Министерство образования Республики Беларусь
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Кафедра инженерной математики

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по информатике
для студентов инженерных специальностей
приборостроительного факультета

В трех частях

Часть 3

Под общей редакцией В.А.Нифагина

Минск 2004

УДК 002.6 + 681.3 (075.8)

Составители:
А.В.Стрелюхин, Л.В.Бокуть, О.Г.Вишневская

Рецензент
доктор технических наук Н.А.Ярмош

Лабораторный практикум включает разделы по основам программирования в среде Delphi, предусмотренные программой курса информатики для студентов приборостроительного факультета.

Часть 1 настоящего издания вышла в свет в 2001 году, часть 2 – в 2002 году.

Предисловие

Современные визуальные системы программирования позволяют создавать достаточно сложные законченные Windows-приложения, причем реализация программного интерфейса во многом возлагается на библиотеки встроенных компонентов. Одной из лучших сред разработки приложений, направленных на решение научно-технических задач, является Delphi.

Данный практикум предназначен для работы со средой Delphi в качестве инструмента создания программ. Он не только знакомит с основными понятиями и архитектурой Delphi, но и способствует получению знаний, необходимых для создания реальных приложений. В практикуме предлагается комплекс работ по освоению базовых элементов языка Object Pascal и его использования в Delphi. Изложение построено таким образом, чтобы студенты сумели понять основные принципы разработки компьютерных программ, не увязнув в многочисленных тонкостях языка программирования. При этом они проходят путь от решения простейших задач до создания собственных приложений.

В лабораторных работах содержатся сведения об алгоритмизации и программировании, типах данных, операторах, процедурах, функциях и модулях, приводятся описания основных конструкций языка, включая логические и математические операторы, рассматриваются структуры управления, контроль циклов, стандартные и пользовательские типы данных, массивы и множества. Изучаются основные "строительные блоки" Delphi: дизайнер форм, инспектор объектов, редактор кода и т.д. Рассматриваются объектная модель Delphi, свойства и методы основных визуальных компонентов, событийное программирование, а также использование этих концепций для создания проектов. Описываются средства разработки Delphi, состав и характеристика элементов проекта приложения, процесс программирования на языке Object Pascal. Даны примеры визуальных компонентов, используемых для создания интерфейса приложений, рассмотрена техника работы с текстовой информацией, кнопками и переключателями, формами, являющимися центральной частью любого приложения, описано создание меню, определены компоненты для работы с массивами и графикой. Даются понятия, используемые в теории баз данных, обсуждаются элементы реляционных баз данных и особен-

ности их использования, описываются методы создания таблиц и приложений баз данных, приемы работы с данными.

Все работы иллюстрированы конкретными примерами, приведены схемы их решения, включающие основные приемы программирования и практической работы в интегрированной среде Delphi, описаны требования к содержанию отчета, приведены контрольные вопросы для самоконтроля, а также индивидуальные задания, выполнение которых позволит получить навыки решения практических задач.

Лабораторная работа № 22

ИНТЕГРИРОВАННАЯ СРЕДА DELPHI. РАЗРАБОТКА ПРИЛОЖЕНИЙ В DELPHI

Цель работы: изучение среды Delphi.

Используемые программные средства: Delphi.

22.1. Теоретические сведения

Delphi относится к системам визуального программирования, которые называются также системами RAD (Rapid Application Development) – быстрая разработка приложений.

Для написания программ в Delphi используется разработанный фирмой Borland язык программирования Object Pascal, основы которого приведены в прил. 1.

Запуск Delphi в среде Windows: **Пуск – Программы – Borland Delphi – Delphi**.

Интегрированная среда разработки Delphi представляет собой многооконную систему. После загрузки интерфейс Delphi имеет 4 окна (рис. 22.1).

При запуске Delphi автоматически создается новый проект Project 1. Название проекта приложения выводится в строке заголовка главного окна. Delphi является однодокументной средой (позволяет работать только с одним проектом приложения).

В **главном окне** Delphi отображаются: **главное меню, панели инструментов, палитра компонентов.**

Главное меню содержит обширный набор команд для доступа к функциям Delphi и предназначено для управления процессом создания программы.

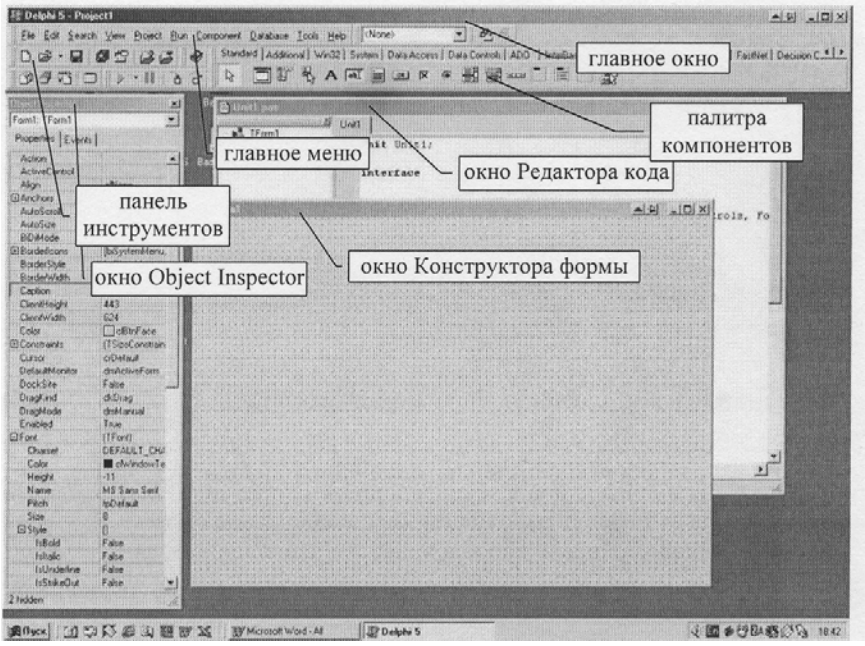

Рис. 22.1. Визуальное изображение среды Delphi

Панели инструментов содержат набор кнопок для вызова часто используемых команд главного меню.

Палитра компонентов содержит наборы компонентов, размещаемых в создаваемых формах. **Компоненты** являются структурными единицами и делятся на **визуальные** и **невизуальные**. Кроме того, все компоненты делятся на группы, каждая из которых в палитре компонентов располагается на отдельной вкладке, а сами компоненты представлены соответствующими пиктограммами. Настройка *Палитры компонентов* проводится через пункты **Component** – **Configure Palette** главного меню.

Для каждого компонента при создании программы выполняются следующие операции:

– выбор компонента в *Палитре компонентов* и размещение его на форме;

– изменение свойств компонента.

Выбор компонента в *Палитре компонентов* выполняется щелчком мыши на нужном компоненте. Для выбора нескольких компо-

нентов в *Палитре компонентов* нужно нажать и удерживать клавишу <Shift>.

Свойства компонента представляет собой атрибуты, определяющие способ отображения и функционирования компонентов при выполнении приложения. В *Object Inspector* приводятся названия всех свойств компонента, которые доступны на этапе разработки программы, и значения, которые они принимают. Свойства, доступные в *Object Inspector*, также можно изменять и при выполнении приложения.

Основные свойства компонентов приведены в прил. 2.

В окне **Конструктора формы** выполняется проектирование формы, для чего на форму помещаются необходимые компоненты. По умолчанию форма имеет имя Form1. **Форма** – контейнер, в котором размещаются визуальные и невизуальные компоненты, при этом сама форма является компонентом типа TForm.

Некоторые свойства компонента Form приведены в табл. 22.1.

Таблица 22.1

Свойства компонента Form	Описание свойств
1	2
`BorderIcons`	определяет набор кнопок, которые имеются в полосе заголовка
`biSystemMenu`	кнопка системного меню
`biMinimize`	кнопка `Свернуть`
`biMaximize`	кнопка `Развернуть`
`biHelp`	кнопка `Справки`
`BorderStyle`	определяет общий вид окна и операции с ним, которые разрешено выполнять пользователю
`bsDialog`	неизменяемое по размерам окно (окна диалогов)
`bsSingle`	окно, размер которого нельзя изменять, потянув курсором мыши край окна, но можно менять кнопками в полосе заголовка
`bsNone`	окно без полосы заголовка, не допускает изменения размера и перемещения по экрану

Окончание табл. 22.1

1	2
bsSizeable	обычный вид окна Windows
bsToolWindow	то же, что и bsSingle, но с полосой заголовка меньшего размера
bsSizeToolWin	то же, что и bsSizeable, но с полосой заголовка меньшего размера и отсутствием кнопок изменения размера
Caption	содержит строку для надписи заголовка
FormStyle	стиль формы
Position	положение окна приложения
poDefault	определяется Windows
poDesigned	определяется разработчиком
poScreenCenter	по центру экрана
WindowState	определяет вид, в котором окно первоначально предъявляется пользователю при выполнении приложения
wsMaximized	окно развернуто на весь экран
wsMinimized	окно свернуто
wsNormal	нормальный вид окна

В окне **редактора кода** содержится исходный текст разрабатываемой программы. Первоначально в нем имеется одна страница Unit1.pas кода для новой формы Form1.

Переключение между окнами *конструктора формы* и *редактора кода* выполняется с помощью функциональной клавиши **F12** или нажатием кнопки  на *Панели инструментов*.

Окно **Object Inspector** (Инспектор объектов) предназначено для задания и отображения свойств компонентов, расположенных на форме, на этапе разработки программы. Окно **Object Inspector** имеет две страницы (вкладки): **Properties** (свойства) для изменения свойств выбранных компонентов и **Events** (события) для определения реакции компонента на то или иное событие. Окно **Object Inspector** можно отобразить/спрятать с помощью нажатия функциональной клавиши **F11**.

Приложение (программа), создаваемое в среде Delphi, состоит из нескольких элементов (файлов), объединенных в **проект** (табл. 22.2).

Таблица 22.2

Название файлов	Расширение файлов
Файл проекта	*.dpr
Файлы описания форм	*.dfm
Файлы модулей форм	*.pas
Файлы модулей (без формы)	*.pas
Файл параметров проекта	*.opt
Файл ресурсов	*.res

Кроме приведенных файлов автоматически могут создаваться их резервные копии, отличительным признаком которых является наличие знака "~" в расширении файла, например *.~dp – резервная копия для dpr-файлов.

Взаимосвязи между файлами проекта показаны на рис. 22.2.

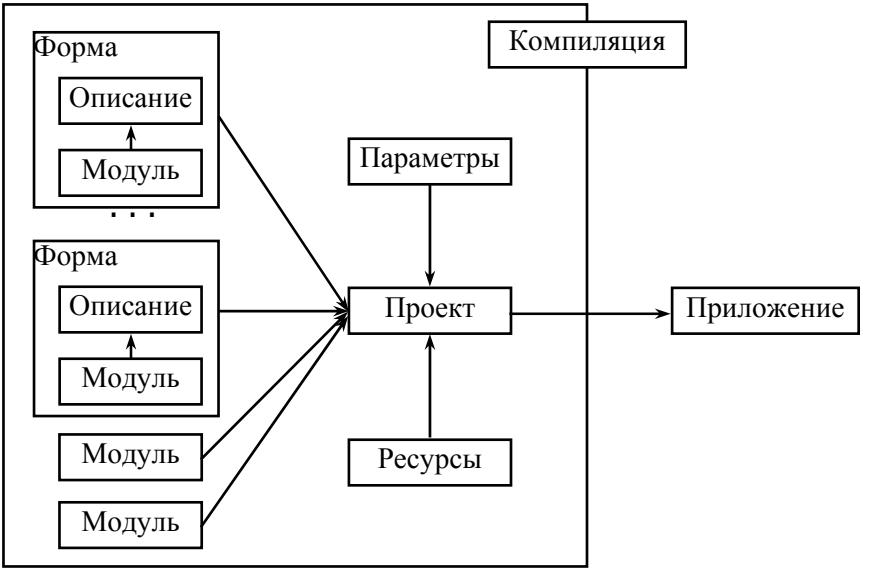

Рис. 22.2. Взаимосвязи между файлами проекта

Файл проекта является основным и представляет собой собственно программу. Имя проекта совпадает с именем файла проекта,

то же название имеют файлы ресурсов и параметров проекта. Файл проекта формируется Delphi автоматически.

Файл описания формы содержит характеристики формы и ее компонентов. Для каждой формы в составе проекта автоматически создаются файл описания формы (расширение *dfm*) и файл модуля (расширение *pas*). При конструировании формы с помощью *Конструктора формы* и *Object Inspector* изменения в *файл описания* вносятся автоматически.

Кроме модулей, входящих в состав форм, можно использовать модули, не связанные с формой. Они оформляются по обычным правилам Object Pascal и сохраняются в отдельных файлах. Для подключения модуля его имя указывается в разделе *Uses* того модуля или проекта, в котором он используется. В отдельных модулях целесообразно размещать процедуры, функции, переменные, общие для нескольких модулей проекта.

В **файле ресурсов** содержатся пиктограммы, растровые изображения и курсоры, которые являются ресурсами Windows.

Файл параметров проекта представляет собой текстовый файл, в котором располагаются параметры проекта и их значения.

Запуск проекта из среды Delphi осуществляется командами меню **Run** – **Run**, нажатием функциональной клавиши **F9** или нажатием кнопки  на *Палитре инструментов*. При этом происходит компиляция проекта и создается готовый к выполнению файл с расширением *exe*.

Для создания нового проекта из среды Delphi надо выполнить команду главного меню **File** – **New Application.**

Чтобы сохранить текущий проект, надо выполнить команду главного меню **File** – **Save all**. Если до сохранения проекту не было присвоено имя, то в открывшемся диалоговом окне будет предложено сохранить файл с исходным тестом (по умолчанию Unit1.pas) (нажать кнопку *Сохранить*), а затем сохранить файл проекта (по умолчанию Project1.dpr) (нажать кнопку *Сохранить*).

В этом же каталоге после компиляции будет сохранено и само приложение (расширение файла *exe*). Имя файла приложения совпадает с именем файла проекта.

Для открытия проекта надо выполнить команду главного меню **File** – **Open Project**, в диалоговом окне выбрать имя проекта и нажать кнопку *Открыть*.

Разработка приложений в Delphi. Понятие событий

Разработка приложений в Delphi состоит из двух этапов:

– создание интерфейса приложения;

– определение функциональности приложения.

Интерфейс приложения определяет способ взаимодействия пользователя и приложения, т.е. внешний вид формы при выполнении приложения и то, каким образом пользователь управляет приложением. Интерфейс создается путем размещения в форме компонентов.

Функциональность приложения определяется процедурами, которые выполняются при возникновении определенных событий.

Простейшее приложение представляет собой каркас (заготовку), обеспечивающее все необходимое для каждого приложения, и является равноправным приложением Windows.

Операционная система Windows работает по принципу обработки возникающих в ней **событий** (например, щелчок мыши на кнопке, выбор пункта меню, нажатие клавиши, изменение размеров окна и т.д.) и передает их выполняющейся программе. Обычно программа ожидает сообщений о событиях и реагирует на них. Сообщения обрабатываются программой не одновременно, а последовательно. Программа в среде Delphi составляется как описание алгоритмов, которые надо выполнить при возникновении какого-либо события для активного приложения (открытие формы, щелчок мыши на компоненте, нажатие клавиши на клавиатуре и т.д.). Например события, возникающие:

onClick – при щелчке левой кнопкой мыши на компоненте или при других способах нажатия на управляющий элемент;

onDblClick – при двойном нажатии левой кнопки мыши;

onKeyDown – при нажатии клавиши на клавиатуре;

onKeyUp – при отпускании клавиши на клавиатуре;

onMouseDown – при нажатии любой кнопки мыши;

onMouseUp – при отпускании кнопки мыши;

onCreate – один раз при создании формы;

onClose – при закрытии формы.

Обработчик событий для компонента выбирается в *Object Inspector* во вкладке *Events* или двойным щелчком левой кнопки мыши на этом компоненте.

С помощью Delphi можно создавать программы в стиле MS-DOS, так называемые **консольные приложения**. Запуск Delphi в режиме консольного приложения приведен в прил. 3.

22.2. Порядок выполнения работы

Изучить структуру интегрированной среды Delphi, свойства основного компонента Form и выполнить контрольный пример.

Контрольный пример. Создать простейшее приложение Windows на основе компонента Form. Изучить основные свойства этого компонента.

Решение

1. Открыть новый проект Delphi: **File** – **New Application**.
2. В *Object Inspector* изменить свойство `Caption` компонента Form1 с 'Form1' на 'Простейшее приложение'.
3. Запустить проект на компиляцию и выполнение с помощью клавиши **F9**.
4. Закрыть приложение, нажав на значок  .
5. С помощью *Object Inspector* для компонента Form1 изменить свойство `Color`, задавая ему различные значения, например `clRed`, `clBlue` и др.
6. Изменить свойства `Height` и `Width` компонента Form1, задавая этим свойствам различные значения. Например,

```
Height = 480, 350, 130;
Width = 120, 200, 400.
```

7. Задавая различные значения свойствам `BorderIcons` и `BorderStyle`, запустить проект на компиляцию и выполнение и проанализировать изменения во внешнем виде окна приложения. Например,

```
BorderStyle = bsSizeable, bsSingle, bsDialog, bsToolWindow;
BorderIcons:
            biSystemMenu = true, false;
            biMinimize = = true, false;
            biMaximize = true, false;
```

8. Аналогично, изменяя свойства `FormStyle`, `Position` и `WindowState`, запустить проект на выполнение и проанализировать изменения во внешнем виде окна приложения. Например,

```
FormStyle = fsNormal, fsStayOnTop
Position = poDefault, poDesigned, poScreenCenter
WindowState = wsNormal, wsMaximized
```

22.3. Содержание отчета

1. Краткий обзор теоретической части.
2. Ответы на контрольные вопросы.

22.4. Контрольные вопросы

1. Для чего используется интегрированная среда Delphi?
2. Перечислить основные вкладки *Палитры компонентов*.
3. Для чего используется *Object Inspector*?
4. Что входит в состав проекта Delphi?
5. Как происходит выбор компонента из *Палитры компонентов* и его размещение на форме?

Лабораторная работа № 23

ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ И РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

Цель работы: приобретение практических навыков программирования в Delphi линейных и разветвляющихся алгоритмов.

Используемые программные средства: Delphi.

23.1. Теоретические сведения

Линейными называются алгоритмы, в которых команды выполняются в последовательном порядке, т.е. одна за другой. Для их программирования используются операторы присваивания. Если в программе предусматривается проверка условий, при которых нарушается порядок выполнения команд в приложении, то такие алгоритмы называются *разветвляющимися*. Для их организации в языке Object Pascal используются операторы условия (**if**) и операторы выбора (**case**) (см прил. 1).

Работа с компонентами

Ввод, редактирование и отображение информации выполняется в специальных полях или областях формы. Для этих целей Delphi предлагает различные компоненты.

Компоненты **Edit** типа TEdit  (панель **Standard)** и **MaskEdit** типа TMaskEdit  (панель **Additional**) представляют собой строку для обработки информации и относятся к **Однострочным редакторам**. Компонент **MaskEdit**, в отличие от Edit, предоставляет возможность ограничения вводимой информации по шаблону.

Основное свойство компонентов – `Text`, используемое для ввода и редактирования данных в текстовом виде (тип **string**).

Для отображения информации без возможности редактирования при выполнении программы используется компонент **Label** типа TLabel  (панель **Standard**). Текст представляет собой **надпись** и чаще всего используется в качестве заголовков для других элементов. Основные свойства компонента Label представлены в табл. 23.1.

Таблица 23.1

Свойства компонента Label	Описание свойств
`Alignment`	определяет способ выравнивания текста внутри компонента:
`taLeftJustify`	по левому краю
`taCenter`	по центру
`taRightJustify`	по правому краю
`AutoSize`	автоматическая коррекция размеров компонента в зависимости от текста надписи
`Caption`	текст надписи

При использовании компонентов ввода-вывода достаточно часто требуется провести преобразование типов. Например, для того чтобы вывести с помощью компонента Edit значение переменной типа real, необходимо сначала получить строковое представление переменной. Это можно сделать с помощью функции

```
FloatToStr(Value:extended):string ,
```

которая преобразует вещественное значение `Value` в строку символов. Перечень основных функций, используемых для преобразования типов, приведен в прил. 4.

Компоненты **Кнопки** являются управляющими элементами и используются для выдачи команд на выполнение определенных функциональных действий. В Delphi имеются различные варианты кнопок: стандартная кнопка **Button** типа TButton  (панель **Standard**), кнопка с рисунком **BitBtn** типа TBitBtn  (панель **Additional**) и кнопка быстрого доступа **SpeedButton** типа TSpeedButton  (панель **Additional**). На поверхности кнопки может содержаться поясняющая надпись (свойство Caption).

Основным для кнопок является событие `OnClick`, возникающее при нажатии на кнопку. При этом кнопка принимает соответствующий вид, подтверждающий действие. Действия, выполняемые в обработчике события OnClick, происходят сразу после отпускания кнопки.

Для организации разветвлений в Delphi используются компоненты в виде кнопок-переключателей, состояние которых (включено-выключено) визуально отражается во время выполнения приложения: **CheckBox** типа TCheckBox  , **RadioButton** типа TRadioButton  и **RadioGroup** типа TRadioGroup  . Компоненты расположены на панели **Standard**.

Основным свойством компонентов **CheckBox** и **RadioButton** является свойство `Checked` типа boolean.

Компонент **RadioGroup** представляет собой группу кнопок, являющихся взаимно исключающими, т.е. при выборе одного переключателя другие становятся невыбранными. Для управления количеством и названиями переключателей используется свойство `Items` типа `TStrings`, которое позволяет получить доступ к отдельным переключателям в группе. Отсчет строк в массиве `Items` начинается с нуля. Для работы со списком заголовков кнопок в режиме проектирования приложения значения свойства `Items` компонента можно изменить, используя **String List Editor**. Для доступа к отдельному переключателю используется свойство `ItemIndex` типа `integer`, содержащее номер переключателя. Количество столбцов для вывода информации определяется свойством `Columns`.

23.2. Порядок выполнения работы

Изучить свойства компонентов, используемых для ввода-вывода информации, для организации разветвлений и управления. Изучить операторы, используемые в *Object Pascal* для программирования линейных и разветвляющихся алгоритмов, выполнить контрольные примеры и задания соответствующего варианта.

Контрольный пример 1. Составить программу для расчета значения *f*:

$$f = z + \frac{z}{z^2 + 1} - 3{,}7 \cdot 10^{-8} + e^{x+z}.$$

Значения *z* и *x* вводятся с клавиатуры.

Решение

1. Открыть новый проект Delphi: **File** – **New Application**.

2. Установить с помощью *Object Inspector* следующие свойства компонента Form1:

```
Form1.Height = 224
Form1.Width = 286
Form1.BorderIcons
    biMaximize = false
Form1.BorderStyle = bsSingle
Form1.Position = poScreenCenter
Form1.Caption = 'Контрольный пример 1'
```

3. Расположить на форме три компонента Edit, три компонента Label и один компонент Button, как показано на рис. 23.1.

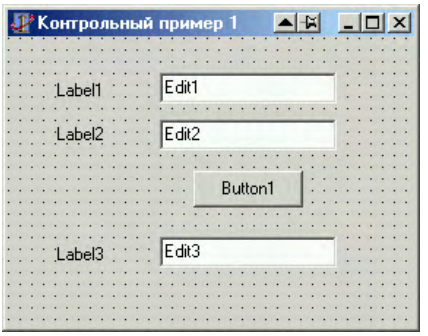

Рис. 23.1. Вид формы

4. Выделяя каждый из компонентов, находящихся на форме, с помощью *Object Inspector* установить для них следующие свойства:

```
Label1.Caption = 'Значение z ='
Label2.Caption = 'Значение x ='
Label3.Caption = 'Результат ='
Edit1.Text = ''
Edit2.Text = ''
Edit3.Text = ''
Button1.Caption = 'Счет'
```

5. Для решения задачи нужно записать обработчик события Button1.Click.

В окне **конструктора формы** следует два раза щелкнуть левой кнопкой мыши на кнопке Button1. В результате в окне **редактора кода** появится 'текст-заготовка' для процедуры Button1Click. Операторы процедуры необходимо записать между begin…end, а описание переменных-идентификаторов – в разделе описания переменных процедуры.

Полный текст процедуры для обработки события Button1.Click имеет вид

```
procedure TForm1.Button1Click(Sender: TObject);
var x,z,f:real;
begin
                              //считываются исходные данные
 z:=StrToFloat(Edit1.Text);//происходит преобразование
                              //переменных
 x:=StrToFloat(Edit2.Text);//строкового типа в
                              //вещественный

                  //вычисление арифметического выражения
 f:=z+z/(sqr(z)+1)-3.7e-8+exp(x+z);

 Edit3.Text:=FloatToStr(f); //результат преобразуется к
                              //переменной строкового типа
                              //и выводится в компопоненте
                              //Edit3 (свойство Text)
end;
```

В процедуре использовалась встроенная в язык *Object Pascal* функция **sqr** для возведения в квадрат значения *z*. Перечень основных встроенных процедур и функций приведен в прил. 4.

6. Запустить проект на компиляцию и выполнение.

7. Задать значения для $z = 3$ (в поле компонента Edit1), $x = 1$ (в поле компонента Edit2) и нажать кнопку **Счет**. Результат выполнения программы показан на рис. 23.2.

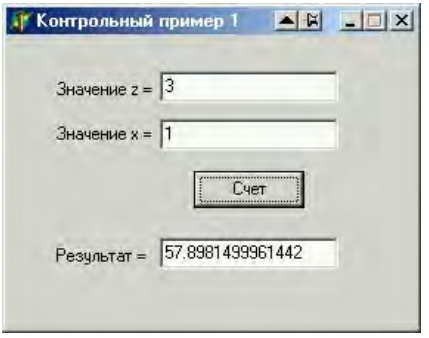

Рис. 23.2. Результат выполнения программы

Контрольный пример 2. Составить программу для расчета значения функции $f(x)$:

$$f(x) = \begin{cases} x + 3 \cdot x, & 1 \le x \le 5, \\ \cos(2 \cdot x + 3), & x < 1, \\ \sin^2(x + 8), & x > 5. \end{cases}$$

Если результат вычисления функции принимает отрицательное значение, предусмотреть возможность отображения абсолютного значения.

Решение

1. Открыть новый проект Delphi: **File** – **New Application**.

2. Установить с помощью *Object Inspector* следующие свойства компонента Form1:

```
Form1.Height = 284
Form1.Width = 276
Form1.BorderIcons
 biMaximize = false
Form1.BorderStyle = bsSingle
Form1.Position = poScreenCenter
Form1.Caption = 'Контрольный пример 2'
```

3. Расположить на форме два компонента Edit, два компонента Label, один компонент Button, один компонент CheckBox и один компонент RadioGroup. Установить для них с помощью *Object Inspector* следующие свойства:

```
Label1.Caption = 'Значение x ='
Label3.Caption = 'Результат ='
Edit1.Text = ''
Edit2.Text = ''
Button1.Caption = 'Счет'
CheckBox1.Width = 209
CheckBox1.Caption = 'Отображать абсолютное значение'
RadioGroup1.Caption = 'f(x)'
```

Для того чтобы задать количество и названия заголовков кнопок-переключателей в компоненте RadioGroup1, надо выделить находящийся на форме компонент RadioGroup1 и в *Object Inspector* для свойства Items нажать кнопку ... – появится окно **String List Editor**, в котором следует построчно ввести названия переключателей (рис. 23.3). Результат проектирования формы показан на рис. 23.4.

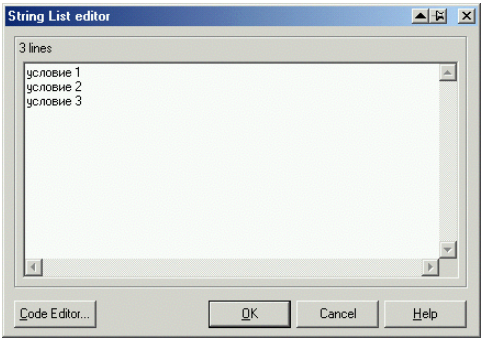

Рис. 23.3. Окно String List Editor

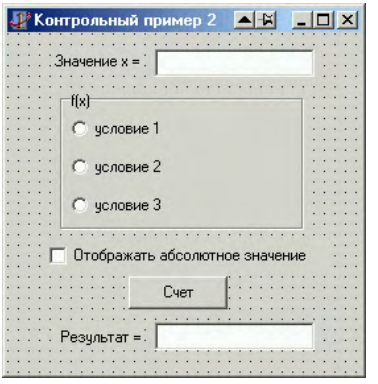

Рис. 23.4. Вид формы

4. Для решения задачи запишем обработчик событий Button1.Click (кнопка **Счет**), щелкнув на компоненте Button1 два раза левой кнопкой мыши. Текст соответствующей процедуры имеет вид

```
procedure TForm1.Button1Click(Sender: TObject);
var x,f:real;
```

```
begin
  x:=StrToFloat(Edit1.Text);
  case RadioGroup1.ItemIndex of
    0:f:=sqr(x)+3*x;
    1:f:=cos(2*x+3);
    2:f:=sqr(sin(x+8));
  end;
  if CheckBox1.Checked then f:=abs(f);
  Edit2.Text:=FloatToStr(f);
end;
```

При выполнении программы анализируется значение RadioGroup1. ItemIndex типа integer, в зависимости от которого идет выполнение программы. Для организации разветвления на языке *Object Pascal* использовался оператор **case**.

Вывод абсолютного значения происходит при анализе свойства Checked компонента CheckBox1 (оператор **if** языка *Object Pascal*).

5. Запустить проект на компиляцию и выполнение.

6. Задать значение x = 0,5, выбрать <условие 2> и нажать кнопку **Счет**. Результат отобразится в окне вывода (компонент Edit2). При изменении состояния переключателя CheckBox1 и последующем нажатии кнопки **Счет** в окне вывода отображается абсолютное значение результата.

Результат выполнения программы при различных состояниях переключателя CheckBox1 показан на рис. 23.5.

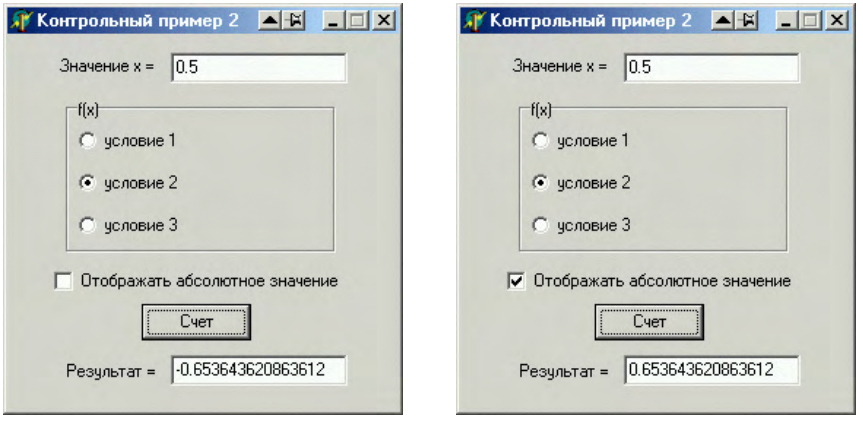

Рис. 23.5. Результат выполнения программы

23.3. Содержание отчета

Отчет должен содержать ответы на контрольные вопросы, тексты программ и результат выполнения соответствующего варианта.

23.4. Контрольные вопросы

1. Какие операторы используются для организации линейных и разветвляющихся алгоритмов и какова их структура?
2. Перечислите основные свойства компонентов Edit, CheckBox, Button и RadioGroup.
3. Укажите назначение кнопок в Delphi.
4. Какие компоненты используются в Delphi для организации разветвлений?

Варианты заданий

Вариант 1

Задание 1

Составить программу для расчета значения *f*:

$$f = \frac{3 \cdot 10^6 + 5}{2 \cdot 10^5 + 7} \cdot (a + b) \cdot \sin\sqrt{x}\,.$$

Значения *a*, *b* и *x* вводятся с клавиатуры.

Задание 2

Составить программу для расчета значения функции *f*(*x*):

$$f(x) = \begin{cases} e^x + 3 \cdot a \cdot b^3, & \text{если } x > 5, \\ b \cdot \mathrm{tg}\left(x^3 - a\right) - e^{-x}, & \text{если } x = 5, \\ \mathrm{tg}(8 \cdot x) + \sin\left(\sqrt{a - b \cdot x}\right), & \text{если } x < 5. \end{cases}$$

Значения *a*, *b* и *x* вводятся с клавиатуры.

Вариант 2

Задание 1
Составить программу для расчета значения *f*:

$$f = \frac{2 \cdot 10^{-5} \cdot \left(a^2 + b\right)}{1 + x^2 + \mathrm{tg}^2(x)}.$$

Значения *a*, *b* и *x* вводятся с клавиатуры.
Задание 2
Составить программу для расчета значения функции *f*(*x*):

$$f(x) = \begin{cases} a \cdot x^3, & \text{если } x < 0{,}2, \\ 16.5 + \cos(x), & \text{если } x = 0{,}2, \\ \sqrt{25 + x}, & \text{если } x > 0{,}2. \end{cases}$$

Значения *a*, *b* и *x* вводятся с клавиатуры.

Вариант 3

Задание 1
Составить программу для расчета значения *f*:

$$f = (a - b) \cdot \left(1 + \frac{1}{1 + 4 \cdot \left(a^2 + \ln(x)\right)}\right) + 13.7 \cdot 10^{-3}.$$

Значения *a*, *b* и *x* вводятся с клавиатуры.
Задание 2
Составить программу для расчета значения функции *f*(*x*):

$$f(x) = \begin{cases} \ln\left(x^2\right) + e^{a \cdot x + b} + \log|a\text{-}b|, & \text{если } 2 \cdot a\text{-}b < 0, \\ \mathrm{tg}\left(a \cdot x - b^2\right) - b \cdot |x|, & \text{если } 2 \cdot a\text{-}b = 0, \\ \mathrm{arctg}(2 \cdot x + 0.5) + \sqrt{a + b \cdot x}, & \text{если } 2 \cdot a\text{-}b > 0, \end{cases}$$

Значения *a*, *b* и *x* вводятся с клавиатуры.

Вариант 4

Задание 1

Составить программу для расчета значения *f*:

$$f = 2 \cdot x \cdot b \cdot \left(x^2 + 1\right) + \sqrt{\mathrm{tg}^2(a) + 0{,}7 \cdot 10^{-3}}\ .$$

Значения *a*, *b* и *x* вводятся с клавиатуры.

Задание 2

Составить программу для расчета значения функции *f*(*x*):

$$f(x) = \begin{cases} a \cdot \sqrt{\sin(x) + \cos^2(x)} + e^{a+\cdot b \cdot x}, & \text{если } \sqrt{b \cdot x^2 - 65} < a, \\ 1 - \ln\left(\sqrt{a \cdot x^2 - b} - \left|x^3\right|\right), & \text{если } \sqrt{b \cdot x^2 - 65} = a, \\ \left(x^3 - b\right) \cdot \cos(3 \cdot x\text{-}0{,}5), & \text{если } \sqrt{b \cdot x^2 - 65} > a. \end{cases}$$

Значения *a*, *b* и *x* вводятся с клавиатуры.

Вариант 5

Задание 1

Составить программу для расчета значения *f*:

$$f = \frac{2 \cdot x^2 + y^2}{1 + x^2 \cdot 10^{-4}} + x^{-4}.$$

Значения *x* и *y* вводятся с клавиатуры.

Задание 2

Составить программу для расчета значения функции *f*(*x*):

$$f(x) = \begin{cases} a^2 - e^x + b^3 + \cos(4 \cdot x - 0{,}2), & \text{если } \left|a^2 - b^2\right| < 10, \\ b \cdot \sin\left(x^3 - a\right) - e^{-x}, & \text{если } \left|a^2 - b^2\right| = 10, \\ \mathrm{ctg}(8 \cdot x) + \sqrt{a - b \cdot x}\ , & \text{если } \left|a^2 - b^2\right| > 10. \end{cases}$$

Значения *a*, *b* и *x* вводятся с клавиатуры.

Вариант 6

Задание 1
Составить программу для расчета значения *f*:

$$f = \frac{2 + 3 \cdot 10^{-15}}{1 + 14 \cdot x^2 + \sqrt[4]{|y|}} + \cos(2 \cdot x \cdot a).$$

Значения *x*, *y* и *a* вводятся с клавиатуры.
Задание 2
Составить программу для расчета значения функции *f*(*x*):

$$f(x) = \begin{cases} \ln(x^2) - e^{a \cdot x + b} + \sin(|a - b|), & \text{если } 2 \cdot a + b < 0, \\ \operatorname{tg}(a \cdot x - b^2) - b \cdot |x^{-3}|, & \text{если } 2 \cdot a + b = 0, \\ \operatorname{tg}(2 \cdot x - 1) + \sqrt{a + b \cdot x}, & \text{если } 2 \cdot a + b > 0. \end{cases}$$

Значения *a*, *b* и *x* вводятся с клавиатуры.

Вариант 7

Задание 1
Составить программу для расчета значения *f*:

$$f = |\cos(x) - \cos(y)| \cdot (1 + z + x^2 + y^2).$$

Значения *x*, *y* и *z* вводятся с клавиатуры.
Задание 2
Составить программу для расчета значения функции *f*(*x*):

$$f(x) = \begin{cases} a \cdot x^3 + b \cdot \ln|2 \cdot x|, & \text{если } \sqrt{a\text{-}b} < x, \\ \sqrt{a + \sin(2 \cdot x)} - e^{|3 \cdot x|}, & \text{если } \sqrt{a\text{-}b} = x, \\ \dfrac{\operatorname{arctg}(5 \cdot x)}{b \cdot \cos(x) + \ln(a \cdot x)}, & \text{если } \sqrt{a\text{-}b} > x. \end{cases}$$

Значения *a*, *b* и *x* вводятся с клавиатуры.

Лабораторная работа № 24

ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ. ИСПОЛЬЗОВАНИЕ ПОДПРОГРАММ

Цель работы: приобретение практических навыков программирования в Delphi циклических алгоритмов и организации подпрограмм.

Используемые программные средства: Delphi.

24.1. Теоретические сведения

Циклическими называются алгоритмы, в которых определенные серии команд повторяются некоторое число раз. Для организации циклов с заданным числом повторений используется оператор **for**. Если неизвестно, сколько раз необходимо повторить цикл, то используются операторы цикла с постусловием **repeat** или с предусловием **while** (см прил. 1).

Вопросы организации программ с использованием процедур и функций описаны в прил. 1.

Работа с компонентами

Для работы с многострочным текстом в Delphi имеются компоненты **Memo** типа TMemo  (панель **Standard**) и **RichEdit** типа TRichEdit  (панель **Win32**), которые относятся к многострочным редакторам. Многострочный редактор предоставляет возможности для ввода, редактирования и отображения информации и позволяет содержать несколько строк.

Для работы с отдельными строками используется свойство `Lines` типа `TStrings`. Для того чтобы изменить значение свойства `Lines` компонента `Memo` в режиме проектирования приложения, используется **String List Editor**, который вызывается с помощью *Object Inspector* (рис. 24.1).

Для добавления новой строки во время выполнения приложения необходимо вызвать метод `Add` (переменная типа `string`) компонента `Memo`:

```
Memo1.Lines.Add('новая строка');
```

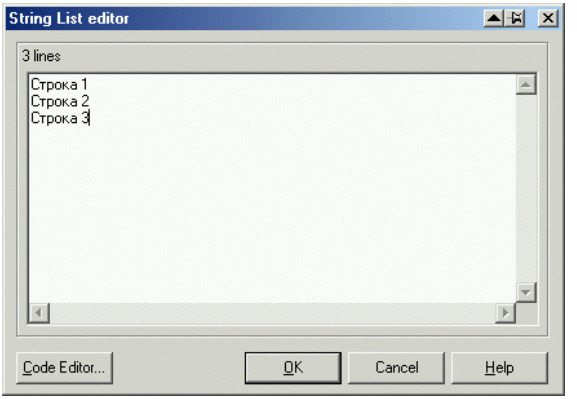

Рис. 24.1. Окно String List Editor

Для выравнивания текста в поле компонента `Memo` используется свойство `Alignment,` которое может принимать значения:

`taCenter` – выровнять по центру;

`taLeftJustify` – выровнять по левому краю;

`taRightJustify` – выровнять по правому краю.

Для просмотра всей информации в компоненте Memo используются свойства `WordWrap` (перенос текста) и `ScrollBars` (полосы прокрутки), задаваемые при помощи окна *Object Inspector* во время разработки приложения или при обращении к этим свойствам компонентов непосредственно во время работы приложения.

Для очистки содержимого компонента Memo используется метод `Clear`:

```
Memo1.Clear.
```

Содержимое компонентов можно загружать из текстового файла и сохранять в нем. Для этого удобно использовать методы

```
Memo1.Lines.LoadFromFile(const FileName:string)
```

и

```
Memo1.Lines.SaveToFile(const FileName:string)
```

класса `TStrings.`

24.2. Порядок выполнения работы

Изучить: операторы, используемые для организации циклов; компонент Memo и его свойства, выполнить контрольные примеры и задания соответствующего варианта.

Контрольный пример 1. Составить программу для расчета $f(x)$. Значения N и x вводятся с клавиатуры. Расчет факториала оформить в виде процедуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h:

$$f(x) = \sum_{n=1}^{N} \frac{x^n}{(n+3)!}.$$

Решение

1. Открыть новый проект Delphi: **File** – **New Application**.

2. Установить с помощью *Object Inspector* следующие свойства компонента Form1:

```
Form1.Height = 302
Form1.Width = 326
Form1.BorderIcons
    biMaximize = false
Form1.BorderStyle = bsSingle
Form1.Position = poScreenCenter
Form1.Caption = 'Контрольный пример 1'
```

3. Расположить на форме следующие компоненты: четыре компонента Edit, четыре компонента Label, один компонент Button и один компонент Memo. Установить для них с помощью *Object Inspector* следующие свойства:

```
Label1.Caption = 'N'
Label2.Caption = 'x1'
Label3.Caption = 'x2'
Label3.Caption = 'h'
Edit1.Text = ''
Edit2.Text = ''
Edit3.Text = ''
Edit4.Text = ''
Button1.Caption = 'Счет'
Memo1.Lines = ''
Memo1.ScrollBars = ssVertical
```

Результат показан на рис. 24.2.

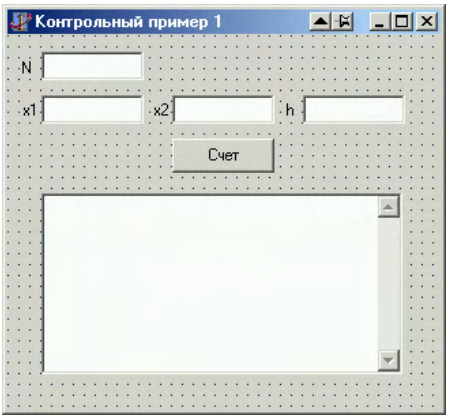

Рис. 24.2. Вид формы

4. Для решения задачи запишем обработчик событий Button1.Click, щелкнув на компоненте Button1 (кнопка **Счет**) два раза левой кнопкой мыши.

Текст соответствующей процедуры имеет вид:

```
procedure TForm1.Button1Click(Sender: TObject);
var x1,x2,h:real;
   N:integer;
   i:integer;
   sum:real;
   ZnFact:integer;
   stroka:string;
begin
  Memo1.Clear;
  N:=StrToInt(Edit1.Text);
  x1:=StrToFloat(Edit2.Text);
  x2:=StrToFloat(Edit3.Text);
  h:=StrToFloat(Edit4.Text);

  repeat
   sum:=0;
   for i:=1 to n do
   begin
      CalkFact(i+3,ZnFact);
       sum:=sum+exp(i*ln(x1))/ZnFact;
   end;
```

```
    stroka:='f('+FloatToStr(x1)+') = '+
            FloatToStrF(sum,ffFixed,7,3);
    Memo1.Lines.Add(stroka);
    x1:=x1+h;
   until x1>x2;
end;
```

Переменные, которые необходимы для решения задачи, описаны в соответствующем разделе процедуры `TForm1.Button1Click`.

Для расчета функции вида x^n использовалось соотношение $e^{(n\cdot\ln(x))}$.

Для расчета факториала использовалась процедура `CalkFact`, которая должна располагаться в тексте модуля выше, чем происходит обращение к ней:

```
procedure CalkFact(Zn:integer; var Rez:integer);
var j: Integer;
begin
  Rez:=1;
  for j := 1 to Zn do Rez:=Rez*j;
end;
```

5. Запустить проект на компиляцию и выполнение.

6. Задать значения для N = 10, $x1$ = 1, $x2$ = 10, h = 0,5 и нажать кнопку **Счет**. Результат выполнения программы показан на рис. 24.3.

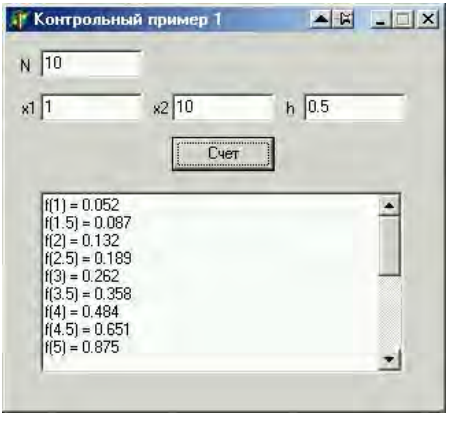

Рис. 24.3. Результат выполнения программы

Контрольный пример 2. Составить программу поиска минимального и максимального значения функции на отрезке [*a*, *b*]. Значения *a*, *b* и шага *h* вводятся с клавиатуры. Расчет функции оформить как подпрограмму-функцию:

$$f(x) = 5 \cdot x^2 - 7 \cdot x + 10 .$$

Решение

1. Открыть новый проект Delphi: **File** – **New Application**.

2. Установить с помощью *Object Inspector* следующие свойства компонента Form1:

```
Form1.Height = 288
Form1.Width = 324
Form1.BorderIcons
     biMaximize = false
Form1.BorderStyle = bsSingle
Form1.Position = poScreenCenter
Form1.Caption = 'Контрольный пример 2'
```

3. Расположить на форме следующие компоненты: три компонента Edit, три компонента Label, один компонент Button и один компонент Memo. Установить для них с помощью *Object Inspector* следующие свойства:

```
Label1.Caption = 'a'
Label2.Caption = 'b'
Label3.Caption = 'h'
Edit1.Text = ''
Edit2.Text = ''
Edit3.Text = ''
Button1.Caption = 'Счет'
Memo1.Lines = ''
Memo1.ScrollBars = ssNone
```

Результат показан на рис. 24.4.

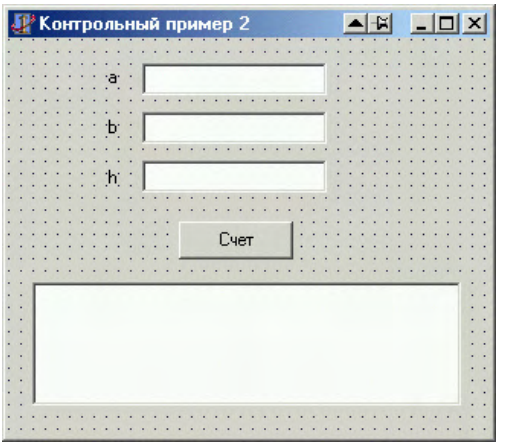

Рис. 24.4. Вид формы

4. Для решения задачи запишем обработчик событий Button1.Click, щелкнув на компоненте Button1 (кнопка **Счет**) два раза левой кнопкой мыши. Текст соответствующей процедуры имеет вид

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,h:real;
    MinX,MinY,MaxX,MaxY:real;
begin
  Memo1.Clear;
  a:=StrToFloat(Edit1.Text);
  b:=StrToFloat(Edit2.Text);
  h:=StrToFloat(Edit3.Text);

  MinY:=1e38;
  MaxY:=-1e38;

  while a<b do
  begin
    if ZnFun(a)<MinY then
    begin
      MinY:=ZnFun(a);
      MinX:=a;
    end;
```

```
    if ZnFun(a)>MaxY then
    begin
      MaxY:=ZnFun(a);
      MaxX:=a;
    end;
    a:=a+h;
  end;
  Memo1.Lines.Add('Миним. знач. y при x = '+
                  FloatToStrF(MinX,ffFixed,10,5)+
                  'y = '+FloatToStrF(MinY,ffFixed,10,5));
  Memo1.Lines.Add('Макс. знач. y при x = '+
                  FloatToStrF(MaxX,ffFixed,10,5)+
                  'y = '+FloatToStrF(MaxY,ffFixed,10,5));
end;
```

Переменные, которые необходимы для решения задачи, описаны в соответствующем разделе процедуры `TForm1.Button1Click`.

Для расчета значений функции использовалась подпрограмма-функция `ZnFun`, которая должна располагаться в тексте модуля ранее, чем происходит обращение к ней:

```
function ZnFun(x:real):real;
begin
  ZnFun:=5*sqr(x)-7*x+100;
end;
```

5. Запустить проект на компиляцию и выполнение.

6. Задать значения для a = -1, b = 2, h = 0,01 и нажать кнопку **Счет**. Результат выполнения программы показан на рис. 24.5.

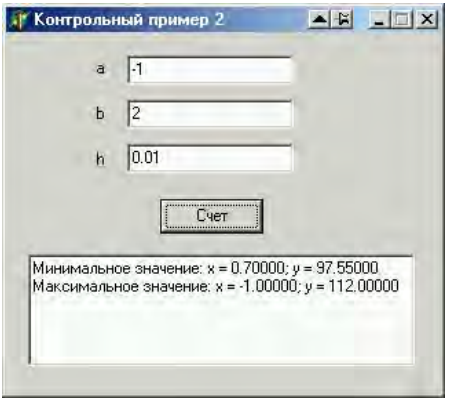

Рис. 24.5. Результат выполнения программы

24.3. Содержание отчета

Отчет должен содержать ответы на контрольные вопросы, тексты программ и результат выполнения задания соответствующего варианта.

24.4. Контрольные вопросы

1. Какие операторы используются для организации циклических алгоритмов и какова их структура?
2. Перечислить основные свойства компонента Memo.
3. Как получить доступ к заданной строке компонента Memo?
4. Что такое многострочный редактор?

Варианты заданий

Вариант 1

Задание 1. Составить программу для расчета $f(x)$. Значение N и x вводятся с клавиатуры. Расчет факториала оформить в виде процедуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h:

$$f(x) = \sum_{n=1}^{N} \frac{(2 \cdot x)^n}{(n+5)!}.$$

Задание 2. Вычислить экстремум функции при следующих условиях:

Вид функции $f(x)$	Вид экстремума	Диапазон изменения аргумента
$2 + x - x^2$	max	[0; 1]

Вывести на печать значение экстремума и значение аргумента, при котором он достигается.

Вариант 2

Задание 1. Составить программу для расчета $f(x)$. Значение N и x вводятся с клавиатуры. Расчет факториала оформить в виде процеду-

ры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h:

$$f(x) = \sum_{n=1}^{N} \frac{2\sqrt{n}x^n}{3(n+8)!}.$$

Задание 2. Вычислить экстремум функции при следующих условиях:

Вид функции $f(x)$	Вид экстремума	Диапазон изменения аргумента
$(1 - x)^4$	min	[0,2; 1,5]

Вывести на печать значение экстремума и значение аргумента, при котором он достигается.

Вариант 3

Задание 1. Составить программу для расчета $f(x)$. Значение N и x вводятся с клавиатуры. Расчет факториала оформить в виде процедуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h:

$$f(x) = \sum_{n=1}^{N} \frac{25e^{nx}}{10!+2n}.$$

Задание 2. Вычислить экстремум функции при следующих условиях.

Вид функции $f(x)$	Вид экстремума	Диапазон изменения аргумента
$2x^3 - 3x^2$	max	[-1; 2]

Вывести на печать значение экстремума и значение аргумента, при котором он достигается.

Вариант 4

Задание 1. Составить программу для расчета $f(x)$. Значение N и x вводятся с клавиатуры. Расчет факториала оформить в виде процеду-

ры. Предусмотреть возможность вывода значений функции в зависимости от *x*, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом *h*:

$$f(x) = \sum_{n=1}^{N} \frac{\cos(2\pi x)}{2^n + \sin(n!)}.$$

Задание 2. Вычислить экстремум функции при следующих условиях:

Вид функции $f(x)$	Вид экстремума	Диапазон изменения аргумента
$x^{1/3}(1 - x)^{2/3}$	max	[-0,2; 1,5]

Вывести на печать значение экстремума и значение аргумента, при котором он достигается.

Вариант 5

Задание 1. Составить программу для расчета $f(x)$. Значение *N* и *x* вводятся с клавиатуры. Расчет факториала оформить в виде процедуры. Предусмотреть возможность вывода значений функции в зависимости от *x*, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом *h*:

$$f(x) = \sum_{n=1}^{N} \frac{\sin^2(n)}{x^n + n!}.$$

Задание 2. Вычислить экстремум функции при следующих условиях:

Вид функции $f(x)$	Вид экстремума	Диапазон изменения аргумента
$x^3 - 6x^2 + 9x + 4$	max	[-0,2; 0,8]

Вывести на печать значение экстремума и значение аргумента, при котором он достигается.

Вариант 6

Задание 1. Составить программу для расчета $f(x)$. Значение *N* и *x* вводятся с клавиатуры. Расчет факториала оформить в виде процеду-

ры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h:

$$f(x) = \sum_{n=1}^{N} \frac{n!}{n!+1} \left(\frac{x}{2} \right)^n .$$

Задание 2. Вычислить экстремум функции при следующих условиях:

Вид функции $f(x)$	Вид экстремума	Диапазон изменения аргумента
$x^3 - 6x^2 + 9x + 4$	min	[2; 4]

Вывести на печать значение экстремума и значение аргумента, при котором он достигается.

Вариант 7

Задание 1. Составить программу для расчета $f(x)$. Значение N и x вводятся с клавиатуры. Расчет факториала оформить в виде процедуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h:

$$f(x) = \sum_{n=1}^{N} \frac{2n \ln(1 + n!)}{n!+x}.$$

Задание 2. Вычислить экстремум функции при следующих условиях:

Вид функции $f(x)$	Вид экстремума	Диапазон изменения аргумента
$5x^3 + 6x^2 - 6x - 7$	min	[2; 4]

Вывести на печать значение экстремума и значение аргумента, при котором он достигается.

Лабораторная работа № 25

ИСПОЛЬЗОВАНИЕ ВИЗУАЛЬНЫХ КОМПОНЕНТОВ ДЛЯ ПРОГРАММИРОВАНИЯ МАССИВОВ

Цель работы: приобретение практических навыков программирования массивов.

Используемые программные средства: Delphi.

25.1. Теоретические сведения. Работа с компонентами

Массивом называется упорядоченная индексированная совокупность однотипных элементов, имеющих общее имя. Элементами массива могут быть данные различных типов. Каждый элемент массива одновременно определяется *именем* массива и *индексом* (номер этого элемента в массиве) или *индексами*, если массив многомерный. Количеством индексных позиций определяется размерность массива (одномерный, двумерный и т.д.) (см. прил. 1).

Работу с массивами данных удобно организовывать в виде таблиц. Для этой цели используются компоненты **StringGrid** типа TStringGrid и **DrawGrid** типа TDrawGrid, отображающие информацию в виде двумерных таблиц. Компоненты расположены на панели **Additional** *Палитры компонентов* и имеют пиктограммы  (StringGrid) и  (DrawGrid).

Компонент **StringGrid** применяется для работы с текстовой информацией, причем ее отображение и хранение производится компонентом автоматически. Компонент **DrawGrid** позволяет отображать любую (текстовую и графическую) информацию, однако вся работа по визуальному отображению объектов возлагается на разработчиков программы. В дальнейшем будем рассматривать компонент StringGrid.

Основным элементом таблицы является ячейка, для доступа к которой используется свойство

```
Cells[ACol,ARow:Integer]:string,
```

где `ACol` и `ARow` – индексы элемента двумерного массива. Индекс `ACol` определяет номер столбца, а индекс `ARow` – номер строки. Свойство `Cells` доступно только во время выполнения программы. Нумерация элементов таблицы начинается с нуля.

Некоторые свойства для работы с компонентом StringGrid приведены в табл. 25.1.

Таблица 25.1

Свойства компонента StringGrid	Описание свойств
`ColCount`	число столбцов в таблице
`RowCount`	число строк в таблице
`FixedCols`	число фиксированных столбцов
`FixedRows`	число фиксированных строк
`ColWidths`	ширина столбца в пикселах
`RowHeights`	высота строки в пикселах
`DefaultColWidth`	значение ширины столбца по умолчанию
`DefaultRowHeight`	значение высоты строк по умолчанию
`BorderStyle`	определяет наличие или отсутствие внешней рамки таблицы

`ScrollBars`
`ssNone`
`ssHorizontal`
`ssVertical`
`ssBoth`
	параметры отображения полосы прокрутки:
	полоса прокрутки не допускается
	допускается горизонтальная полоса прокрутки
	допускается вертикальная полоса прокрутки
	допускаются обе полосы прокрутки

`Options`
`goVertLine`
`goHorzLine`
`goEditing`
	опции для доступа к параметрам таблицы для их настройки:
	отображение в сетке вертикальных разделительных линий
	отображение в сетке горизонтальных разделительных линий
	возможность редактирования содержания ячейки с клавиатуры

В качестве фиксированных элементов задаются крайние левые столбцы и верхние строки, что используется при оформлении заголовков. В них запрещен ввод значений с клавиатуры.

Для визуального выделения функционально связанных компонентов или в качестве средств визуального группирования используются компоненты **GroupBox**  или **Panel**  . Компоненты расположены на панели **Standard**. Компонент **GroupBox** представляет

собой прямоугольную рамку с заголовком (свойство `Caption`) в левом верхнем углу и служит только для объединения содержащихся в нем элементов. Компонент **Panel** предназначен для объединения произвольных элементов управления и возможного их перемещения по форме и для стыковки с другими панелями. Кроме того, компонент **Panel** может использоваться для создания рамок разнообразного вида (свойства `BevelInner`, `BevelOuter`, `BevelWidth` компонента).

25.2. Порядок выполнения работы

Изучить компонент StringGrid и его свойства, выполнить контрольный пример и задания соответствующего варианта.

Контрольный пример. Для заданной матрицы **A**(a_{ij}) сформировать матрицу **B**(b_{ij}) так, чтобы

$$b_{ij} = \frac{a_{ij}^2}{a_{ij} + P \cdot a_{ij}^2 + Q}.$$

Вычислить сумму элементов нечетных строк матрицы **B**. Матрица **A** заполняется случайными значениями в диапазоне от 0 до 10. Предусмотреть ввод параметров *P* и *Q*.

Решение

1. Открыть новый проект Delphi: **File** – **New Application**.

2. Установить с помощью *Object Inspector* следующие свойства компонента Form1:

```
Form1.Height = 316
Form1.Width = 586
Form1.BorderIcons
     biMaximize = false
Form1.BorderStyle = bsSingle
```

3. Расположить на форме следующие компоненты: два компонента GroupBox, пять компонентов Edit, пять компонентов Label, два компонента StringGrid и два компонента Button. Установить для них с помощью *Object Inspector* следующие свойства:

```
GroupBox1.Align = alLeft
GroupBox1.Width = 289
GroupBox1.Caption = 'Исходные данные'
Label1.Caption = 'К-во строк:'
Label2.Caption = 'К-во столбцов:'
Edit1.Width = 60
Edit2.Width = 60
Button1.Caption = 'Задать'
StringGrid1.Left = 10
StringGrid1.Top = 71
StringGrid1.Height = 177
StringGrid1.Width = 271
StringGrid1.ScrollBars = ssBoth
Label3.Caption = 'P='
Label4.Caption = 'Q='
Edit3.Width = 60
Edit4.Width = 60
Button2.Caption = 'Вычислить'

GroupBox2.Align = alClient
GroupBox2.Caption = 'Результат'
Label5.Caption = 'Сумма элементов='
StringGrid2.Left = 10
StringGrid2.Top = 71
StringGrid2.ScrollBars = ssBoth
```

Результат показан на рис. 25.1.

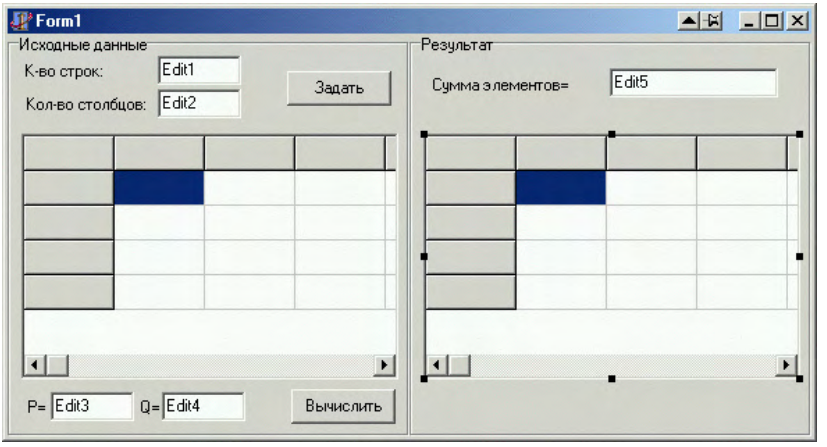

Рис. 25.1. Вид формы

При запуске программы в момент создания формы возникает событие OnCreate (создать форму). Обработка этого события используется для настройки компонентов формы, для задания начальных значений и т.д.

Создадим обработчик этого события, во время выполнения которого будут установлены некоторые начальные значения компонентов. Текст соответствующей процедуры имеет вид

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Form1.Caption:='Работа с массивами';
  Edit1.Text:='';
  Edit2.Text:='';
  Edit3.Text:='';
  Edit4.Text:='';
  Edit5.Text:='';

  StringGrid1.Cells[0,0]:='A';
  StringGrid2.Cells[0,0]:='B';

  StringGrid2.Height:=StringGrid1.Height;
  StringGrid2.Width:=StringGrid1.Width;
end;
```

4. Запишем обработчик события `Button1.Click` (кнопка **Задать**), в процессе выполнения которого устанавливается размерность массива, число столбцов и число строк в компоненте StringGrid1, формируется матрица **A** со случайными значениями в диапазоне от 0 до 10 и происходит заполнение ячеек компонента StringGrid1 элементами матрицы **A**.

В разделе описания глобальных переменных должны быть описаны следующие константы, типы и переменные:

```
const MaxMN=10; //максимальная размерность массива
      K=10;     //диапазон генерации случайного числа

Type Mas=array [1..MaxMN,1..MaxMN] of real;
var
  Form1: TForm1;
  M,N:integer;   //количество строк и столбцов в матрицах
  i,j:integer;
  A,B:Mas;       //массивы
  P,Q:real;      //переменные для параметров P и Q
  Sum:real;
```

Текст процедуры `TForm1.Button1Click` имеет вид

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  M:=StrToInt(Edit1.Text);
  N:=StrToInt(Edit2.Text);

  StringGrid1.RowCount:=M+StringGrid1.FixedRows;
  StringGrid1.ColCount:=N+StringGrid1.FixedCols;
  for i:=1 to M do StringGrid1.Cells[0,i]:='i='+IntToStr(i);
  for j:=1 to N do StringGrid1.Cells[j,0]:='j='+IntToStr(j);

  Randomize;
  for i:=1 to M do
  for j:=1 to N do
  begin
    A[i,j]:=random(K);
    StringGrid1.Cells[j,i]:=FloatToStr(A[i,j]);
  end;
end;
```

Процедура Randomize используется для смены базы генерации случайных чисел.

5. Для формирования матрицы B и вычисления суммы элементов нечетных строк в этой матрице необходимо записать процедуру, являющуюся обработчиком события `Button2.Click` (кнопка **Вычислить**). Текст процедуры приведен ниже:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  P:=StrToFloat(Edit3.Text);
  Q:=StrToFloat(Edit4.Text);

  StringGrid2.RowCount:=StringGrid1.RowCount;
  StringGrid2.ColCount:=StringGrid1.ColCount;
  for i:=1 to M do StringGrid2.Cells[0,i]:='i='+IntToStr(i);
  for j:=1 to N do StringGrid2.Cells[j,0]:='j='+IntToStr(j);

  for i:=1 to M do
  for j:=1 to N do
  begin
    B[i,j]:=sqr(A[i,j])/(A[i,j]+P*sqr(A[i,j])+Q);
   StringGrid2.Cells[j,i]:=FloatToStrF(B[i,j],ffFixed,5,3);
  end;
```

```
  Sum:=0;
  for i:=1 to M do
  for j:=1 to N do
  begin
    if odd(i) then Sum:=Sum+B[i,j];
  end;

  Edit5.Text:=FloatToStr(Sum);
end;
```

6. Запустить проект на выполнение. Результат выполнения программы показан на рис. 25.2.

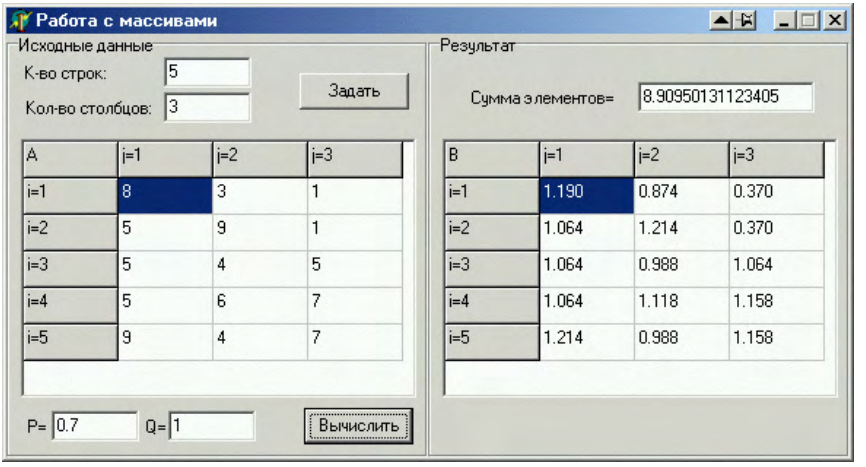

Рис. 25.2. Результат выполнения программы

25.3. Содержание отчета

Отчет должен содержать ответы на контрольные вопросы, тексты программ и результат выполнения задания соответствующего варианта.

25.4. Контрольные вопросы

1. Приведите определение массива.
2. Какие данные могут быть элементами массива?
3. С какой целью применяются компоненты StringGrid и DrawGrid?
4. Назовите основные свойства компонентов StringGrid и DrawGrid.

5. В каких разделах программы и модуля описываются локальные и глобальные переменные?

Варианты заданий

Вариант 1

Написать программу, вычисляющую значение матрицы

$$\mathbf{C} = \mathbf{A} + k \cdot \mathbf{B}.$$

Найти произведение всех элементов матрицы **C** (для вывода значения использовать компонент Edit). Предусмотреть ввод параметра *k*.

Вариант 2

Для заданной матрицы $\mathbf{A}(a_{ij})$ сформировать матрицу $\mathbf{B}(b_{ij})$ так, чтобы

$$b_{ij} = a_{ij}^2 + a_{ij} \cdot K .$$

Вычислить сумму элементов нечетных строк матрицы **B**. Предусмотреть ввод параметра *K*.

$$A = \begin{pmatrix} 2 & 6 & 0 \\ 5 & 7 & 4 \\ 3 & -5 & 1 \end{pmatrix}.$$

Вариант 3

Написать программу, вычисляющую произведение двух матриц. Значения исходных матриц заполняется случайными значениями в диапазоне от 0 до 100.

Вариант 4

Для заданной матрицы **A** посчитать количество элементов, меньших нуля, равных нулю и больших нуля.

$$A = \begin{pmatrix} -4 & 4.6 & 2 & -4 \\ 0 & 7 & 9 & -5.5 \\ 3 & 0 & 11 & 7 \\ 1 & 6 & 2 & 8 \end{pmatrix}.$$

Для вывода результатов использовать компонент Edit.

Вариант 5

Для заданной матрицы **A** [5×5] найти минимальный элемент (и его номер) и максимальный элемент (и его номер). Для вывода результатов использовать компонент Edit.

Вариант 6

Для заданной матрицы **A** сформировать матрицу **B** так, чтобы

$$b_{ij} = \begin{cases} 10 \cdot a_{ij}, & \text{если } a_{ij} \geq 10; \\ 0, & \text{если } a_{ij} < 10. \end{cases}$$

Значения матрицы A вводятся с клавиатуры.

Вариант 7

Найти матрицу, транспонированную относительно исходной. Предусмотреть ввод исходной матрицы различной размерности. Если исходная матрица квадратная (для размерности не более 3), то вычислить ее определитель. В противоположном случае выдать сообщение 'Матрица не является квадратной или ее размерность больше 3'.

Лабораторная работа № 26

ПОСТРОЕНИЕ ДВУМЕРНЫХ ГРАФИКОВ В DELPHI

Цель работы: изучение возможностей графического отображения информации с помощью компонента Chart.

Используемые программные средства: Delphi.

26.1. Теоретические сведения. Работа с компонентами

Компонент-диаграмма **Chart** типа TChart предназначен для работы с графиками и диаграммами различных типов и служит для графического представления результатов. Компонент находится на панели **Additional** *Палитры компонентов* и имеет пиктограмму .

Компонент содержит большое количество разнообразных свойств, многие из которых являются объектами и имеют свои свойства. Установка значений этих свойств выполняется с помощью редактора **Editing Chart** (рис. 26.1) во время разработки программы (приложения) либо при обращении к свойствам компонента во время ее выполнения. Всю работу по отображению графиков, построению и разметке координатных осей, сетки, подписей берет на себя компонент Chart. Разработчику программы требуется задать тип диаграммы и источник данных.

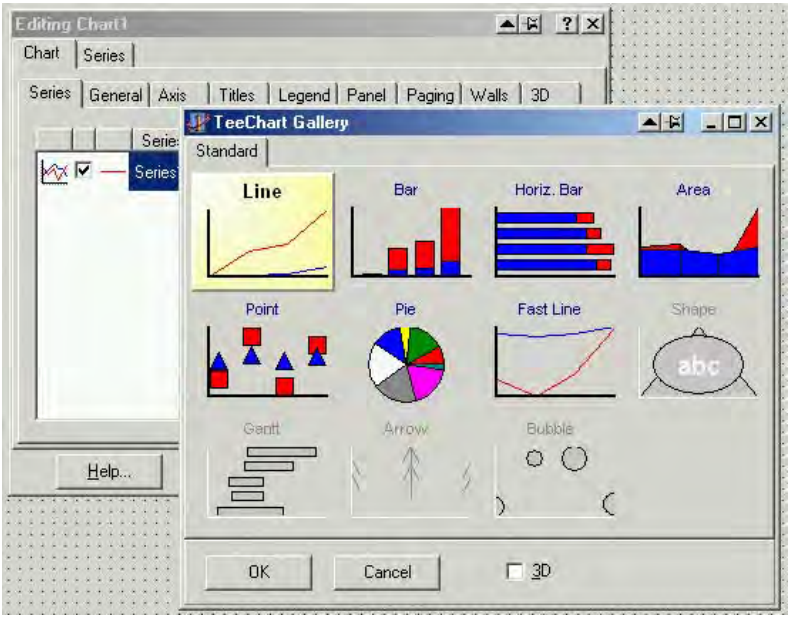

Рис. 26.1. Окно редактора Editing Chart

Для представления данных, заданных таблично или с использованием функции в виде линии используется переменная **Series1** ти-

па **TLineSeries**, которая описывает последовательность значений, отображающихся на диаграмме.

Добавление новой точки к серии выполняется с помощью метода **Add**:

```
function AddXY(Const AXValue,AYValue:Double;
               Const AXLabel:String;
               AColor:TColor),
```

где `AXValue`, `AYValue` – параметры, определяющие координаты точки по осям *OX* и *OY*, `AXLabel` – необязательный параметр; `AColor` – цвет группы, к которой принадлежит точка.

Связь между диаграммой и программным кодом происходит следующим образом. При создании каждой серии данных с помощью редактора Editing Chart в разделе TForm1 появляется новая переменная Series<n> (где <n> – номер серии) соответствующего типа. Например, для отображения серии данных в виде точек переменная Series1 будет иметь тип TPointSeries (точечное представление). Некоторые свойства компонента Chart приведены в табл. 26.1.

Таблица 26.1

Свойства компонента Chart	Описание свойств
`Title.Text`	задание заголовка диаграммы
`Title.Aligment`	выравнивание заголовка
`<NameAxis>.Automatic`	автоматическое определение параметров по оси
`<NameAxis>.Minimum`	задание минимального значения по оси
`<NameAxis>.Maximum`	задание максимального значения по оси
`<NameAxis>.Increment`	задание шага разметки по оси
Под `<NameAxis>` понимается нижняя (`BottomAxis`), левая (`LeftAxis`), правая (`RightAxis`) или верхняя (`TopAxis`) координатная ось	

26.2. Порядок выполнения работы

Изучить компонент Chart и его свойства, выполнить контрольные примеры и задания соответствующего варианта.

Контрольный пример 1. Составить программу, отображающую графики функций $f_1(x) = \sin(x)$ и $f_2(x) = \cos(x)$ в интервале $[a, b]$ с заданным шагом h.

Решение

1. Открыть новый проект Delphi: **File** – **New Application**.

2. Расположить на форме следующие компоненты: три компонента Edit, три компонента Label, компонент Chart и компонент Button. Установить для них следующие свойства:

```
Label1.Caption = 'a'
Label2.Caption = 'b'
Label3.Caption = 'h'
Edit1.Text = ''
Edit2.Text = ''
Edit3.Text = ''
Button1.Caption = 'Построить'
```

Результат показан на рис. 26.2.

Рис. 26.2. Вид формы

Для изменения параметров компонента Chart необходимо два раза щелкнуть на нем левой кнопкой мыши (или один раз правой кнопкой и в контекстном меню выбрать пункт **Edit Chart**). В открывшемся окне редактирования **Editing Chart 1** создать два объекта: **Series1** и **Series2**, щелкнув на кнопке **Add**, находящейся на вкладке **Series**. В качестве типа графика выбрать **Line**, отключив трехмерное представление с помощью переключателя **3D**. Для изменения имен серий (на **f1** и **f2**) используется кнопка **Title**. Редакти-

рование завершается нажатием кнопки **Close**. Первоначально на графиках отображаются случайные значения.

3. Для решения задачи запишем обработчик событий Button1.Click, щелкнув на компоненте Button1 (кнопка **Построить**) два раза левой кнопкой мыши. Текст соответствующей процедуры имеет вид

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,h:double;
var x,f1,f2:double;
begin
                    //удаление всех значений в ряду данных
  Series1.Clear;
  Series2.Clear;
                    //задание значений границ и шага
  a:=StrToFloat(Edit1.Text);
  b:=StrToFloat(Edit2.Text);
  h:=StrToFloat(Edit3.Text);
                    //расчет значений функций
  x:=a;
  repeat
    f1:=sin(x);
    Series1.AddXY(x,f1,'',clRed);
    f2:=cos(x);
    Series2.AddXY(x,f2,'',clBlue);
    x:=x+h;
  until x>b;
                    //задание названия диаграммы
  Chart1.Title.Text.Clear;
  Chart1.Title.Text.Add('Графики функций f1 и f2. Шаг =
                        '+FloatToStr(h));
                    //установка параметров нижней оси
  Chart1.BottomAxis.Automatic:=false;
  Chart1.BottomAxis.Maximum:=b;
  Chart1.BottomAxis.Minimum:=a;
  Chart1.BottomAxis.Increment:=(Chart1.BottomAxis.Maximum
                             -Chart1.BottomAxis.Minimum)/2;
                    //установка параметров левой оси
  Chart1.LeftAxis.Automatic:=false;
  Chart1.LeftAxis.Minimum:=-1;
  Chart1.LeftAxis.Maximum:=1;
  Chart1.LeftAxis.Increment:=0.5;
end;
```

4. Запустить проект на компиляцию и выполнение.

5. Задать значения $a = 0$, $b = 6.28$, $h = 0{,}1$ и нажать кнопку **Построить**. График зависимостей будет иметь вид, как на рис. 26.3.

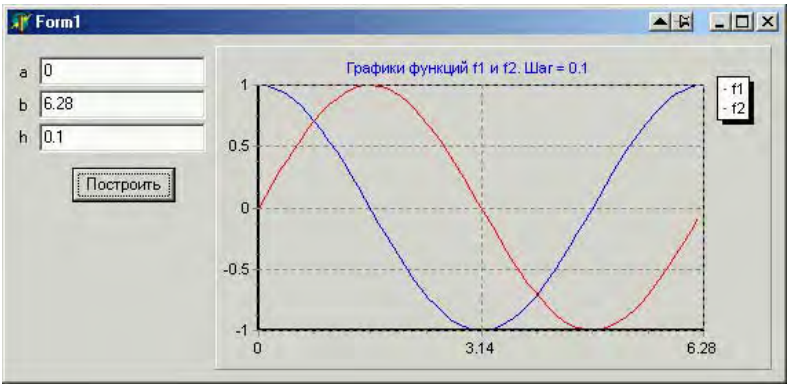

Рис. 26.3. Результат выполнения программы

Контрольный пример 2. Построить график функции, заданной таблично. Для ввода значений использовать компонент StringGrid.

Решение

1. Открыть новый проект Delphi: **File** – **New Application**.

2. Расположить на форме следующие компоненты: Edit, Label, Chart, CheckBox, StringGrid и два компонента Button в соответствии с рис. 26.4:

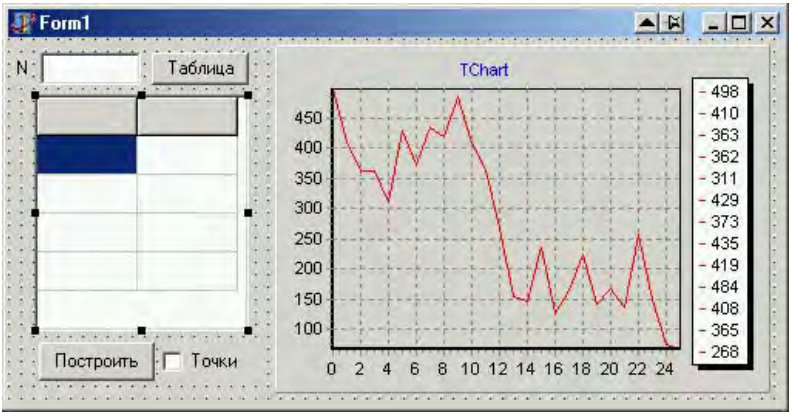

Рис. 26.4. Вид формы

3. Установить в *Object Inspector* следующие свойства компонентов:

```
Label1.Caption = 'N'
Button1.Caption = 'Таблица'
Button2.Caption = 'Построить'
Edit1.Text = ''
StringGrid1.Options
      goEditing = true
StringGrid1.ColCount = 2
StringGrid1.FixedCols = 0
CheckBox1.Caption = 'Точки'
```

Для компонента Chart, используя **Editing Chart 1,** создать объект **Series1**, выбрав в качестве типа графика **Line**.

4. Записать обработчики событий `Button1.Click` (кнопка **Таблица**) и `Button2.Click` (кнопка **Построить**), текст которых приведен ниже:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  StringGrid1.RowCount:=StrToInt(Edit1.Text)+1;
  StringGrid1.Cells[0,0]:='x';
  StringGrid1.Cells[1,0]:='y';
end;

procedure TForm1.Button2Click(Sender: TObject);
var i:longint;
begin
  Series1.Clear;
  for i:=1 to StringGrid1.RowCount-1 do
      Series1.AddXY(StrToFloat(StringGrid1.Cells[0,i]),
                    StrToFloat(StringGrid1.Cells[1,i]),
                    '',clGreen);

  Chart1.Title.Text.Clear;
end;
```

5. Для отображения точек на графике используем метод Visible компонента типа TSeriesPointer, входящего в состав компонента Chart. Обработчик соответствующего события имеет вид

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  if CheckBox1.Checked=true then Series1.Pointer.Visible:=true
  else Series1.Pointer.Visible:=false;
end;
```

6. Запустить проект на компиляцию и выполнение.

7. Задать значение для N = 6, нажать кнопку **Таблица** и заполнить таблицу следующими значениями:

x	0	3	4	7	10	12
y	0	14	-4	10	12	7

8. После нажатия на кнопку **Построить** отобразится графическая зависимость исходных данных. При изменении состояния переключателя **Точки** график имеет вид "линия с точкой", как показано на рис. 26.5.

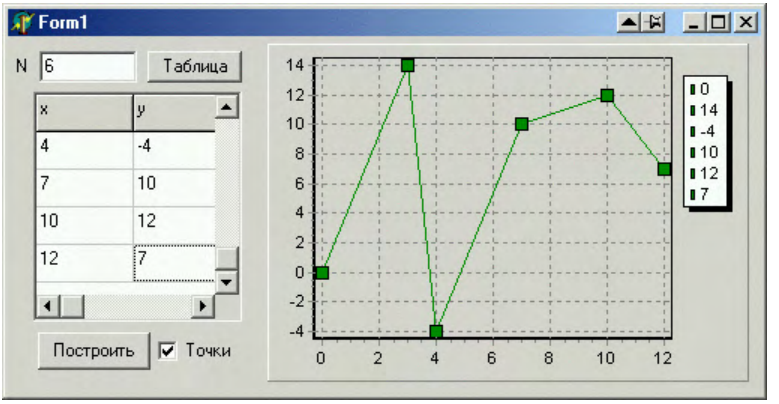

Рис. 26.5. Результат выполнения программы

26.3. Содержание отчета

Отчет должен содержать ответы на контрольные вопросы, тексты программ и выполнение заданий соответствующего варианта.

26.4. Контрольные вопросы

1. С какой целью применяется компонент Chart?
2. Можно ли в *Object Inspector* устанавливать свойства отображения осей?
3. Можно ли на форме располагать два компонента Chart? Если нет, то почему?

4. Разрешается ли в выполнении программы изменять тип диаграммы?

5. Какие параметры задаются на панели Legend в Editing Chart и какие параметры графика можно редактировать с ее помощью?

Варианты заданий

Вариант 1

Построить на одном графике функции $f_1(x) = \exp(x)$ и $f_2(x) = \ln(x)$ на интервале [0,1, 1]. Шаг $h = 0{,}01$

Вариант 2

Построить на одном графике функции $f_1(x) = x$ и $f_2(x) = |x|$ на интервале [-10, 10].

Вариант 3

Построить на одном графике три функции: $f_1(x) = x$, $f_2(x) = x^2$, $f_1(x) = x^3$ на интервале [-20, 20].

Вариант 4

Построить на одном графике две функции, заданные таблично. Значения функций задаются с помощью компонента StringGrid.

Вариант 5

Построить график функции $y = a \cdot x^2 + b \cdot x + c$. Значения параметров a, b, c задаются с клавиатуры (использовать компонент Edit).

Вариант 6

Построить зависимость $I(\varphi) = I_0 \dfrac{\sin^2(\pi \cdot a \cdot \sin(\varphi))}{(\pi \cdot a \cdot \sin(\varphi))^2}$. Предусмотреть возможность задания параметров I_0 и a. Результат представить в графическом виде (компонент Chart) и табличном (компонент StringGrid).

Вариант 7

Построить круговую диаграмму реализации следующей продукции: кофе – 20 %, чай – 35 %, напитки – 45 %. Использовать компонент типа **TPieSeries** (круговая диаграмма). Для добавления значений использовать метод Add, который выглядит следующим образом:

```
function Add(Const PieValue:Double;
             Const APieLabel:String;
             AColor: TColor) ,
```

где `PieValue` – величина элемента ряда данных; `APieLabel` – необязательный параметр; `AColor` – цвет группы, к которой принадлежит точка. Для ввода значений использовать компонент Edit.

Лабораторная работа № 27

ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ЗАПИСЕЙ И ФАЙЛОВ. ПРИМЕНЕНИЕ РАЗВИТЫХ ЭЛЕМЕНТОВ ИНТЕРФЕЙСА ПРИ РАЗРАБОТКЕ ПРИЛОЖЕНИЙ

Цель работы: приобретение практических навыков программирования с использованием записей и файлов.

Используемые программные средства: Delphi.

27.1. Теоретические сведения

Вопросы организации структур типа 'запись', а также работа с переменными файлового типа, описаны в прил. 1.

Работа с компонентами

Списком называется упорядоченная совокупность элементов, являющихся тестовыми строками. Для работы с простым списком в Delphi используется компонент **ListBox**  (панель **Standard**). Некоторые свойства для работы с компонентом ListBox приведены в табл. 27.1.

Таблица 27.1

Свойства компонента ListBox	Описание свойств
`Columns`	определяет число колонок, которые одновременно видны в области списка
`ItemIndex`	определяет выбранный элемент в списке
`Items`	представляет собой массив строк и определяет количество элементов списка и их содержимое
`MultiSelect`	разрешает или отменяет выбор нескольких элементов
`Sorted`	определяет наличие или отсутствие сортировки элементов списка

Отсчет элементов в списке начинается с нуля.

Для работы со свойством Items в режиме проектирования приложения можно использовать **String List Editor** (как и для компонента Memo).

Чтобы добавить новую строку во время выполнения приложения, необходимо вызвать метод `Add` (переменная типа `string`) компонента:

```
ListBox1.Items.Add('новая строка');
```

Для удаления всех строк списка используется метод Clear:

`ListBox1.Clear` или `ListBox1.Items.Clear`.

Содержимое компонента ListBox можно загружать из текстового файла и сохранять в текстовом файле. Для этого используются методы:

`LoadFromFile(const FileName:string)` и `SaveToFile(const FileName:string)` класса `TStrings`.

Практически все приложения Windows имеют свое меню, представляющее собой список пунктов, объединенных по функциональному признаку. Обычно в меню имеется главное меню и несколько контекстных меню. **Главное меню** используется для управления работой всего приложения и располагается в верхней части формы под ее заголовком. **Контекстное меню** появляется при размещении

указателя в области некоторого управляющего элемента и нажатии правой кнопки мыши и служит для управления отдельным интерфейсным элементом.

Для создания и изменения главного меню в процессе разработки приложения используется компонент MainMenu  . Контекстное меню в Delphi представляется компонентом PopupMenu  . Компоненты для работы с меню расположены на панели **Standard**.

Пункты меню представляют собой объекты типа TMenuItem. Некоторые свойства пунктов меню приведены в табл. 27.2.

Таблица 27.2

Свойства объектов TMenuItem	Описание свойств
`Caption`	строка текста, отображаемая как название (заголовок) пункта меню. Если в качестве названия указать символ '-', то на месте соответствующего пункта меню отображается разделительная линия. Знак & используется для подчеркивания символа в строке пункта меню (для выбора пунктов меню с использованием клавиши Alt)
`Bitmap`	определяет изображение пиктограммы, размещаемое слева от названия меню
`Break`	определяет, разделяется ли меню на колонки
`ShortCut`	определяет комбинацию клавиш для активизации пункта меню

Основным событием, связанным с пунктом меню, является событие `OnClick`, возникающее при выборе пункта меню с помощью клавиатуры или мыши.

В Delphi имеется ряд компонентов, находящихся на панели **Dialogs** *Палитры компонентов* и реализующих диалоги общего назначения. Эти диалоги используются многими приложениями Windows для выполнения таких стандартных операций, как открытие, сохранение и печать файлов. Чтобы можно было использовать стандартный диалог, соответствующий ему компонент должен быть помещен на форму. Для вызова любого стандартного диалога используется метод `Execute`.

Основные свойства компонентов `OpenDialog`  и `SaveDialog`  приведены в табл. 27.3.

Таблица 27.3

Свойства компонентов OpenDialog и SaveDialog	Тип	Описание свойств
`DefaultExt`	string	задает расширение, автоматически подставляемое к имени файла, если оно не указано
`FileName`	string	указывает имя и полный путь файла, выбранного в диалоге
`Filter`	string	задает маски имен файлов
`FilterIndex`	integer	указывает, какая из масок фильтра отображается в списке
`InitialDir`	integer	определяет каталог, содержимое которого отображается при вызове окна диалога
`Options`	TOpen-Options	используется для настройки параметров, управляющих внешним видом и функциональными возможностями диалога
`ofOverwritePrompt`		предупреждает пользователя, что файл уже существует и требует подтверждения
`ofNoChangeDir`		вызывает текущий каталог при открытии
`onAllowMultiSelect`		разрешает одновременно выбрать из списка более одного файла
`onPathMustExist`		разрешает указывать файлы только из существующих каталогов
`onFileMustExist`		разрешает указывать только существующие файлы
`onCreatePrompt`		при вводе несуществующего имени файла выдает запрос на создание файла
`Title`	string	задает заголовок окна

Для формирования фильтра используется **Filter Editor** (редактор фильтра) (рис. 27.1), вызываемый через *Object Inspector* в области свойства Filter. Рабочее поле редактора представляет собой таблицу, состоящую из двух колонок. В **области Filter Name** вводится описательный текст, поясняющий маску фильтра, а в области **Filter** – сама маска для отображения файлов. Несколько масок разделяются знаком ';'.

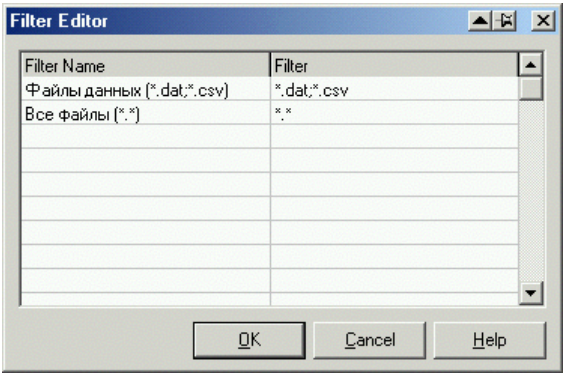

Рис. 27.1. Окно редактора фильтра

Компоненты `OpenPictureDialog` и `SavePictureDialog` вызывают стандартные диалоги открытия и сохранения графических файлов и отличаются от `OpenDialog` и `SaveDialog` только видом окон и установленными значениями свойства `Filter`.

Работа с параметрами шрифта в Delphi реализуется с помощью компонента **FontDialog**.

Для отображения ряда простых диалогов общего назначения в Delphi имеется ряд соответствующих процедур и функций.

Процедура

```
ShowMessage(const Msg:string)
```

отображает простейшее окно сообщений, содержащее выводимое сообщение `Msg` строкового типа.

Функция

```
function MessageDlg(const Msg:string;DlgType:TMsgDlgType;
Buttons:TMsgDlgButtons;HelpCtx:Longint):Word;
```

отображает окно сообщений в центре экрана и позволяет получить ответ пользователя. Параметр `Msg` содержит выводимое сообщение. Тип окна определяется параметром `DlgType`, который может принимать значения `mtWarning`, `mtError` и др. Параметр `Buttons` задает набор кнопок окна.

27.2. Порядок выполнения работы

Изучить описанные компоненты и их свойства, выполнить контрольный пример и задания соответствующего варианта.

Контрольный пример. Написать программу записи (чтения) в файл (из файла) информации о марках сталей и содержании в них легирующих элементов (химическом составе). Интерфейс программы организовать с использованием меню. Для работы с данными использовать структуру типа 'запись':

Марка стали	Химический состав, %			
	Углерод (C)	Марганец (Mn)	Кремний (Si)	Хром (Cr)
15Х	0,15	0,5	0,2	0,75
30Х	0,33	0,8	0,37	1
20ХГСА	0,23	1,1	1,2	0,25
20ХН4ФА	0,24	0,55	0,37	1,1

Решение

1. Открыть новый проект Delphi: **File** – **New Application**.

2. Установить с помощью *Object Inspector* следующие свойства компонента Form1:

```
Form1.Height = 310
Form1.Width = 336
Form1.BorderIcons
      biMaximize = false
Form1.BorderStyle = bsSingle
Form1.Position = poScreenCenter
```

3. Расположить на форме компоненты MainMenu и PopupMenu.

Для организации пунктов главного меню нужно два раза щелкнуть левой кнопкой мыши на компоненте MainMenu и, используя окно *Object Inspector*, ввести свойства каждого пункта меню. Все изменения автоматически отображаются в окне Form1.MainMenu1.

```
N1.Caption = '&Файл'
N2.Caption = '&Новый'
N2.ShortCut = Ctrl+N'
N3.Caption = '-'
N4.Caption = '&Открыть...'
N4.ShortCut = Ctrl+O
N5.Caption = 'Сохранить &как...'
N6.Caption = '-'
N7.Caption = 'В&ыход'
N8.Caption = '&Правка'
N9.Caption = 'До&бавить запись'
N10.Caption = 'И&зменить запись'
N11.Caption = '-'
N12.Caption = '&Шрифт...'
```

Результат показан на рис. 27.2.

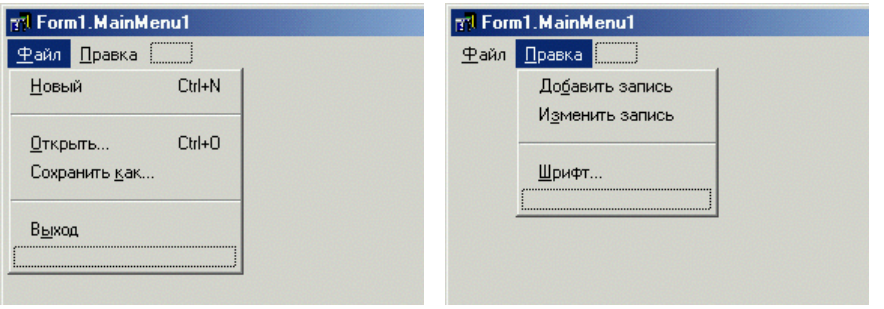

Рис. 27.2. Вид меню при конструировании

4. Для организации контекстного меню необходимо два раза щелкнуть левой кнопкой мыши на компоненте PopupMenu и ввести свойства пунктов меню, используя окно *Object Inspector*:

```
N13.Caption = '&Шрифт...'
```

5. Расположить на форме следующие компоненты: два компонента GroupBox, один компонент Panel, один компонент ListBox и установить с помощью *Object Inspector* для них следующие свойства:

```
GroupBox1.Align = alTop
GroupBox1.Height = 49
GroupBox1.Caption = 'Марка стали'
```

```
GroupBox2.Align = alTop
GroupBox2.Height = 80
GroupBox2.Caption = 'Химический состав, %'
Panel1.BevelInner = bvRaised
Panel1.BevelOuter = bvLowered
Panel1.Caption = ''
Panel1.Height = 71
ListBox1.Align = alClient
```

6. Расположить на форме следующие компоненты: пять компонентов Edit, восемь компонентов Label и два компонента Button. Вид формы приложения с компонентами (с заданными свойствами) показан на рис. 27.3.

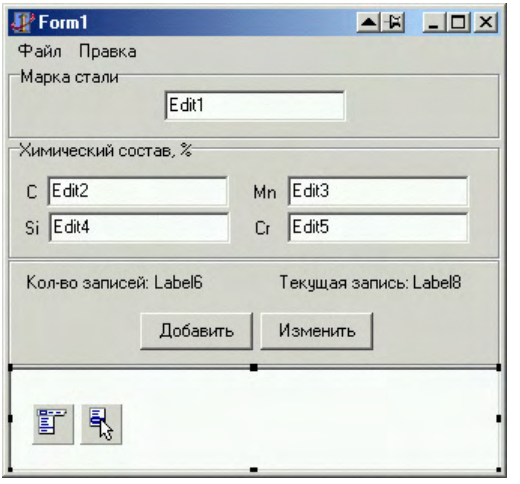

Рис. 27.3. Вид формы

Для этих компонентов установить следующие свойства:

```
Label1.Caption = 'C'
Label2.Caption = 'Mn'
Label3.Caption = 'Si'
Label4.Caption = 'Cr'
Label5.Caption = 'Кол-во записей:'
Label7.Caption = 'Текущая запись:'
Button1.Caption = 'Добавить'
Button2.Caption = 'Изменить'
```

7. В раздел описания ввести соответствующие переменные-идентификаторы, которые должны быть описаны в разделе модуля для глобальных переменных:

```
const MaxRec=10;        //максимальное количество записей
Type TSteel=record      //объявление типа 'запись'
          Name:string[10];    //поле марки стали
          C:real; //поле содержания углерода
          Mn:real;      //поле содержания марганца
          Si:real;      //поле содержания кремния
          Cr:real;      //поле содержания хрома
           end;
var
  Form1: TForm1;
                      // объявление переменной типа запись
  Steel:array [1.. MaxRec] of TSteel;
  Fr:file of TSteel;    //файловая переменная и ее тип
  FileRecName:string;
  NumRecord:integer;    //количество записей
```

8. Создать обработчик события OnCreate для формы. Процедура `ClearAll` используется для задания начальных значений. Текст процедуры `FormCreate` имеет вид

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Form1.Caption:='Контрольный пример;'
  ClearAll;
end;
```

Процедура `ClearAll` будет использоваться в программе в дальнейшем. Сама процедура должна располагаться в тексте модуля выше, чем к ней происходит обращение.

```
Procedure ClearAll;
begin
  Form1.Edit1.Text:='';
  Form1.Edit2.Text:='';
  Form1.Edit3.Text:='';
  Form1.Edit4.Text:='';
  Form1.Edit5.Text:='';
  NumRecord:=0;
  Form1.Label6.Caption:='';
  Form1.Label8.Caption:='';
  Form1.ListBox1.Clear;
end;
```

9. Создать обработчик события Button1.Click (кнопка **Добавить**) для ввода новой записи:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  NumRecord:=NumRecord+1;
  if NumRecord>MaxRec then
     MessageDlg('Количество записей больше '+IntToStr
                (MaxRec),mtError,[mbOk],0)
  else begin
         with Steel[NumRecord] do
         begin
           Name:=Edit1.Text;
           C:=StrToFloat(Edit2.Text);
           Mn:=StrToFloat(Edit3.Text);
           Si:=StrToFloat(Edit4.Text);
           Cr:=StrToFloat(Edit5.Text);
           ListBox1.Items.Add(ViewRecord(NumRecord));
         end;
         Label6.Caption:=IntToStr(NumRecord);
         Edit1.Text:='';
         Edit2.Text:='';
         Edit3.Text:='';
         Edit4.Text:='';
         Edit5.Text:='';
        end;
end;
```

Для создания сообщения о превышении количества допустимых записей использовалась функция MessageDlg.

Для отображения записи с помощью компонента ListBox использовался метод Add компонента. В качестве переменной строкового типа берется результат выполнения функции ViewRecord. Описание процедуры ViewRecord в теле модуля должно предшествовать ее вызову.

```
Function ViewRecord(k:integer):string;
begin
  with Steel[k] do
  begin
   ViewRecord:=Name+' - C:'+FloatToStr(C)
               +'% Mn:'+FloatToStr(Mn)
               +'% Si:'+FloatToStr(Si)
               +'% Cr:'+FloatToStr(Cr)+'%';
  end;
end;
```

10. Создать обработчик события OnClick для компонента ListBox1, обеспечивающий отображение выбранной мышью записи в компонентах Edit1 – Edit5 и номера текущей записи:

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
   with Steel[ListBox1.ItemIndex+1] do
   begin
     Edit1.Text:=Name;
     Edit2.Text:=FloatToStr(C);
     Edit3.Text:=FloatToStr(Mn);
     Edit4.Text:=FloatToStr(Si);
     Edit5.Text:=FloatToStr(Cr);
   end;
   Label8.Caption:=IntToStr(ListBox1.ItemIndex+1);
end;
```

11. Создать обработчик события `Button2.Click` (кнопка **Изменить**) для возможности изменения введенных записей:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  if NumRecord >=1 then
  begin
     with Steel[ListBox1.ItemIndex+1] do
     begin
       Name:=Edit1.Text;
       C:=StrToFloat(Edit2.Text);
       Mn:=StrToFloat(Edit3.Text);
       Si:=StrToFloat(Edit4.Text);
       Cr:=StrToFloat(Edit5.Text);
     end;
     ListBox1.Items[ListBox1.ItemIndex]:=
                    ViewRecord(ListBox1.ItemIndex+1);
  end;
end;
```

Условие `NumRecord>=1` использовано для того, чтобы избежать ошибки во время обращения к несуществующей записи.

12. Для сохранения в файле созданных записей, а также для открытия существующего файла с записями разместить на форме компоненты OpenDialog и SaveDialog и задать для них следующие свойства:

```
   OpenDialog1.Filter = 'Файлы данных (*.dat)|*.dat'
   OpenDialog1.Options
       ofOverwritePrompt = true
       ofNoChangeDir = true
       onPathMustExist = true
       onFileMustExist = true
```

Аналогичные свойства записываются для компонента SaveDialog1.

13. Записать обработчики событий для соответствующих пунктов меню:

```
procedure TForm1.N2Click(Sender: TObject); {пункт меню 'Новый'}
begin
  ClearAll;
end;

procedure TForm1.N4Click(Sender: TObject); {пункт меню 'Открыть'}
var i:integer;
begin
  if OpenDialog1.Execute then
  begin
    ClearAll;
    FileRecName:=OpenDialog1.FileName;
    AssignFile(Fr,FileRecName);
    Reset(Fr);
    NumRecord:=0;
    While not Eof(Fr) do
    begin
      NumRecord:=NumRecord+1;
      read(fr,Steel[NumRecord]);
      ListBox1.Items.Add(ViewRecord(NumRecord));
    end;
    CloseFile(Fr);
    Label6.Caption:=IntToStr(NumRecord);
    MessageDlg('Файл '+FileRecName+' открыт.',
                mtInformation,[mbOk],0);
  end;
end;

procedure TForm1.N5Click(Sender: TObject); {пункт меню
                                    'Сохранить как...'}
var i:integer;
begin
```

```
  if SaveDialog1.Execute then
  begin
    FileRecName:=SaveDialog1.FileName;
    AssignFile(Fr,FileRecName);
    Rewrite(Fr);
    for i:=1 to NumRecord do write(Fr,Steel[i]);
    CloseFile(Fr);
    MessageDlg('Файл '+FileRecName+' сохранен.',
               mtInformation,[mbOk],0);
  end;
end;
```

14. В качестве обработчиков событий для пунктов меню 'Добавить запись' и 'Изменить запись' можно использовать обработчики событий для `Button1.Click` и `Button2.Click`. Для этого с помощью *Object Inspector* (вкладка Events) установить:

```
N9.OnClick = Button1Click
N10.OnClick = Button2Click
```

15. Для изменения параметров шрифта отображаемой в ListBox1 информации надо поместить на форму компонент FontDialog и записать процедуру-обработчик, вызывающую диалоговое окно 'Шрифт':

```
procedure TForm1.N12Click(Sender: TObject);
begin
  if FontDialog1.Execute then
     ListBox1.Font.Assign(FontDialog1.Font);
end;
```

Для вызова контекстного меню при нажатии правой кнопки мыши на компоненте ListBox1 во время выполнения программы нужно установить следующее свойство компонента:

```
ListBox1.Popup = PopupMenu1
```

16. Обработчик события для пункта меню 'Выход' имеет вид:

```
procedure TForm1.N7Click(Sender: TObject);
begin
  Close;
end;
```

17. Запустить проект на выполнение. Окно приложения с введенными записями показано на рис. 27.4.

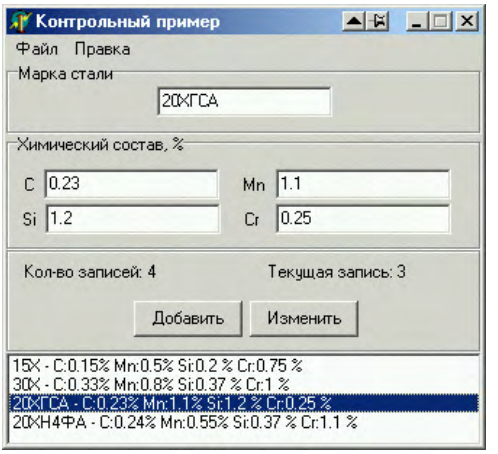

Рис. 27.4. Результат выполнения программы

27.3. Содержание отчета

Отчет должен содержать ответы на контрольные вопросы, тексты программ и результат выполнения задания соответствующего варианта.

27.4. Контрольные вопросы

1. Как в *Object Pascal* организуется структура типа 'запись'?
2. Чем отличаются текстовые, типизированные и нетипизированные файлы?
3. Дать определение файловой переменной.
4. Описать процесс построения главного меню на этапе разработки приложения.

Варианты заданий

Вариант 1

Написать программу для работы с телефонным справочником. Для работы с данными использовать структуру типа 'запись'. Поля должны содержать информацию о фамилии, домашнем и рабочем телефоне. Работу с файлом организовать с помощью меню.

Вариант 2

Составить ведомость студентов, сдавших экзамены за семестр. Для работы с данными использовать структуру типа 'запись'. Поля должны содержать информацию о фамилиях, об оценках по трем предметам и среднем балле. Работу с файлом организовать с помощью меню.

Вариант 3

Составить расписание движения автобусов. Для работы с данными использовать структуру типа 'запись'. Поля должны содержать информацию о фамилии водителе, марке автобуса, времени выхода в рейс и времени возвращения. Работу с файлом организовать с помощью меню.

Вариант 4

Составить ведомость сотрудников, работающих на предприятии. Для работы с данными использовать структуру типа 'запись'. Поля должны содержать информацию о фамилии, должности, наименовании отдела и дате начала работы. Работу с файлом организовать с помощью меню.

Вариант 5

Составить ведомость, содержащую информацию о производимом предприятием товаре за квартал. Для работы с данными использовать структуру типа 'запись'. Поля должны содержать информацию о подразделении, количестве товара в каждом месяце и суммарном значении производимого товара за квартал. Работу с файлом организовать с помощью меню.

Вариант 6

Составить ведомость книг в библиотеке. Для работы с данными использовать структуру типа 'запись'. Поля должны содержать информацию об авторе, названии и дате (год) выпуска книги. Работу с файлом организовать с помощью меню.

Вариант 7

Составить ведомость зарплаты сотрудников. Для работы с данными использовать структуру типа 'запись'. Поля должны содержать информацию о фамилии, начисленной заработной плате по тарифной ставке и полученной сумме после уплаты налога 13%. Работу с файлом организовать с помощью меню.

Лабораторная работа № 28

ИСПОЛЬЗОВАНИЕ СРЕДСТВ DELPHI ДЛЯ РАБОТЫ С ЛОКАЛЬНЫМИ БАЗАМИ ДАННЫХ

Цель работы: изучить основы проектирования локальных баз данных.

Используемые программные средства: Delphi.

28.1. Теоретические сведения

База данных (БД) – это совокупность записей различного типа, организованных по определенным правилам и обеспечивающих хранение и целостность информации.

Система управления базой данных (СУБД) – это совокупность языковых и программных средств, предназначенных для создания, ведения и использования БД.

В зависимости от вида организации данных различают иерархическую, сетевую, реляционную и объектно-ориентируемую модели БД.

Реляционная БД представляет собой совокупность таблиц, связанных отношениями. К достоинствам реляционной БД относятся простота, гибкость структуры и удобство реализации на компьютере. **Таблица** – это двумерный массив, где строки образованы отдельными **записями**, а столбцы – **полями** этой записи. Таблицы хранятся в файлах на жестком диске и похожи на отдельные документы или электронные таблицы, однако в отличие от последних поддерживают многопользовательский режим доступа. Во избежание дублирования информации в таблицах, в реляционных БД определяются **ключи** и **индексы**. **Ключ** – это поле (комбинация полей), данные в котором(ых) однозначно идентифицируют каждую

запись в таблице. **Индекс**, как и ключ, строится по полям таблицы, однако он может допускать повторение значений составляющих его полей. Индекс служит для сортировки таблиц по индексным полям. В простой БД поля можно разместить в одной таблице. В сложной БД поля распределены по нескольким таблицам.

При создании программ, работающих с базами данных, в Delphi используется механизм **Borland Database Engine** (BDE), реализованный в виде набора библиотек, обеспечивающий простой и удобный доступ к базам данных независимо от их архитектуры. Проблема передачи в программу информации о месте нахождения файлов базы данных решается путем использования **псевдонима** (**Alias**) базы данных. **Псевдоним** – это короткое имя, поставленное в соответствие полному имени каталога базы данных, т.е. каталога, в котором находятся файлы базы данных. Для создания и связи псевдонима с каталогом базы данных используется утилита **BDE Administrator**.

Delphi не имеет своего формата таблиц, однако поддерживает два вида локальных таблиц – dBase и Paradox. Таблицы Paradox являются достаточно развитыми и удобными при создании локальных БД.

Для каждого поля таблицы необходимо задать имя, тип и размер поля. Тип поля определяет тип данных, которые могут быть помещены в поле. В Paradox имеется достаточно широкий выбор типов полей, используемых для хранения данных. Некоторые из них приведены в табл. 28.1.

Таблица 28.1

Типы полей таблиц Paradox 7	Обозначение	Описание значений
Alpha	A	Строка символов. Длина не более 255 символов
Number	N	Число с плавающей точкой
Date	D	Дата
Autoincrement	+	Автоинкрементное поле. При добавлении к таблице новой записи в поле автоматически записывается число, на единицу большее, чем находится в соответствующем поле последней добавленной записи

Имя поля в таблице Paradox должно состоять из букв и цифр и начинаться с буквы. Максимальная длина имени поля – 25 символов. Таблица не может содержать два поля с одинаковым именем.

Создание таблиц производится при помощи входящей в состав Delphi утилиты **Database Desktop**. После выбора типа таблицы, в диалоговом окне **Create Paradox 7 Table** следует определить структуру записей таблицы. Тип поля выбирается из списка при нажатии правой кнопки мыши в колонке **Type** или при нажатии клавиши **Пробел**. Файлы таблиц Paradox, хранящихся на диске, имеют расширение *.db.

Таким образом, процесс создания новой базы данных состоит из следующих этапов:

1 – создание каталога; 2 – создание псевдонима; 3 – создание таблицы (таблиц).

Методика разработки в Delphi приложения для управления базой данных ничем не отличается от методики создания обычной программы.

Просмотр баз данных организуется или в режиме формы, или в режиме таблицы. В режиме формы можно видеть только одну запись, а в режиме таблицы – несколько записей одновременно. Довольно часто эти два режима комбинируют.

При работе с базой данных интересует, как правило, не все содержимое базы, а некоторая конкретная информация. Выборка нужной информации производится путем выполнения **запросов**. Текст запроса строится с помощью *структурированного языка запросов* **SQL** (не входит в рассмотрение данной работы). Кроме того, состав записей в наборе данных зависит от установленных ограничений, которые вводятся с помощью **фильтров**. Delphi предоставляет широкие возможности для выполнения различных вариантов фильтрации.

Работа с компонентами

Для создания приложений, работающих с БД, в Delphi имеется ряд компонентов (визуальных и невизуальных) и специальных объектов. Основные компоненты, используемые для работы с локальными базами данных, находятся на страницах **Data Access** (рис. 28.1) и **Data Controls** (рис. 28.2) *Палитры компонентов*.

Рис. 28.1. Страница компонентов **Data Access**

Рис. 28.2. Страница компонентов **Data Controls**

Рассмотрим свойства основных компонентов для работы с БД.

Компонент **Table** (панель **Data Access** *Палитры компонентов*) – набор данных, связанный с одной таблицей БД:

Свойства	Описание
Active	признак активизации/деактивизации файла данных (таблицы)
DatabaseName	имя БД. В качестве значения свойства используется псевдоним БД
Filter	текст фильтра
Filtered	признак активизации/деактивизации фильтра
ReadOnly	признак активизации/деактивизации редактирования таблицы
TableName	имя файла данных (таблицы данных)
Методы	
Next, Prior, First, Last	используются для перемещения указателя соответственно на следующую, предыдущую, первую, последнюю записи набора данных

Компонент **DataSource** (панель **Data Access** *Палитры компонентов*) обеспечивает связь таблиц БД с компонентами просмотра и редактирования содержимого полей БД:

Свойства	Описание
DataSet	имя компонента, представляющего собой входные данные

Компонент **DBGrid** (панель **Data Controls** *Палитры компонентов*) – обеспечивает представление БД в виде таблицы:

Свойства	Описание
DataSource	имя компонента – источника данных

Компонент **DBNavigator** (панель **Data Controls** *Палитры компонентов*) представляет собой набор кнопок для перемещения по записям и их редактирования:

Свойства	Описание
DataSource	имя компонента – источника данных
Hints	массив строк, хранящий всплывающие подсказки для каждой из кнопок. Для активации требуется, чтобы свойство компонента ShowHint было установлено в true

Компоненты **DBEdit** и **DBText** (панель **Data Controls** *Палитры компонентов*) используются для просмотра и редактирования полей записи:

Свойства	Описание
DataField	имя поля для отображения/редактирования
DataSource	имя компонента – источника данных

Взаимосвязь компонентов приложения и таблицы БД и используемые при этом свойства компонентов показаны на рис. 28.3.

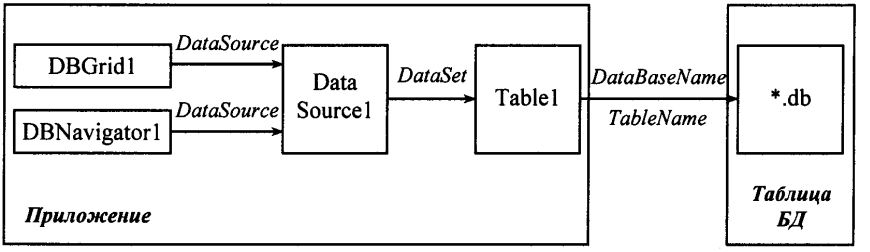

Рис. 28.3. Связь компонентов приложения и таблицы БД

28.2. Порядок выполнения работы

Изучить компоненты Delphi, используемые для управления базой данных, выполнить контрольные примеры и задания соответствующего варианта.

Контрольный пример 1. Разработать программу управления базой данных "Записная книжка". В базе должны содержаться: информация о фамилии, имени, телефоне, адресе электронной почты, дате рождения и графическое изображение.

Решение

1. Перед тем как приступить непосредственно к разработке приложения управления базой данных, необходимо, используя утилиты BDE Administrator и Database Desktop, создать псевдоним базы данных и сам файл данных (таблицу).

1.1. Запустить из Windows утилиту BDE Administrator:

Пуск – Программы – Borland Delphi 5 – BDE Administrator

В левой части окна, на вкладке **Databases**, перечислены зарезервированные на данном компьютере псевдонимы. Чтобы создать новый псевдоним, надо выполнить команду **New** из меню **Object**. В открывшемся диалоговом окне **New Database Alias** выбрать драйвер **Standard** и нажать кнопку **OK**. После этого надо изменить имя **Standard1** на новое имя **DBSprav** и указать в правом окне в поле **PATH** путь к файлам БД. В данном случае ввести путь **c:\Spravochnik** или воспользоваться стандартным диалоговым окном ввода, находящимся в конце поля **PATH** (каталог должен существовать). Чтобы данный псевдоним был зарегистрирован в файле конфигурации, в меню **Object** выбрать команду **Apply**. Результат создания нового псевдонима показан на рис. 28.4.

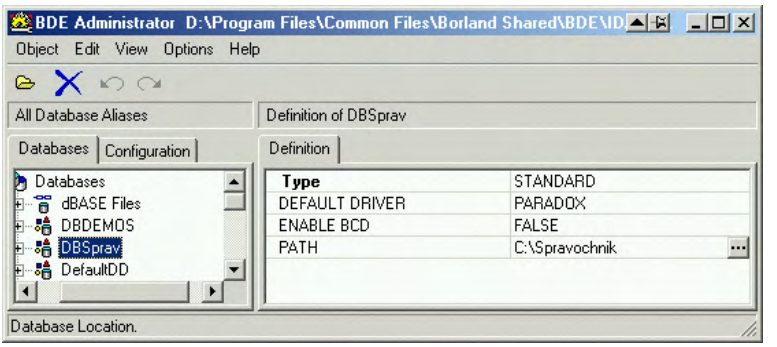

Рис. 28.4. Результат создания псевдонима

1.2. Для создания файла данных (таблицы) необходимо:

– открыть новый проект Delphi: **File** – **New Application**;

– из меню Delphi запустить утилиту Database Desktop: **Tools** – **Database Desktop**;

– в окне Database Desktop из меню **File** выбрать команду **New** и в появившемся списке выбрать тип создаваемого файла **Table** (рис. 28.5).

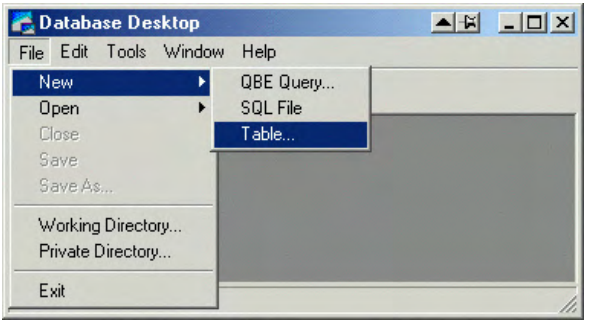

Рис. 28.5. Диалоговое окно Database Desktop

В открывшемся диалоговом окне **Create Table** выбрать тип создаваемой таблицы (**Paradox 7**) и нажать **OK**. В открывшемся диалоговом окне (рис. 28.6) можно определять структуру записей таблицы.

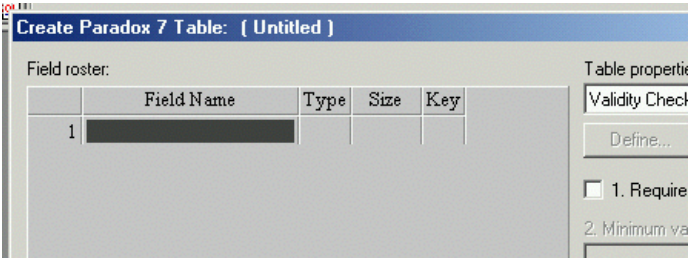

Рис. 28.6. Диалоговое окно **Create Paradox 7 Table**

В табл. 28.2 перечислены поля таблицы для базы данных "Записная книжка". Тип поля (**Type**) задается нажатием правой кнопки мыши или клавишей **Пробел**.

Таблица 28.2

Поле (Field Name)	Тип (Type)	Размер (Size)	Описание
LastName	A	20	фамилия
FirstName	A	15	имя
Tel	N		телефон
Email	A	20	e-mail
Birthday	D		дата рождения
Image	A	12	имя файла иллюстрации

После создания таблицы ее нужно сохранить. Для этого надо нажать кнопку **Save As** в окне **Create Paradox 7 Table**, указать директорию для сохранения имени псевдонима (Alias). В настоящем примере выбрать имя псевдонима **DBSprav** и сохранить таблицу в каталоге **c:\Spravochnik** под именем **sprav.db**.

2. На форме (рис. 28.7) расположить компоненты Table1, DataSource1, DBGrid1, DBNavigator1, CheckBox1 и установить с помощью *Object Inspector* следующие свойства, определяющие внешний вид и поведение компонентов:

```
Form1.Caption = 'Записная книжка'
Form1.Height = 160
Form1.Width = 600
DBGrid1.Align = alTop
DBGrid1.Height = 97
CheckBox1.Caption = 'Разрешить редактирование'
CheckBox1.Width = 169
CheckBox1.Checked = false
DBNavigator1.ShowHint = true
DBNavigator1.Flat = true
Table1.ReadOnly = true
```

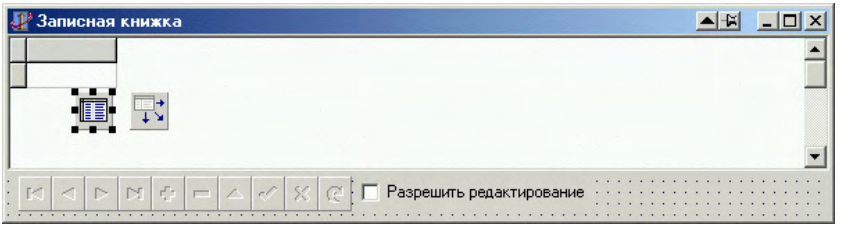

Рис. 28.7. Вид формы

3. Для задания связей между компонентом Table1 и таблицей БД, между компонентом DataSource1 и Table1, между компонентами DBGrid1, DBNavigator1 и DataSource1 надо установить для них следующие свойства:

```
Table1.DatabaseName = 'DBSprav'
Table1.TableName = 'sprav.db'
DataSource1.DataSet = Table1
DBGrid1.DataSource = DataSource1
DBNavigator1.DataSource = DataSource1
Table1.Active = true
```

4. Для редактирования записей, их добавления и удаления написать обработчик события OnClick компонента CheckBox1. Текст процедуры имеет вид:

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  if CheckBox1.Checked then
   with Table1 do
   begin
     Active:=false;
     ReadOnly:=false;
     Active:=true;
   end
  else with Table1 do
        begin
          Active:=false;
          ReadOnly:=true;
          Active:=true;
        end;
end;
```

5. Запустить проект на компиляцию и выполнение. Изменение состояния переключателя **Разрешить редактирование** позволяет использовать компонент DBNavigator1 для работы с записями (редактирование, создание новых, удаление) или запрещает редактировать содержание таблицы данных.

6. Чтобы добавить новую запись, необходимо нажать кнопку  и в новой строке начать ввод информации. Заполнить таблицу следующей информацией:

LastName	FirstName	Tel	Email	Birthday	Image
Зверев	Дима	746323	zvd@mail.ru	17.12.1975	2.ico
Иванова	Наташа	384736	iv_nat@mail.ru	05.08.1972	1.ico
Сергеева	Алла	936523	alla@yandex.ru	13.04.1970	4.ico
Петров	Иван	965432	petrov@yahoo.com	25.11.1976	3.ico
Смирнов	Олег	793546	smirnov@rambler.ru	05.07.1980	5.ico
Иванов	Андрей	573865	ivanov_a@mail.ru	01.01.1970	6.ico
Иванов	Сергей	485325	ivanov_s@mail.ru	03.03.1974	7.ico
Никитина	Ольга	298476	nick@mail.com	08.12.1965	9.ico
Петренко	Сергей	163456	petr@yandex.ru	24.03.1967	8.ico

После окончания редактирования убрать флажок в переключателе **Разрешить редактирование**. Переход между записями осуществляется с помощью кнопок компонента DBNavigator. Окончательное окно приложения, позволяющего работать с базой данных в режиме таблицы, показано на рис. 28.8.

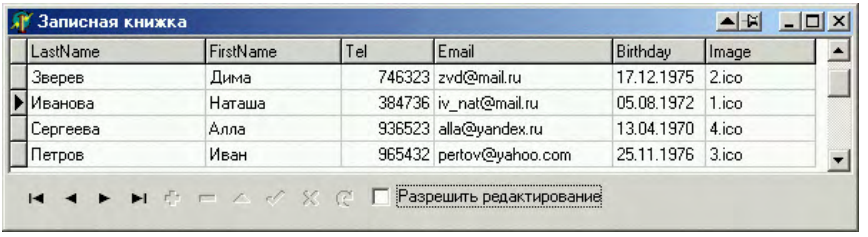

Рис. 28.8. Результат выполнения программы

Контрольный пример 2. Дополнить имеющуюся программу компонентами, позволяющими просматривать и редактировать базу данных в режиме формы. Для управления базой данных необходимо использовать управляющие кнопки.

Решение

1. С помощью *Object Inspector* установить свойство формы `Form1.Width = 324.` Расположить на форме компоненты DBEdit1, DBEdit2, DBEdit3, DBEdit4, Label1, Label2, Label3, Label4, Image1, Button1, Button 2, как показано на рис. 28.9, и установить для них следующие свойства:

```
Label1.Caption = 'Фамилия'
Label2.Caption = 'Имя'
Label3.Caption = 'Эл. почта'
Label4.Caption = 'Изображение'
Image1.Height = 32
Image1.Width = 32
Button1.Caption = 'Предыдущая'
Button1.Height = 25
Button1.Width = 75
Button2.Caption = 'Следующая'
Button2.Height = 25
Button2.Width = 75
```

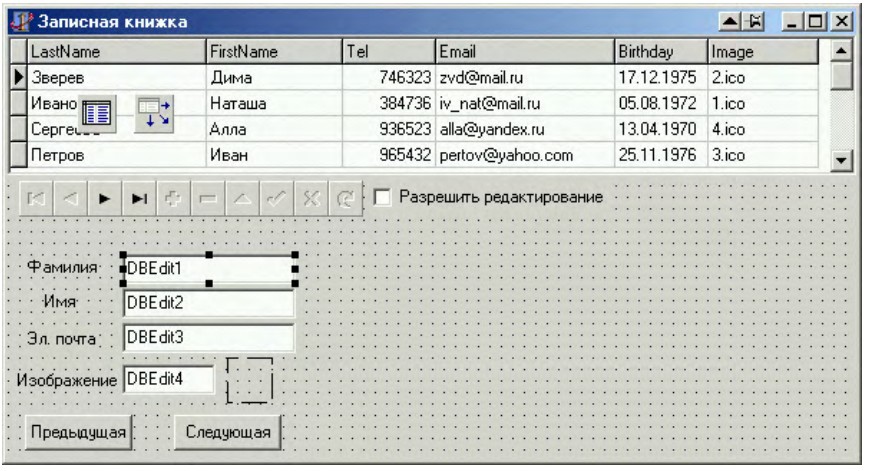

Рис. 28.9. Вид формы

2. Для задания связей между компонентами, обеспечивающими просмотр и редактирование полей базы данных, и компонентом источника данных DataSourse1, установить для них следующие свойства:

```
DBEdit1.DataSource = DataSource1
DBEdit1.DataField = LastName
DBEdit2.DataSource = DataSource1
DBEdit2.DataField = FirstName
DBEdit3.DataSource = DataSource1
DBEdit3.DataField = Email
DBEdit4.DataSource = DataSource1
DBEdit4.DataField = Image
```

3. Записать обработчики событий OnClick компонентов Button1 и Button2 для вызова соответствующих методов управления набором данных. В этом случае при нажатии на кнопки **Предыдущая** или **Следующая** будет осуществляться переход соответственно к предыдущей или следующей записи набора данных Table1.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Table1.Prior;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Table1.Next;
end;
```

4. В поле **Image** таблицы данных хранится имя файла графического изображения, при этом сами файлы должны существовать. В настоящей работе считаем, что файлы хранятся в каталоге **c:\Spravochnik\Image.** Для отображения графического изображения необходимо выполнить процедуру TForm1.Table1AfterScroll, которая обеспечивает обработку события **AfterScroll** для компонента Table1. Это событие происходит всякий раз при переходе между записями таблицы.

Для записи обработчика события надо выделить левой кнопкой мыши компонент Table1, в окне *Object Inspector* выделить вкладку Events и два раза щелкнуть левой кнопкой мыши в поле ввода для свойства **AfterScroll**. Листинг соответствующей процедуры имеет вид:

```
procedure TForm1.Table1AfterScroll(DataSet: TDataSet);
begin
  try
    Image1.Picture.LoadFromFile(ImagePath+DBEdit4.Text);
    Image1.Visible:=true;
  except
     Image1.Visible:=false;
  end;
end;
```

Процедура TForm1.Table1AfterScroll содержит конструкцию `try...except`, чтобы обработать исключительную ситуацию в случае отсутствия изображения, например при добавлении в таблицу

новой записи. ImagePath – переменная строкового типа, которая содержит полный путь к файлу с изображением (имени диска, названия каталога, имени файла). Для ее задания нужно записать процедуру TForm1.Table1BeforeOpen, которая обеспечивает обработку события **BeforeOpen** для компонента Table1. Это событие возникает всякий раз перед открытием таблицы с данными. Запись обработчика события **BeforeOpen** выполняется так же, как и для обработчика события **AfterScroll**. Текст соответствующей процедуры имеет вид

```
procedure TForm1.Table1BeforeOpen(DataSet: TDataSet);
begin
  ImagePath:=ExtractFilePath(ParamStr(0))+'image\';
end;
```

Переменная `ImagePath:string` должна быть описана в модуле в разделе описания глобальных переменных.

В процедуре TForm1.Table1BeforeOpen использовалась `function ExtractFilePath(const FileName:string):string`, возвращающая переменную строкового типа, содержащую название текущего каталога.

5. Запустить проект на компиляцию и выполнение. Для перемещения по записям можно использовать кнопки **Предыдущая** или **Следующая**. При этом в соответствующих компонентах будет отображаться содержимое полей конкретной записи. Окно приложения показано на рис. 28.10.

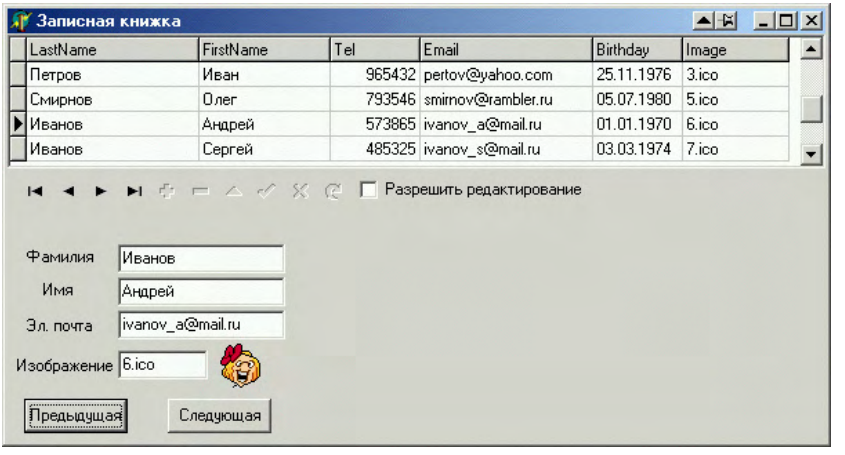

Рис. 28.10. Результат выполнения программы

Контрольный пример 3. Дополнить имеющуюся программу компонентами, позволяющими проводить фильтрацию содержимого базы данных.

Решение

1. Расположить на форме компоненты Edit1, Label5, Button3, Button4. С помощью *Object Inspector* установить следующие свойства компонентов:

```
Label5.Caption = 'Фильтр'
Edit1.Text = ''
Button3.Caption = 'Сбросить фильтр'
Button3.Height = 25
Button3.Width = 105
Button4.Caption = 'Применить фильтр'
Button4.Height = 25
Button4.Width = 105
```

Результат показан на рис. 28.11.

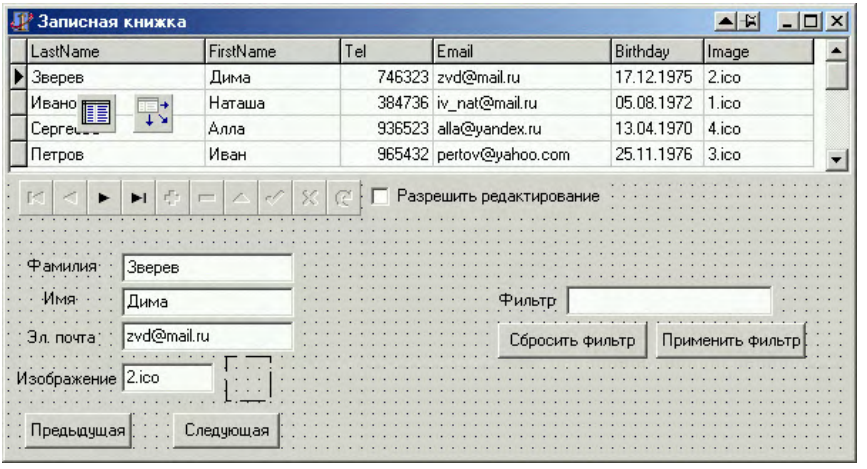

Рис. 28.11. Вид формы

Компонент Edit1 используется для ввода выражения, по которому будет проводиться фильтрация.

2. Записать соответствующие обработчики событий для компонентов Button3 и Button4:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Table1.Filtered:=false;
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
  Table1.Filtered:=true;
  Table1.Filter:=Edit1.Text;
end;
```

3. Запустить проект на компиляцию и выполнение.
4. Ввести выражение, описывающее условие фильтрации:

```
Birthday>'01.01.1975',
```

и нажать кнопку **Применить фильтр**. Результат показан на рис. 28.12.

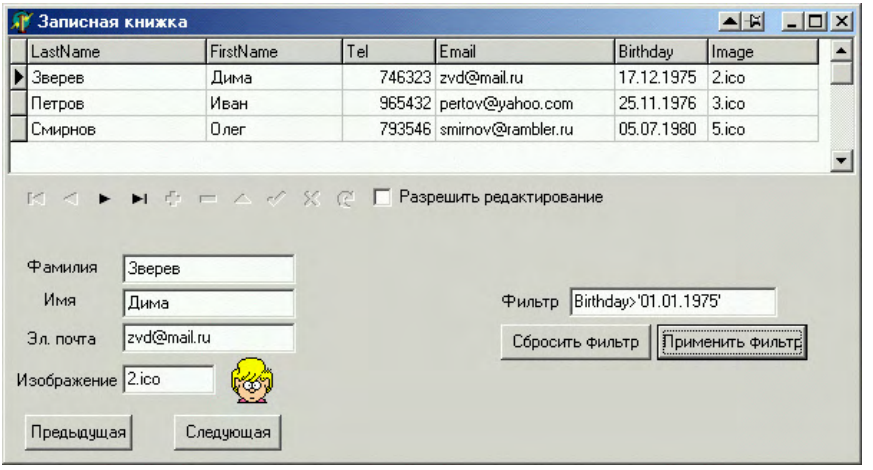

Рис. 28.12. Результат выполнения фильтрации

После нажатия кнопки **Сбросить фильтр** в таблице отображаются все записи, имеющиеся в таблице.

28.3. Содержание отчета

Отчет должен содержать ответы на контрольные вопросы, тексты программ и решение соответствующего варианта.

28.4. Контрольные вопросы

1. В чем заключается отличие между БД и СУБД
2. Что называется псевдонимом БД?
3. Какие компоненты обеспечивают доступ к файлам данных (таблицам) и где они расположены на *Палитре компонентов*?
4. Какие компоненты обеспечивают просмотр и редактирование содержимого полей БД? Где расположены эти компоненты?
5. Чем отличаются просмотры БД в режиме таблицы и в режиме формы?
6. Что называется выборкой информации из БД?

Варианты заданий

Вариант 1

Разработать программу управления базой данных, содержащей информацию об учениках в классе, изучаемых предметах и успеваемости. Обеспечить просмотр базы данных в режиме формы и возможность продвижения по базе данных.

Вариант 2

Разработать программу управления базой данных, содержащей информацию о товаре, производителе, годе выпуска и цене. Обеспечить просмотр базы данных в режиме таблицы и возможность фильтрации исходных данных.

Вариант 3

Разработать программу управления базой данных, содержащей информацию о студентах и оценках, полученных на экзамене по трем предметам. Обеспечить просмотр базы данных в режиме таблицы и возможность фильтрации исходных данных.

Вариант 4

Разработать программу управления базой данных, содержащей информацию о книгах в библиотеке. В полях таблицы должна находиться информация об авторах, названии и годе издания. Обеспе-

чить просмотр базы данных в режиме формы и возможность фильтрации исходных данных.

Вариант 5

Разработать программу управления базой данных, содержащей информацию о студентах в группе. В полях таблицы должна находиться информация о фамилии, номере зачетной книжки, дате рождения и телефоне. Обеспечить просмотр базы данных в режиме формы и возможность фильтрации исходных данных.

Вариант 6

Разработать программу управления базой данных, содержащей информацию об учениках в школе. В полях таблицы должна находиться следующая информация: фамилии учеников, рост, вес, пол. Обеспечить просмотр базы данных в режиме формы и возможность продвижения по базе данных.

Вариант 7

Разработать программу управления базой данных, содержащей информацию об учете разговоров по телефону. В полях таблицы должна находиться следующая информация: о дате звонка, фамилии звонившего и времени разговора. Обеспечить просмотр базы данных в режиме таблицы и возможность продвижения по базе данных, а также фильтрации исходных данных.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

Основные понятия языка программирования Object Pascal

Язык программирования Pascal разработан в 1971 г. известным специалистом по вычислительной технике Никлаусом Виртом (Niclaus Wirth) и был предназначен для обучения специалистов методам разработки компиляторов. (Компилятор – программа, переводящая инструкции языка программирования на язык инструкций процессора вычислительной машины.) Простота языка, ясность и логичность послужили основой для его широкого использования. Совершенствование языка Pascal в части объектно-ориентированного программирования (ООП) привело к созданию Object Pascal.

Основные элементы языка Object Pascal

– **алфавит**, включающий прописные (A – Z), строчные (a – z) буквы латинского алфавита, знак подчеркивания "_", десятичные цифры (0 – 9) и специальные символы (**.** + - = > { } ( ) ; и т.д.). Шестнадцатеричные цифры строятся из десятичных цифр и букв от A(a) до F(f).

Специальные символы (каждый в отдельности либо в комбинации друг с другом) позволяют описывать различные операции, например (* *) – совокупность знаков для записи комментариев.

– **словарь языка**, включающий приблизительно 80 зарезервированных (ключевых) слов, которые имеют фиксированное начертание и раз и навсегда определенный смысл. Например begin, function, type, with, var и т.д.

Важным понятием языка являются **идентификаторы**, которые бывают стандартные и идентификаторы пользователя.

Стандартные идентификаторы используются для обозначения заранее определенных разработчиками конструкций языка и в отличие от зарезервированных слов могут быть переопределены.

Идентификаторы пользователя применяются для обозначения имен меток, констант, переменных, процедур, функций и типов данных. Эти имена задаются разработчиком программы и подчиняются следующим правилам:

- идентификатор составляется из букв и цифр;

- идентификатор всегда начинается только с буквы или знака подчеркивания (исключением являются метки, которыми могут быть целые числа без знака в диапазоне 0 – 9999);
- между двумя идентификаторами в программе должен быть, по крайней мере, один разделитель.

Примеры идентификаторов, определенных пользователем:

```
Label1
1Max  – ошибка (если не метка)
NumPoint
Kol.Uzla.Setki  – ошибка
```

Нетипизированными константами называются элементы данных, которые установлены в описательной части программы с использованием зарезервированного слова **const** и не изменяются в процессе выполнения программы. Зарезервированные константы true (истина), false (ложь) и другие распознаются компилятором автоматически без предварительного описания.

Типизированные константы – элементы данных, тип которых описывается в разделе **const** программы и которые могут изменять свои значения в процессе выполнения программы. По сути, типизированная константа равнозначна переменной с заранее инициализированным значением. В дальнейшем действия над ней могут производиться так же, как и над переменными.

В отличие от констант **переменные** могут менять свои значения в процессе выполнения программы. Тип переменной должен быть определен перед тем, как с ней будут выполняться какие-либо действия. Для описания переменных используется зарезервированное слово **var**.

Типы данных в Object Pascal

Каждый элемент данных, используемый в программе, принадлежит к определенному типу данных, при этом тип переменной указывается при ее описании, а тип констант определяется компилятором автоматически по указанному значению. **Тип** определяет множество значений, которые могут принимать элементы программы, и совокупность операций, допустимых над этими значениями.

Целочисленные типы представляют собой значения, которые могут занимать в памяти следующий объем:

Обозначение	Диапазон	Представление в памяти, байт
shovtint	-128 ... +127	1
smallint	-32768 ... +32767	2
integer	-2147483648...+2147483647	4
longint	-2147483648...+2147483647	4
byte	0 ... +255	1
word	0 ... +65535	2
longword	0 ... 4294967295	4

Для записи целых чисел можно использовать цифры и знаки " + " и " - ". Если знак числа отсутствует, то число считается положительным. Для записи чисел в шестнадцатеричной системе перед ним ставится знак $ (без пробела), а допустимый диапазон значений – $00000000 ... $FFFFFFFF.

Вещественные типы представляют собой вещественные значения, которые используются в арифметических выражениях и могут занимать в памяти следующий объем:

Обозначение	Минимальное значение	Максимальное значение	Точность (число цифр мантиссы)	Память, байт
real	$2{,}9\cdot10^{-39}$	$1{,}7\cdot10^{38}$	11-12	6
single	$1{,}79\cdot10^{-45}$	$3{,}4\cdot10^{38}$	7-8	4
double	$5{,}0\cdot10^{-324}$	$1{,}7\cdot10^{308}$	15-16	8
extended	$3{,}6\cdot10^{-4951}$	$1{,}1\cdot10^{4932}$	19-20	10

Запись вещественных чисел возможна в форме с **фиксированной** или **плавающей точкой.** Для вещественных чисел с фиксированной точкой действия записываются по обычным правилам арифметики. Целая часть отделяется от дробной точкой. Перед числом может ставиться знак " + " или " - ". Для записи вещественных чисел с плавающей точкой указывается порядок числа со значением, отделенным от мантиссы знаком E или e.

Булевский тип данных представлен двумя значениями: **true** (истина) и **false** (ложь). Они широко применяются в логических выражениях отношения.

К значениям **литерного типа** относятся элементы из набора литер, т.е. отдельные символы, и их последовательности (строки). Символы имеют тип **char** и записываются в виде знака, взятого в одиночные кавычки, например, '5', 'S' и т.д.

Язык программирования Object Pascal поддерживает три **строковых** типа: **shortsting** (или **string**), **longstring и widestring.** Типы **shortsting** (**string**) представляют собой *статически* размещенные в памяти компьютера строки длиной от 0 до 255 символов. Тип **longstring и widestring** представляют собой *динамически* размещаемые в памяти компьютера строки, длина которых ограничена только объемом свободной памяти (отличие между ними заключается только в кодировке символов). Строка может быть *пустой*, т.е. не содержащей ни одного символа (записывается, как две идущие подряд одиночные кавычки ''). Для явного задания длины строки после ключевого слова *string* в квадратных скобках задается число, определяющее эту длину: string[20]. Для такой строки на этапе компиляции будет выделяться область памяти в 20 символов (вписать 21 символ в нее нельзя, меньше можно, но объем памяти остается неизменным).

Перечисляемый тип задается непосредственно перечислением всех значений (имен), которые может принимать переменная данного типа.

Формат описания перечисляемого типа:

```
type <имя_типа>=(<имя_1>,...,<имя_n>);
```

Пример:

```
type Season=(Winter,Spring,Summer,Autumn);
```

Интервальный тип описывается путем задания двух констант, определяющих границы допустимых для данного типа значений. Эти границы и определяют интервал (диапазон) значений.

Формат описания интервального типа:

```
type <имя_типа>=<константа_1>..<константа_n>;
```

Пример:

```
type Day=1..31;
```

Массивом называется упорядоченная индексированная совокупность однотипных элементов, имеющих общее имя. Элементами массива могут быть данные различных типов. Каждый элемент массива одновременно определяется *именем* массива и *индексом* (номер этого элемента в массиве) или *индексами*, если массив многомерный. Количество индексных позиций определяет размерность массива (одномерный, двумерный и т.д.).

Статическим массивом называется массив, границы индексов и, соответственно, размеры которого задаются при объявлении, т.е. они известны до компиляции программы.

Формат:

```
type <имя_типа>=array [тип_индекса] of <тип_элементов>;
var <идентификатор>:<имя_типа>;
```

или

```
var
  <идентификатор>:array [тип_индексов] of <тип_элементов>;
```

Пример:

```
type Mas1=array [1..10] of real;
     Mas2=array [1..10,1..10] of real;
var m1:Mas1;
    m2:Mas2;
    m3:array [1..100] of integer;
```

Динамический массив представляет собой массив, для которого при объявлении указывается только тип его элементов, а размер его определяется при выполнении программы. Задание размера динамического массива во время выполнения программы производится процедурой **SetLength**(var M; NewLength: Integer), которая для динамического массива M устанавливает новый размер, равный *NewLength*. Выполнять операции с динамическим массивом и его элементами можно только после задания размеров этого массива.

Формат:

```
type <имя_типа>=array of <тип_элементов>;
var <идентификатор>:<имя_типа>;
```

или

```
var <идентификатор>:array of <тип_элементов>;
```

Пример:

```
type Mas=array of real;
var M:Mas;
   ...
    SetLength(M,100);
   ...
```

Для многомерного динамического массива установка новых размеров выполняется для каждого индекса.

Действия над массивом выполняются обычно поэлементно, как правило с использованием операторов цикла.

Множества представляет собой совокупность элементов, выбранных из определенного набора значений. Все элементы множе-

ства имеют порядковый тип; количество элементов множества не может превышать 256. В выражениях значения элементов множества указываются в квадратных скобках. Если множество не имеет элементов, оно называется пустым и обозначается [].

Формат:

```
type <имя_типа>=set of <элемент_1;...,элемент_n>;
var <идентификатор>:<имя_типа>;
```

или

```
var <идентификатор>:set of <элемент_1,...,элемент_n>;
```

Пример:

```
type Date=set of 1..31;
var Day:Date;
```

При работе с множествами допускается использование операций отношения, объединения, пересечения, разность множеств и операции *in*. Результатом выражений с применением этих операций является значение *true* или *false*.

Записи объединяют фиксированное число элементов данных других типов. Отдельные элементы записи имеют имена и называются **полями**. Различают фиксированные и вариантные записи.

Фиксированная запись состоит из конечного числа полей, ее объявление имеет следующий вид:

Формат:

```
type <имя_типа>=record
        <идентификатор_поля_1>:<тип_поля_1>;
        ...
        <идентификатор_поля_n>:<тип_поля_n>;
     end;
var <идентификатор>:<имя_типа>;
```

Пример:

```
type Elements=record
        N:integer;
        KoordX:real;
        KoordY:real;
        KoordZ:real;
     end;
var Element:Elements;
```

Вариантная запись, так же как и фиксированная, имеет конечное число полей, однако позволяет по-разному интерпретировать области памяти, занимаемые полями.

Формат:

```
type <имя_типа>=record
        <идентификатор_поля>:<тип_поля>;
        case <поле_признака>:<имя_типа> of
            <Вариант_1>:(поле_1:тип_1);
            <Вариант_2>:(поле_2:тип_2);
      end;
var <идентификатор>:<имя_типа>;
```

Пример:

```
type Elements=record
       N:integer
       case Flag:boolean of
           true:(usel1,usel2,usel3:integer);
           false:(usel1,usel2,usel3,usel4:real);
    end;
var Element:Elements;
```

Для обращения к конкретному полю необходимо указать имя записи и имя поля, т.е. имя поля является составным. С полем можно выполнять те же операции, что и при переменной этого поля:

```
Element.N:=5;
Element.KoordY:=10;
Element.KoordX:=Element.KoordY;
```

Зарезервированное слово **with** позволяет использовать в тексте программы имена полей без указания имени переменной-записи.

Формат:

```
with <переменная_типа_запись> do
begin
   <операторы>;
end;
```

Пример:

```
with Element do
begin
   N:=5;
   KoordY:=10;
   KoordX:=KoordY;
end;
```

Язык Pascal допускает вложение записей друг в друга.

Указатель представляет собой переменную, значением которой является адрес начала размещения некоторых данных в основной памяти, т.е. указатель содержит ссылку на соответствующий объект. Иными словами, указатель только указывает на место в памяти,

а получить доступ к данным можно с помощью специальных операций. Переменные типа "указатель" являются динамическими, т.е. их значения определяются во время выполнения программы. Указатели могут ссылаться на данные любого типа. Переменная-указатель, как и любая переменная, должна быть объявлена в разделе объявления переменных.

Формат:

```
var <имя>:^<тип>;
```

Пример:

```
var M1:^integer;
    S:^real;
```

Для выделения памяти для динамической переменной используется процедура **new**, содержащая только один параметр – указатель на переменную того типа, память для которой надо выделить. Для освобождения памяти, занимаемой динамической переменной, используется процедура **dispose**, содержащая указатель на динамическую переменную.

Пример:

```
var m1,m2,m3:^real;     {указатель на переменные типа real}
begin
   new(m1);             {создание динамических переменных}
   new(m2);
   new(m3);
   m1^:=10;             {работа с динамическими переменными}
   m2^:=20;
   m3^:=m1^+m2^;
   dispose(m1);        {уничтожение динамических переменных}
   dispose(m2);
   dispose(m3);
end;
```

Файл представляет собой именованную последовательность однотипных элементов, размещенных на внешнем устройстве, чаще всего на диске. В отличие от одномерного динамического массива, он размещается не в оперативной, а во внешней памяти и не требует предварительного указания размера. Для выполнения операций с конкретным файлом, размещенным на диске, используется **файловая переменная**, которая после ее описания связывается с некоторым файлом. Операции с файловой переменной приводят к соответствующим изменениям в файле. После завершения всех операций связь между файловой переменной и файлом разрывается.

В зависимости от типа элементов различают **текстовые**, **типизированные** и **нетипизированные** файлы. **Текстовый файл** содержит строки символов переменной длины; **типизированный файл** составляют элементы указанного типа (кроме файлового); в **нетипизированном файле** содержаться элементы, тип которых не указан.

Пример:

```
var                 {примеры задания файловой переменной}
  f1:TextFile;      {для работы с текстовым файлом}
  f2:File of TMyOutFile;  {для работы с типизированным
                           файлом. Структура TMyOutFile
                           должна быть описана}
  f3:File of real; {для работы с типизированным файлом}
  f4:File;         {для работы с нетипизированным файлом}
```

Структура Object Pascal программы

Программа – это ряд последовательных инструкций, реализующих набор предписаний, однозначно определяющих содержание и последовательность выполнения операций для решения определенных задач.

Каждая программа состоит из заголовка и блока, который делится на **описательную** и **исполнительную** части.

Раздел описаний включает описание переменных, констант, меток, подпрограмм, типов данных и других объектов, используемых в программе.

Часть программы, выполняющая какие-либо действия, называется **разделом операторов**.

В общем случае **структуру Object Pascal программы** можно представить следующим образом:

```
program <имя>                          – заголовок программы;
uses    <список модулей>               – раздел подключения модулей;
label   <список меток>                 – раздел объявления меток;
const   <список констант>              – раздел объявления констант;
type    <описание типов>               – раздел описания типов данных;
var     <объявление переменных>        – раздел объявления переменных;
procedure <описание процедур>          – раздел описания процедур;
function <описание функций>            – раздел описания функций;
begin                                  – тело программы.
  <раздел операторов>
end.
```

В конце раздела ставится " ; ".

Комментарии в программе представляют собой пояснительный текст, который можно записывать в любом месте программы, где разрешен пробел. Текст комментария ограничен символами (* *) или { }, может содержать любые символы и занимать несколько строк. Любой раздел, кроме раздела операторов, может отсутствовать. Раздел **Uses** всегда расположен после заголовка программы. Разделы описаний могут следовать в произвольном порядке.

Раздел подключения модулей состоит из зарезервированного слова **uses** и списка имен подключаемых стандартных и (или) определенных пользователем библиотечных модулей.

Формат:

```
uses <имя_1>,<имя_2>,<имя_3>;
```

Пример:

```
uses MyGraph,Dos,System,MyLib;
```

Раздел объявления меток начинается с зарезервированного слова **label,** за которым следуют имена меток. Метку можно поставить перед любым оператором в теле программы, что позволяет выполнить прямой переход на этот оператор с помощью оператора **goto**.

Формат:

```
label <имя_1>,<имя_2>,...;
```

Пример:

```
label M1,LL;
...
begin      {тело программы}
   ...
   goto M1;
   ...
   goto LL;
   ...
   M1:<оператор_1>;
   ...
   LL:<оператор_2>;
   ...
end.       {тело программы}
```

Раздел объявления констант начинается с использования зарезервированного слова **const**.

Формат (для **нетипизированных** констант):

```
const <идентификатор_1>=<значение_1>;
      <идентификатор_2>=<значение_2>;
```

Пример:

```
const
    Length=5;
    Stroka='наименование';
```

Формат (для **типизированных** констант):

```
const <идентификатор_1>:<тип_1>=<значение_1>;
      <идентификатор_2>:<тип_2>=<значение_2>;
```

Пример:

```
const
   Length:integer=5;
   Stroka:string='наименование';
```

Раздел описания типов данных начинается с использования зарезервированного слова **type**. В нем указываются типы данных, определенные разработчиком программы. Кроме того, типы могут быть описаны неявно в разделе описания переменных. Каждое описание задает множество значений и связывает с этим множеством некоторое имя типа. В языке имеется ряд стандартных типов, не требующих предварительного описания, например *integer*, *real*, *boolean* и другие.

Формат:

```
type <имя_типа>=<значение_типа>;
```

Пример:

```
type
  Matrix=array [1..10,1..10] of real;
  Month=1..12;
  Char=('a'..'z');
```

Раздел описания переменных начинается с зарезервированного слова **var**. Каждая встречающаяся в программе переменная должна быть объявлена. Описание обязательно предшествует использованию переменной.

Формат:

```
var <идентификатор_1>:<тип_1>;
    <идентификатор_2>:<тип_2>;
```

Пример:

```
var i1,j1,i2,j2:integer;
    z1,z2:boolean;
    sml:char;
    ss:string;
    mass:array [1..4] of real;
```

В **разделе описания процедур и функций** размещаются тела подпрограмм, описание которых дано в соответствующем разделе.

Раздел операторов в языке Pascal является основным и начинается с зарезервированного слова **begin.** Далее следуют операторы языка, отделенные друг от друга " ; " (точка с запятой). Завершается раздел зарезервированным словом **end** и " . " (точка).

Формат:

```
begin
   <оператор_1>;
   <оператор_2>;
   ...
   <оператор_n>;
end.
```

Пример:

```
begin
   ...
   a:=10;
   b:=15;
   c:=b/a;
   ...
end.
```

Модуль. Структура модуля

Кроме программ, структура которых рассмотрена выше, средства языка позволяют создавать **модули**, которые служат средством создания библиотеки подпрограмм (процедур и функций). В отличие от программы, модуль не может быть скомпилирован в выполняемый файл, зато его элементы можно использовать в программе или других модулях. Для использования средств модуля его необходимо подключить, указав имя этого модуля в разделе *uses*. Создание библиотечного модуля требует определенной структурной организации. Общая структура модуля имеет вид:

```
unit <имя модуля>;
interface                                  {раздел интерфейса}
uses          <список модулей>;
const         <список констант>;
type          <описание типов>;
var           <объявление переменных>;
<заголовки процедур>
<заголовки функций>
implementation                             {раздел реализации}
uses          <список модулей>
```

```
const          <список констант>
type           <описание типов>
var            <объявление переменных>
<описание процедур>
<описание функций>
initialization                    {раздел инициализации}
<операторы>
finalization                      {раздел деинициализации}
<операторы>
End.
```

В разделе **интерфейса** размещаются описания идентификаторов, которые должны быть доступны всем модулям и программам, использующим этот модуль, и содержащим его имя в списке *uses*. В разделе также объявляются типы, константы, переменные и подпрограммы, при этом для подпрограмм указываются только их заголовки. Другие доступные модули указываются в списке uses. Раздел начинается с зарегистрированного слова **interface**.

В разделе **реализации** располагаются подпрограммы, заголовки которых были приведены в разделе *interface*, типы, переменные, константы и подпрограммы, которые используются только в этом модуле. Раздел начинается со слова **implementation**.

Раздел **инициализации** начинается с зарегистрированного слова **initialization** и содержит операторы, выполняемые в начале работы программы, которая подключает этот модуль. При наличии раздела инициализации в модуле можно использовать раздел **деинициализации**, который начинается зарезервированным словом **finalization** и содержит операторы, выполняемые при завершении программы. Разделы *initialization* и *finalization* не являются обязательными.

Операторы

Оператор присваивания предписывает вычислить выражение, заданное в правой части, и присвоить результат переменной, имя которой расположено в левой части оператора. Переменная и выражение должны иметь совместимый тип.

Формат:

```
<имя_переменной>:=<выражение>;
```

Вместо имени переменной можно указывать элемент массива или поле записи. Знак присваивания "**:=**" означает, что сначала вы-

числяется значение выражения, а затем оно присваивается указанной переменной.

Пример:

```
var u:integer;
    x:real;
    str:string;
    ...
    n:=sqr(n+1)+sqrt(n);
    x:=-10.756;
    str:='дата'+''+'время';
```

Оператор перехода (goto) предназначен для изменения порядка выполнения операторов программы и используется в случаях, когда после выполнения некоторого оператора требуется выполнить не следующий по порядку, а какой-либо другой помеченный меткой оператор.

Формат:

```
goto:=<метка>;
```

Пустой оператор представляет собой точку с запятой и может быть расположен в любом листе программы, где допускается наличие оператора. Пустой оператор не выполняет никаких действий, может быть помечен меткой и может быть использован для передачи управления в конец цикла или составного оператора.

Составной оператор представляет собой группу из произвольного числа любых операторов, отделенных друг от друга точкой с запятой и ограниченных операторными скобками **begin** и **end**. Составной оператор воспринимается как единое целое независимо от числа входящих в него операторов и может располагаться в любом месте программы, где допускается наличие оператора. Наиболее часто составной оператор используется в условных операторах и операторах цикла.

Формат:

```
begin
   <оператор_1>;
   ...
   <оператор_n>;
end;
```

Составные операторы могут вкладываться друг в друга, при этом на глубину вложенности составных операторов ограничений не накладывается.

Условный оператор (if) обеспечивает выполнение или невыполнение некоторых операторов в зависимости от соблюдения определенных условий

Формат:

```
if <условие> then <оператор_1> [else <оператор_2>];
```

Если условие истинно (*true*), то выполняется <оператор_1>, в противном случае – <оператор_2>. Оба оператора могут быть составными.

Условия представляют собой логические выражения. В них происходит сравнение значений выражений, вызов функций, возвращающих значение типа *boolean* и комбинирование этих значений с помощью логических операций.

Основными операциями сравнений являются (используются без кавычек): "=" – равно; "<>" –неравно; ">" – больше; "<"– меньше; ">=" – больше или равно; "<=" – меньше или равно; "in" – принадлежность.

Если используются логические операции **or** (*арифметическое или*), **and** (*арифметическое и*) и так далее, то связываемые ими условия заключаются в круглые скобки.

Пример:

```
if x>10...;
if (I>=1) and (I<=20)...;
```

Для организации разветвлений на три и более направлений, используется несколько условных операторов, вложенных друг в друга.

Пример:

```
if a>b then
begin
   if a>d then c:=10 else c:=7;
end
else c:=5;
```

Оператор выбора (case) является обобщением условного оператора и позволяет сделать выбор из произвольного числа имеющихся вариантов.

Формат:

```
case <выражение-селектор>
  <список_1>:<оператор_1> (<составной_оператор_1>);
  ...
  <список_n>:<оператор_n> (<составной_оператор_n>);
else                          {раздел может отсутствовать}
  <оператор> (<составной_оператор_n>);
end;
```

Выражение-селектор должно быть порядкового типа. Каждый вариант представляет собой список констант и отделенного от него символом двоеточия оператора (составного оператора). Список констант выбора состоит из произвольного количества значений и диапазонов, отделенных друг от друга запятыми. Границы диапазона записываются двумя константами через разделитель "..". Тип констант должен соответствовать типу выражения-селектора.

Оператор выбора выполняется следующим образом:

1. Вычисляется значение выражения-селектора.

2. Производится последовательный просмотр вариантов на предмет совпадения значений селектора с константами и значениями из диапазонов соответствующего списка.

3. Если для очередного варианта этот поиск успешный, то выполняется оператор этого варианта. После этого выполнение оператора выбора заканчивается.

4. Если все проверки оказались безуспешными, то выполняется оператор, состоящий после слова else (при его наличии).

Пример:

```
case i of
   1..10:ss:="Вариант 1";
   11,12:ss:="Вариант 2";
else ss:="Вариант 3";
end;
```

Оператор цикла с параметром (for) имеет два формата:

```
for <параметр_1>:=<выражение_1> to <выражение_2> do
<оператор> (<составной оператор>);
```

или

```
for <параметр_2> := <выражение_3> downto <выраже-
ние)4> do <оператор> (<составной оператор>);
```

Параметр_1(2) представляет собой переменную порядкового типа и называется параметром цикла. Выражение_1(3) и выражение_2(4) являются соответственно *начальным* и *конечным* значениями параметра цикла и должны иметь тип, совместимый с типом параметра цикла. Тело цикла расположено после слова do, которое выполняется каждый раз до полного перебора всех значений параметра цикла от начального до конечного значений. Шаг параметра всегда равен 1 для первого формата и –1 для второго формата. Цикл может не выполниться ни разу, если выражение_1 больше

выражения_2 или выражение_3 меньше выражения_4. Циклы разрешено вкладывать один в другой.

Пример:

```
for k:=1 to 10 do s:=s+1;

for i:=1 to 20 do
for j:=1 to 30 do
begin
   mas1[i,j]:=mas2[i,j];
   mas2[i,j]:=0;
end;
```

Оператор цикла с постусловием (repeat) целесообразно использовать, когда тело цикла необходимо выполнить не менее одного раза и заранее неизвестно общее количество повторений цикла.

Формат:

```
repeat
   <оператор_1>;
   ...
   <оператор_n>;
until <условие>;
```

Условие – это выражение логического типа. Тело цикла заключено между словами repeat и until. Операторы тела цикла выполняются до тех пор, пока условие не примет значение *true*. В теле цикла может находиться произвольное число операторов. По крайней мере один из операторов тела цикла должен влиять на значение условия.

Пример:

```
...
y:=1;
pr:=1;
repeat
   pr:=pr*y;
   y:=y+1;
until y>=20;
```

Оператор цикла с предусловием (while) аналогичен оператору repeat ... until, но проверка условия выполняется в начале оператора. Оператор целесообразно использовать в случаях, когда число повторений тела цикла заранее неизвестно и тело цикла может не выполняться.

Формат:

```
while <условие> do
   <оператор>;
```

Операторы в теле цикла выполняются до тех пор, пока `условие` не примет значение *false*. Если до оператора условие не выполняется, то тело цикла вообще не выполняется и происходит переход на оператор, следующий за оператором цикла.

Пример:

```
y:=1;
pr:=1;
while y<=20 do
begin
   pr:=pr*y;
   y:=y+1;
end;
```

Оператор доступа (**with)** служит для быстрой и удобной работы с составными частями объектов, например с полями записей. Работа с оператором *with* показана в разделе описания типа данных **record**.

Операторы ввода-вывода

Работа с файлами (считывание из них данных, запись, поиск и другие операции) может быть организована двумя способами.

Первый способ заключается в использовании стандартных подпрограмм, позволяющих записывать содержимое переменных в файлы и считывать их обратно в переменные.

Перед тем как выполнить **операции ввода-вывода**, программе надо сообщить, где расположен файл. Для этого файловая переменная должна быть связана с именем файла с помощью процедуры `AssignFile`:

```
AssignFile(fp,'c:\result.dat');
```

где `fp` – имя файловой переменной; `'c:\result.dat'` – строка (или переменная строкового типа), содержащая название файла.

Для открытия файла в режиме записи используется процедура `Rewrite(fp)`; в режиме чтения – `Reset(fp)`.

При завершении работы с файлом его надо закрыть. Это выполняется вызовом процедуры `CloseFile(fp)`.

Операторы чтения **read** и **readln** обеспечивают ввод числовых данных, символов, строки и т.д. для последующей их обработки программой.

Формат:

`read(fp,X1,X2,...,Xn)` или `readln(fp,X1,X2,...,Xn)`, где `X1,...,Xn` – переменные; `fp` – имя файловой переменной.

Отличие оператора *readln* заключается в том, что после считывания последнего в списке значения для одного оператора *readln* данные для следующего оператора *readln* будут считываться с начала новой строки.

С помощью операторов вывода **write** и **writeln** производят вывод числовых данных, символов, строк и булевских значений.

Формат:

`write(fp,Y1,Y2,...,Yn)` или `writeln(fp,Y1,Y2,...,Yn)`, где `Y1,...,Yn` – переменные; `fp` – имя файловой переменной.

Оператор записи *writeln* аналогичен оператору *write*, но после вывода последнего в списке значения для следующего оператора *writeln* происходит перевод курсора к началу следующей строки. Оператор *writeln*, записанный без параметров, вызывает перевод строки.

Пример:

```
var a,b,c,d,rez:real;
var fp:TextFile;
    StrFileName:string;
begin
  StrFileName:='c:\test.txt';
  AssignFile(fp,StrFileName);
  Reset(fp);
  read(fp,a,b);
  CloseFile(fp);

  rez:=a+b;
  c:=rez-a;
  d:=rez+b;

  AssignFile(fp,StrFileName);
  Rewrite(fp);
  write(fp,c,d);
  CloseFile(fp);
end.
```

При использовании операторов read, readln, write, writeln без указания файловой переменной происходит считывание значений с клавиатуры и вывод на экран, что используется только при разработке программ, работающих под управлением MS-DOS.

Второй способ заключается в объектном подходе, с помощью которого можно одинаково работать с любым внешним хранилищем данных. Для непосредственной работы с файлами многие объекты предоставляют соответствующие методы, например

`LoadFromFile(const FileName:string)` – загрузить из файла или

`SaveToFile(const FileName:string)` – сохранить в файле.

В таких методах файловая переменная не нужна, а в параметре `FileName` указывается имя файла.

Подпрограммы

Подпрограммой называется именованная логически законченная группа операторов языка, которую можно вызвать для выполнения по имени любое количество раз из различных мест программы. В общем случае подпрограмма имеет такую же структуру, как и программа. Для организации подпрограмм используются процедуры и функции.

Процедура – это независимая поименованная часть программы, предназначенная для выполнения определенных действий. Для описания подпрограмм используется зарезервированное слово **procedure**.

Функция аналогична процедуре, но имеет два отличия: функция передает в точку вызова результат своей работы, имя функции может входить в выражение как операнд. Для описания подпрограмм используется зарезервированное слово **function**.

Формат Procedure:

```
procedure<имя процедуры>(параметры);
     <раздел описаний>
begin
     <раздел операторов>
end;
```

Формат Function:

```
function <имя функции>(параметры):<тип результата>;
     <раздел описаний>
begin
     <раздел операторов>
end;
```

Все процедуры и функции подразделяются на две группы: **встроенные** и **определенные пользователем**.

Встроенные (стандартные) процедуры и **функции** являются частью языка Object Pascal и могут вызываться по имени без предварительного определения в разделе описаний.

Процедуры и функции пользователя определяются самим разработчиком программы в соответствии с синтаксисом языка, представляют собой локальный блок и требуют обязательного описания перед использованием.

Поскольку подпрограммы должны обладать определенной независимостью при использовании переменных, то при их введении возникает разделение данных и их типов на **глобальные** и **локальные**. **Глобальные** константы, типы, переменные – это те, которые объявлены в головной программе. **Локальные** – это константы, типы и переменные, существующие только внутри подпрограммы и объявленные либо в списке параметров, либо в соответствующих разделах блока описаний этой подпрограммы. При совпадении имен локальной и глобальной переменной сильнее оказывается локальное имя.

Группа параметров, перед которыми *отсутствует* зарезервированное слово **var** и за которыми следует тип, называется **параметрами-значениями**. Параметры-значения обрабатываются как локальная по отношению к процедуре или функции переменная. Изменения формальных параметров-значений не влияют на значения соответствующих фактических параметров.

Группа параметров, перед которыми следует ключевое слово **var** и за которыми следует тип, называется **параметрами-переменными**. Параметр-переменная используется в том случае, если значение должно быть передано из процедуры в вызывающий блок.

Пример:

```
program test;
var Zn1,Zn2,Zn3,Zn4:real;
procedure Sum(a,b:real; var c:real);
begin
   c:=a+b;
end;
function Proiz(a,b:real):real;
begin
   Proiz:=a*b;
end;
```

```
begin
   Zn1:=5;
   Zn2:=7;
   Sum(Zn1,Zn2,Zn3);
   Zn4:=Proiz(Zn1,Zn2);
end;
```

Параметры могут иметь любой тип, включая структурированный. Группы параметров в описании подпрограмм разделяются точкой с запятой.

Перечень основных встроенных процедур и функций приведен в прил. 4.

Классы. Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) – это методика разработки программ, в основе которой лежит понятие объекта как некоторой структуры, описывающей объект и его поведение.

Основными принципами ООП являются:

инкапсуляция – объединение данных и обрабатывающих их методов внутри класса. Это означает, что объединяются и помещаются внутри класса (инкапсулируются) поля, свойства и методы. При этом класс приобретает определенную функциональность;

наследование – порождение новых объектов-потомков от существующих объектов-родителей. При этом потомок берет от родителя все его поля, свойства и методы, которые можно использовать в неизменном виде или переопределять. Удалить какие-то элементы родителя в потомке нельзя;

полиморфизм, заключающийся в том, что методы различных объектов могут иметь одинаковые имена, но различное содержание. Это достигается переопределением родительского метода в классе-потомке. В результате родитель и потомок ведут себя по-разному.

Каждый объект имеет свои свойства, т.е. характеристики (атрибуты), методы, определяющие поведение этого объекта, и события, на которые он реагирует.

В языке *Object Pascal* имеется четкое разграничение между понятиями объекта и класса. **Класс** – это сложная структура, включающая помимо описания данных описание процедур и функций, которые могут быть выполнены над представителем класса – **объектом** (экземпляром класса). Класс имеет в своем составе поля,

свойства и методы. **Поля** класса аналогичны полям записи и служат для хранения информации об объекте. **Методами** называются процедуры и функции, предназначенные для обработки полей. **Свойства** реализуют механизм доступа к полям и занимают промежуточное положение между полями и методами. С одной стороны, свойства можно использовать как поля, присваивая им значения с помощью оператора присваивания; с другой стороны, внутри класса доступ к значениям свойств выполняется методами класса.

Описание класса имеет следующую структуру:

```
Type <имя класса> = class (<имя класса-родителя>)
private
<содержит частные описания членов класса, которые доступны только в том модуле, где данный класс описан>
protected
<содержит защищенные описания членов класса, которые доступны также внутри методов класса являющихся наследниками данного класса и описанных в других модулях>
public
<содержит общедоступные описания членов класса, которые доступны в любом месте программы, где доступен сам класс >
published
<содержит опубликованные описания членов класса, которые доступны для редактирования и изменения значений во время проектирования приложения>
end;
```

Метод, объявленный в классе, может вызываться различными способами. Вид метода определяется модификатором, который указывается в описании класса после заголовка метода и отделяется от заголовка точкой с запятой. По умолчанию все считаются статическими и вызываются как обычные подпрограммы. Примеры некоторых модификаторов:

`virtual` – виртуальный метод;
`dynamic` – динамический метод;
`override` – перекрывающий метод;
`message` –метод обработки сообщения;
`abstract` – абстрактный метод.

Объекты содержат информацию о собственном типе и наследовании, которая доступна во время выполнения. Ее важность заклю-

чается в том, что для каждого объекта можно выполнить только определенный набор операций, зависящий от типа этого объекта.

Большинству методов при вызове передается параметр `Sender`, имеющий тип `TObject`. Для выполнения операций с этим параметром его тип необходимо преобразовать к типу того объекта, для которого выполняются эти операции. Для работы с типами в языке *Object Pascal* служат операторы:

<объект> **is** <класс > – проверяет, принадлежит ли указанный объект указанному классу или одному из его потомков;

<объект> **as** <класс > –предназначен для приведения одного типа к другому. Тип объекта приводится к типу класса.

ПРИЛОЖЕНИЕ 2

Общие свойства компонентов

Многие стандартные визуальные компоненты имеют одинаковые свойства, основные из которых описаны ниже.

Свойство **Align** задает способ выравнивания компонента внутри формы или другого компонента. Может принимать одно из следующих значений:

Значение	Описание
alNone	Выравнивание не используется. Компонент располагается на том месте, куда был помещен во время создания программы. Принимается по умолчанию
alTop	Компонент перемещается в верхнюю часть формы, и его ширина становится равной ширине формы. Высота компонента не изменяется
alBottom	Компонент перемещается в нижнюю часть формы, и его ширина становится равной ширине формы. Высота компонента не изменяется
alLeft	Компонент перемещается в левую часть формы, и его высота становится равной высоте формы. Ширина компонента не изменяется
alRight	Компонент перемещается в правую часть формы, и его высота становится равной высоте формы. Ширина компонента не изменяется
alClient	Компонент занимает всю рабочую область формы

Свойство **Color** задает цвет фона формы или цвет компонента или графического объекта. Может принимать одно из следующих значений:

Значение	Цвет	Значение	Цвет
clBlack	черный	clSilver	серебряный
clMaroon	темно-красный	clRed	красный
clGreen	зеленый	clLime	ярко-зеленый
clOlive	оливковый	clYellow	желтый
clNavy	темно-синий	clBlue	голубой
clPurple	фиолетовый	clFuchsia	сиреневый
clTeal	сине-зеленый	clAqua	ярко-голубой
clGray	серый	clWhite	белый

Помимо перечисленных существует набор цветов, определяемых цветовой схемой Windows. Свойство Color может также задаваться шестнадцатеричными значениями.

Свойство **Ctl3D** задает вид компонента. Если значение этого свойства равно False, то компонент имеет двумерный вид, если True – трехмерный.

Свойство **Cursor** определяет вид курсора, который он будет иметь, находясь в активной области компонента.

Свойство **DragMode** определяет режим поддержки протокола drag-and-drop.

Свойство **Enabled** определяет активность компонента. Если это свойство имеет значение True, то компонент реагирует на сообщения от мыши, клавиатуры и таймера. В противном случае (значение False) эти сообщения игнорируются.

Свойство **Font** определяет шрифт текста, отображающегося на визуальном компоненте.

Свойство **Height** задает вертикальный размер компонента или формы.

Свойство **Hint** задает текст, который будет отображаться, если курсор находится в области компонента. Свойство ShowHint должно иметь значение true.

Свойство **Left –** задает горизонтальную координату левого угла компонента относительно формы в пикселях. Для форм это значение указывается относительно экрана.

Свойство **Name** задает имя компонента, используемое в программе.

Свойство **PopupMenu** задает название локального меню, которое будет отображаться при нажатии правой кнопки мыши. Локальное меню отображается только в случае, когда свойство AutoPopup имеет значение True или когда вызывается метод Popup.

Свойство **ReadOnly о**пределяет, разрешено ли управляющему элементу, связанному с вводом и редактированием информации, изменять находящийся в нем текст.

Свойство **TabOrder** задает порядок получения компонентами фокуса при нажатии клавиши Tab во время выполнения приложения.

Свойство **Top** задает вертикальную координату левого верхнего угла интерфейсного элемента относительно формы в пикселях. Для формы это значение указывается относительно экрана.

Свойство **Visible** определяет, видим ли компонент на экране.

Свойство **Width** задает горизонтальный размер интерфейсного элемента или формы в пикселях.

ПРИЛОЖЕНИЕ 3

Запуск Delphi в режиме консольного приложения

Консольным называется приложение, имитирующее работу в текстовом режиме. При запуске консольного приложения Windows выделяет окно как для DOS-программы. Вывод-ввод данных осуществляется с помощью процедур `read`, `readln`, `write`, `writeln`.

Для создания нового консольного приложения нобходимо

1. Запустить **Delphi** в среде Windows:

Пуск – Программы – Borland Delphi – Delphi.

2. В меню Delphi выполнить следующие действия:

File – New.

3. В открывшемся окне (рис. П.1) выбрать **Console Application**. В результате создается новый проект, состоящий из одного файла с расширением **dpr**. Этот файл и является консольной программой. Первоначально он содержит следующий код:

```
program Project2;
{$APPTYPE CONSOLE}
uses SysUtils;

begin
// Insert user code here
end.
```

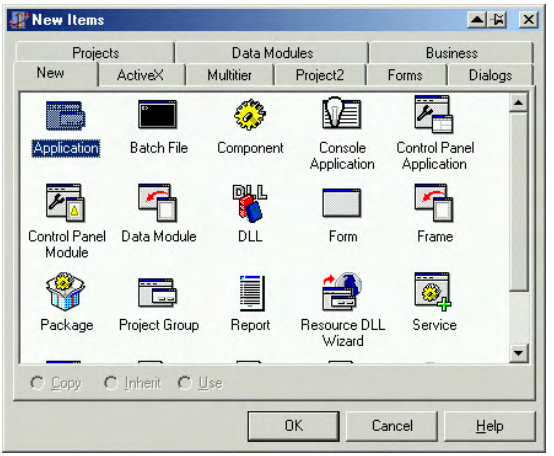

Рис. П.1

Директива {$APPTYPE CONSOLE} сообщает компилятору, что Delphi работает в консольном режиме.

Необходимый код программы, за исключением раздела описания, записывается в разделе begin...end.

ПРИЛОЖЕНИЕ 4

Перечень основных встроенных процедур и функций

Арифметические процедуры и функции

abs(x)	вычисление абсолютных величин x
arctan(x)	вычисление угла, тангенс которого равен x
cos(x), sin(x)	вычисление косинуса и синуса x
exp(x)	вычисление функции e^x
frac(x)	вычисление дробной части x
int(x)	вычисление целой части x
ln(x)	вычисление натурального логарифма x
odd(I)	возвращает true, если аргумент – нечетное число
pi	возвращает значение числа π
random	генерирует случайное число из диапазона 0...0,99. Тип результата вещественный
random(I)	генерирует значение случайного числа из диапазона 0...I
randomize	процедура для загрузки новой базы в генератор случайных чисел
sqr(x)	возведение в квадрат значения x
sqrt(x)	вычисление квадратного корня из x

x – целочисленные и вещественные типы;

I – целочисленные типы.

Функции преобразования типов

function round (x:extended):integer	возвращает значение x, определенное до ближайшего целого числа
function trunc (x:extended):integer	возвращает ближайшее целое число, меньшее или равное x, если x >= 0, и большее или равное x, если x < 0

Процедуры и функции для работы со строковыми переменными

function Copy(s:string; index, count: integer):string	выделяет из строки s подстроку длиной count, начиная с символа в позиции index
function Length(s:string):integer	возвращает текущую длину строки s
function Concat(s1,s2,..,sn:string):string	возвращает строку, представляющую собой сцепление строк s1, s2, ..., sn
function Pos(s1,s2:string):integer	определяет первое появление в строке s2 подстроки s1. Результат равен номеру позиции
procedure Delete(s:string; poz, n:integer)	удаляет n символов строки s начиная с позиции poz
procedure Insert(s1,s2:string; poz:integer)	вставляет строку s1 в строку s2 начиная с позиции poz

Функции преобразования типов

function StrToInt(s:string):integer	преобразует строку s в целое число
function IntToStr(I:integer):string	преобразует значение целочисленного выражения I в строку
function StrToFloat(s:string):extended	преобразует строку s в вещественное число
function FloatToStr(x:extended):string	преобразует значение вещественного выражения x в строку
function FloatToStrF(Value:Extended; Format: TFloatFormat; Precision, Digits:Integer):string	преобразует значение вещественного выражения x в строку с учетом параметров Precision и Digits
Format	форматы изображения числа
ffExponent	научный формат
ffFixed	формат с десятичной точкой
ffGeneral	общий цифровой формат
ffNumber	числовой формат
ffCurrecy	денежный формат

Содержание

Предисловие. 3
Лабораторная работа № 22.
Интегрированная среда Delphi. Разработка приложений
в Delphi. 4
Лабораторная работа № 23.
Программирование линейных и разветвляющихся
алгоритмов. 12
Лабораторная работа № 24.
Программирование циклических алгоритмов. использование
подпрограмм. 24
Лабораторная работа № 25.
Использование визуальных компонентов
для программирования массивов. 36
Лабораторная работа № 26.
Построение двумерных графиков в Delphi. 44
Лабораторная работа № 27.
Программирование с использованием записей и файлов.
Применение развитых элементов интерфейса
при разработке приложений. 53
Лабораторная работа № 28.
Использование средств Delphi для работы с локальными
базами данных. 68
Приложения 85
Приложение 1. Основные понятия языка программирования
Object Pascal. 85
Приложение 2. Общие свойства компонентов. 108
Приложение 3. Запуск Delphi в режиме консольного
приложения. 110
Приложение 4. Перечень основных встроенных процедур
и функций. 111

Учебное издание

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по информатике
для студентов инженерных специальностей
приборостроительного факультета

В трех частях

Часть 3

Составители: СТРЕЛЮХИН Александр Владиславович
БОКУТЬ Людмила Валентиновна
ВИШНЕВСКАЯ Ольга Геннадьевна

Под общей редакцией В.А.Нифагина

Редактор Е.И.Кортель. Корректор М.П.Антонова
Компьютерная верстка Н.А.Школьниковой
Подписано в печать 03.05.2004.
Формат 60x84 1/16. Бумага типографская № 2.
Печать офсетная. Гарнитура Таймс.
Усл. печ. л. 6,6. Уч.-изд. л. 5,2. Тираж 200. Заказ 35.
Издатель и полиграфическое исполнение:
Белорусский национальный технический университет.
Лицензия № 02330/0056957 от 01.04.2004.
220013, Минск, проспект Ф.Скорины, 65.