Министерство образования и науки Российской Федерации
Филиал федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Красноярский государственный педагогический университет им. В.П. Астафьева» в г. Железногорске
Федеральное государственное бюджетное учреждение науки Институт физики им. Л.В. Киренского СО РАН

С.В. Бутаков

РАЗРАБОТКА WINDOWS-ПРИЛОЖЕНИЙ В СРЕДЕ DELPHI

Лабораторный практикум: учебное пособие

Рекомендовано Сибирским региональным учебно-методическим центром высшего профессионального образования для межвузовского использования в качестве учебного пособия для студентов, обучающихся по специальности 050202 – «Информатика»

Железногорск 2012

УДК 004.41
ББК 32.81 я 73
Б 93

Печатается по решению редакционно-издательского совета ФГБОУ ВПО «Красноярский государственный педагогический университет им. В.П. Астафьева»

Рецензенты:

Г.И. Цугленок,

доктор технических наук, профессор, заведующий кафедрой высшей и прикладной математики ФГБОУ ВПО «Красноярский государственный аграрный университет»

И.О. Богульский,

доктор физико-математических наук, профессор, ведущий научный сотрудник Института вычислительного моделирования СО РАН

Б 93 **Бутаков С.В.**

Разработка Windows-приложений в среде Delphi. Лабораторный практикум: учебное пособие / С.В. Бутаков. Изд. 2-е, испр. и доп. – Красноярск: РИО КГПУ им. В.П. Астафьева, 2012. – 108 с.: 14 ил.; 10 наимен. библ.

Предназначено для аудиторной и самостоятельной работы студентов педагогических вузов в рамках дисциплины предметной подготовки «Программирование» или спецкурса по программированию специальности 050202 – «Информатика», а также для учащихся старших классов и всех желающих самостоятельно изучать среду разработки Delphi.

УДК 004.41
ББК 32.81 я 73

ISBN 5-98997-008-0

Содержание

Предисловие

Практикум предназначен для аудиторной и самостоятельной работы студентов педагогических вузов, обучающихся по специальности 050202 «Информатика» и изучающих дисциплину «Программирование» или спецкурс по программированию. Он содержит десять лабораторных работ, выполняемых в интегрированной среде разработки Delphi.

Во введении содержатся краткие сведения об интегрированной среде разработки Delphi.

В лабораторных работах рассмотрены примеры разработки большинства стандартных прикладных приложений для Windows, таких как калькулятор, текстовый редактор, вьюеры текстовых и графический файлов, программа научной графики, секундомер, универсальный проигрыватель, графический редактор. Для создания этих приложений в лабораторных работах применяются компоненты общего назначения и компоненты, использующие некоторые системные средства Windows, размещенные на страницах Standard, Additional, Win32, System, Dialogs палитры компонентов Delphi. Лабораторные работы расположены в порядке изучения компонентов, содержащихся на этих страницах.

Выполнять лабораторные работы рекомендуется строго последовательно, начиная с первой, так как содержание последующей работы основано на материале, изученном при выполнении предыдущей.

После каждой лабораторной работы приведены задания для самостоятельной работы, выполнение которых закрепляют знания и практические навыки, полученные в ходе выполнения работ.

В Приложении представлены справочные сведения о свойствах и событиях формы и некоторых процедурах и функциях Object Pascal для работы со строками.

Введение

Интегрированная среда разработки (Integrated Development Environment – IDE) Delphi – это комплекс программных (инструментальных) средств, предназначенных для создания (проектирования, отладки и выполнения) приложений на языке программирования Object Pascal.

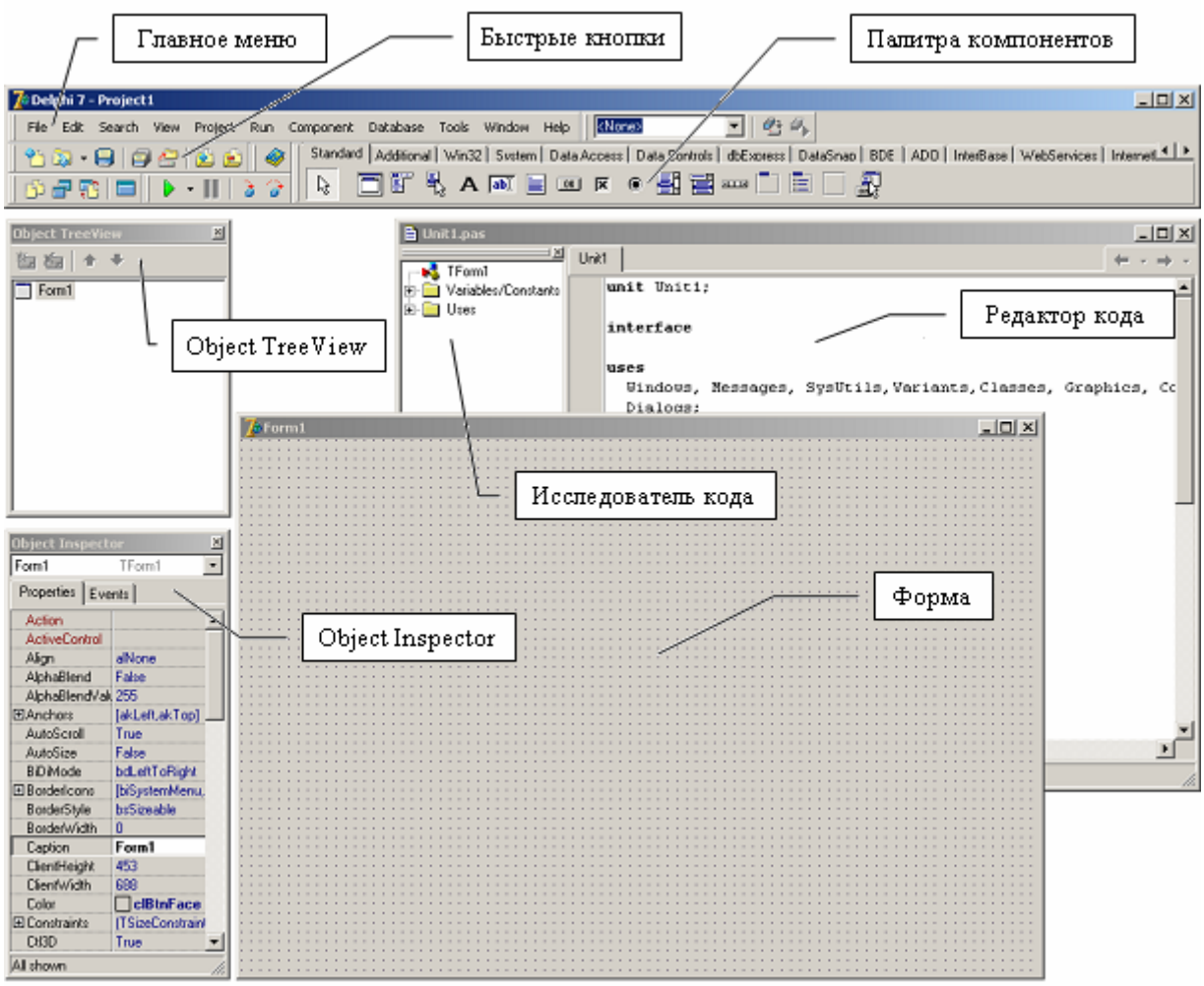

Рис. 1. Интегрированная среда разработки Delphi 7

На рис. 1 представлен вид Интегрированной среды разработки Delphi 7. В верхней части IDE расположены главное меню Delphi и инструментальные панели, содержащие набор быстрых кнопок, дублирующих наиболее часто используемые команды главного меню, и палитру компонентов библиотеки визуальных компонентов (Visual Component Library – VCL). Палитра компонентов имеет ряд страниц, снабженных вкладками. На этих

страницах сгруппированы пиктограммы всех визуальных и невизуальных компонентов, содержащихся в Delphi.

В левой части IDE расположено окно Object TreeView, которое предназначено для просмотра иерархической структуры компонентов, используемых в приложении. Ниже находится окно Object Inspector – Инспектор объектов, с помощью которого можно просматривать и задавать значения свойств и управлять событиями у используемых компонентов. Окно Object Inspector имеет две страницы: страницу свойств – Properties и страницу событий – Events. Для активизации окна Object Inspector используйте клавишу быстрого доступа F11.

Основную часть IDE занимает окно формы, за ним расположено окно Редактора кода. Обычно оно полностью перекрывается формой. На рис. 1 форма сдвинута так, чтобы было видно окно редактора. Для переключения между окном формы и окном Редактора кода используйте клавишу F12.

Форма является основой приложений Delphi, на которой размещаются другие компоненты. Во время проектирования приложения на форме видна сетка из точек. По узлам этой сетки выравниваются компоненты, помещаемые на форму. Во время выполнения приложения эта сетка не видна.

Редактор кода – это программный редактор. В окне Редактора кода жирным шрифтом ключевые слова Object Pascal, синим курсивом – комментарии, в заголовке окна отображается имя текущего файла. В верхней части окна редактора находятся вкладки, указывающие на текущую страницу. Приложения Delphi могут использовать много исходных файлов, и вкладки помогают переходить от одного из них к другому. В левой части Редактора кода расположено встроенное в него окно Исследователя кода.

Для получения контекстной справки о свойствах и событиях компонентов выделите в Object Inspector интересующее вас свойство или событие и нажмите клавишу F1. Для получения контекстной справки о каком-либо слове кода (ключевом слове Object Pascal, имени встроенной процедуры или функции и т.п.) установите курсор на это слово в Редакторе кода и нажмите клавишу F1.

Лабораторная работа №1

Разработка программы Калькулятор

Задание. В среде Delphi разработать приложение Калькулятор, выполняющий четыре арифметических действия (сложение, вычитание, умножение и деление).

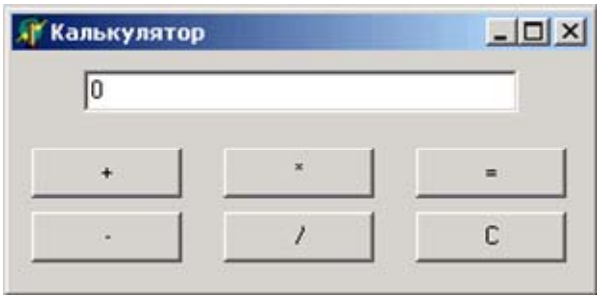

Рис. 2. Калькулятор

1. Запустите IDE Delphi с помощью главного меню Windows (кнопка Пуск > Программы > Borland Delphi).

2. Сохраните проект с именем Calculator в отдельной папке. Для этого выполните команду File > Save Project As главного меню Delphi. При запросе имени сохраняемого модуля оставьте имя по умолчанию (Unit1.pas), а на запрос имени сохраняемого проекта вместо предлагаемого по умолчанию Project1.dpr напишите Calculator.dpr.

3. Задайте заголовок окна приложения (формы). Для этого в Object Inspector у формы в свойстве **Caption** вместо значения по умолчанию *Form1* напишите слово *Калькулятор.*

4. Измените размеры формы, установив следующие значения ее свойствам: **Height**:=*150*, **Width**:=*300*.

5. Поместите на форму кнопку «+» (см. рис. 2). Для этого на странице Standard палитры компонентов выделите пиктограмму кнопки  и затем щелкните курсором мыши в нужной части формы. На форме появится кнопка, которой Delphi присвоит имя по умолчанию **Button1**. Выделите на форме кнопку **Button1**. Перейдите в Object Inspector и измените значение ее свойства **Caption** на +, при этом на кнопке появится надпись «+».

6. Перенесите на форму еще пять кнопок **Button2**, **Button3**, **Button4**, **Button5, Button6**, выполнив аналогичные действия. Это будут кнопки «–», «*», «/», «=», «C», соответственно.

7. Поместите на форму компонент **Edit**  . (**Edit1**). Это будет поле для ввода пользователем чисел и вывода результатов вычислений. Измените ширину компонента с помощью мыши или задав его свойству **Width** значение *220*. В свойстве **Text** в место *Text1* запишите *0* – этот символ будет отображаться в поле ввода после запуска вашего приложения.

8. На этом создание графического интерфейса завершено. Теперь перейдем к написанию обработчиков событий. Не забывайте периодически сохранять ваш проект командой File > Save или File > Save All главного меню Delphi или соответствующими быстрыми кнопками  или  .

9. У кнопки **Button1** («+») создайте событие **OnClick**, которое происходит при щелчке мыши на компоненте**.** Для этого выделите компонент **Button1** на форме, перейдите в Object Inspector, откройте в нем вкладку Events (События), найдите событие **OnClick** и сделайте двойной щелчок в окне справа от имени этого события. Это стандартный способ создания обработчиков любых событий. Перейти в обработчик события **OnClick** (только этого события) можно и иначе: сделать двойной щелчок мышью на компоненте **Button1**. В обоих случаях вы перейдете в окно Редактора кода, в котором появится текст процедуры – обработчика события:

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

end;
```

10. В обработчик этого события между операторами **begin** и **end** вставьте код:

```
Try
X:=StrToFloat(Edit1.Text);
```

```
Except
  on  EConvertError  do
   begin
     ShowMessage('Введено не числовое значение');
     Exit;
   end;
end;
act:=1;
```

Код, который вы написали, означает следующее. Конструкция **Try**…**Except** – это блоки обработки исключительных ситуаций. Они предназначены для корректной обработки программой возможных ошибок, в данном случае ошибки, которая может возникнуть, если пользователь в место числа введет набор символов, например, «1ю5». В блок **Try** помещаются операторы, при выполнении которых может возникнуть ошибка, а в блок **Except** – операторы, которые будут выполняться в случае возникновения ошибки, например, выдача пользователю сообщения об ошибке. С помощью оператора **on…do**, помещаемого в этот блок, можно производить выборочную обработку различных исключительных ситуаций.

Разберем более подробно приведенный код. В блок **Try** помещен оператор: `X:=StrToFloat(Edit1.Text);` Слева от оператора присвоения (символы «:=») записана переменная `X` – переменная, в которой в программе будет храниться значение первого числа, введенного пользователем. Справа – функция **StrToFloat**, преобразующая строку в действительное число. Все, что введет пользователь в поле ввода (компонент **Edit1**), хранится в виде строки в свойстве **Text** этого компонента. Чтобы обратиться к свойству компонента, на первом месте записывается имя компонента, затем ставится точка, а затем без пробела записывается имя свойства*. В

* Во время написания кода, если вы поставите точку после имени объекта, всплывает подсказка, содержащая список всех свойств и методов, относящихся к этому объекту. Вы можете дальше не писать полностью имя нужного свойства или метода, а выбрать его из этого списка и нажать клавишу Enter, при этом выбранное имя вставится в текст.

данном случае в свойстве **Text** компонента **Edit1** (обращаемся к этому свойству как `Edit1.Text`) будет храниться набор символов, введенный пользователем, а функция **StrToFloat** преобразует их в числовое значение, которое присваивается переменной `X`.

В блоке **Except** оператор **on** `EConvertError` **do** производит обработку одной из исключительных ситуаций – класса EConvertError, которая возникает при попытке преобразовать в число строку из буквенных символов. В случае возникновении такой ошибки выполняются операторы, следующие после **do** и объединенные в блок **begin**…**end**. А именно, процедура **ShowMessage** в случае указанной ошибки в диалоговом окне выводит сообщение «Введено не числовое значение», а процедура **Exit** прекращает дальнейшее выполнение процедуры – обработчика события.

Оператор **end** завершает блоки обработки исключительных ситуаций **Try**…**Except**.

И, в завершение выполнения процедуры, если не возникло ошибки преобразования, оператор `act:=1;` присваивает переменной `act` условный код нажатой кнопки. Для кнопки «+» примем его равным *1*.

11. Для кнопок **Button2**, **Button3**, **Button4** («–», «*», «/») создайте события **OnClick**, выполнив такие же действия, как п. 9. В обработчиках событий напишите следующий код (текст в фигурных скобках *{}* и после *//* можно не писать – это комментарии).

У кнопки **Button2** («–»):

```
Try
X:=StrToFloat(Edit1.Text);
Except
  on  EConvertError  do
   begin
     ShowMessage('Введено не числовое значение');
     Exit;
   end;
end;
act:=2; // условный код кнопки «–»
```

У кнопки **Button3** («*»):

```
Try
X:=StrToFloat(Edit1.Text);
Except
  on  EConvertError  do
   begin
     ShowMessage('Введено не числовое значение');
     Exit;
   end;
end;
act:=3;  {условный код кнопки «*»}
```

У кнопки **Button4** («/»):

```
Try
X:=StrToFloat(Edit1.Text);
Except
  on  EConvertError  do
   begin
     ShowMessage('Введено не числовое значение');
     Exit;
   end;
end;
act:=4;  // условный код кнопки «/»
```

12. Создайте событие **OnClick** у кнопки **Button5** («=»). В его обработчике напишите:

```
Try
Y:=StrToFloat(Edit1.Text);
Except
  on  EConvertError  do
   begin
     ShowMessage('Введено не числовое значение');
     Exit;
   end;
end;
  Case act of
  1: Z:=X+Y;
  2: Z:=X-Y;
  3: Z:=X*Y;
  4: Z:=X/Y;
  end;
Edit1.Text:=FloatToStr(Z);
```

Прокомментируем приведенный код. Вначале идут уже знакомые блоки обработки исключительных ситуаций **Try**…**Except**. В блоке **Try** за-

писан оператор `Y:=StrToFloat(Edit1.Text)`, который присваивает переменной `Y` значение второго числа, введенного пользователем.

Оператор выбора **Case** производит арифметические действия с веденными пользователем числами, которые хранятся в переменных `X` и `Y`, в зависимости от кода (переменная `act`) ранее нажатой кнопки действия («+», «–», «*», «/»). Результат вычисления заносится в переменную `Z`.

В последней строке `Edit1.Text:=FloatToStr(Z);` числовое значение переменной `Z` с помощью функции **FloatToStr** (функция, обратная **StrToFloat**) преобразуется в строковое и присваивается свойству **Text** компонента **Edit1**, т.е. результат вычисления выводится на экран в поле ввода.

13. Создайте событие **OnClick** у кнопки **Button6** («C» – очистка). В его обработчике напишите:

```
X:=0;
Y:=0;
Z:=0;
act:=0;
Edit1.Text:='0';
```

Эта кнопка производит обнуление всех значений переменных, сброс кода нажатой кнопки действия и выводит на экран в поле ввода символ «0».

14. В заключение опишем все переменные, введенные нами в программе, – это `act`, `X`, `Y`, `Z`, так, чтобы они были доступны внутри модуля. Для этого в окне Редактора кода, в тексте модуля, ниже ключевого слова **implementation** запишите:

```
var
  act: integer;
  X,Y,Z: real;
```

Таким образом, мы объявили `act` переменной целого типа, а `X`, `Y`, `Z` – действительного типа.

15. Запустите ваше приложение быстрой кнопкой Run  или клавишей быстрого доступа F9 или командой главного меню Run > Run. В случае возникновения ошибок при компиляции отладьте приложение – проверьте еще раз правильность написания кода (п. 9 – 14). Простейший калькулятор готов! Убедитесь в его работоспособности. Введите в поле ввода первое число, нажмите кнопку действия (любую из «+», «–», «*», «/») , введите второе число и нажмите кнопку «=». Посмотрите в окне ввода результат выполнения вычислений.

Задания для самостоятельной работы

Задача 1. В среде Delphi разработать приложение Калькулятор, выполняющий четыре арифметических действия (сложение, вычитание, умножение и деление). Ввод чисел должен осуществляться только с помощью кнопок калькулятора.

Лабораторная работа №2

Разработка однострочного текстового редактора

Задание. В среде Delphi разработать приложение, позволяющее считывать из файла, редактировать и сохранять в файл одну текстовую строку.

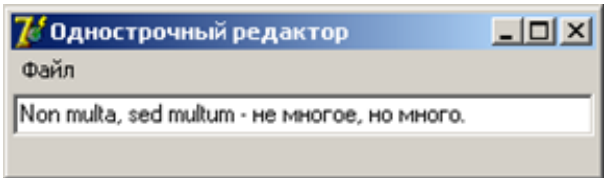

Рис. 3. Однострочный текстовый редактор

1. Запустите IDE Delphi.

2. Сохраните проект с именем Edit в отдельной папке (см. п. 2 лабораторной работы № 1).

3. Задайте заголовок окна приложения (формы). Для этого в Object Inspector у формы измените значение ее свойства **Caption** на *Однострочный редактор.* Измените размеры формы, установив следующие значения ее свойствам: **Height**:=*95*, **Width**:=*300*.

4. Поместите на форму компонент **MainMenu**  со страницы Standard палитры компонентов (имя этого компонента будет **MainMenu1**). Это невизуальный компонент и предназначен для проектирования главного меню. Такие компоненты после помещения на форму остаются в виде пиктограммы и не видны во время выполнения программы. Поэтому не имеет значения, где он будут размещен на форме.

Создадим раздел меню Файл, содержащий три раздела (команды) Открыть…, Сохранить как… и Выход (многоточие означает, что эта команда меню должна вызывать диалоговое окно). Щелкните два раза по компоненту **MainMenu1** на форме, при этом откроется окно Конструктора меню. Перейдите в Object Inspector и в свойстве **Caption** напишите *Файл*. Перейдите в окно Конструктора меню, в котором появится название соз-

данного раздела. Поместите курсор в рамку из точек, обозначающую место нового раздела, которая расположена ниже раздела Файл. В Object Inspector в свойстве **Caption** создаваемого раздела (каждый раздел меню является объектом и обладает своими свойствами, методами и событиями) напишите *Открыть*….Подобным образом, ниже этого раздела, создайте раздел Сохранить как… и Выход. После этого закройте окно Конструктора меню. В верхней части формы появится главное меню, содержащее созданные разделы (см. рис. 3).

5. Поместите на форму компонент **Edit**  . (**Edit1**). Это будет поле для ввода однострочных текстов. Измените ширину компонента, задав его свойству **Width** значение *290*. Сотрите содержимое его свойства **Text**. Выделите компонент и, удерживая на клавиатуре клавишу Ctrl, клавишами со стрелками выровняйте его положение на форме.

6. Со страницы Dialogs палитры компонентов перенесите на форму компоненты, реализующие диалоговые окна открытия файлов **OpenDialog**  (**OpenDialog1**) и сохранения файлов **SaveDialog**  (**SaveDialog1**). Это невизуальные компоненты и могут быть размещены в любом месте формы.

7. Выполните настройку диалогов. Для этого выделите компонент **OpenDialog1**. С помощью его свойства **Filter** задайте типы файлов, появляющиеся в диалоге в выпадающем списке Тип файла. Для этого в Object Inspector выделите это свойство и нажмите на кнопку с многоточием  рядом имени этого свойства, чтобы вызвать редактор фильтров. В открывшемся окне редактора задайте два фильтра, как показано в таблице:

Filter Name	Filter
Текстовые файлы	*.txt
Все файлы	*.*

В левой панели Filter Name записывается текст, который увидит пользователь в выпадающем списке Тип файла диалога, а в правой панели

Filter – шаблоны фильтра. Вы задали два фильтра: все текстовые файлы с расширением .txt (* означает любое имя) и все файлы с любым именем и любым расширением.

С помощью свойства **FilterIndex** задайте номер фильтра, который будет по умолчанию показан пользователю в момент открытия диалога, например **FilterIndex**:=*1*.

Теперь точно также настройте диалог **SaveDialog1**. Кроме этого, в значение его свойства **DefaultExt** напишите *txt*. Это свойство определяет расширение, которое будет по умолчанию добавляться к именам файлов при их сохранении (если значение этого свойства не задано, то пользователь должен будет указывать в диалоге полное имя файла с расширением). Щелкните по знаку плюс рядом со свойством **Options** и включите опцию **ofOverwritePrompt**, выбрав из раскрывающегося списка рядом с именем этого свойства значение *true*. Теперь, в случае если пользователь при сохранении файла напишет имя уже существующего файла, об этом будет выдано сообщение.

На этом создание графического интерфейса завершено. Не забывайте периодически сохранять ваш проект!

8. Сначала объявим переменные, которые будут использоваться в программе: файловую переменную `FileVar` типа Текстовый файл, используемую для доступа к файлам, и строковую переменную `St`, в которой будет храниться текстовая строка. Для этого в тексте модуля, ниже ключевого слова **implementation**, запишите:

```
var
  FileVar: TextFile;
  St: string;
```

9. В главном меню, которое вы спроектировали в п. 4, у раздела Открыть… создайте событие **OnClick**, щелкнув на нем мышью. Команда

Открыть… должна вызывать диалоговое окно открытия файлов, считывать из файла, выбранного пользователем, одну текстовую строку и выводить ее в поле ввода (компонент **Edit1**). В обработчик этого события между операторами **begin** и **end** напишите:

```
If OpenDialog1.Execute Then
begin
AssignFile(FileVar,OpenDialog1.FileName);
Reset(FileVar);
Read(FileVar,St);
Edit1.Text:=St;
CloseFile(FileVar);
end;
```

Прокомментируем приведенный выше код. С помощью условного оператора **If** вызывается диалоговые окно открытия файлов **OpenDialog1**. Если пользователь выбирает в диалоге файл и нажимает на кнопку Открыть, то выполняются операторы, идущие после слова **Then** и объединенные в блок **begin**…**end**. Имя файла, выбранного пользователем в диалоге, содержится в виде строки в свойстве **FileName** этого диалога. Процедура **AssignFile** связывает это имя с файловой переменной `FileVar`. Процедура **Reset** открывает выбранный файл. Процедура **Read** читает из файла текстовую строку в переменную `St`. Далее, значение этой переменной присваивается свойству **Text** компонента **Edit1** (выводится на экран). Процедура **CloseFile** закрывает ранее открытый файл.

10. Создайте событие **OnClick** у раздела главного меню Сохранить как…. Эта команда должна вызывать диалоговое окно сохранения файлов и записывать в файл, выбранный пользователем, одну текстовую строку из поля ввода. В обработчик этого события между операторами **begin** и **end** напишите (комментарии к коду приведены в фигурных скобках *{}* и после *//*):

```
If SaveDialog1.Execute Then // Вызывает диалоговое окно
Begin
{Связывает имя файла с файловой переменной FileVar}
AssignFile(FileVar,SaveDialog1.FileName);
{Создает и открывает новый файл, связный с файловой переменной FileVar }
Rewrite(FileVar);
{Присваивает переменной St строку, содержащуюся в поле ввода}
St:=Edit1.Text;
{Записывает значение переменной St в файл, связанный с файловой переменной FileVar}
Write(FileVar,St);
CloseFile(FileVar); // Закрывает файл
end;
```

11. Создайте событие **OnClick** у раздела главного меню Выход. Эта команда должна закрывать приложение. В обработчик события запишите:

```
Close;
```

12. Запустите ваше приложение. Убедитесь в его работоспособности. Введите в поле ввода любую текстовую строку. Сохраните ее в файл с помощью команды главного меню Сохранить как. Просмотрите сохраненный файл с помощью стандартных средств Windows, например Блокнота, и убедитесь, что введенная вами строка содержится в файле. Сотрите текст в поле ввода вашего однострочного редактора и откройте с помощью команды Открыть ранее сохраненный файл. Убедитесь, что тестовая строка из файла считалась правильно. Закройте ваше приложение с помощью команды Выход.

Задания для самостоятельной работы

Задача 2. В среде Delphi разработать приложение, содержащее текстовое поле, которое позволяет пользователю ввести номер телефона только в соответствии с шаблоном: (*-****)-**-**-**, где * – любая цифра, и сохранить этот номер в виде текстовой строки в файл или открыть его из файла.

Указание: вместо компонента **Edit** использовать компонент **MaskEdit**  – маскированный ввод, расположенный на странице Additional палитры компонентов Delphi. Этот компонент используется для ввода символов в соответствии с шаблоном.

Лабораторная работа №3

Разработка вьюера текстовых файлов

Задание. В среде Delphi разработать вьюер текстовых файлов, позволяющий пользователю просматривать содержимое текстовых файлов, выделять весь текст, копировать его в буфер обмена.

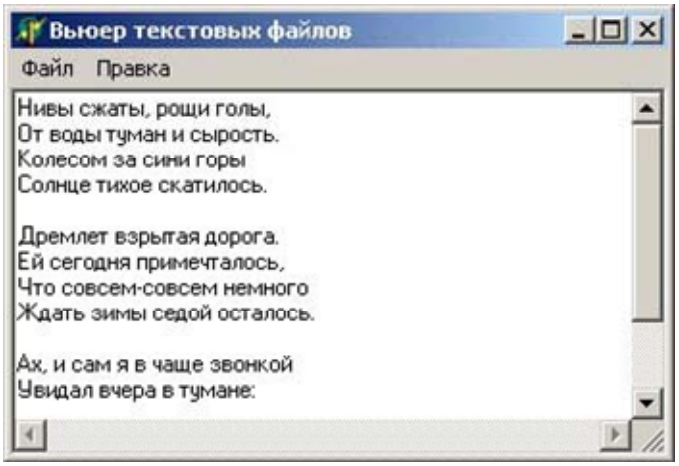

Рис. 4. Вьюер текстовых файлов

1. Запустите IDE Delphi.

2. Сохраните проект с именем Lister в отдельной папке.

3. В заголовке окна приложения напишите *Вьюер текстовых файлов*.

4. Поместите на форму компонент **MainMenu** (**MainMenu1**) – главное меню. С помощью Конструктора меню создайте раздел главного меню Файл, содержащий две команды Открыть… и Выход и раздел Правка, содержащий две команды Копировать и Выделить все (см. п. 4 лабораторной работы № 2).

5. Со страницы Standard палитры компонентов поместите на форму компонент **Memo** (**Memo1**) – многострочное окно редактирования, предназначенное для ввода и отображения многострочных тестов.

Измените значение следующих его свойств:

ReadOnly:=*true* – запрет на редактирование пользователем просматриваемого теста.

HideSelection:=*false* – помечать цветом выделенный текст.

ScrollBars:=*ssBoth* – наличие горизонтальной и вертикальной полос прокрутки.

Align:=*alClient* – занимать всю свободную площадь формы – клиентскую область.

Удалите текст «Memo1», содержащийся в окне **Memo1**. Для этого в Object Inspector выделите свойство **Lines** этого компонента. Это свойство содержит текст окна **Memo** в виде списка строк. Нажмите кнопку с многоточием  около имени этого свойства и сотрите текст в открывшемся окне редактирования списка строк. Нажмите кнопку Ok, чтобы принять изменения и закрыть окно редактирования.

6. Поместите на форму диалог **OpenDialog**  (**OpenDialog1**) и выполните настройку фильтра типов файлов, как описано в п. 7 лабораторной работы №2.

На этом создание графического интерфейса завершено. Вид вьюера приведен на рис. 4.

7. Для команды Открыть… раздела меню Файл в обработчик события **OnClick** напишите следующий код:

```
If OpenDialog1.Execute Then
Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

В первой строке кода записан условный оператор **If**, который вызывает диалоговое окно открытия файлов **OpenDialog1**. Во второй строке записана команда, которая производит загрузку текста в окно **Memo1** из файла, выбранного пользователем, с помощью специального метода **LoadFromFile**.

8. Для команды Выход раздела меню Файл в обработчик напишите следующий код:

```
Close;
```

9. Для команды Копировать раздела меню Правка в обработчик напишите следующий код:

```
Memo1.CopyToClipboard;
```

Эта команда копирует в буфер обмена текст, выделенный пользователем в окне **Memo1**.

10. Для команды Выделить все раздела меню Правка в обработчик напишите следующий код:

```
Memo1.SelectAll;
```

Эта команда выделяет весь текст, содержащийся в окне редактирования.

11. На этом создание вьюера тестовых файлов завершено. Запустите ваше приложение. Убедитесь в его работоспособности. Загрузите в него текст из любого текстового файла. Выделите весь текст, скопируйте его в буфер обмена. Вставьте текст из буфера в стандартный текстовый редактор Блокнот.

Задания для самостоятельной работы

Задача 3. В среде Delphi разработать вьюер текстовых файлов, позволяющий считать из файла, выбранного пользователем, текст в текстовом формате и формате RTF, просмотреть текст, выделить весь текст и скопировать его в буфер обмена.

Указание: в место компонента **Memo** использовать компонент **RichEdit**  , расположенный на странице Win32 палитры компонентов Delphi. Основное отличие этого компонента – возможность работы с текстом в обогащенном формате RTF.

Лабораторная работа №4

Разработка простейшего текстового редактора

Задание. В среде Delphi разработать простейший текстовый редактор, позволяющий считать из текстового файла, выбранного пользователем, текст (несколько строк), редактировать его, копировать в буфер обмена выделенный фрагмент, сохранять текст в текстовый файл, выводить на принтер, производить поиск заданного фрагмента текста.

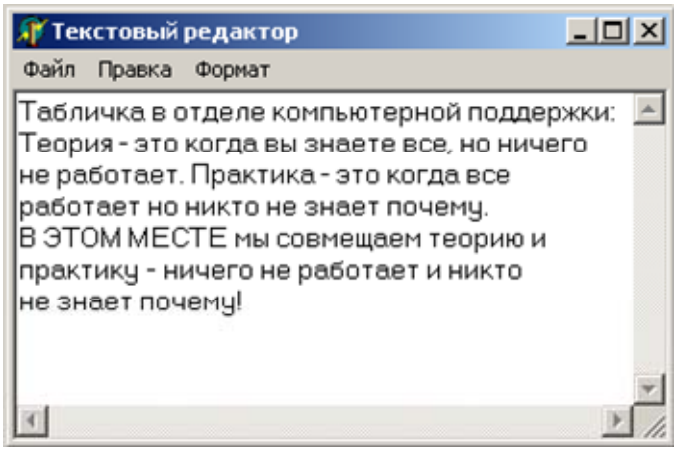

Рис. 5. Простейший текстовый редактор

Примечание: для выполнения этой лабораторной работы можно взять за основу предыдущий проект, так как в Текстовом редакторе будут использоваться те же компоненты и некоторые команды меню, что и во Вьюере текстовых файлов (лабораторная работа №3). Для этого в главном меню Delphi нужно открыть проект Lister с помощью команды File > Open Project или выбрать из списка недавно использовавшихся в каскадном меню File > Reopen. После этого в главном меню Delphi выполнить команду File > Save Project As и сохранить проект с новым именем TextEditor в новую папку. Затем выполнить команду File > Save As и сохранить в эту же новую папку файл модуля. Имя модуля можно оставить прежним (Unit1.pas).

Вы можете переименовать в новую папку предыдущий проект и выполнить нужные изменения в соответствии с пунктами 3 – 11 этой лабора-

торной работы или создать новые проект и сделать все сначала, как описано ниже в п 1 – 11.

1. Запустите IDE Delphi.

2. Сохраните проект с именем TextEditor в отдельной папке.

3. В заголовке окна приложения напишите *Текстовый редактор*.

4. Поместите на форму компонент **MainMenu**  (**MainMenu1**) – главное меню.

С помощью Конструктора меню создайте разделы главного меню:

раздел Файл, содержащий команды: Создать, Открыть…, Сохранить как…, Установки принтера…, Печать…, Выход;

раздел Правка, содержащий команды: Копировать, Вставить, Выделить все, Найти…;

раздел Формат, содержащий команду Шрифт….

После разделов Сохранить как…, Печать… и Выделить все вставьте разделители. Для этого с помощью Конструктора меню создайте новый раздел. В значение его свойства **Caption** напишите символ минус «–».

Примечание: если разделы уже созданы, то между ними можно вставить новый раздел с помощью всплывающего (контекстного) меню. Для этого в Конструкторе меню выделите раздел меню, перед которым требуется вставить новый раздел, щелкните правой кнопкой мыши и выберите из всплывающего меню команду Insert.

5. Поместите на форму компонент **Memo**  (**Memo1**) – многострочное окно редактирования.

Измените значение следующих его свойств:

HideSelection:=*false* – помечать цветом выделенный текст.

ScrollBars=*ssBoth* – наличие горизонтальной и вертикальной полос прокрутки.

Align:=*alClient* – занимать всю свободную площадь формы – клиентскую область.

ReadOnly:=*false* – возможность редактировать текст.

Удалите текст «Memo1», содержащийся в окне **Memo1**, очистив содержимое свойства **Lines**, как это описано в п. 5 лабораторной работы №3.

6. Поместите на форму диалоги **OpenDialog**  (**OpenDialog1**) и **SaveDialog**  (**SaveDialog1**). Выполните настройку их свойств, как описано в п. 7 лабораторной работы №2.

7. Со страницы Dialogs палитры компонентов поместите на форму следующие диалоги: **PrinterSetupDialog**  (**PrinterSetupDialog1**) – диалог Настройка печати, **PrintDialog**  (**PrintDialog1**) – диалог печати, **FindDialog**  (**FindDialog1**) – диалог поиска текста, **FontDialog**  (**FontDialog1**) – диалог выбора шрифта. Все эти компоненты невизуальные, и поэтому местоположение их на форме не имеет значения.

8. Для упрощения алгоритма поиска текста, который будет реализован в п.19, удалим из диалога **FindDialog1** некоторые опции: Только слово целиком, С учетом регистра, Вверх, Вниз. Для этого установите значение *true* следующим его свойствам из группы **Options**:

frHideWholeWord:=*true* – отключение опции Только слово целиком;

frHideMatchCase:=*true*– отключение опции С учетом регистра;

frHideUpDown:=*true* – отключение опций Вверх и Вниз.

На этом создание графического интерфейса завершено. Вид текстового редактора приведен на рис. 5.

9. Для команды Создать… раздела меню Файл в обработчик события **OnClick** напишите следующий код:

```
Memo1.Clear;
```

Этот метод производит очистку текста в окне редактирования **Memo1**.

10. Для команды Открыть… раздела меню Файл в обработчик события **OnClick** напишите следующий код:

```
If OpenDialog1.Execute Then
Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

В этом обработчике производится загрузка текста из файла, выбранного пользователем в диалоге **OpenDialog1**, в окно **Memo1**.

11. Для команды Сохранить как… раздела меню Файл в обработчик напишите следующий код:

```
If SaveDialog1.Execute Then
Memo1.Lines.SaveToFile(SaveDialog1.FileName);
```

С помощью метода **SaveToFile** производится сохранение текста из окна **Memo1** в файл, выбранный пользователем в диалоге **SaveDialog1**.

12. Для команды Установки принтера… раздела меню Файл в обработчик напишите следующий код:

```
PrinterSetupDialog1.Execute;
```

Эта строка вызывает диалог Настройка печати – **PrinterSetupDialog1**.

13. Теперь приступим к написанию обработчика события команды Печать… раздела меню Файл.

Печать в Delphi может осуществляться несколькими способами. Некоторые компоненты имеют методы, обеспечивающие печать хранящихся в них данных. К таким компонентам относится **RichEdit**, расположенный на

странице Win32 палитры компонентов Delphi. Чтобы распечатать текст, содержащийся в окне этого компонента, нужно просто вызвать метод **Print** с помощью оператора: **RichEdit1.Print('Имя задания, отображаемое в очереди печати')**. К сожалению, используемый нами компонент **Memo** такого метода не имеет. Поэтому мы применим другой способ – с помощью функций файлового ввода–вывода, подобно тому, как это было описано в п. 10 лабораторной работы №2. Однако связывание выходного потока будет происходить не с текстовым файлом, а с принтером. Этот метод позволяет печатать только простой текст, не содержащий графики и других объектов.

Сначала подключите модуль Delphi, который называется **Printers**. Для этого в начале текста модуля, после ключевого слова **Uses**, к уже подключенным модулям через запятую добавьте модуль **Printers**. После этого этот фрагмент кода должен выглядеть, например, так:

```
uses
  Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, Menus, StdCtrls,
ComCtrls, Printers;
```

Затем создайте событие **OnClick** для команды Печать… раздела меню Файл и напишите код, выполняющий печать текста, хранящегося в компоненте **Memo1**. Обработчик этого события должен иметь вид:

```
procedure TForm1.N11Click(Sender: TObject); {эту строку
писать не надо, так как это заголовок события, который автоматически
добавлен Delphi при создании события}
Var
PrintTxt:TextFile;
i:integer;
begin // эту строку писать не надо, она добавлена Delphi автоматически
```

```
If PrintDialog1.Execute Then
begin
AssignPrn(PrintTxt);
Rewrite(PrintTxt);
For i:=0 to Memo1.Lines.Count -1 do
Writeln(PrintTxt,Memo1.Lines[i]);
CloseFile(PrintTxt);
end;
end; // эту строку писать не надо, она добавлена Delphi автоматически
```

Прокомментируем приведенный код. После заголовка процедуры – обработчика события – мы объявили файловую переменную `PrintTxt` типа `TextFile` и управляющую переменную `i` целого типа, в которой будет храниться текущее состояние счетчика цикла **For**.

В исполняемой части процедуры, после оператора **begin**, с помощью условного оператора **If** вызывается диалог печати **PrintDialog1**. Если пользователь в диалоговом окне нажимает на кнопку ОК, то выполняется вывод текста на принтер с помощью операторов, идущих после слова **Then** и объединенных в блок **begin**…**end**. Для этого используется разновидность команды **Assign** – **AssignPrn**. Она настраивает переменную `PrintTxt` на порт принтера, обращаясь к нему как к файлу. Затем порт принтера открывается командой **Rewrite**.

Далее следует оператор цикла **For**. В свойстве **Count** хранится количество строк текста, содержащегося в свойстве **Lines** компонента **Memo1**. Все индексы в Delphi начинаются с нуля, поэтому первая строка будет иметь индекс равный *0*, а последняя *Memo1.Lines.Count-1*. Таким образом, оператор цикла **For** выполняется *Memo1.Lines.Count* раз, изменяя значения управляющей переменной `i` от *0* до *Memo1.Lines.Count-1*. При этом процедура **Writeln** построчно, с перовой строки до последней, передает текст на принтер.

Командой **CloseFile** порт принтера закрывается.

14. Для команды Выход раздела меню Файл в обработчик напишите следующий код:

```
Close;
```

15. Для команды Копировать раздела меню Правка в обработчик напишите следующий код:

```
Memo1.CopyToClipboard;
```

Эта команда копирует в буфер обмена текст, выделенный пользователем в окне **Memo1**.

16. Для команды Вставить раздела меню Правка в обработчик напишите следующий код:

```
Memo1.PasteFromClipboard;
```

Эта команда вставляет в окно **Memo1** текст из буфера обмена.

17. Для команды Выделить все раздела меню Правка в обработчик напишите следующий код:

```
Memo1.SelectAll;
```

Эта команда выделяет весь текст, содержащийся в окне редактирования.

18. Для команды Найти… раздела меню Правка в обработчик напишите следующий код:

```
FindDialog1.Execute;
```

Эта команда вызывает диалог поиска текста **FindDialog1**.

19. Реализуем механизм поиска заданного фрагмента текста. Сам по себе компонент **FindDialog** не осуществляет поиска, а обеспечивает диалог с пользователем. Поиск должен осуществляться программно. Для этого

можно использовать некоторые функции для работы со строками Object Pascal*, синтаксис и описание которых приведено ниже.

Pos(SubS, S: string): integer – возвращает позицию, начиная с которой в строке **S** располагается подстрока **SubS**. Если **S** не содержит **SubS**, то функция возвращает *0*.

Copy(S: string; Index, Count: integer): string – возвращает подстроку из строки **S**, начиная с позиции и длиной **Count** символов.

Length(S: string): integer – возвращает число символов в строке **S**.

У компонента **FindDialog1** создайте событие **OnFind**, которое происходит, когда пользователь нажал в диалоге кнопку Найти далее. Текст кода этого обработчика, реализующего поиск, должен иметь следующий вид:

```
{эту строку писать не надо, она автоматически добавлена при создании события}
procedure TForm1.FindDialog1Find(Sender: TObject);
Var
StartPos,SearchPos:integer;
begin // эту строку писать не надо, она добавлена Delphi автоматически
{Запоминание начальной позиции поиска}
StartPos:=Memo1.SelStart+Memo1.SelLength;
{Определение позиции искомого фрагмента текста}
SearchPos:=Pos(FindDialog1.FindText,
Copy(Memo1.Lines.Text,StartPos+1,
Memo1.GetTextLen-StartPos));
{Проверка наличия искомой строки в тексте}
If SearchPos=0 Then
```

* У компонента **RichEdit** есть специальная функция, с помощью которой можно выполнять поиск текста: **FindText(const SearchStr: string; StartPos, Length: Integer; Options: TSearchTypes): Integer;** Эта функция возвращает позицию, начиная с которой в тексте располагается искомая строка **SearchStr**. Поиск производится на отрезке текста длиной **Length**, начиная с позиции **StartPos**. С помощью **Options** можно задать условия поиска. Этот параметр может принимать значения: [*stMatchCase*] – учитывая регистр; [*stWholeWord*] – слово полностью.

```
{Если строка не найдена, то пользователю выдается сообщение}
Application.MessageBox('Не удается найти заданный
фрагмент текста', 'Текстовый редактор',
MB_Ok+ MB_ICONINFORMATION)
Else
  {Если строка найдена}
  begin
    {Определение начала выделения текста}
    Memo1.SelStart:=StartPos+SearchPos-1;
    {Выделение найденного текста}
    Memo1.SelLength:=Length(FindDialog1.FindText);
  end;
{Активизация окна редактирования (передача ему фокуса)}
Memo1.SetFocus;
end; // эту строку писать не надо, она добавлена Delphi автоматически
```

В обработчике, приведенном выше, вначале мы объявили две переменных целого типа: `StartPos` и `SearchPos`. В переменной `StartPos` будет храниться позиция, начиная с которой будет производиться поиск, а в переменной `SearchPos` позиция, начиная с которой в тексте располагается найденный фрагмент текста.

Код исполняемой части процедуры-обработчика начинается с сохранения в переменной `StartPos` позиции, с которой будет выполняться поиск. Свойство **SelStart** компонента **Memo** содержит позицию курсора в тексте, с которой начинается поиск (или начало прошлого выделения текста, если производится дальнейший поиск в оставшейся части текста), а **SelLength** – количество выделенных символов при предыдущем поиске (в начале поиска значение этого свойства равно нулю).

Строковая функция **Pos** возвращает позицию, с которой в тексте начинается первый символ искомого фрагмента. Текст, заданный пользовате-

лем для поиска, хранится в свойстве **FindText** диалога **FindDialog1**. Весь текст окна редактирования, содержащийся в свойстве **Lines** компонента **Memo1**, в виде одной строки хранится в свойстве **Text**, а количество символов в нем определяется методом **GetTextLen** этого компонента. Поиск выполняется не во всем тексте, а только начиная с позиции курсора или при продолжении поиска с позиции, расположенной после последнего символа найденного фрагмента, и до конца текста. Для этого используется функция **Copy**. Начало поиска определяется переменной `StartPos`. К ней прибавляется единица, так позиции курсора и символа в строке различаются (см. рис. 6). Длина оставшейся части текста, в которой выполняется поиск, определяется как разность длины всей строки и позиции начала поиска.

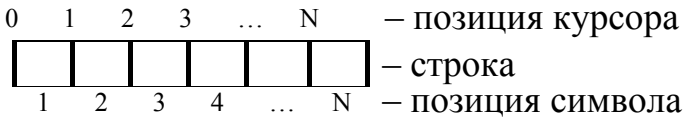

Рис. 6.

Далее выполняется проверка наличия искомой строки в тексте. Если искомый фрагмент текста не найден, то функция **Pos** в предыдущем операторе возвращает в переменную `SearchPos` значение *0*. В этом случае с помощью метода **Application.MessageBox** пользователю выдается сообщение «Не удается найти заданный фрагмент текста». Диалоговое окно, вызываемое этим методом, будет содержать заголовок «Текстовый редактор», указывающий на приложение, из которого она вызвано, кнопку OK, закрывающую окно, и пиктограмму в виде символа i в круге. Наличие кнопки и пиктограммы определяется параметрами-флагами *MB_Ok+ MB_ICONINFORMATION*.

Если текст, заданный пользователем для поиска, обнаружен, то он выделяется в окне редактирования. Для этого свойству **SelStart**, которое определяет начало выделения, присваивается позиция найденного фраг-

мента текста, отсчитываемая от начала всего текста. Она складывается из позиции, с которой начинался поиск `StartPos`, позиции первого символа найденного фрагмента `SearchPos`, которая отсчитывалась от позиции начала поиска. Из этой суммы вычитается единица, так как в этот раз мы определяем позицию курсора, с которой начнется выделение, а не позицию символа в строке (см. рис. 6). Затем с помощью свойства **SelLength** выполняется выделение фрагмента текста. Количество выделяемых символов должно соответствовать длине искомого фрагмента, содержащегося в свойстве **FindText** диалога **FindDialog1**. Эта длина определяется строковой функцией **Length**.

В завершение процедуры-обработчика управление (фокус) от диалогового окна **FindDialog1** предается окну редактирования **Memo1** с помощью метода **SetFocus**.

Если пользователь в диалоговом окне **FindDialog1** повторно нажмет кнопку Найти далее, то у этого диалога снова произойдет событие **OnFind**, и процедура-обработчик события выполнится снова, продолжая поиск в оставшейся части текста.

20. Для команды Шрифт… раздела меню Формат в обработчик напишите следующий код:

```
If FontDialog1.Execute Then
Memo1.Font:=FontDialog1.Font;
```

Эта команда вызывает диалоговое окно выбора шрифта **FontDialog1**. Все атрибуты шрифта хранятся в его свойстве **Font**. После того как пользователь изменит атрибуты и нажмет кнопку ОК, значение этого свойства присваивается свойству **Font** компонента **Memo1**, изменяя шрифт в окне редактирования.

21. На этом создание простейшего тестового редактора завершено. Запустите ваше приложение. Убедитесь в его работоспособности. Напиши-

те любой текст в окне редактирования. Выделите фрагмент этого текста, скопируйте его в буфер обмена. Вставьте текст из буфера. Выполните поиск любого фрагмента текста. Измените атрибуты шрифта текста. Напечатайте текст. Сохраните текст в файл. Очистите окно редактирования командой Создать. Откройте ранее сохраненный файл.

Задания для самостоятельной работы

Задача 4. В среде Delphi разработать простейший текстовый редактор, позволяющий считывать из файла, выбранного пользователем, текст в текстовом формате и формате RTF, редактировать его, изменять шрифт абзаца, копировать в буфер обмена выделенный фрагмент, вставлять текст из буфера, производить поиск текста, сохранять текст в файл текстового формата и формата RTF, выводить на принтер.

Указание: в место компонента **Memo** использовать компонент **RichEdit**  , расположенный на странице Win32 палитры компонентов Delphi. Основное отличие этого компонента – возможность работы с текстом в обогащенном формате RTF.

Лабораторная работа №5

Разработка вьюера графических файлов

Задание. В среде Delphi разработать вьюер графических файлов, позволяющий просматривать выбранные пользователем рисунки в форматах *.bmp, *.jpg.

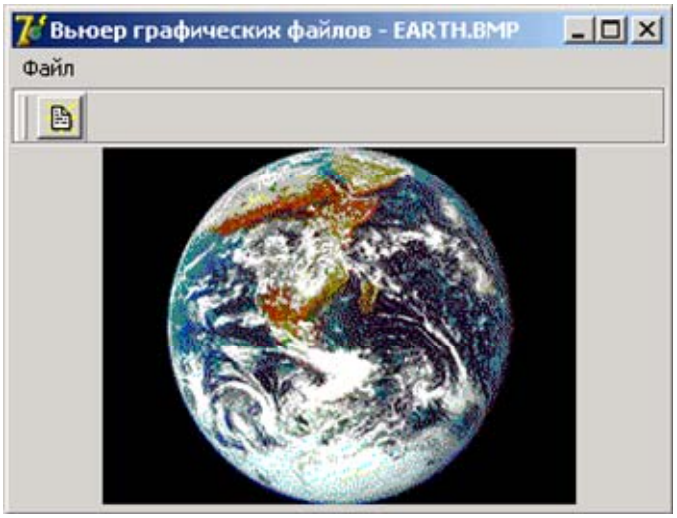

Рис. 7. Вьюер графических файлов

1. Запустите IDE Delphi.

2. Сохраните проект с именем Viewer в отдельной папке.

3. В заголовке окна приложения напишите *Вьюер графических файлов*.

4. Поместите на форму компонент **MainMenu** (**MainMenu1**) – главное меню. С помощью Конструктора меню создайте раздел главного меню Файл, содержащий две команды: Открыть… и Выход.

5. Со страницы Dialogs палитры компонентов поместите на форму диалог **OpenPictureDialog** (**OpenPictureDialog1**) – специализированный диалог открытия графических файлов. У этого диалога имеется возможность предпросмотра рисунка в процессе выбора файла, а также в свойстве **Filter** уже заданы типы графических файлов.

6. Со страницы Additional палитры компонентов поместите на форму компонент **ControlBar** (**ControlBar1**) – перестраиваемую инструмен-

тальную панель. С помощью этого компонента в данной лабораторной работе будет создаваться обычная инструментальная панель, предназначенная для быстрого доступа пользователей к часто используемым командам главного меню. Следует заметить, что в Delhi для этих целей существует специальный компонент **ToolBar**, расположенный на странице Win32 палитры компонентов. Однако создание инструментальных панелей с помощью этого компонента будет рассмотрено позже, в лабораторной работе № 7.

У компонента **ControlBar1** задайте свойству **Align** значение *alTop*, чтобы поместить инструментальную панель вверху формы, под главным меню. А в значение свойства **Height** напишите число *29*.

7. Продолжим создание инструментальной панели. На инструментальную панель – компонент **ControlBar1** – со страницы Additional палитры компонентов поместите компонент **SpeedButton**  (**SpeedButton1**) – быструю кнопку. Эта кнопка будет дублировать команду Открыть… главного меню. Зададим пиктограмму, которая будет на этой кнопке. Для этого в Object Inspector нажмите кнопку с многоточием  рядом со свойством **Glyph** этого компонента. При этом откроется окно Редактора рисунка. В нем нажмите кнопку Load. И выберите на диске, на котором установлена Delphi, в папке \Program Files \Common Files \Borland Shared \Images \Buttons\ файл с изображением fileopen.bmp. В этой папке расположены графические файлы, поставляемые с Delphi и содержащие пиктограммы для кнопок. После того как вы выбрали изображение, нажмите в Редакторе рисунка кнопку OK, и выбранная пиктограмма появится на кнопке **SpeedButton1**.

8. Со страницы Standard палитры компонентов поместите на форму компонент **ActionList**  (**ActionList1**) – список событий. Этот компонент используется, когда требуется выполнить одно и то же действие разными

элементами управления. Как, например, в нашем случае открыть файл командой меню или быстрой кнопкой инструментальной панели. Если не использовать этот компонент, то пришлось бы писать один и тот же код в обработчики события **OnClick** как у команды меню Открыть…, так и у быстрой кнопки. Компонент **ActionList** не визуальный.

Щелкните два раза по компоненту **ActionList1** и в открывшемся редакторе нажмите быструю кнопку  New Action , чтобы создать новое действие **Action1**. В правом окне Actions редактора выделите имя созданного действия **Action1**. В Object Inspector сотрите значение свойства **Caption**. Закройте окно редактора.

Теперь свяжем созданное действие с элементами управления – командой главного меню Открыть… и быстрой кнопкой **SpeedButton1**. Перейдите к компоненту **MainMenu1** и двойным щелчком откройте Конструктор меню. Выберите пункт Открыть…. В его свойстве **Action**, рядом с именем свойства, раскройте выпадающий список и выберите в нем действие *Action1*. В свойстве **Caption** этого раздела меню напишите *Открыть*...., так как содержащееся там до этого значение было замещено пустым значением из свойства **Caption** действия **Action1**. Закройте окно Конструктора меню. Перейдите к компоненту **SpeedButton1**. В его свойстве **Action** также выберите действие *Action1*.

9. Со страницы Additional палитры компонентов поместите на форму компонент **Image**  (**Image 1**) – компонент, предназначенный для отображения графики: битовых матриц, пиктограмм.

Измените значение следующих его свойств:

Align=*alClient* – развернуть компонент на всю свободную площадь формы – клиентскую область.

Center=*true* – включение центрирования изображения по площади компонента **Image1**, если размер рисунка меньше размера компонента.

На этом создание графического интерфейса завершено.

10. Чтобы была возможность просматривать графические файлы в формате *.jpg, подключим стандартный модуль **JPEG**. Для этого в начале текста модуля, после ключевого слова **Uses**, к уже подключенным модулям через запятую добавьте модуль **JPEG**, после чего этот фрагмент кода должен выглядеть, например, так:

```
uses
  Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, Menus, StdCtrls,
ComCtrls, JPEG;
```

11. Теперь у действия **Action1** создадим событие **OnExecute** и в его обработчик запишем код, который будет выполняться как при выборе пользователем пункта меню Открыть…, так и при нажатии им на соответствующую быструю кнопку инструментальной панели. Для этого двойным щелчком мыши по компоненту **ActionList1** откройте окно редактора списка событий. В правом окне Actions редактора выделите имя действия **Action1**. В Object Inspector на вкладке Events сделайте двойной щелчок в окне справа от имени события **OnExecute**.

В обработчик события напишите следующий код:

```
If OpenPictureDialog1.Execute Then
begin
Image1.Picture.Assign(nil);
Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
Form1.Caption:= 'Вьюер графических файлов' +' - '+
ExtractFileName(OpenPictureDialog1.FileName);
end;
```

Вначале вызывается диалог открытия графических файлов. Затем, в случае, если пользователь выбрал файл, производится удаление предыдущего изображения. Для этого свойству **Picture**, которое содержит изобра-

жение, с помощью метода **Assign** присваивается предопределенная константа **nil**. После этого в свойство **Picture** с помощью метода **LoadFromFile** загружается изображение из файла.

Последние две строки изменяют заголовок окна приложения. Обычно в приложениях, просматривающих файлы, текст заголовка содержит имя приложения и имя просматриваемого файла, разделенные дефисом. Для этого к свойству **Caption** формы **Form1**, добавляются строки содержащие имя приложения, дефис, а также строка, содержащая имя и расширение открытого файла.

Строка, содержащая имя и расширение файла, выделяется из его полного имени функцией **ExtractFileName**. Сцепление строк – конкатенация – производится с помощью знака «+».

Примечание: в редакторах документов, наоборот, в тексте заголовка на первом месте располагается имя документа, затем через дефис – имя приложения. Если приложение не связано с каким-либо документом, то в тесте заголовка содержится только имя приложения.

12. Для команды Выход раздела меню Файл в обработчик напишите следующий код:

```
Close;
```

13. На этом создание простейшего вьюера графических файлов завершено. Запустите ваше приложение. Убедитесь в его работоспособности. С помощью команды меню откройте графический файл в формате *.bmp, например, Earth.bmp, расположенный в папке \Program Files \Common Files \Borland Shared \Images \Splash \16Color\ диска, на котором установлена Delphi (вид вьюера с этим рисунком приведен на рис. 7). Обратите внимание на имя открытого файла в заголовке окна вашего приложения. Откройте с помощью быстрой кнопки инструментальной панели какой-нибудь другой рисунок в формате *.jpg. При выборе графических файлов

обратите внимание в диалоговом окне на панель предварительного просмотра рисунков.

Задания для самостоятельной работы

Задача 5. В среде Delphi разработать вьюер графических файлов, позволяющий просматривать выбранные пользователем рисунки в форматах *.bmp, *.jpg. Большие по размеру изображения должны полностью помещаться в область просмотра, кроме того, предусмотреть опцию, позволяющую пользователю разместить рисунок в углу или по центру окна.

Лабораторная работа №6

Разработка простейшей программы научной графики

Задание. В среде Delphi разработать приложение, позволяющее строить двухмерные графики по числовым данным, введенным пользователем с клавиатуры, сохранять эти данные в файл (формат файла: два столбца, разделенные пробелом и содержащие данные по осям X и Y) и строить двухмерные графики по числовым данным, считанным из этого файла.

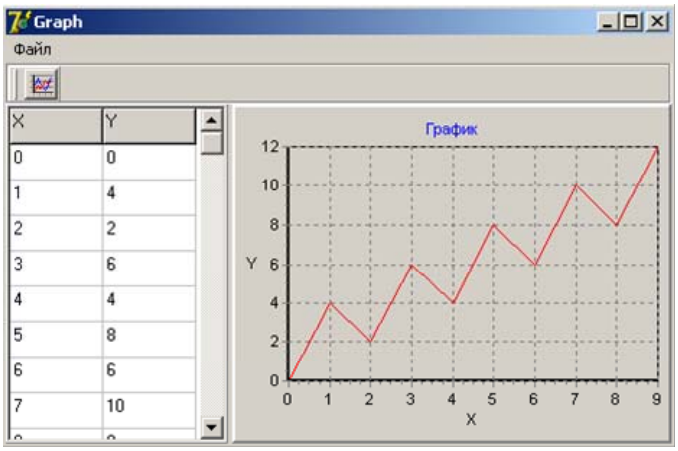

Рис. 8. Простейшая программа научной графики

Для упрощения задачи реализуем программу, которая будет строить график по десяти точкам, т.е. по десяти парам координат X и Y, считанным из файла или введенным с клавиатуры.

1. Запустите IDE Delphi.

2. Сохраните проект с именем Graph в отдельной папке.

3. В заголовке окна приложения напишите *Graph*.

4. Поместите на форму компонент **MainMenu** (**MainMenu1**) – главное меню. С помощью Конструктора меню создайте раздел главного меню Файл, содержащий три команды: Открыть…, Сохранить как… и Выход. Между разделами Сохранить как… и Выход вставьте разделитель.

5. Со страницы Additional палитры компонентов поместите на форму компонент **ControlBar**  (**ControlBar1**) – перестраиваемую инструментальную панель.

Измените значение следующих его свойств:

Align:=*alTop* – поместить инструментальную панель вверху формы, под главным меню.

Height:=*29* – задать высоту панели.

6. На компонент **ControlBar1** со страницы Additional палитры компонентов поместите компонент **SpeedButton**  (**SpeedButton1**) – быструю кнопку. Задайте для этой кнопки пиктограмму, загрузив в свойство **Glyph** файл с изображением grphline.bmp из папки \Program Files \Common Files \Borland Shared \Images \Buttons\ на диске, на котором установлена Delphi. С помощью этой кнопки мы будем строить график.

Создадим всплывающее окно подсказки о назначении этой кнопки. Ярлычок с текстом подсказки будет появляться, когда пользователь задержит курсор мыши над этой кнопкой. Для этого в свойстве **Hint** этой быстрой кнопки запишите текст подсказки *Построить график* и включите показ всплывающего окна подсказки, выбрав значение *true* в свойстве **ShowHint**.

7. Поместите на форму диалоги **OpenDialog**  (**OpenDialog1**) и **SaveDialog**  (**SaveDialog1**).

8. Выполните настройку диалогов. В свойстве **Filter** обоих диалогов задайте два фильтра: файлы данных с расширением dat (*.dat) и все файлы с любым именем и любым расширением (*.*). Кроме того, для диалога **SaveDialog1** в свойстве **DefaultExt** напишите значение *dat* – расширение, которое будет по умолчанию добавляться к именам файлов при их сохра-

нении, а в свойстве **Options** и включите опцию **ofOverwritePrompt** – запрос на перезапись уже существующего файла.

9. Поместите на форму компонент **Panel**  (**Panel1**) – панель со страницы Standard палитры компонентов. Этот компонент предназначен для планировки графического интерфейса пользователя. С его помощью можно компоновать различные элементы управления в окне формы. Свойству **Align** этого компонента задайте значение *alLeft*, чтобы поместить панель в левой части формы. Задайте ширину панели, записав в ее свойство **Width** значение *155*. Сотрите значение свойства **Caption**, чтобы убрать надпись «Panel1» с панели.

10. Со страницы Additional поместите на форму компонент **Splitter**  (**Splitter1**) – разделитель панелей, который используется для создания панелей с изменяемыми пользователем размерами. Задайте выравнивание этого компонента по левому краю, выбрав в его свойстве **Align** значение *alLeft*. При этом разделитель прижмется к краю панели **Panel1**.

11. Поместите на форму еще один компонент **Panel**  (**Panel2**) со страницы Standard палитры компонентов.

Измените значение следующих его свойств:

Align:=*alClient* – развернуть панель на всю свободную площадь формы – клиентскую область.

Caption:=’’ – убрать надпись «Panel2» с панели.

Теперь, если запустить приложение, то с помощью мыши, потянув за разделитель, можно изменить положение границы между панелями **Panel1** и **Panel2**. Кроме того, размеры панелей будут автоматически изменяться при изменении размеров формы, например, при разворачивании окна на весь экран.

12. Поместите на компонент **Panel1** (левую панель) компонент **StringGrid**  (**StringGrid1**) – таблицу строк, расположенную на странице Additional.

В этой таблице будут содержаться координаты по осям X и Y, загруженные в нее из файла или введенные с клавиатуры, по которым будет построен график.

Зададим в таблице два столбца и одиннадцать строк. Для этого у компонента **StringGrid1** измените значение следующих ее свойств:

ColCount:=*2* – число столбцов;

RowCount:=*11* – число строк.

Верхняя строка таблицы будет предназначаться для отображения заголовков столбцов, поэтому оставим ее фиксированной, то есть всегда видимой сверху таблицы (эта строка выделена серым цветом).

Уберите фиксацию с первого столбца таблицы, для этого в свойстве **FixedCols** задайте значение, равное *0*.

У свойства **Align** выберите значение *alClient*, чтобы таблица занимала всею площадь панели.

Для того чтобы иметь возможность редактировать содержимое таблицы, в свойстве **Options** включите опцию **goEditing**, задав ей значение *true*.

13. Со страницы Additional поместите на компонент **Panel2** (правую панель) компонент **Chart**  (**Chart1**) – диаграмма, предназначенный для построения диаграмм и графиков. У его свойства **Align** выберите значение *alClient*, чтобы развернуть компонент на всю клиентскую область панели **Panel2**.

14. Выполним настройку компонента **Chart1**. Вызовите вспомогательное окно Редактора диаграмм, сделав двойной щелчок мышью на компоненте **Chart1**.

Добавьте специальный объект **Series** – серию, с помощью которой в виде кривой будут отображаться данные на графике. Каждая серия соответствует одной кривой на графике. Для этого на странице Chart Редактора диаграмм на вкладке Series щелкните на кнопке Add. В открывшемся еще одном окне – TeeChart Gallery – выберите тип диаграммы Line – точечную диаграмму со значениями, соединенными линиями. Нажмите кнопку Ok.

Подпишите оси. Для этого перейдите на вкладку Axis страницы Chart Редактора Диаграмм. На ней в группе радиокнопок Axis выберите радиокнопку Left – левую ось (она уже должна быть выбранной по умолчанию). После этого перейдите еще на одну вкладку Title, которая расположена на этой же вкладке. На ней в поле Title: напишите *Y* – название левой (вертикальной) оси. Измените ориентацию этой надписи, выбрав в списке числовых значений Angle: значение *0*. Подпишите нижнюю (горизонтальную) ось, выбрав в группе Axis радиокнопку Bottom. Для этой оси на вкладке Title в поле Title: напишите *X* – название нижней оси.

Задайте заголовок всей диаграмме. Для этого перейдите на вкладку Titles страницы Chart Редактора Диаграмм. В текстовом поле на этой странице вместо *TChart* напишите название диаграммы *График*.

Удалите легенду диаграммы (список обозначений), расположенную в левой части диаграммы. Для этого перейдите на вкладку Legend и снимите флажок с переключателя Visible.

Отключите трехмерное отображение графиков, сняв флажок с переключателя 3 Dimensions на вкладке 3D.

Закройте окно Редактора диаграмм, нажав кнопку Close.

На этом создание графического интерфейса закончено (см. рис. 8). Перейдем к написанию обработчиков событий. Не забывайте периодически сохранять ваш проект.

15. В Object Inspector в списке компонентов выберите форму **Form1** и создайте для нее событие **OnCreate**, которое происходит при создании формы. В обработчик этого события запишите:

```
StringGrid1.Cells[0,0]:='X';
StringGrid1.Cells[1,0]:='Y';
```

Для того чтобы программно занести текст в какую-либо ячейку таблицы строк **StringGrid1**, нужно ее свойству **Cells**[индекс столбца, индекс строки] присвоить значение строкового типа (индексы в Delphi – целые числа, начинающиеся с нуля). Поэтому код, записанный в этом обработчике, после запуска программы задает заголовки столбцов таблицы, записывая в ячейки фиксированной строки таблицы **StringGrid1** значения «X» и «Y».

16. Опишите глобальную переменную `FileVar` типа Текстовый файл, которая будет использоваться в этой программе для доступа к файлам. Для этого в тексте модуля, ниже ключевого слова **implementation**, запишите:

```
var
  FileVar: TextFile;
```

17. Создайте событие **OnClick** для команды Открыть… раздела меню Файл. Обработчик этого события должен иметь вид:

```
procedure TForm1.N2Click(Sender: TObject); {эту строку писать не надо, она автоматически добавлена при создании события}
Var // Блок описания переменных
i:integer; // Переменная-счетчик цикла
X,Y:real;// Переменные, в которые будут считываться данные из файла
begin // эту строку писать не надо, она добавлена Delphi автоматически
If OpenDialog1.Execute Then // Вызывает диалоговое окно
 begin
```

```
{Связывает имя файла с файловой переменной FileVar}
   AssignFile(FileVar,OpenDialog1.FileName);
{Открывает выбранный файл, связанный с файловой переменной FileVar}
   Reset(FileVar);
   For i:=1 to 10 do // Цикл, построчно заполняющий таблицу
     Begin
{Считывает данные из файла в переменные X и Y}
         Readln(FileVar,X,Y);
{Заносит значения, считанные из файла, в ячейки таблицы}
         StringGrid1.Cells[0,i]:=FloatToStr(X);
         StringGrid1.Cells[1,i]:=FloatToStr(Y);
     end;
   CloseFile(FileVar); // Закрывает файл
 end;
end; // эту строку писать не надо, она добавлена Delphi автоматически
```

Данный обработчик считывает из файла, выбранного пользователем, координаты точек и заносит их в таблицу **StringGrid1**. В начале выполнения процедуры-обработчика вызывается диалоговое окно открытия файлов **OpenDialog1**. Затем, если пользователь выбирает файл, этот файл открывается, и с помощью цикла с **For** из него построчно считываются данные в переменные вещественного типа `X` и `Y`, значения которых, после преобразования в строковые значения с помощью функции **FloatToStr**, присваиваются соответствующим ячейкам таблицы. В конце процедуры-обработчика открытый для чтения файл закрывается.

18. Создайте событие **OnClick** для команды Сохранить как… раздела меню Файл. Обработчик этого события должен иметь вид:

```
procedure TForm1.N3Click(Sender: TObject); {эту строку писать не надо, она автоматически добавлена при создании события}
   Var
```

```
i:integer; // Переменная-счетчик цикла
begin // эту строку писать не надо, она добавлена Delphi автоматически
If SaveDialog1.Execute Then // Вызывает диалоговое окно
 Begin
{Связывает имя файла с файловой переменной FileVar}
   AssignFile(FileVar,SaveDialog1.FileName);
{Создает и открывает новый файл, связанный с файловой переменной FileVar }
   Rewrite(FileVar);
   For i:=1 to 10 do {Цикл, построчно сохраняющий значения из таблицы в файл}
{Записывает значение из таблицы в файл, связанный с файловой переменной FileVar}
   Writeln(FileVar,StringGrid1.Cells[0,i],'  '{'пробел'},
   StringGrid1.Cells[1,i]);
   CloseFile(FileVar); // Закрывает файл
 end;
end; // эту строку писать не надо, она добавлена Delphi автоматически
```

Прокомментируем приведенный код. После вызова диалогового окна сохранения файлов и создания и открытия файла для записи в него с помощью цикла **For** построчно записываются значения ячеек таблицы **StringGrid1**, разделяемые пробелом. В конце процедуры-обработчика открытый для чтения файл закрывается.

19. Для команды Выход раздела меню Файл в обработчик напишите следующий код:

```
Close;
```

20. Создайте событие **OnClick** для быстрой кнопки **SpeedButton1**. Обработчик этого события должен иметь вид:

```
procedure TForm1.SpeedButton1Click(Sender:TObject);{эту
```
строку писать не надо, она автоматически добавлена при создании события}

```
Var i:integer; // Переменная-счетчик цикла
begin // эту строку писать не надо, она добавлена Delphi автоматически
Chart1.Series[0].Clear; // Программная очистка серии (графика)
For i:=1 to 10 do //Программное заполнение серии – построение графика
Chart1.Series[0].AddXY(StrToFloat(StringGrid1.Cells[0,i]),
StrToFloat(StringGrid1.Cells[1,i]),'',clTeeColor);
end; // эту строку писать не надо, она добавлена Delphi автоматически
```

После нажатия пользователем на быструю кнопку **SpeedButton1** эта процедура-обработчик выполняет построение графика (заполняет серию) по данным, которые содержаться в таблице **StringGrid1**. Сначала с помощью метода **Clear** серия очищается (удаляются предыдущие данные). Затем с помощью цикла **For** производится поточечное заполнение серии, используя метод **AddXY**, который добавляет в диаграмму новую точку. Синтаксис этого метода:

AddXY (X, Y: Double; XLabel: String; Color: TColor): Longint;

где **X** и **Y** – переменные вещественного типа, содержащие значения координат текущей точки, **XLabel** – название текущей точки, которое отображается на диаграмме и в легенде (это параметр не обязательный и в данном примере является пустым – *''*), **Color** – цвет графика (*clTeeColor* – цвет серии, заданный при ее создании в окне Редактора диаграмм).

21. На этом создание простейшей программы научной графики завершено. Запустите ваше приложение. Заполните таблицу координат, введя с клавиатуры, например, следующие значения: $X_1=0$, $Y_1=0$; $X_2=1$, $Y_2=1$; $X_3=2$, $Y_3=2$ и т.д. Задержите курсор мыши над быстрой кнопкой, дождавшись появления окна подсказки о ее назначении. Постройте график, нажав на быструю кнопку.

С помощью мыши потяните за разделитель между панелями, на которых расположены таблица и диаграмма, и измените положение границы между этими панелями.

Измените масштаб изображения графика. Для этого, нажав и удерживая левую кнопку мыши, переместите курсор мыши с левого верхнего в правый нижний угол прямоугольной области, которую вы хотите увидеть подробно. Восстановите масштаб изображения, удерживая левую кнопку мыши и перемещая курсор в противоположном направлении.

Переместите наблюдаемую часть изображения графика, перемещая курсор по диаграмме при нажатой правой кнопке мыши. Возврат к первоначальной позиции изображения аналогичен восстановлению масштаба.

Измените в таблице несколько значений координаты Y. Сохраните координаты в файл. Откройте этот файл и снова постройте график.

Задания для самостоятельной работы

Задача 6. В среде Delphi разработать приложение, позволяющее строить двухмерные графики по числовым данным, введенным пользователем с клавиатуры, сохранять эти данные в файл и строить двухмерные графики по числовым данным, считанным из этого файла. Файл должен содержать координаты точек по осям X и Y в виде дух столбцов, разделенных пробелом, количество строк произвольное.

Лабораторная работа №7

Разработка программы, рисующей различные изображения

Задание. В среде Delphi разработать приложение, рисующее домик и ракету.

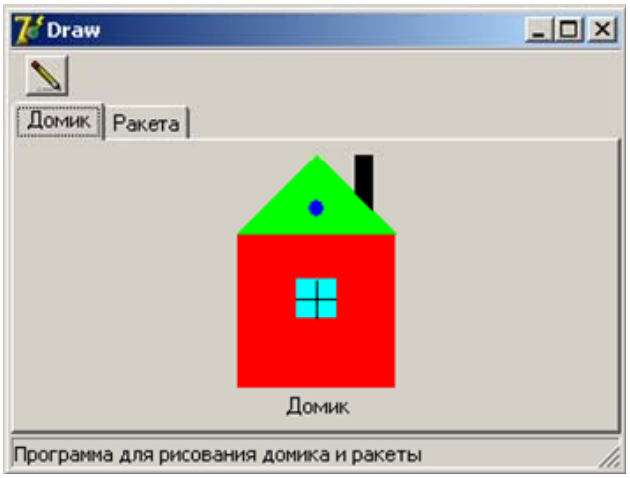

Рис. 9. Программа, рисующая различные изображения

1. Запустите IDE Delphi.

2. Сохраните проект с именем Draw в отдельной папке.

3. В заголовке окна приложения напишите *Draw*.

4. Со страницы Win32 палитры компонентов поместите на форму компонент **ToolBar** (**ToolBar1**) – инструментальную панель. Задайте в свойстве **Indent** этого компонента отступ от левого края инструментальной панели до элементов управления, которые могут быть расположены на ней, равным *10* пикселей.

5. Создадим на инструментальной панели **ToolBar1** кнопку. Для этого щелкните на этом компоненте правой кнопкой мыши и выберите из всплывающего меню команду NewButton, создав на панели кнопку **ToolButton1**. При нажатии пользователем на эту кнопку будет выполняться рисование изображений.

6. Зададим пиктограмму, которая будет на этой кнопке. Для этого со страницы Win32 палитры компонентов поместите на форму компонент **ImageList** (**ImageList1**) – список изображений. Этот компонент не визуальный. Щелкните два раза по этому компоненту, вызвав редактор списка изображений. Добавьте в список пиктограмму, нажав на кнопку Add и

выбрав в открывшемся диалоговом окне файл pencil.bmp из папки \Program Files \Common Files \Borland Shared \Images \Buttons\ на диске, на котором установлена Delphi. Этот файл содержит два изображения, поэтому при его добавлении в список изображений будет выдан запрос «Bitmap dimensions for PENCIL.BMP are greater then imagelist dimensions. Separate into 2 separate bitmaps?» (Размерность изображения PENCIL.BMP больше размерности списка. Разделить на две отдельных битовых матрицы?). Нажмите на кнопку Yes, чтобы битовая матрица автоматически разделилась на два изображения, при этом первое (цветное) получит индекс *0*, а второе (серое) получит индекс *1*. Закройте редактор списка изображений, нажав на кнопку OK.

Выделите компонент **ToolBar1**. В его свойстве **Images** выберите список изображений **ImageList1**.

Выделите кнопку **ToolButton1** и запишите в ее свойство **ImageIndex** значение *0* – индекс цветной пиктограммы.

7. Со страницы Win32 палитры компонентов поместите на форму компонент **StatusBar**  **StatusBar1** – строку состояния. В Object Inspector нажмите на кнопку с многоточием  рядом с именем свойства **Panels** этого компонента. В открывшемся окне редактора панелей нажмите быструю кнопку Add New (Ins)  , расположенную слева, чтобы добавить новую панель **StatusBar1.Panels[0]** в строку состояния. Каждая панель имеет свойство **Text**, в которое заносится отображаемый в панели строки состояния текст. В значение этого свойства для созданной панели запишите *Программа для рисования домика и ракеты*.

8. Поместите на форму компонент **PageControl**  (**PageControl1**) – многостраничное окно, – расположенный на странице Win32. Этот компонент позволяет создавать страницы с вкладками. У его свойства **Align** вы-

берите значение *alClient*, чтобы развернуть компонент на всю клиентскую область формы.

Для создания страницы щелкните по компоненту правой кнопкой мыши и во всплывающем меню выберите команду New Page. Созданная вами страница будет иметь имя **TabSheet1**. Аналогично создайте вторую страницу **TabSheet2**. Каждая страница является контейнером для размещения любых элементов управления. Выделите первую странницу **TabSheet1**, щелкнув мышью по ее вкладке, затем в середине страницы. В ее свойстве **Caption** напишите ее название *Домик*. Выделите вторую страницу **TabSheet2** и задайте ей заголовок *Ракета*.

9. Выделите страницу **TabSheet1.** На нее со страницы System палитры компонентов поместите компонент **PaintBox**  (**PaintBox1**) – окно для рисования, используемое для создания на форме некоторой области, в которой можно рисовать. В свойстве **Align** выберите значение *alClient*, чтобы развернуть компонент на всю клиентскую область страницы.

Компонент **PaintBox** имеет свойство **Canvas** – поверхность (холст, канва) для рисования, представляющее собой область компонента, на которой можно рисовать или отображать готовые изображения (это свойство также имеют компоненты **Form** и **Image**). Свойство **Canvas** не отражено в Object Inspector. Рисование производится с помощью методов и свойств, которые имеются у этого свойства. Каждая точка холста, канвы имеет координаты X и Y, измеряемые в пикселях. Начало системы координат находится в левом верхнем углу холста. Координата X возрастает при перемещении слева направо, а координата Y – при перемещении сверху вниз (см. рис. 10).

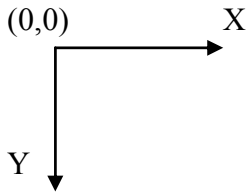

Рис. 10. Система координат холста

10. Выделите вторую страницу **TabSheet2.** На нее со страницы System палитры компонентов поместите компонент **PaintBox**  (**PaintBox2**)

– еще одно окно для рисования. В свойстве **Align** этого компонента выберите значение *alClient*. Теперь снова выделите первую страницу **TabSheet1.**

На этом создание графического интерфейса закончено. Перейдем к написанию обработчиков событий.

11. Создайте событие **OnClick** у кнопки **ToolButton1**.

В обработчике этого события мы будем писать методы холста **Canvas**, с помощью которых будет выполняться рисование изображений. Так как холст **Canvas** является свойством компонента **PaintBox**, то при обращении к методам и свойствам холста мы должны каждый раз указывать ссылки на эти объекты, например, `PaintBox1.Canvas.…` далее записывая имя конкретного метода или свойства. Для сокращения записи при обращении к свойствам и методам объектов, мы будем использовать оператор **With**, который может записываться следующим образом:

With Объект **do begin** Метод1; Метод2; … МетодN; **end**;

где Метод1, Метод2, … МетодN – методы (или свойства) объекта Объект, которые записываются без ссылки на него.

Код обработчика события **OnClick** кнопки **ToolButton1**, который рисует на холсте домик, должен иметь вид:

```
{Эту строку писать не надо, она автоматически добавлена при создании события}
procedure TForm1.ToolButton1Click(Sender: TObject);
Var MaxX, MaxY, Xn, Yn : integer;
begin //Эту строку писать не надо, она добавлена Delphi автоматически
MaxX:=PaintBox1.Width; //Максимальная горизонтальная координата
MaxY:=PaintBox1.Height; //Максимальная вертикальная координата
{Координаты центра экрана. Так как координаты должны быть целыми числами, функция Round округляет результат их вычисления до целого числа}
Xn:=Round(MaxX/2);
Yn:=Round(MaxY/2);
```

```
With PaintBox1.Canvas do
  begin
    Pen.Width:=1; // Установка толщины линии (1 пиксель)
    {Рисование стены домика}
    Pen.Color:=clRed; // Установка цвета линии (красный)
    Brush.Color:=clRed; {Установка цвета кисти - цвета заполнения (красный)}
    Rectangle(Xn-100,Yn+150,Xn+100,Yn-50); {Рисование прямоугольника. Первыми указываются координаты верхней левой точки, затем координаты нижней правой точки прямоугольника}
    {Рисование проема окна домика}
    Pen.Color:=clAqua; // Установка цвета линии (голубой)
    Brush.Color:= clAqua; // Установка цвета кисти (голубой)
    FillRect(Rect(Xn-25,Yn+10,Xn+25,Yn+60)); {Закрашивание прямоугольной области. Первыми указываются координаты верхней левой точки, затем координаты нижней правой точки прямоугольника}
    {Рисование рамы окна}
    Pen.Color:=clBlack; // Установка цвета линии
    MoveTo(Xn,Yn+60); {Смещает указатель к точке с указанными координатами}
    LineTo(Xn,Yn+10); {Рисует линию от текущего указателя к точке с указанными координатами}
    MoveTo(Xn-25,Yn+35); {Смещает указатель к точке с указанными координатами}
    LineTo(Xn+25,Yn+35); {Рисует линию от текущего указателя к точке с указанными координатами}
    {Рисование крыши домика}
    Pen.Color:=clLime; // Установка цвета линии (лимонно-зеленый)
    Brush.Color:=clLime; // Установка цвета кисти
```

```
        Polygon([Point(Xn-100,Yn-50),     Point(Xn,Yn-150),
Point(Xn+100,Yn-50)]); // Рисование многоугольника (треугольника)
        {Рисование круглого окошка на крыше домика}
        Pen.Color:=clBlue; // Установка цвета линии (синий)
        Brush.Color:=clBlue; // Установка цвета кисти (синий)
        Ellipse(Xn-10,Yn-80-10,Xn+10,Yn-80+10);     {Рисование
эллипса (круга). Первыми указываются координаты верхней левой точки, затем
координаты нижней правой точки прямоугольника, в который будет вписан эл-
липс}
        {Рисование трубы на крыше}
        Pen.Color:=clBlack; // Установка цвета линии (черный)
        {Рисование ломаной линии. Так как координаты конца линии совпа-
дают с координатами начала, рисуется замкнутая линия}
        PolyLine([Point(Xn+50,Yn-100),Point(Xn+50,Yn-150),
Point(Xn+70,Yn-150),Point(Xn+70,Yn-80),Point(Xn+50,Yn-100)]);
        Brush.Color:=clBlack; // Установка цвета кисти (черный)
        FloodFill(Xn+60,Yn-100,clBlack,fsBorder);  {Закраши-
вание текущим цветом произвольной замкнутой области. Вначале указываются
координаты произвольной внутренней точки области. Граница области опреде-
ляется сочетанием цвета clBlack и параметра fsBorder, определяющего, что за-
крашивается область, имеющая цвет, не равный clBlack, а на цвете clBlack за-
крашивание останавливается}
        {Вывод надписи «Домик» внизу рисунка}
        Pen.Color:=clBlack; // Установка цвета линии (черный)
        Brush.Color:=clBtnFace; {Установка цвета кисти (систем-
ный цвет, используемый по умолчанию для формы)}
        TextOut(Xn-25,Yn+155,'Домик'); {Вывод текста в точку с
указанными координатами}
        end;
    end; // Эту строку писать не надо, она добавлена Delphi автоматически
```

12. В окне для рисования **PaintBox2**, которое расположено на второй странице, нарисуйте ракету. Для этого в тот же, что и в предыдущем пункте обработчик события **OnClick** кнопки **ToolButton1**, самостоятельно разработайте и добавьте соответствующий код.

13. На этом создание программы, рисующей различные изображения, завершено (рис. 9). Запустите ваше приложение. Убедитесь в его работоспособности. Щелкните по вкладке с именем *Домик*. Нажмите на кнопку инструментальной панели, чтобы нарисовать домик. Щелкните по вкладке с именем *Ракета* и нарисуйте ракету.

Примечание: разверните окно вашего приложения с нарисованным изображением на весь экран, при этом рисунок исчезнет. То же самое произойдет, если перекрыть окно вашего приложения другим приложением. При частичном перекрытии исчезает только часть изображения, которая была закрыта другим окном. Это связано с тем, что операционная система Windows сама не перерисовывает содержимое окна приложения, а сообщает приложению о произошедших изменениях, и оно должно самостоятельно заботиться об этом. В некоторых компонентах, таких как **Image**, уже предусмотрены все необходимые действия, автоматически выполняющие перерисовку испорченного изображения. При рисовании на холсте других оконных компонентов, например, используемого нами **PaintBox**, эти меры должен предпринимать сам разработчик приложения. Для этого у компонента **PaintBox** имеется событие **OnPaint**, которое происходит, когда требуется обновление окна при перерисовке компонента. В обработчике этого события нужно перерисовать изображение. В нашем приложении это можно сделать, повторно написав код, рисующий изображение, или использовать код, уже написанный в обработчике события **OnClick** у кнопки **ToolButton1**. В последнем случае нужно выделить один из компонентов **PaintBox**, например **PaintBox1,** и в Object Inspector, в поле рядом с именем события **OnPaint**, выбрать значение *ToolButton1Click*. – имя раннее созданной процедуры-обработчика события нажатия на кнопку. Теперь при изме-

нении размеров окна приложения или его перекрытии другим окном изображение будет перерисовываться. Кроме того, после запуска приложения изображение уже будет нарисовано, так как событие **OnPaint** происходит и при создании формы, поэтому отпадает необходимость нажимать быструю кнопку на инструментальной панели для рисования.

Задания для самостоятельной работы

Задача 7. В среде Delphi разработать приложение, рисующее новогоднюю елку.

Лабораторная работа №8

Разработка программы Секундомер

Задание. В среде Delphi разработать приложение «Секундомер», позволяющее пользователю включать, выключать отсчет времени и сбрасывать показания.

Рис. 11. Секундомер

1. Запустите IDE Delphi.

2. Сохраните проект с именем Timer в отдельной папке.

3. В заголовке окна приложения напишите *Секундомер*. Измените размеры формы, установив следующие значения ее свойствам: **Height**:=*90*, **Width**:=*260*.

4. Со страницы Additional палитры компонентов поместите на форму компонент **BitBtn** (**BitBtn1**) – кнопка с графикой. В свойстве **Kind**, которое определяет тип кнопки, выберите значение *bkIgnore*, при этом на кнопке появится пиктограмма в виде человечка и надпись «Ignore». Измените эту надпись, написав в свойстве **Caption** новое значение *Старт*.

5. Со страницы Additional палитры компонентов поместите на форму вторую кнопку с графикой **BitBtn2**. В свойстве **Kind** выберите значение *bkNo*, а свойстве **Caption** напишите *Стоп*.

6. Поместите на форму третью кнопку с графикой **BitBtn3**. В свойстве **Kind** выберите значение *bkAbort*, а в свойстве **Caption** напишите *Сброс*.

7. Выровняем положение кнопок на форме. Вначале выделим их в группу. Для этого, удерживая клавишу Shift, щелкните мышкой по каждой кнопке, начиная с первой по порядку. Затем в главном меню Delphi выпол-

ните команду Edit > Align… – выравнивание размещения. В открывшемся окне Alignment, в его левой части Horizontal (выравнивание компонентов по горизонтали), выберите вариант Space equally – разместить с равными интервалами между компонентами. А в правой части окна – Vertical (выравнивание компонентов по вертикали) выберите вариант Tops – выровнять компоненты по их верхним сторонам. Нажмите кнопку OK, чтобы выполнить выравнивание компонентов. После этого, не снимая выделения с группы и удерживая клавишу Ctrl, с помощью клавиш со стрелочками переместите всю группу кнопок в нижнюю часть формы, как показано на рис. 11.

8. Со страницы Additional палитры компонентов поместите на форму в ее верхнюю часть компонент **StaticText**  (**StaticText1**) – метка с бордюром. В значение его свойства **Caption** запишите *0*. Измените размер шрифта надписи. Для этого в свойстве **Font** в подсвойстве **Size** напишите значение *12*.

Примечание: Компонент **StaticText** аналогичен компоненту **Label**, но обладает дополнительными возможностями, например, такими, как автоматический перенос слов длинной надписи на новую строку (при значении свойства **AutoSize**:=*false*), задание стиля бордюра (рамки текста).

Задайте стиль бордюра, выбрав у свойства **BorderStyle** значение *sbsSunken*.

9. Со страницы System палитры компонентов поместите на форму компонент **Timer**  (**Timer1**) –таймер. Этот компонент невизуальный, поэтому может быть размещен в любом месте формы. Если таймер включен (свойство **Enabled**:= true), то периодически, через интервал времени, заданный в миллисекундах в свойстве **Interval** (по умолчанию значение равно 1000 mc=1 c), таймер срабатывает, вызывая событие **OnTimer**.

Чтобы после запуска программы до нажатия соответствующей кнопки таймер был выключен, установите у его свойства **Enabled** значение *false*.

На этом создание графического интерфейса закончено. Перейдем к написанию обработчиков событий.

10. У кнопки **BitBtn1** («Старт») создаете событие **OnClick**. В обработчик этого события запишите:

```
Timer1.Enabled:=true;
```

При нажатии на эту кнопку таймер включается.

11. У кнопки **BitBtn2** («Стоп») создаете событие **OnClick**. В обработчик этого события запишите:

```
Timer1.Enabled:=false;
```

При нажатии на эту кнопку таймер выключается.

12. У компонента **Timer1** создайте событие **OnTimer**. В обработчик этого события запишите:

```
Timer1.Tag:=Timer1.Tag+1;
StaticText1.Caption:=IntToStr(Timer1.Tag);
```

Свойство **Tag** – это свойство, имеющееся у многих компонентов, которое разработчик приложения может использовать по своему усмотрению. В его значении может храниться любое целое число.

В данном приложении в значении этого свойства мы будем сохранять количество срабатываний таймера, что записано в первой строке этого обработчика события. Так, интервал срабатывания таймера, определяемый в свойстве **Interval**, равен одной секунде, и количество его срабатываний равно количеству секунд, прошедших с момента его включения.

Во второй строке приведенного кода текущее значение свойства **Tag** – количество секунд, предварительно преобразованное из целого числа в строку с помощью функции **IntToStr** – выводится в метку **StaticText1**.

13. У кнопки **BitBtn3** («Сброс») создаете событие **OnClick**. В обработчик этого события запишите:

```
Timer1.Enabled:=false;
Timer1.Tag:=0;
StaticText1.Caption:='0';
```

При нажатии на эту кнопку таймер выключается, значение свойства **Tag** обнуляется и на экран выводится надпись «0».

14. На этом создание программы Секундомер завершено. Запустите ваше приложение. Убедитесь в его работоспособности. Включите секундомер, нажав на кнопку «Старт». Остановите секундомер, нажав на кнопку «Стоп». Продолжите отсчет времени, снова нажав на кнопку «Старт». Обратите внимание, что отображение времени ведется только в секундах, т.е. после 59 секунды на экран выводится значение 60, 61 и т.д., не выделяя минуты. Остановите секундомер и сбросьте показания секундомера, нажав на кнопку «Сброс».

Задания для самостоятельной работы

Задача 8. В среде Delphi разработать приложение Секундомер, отображающее время отсчета в минутах и секундах и позволяющее пользователю включать, выключать отсчет времени, сбрасывать показания.

Лабораторная работа №9

Разработка программы Универсальный проигрыватель

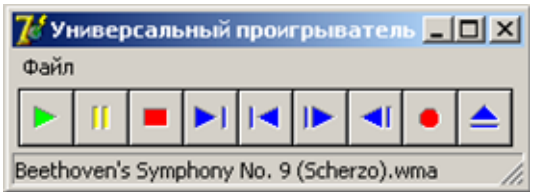

Рис. 12. Универсальный проигрыватель

Задание. В среде Delphi разработать приложение Универсальный проигрыватель, позволяющее воспроизводить различные форматы звуковых и видеофайлов. Выбор пользователем мультимедиа-файлов должен осуществляться с помощью диалогового окна, вызываемого командой меню Открыть.

1. Запустите IDE Delphi.

2. Сохраните проект с именем Mediaplayer в отдельной папке.

3. В заголовке окна приложения напишите *Универсальный проигрыватель*. Измените размеры формы, установив следующие значения ее свойствам: **Height**:=*105*, **Width**:=*270*.

4. Со страницы Standard палитры компонентов поместите на форму компонент **MainMenu**  (**MainMenu1**) – главное меню. Создайте раздел главного меню Файл, содержащий две команды: Открыть… и Выход. Между этими разделами вставьте разделитель.

5. Со страницы Win32 палитры компонентов поместите на форму компонент **StatusBar**  **StatusBar1** – строку состояния. С помощью редактора панелей добавьте новую панель **StatusBar1.Panels[0]** в строку состояния (см. п. 7 лабораторной работы № 7).

6. Поместите на форму диалог открытия файлов **OpenDialog**  (**OpenDialog1**) со страницы Dialogs палитры компонентов. В свойстве **Filter** этого диалога задайте несколько фильтров: Файлы мультимедиа (*.wav;

*.mp3; *.wma; *.midi; *.mid; *.avi; *.mpeg; *.mpg), Волновые файлы (*.wav), MPEG аудио файлы (*.mp3), Файлы Windows Media Audio (*.wma), MIDI файлы (*.midi; *.mid), Видео файлы (*.avi), MPEG видео файлы (*.mpeg; *.mpg).

7. Со страницы System палитры компонентов поместите на форму компонент **MediaPlayer**  (**MediaPlayer1**) – аудио- и видео-плеер. Этот компонент является панелью управления воспроизведением звуковых и видеофайлов, а само воспроизведение осуществляется с помощью встроенной в операционную систему Windows технологии Media Control Interface (MCI) – интерфейса управления носителями, используя установленные в системе кодеки. Панель компонента **MediaPlayer**, помещенного на форму, похожа на стандартную панель управления проигрывателями и содержит следующий набор кнопок (слева направо): Play  – воспроизведение, Pause  – пауза воспроизведения или записи, Stop  – остановка воспроизведения или записи, Next  – переход на следующий трек или на конец, Prev  – переход на предыдущий трек или на начало, Step  – перемещение вперед на заданное число кадров, Back  – перемещение назад на заданное число кадров, Record  – начало записи, Eject  – освобождение объекта, загруженного в устройство.

На этом создание графического интерфейса закончено (см. рис. 12). Перейдем к написанию обработчиков событий.

8. Создайте событие **OnClick** для команды Открыть… раздела меню Файл. В обработчик этого события напишите следующий код:

```
If OpenDialog1.Execute Then // Вызывает диалоговое окно
begin
```

{Закрывает мультимедиа-устройства, которые на данный момент, возможно, открыты}

```
MediaPlayer1.Close;
```

{Устанавливает имя выбранного файла}

```
MediaPlayer1.FileName:=OpenDialog1.FileName;
```

{Выводит имя выбранного файла на панель строки состояния}

```
StatusBar1.Panels[0].Text:=
ExtractFileName(OpenDialog1.FileName);
```

{Открывает файл для воспроизведения с помощью метода Open}

```
MediaPlayer1.Open;
```

end;

9. Для команды Выход раздела меню Файл в обработчик события **OnClick** напишите следующий код:

{Закрывает мультимедиа-устройства, которые на данный момент, возможно, открыты}

```
MediaPlayer1.Close;
Close; // Закрывает приложение
```

10. На этом создание программы Универсальный проигрыватель завершено. Запустите ваше приложение. Убедитесь в его работоспособности. Откройте звуковой файл, например, Beethoven's Symphony No. 9 (Scherzo).wma, поставляемый вместе с Windows и расположенный в папке Мои документы \Моя музыка \Образцы музыки. Нажмите на кнопку Play  , чтобы начать воспроизведение. Приостановите воспроизведение, нажав на кнопку Pause  . Еще раз нажмите на кнопку Pause  , чтобы возобновить воспроизведение. Остановите воспроизведение, нажав на кнопку Stop  . Обратите внимание на строку состояния вашего приложения, в которой должно быть написано имя выбранного файла.

Откройте видео-файл, например, speedis.avi из папки \Program Files \Borland \Delphi7 \Demos \CoolStuf\ с диска, на котором установлена Delphi. Выполните те же действия, что и со звуковым файлом. Воспроизведение видео-файлов по умолчанию осуществляется в отдельном окне, автоматически создаваемом компонентом **MediaPlayer**.

Задания для самостоятельной работы

Задача 9. В среде Delphi разработать приложение Универсальный проигрыватель, позволяющий воспроизводить различные форматы звуковых и видео-файлов. Выбор пользователем мультимедиа-файлов должен осуществляться с помощью диалогового окна, вызываемого командой меню Открыть. Предоставить пользователю возможность выбирать место показа видео-файла: в отдельном окне или непосредственно в окне приложения, в специально отведенной для этого области.

Лабораторная работа №10

Разработка простейшего графического редактора

Задание. В среде Delphi разработать простейший графический редактор, позволяющий сохранить созданный рисунок, открыть ранее созданный рисунок, очистить рабочее поле (холст) и имеющий следующие инструменты: палитру цветов, окна основного и вспомогательного цветов, заливку, карандаш, ластик, полый прямоугольник, закрашенный прямоугольник (см. рис. 13).

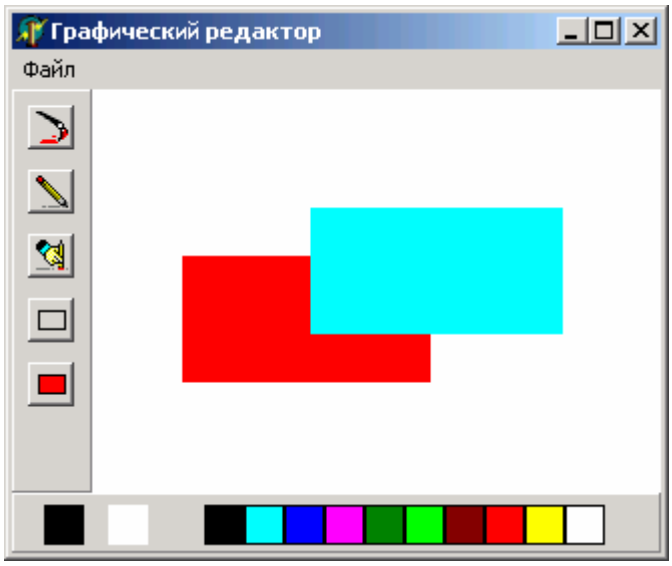

Рис. 13. Простейший графический редактор

1. Запустите IDE Delphi.

2. Сохраните проект с именем GraphEdit в отдельной папке.

3. В заголовке окна приложения напишите *Графический редактор*.

4. Поместите на форму компонент **MainMenu**  (**MainMenu1**) – главное меню. Создайте раздел меню Файл, а в нем четыре команды: Создать, Открыть..., Сохранить как… и Выход.

5. Поместите на форму три компонента **Panel**  (**Panel1**, **Panel2** и **Panel3,** соответственно). Задайте в Object Inspector значения следующим их свойствам:

у **Panel1**: **Align**:=*alBottom*, **Height**:=*30*, **Caption:**=’’ (все стереть);

у **Panel2**: **Align**:=*alLeft*, **Width:**=*40*, **Caption**:=’’;

у **Panel3**: **Align**:=*alClient*, **BevelOuter**:=*bvNone*, **Caption**:=’’.

Панели были помещены для удобства компоновки компонентов на форме и их выравнивания при изменении пользователем размеров окна приложения. На этих панелях мы будем размещать все другие компоненты.

6. Поместите на форму компонент **OpenPictureDialog**  (**OpenPictureDialog1**).

Поместите на форму компонент **SavePictureDialog**  (**SavePictureDialog1**). Запишите в его свойство **DefaultExt** расширение, которое будет у сохраняемых файлов по умолчанию: *bmp*.

7. Поместите на **Panel1** два компонента **Image**  (**Image1** и **Image2**, соответственно) и расположите их в левой части панели. Задайте в Object Inspector их размеры **Height**:=*20* и **Width**:=*20*. Это будут окна основного и вспомогательных цветов.

8. Поместите на **Panel1** еще один компонент **Image** (**Image3**) и расположите его правее первых двух на одном с ними уровне. Его высоту задайте такой же, что и у первых двух компонентов **Image** (**Height**:=*20*), а длину – в 10 раз большую (**Width**:=*200*). Это будет палитра цветов.

9. Поместите на **Panel3** еще один компонент **Image** (**Image4**). Задайте его свойство **Align**:=*alClient*. Это будет рабочее поле или холст для рисунков.

10. Поместите на **Panel2** пять быстрых кнопок **SpeedButton**  (**SpeedButton1**, **SpeedButton2**, **SpeedButton3**, **SpeedButton4**, **SpeedButton5**) и расположите их сверху вниз. У всех кнопок установите свойства **GroupIndex** равным *1*, а свойства **AllowAllUp** – *true*. Эти свойства обеспечат кнопкам возможность фиксироваться в нажатом и не нажатом состояниях, причем одновременно только одна из этих кнопок может находиться в нажатом состоянии.

У **SpeedButton1** загрузите в свойство **Glyph** файла пиктограммы brush.bmp из папки \Program Files \Common Files \Borland Shared \Images \Buttons\ с диска, на котором установлена Delphi. Эта кнопка будет соответствовать «Заливке» – типичному инструменту графических редакторов.

У **SpeedButton2** загрузите в свойство **Glyph** файл пиктограммы pencil.bmp из папки \Program Files \Common Files \Borland Shared \Images \Buttons\ с диска, на котором установлена Delphi. Эта кнопка будет соответствовать инструменту «Карандаш».

У **SpeedButton3** загрузите в свойство **Glyph** файл пиктограммы erase.bmp из папки \Program Files \Common Files \Borland Shared \Images \Buttons\ с диска, на котором установлена Delphi. Эта кнопка будет соответствовать инструменту «Ластик».

У **SpeedButton4** загрузите в свойство **Glyph** пиктограмму  , предварительно создав ее. Для этого запустите встроенный редактор изображений, выполнив команду Image Editor в разделе Tolls главного меню Delphi. Затем создайте графический файл размером 16 на 16 точек, выполнив в главном меню редактора действия File > New > Bitmap File (.bmp). Нарисуйте пиктограмму и сохраните ее под именем *frame.bmp* (для удобства рисования можно увеличить масштаб изображения командой View > Zoom In). Кнопка **SpeedButton4** будет предназначаться для рисования прямоугольной рамки (полого прямоугольника).

У **SpeedButton5** загрузите в свойство **Glyph** пиктограмму  , также предварительной создав ее под именем *Rect.bmp*. Эта кнопка будет предназначаться для рисования закрашенного прямоугольника.

Создайте всплывающие подсказки о назначении каждой кнопки. Для этого установите у всех кнопок свойство **ShowHint**, равное *true*, а у каждой

кнопки в свойстве **Hint** напишите текст подсказки: *Заливка*, *Карандаш*, *Ластик*, *Рамка*, *Прямоугольник*.

На этом создание графического интерфейса завершено. Теперь перейдем к написанию обработчиков событий.

11. Создайте у формы событие **OnCreate**. Это событие происходит при создании формы, в момент запуска пользователем приложения.

Для этого события напишите следующий обработчик:

```
procedure TForm1.FormCreate(Sender: TObject); {Эту строку писать не надо, она автоматически добавлена при создании события}
Var i:integer; // Переменная-счетчик цикла
begin // эту строку писать не надо, она добавлена Delphi автоматически
{Задание свойств кисти основного и вспомогательного цветов: черный и белый}
Image1.Canvas.Brush.Color:=clBlack;
Image2.Canvas.Brush.Color:=clWhite;
{Заполнение окон основного и вспомогательного цветов соответствующим цветом}
Image1.Canvas.FillRect(Rect(0,0,20,20));
Image2.Canvas.FillRect(Rect(0,0,20,20));
{Закраска элементов палитры цветов: для каждого элемента палитры задается свой цвет и элемент заполняется этим цветом с помощью процедуры Rectangle}
With Image3.Canvas do
  For i:=1 to 10 do
    begin
      Case i of
        1:Brush.Color:=clBlack; // черный
        2:Brush.Color:=clAqua; // голубой
        3:Brush.Color:=clBlue; // синий
        4:Brush.Color:=clFuchsia; // сиреневый
        5:Brush.Color:=clGreen; // зеленый
        6:Brush.Color:=clLime; // лимонно-зеленый
```

```
            7:Brush.Color:=clMaroon; // темно-бордовый
            8:Brush.Color:=clRed; // красный
            9:Brush.Color:=clYellow; // желтый
            10:Brush.Color:=clWhite; // белый
          end;
          Rectangle((i-1)*20,0,i*20,20);
        end;
    {Закрашивание холста белым цветом}
    Image4.Canvas.Brush.Color:=clWhite;
    Image4.Canvas.FillRect
    (Rect(0,0,Image4.Width,Image4.Height));
    end; // Эту строку писать не надо, она добавлена Delphi автоматически
```

12 Создайте событие **OnClick** для команды Создать раздела меню Файл. В обработчик этого события напишите следующий код:

```
{Удаление предыдущего изображения с холста}
Image4.Picture.Assign(nil);
{Закрашивание холста белым цветом}
Image4.Canvas.Brush.Color:=clWhite;
Image4.Canvas.FillRect
(Rect(0,0,Image4.Width,Image4.Height));
```

13. Для команды меню Открыть… в обработчик события напишите следующий код:

```
If OpenPictureDialog1.Execute Then // Вызов диалогового окна
{Загрузка в компонент Image4 (холст) файла рисунка, выбранного пользователем в диалоговом окне}
Image4.Picture.Bitmap.
LoadFromFile(OpenPictureDialog1.FileName);
```

14. Для команды меню Сохранить как… в обработчик события напишите следующий код:

```
If SavePictureDialog1.Execute Then // Вызов диалогового окна
{Сохранение в файл изображения из компонента Image4 (холст)}
Image4.Picture.Bitmap.
SaveToFile(SavePictureDialog1.FileName);
```

15. Для команды меню Выход в обработчик напишите следующий код:

```
Close;
```

16. У компонента **Image3** создайте событие **OnMouseDown**. В обработчик этого события вставьте код:

```
If (Button=mbLeft) Then // Нажата левая кнопка мыши*
  begin
    {Установка основного цвета}
    Image1.Canvas.Brush.Color:=
    Image3.Canvas.Pixels[X,Y];
    Image1.Canvas.FillRect(Rect(0,0,20,20));
  end
Else // Нажата правая кнопка мыши
  begin
    {Установка вспомогательного цвета}
    Image2.Canvas.Brush.Color:=
    Image3.Canvas.Pixels[X,Y];
    Image2.Canvas.FillRect(Rect(0,0,20,20));
  end;
```

Событие **OnMouseDown** происходит при нажатии клавиши мыши над компонентом. Поэтому, когда над элементом палитры цветов (**Image3**)

∗ Button – это параметр, который передается в обработчик события и определяет, какая кнопка мыши в данный момент нажата. X, Y – координаты курсора мыши, которые также передаются в обработчик (см. заголовок обработчика события **OnMouseDown**: `procedure TForm1.Image3MouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer))`.

нажата левая кнопка мыши, с помощью свойства **Pixels** определяется цвет этого элемента, и этот цвет присваивается свойству кисти компонента **Image1** (окно основного цвета). Затем это окно закрашивается с помощью процедуры **FillRect**. Когда нажата правая кнопка мыши – закрашивается окно вспомогательного цвета (**Image2**).

Теперь перейдем к написанию кода, отвечающего за рисование изображения на холсте (компонент **Image4**).

17. У компонента **Image4** создайте событие **OnMouseDown**. Здесь мы опишем, что будет происходить при нажатии кнопки мыши над холстом, в зависимости от того, какая из быстрых кнопок («инструментов») на **Panel2** нажата. В обработчике созданного события запишите:

```
If SpeedButton1.Down Then // Если нажата быстрая кнопка Заливка
  begin
    {Установка цвета закрашивания}
    If Button=mbLeft Then // Нажата левая кнопка мыши
{Основным цветом}
      Image4.Canvas.Brush.Color:=Image1.Canvas.Brush.Color
    Else
{Вспомогательным цветом}
      Image4.Canvas.Brush.Color:=Image2.Canvas.Brush.Color;
{Закрашивание области, которая имеет цвет точки, в которой находится курсор мыши, а на других цветах заполнение останавливается}
    Image4.Canvas.
    FloodFill(X,Y,Image4.Canvas.Pixels[X,Y],fsSurface);
  end;

If SpeedButton2.Down Then // если нажата кнопка Карандаш
{Смещает позицию пера к точке, в которой находится курсор мыши и с которой начнется рисование «карандашом» в следующем обработчике}
Image4.Canvas.MoveTo(X,Y);
```

```
If SpeedButton3.Down Then // если нажата быстрая кнопка Ластик
 begin
```

{Задание свойству кисти холста (Image4) цвета фона (вспомогательного), которым будет стираться изображение}

```
    Image4.Canvas.Brush.Color:=Image2.Canvas.Brush.Color;
```

{Создание изображения «ластика» - небольшой квадратной рамки (12 на 12 пикселей), нарисованной точками}

```
    Image4.Canvas.DrawFocusRect(Rect(X-6,Y-6,X+6,Y+6));
```

{Закрашивание области внутри рамки цветом фона – «стирание»}

```
    Image4.Canvas.FillRect(Rect(X-5,Y-5,X+5,Y+5));
```

{Запоминание текущих координат курсора мыши, используя которые была изображена рамка «ластика»}

```
    Xb:=X; Yb:=Y;
  end;
```

```
If SpeedButton4.Down or SpeedButton5.Down Then
```

{Если нажаты быстрые кнопки рисования полого или закрашенного прямоугольника}

```
   begin
```

{Установка вспомогательного цвета свойству кисти холста (Image4). Этим цветом будет отображаться точечная рамка – «проект» прямоугольника (см. следующий обработчик)}

```
    Image4.Canvas.Brush.Color:=Image2.Canvas.Brush.Color;
```

{Запоминание текущих координат курсора мыши, которые будут использоваться для изображения прямоугольника (см. следующий обработчик)}

```
    Xb:=X; Yb:=Y;
    Xe:=X; Ye:=Y;
  end;
```

18. В предыдущем обработчике мы использовали переменные `Xb`, `Yb`, `Xe`, `Ye`, которые нигде еще не описаны. Опишите их глобальными переменными целого типа. Для этого в тексте модуля, ниже ключевого

слова **implementation**, запишите:

```
var
  Xb,Yb,Xe,Ye : integer;
```

19. У компонента **Image4** создайте событие **OnMouseMove**. Это событие многократно происходит при перемещении курсора мыши над компонентом – в данном случае над холстом. В обработчике созданного события запишите:

{Выход из обработчика, если не нажата левая кнопка мыши, так как в противном случае будут выполняться нежелательные действия, например, рисоваться линия, когда кнопка мыши еще не нажата, и т.п.}*

```
If not (ssLeft in Shift) Then exit;


{Режим карандаша}
If SpeedButton2.Down Then
  begin
{Установка цвета рисуемой линии (основной цвет)}
    Image4.Canvas.Pen.Color:=Image1.Canvas.Brush.Color;
{Рисование линии от предыдущей позиции пера к текущим координатам курсора мыши}
    Image4.Canvas.LineTo(X,Y);
  end;


{Режим ластика}
If SpeedButton3.Down Then
  begin
```

---

* Shift – это параметр, который передается в обработчик события **OnMouseMove** и определяет, какие вспомогательные клавиши на клавиатуре были нажаты (включая и кнопки мыши) (см. заголовок обработчика: `procedure TForm1.Image4MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);`). ssLeft – это значение параметра Shift, означающее, что была нажата левая кнопка.

{Удаление предыдущего изображения «ластика». При повторной прорисовке методом DrawFocusRect, используя прежние координаты, хранящиеся в Xb и Yb, предыдущее изображение удаляется}

```
Image4.Canvas.
DrawFocusRect(Rect(Xb-6,Yb-6,Xb+6,Yb+6));
```

{Запоминание новых координат курсора мыши}

```
Xb:=X; Yb:=Y;
```

{Создание нового изображения «ластика» по текущим координатам}

```
Image4.Canvas.
DrawFocusRect(Rect(Xb-6,Yb-6,Xb+6,Yb+6));
```

{Закрашивание области внутри нового «ластика» цветом фона – «стирание»}

```
Image4.Canvas.FillRect(Rect(Xb-5,Yb-5,Xb+5,Yb+5));
end;
```

{Режим прямоугольников}

```
If SpeedButton4.Down or SpeedButton5.Down Then
```

{Здесь будет создаваться рамка, изображенная точками и изменяющаяся в размерах, по мере перемещения мыши с нажатой левой кнопкой – «проект» рисуемого прямоугольника}

```
begin
```

{Удаление предыдущего изображения рамки, как в режиме ластика. Первоначальные координаты Xb, Yb, Xe,Ye были «запомнены» в предыдущем обработчике}

```
Image4.Canvas.DrawFocusRect(Rect(Xb,Yb,Xe,Ye));
```

{Запоминание новых координат нижней правой точки рамки. Координаты верхней левой точки – Xb, Yb – остаются прежними (сохраненными в предыдущем обработчике), так как «проект» прямоугольника растягивается по мере перемещения мыши от точки, в которой пользователь нажал ее левую кнопку}

```
Xe:=X; Ye:=Y;
```

{Создание нового изображения рамки по текущим координатам ее нижней правой точки}

```
        Image4.Canvas.DrawFocusRect(Rect(Xb,Yb,Xe,Ye));
    end;
```

20. У компонента **Image4** создайте событие **OnMouseUp**. Это событие происходит при отпускании ранее нажатой кнопки над компонентом. В обработчике созданного события запишите:

```
{Выход из обработчика, если отпускается не левая кнопка мыши}
If Button<>mbLeft Then Exit;

{Режим ластика}
If SpeedButton3.Down Then
{Удаление изображения «ластика»}
Image4.Canvas.DrawFocusRect(Rect(Xb-6,Yb-6,Xb+6,Yb+6));

{Режим полого прямоугольника}
If SpeedButton4.Down Then
  begin
```

{Установка основного цвета свойству кисти Brush для рисования рамки полого прямоугольника (напомним, что при рисовании методом FrameRect рамка рисуется цветом кисти Brush, а внутренняя область не закрашивается)}

```
        Image4.Canvas.Brush.Color:=Image1.Canvas.Brush.Color;
```

{Рисование полого прямоугольника}

```
        Image4.Canvas.FrameRect(Rect(Xb,Yb,X,Y));
```

{Восстановление цвета у свойства Brush компонента Image4 как вспомогательного}

```
        Image4.Canvas.Brush.Color:=
        Image2.Canvas.Brush.Color;
    end;
```

```
{режим закрашенного прямоугольника}
If SpeedButton5.Down Then
  begin
{Установка основного цвета перу Pen, которым будет рисоваться рамка закрашенного прямоугольника}
    Image4.Canvas.Pen.Color:=Image1.Canvas.Brush.Color;
{Установка вспомогательного цвета кисти Brush, которым будет закрашиваться внутренняя область прямоугольника}
    Image4.Canvas.Brush.Color:=Image2.Canvas.Brush.Color;
{Рисование закрашенного прямоугольника}
    Image4.Canvas.Rectangle(Xb,Yb,X,Y);
  end;
```

21. На этом создание простейшего графического редактора завершено. Запустите ваше приложение. Убедитесь в его работоспособности. Проверьте установку цветов. При щелчке левой или правой кнопкой мыши на палитре цветов у вас будет меняться соответственно основной или вспомогательный цвета. Нарисуйте произвольную кривую инструментом Карандаш. Создайте закрашенный прямоугольник (в процессе рисования, пока вы удерживаете левую кнопку мыши, за ее курсором будет растягиваться точечная рамка – «проект» прямоугольника, в момент отпускания кнопки мыши у вас отобразится прямоугольник, граница которого будет нарисована основным цветом, а внутренняя часть залита вспомогательным). Нарисуйте полый прямоугольник и закрасьте его с помощью инструмента Заливка. Сотрите часть изображения с помощью инструмента Ластик, при этом обратите внимание на изображение ластика в виде точечной рамки около курсора мыши. Сохраните изображение в файл. Очистите холст командой главного меню Создать. Загрузите произвольный рисунок в формате *.bmp, например, technlgy.bmp из папки \Program Files \Common Files \Borland Shared \Images \Splash \16Color\ с диска, на котором установлена Delphi. Отредактируйте это изображение.

Задания для самостоятельной работы

Задача 10. В среде Delphi разработать простейший графический редактор, позволяющий сохранить созданный рисунок, открыть ранее созданный рисунок, очистить рабочее поле (холст) и имеющий следующие инструменты: Палитру цветов, окна Основного и Вспомогательного цветов, Заливку, Карандаш, Ластик, Полый прямоугольник, Закрашенный прямоугольник, Линия (рисование прямых линий), Выбор цвета (установка цвета любой точки рисунка в качестве основного или вспомогательного).

Лабораторная работа №11

Создание простейшей анимации.

Задание. В среде Delphi разработать приложение с простейшей анимацией – условным изображением шагающего человека.

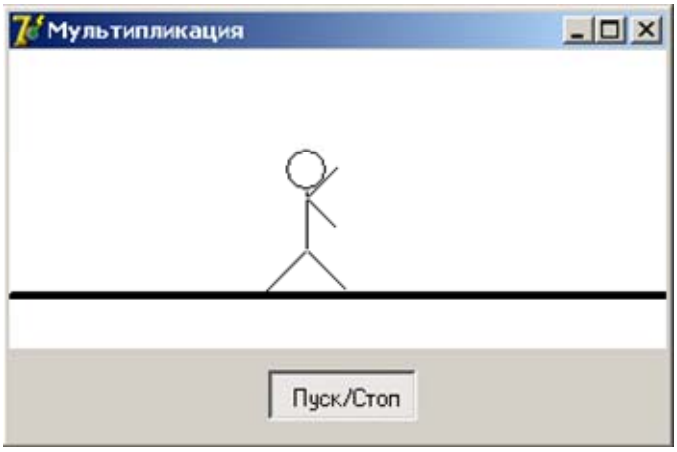

Рис. 14. Программа, использующая простейшую анимацию

1. Запустите IDE Delphi.

2. Сохраните проект с именем Animation в отдельной папке.

3. В заголовке окна приложения напишите *Мультипликация*. Измените размеры формы, установив следующие значения ее свойствам: **Height**:=*230*, **Width**:=*340*.

4. Со страницы Additional палитры компонентов поместите на форму компонент **Image** (**Image1**) – изображение. Выровняйте компонент по верхней части формы, установив его свойству **Align** значение *alTop*. Измените высоту компонента, задав его свойству **Height** значение *150*.

5. С той же страницы палитры компонентов поместите в нижнюю часть формы, как показано на рисунке (рис. 14), компонент **SpeedButton** (**SpeedButton1**) – быструю кнопку. В ее свойстве **Caption** напишите *Пуск/Стоп*. Измените ширину и высоту кнопки, установив следующие значения ее свойствам: **Height**:=*25*, **Width**:=*75*. Для того чтобы кнопка фикси-

ровалась в нажатом состоянии, указывая пользователю запущена ли анимация, задайте следующие значения ее свойствам:

GroupIndex:=*1* – устанавливает индекс группы кнопок (чтобы кнопка фиксировалась, значение этого свойства должно быть больше нуля),

AllowAllUp:=*true* – разрешает отжатое (нормальное) состоянии кнопки.

6. Со страницы System палитры компонентов поместите на форму компонент **Timer**  (**Timer1**) – таймер. Этот компонент невизуальный, поэтому может быть размещен в любом месте формы.

Таймер будет задавать темп смены кадров, равный 1 кадр в 0,5 сек. Для этого установите в его свойстве **Interval** значение, равное *500*.

Чтобы после запуска программы до нажатия кнопки таймер был выключен, установите у его свойства **Enabled** значение *false*.

На этом создание графического интерфейса закончено. Перейдем к написанию обработчиков событий.

7. Опишем константы и переменные, которые будут использоваться в приложении. Для этого в окне Редактора кода, в тексте модуля, ниже ключевого слова **implementation** запишите:

```
const
  Ypos=120;              // Y-координата "земли"
  L=20;                  // Длина ноги
  Hmen=30;               // Высота тела
  H=15;                  // Длина руки
  Rhead=10;              // Радиус головы
Var
  Num:word;              // Номер кадра
  Revers:integer;        // Направление движения
  Xpos:word;             // X-координата туловища
```

8. После этого напишите код процедуры, рисующей положение человека:

```
Procedure DrawMan; // Процедура, рисующая положение человека
Var Yhead:word;    // Y-координата низа головы
begin
 With Form1.Image1.Canvas do
  begin
   Case Num of
    0: begin       // 1-ый кадр
        Yhead:=Ypos-L-Hmen;    // Y-координата низа головы
        MoveTo(Xpos-L,Ypos);
        LineTo(Xpos,Ypos-L);   // Нога
        LineTo(Xpos+L,Ypos);   // Другая нога
        MoveTo(Xpos,Ypos-L);
        LineTo(Xpos,Yhead);    // Туловище
        MoveTo(Xpos+Revers*H,Yhead+4-H);
        LineTo(Xpos,Yhead+4); // Рука
        LineTo(Xpos+Revers*H,Yhead+4+H); // Другая рука
        Ellipse(Xpos-Rhead,Yhead,Xpos+Rhead,Yhead-
2*Rhead); // Голова
       end;
    1: begin       // 2-ой кадр
        Yhead:=Ypos-L-Hmen;    // Y-координата низа головы
        MoveTo(Xpos,Ypos);
        LineTo(Xpos,Yhead);    // Туловище + ноги
        MoveTo(Xpos,Yhead+4);
        LineTo(Xpos+revers*L,Yhead+4);   // Руки
        Ellipse(Xpos-Rhead,Yhead,Xpos+Rhead,Yhead-
2*Rhead); // Голова
       end;
```

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

Эта процедура рисует то или иное положение движущегося человека, определяемое оператором выбора **Case** в зависимости от номера «кадра» Num. В этой анимации для изображения движения человека используется всего два «кадра».

9. В Object Inspector в списке компонентов выберите форму **Form1** и создайте для нее событие **OnCreate**, которое происходит при создании формы. В обработчик этого события запишите:

```
{Задание переменным начальных значений}
Num:=0;          // Номер кадра
Revers:=1;       // Направление движения
Xpos:=2*H;       // Координата туловища

With Image1.Canvas do
  begin
    Pen.Width:=4;  {Установка толщины линии, изображающей "землю" (4 пикселя)}
    MoveTo(0,Ypos+3);
    LineTo(ClientWidth,Ypos+3); // Линия "земли"
    Pen.Width:=1;  {Установка "стандартной" толщины линии (1 пиксель)}
    Pen.Mode:=pmNotXor; {Установка режима пера: при повторной прорисовки с теми же координатами, изображение линии стирается}
  end;
DrawMan; // Рисование исходного положения человека
```

В этой процедуре задаются начальные значения переменным и рисуется линия, отображающая «землю», по которой будет ходить наш человек. Затем устанавливается режим пера *pmNotXor* – сложение с фоном по ин-

версному исключающему ИЛИ, т.е. режим рисования, когда при повторной прорисовке с теми же координатами изображение будет стираться. И в заключение вызывается процедура DrawMan, которая рисует исходное положение человека.

10. Создайте событие **OnClick** для быстрой кнопки **SpeedButton1**. В обработчик этого события запишите:

```
Timer1.Enabled:= not Timer1.Enabled; {Включение / выключение таймера}
```

При нажатии на эту кнопку таймер будет включаться, запуская анимацию, а при повторном нажатии выключаться – останавливая анимацию.

11. У компонента **Timer1** создайте событие **OnTimer**. В обработчик этого события запишите:

```
DrawMan; {Стирание предыдущего положения человек (повторная прорисовка с теми же координатами)}
If (Xpos>=Image1.Width-L)or(Xpos<=L) Then Revers:=-Revers; // Смена направления движения
Xpos:=Xpos+Revers*L; // Изменение текущей x-координаты туловища
Num:=1-Num; // Изменение номера кадра
DrawMan; // Рисование нового положения человека
```

Это событие происходит с периодичностью, заданной свойством **Interval** компонента **Timer1**, т.е. один раз в 0,5 сек, задавая темп смены кадров. При возникновении этого события вначале вызывается процедура DrawMan, прорисовывая изображение человека с прежними координатами, т.е. стирая его, так как установлен режим пера *pmNotXor*. Затем выполняется проверка текущей x-координаты положения человека `Xpos`. Если эта координата отстоит от кого-либо конца холста компонента **Image1** на величину, меньшую, чем значение `L` – длину ноги человечка, то изменяется знак переменной `Revers`, характеризующей направление движения. Если

`Revers=1` – человек шагает вправо, если `Revers=-1` – человек шагает влево. После этого позиция `Xpos` изменяется на величину `Revers*L`, т.е. на шаг вправо или влево. Изменяется переменная `Num`, которая указывает номер изображаемого «кадра»: 0 или 1. В заключение вызывается процедура DrawMan, которая рисует указанный «кадр» в указанной позиции `Xpos`.

12. На этом создание приложения с простейшей анимацией завершено. Запустите ваше приложение. Включите анимацию, нажав на кнопку «Пуск/Стоп». Обратите внимание, что кнопка зафиксировалась в нажатом состоянии. Проследите за сменой направления движение человека, когда он «доходит» до «стенки». Остановите анимацию, отжав кнопку.

Задания для самостоятельной работы

Задача 11. В среде Delphi разработать приложение с простейшей анимацией – условным изображением часов с качающимся маятником и движущейся часовой стрелкой, используя буферизацию изображения.

Приложения

Свойства Формы

1. **Action** – определяет действие, связанное с элементом управления (в данном случае – с формой).

2. **ActiveControl** – определяет, какой из размещенных на форме компонентов будет иметь фокус (являться активным) сразу после запуска приложения.

3. **Align** – определяет способ выравнивания компонента (в данном случае – формы) внутри контейнера (родительского компонента) при изменении размеров последнего. Возможные значения:

alBottom – занимать всю нижнюю часть контейнера;

alClient – занимать всю клиентскую область – свободную площадь контейнера;

alLeft – занимать всю левую часть контейнера;

alNone – нет выравнивания (по умолчанию);

alRight – занимать всю правую часть контейнера;

alTop – занимать всю верхнюю часть контейнера.

4. **AlphaBlend** – определяет, является ли форма полупрозрачной.

5. **AlphaBlendValue** – определяет степень прозрачности формы (от 0 до 255). Это свойство связано с предыдущим – **AlphaBlend**.

6. **Anchors** – определяет привязку компонента (в данном случае – формы) к родительскому при изменении размеров последнего. Содержит следующие подсвойства:

akLeft – привязать к левому краю родительского компонента;

akTop – привязать к верхнему краю родительского компонента;

akRight – привязать к правому краю родительского компонента;

akBottom – привязать к нижнему краю родительского компонента.

7. **AutoScroll** – определяет, будут ли автоматически появляться полосы прокрутки, если размеры компонента (в данном случае – формы) недостаточны, чтобы показать все размещенные на нем элементы управления.

8. **AutoSize** – определяет, будут ли размеры компонента (в данном случае – формы) автоматически адаптироваться к его содержимому.

9. **BiDiMode** – определяет направление чтения и выравнивания текстов. Возможные значения:

bdLeftToRight – порядок чтения слева направо, выравнивание не изменяется, полоса прокрутки размещается справа;

bdRightToLeft – порядок чтения справа налево, выравнивание изменяется, полоса прокрутки размещается слева;

bdRightToLeftNoAlign – порядок чтения справа налево, выравнивание не изменяется, полоса прокрутки размещается слева;

bdRightToLeftReadingOnly – порядок чтения справа налево, выравнивание и положение полосы прокрутки не изменяется.

10. **BorderIcons** – определяет комбинацию системных кнопок в полосе заголовка формы. Содержит следующие подсвойства:

biSystemMenu – определяет наличие Системного меню;

biMinimize – определяет наличие кнопки Свернуть;

biMaximize – определяет наличие кнопки Развернуть;

biHelp – определяет наличие кнопки Справка, если подсвойства **biMinimize** и **biMaximize** выключены (имеют значение *false*) или свойство **BorderStyle** (см. п. 11) имеет значение *bsDialog*.

Определенные комбинации подсвойств **BorderIcons** и свойства **BorderStyle** взаимно исключительны. Например, **BorderStyle** со значением *bsDialog* и **biMinimize**:=*true*, **biMaximize**:=*true*.

11. **BorderStyle** – задает стиль рамки (внешней границы) формы и определяет, могут ли изменяться размеры формы с помощью мыши. Возможные значения:

bsDialog – размеры не изменяемые, рамка стандартная для диалогового окна;

bsNone – размеры не изменяемые, рамка отсутствует, полоса заголовка отсутствует;

bsSingle – размеры не изменяемые, рамка имеет одинарную толщину;

bsSizeable – стандартная изменяемая рамка;

bsSizeToolWin – то же, что и *bsSizeable*, но с полосой заголовка меньшего размера;

bsToolWindow – то же, что и *bsSingle*, но с полосой заголовка меньшего размера.

12. **BorderWidth** – определяет ширину рамки формы в пикселях.

13. **Caption** – определяет текст заголовка компонента (в данном случае – формы).

14. **ClientHeight** – задает высоту клиентской области в пикселях.

15. **ClientWidth** – задает ширину клиентской области в пикселях.

16. **Color** – определяет цвет фона компонента (в данном случае – формы). Возможные значения:

статические цвета:

clBlack – черный;

clMaroon – темно-бордовый;

clGreen – зеленый;

clOlive – оливково-зеленый;

clNavy – темно-синий;

clPurple – пурпурный;

clTeal – морской волны;

clGray – серый;

clSilver – серебряный;

clRed – красный;

clLime – лимонно-зеленый;

clYellow – желтый;

clBlue – синий;

clFuchsia – сиреневый;

clAqua – голубой;

clWhite – белый;

clMoneyGreen – медно-зеленый;

clSkyBlue – небесно-голубой;

clCream – кремовый;

clMedGray – средне-серый;

системные цвета (определяются цветовой схемой в свойствах Экрана Панели управления Windows):

clActiveBorder – текущий цвет бордюра активного окна;

clActiveCaption – текущий цвет фона полосы заголовка в активном окне;

clAppWorkspace – текущий цвет рабочей области приложений;

clBackground – текущий цвет фона стола Windows;

clBtnFace – текущий цвет поверхности кнопок;

clBtnHighlight – текущий цвет выделенной кнопки;

clBtnShadow – текущий цвет теней, отбрасываемых кнопками;

clBtnText – текущий цвет текста кнопок;

clCaptionText – текущий цвет текста заголовка в активном окне;

clDefault – цвет по умолчанию;

clGradientActiveCaption – цвет правой стороны полосы заголовка активного окна в Windows 98 или Windows 2000;

clGradientInactiveCaption – цвет правой стороны полосы заголовка неактивного окна в Windows 98 или Windows 2000;

clGrayText – текущий цвет текста недоступных элементом;

clHighlight – текущий цвет фона выделенного текста;

clHighlightText – текущий цвет выделенного текста;

clHotLight – текущий цвет, которым отображаются элементы при наведении на них мышью (например, подписи к значкам файлов и папок при использовании одиночного щелчка);

clInactiveBorder – текущий цвет бордюра неактивного окна;

clInactiveCaption – текущий цвет фона полосы заголовка в неактивном окне;

clInactiveCaptionText – текущий цвет текста заголовка в неактивном окне;

clInfoBk – цвет фона советов (только для Windows 95 или NT 4.0).

*clInfoTex*t – цвет текста советов (только для Windows 95 или NT 4.0);

clMenu – текущий цвет фона меню;

clMenuBar – текущий цвет фона для строки меню в Windows XP, когда меню показываются как одноуровневые (линейное);

clMenuHighlight – текущий цвет, используемый для выделения пунктов меню в Windows XP, когда меню показывается как одноуровневое (линейное);

clMenuText – текущий цвет текста меню;

clNone – белый в Windows 9x, черный в NT;

clScrollBar – текущий цвет полос прокрутки;

cl3DDkShadow – цвет темных теней трехмерных элементом (только для Windows 95 или NT 4.0);

cl3DLight – светлый цвет на краях освещенных трехмерных элементов (только для Windows 95 или NT 4.0);

clWindow – текущий цвет фона окон;

clWindowFrame – текущий цвет рамок окон;

clWindowText – текущий цвет текста окон;

17. **Constraints** – задает ограничения на допустимые изменения размеров компонента (в данном случае – формы) при изменениях размеров окна приложения. Содержит следующие подсвойства:

MaxHeight – максимальная высота компонента в пикселях;

MaxWidth – максимальная ширина компонента в пикселях;

MinHeight – минимальная высота компонента в пикселях;

MinWidth – минимальная ширина компонента в пикселях;

18. **Ctl3D** – определяет, имеет ли компонент трехмерный или двухмерный вид. Это свойство используется для обеспечения обратной совместимости с 16-разрядными операционными системами. Не используется для 32-битовых версий Windows или NT4.0 и более поздних.

19. **Cursor** – определяет изображение курсора мыши, когда он расположен в области компонента (в данном случае – формы). Возможные значения:

crAppStar –	 ;	*crNo* –	 ;
crArrow –	 ;	*crNoDrop* –	 ;
crCross –	 ;	*crSizeAll* –	 ;
crDefault –	Курсор по умолчанию. Обычно это *crArrow*;	*crSizeNESW* –	 ;
crDrag –	 ;	*crSizeNS* –	 ;
crHandPoint –	 ;	*crSizeNWSE* –	;
crHelp –	;	*crSizeWE* –	;
crHourGlass –	;	*crSQLWait* –	;

crHSplit –  ; *crUpArrow* –  ;

crIBeam –  ; *crVSplit* –  .

crMultiDrag –  ;

20. **DefaultMonitor** – определяет режимы работы формы в многомониторных системах. Возможные значения:

dmActiveForm – форма отображается на том же мониторе, что и текущая активная форма;

dmDesktop – не делаются попытки разместить форму на определенном мониторе;

dmMainForm – форма отображается на том же мониторе, что и главная форма;

dmPrimary – форма отображается на мониторе, который стоит первым в свойстве **Monitor** глобального объекта **Screen**.

21. **DockSite** – определяет, является ли компонент (в данном случае – форма) приемником (контейнером) для других компонентов (клиентов) при перетаскивании и встраивании оконных объектов, используя технологию работы с экранными объектами в Windows с помощью мыши Drag-and-Dock (переместить и встроить).

22. **DragKind** – определяет, будет ли компонент-клиент (в данном случае – форма) перетаскиваться по технологии Drag-and-Dock (переместить и встроить) или Drag-and-Drop (переместить и оставить). Возможные значения:

dkDock – компонент перетаскивается для встраивания (по технологии Drag-and-Dock);

dkDrag – компонент перетаскивается обычным образом (по технологии Drag-and-Drop).

23. **DragMode** – определяет поведение компонента (в данном случае – формы) в процессе его перетаскивания по технологии Drag-and-Dock (переместить и встроить) или Drag-and-Drop (переместить и оставить). Возможные значения:

dmAutomatic – перемещение компонента начинается автоматически, когда пользователь начнет его перетаскивать с помощью мыши;

dmManual – компонент не может быть перемещен, пока приложение не вызовет метод **BeginDrag**.

24. **Enabled** – определяет доступность компонента (в данном случае – формы) для пользователя – реагирует ли он на события, связанные с мышью, клавиатурой и таймером.

25. **Font** – определяет атрибуты шрифта. Содержит следующие подсвойства:

Charset – определяет набор символов шрифта. Возможные значения (константы множества шрифтов):

ANSI_CHARSET – символы ANSI;

ARABIC_CHARSET – арабские символы (не доступно в NT 3.51);

BALTIC_CHARSET – балтийские символы (не доступно в NT 3.51);

CHINESEBIG5_CHARSET – символы китайского традиционного письма (Тайвань);

DEFAULT_CHARSET – шрифт, выбранный основным. Если описанный шрифт будет недоступен на системе, то Windows заменит его другим шрифтом.

EASTEUROPE_CHARSET – включает диакритические знаки для восточноевропейских стран (недоступно в NT 3.51);

GB2312_CHARSET – символы китайского упрощенного письма (материковый Китай);

GREEK_CHARSET – греческие символы (недоступно в NT 3.51);

HANGEUL_CHARSET – символы корейского письма (Wan sung);

HEBREW_CHARSET –символы еврейского письма (недоступно в NT 3.51);

JOHAB_CHARSET – символы корейского письма (Johab) (недоступно в NT 3.51);

MAC_CHARSET –символы Макинтош (недоступно в NT 3.51);

OEM_CHARSET – зависит от кодовой страницы операционной системы;

RUSSIAN_CHARSET – кириллические символы (недоступно в NT 3.51);

SHIFTJIS_CHARSET – символы японского письма (shift-JIS);

SYMBOL_CHARSET – стандартный набор символов;

THAI_CHARSET – символы тайского письма (недоступно в NT 3.51);

TURKISH_CHARSET – турецкие символы (недоступно в NT 3.51);

Color – определяет цвет текста (перечень возможных значений см. в п. 16);

Height – определяет высоту шрифта в пикселях. Если значение этого свойства задано отрицательным, то в размер не входит верхний пиксель каждой строки. Для задания размера шрифта можно использовать свойство **Size** – размер шрифта в пунктах, принятых в Windows. Значение свойства **Height** связано со значениями свойств **Size** и **PixelsPerInch** (число пикселей на дюйм) соотношением:

```
Font.Height:=-Font.Size*Font.PixelsPerInch/72;
```

Name – название шрифта;

Pitch – определяет расстояние между символами. Возможные значения:

fpDefault – расстояние между символами устанавливается по умолчанию, которое зависит от шрифта, определенного в свойстве **Name** шрифта объекта;

fpFixed – расстояние между символами устанавливается фиксированным, все символы шрифта имеют одну и ту же ширину;

fpVariable – расстояние между символами устанавливается переменным, символы в шрифте имеют разную ширину;

Size – определяет размер шрифта в пунктах (пт) (1 пт =1/72 дюйма, 1 дюйм =2,54 см). Если значение этого свойства задано отрицательным, то в размер входит верхний пиксель каждой строки. Для задания размера шрифта можно использовать свойство **Height** – размер шрифта в пикселях. Значение свойства **Size** связано со значениями свойств **Height** и **PixelsPerInch** (число пикселей на дюйм) соотношением:

```
Font.Size:=-Font.Height*72/Font.PixelsPerInch;
```

Style – определяет стиль, характер начертания символов заданного шрифта. Содержит следующие подсвойства:

fsBold – полужирный;

fsItalic – курсив;

fsUnderline – подчеркнутый;

fsStrikeOut – перечеркнутый горизонтальной прямой, зачеркнутый.

26. **FormStyle** – определяет стиль формы. Возможные значения:

fsMDIChild – форма – дочернее окно MDI (Multi Document Interface – интерфейс множества документов);

fsMDIForm – форма – родительское окно MDI;

fsNormal – форма – обычное окно;

fsStaOnTop – форма находится всегда поверх других окон.

27. **Height** – определяет высоту компонента (в данном случае – формы) в пикселях.

28. **HelpContext** – определяет ID номер темы, используемый в контекстно-зависимой справке. Это свойство предназначено для систем помощи, которые используют ID номера тем. Для систем помощи, которые используют ключевые слова тем, см. свойство **HelpKeyword**.

29. **HelpFile** – определяет имя файла, содержащего темы справочной системы для данной формы.

30. **HelpKeyword** – определяет ключевое слово темы, используемое в контекстно-зависимой справке. Это свойство предназначено для систем помощи, которые используют ключевые слова тем. Для систем помощи, которые используют ID номера тем, см. свойство **HelpContext**.

31. **HelpType** – определяет, использует ли справочная система ID номера или ключевые слова тем контекстно-зависимой справки. Возможные значения:

htContext – темы справочной системы идентифицируются ID номерами;

htKeyword – темы справочной системы идентифицируются ключевыми словами.

32. **Hint** – содержит текст, отображаемый во всплывающем окне подсказки или в строке состояния.

33. **HorzScrollBar** – определяет свойства горизонтальной полосы прокрутки. Содержит следующие подсвойства:

ButtonSize – определяет размеры кнопок в пикселях в полосе прокрутки;

Color – определяет цвет полосы прокрутки;

Increment – определяет, насколько переместится изображение, когда пользователь щелкает по одной из кнопок со стрелками на краю полосы прокрутки;

Margin – определяет минимальное число пикселей, которые должны отделять каждый компонент, содержащийся на форме, от ее края, до появления полос прокрутки;

ParentColor – определяет, задается ли цвет полосы прокрутки цветом, определенным в свойстве **Color** родительского компонента;

Position – определяет положение бегунка на полосе прокрутки;

Range – определяет диапазон полосы прокрутки;

Size – определяет размеры полосы прокрутки;

Smooth – определяет, должно ли динамически изменяться приращение прокрутки, когда пользователь щелкает мышкой по полосе прокрутки рядом с бегунком;

Style – определяет стиль полосы прокрутки;

ThumbSize – определяет размер бегунка;

Tracking – определяет, перемещается ли изображение в момент перемещения бегунка до того как пользователь его отпустит;

Visible – определяет видимость полосы прокрутки.

34. **Icon** – определяет пиктограмму для формы.

35. **KeyPreview** – определяет, должна ли форма получать события клавиатуры до того как их получит активный элемент управления.

36. **Left** – определяет координаты левого края компонента (в данном случае – формы) в пикселях.

37. **Menu** – определяет компонент главного меню для формы.

38. **Name** – определяет имя компонента (в данном случае – формы), по которому на него ссылаются другие компоненты.

39. **ObjectMenuItem** – определяет элемент меню, который изменяет свое состояние в результате работы на форме объекта OLE (Object Linking and Embedding – технология связывания и встраивания объектов).

40. **OldCreateOrder** – определяет порядок событий **OnCreate** и **OnDestroy**. Когда значение этого свойства равно *false* (по умолчанию), событие **onCreate** происходит после того как все конструкторы закончены, а событие **onDestroy** происходит прежде чем любые деструкторы вызываются.

41. **ParentBiDiMode** – определяет, действует ли для компонента (в данном случае – для формы) значение свойства **BiDiMode** родительского компонента.

42. **ParentFont** – определяет, будет ли для компонента (в данном случае – для формы) использоваться шрифт родительского компонента.

43. **PixelsPerInch** – определяет количество пикселей на один дюйм, которое используется при масштабировании формы на экране (если свойство **Scaled**:=*true*).

44. **PopupMenu** – определяет связанный с компонентом (в данном случае – с формой) компонент всплывающего меню.

45. **Position** – определяет размер и положение формы на экране после запуска приложения (если свойство **WindowState**:=*wsNormal*). Возможные значения:

poDefault – положение формы на экране, ее высота и ширина определяются операционной системой;

poDefaultPosOnly – размер формы остается таким, каким он был задан во время разработки, а ее положение на экране определяется операционной системой;

poDefaultSizeOnly – положение формы на экране остается таким, каким оно было задано во время разработки, а ее размер определяется операционной системой;

poDesigned – положение формы на экране, ее высота и ширина остаются такими, какими они были заданы во время разработки;

poDesktopCenter – форма расположена в центре экрана, а ее размер остается таким, каким он был задан во время разработки (расположение формы не адаптируется к многомониторным системам, определяемым свойством **DefaultMonitor**);

poMainFormCentre – форма расположена в центре главной формы приложения, а ее размер остается таким, каким он был задан во время разработки. Это значение может использоваться только для вторичных форм;

poOwnerFormCentre – форма расположена в центре формы, указанной в свойстве **Owner** (это свойство не отображается в Object Inspector), а ее размер остается таким, каким он был задан во время разработки;

poScreenCenter – форма расположена в центре экрана, а ее размер остается таким, каким он был задан во время разработки (расположение формы адаптируется к многомониторным системам, определяемым свойством **DefaultMonitor**);

46. **PrintScale** – определяет масштаб формы при печати. Возможные значения:

poNone – изображение формы при печати не масштабируется;

poPrintToFit – изображение формы печатается того же размера, который виден на экране;

poProportional – изображение формы при печати масштабируется пропорционально ее размерам на экране.

47. **Scaled** – определяет, изменяются ли размеры формы в соответствии со значением свойства **PixelsPerInch**.

48. **ScreenSnap** – определяет, фиксируется ли форма у границ экрана, когда пользователь перемещает форму. Для управления расстоянием фиксации используется свойство **SnapBuffer**.

49. **ShowHint** – включает или отключает показ всплывающего окна подсказки при задержке курсора мыши над компонентом.

50. **SnapBuffer** – определяет максимальное число пикселей между границей формы и экраном, после которого наступает фиксация формы к границе экрана.

51. **Tag** – свойство, которое разработчик приложения может использовать по своему усмотрению. В его значении может храниться любое целое число.

52. **Top** – определяет координаты верхнего края компонента (в данном случае – формы) в пикселях.

53. **TransparentColor** – определяет, будет ли цвет на форме прозрачным. В свойстве **TransparentColorValue** указывается, какой именно цвет будет прозрачным.

54. **TransparentColorValue** – определяет цвет, который будет на форме прозрачным, если значение свойства **TransparentColor**:=*true*.

55. **UseDockManager** – определяет, используется ли диспетчер встраивания для управления процессом встраивания объектов-клиентов в компонент-приемник (контейнер) по технологии Drag-and-Dock – переместить и встроить.

56. **VertScrollBar** – определяет свойства вертикальной полосы прокрутки. Описание содержащихся подсвойств см. в свойстве **HorzScrollBar**.

57. **Visible** – определяет, видим или невидим компонент (в данном случае – форма) во время выполнения приложения.

58. **Width** – определяет горизонтальный размер компонента (в данном случае – формы) в пикселях.

59. **WindowMenu** – определяет стандартное меню Окно для родительской формы MDI (Multi Document Interface – интерфейс множества документов).

60. **WindowState** – определяет вид окна формы после запуска приложения. Возможные значения:

wsMaximized – форма развернута;

w Minimized – форма свернута;

wsNormal – форма находится в нормальном состоянии.

События Формы

1. **Action** – обеспечивает доступ к событиям действия, связанного с элементом управления (в данном случае – с формой). Это действие определяется в свойстве **Action (**см. п. Свойства Формы).

2. **ActiveControl** – обеспечивает доступ к событиям компонента, который будет иметь фокус (является активным) сразу после запуска приложения. Этот компонент определяется в свойстве **ActiveControl** (см. п. Свойства Формы).

3. **Menu** – обеспечивает доступ к событиям главного меню формы. Этот компонент определяется в свойстве **Menu** (см. п. Свойства Формы).

4. **ObjectMenuItem** – обеспечивает доступ к событиям элемента меню, который изменяет свое состояние в результате работы на форме объекта OLE (Object Linking and Embedding – технология связывания и встраивания объектов). Этот компонент определяется в свойстве **ObjectMenuItem** (см. п. Свойства Формы).

5. **OnActivate** – происходит при передаче форме фокуса (когда форма становится активной).

6. **OnCanResize** – происходит перед изменением размеров компонента (в данном случае – формы).

7. **OnClick** – происходит при щелчке мыши на компоненте (в данном случае – на форме).

8. **OnClose** – происходит при закрытии формы.

9. **OnCloseQuery** – происходит перед закрытием формы.

10. **OnConstrainedResize** – происходит при изменении размеров компонента (в данном случае – формы).

11. **OnContextPropup** – происходит при вызове всплывающего меню.

12. **OnCreate** – происходит при создании формы.

13. **OnDblClick** – происходит при двойном щелчке кнопкой мыши на компоненте (в данном случае – на форме).

14. **OnDeactivate** – происходит, когда форма теряет фокус.

15. **OnDestroy** – происходит при уничтожении формы.

16. **OnDockDrop** – происходит, когда компонент-клиент начинает встраиваться в компонент-приемник (в данном случае – в форму) по технологии Drag-and-Dock (переместить и встроить).

17. **OnDockOver** – происходит и повторяется, пока пользователь перемещает компонент-клиент над компонентом-приемником (в данном случае – над формой) по технологии Drag-and-Dock (переместить и встроить).

18. **OnDragDrop** – происходит в момент отпускания пользователем перетаскиваемого компонента над компонентом (в данном случае – над формой) по технологии Drag-and-Drop (переместить и оставить).

19. **OnDragOver** – происходит и повторяется, пока пользователь перемещает перетаскиваемый компонент над компонентом (в данном случае – над формой) по технологии Drag-and-Drop (переместить и оставить).

20. **OnEndDock** – происходит после завершения встраивания компонента-клиента по технологии Drag-and-Dock (переместить и встроить) или в случае отмены перетаскивания.

21. **OnGetSiteInfo** – происходит в момент начала перетаскивания компонента-клиента и непрерывно повторяется в процессе перетаскивания по технологии Drag-and-Dock (переместить и встроить).

22. **OnHelp** – происходит при обращении из формы к справочной системе.

23. **OnHide** – происходит при исчезновении формы с экрана (то есть, когда значение свойства **Visible** принимает значение *false*).

24. **OnKeyDown** – происходит при нажатии пользователем любой клавиши клавиатуры (при этом в обработчик события передается код нажатой клавиши).

25. **OnKeyPress** – происходит при нажатии пользователем любой клавиши символа на клавиатуре, кроме функциональных (при этом в обработчик события передается символ нажатой клавиши).

26. **OnKeyUp** – происходит при отпускании любой клавиши, ранее нажатой на клавиатуре.

27. **OnMouseDown** – происходит при нажатии пользователем клавиши мыши над компонентом (в данном случае – над формой).

28. **OnMouseMove** – происходит при перемещении пользователем курсора мыши над компонентом (в данном случае – над формой).

29. **OnMouseUp** – происходит при отпускании пользователем ранее нажатой клавиши мыши над компонентом (в данном случае – над формой).

30. **OnMouseWheal** – происходит при вращении пользователем колеса прокрутки мыши.

31. **OnMouseWhealDown** – происходит при нажатии пользователем на колесо прокрутки мыши.

32. **OnMouseWhealUp** – происходит при отпускании пользователем ранее нажатого колеса прокрутки мыши.

33. **OnPaint** – происходит при получении сообщения Windows о необходимости перерисовки компонента (в данном случае – формы).

34. **OnResize** – происходит немедленно после изменения размеров компонента (в данном случае – формы).

35. **OnShortCut** – происходит, когда пользователь нажимает клавишу на клавиатуре (до события **OnKeyDown**).

36. **OnShow** – происходит при появлении формы на экране (то есть, когда значение свойства **Visible** принимает значение *true*).

37. **OnStartDock** – происходит, когда пользователь начинает процесс перетаскивания компонента (в данном случае – формы) по технологии Drag-and-Dock (переместить и встроить).

38. **OnUnDock** – происходит в компоненте-приемнике (в данном случае – формы) в момент, когда пользователь начинает перетаскивать компонент-клиент из компонента-приемника.

39. **PopupMenu** – обеспечивает доступ к событиям всплывающего меню, которое связано в данном случае с формой. Этот компонент определяется в свойстве **PopupMenu** (см. п. Свойства Формы).

40. **WindowMenu** – обеспечивает доступ к событиям меню Окно для родительской формы MDI (Multi Document Interface – интерфейс множества документов). Это меню определяется в свойстве **WindowMenu** (см. п. Свойства Формы).

Некоторые процедуры и функции Object Pascal для работы со строками.

Процедуры

Delete(Var S: string; Index, Count: integer);

Удаляет **Count** символов из строки **S**, начиная с позиции **Index**.

Insert(SubS: string; Var S: string; Index: integer);

Вставляет подстроку **SubS** в строку **S**, начиная с позиции **Index**.

Str(X [: Width [: Decimals]]; Var S: string);

Преобразует численное значение **X** в строку **S**. Необязательные параметры: **Width** – ширина поля, **Decimals** – число цифр после разделителя целой и дробной частей.

Val(S: string; Var X; Var Code: integer);

Преобразует строку **S** в числовое значение **X**. Параметр **Code** содержит признак ошибки преобразования (**Code**=*0* – нет ошибки).

Функции

CompareText(const S1, S2: string): integer;

Сравнивает две строки **S1** и **S2** без учета регистра. Возвращает значение < *0*, если **S1**<**S2**, *0* если **S1**=**S2**, и > **0** если **S1**>**S2**.

Concat(S1 [, S2,…, Sn]: string): string;

Возвращает строку, склеенную из строк **S1**, **S2**,…, **Sn**. Идентична операции «+» для строк.

Copy(S: string; Index, Count: integer): string;

Возвращает подстроку из строки **S**, начиная с позиции **Index** и длиной **Count** символов.

FloatToStr(X: extended): string;

Преобразует численное значение **X** в строку с точностью 15 цифр.

FloatToStrF(X: extended, Format: TFloatFormat; Precision, Digits: integer): string;

Преобразует численное значение **X** в строку, используя параметр **Format** с точностью **Precision** и числом цифр **Digits**.

Возможные значения параметра **Format**:

ffGeneral – общий формат чисел. Значение преобразуется в формат научный *ffExponent* или с фиксированной точкой *ffFixed*, в зависимости от того, какой из них дает более короткую запись. В научном формате общее число цифр (включая цифру перед точкой) равно числу, указанному в параметре **Precision** (точность). По умолчанию точность равна 15. Заключающие нули после десятичной точки в результирующей строке отбрасываются, а сама десятичная точка появляется, только если это необходимо. Результирующая строка использует формат с фиксированной точкой, если число цифр слева от десятичной точки не превышает заданной точности и если значение числа не меньше 0.00001. В остальных случаях используется научный формат, а параметр **Digits** определяет минимальное количество цифр в экспоненте (от 0 до 4);

ffExponent – научный формат. Значение преобразуется в строку вида «–d.ddd...E+ddd», где d означает цифру. Иначе говоря, число представляется в виде $–d.ddd...\cdot 10^{+ddd}$. Отрицательные числа начинаются со знака «–».

Перед десятичной точкой всегда имеется одна цифра. Общее число цифр перед экспонентой (включая цифру перед точкой) определяется параметром **Precision**. По умолчанию точность равна 15. После символа «E» обязательно следует знак «+» или «–». Параметр **Digits** определяет минимальное количество цифр в экспоненте (от 0 до 4);

ffFixed – формат с фиксированной точкой. Значение преобразуется в строку вида «–ddd.ddd...», где d означает цифру. Отрицательные числа начинаются со знака «–». Перед десятичной точкой всегда имеется одна цифра. Число цифр после десятичной точки определяется параметром **Precision** (от 0 до 18). По умолчанию точность равна 2. Если количество цифр слева от десятичной точки большее, чем указанная точность, то результирующая строка будет использовать научный формат;

ffNumber – числовой формат. Значение преобразуется в строку вида «–d,ddd,ddd.ddd...», где d означает цифру. Этот формат подобен формату с фиксированной точкой *ffFixed*, но отличается наличием в результирующей строке разделителей тысяч;

ffCurrency – денежный формат. Значение преобразуется в строку, представляющую собой денежную сумму «–d ddd ddd.ddp.», где d означает цифру. Преобразование определяется глобальными переменными **CurrencyString**, **CurrencyFormat**, **NegCurrFormat**, **ThousandSeparator**, **DecimalSeparator** и **CurrencyDecimals**, которые задаются на вкладке Региональные параметры объекта Язык и региональные стандарты Панели управления Windows. Если задан параметр **Precision**, то он заменяет значение, содержащееся в глобальной переменной **CurrencyDecimals**. Количество цифр после десятичной точки определяется параметром **Digits** (от 0 до 18).

IntToStr(X: integer): string;

Преобразует целое число **X** в строку.

Length(S: string): integer;

Возвращает число символов в строке **S**.

Pos(SubS, S: string): integer;

Возвращает позицию, начиная с которой в строке **S** располагается подстрока **SubS**. Если **S** не содержит **SubS**, то функция возвращает *0*.

StrToFloat(const S: string): extended;

Преобразует строку **S** в действительное число.

StrToInt(const S: string): integer;

Преобразует строку **S** в целое число.

ExtractFileName(const FullFileName: string): string;

Возвращает в виде строки имя и расширение из полного имени файла.

ExtractFileExt(const FullFileName: string): string;

Возвращает в виде строки расширение из полного имени файла.

ExtractFilePath(const FullFileName: string): string;

Возвращает в виде строки путь из полного имени файла.

Библиографический список

Основная:

1. *Мануйлов, В.Г.* Разработка программного обеспечения на Паскале: учебное пособие для преподавателей, студентов, старшеклассников / В.Г. Мануйлов. – М.: Приор, 1996.

2. *Фаронов, В.В.* Основы Турбо-Паскаля. Книга 1 / В.В. Фаронов. – М.: МВТУ. – Фесто дидактик, 1992.

3. *Оузьер, Д.* Delphi 3. Освой самостоятельно / Ден Оузьер и др. – М.: Бином, 1998.

4. *Архангельский, А.Я.* Программирование Delphi 5 / А.Я. Архангельский. – М.: Бином, 2000.

5. *Дарахвелидзе, П.Г.* Программирование в Delphi 4. / П.Г. Дарахвелидзе, Е.П. Марков. – СПб.: БХВ – Санкт Петербург, 1999.

6. *Архангельский, А.Я.* Приемы программирования в Delphi на основе VCL / А.Я. Архангельский. – М.: Бином-Пресс, 2006.

Дополнительная:

1. *Васильев, П.П.* Турбо паскаль в примерах и задачах: освой самостоятельно: учеб. пособие / П.П. Васильев. – М.: Финансы и статистика, 2002.

2. *Немнюгин*, С.А. Turbo Pascal: практикум / С.А. Немнюгин. – СПб.: Питер, 2002.

3. *Рубенкинг, Нейл Дж.* Delphi 3 для “чайников” / Нейл Дж. Рубенкинг. – Киев-Москва: Диалектика, 1997.

4. *Лалетин, Н.В*. Технология программирования: учебное пособие / Н.В. Лалетин; Краснояр. гос. пед. ун-т им. В.П. Астафьева. – Красноярск, 2005. – 84 с.

Сергей Владимирович Бутаков

РАЗРАБОТКА WINDOWS-ПРИЛОЖЕНИЙ В СРЕДЕ DELPHI

Лабораторный практикум

Редактор Н.А. Агафонова

Корректор С.А. Бовкун

660049, г. Красноярск, ул. Ады Лебедевой, 89.

Редакционно-издательский отдел КГПУ им. В.П. Астафьева,

тел. (391) 217-17-52

Формат 60×84 $^{1}/_{16}$.

Усл. печ. л. 6,8.

Для заметок