курс

begin

  k := StrToFloat(Edit2.Text);

  if RadioButton1.Checked then
    begin
      // пересчет из долларов в рубли
      usd := StrToFloat(Edit1.Text);
      rub := usd * k;
      Label3.Caption := FloatToStrF(usd,ffFixed,6,2) +
                        '$ = ' +
                        FloatToStrF(rub,ffCurrency,6,2);
    end
  else begin
      // пересчет из рублей в доллары
      rub := StrToFloat(Edit1.Text);
      usd := rub / k;
      Label3.Caption := FloatToStrF(rub,ffCurrency,6,2) +
                        ' = ' +
                        FloatToStrF(usd,ffFixed,6,2) + '$';

  end;
end;

// процедура обрабатывает событие EditChange
//  компонентов Edit1 и Edit2
```

```
procedure TForm1.EditChange(Sender: TObject);
begin
  // если в каком-либо из полей Edit нет данных,
  // сделать кнопку Button1 недоступной
  if (Length (Edit1.Text) = 0) or  (Length (Edit2.Text) = 0)
     then Button1.Enabled := False
     else Button1.Enabled := True;

  Label3.Caption := '';
end;

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

       '.',',':
            // Обработку десятичного разделителя
            // сделаем "интеллектуальной". Заменим точку
            // и запятую на символ
            // FormatSettings.DecimalSeparator — символ,
            // который при текущей настройке операционной
            // системы должен использоваться при записи
            // дробных чисел.
            begin
               Key := FormatSettings.DecimalSeparator;
               // проверим, введен ли уже в поле Edit
               // десятичный разделитель
               if
pos(FormatSettings.DecimalSeparator,Edit1.Text) <> 0
                  then Key := #0;
            end;
        #13: // клавиша <Enter>
             Edit2.SetFocus    // Переместить курсор
                               // в поле Курс

        else Key := #0; // остальные символы запрещены
    end;
end;
```

```
// нажатие клавиши в коле Курс
procedure TForm1.Edit2KeyPress(Sender: TObject;
                               var Key: Char);
begin
  case Key of
       '0'..'9', #8: ;  // цифры и <Backspace>

       '.',',':
          begin
              Key := FormatSettings.DecimalSeparator;
              if
pos(FormatSettings.DecimalSeparator,Edit1.Text) <> 0
                 then Key := #0;
           end;
        #13: // клавиша <Enter>
                Edit2.SetFocus  // Переместить фокус
                                // на кнопку Ok

        else Key := #0; // остальные символы запрещены
    end;
end;
```

14. Написать программу, которая вычисляет доход по вкладу сроком на 1, 2, 3, 6 месяцев или на один год (предполагается, что процентная ставка зависит от срока вклада). Рекомендуемый вид формы приведен на рис. 1.12.

```
// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
var
  sum: real;         // суммма вклада
  period: integer;   // срок вклада (месяцев)
  percent: real;     // процент (годовых)
  profit: real;      // доход
  sum2: real;        // сумма в конце срока вклада

begin
  sum := StrToFloat(Edit1.Text);
  case RadioGroup1.ItemIndex of
```

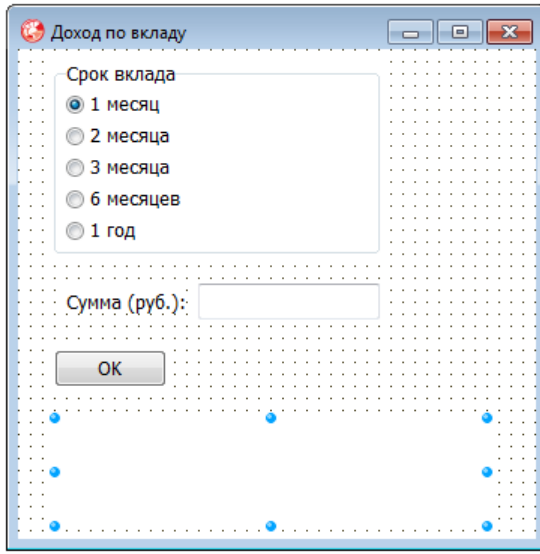


Рис. 1.12. Форма программы **Доход по вкладу**

```
  0: begin
       period := 1;
       percent := 8;
     end;
  1: begin
       period := 2;
       percent := 8.5;
     end;
  2: begin
       period := 3;
       percent := 9;
     end;
  3: begin
       period := 6;
       percent := 10 ;
     end;
  4: begin
       period := 12;
       percent := 11;
     end;
end;
```

```
   profit := sum * percent/100/12 * period;
   sum2 := sum + profit;

   Label2.Caption := 'Сумма вклада: ' +
              FloatToStrF(sum, ffCurrency,6,2) + #13 +
             'Срок вклада: ' + IntToStr(period) + 'мес.' +
              #13 + 'Процентная ставка: ' +
              FloatToStrF(percent, ffFixed,6,2) + '%' + #13 +
             'Доход: ' + FloatToStrF(profit, ffCurrency,6,2);
end;

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

       '.',',':
            // Обработку десятичного разделителя
            // сделаем "интеллектуальной". Замените точку
            // и запятую на символ
            // FormatSettings.DecimalSeparator — символ,
            // который при текущей настройке операционной
            // системы должен использоваться при записи
            // дробных чисел.
            begin
               Key := FormatSettings.DecimalSeparator;
               // проверим, введен ли уже в поле Edit
               // десятичный разделитель
               if
pos(FormatSettings.DecimalSeparator,Edit1.Text) <> 0
                  then Key := #0;
            end;
        #13: // клавиша <Enter>
                Button1.SetFocus; // Установить фокус
                                  // на Button1

       else Key := #0; // остальные символы запрещены
    end;
end;
```

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  if Length(Edit1.Text) = 0
     then Button1.Enabled := False
     else Button1.Enabled := True
end;
```

15. Написать программу, которая вычисляет стоимость автомобиля с учетом дополнительных опций. Рекомендуемый вид формы приведен на рис. 1.13.

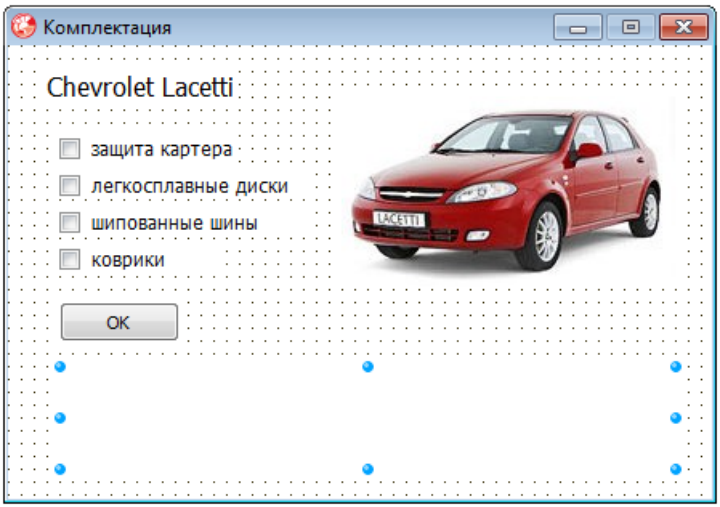


Рис. 1.13. Форма программы **Авто**

```
procedure TForm1.Button1Click(Sender: TObject);
var
    cena: real;      // цена в базовой комплектации
    dop: real;       // сумма за доп. оборудование
    discount: real; // скидка
    total: real;     // общая сумма

    st: string;      // сообщение — результат расчета
begin
    cena := 415000;
    dop := 0;
```

```
  if (CheckBox1.Checked)
  then
              // защита картера
    dop := dop + 4500;

  if (CheckBox2.Checked)
  then
              // зимние шины
    dop := dop +  12000;

  if (CheckBox3.Checked)
  then
              // литые диски
    dop := dop + 12000;

  if (CheckBox4.Checked)
  then
              // коврики
     dop := dop + 1200;

     total := cena + dop;

st := 'Цена в выбранной комплектации: ' +
                FloatToStrF(total, ffCurrency, 6,2);
     if ( dop <> 0) then

     st := st + #10+ 'В том числе доп. оборудование: '
                      + FloatToStrF(dop, ffCurrency, 6,2);

     if ((CheckBox1.Checked) and (CheckBox2.Checked) and
      (CheckBox3.Checked) and (CheckBox4.Checked))
then
  begin
   // скидка предоставляется, если
   // выбраны все опции
   discount := dop * 0.1;
   total := total — discount;
   st := st + #10 + 'Скидка на доп. оборудование (10%): ' +
             FloatToStrF(discount, ffCurrency, 6,2) +
```

```
                #10 + 'Итого: ' +
                FloatToStrF(total, ffCurrency, 6,2);
    end;

        Label2.Caption := st;

end;

// щелчок на компоненте CheckBox.
// Функция обрабатывает событие Click
// на компонентах checkBox1 — checkBox4
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  Label2.Caption := '';
end;
```

16. Написать программу, вычисляющую стоимость жалюзи в зависимости от размера и материала (пластик, алюминий, текстиль, бамбук, соломка), из которого они изготовлены. Рекомендуемый вид формы приведен на рис. 1.14.

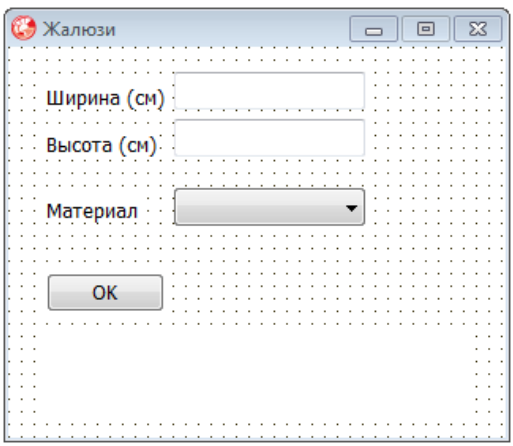


Рис. 1.14. Форма программы **Жалюзи**

```
// щелчок на кнопке OK
procedure TForm1.Button1Click(Sender: TObject);
var
  w,h: real;    // ширина, высота (см)
```

```
  s: real;      // площадь (кв.м.)
  c: real;      // цена за 1 кв.м.
  summ: real;   // сумма
  st: string;   // сообщение
begin

  if   ( Length(Edit1.Text) = 0 )  // не задана ширина
    or ( Length(Edit1.Text) = 0 )  // не задана высота
    or ( ComboBox1.ItemIndex = -1) // не выбран материал
  then begin
    ShowMessage('Надо задать ширину, высоту и выбрать
материал');
    exit;
  end;

  // вычислить площадь
  w := StrToFloat(Edit1.Text);
  h := StrToFloat(Edit2.Text);
  s := w * h / 10000;

  // определить цену
  // ItemIndex — номер элемента, выбранного в списке
  case ComboBox1.ItemIndex of
    0: c := 700; // алюминий
    1: c := 450; // пластик
    2: c := 300; // текстиль
  end;

  // расчет
  summ := s * c;

  st := 'Размер: ' + Edit1.Text + 'x' + Edit2.Text + ' см' +
        #10 + 'Материал: ' + ComboBox1.Text + #10 +
        'Сумма: ' + FloatToStrF(summ, ffCurrency, 6,2);

  Label4.Caption := st;

end;
```

17. Напишите программу, которая позволяет рассчитать тариф ОСАГО (обязательное страхование гражданской ответственности владельца транспортного средства). Рекомендуемый вид формы приведен на рис. 1.15.

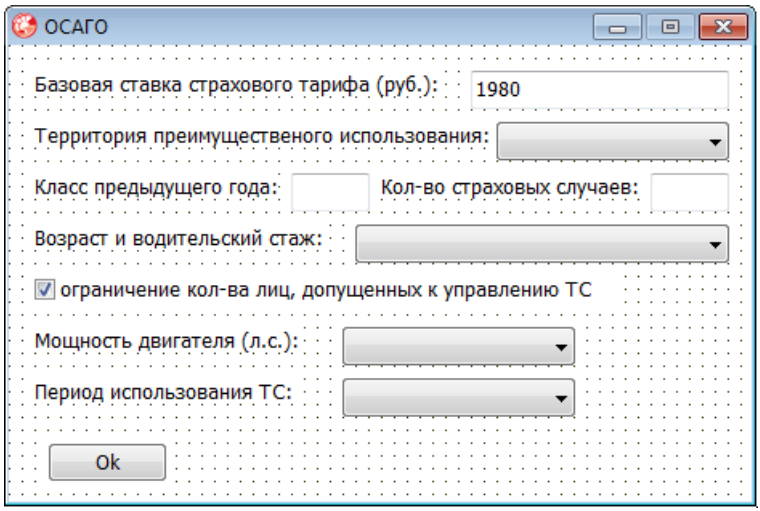


Рис. 1.15. Форма программы **ОСАГО**

```
var
  // коэффициент региона
  kt  : array[0..7] of double =
    (1, 1.8, 1.6, 1, 1.3, 1, 1, 1.8);

  // таблица определения коэффициента страхового тарифа.
  // 1-й год — 3-й класс, 2-й год (если не было страховых
  // случаев) — 4-й класс и т. д. Если страховой случай был,
  // то класс — ячейка таблицы на пересечении строки
  // предыдущего года и столбца, соответствующего
  // кол-ву страховых случаев.

  // класс безаварийности
  cb  : array[0..5, 0..4] of integer =
   ((1, -1, -1, -1, -1),
    (2, -1, -1, -1, -1),
```

```
    (3,  1, -1, -1, -1),
    (4,  1, -1, -1, -1),
    (5,  2,  1, -1, -1),
    (6,  3,  1, -1, -1));

  // коэффициент безаварийности для классов -1, 1 — 6
  kb : array[0..6] of double =
    (2.45, 2.3, 1.55, 1.4, 1, 0.95, 0.9);

  // коэффициент водительского стажа
  kvs : array[0..3] of double =
    (1.3, 1.2, 1.15, 1);

  // коэффициент мощности двигателя
  km : array[0..6] of double =
    (0.5, 0.7, 1, 1.3, 1.5, 1.7, 1.9);

  // коэффициент периода использования ТС
  ks : array[0..4] of double =
    (0.7, 0.8, 0.9, 0.95, 1);

procedure TForm1.FormCreate(Sender: TObject);
begin
  // настройка списка Территория использования
  ComboBox1.Style := csDropDownList;
  ComboBox1.Items.Add('Ленинградская обл.');
  ComboBox1.Items.Add('Москва');
  ComboBox1.Items.Add('Московская обл.');
  ComboBox1.Items.Add('Мурманск');
  ComboBox1.Items.Add('Нижний Новгород');
  ComboBox1.Items.Add('Ростов-на-Дону');
  ComboBox1.Items.Add('Самара');
  ComboBox1.Items.Add('Санкт-Петербург');

  // настройка списка Возраст и стаж
  ComboBox2.Style := csDropDownList;
  ComboBox2.Items.Add('до 22 лет, стаж менее 2 лет');
  ComboBox2.Items.Add('до 22 лет, стаж свыше 2 лет');
  ComboBox2.Items.Add('от 22 лет, стаж менее 2 лет');
  ComboBox2.Items.Add('от 22 лет, стаж свыше 2 лет');
```

```
  // настройка списка Мощность двигателя
  ComboBox3.Style := csDropDownList;
  ComboBox3.Items.Add('до 50 включительно');
  ComboBox3.Items.Add('свыше 50 до 70 включительно');
  ComboBox3.Items.Add('свыше 70 до 95 включительно');
  ComboBox3.Items.Add('свыше 95 до 120 включительно');
  ComboBox3.Items.Add('свыше 120 до 160 включительно');
  ComboBox3.Items.Add('свыше 160 до 200 включительно');
  ComboBox3.Items.Add('свыше 200');

  // настройка списка Период использования ТС
  ComboBox4.Style := csDropDownList;
  ComboBox4.Items.Add('6 месяцев');
  ComboBox4.Items.Add('7 месяцев');
  ComboBox4.Items.Add('8 месяцев');
  ComboBox4.Items.Add('9 месяцев');
  ComboBox4.Items.Add('более 9 месяцев');

end;

procedure TForm1.Button1Click(Sender: TObject);
var
  atb  : double;  // базовая ставка тарифа
  akt  : double;  // коэф. тарифа
  akb  : double;  // коэф. безаварийности
  akvs : double;  // коэф. водительского стажа
  ako  : double;  // коэф., учитывающий количество лиц,
                  // допущенных к управлению ТС
  akm  : double;  // коэф. мощности двигателя
  aks  : double;  // коэф., учитывающий период
                  // использования ТС

  pcb : integer;  // предыдущий класс безаварийности
  ccb : integer;  // текущий класс безаварийности
  nss : integer;  // количество страховых случаев
                  // предыдущего страхового периода

  t   : double;   // тариф
begin
```

```
try
  // базовая ставка тарифа
  atb := StrToFloat(Edit1.Text);

  // коэф. тарифа (зависит от выбранного региона)
  akt := kt[ComboBox1.ItemIndex];

  // текущий класс безаварийности
  pcb := StrToInt(Edit2.Text);
  nss := StrToInt(Edit3.Text);
  ccb := cb[pcb,nss];

  // коэф. безаварийности
  if (ccb <> -1) then
    akb := kb[ccb]
  else
    akb := kb[0];

  // коэф. водительского стажа
  akvs := kvs[ComboBox2.ItemIndex];

  // коэф., учитывающий количество лиц,
  // допущенных к управлению ТС
  if CheckBox1.Checked then ako := 1
  else ako := 1.5;

  // коэф. мощности двигателя
  akm := km[ComboBox3.ItemIndex];

  // коэф. периода использования ТС
  aks := ks[ComboBox4.ItemIndex];

except
   on e: EConvertError do begin
     ShowMessage('Ошибка исходных данных.' +  chr(10) +
      'Проверьте, все ли поля формы заполнены.');
    exit;
  end;
end;
```

```
  // вычисляем тариф
  t := atb * akt * akb * akvs * ako * akm * aks;

  ShowMessage(
    'Базовая ставка тарифа: ' +
     FloatToStrF(atb,ffCurrency,6,2) + chr(10) +
    'Коэф. тарифа: ' +
     FloatToStrF(akt,ffFixed,2,2) + chr(10) +
    'Коэф. безаварийности: ' +
     FloatToStr(akb) + chr(10) +
    'Коэф. водительского стажа: ' +
     FloatToStr(akvs) + chr(10) +
    'Коэф. кол-ва лиц, допущенных к управлению: ' +
     FloatToStr(ako) + chr(10) +
    'Коэф. мощности двигателя: ' +
     FloatToStr(akm) + chr(10) +
    'Коэф. периода использования ТС: ' +
     FloatToStr(aks) + chr(10) + chr(10) +
    'Тариф: ' + FloatToStrF(t, ffCurrency, 4,2));
end;
```

18. Напишите программу-калькулятор, выполняющую сложение и вычитание. Рекомендуемый вид формы приведен на рис. 1.16.

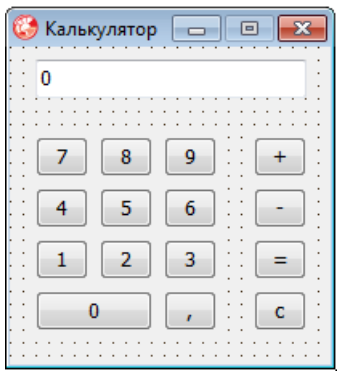


Рис. 1.16. Форма программы **Калькулятор**

К этой задаче даны два варианта решения. В первом варианте для каждой цифровой кнопки создана отдельная процедура обработ-

ки события `Click`. Во втором варианте событие `Click` всех цифровых кнопок обрабатывает одна процедура, что позволило значительно сократить размер программы.

```
// ************************************************
// 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.
```

19. Напишите программу **Время**, в окне которой отображается текущее время (рис. 1.17).

```
procedure TForm1.FormCreate(Sender: TObject);
var
  Time: TDateTime;  // текущее время
begin
  Time := Now();    // получить системное время
  Label1.Caption := FormatDateTime('hh:mm:ss',Time);

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

end;

// обработка сигнала таймера
procedure TForm1.Timer1Timer(Sender: TObject);
var
  Time: TDateTime;  // текущее время
begin
  Time := Now();    // получить системное время
  Label1.Caption := FormatDateTime('hh:mm:ss',Time);
end;
```

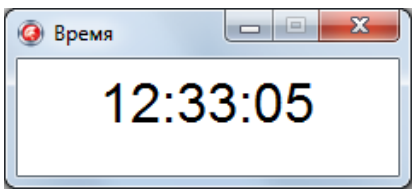


Рис. 1.17. Окно программы **Время**

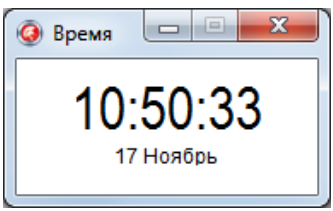


Рис. 1.18. В окне программы **Время** отображается текущие время и дата

20. Усовершенствуйте программу **Время** так, чтобы в ее окне отображалось не только время, но и дата (рис. 1.18).

```
procedure TForm1.FormCreate(Sender: TObject);
var
  Time: TDateTime;  // текущее время
begin
  Time := Now();   // получить системное время

  // отобразить время в поле Label1
  Label1.Caption := FormatDateTime('hh:mm:ss',Time);

  // отобразить дату в поле Label2
  Label2.Caption := FormatDateTime('dd.mm.yyyy',Time);

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

// обработка сигнала таймера
procedure TForm1.Timer1Timer(Sender: TObject);
var
  Time: TDateTime;  // текущее время
begin
  Time := Now();   // получить системное время
  Label1.Caption := FormatDateTime('hh:mmmm:ss',Time);
end;
```

21. Измените программу **Время** так, чтобы в ее окне секунды не отображались. Вместе с тем, чтобы пользователь видел, что часы идут, двоеточие, разделяющее часы и минуты, должно мигать. В окне программы также должна отображаться дата и день недели (рис. 1.19).

```
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');

var
  // время на индикаторе
  hh: integer; // часы
  mm: integer; // минуты

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

  Present:= Now(); // получить текущую дату

  hh := HourOf(Time);
  label1.Caption := IntToStr(hh);

  mm := MinuteOf(Time);
  label3.Caption := IntToStr(mm);

  DecodeDate(Present, Year, Month, Day);
  Dw := DayOfWeek(Present);

  // отобразить дату и день недели
  Label4.Caption := IntToStr(Day)+ ' ' +
              stMonth[Month] + ' '+ IntToStr(Year)+
              ' года, ' + stDay[Dw];
```

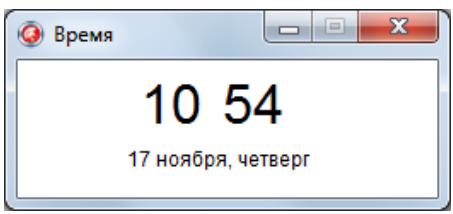


Рис. 1.19. В окне программы **Время** отображается текущие время, дата и день недели

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

// обработка сигнала таймера
procedure TForm1.Timer1Timer(Sender: TObject);
var
  Time: TDateTime;  // текущее время
begin
  Time := Now();   // получить системное время

  if hh <> HourOf(Time) then
  begin
    hh := HourOf(Time);
    label1.Caption := IntToStr(hh);
  end;

  if mm <> MinuteOf(Time) then
  begin
    mm := MinuteOf(Time);
    label3.Caption := IntToStr(mm);
  end;

  // скрыть/показать двоеточие
  Label2.Visible := not Label2.Visible;

end;
```

22. Напишите программу **Таймер**. Форма программы приведена на рис. 1.20, а на рис. 1.21 — ее окно во время установки интервала и в процессе отсчета времени.

```
implementation

{$R *.dfm}

{ Во время создания формы свойству Visible компонента Label3,
  в котором во время отсчета отображается оставшееся время,
  надо присвоить значение False }
```

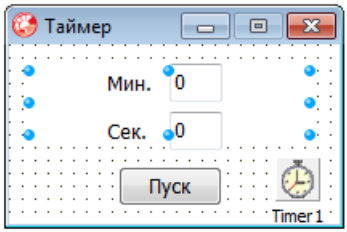
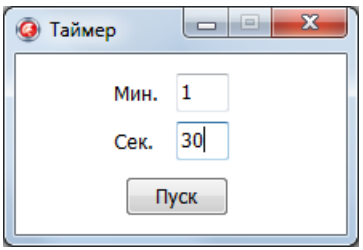
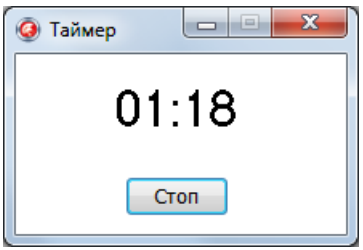


Рис. 1.20. Форма программы **Таймер**



а



б

Рис. 1.21. Окно программы **Таймер** во время установки интервала (*а*) и в процессе отсчета времени (*б*)

```
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.
```

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

```
implementation

{$R *.dfm}

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

// Конструктор формы.
// Создает компонент MediaPlayer и загружает в него 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;

// выводит в поле Label3 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 WAW-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 индикатор
      // 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 индикатор
      // 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.
```

24. Напишите программу **Таймер**. Для ввода интервала используйте компоненты UpDown. Рекомендуемый вид формы приведен на рис. 1.22.

```
{ 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
```

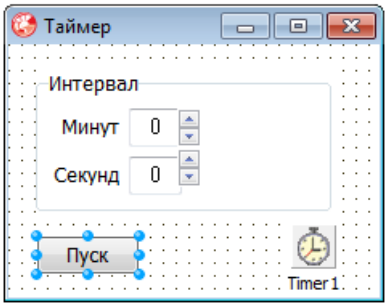


Рис. 1.22. Форма программы **Таймер**

```
// выводит в заголовок окна программа
// 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

- Для вывода графики следует использовать компонент `PaintBox`.
- Графика выводится на графическую поверхность компонента `PaintBox`, которая представляет собой объект `TCanvas`. Доступ к крафической поверхности (объекту типа `TCanvas`) обеспечивает свойство `Canvas`.
- Для того чтобы на графической поверхности объекта появилась 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 процедура обработки события `Paint`.

25. Напишите программу, которая на поверхности формы рисует флаг Российской Федерации. Флаг и подпись должны быть размещены в центре окна по горизонтали (рис. 1.23).

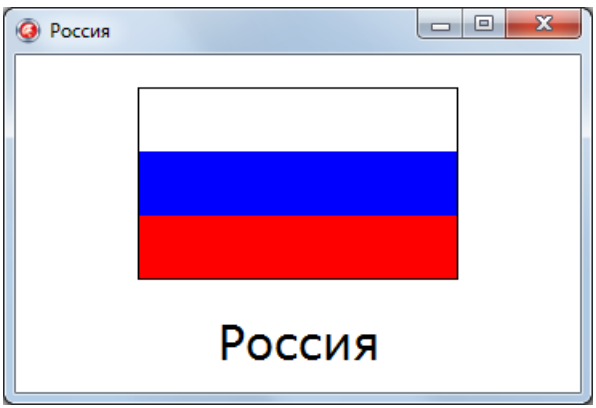


Рис. 1.23. Вывод рисунка на форму

```
// обработка события Paint
// процедура рисует флаг Российской Федерации
procedure TForm1.FormPaint(Sender: TObject);
const
    W = 200; // ширина флага (полосы)
    H = 40;  // высота полосы


var
    x,y: integer; // левый верхний угол

     st: string;
     wt: integer; // ширина области вывода текста
     xt: integer;
```

```
begin

    // вычислить X координату левого верхнего угла флага
    x := Round((ClientWidth — W) / 2);
    y:= 20;

    with Form1.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+W,y+H);
        Brush.Color := clBlue;
        Pen.Color   := clBlue;
        Rectangle(x,y+H,x+W,y+2*H);
        Brush.Color := clRed;
        Pen.Color   := clRed;
        Rectangle(x,y+2*H,x+W,y+3*H);
        // контур
        Pen.Color := clBlack;
        Brush.Style := bsClear; // "прозрачная" кисть
        Rectangle(x,y,x+W,y+h*3);

        // текст выведем два раза: сначала белым,
        // затем (со смещением) — черным
        // В результате у букв будет тень
        Font.Size := 22;
        Font.Name := 'Verdana';
        st := 'Россия';
        wt := Canvas.TextWidth(st); // ширина области вывода
                                    // текста
        // чтобы текст был в середине окна по горизонтали
        xt := (Form1.ClientWidth — wt ) div 2;

        Font.Color := clWhite;
        TextOut(xt,y+3*H+20, st);
        Font.Color := clBlack;
```

```
        TextOut(xt+1,y+3*H+20+1, st);
    end;
end;
```

26. Напишите программу, в окне которой в точке щелчка кнопкой мыши вычерчивается контур пятиконечной звезды (рис. 1.24). В качестве графической поверхности используйте форму.

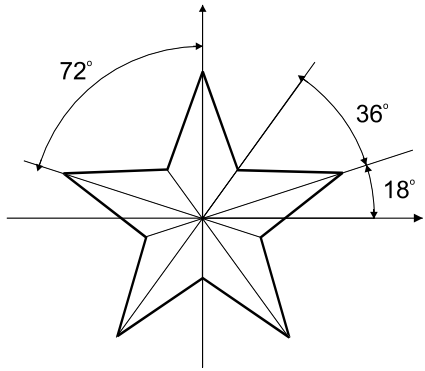


Рис. 1.24. Звезда

```
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.
```

27. Напишите программу **Тир**. В ее окне случайным образом должна "прыгать" цель, например веселая рожица (рис. 1.25), на которой пользователь может сделать щелчок кнопкой мыши. Движение рожицы должно прекратиться после того, как пользователь сделает 10 щелчков кнопкой мыши.

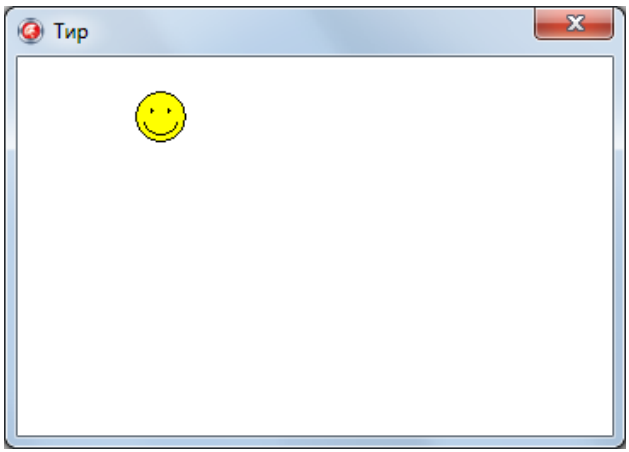


Рис. 1.25. Окно программы **Тир**

```
type
  TForm1 = class(TForm)
    Timer: TTimer;
    Label1: TLabel;
    Button1: TButton;
    procedure TimerTimer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
```

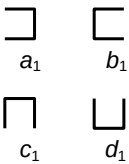
```
    procedure FormMouseDown(Sender: TObject;
                            Button: TMouseButton;
                            Shift: TShiftState;
                            X, Y: Integer);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }

    { ****************************
      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.
   Canvas.Pen.Color := Form1.Color;   // цвет окружности
   Canvas.Brush.Color := Form1.Color; // 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)
```

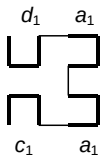
```
  then begin
       // щелчок по рожице
       inc(p);
end;

  Form1.Caption := 'Выстрелов: '+IntToStr(n) +
                        ' Попаданий: ' + IntToStr(p);
  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
    n := 0; // выстрелов
    p := 0; // попаданий
    Label1.Visible := False;  // скрыть сообщение
    Button1.Visible := False; // скрыть кнопку
    Timer.Enabled := True;    // пуск таймера
end;
```

28. Напишите программу, которая на поверхности формы вычерчивает кривую Гильберта (рис. 1.26). Пример кривой Гильберта пятого порядка приведен на рис. 1.27.

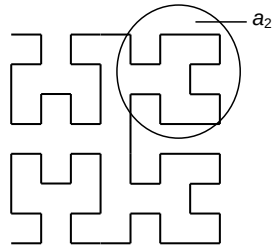
```
var
  p: integer = 5;  // порядок кривой
  u: integer = 7;  // длина штриха
```



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. 1.26. Кривые Гильберта

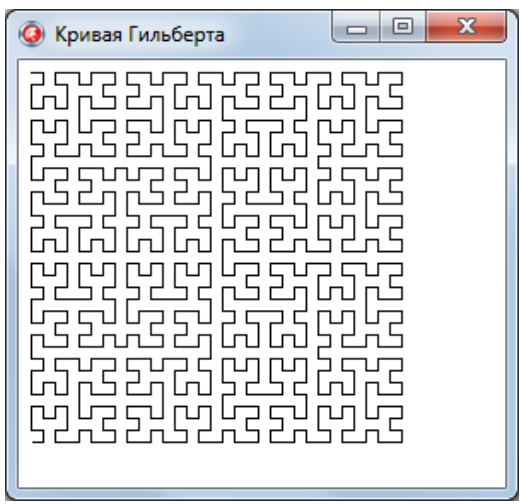


Рис. 1.27. Кривая Гильберта пятого порядка

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

29. Напишите программу, которая на поверхность формы выводит изображение оцифрованной координатной сетки (рис. 1.28). Программа должна быть спроектирована таким образом, чтобы при изменении размера окна сетка перерисовывалась и занимала всю область окна.

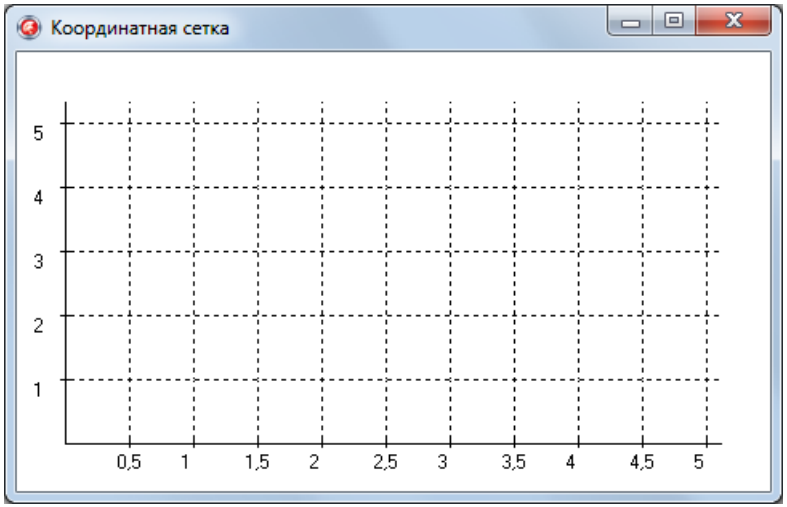


Рис. 1.28. Координатная сетка

```
// обработка события Paint
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 неоцифрованных линий сетки
 dcross:integer; // количество неоцифрованных линий
                 // между оцифрованными
begin
 x0:=30;
 y0 := Form1.ClientHeight - 30;

 h := Form1.ClientHeight - 60;
 w := Form1.ClientWidth - 60;
```

```
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.

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;
```

30. Напишите программу, которая на поверхности формы вычерчивает график функции, например $2\sin(x)e^{x/5}$. Окно программы приведено на рис. 1.29. Программа должна быть спроектирована таким образом, чтобы при изменении размера окна график перерисовывался и занимал всю область окна.

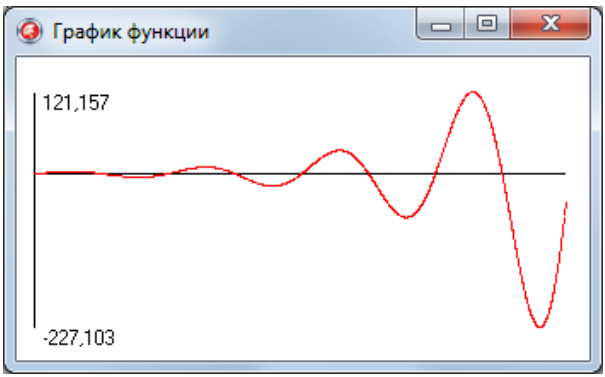


Рис. 1.29. Окно программы **График функции**

```
// 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;          // ширина
 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;
```

31. Напишите программу, которая строит диаграмму, отображающую значения четырех категорий, например, результаты экзамена (категории: отлично, хорошо, удовлетворительно, неудовлетворительно). Для ввода исходных данных используйте компонент `StringGrid`. Пример окна программы во время ее работы приведен на рис. 1.30.

```
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
```

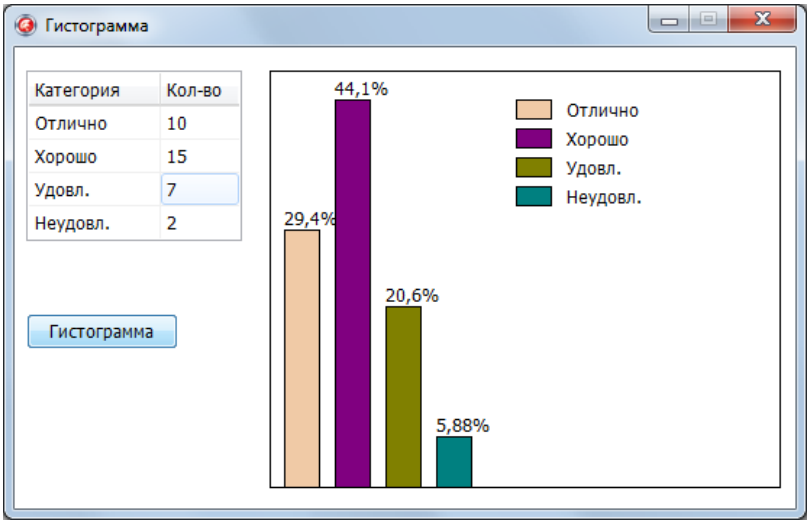


Рис. 1.30. Окно программы **Гистограмма**

```
  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 := FormatSettings.DecimalSeparator;
              if Pos(FormatSettings.DecimalSeparator,
                 StringGrid1.Cells[StringGrid1.Row,
                 StringGrid1.Col]) <> 0
              then Key := Char(0);
         end;
      else Key := Char(0);
      end;
end;

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

32. Напишите программу, которая отражает ряд данных, например, результаты сдачи экзамена в виде круговой диаграммы (рис. 1.31).

```
implementation

{$R *.dfm}

const
NR = 4; // кол-во строк в таблице
var
kat: array[1..NR] of string =
('Отлично','Хорошо','Удовлетворительно','Неудовлетворительно');
n: array[1..NR] of real;  // значения категорий
p: array[1..NR] of real;  // процент категории в общей сумме
```

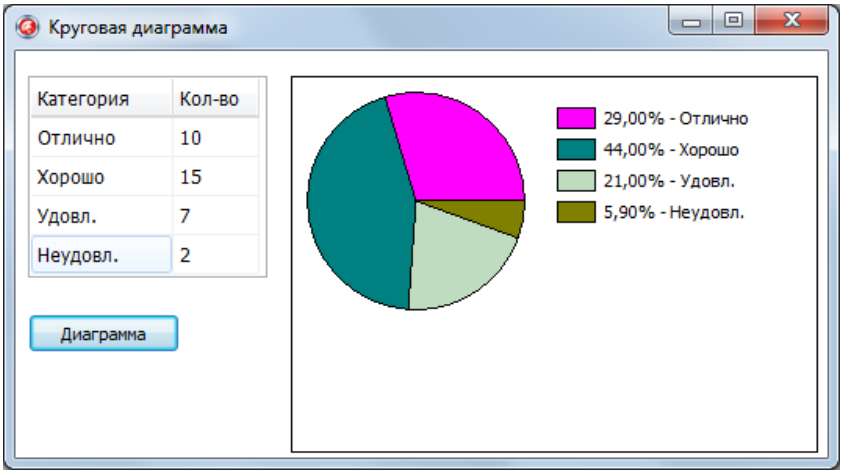


Рис. 1.31. Окно программы **Круговая диаграмма**

```
// цвет секторов диаграммы
BarColor: array[1..4] of TColor =
(clFuchsia,clTeal,clMoneyGreen,clOlive);

// ввод и обработка
// если исх. 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;

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

// круговая диаграмма
procedure cdiagr;
var
x1,y1,x2,y2 : integer; // круг (эллипс)
x0,y0: integer;        // центр круга (эллипса)
x3,y3,x4,y4 : integer; // линии, ограничивающие сектор
a1,a2: integer;        // угол начала и конца сектора

x,y: integer;   // координаты области вывода
                // легенды (левый верхний угол)
i: integer;
r: integer;

st: string;
begin
x1:= 10; y1:=10;
x2:= 150; y2 :=150;
```

```
x0 := x1 + (x2-x1) div 2;
y0 := y1 + (y2-y1) div 2;

r := (x2-x1) div 2;
with Form1.Image1 do
begin
  Canvas.Brush.Color := clWindow;
  Canvas.Rectangle(0,0,Width,Height);

  a1 := 1; // строим от оси OX

  x3 := x2;
  y3 := y1 + r;

  for i:=1 to 4 do
  begin
    if i < 4 then
        a2 := a1 + Round(360 * P[i])
      else
        a2 := 360;

    x4:= x0  + Round( r* cos(a2*PI/180));
    y4:= y0  - Round( r* sin(a2*PI/180));

    Canvas.Brush.Color := BarColor[i];
    Canvas.Pie(x1,y1,x2,y2,x3,y3,x4,y4);

    // следующий сектор - от конца текущего
    a1 := a2;
    x3:=x4;
    y3:=y4;

  end;

  // Легенда.
  // Здесь x - коорд. левого угла
  // прямоугольника легенды
  x := x2 + 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;

    st :=  FloatToStrF(p[i]*100,ffFixed,2,2) + '% — ' +
             Form1.StringGrid1.Cells[0,i];

     Canvas.TextOut(x+30,y, st);

    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';
// и содержимое первого столбца
StringGrid1.Cells[0,1]  := kat[1];
StringGrid1.Cells[0,2]  := kat[2];
StringGrid1.Cells[0,3]  := kat[3];
StringGrid1.Cells[0,4]  := kat[4];

StringGrid1.Cells[1,1]  := '10';
StringGrid1.Cells[1,2]  := '15';
StringGrid1.Cells[1,3]  := '7';
StringGrid1.Cells[1,4]  := '2';
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  // исх. данные введены
    then cdiagr; // строим диаграмму
end;

end.
```

33. Напишите программу, которая строит график изменения курса доллара за последние 10 дней (рис. 1.32). Данные для построения графика находятся в текстовом файле, причем значения за последние дни находятся в начале файла (чтобы построить график, надо загрузить первые 10 элементов данных).

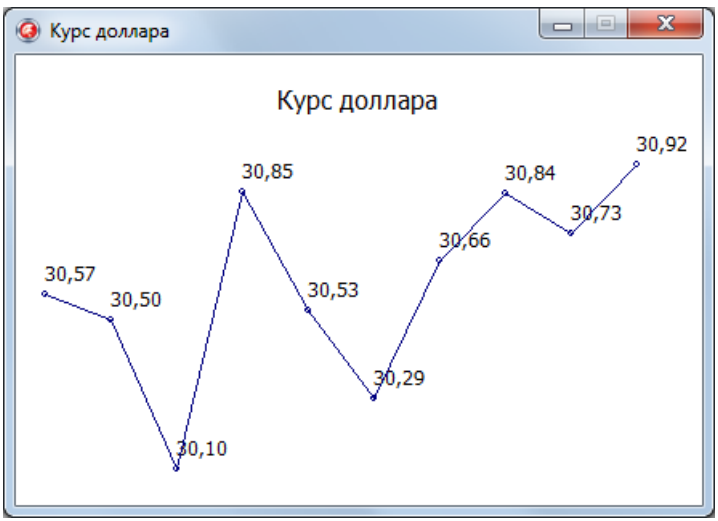


Рис. 1.32. График изменения курса доллара

```
const
  HB=10;
var
  kurs: array[1..HB] of real;
  date: array[1..HB] of TDate;

// конструктор
procedure TForm1.FormCreate(Sender: TObject);
var
  i: integer;
  f: TextFile;
  st: string;

begin
  // предполагается, что файл данных находится в том же
  // каталоге, что и программа (EXE-файл)
  AssignFile(f,'usd.txt');
  try
    Reset(f);
  except
    on E: EInOutError do
    begin
      ShowMessage('Ошибка доступа к файлу данных');
      exit;
    end;
  end;

  // ввод данных
  for i:= HB downto 1 do
  begin
    readln(f,st);
    Date[i] := StrToDate(st);
    readln(f,st);
    kurs[i] := StrToFloat(st);
  end;
  CloseFile(f);
end;
```

```
procedure TForm1.PaintBox1Paint(Sender: TObject);
var
    n: integer ;      // количество точек
    x,y: integer;     // координаты точки
    dx: integer;      // шаг по X

    min,max: integer; // индекс мин. и макс. элементов
    m: real;          // масштаб

    i: integer;
    st: string;

begin
    with PaintBox1.Canvas do begin
      // заголовок
      Font.Name := 'Tahoma';
      Font.Size := 12;
      x := Round((PaintBox1.Width -
                  TextWidth('Курс доллара'))/2);
      y := 10;
      Brush.Style := bsClear;
      TextOut(x,y,'Курс доллара');

      // подзаголовок
      y := y + TextHeight('Курс доллара');
      Font.Size := 9;
      st := '('+ DateToStr(Date[1]) +
              ' — ' + DateToStr(Date[HB])+ ')';
      x := Round((PaintBox1.Width — TextWidth(st))/2);
      Brush.Style := bsClear;
      TextOut(x,y,st);

      // определить кол-во элементов массива
      n := Round( sizeof(kurs) / sizeof(real));

     // найти минимальное и максимальное значения ряда данных
      min := 1;  // пусть первый элемент минимальный
      max := 1;  // пусть первый элемент максимальный
```

```
for i := 1 to n do
  begin
    if (kurs[i] < kurs[min]) then min := i;
    if (kurs[i] > kurs[max]) then max := i;
  end;

 { Если разница между минимальным и максимальным
  значениями незначительна,
  то график получается ненаглядным.
  В этом случае можно построить не абсолютные значения,
  а отклонения от минимального значения ряда. }

  Font.Size := 9;
  Pen.Width := 1;
  Pen.Color := clNavy;

  dx:= Round((PaintBox1.Width — 40) / (n-1));

  // Вычислим масштаб.
  // Для построения графика будем использовать
  // не всю область компонента, а ее нижнюю часть.
  // Верхняя область высотой 60 пикселов используется
  // для отображения заголовка
  m := (PaintBox1.Height — 70) / (kurs[max] -
               kurs[min]);

  x := 10;
  for i := 1 to n do
    begin
      y := PaintBox1.Height -
               Round( (kurs[i] — kurs[min]) * m)-10;

      // поставить точку
      // Rectangle(x-2,y-2,x+2,y+2); // квадрат
      Ellipse(x-2,y-2,x+2,y+2); // окружность
      if (i <> 1) then
          LineTo(x,y);

      // подпись данных
      if ( ( i = 1) or (kurs[i] <> kurs[i-1])) then
```

```
            begin
            st := FloatToStrF(kurs[i],ffFixed,4,2);
            Brush.Style := bsClear;
            TextOut(x,y-20,st);
          end;
      { т. к. метод TextOut меняет положение точки (карандаша),
        из которой рисует метод LineTo, то после вывода текста
        надо переместить указатель (карандаш) в точку (x,y) }
         MoveTo(x,y);
         x := x + dx;
    end;
  end;
end;
```

34. Напишите программу, по поверхности окна которой перемещается графический объект, например корабль (рис. 1.33). Вид окна программы приведен на рис. 1.34.

```
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)
```

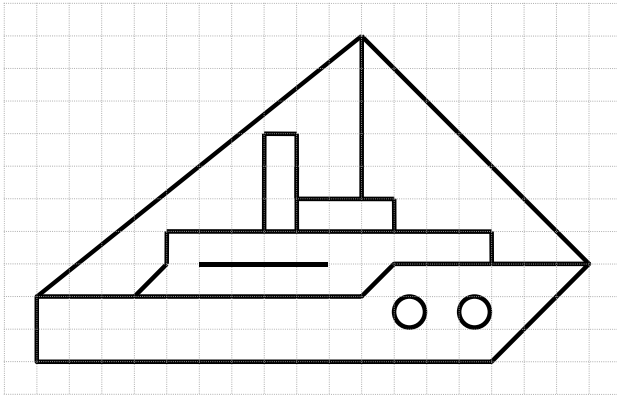


Рис. 1.33. Корабль

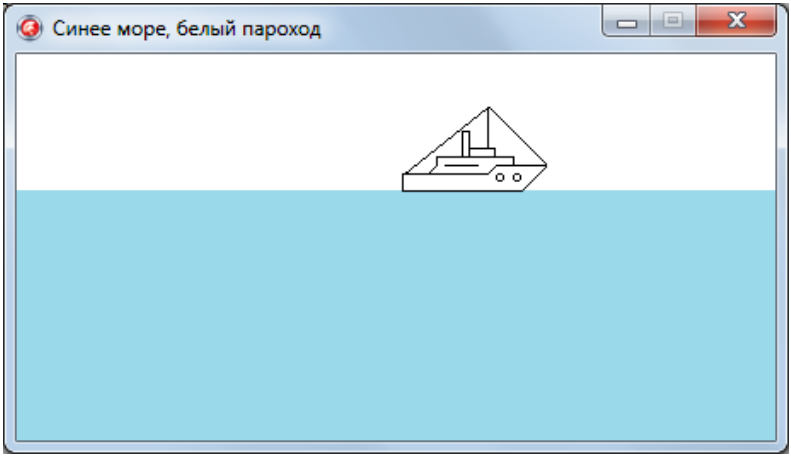


Рис. 1.34. Простая мультипликация — корабль, плывущий по морю

```
// 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
  // с помощью 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.
```

35. Напишите программу, которая в диалоговом окне выводит изображение идущих часов с часовой, минутной и секундной стрелками (рис. 1.35).

```
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);
```

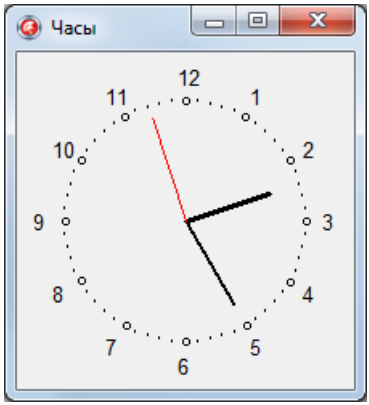


Рис. 1.35. Окно программы **Часы**

```
    // 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 := Form1.Color;
  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 := clRed;
  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;
```

36. Напишите программу, в окне которой летит самолет (рис. 1.36). Изображение самолета и фона (рис. 1.37) должны загружаться из файла.

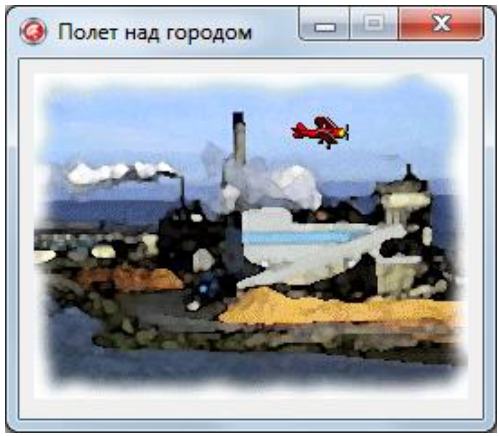


Рис. 1.36. Летящий над городом самолет

Рис. 1.37. Самолет и фоновый рисунок

```
var
  Back, Plane: 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

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

  // загрузить и вывести фон
  Back.LoadFromFile('factory.bmp');
  PaintBox1.Width  := Back.Width;
  PaintBox1.Height := Back.Height;
  PaintBox1.Canvas.Draw(0,0,Back);

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

  // определим "прозрачный" цвет
  Plane.Transparent := True;
  // прозрачный цвет картинки определяет
  // левый верхний пиксел картинки
  Plane.TransParentColor := Plane.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
     PaintBox1.Canvas.CopyRect(BackRct,Back.Canvas,BackRct);

     x:=x+2;
     if x > PaintBox1.Width then x:=-W;

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

     // выведем рисунок
    PaintBox1.Canvas.Draw(x,y,Plane);
end;

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

// обработка события Paint
procedure TForm1.FormPaint(Sender: TObject);
begin
  PaintBox1.Canvas.Draw(0,0,Back);
  PaintBox1.Canvas.Draw(x,y,Plane);
end;
```

37. Напишите программу, в окне которой отображается баннер, например вращающийся текст (рис. 1.38). Кадры баннера должны загружаться из файла. Пример файла баннера приведен на рис. 1.39.

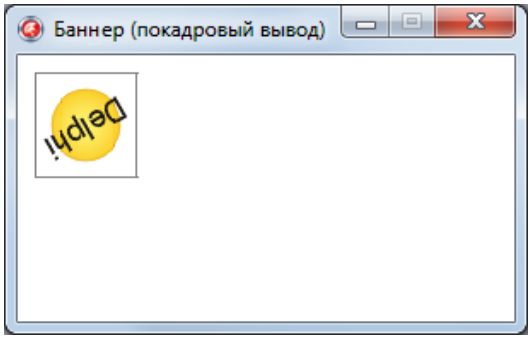


Рис. 1.38. Баннер

Рис. 1.39. Кадры баннера — 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 (рис. 1.40).

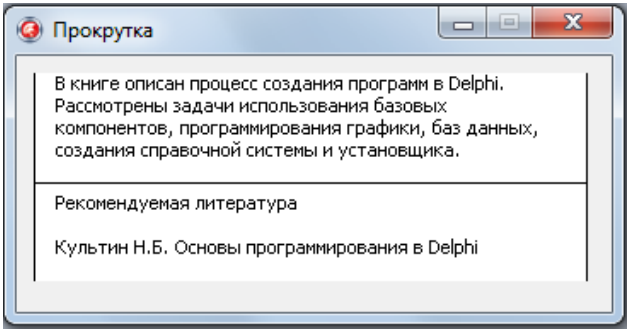


Рис. 1.40. Окно программы **Прокрутка**

```
implementation
{$R *.dfm}
var
    pic :TBitMap;        // прокручиваемая картинка
    sRect,dRect: TRect; // область-источник
    t: integer;          // смещение по вертикали
                         // отображаемой области
                         // относительно начала плаката
```

```
    hb: integer;  // высота области-картинки
    hr: integer;  // высота прокручиваемого "плаката"

    //  В файле плакат должен быть продублирован по вертикали
    //  два раза. Высота плаката должна быть
    //  больше или равна высоте области просмотра

procedure TForm1.FormCreate(Sender: TObject);
begin
    pic := TBitMap.Create;
    pic.LoadFromFile('baner-2.bmp');   // загрузить картинку

    hb := pic.Height div 2; // предполагается, что плакат
                    // продублирован в файле два раза
    hr := pic.Height div 2;

     // положение и размер области, в которой
     // прокручивается картинка
    dRect := Bounds(10,10,pic.Width,HB);

    sRect := Rect(0,0,pic.Width,HB);  // отображаемая область

    t:=0;
end;

// сигнал от таймера
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    // отобразить часть картинки
    Canvas.CopyRect(dRect,pic.Canvas,sRect);
    inc(t);

    if t = hr // высота плаката
        then t:=0;
    sRect := Bounds(0,t,pic.Width,HB);  // следующий кадр
end;
```

39. Напишите программу, в окне которой в стиле бегущей строки прокручивается битовый образ (рис. 1.41). Битовый образ должен загружаться из ресурса программы (подготовить файл ресурса можно, например, с помощью утилиты Image Editor).

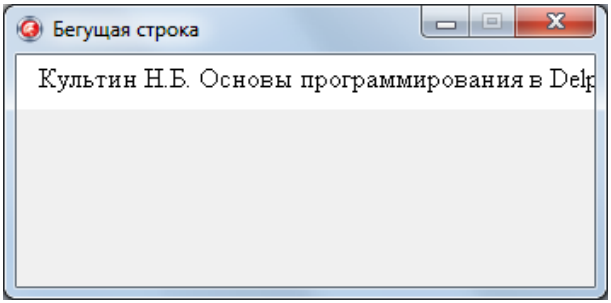


Рис. 1.41. Окно программы **Бегущая строка**

```
{ 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
                // с помощью Image Editor.
            // Внимание! 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,'BANER_2');
                                      // 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;

end.
```

40. Напишите программу, используя которую можно просмотреть иллюстрации, находящиеся в одном из каталогов компьютера. Выбор папки, содержащей картинки, должен осуществляться в стандартном окне **Обзор папок**. Окно программы приведено на рис. 1.42.

Рис. 1.42. Окно программы **Просмотр иллюстраций**

```
unit Unit1;

interface

uses

  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, Buttons, ExtCtrls,
  FileCtrl, Jpeg;
  // ссылки на модуль FileCtrl и Jpeg вставлены вручную!!!
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Image1: TImage;
    SpeedButton2: TSpeedButton;
    SpeedButton1: TSpeedButton;
    SpeedButton3: TSpeedButton;
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
```

```
  private
    aPath: string;   // каталог, который выбрал пользователь

    Pictures : TStringList;  // список иллюстраций
    n: integer; // номер иллюстрации, отображаемой
                // в данный момет
    procedure ShowPicture(i: integer);

  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
begin
  Pictures := TStringList.Create;
  SpeedButton2.Enabled := False;
  SpeedButton3.Enabled := False;
end;

// щелчок на кнопке "Папка"
procedure TForm1.SpeedButton1Click(Sender: TObject);
var
  aSearchRec : TSearchRec; // информация о файле
  r: integer;
begin
  if SelectDirectory('Выберите каталог', '',aPath) then
  begin
    aPath := aPath + '\';
    Form1.Caption := 'Просмотр иллюстраций — ' + aPath;

    // сформировать список иллюстраций
    r := FindFirst(aPath+'*.jpg',faAnyFile,aSearchRec);
```

```
    if r = 0 then
      begin
        // в указанном каталоге есть JPG-файл
        Pictures.Clear; // очистить список иллюстраций
        Pictures.Add(aSearchRec.Name); // добавить имя файла
                                       // в список иллюстраций
      end;

    // получить имена остальных JPG-файлов
    repeat
      r := FindNext(aSearchRec); // получить имя
                                 // следующего JPG-файла
      if r = 0 then
          Pictures.Add(aSearchRec.Name);
    until ( r <> 0);

    if Pictures.Count > 1 then
        SpeedButton2.Enabled := True;

    // отобразить иллюстрацию
     n := 0; // номер отображаемой иллюстрации
    ShowPicture(n);

    if Pictures.Count = 1 then
        SpeedButton2.Enabled := False;
  end;
end;

// вывод следующей картинки
procedure TForm1.SpeedButton2Click(Sender: TObject);
begin

   n := n+1;

   ShowPicture(n); // вывести картинку
   if n = Pictures.Count-1 then
      SpeedButton2.Enabled := False;

   // если кнопка "Предыдущая" не доступна,
   // сделать ее доступной
```

```
   if (n > 0 ) and SpeedButton3.Enabled = False then
        SpeedButton3.Enabled := True;
end;

// вывод предыдущей картинки
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin

   n := n-1;

   ShowPicture(n);  // вывести картинку

   if n = 0 then
      SpeedButton3.Enabled := False;

   // если кнопка "Следующая" не доступна,
   // сделать ее доступной
   if (n < Pictures.Count) and SpeedButton2.Enabled = False
then
        SpeedButton2.Enabled := True;
end;

// выводит i-ю иллюстрацию
procedure TForm1.ShowPicture(i: integer);
begin
    try
       Image1.Picture.LoadFromFile(aPath + Pictures[i]);
       Caption := 'Просмотр иллюстраций: ' + aPath +
                  Pictures[i];
 except on EInvalidGraphic do Image1.Picture.Graphic := nil;
   end;
end;

end.
```

41. Напишите программу, в окне которой фоновый рисунок формируется путем многократного отображения картинки по принципу кафельной плитки (рис. 1.43).

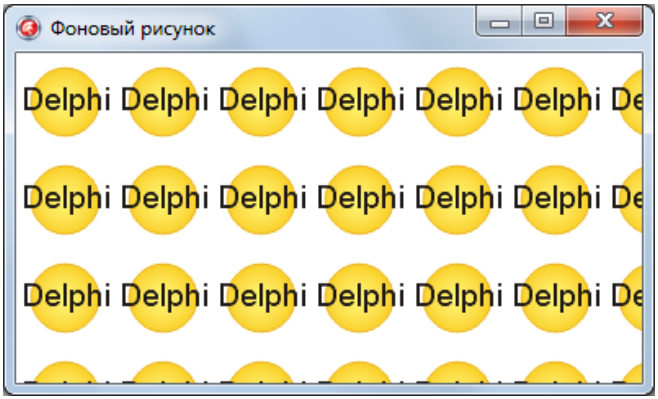


Рис. 1.43. Фоновый рисунок, сформированный путем многократного отображения небольшой картинки

```
implementation
var
    Back : TBitmap; // фоновая картинка
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
   Back := TBitmap.Create;
   Back.LoadFromFile('back.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

- 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 окне которой можно просмотреть простой, не содержащий звук, AVI-ролик. В окне программы, помимо самого ролика, должна отображаться информация о файле (рис. 1.44).

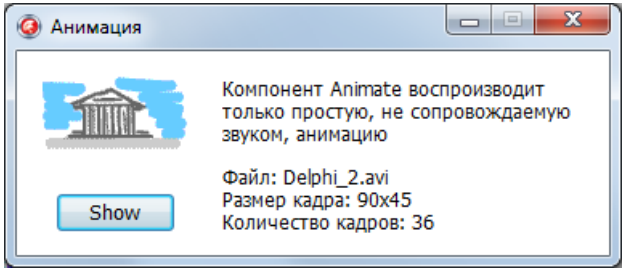


Рис. 1.44. Окно программы **Анимация**

```
// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
const
  fn = 'Delphi_2.avi';
var
  st: string;
begin
  try
      Animate1.FileName := fn;
      st := 'Файл: ' + fn + #13 + 'Размер кадра: ' +
            IntToStr(Animate1.Width) + 'x'+
```

```
            IntToStr(Animate1.Height) + #13 +
            'Количество кадров: ' +
            IntToStr(Animate1.FrameCount);
      Label2.Caption := st;
    except on e: Exception do
    begin
      Label1.Caption := 'Ошибка загрузки файла ' +
                        fn + #13 +
                      'Файл недоступен или содержит анимацию,
                       которая сопровождается звуком.';
      Button1.Enabled := False;
    end;
  end;
end;

// обработка события Paint
procedure TForm1.FormPaint(Sender: TObject);
begin
  // активизировать воспроизведение анимации
  // с первого по последний кадр
  Animate1.Play(1, Animate1.FrameCount,1);

  // отобразить последний кадр анимации
  //  Animate1.StartFrame := Animate1.FrameCount;
end;

// щелчок на кнопке Show
procedure TForm1.Button1Click(Sender: TObject);
begin
  // активизировать воспроизведение анимации
  // с первого по последний кадр
  Animate1.Play(1, Animate1.FrameCount,1);
end;
```

43. Напишите программу, в окне которой при ее запуске отображается AVI-ролик — сопровождаемая звуком анимация (рис. 1.45). Для воспроизведения анимации используйте компонент `MediaPlayer`, в качестве экрана — компонент `Panel`.

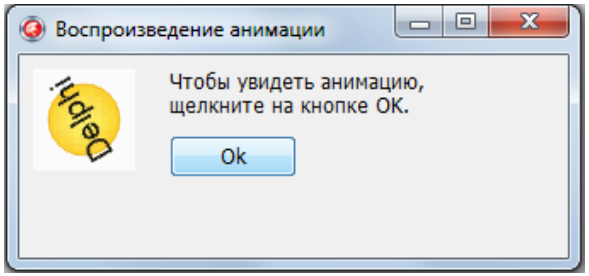


Рис. 1.45. Окно программы **Воспроизведение анимации**. В качестве экрана используется компонент `Panel`

```
procedure TForm1.FormCreate(Sender: TObject);
begin
       Button1.Enabled := False;
       // настройка компонента MediaPlayer
       MediaPlayer1.Visible := False;
       MediaPlayer1.Display := Panel1;
       MediaPlayer1.FileName := 'delphi_s.avi';
       try
           MediaPlayer1.Open;
           Button1.Enabled := True;
           // Размер области отображения анимации
           // должен соответствовать размеру кадра анимации.
           // Размер кадра в файле delphi_s.avi — 60x60

           // зададим размер области вывода анимации
           MediaPlayer1.DisplayRect:=Rect(0,0,60,60);
       except
         on e: exception do
           begin
              Label1.Caption := 'Нет файла delphi_s.avi';

           end;
       end;
end;

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

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

44. Напишите программу, используя которую можно просматривать AVI-анимацию в реальном масштабе времени или по кадрам (рис. 1.46). Для отображения анимации используйте компонент `Animate`. Обратите внимание, что этот компонент работает только с AVI-файлами, в которых нет звука. Для выбора файла анимации используйте стандартное диалоговое окно **Открытие файла**. Форма программы приведена на рис. 1.47.

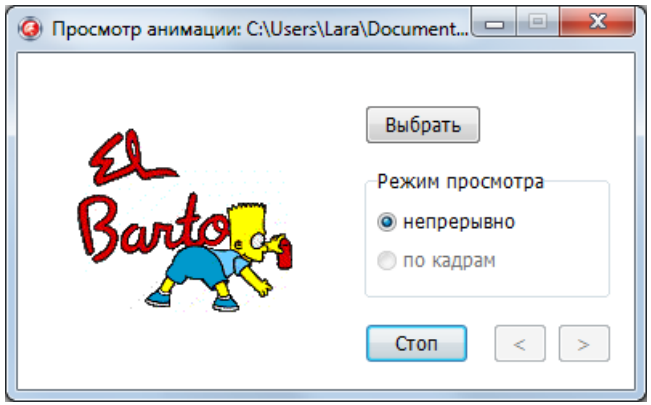


Рис. 1.46. Окно программы **Просмотр анимации**

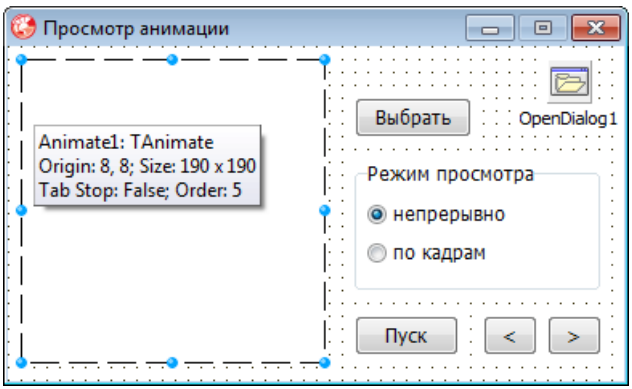


Рис. 1.47. Форма программы **Просмотр анимации**

```
var
   fName: string;        // файл анимации
   CFrame: integer;  // номер отображаемого кадра

// Щелчок на кнопке Выбрать открывает
// стандартное диалоговое окно Открытие файла
procedure TForm1.Button4Click(Sender: TObject);
begin
    OpenDialog1.Title := 'Выбрать AVI-файл';
    OpenDialog1.InitialDir :='';
    if OpenDialog1.Execute then
    begin
        fName := OpenDialog1.FileName;
        //OpenAVI;
        Button1.Enabled := False;
    Button2.Enabled := False;
    Button3.Enabled := False;
    RadioButton1.Enabled := False;
    RadioButton2.Enabled := False;
    try
        Animate1.FileName := fName;
    except
      on Exception do
      begin
        MessageDlg('Ошибка формата AVI-файла.'+
           #13+'(Анимация не должна сопровождаться звуком.)',
           mtError,[mbOk],0);
        exit;
      end;
    end;

    Form1.Caption := 'Просмотр анимации: ' +  fName;
    Button1.Enabled := True;
    RadioButton1.Enabled := True;
    RadioButton2.Enabled := True;
    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
          // вывод с первого до последнего кадра:
          Animate1.StartFrame:=1; // первый
          Animate1.StopFrame:=Animate1.FrameCount;//последний
          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 просмотра по кадрам
     Form1.Button3.Enabled:=False;
     Form1.Button2.Enabled:=False;
end;

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

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

```
// 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;

// 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;
```

45. Напишите программу, используя которую можно прослушать звуковые файлы Windows. Рекомендуемый вид диалогового окна программы приведен на рис. 1.48.

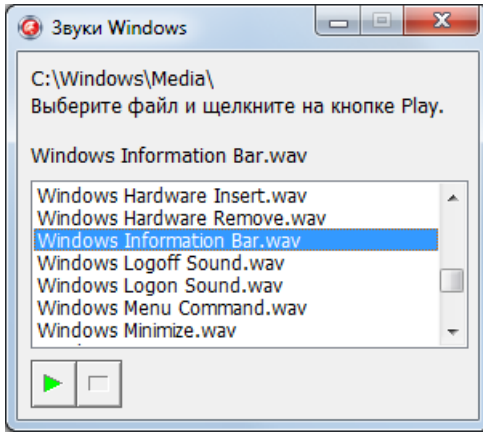


Рис. 1.48. Окно программы **Звуки Windows**

```
unit WinSound_;

interface

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

type
  TForm1 = class(TForm)
    MediaPlayer1: TMediaPlayer;
    ListBox1: TListBox;
    Label2: TLabel;
    Label1: TLabel;
    Label3: TLabel;
    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;

// Внимание! Добавить в список используемых модулей
// ссылку на SHFolder!
function GetSpecialFolderPath(folder : integer) : string;
const
  SHGFP_TYPE_CURRENT = 0;
var
  path: array [0..MAX_PATH] of char;
begin
  if SUCCEEDED(
  SHGetFolderPath(0,folder,0,SHGFP_TYPE_CURRENT,@path[0]))
then
    Result := path
  else
    Result := '';
end;


{ формирует список WAV-файлов,
  находящихся в подкаталоге Windows\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
  soundpath := GetSpecialFolderPath(CSIDL_WINDOWS);
```

```
  soundpath := soundpath + '\Media\';
  label3.Caption := soundpath;

  // 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
          ListBox1.Items.Add(SearchRec.Name);

      // пока в каталоге есть другие файлы с расширением wav
          while (FindNext(SearchRec) = 0) do
              ListBox1.Items.Add(SearchRec.Name);

          ListBox1.ItemIndex := 0;
        Label2.Caption := ListBox1.Items[ListBox1.ItemIndex];
     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];
    MediaPlayer1.FileName := soundpath +
          ListBox1.Items[ListBox1.itemIndex];
    MediaPlayer1.Open;
    MediaPlayer1.Play;
end;

// щелчок на кнопке компонента MediaPlayer
procedure TForm1.MediaPlayer1Click(Sender: TObject;
                                   Button: TMPBtnType;
                               var DoDefault: Boolean);
begin
     if (Button = btPlay) then
     begin
          // нажата кнопка Play
          MediaPlayer1.FileName :=
             soundpath+ListBox1.Items[ListBox1.itemIndex];
```

```
            MediaPlayer1.Open;
        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 окна программы приведен на рис. 1.49.

Рис. 1.49. Окно программы **Video Player**

```
unit Vp_;

interface

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

```
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 byte;    // 'RIFF'
        nu1:  array[1..5] of dword;   // не используется
        AVIH: array[1..4] of byte;    // 'avih'
        nu2:  array[1..9] of dword;   // не используется
        Width:  dword;
        Height: dword;
    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;

    Form1.Caption := OpenDialog.FileName;
    // 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;

    // активизировать процесс воспроизведения
    SpeedButton1.Down := True;  // "нажать" кнопку Play
    MediaPlayer.Play;

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 **MP3 Player**. Программа должна обеспечивать возможность выбора каталога, в котором находятся MP3-файлы, а также регулировку громкости звука в окне программы. Для отображения списка MP3-файлов используйте компонент `ListBox`, а для управления медиаплеером — кнопки `SpeedButton`. Окно программы приведено на рис. 1.50.

```
unit mp3p_;
interface
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)

    SpeedButton1: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton4: TSpeedButton;
    ListBox1: TListBox;
    TrackBar1: TTrackBar;
```

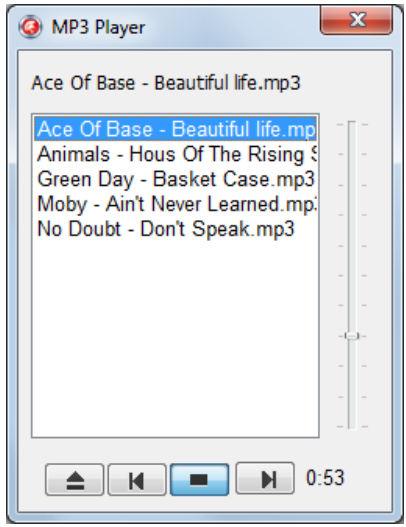


Рис. 1.50. Окно программы **MP3 Player**

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

    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);

  private
    { Private declarations }

    // эти объявления вставлены сюда вручную

    MediaPlayer1: TMediaPlayer; // медиаплеер

    procedure Play; // воспроизведение
    procedure PlayList(Path: string); // формирует список
                                      // MP3-файлов
```

```
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

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

// конструктор формы
procedure TForm1.FormCreate(Sender: TObject);
begin
   // создать медиаплеер
   MediaPlayer1 := TMediaPlayer.Create(self);
   MediaPlayer1.Parent := Form1;
   MediaPlayer1.Visible := FALSE;

   // настроить кнопку так, чтобы ее можно было "отжать"
   SpeedButton2.AllowAllUp := True;
   SpeedButton2.GroupIndex := 1;

   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;
```

```
// 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;


// 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
procedure TForm1.SpeedButton4Click(Sender: TObject);
var
  Root: string;       // корневой каталог
  pwRoot : PWideChar;
  Dir: string;
Begin
  // выбрать папку, в которой находятся MP3-файлы
  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

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

48. Напишите программу, которая позволяет увидеть содержимое текстового файла. Для отображения текста используйте компонент `Memo`, для получения от пользователя имени файла — стандартное диалоговое окно **Открыть файл**. Рекомендуемый вид формы приведен на рис. 1.51.

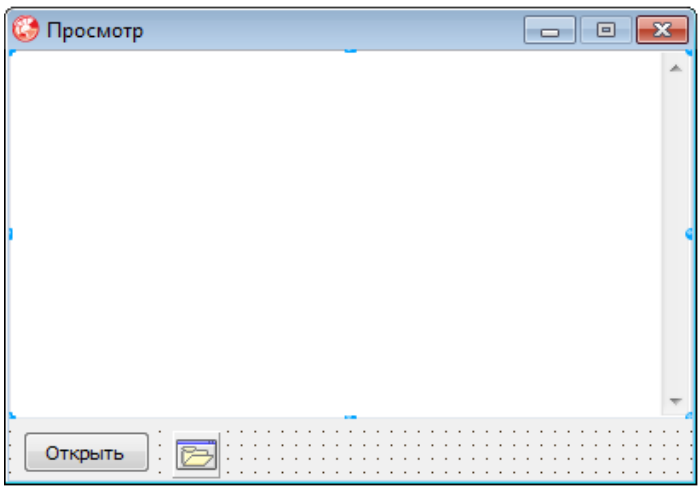


Рис. 1.51. Форма программы **Просмотр**

```
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.
```

49. Напишите программу, которая позволяет увидеть и, в случае необходимости, изменить содержимое текстового файла. Рекомендуемый вид формы приведен на рис. 1.52.

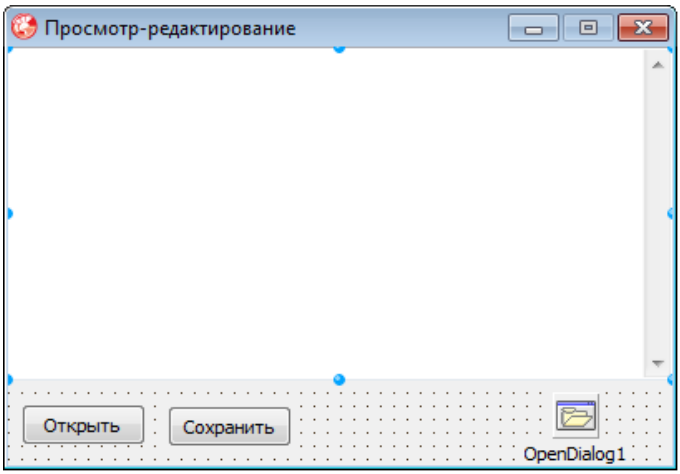


Рис. 1.52. Форма программы **Просмотр-редактирование**

```
implementation

{$R *.dfm}

var
  fName: string; // имя файла

// 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
```

```
   buf: String[80];   // буфер для чтения строк

begin
   OpenDialog1.Options := [ofFileMustExist];
   if OpenDialog1.Execute then
    begin
      Memo1.Lines.Clear;
      // пользователь закрыл окно Открыть,
      // сделав щелчок на кнопке Ok
      fName := OpenDialog1.FileName;
      AssignFile(f, fName);
      Reset(f);
      // 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
      Button2.Enabled := True;
    end;
end;

// щелчок на кнопке Сохранить
procedure TForm1.Button2Click(Sender: TObject);
var
   f: TextFile;       // файл
   i: integer;
begin
   AssignFile(f, fName);

   Rewrite(f);  // открыть для перезаписи

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

```
   CloseFile(f); // закрыть файл

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

end.
```

50. Напишите программу, которая в указанном пользователем каталоге и его подкаталогах выполняет поиск заданного файла или соответствующих указанной маске файлов. Рекомендуемый вид формы программы приведен на рис. 1.53. Для ввода имени каталога во время работы программы используйте стандартное диалоговое окно **Обзор папок**.

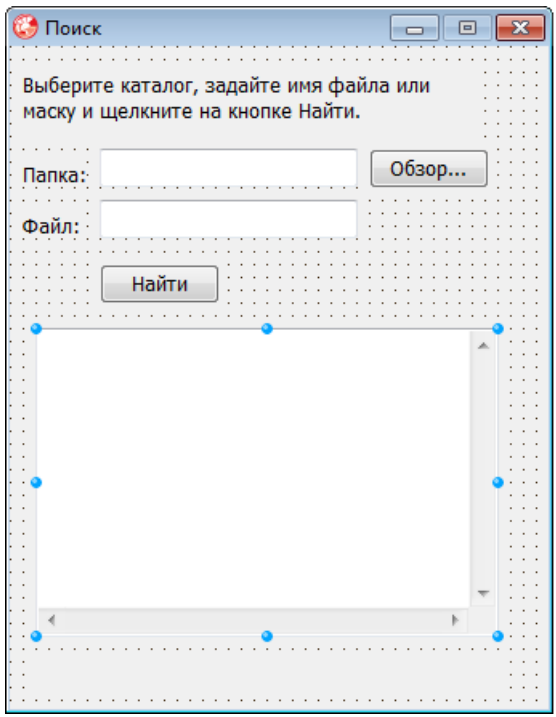


Рис. 1.53. Форма программы **Поиск**

```
// 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

// 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

51. 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. 1.54). Задача игрока состоит в том, чтобы, не вынимая фишки из коробочки, выстроить фишки в правильном порядке (рис. 1.55). Напишите программу **Игра 15** (рис. 1.56).

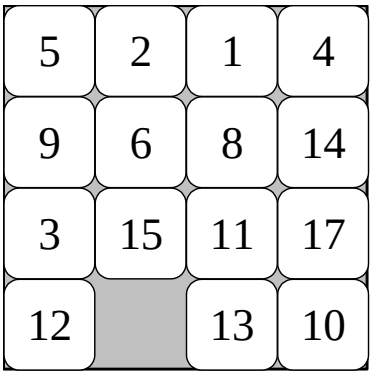


Рис. 1.54. В начале игры фишки перемешаны

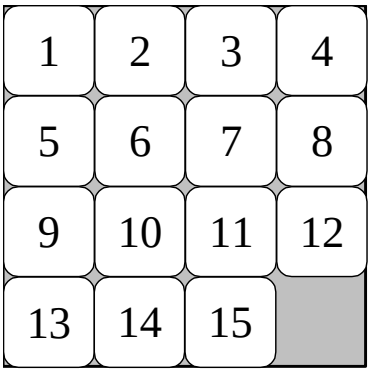


Рис. 1.55. Правильный порядок фишек

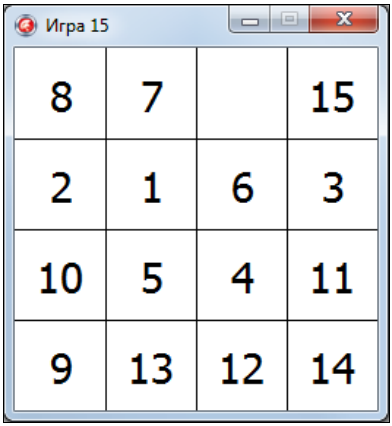


Рис. 1.56. Окно программы **Игра 15**

```
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);
  private
    { Private declarations }
    // эти объявления вставлены сюда вручную
    procedure ShowPole;
    procedure Mixer;
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
```

```
implementation
{$R *.dfm}
const
    H = 4;   W = 4;   // размер поля — 4x4
    CH = 62; CW = 62; // 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:=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
// 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 Create
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 Paint
procedure TForm1.FormPaint(Sender: TObject);
begin
    Form1.ShowPole;
end;
end.
```

52. Напишите программу **Собери картинку** — аналог игры "15", в которой игрок будет перемещать не цифры, а фрагменты картинки (рис. 1.57).

```
{
  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) Никита Культин, 2003-2012.
}
```

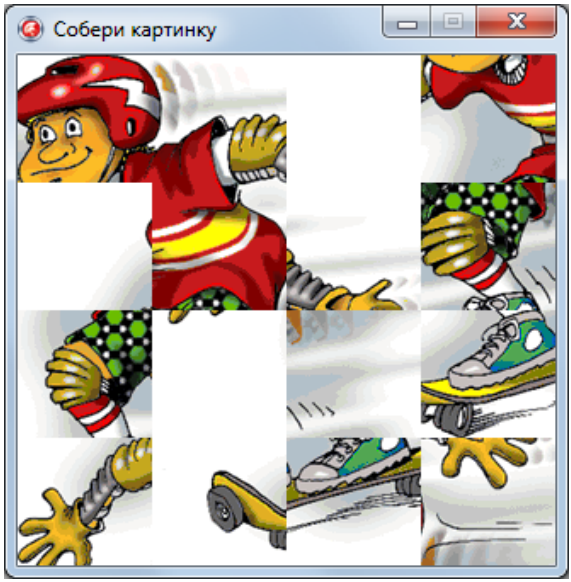


Рис. 1.57. Окно программы **Собери картинку**

```
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);
  private
    { Private declarations }
    // эти объявления вставлены сюда вручную
    procedure ShowPole;
    procedure Mixer;
    procedure NewGame;
```

```
  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 Create
procedure TForm1.FormCreate(Sender: TObject);
begin
    pic := TBitMap.Create;
    NewGame;
end;

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

53. 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. Форма и окно программы приведены на рис. 1.58 и 1.59.

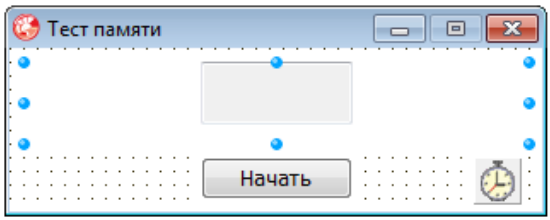


Рис. 1.58. Форма программы **Тест памяти**

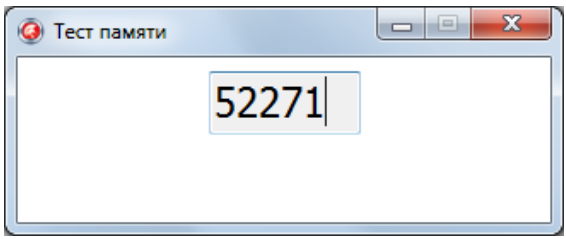


Рис. 1.59. Окно программы **Тест памяти**

```
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.
```

54. Усовершенствуйте программу **Тест памяти**, сделайте ее "интеллектуальной". Сначала программа 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.

55. 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. 1.60). Если картинки в открытых клетках одинаковые, то эти

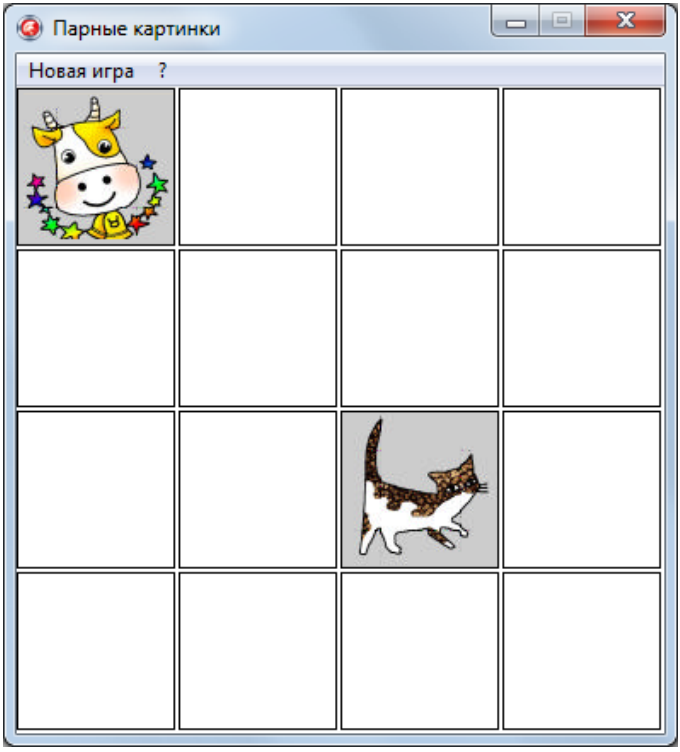


Рис. 1.60. Игровое поле программы **Парные картинки**

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.

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

В приведенной реализации игры все картинки квадратные и находятся в одном файле (рис. 1.61). 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. 1.62.

Рис. 1.61. Все картинки находятся в одном файле

```
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;
```

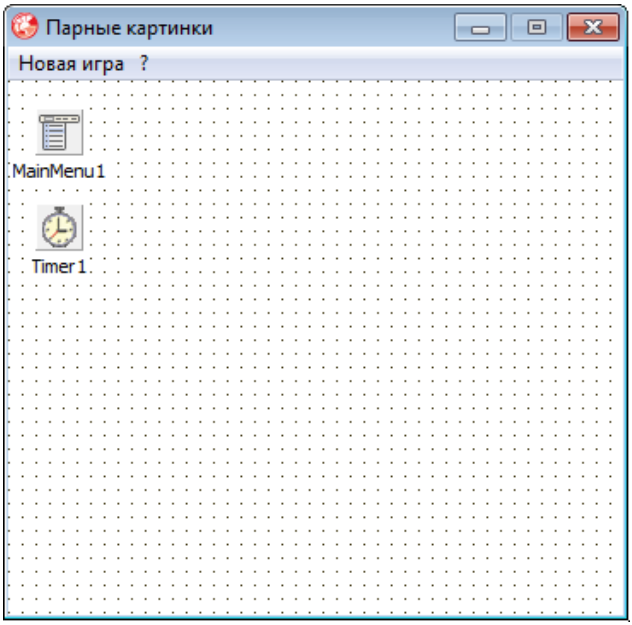


Рис. 1.62. Форма программы **Парные картинки**

```
  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}

// конструктор 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;

// 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;

// 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.
```

56. 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. 1.63). 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. Вид формы программы приведен на рис. 1.64.

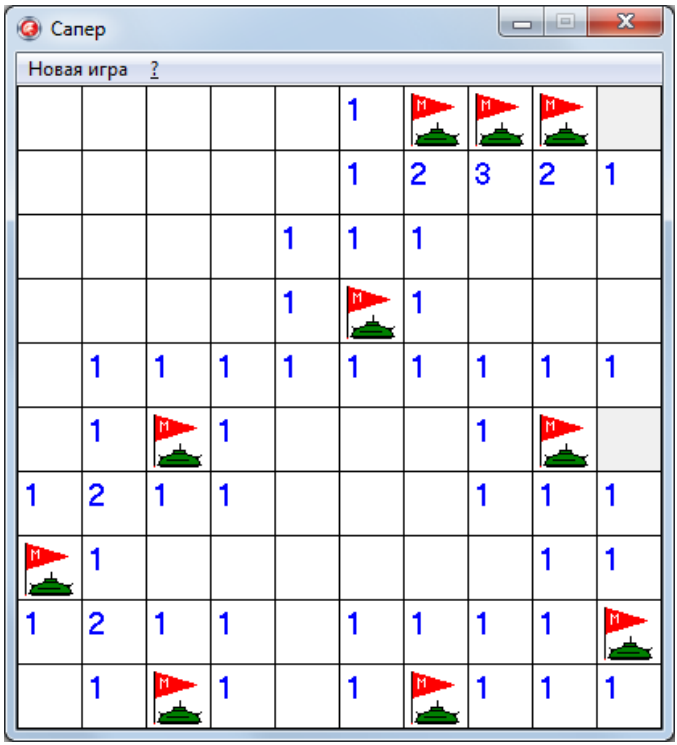


Рис. 1.63. Вид окна программы в конце игры — все мины найдены

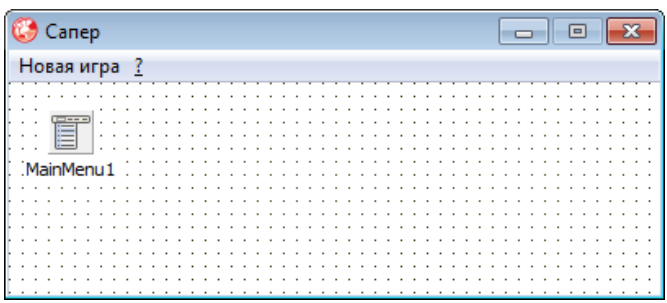


Рис. 1.64. Главная форма программы **Сапер**

```
unit saper_;
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


Procedure NewGame(); forward; // генерирует новое поле
// 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 := clBtnFace;
             Canvas.Rectangle(x-1,y-1,x+W,y+H);
             exit;
           end;

       if Pole[row,col] < 100  then
        begin
          Canvas.Brush.Color := clBtnFace;  // не открытые -
                                            // серые
          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;       // количество 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 точек флажка
       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 М
            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
    ClientHeight := H*MR + 1;
    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 := clBtnFace;
               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;
                  // поставили флаг
                  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.
```

57. Напишите программу **Будильник**. После того как пользователь установит время сигнала, задаст текст напоминания и сделает щелчок на кнопке **OK** (рис. 1.65), окно программы должно исчезнуть с экрана. В установленное время на экране должно появиться окно с сообщением (рис. 1.66). Появление окна должно сопровождаться звуковым сигналом.

```
unit Alarm2_1_;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Dialogs, StdCtrls,
  ExtCtrls, ComCtrls, DateUtils;
```

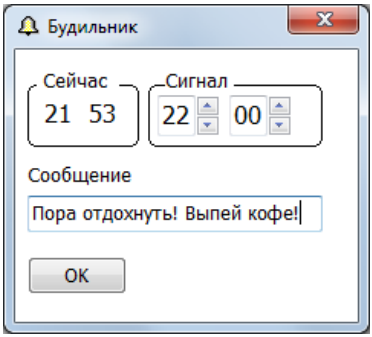


Рис. 1.65. Программа **Будильник**

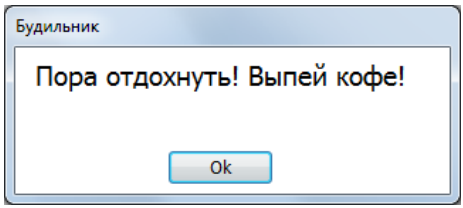


Рис. 1.66. Сообщение программы **Будильник**

```
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.
```

58. Напишите программу **Будильник**. После того как пользователь установит время сигнала и сделает щелчок на кнопке **OK** (рис. 1.67), окно программы должно закрыться, а ее значок появится на системной панели (рис. 1.68). В заданный момент

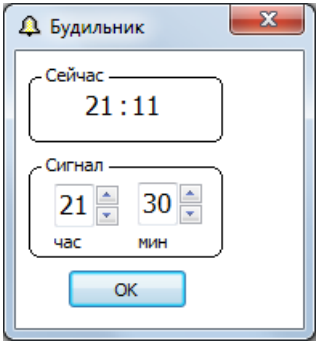


Рис. 1.67. Окно программы **Будильник**

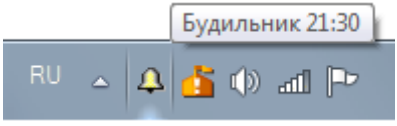


Рис. 1.68. Значок программы **Будильник** во время ее работы отображается на системной панели

времени окно программы должно появиться на экране. 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;

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);
  private
    { Private declarations }
```

```
    // **** эти объявления вставлены сюда вручную
    procedure CreateTrayIcon(n: integer; Tip: String);
    procedure DeleteTrayIcon(n: integer);
    procedure SetSound;
    // ***************************************

  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
const
    WM_MYTRAYNOTIFY = WM_USER + 123;

// 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);
       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;

// помещает картинку на System Tray
procedure TForm1.CreateTrayIcon(n: integer; Tip: string);
var
    nidata: TNotifyIconData;
begin
    // заполним структуру nidata,
    // поля которой определяют значок
    // в 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;
        StrPCopy(szTip,Tip);  // копируем Ansi-строку
                              // в nul-terminated-строку
    end;
    Shell_NotifyIcon(NIM_ADD, @nidata);
end;

// удаляет картинку из System Tray
procedure TForm1.DeleteTrayIcon(n: integer);
```

```
var
    nidata: TNotifyIconData;
begin
    // заполним структуру nidata,
    // поля которой определяют значок
    // в 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;

// показать время сигнала будильника — часы
procedure TForm1.UpDown1Changing(Sender: TObject;
                                 var AllowChange: Boolean);
begin
      if UpDown1.Position < 10
        then Edit1.Text := '0' + IntToStr(UpDown1.Position)
        else Edit1.Text := IntToStr(UpDown1.Position);
end;

// показать время сигнала будильника — минуты
procedure TForm1.UpDown2Click(Sender: TObject;
                              Button: TUDBtnType);
begin
    if UpDown2.Position < 10
        then Edit2.Text:= '0' + IntToStr(UpDown2.Position)
        else Edit2.Text := 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 в System Tray
    CreateTrayIcon(1,'Будильник '+
       Edit1.Text+':'+Edit2.Text);

    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.
```

59. Напишите программу **Экзаменатор**, позволяющую автоматизировать процесс проверки знаний (тестирования). Выбор ответа из нескольких возможных вариантов должен осуществляться с помощью переключателя (рис. 1.69).

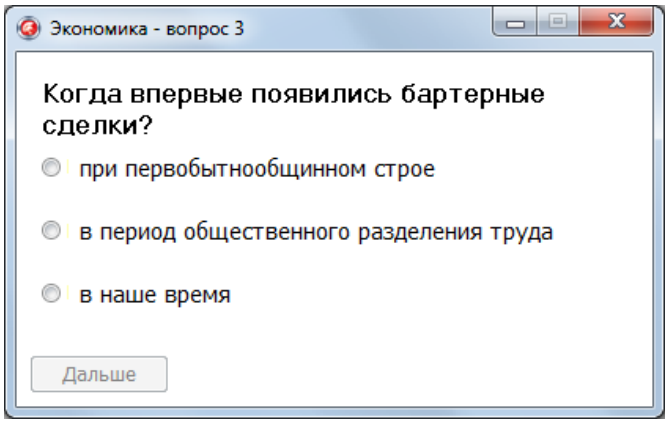


Рис. 1.69. Окно программы **Экзаменатор**

Тест — последовательность вопросов и соответствующих им вариантов ответа, должен находиться в текстовом файле. Количество вопросов теста не ограничено. Программа тестирования должна получать имя файла теста из командной строки.

Далее приведен пример файла теста. Первая строка — это название теста. Потом следует описание четырех уровней оценки. Для каждого уровня задается сообщение и количество правильных ответов, необходимых для его достижения. Далее следуют вопро-

сы. Каждый вопрос представляет собой текст вопроса, за которым следуют три варианта ответа. После каждого ответа указывается количестов баллов, добавляемое к общей оценке, в случае выбора этого варианта ответа (в простейшем случае за выбор правильного ответа к общей сумме добавляется 1, за выбор неправильного — 0). Необходимо обратить внимание на то, что вопрос и варианты ответа в файле теста представлены одной строкой.

```
Экономика
Вы правильно ответили на все вопросы. Оценка — ОТЛИЧНО!
6
На некоторые вопросы вы ответили неверно. Оценка — ХОРОШО.
5
На многие 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)
  title: string;   // название теста
  nq: integer;     // количество вопросов в тесте
  right: integer; // количество правильных ответов

  level: array[1..4] of integer;// кол-во правильных ответов,
                        // необходимое для достижения уровня
  mes: array[1..4] of string;      // сообщение об оценке

  buf: string;

// читает из файла вопрос, варианты ответа
// и выводит их в поля формы
function NextQw : boolean;
begin
  if not EOF(f) then
  begin
     // счетчик общего количества вопросов
    nq:= nq + 1;

    Form1.Caption := Title + ' — вопрос ' + IntToStr(nq);
    // 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);

    // кнопка "Дальше" недоступна,
    // пока не выбран один из вариантов ответа
    Form1.Button1.Enabled := False;

    // ни одна из радиокнопок не выбрана
    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 файл 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);
  title := 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 Далее теперь доступна
end;

procedure TForm1.RadioButton2Click(Sender: TObject);
begin
  Button1.Enabled := True;
end;

procedure TForm1.RadioButton3Click(Sender: TObject);
begin
  Button1.Enabled := True;
end;

end.
```

60. Напишите программу тестирования, в которой вопрос может сопровождаться иллюстрацией, а количество вариантов ответа к каждому вопросу может быть от двух до четырех. Пример окна программы приведен на рис. 1.70.

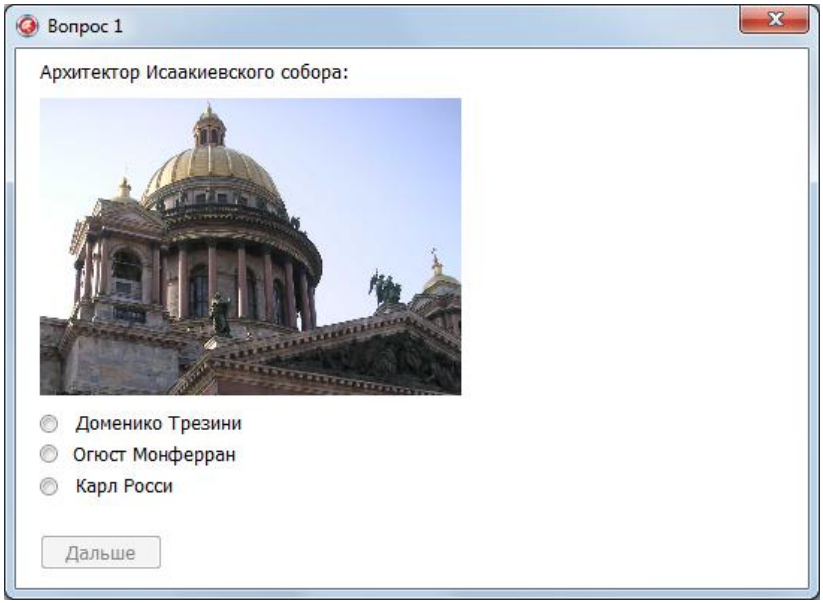


Рис. 1.70. Программа **Экзаменатор**.
Испытуемый должен выбрать правильный ответ

Вопросы теста должны находиться в текстовом файле, имя которого будет 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 теста выводится в заголовке стартового окна программы тестирования.

Раздел заголовка содержит общую информацию о тесте, например, о его назначении. Заголовок представляет собой одну строку (абзац) текста.

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 (оценка) 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 в следующей строке находятся три числа. Первое число — количество вариантов ответа, второе — номер правильного ответа, третье — признак наличия иллюстрации (если вопрос сопровождается иллюстрацией, то третье число — единица, если иллюстрации нет — ноль). Если к вопросу есть иллюстрация, то в следующей строке находится имя файла иллюстрации. Далее следуют варианты ответа.

Вот пример вопроса:

```
Архитектор Исаакиевского собора:
3 2 1
is.jpg
Доменико Трезини
Огюст Монферран
Карл Росси
```

В приведенном примере к вопросу дается три варианта ответа, правильным является второй ответ, вопрос сопровождается иллюстрацией.

Вид формы программы тестирования приведен на рис. 1.71.

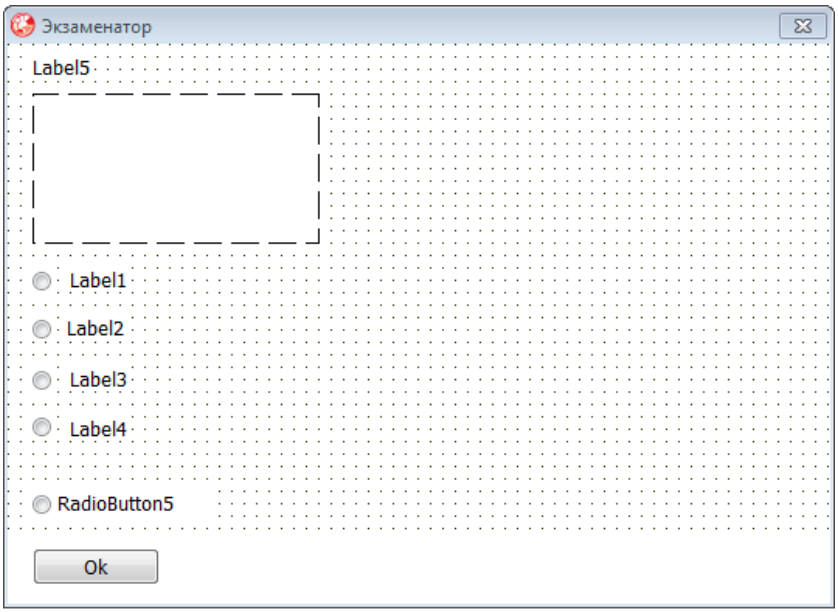


Рис. 1.71. Форма программы **Экзаменатор**

```
//  Универсальная программа тестирования.
//  (c) Культин Н.Б., 2003 — 2012
//
//  Тест загружается из файла, имя которого указано
//  в команде запуска программы.

unit ExamMainForn;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls,
  jpeg; // обеспечивает отображение JPG-иллюстраций
```

```
type
  TForm1 = class(TForm)

    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;

    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioButton3: TRadioButton;
    RadioButton4: TRadioButton;
    RadioButton5: TRadioButton;

    Image1: TImage;
    Button1: TButton;
    Panel1: TPanel;

    procedure FormActivate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure RadioButtonClick(Sender: TObject);

  private
    // Эти объявления вставлены сюда вручную
    procedure Info;
    function VoprosToScr: boolean;
    procedure ShowPicture; // выводит иллюстрацию
    procedure ResetForm;   // очистка формы перед
                           // выводом вопроса
    procedure Itog;        // результат тестирования
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation
{$R *.DFM}

var
  f:TextFile;
  fn:string; // имя файла вопросов

  level:array[1..4] of integer; // сумма, соответствующая
                                // уровню
  mes:array[1..4] of string;  // сообщение, соответствующее
                              // уровню

  vopr: record
    text: string;  // вопрос
    src: string;   // иллюстрация
    otv: array[1..4] of string; // варианты ответа
    notv: integer;  // количество вариантов ответа
    right: integer; // номер правильного ответа
  end;

  cv: integer = 0; // номер вопроса
  otv : integer;   // номер выбранного ответа
  sum: integer;

procedure TForm1.FormActivate(Sender: TObject);
  begin
     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;

          // ключ /t активизирует режим отладки/обучения
          if ParamStr(2) = '/t' then
              Label6.Visible := True;

          ResetForm;
          Info;     // прочитать и вывести информацию о тесте
          GetLevel; // прочитать информацию об уровнях оценок
     end;
end;

// информация о тесте
procedure Tform1.Info;
var
   s: string;
begin
   readln(f,s);
   Form1.Caption := s;
   readln(f,s);
   Label5.caption:=s;
end;

// прочитать из файла информацию об оценках
Procedure GetLevel;
var
   i:integer;
begin
     for i:=1 to 4 do
     begin
         readln(f,level[i]); // оценка
         readln(f,mes[i]);   // сообщение

     end;
end;
```

```
// отображение 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.Height > h
        then Image1.Height:=h
        else Image1.Height:= Image1.Picture.Height;
  if Image1.Picture.Width > w
        then Image1.Width:=w
        else Image1.Width:=Image1.Picture.Width;

  Image1.Visible := True;
end;

// вывести вопрос
function TForm1.VoprosToScr: boolean;
  var
   i: integer;
```

```
  p: integer;
begin

    if EOF(f) then
    begin
      // достигнут конец файла
      VoprosToScr := False;
      exit;
    end;

    readln(f, vopr.text); // прочитать вопрос

    if  vopr.text = '*' then
    begin
      // признак конца теста
      VoprosToScr := False;
      exit;
    end;

    cv := cv + 1;
    caption:='Вопрос ' + IntToStr(cv);
    Label5.caption:= vopr.text;  // вывести вопрос

    // прочитать информацию об ответе:
    // количество вариантов, номер правильного ответа
    // и признак наличия иллюстрации
    readln(f,vopr.notv,vopr.right, p);
    if p <> 0 then // есть иллюстрация
    begin
       readln(f,vopr.src);  // имя файла иллюстрации
       Image1.Tag:=1;
         try
           { Иллюстрацию прочитаем, но выведем только после
             того, как прочитаем альтернативные ответы
             и определим максимально возможный размер
             области формы, который можно использовать
             для отображения иллюстрации.
           }
```

```
        Image1.Picture.LoadFromFile(vopr.src);
        except
          on E:EFOpenError do
             Image1.Tag:=0;
      end

end
else Image1.Tag := 0; // нет иллюстрации

// читаем варианты ответа
for i:= 1 to vopr.notv do
   readln(f,vopr.otv[i]);

for i:= 1 to vopr.notv do
   case i of
       1: Label1.caption:= vopr.otv[i];
       2: Label2.caption:= vopr.otv[i];
       3: Label3.caption:= vopr.otv[i];
       4: Label4.caption:= vopr.otv[i];
   end;

// здесь прочитана иллюстрация и альтернативные ответы

// текст вопроса уже выведен
if Image1.Tag =1  // есть иллюстрация к вопросу
   then ShowPicture;

// вывод альтернативных ответов
if Label1.Caption <> ''
then begin
   if 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 Label2.Caption <> ''
then begin
   Label2.top:=Label1.top+ Label1.height+5;
```

```
        RadioButton2.top:=Label2.top;
        Label2.visible:=TRUE;
        RadioButton2.Visible:=TRUE;
      end;

      if Label3.Caption <> ''
      then begin
        Label3.top:=Label2.top+ Label2.height+5;
        RadioButton3.top:=Label3.top;
        Label3.visible:=TRUE;
        RadioButton3.Visible:=TRUE;
      end;

      if Label4.Caption <> ''
      then begin
        Label4.top:=Label3.top+ Label3.height+5;
        RadioButton4.top:=Label4.top;
        Label4.visible:=TRUE;
        RadioButton4.Visible:=TRUE;
      end;

      Label6.Caption := '';
      VoprosToScr := True;
  end;

Procedure TForm1.ResetForm;
  begin
      // сделать невидимыми все метки и радиокнопки
      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 вопросов: '+ IntToStr(cv) + #13 +
        'Правильных ответов: ' + IntToStr(sum);
   i:=1;
   while (sum < level[i]) and (i<4) do
         i:=i+1;
   buf:=buf+ #13+ mes[i];
   Label5.Top:= 20;
   Label5.caption:=buf;
  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
                if otv = vopr.right then
                   sum := sum + 1;
                RadioButton5.Checked:=TRUE;
                Button1.Enabled:=False;
                ResetForm;
                if not VoprosToScr then
                      begin
                         closefile(f);
                         Button1.caption:='Ok';
                         Form1.caption:='Результат';
                         Button1.tag:=2;
                         Button1.Enabled:=TRUE;
                         Label6.Visible := False;
                         Itog; // вывести результат
                      end;
            end;
        2: begin // завершение работы
             Form1.Close;
            end;
     end;
end;

// Процедура обработки события Click
// 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;
  Label6.Caption := 'Выбран ответ: ' + IntToStr(otv) +
           ' Правильный ответ: ' + IntToStr(vopr.right);
end;

end.
```

БАЗЫ ДАННЫХ

Общие замечания

- Для доступа к базе данных можно использовать различные технологии, например ADO.
- Доступ к базе данных на основе технологии ADO обеспечивают компоненты `ADOConnection` и `ADODataSet`.
- Отображение и редактирование информации в табличной форме обеспечивает компонент `DBGrid`.
- Отображение и редактирование отдельных полей обеспечивают компоненты `DBEdit`, `DBMemo`.
- Связь компонентов доступа к данным и отображения данных обеспечивает компонент `DataSource`.
- Создать базу данных и наполнить ее информацией можно с помощью соответствующей СУБД, например Microsoft Access.

61. Напишите программу работы с базой данных Microsoft Access **Контакты** (рис. 1.72).

Для создания базы данных используйте Microsoft Access. Характеристики полей таблицы `contacts` приведены в табл. 1.1, форма программы — на рис. 1.73, значения свойств компонентов — в табл. 1.2.

Таблица 1.1. *Поля таблицы* `contacts`

Поле	Тип	Размер	Информация
Name	Текстовый	50	Имя
Phone	Текстовый	50	Телефон
Email	Текстовый	50	Электронная почта
Image	Текстовый	30	Фотография (имя файла)

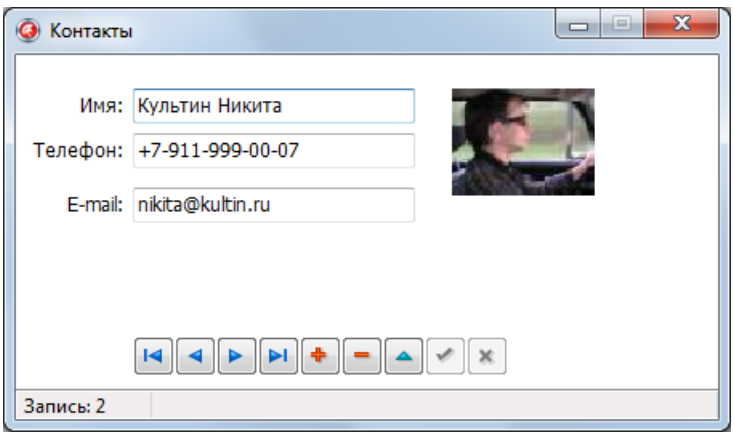


Рис. 1.72. Окно программы работы с базой данных **Контакты**

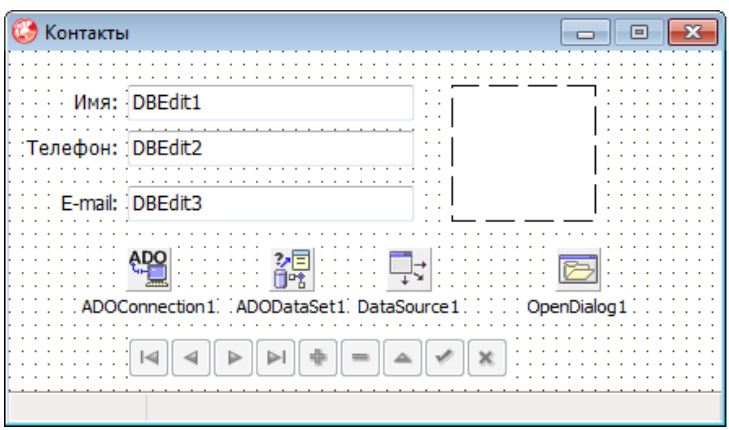


Рис. 1.73. Форма программы работы с базой данных **Контакты**

Таблица 1.2. *Значения свойств компонентов*

Свойство	**Значение**
Name	ADOConnection1
ConnectionString	Provider=Microsoft.Jet.OLEDB.4.0; Data Source=contacts.mdb; Persist Security Info=False
LoginPrompt	False
Name	ADODataSet1
Connection	ADOConnection1
CommandText	SELECT * FROM contacts
Name	DataSource1
DataSet	ADODataSet11
Name	DBEdit1
DataSource	DataSource1
DataField	Name
AutoSelect	False
ReadOnly	True
Name	DBEdit2
DataSource	DataSource1
DataField	Phone
AutoSelect	False
ReadOnly	True
Name	DBEdit3
DataSource	DataSource1
DataField	Email
AutoSelect	False
ReadOnly	True
Name	DBNavigator1
DataSource	DataSource1
VisibleButtons.bnRefresh	False
ShowHint	True

Таблица 1.2 *(окончание)*

Свойство	Значение
Hints	Первая запись Предыдущая запись Следующая запись Последняя запись Добавить запись Удалить запись Редактировать запись Сохранить изменения Отменить изменения
Name	Image1
Proportional	True
Stretch	True
Enabled	False
Hint	Щелкните, чтобы изменить
Center	True
ShowHint	False

```
unit MainForm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, DB, ADODB, Grids, DBGrids,
  ExtCtrls, DBCtrls, StdCtrls, ComCtrls, Mask, ExtDlgs;

type
  TForm1 = class(TForm)
    ADOConnection1: TADOConnection;
    ADODataSet1: TADODataSet;
    DataSource1: TDataSource;
```

```
    Label1: TLabel;
    Label2: TLabel;
    DBNavigator1: TDBNavigator;
    StatusBar1: TStatusBar;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    Image1: TImage;
    OpenDialog1: TOpenDialog;
    DBEdit3: TDBEdit;
    Label3: TLabel;
    procedure FormClose(Sender: TObject;
                        var Action: TCloseAction);
    procedure FormActivate(Sender: TObject);
    procedure ADODataSet1AfterScroll(DataSet: TDataSet);
    procedure Image1Click(Sender: TObject);
    procedure DBNavigator1Click(Sender: TObject;
                                Button: TNavigateBtn);
    procedure ADOConnection1BeforeConnect(Sender: TObject);
  private
    { Private declarations }
    aPath: string;
    procedure ShowImage(img: string); // отображает картинку
                                      // в поле Image1
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// ********************
uses jpeg,
     IniFiles,
     StrUtils;
```

```
// начало работы программы
procedure TForm1.FormActivate(Sender: TObject);
begin
  aPath :=''; // предполагается, что БД и каталог Images
              // находятся в каталоге программы
  try
    ADOConnection1.Open;
    ADODataSet1.Open;
    StatusBar1.Panels[0].Text := ' Запись: 1';
  except
    on e:Exception do begin
        DBEdit1.Enabled := False;
        DBEdit2.Enabled := False;
        DBNavigator1.Enabled := False;
        MessageDlg('Нет файла БД ' +
           aPath + 'contacts.mdb',mtError,[mbOk],0);
    end;
  end;
end;

// событие возникает после перехода к другой записи
procedure TForm1.ADODataSet1AfterScroll(DataSet: TDataSet);
var
    img: string;
begin
    if ADODataSet1.RecNo <> -1 then
    begin
       StatusBar1.Panels[0].Text := ' Запись: ' +
                            IntToStr(ADODataSet1.RecNo);

       if ADODataSet1.FieldValues['img'] <> Null then
            img := ADODataSet1.FieldValues['img']
       else
            img := '';
       ShowImage(img);
    end
    else
       StatusBar1.Panels[0].Text := ' Новая запись'
end;
```

```
// отображает иллюстрацию
Procedure TForm1.ShowImage(img: string);
begin
  if img = '' then
    img := 'nobody.jpg';
  try
     Image1.Picture.LoadFromFile(aPath+'images\'+img);
  finally

  end;
end;

// щелчок на кнопке компонента DBNavigator
procedure TForm1.DBNavigator1Click(Sender: TObject;
                                   Button: TNavigateBtn);
begin
  case Button of
    nbInsert, nbDelete, nbEdit:
          begin
             DBEdit1.ReadOnly := False;
             DBEdit2.ReadOnly := False;
             DBEdit3.ReadOnly := False;
             Image1.Enabled := True;
             Image1.ShowHint := True;
             if Button = nbInsert then
                ShowImage('nobody.jpg');
          end ;
    nbPost, nbCancel:
          begin
             DBEdit1.ReadOnly := True;
             DBEdit2.ReadOnly := True;
             DBEdit3.ReadOnly := True;
             Image1.Enabled := False;
             Image1.ShowHint := False;
          end ;
  end;
end;

// щелчок в поле компонента Image (выбор картинки)
procedure TForm1.Image1Click(Sender: TObject);
```

```
var
  nFileName: string;
begin
  OpenDialog1.FileName := '*.jpg';
  if OpenDialog1.Execute then
  begin
      // пользователь выбрал изображение
      nFileName := ExtractFileName(OpenDialog1.FileName);
      CopyFile( PChar(OpenDialog1.FileName), PChar(aPath +
'images\'+ nFileName), false);
      ShowImage(nFileName);
      ADODataSet1.FieldValues['image'] := nFileName;
  end;
end;

// завершение работы программы
procedure TForm1.FormClose(Sender: TObject;
                           var Action: TCloseAction);
begin
   if ADODataSet1.State = dsEdit then  // пользователь
                               // не завершил редактирование
    begin
      // записать редактируемую запись
      ADODataset1.UpdateBatch(arCurrent);
    end;
end;

procedure TForm1.ADOConnection1BeforeConnect(Sender: TObject);
var
  p1,p2: integer;
  IniFile: TIniFile;
  fn: string;          // имя INI-файла
  st: string;          // строка соединения
begin

    // загрузить строку соединения из INI-файла
    // INI-файл должен находиться в том же каталоге,
    // что и EXE-файл
    // и его имя совпадает с именем EXE-файла
    p1 := Pos('.exe', Application.ExeName);
```

```
    fn := Copy(Application.ExeName, 1, p1-1) + '.ini';
    IniFile := TIniFile.Create(fn);

    // прочитаем из INI-файла
    // имя каталога, в котором находится БД
    // ключ aPath находится в секции data
    aPath := IniFile.ReadString('data','aPath','');

    if aPath = '' then
      MessageDlg('Нет файла: '+ fn ,mtError,[mbOk],0);

    st := ADOConnection1.ConnectionString;
    p1 := Pos('Data Source',st);
    p2 := PosEx(';',st,p1);

    Delete(st,p1,p2-p1);
    Insert('Data Source='+ aPath+ 'contacts.mdb',st,p1);

    ADOConnection1.ConnectionString := st;

end;

end.
```

62. Напишите программу работы с базой данных **Записная книжка**. Для создания базы данных используйте Microsoft Access. Характеристики полей таблицы `Phones` приведены в табл. 1.3, форма программы — на рис. 1.74, значения свойств компонентов — в табл. 1.4.

Таблица 1.3. *Поля таблицы* `Phones`

Поле	Тип	Размер	Информация
Name	Текстовый	50	Имя
Phone	Текстовый	30	Телефон

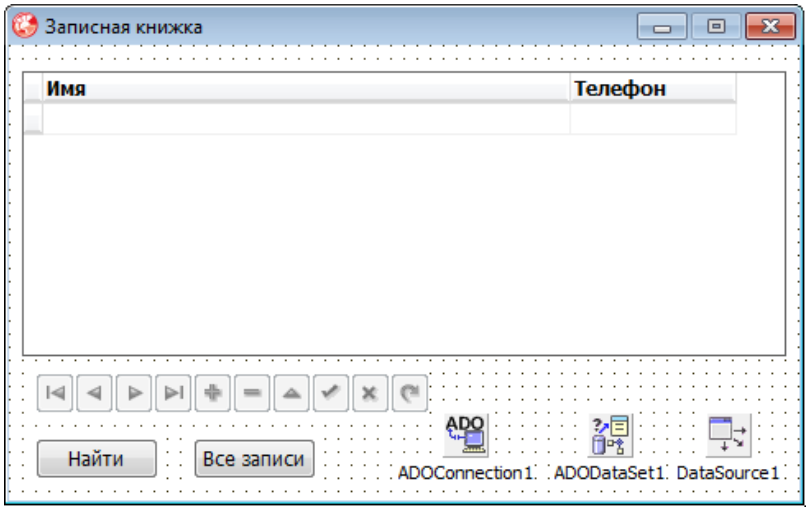


Рис. 1.74. Форма программы **Записная книжка**

Таблица 1.4. *Свойства компонентов*

Свойство	**Значение**
Name	ADOConnection1
ConnectionString	Provider=Microsoft.Jet.OLEDB.4.0; Data Source=D:\Database\phones.mdb; Persist Security Info=False
LoginPrompt	False
Name	ADODataSet1
Connection	ADOConnection1
CommandText	SELECT * FROM Phones ORDER BY Name
Name	DataSource1
DataSet	ADODataSet1
Name	DBNavigator1
DataSource	DataSource1
VisibleButtons.bnRefresh	False
ShowHint	True

Таблица 1.4 *(окончание)*

Свойство	Значение
Hints	Первая запись Предыдущая запись Следующая запись Последняя запись Добавить запись Удалить запись Редактировать запись Сохранить изменения Отменить изменения
Name	DBGrid1
DataSource	DataSource1
Font.Name	Tahoma
Font.Size	9
Columns[0].FieldName	Name
Columns[0].TitleCaption	Имя
Columns[0].Title.Width	320
Columns[0].Title.Font.Style	fsBold
Columns[1].FieldName	Phone
Columns[1].TitleCaption	Телефон
Columns[1].Width	100
Columns[1].Title.Font.Style	fsBold

```
unit ntbk;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, DB, ADODB, Grids, DBGrids,
  ExtCtrls, DBCtrls, StdCtrls;
```

```
type
  TForm1 = class(TForm)
    ADOConnection1: TADOConnection;
    ADODataSet1: TADODataSet;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Button1: TButton;
    Button2: TButton;
    DBNavigator1: TDBNavigator;
    procedure Button2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormClose(Sender: TObject;
                        var Action: TCloseAction);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// начало работы программы
procedure TForm1.FormActivate(Sender: TObject);
begin
  try
    ADOConnection1.Open;
    ADODataSet1.Open;
  except
    on e:Exception do begin
        DBGrid1.Enabled := False;
        MessageDlg(e.Message,mtError,[mbOk],0);
    end;
  end;
end;
```

```
// щелчок на кнопке Найти
procedure TForm1.Button1Click(Sender: TObject);
var
   st: string;
begin

  st := InputBox('Поиск','Введите имя или фамилию','');

  if st <> '' then
    // пользователь ввел критерий запроса
    begin
      // фильтр
      ADODataSet1.Filtered := False;
      ADODataSet1.Filter :='name Like ''%' + st + '%''';
      ADODataSet1.Filtered := True;
      if ADODataSet1.RecordCount = 0 then
        begin
          // В базе данных нет записей,
          // удовлетворяющих критерию запроса.
          ADODataSet1.Filtered := False;
          ShowMessage('В БД нет записей, удовлетворяющих
                       критерию запроса.');
        end;

    end;
end;

// щелчок на кнопке Все записи
procedure TForm1.Button2Click(Sender: TObject);
begin
  ADODataSet1.Filtered := False;

end;

// завершение работы программы
procedure TForm1.FormClose(Sender: TObject;
                           var Action: TCloseAction);
begin
   if DBGrid1.EditorMode then  // пользователь не завершил
                               // редактирование
```

```
    begin
      DBGrid1.EditorMode := False;
      // записать редактируемую запись
      ADODataset1.UpdateBatch(arCurrent);
    end;
end;

end.
```

63. Напишите программу работы с базой данных **Ежедневник** (для создания базы данных используйте Microsoft Access). Сразу после запуска программа должна вывести список дел, запланированных на сегодня и ближайшие дни. Характеристики полей таблицы `Tasks` приведены в табл. 1.5, форма программы — на рис. 1.75, значения свойств компонентов — в табл. 1.6.

Таблица 1.5. *Поля таблицы* `Tasks`

Поле	Тип	Размер	Информация
Task	Текстовый	50	Задача
Date	Дата/время	—	Дата, на которую задача запланирована

Таблица 1.6. *Свойства компонентов*

Свойство	Значение
Name	ADOConnection1
ConnectionString	Provider=Microsoft.Jet.OLEDB.4.0; Data Source=tasks.mdb; Persist Security Info=False
LoginPrompt	False
Name	ADOTable1
Connection	ADOConnection1
TableName1	Tasks
Name	DataSource1
DataSet	ADODataSet1

***Таблица 1.6** (окончание)*

Свойство	Значение
Name	DBNavigator1
DataSource	DataSource1
VisibleButtons.bnRefresh	False
Name	DBGrid1
DataSource	DataSource1
Font.Name	Tahoma
Font.Size	9
Columns[0].FieldName	Date
Columns[0].TitleCaption	Когда
Columns[0].Title.Width	100
Columns[1].FieldName	Task
Columns[1].TitleCaption	Что
Columns[1].Width	3200

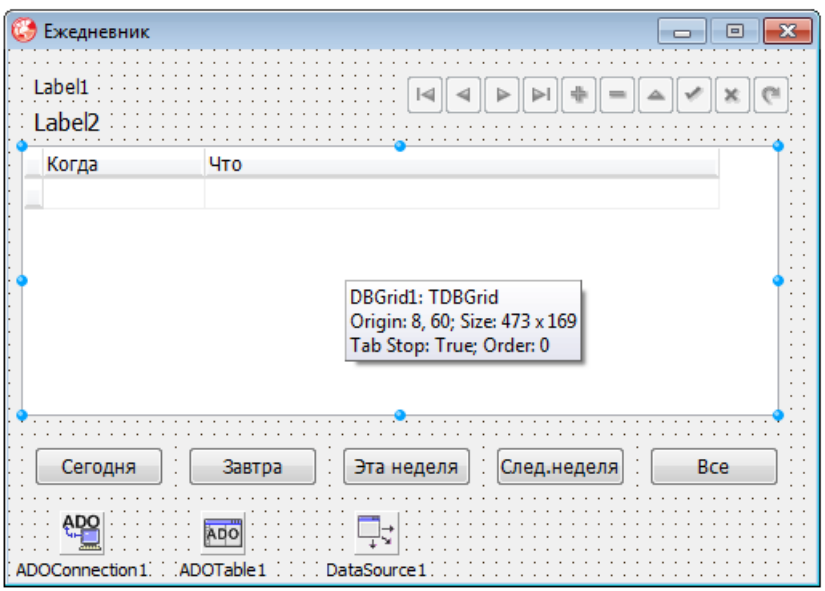


Рис. 1.75. Форма программы **Ежедневник**

```
// Главный модуль программы Организатор.
// Чтобы его увидеть, выберите в меню
// Project команду View Source.
program tasks;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  SysUtils,
  Dialogs,
  DateUtils,
  DBTables;

{$R *.res}

var
  Present: TDateTime;       // сегодня
  NextDay: TDateTime;       // следующий день
  Year, Month, Day : Word;  // год, месяц, день

begin
  Application.Initialize;
  Application.Title := 'Ежедневник';
  Application.CreateForm(TForm1, Form1);

// определим текущую дату
  Present:= Now; // Now — функция,
                 // возвращает текущую дату и время
  DecodeDate(Present, Year, Month, Day);

  // для пятницы и субботы следующим днем
  // будем считать понедельник
  case DayOfWeek(Present) of
    6:   NextDay := IncDay(Present,3);
    7:   NextDay := IncDay(Present,2);
    else NextDay := IncDay(Present,1)
  end;
```

```
  Form1.ADOTable1.Open;

  // выбрать задачи, запланированные
  // на сегодня и ближайшие дни
  Form1.ADOTable1.Filter :=
   '(Date >= #'+ FormatDateTime('dd/mm/yyyy',Present)+ '#)
    and ('+ 'Date <= #'+
    FormatDateTime('dd/mm/yyyy',NextDay)+'#)';
  Form1.ADOTable1.Filtered := True;

  Application.Run;

end.

{
 В окне программы сразу после ее запуска отображается
 список дел, запланированных на сегодня и ближайшие дни.
 Ближайшим днем для понедельника, вторника, среды, четверга
 и воскресенья является следующий день,
 для пятницы и субботы — воскресенье.
 Выбор осуществляется путем фильтрации записей.
 Настройку фильтра в начале работы программы выполняет
 главная процедура приложения (чтобы ее увидеть,
 выберите в меню Project команду View Source).
}
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, DB, Grids, DBGrids, DBTables,
  StdCtrls, DBCtrls, ExtCtrls, ADODB;

type
  TForm1 = class(TForm)
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Label1: TLabel;
```

```
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Label2: TLabel;
    ADOConnection1: TADOConnection;
    ADOTable1: TADOTable;
    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 := 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
    Label2.Caption := 'Сегодня';
    st:= FormatDateTime('dd/mm/yyyy',Now);

    ADOTable1.Filtered := False;
    ADOTable1.Filter := 'Date = #' +st+ '#';
    ADOTable1.Filtered := True;
    if ADOTable1.RecordCount = 0 then
     ShowMessage('На сегодня никаких дел не запланировано.');
end;

// завтра
procedure TForm1.Button3Click(Sender: TObject);
var
  Present,              // сегодня
  Tomorrow: TDateTime;  // завтра

begin
  Label2.Caption := 'Завтра';
  Present:= Now();
  Tomorrow := IncDay(Present);  // завтра
```

```
  ADOTable1.Filtered := False;
  ADOTable1.Filter := 'Date = #'+
              FormatDateTime('dd/mm/yyyy',Tomorrow)+ ' #';
  ADOTable1.Filtered := True;
  if ADOTable1.RecordCount = 0 then
    ShowMessage('На завтра никаких дел не запланировано.');
end;

// на этой неделе
procedure TForm1.Button4Click(Sender: TObject);
var
  Present: TDateTime;
  EndOfWeek: TDateTime;

begin
  Label2.Caption := 'На этой неделе';
  Present:= Now();
  EndOfWeek :=
StartOfAWeek(YearOf(Present),WeekOf(Present)+1);

  ADOTable1.Filtered := False;
  ADOTable1.Filter := '(Date >= #'+
            FormatDateTime('dd/mm/yyyy',Present)+'#)
            and ('+ 'Date <= #'+
            FormatDateTime('dd/mm/yyyy',EndOfWeek)+'#)';
  ADOTable1.Filtered := True;

  if ADOTable1.RecordCount = 0 then
 ShowMessage ('На эту неделю никаких дел не запланировано.');
end;

// на следующей неделе
procedure TForm1.Button1Click(Sender: TObject);
var
  Present: TDateTime;
  d1, d2: TDateTime;  // даты начала и конца следующей недели

 begin
  Form1.Label2.Caption := 'На следующей неделе';
  Present:= Now();
```

```
  d1 := StartOfAWeek(YearOf(Present),WeekOf(Present)+1);
  d2 := StartOfAWeek(YearOf(Present),WeekOf(Present)+2);

  ADOTable1.Filtered := False;
  ADOTable1.Filter :=
         '(Date >= #'+ FormatDateTime('dd/mm/yyyy',d1)+
         '#) and ('+
         'Date <= #'+ FormatDateTime('dd/mm/yyyy',d2)+'#)';
  ADOTable1.Filtered := True;

  if ADOTable1.RecordCount = 0 then
    ShowMessage('На следующую неделю никаких дел не
запланировано.');
end;

// показать все записи
procedure TForm1.Button5Click(Sender: TObject);
begin
        Label2.Caption := 'Все, что намечено сделать';
        ADOTable1.Filtered := False;
end;

end.
```

ПЕЧАТЬ СЧЕТА

64. Напишите программу, используя которую можно подготовить и распечатать счет на оплату покупки. Рекомендуемый вид формы программы приведен на рис. 1.76.

```
unit schet_;

interface

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

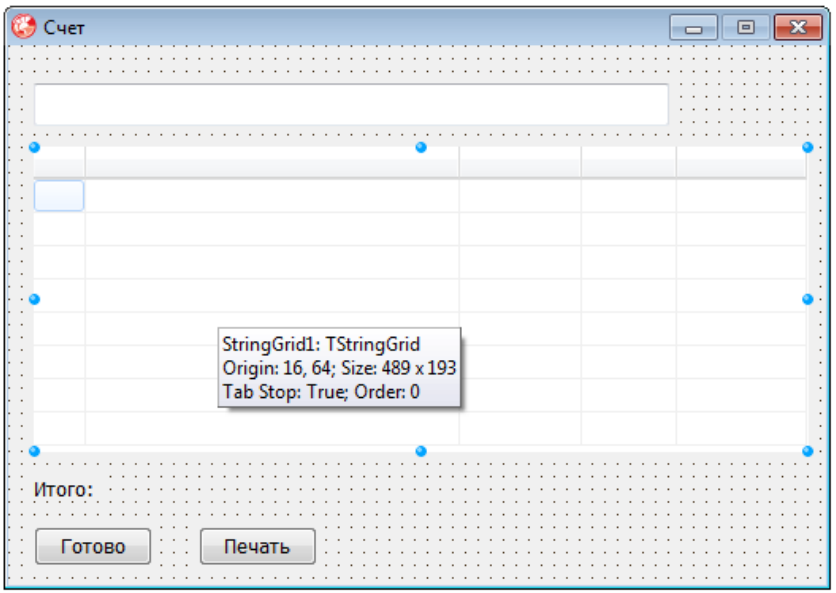


Рис. 1.76. Форма программы **Счет**

```
type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.dfm}

uses Printers;

// конструктор формы
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, умножить 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);

        px := x1;
        py := y2 + Round(20 * ky);
        Canvas.TextOut(px,py, Label1.Caption);

        EndDoc;  // закрыть печать
    end;
end;
```

```
// щелчок на кнопке Готово
procedure TForm1.Button1Click(Sender: TObject);
var
    i: integer;
    cena, kol, sum : real; // цена, количество, сумма
    itogo: real;  // итог

begin
    itogo := 0;

    for i := 1 to StringGrid1.RowCount do
    begin
        cena := 0;
        kol := 0;
        if StringGrid1.Cells[2,i] <> '' then
              cena := StrToFloat(StringGrid1.Cells[2,i]);
        if StringGrid1.Cells[3,i] <> '' then
              kol := StrToFloat(StringGrid1.Cells[3,i]);
        if  (cena <> 0) and (kol <> 0) then
            begin
                sum := cena * kol;
                itogo := itogo + sum;
                StringGrid1.Cells[4,i] :=
                         FloatToStrF(sum,ffCurrency,4,2);
            end;
     end;
     Label1.Caption := 'Итого: ' +
                         FloatToStrF(itogo,ffCurrency,4,2);
end;

end.
```

Часть 2

Справочник

Справочник содержит описание базовых компонентов, компонентов доступа к данным, а также описание часто используемых функций. Описание других компонентов можно найти в справочной системе среды разработки Delphi.

ФОРМА

Форма (объект типа `TForm`) является основой программы. Свойства формы (табл. 2.1) определяют вид окна программы.

Таблица 2.1. *Свойства формы (объекта* `TForm`*)*

Свойство	Описание
`Name`	Имя (идентификатор) формы. Используется для доступа к форме, ее свойствам и методам, а также для 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
`Height`	Высота формы
`Position`	Положение окна в момент первого его появления на экране (`poCenterScreen` — в центре экрана; `poOwnerFormCenter` — в центре родительского окна; `poDesigned` — положение окна определяют значения свойств `Top` и `Left`)

Таблица 2.1 *(окончание)*

Свойство	Описание
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 вообще отсутствовать (bsNone). Если у окна обычная граница, то во время работы программы пользователь может с помощью 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 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, используемый "по умолчанию" компонентами, находящимися на поверхности формы. Изменение свойства Font формы приводит к автоматическому изменению свойства Font компонента, располагающегося на поверхности формы. То есть компоненты наследуют свойство Font от формы (имеется возможность запретить наследование)
Canvas	Поверхность, на которую можно вывести графику

БАЗОВЫЕ КОМПОНЕНТЫ

В этом разделе приведено краткое описание базовых компонентов. Подробное описание этих и других компонентов можно найти в справочной системе.

Label

Компонент `Label` (рис. 2.1) предназначен для вывода текста на поверхность формы. Свойства компонента (табл. 2.2) определяют вид и расположение текста.

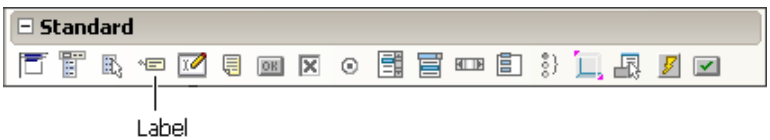


Рис. 2.1. Компонент `Label`

Таблица 2.2. *Свойства компонента* `Label`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Caption`	Текст, отображаемый в поле компонента
`Left`	Расстояние от левой границы поля вывода до левой границы формы
`Top`	Расстояние от верхней границы поля вывода до верхней границы формы
`Height`	Высота поля вывода
`Width`	Ширина поля вывода
`AutoSize`	Признак того, что размер поля определяется его содержимым
`WordWrap`	Признак того, что слова, которые не помещаются в текущей строке, автоматически переносятся на следующую строку (значение свойства `AutoSize` должно быть `False`)

Таблица 2.2 *(окончание)*

Свойство	Описание
Alignment	Задает способ выравнивания текста внутри поля. Текст может быть выровнен по левому краю (taLeftJustify), по центру (taCenter) или по правому краю (taRightJustify)
LayOut	Задает способ размещения текста по вертикали. Текст может быть прижат к верхней границе поля компонента (tlTop), к нижней границе (tlBottom) или размещен по центру (tlCenter)
Font	Шрифт, используемый для отображения текста. Уточняющие свойства определяют шрифт (Name), размер (Size), стиль (Style) и цвет символов (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.

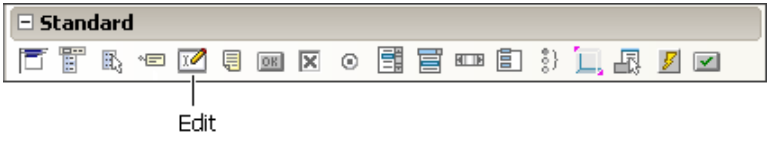


Рис. 2.2. Компонент Edit

Таблица 2.3. *Свойства компонента* `Edit`

Свойство	Описание
`Name`	Имя компонента. Используется 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.

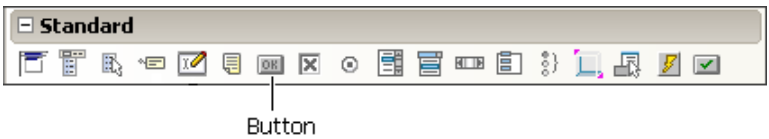


Рис. 2.3. Компонент `Button`

Таблица 2.4. *Свойства компонента* `Button`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Caption`	Текст на кнопке
`Left`	Расстояние от левой границы кнопки до левой границы формы
`Top`	Расстояние от верхней границы кнопки до верхней границы формы
`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, значение свойства `ShowHint` должно быть `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.

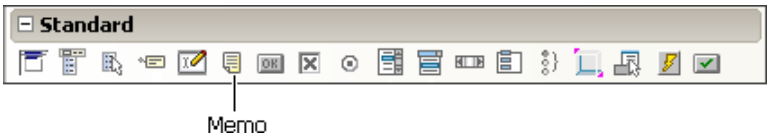


Рис. 2.4. Компонент `Memo`

Таблица 2.5. *Свойства компонента* `Memo`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента
`Text`	Текст, находящийся в поле **Memo** (свойство доступно только во время работы программы)
`Lines`	Массив строк, соответствующий содержимому поля. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля
`Lines.Count`	Количество строк текста в поле **Memo** (количество элементов в массиве `Lines`)
`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 (переключатель), состояние которой определяется состоянием других кнопок группы. Свойства компонента приведены в табл. 2.6.

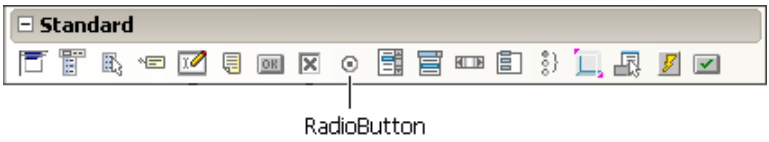


Рис. 2.5. Компонент `RadioButton`

Если в диалоговом окне надо организовать несколько групп переключателей, то каждую группу следует представить компонентом RadioGroup.

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 переключателя до левой границы формы
`Top`	Расстояние от верхней границы переключателя 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 (флажок). Свойства компонента приведены в табл. 2.7.

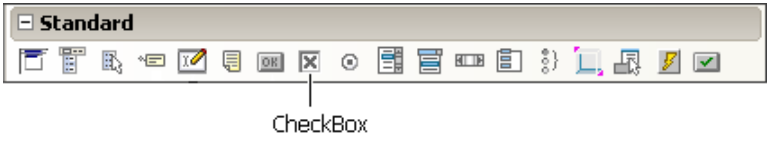


Рис. 2.6. Компонент `CheckBox`

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 значение `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
`Items.Count`	Количество элементов списка
`ItemIndex`	Номер выбранного элемента (элементы списка нумеруются с нуля). Если в списке ни один из элементов не выбран, то значение свойства равно –1 (минус один)
`Sorted`	Признак необходимости автоматической сортировки (`True`) списка после добавления очередного элемента
`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 выбора из списка. Свойства компонента приведены в табл. 2.9.

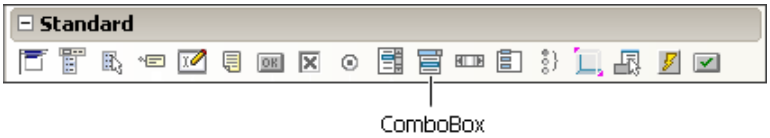


Рис. 2.8. Компонент `ComboBox`

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 –1 (минус один)
`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)
`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
`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

Таблица 2.10 *(окончание)*

Свойство	Описание
`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 JPEG, надо подключить модуль `JPEG` — включить в текст программы директиву `uses JPEG`). 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`, `Stretch` 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`
`Stretch`	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
`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 `Timer`. 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 `Timer`. Задается в миллисекундах
`Enabled`	Разрешение работы. Разрешает (значение `True`) или запрещает (значение `False`) генерацию события `Timer`

SpeedButton

Компонент `SpeedButton` (рис. 2.12) представляет собой кнопку, на поверхности которой находится картинка. Свойства компонента приведены в табл. 2.13.

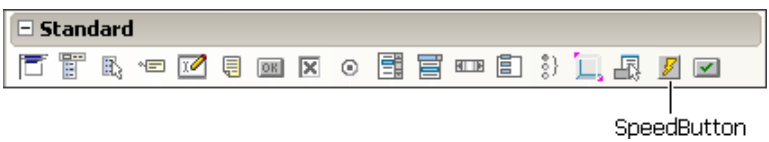


Рис. 2.12. Компонент `SpeedButton`

Таблица 2.13. *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. 2.13)

Таблица 2.13 *(окончание)*

Свойство	Описание
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. Чтобы кнопку можно было зафиксировать, значение свойства GroupIndex не должно быть равно нулю
Down	Идентификатор состояния кнопки. Изменить значение свойства можно, если значение свойства GroupIndex не равно 0
AllowAllUp	Свойство определяет возможность отжатия кнопки. Если кнопка нажата и значение свойства равно True, то кнопку можно отжать
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
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, значение свойства ShowHint должно быть True)
ShowHint	Разрешает (True) или запрещает (False) отображение подсказки при позиционировании указателя на кнопке

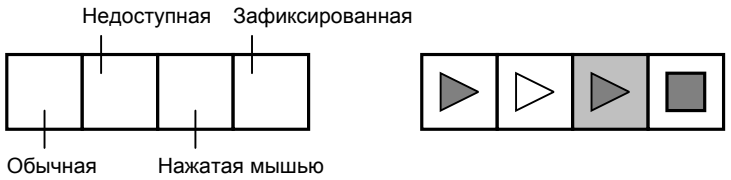


Рис. 2.13. Структура и пример битового образа `Glyph`: картинки, соответствующие состоянию кнопки

UpDown

Компонент `UpDown` (рис. 2.14) 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. 2.14.

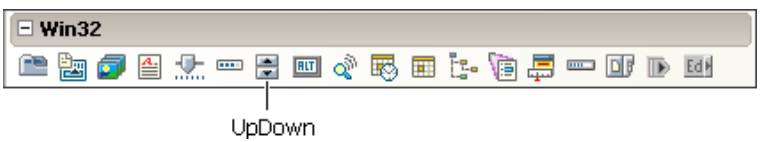


Рис. 2.14. Компонент `UpDown`

Таблица 2.14. *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

Таблица 2.14 *(окончание)*

Свойство	Описание
`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`)

OpenDialog

Компонент `OpenDialog` (рис. 2.15) представляет собой диалог **Открыть**. Свойства компонента приведены в табл. 2.15. Отображение диалога обеспечивает метод `Execute`, значение которого позволяет определить, щелчком на какой кнопке, **Открыть** или **Отмена**, пользователь закрыл диалог.

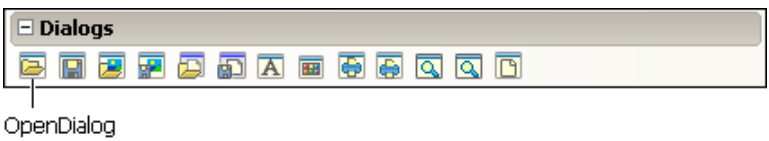


Рис. 2.15. Компонент `OpenDialog`

Таблица 2.15. *Свойства компонента* `OpenDialog`

Свойство	Описание
`Title`	Текст в заголовке окна. Если значение свойства не указано, то в заголовке отображается текст **Открыть**
`Filter`	Свойство задает список фильтров имен файлов. В списке файлов отображаются только те файлы, имена которых соответствуют выбранному (текущему) фильтру. Во время отображения диалога пользователь может выбрать фильтр в списке **Тип файлов**. Каждый фильтр задается строкой вида *описание*`|`*маска*, например `Текст|*.txt`

Таблица 2.15 *(окончание)*

Свойство	Описание
`FilterIndex`	Если в списке `Filter` несколько элементов (например, `Текст|*.txt|Все файлы|*.*`), то значение свойства задает фильтр, который используется в момент появления диалога на экране
`InitialDir`	Каталог, содержимое которого отображается при появлении диалога на экране. Если значение свойства не указано, то в окне диалога отображается содержимое папки Мои документы
`FileNane`	Имя файла, выбранного пользователем

SaveDialog

Компонент `SaveDialog` (рис. 2.16) представляет собой диалог **Сохранить**. Свойства компонента приведены в табл. 2.16. Отображение диалога обеспечивает метод `Execute`, значение которого позволяет определить, щелчком на какой кнопке, **Сохранить** или **Отмена**, пользователь закрыл диалог.

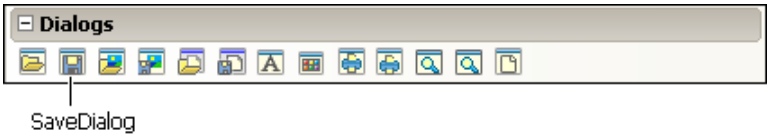


Рис. 2.16. Компонент `SaveDialog`

Таблица 2.16. Свойства компонента *`SaveDialog`*

Свойство	Описание
`Title`	Текст в заголовке окна. Если значение свойства не указано, то в заголовке отображается текст **Сохранить как**
`Filter`	Свойство задает список фильтров имен файлов. В списке файлов отображаются только те файлы, имена которых соответствуют выбранному (текущему) фильтру. Во время отображения диалога пользователь может выбрать фильтр в списке **Тип файлов**. Каждый фильтр задается строкой вида *описание*`|`*маска*, например `Текст|*.txt`

Таблица 2.16 *(окончание)*

Свойство	Описание
`FilterIndex`	Если в списке `Filter` несколько элементов (например, `Текст|*.txt|Все файлы|*.*`), то значение свойства задает фильтр, который используется в момент появления диалога на экране
`InitialDir`	Каталог, содержимое которого отображается при появлении диалога на экране. Если значение свойства не указано, то в окне диалога отображается содержимое папки Мои документы
`FileNane`	Имя файла, введенное пользователем в поле **Имя файла**
`DefaultExt`	Расширение, которое будет добавлено к имени файла, если в поле **Имя файла** пользователь не задаст расширение файла

Animate

Компонент `Animate` (рис. 2.17) позволяет воспроизводить простую, не сопровождаемую звуком анимацию, кадры которой находятся в AVI-файле. Свойства компонента приведены в табл. 2.17.

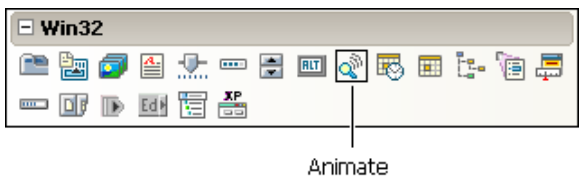


Рис. 2.17. Компонент `Animate`

Таблица 2.17. *Свойства компонента* `Animate`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента и управления его поведением

Таблица 2.17 *(окончание)*

Свойство	Описание
`FileName`	Имя AVI-файла, в котором находится анимация, отображаемая с помощью 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. 2.18) позволяет воспроизвести видеоролик, звук и сопровождаемую звуком анимацию. Свойства компонента приведены в табл. 2.18.

Рис. 2.18. Компонент `MediaPlayer`

Таблица 2.18. Свойства компонента `MediaPlayer`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к свойствам компонента и управления работой плеера

Таблица 2.18 *(окончание)*

Свойство	Описание
`DeviceType`	Тип устройства. Определяет конкретное устройство, которое представляет собой компонент `MediaPlayer`. Тип устройства задается именованной константой: `dtAutoSelect` — тип устройства определяется автоматически; `dtWaveAudio` — проигрыватель звука; `dtAVIVideo` — видеопроигрыватель; `dtCDAudio` — CD-проигрыватель
`FileName`	Имя файла, в котором находится воспроизводимый звуковой фрагмент или видеоролик
`AutoOpen`	Признак автоматического открытия файла видеоролика или звукового фрагмента сразу после запуска программы
`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

КОМПОНЕНТЫ ДОСТУПА/МАНИПУЛИРОВАНИЯ ДАННЫМИ

ADOConnection

Компонент `ADOConnection` (рис. 2.19) обеспечивает соединение с базой данных. Свойства компонента приведены в табл. 2.19.

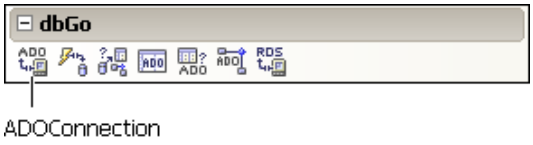


Рис. 2.19. Компонент `ADOConnection`

Таблица 2.19. *Свойства компонента* `ADOConnection`

Свойство	Описание
`ConnectionString`	Строка соединения. Содержит информацию, необходимую для подключения к базе данных
`LoginPrompt`	Признак необходимости запросить имя и пароль пользователя в момент подключения к базе данных. Если значение свойства равно `False`, то окно **Login** в момент подключения к базе данных не отображается
`Mode`	Режим соединения. Соединение с базой данных может быть открыто для чтения (`cmRead`), записи (`cmWrite`), чтения/записи (`cmReadWrite`)
`Connected`	Признак того, что соединение установлено

ADOTable

Компонент `ADOTable` (рис. 2.20) представляет собой таблицу (все столбцы) базы данных (обеспечивает доступ к таблице). Свойства компонента приведены в табл. 2.20.

Рис. 2.20. Компонент `ADOTable`

Таблица 2.20. *Свойства компонента* `ADOTable`

Свойство	Описание
`Connection`	Ссылка на компонент (`ADOConnection`), который обеспечивает соединение с источником (базой) данных
`ConnectionString`	Строка соединения. Содержит информацию о базе данных, элементом которой является таблица, подключение к которой обеспечивает компонент
`TableName`	Таблица базы данных

Таблица 2.20 *(окончание)*

Свойство	Описание
`Filter`	Фильтр — условие отбора записей из таблицы. Позволяет скрыть записи, не удовлетворяющие критерию отбора
`Filtered`	Признак фильтрации записей. Если значение свойства равно `True`, то записи, не удовлетворяющие критерию запроса, не отображаются
`ReadOnly`	Запрещает (`True`) или разрешает (`False`) изменение данных таблицы
`Active`	Признак того, что соединение с таблицей активно

ADODataSet

Компонент `ADODataSet` (рис. 2.21) хранит данные, полученные из базы данных в результате выполнения сервером SQL-команды. В отличие от компонента `ADOTable`, который представляет собой всю таблицу (все столбцы), компонент позволяет прочитать только нужные столбцы или может быть заполнен данными из разных таблиц. Свойства компонента `ADODataSet` приведены в табл. 2.21.

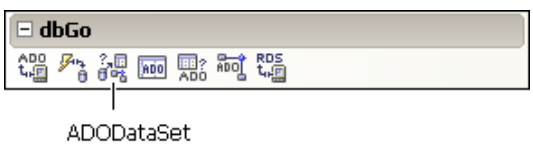


Рис. 2.21. Компонент `ADODataSet`

Таблица 2.21. *Свойства компонента* `ADODataSet`

Свойство	Описание
`Connection`	Ссылка на компонент (`ADOConnection`), который обеспечивает соединение с источником (базой) данных
`CommandText`	Команда, которая направляется серверу
`Parameters`	Параметры команды

Таблица 2.21 *(окончание)*

Свойство	Описание
`Filter`	Фильтр. Позволяет скрыть записи, не удовлетворяющие критерию отбора
`Filtered`	Признак использования фильтра
`Activate`	Открывает или делает недоступным набор данных

ADOQuery

Компонент `ADOQuery` (рис. 2.22) позволяет направить запрос серверу базы данных. Обычно он используется, если данные, которые надо получить из базы данных, распределены по нескольким таблицам (если данные находятся в одной таблице, то рекомендуется использовать компонент `ADOTable`). Свойства компонента приведены в табл. 2.22.

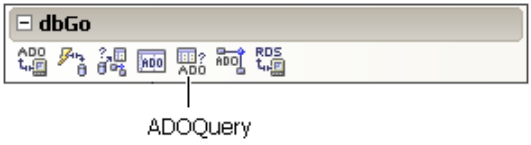


Рис. 2.22. Компонент `ADOQuery`

Таблица 2.22. Свойства компонента *`ADOQuery`*

Свойство	Описание
`Connection`	Ссылка на компонент (`ADOConnection`), который обеспечивает соединение с источником (базой) данных
`ConnectionString`	Строка соединения. Содержит информацию о базе данных, которой направляется запрос (SQL-команда)
`SQL`	SQL-команда (запрос), которая направляется серверу
`Filter`	Фильтр — условие отбора записей из таблицы. Позволяет скрыть записи, не удовлетворяющие критерию отбора

Таблица 2.22 *(окончание)*

Свойство	Описание
Filtered	Признак фильтрации записей. Если значение свойства равно True, то записи, не удовлетворяющие критерию запроса, не отображаются
Active	Признак того, что соединение с таблицей активно

DataSource

Компонент DataSource (рис. 2.23) обеспечивает связь между данными, представленными, например, компонентом ADODataSet, ADOTable или ADOQuery и компонентом, обеспечивающим отображение данных, например DBGrid, DBEdit, DBMemo или DBText. Свойства компонента приведены в табл. 2.23.

Рис. 2.23. Компонент DataSource

Таблица 2.23. Свойства компонента *DataSource*

Свойство	Определяет
Name	Имя компонента. Используется компонентом отображения данных для доступа к компоненту DataSource и, следовательно, к данным, связь с которыми он обеспечивает
DataSet	Компонент, представляющий собой данные (например, ADODataSet, ADOTable или ADOQuery)

DBEdit, DBMemo, DBText

Компоненты DBEdit и DBMemo обеспечивают просмотр и редактирование полей записи базы данных, компонент DBText — только просмотр (рис. 2.24). Свойства компонентов приведены в табл. 2.24.

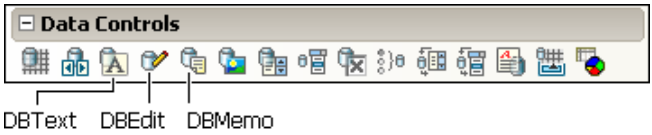


Рис. 2.24. Компоненты просмотра и редактирования полей БД

Таблица 2.24. *Свойства компонентов* `DBText`, `DBEdit` *и* `DBMemo`

Свойство	Определяет
`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. 2.25) используется для просмотра и редактирования базы данных в режиме таблицы. Свойства компонента приведены в табл. 2.25.

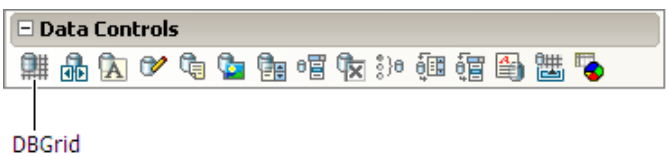


Рис. 2.25. Компонент `DBGrid`

Таблица 2.25. *Свойства компонента* `DBGrid`

Свойство	Описание
`Name`	Имя компонента
`DataSource`	Источник отображаемых в таблице данных (компонент `DataSource`)
`Columns`	Свойство `Columns` представляет собой массив объектов типа `TColumn`, каждый из которых определяет колонку таблицы и отображаемую в ней информацию (см. табл. 2.26)

Таблица 2.25 *(окончание)*

Свойство	Описание
Options	Определяет настройку компонента
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

Таблица 2.26. *Свойства объекта* `TColumn`

Свойство	Определяет
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
Alignment	Способ выравнивания текста в ячейках колонки. Текст может быть выровнен 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 выровнен 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. 2.26 и 2.27) обеспечивает перемещение указателя текущей записи, активизацию режима редактирования, добавление и удаление записей. Компонент представляет собой совокупность командных кнопок (табл. 2.27). Свойства компонента приведены в табл. 2.28.

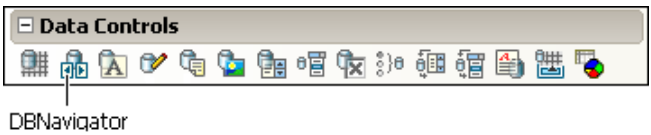


Рис. 2.26. Значок компонента DBNavigator

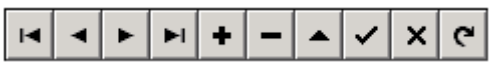


Рис. 2.27. Компонент DBNavigator

Таблица 2.27. *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

Таблица 2.27 *(окончание)*

Кнопка		Обозначение	Действие
▲	**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 2.28. *Свойства компонента* `DBNavigator`

Свойство	Определяет
`Name`	Имя компонента. Используется для доступа к свойствам компонента
`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

PaintBox

Компонент `PaintBox` (рис. 2.28) представляет собой область, в которую программа может вывести графику. Вывод графики обеспечивают методы свойства `Canvas`.

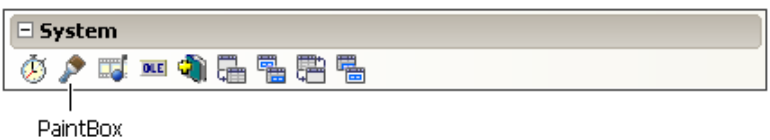


Рис. 2.28. Компонент `PaintBox`

Canvas

Canvas — это объект TCanvas, поверхность (формы, компонента PaintBox или компонента Image), на которую программа может вывести графику.

Вывод графики обеспечивают методы объекта TCanvas (табл. 2.29), свойства объекта TCanvas (табл. 2.30) определяют вид графических элементов.

Таблица 2.29. *Методы объекта* TCanvas

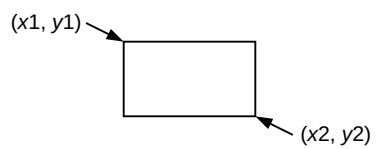
Метод	Описание
TextOut(x,y,s)	Выводит строку s от точки с координатами (x, y). Шрифт определяет свойство Font поверхности (Canvas), на которую выводится тест, цвет закраски области вывода текста — свойство Brush
Draw(x,y,b)	Выводит от точки с координатами (x, y) битовый образ b. Если значение свойства Transparent поверхности, на которую выполняется вывод, равно True, то точки, цвет которых совпадает с цветом левой нижней точки битового образа, не отображаются
LineTo(x,y)	Вычерчивает линию из текущей точки в точку с указанными координатами. Чтобы начертить линию из точки (x1, y1) в точку (x2, y2), надо сначала вызвать метод MoveTo(x1,y2), затем — LineTo(x2,y2). Вид линии определяет свойство Pen
MoveTo(x,y)	Перемещает указатель текущей точки (карандаш) в точку с указанными координатами
Rectangle(x1,y1, 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 

Таблица 2.29 *(продолжение)*

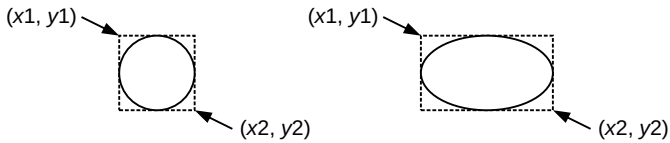
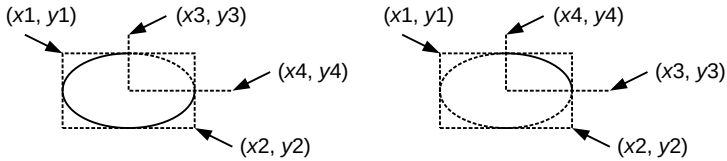
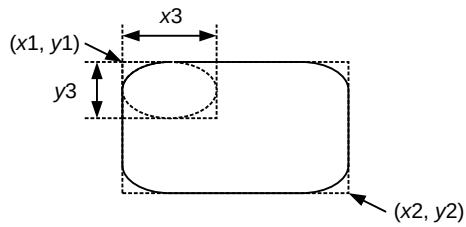
Метод	Описание
`Ellipse(x1,y1, x2,y2)`	Вычерчивает эллипс, окружность или круг. Параметры `x1`, `y1`, `x2` и `y2` задают размер прямоугольника, в который вписывается эллипс. Вид линии определяет свойство `Pen` 
`Arc(x1,y1,x2,y2, x3,y3,x4,y4)`	Вычерчивает дугу. Параметры `x1`, `y1`, `x2`, `y2` определяют эллипс, из которого вырезается дуга, параметры `x3`, `y3`, `x4` и `y4` — координаты концов дуги. Дуга вычерчивается против часовой стрелки от точки (`x3`, `y3`) к точке (`x4`, `y4`). Вид линии (границы) 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 2.29 *(окончание)*

Метод	Описание
`PolyLine(pl)`	Вычерчивает ломаную линию. 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 одинаковыми. Вид границы определяет свойство `Pen`, цвет и стиль закраски внутренней области — свойство `Brush`

Таблица 2.30. *Свойства объекта* `TCanvas`

Свойство	Описание
`Pen`	Свойство `Pen` представляет собой объект `TPen` (см. табл. 2.31), свойства которого определяют цвет, толщину и стиль линий, вычерчиваемых методами вывода графических примитивов
`Brush`	Свойство `Brush` представляет собой объект `Brush` (см. табл. 2.32), 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
`Transparent`	Признак использования "прозрачного" цвета при выводе битового образа методом `Draw`. Если значение свойства равно `True`, то точки, цвет которых совпадает с цветом левой нижней точки битового образа, не отображаются

Pen

Объект `Pen` (карандаш) является свойством объекта `Canvas`. Свойства объекта `Pen` (табл. 2.31) определяют цвет, стиль и толщину

линий, вычерчиваемых методами вывода графических примитивов.

Таблица 2.31. *Свойства объекта* `Pen`

Свойство	Описание
`Color`	Цвет линии: `clBlack` — черный; `clMaroon` — каштановый; `clGreen` — зеленый; `clOlive` — оливковый; `clNavy` — темно-синий; `clPurple` — фиолетовый; `clTeal` — зелено-голубой; `clGray` — серый; `clSilver` — серебристый; `clRed` — красный; `clLime` — салатовый; `clBlue` — синий; `clFuchsia` — ярко-розовый; `clAqua` — бирюзовый; `clWhite` — белый
`Style`	Стиль (вид) линии: `psSolid` — сплошная; `psDash` — пунктирная (длинные штрихи); `psDot` — пунктирная (короткие штрихи); `psDashDot` — пунктирная (чередование длинного и короткого штрихов); `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

Объект `Brush` (Кисть) является свойством объекта `Canvas`. Свойства объекта `Brush` (табл. 2.32) определяют цвет, стиль закраски внутренних областей контуров, вычерчиваемых методами вывода графических примитивов.

Таблица 2.32. *Свойства объекта* `Brush`

Свойство	Определяет
`Color`	Цвет закраски области: `clBlack` — черный; `clMaroon` — каштановый; `clGreen` — зеленый; `clOlive` — оливковый; `clNavy` — темно-синий; `clPurple` — фиолетовый; `clTeal` — зелено-голубой; `clGray` — серый; `clSilver` — серебристый; `clRed` — красный; `clLime` — салатовый; `clBlue` — синий; `clFuchsia` — ярко-розовый; `clAqua` — бирюзовый; `clWhite` — белый

Таблица 2.32 *(окончание)*

Свойство	Определяет
`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

Цвет

Цвет линии (свойство `Pen.Color`) или цвет закраски области (свойство `Brush.Color`) можно задать, указав в качестве значения свойства именованную константу типа `TColor` (табл. 2.33). Вместо конкретного цвета можно указать цвет элемента графического интерфейса, тем самым "привязать" цвет к цветовой схеме операционной системы.

Таблица 2.33. *Константы* `TColor`

Константа	Цвет	Константа	Цвет
`clBlack`	Черный	`clSilver`	Серебристый
`clMaroon`	Каштановый	`clRed`	Красный
`clGreen`	Зеленый	`clLime`	Салатовый
`clOlive`	Оливковый	`clBlue`	Синий
`clNavy`	Темно-синий	`clFuchsia`	Ярко-розовый
`clPurple`	Фиолетовый	`clAqua`	Бирюзовый
`clTeal`	Зелено-голубой	`clWhite`	Белый
`clGray`	Серый		

ФУНКЦИИ

В этом разделе приведено краткое описание часто используемых функций (табл. 2.34—2.37). Подробное их описание можно найти в справочной системе.

Функции ввода и вывода

Таблица 2.34. *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 равно `0`. Значением функции может быть одна из констант: `mrAbort`, `mrYes`, `mrOk`, `mrRetry`, `mrNo`, `mrCancel`, `mrIgnore` или `mrAll`, обозначающая соответствующую командную кнопку

Математические функции

Таблица 2.35. *Математические функции*

Функция	Значение
`abs(n)`	Абсолютное значение `n`
`sqrt(n)`	Квадратный корень из `n`
`exp(n)`	Экспонента `n`
`random(n)`	Случайное целое число в диапазоне от 0 до `n-1` (перед первым обращением к функции необходимо вызвать процедуру `randomize`, которая выполнит инициализацию программного генератора случайных чисел)
`sin(α)`	Синус выраженного в радианах угла `α`
`cos(α)`	Косинус выраженного в радианах угла `α`
`tan(α)`	Тангенс выраженного в радианах угла `α`
`asin(n)`, `acos(n)`, `atan(n)`	Угол (в радианах), синус, косинус и тангенс которого равен `n`

Аргумент тригонометрических функций (угол) должен быть выражен в радианах. Для преобразования величины угла из градусов в радианы следует воспользоваться формулой `(a*3.1415256)/180`, где `a` — величина угла в градусах; `3.1415926` — число π. Вместо десятичной константы `3.1415926` можно использовать стандартную именованную константу `PI`.

Функции преобразования

Таблица 2.36. *Функции преобразования*

Функция	Значение
`IntToStr(k)`	Строка, являющаяся изображением целого `k`
`FloatToStr(n)`	Строка, являющаяся изображением вещественного `n`

Таблица 2.36 *(окончание)*

Функция	Значение
`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`
`StrToInt(s)`	Целое, изображением которого является строка `s`
`StrToFloat(s)`	Вещественное, изображением которого является строка `s`

Функции манипулирования датами и временем

Большинству функций манипулирования датами в качестве параметра передается переменная типа `TDateTime`, которая хранит информацию о дате и времени.

Для того чтобы в программе были доступны функции `DayOf`, `WeekOf`, `MonthOf` и некоторые другие, например `CompareTime`, в ее текст надо включить ссылку на модуль `DateUtils` (указать имя модуля в директиве `uses`).

Таблица 2.37. *Функции манипулирования датами и временем*

Функция	Значение
`Now()`	Системная дата и время — значение типа `TDateTime`
`DateToStr(dt)`	Строка символов, изображающая дату в формате `dd.mm.yyyy`

Таблица 2.37 *(продолжение)*

Функция	Значение
`TimeToStr(dt)`	Строка символов, изображающая время в формате `hh:mm:ss`
`FormatDateTime(s,dt)`	Строка символов, представляющая собой дату или время. Способ представления задает строка `s`. Например, строка `dd.mm.yyyy` задает, что значением функции является дата, а строка `hh:mm` — время. Помимо символов формата, в строке `s` могут быть и другие символы, например, если необходимо отобразить текущее время, то строка форматирования может выглядеть так: `Сейчас hh:mm`. Форматы: `d` — число, одна или две цифры; `dd` — число, две цифры; `ddd` — сокращенное название дня недели; `dddd` — полное название дня недели; `m` — месяц, одна или две цифры; `mm` — месяц, две цифры; `mmm` — сокращенное название месяца; `mmmm` — полное название месяца
`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 (`w` — номер недели, отсчитываемый от начала года)
`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 2.37 *(окончание)*

Функция	Значение
`DecodeTime(dt,h,m,s,ms)`	Возвращает время (часы, минуты, секунды и миллисекунды), представленное отдельными числами
`EncodeDate(y,m,d)`	Значение типа `TDateTime`, соответствующее указанной дате (`y` — год, `m` — месяц, `d` — день)
`EncodeTime(h,m,s,ms)`	Значение типа `TDateTime`, соответствующее указанному времени (`h` — часы; `m` — минуты; `s` — секунды; `ms` — миллисекунды)
`CompareDate(d1,d2)`	Сравнивает две даты (значения типа `TDateTime`). Если даты совпадают, значение функции равно нулю, если `d1 < d2` (например, `d1 = 01.06.2012`, а `d2 = 05.06.2012`), то значение функции — "минус один", если `d2 < d1`, то значение функции — "единица"
`CompareTime(t1,t2)`	Сравнивает два временных значения (значения типа `TDateTime`). Если времена совпадают, значение функции равно нулю, если `t1 < t2` (например, `t1` = 10:00, а `t2` = 10:30), то значение функции — "минус один", если `t2 < t1`, то значение функции — "единица"

СОБЫТИЯ

Таблица 2.38. *События*

Событие	Происходит
`Click`	При щелчке кнопкой мыши
`DblClick`	При двойном щелчке кнопкой мыши
`MouseDown`	При нажатии кнопки мыши
`MouseUp`	При отпускании кнопки мыши
`MouseMove`	При перемещении мыши
`KeyPress`	При нажатии клавиши клавиатуры

Таблица 2.38 *(окончание)*

Событие	Происходит
KeyDown	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)
KeyUp	При отпускании нажатой клавиши клавиатуры
Create	При создании объекта (формы, элемента управления). Процедура обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий
Paint	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
Enter	При получении элементом управления фокуса
Exit	При потере элементом управления фокуса

ИСКЛЮЧЕНИЯ

Таблица 2.39. *Типичные исключения*

Тип исключения	Возникает
EConvertError	При выполнении преобразования строки в число, если преобразуемая величина не может быть приведена к требуемому виду. Наиболее часто возникает при преобразовании строки в дробное число, если в качестве разделителя целой и дробной частей указан неверный символ
EZeroDivide	Деление на ноль. При выполнении операции деления, если делитель равен нулю (если и делитель, и делимое равны нулю, то возникает исключение EInvalidOp)
EFOpenError	При обращении к файлу, например при попытке загрузить файл иллюстрации с помощью метода LoadFromFile. Наиболее частой причиной является отсутствие требуемого файла или, в случае использования сменного диска, отсутствие диска в накопителе

Таблица 2.39 *(окончание)*

Тип исключения	Возникает
`EInOutError`	При обращении к файлу, например при попытке открыть для чтения несуществующий файл
`EOLEException`	При выполнении операций с базой данных, например при попытке открыть несуществующую базу данных, если для доступа к базе данных используются ADO-компоненты (чтобы иметь возможность обработки этого исключения, в директиву `uses` надо добавить ссылку на модуль `ComObj`)

ПРИЛОЖЕНИЕ

Описание электронного архива

Электронный архив к книге выложен на FTP-сервер издательства по адресу: **ftp://85.249.45.166/9785977508117.zip**. Ссылка доступна и со страницы книги на сайте **www.bhv.ru**.

Электронный архив содержит проекты, приведенные в книге. Каждый проект находится в отдельном каталоге.

После компиляции большинство программ не требуют для своей работы никакой дополнительной настройки. Для некоторых программ, например программ работы с базами данных, необходимо указать, где находился файл базы данных.

Для активной работы, чтобы иметь возможность вносить изменения в программы, скопируйте каталоги проектов на жесткий диск компьютера в папку Projects, которая при стандартной установке Delphi находится в папке Документы\RAD Studio.

Предметный указатель

Г

Д

З

И

К

Л

М

О

П

Р

С

Т

Ф

Ц

Ч

Э