Министерство образования Республики Беларусь
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Инженерная математика»

ИНФОРМАТИКА

Лабораторный практикум

для студентов инженерных специальностей

приборостроительного факультета

В 2 частях

Часть 2

Учебное электронное издание

М и н с к 2 0 1 0

УДК 004(076.5)
ББК 32.97 я 7
И 74

С о с т а в и т е л и :
О.В. Дубровина, Н.К. Прихач, А.С. Гусейнова, И.В. Васильев, А.В. Стрелюхин, Л.В. Бокуть, О.Г. Вишневская

Под общей редакцией В.А. Нифагина

Р е ц е н з е н т ы :
В.И. Юринок, доцент кафедры «Математика № 1» БНТУ, кандидат технических наук;

И.Р. Лукьянович, доцент кафедры «Информационные технологии» БГУ, кандидат технических наук

Лабораторный практикум включает разделы по основам программирования в среде Delphi, предусмотренные программой курса информатики для студентов приборостроительного факультета. Каждая работа практикума четко структурирована, содержит теоретические сведения, порядок выполнения работы, контрольные вопросы и варианты заданий, что позволяет часть работы выполнять студентами под руководством преподавателя, а другая часть предназначена для самостоятельной работы. Это позволяет привить устойчивые навыки программирования.

Белорусский национальный технический университет
пр-т Независимости, 65, г. Минск, Республика Беларусь
Тел.(017)292-77-52 факс (017)292-91-37
E-mail: emd@bntu.by
http://www.bntu.by/ru/struktura/facult/psf/chairs/im/
Регистрационный № БНТУ/ПСФ85-7.2010

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Лабораторный практикум содержит методические материалы по объектно-ориентированному программированию визуальной среде Delphi, версий 4-7.

Описаны интерфейс системы Delphi, состав и характеристика элементов проекта приложения, приемы программирования на языке Delphi. Рассматриваются визуальные компоненты, используемые для создания интерфейса приложений; компоненты и техника работы с текстовой информацией, а также формами. Описываются развитые элементы для ввода и вывода информации, использование графики, работа с файлами и каталогами. Даются понятия, связанные с реляционными базами данных. Рассматривается создание приложений с базами данных.

Сформулирована методика построения прикладных программ, реализующих текстовые и графические компоненты, работу с файлами и базами данных, разработку алгоритмов сортировки. Рассмотрены структура файла проекта, интерфейс формы приложения, объектно-ориентированный подход при конструировании приложений, основные операторы языка и их использование, визуальные и невизуальные компоненты среды. Разобраны компоненты, свойства, методы и события, используемые при разработке различных программ. Уделяется внимание работе с базами данных с применением технологий BDE. Практикум снабжен приложениями, которые содержат описание базовых компонентов и часто используемых функций. Включены многочисленные примеры иллюстрирующие работу с используемыми структурами языка и компонентами.

Методическое пособие рассчитано на студентов первых курсов общетехнических специальностей приборостроительного профиля, владеющих только основами какого-нибудь языка программирования, которые будут применять среду программирования Delphi при создании конкретных приложений в прикладных областях. Может использоваться в качестве учебного пособия в ВУЗах, а также как пособие для самообучения. Благодаря большому количеству затронутых тем, подробному изложению и многочисленным примерам пособие будет полезно пользователям различного уровня.

Л а б о р а т о р н а я р а б о т а № 1

ИНТЕГРИРОВАННАЯ СРЕДА ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ BORLAND DELPHI. РАЗРАБОТКА ПРИЛОЖЕНИЙ

Цель работы: изучение среды Borland Delphi.

Используемые программные средства: Borland Delphi**.**

1.1. 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, основные элементы которого приведены в Приложении А.

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

Интегрированная среда разработки Delphi представляет собой много-оконную систему. После загрузки интерфейс Delphi имеет 4 окна (рис. 1.1).

При запуске Delphi автоматически создается новый проект Project1. 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. **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.

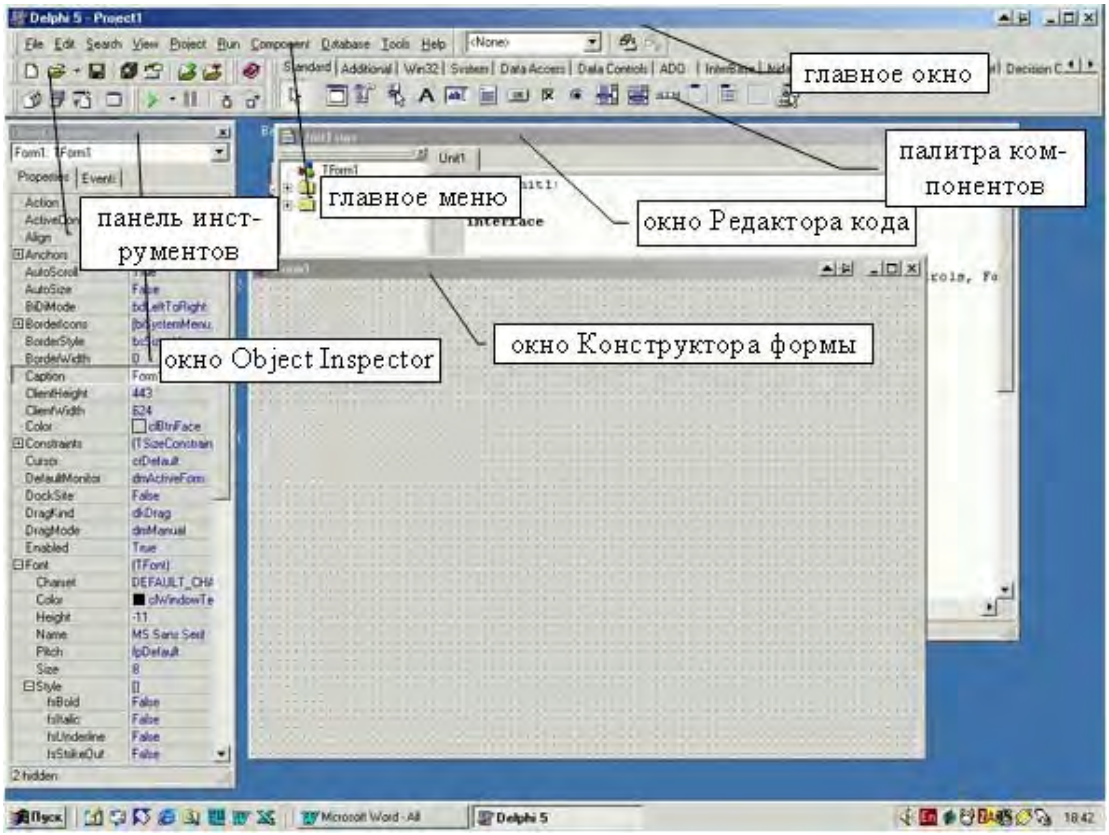

Рис. 1.1. Интерфейс Delphi

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 компонентов перед их выбором 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 Приложении В.

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. 1.1.

Таблица 1.1

Свойства компонента `Form`

`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
`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`	определятся 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** (Инспектора 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. 1.2).

Таблица 1.2

Название файлов	Расширения 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. 1.2.

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.

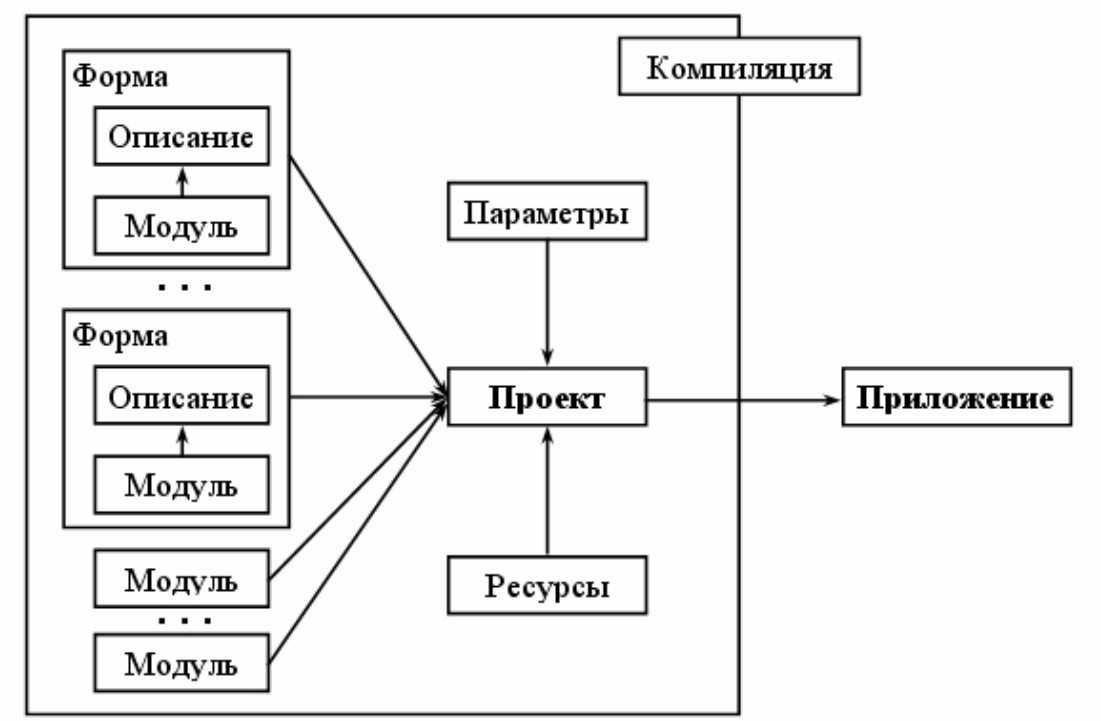

Рис. 1.2. Взаимосвязи между файлами проекта

Кроме модулей, входящих в состав форм, можно использовать модули, не связанные ни с какой 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 и т.д., 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** или 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 текстом (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**: щелчок мыши на кнопке, выбор пункта меню, нажатие клавиши, изменение размеров окна – 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.):

- `onClick` – событие нажатия, возникающее при щелчке (левой) кнопкой мыши на компоненте или при других способах нажатия на управляющий элемент;
- `onMouseDown` – событие, возникающее при нажатии любой кнопки мыши;
- `onMouseUp` – событие, возникающее при отпускании кнопки мыши;
- `onDblClick` – событие, возникающее при двойном нажатии (левой) кнопки мыши;
- `onKeyDown` – событие, возникающее при нажатии клавиши на клавиатуре;
- `onKeyUp` – событие, возникающее при отпускании клавиши на клавиатуре;
- `onCreate` – событие, возникающее один раз при создании формы;
- `onClose` – событие, возникающее 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 операционной системы MSDOS, так называемые *консольные приложения*. Запуск Delphi в режиме консольного приложения приведен в Приложении Г.

1.2. Порядок выполнения работы

Изучить структуру интегрированной среды Delphi, свойства основного компонента `Form` и выполнить контрольные примеры.

Контрольный пример 1.1

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 т.д.

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
```

Контрольный пример 1.2

Составить программу для вычисления площади круга произвольного радиуса.

Решение

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

2. Поместить на форму четыре компонента: `Label1`, `Label2`, `Edit1` и `Button1` (рис. 1.3).

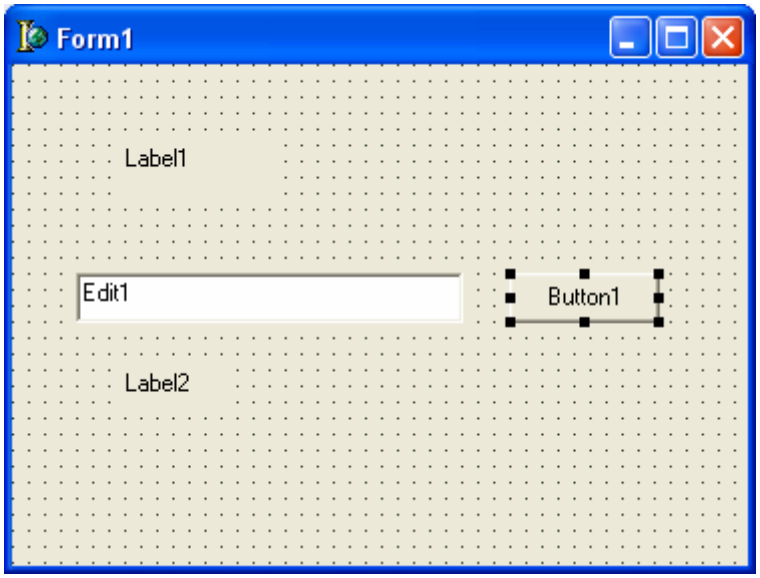

Рис. 1.3. Вид формы для контрольного примера 1.2

При помещении компонентов на форму можно сразу же задавать их размеры. Для этого после выбора компонента в *Палитре компонентов* следует указать на форме прямоугольную область, которую займет компонент. Левый угол области определяется щелчком левой клавиши мыши, затем, не отпуская клавиши мыши, нужно переместить указатель мыши в правый нижний угол области.

3. В *Object Inspector* изменить свойство `Caption` компонента `Form1` с 'Form1' на «Вычисление площади круга».

4. Изменить размеры формы. Для этого можно подвести курсор мыши к любому краю формы и, не отпуская левой клавиши мыши, изменить размеры формы. При этом автоматически будут изменяться свойства `Height` (`Высота`) и `Width` (`Ширина`), находящиеся в *Инспекторе объектов*. Положить указанные свойства равными соответственно 350 и 400 (пикселей).

Свойства Left и Top задают расстояние от левого верхнего угла монитора до левого верхнего угла формы соответственно. Установить их значения равными 300 и 200 (пикселей) соответственно. В результате форма будет находиться примерно на середине экрана.

5. Установить следующие свойства компонентов Label1 и Label2:

	Label1	Label2
Height	57	Любое число
Width	129	Любое число
Left	131	100
Top	34	218

Как и для формы, эти значения можно установить, перемещая компоненты по форме и изменяя их размеры при помощи мыши, либо прибегнув к помощи *Инспектора объектов*. Кроме того, можно воспользоваться панелью инструментов Align.

Свойство Caption является главным для метки и содержит отображаемый ею текст. Для метки Label1 установить свойство Caption равным «Введите радиус круга и нажмите кнопку Счет». Для метки Label2 свойство Caption будет определяться на этапе выполнения программы.

Свойство AutoSize метки определяет, будет ли размер метки устанавливать автоматически, в зависимости от длины символьной строки, помещенной туда.

Свойство WordWrap (Перенос слов) разрешает, либо запрещает перенос слов, если строка не помещается в метку и свойство AutoSize равно False.

Свойство Aligment (Выравнивание) определяет, как будет выровнен текст внутри метки: по левому краю, по центру или по правому краю.

	Label1	Label2
AutoSize	False	True
WordWrap	True	False
Aligment	taCenter	taLeftJustify

Для того чтобы установить характеристики шрифта, выбрать свойство Font в Инспекторе объектов. Щелкнуть по кнопке с тремя точками, появившейся в правой колонке. На экране появится окно «Выбор шрифта».

При помощи этого окна установить следующие характеристики шрифтов для меток Label1 и Label2:

	Label1	Label2
Шрифт	Times New Roman	Arial
Начертание	Полужирный	Курсив
Размер	10	11

6. Установить в Инспекторе объектов для компонентов Edit1 и Button1 следующие значения свойств Height, Width, Left и Top:

	Edit1	Button1
Height	21	25
Width	193	75
Left	31	285
Top	146	146

Свойство Text является основным для компонента Edit1 и предназначено для ввода (или вывода) символьных строк. Задать в качестве значения этого свойства пустую строку.

Для компонента Button1 в качестве значения свойства Caption положить символьную строку **Счет**.

7. Все использующиеся в программе компоненты обладают именами, задаваемыми в имеющемся у них свойстве Name (Имя). Имена компонентов генерируются автоматически интегрированной средой Delphi при создании компонента, например при размещении компонента на форме. Для образования имени компонента используется имя класса с отброшенной первой буквой T. В конце имени добавляется цифра, указывающая под каким порядковым номером в своем классе появился на свет компонент.

8. Чтобы создать заготовку обработчика события `OnClick`, необходимо выполнить двойной щелчок мышью по кнопке `Button1`. В результате окно редактора кода станет активным и будет содержать заготовку обработчика события.

Добавить в заготовку код для вычисления площади круга:

```
procedure TForm1.Button1Click(Sender: TObject);
 var r,s:real;
begin
 r:=StrToFloat(Edit1.Text);
 s:=pi*sqr(r);
 Label2.Caption:='Площадь круга равна' +
          FloatToStrF(s,ffGeneral,7,2);
end;
```

9. Перед запуском программы ее необходимо сохранить. Для этого выполнить команду главного меню: **File – Save All**.

10. Сохранив проект, запустить его на выполнение. В случае отсутствия ошибок, на экране монитора появится окно программы (рис. 1.4).

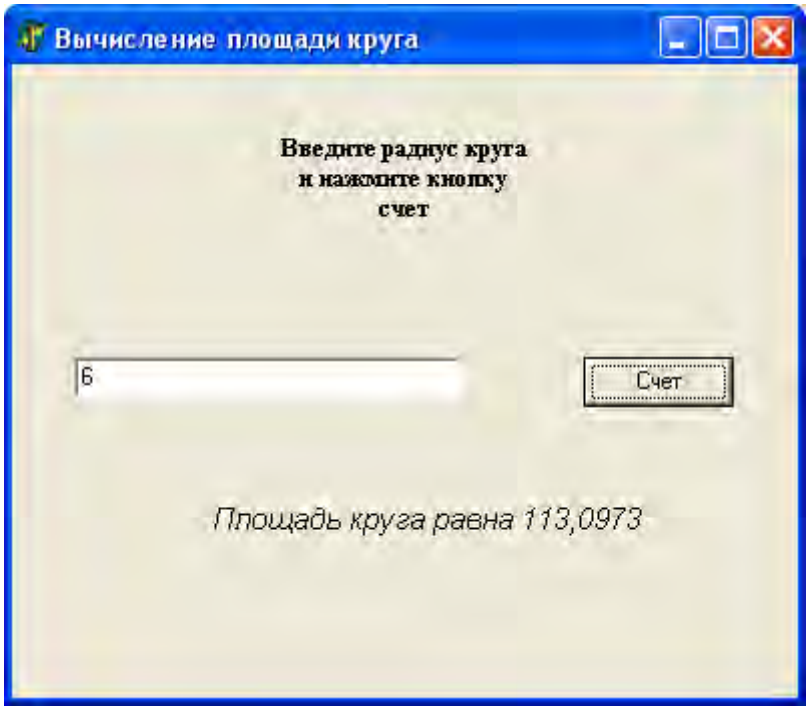

Рис. 1.4. Результат выполнения программы для контрольного примера 1.2

1.3. Содержание отчета

1. Краткий обзор по теоретической части.

2. Ответы на контрольные вопросы.

1.4. 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 №2
ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ И РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

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

Используемые программные средства: Borland Delphi.

2.1. Теоретические сведения

Линейными называются алгоритмы, в которых команды выполняются в последовательном порядке, т.е. одна за одной. Для их программирования используются операторы присваивания. Если в программе предусматривается проверка некоторых 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`) (Приложение А).

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` приведены в табл. 2.1.

Таблица 2.1

Основные свойства компонента `Label`

Свойство	Описание
`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 теста 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 переменой. Это можно сделать с помощью функции

```
FloatToStr(Value: real):string;
```

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

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 надпись, поясняющая назначение кнопки (свойство `Caption`).

Основным для кнопок являются 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 редактор **StringListEditor**. Для доступа к отдельному переключателю используется свойство `ItemIndex` типа `integer`, содержащее номер переключателя. Количество столбцов для вывода информации определяется свойством `Columns`.

2.2. 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 2.1

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`.

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 два раза щелкнуть левой кнопкой мыши на кнопке `Button1`. В результате в окне редактора кода появится 'текст-заготовка' для процедуры `Button1Click`. Операторы процедуры записать между `begin…end`, а описание переменных-идентификаторов записать 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 из строкового типа в вещественный}
   x:=StrToFloat(Edit2.Text);
    // 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 компонент Edit3 }
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 Приложении Б.

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. 2.1.

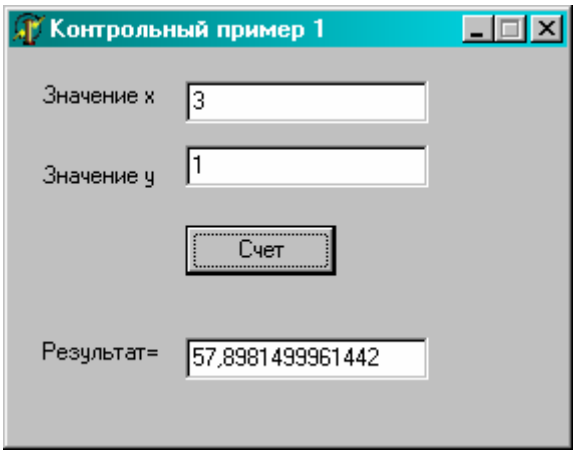

Рис. 2.1. Результат выполнения программы
для контрольного примера 2.1

Контрольный пример 2.2

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

$$f(x)=\begin{cases}\dfrac{1}{x+3} & x<-2,\\ 2\cdot x^3 & -2\le x\le 2,\\ \lg(x)+e^x & x>2.\end{cases}$$

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`. Установить с помощью *Object Inspector* для них следующие свойства:

```
Label1.Caption = 'Значение x ='
Label2.Caption = 'Результат ='
Edit1.Text = ''
Edit2.Text = ''
Button1.Caption = 'Вычислить'.
```

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

```
procedure TForm1.Button1Click(Sender: TObject);
var x,y:extended;
begin
x:=StrToFloat(edit1.Text);
if x<-2 then y:=1/(x+3)
        else if (x>=-2) and (x<=2) then y:=2*x*sqr(x)
                                   else y:=ln(x)/ln(10)+exp(x);

edit2.Text:=FloatToStrF(y,fffixed,9,4);
```

```
  end
end;
```

5. Запустить проект на компиляцию и выполнение.

6. Результат выполнения программы показан на рис. 2.2.

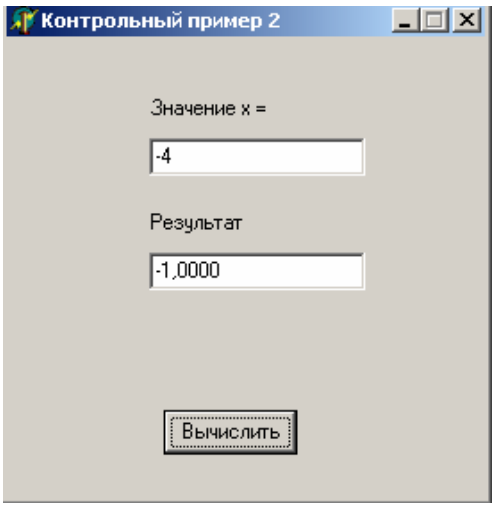

Рис. 2.2. Результат выполнения программы
для контрольного примера 2.2

Условный оператор может применяться для отслеживания в программе так называемых исключительных ситуаций: например, деление на ноль. В нашем примере при $x = -3$ возникает такая ситуация. Если мы попробуем ввести значение $x = -3$, программа выдаст сообщение об ошибке (рис. 2.3).

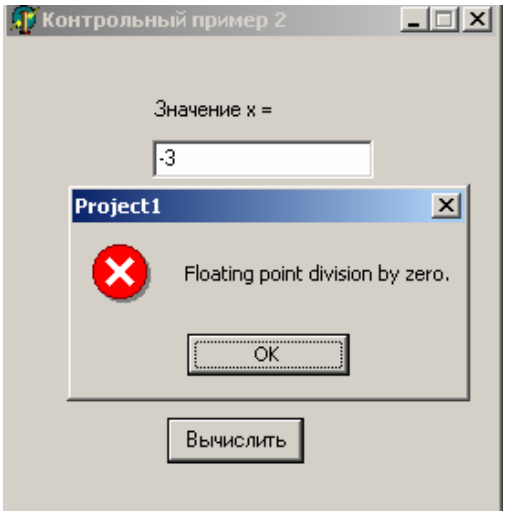

Рис. 2.3. Сообщение об ошибке

Чтобы избежать такой ситуации необходимо, чтобы перед вычислением значения функции осуществлялась проверка равенства нулю делителя в выражении $\frac{1}{x+3}$. Вставим в процедуру обработчика события `TForm1.Button1Click` оператор

```
if x= - 3 then begin
edit2.Text:= ' На ноль делить нельзя!';
exit;
end;
```

После запуска программы на выполнение, получим

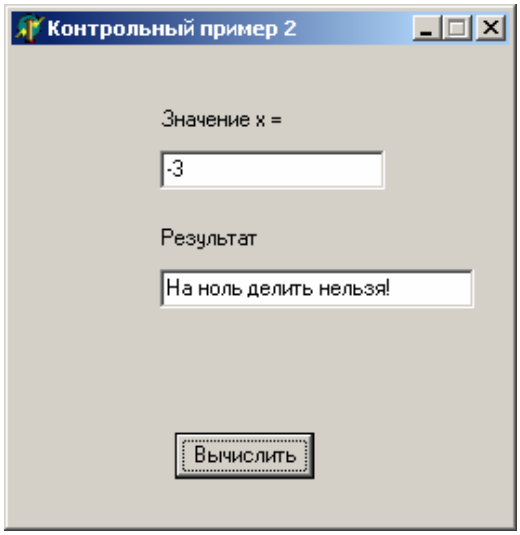

Рис. 2.4. Результат выполнения программы
для контрольного примера 2.2 при исключительной ситуации

Отредактируем текст модуля таким образом, чтобы перед выполнением вычислений выполнялась проверка, задано ли значение. Если значение не задано, то следует вывести сообщение об этом в отдельном окне. Поместим перед оператором присваивания `x:=StrToFloat (edit1.Text);` строку:

```
if Edit1.Text <> ' ' then begin
```

которая проверяет значения свойств `Edit1.Text`. Если эти значения не пустые, то вычисляется значение функции, в противном случае управление передается следующему фрагменту программы, который нужно вставить перед последним оператором `end;`

```
else ShowMessage('Не заданы значения!');
```

Примечание. Для вывода текста в отдельном окне сообщения использовалась процедура `ShowMessage('const Msg: string')`, которая отображает модальное окно сообщения с кнопкой *OK* (рис. 2.5).

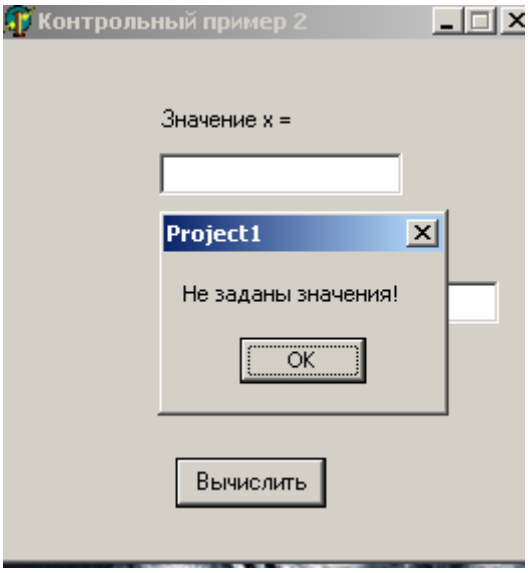

Рис. 2.5. Модальное окно сообщения с кнопкой *OK*

```
procedure TForm1.Button1Click(Sender: TObject);
var x,y:extended;
begin
  if Edit1.Text <> '' then begin
  x:=StrToFloat(edit1.Text);
  if x=-3 then begin
               edit2.Text:='На ноль делить нельзя!';
               exit;
               end;
  if x<-2 then y:=1/(x+3)
          else if (x>=-2) and (x<=2) then y:=2*x*sqr(x)
                                     else
                                  y:=ln(x)/ln(10)+exp(x);
  edit2.Text:=FloatToStrF(y,fffixed,9,4);
  end
  else ShowMessage('Не заданы значения!');
end;
```

Сохраним, откомпилируем и запустим приложение на выполнение.

Контрольный пример 2.3

Создать приложение, обеспечивающее ввод двух целых чисел и выполнение над ними арифметических операций сложения, вычитания, умножения и вещественного деления. Для выбора операции использовать переключатели, вывод сообщения об ошибке при вводе делителя, равного нулю, выполнить в отдельном окне сообщений.

Решение.

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

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

```
Form1.Height = 316
Form1.Width = 528
Form1.BorderIcons
   biMaximize = false
Form1.BorderStyle = bsSingle
Form1.Position = poScreenCenter
Form1.Caption = 'Калькулятор'.
```

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

```
Label1.Caption = 'Операнд'
Label2.Caption = 'Операнд'
Label3.Caption = 'Вычислить'
Edit1.Text = ''
Edit2.Text = ''
Edit3.Text = ''
Button1.Caption = 'Вычислить'
RadioGroup1.Caption= 'Операция'.
```

Так как количество переключателей в группе и надписи около них определяются свойством `Items`, выберем в Инспекторе объектов компонент `RadioGroup1`, а на странице свойств – свойство `Items` (Список элементов). В окне **String List Editor** введем список элементов – символов арифметических операций: +, -, *, / (рис. 2.6).

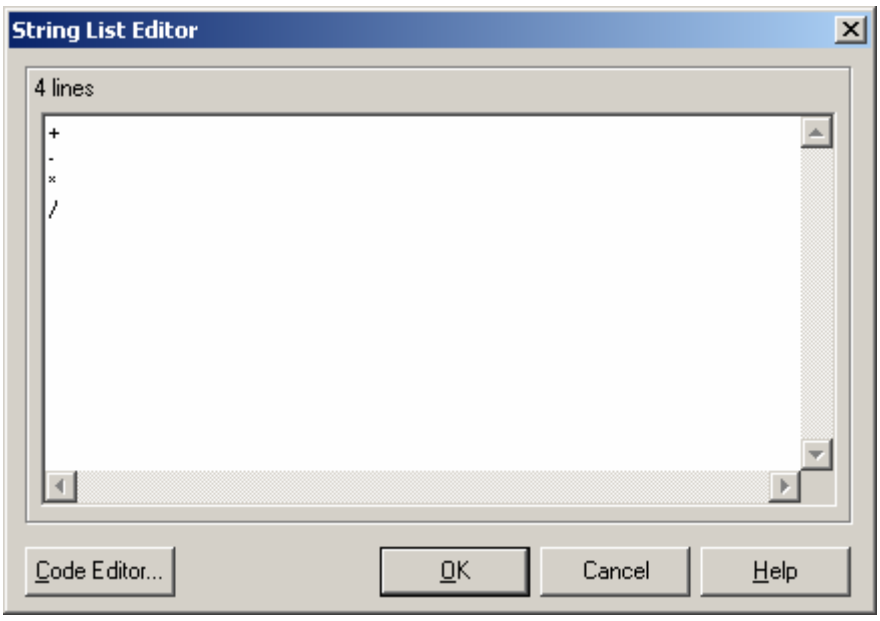

Рис. 2.6. Список элементов в окне **String List Editor**

В окне Инспектора объектов зададим для свойства `RadioGroup1.ItemIndex` значение 0, чтобы сделать первую кнопку выбранной по умолчанию. Зададим размер символов компонента `RadioGroup`, установив для свойства `RadioGroup1.Font.Size` значение 12 пунктов.

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

```
// Обработчик события щелчка на кнопке  Button1
procedure TForm1.Button1Click(Sender: TObject);
var
 a,b:integer;  // 2 операнда – целое число
 c:extended;  // результат арифметических операций
begin
  if (Edit1.Text <> '') and (Edit2.Text <> '') then
   begin
   a:=StrToInt(Edit1.Text); // если значения заданы
   b:=StrToInt(Edit2.Text); {преобразование текстовой  строки в
целое число }
   Edit3.Text:=''; {очистить от результата предыдущих
вычислений}
   {выбор  операции  в  зависимости  от  значения  свойства
RadioGroup1.ItemIndex}
   Case RadioGroup1.ItemIndex of
      0: c:=a+b;                  // сложение
```

```
      1: c:=a-b;                    // вычитание
      2: c:=a*b;                   // умножение
      3: if b=0 then begin
                ShowMessage('На ноль делить нельзя!');
                exit; end
                else c:=a/b;     // деление
    end;
   // вывод результата операций
   Edit3.Text:=FloatToStrF(c,ffGeneral,10,4);
   end
   else ShowMessage('Не заданы значения!');
end;
```

5. Запустить проект на компиляцию и выполнение.

6. Результат выполнения программы показан на рис. 2.7.

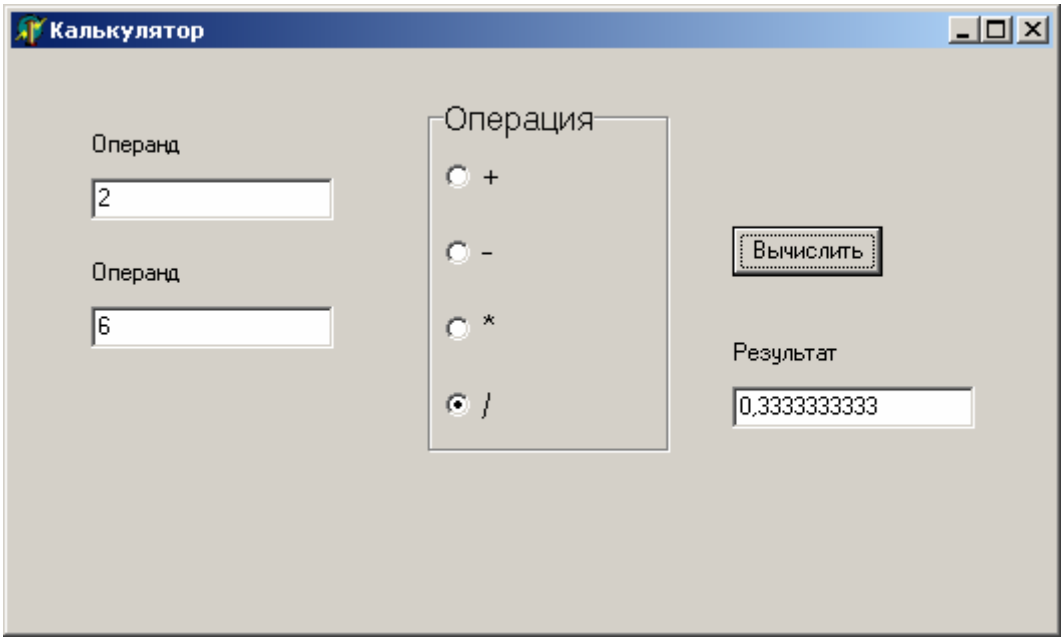

Рис. 2.7. Результат выполнения программы
для контрольного примера 2.3

2.3. Содержание отчета

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

2.4. Контрольные вопросы

1. Что такое оператор? Чем различаются простые и структурные операторы?
2. Опишите оператор присваивания, его назначение и порядок выполнения.

3. Что представляет собой составной оператор? Как ограничиваются операторы, объединенные в составной оператор?

4. Какие операторы используются для организации линейных разветвляющихся алгоритмов и какова их структура?

5. Какие компоненты используются в Delphi для организации разветвлений?

6. Перечислите основные свойства компонентов `Edit`, `CheckBox`, `Button` и `RadioGroup`?

7. Укажите назначение кнопок в Delphi.

2.5. Варианты заданий

Вариант 1

Задание 1. Составить программу для вычисления функции

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

Значения a, b и x вводятся с клавиатуры.

Задание 2. Составить программу для вычисления функции

$$y(x) = \begin{cases} \dfrac{1}{\sin(x) + a}, & x < 0 \\ \lg(x) + \exp(b \cdot x) & 0 \le x \le 2\,. \\ tg(8 \cdot x) + \sqrt{a - bx} & x > 2 \end{cases}$$

Значения a, b и x вводятся с клавиатуры.

Задание 3. По дате (месяц и день рождения) определите знак зодиака.

21.03 – 20.04: Овен; *21.04 – 20.05: Телец;* *21.05 – 21.06: Близнецы;*

22.06 – 22.07: Рак; *23.07 – 23.08: Лев;* *24.08 – 23.09: Дева;*

24.09 – 23.10: Весы; *24.10 – 22.11: Скорпион;* *23.11 – 21.12: Стрелец;*

22.12 – 20.01: Козерог; *21.01 – 19.02: Водолей;* *20.02 – 20.03: Рыбы.*

Использовать оператор выбора `case`.

Вариант 2

Задание 1. Составить программу для вычисления функции

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

Значения a, b и x вводятся с клавиатуры.

Задание 2. Составить программу для вычисления функции

$$y(x) = \begin{cases} e^x + \dfrac{1}{x+1}, & -2 \le x < 3 \\ \sqrt{x} + b \cdot sh(x) & 3 \le x \le 5 \\ |b| + \dfrac{1}{x^3} + a \cdot x & x > 5 \end{cases}$$

или выведите сообщение: «Функция не определена в данной точке». Значения a, b и x вводятся с клавиатуры.

Задание 3. По номеру n ($n > 0$) некоторого года определить c – номер его столетия (учесть, что, к примеру, началом XX столетия был 1901, а не 1900 год!). Использовать оператор выбора `case`.

Вариант 3

Задание 1. Составить программу для вычисления функции

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

Значения a, b и x вводятся с клавиатуры.

Задание 2. Составить программу для вычисления функции

$$y(x) = \begin{cases} \ln(x^2) - e^{ax} + \dfrac{1}{x} & -1 \le x < 1 \\ tg(a - b \cdot x^2) - b \cdot |x| & x < -1 \\ arctg(a \cdot x) + \sqrt{a + b \cdot x} & x > 1 \end{cases}$$

или выведите сообщение: «Функция не определена в данной точке». Значения a, b и x вводятся с клавиатуры.

Задание 3. Составьте программу вычисления по заданному радиусу и значению переменной k площади круга (если k = 1), длины окружности (если k = 2) или объема шара (если k = 3). Использовать оператор выбора `case`.

Вариант 4

Задание 1. Составить программу для вычисления функции

$$f = 2xb\left(x^2+1\right)+\sqrt{\mathrm{tg}^2 a + 0.7\cdot 10^{-3}}\,.$$

Значения a, b и x вводятся с клавиатуры.

Задание 2. Составить программу для вычисления функции

$$y(x)=\begin{cases}\pi\cdot x+\lg(a\cdot x^3) & 0\le x\le 1.5\\ 1-\ln\left(\sqrt{x^2+1}+b\right)-|a-x| & x>1.5\\ e^x+\sin(x)+\dfrac{1}{|b\cdot\cos(x)|} & x<0\end{cases}$$

или выведите сообщение: «Функция не определена в данной точке». Значения a, b и x вводятся с клавиатуры.

Задание 3. Написать программу нахождения числа дней в месяце, если даны: номер месяца n – целое число от 1 до 12; целое число a, равное единице для високосного года и равное нулю в противоположном случае. Использовать оператор выбора `case`.

Вариант 5

Задание 1. Составить программу для вычисления функции

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

Значение переменных x и y вводятся с клавиатуры.

Задание 2. Составить программу для вычисления функции

$$y(x)=\begin{cases}\pi\cdot x^2-\dfrac{1}{x-1}+\cos(a\cdot x+b) & x<1.4\\ b\cdot\sin\left(x^3-a\right)-e^{-x} & 1.4\le x\le 2\\ \ln\left(x+\sqrt{x+a}\right) & x>2\end{cases}$$

или выведите сообщение: «Функция не определена в данной точке». Значения a, b и x вводятся с клавиатуры.

Задание 3. Для целого числа *k* от 1 до 99 вывести фразу “мне *k* лет”, учитывая при этом, что при некоторых значениях *k* слово “лет” надо заменить на слово “год” или “года”. Использовать оператор выбора `case`.

Вариант 6

Задание 1. Составить программу для вычисления функции

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

Значения a, x и y вводятся с клавиатуры.

Задание 2. Составить программу для вычисления функции

$$y(x)=\begin{cases}a\cdot\lg(x)+\sqrt[3]{\sin(x)} & 0<x\le 1\\ \dfrac{2\cdot a\cdot\cos(x)}{2-x}+e^x & 1<x\le 3\\ tg(2\cdot x-1)+\sqrt{a\cdot x+b} & x>3\end{cases}$$

или выведите сообщение: «Функция не определена в данной точке». Значения a, b и x вводятся с клавиатуры.

Задание 3. Для натурального числа k напечатать фразу «мы нашли k грибов в лесу», согласовав окончание слова «гриб» с числом k. Использовать оператор выбора `case`.

Вариант 7

Задание 1. Составить программу для расчета значения

$$f = |\cos x - \cos y| \cdot \left(1 + z + x^2 + y^3\right).$$

Значения x, y и z вводятся с клавиатуры.

Задание 2. Составить программу для вычисления функции

$$y(x) = \begin{cases} ax^3 + b \cdot \ln|2 \cdot x| & x < 0.5 \\ \sqrt{a + \sin(x)} - e^{b \cdot x} & 0.5 \le x \le 1 \\ \dfrac{arctg(5 \cdot x)}{b \cdot \cos(x) + \ln(a \cdot x)} & x > 1 \end{cases}$$

или выведите сообщение: «Функция не определена в данной точке». Значения a, b и x вводятся с клавиатуры.

Задание 3. Напишите программу, которая анализирует возраст человека и относит его к одной из четырех групп: дошкольник, ученик, работник, пенсионер. Возраст вводится с клавиатуры. Использовать оператор выбора `case`.

Вариант 8

Задание 1. Составить программу для расчета значения w:

$$w = |\cos x - \cos y|^{\left(1 + 2\sin^2 y\right)} \left(1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4}\right).$$

Значения x, y и z вводятся с клавиатуры.

Задание 2. Составить программу для вычисления функции

$$y(x) = \begin{cases} \dfrac{\ln^3 x + x}{\sqrt{x+1}} & x \le 0.4 \\ \dfrac{\sqrt{x+t}}{e^x + t \cdot \sin^2 x} & 0.4 < x < 1 \\ arctg(t \cdot x) - \dfrac{1}{|x - 1.5|} & x \ge 1 \end{cases}$$

или выведите сообщение: «Функция не определена в данной точке». Значения t и x вводятся с клавиатуры.

Задание 3. В зависимости от стажа работы на предприятии введена надбавка в размере:

для работающих от 5 до 10 лет – 10%;

для работающих от 10 до 15 лет – 15%;

для работающих свыше 15 лет – 20%.

Составить программу, которая по заданному стажу работы определит размер надбавки в процентах. Использовать оператор выбора `case`.

Лабораторная работа №3

ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Цель работы: приобретение практических навыков использования операторов циклов.

Используемые программные средства: Borland Delphi.

3.1. 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` (Приложение А).

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 **StringListEditor**, который вызывается с помощью **Object Inspector**:

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

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

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), которые задаются, используя окно *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 текстовом файле. Для этого удобно использовать методы

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

и

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

класса TStrings.

3.2. Порядок выполнения работы

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

Контрольный пример 3.1

Составить программу для расчета $f(x)$. Значения N и x вводятся с клавиатуры. Предусмотреть возможность вывода значения функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h:

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

Решение.

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

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

```
Form1.Height = 302
Form1.Wight = 326
Form1.BorderIcons
```

```
biMaximize = false
Form1BorderStyle = bsSingle
Form1.Position = poScreenCenter
Form1.Caption = ‘ Контрольный пример 1а’
```

3. Рассмотрим по отдельности программирование каждого из слагаемых этого примера.

3.1. Вычисление $n!$. Вводится число n. Переменной f, предназначенной для хранения значения последовательности чисел, присваивается начальное значение, равное единице. Затем организуется цикл, параметром которого выступает переменная i. Если значение параметра цикла меньше или равно n, то выполняется оператор тела цикла, в котором из участка памяти с именем f считывается предыдущее значение произведения, умножается на текущее значение параметра цикла, а результат снова помещается в участок памяти с именем f. Когда параметр i становится больше n, цикл заканчивается.

```
f:=1;
for i:=1 to n do f:=f*i;
```

3.2. Вычисление степени a^n. Здесь a – вещественное число, которое необходимо возвести в целую положительную степень. Для того чтобы получить целую степень n числа a, нужно умножить его само на себя n раз. Результат будет храниться в участке памяти с именем st. При выполнении очередного цикла из этого участка предыдущее значение будет считываться, умножаться на основание степени a и снова записываться в участок памяти st. Цикл выполняется n раз.

```
st:=1;
for i:=1 to n do st:=st*x;
```

3.3. Суммирование. При сложении нескольких чисел необходимо накапливать результат в определенном участке памяти, каждый раз считывая оттуда предыдущее значение суммы и прибавляя к нему следующее слагаемое. Для выполнения первого оператора накапливания суммы из участка памяти необходимо взять такое число, которое не влияло бы на результат сложения. Другими словами, перед началом цикла переменной, предназначенной для накапливания суммы, необходимо присвоить нуль.

3.4. Объединив, получим:

```
st:=1; s:=0; f:=1;
for i:=1 to n do begin
   st:=st*x; f:=f*i;
   s:=s+st/f;
end;
```

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

```
Label1.Caption = 'N'
Label2.Caption = 'x'
Label3.Caption = 'Сумма'
Edit1.Text = ''
Edit2.Text = ''
Edit3.Text = ''
Button1.Caption = 'Счет'
BitBtn1.Caption:=’&Закрыть’
```

5. 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 n,i:integer;x,s,st,f:extended;
begin
 n:=StrToInt(Edit1.Text);
 x:=StrToFloat(Edit2.Text);
 st:=1; s:=0; f:=1;
 for i:=1 to n do begin
  st:=st*x; f:=f*i;
  s:=s+st/f;
 end;
 Edit3.Text:=FloatToStrF(s,ffFixed,7,3);
end;
```

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

6. Запустить проект на компиляцию и выполнение.

7. Задать значения для $N = 5$, $x = 2$ и нажать кнопку **Счет**.

8. Теперь нужно посчитать значение данной суммы на отрезке $[x_1; x_2]$.

Открыть новый проект Delphi: File – New Application .

9. 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-б'
```

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

11. 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,x,h,s,st,f:extended; n,i:integer;
stroka:string;
begin
  Memo1.Clear;
  n:=StrToInt(Edit1.Text);
  x1:=StrToFloat(Edit2.Text);
  x2:=StrToFloat(Edit3.Text);
  h:=StrToFloat(Edit4.Text);
  x:=x1;
  repeat
   st:=1; s:=0; f:=1;
   for i:=1 to n do begin
    st:=st*x; f:=f*i;
    s:=s+st/f;
   end;
```

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

12. Запустить проект на компиляцию и выполнение.

13. Задать значения для $N=10$, $x1=1$, $x2=10$, $h=0.5$ и нажать кнопку Счет. Результат выполнения программы показан на рис. 3.1:

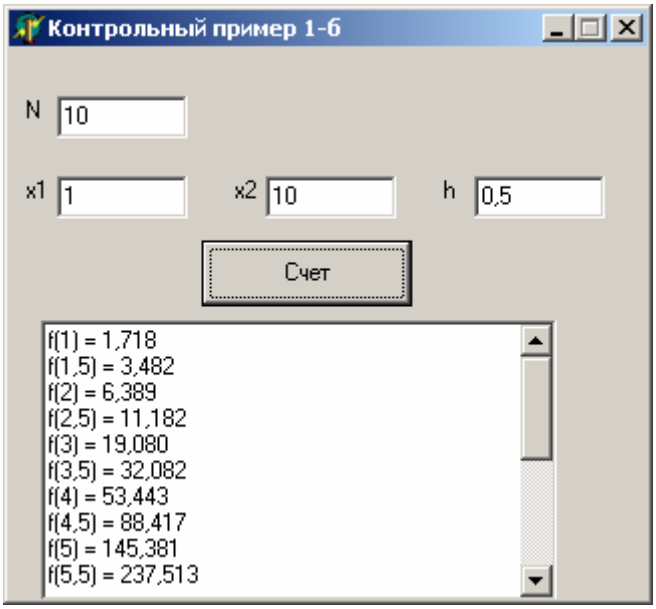

Рис. 3.1. Результат выполнения программы
для контрольного задания 3.1

Контрольный пример 3.2

Корень некоторого уравнения находится последовательными приближениями по формуле $x_{n+1}=\sqrt[4]{29+3x_n{}^2+8x_n}$. Написать программу для нахождения такого приближения корня, при котором разность по модулю между двумя соседними приближениями не превосходит 10^{-6}, а начальное приближение $x_0=1.92$. Вывести на экран корень уравнения до 5-го знака и число итераций.

Решение.

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

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

```
Form1.Height = 323
Form1.Wight = 268
Form1.BorderIcons
   biMaximize = false
Form1BorderStyle = bsSingle
Form1.Position = poScreenCenter
Form1.Caption = ‘ Последовательные приближения’.
```

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

```
Label1.Caption = 'начальное приближение'
Label2.Caption = 'решение'
Label3.Caption = 'количество итераций'
Label3.Caption = 'Проверка: подстановка в исходное уравнение'
Edit1.Text = ''
Edit2.Text = ''
Edit3.Text = ''
Edit4.Text = ''
Button1.Caption = 'Решение'
```

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

Текст соответствующей процедуры имеет вид:

```
procedure TForm1.Button1Click(Sender: TObject);
var x0,x1,eps,f:extended; n:integer;
begin
  x0:=StrToFloat(Edit1.Text);
  eps:=1E-6;
  x1:=exp(ln(29+3*sqr(x0)+8*x0)*(1/4));
  n:=1;
   while abs(x0-x1)>eps do
   begin
     x0:=x1;
     x1:=exp(ln(29+3*sqr(x0)+8*x0)*(1/4));
     n:=n+1;
   end;
  Edit2.Text:=FloatToStrF(x1,fffixed,10,4);
  Edit3.Text:=IntToStr(n-1);
  f:=sqr(sqr(x1))-3*sqr(x1)-8*x1-29;
  Edit4.Text:=FloatToStrF(f,fffixed,10,6);
```

```
end;
```

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

5. Запустить проект на компиляцию и выполнение.

6. Задать значение начального приближения и нажать кнопку **Решение**. Результат выполнения программы показан на рис. 3.2.

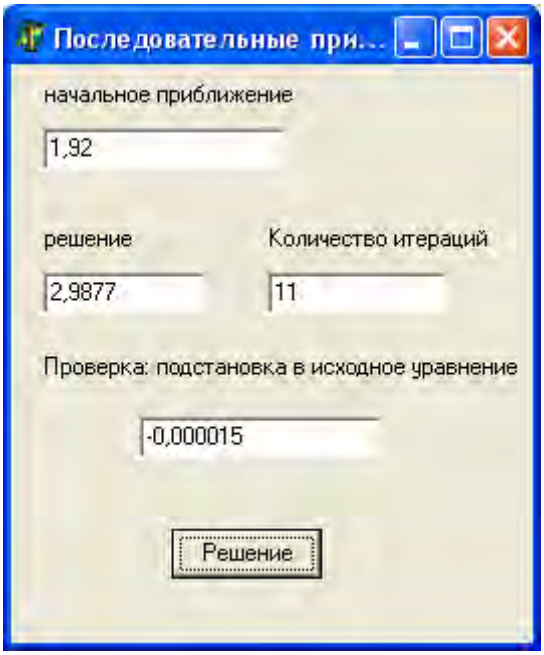

Рис. 3.2. Результат выполнения программы
для контрольного задания 3.2

3.3. Содержание отчета

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

3.4. Контрольные вопросы

1. Каково назначение операторов повторений (циклов)?

2. В чем различия операторов повтора `while` и `repeat`?

3. В каких случаях предпочтительнее использовать для организации циклов оператор повтора `for`? Что записывается в заголовке этого оператора?

4. Какие ограничения накладываются на использование параметра цикла в цикле `for`?

5. Что такое вложенные циклы? Какие дополнительные условия необходимо соблюдать при организации вложенных циклов?

6. Перечислите основные свойства компонента `Memo`.

7. Как получить доступ к заданной строке компоненте `Memo`?

8. Что такое многострочный редактор?

3.5. Варианты заданий

Вариант 1

Задание 1. Составить программу для расчета функции $f(x)=\sum_{k=1}^{N}\frac{x^k}{(k+3)!}$. Значения N и x вводятся с клавиатуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h.

Задание 2. Вычислить $x=\sqrt[3]{a}$ для заданного значения a по рекуррентному соотношению Ньютона: $x_{n+1}=\frac{1}{3}\left(x_n+2\cdot\sqrt{\frac{a}{x_n}}\right)$, $x_0=a$. Сколько итераций надо выполнить, чтобы для заданной погрешности ε выполнялось соотношение: $|x_{n+1}-x_n|<\varepsilon$?

Вариант 2

Задание 1. Составить программу для расчета $f(x)=\sum_{k=1}^{N}\frac{2\sqrt{k}x^k}{3(k+8)!}$. Значения N и x вводятся с клавиатуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h.

Задание 2. Пусть $x_1 = 0{,}3$, $x_2 = -0{,}3$, $x_i = i + \sin(x_{i-1})$, $i = 3,4,...$ Найти $x_1 \cdot x_2 \cdot x_3 \cdot \ldots \cdot x_n$. Число n вводится с клавиатуры. Элементы последовательности вывести в компонент Memo.

Вариант 3

Задание 1. Составить программу для расчета $f(x) = \sum_{k=1}^{N} \frac{5e^{kx}}{3^k + (k+1)!}$. Значения N и x вводятся с клавиатуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h.

Задание 2. Корень некоторого уравнения находится последовательными приближениями по формуле $x_{n+1} = \sqrt[3]{x_n} + 0{,}1$. Написать программу для нахождения такого приближения корня, при котором разность по модулю между двумя соседними приближениями не превосходит $\varepsilon = 10^{-4}$, а $x_0 = 1{,}1$.

Вариант 4

Задание 1. Составить программу для расчета $f(x) = \sum_{k=1}^{N} \frac{\cos(2\pi k)}{2^k + \sin(k)!}$. Значения N и x вводятся с клавиатуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h.

Задание 2. Для заданного $x > 1$ вычислить $y = \sqrt{x}$ по итерационной формуле $y_i = \frac{1}{2} \cdot \left(y_{i-1} + \frac{x}{y_{i-1}} \right)$ с заданной погрешностью ε, задав начальное приближение $y_0 = x$. Сравнить с результатом использования встроенной функции. Сколько итераций пришлось выполнить?

Вариант 5

Задание 1. Составить программу для расчета $f(x)=\sum_{k=1}^{N}\frac{\sin^2(k)}{x^k+k!}$. Значения N и x вводятся с клавиатуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h.

Задание 2. Вычислите $x_1+x_2+\ldots+x_{20}$, если последовательность $x_1, x_2, \ldots$ образована по следующему закону: $x_1=0,\quad x_2=\frac{5}{8},\quad x_i=\frac{x_{i-2}}{2}+\frac{3}{4}\cdot x_{i-1},$ $i=3,4,\ldots$ Элементы последовательности вывести в компонент `Memo`.

Вариант 6

Задание 1. Составить программу для расчета $f(x)=\sum_{k=1}^{N}\frac{k!}{k!+1}\left(\frac{x}{2}\right)^k$. Значения N и x вводятся с клавиатуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h.

Задание 2. Пусть $a_0=a_1=1;\quad a_k=a_{k-1}+\frac{a_{k-1}}{2^{k-1}}$, где $k=2,3,\ldots.$ Написать программу нахождения произведения $a_0\cdot a_1\cdot\ldots\cdot a_n$. Число n вводится с клавиатуры. Числа a_k вывести в компонент `Memo`.

Вариант 7

Задание 1. Составить программу для расчета $f(x)=\sum_{k=1}^{N}\frac{2k+1}{(2k)!}x^{2k}$. Значения N и x вводятся с клавиатуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h.

Задание 2. Дано целое $k\geq 0$. Вывести на печать k-ый член последовательности, задаваемой формулами: $x_0=1;\ x_n=n\cdot x_{n-1}+\frac{1}{n},\ n\geq 1.$

Вариант 8

Задание 1. Составить программу для расчета $f(x)=\sum_{k=1}^{N}\frac{(x\ln a)^n}{k!}$. Значения N, a и x вводятся с клавиатуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h.

Задание 2. Корень некоторого уравнения находится последовательными приближениями по формуле $x_{n+1}=\frac{2-x_n^3}{5}$. Написать программу для нахождения такого приближения корня, при котором разность по модулю между двумя соседними приближениями не превосходит 10^{-5}, а начальное приближение $x_0=1$.

Вариант 9

Задание 1. Составить программу для расчета $f(x)=\sum_{k=1}^{N}\frac{x^{2k}k^2}{(2k+1)!}$. Значения N и x вводятся с клавиатуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h.

Задание 2. Пусть дано натуральное число n. Найдите $a_1b_1+a_2b_2+\ldots+a_nb_n$, если $a_1=b_1=1$, $a_k=\frac{1}{2}\cdot\left(\sqrt{b_{k-1}}+\frac{1}{2}\cdot a_{k-1}\right)$, $b_k=2\cdot a_{k-1}^2+b_{k-1}$, $k=1,2,\ldots n$. Значения чисел a_i, b_i вывести в компонент `Memo`.

Вариант 10

Задание 1. Составить программу для расчета $f(x)=\sum_{k=1}^{N}\frac{x^k+tg(kx)}{k!}$. Значения N и x вводятся с клавиатуры. Предусмотреть возможность вывода значений функции в зависимости от x, изменяющегося в диапазоне $[x_1, x_2]$ с заданным шагом h.

Задание 2. Дано вещественное положительное число b. Последовательность $a_1, a_2, \ldots$ образована по закону: $a_1 = b, \quad a_i = a_{i-1} - \frac{1}{\sqrt{i}}, \quad i = 2,3,\ldots$ Написать программу нахождения первого отрицательного члена последовательности. Значения элементов последовательности вывести в компонент `Memo`.

Л а б о р а т о р н а я р а б о т а № 4

ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ПОДПРОГРАММ

Цель работы: приобретение практических навыков реализации процедур и функций.

Используемые программные средства: Borland Delphi.

4.1. Теоретические сведения

В практике программирования часто возникают ситуации, когда одну и ту же группу операторов, реализующих определенную цель, требуется повторить без изменений в нескольких местах программы. Для избавления от столь нерациональной траты времени была предложена концепция подпрограммы, которые в языке OP бывают двух типов: **процедуры и функции**.

Процедура – это независимая именованная часть программы, которую можно вызвать по имени для выполнения определенных действий. Структура процедуры повторяет структуру программы. Процедура не может выступать в качестве операнда в выражении. Упоминание имени процедуры в тексте программы приводит к активизации процедуры и называется ее **вызовом**.

Функция сходна с процедурой, но имеет 2 отличия: функция передает в точку вызова скалярное значение; имя функции может входить в выражение в качестве операнда.

Итак, отличие подпрограмм-процедур от подпрограмм-функций состоит в том, что процедуры служат для задания совокупности действий, направленных на изменение внешней по отношению к ним программной обстановки, а функции, являясь частным случаем процедур, обязательно возвращают в точку вызова основной программы единственный результат как значение имени этой функции.

Все процедуры и функции языка **Object Pascal** делятся на две группы: встроенные (стандартные) и определенные пользователем. Первые входят в состав языка и вызываются для выполнения по строго фиксированному имени. Вторые разрабатываются и именуются самим пользователем.

Для использования стандартной процедуры или функции к программе подключается тот или иной специализированный библиотечный модуль, в который входит данная стандартная процедура или функция, для чего имя специализированного библиотечного модуля указывается в разделе *uses*. Затем в программе осуществляется вызов процедуры или функции, для чего записывается ее имя и указываются фактические параметры, например: `Pi, Sin(x), Inc(x,5), Chr(125).` Так как после выполнения функции ее значение присваивается имени, то имя функции используется в выражении.

Если в программе возникает необходимость частого обращения к некоторой группе операторов, то рационально сгруппировать такую группу операторов в самостоятельный блок, к которому можно обращаться, указывая его имя. Такие разработанные программистом самостоятельные программные блоки называются *подпрограммами пользователя*. Они являются основами модульного программирования. Разбивая задачу на части и оформляя логически обособленные модули в виде процедур и функций, программист реализует основные принципы широко используемого в практике системного подхода и методов нисходящего проектирования.

При вызове подпрограммы, определенной программистом, работа главной программы на некоторое время приостанавливается и начинает выполняться вызванная подпрограмма. Она обрабатывает данные, переданные ей из главной программы. По завершении выполнения подпрограмма-функция возвращает главной программе результат (подпрограмма-процедура не возвращает явно результирующего значения).

Передача данных из главной программы в подпрограмму и возврат результата выполнения функции осуществляется с помощью параметров. *Параметром* называется переменная, которой присваивается некоторое значение в рамках указанного применения. Различают *формальные параметры* – параметры, определенные в заголовке подпрограммы, и *фактические параметры* – выражения, задающие конкретные значения при обращении к подпрограмме.

При обращении к подпрограмме ее формальные параметры замещаются фактическими, переданными из главной программы.

4.1.1. Процедуры

Описание процедуры состоит из заголовка процедуры и тела процедуры. Заголовок включает в себя служебное слово procedure, имя процедуры и заключенные в круглые скобки список формальных параметров *с указанием их типов*:

```
procedure <имя> (<список формальных параметров>);
```

Формальные параметры отделяются точкой с запятой (список однотипных параметров может быть перечислен через запятую). После заголовка идут разделы описаний (констант, типов, переменных, процедур и функций, используемых в процедуре) и операторы языка OP, реализующие алгоритм процедуры.

Например:

```
procedure f1 (x: real; var y: real; var c: real);
begin
    y:=sin(x)/cos(x);
    c:=ln(x)/ln(10);
end;
```

Формальные параметры *нельзя* описывать в разделе описаний процедуры.

Для обращения к процедуре необходимо использовать оператор вызова процедуры. Он имеет следующий вид:

```
<имя процедуры> (<список фактических параметров>);
```

Например:

```
f1(r, r1, r2);
```

Фактические параметры в списке отделяются друг от друга запятой. Механизм применения формальных-фактических параметров обеспечивает замену первых параметров последними, что позволяет выполнять процедуру с различными данными. Между фактическими параметрами в операторе вызова процедуры и формальными параметрами в заголовке процедуры устанавливается взаимно однозначное соответствие.

Замечание. Количество, типы и порядок следования формальных и фактических параметров должны совпадать.

4.1.2. Функции

Функция, определенная пользователем, состоит из заголовка и тела функции. Заголовок содержит зарезервированное слово function, идентификатор (имя) функции, заключенный в круглые скобки, необязательный список формальных параметров и тип возвращаемого функцией значения:

```
function <имя> (список формальных параметров): <тип результата>;
```

Тело функции представляет собой локальный блок, по структуре аналогичный программе.

Например:

```
function tan (c: real): real;
  begin
    tan:=sin(c)/ cos(c);
  end;
```

В теле функции всегда должен быть *один* оператор, присваивающий значение имени функции.

Обращение к функции осуществляется по имени с указанием списка фактических параметров. Количество, типы и порядок следования формальных и фактических параметров должны совпадать:

```
<имя функции> (<список фактических параметров>);
```

Например:

```
tg: =tan(x);
```

При использовании процедур и функций переменные объявляются несколько раз в основной программе и подпрограммах.

Переменные и типы, определенные в основной программе до объявления процедур и функций, называются *глобальными* – они доступны всем функциям и процедурам. Переменные, определенные в какой-либо подпрограмме после раздела описания процедур и функций, называются *локальными*.

Для правильного определения области действия идентификаторов (переменных) необходимо придерживаться следующих правил:

- каждая переменная должна быть описана перед тем, как она будет использована;
- областью действия переменной является та подпрограмма, в которой она будет описана;
- все переменные в подпрограммах должны быть уникальными;
- одна и та же переменная может быть по-разному определена в каждой из подпрограмм;
- если имя подпрограммы совпадает с названием стандартной подпрограммы, то последняя игнорируется, а выполняется подпрограмма пользователя;
- если внутри какой-либо процедуры встречается переменная с таким же именем, что и глобальная переменная, то внутри процедуры будет действовать локальное описание;
- каждая подпрограмма может изменить значение глобальной переменной.

4.1.3. Рекурсивные процедуры

Рекурсия (рекурсивная процедура или функция) возникает, если функция или процедура вызывает саму себя. **Прямая рекурсия** может вызывать себя непосредственно, например:

```
function Factorial (n: LongInt) : LongInt;
begin
     Factorial:=n*Factorial(n-1);
end;
```

Рекурсивная процедура также может вызывать себя **косвенно**, вызывая вторую процедуру, которая, в свою очередь, вызывает первую:

```
procedure Ping(n: Integer);
begin
    Pong(n-1);
end;

procedure Pong(n: Integer);
begin
    Ping(n div 2);
end;
```

Классическим примером рекурсии является **вычисление факториала**. **Факториал числа** n – это произведение целых чисел от 1 до n: $1 \cdot 2 \cdot 3 \cdot \ldots (n-1) \cdot n$. Приведенное выражение можно переписать следующим образом: $n! = n \cdot ((n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1) = n \cdot (n-1)!$

Рекурсивная функция вычисления факториала:

```
function Factorial (N: Integer): Integer;
begin
  if (N<=0) then
     Factorial := 1
  else
     Factorial=N*Factorial(N-1);
end;
```

Функция сначала проверяет число на условие $N \le 0$. Для чисел, меньших нуля факториал не определен, но это условие проверяется для подстраховки. Если бы функция проверила только условие равенства числа нулю, то для отрицательных чисел рекурсия была бы бесконечной.

Если входное значение меньше или равно нулю, функция возвращает значение, равное 1. Иначе значение функции равно произведению входного значения на факториал, уменьшенный на единицу.

Существуют два фактора, которые гарантируют, что эта рекурсивная функция в конце концов остановится. Во-первых, при каждом последующем вызове значение параметра N уменьшается на единицу. Во-вторых, значение N ограничено нулем. Когда значение достигает 0, функция заканчивает рекурсию. Такое условие, как $N \le 0$, которое останавливает **рекурсию, называется основным условием** или **условием остановки**.

При каждом вызове подпрограммы система сохраняет некоторые значения в **стеке** (**стек** – упорядоченный список, в котором элементы добавляются и удаляются с одного и того же конца списка). Если рекурсивная процедура вызывается много раз, она может заполнить весь стек и вызвать ошибку переполнения стека.

Количество вызовов рекурсивной функции зависит от объема памяти компьютера и количества данных, которые программа помещает в стек.

4.2. Порядок выполнения работы

Изучить операторы, используемые для организации подпрограмм, выполнить контрольные примеры и задания соответствующего варианта.

Контрольный пример 4.1

Написать программу вычисления выражения $Z = \frac{A^5 + A^{-3}}{2 \cdot A^M}$, в котором возведение в степень выполняется функцией `Step`.

Решение.

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

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

```
Form1.Height = 345
Form1.Width = 396
Form1.BorderIcons
  biMaximize = false
Form1.BorderStyle = bsSingle
Form1.Position = poScreenCenter
Form1.Caption = 'Контрольный пример 1'.
```

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

```
Label1.Caption = 'A'
Label2.Caption = 'M'
Label3.Caption = 'Z'
Edit1.Text = ''
Edit2.Text = ''
Edit3.Text = ''
Button1.Caption = 'Выполнить'.
```

4. Текст программы приведен ниже:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Edit1: TEdit;
```

```
    Label1: TLabel;
    Label2: TLabel;
    Edit2: TEdit;
    Button1: TButton;
    Label3: TLabel;
    Edit3: TEdit;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  M:integer;
  A,Z,R:real;
implementation

{$R *.dfm}
  // Функция вычисления степени.
  // N,X – формальные параметры,
  // результат, возвращаемый функцией в точку вызова
  // имеет вещественный тип.
  function Step(N:integer; X:real):real;
   var i:integer; y:real;
  begin
   y:=1;
   for i:=1 to N do
    y:=y*x;
    step:=y; // присваивание функции результата
             // вычисления степени
   end; // Step

procedure TForm1.Button1Click(Sender: TObject);
begin
// ввод значения числа A и показателя степени M
 A:=StrToFloat(Edit1.Text);
 M:=StrToInt(Edit2.Text);
 // Вызов функции с передачей ей фактических параметров
 Z:=Step(5,A);
 Z:= Z+Step(3,1/A);
 if M=0 then R:=1
        else if M>0 then R:=Step(M,A)
                    else R:=Step(-M,1/A);
 Z:=Z/(2*R);
 Edit3.Text:=FloatToStrF(Z,fffixed,7,5);
end;
```

```
end.
```

5. Запустить программу на компиляцию и выполнение.

Контрольный пример 4.2

Напишите процедуру, которая по заданному интервалу и функции определяет «ноль» функции с заданной точностью, используя метод деления отрезка пополам. Известно, что на границах интервала функция принимает значения, отличные по знаку. Определите «ноль» функции $\sin(x) = 0.2 \cdot x$ (наименьший положительный корень) на произвольном интервале $[a;b]$. Числа a и b вводятся с клавиатуры.

Метод деления отрезка пополам. Пусть уравнение $F(x) = 0$ имеет на отрезке $[a,b]$ единственный корень, причем функция $F(x)$ на этом отрезке непрерывна. Разделим отрезок $[a,b]$ пополам точкой $c = {(a+b)}/{2}$. Если $F(c) \neq 0$, то возможны два случая: либо $F(x)$ меняет знак на отрезке $[a,c]$, либо на отрезке $[c,b]$. Выбирая в каждом случае тот из отрезков, на котором функция меняет знак, и, продолжая процесс деления отрезка пополам дальше, можно дойти до сколь угодно малого отрезка, содержащего корень уравнения. Если на каком-то этапе процесса получен отрезок $[\alpha,\beta]$, содержащий корень, то, приняв приближенно $x = {(\alpha+\beta)}/{2}$, получим ошибку, не превышающую значения $d = {(\beta-\alpha)}/{2}$. Уточненный корень исходного уравнения: $x = x + d$.

Решение.

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

2. Установить с помощью *Object Inspector* следующие свойства формы**:**

```
Form1.Height = 392
Form1.Width =  289
Form1.BorderIcons
biMaximize = false
Form1.BorderStyle = bsSingle
Form1.Position = poScreenCenter
Form1.Caption = 'Метод половинного деления'
```

3. Расположить на форме следующие компоненты: три компонента Edit, три компонента Label, один компонент Button и один компонент BitBtn. Установить с помощью инспектора объектов следующие свойства:

```
Label1.Caption = 'a'
Label2.Caption = 'b'
Label3.Caption = 'решение'
Edit1.Text = ''
Edit2.Text = ''
Edit3.Text = ''
Button1.Caption = 'Счет'
BitBtn1.Kind = 'bkClose'
BitBtn1. Caption = '&Закрыть'.
```

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,x1,x2,y:real;
begin
 a:=StrToFloat(edit1.Text);
 b:=StrToFloat(edit2.Text);
 x1:=a;
 x2:=b;
 zero(x1,x2,y);
 edit3.Text:=FloatToStrF(y,fffixed,8,4);
end;
```

Переменные, которые необходимы для решения задачи, описаны в разделе глобальных переменных приложения.

Для реализации метода половинного деления использовалась подпрограмма-процедура Zero, которая должна располагаться в тексте модуля до того, как к ней происходит обращение**:**

```
procedure zero(a,b:real; var res:real);
const eps=1E-5;
var
  s:boolean;x3:real;
  function f(x:real):real;
   begin
     f:=sin(2*x)-0.2*x;
   end; // f
  begin
   s:=f(a)< 0;
   repeat
```

```
        x1:=(a+b)/2;
        x3:=f(x1);
        if (x3<0)=s then a:=x1
                    else b:=x1;
       until abs(a-b)<eps;
       res:=x1;
      end; // zero
```

Раздел констант приложения содержит величину допустимой погрешности вычислений `EPS`. В процедуру `Zero` вложена функция `f`, которая задается пользователем как левая часть уравнения $f(x)=0$. При обращении к процедуре `Zero` происходит отделение корня: булева константа `S` принимает значение `True`, если $f(a)<0$ и `False` при $f(a)>0$f. Операторы, стоящие между `repeat` и `until`, повторяются до тех пор, пока выражение `until` не примет значение `True` при проверке стоящего за ним условия, что означает достижение нужной погрешности вычислений.

5. Запустить проект на компиляцию и выполнение.

Контрольный пример 4.3

Написать рекурсивную функцию вычисления *наибольшего общего делителя* по алгоритму Эйлера:

Если *B* делится на *A* нацело, то *НОД (A, B) = A*.

В противном случае *НОД(A, B) = НОД(B mod A, A)*.

Решение.

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

2. Установить с помощью *Object Inspector* следующие свойства формы:

```
Form1.Height = 347
Form1.Width =  359
Form1.BorderIcons
 biMaximize = false
Form1.BorderStyle = bsSingle
Form1.Position = poScreenCenter
Form1.Caption  =  'Нахождение наибольшего общего делителя'.
```

3. Расположить на форме следующие компоненты: три компонента `Edit`, три компонента `Label`, один компонент `Button` и один компонент `BitBtn`. Установить для них следующие свойства:

```
Label1.Caption = 'a'
Label2.Caption = 'b'
Label3.Caption = 'НОД'
Edit1.Text = ''
Edit2.Text = ''
Edit3.Text = ''
Button1.Caption = 'Вычислить НОД'
BitBtn1.Kind = 'bkClose'
BitBtn1. Caption = '&Закрыть'.
```

4. Текст программы приведен ниже:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, StdCtrls, Buttons;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    BitBtn1: TBitBtn;
    Label3: TLabel;
    Edit3: TEdit;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
 function NOD ( A, B : LongInt) : LongInt;
  begin
    if ( B mod A) = 0 then  // если A делит B нацело, то
значение вычислено
     NOD:=A
```

```
     else     //     в противном случае функция вычисляется
рекурсивно
      NOD:=NOD(B mod A, A);
   end;
procedure TForm1.Button1Click(Sender: TObject);
 var a,b:longint; f:longint;
begin
  a:=StrToInt(edit1.Text);
  b:=StrToInt(edit2.Text);
  f:=NOD(a,b);
  Edit3.Text:=IntToStr(f);
end;
end.
```

5. Запустить проект на компиляцию и выполнение.

4.3. Содержание отчета

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

4.4. Контрольные вопросы

1. Что называется подпрограммой? В чем состоит сходство и различие подпрограмм-процедур и подпрограмм-функций в языке Object Pascal?

2. В чем различие между стандартными и определенными пользователем программами?

3. Опишите последовательность событий при вызове процедуры или функции.

4. Что называется параметром и каково его назначение? Что такое формальные и фактические параметры, какова их взаимосвязь?

5. Каковы отличия параметров-переменных от параметров-значений, особенности их описания и применения?

6. Чем различаются локальные и глобальные параметры? Какова область их действия?

7. Что такое рекурсия?

8. Каковы особенности параметров-процедур и параметров-функций?

4.5. Варианты заданий

Вариант 1

Задание 1. Написать программу для вычисления выражения $z(x) = (sign(x) + sign(y)) \cdot sign(x + y)$. При решении задачи определите и используйте функцию *sign* (знак числа): $sign(x) = \begin{cases} -1, & x < 0, \\ 0, & x = 0, \\ 1, & x > 0. \end{cases}$

Задание 2. Написать процедуру вычисления определенного интеграла $I = \int_a^b \sin^2 x \cdot \frac{1}{1+x^2} dx$ методом прямоугольников: $I \approx \frac{b-a}{n} \cdot (y_0 + y_1 + \ldots + y_n)$, где n – количество отрезков разбиения; $y_0, y_1, \ldots, y_n$ – значения функции на концах отрезков. Числа a и b – произвольные, вводятся с клавиатуры.

Задание 3*. Напишите рекурсивную функцию вычисления i-го число Фибоначчи. Вычислите $f(k), \quad k = 15, 20, 30, 40$. Функция $f(n)$ определена для целых чисел следующим образом: $f(n) = \begin{cases} 1, n = 1 \\ \sum_2^n f(n \ div \ 2), \quad n \geq 2 \end{cases}$.

Вариант 2

Задание 1. Написать программу вычисления выражения $H(s,t) + (\max(H^2(s-t, s \cdot t), H^4(s-t, s+t)) + H(1,1)$, где $H(a,b) = \frac{a}{1+b^2} + \frac{b}{1+a^2} - (a-b)^3$. вычисление величин $H(a,b)$ и $\max(a,b)$ оформить в виде подпрограммы-функции. Числа s и t вводятся с клавиатуры.

Задание 2. Написать процедуру вычисления определенного интеграла $I = \int_a^b ctg\left(x^2 + 4\right) dx$ методом трапеций: $I \approx \frac{b-a}{2 \cdot n} \cdot (y_0 + 2 \cdot y_1 + \ldots + 2 \cdot y_{n-1} + y_n)$, где n – количество отрезков разбиения; $y_0, y_1, \ldots, y_n$ – значения функции на концах отрезков. Числа a и b – произвольные, вводятся с клавиатуры.

Задание 3*. Создайте программу вычисления числа сочетаний из N по M. Число сочетаний определяется по формуле $C_N^M = \frac{N!}{M!\cdot(N-M)!}$. Для вычисления факториала напишите рекурсивную функцию.

Вариант 3

Задание 1. Написать программу вычисления выражения $u = \frac{\max^2(x,y,z) - 2^x \cdot \min(x,y,z)}{\sin 2 + \max(x,y,z)/\min(x,y,z)}$ по заданным с клавиатуры числам x, y, z. Нахождение максимального и минимального значения из трех чисел оформить в виде подпрограмм-функций $\max(x,y,z)$ и $\min(x,y,z)$.

Задание 2. Написать процедуру вычисления определенного интеграла $\int_a^b 3\sin(x^2+1)dx$ по формуле Симпсона $I \approx \frac{b-a}{6\cdot n}\cdot(y_0 + 4\cdot y_1 + 2\cdot y_2 + \ldots + 4\cdot y_{2n-1} + y_{2n})$, где $2\cdot n$ – количество отрезков разбиения; $y_0, y_1, \ldots, y_{2n}$ – значения функции на концах отрезков. Числа a и b – произвольные, вводятся с клавиатуры.

Задание 3*. Написать программу нахождения наименьшего общего кратного (НОК) двух натуральных чисел, используя рекурсивный алгоритм нахождения наибольшего общего делителя как вспомогательный. Для любых натуральных чисел a, b справедливо тождество $a \cdot b = НОД(a,b) \cdot НОК(a,b)$.

Вариант 4

Задание 1. Написать программу вычисления выражения $u = \frac{\min\left(\frac{x+y+z}{3}, xyz, z\right)}{\max^2(x+y+z, xyz, y) + \min(x,y,z)}$. Нахождение максимального и минимального значения из трех чисел оформить в виде подпрограмм-функций $\max(a,b,c)$ и $\min(a,b,c)$.

Задание 2. Написать процедуру вычисления определенного интеграла $I = \int_a^b \frac{\cos x}{1+x} dx$ методом трапеций: $I \approx \frac{b-a}{2 \cdot n} \cdot \left(y_0 + 2 \cdot y_1 + \ldots + 2 \cdot y_{n-1} + y_n\right)$, где n – количество отрезков разбиения; $y_0, y_1, \ldots, y_n$ – значения функции на концах отрезков. Числа a и b – произвольные, вводятся с клавиатуры.

Задание 3*. Написать рекурсивную функцию, которая по заданному вещественному n вычисляет величину x^n согласно формуле

$$x^n = \begin{cases} 1, & n = 0 \\ \frac{1}{x^{|n|}}, & n < 0 \\ x \cdot \left(x^{n-1}\right), & n > 0 \end{cases}.$$

Вариант 5

Задание 1. Даны координаты вершин многоугольника $(x_1, y_1); (x_2, y_2); \ldots (x_5, y_5)$. Найти его периметр. Вычисление расстояния между вершинами оформить в виде подпрограммы-функции.

Задание 2. Написать процедуру вычисления определенного интеграла $I = \int_a^b \frac{\ln x}{x} dx$ по формуле Симпсона $I \approx \frac{b-a}{6 \cdot n} \cdot \left(y_0 + 4 \cdot y_1 + 2 \cdot y_2 + \ldots + 4 \cdot y_{2n-1} + y_{2n}\right)$, где $2 \cdot n$ – количество отрезков разбиения; $y_0, y_1, \ldots, y_{2n}$ – значения функции на концах отрезков. Числа a и b – произвольные, вводятся с клавиатуры.

Задание 3*. Написать рекурсивную функцию, которая вычисляет $y = \sqrt[k]{x}$ по следующей формуле: $y_0 = 1$; $y_{n+1} = y_n + \frac{\frac{x}{y_n^{k-1}} - y_n}{k}$. За ответ принять приближение, для которого выполняется условие $|y_n - y_{n+1}| < 0.0001$.

Вариант 6

Задание 1. Даны действительные числа s, t. Написать программу вычисления выражения $f(t,-2s,1{,}17)+f(2{,}2,t,s-t)$. Вычисление функции $f(a,b,c)=\dfrac{2a-b-\sin c}{5+|c|}$ оформить в виде подпрограммы-функции.

Задание 2. Написать процедуру вычисления определенного интеграла $I=\int_a^b \sqrt{1-0.5\cdot\sin^2 x}\,dx$ методом прямоугольников: $I\approx\dfrac{b-a}{n}\cdot(y_0+y_1+\ldots+y_n)$, где n – количество отрезков разбиения; $y_0, y_1, \ldots, y_n$ – значения функции на концах отрезков. Числа a и b – произвольные, вводятся с клавиатуры.

Задание 3*. Задано вещественное число $a>0$. Вычислить $\dfrac{\sqrt[3]{a}-\sqrt[6]{a^2+1}}{\sqrt[7]{a}}$. Вычисление корней $y=\sqrt[k]{x}$ оформить в виде подпрограммы. Корни вычислять с точностью $E=0.00001$ по итерационной формуле: $y_0=1;\ y_{n+1}=y_n+\dfrac{\left(\dfrac{x}{y_n^{k-1}}-y_n\right)}{k}$, приняв за ответ приближение, для которого $|y_{n+1}-y_n|<E$. Вычисление y^k оформить в виде рекурсивной подпрограммы.

Вариант 7

Задание 1. Для двух квадратных уравнений $a_1x^2+b_1x+c_1=0$ и $a_2x^2+b_2x+c_2=0$ определить, имеют ли они общие корни. Вывести на экран корни уравнения, которые не совпадают. Решение уравнений оформить в виде подпрограммы-функции. Числа a_1, b_1, c_1, a_2, b_2, c_2 вводятся с клавиатуры.

Задание 2. Написать процедуру вычисления определенного интеграла $I=\int_a^b e^{-x^2}dx$ по формуле Симпсона $I\approx\dfrac{b-a}{6\cdot n}\cdot(y_0+4\cdot y_1+2\cdot y_2+\ldots+4\cdot y_{2n-1}+y_{2n})$, где

$2 \cdot n$ – количество отрезков разбиения; $y_0, y_1, \ldots, y_{2n}$ – значения функции на концах отрезков. Числа a и b – произвольные, вводятся с клавиатуры.

Задание 3*. Написать процедуру сложения двух дробей, результатом которого является несократимая правильная дробь. Использовать рекурсивную программу нахождения НОД.

Вариант 8

Задание 1. Написать программу вычисления выражения $\text{sh}\, x + \text{tg}(x+1) - \dfrac{\text{ctg}^2\left(2 + \text{sh}(x-1)\right)}{\sqrt{|x+1|}}$. Вычисление $\text{sh}\, x = \dfrac{e^x - e^{-x}}{2}$; $\text{tg}x = \dfrac{\sin x}{\cos x}$ и $\text{ctg}x = \dfrac{\cos x}{\sin x}$ оформить в виде подпрограммы-функции.

Задание 2. Написать процедуру вычисления определенного интеграла $I = \int_a^b \dfrac{x}{\cos^3 x} dx$ методом прямоугольников: $I \approx \dfrac{b-a}{n} \cdot \left(y_0 + y_1 + \ldots + y_n\right)$, где n – количество отрезков разбиения; $y_0, y_1, \ldots, y_n$ – значения функции на концах отрезков. Числа a и b – произвольные, вводятся с клавиатуры.

Задание 3*. Написать программу вычисления выражения $\left(\dfrac{A}{B+C} - \dfrac{C}{A-C}\right) \cdot \dfrac{E}{F}$ в виде правильной дроби, где A, B, C, E, F – целые числа. Сложение двух дробей оформить как подпрограмму-функцию. Использовать рекурсивную программу нахождения НОД.

Лабораторная работа №5
ИСПОЛЬЗОВАНИЕ ВИЗУАЛЬНЫХ КОМПОНЕНТОВ ДЛЯ ПРОГРАММИРОВАНИЯ МАССИВОВ

Цель работы: приобретение практических навыков проетирования и программирования массивов.

Используемые программные средства: Borland Delphi.

5.1. 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. Количество индексных позиций определяет размерность массива (одномерный, двумерный и т.д.) (см. Приложение А).

Работу с массивами данных удобно организовывать в виде таблиц. Для этой цели предназначены 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 `DrawGrid` позволяет отображать любую (текстовую и графическую) информацию, однако вся работа по визуальному отображению объектов возлагается на разработчиков программы. Компонент `StringGrid` применяется для работы с текстовой информацией, причем ее отображение и хранение производится компонентом автоматически. 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 компонента StringGrid приведены в табл. 5.1.

Таблица 5.1

Свойства компонента StringGrid

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	параметры отображения полосы прокрутки :
ssNone	полоса прокрутки не допускается
ssHorizontal	допускается горизонтальная полоса прокрутки
ssVertical	допускается вертикальная полоса прокрутки
ssBoth	допускаются обе полосы прокрутки
Options	опции для доступа к параметрам таблицы для их настройки :
goVertLine	отображение в сетке вертикальных разделительных линий
goEditing	возможность редактирования содержания ячейки с клавиатуры
goHorzLine	отображение в сетке горизонтальных разделительных линий

В качестве фиксированных элементов используются 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  . 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 с

возможностями их перемещения по форме и стыковкой с другими панелями. Кроме того, компонент Panel может использоваться для создания рамок разнообразного вида (свойства BevelInner, BevelOuter , BevelWidth компонента).

5.2. Порядок выполнения работы

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

Контрольный пример 5.1

Сформировать матрицу A размерности $n \times n$, для которой

$$a_{ij} = \begin{cases} j^2 \cdot i, & i < j \\ 2, & i = j \\ \dfrac{i^3}{j}, & i > j \end{cases}$$

Решение.

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

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

```
Form1.Height = 480
Form1.Width = 487
Form1.BorderIcons
biMaximize = false
Form1.BorderStyle = bsSingle
```

3. На форме расположить следующие компоненты: один компонент Edit, два компонента Label, один компонент StringGrid, два компонента Button и один компонент BitBtn установить с помощью *Object Inspector* для них следующие свойства:

```
Label1.Caption = 'Размерность матрицы A'
Label2.Caption = 'Матрица A'
Edit1.Width = 120
Button1.Caption = 'Изменить размерность матрицы A'
StringGrid1.Left = 50
StringGrid1.Top = 150
```

```
StringGrid1.Height = 153
StringGrid1.Width = 313
StringGrid1.ScrollBars = ssBoth
Button2.Caption = 'Получение матрицы A'
BitBtn1.Kind = 'bkClose'
BitBtn1. Caption = '&Закрыть'.
```

Результат показан на рис. 5.1.

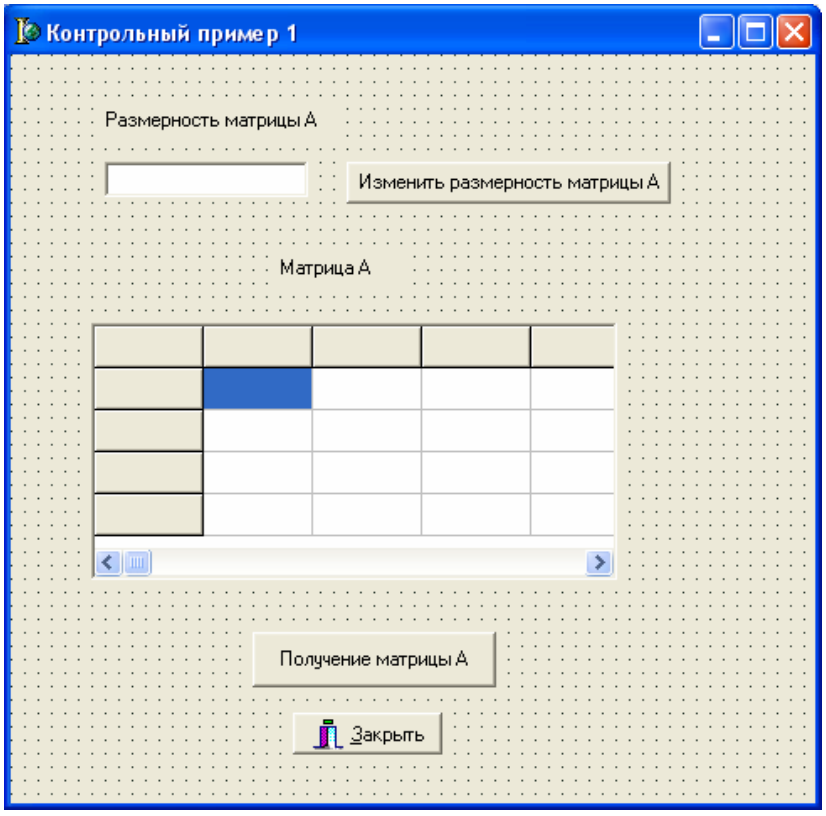

Рис. 5.1. Вид формы для контрольного примера 5.1

4. Запишем обработчик события `Button1.Click` (кнопка **Изменить размерность матрицы A**), в процессе выполнения которого устанавливается размерность матрицы, число столбцов и число строк в компоненте `StringGrid1`.

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

```
type
 mas=array[1..10,1..10] of real;
var
  Form1: TForm1;
  N: integer;
```

```
     A:mas;
```
Текст процедуры `TForm1.Button1Click` имеет вид:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 i:integer;
begin
 N:=StrToInt(Edit1.Text);// размерность матрицы A
{Задание числа строк и столбцов в таблице}
 StringGrid1.ColCount:=N+1;
 StringGrid1.RowCount:=N+1;
{Ввод в левую верхнюю ячейку таблицы названия массива}
 StringGrid1.Cells[0,0]:='Массив A:';
{Заполнение левого и верхнего столбцов поясняющими записями}
     for i:=1 to N do begin
       StringGrid1.Cells[0,i]:=' i= '+IntToStr(i);
       StringGrid1.Cells[i,0]:=' j= '+IntToStr(i);
                      end;
end;
```

5. Для формирования матрицы *A* нужно записать процедуру, являющуюся обработчиком события `Button2.Click` (кнопка **Получение матрицы А**). Текст процедуры приведен ниже:

```
procedure TForm1.Button2Click(Sender: TObject);
 var i,j:integer;
begin
  for i:=1 to n do
   for j:= 1 to n do
   begin
    if i<j then A[i,j]:=sqr(j)*i;
    if i=j then A[i,j]:=2;
    if i>j then A[i,j]:=i*sqr(i)/j;
   end;
{Вывод результата в таблицу  StringGrid1}
for i:=1 to N do
 for j:=1 to N do
  StringGrid1.Cells[j,i]:=FloatToStrf(A[i,j],fffixed,6,2);
end;
```

6. Запустить проект на выполнение. Результат выполнения программы показан на рис. 5.2.

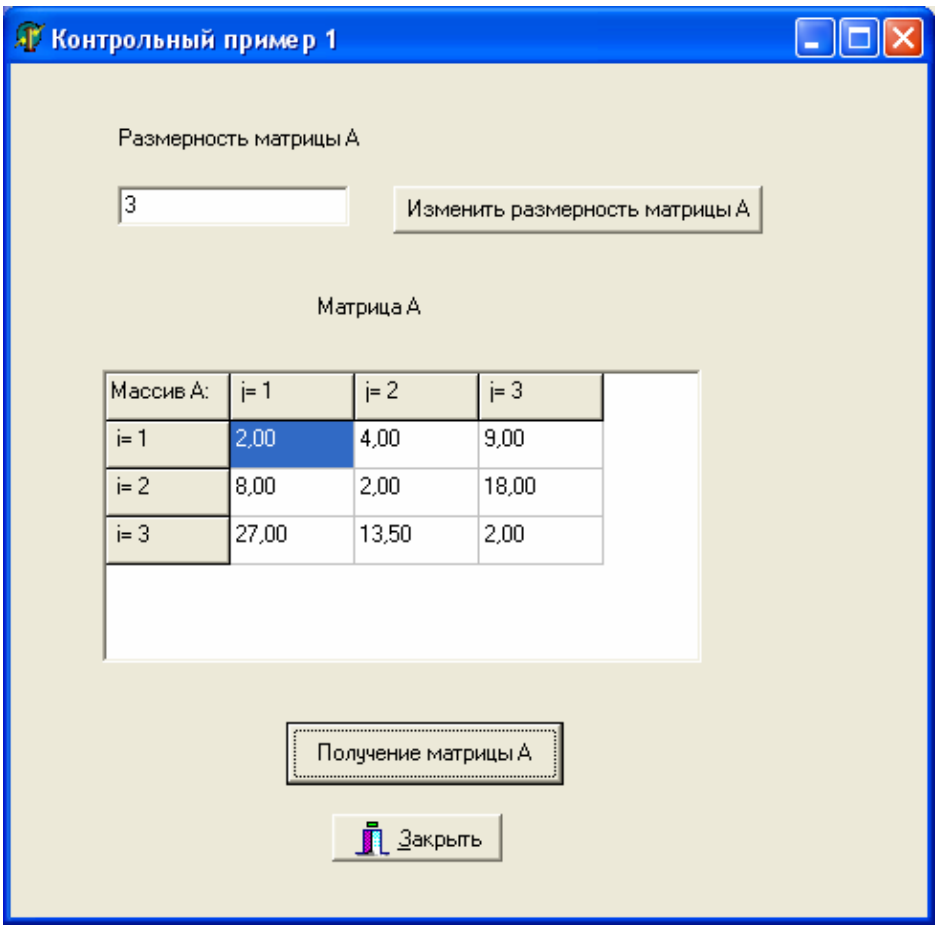

Рис. 5.2. Результат выполнения программы
для контрольного примера 5.1

Контрольный пример 5.2

Создать программу для определения вектора $\vec{Y} = A * \vec{B}$, где A – квадратная матрица размерности $N \times N$, а Y, B – одномерные массивы размерности N. Элементы вектора Y определяются по формуле $Y_i = \sum_{j=1}^{N} A_{ij} \cdot B_i$. Значения N вводить в компонент `Edit`, элементы массивов A и B – в компонент `StringGrid`. Результат, после нажатия кнопки типа `Button`, вывести в компонент `StringGrid`.

Решение.

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

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

```
Form1.Height = 395
Form1.Width = 585
Form1.BorderIcons
biMaximize = false
Form1.BorderStyle = bsSingle
```

3. На форме расположить следующие компоненты: один компонент `Edit`, один компонент `Label`, три компонента `StringGrid`, два компонента `Button` и один компонент `BitBtn` установить с помощью *Object Inspector* для них следующие свойства:

```
Label1.Caption = 'Размерность массива'
Edit1.Width = 50
Button1.Caption = 'Изменить размерность массива'
StringGrid1.ColCount = 2
StringGrid1.RowCount = 2
StringGrid1.FixedCols = 1
StringGrid1.FixedRows = 1
StringGrid2.ColCount = 1
StringGrid2.RowCount = 2
StringGrid2.FixedCols = 0
StringGrid2.FixedRows = 1
StringGrid3.ColCount = 1
StringGrid3.RowCount = 2
StringGrid3.FixedCols = 0
StringGrid3.FixedRows = 1
Button2.Caption = 'Вычислить A'
BitBtn1.Kind = 'bkClose'
BitBtn1. Caption = '&Закрыть'.
```

По умолчанию в компонент `StringGrid` запрещен ввод информации с клавиатуры, поэтому необходимо свойство `Options goEditing` для компонентов `StringGrid1` и `StringGrid2` установить в положение `True`.

4. Текст программы имеет вид

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, Buttons, Grids, StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Button1: TButton;
```

```
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
    StringGrid3: TStringGrid;
    Button2: TButton;
    BitBtn1: TBitBtn;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
const
  Nmax=10;        // Максимальная размерность массива
Type
  Mas2 = array[1..Nmax,1..Nmax] of extended; // Объявление типа
двумерного массива размерностью Nmax
  Mas1 = array[1..Nmax] of extended;         // Объявление типа
одномерного массива размерностью Nmax
var
  Form1: TForm1;
  A : Mas2;         // Объявление двумерного массива
  B,Y : Mas1;     // Объявление одномерного массива
  N: integer;

implementation

{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
 var i,j:integer;
begin
  N:=StrToInt(Edit1.Text);
 {Задание числа строк и столбцов в таблицах}
   StringGrid1.ColCount:=N+1;
   StringGrid1.RowCount:=N+1;
   StringGrid2.RowCount:=N+1;
   StringGrid3.RowCount:=N+1;
 { Ввод в левую верхнюю ячейку таблицы названия массива}
   StringGrid1.Cells[0,0]:='Массив A:';
   StringGrid2.Cells[0,0]:='Массив B:';
   StringGrid3.Cells[0,0]:='Массив Y:';
 {Заполнение верхнего и левого столбцов поясняющими подписями}
   for i:=1 to N do begin
    StringGrid1.Cells[0,i]:=' i= '+IntToStr(i);
    StringGrid1.Cells[i,0]:=' j= '+IntToStr(i);
                    end;
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
  var s: extended;i,j:integer;
begin
    { Заполнение массива A элементами из таблицы StringGrid1}
  for i:=1 to N do
   for j:=1 to N do
     A[i,j]:=StrToFloat(StringGrid1.Cells[j,i]);
  { Заполнение массива B элементами из таблицы StringGrid2}
  for i:=1 to N do
     B[i]:=StrToFloat(StringGrid2.Cells[0,i]);
   {Умножение массива A на массив B}
  for i:=1 to N do begin
      s:=0;
   for j:=1 to N do s:=s+A[i,j]*B[j];
         Y[i]:=s;
            {Вывод результата в таблицу StringGrid3}

StringGrid3.Cells[0,i]:=FloatToStrF(y[i],fffixed,6,2);
                    end;
end;
end.
```

5. Запустить проект на выполнение. Результат выполнения программы показан на рис. 5.3.

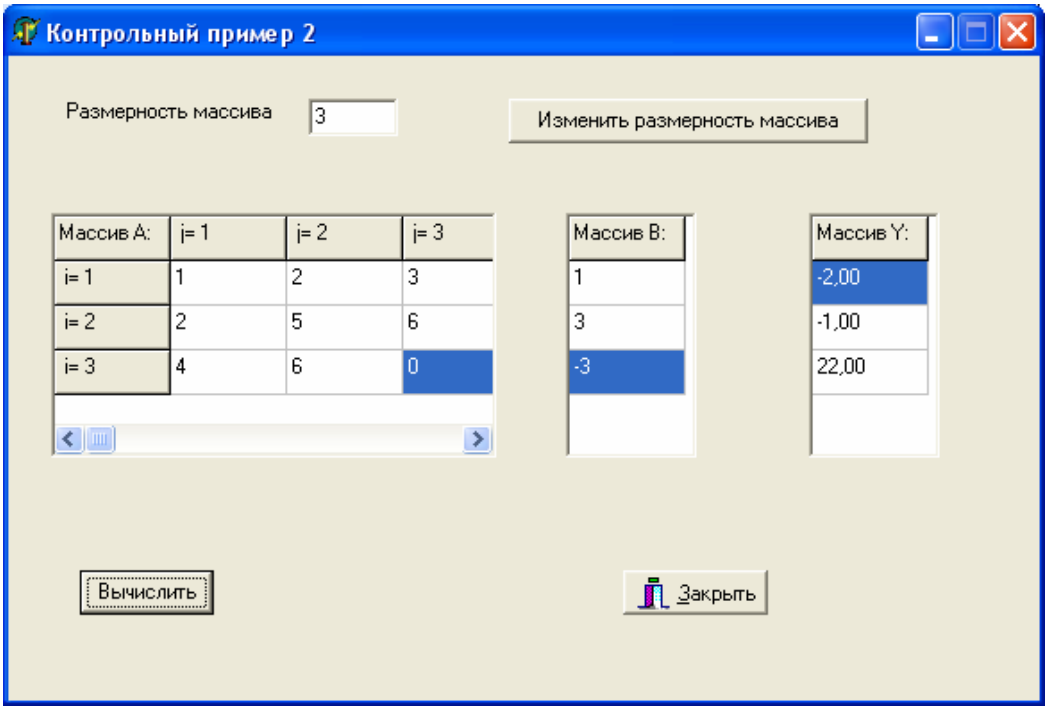

Рис. 5.3. Результат выполнения программы для контрольного примера 5.2

Контрольный пример 5.3

Для заданной матрицы $A(a_{ij})$ сформировать матрицу $B(b_{ij})$ так, чтобы $b_{ij} = \dfrac{a_{ij}^2}{a_{ij} + P \cdot a_{ij}^2 + Q}$. 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. На форме расположить следующие компоненты: два компонента `GroupBox`, пять компонентов `Edit`, пять компонента `Label`, два компонента `StringGrid` и два компонента `Button` и установить с помощью *Object Inspector* для них 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'
Label4.Caption = 'Сумма элементов='
StringGrid2.Left = 10
StringGrid2.Top = 71
StringGrid2.ScrollBars = ssBoth
```

4. При запуске программы в момент создания формы возникает событие `OnCreate` (создание формы). 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;
```

5. 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.

6. Для формирования матрицы *B* и вычисления суммы элементов нечетных строк в этой матрице нужно 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;
```

7. Запустить проект на выполнение. Результат выполнения программы показан на рис. 5.4.

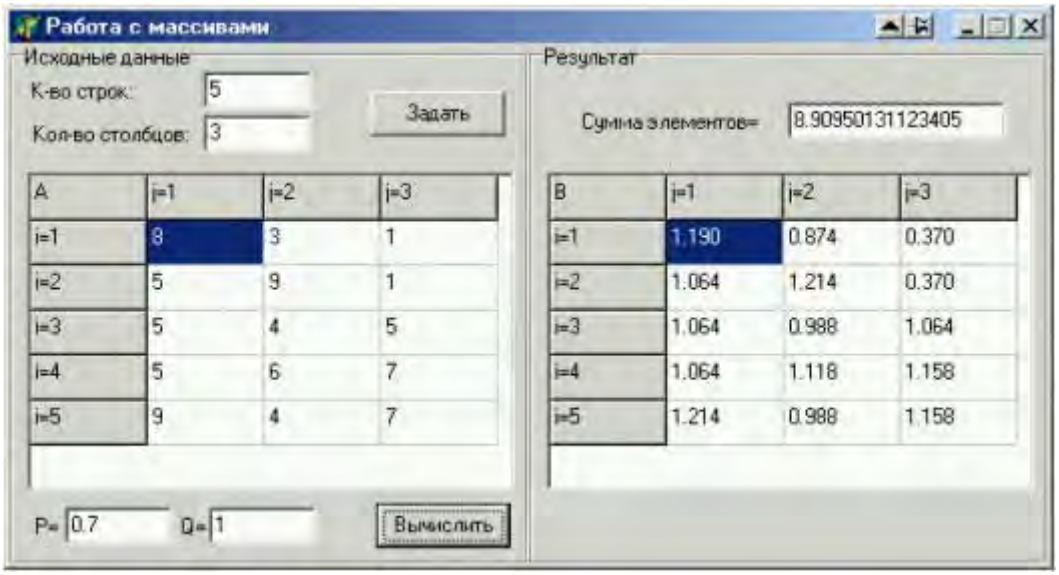

Рис. 5.4. Результат выполнения программы
для контрольного примера 5.3

5.3. Содержание отчета

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

5.4. Контрольные вопросы

1. Приведите определение массива.
2. Какие данные могут быть элементами массива?
3. С какой целью применяется компоненты `StringGrid` и `DrawGrid`?

4. Какие основные свойства компонентов `StringGrid` и `DrawGrid`?

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

5.5. Варианты заданий

Вариант 1

Задание 1. Сформировать матрицу A размерности $m \times n$, для которой

$$a_{ij} = \begin{cases} \cos(i^2 + n \cdot j) & i < j \\ 2i/(j+1) & i = j \\ \ln(i \cdot j) & i > j \end{cases}.$$

Числа m и n вводятся с клавиатуры.

Задание 2. Написать программу, вычисляющую значение матрицы $C = A + k \cdot B$. Найти произведение всех элементов матрицы C (для вывода значения использовать компонент `Edit`). Элементы матриц A, B и параметр k вводятся с клавиатуры.

Задание 3*. Написать программу, вычисляющую произведение двух матриц произвольной размерности. Элементы матриц вводятся с клавиатуры в компонент `StringGrid`. Предусмотреть возможность ввода с клавиатуры количества строк и столбцов заданных матриц.

Вариант 2

Задание 1. Сформировать матрицу A размерности $n \times n$, для которой

$$a_{ij} = \begin{cases} \sin\left(i + j/2\right) & i < j \\ i \cdot j & i = j \\ 2 \cdot i + 1/j & i > j \end{cases}.$$

Число n вводится с клавиатуры.

Задание 2. Задана вещественная матрица A размерности $m \times n$. Написать программу нахождения суммы элементов строки матрицы, в которой расположен

элемент с наименьшим значением. Предполагается, что этот элемент единственный. Элементы матрицы A и числа m и n вводятся с клавиатуры. Для вывода результата использовать компонент `Edit`.

Задание 3*. Написать программу, вычисляющую произведение двух матриц произвольной размерности. Элементы матриц вводятся с клавиатуры в компонент `StringGrid`. Предусмотреть возможность ввода с клавиатуры количества строк и столбцов заданных матриц.

Вариант 3

Задание 1. Сформировать матрицу A размерности $n \times n$, для которой

$$a_{ij} = \begin{cases} i - \dfrac{3}{1 + 1/j} & i < j \\ 2j & i = j \\ tg(i + j) & i > j \end{cases}.$$

Число n вводится с клавиатуры.

Задание 2. Дана вещественная матрица A размерности $m \times n$, все элементы которой различны. Написать программу нахождения номера столбца, в котором находится наибольшее количество отрицательных элементов. Элементы матрицы A и числа m и n вводятся с клавиатуры.

Задание 3*. Написать программу, вычисляющую произведение двух матриц произвольной размерности. Элементы матриц вводятся с клавиатуры в компонент `StringGrid`. Предусмотреть возможность ввода с клавиатуры количества строк и столбцов заданных матриц.

Вариант 4

Задание 1. Сформировать матрицу A размерности $n \times n$, для которой

$$a_{ij} = \begin{cases} j - \dfrac{3 \cdot e^i}{7} & i < j \\ \dfrac{i}{1+j} & i = j \\ \sin^2 i & i > j \end{cases}.$$

Число n вводится с клавиатуры.

Задание 2. Для матрицы A размерности $m \times n$ посчитать количество элементов меньше нуля и равных нулю. Элементы матрицы A и числа m, n вводятся с клавиатуры. Для вывода результатов использовать компонент `Edit`.

Задание 3*. Написать программу, вычисляющую произведение двух матриц произвольной размерности. Элементы матриц вводятся с клавиатуры в компонент `StringGrid`. Предусмотреть возможность ввода с клавиатуры количества строк и столбцов заданных матриц.

Вариант 5

Задание 1. Сформировать матрицу A размерности $m \times n$, для которой

$$a_{ij} = \begin{cases} \sin\left(\dfrac{j^2 - i^2}{i}\right), & i < j \\ i, & i = j \\ \arcsin\dfrac{i \cdot j}{i + 2 \cdot j}, & i > j \end{cases}.$$

Числа m и n вводятся с клавиатуры.

Задание 2. Для заданной матрицы A [m , n] найти минимальный элемент (и его номер) и максимальный элемент (и его номер). Для вывода результатов использовать компонент `Edit`. Элементы матрицы A и числа m, n вводятся с клавиатуры.

Задание 3*. Написать программу, вычисляющую произведение двух матриц произвольной размерности. Элементы матриц вводятся с клавиатуры в компонент

StringGrid. Предусмотреть возможность ввода с клавиатуры количества строк и столбцов заданных матриц.

Вариант 6

Задание 1. Сформировать матрицу A размерности $m \times n$, для которой

$$a_{ij} = \begin{cases} (i+j)\cdot \sin i, & i < j \\ \frac{1}{3} j, & i = j \\ \frac{i \cdot j}{i + 2 \cdot j}, & i > j \end{cases}.$$

Числа m и n вводятся с клавиатуры.

Задание 2. Для матрицы A размерности $m \times n$ посчитать количество положительных элементов в каждом столбце. Элементы матрицы A и числа m, n вводятся с клавиатуры. Для вывода результатов использовать компонент Edit.

Задание 3*. Написать программу, вычисляющую произведение двух матриц произвольной размерности. Элементы матриц вводятся с клавиатуры в компонент StringGrid. Предусмотреть возможность ввода с клавиатуры количества строк и столбцов заданных матриц.

Вариант 7

Задание 1. Сформировать матрицу A размерности $n \times n$, для которой

$$a_{ij} = \begin{cases} ctgij, & i < j \\ j/2, & i = j \\ \frac{2i + 5j}{i^2}, & i > j \end{cases}.$$

Число n вводится с клавиатуры.

Задание 2. 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'*. Элементы матрицы вводятся с клавиатуры.

Задание 3*. Написать программу, вычисляющую произведение двух матриц произвольной размерности. Элементы матриц вводятся с клавиатуры в компонент `StringGrid`. Предусмотреть возможность ввода с клавиатуры количества строк и столбцов заданных матриц.

Вариант 8

Задание 1. Сформировать матрицу A размерности $m \times n$, для которой

$$a_{ij} = \begin{cases} \sin(i+j), & i < j \\ 1, & i = j \\ arctg\dfrac{i+j}{2i+3j}, & i > j \end{cases}.$$

Числа m и n вводятся с клавиатуры.

Задание 2. Задана вещественная квадратная матрица порядка n. Написать программу нахождения среднего арифметического наибольшего и наименьшего значений ее элементов. Элементы матрицы и число n вводятся с клавиатуры. Для вывода результата использовать компонент `Edit`.

Задание 3.* Написать программу, вычисляющую произведение двух матриц произвольной размерности. Элементы матриц вводятся с клавиатуры в компонент `StringGrid`. Предусмотреть возможность ввода с клавиатуры количества строк и столбцов заданных матриц.

Лабораторная работа №6

ПОСТРОЕНИЕ ДИАГРАММ И ГРАФИКОВ ФУНКЦИЙ

Цель работы: приобретение практических навыков по построению диаграмм и графиков функций.

Используемые программные средства: Borland Delphi.

6.1. 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. 6.1) 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 и т.д. берет на себя компонент Chart. Разработчику программы требуется задать тип диаграммы и источник данных.

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

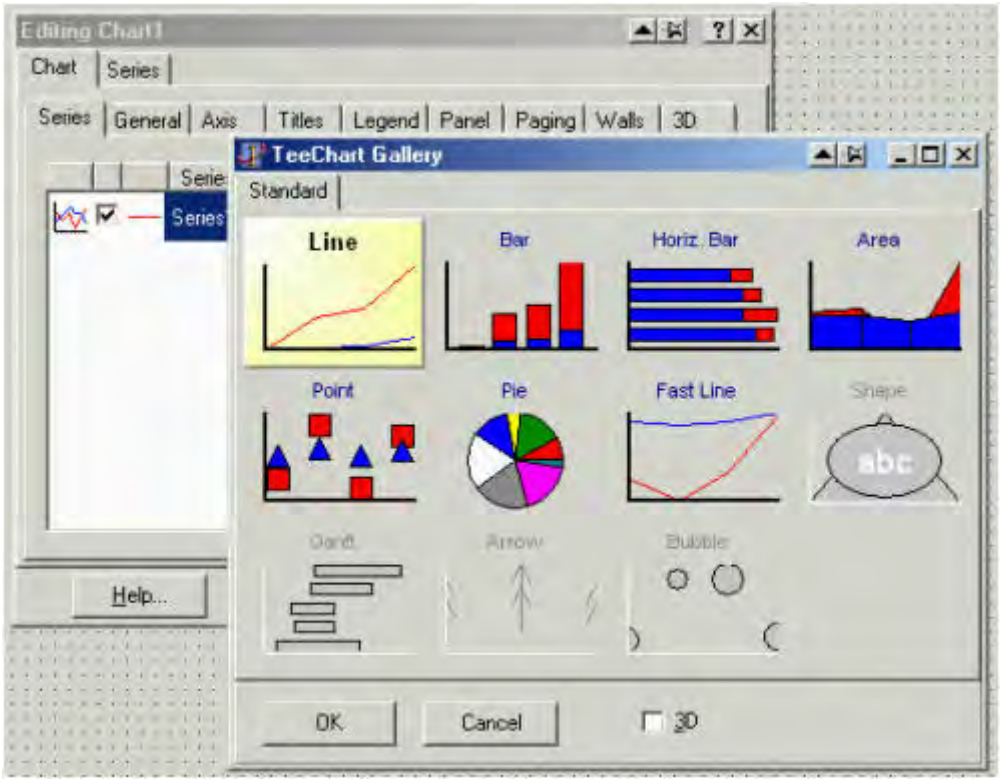

Рис. 6.1. Окно редактора Editing Chart

Аналогично для добавления нового сектора в круговой диаграмме так же можно воспользоваться функцией ***Add***:

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

где `PieValue` – величина сектора данных , `APieLabel` – необязательный параметр, `AColor` – цвет сектора.

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. 6.1.

Таблица 6.1

Свойства компонента Chart

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.

6.2. Порядок выполнения работы

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

Контрольный пример 6.1

Составить программу, отображающую графики функций $f_1 = \sin x$ и $f_1 = \cos x$ в интервале $[a, b]$ с заданным шагом *h*.

Решение.

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

2. На форме расположить следующие компоненты: три компонента Edit, три компонента Label, компонент Chart и компонент Button и установить для них следующие свойства:

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

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 Chart1 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. Редактирование 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.Minimum:=a;
  Chart1.BottomAxis.Maximum:=b;
```

```
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, показанный на рис. 6.2.

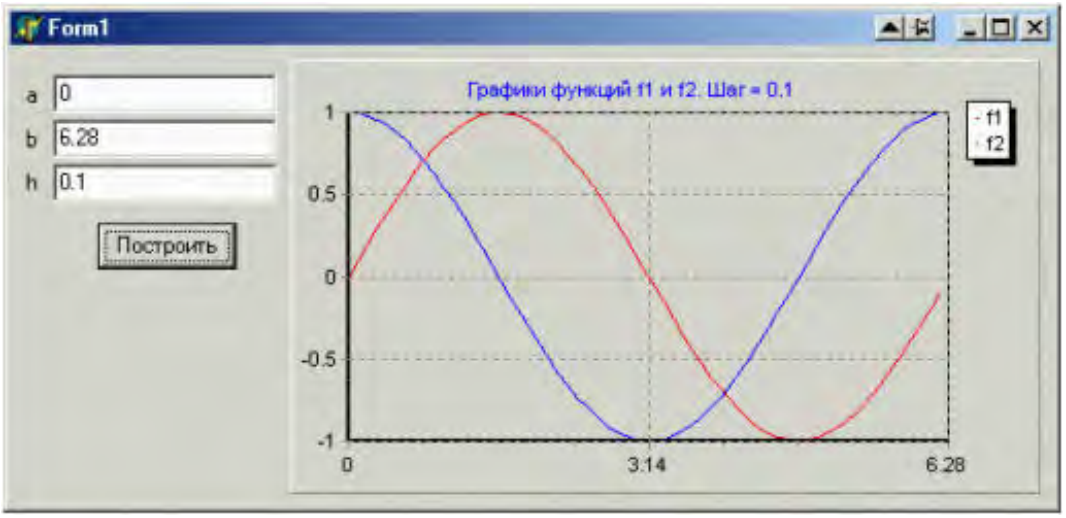

Рис. 6.2. Результат выполнения программы для контрольного примера 6.1

Контрольный пример 6.2

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

Решение.

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

2. На форме расположить следующие компоненты: компонент `Edit`, компонент `Label`, компонент `Chart`, компонент `CheckBox`, компонент `StringGrid` и два компонента `Button`.

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 EditingChart1, создать объект **Series1**, выбрав в качестве типа графика **Line**.

4. Запишем 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,j: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 использовался 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 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. 6.3.

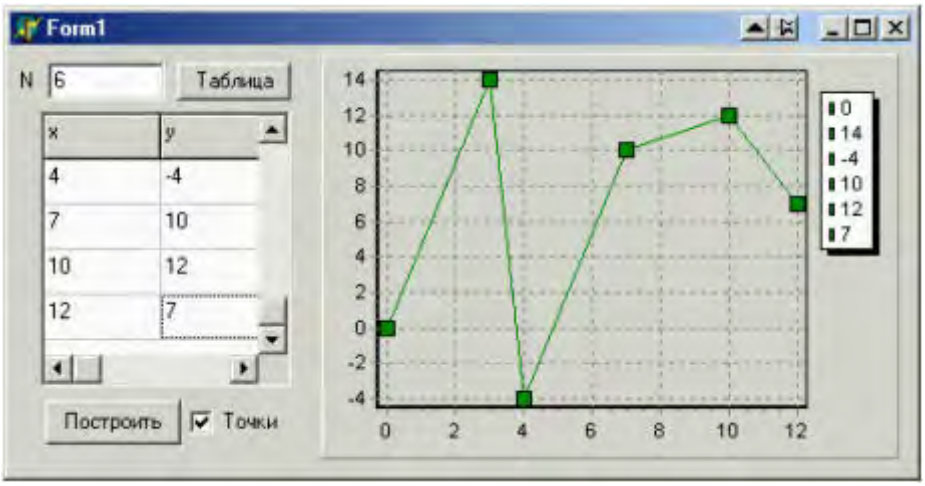

Рис. 6.3. Результат выполнения программы для контрольного примера 6.2

Контрольный пример 6.3

Построить круговую диаграмму реализации следующей продукции: гречка – 20%, пшено – 35%, рис – 45%. Использовать компонент типа T PieSeries (круговая диаграмма).

Решение.

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

2. На форме расположить следующие компоненты: три компонента `Edit`, три компонента `Label`, компонент `Chart`, компонент `Button`. В качестве типа графика выбрать `Pie`.

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,c:real;
begin
a:=strtofloat(edit1.Text);
b:=strtofloat(edit2.Text);
c:=strtofloat(edit3.Text);
```

```
With Series1 do   // with – оператор присоединения (with
<переменная> do   <оператор>)
Begin
   Clear ;
    AddPie(  a, ‘гречка’ , clRed ) ;
    AddPie(  b, 'пшено',  clyellow ) ;
    AddPie(  c, 'рис', clGreen ) ;
end;
```

Контрольный пример 6.4

Построить график функции $r = \sin k\varphi$ в полярной системе координат k и φ вводятся с клавиатуры.

Решение.

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

2. На форме расположить следующие компоненты: четыре компонента `Edit`, четыре компонента `Label`, (график рисуется на форме, перерисовывается при изменении параметра n в `Edit1` (рис. 6.4)).

```
procedure TForm1.FormPaint(Sender: TObject);
const
  XScale = 200;          // масштаб по горизонтали
  XShift = XScale;       // сдвиг по горизонтали
  YScale = 200;          // масштаб по вертикали
  YShift = YScale;       // сдвиг по вертикали
  ER = 2;                // радиус круга
var
  n, i, X, Y: Integer;
  r, fi, fi_0, fi_n, h, k: Real;
begin
  try
    n := StrToInt(Edit1.Text);
    k := StrToFloat(Edit4.Text);
    fi_0 := StrToFloat(Edit2.Text);
    fi_n := StrToFloat(Edit3.Text);
    h := Abs(fi_n - fi_0) / n;          // шаг аргумента
    with Canvas do
    begin
      Pen.Color := clNavy;              // цвет карандаша
      Brush. Color := clLime;           // цвет заливки
    end;
    for i := 0 to n do
    begin
      fi := fi_0 + i * h;               // расчет текущего
значения аргумента
```

```
      r := Sin(k * fi);                  // расчет текущего
значения функции
      X := Round(XScale * r * Cos(fi) + XShift);         //
преобразование
  Y := Round(YScale * r * Sin(fi) + YShift);
//координат
      Canvas.Ellipse(X - ER, Y - ER, X + ER, Y + ER);    //
рисуем эллипс (в нашем случае круг)
    end;
  except
    on e: EConvertError do
      MessageDlg('Нужно вводить числа', mtError, [mbOK], 0);
// обработка ошибки ввода числа
    else
      raise;
  end;
end;
```

При изменении пользователем параметра n в компоненте `Edit1` форма перерисовывается.

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  Invalidate;          // перерисовка формы
end;

procedure TForm1.Edit4Change(Sender: TObject);
begin
  Caption := Format('График функции  r=Sin(kφ)', [Edit4.Text]);
  Edit1Change(Self);
end;
```

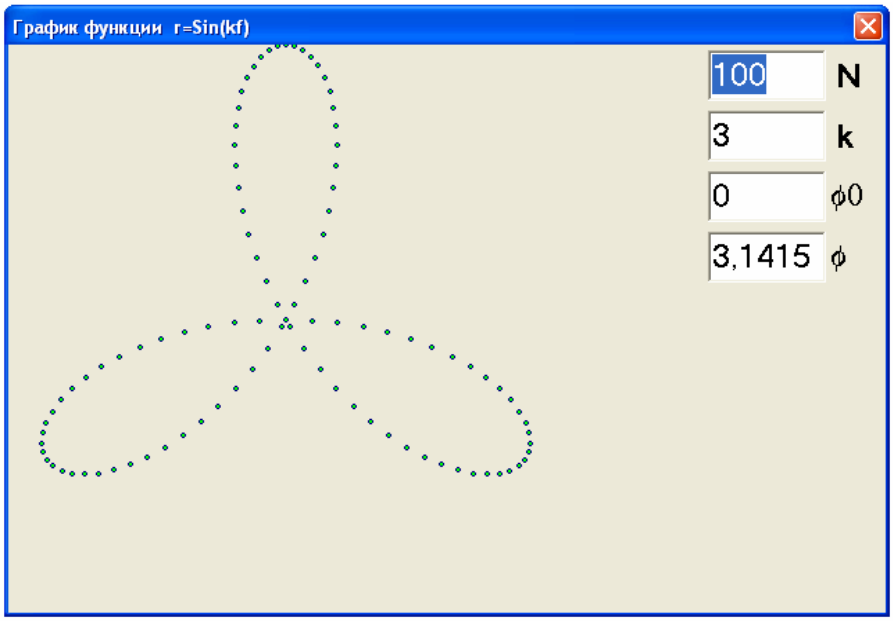

Рис. 6.4. Результат выполнения программы для контрольного примера 6.3

6.3. Содержание отчета

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

6.4. 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 во выполнении программы изменять тип диаграммы?

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

6.5. Варианты заданий

Вариант 1

Задание 1. Построить на одном графике функции $f_1 = e^x$ и $f_2 = \ln x$ $f_2 = \ln(x)$ на интервале [0,1;1]. Шаг h = 0.01.

Задание 2. В полярной системе координат построить график спирали Архимеда $r = \frac{a}{\varphi^2}$.

Вариант 2

Задание 1. Построить на одном графике функции $f_1 = x^3$ и $f_2 = |x|$ на интервале $[-10, 10]$.

Задание 2. В полярной системе координат построить график улитки Паскаля $r = 2a\cos\varphi$.

Вариант 3

Задание 1. Построить на одном графике три функции: $f_1 = x$, $f_2 = x^2$, $f_3 = x^3$ на интервале $[-20, 20]$.

Задание 2. В полярной системе координат построить график спирали Галилея $r = a\varphi - l$, $l \geq 0$.

Вариант 4

Задание 1. Построить на одном графике две функции, заданные таблично. Значения функций задаются с помощью компонента `StringGrid`.

Задание 2. В полярной системе координат построить график строфоиды $r = \dfrac{a}{\cos\varphi} + a\,\mathrm{tg}\varphi$ для $a = 1$.

Вариант 5

Задание 1. Построить график функции $y = ax^2 + bx + c$. Значения параметров *a*, *b*, *c* задаются с клавиатуры (использовать компонент `Edit`).

Задание 2. В полярной системе координат построить график кардиоиды $r = 2a(1 - \cos\varphi)$ для $a = 3$.

Вариант 6

Задание 1. Построить зависимость $I(\varphi) = I_0 \dfrac{\sin^2(\pi a \sin\varphi)}{(\pi a \sin\varphi)^2}$. Предусмотреть возможность задания параметров I_0 и *a*. Результат представить в графическом виде (компонент `Chart`) и табличном (компонент `StringGrid`).

Задание 2. В полярной системе координат построить график логарифмической спирали $r = a^{\varphi}$ для $a = 2$.

Вариант 7

Задание 1. Построить круговую диаграмму реализации следующей продукции: кофе – 20%, чай – 35%, напитки – 45%. Использовать компонент типа `TPieSeries` (круговая диаграмма).

Задание 2. В полярной системе координат построить график спирали «жезл» $f = a\sqrt{\varphi}$ для $a = 4$.

Вариант 8

Задание 1. Построить графики реализации книг в двух книжных магазинах по месяцам. Использовать компонент типа `TPieSeries` (круговая диаграмма).

Задание 2. В полярной системе координат построить график гиперболической спирали $r = \frac{a}{\varphi}$.

Лабораторная работа №7

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

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

Используемые программные средства: Borland Delphi.

7.1. Теоретические сведения

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

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. 7.1.

Таблица 7.1

Свойства компонента `ListBox`

`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** (аналогично компоненту `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 представляет собой объекты типа `TMenuItem`. Некоторые свойства пунктов меню приведены в табл. 7.2.

Таблица 7.2

Свойства объектов `TMenuItem`

`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 (используется 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*, 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. 7.3.

Таблица 7.3

Свойства компонентов `OpenDialog` и `SaveDialog`

Свойства	Тип	Описание
1	*2*	*3*
`DefaultExt`	`string`	задает расширение, автоматически подставляемое к имени файла, если не указано расширение
`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`	`TOpenOptions`	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

Окончание табл. 7.3.

1	*2*	*3*
`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. 7.1), 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 ';'.

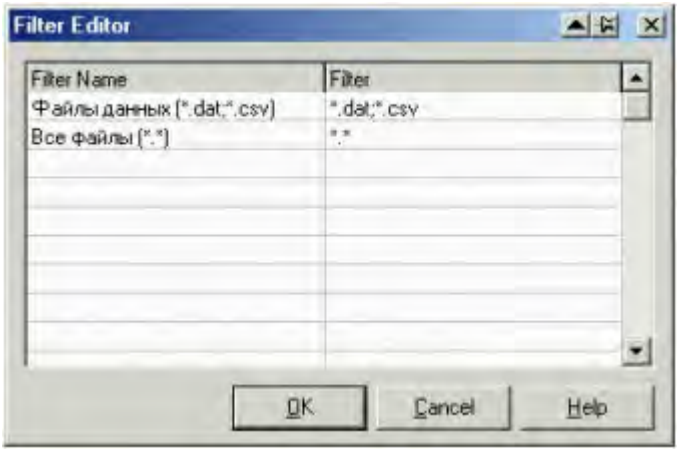

Рис. 7.1. Окно редактора фильтра

Компоненты `OpenPictureDialog`  и `SavePictureDialog`  вызывают стандартные диалоги открытия и сохранения графических файлов. От компонентов `OpenDialog` и `SaveDialog` отличаются 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.

`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.

7.2. Порядок выполнения работы

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

Контрольный пример 7.1

Написать программу ввода (чтения) в файл (из файла) информации о марках сталей и содержании в них легирующих элементов (химическом составе) (табл. 7.4). Интерфейс программы организовать с использованием меню. Для работы с данными использовать структуру типа 'запись'.

Таблица 7.4

Марка стали и ее химический состав

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 7.2.

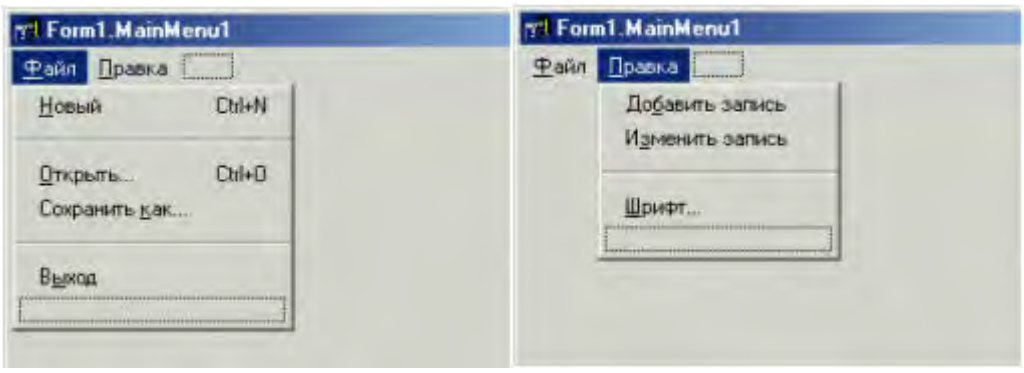

Рис. 7.2. Вид меню при конструировании

4. Для организации контекстного меню нужно 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. 7.3.

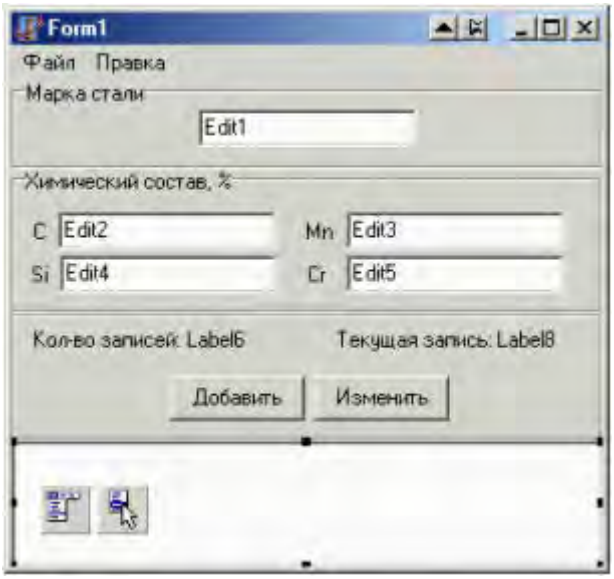

Рис. 7.3. Вид формы для контрольного примера 7.1

```
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; {максимальная 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;
  Steel: array [1.. MaxRec] of TSteel; {объявление переменной типа запись}
  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 созданных записей в файл, 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 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. 7.4.

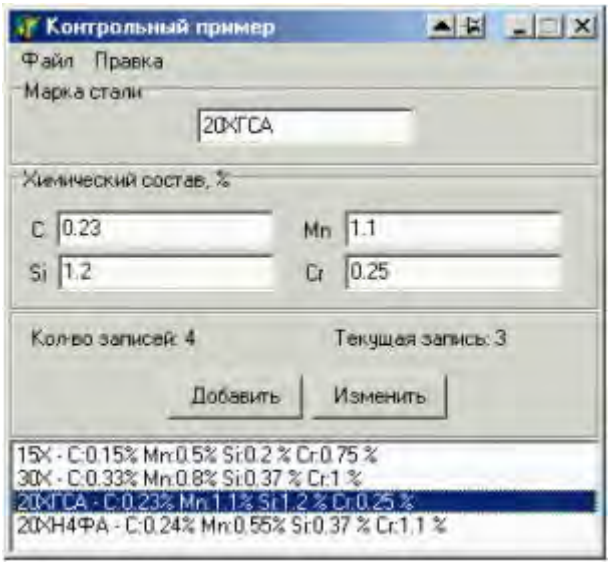

Рис. 7.4. Результат выполнения программы для контрольного примера 7.1

7.3. Содержание отчета

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

7.4. Контрольные вопросы

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

7.5. 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 фамилии, оценкам трем предметам и среднему баллу. 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 предприятиями 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 значения 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 годе 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 платы по тарифной ставке и полученной сумме после уплаты налога 13%. Работу с файлом организовать с помощью меню.

Л а б о р а т о р н а я р а б о т а № 8
ИСПОЛЬЗОВАНИЕ СРЕДСТВ DELPHI ДЛЯ РАБОТЫ С ЛОКАЛЬНЫМИ БАЗАМИ ДАННЫХ

Цель работы: изучить основы проектирования локальных баз данных.

Используемые программные средства: Borland Delphi.

8.1. Теоретические сведения

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

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 (табл. 8.1).

Таблица 8.1

Некоторые типы полей таблиц Paradox 7

Тип	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 должно состоять из букв и цифр и начинаться с буквы. Максимальная длина имени поля – 25 символов. В таблице не может быть 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).

Разработка в Delphi приложения для управления базой данных ничем не отличается от создания обычной программы. Просмотр баз данных обычно 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, как правило, интересует не все содержимое базы данных, 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.

8.2. 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 вкладках **Data Access** (рис. 8.1) и **Data Controls** (рис. 8.2) *Палитры компонентов*.

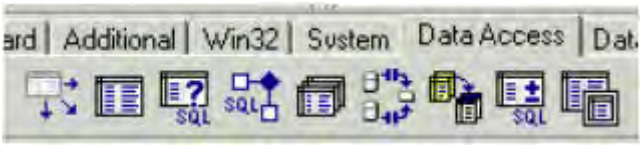

Рис. 8.1. Страница компонентов **Data Access**

Рис. 8.2. Страница компонентов **Data Controls**

Некоторые свойства основных компонентов для работы с БД приведены ниже (см. табл. 8.2).

Таблица 8.2

Свойства основных компонентов для работы с 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	Описание
`Next, Prior, First, Last`	используются для перемещения указателя соответственно на следующую, предыдущую, первую, последнюю записи набора данных

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

Компонент `DataSource`  (панель **Data Access** *Палитры компонентов*) – обеспечивает связь таблиц БД с компонентами просмотра и редактирования содержимого полей БД. Имя компонента, представляющего собой входные данные, хранится в свойстве `DataSet`.

Компонент `DBGrid`  (панель **Data Controls** *Палитры компонентов*) – обеспечивает представление БД в виде таблицы. Имя компонента, который является источником данных, хранится в свойстве `DataSource`.

Компонент `DBNavigator`  (панель **Data Controls** *Палитры компоненто*) представляет собой набор кнопок для перемещения по записям и их редактирования. В свойстве `DataSource` хранится имя компонента – источника данных, в свойстве `Hints` – массив строк, хранящий всплывающие подсказки для каждой из кнопок. Для активации требуется, чтобы свойству компонента было установлено значение `true`.

Компоненты DBEdit  и DBText  (панель **Data Controls** *Палитры компонентов*) – используются для просмотра и редактирования полей записи. Свойство DataField содержит имя поля для отображения/редактирования, в свойстве DataSource хранится имя компонента – источника данных.

Связь компонентов приложения и таблицы БД показана на рис. 8.3.

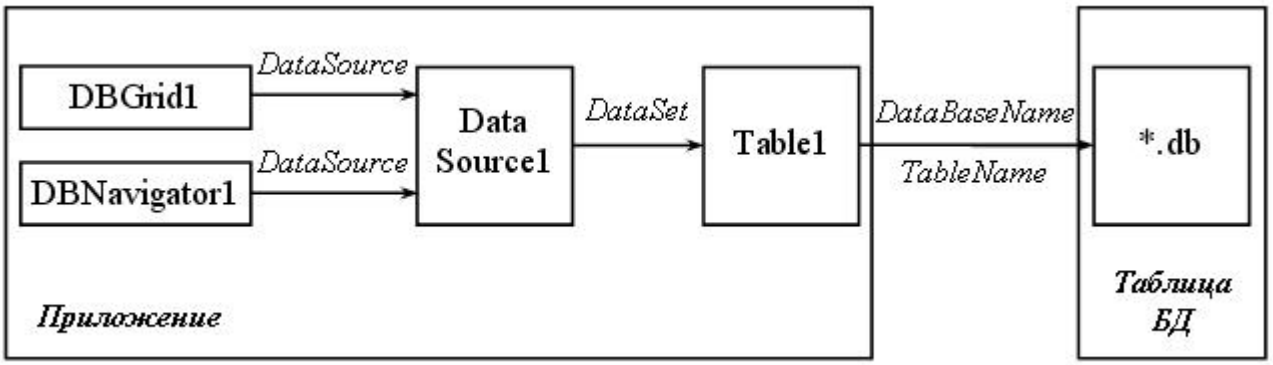

Рис. 8.3. Связь компонентов приложения и таблицы БД

8.3. Порядок выполнения работы

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

Контрольный пример 8.1

Разработать программу управления базой данных "Записная книжка". В базе должна 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 базы данных. В данном случае ввести путь c:Spravochnik или воспользоваться стандартным диалоговым окном ввода, находящемся 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. 8.4.

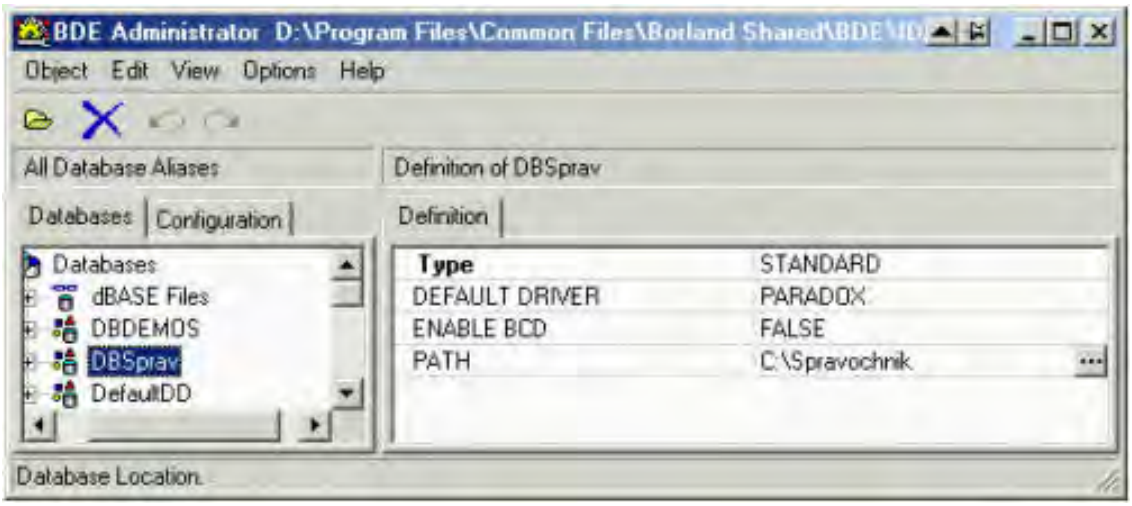

Рис. 8.4. Результат создания псевдонима

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. 8.5).

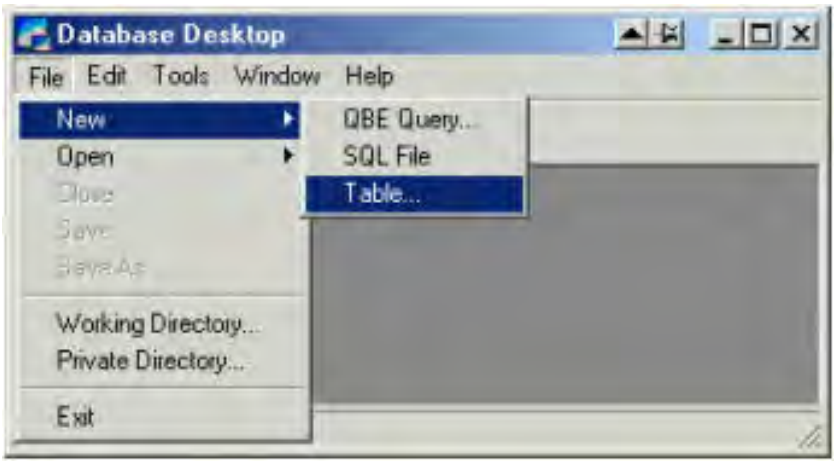

Рис. 8.5. Диалоговое окно Database Desktop

В открывшемся диалоговом окне **Create Table** выбрать тип создаваемой таблицы (Paradox 7) и нажать **OK**. В открывшемся диалоговом окне (рис. 8.6) можно определять структуру записей таблицы.

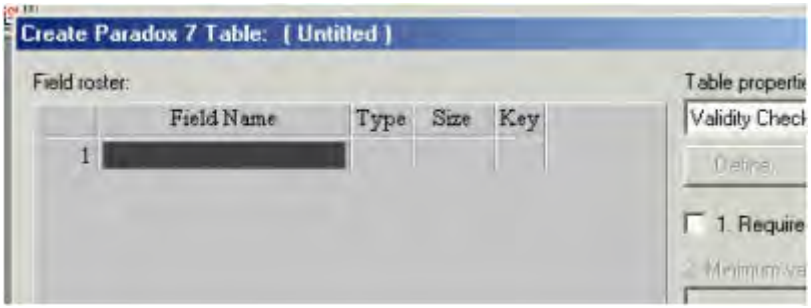

Рис. 8.6. Диалоговое окно **Create Paradox 7 Table**

В табл. 8.3 перечислены поля таблицы для базы данных "Записная книжка". Тип поля (**Type**) задается нажатием правой кнопки мыши или клавишей **Пробел**.

Таблица 8.3

Поля таблицы для базы данных "Записная книжка"

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		день 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 и 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. 8.7) 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
```

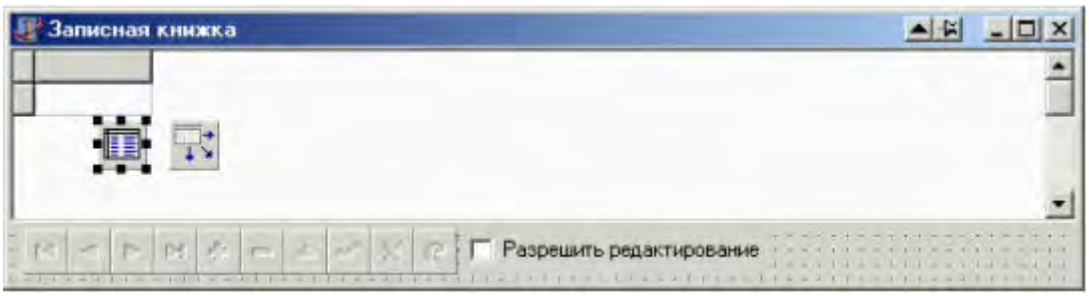

Рис. 8.7. Вид формы для контрольного примера 8.1

3. Для задания связей между компонентом `Table1` и таблицей БД, между компонентом `DataSource1` и `Table1`, между компонентами `DBGrid1`, `DBNavigator1` и `DataSource1`, необходимоо установить для этих компонентов следующие свойства:

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

4. Для возможности 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 (редактирования, создания новых, удаления) или запрещает редактировать содержание таблицы данных.

6. Для того, чтобы добавить новою запись, надо нажать кнопку и в новой строке начать ввод информации. Заполнить таблицу информацией, приведенной в табл. 8.4.

Таблица 8.4

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. 8.8.

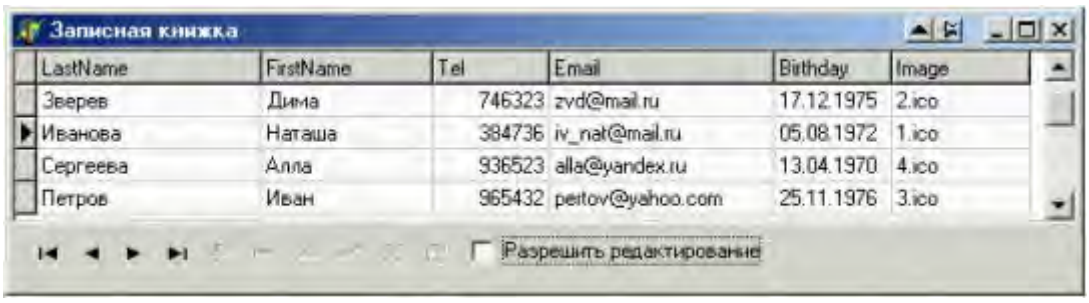

Рис. 8.8. Результат выполнения программы для контрольного примера 8.1

Контрольный пример 8.2

Дополнить имеющуюся программу компонентами, позволяющими просматривать и редактировать базу данных в режиме формы. Для управления базой данных 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`, `Button2` как показано на рис. 8.9 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
```

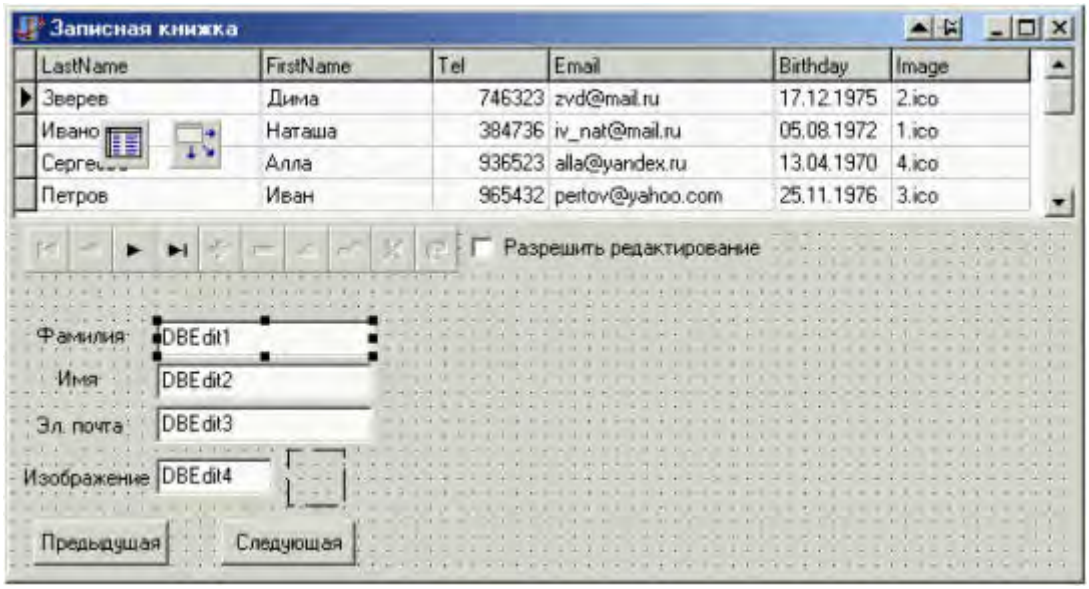

Рис. 8.9. Вид формы для контрольного примера 8.2

2. Для задания связей между компонентами, обеспечивающими просмотр и редактирование полей базы данных, и компонентом источника данных DataSourse1, установить для этих компонентов 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 данном случае, при нажатии на кнопки **Предыдущая** и **Следующая** будет происходить переход соответственно к предыдущей и следующей записям 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:SpravochnikImage**. 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 (имя диска, название каталога, имя 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. Тест соответствующей процедуры имеет вид:

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

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

В процедуре TForm1.Table1BeforeOpen использовалась функция 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 теперь можно использовать кнопки **Предыдущая** и **Следующая**. При этом в соответствующих компонентах будет отображаться содержимое

соответствующих полей конкретной записи. Окно приложения показано на рис. 8.10.

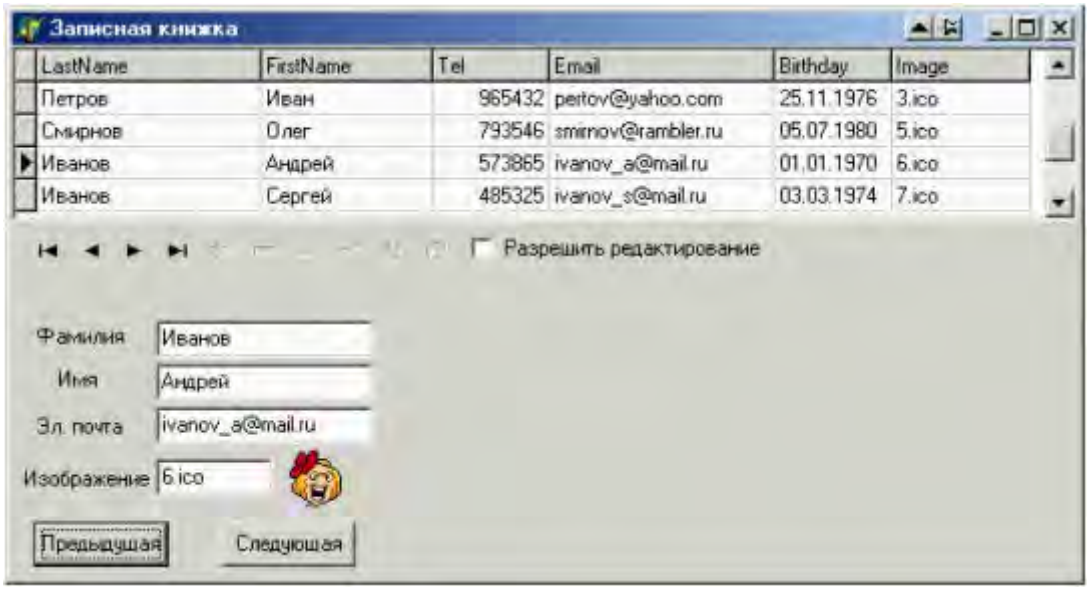

Рис. 8.10. Результат выполнения программы для контрольного примера 8.2

Контрольный пример 8.3

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. 8.11.

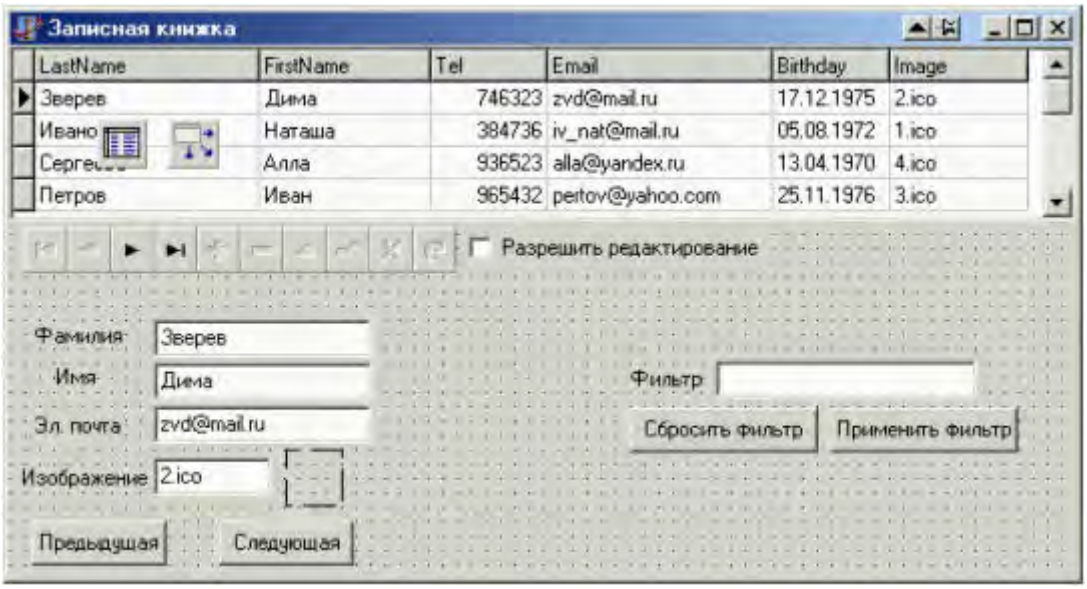

Рис. 8.11. Вид формы для контрольного примера 8.3

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. 8.12.

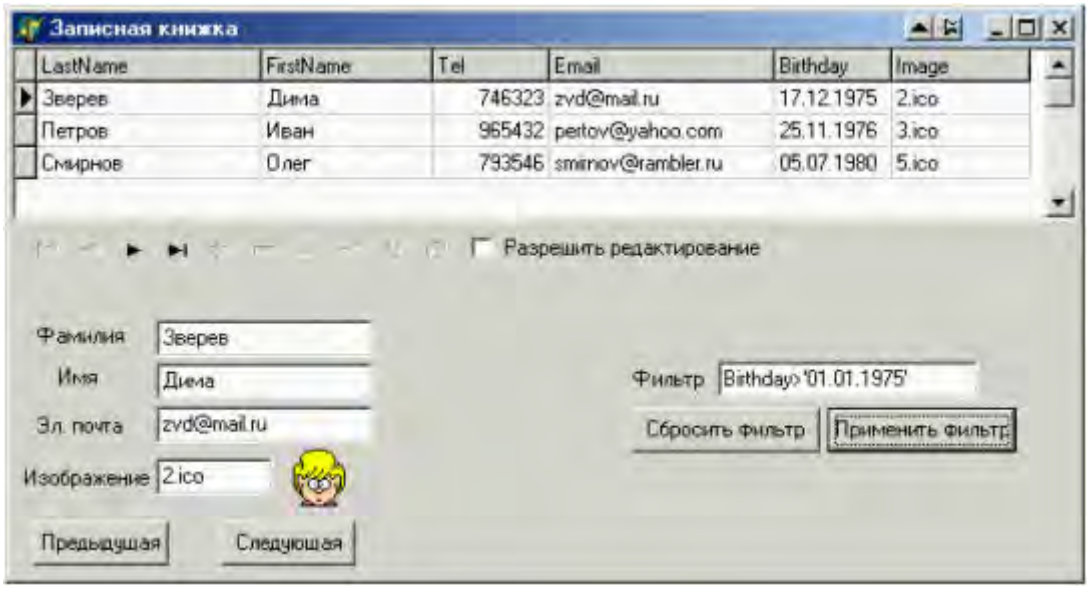

Рис. 8.12. Результат выполнения фильтрации

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

8.4. Содержание отчета

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

8.5. Контрольные вопросы

1. В чем заключается отличием 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 отличается просмотри БД в режиме таблицы и в режиме формы?
6. Что называется выборкой информации из БД?

8.6. Варианты заданий

Вариант 1

Разработать программу управления базой данных, содержащей информацию об учениках в классе, предметам и успеваемости. Обеспечить просмотр базы данных в режиме формы с возможностями навигации 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. Поля таблицы должны содержать информацию об авторах, названии и год издания. Обеспечить просмотр базы данных в режиме формы и возможность фильтрации исходных данных.

Вариант 5

Разработать программу управления базой данных, содержащей информацию о студентах в группе. Поля таблицы должны содержать информацию о фамилии, номере зачетной книжки, дне рождения и телефоне. Обеспечить просмотр базы данных в режиме формы и возможность фильтрации исходных данных.

Вариант 6

Разработать программу управления базой данных, содержащей информацию о учениках в школе. Поля таблицы должны содержать следующую информацию: фамилии учеников, рост, вес, пол. Обеспечить просмотр базы данных в режиме формы и возможность навигации по базе данных.

Вариант 7

Разработать программу управления базой данных, содержащей информацию об учетах разговоров по телефону. Поля таблицы должны содержать следующую информацию о дате звонка, фамилии звонившего и времени разговора. Обеспечить просмотр базы данных в режиме таблицы с возможностями навигации по базе данных и фильтрации исходных данных.

Л а б о р а т о р н а я р а б о т а № 9
АЛГОРИТМЫ СОРТИРОВОК МАССИВОВ ДАННЫХ

Цель работы: знакомство с основными алгоритмами сортировок массивов данных, классификация этих алгоритмов в соответствии с их эффективностью.

Используемые программные средства: Borland Delphi.

9.1. Теоретические сведения

9.1.1. Методы сортировки

Под *сортировкой* обычно понимают процесс целенаправленной перестановки элементов данных в определенном порядке по возрастанию или убыванию согласно определенным линейным отношениям их порядка, таким как отношения ≤ или ≥ для чисел. Цель сортировки – облегчить в дальнейшем поиск элемента в отсортированной (упорядоченной) совокупности элементов. Это напоминает поиск элемента в словаре, телефонном справочнике, ведомостях и т.д.

Методы сортировки необходимы при обработке данных. С сортировкой связаны многие фундаментальные приемы построения алгоритмов. Сортировка является идеальным примером огромного разнообразия алгоритмов, которые исполняют одну и тую же задачу, многие из которых в некотором смысле являются оптимальными, а большинство имеет некоторые преимущества по сравнению с остальными. Поэтому на примере сортировки можно убедиться в необходимости сравнительного анализа алгоритмов. Здесь можно увидеть, как при помощи усложнения алгоритма можно получить существенное увеличение эффективности при сравнении с более простыми и очевидными алгоритмами.

Зависимость выбора алгоритма от структуры данных – явление довольно частое, и в случае сортировки это особо сильно ощущается, поэтому методы сортировки обычно разделяют на две категории:

- сортировка данных типа массив;
- сортировка последовательных файлов.

Эти два класса часто называют *внутренней* и *внешней сортировкой*, поскольку массивы размещаются в оперативной памяти быстрого доступа к

каждому элементу, а последовательные файлы сохраняются в более медленной, но более емкой памяти, когда доступ имеем только к текущему элементу.

Сортировка строк – это ранжирование данных строкового типа по алфавиту (фактически по номеру составляющих строку символов в ASCII-коде).

Отметим, что сортировка строк или элементов файла с прямым (а не последовательным) доступом во всем напоминает сортировку массивов, поскольку доступ имеем к каждому элементу.

Введем систему обозначений, которую будем использовать в дальнейшем. Пусть нам даны элементы $a_1, a_2, \ldots, a_n$.

Сортировка по возрастанию (убыванию) означает перестановку этих элементов в таком порядке: $a_{k_1}, a_{k_2}, \ldots, a_{k_n}$, что при заданной функции упорядочения f справедливы отношения: $f(a_{k_1}) \le f(a_{k_2}) \le \ldots \le f(a_{k_n})$ (или наоборот $f(a_{k_1}) \ge f(a_{k_2}) \ge \ldots \ge f(a_{k_n})$).

Обычно функция упорядочения не подсчитывается по какому-то специальному правилу, а присутствует в каждом элементе в виде явной компоненты (поля элемента), которую называют ключом элемента. Таким образом, для представления элемента a_i особенно хорошо подходит структура записи. Для дальнейшего рассмотрения материала определим тип *item*, который будет использоваться в алгоритмах сортировки так:

```
Const n = ...;
Type inf = record
           ...
           end ;
item = record
           key : integer;
           Zmest : inf; {описание «других компонентов»}
           end;
Index = 0..n;
```

«Другие компоненты» – это все существенные данные об элементе. Поле *key* – ключ, и служит только для идентификации элемента. Но если мы говорим об алгоритмах сортировки, ключ для нас – единственная существенная компонента, и у нас нет необходимости определять остальные. Тип ключа может быть любым

типом, для которого заданы отношения всеобщего порядка «больше или равно» («меньше или равно»).

Введём еще такое обозначение:

```
Var A: array [1..n] of item;
```

Метод сортировки будем называть устойчивым, если относительный порядок элементов с одинаковыми ключами не изменяется при сортировке. Устойчивость сортировки часто бывает желательна, если элементы упорядочены (отсортированы) по каким-то второстепенным ключам, т.е. по свойствам, которые не отражены в первоначальном ключе.

Разберем подробно некоторые интересные алгоритмы сортировки.

9.1.2. Сортировки массивов

Основное требование для методов сортировки массивов – экономное использование памяти. Это значит, что перераспределение элементов нужно выполнять на том же месте {in site}, и поэтому методы, которые пересылают элементы из массива *A* в массив *B* для нас не представляют интереса.

Таким образом, выбирая метод сортировки и руководствуясь критериями экономии памяти, классификацию алгоритмов мы будем проводить в соответствии с их *эффективностью*, это значит *экономией времени* и *быстротой действия*. При этом будем подсчитывать *C – необходимое количество сравнений ключей* и *M – пересылок элементов* (что отнимает много времени). Это будут функции от *n* – числа сортируемых элементов.

Методы, сортирующие элементы in site, можно разбить на три основных класса в зависимости от заложенного в их основе приема (базового алгоритма):

- сортировка обменом;
- сортировка включениями;
- сортировка выбором.

Рассмотрим методы сортировки массивов в соответствии с этой классификацией и сразу будем отыскивать те подходы, которые позволяют улучшать базовый алгоритм.

При различных методах сортировки интерес будут вызывать такие вопросы:

1) Как ведет себя алгоритм в крайних ситуациях:
   - массив упорядочен в нужном порядке;
   - массив упорядочен в обратном порядке;
   - массив не упорядочен.
2) На каких массивах алгоритм дает:
   - min количество действий;
   - max количество действий;
   - чему равняется среднее количество действий.

9.2. Сортировки обменом

9.2.1. Сортировка простым обменом (метод пузырька)

Идея: *упорядоченный массив A получается из исходного массива A систематическим обменом пары рядом находящихся элементов, которые не соответствуют нужному порядку упорядочения, пока такие пары находятся при просмотре массива.*

Просмотр массива будем осуществлять справа налево (можно и наоборот). Этот алгоритм называется *сортировка «пузырьком»*, поскольку минимальный элемент как бы всплывает (как пузырек) в начало массива при просмотре этого массива.

Пример

Рассмотрим работу алгоритма на следующем примере (n = 8): 44, 55, 12, 42, 94, 18, 06, 67. В табл. 9.1 в столбцах отражены промежуточные состояния сортируемого массива.

Таблица 9.1

Пример сортировки методом пузырька

Начальные ключи	i = 2	i = 3	i = 4	i = 5	i = 6	i = 7	i = 8
44	06	06	06	06	06	06	06
55	44	12	12	12	12	12	12
12	55	44	18	18	18	18	18
42	12	55	44	42	42	42	42
94	42	18	55	44	44	44	44
18	94	42	42	55	55	55	55
06	18	94	67	67	67	67	67
67	67	67	94	94	94	94	94

Подсчитаем количество сравнений:

$$C = (n-1) + (n-2) + ... + 2 + 1 = \frac{n-1+1}{2}(n-1) = \frac{n^2 - n}{2} = O(n^2).$$

Количество пересылок (обменов):

$$M_{\min} = 0;\ M_{\max} = \frac{3}{2}(n^2 - n);\ M_{cp} = \frac{3}{4}(n^2 - n);\ M_{\max} = 3C.$$

Можно заметить, что массив был отсортирован на некотором промежуточном шаге, однако алгоритм построен так, что это явление в нем не анализируется.

9.2.2. Сортировка простым обменом с флагом

Идея: *этот метод является оптимизацией предыдущего метода – нужно запомнить, проводился ли при данном просмотре какой-нибудь обмен, и если нет, то можно досрочно завершить работу.*

9.2.3. Сортировка простым обменом с границей

Идея: *нужно запомнить не только факт обмена, но и запомнить индекс последнего обмена, так как понятно, что все пары соседних элементов с индексами меньшими этого индекса уже размещены в нужном порядке. Поэтому просмотр можно заканчивать на этом индексе вместо того, чтобы двигаться до конца.*

9.2.4. Шейкер-сортировка

Следующий пример: 94, 55, 10, 12, 08, 44, 06, 67 (n = 8) демонстрирует такое свойство, что при просмотре слева направо легкий элемент с каждым разом сдвигается на один шаг, а тяжелый сразу попадает в конец.

1-й просмотр: 55, 10, 12, 08, 44, 06, 67, 94.

2-й просмотр: 10, 12, 08, 44, 06, 55, 67, 94.

3-й просмотр: 10, 08, 12, 06, 44, 55, 67, 94.

Отсюда сразу виден следующий подход.

Идея: используем смену направлений просмотра (→,←), поскольку один плохо размещенный «пузырек», «тяжелый» в «легком» конце, будет опускаться на нужное место за один проход, а «легкий» пузырек в «тяжелом» конце будет опускаться каждый раз только на один шаг.

Пример

а) 94, 06, 12, 18, 42, 44, 55, 67; просмотр → сортирует массив за 1 проход;

б) 12, 18, 42, 44, 55, 67, 94, 06; тут понадобилось бы 7 проходов.

Схема работы алгоритма:

$\{a_1, a_2, a_3, ..., a_k, ..., a_n\}$

{1-й проход «←»} i:=n..2; id – индекс последней перестановки, значит, $\{a_1, a_2, a_3, ..., a_{id}\}$ уже упорядочен;

{2-й проход «→»} left:=id; i:=left..n; id – индекс последней перестановки, $\{a_{left}, ..., a_{id}\}$ – осталось упорядочить. И так далее.

9.2.5. Быстрая сортировка

Рассмотрим последовательность элементов, отсортированных в обратном порядке. Как их быстро отсортировать в нужном порядке? Правильно, нужно выполнить обмен равноудаленных от концов элементов. Это наводит на определенные мысли, и получается такой улучшенный алгоритм:

Идея: выбираем некоторый элемент x данных – опорный элемент. Двигаемся по совокупности элементов слева направо, пока не встретим элемент $a_i \geq x$, а потом – справа налево, пока не найдем элемент $a_j < x$. Если I < j, то

обменяем a_i *и* a_j. *Продолжаем просмотр далее от позиции* i+1 *до* j-1. *Выполняем такой поиск до тех пор, пока два просмотра не встретятся.*

Пример

Пусть в следующем примере: 55, 12, 42, 94, 06, 18, 44, 67 опорный элемент равен 42. Получим такую последовательность перемещений:

- 55, 12, 42, 94, 06, 18, 44, 67;
- 18, 12, 42, 94, 06, 55, 44, 67;
- 18, 12, 06 | 94, 42, 55, 44, 67.

Дальше просмотр слева направо заканчивается на элементе 94, а справа налево на элементе 6. Они не обмениваются. Получили две части последовательности:

- в одной числа меньшие, чем *x*;
- во второй числа не меньшие, чем *x*.

Поделив массив опорным элементом на две части, нужно сделать то же самое с каждой из них, а затем с частями этих частей и т.д., пока каждая часть не будет содержать только один элемент.

9.3. Сортировки включениями

9.3.1. Сортировка простым включением

<u>Идея:</u> элементы условно разделяются на готовую последовательность $a_1, a_2, ..., a_{i-1}$ *и входящую последовательность* $a_i, ..., a_n$. *На каждом шаге, начиная с* i=2,3,…, *берут* a_i *и вставляют его на подходящее место в готовую последовательность.*

Готовая последовательность $a_1, a_2, ..., a_{i-1}$ упорядочена в порядке роста. Элемент *x* нужно вставить в нее, не нарушая порядок, например, «просеять». Заметим, что «просеивание» может закончиться при двух условиях:

- найден элемент a_j такой, что $a_j < x \le a_{j+1}$, $1 \le j \le i-1$;
- достигли левого конца готовой последовательности.

Когда становится ясно, куда нужно вставить элемент *x*, нужно освободить для него место, перемещая элементы $a_{j+1},...,a_{i-1}$ на шаг вправо. Отметим, что это перемещение можно выполнять сразу при сравнении.

9.3.2. Сортировка бинарными вставками

Идея: *можно получить улучшение предыдущего алгоритма, если поиск места вставки элемента x проводить бинарными делениями.*

9.3.3. Сортировка Шелла

Следующий алгоритм представляет собой сортировку включениями с уменьшающимся приращением.

Идея: *Сначала отдельно группируются и сортируются все элементы, которые отстоят один от другого на 4 позиции. Это четвертная сортировка (выполняем, пока не упорядочим). Дальше – двоичная сортировка (сортируются элементы, которые отстоят на дистанции 2; выполняем до упорядочения). На последнем проходе – обычная сортировка (одинарная).*

На первый взгляд добавились 2 лишних шага (четвертная и двоичная сортировки), однако на каждом этапе либо сортируется относительно мало элементов, либо элементы уже довольно хорошо упорядочены.

Пример

Над следующим массивом выполним четвертную сортировку:

- 44, 55, 12, 42, 94, 18, 06, 67.

Результатом будет такой массив:

- 44, 18, 06, 42, 94, 55, 12, 67.

Выполним двоичную сортировку и получим следующий массив:

- 06, 18, 12, 42, 44, 55, 94, 67.

Выполним одинарную сортировку и получим полностью отсортированный массив:

- 06, 12, 18, 42, 44, 55 67, 94.

Заметим, что продвижение «легкого» элемента вперед идет довольно быстрыми темпами.

Очевидно, что этот метод дает в результате упорядоченный массив. Также ясно, что каждый проход будет использовать результаты предыдущего прохода, поскольку каждая *i*-я сортировка объединяет две группы, отсортированные предыдущей сортировкой. Анализ показал, что приемлема произвольная последовательность приращений, с условием, что последнее приращение равно 1, так как в худшем случае вся работа будет выполнена при последнем проходе. Отметим также, что, как показывает практика, если приращение не является степенью двойки, то можно получить гораздо лучшие результаты сортировки.

9.4. Сортировки выбором

9.4.1. Сортировка простым выбором

Идея: среди n элементов выбирается элемент с наименьшим ключом, и меняется местами с первым. Затем действие повторяется с оставшимися n-1 элементами, n-2 элементами и так далее, пока не останется один последний элемент.

Пример

i = 1 44, 55, 12, 42, 94, 18, 06, 67;
i = 2 06, 55, 12, 42, 94, 18, 44, 67;
i = 3 06, 12, 55, 42, 94, 18, 44, 67;
i = 4 06, 12, 18, 42, 94, 55, 44, 67;
i = 5 06, 12, 18, 42, 94, 55, 44, 67;
i = 6 06, 12, 18, 42, 44, 55, 94, 67;
i = 7 06, 12, 18, 42, 44, 55, 94, 67;
i = 8 06, 12, 18, 42, 44, 55, 67, 94.

9.4.2. Пирамидальная сортировка

Улучшить предложенный выше метод можно, если после каждого сравнения сохранять больше информации об элементах. Например, *1-й шаг: при помощи n/2 сравнений можно определить наименьший ключ из каждой пары; 2-й шаг: при помощи n/4 сравнений из выбранных ключей определить наименьший из каждой пары и так далее. Получим за меньшее количество сравнений наименьший из n элементов в корне дерева.*

На втором этапе мы спускаемся по пути, указанному наименьшим ключом и исключаем его, заменяя, например, на ∞. Дальше по той же схеме находим наименьший среди оставшихся элементов. После n таких шагов дерево становится пустым (состоит из дырок), значит процесс сортировки закончился.

Каждый из n шагов требует $\log_2 n$ сравнений (так как уменьшение шло каждый раз в два раза), тогда за n шагов получаем $n \cdot \log_2 n$ сравнений, в то время как первоначальный метод требует n^2 сравнений.

Однако реализация этого метода требует дополнительную память для хранения дерева – информации после каждого из сравнений. Поэтому нужно найти более удобный метод хранения дерева и каким-то образом освободиться от необходимости сохранять «дырки».

Оригинальный метод нашел Дж. Вильямс и назвал его *пирамидальной сортировкой*.

Пирамида определяется как последовательность ключей $h_l, h_{l+1}, ..., h_r$, такая, что выполняются условия:

$$\begin{cases} h_i \le h_{2i} \\ h_i \le h_{2i+1} \end{cases}, \qquad (*)$$

для всех $i = l,...,r/2$.

Отсюда $h_1 = \min(h_1,...,h_n)$. Фактически получается дерево, изображенное на рис. 9.1.

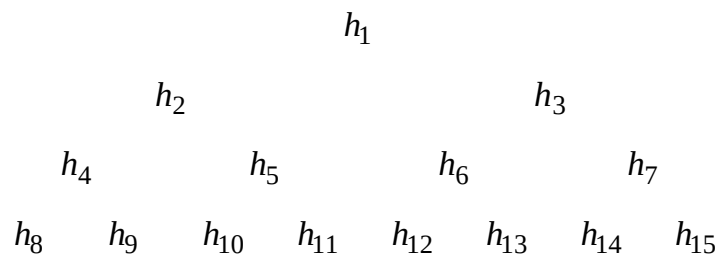

Рис. 9.1. Дерево

Встают вопросы: как строить и хранить пирамиду. Оказалось, что это можно делать in site.

Допустим, что задана пирамида с элементами $h_{l+1},...,h_r$, (l и r – фиксированные). Нужно добавить новый элемент x так, чтобы сформировать расширенную на этот элемент пирамиду $h_l,h_{l+1},...,h_r$. *Если новый элемент сначала добавить в вершину дерева, а затем «просеивать» по пути, на котором находятся меньшие в сравнении с ним элементы, которые одновременно поднимаются вверх, тогда сформируется новая (расширенная на один элемент) пирамида, так как при этом выполняется условие (*).*

Пример

Пусть задана пирамида $h_2,...,h_7$; $h_2 = 42$, $h_3 = 6$, $h_4 = 55$, $h_5 = 94$, $h_6 = 18$, $h_7 = 12$.

Нужно добавить в нее новый элемент 44. Берем $h_1 = 44$ (рис. 9.2).

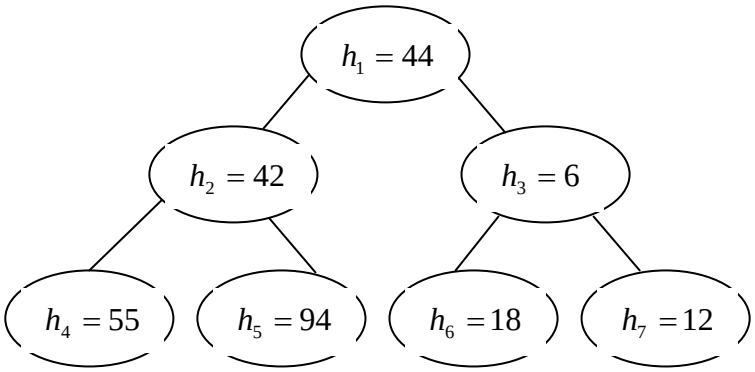

Рис. 9.2. Исходная расширенная пирамида

По условию (*) $h_1 = 44$ сравнивается с меньшим из $h_2 = 42$ и $h_3 = 6$. Это $h_3 = 6$, и он обменивается с $h_1 = 44$. Получится $h_1 = 6$ и $h_3 = 44$. Дальше $h_3 = 44$ сравнивается с меньшим из $h_6 = 18$ и $h_7 = 12$. Это $h_7 = 12$, и он обменивается с $h_3 = 44$. Получилась расширенная пирамида 6, 42, 12, 55, 94, 18, 44 (рис. 9.3).

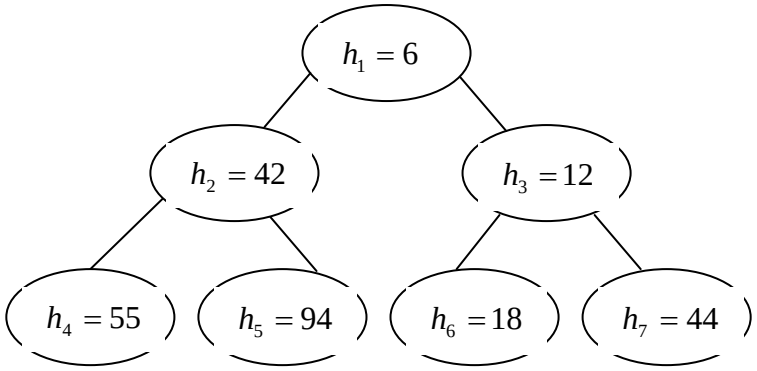

Рис. 9.3. Расширенная пирамида после проведения «просеивания»

Основываясь на этом приеме просеивания, красивый способ построения пирамиды in site был предложен Р.У. Флойдом. Рассмотрим его.

Дан массив $h_1,...,h_n$. Ясно, что элементы $h_{n/2+1},...,h_n$ уже образовывают пирамиду, поскольку для $n/2+1 \le i \le n$ тут не существует элементов с номерами *2i* и *2i+1*. (1-й этап). *Начиная с элемента i = n/2 с шагом -1 до 1 будем просеивать через предыдущую пирамиду очередной* h_i *элемент*. В конце этой процедуры в вершину пирамиды вытолкнется самый маленький элемент. Но последовательность еще не отсортирована. Сделаем следующее. (2-й этап). *Обменяем первый элемент с последним и просеем его через пирамиду, не трогая последнего. Новый первый элемент обменяем с предпоследним и просеем его через пирамиду, не трогая двух последних. И так сделаем n-1 раз.* Получим упорядоченный массив по спаданию значений.

Отобразим первый этап на следующем примере.

Пример

44 55 12 42 | 94 18 06 67 (h_4 сравнивается с h_8 и остается на месте);

44 55 12 | 42 94 18 06 67 (h_3 сравнивается с меньшим из h_6 и h_7, и обменивается с h_7);

44 55 | 06 42 94 18 12 67;

44 | 42 06 55 94 18 12 67;

06 42 12 55 94 18 44 67.

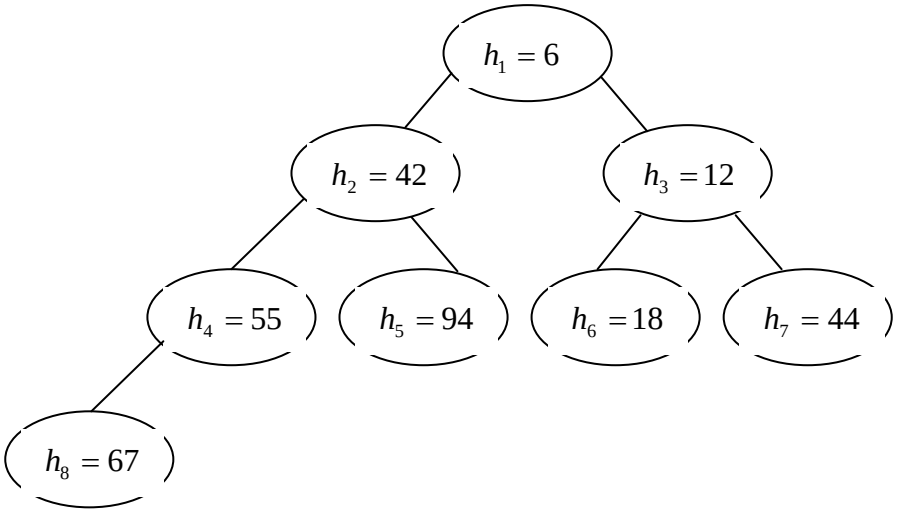

Рис. 9.4. Пирамида с самым маленьким элементом в вершине

Этим мы сформировали пирамиду и наименьший элемент вытолкнули наверх (рис. 9.4). Дальше будем обменивать его согласно этапу 2.

67 42 12 55 94 18 44 | 06 → 12 42 67 55 94 18 44 | 06 →
12 42 18 55 94 67 44 | 06 → 44 42 18 55 94 67 | 12 06 →
18 42 44 55 94 67 | 12 06 → 18 42 44 55 94 67 | 12 06 →
67 42 44 55 94 | 18 12 06 → 42 55 44 67 94 | 18 12 06 →
94 55 44 67 | 42 18 12 06 → 44 55 94 67 | 42 18 12 06 →
67 55 94 | 44 42 18 12 06 → 55 67 94 | 44 42 18 12 06 →
94 67 | 55 44 32 18 12 06 → 67 94 | 55 44 42 18 12 06 →
94 | 67 55 44 32 18 12 06.

Получили упорядоченный массив по убыванию значений.

Значит, чтобы получить упорядочение в другую сторону, нужно изменить условие (*) на противоположное.

9.5. Сравнительный анализ сортировок

Мы получили следующие алгоритмы.

I. Сортировки обменом:

- сортировка простым обменом (метод пузырька);
- сортировка простым обменом с флагом;
- сортировка простым обменом с границей;
- шейкер-сортировка;
- быстрая сортировка.

II. Сортировки включениями:

- сортировка простым включением;
- сортировка бинарными вставками;
- сортировка Шелла.

III. Сортировки выбором:

- сортировка простым выбором;
- пирамидальная сортировка.

Попробуем провести сравнительный анализ их эффективности.

Пусть n – по прежнему обозначает число сортируемых элементов, а C и M – соответственно количество необходимых сравнений ключей и пересылок элементов. Для всех трех простых методов сортировки можно дать замкнутые аналитические формулы. Они приведены в табл. 9.2 (заголовки столбцов Min, Max, Средн. определяют максимумы, минимумы и ожидаемые средние значения для всех $n!$ перестановок n элементов).

Таблица 9.2

Сравнение простых методов сортировки

Метод сортировки	Min	Средн	Max
Простые включения	$C = n - 1$ $M = 2(n - 1)$	$(n^2 + n - 2)/4$ $(n^2 - 9n - 10)/4$	$(n^2 - n)/2 - 1$ $(n^2 + 3n - 4)/2$
Простой выбор	$C = (n^2 - n)/2$ $M = 3(n - 1)$	$(n^2 - n)/2$ $n(\ln n + 0{,}57)$	$(n^2 - n)/2$ $n^2/4 + 3(n - 1)$
Простой обмен (метод пузырька)	$C = (n^2 - n)/2$ $M = 0$	$(n^2 - n)/2$ $(n^2 - n) \cdot 0{,}75$	$(n^2 - n)/2$ $(n^2 - n) \cdot 1{,}5$

Для усовершенствованных методов нет достаточно простых и точных формул (некоторые приблизительные оценки см. в приложении, таблица 4). Все, что можно сказать, это то, что стоимость вычислений равна $c_i \cdot n^{1.2}$ в случае сортировки Шелла и $c_i \cdot n \cdot \log n$ в случаях пирамидальной и быстрой сортировок.

Эти формулы дают лишь приблизительную оценку эффективности как функции от n; они допускают классификацию алгоритмов сортировки на простые (n^2) и усовершенствованные, или «логарифмические» ($n \cdot \log n$). Однако для практических целей полезно иметь некоторые экспериментальные данные, которые могут пролить свет на коэффициенты c_i, позволяющие проводить дальнейшую оценку различных методов.

Рассмотрим экспериментально полученные данные на примере массивов из 160, 2560 и 10240 элементов (см. приложение, таблицы 1-3). Массивы различной длины и различной начальной упорядоченности (по возрастанию, по убыванию, не упорядоченные) были отсортированы в порядке возрастания и убывания.

Анализируя полученные данные, нетрудно заметить, что сортировка методом пузырька (с улучшениями) определенно является наихудшей среди всех сравниваемых методов, и даже ее улучшенная версия – шейкер-сортировка – все-таки хуже, чем сортировка простыми включениями и простым выбором (кроме патологического случая сортировки уже рассортированного массива).

Очевидно также, что преимущество сортировки бинарными включениями по сравнению с сортировкой простыми включениями ничтожно, хотя оба этих метода заметно превосходят алгоритмы сортировки обменом.

Наиболее выгодными алгоритмами являются быстрая и пирамидальная сортировки, причем быстрая сортировка превосходит пирамидальную в отношении 2 к 3. Она сортирует массив с элементами, расположенными в обратном порядке практически так же, как уже рассортированный массив.

Результаты практически того же порядка показывает сортировка Шелла, особенно в случае уже рассортированного массива. Однако в случае неупорядоченного массива ее эффективность заметно снижается при увеличении количества элементов массива.

9.6. Контрольные вопросы

1. Что такое внешняя, и что такое внутренняя сортировка?
2. Какие алгоритмы сортировки называются устойчивыми?
3. Что положено в основу сортировки строк символов?
4. Чем характеризуются методы сортировки массивов in site?
5. Перечислите алгоритмы сортировки из группы сортировка массивов обменом.
6. Перечислите алгоритмы сортировки из группы сортировка массивов выбором.
7. Перечислите алгоритмы сортировки из группы сортировка массивов включениями.

Приложения

Приложение А

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 включает 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 включает 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 данных представляет собой значения, которые могут использоваться в арифметических выражениях и занимать в памяти от 1 до 4 байт (табл. А.1)

Таблица А.1

Целочисленные типы

Тип	Диапазон	Представление в памяти, байт
`shortint`	-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 "+" (плюс) и "-" (минус). 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 данных представляют собой вещественные значения, которые используются в арифметических выражениях и могут занимать в памяти от 4 до 6 байт (табл. А.2).

Таблица А.2

Вещественные типы

Тип	Минимальное значение	Максимальное значение	Точность (число цифр мантиссы)	Память, байт
`real`	2,9·10 -39	1,7·10 38	11-12	6
`single`	1,79·10 -45	3,4·10 38	7-8	4
`double`	5,0·10 -324	1,7·10 308	15-16	8
`extended`	3,6·10 -4951	1,1·10 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 (логический) 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. Они 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`. Тип `shortsting` (`string`) представляет собой *статически* размещенные в памяти компьютера строки длиной от 0 до 255 символов. Типы `longstring` и `widestring` представляет 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 множества 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 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 ";" (точка с запятой).

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 использованием 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 использованием 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 содержащих 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 вычисляется 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 ограниченную 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 от друга, при этом на глубину вложенности составных операторов ограничений не накладывается.

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

Формат:

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

Если условие истинно (равно `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*), 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 направления, используются несколько условных операторов, вложенных друг в друга.

Пример:

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

Оператор выбора 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, отделенных 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 со счетчиком (с параметром) имеет два формата:

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

или

```
for <параметр_2> := <выражение_3> downto <выражение)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 выражения2 или выражение_3 меньше выражения_4 . Циклы разрешено вкладывать один в другой.

Пример:

```
for k:=1 to 10 do k:=k+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 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 ключевыми 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 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 перед первым выполнением 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 он расположен. 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 и путь к нему.

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`** обеспечивает ввод числовых данных, символов, строки 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`** производит 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 в консольном режиме.

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 файл .

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

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 используются зарезервированное слово `procedure`.

Формат:

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

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

Формат:

```
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, 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**. Параметр-значение обрабатывается 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;
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 Приложении Б.

Классы. Объектно-ориентированное программирование (ООП)

Объектно-ориентированное программирование (ООП) – это методика разработки программ, в основе которое 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** заключается в порождении 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** заключается 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 *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 как 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 заключается в том, что для каждого объекта можно выполнить только определенный набор операций, зависящий от типа этого объекта.

Большинству методов пи вызове передаются параметр `Sender`, имеющий тип `TObject`. Для выполнения с этим параметром операций, его тип необходимо преобразовать к типу того объекта, для которого выполняются эти операции. Для работы с типами в *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 Б

Перечень основных встроенных процедур и функций

Таблица Б.1

Арифметические процедуры и функции

`Abs(x)`	вычисление абсолютной величины (модуля) числа `x`
`arctan(x)`	вычисление угла, тангенс которого равен `x`
`cos(x), sin(x)`	вычисление косинуса и синуса `x`
`Exp(x)`	вычисление экспоненциальной функции e^x
`frac(x)`	вычисление дробной части числа `x`
`int(x)`	вычисление целой части числа `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.

Таблица Б.2

Функции преобразования типов для работы со скалярными переменными

`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`

Таблица Б.3

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`

Таблица Б.4

Функции преобразования типов для работы со строковыми переменными

`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`

Таблица Б.5

Форматы изображения числа

`ffExponent`	научный формат
`ffFixed`	формат с десятичной точкой
`ffGeneral`	общий цифровой формат
`ffNumber`	числовой формат
`ffCurrecy`	денежный формат

Приложение В

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 значений, приведенных в табл. В.1.

Таблица В.1

Значения свойства `Align`

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 и 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 значений, приведенных в табл. В.2.

Таблица В.2

Значения свойства `Color`

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 в таблице цветов, 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 , 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`, компонент реагирует на сообщения от мыши, клавиатуры и таймера. В противном случае (значение `False`) эти сообщения игнорируются.

Свойство `Font` – определяет шрифт текста, отображающегося на физуальном 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 определяет, 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 Г

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 необходимо

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

Пуск – Программы – Borland Delphi – Delphi .

2. В меню Delphi выполнить следующие действия:

File – New .

3. В открывшемся окне выбрать– **Console Application** .

В результате создается новый проект, состоящий из одного файла с расширением **pr** . Этот файл и является консольной программой. Первоначально он содержит следующий код:

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

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

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

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

ЛИТЕРАТУРА

1. Архангельский, А.Я. Delphi 6: Справочное пособие / А.Я. Архангельский. – М.: БИНОМ, 2001. – 1024 с.

2. Бобровский, С.И. Delphi 7. Учебный курс / С.И. Бобровский. – Санкт-Петербург: ПИТЕР, 2008. – 736 с.

3. Информатика. Учебник для вузов. Базовый курс / Под ред. С.В. Симоновича. – 2-е изд. Санкт-Петербург: ПИТЕР, 2000. – 640 с.

4. Культин, Н.Б. Основы программирования в Delphi 7 / Н.Б. Культин. – СПб.: БХВ-Петербург, 2009. – 640 с.

5. Фаронов, В.В. Delphi. Программирование на языке высокого уровня. Учебник для вузов / В.В. Фаронов. – Санкт-Петербург: ПИТЕР, 2003. – 640 с.

6. Хомоненко, А.Д. Самоучитель Delphi / А.Д. Хомоненко А., В.Э. Гофман. – Санкт-Петербург: БХВ-Петербург, 2008. – 576 с.