ВЫСШЕЕ ОБРАЗОВАНИЕ

серия основана в 1996 г.

Г.М. ЭЙДЛИНА
К.А. МИЛОРАДОВ

DELPHI:
ПРОГРАММИРОВАНИЕ В ПРИМЕРАХ И ЗАДАЧАХ.
ПРАКТИКУМ

Учебное пособие

Соответствует Федеральному государственному образовательному стандарту 3-го поколения

Москва
РИОР
ИНФРА-М

УДК 681.3.068+800.92(075.8)
ББК 32.973.26-0.18я73
Э11

Рецензент: Волков А.К. — к.т.н., профессор

Эйдлина Г.М., Милорадов К.А.

Э11 Delphi: программирование в примерах и задачах. Практикум: Учеб. пособие. — М.: РИОР: ИНФРА-М, 2012. — 116 с. — (Высшее образование: Бакалавриат).

ISBN 978-5-369-01084-6 (РИОР)
ISBN 978-5-16-006045-3 (ИНФРА-М)

Излагаются основные приемы разработки программного обеспечения с помощью Delphi. Рассмотрены примеры разработки интерактивных Windows-приложений и приложений баз данных. Приводятся задачи и упражнения для самостоятельной работы.

Адресовано студентам экономических специальностей и всем читателям, начинающим изучение программирования в Delphi.

УДК 681.3.068+800.92(075.8)
ББК 32.973.26-0.18я73

ISBN 978-5-369-01084-6 (РИОР)
ISBN 978-5-16-006045-3 (ИНФРА-М)

ВВЕДЕНИЕ

В настоящее время уже практически нет области человеческой деятельности, где бы не нашли свое применение компьютеры. Поэтому как никогда необходима разносторонняя подготовка специалистов по использованию компьютеров.

Безусловно, удобно, когда для решения разнообразных профессиональных задач предлагаются готовые программные средства, и по этому пути идет внедрение вычислительной техники. Однако даже в этом случае пользователю необходимо представление о том, как рождаются программы, он должен быть знаком с принципами устройства и работы ПК и основами программирования.

С другой стороны, для решения далеко не всех вычислительных и логических задач существуют готовые программные средства. В этой связи пользователям довольно часто приходится самостоятельно программировать решение своих задач, а не искать исполнителей в сфере профессиональных программистов.

Студенты экономико-математического факультета направления «Экономика» (профиль «Математические методы в экономике») и направления «Прикладная математика и информатика» общего профиля углубленно изучают математические дисциплины, занимаются экономико-математическим моделированием, и не всегда для решения тех или иных задач существуют прикладные пакеты. Очень часто студентам приходится самостоятельно составлять программы для решения задач, особенно, если они участвуют в каких-либо научных исследованиях. Нередко им приходится сталкиваться с проблемами программирования и после окончания обучения на рабочих местах.

Задания, предлагаемые в данном пособии, содержат набор задач, предназначенных для отработки основных приемов программирования. Примеры программ написаны на Delphi 7.

Практикум состоит из шести разделов. В первом разделе изложен теоретический материал, дано описание системы программирования Delphi, структуры программного проекта в Delphi.

В втором разделе практикума приведены лабораторные работы, нацеленные на изучение языка программирования Delphi и приемов разработки интерактивных Windows-приложений с использованием стандартных компонентов VCL, и методические указания к их выполнению.

В третьем разделе практикума приведены лабораторные работы, ориентированные на изучение приемов разработки приложений для работы с базами данных, и методические указания к их выполнению.

После выполнения лабораторных работ рекомендуется ответить на контрольные вопросы.

Четвертый раздел практикума содержит задачи и упражнения для самостоятельной работы.

Пятый раздел практикума содержит примеры программ для выполнения упражнений из раздела 4.

Шестой раздел практикума содержит примерную тематику курсовых работ.

Список литературы содержит перечень книг, успешно зарекомендовавших себя в учебном процессе.

Важным источником знаний для программистов является Интернет. Перечисленные в практикуме интернет-ресурсы окажутся полезными программистам с разным уровнем подготовки.

Не следует забывать и о таком важном источнике информации для программиста, как электронная справочная система Delphi.

В тексте практикума даются общепринятая терминология и обозначения. Для обозначения клавиш, названий команд меню и кнопок используется полужирный шрифт. Данные, задаваемые пользователем, выделены курсивом. Следует обратить внимание, что в некоторых листингах исходный код на языке программирования Delphi может не умещаться на одной строке.

1 . СИСТЕМА ПРОГРАММИРОВАНИЯ DELPHI

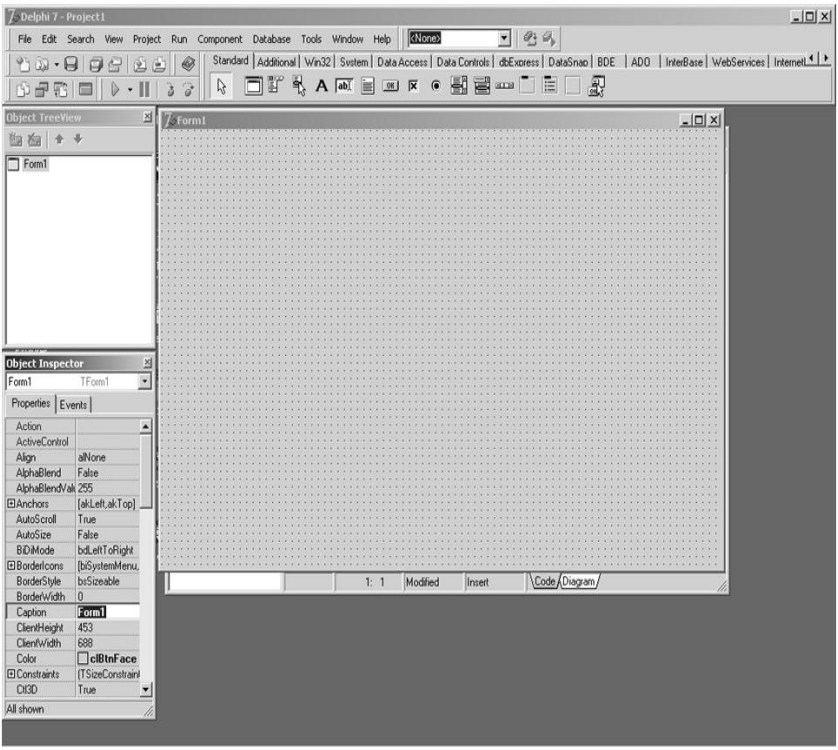

Рис. 1.1. Вид окна Delphi после запуска

Процесс создания программы в среде Delphi разбивается на две фазы: фазу конструирования формы и фазу кодирования.

Конструирование формы осуществляется с помощью выбора компонентов из палитры и размещения их на форме.

Программист может перемещать любой размещенный на форме компонент и изменять его размеры с помощью мыши. Чтобы придать компоненту нужные свойства, используется страница Properties Инспектора объектов. Небольшой знак «+» слева от названия свойства указывает, что данное свойство имеет подсвойства. При двойном щелчке на таком свойстве открывается и закрывается список этих подсвойств. Кнопка подстановки, расположенная справа от значения свойства, указывает, что при двойном щелчке на значении или самой кнопке можно открыть диалоговое окно, позволяющее

изменить значение свойства. Некоторые свойства имеют и подсвойства, и кнопки подстановки.

Чтобы компонент мог откликаться на то или иное событие, программист должен создать обработчик события и указать его имя на странице **Events** Инспектора объектов.

События определяют действия объекта во время работы программы. При двойном щелчке на значении события в исходный код программы будет вставлена заготовка обработчика события. Обработчик события оформляется в виде процедуры, имеющей составное имя. Первая часть имени представляет собой имя класса для формы, вторая часть отделяется от первой точкой и может быть произвольной. Если Delphi автоматически формирует заготовку для обработчика, то вторая часть имени представляет собой объединение имени компонента и имени события без предлога ON.

Тело процедуры ограничено словами **begin … end** и состоит из отдельных предложений (операторов) языка Delphi. В конце каждого предложения ставится точка с запятой.

Свойства компонента могут изменяться внутри работающей программы.

Формы

Каждый проект обладает, по меньшей мере, одной формой, являющейся главным окном программы. Но проекты могут иметь несколько форм.

Форма является компонентом, но она не присутствует в палитре компонентов. Форма создается при выполнении нового проекта или команды **File — New Form**.

Для каждой формы имеется соответствующий модуль, содержащий код формы. Значение свойства **Name** и имя файла модуля формы должны различаться. (Совет: для **Name** используйте имена, подобные **EntryFrm**, а для файла модуля — **Entry.pas**).

Форма — это не просто реализация окна программы. Она обладает свойствами и событиями, которые можно программировать для определения характеристик окна и выполнения действий.

Шаблоны форм полезны для создания часто используемых экранных форм, прототипов и демонстрационных программ. Вы можете выбрать один из готовых шаблонов форм или образовать собственный.

Любую форму можно сделать главным окном приложения. Например, в качестве простого и ясного интерфейса приложения, не требующего полосы меню, можно использовать диалоговое окно.

Событие **OnCloseQuery** удобно для подтверждения закрытия окна или отказа от него. Употребляйте данное событие для предотвращения потери данных.

Для того чтобы окно оставалось на переднем плане (поверх других окон приложения), свойству **FormStyle** формы данного окна присвойте значение *fsStayOnTop*. Это можно сделать в окне Object Inspector или во время работы программы.

Обычно Delphi автоматически создает экземпляры всех форм во время запуска программы. В больших приложениях это сказывается на памяти и других ресурсах. Имя формы следует удалять из списка автосоздания (Auto-create forms), чтобы рационально использовать память. Выберите **View — Project Manager — Options — Forms — SplashForm** и щелкните на кнопке с изображением направленной вправо стрелки, чтобы перевести данную форму в Available forms.

Структура программ Delphi

Любая программа в Delphi состоит из файла проекта (.DPR) и одного или нескольких модулей, содержащих исходный текст программы (файлы с расширением .PAS). Каждый из таких файлов описывает программную единицу Object Pascal. В состав проекта также входят различные файлы ресурсов, например файлы с расширением RES. Поэтому перед созданием нового приложения следует создавать отдельную папку и все файлы проекта сохранять в отдельной папке.

Структура проекта

Файл проекта представляет собой программу, написанную на языке Object Pascal и предназначенную для обработки компилятором. Эта программа автоматически создается Delphi и содержит лишь несколько строк. Чтобы увидеть их, запустите Delphi и щелкните по опции **View — Project Source** главного меню. Delphi покажет окно кода с закладкой Project1, содержащее такой текст:

```
Program Project1;
Uses
    Forms,
    Unit1 in 'Unit1.pas' {fmExample};
{$R *.res}
begin
    Application.Initialize;
    Application.CreateForm (TfmExample, fmExample);
    Application.Run;
End.
```

В окне кода жирным шрифтом выделяются так называемые зарезервированные слова.

Комментарии в Object Pascal ограничиваются следующими символами:

(*…*)

{…}

//, — указывают, что комментарий располагается за ними и продолжается до конца текущей строки.

Модуль Forms является стандартным, а модуль Unit1 — новым, ранее неизвестным, и Delphi в этом случае указывает также имя файла с текстом модуля.

В теле программы — три исполняемых оператора, каждый из них реализует обращение к одному из методов объекта Application.

Объекты

Объектом называется специальным образом оформленный фрагмент программы, заключающий в себе данные и подпрограммы для их обработки. Данные называются полями объекта, а подпрограммы — его методами. Объект в целом предназначен для решения какой-либо конкретной задачи и воспринимается в программе как неделимое целое. Объекты придуманы для того, чтобы увеличить производительность труда программиста и одновременно повысить качество разрабатываемых им программ. Два главных свойства объекта — функциональность и неделимость — делают его самостоятельной или даже самодостаточной частью программы и позволяют легко переносить объект из одной программы в другую. Разработчики Delphi придумали сотни объектов, которые можно рассматривать как кирпичики, из которых программист строит многоэтажное здание программы. Такой принцип построения программ называется ООП (объектно-ориентированное программирование).

В объекте **Application** собраны данные и подпрограммы, необходимые для нормального функционирования Windows-программы в целом. Delphi автоматически создает объект-программу **Application** для каждого нового проекта. С помощью его метода **Initialize** программа осуществляет ряд вспомогательных действий, необходимых для работы под управлением операционной системы Windows.

Строка **Application.Initialize**; означает обращение к методу **Initialize** объекта **Application**.

Метод **CreateForm** объекта **Application** создает и показывает на экране окно главной формы, а метод **Run** реализует бесконечный цикл получения и обработки поступающих от Windows сообщений о действиях пользователя. Когда пользователь щелкнет по кнопке **Close**, Windows передаст программе специальное сообщение, кото-

рое заставит программу прекратить работу и освободить назначенные ей системные ресурсы (память и процессор).

Файл проекта полностью формируется самой Delphi и не предназначен для редактирования.

Структура модуля

Модули — это программные единицы, предназначенные для размещения фрагментов программ. С помощью содержащегося в них программного кода реализуется вся поведенческая сторона программы. Любой модуль имеет следующую структуру:

Заголовок
Секция интерфейсных объявлений
Секция реализаций
Терминатор (это **end.**)

Следующий фрагмент программы является синтаксически правильным вариантом модуля:

```
Unit Unit1;
Interface
// Секция интерфейсных объявлений
implementation
// Секция реализаций
begin
// Секция инициации
end.
```

В секции интерфейсных объявлений описываются типы, классы, процедуры и функции, которые будут «видны» другим программным модулям, а в секции реализаций раскрывается механизм работы этих элементов. Разделение модуля на две секции обеспечивает удобный механизм обмена алгоритмами между отдельными частями одной программы. Он также реализует средство обмена программными разработками между отдельными программистами. Получив «посторонний» модуль, программист получает доступ только к его интерфейсной части, в которой содержатся объявления элементов. Детали реализации объявленных процедур, функций, классов скрыты в секции реализаций и недоступны другим модулям.

Секция инициации используется крайне редко и обычно опускается вместе с открывающим ее словом **begin**. Если щелкнуть по закладке **Unit1** окна кода, то увидите такой текст:

```
Unit Unit1;
Interface
Uses
```

```
  Windows, Messages, SysUnils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls;
Type
  TfmExample1 = class (TForm)
    Panel1: TPanel1;
    Panel2: TPanel2;
    EdInput: TEdit;
    LbOutput: TLabel;
    MmOutput: TMemo;
    BbRun: TBitBth;
    BbClose: TBitBth;
Private
{ Private declarations}
public
{ public declarations}
end;
var
  fmExample1: TfmExample1;
implementation
{$R *.DFM}
end.
```

Весь этот текст сформирован Delphi, но в отличие от файла проекта программист может его изменять, придавая программе нужную функциональность.

В интерфейсной секции описан один тип (класс **TfmExample1**) и один объект (переменная **fmExample1**).

Вот описание класса:

```
Type
  TfmExample1 = class (TForm)
Panel1: TPanel1;
    Panel2: TPanel2;
    EdInput: TEdit;
    LbOutput: TLabel;
    MmOutput: TMemo;
    BbRun: TBitBth;
    BbClose: TBitBth;
Private
{ Private declarations}
public
{ public declarations}
end;
```

Классы

Классы являются основным инструментом реализации мощных возможностей Delphi. Классами в Delphi называются функционально законченные фрагменты программ, служащие образцами для создания подобных себе экземпляров. Однажды создав класс, программист может включать его экземпляры (копии) в разные программы или в разные места одной и той же программы. Такой подход способствует максимально высокой продуктивности программирования за счет использования ранее написанных фрагментов программ. В состав Delphi входят более сотни классов, созданных программистами корпорации Borland (так называемых стандартных классов). Совокупность стандартных классов определяет мощные возможности этой системы программирования.

Класс является образцом, по которому создаются объекты, и наоборот, объект — это экземпляр реализации класса. Образцы для создания элементов программы в Object Pascal называются типами, таким образом, класс — это тип, определяемый пользователем (разработчиком). Перед его объявлением стоит зарезервированное слово **TYPE** (тип), извещающее компилятор о начале раздела описания типов.

Стандартный класс **TForm** реализует все нужное для создания и функционирования пустого Windows-окна. Класс **TfmExamples1** порожден от этого класса, о чем свидетельствует строка (она создает экземпляр **TForm**):

TfmExamples1 = class (Tform),

в которой за зарезервированным словом **class** в скобках указывается имя родительского класса. Термин «порожден» означает, что класс **TfmExamples1** унаследовал все возможности родительского класса **TForm** и добавил к ним собственные в виде дополнительных компонентов, которые мы вставили в форму **fmExample**. Перечень вставленных нами компонентов и составляет значительную часть описания класса.

Свойство наследования классами-потомками всех свойств родительского класса и обогащения их новыми возможностями является одним из фундаментальных принципов ООП. От наследника может быть порожден новый наследник, который внесет свою лепту в виде дополнительных программных заготовок и т.д. В результате создается ветвящаяся иерархия классов, на вершине которой располагается самый простой класс **TObject** (все остальные классы в Delphi порождены от этого единственного прародителя), а на самой нижней ступени иерархии — мощные классы-потомки, которым по плечу решение любых проблем.

Объект **fmExample** формально относится к элементам программы, которые называются переменными (**VARables**).

Следует отметить, что текст модуля доступен как Delphi, так и программисту. Delphi автоматически вставляет в текст модуля описание любого добавленного к форме компонента, а также создает заготовки для обработчиков событий; программист может добавлять свои методы в ранее объявленные классы, наполнять обработчики событий конкретным содержанием, вставлять собственные переменные, типы, константы и т.д. совместное с Delphi владение текстом модуля будет вполне успешным, если программист будет соблюдать простое правило: он не должен удалять строки, которые вставлены не им, а Delphi.

2. РАЗРАБОТКА ИНТЕРАКТИВНЫХ WINDOWS-ПРИЛОЖЕНИЙ

Лабораторная работа 2.1. Основные приемы визуального программирования

Настройка среды Delphi

1. Создайте на рабочем диске папку Prog1.

2. Загрузите Delphi (**Пуск — Программы — Borland Delphi 7 — Delphi 7**).

3. Выберите команду **Tools — Environment Options**, закладка **Preferences**. Установите флажок **Editor Files** в группе **Autosave Options** для автоматического сохранения текста кода программы и флажок **Show Compiler progress** в группе **Compiling and Running** для просмотра процесса компиляции. Для ввода в текст комментариев на русском языке установите шрифт *Courier New Cyr* (команда **Tools ─ Editor Options**, закладка **Display**, список **Editor Font**).

4. Выберите команду **File — New — Application**.

Изменение заголовка окна программы

5. Щелкните по закладке **Properties** окна **Инспектора объектов**.

6. В левой колонке выберите свойство **Caption** (заголовок).

7. Введите новое имя "*Моя первая программа*" в выделенной области правой колонки.

8. Нажмите **Enter**.

9. Сохраните проект в папке Prog1 с помощью кнопки **Save all** на панели инструментов главного окна. На запрос об имени файла введите имя *main*. Расширение .pas автоматически добавится. Затем Delphi запросит имя проекта. Введите *prg1*. Автоматически добавится расширение **.dpr**.

Запуск приложения

10. Нажмите **F9** или выберите команду **Run** из меню (или щелкните на кнопке **Run**). Не обязательно завершать проект перед его запуском. На экране появится результат вашей работы.

11. Выход из приложения — один из трех способов:

– двойной щелчок на кнопке системного меню;
– команда **Закрыть (Close)** из системного меню;
– щелчок на кнопке закрытия окна в верхнем правом углу.

После запуска приложений из среды Delphi их следует закрывать, чтобы иметь возможность вернуться к режиму программирования.

Размещение нового компонента

12. В случае если вы закрыли проект prg1.dpr, откройте его командой **File — Open** (или кнопкой **Open Project**).

13. Выберите страницу (закладку) **Standard** на палитре компонентов и щелкните по кнопке A, которая отображает объект **Label** (метка), предназначенный для размещения надписи.

14. Разместите компонент в окне форм, щелкнув мышью левее и выше его центра. Протяните и отпустите мышь.

15. Введите надпись «Я программирую на Delphi» с помощью строки **Caption** (заголовок) инспектора объектов.

16. Измените шрифт и цвет надписи (на закладке **Properties** окна инспектора объектов выберите свойство **Font** и с помощью кнопки раскройте диалоговое окно настройки шрифта. В списке **Size** (размер) выберите высоту шрифта в 24 пункта, а в списке **Color** (цвет), выберите цвет, например красный.

17. Щелкнув мышью по надписи в окне форм и не отпуская левую кнопку мыши, переместите прямоугольник с надписью в центр окна. Таким образом, вы измените расположение компонента **Label** (свойства **Left —** слева и **Top —** сверху).

18. Используя черные квадратики, обрамляющие выделенную мышью надпись, измените размер надписи (свойства **Width —** ширина и **Height —** высота). Нажмите **F9**.

19. Сделайте надпись невидимой. Для этого присвойте свойству **Visible** значение *False*. Результат увидите только при запуске приложения.

Обработка событий

20. Добавьте к программе новый визуальный компонент **BitBtn** (кнопка с графическим изображением), нажав пиктограмму **OK** на странице **Additional** палитры компонентов.

21. Выберите тип кнопки (свойство **Kind**, затем в появившемся меню выберите *bkClose*).

22. Измените имя компонента, выбрав свойство **Name** и введя **btClose**.

23. Чтобы при щелчке по кнопке всегда осуществлялось завершение программы, надо в окне редактирования изменить код модуля **Main.pas**, добавив в тело процедуры с именем

TForm1.btCloseClick (Sender : TObject);

(между **begin** и **end**) оператор *Close*. Для этого щелкните дважды по кнопке и введите оператор в окно кода программы.

24. С помощью свойства **Caption** измените надпись на кнопке, введя текст «Выход».

25. Создайте новую кнопку (страница **Standard**, пиктограмма **OK**). Присвойте ей имя *btShow*. При нажатии кнопки надпись «Я программирую на Delphi» станет видимой, но для этого в окне **Object TreeView** дважды щелкните мышью по новой кнопке и

вставьте в тело процедуры TForm1.btShowClick (Sender:TObject) следующую строку:

Label1.Visible:=True;

26. Расположите новую кнопку в окне формы, используя страницу **Standard**. Измените имя компонента на *btBeep*.

27. Сделайте эту кнопку «звучащей» (обработайте щелчок мышью по кнопке другим способом): выберите в окне инспектора объектов на закладке **Events** (событие) из списка действий, которые сможет выполнить эта кнопка, событие **OnClick** и дважды щелкните на пустом пространстве в правой колонке. В тело процедуры

TForm1.btBeepClick(Sender: TObject);

вставьте оператор *MessageBeep(0)*, который извлекает из динамика короткий звуковой сигнал (бип).

28. Создайте динамическое изменение надписи на кнопке (надпись изменяется на этапе выполнения программы, а не на этапе ее конструирования), добавив обработчик события **OnCreate**, которое возникает после создания формы, но до появления ее на экране. Для этого выберите в окне инспектора объектов компонент **Form1**. На закладке **Events** этого компонента дважды щелкните справа от события **OnCreate** (создание формы) и добавьте в тело процедуры

TForm1.FormCreate(Sender: TObject);

оператор присваивания *BtBeep.Caption:='Звук'*.

29. Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Как создать новый проект в Delphi?
2. Как запустить программу на выполнение?
3. Как изменить свойства компонента Delphi?

Лабораторная работа 2.2. Использование стандартных компонентов Delphi

Приложения, использующие визуальные компоненты:

MemoPad (стр. **Standard**) **—** текстовый редактор, основанный на использовании компонента Memo. Демонстрирует создание команд главного и всплывающего меню, а также считывание и запись текстовых файлов.

BitView (стр. **Standard, Additional, Dialogs**) **—** программа просмотра файлов с растровым изображением. Позволяет открыть и отобразить на экране любой файл Windows, содержащий графику в стандартных форматах. Демонстрирует программирование с использованием графических и диалоговых компонентов.

Dclock (стр. **Standard, System**) **—** цифровые часы. Демонстрирует программирование компонента **Timer**, работающего параллельно с другими задачами.

Создание приложения ***MemoPad***

1. Создайте папку **Memopad**.
2. Начните новый проект.
3. Измените свойство **Caption** на *MemoPad*.

4. Свойство **Name** измените на *MainForm*.

5. Сохраните проект: в качестве имени файла при этом задайте *main.pas*, имя проекта **—** *memopad.dpr*.

6. На форме разместите компоненты **MainMenu** и **PopupMenu** (объекты **MainMenu1** и **PopupMenu1**). Во время работы программы данные пиктограммы невидимы.

7. Дважды щелкните на объекте **MainMenu1**. Откроется окно **MainForm.MainMenu1**. Для создания меню введите *&File* и нажмите **Enter**, чтобы изменить свойство **Caption** этого объекта меню. **&** **—** перед соответствующей буквой обеспечивает доступ к этому элементу совместно с клавишей **Alt**.

8. Ниже пункта **File** введите **&Save** для создания пункта **Save**. Для клавишного эквивалента в свойство **ShortCut** введите или выберите *F2*.

9. Аналогично введите *E&xit*.

10. Точно также создайте меню **Help**, а в нем пункт **About F1**.

11. Закройте окно **MainForm.MainMenu1**, чтобы посмотреть вновь созданное меню в окне формы. Можно проверить работу меню с помощью клавиатуры, но не щелкая на пунктах меню, так как может появиться окно редактирования модуля с заготовкой для обработчика события. Но если это произошло, вернитесь в окно формы.

12. Дважды щелкните на объекте **PopupMenu1**. Введите в это окно *About, Save, Exit*.

13. Свойству **PopupMenu** формы присвойте *PopupMenu1*.

14. Клавишей **F9** скомпилируйте и запустите приложение, которое пока не завершено. Команды меню программы не работают. Щелчок правой кнопкой мыши в области формы вызывает всплывающее меню, которое в данный момент не работает.

Первый этап создания приложения **MemoPad** завершен.

Далее выполним следующее:

15. На форме разместите метку (компонент **Label**) и ее свойству **Name** присвойте значение *MemoLabel*. Свойству **Caption** метки Delphi присвоила то же значение.

16. Нежелательно, чтобы объект показывал свое внутреннее имя. Для изменения отображаемого заголовка метки ее свойству **Caption** присвойте значение Memos.

17. Разместите на форме примечание (компонент **Memo**). **Name —** *Memo1*. Все символы значения свойства **Lines** следует удалить так, чтобы область ввода была пустой (щелчок по кнопке с многоточием и очистить открывшееся окно редактирования, **ОК**).

18. Измените размер **Memo1**. Откомпилируйте программу.

19. Для того чтобы создать код, который будет выполняться при выборе пунктов меню, необходимо в окне формы выбрать нужный пункт, например **File — Save**, и щелкнуть на нем. В текст программы вставить заготовку процедуры, называемой обработчиком события, а курсор размещается там, где следует ввести оператор, который будет определять действия данного пункта меню. Между **begin** и **end** введите:

Memo1.Lines.SaveToFile(‘memos.txt’);

20. Введенный оператор сохраняет в файле *memos.txt* строки, содержащиеся в объекте **Memo1**. Эти строки представлены объектом **Lines**, принадлежащим **Memo1**, а **SaveToFile – Lines**. Оператор передает строку в скобках методу **SaveToFile**.

21. Щелкнув на пункте **File — Exit**, между **begin end**, введите Close;. Эта процедура предназначена для закрытия окна. В данном случае она завершает работу программы.

22. Щелкнув на пункте **Help — About**, между **begin** и **end** введите в одну строку:

MessageDlg(‘Memo Pad’#13#10’© 2003 by Tom Swan’#13#10’ V. 1.0’, mtInformation,[mbOK],0);

Символ авторских прав © нужно вставить из таблицы символов Windows. Данный оператор создает диалоговое окно с информацией об авторских правах и номере версии приложения **MemoPad**.

23. Дважды щелкните на объекте **PopupMenu1**. Выберите закладку **Events**. Для пункта меню **About** событию **OnClick** присвойте значение *About1Click*. Имя обработчика события можно ввести вручную или выбрать из списка. Аналогичную работу проделать и с другими пунктами меню.

24. Закройте окно. Щелкните на форме. Дважды щелкните на пустой строке ввода справа от события **OnActivate** и введите следующий оператор:

```
If  FileExists('memos1.txt')  then
   Memo1.Lines.LoadFromFile('memos1.txt')
else
   Memo1.Lines.SaveToFile('memos1.txt');
```

25. Приведенный оператор вызывает функцию **FileExists**, которая возвращает **true**, если указанный файл имеется на диске, или **false**, если его там нет. В данном случае создается новый пустой файл.

26. Выберите форму и дважды щелкните на пустой строке справа от события **OnClose**. В текст программы вставится заготовка процедуры, выполняемой при закрытии окна. Введите:

Memo1.Lines.SaveToFile('memos1.txt');

чтобы сохранить объект **Lines** в указанном файле. Введенный в окно текст при запущенной программе сохраняется в файле.

27. Откомпилируйте программу.

Компоненты страницы **Additional** и **Dialogs**

С помощью этих двух категорий создадим приложение, позволяющее просматривать растровые изображения.

На странице **Additional** находятся графические компоненты, используемые для создания растровых изображений и геометрических фигур — прямоугольники, окружности, линии.

Страница **Dialogs** содержит компоненты, которые взаимодействуют с диалоговыми окнами Windows общего назначения (общими диалогами — перемещение по каталогам диска, выбор файлов, назначение шрифтов и цветов, поиск или замена значений в документах).

Использование компонента **Image**

Сначала надо очистить рабочую область (**File — New — Application**), а затем на пустой форме разместить объект **Image** (стр. **Additional**). Поскольку размеры образа определяются во время работы программы с его содержимым, объект изображается в виде пунктирного контура, которого нет в завершенной программе. Выделите свойство **Picture**, щелкните на кнопке подстановки, чтобы открыть **Picture Editor**, имеющийся в Delphi, который используется для импортирования в приложение любого растрового изображения. Кнопка **Load** позволяет выбрать имя файла. **Save —** скопировать выбранный файл в другой файл на диске. **Clear —** очистить ранее загруженное изображение. Кнопка **Cancel —** закрывает окно без сохранения, **ОК —** загружает растровое изображение в объект.

Приложение **BitView**

1. Создайте папку **Bitview** для хранения файлов проекта. Выполните команду **File — New — Application**.

2. Свойствам **Caption** и **Name** присвойте соответственно значения *Bitmap Viewer* и *MainForm*.

3. Файлам модуля и проекта назначьте соответственно имена *main* и *bitview*.

4. Значение *wsNormal* свойства **WindowState** формы замените на *wsMaximized* (окно будет распахнуто на весь экран).

5. Выберите страницу **Additional** и разместите на форме компонент **Image**. Свойству **Name** объекта присвойте *BitImage*. Месторасположение объекта и его размеры не имеют значения.

6. Свойству **Align** объекта **BitImage** присвойте значение *alClient*. В результате компонент автоматически получит размеры клиентной области формы, которая представляет собой пространство в пределах границ окна.

7. Свойству **Stretch** присвойте значение *true*, чтобы программа выводила изображение в пределах всего пространства компонента.

8. Разместите на форме главное меню **MainMenu1** (компонент **MainMenu**), находящееся на странице **Standard**. Дважды щелкните на пиктограмме меню и создайте подменю **File**, содержащее два пункта **— Open** и **Exit**.

9. Разместите на форме общий диалог открытия файла **OpenDialog1** (компонент **OpenDialog** страницы **Dialogs**). Объекты **MainMenu1** и **OpenDialog1** можно разместить в любом месте.

10. Выберите объект **OpenDialog** и щелкните на кнопке подстановки, находяйщейся рядом со значением свойства **Filter**. В результате откроется окно **Filter Editor** для создания фильтров имен файлов. Например, для представления файлов с растровыми изображениями необходимо указать в колонке **Filter Name — Bitmap files** (это то, что будет написано в строке **Тип файла** окна **Открыть**), в колонке **Filter —** **.bmp*.

11. Теперь можно откомпилировать программу.

12. В окне формы **—** но не в работающей программе **—** щелкните на пункте **File — Open**, чтобы создать соответствующий обработчик события, между словами **begin** и **end** заготовки обработчика введите следующий код:

```
if OpenDialog1.Execute then
begin
   BitImage.Picture.LoadFromFile(OpenDialog1.Filename);
   Caption := OpenDialog1.Filename;
end;
```

13. Предшествующий оператор **if** активизирует объект **OpenDialog1**, вызывая его метод **Execute**. Если пользователь закрывает диалоговое окно, щелкнув на кнопке **ОК**, функция возвращает значение **true**. В этом случае программа выполняет два следующих оператора внутри следующих **begin** и **end**. Первый оператор вызыва-

ет метод **LoadFromFile** объекта **Picture** для загрузки файла, указанного в свойстве **Filename** объекта **OpenDialog1**. Второй назначает это же имя в качестве заголовка формы, тем самым заменяя заголовок окна полным именем файла с растровым изображением.

14. Вернитесь в окно формы и в созданной заготовке обработчика события вставьте оператор *Close* между **begin** и **end**. Выход из приложения — команда меню **File — Exit**.

15. Сохраните завершенный проект, откомпилируйте и запустите.

Компоненты страницы **System**

Компоненты этой категории позволяют получить доступ к программному и аппаратному обеспечению компьютера. Среди них: таймер (компонент **Timer**), предназначенный для организации фоновых процессов, художественный набор (компонент **PaintBox**) — для работы с графикой, а также ряд других компонентов для создания пользовательских средств работы с дисками, каталогами и файлами. Кроме того, имеется медиаплейер (компонент **MediaPlayer**), используемый в приложениях мультимедиа, и компоненты поддержки OLE и DDE — для совместного использования данных.

Использование компонента **Timer**

Этот компонент является одним из простейших. Он имеет лишь несколько свойств и единственное событие. Можно создать один или более таймеров, которые будут работать параллельно с другими приложениями или в фоновом режиме.

Для создания таймера выберите компонент **Timer** на странице **System**. Разместите на пустой форме объект **Timer**, можно изменить его имя. Для свойства **Interval** установите нужное значение частоты события от 1 до 64 435. При значении 1000 создается односекундный таймер, при значении 100 событие генерируется с частотой примерно $^1/_{10}$ секунды. У компонента **Timer** имеется только одно событие — **OnTimer**. Дважды щелкните на значении события и вставьте операторы в полученной заготовке обработчика между **begin** и **end**, которые будут выполняться с частотой, определенной для компонента Timer.

Приложение **Dclock**

1. Создайте папку **dclock**. Файлам модуля и проекта присвойте имена соответственно *main* и *dclock*. Свойствам формы **Caption** и **Name** присвойте значения соответственно *Clock* и *MainForm*.

Для удаления кнопок минимизации и максимизации окна выберите форму и дважды щелкните на свойстве **BorderIcons** (указатель мыши разместите на имени свойства слева, а не на его значении справа). Обратите внимание на знак «+» слева от имени свойства,

который после двойного щелчка меняется на «-». «+» и указывает, что данное свойство имеет одно или больше скрытых подсвойств, а «-» означает, что свойства отображены на экране. В данном случае имеется три таких значения. Установите **biSystemMenu** в *true* (значение по умолчанию), а **biMinimize** и **biMaximize** присвойте значение *False*. Чтобы увидеть эти изменения, нужно запустить программу.

Чтобы изменить размер формы, надо в свойстве **BorderStyle** выделить значение *bsSingle*. Результат виден при запуске.

Разместите на форме метку (компонент **Label**, страница **Standard**). Свойствам **Caption** и **Name** объекта **Label** присвойте значение соответственно *00:00:00 AM* и *TimeLabel1*. В результате вы получите метку необходимого размера, чтобы во время работы программы нашлось место для всех цифр, обозначающих время.

Выберите свойство **Font** объекта **TimeLabel**. Для того чтобы открыть диалоговое окно выбора шрифта, щелкните на кнопке подстановки. Выберите для своих часов шрифт, размер, стиль.

Поместите на форму таймер (**Timer**).

Выберите объект **Timer1** и щелкните на закладке страницы **Events** в окне **Object Inspector**. В результате вы увидите только событие **OnTimer**. Для создания процедуры в исходном коде программы дважды щелкните на значении справа от события. Следующую строку введите после ключевого слова **begin**:

TimeLabel1.Caption:= TimeToStr(Time);

Предыдущий код конвертирует текущее время в строку и присваивает свойству **Caption** метки результат в качестве значения. В результате время отображается на экране.

Задайте окну формы необходимые размеры. (Если окно скрыто, найдите его с помощью команды **View — Forms**). Пиктограмма **Timer1** необходима во время работы программы, и ее можно переместить в любое место.

Нажмите **F9** для запуска программы на выполнение.

Контрольные вопросы

1. Как добавить контекстное меню в проект Delphi?
2. Как добавить окно диалога в проект Delphi?
3. Для чего нужен компонент Timer?

Лабораторная работа 2.3. Репозиторий Delphi

Создание формы и сохранение ее в архиве

1. Откройте новый проект.
2. Для объекта **Форма** установите следующие свойства:
   - **Caption —** *Учебная программа;*
   - **Name —** *fmExample;*
   - **Height —** *320;*
   - **Width —** *440;*
   - **Position —** *poScreenCenter.*
3. Добавьте две панели: сначала нижнюю, затем верхнюю (компонент **Panel** на стр. **Standard**) без заголовков (очистите свойство **Caption**). Расположите их внизу окна формы (свойство **Align** в значение *alBottom*).
4. Для верхней панели установите свойство **BevelOuter** (нижняя кромка) в значение *byNone* (без нижней кромки), чтобы панель и метка, которую мы расположим на ней, воспринимались как одно целое.
5. На верхней панели разместите строку ввода (компонент **Edit** на стр. **Standard**), очистите свойство **Text** и присвойте значения следующим свойствам:
   - **Name —** *edInput;*
   - **Width —** *250;*
   - **Left —** *87;*
   - **Top —** *12.*
6. На свободное место формы над верхней панелью разместите метку (компонент **Label**) для ввода текста, расположите ее внизу формы (**Align —** *alBottom*), присвойте ей внутреннее имя *lbOutput* (свойство **Name**) и очистите заголовок (свойство **Caption**).
7. Добавьте в окно формы пустой многострочный редактор (компонент **Memo** на стр. **Standard**). Задайте ему внутреннее имя **mmOutput**. Расположите его во всей оставшейся свободной области окна программы (свойство **Align** в значение *AlClient*), очистите свойство **Lines** (щелчок по кнопке с многоточием и очистить открывшееся окно редактирования, **ОК**), поместите горизонтальные и вертикальные линейки прокрутки (свойство **ScrollBars** в значение *ssBoth* **—** обе). Отмените перенос слов (свойство **WordWrap** в значение *False* **—** нет).
8. На нижнюю панель поместите две командные кнопки с надписью и пиктограммой (компонент **BitButton**, стр. **Additional**): кнопку **OK** (свойство **Kind** в значение *bkOk*) и кнопку **Close** (свойство **Kind** в значение *bkClose*); укажите их расположение (для первой **Left —**

24, **Top —** 8, для второй **— Left —** 120, **Top —** 8); присвойте им внутренние имена: *bbRun* и *bbClose*.

9. Сохраните созданную форму в архиве (репозитории, который служит для накопления типовых форм и проектов) **Delphi** (в папке **C:\Program files\Borland\Objrepos**) под именем **example+<номер группы>.pas** с помощью команд **File — Save As.**

10. Зарегистрируйте форму в архиве с помощью команды **Add to Repository** контекстного меню (щелкните по форме правой кнопкой мыши). В строке **Title** диалогового окна регистрации введите имя формы *fmExample*, в строке **Description** (пояснение): «*Прототип главной формы для учебной программы*», в списке **Page** выберите **Forms**, в строке **Author —** свою фамилию. Сохраните. Теперь при создании нового проекта командой **File — New — Application** эту форму можно выбирать вместо стандартной (пустой).

11. Для того, чтобы определить форму **fmExample** как главную и умалчиваемую при создании нового проекта при выполнении команды **File - New - Application**, в диалоговом окне команды **Tools - Repository** в списке **Pages** выберите *Forms*, найдите свою форму и установите переключатели **New Form** и **Main Form**.

Ввод и вывод текста

12. Создайте папку **prog2.**

13. Загрузите **Delphi.**

14. Выберите **File — New — Application**.

15. Если подготовленная форма «Учебная программа» не загружена, загрузите ее командой **File — Open.**

16. Чтобы при нажатии кнопки **OK** (компонент **bbRun**), введенный в строку **edInput** текст был помещен в строку **lbOutput** и редактор **mmOutput**, надо изменить обработчик событий **onClick** этой кнопки. Для этого выполните двойной щелчок по кнопке **OK.** В окне редактирования кода измените обработчик событий **—** процедуру *TfmExample.bbRunClick*, введя следующие строки между *begin* и *end*:

```
// поместите введенный текст в заголовок метки
   lbOutput.Caption:=edInput.Text;
   //и в строку многострочного редактора
   mmOutput.Lines.Add(edInput.Text);
edInput.Text:='';  // очищаем строку ввода
   // передаем ей фокус ввода (связываем с клавиатурой)
   edInput.SetFocus;
```

17. Запустите программу.

18. Обратите внимание, что в начале строка *edInput* не имеет фокуса ввода.

19. Измените код программы, добавив обработчик событий **OnActivate** для формы **fmExample**, и установите в нем фокус ввода для строки **edInput** (раскройте список выбора в верхней части окна **Инспектора объектов**, выберите в нем компонент **fmExample** и дважды щелкните по правой колонке свойства **OnActivate** на вкладке **Events** этого окна):

```
procedure TfmExample.FormActivate(Sender:TObject);
begin
    edInput.SetFocus
end;
```

20. Еще раз запустите программу и проверьте ее работу.

Контрольные вопросы

1. Для чего и как используется репозиторий?
2. Как создать шаблон формы?
3. Как использовать созданный шаблон формы?

Лабораторная работа 2.4. Разработка интерактивного Windows-приложения

Задание. Вычислить произведение двух введенных целых чисел.

1. Создайте папку **prog3**.
2. Загрузите **Delphi.**
3. Выберите **File — New — Application**.
4. Добавьте для вычисления результата на нижнюю панель еще одну кнопку типа **BitBtn** так, чтобы она полностью закрывала кнопку **bbRun**, и назовите ее *bbResult* (свойство **Name**). Установите ее вид в **bkOk** (свойство **Kind**). Поместите в свойство **Visible** значение *false*, чтобы сделать ее невидимой.
5. Для того чтобы после ввода первого операнда (множимого) и после нажатия кнопки **Ok** (компонент **bbRun**) она была заменена на кнопку **bbResult** (ввод множителя и вычисление результата), добавьте в обработчик событий (двойной щелчок в правой части события **OnClick**, чтобы войти в программный код) кнопки **bbRun** свойство **Visible** со значением *False*, а кнопки **bbResult —** со значением *True*:

```
bbRun.Visible := false; // в TfmExample.bbRunClick
bbResult.Visible := true; //  в TfmExample.BbresultClick
```

6. Запустите проект.
7. Для контроля за правильностью ввода чисел удалите компонент **Edit** (редактор **edInput**) на форме (щелкните по нему мышью и

нажмите кл. **Delete**) и замените его на компонент **MaskEdit —** ввод по маске (стр. **Additional**).

8. Присвойте ему прежнее имя **edInput** (свойство **Name**) и раскройте диалоговое окно свойства **EditMask** (сделать щелчок по «педальке»).

9. В диалоговом окне в строке **Input Mask Editor** введите маску *#999999;1*, которая воспринимает только знаки «+» или «–» в начале числа (элемент маски #) и цифры (элемент 9). 1 указывает на то, что в текст будет помещена строка, соответствующая шаблону (0 **—** исходный). В окно вместо символа заполнителя «_» введите пробел.

10. Измените код обработчика событий **OnClick** кнопки **bbResult**:

```
begin
{bbResult.Visible := true; }
// функция Trim возвращает строку без ограничивающих пробелов
// для ввода вещественных чисел используйте функцию StrToFloat
y:=StrToInt(Trim(edInput.Text));
mmOutput.Lines.Add ('2-й операнд: '+edInput.Text);
// вычисляем и показываем результат:
mmOutput.Lines.Add('Результат:  '
+ IntToStr(x) + '*' + IntToStr(y) + '=' + IntToStr(x*y));
edInput.Text := '';
edInput.SetFocus;
lbOutput.Caption := 'Введите 1-й операнд';
bbResult.Hide;  // прячем кнопку bbResult
bbRun.Show;  // показываем кнопку bbRun
end;
```

11. Измените код обработчика событий кнопки **bbRun**, добавив в него:

- преобразование веденной строки в целое число *x*;
- сообщение о вводе первого операнда в многострочный редактор;
- очистку строки ввода;
- установку фокуса на строку ввода;
- приглашение к вводу второго операнда;
- сокрытие кнопки **bbRun**;
- отображение кнопки **bbResult**.

12. Измените обработчик события **OnActivate** для формы **fmExample**, поместив в **lbOutput** следующий текст: *«Введите 1-й операнд»* и передав фокус ввода компоненту **edInput** в момент старта программы:

edInput.SetFocus;
lbOutput.Caption := 'Введите 1-й операнд';

13. Добавьте в описание класса **TfmExample** (в окне кода программы подняться вверх по модулю) в секцию **private** (личные для методов данного класса и программ модуля) новые поля данных *x, y:*

x, y : integer;

14. Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Как преобразовать целое число в вещественное?
2. Как проверить правильность ввода данных пользователем?
3. Что такое фокус ввода?

Лабораторная работа 2.5. Разработка расчетной программы

Задание. Составить программу «*Угадай число*», которая выбирает случайным образом целое число в диапазоне 0… 100 и запоминает его. После ввода пользователем числа программа сообщает, больше, меньше или равно оно заданному. Ввод продолжается до угадывания.

1. Создайте новое приложение (**File — New — Application)**.
2. Измените форму **fmExample**: поместите вместо компонента edInput типа **Edit** компонент **MaskEdit** (стр. **Additional**), назовите его **edInput** (свойство **Name**), укажите маску 099;1 и замените заполнитель на пробел (свойство **EditMask**).
3. В секции **private** класса **TfmExample** объявите переменную целого типа, в которой будет находиться случайное число:

x : integer;

4. Измените обработчик событий **OnActivate** формы **fmExample** (активизация окна программы), поместив в него операторы, выполняющие следующие действия:

– инициализацию датчика случайных чисел (randomize;);

– формирование случайного числа с помощью функции *random(101);*

– передачу строке **edInput** фокуса ввода;

– вывод сообщения "*Введите число*" в компонент **lbOutput.**

Текст процедуры:

```
procedure TfmExample.FormActivate(Sender: TObject);
begin
   randomize;
   x := random(101); {загадываем случайное число}
```

```
    edInput.SetFocus;  {Передаем строке edInput фокус ввода}
    Caption := 'Угадай целое число в диапазоне 0...100';
    lbOutput.Caption := 'Введите число:';
  end;
```

5. Измените названия главной формы (например, «*Угадай число*»).

6. Измените обработчик события **OnClick** кнопки **bbRun,** добавив в него операторы для того, чтобы:

– преобразовать введенный текст в целое число;

– очистить ввод;

– установить фокус ввода;

– сравнить введенное число с заданным и вывести сообщение о результате сравнения;

– если число угадано, то добавить звуковой сигнал для привлечения внимания, вывод сообщения «*Вы угадали*» и завершение работы.

Текст процедуры:

```
procedure TfmExample.bbrunClick(Sender: TObject);
var
  y:integer;
begin
   if  edinput.text='' then
   exit; //  если нет входного текста, прекращаем работу
   y:=strtoint (trim(edinput.text)); //преобразуем ввод в число
   edinput.text := ''; //очищаем ввод
   edinput.setfocus; //устанавливаем фокус ввода
// у каждого компонента Delphi есть целочисленное свойство tag
//  параметр Tag=0 означает угадывание числа, иначе - ответ
// на вопрос, будет ли пользователь играть после угадывания
  iIf  tag=0  then
  if x<y then         // Угадывание числа
    mmoutput.lines.add('x < '+inttostr(y))
   else if x>y then
    mmoutput.lines.add('x > '+inttostr(y))
    else
    begin
       mmoutput.lines.add('x = '+inttostr(y));
       messagebeep(0);  //бип для привлечения внимания
     lboutput.caption := 'Вы угадали! Введите 1, если хотите повторить:';
       tag := 1;  // следующий ввод - ответ на вопрос
    end
    else
```

```
      if y = 1 then
  begin
        x := random(101); // новое число
        lboutput.caption := 'Введите число:';
        tag := 0;  // следующий ввод - угадывание
      end
 else
   close;
end;
```

7. Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Как получить случайные числа в программе на Delphi?
2. Чем отличаются компоненты Edit и MaskEdit?
3. Как очистить строку ввода?

Лабораторная работа 2.6. Разработка программы-калькулятора

Задание. Составить программу, имитирующую работу микрокалькулятора. Программа вводит два операнда, знак математической операции **(+, –, *, /)** и вычисляет результат.

1. Создайте новую папку.
2. Для отмены стандартной обработки исключительных ситуаций (ошибок) отключите переключатель **Stop on Delphi Exception** в команде **Tools — Debugger Options — Language Exception**.
3. Выберите команду **File — New — Application**.
4. Измените следующие свойства компонента **edInput**:

– **Name —** *edInput1*;
– **Width —** *121*;
– **Left —** *64*;
– очистите свойство **Text**.

5. Справа от компонента **edInput1** поместите компонент **ComboBox** (стр. **Standard**), назовите его **cbSign**, установите **Width=41** и очистите свойство **Text**.
6. Введите в диалоговое окно свойства **Items** компонента **ComboBox** четыре строки со знаками арифметических действий.
7. Справа от компонента **cbSign** поместите компонент **Edit**, назовите его **edInput2** и очистите свойство **Text**.
8. Включите в обработчик события **bbRunClick** следующие действия:

– завершение работы, если не введены операнды и знак операции:

```
if (edInput1.Text='') or (edInput2.Text='')
or (cbSign.ItemIndex<0) then Exit;
```

– используя средства обработки исключительных ситуаций, проверьте правильность ввода первого операнда:

```
try //вычислить
  x:=StrtoFloat(Trim(edInput1.Text));
except //если возникла ошибка
       //  вывод сообщения
  ShowMessage('ошибка в записи числа '+edInput1.Text);
  edInput1.SetFocus;
  exit;        //завершение работы bbRun.Click
end;
```

– проверьте правильность ввода второго операнда, используя обработчик исключительных ситуаций;

– вычислите результат, используя оператор *case*, и обработайте исключительную ситуацию, которая может возникнуть при делении:

```
case cbSign.ItemIndex of
     0: z := x + y;
     1: z := x – y;
     2: z := x * y;
     3: try z := x/y;
except
     z:=1.1e + 38; //бесконечность при делении на 0
end;
end; //case
```

– выведите результат в **lbOutput** и в **mmOutput** так:

Если z=1.1e+38, то

<1-й операнд> <знак> <2-й операнд>= «бесконечность»,

в противном случае — просто *z*.

Для вывода знака операции выберите элемент (строку со знаком операции) объекта **cbSign** следующим образом:

cbSign.Items[cbSign.ItemIndex].

Для вывода вещественного числа используйте процедуру форматного преобразования *FloatToStrF(z,ffFixed,15,3)*. Здесь *ffFixed* — формат числа с фиксированной точкой, 15 — общее количество цифр в числе (*max double*), 3 — число знаков после запятой;

– очистите ввод:

```
edInput1.Text := '';
edInput2.Text := '';
cbSign.ItemIndex := –1;
```

Замечание: не забудьте объявить в обработчике событий локальные переменные *x, y, z*.

9. Запустите программу. Фокус ввода устанавливайте клавишей **Tab** или мышью. При вводе дробных чисел используйте запятую в качестве разделителя целой и дробной части.

Текст процедуры:

```
procedure Tfmexample.BBrunClick(Sender: TObject);
var
x,y,z : real;
begin
{завершение работы, если не введены операнды и знак операции}
if (edInput1.Text='') or (edInput2.Text='') or
(cbSign.ItemIndex<0) then Exit;
{используя средства обработки исключительных ситуаций, проверьте правильность ввода 1-го операнда:}
{ try  — это указание на начало блока обработки исключительных ситуаций}
try //вычислить
      x:=StrToFloat(Trim(edInput1.Text));
{except — Это действия по обработке исключительных ситуаций}
except //если возникла ошибка
      ShowMessage('ошибка в записи числа'+edInput1.Text);//вывод
      сообщения
          edInput1.SetFocus;
          exit; //завершение работы bbRun.Click
end;
{ проверьте правильность ввода 2-го операнда, используя обработчик исключительных ситуаций;}
{вычислите результат, используя оператор case, и обработайте исключительную ситуацию,
 которая может возникнуть при делении}
try
      y:=StrToFloat(Trim(edInput2.Text));
 except
      ShowMessage('ошибка в записи числа'+edInput2.Text);//вывод
      сообщения
          edInput2.SetFocus;
          exit; //завершение работы bbRun.Click
end;
       // все правильно: вычисляем результат
      case cbSign.ItemIndex of
0: z := x + y;
1: z := x – y;
2: z := x * y;
3: try z := x / y;
```

```
   except
z:=1.1e+38; //бесконечность при делении на 0
   end;
end; //case
      // показываем результат
lbOutput.Caption:= Trim(edInput1.Text)+' '+
cbSign.Items[cbSign.ItemIndex]+ ' '+ Trim(edInput2.Text)+ '= ';
if z>= 1.1e+38 then
      lbOutput.Caption:=lbOutput.Caption+ 'бесконечность'
    else
      lbOutput.Caption:=lbOutput.Caption+FloatToStr(z);
mmOutput.Lines.Add(lbOutput.Caption);
// очищаем ввод
edInput1.Text := '';
edInput2.Tex t:= '';
cbSign.ItemIndex := –1;
end;
end.
```

10. Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Что такое исключительная ситуация?
2. Как преобразовать вещественное число в целое?
3. Как используется оператор выбора case?

Лабораторная работа 2.7. Разработка расчетной программы с использованием циклов

Задание. Вычислить две суммы:

1) Сумму $\sum_{i=1}^{n} i$, где *n* вводится с клавиатуры;

2) Сумму $\frac{\sin nx}{\sqrt{n}}$, где n = 1,3,5,…, с введенной точностью eps.

1. Для выполнения задания измените форму следующим образом:

– на верхнюю панель поместите справа от компонента **edInput** кнопку **BitBtn** (стр. **Additional**), установите для нее свойство **Kind** в **bkOk** и измените ее имя на **bbRun1**;

– добавьте еще одну кнопку типа **BitBtn** так, чтобы она закрывала кнопку **bbRun1** и назовите ее **bbRun2**;

– удалите нижнюю панель, а верхнюю скройте с помощью свойства **Visible**.

2. Добавьте меню, расположив его в верхней части окна.

Создание меню

1. На форме разместите компонент **mainMenu** (стр. **Standard**), который определяет главное меню формы.

2. Дважды щелкните по компоненту **mainMenu1** левой кнопкой мыши (либо справа от строки **Items Инспектора объектов**). Появится окно конструктора меню.

3. Перейдите в окно **Инспектора объектов** и введите в строку **Caption** название первого пункта меню **&Решение** (**&** позволяет выбрать этот пункт кл. **Alt — <p>** (где **p —** выделенная буква). Нажмите **Enter**.

4. Введите первую подопцию пункта меню **Решение**, назвав ее «**Задача1**», а затем вторую с заголовком «**Задача2**» (нажимайте **Enter** после каждого ввода).

5. Щелкните мышью справа от пункта меню **Решение** и введите название второго пункта меню: **&Выход**.

6. Измените свойство **Name** всех опций меню, назвав их соответственно **mnRun** (Решение), **mnExit** (Выход), **mnTask1** (Задача1), **MnTask2** (Задача2).

7. Закройте окно конструктора меню.

Обработка событий

1. Для создания кода, который будет выполняться при выборе первой подопции **Задача1**, сделайте на ней двойной щелчок.

2. В окне редактирования кода программы в тело процедуры (метода) **mnTask1Click** класса **TfmExample** введите операторы, выполняющие следующие действия:

– очистите многострочный редактор методом

mmOutput.Clear;

– очистите строку ввода **edInput**:

edInput.Clear;

– очистите строку вывода:

LbOutput.Caption:='';

– покажите панель **Panel2** (процедура **Show**) и метку **lbOutput** (свойство **Visible**);

– поместите в заголовок компонента **lbOutput** следующий текст: «*Введите n*»;

– скройте кнопку **bbRun2** и покажите **bbRun1**;

– поместите фокус ввода в компонент **edInput**.

3. Создайте обработчик события **bbRun1Click**, добавив в него следующие действия:

– завершите программу, если в строку ввода (компонент **edInput**) ничего не введено;

– преобразуйте введенную строку в целое *n*, используя обработчик исключительный ситуаций *try — except*;

– вычислите сумму;

– выведите значение суммы в строку **lbOutput**, а в **mmOutput** поместите следующее сообщение:

– «*Сумма всех целых чисел от 1 до ___ равна ____* »;

– скройте панель **lbOutput**.

4. Создайте обработчик события для подменю **Задача2** (компонент **mnTask2**), скрыв кнопку **bbRun1** и показав **bbRun2**. В заголовок компонента **lbOutput** поместите «*Введите точность*».

5. Создайте обработчик события **bbRun2Click** для вычисления суммы ряда с заданной точностью. Задайте любое значение *x* с помощью оператора *Const*. Выведите значения суммы, используя функцию *FloatToStrF*.

Текст процедуры для обработки события:

```
procedure Tfmexample.bbrun2Click(Sender: TObject);
const x=0.5;
var i,k:integer;
   d,s1,eps:real;
begin
s1:=0;
if edinput.text='' then exit;
try //вычислить
  eps:=strtoFloat(trim(edinput.text)); //преобразуем ввод в число
{except - Это действия по обработке исключительных ситуаций}
except //если возникла ошибка
        ShowMessage('ошибка в записи числа '+edInput.Text);//вывод
сообщения
        edInput.SetFocus;
        exit; //завершение работы bbRun.Click
end;
n:=1; s1:=0; k:=0;
repeat
   d:= (sin(n*x))/sqrt(n);
   n:= n+2; k:=k+1;
   s1:= s1+d;
```

```
until d<=eps;
s1:= s1 - d;
dec(k);
lboutput.caption:=Floattostr(s1);
mmoutput.visible:=true;
    mmoutput.Lines.add(' Сумма равна '+ FloattostrF(s1,fffixed,7,5)+
    #13#10 + ' Число итераций ' +  inttostr(k));
lbOutput.Caption:='';
Panel2.Visible:=false;
end;
```

Замечание: для создания обработчиков событий в задаче 2 используйте копирование операторов соответствующих методов задачи 1. Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Как добавить меню в проект Delphi?
2. С помощью какого цикла вычисляется первая сумма?
3. С помощью какого цикла вычисляется вторая сумма?

Лабораторная работа 2.8. Разработка расчетной программы с использованием массивов

Задание. Определите для *n* случайных чисел их среднее арифметическое значение, а также максимальное и минимальное.

Создание формы

1. Удалите компоненты **edInput** и **lbOutput**.
2. Разместите на панели **Panel1** список множественного выбора (компонент класса **TlistBox**, стр. **Standard**) с именем **lbxInput**. Установите свойство **MultiSelect** в значение **true** (разрешает выбор нескольких элементов). В свойство **Items** компонента **ListBox** введите следующие строки:
   - среднее значение;
   - минимальное значение;
   - максимальное значение.

Обработка событий

1. В секции реализаций модуля **Implementation** задайте количество случайных чисел с помощью служебного слова *const* и объявите одномерный массив целых чисел.
2. В обработчике событий **FormActivate** (для нахождения обработчика событий **FormActivate** выберете в **Object Inspector** объ-

ект f**mExample**. В закладке **Events** сделайте двойной щелчок справа от **onActivate**):

- сформируйте одномерный массив с помощью датчика случайных чисел. Для определения случайного числа используйте процедуру *randomize* и функцию *random(101)*, которая генерирует случайное число, равномерно распределенное в диапазоне от 0 до 100. Используйте оператор цикла *for*;
- выведите значения сформированного массива в компонент **mmOutput**;
- установите фокус ввода в компонент **lbxInput**.

3. Измените обработчик события **bbRun**:

- разместите вычисления среднеарифметического, минимального и максимального значений в зависимости от сделанного выбора в компоненте **lbxInput** следующим образом:

```
if lbxInput.Selected[0] then
….//вычисляем среднеарифметическое значение
begin
x:=0;
 for i:=1 to n do
 x:=x+a[i];
 x:=x/n;
 mmOutput.Lines.Add('среднее значение ='+' '+FloatToStr(x));
end;
    if lbxInput.Selected[1] then
    .//вычисляем минимальное значение, используя функцию
    MinIntValue //модуля Math
    if lbxInput.Selected[2] then
    .//вычисляем максимальное значение, используя функцию Max-
    IntValue //модуля Math
```

Здесь свойство *Selected[i]* содержит признак выбора элемента списка с номером *i* (значение *true* или *false*);

- результаты решения поместите в компонент **mmOutput** в следующем виде:

Среднее = …..… Min = …..… Max = …..…

- передайте фокус ввода компоненту **lbxInput**.

Замечание. Модуль Math должен быть включен в список модулей в секции uses раздела Interface или Implementation проекта.

Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Для чего используется секция Implementation?
2. Каким образом производится выбор вычисляемого значения?
3. Как вычисляются минимальное и максимальное значения?

Лабораторная работа 2.9. Разработка расчетной программы с использованием пользовательских процедур и функций

Задание. Используя подпрограммы формирования квадратной матрицы, вычисления следа матрицы и вывода матрицы в многострочный редактор, создайте две матрицы и выведите ту, у которой след наименьший.

1. Удалите верхнюю панель **Panel2**.
2. Для отображения небольшого справочного окна (ярлычка рядом с объектом) поместите в свойство **Hint** компонента **bbRun** текст «*Вычисление следа*», а в компонент **bbClose —** «*Выход*». Присвойте свойству **ShowHint** этих компонентов значение *true*.
3. В секции реализаций модуля **Implementation** задайте с помощью служебного слова *const* размерность квадратной матрицы (количество строк/столбцов), опишите тип целочисленной квадратной матрицы и объявите две переменные этого типа.
4. Далее опишите подпрограмму вывода матрицы в компонент **Memo**. Не забудьте при обращении к компоненту **Memo** указать его принадлежность классу **TfmExample**.
5. В обработчике события **OnActivate** формы в разделе описаний (между заголовком процедуры и словом *begin*) опишите подпрограмму формирования квадратной матрицы, используя датчик случайных чисел.
6. Добавьте в обработчик событий **OnActivate** формы операторы формирования первой и второй матриц (обращение к подпрограмме формирования) и их вывод в компонент **Memo** (обращение к подпрограмме вывода матрицы).
7. В обработчике события **bbRunClick** опишите подпрограмму определение следа матрицы.
8. Добавьте в обработчик события **bbRunClick** следующие операторы:
   - вычисление следа первой матрицы с помощью соответствующей подпрограммы;
   - вычисление следа второй матрицы;
   - вывод результатов вычислений в компонент **lbOutput**;
   - очистку компонента **mmOutput**;
   - вывод матрицы с наименьшим следом (используя подпрограмму вывода матрицы).
9. Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Что такое подпрограмма?
2. Почему в работе используются продпрограммы?
3. Как вычисляется след матрицы?

Лабораторная работа 2.10. Представление и обработка табличных данных с помощью компонентов Delphi

Задание. Используя компонент TStringGrid, сформировать две квадратные матрицы, вычислить след матриц, транспонировать и вывести ту, у которой след наименьший.

Замечание. Используйте результаты лабораторной работы 2.9.

1. В секции реализаций модуля **Implementation** задайте с помощью служебного слова *const* размерность квадратной матрицы (количество строк/столбцов), опишите тип целочисленной квадратной матрицы и объявите две переменные этого типа.
2. Удалите ненужные визуальные компоненты и поместите на форму три компонента класса TStringGrid (стр. **Additional**) для ввода исходных матриц и вывода результирующей матрицы.
3. В обработчике события **OnActivate** формы задайте размерность компонентов для работы с матрицами (используйте поля **RowCount** и **ColCount**).
4. Введите с клавиатуры элементы двух исходных матриц (для редактирования ячеек компонентов класса TStringGrid задайте значение **Options — goEditing = true**).
5. Реализуйте в виде отдельной процедуры процедуру определения следа матрицы.
6. В обработчике события **bbRunClick** реализуйте вызов процедур определения следа матрицы для двух исходных матриц. Выведите на экран результаты расчетов.
7. Поместите на форму еще одну кнопку. В обработчике события **OnClick** реализуйте вывод на экран транспонированной матрицы с наименьшим следом.
8. Запустите программу на выполнение и проверьте правильность ее работы
9. Используя метод **OnGetEditMask** для проверки вводимых в ячейки данных, разрешите ввод только целых чисел.
10. Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Какие компоненты Delphi можно использовать для представления и обработки табличных данных?
2. Как определить количество строк и столбцов компонента TStringGrid?
3. Как проверить правильность ввода данных в ячейки компонента TStringGrid?

Лабораторная работа 2.11. Обработка текстовых файлов

Задание 1. Составить программу, которая считывает текст из файла, создает и сохраняет текст в файле.

1. Начните новый проект (не используйте шаблон «Учебная программа»).
2. Измените свойство **Caption** формы на текст «*Текстовый редактор*». Свойство **Name** компонента **Form** измените на **fmMain**.
3. На форме разместите компоненты **MainMenu —** главное иерархическое меню **—** с пунктами *Файл* (подопции *Создать*, *Открыть* и *Сохранить*) и *Выход*. Присвойте объектам соответствующие имена **mnFile**, **mnNew**, **mnOpen**, **mnSave**, **mnExit**.
4. Разметите на форме компонент **PopupMenu** (стр. **Standard**) **—** контекстное вспомогательное меню, которое появляется при нажатии правой кнопки мыши. Дважды щелкните на компоненте **PopupMenu1**. Введите в это окно *Создать, Открыть, Сохранить, Выход*.
5. Свойству **PopupMenu** формы присвойте имя компонента **— PopupMenu1**.
6. Разместите на форме компоненты **OpenDialog** и **SaveDialog** (стр. **Dialog**).
7. Разместите на всем свободном пространстве формы многострочный редактор (компонент **Memo**) с именем **Name — Memo1**. Очистите область ввода (свойство **Lines**). Добавьте вертикальные и горизонтальные линии скроллинга.
8. Для команды **Открыть** пункта меню **Файл** создайте обработчик события, содержащего следующие операторы:

```
if  not OpenDialog1.Execute then exit;
if FileExists(OpenDialog1.FileName) then
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName)
else
begin
ShowMessage('Ошибка при открытии файла' + OpenDialog1.FileName);
```

exit
end;

9. Функция *Execute* любого модального окна (в данном случае диалогового окна) возвращает *true* при успешном завершении диалога и *false* при нажатии клавиши **Отмена**. Функция *FileExists* возвращает *true*, если файл с выбранным в окне диалога именем *FileName* имеется на диске, или *false*, если его там нет (в данном случае создается новый пустой файл). Метод *LoadFronFile* загружает текст из файла и помещает его в компонент **Memo**.

10. Для команды **Сохранить** создайте обработчик события, содержащий следующие операторы:

if not SaveDialog1.Execute then exit;
Memo1.Lines.SaveToFile(SaveDialog1.FileName); {сохранение в файле текста из окна редактора}

11. Создайте обработчик события для пункта меню **Создать**, введя в него очистку компонента **Memo1**.

12. Создайте обработчик события пункта меню **Выход**.

13. Чтобы иметь возможность отмены решения о завершении программы при выборе пункта меню **Выход**, создайте для формы следующий обработчик события **OnCloseQuery**:

```
if MessageDlg('Завершить программу?',
mtConfirmation, [mbYes,mbNo],0) = mrYes
then CanClose:=true
else CanClose:=false;
```

14. Функция MessageDlg создает диалоговое окно с сообщением и двумя кнопками, позволяющими либо завершить программу (CanClose:=true) при нажатии кнопки Yes, либо продолжить работу (выбор кнопки No).

15. Путем копирования создайте такие же обработчики для всех команд контекстного меню.

Задание 2. Добавьте в программу возможность преобразования строчных букв текстового файла в прописные, т.е. заглавные.

1. Добавьте в пункт меню **Файл** и в контекстное меню еще одну подопцию *Преобразовать*.

2. В секцию **Implementation** модуля поместите подпрограмму замены строчных букв прописными.

```
Procedure Perevod(var s:string);

begin
   s:= ANSIUpperCase(s);
end;
```

3. В обработчик события **OnClick** команды **Преобразовать** в раздел описаний введите описание переменной *S* типа *String* и файловой переменной типа *TextFile.*

4. В тело обработчика события добавьте следующие операторы:

– связывание файловой переменной с выбранным файлом и открытие его на чтение (предупредите возникновение исключительной ситуации, если имя файла не определено);

– построчное считывание в цикле текста, перевод каждой строки с помощью подпрограммы и вывод результата в компонент **Memo1**;

– закрытие файла командой.

Текст процедуры:

```
procedure TfmMain.N5Click(Sender: TObject);
var s:string;
    T:TextFile;
begin
    if not OpenDialog1.Execute then Exit;
    if FileExists(OpenDialog1.FileName) then
      Memo1.Lines.LoadFromFile(OpenDialog1.FileName)
    else
    begin
     ShowMessage('Ошибка при открытии файла
    '+OpenDialog1.FileName);
      Exit;
    end;
    Memo1.Lines.Clear;
    AssignFile(T,OpenDialog1.FileName);
    Reset(T);
    while not(EOF(T)) do
    begin
        Readln(T,S);
        perevod(s);
        Memo1.Lines.Add(S);
    end;
       CloseFile(T);
end;
```

5. Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Как происходит выбор файла?
2. Каким образом текст считывается из файла?
3. Каким образом текст сохраняется в файле?

Лабораторная работа 2.12. Обработка типизированных файлов

Задание. Создать картотеку книг. В каждой карточке должна содержаться следующая информация:

- фамилия и инициалы автора;
- заголовок книги; место и год издания;
- количество страниц.

Вывести на экран:

- название и авторов книг, выпущенных после 1990 г.;
- всю информацию о книгах, выпущенных московскими издательствами;
- определить общее количество книг, выпущенных в Москве после 1990 г.

Создание формы

1. Внизу окна расположите строку статуса (компонент **StatusBar** на стр. **Win32**) со следующими свойствами:
   - **Name —** *sbHint;*
   - **BidiMode —** bdRightT*oLefReadingOnly* (текст на панели выровнен слева);
   - **SimplePanel —** *true* (строка статуса содержит одну панель);
   - **Align —** *alBottom*.
2. Вверху окна поместите меню с пунктами: «*Ввод*» (имя **mnInput**), «*Вывод*» (имя **mnOutput**) и «*Выход*» (имя **— mnExit**) и следующими подпунктами пункта «*Вывод*»: «После 1990 г.» (имя **— mnLast90**); «*Москва*» (имя **— mnMoscow**); «*Количество*» (имя **— mnCount**).
3. Для каждой опции меню создайте развернутую подсказку. которая будет отображаться в строке статуса, т.е. задайте свойство **Hint**, введя в него для соответствующей опции пояснительный текст: «*Создание каталога книг*», «*Просмотр каталога*», «*Завершение работы*». Для формы в свойство **Hint** поместите следующий текст: «*Для перехода в главное меню нажмите кл.* **Alt**».
4. На всей свободной области окна расположите компонент **Memo** с полосами скроллинга. В свойство **Hint** поместите текст «*Для перехода в главное меню нажмите кл*. **Alt**».
5. Расположите в центре окна компонент **GroupBox** (контейнер) с заголовком «*Ввод данных*», свойством **Visible** в значении *false* и свойством **Hint** «*Ввод данных в картотеку*».
6. В контейнере расположите друг под другом пять меток с названиями: «*Автор*», «*Название*», «*Издательство*», «*Год*», «*Стр*.».
7. Рядом с метками «*Автор*», «*Название*», «*Год*» и «*Стр*.» поместите строки ввода (компоненты **Edit** с именами **edAutor**, **edTitle**,

edYear, **edSize**). Очистите свойство **Text** компонентов. Введите пояснительный текст в свойство **Hint** для каждого компонента.

8. Рядом с меткой «*Издательство*» расположите компонет **ComboBox** со списком (свойство **Items**) выбора крупнейших издательских центров (городов) и именем **cbCity**. Очистите свойство **Text** компонента и задайте свойство **Hint**.

9. На компонент **edYear** поместите компонент **UpDown** (стр. **Win32**), предназначенный для регулирования числовой величины. Присвойте следующие свойства этому компоненту:

– **Associate —** *edYear* (определяет связанный компонент);

– **Increment —** *1* (определяет шаг изменения числовой величины);

– **Max —** *2003* (максимальное значение числовой величины);

– **Min —** *1900* (минимальное значение числовой величины);

– **Position —** *1990* (текущее значение числовой величины);

– **Wrap —** *false* (разрешает выход числовой величины за установленные границы *Max* и *Min*);

– **Thousands —** *false* (отмена разделителя тысяч в числе).

10. Внизу контейнера **GroupBox1** разместите две кнопки типа **BitButton** с текстом «*Добавить*» и «*Закрыть*», с именами соответственно **bbAdd** и **bbClose**, и с пояснительным текстом (свойство **Hint**).

Обработка событий

11. В исполняемой части модуля (секции **Implementation**) поместите описание типа записи **Tbook**, объявите переменную **Book** типа **Tbook** и типизированный файл **F**. Тип **Tbook** объявите выше следующим образом:

```
type TBook=record
       Autor:array[1..25]of char;
       Title:array[1..25]of char;
       Publ:array[1..25]of char;
       Year:integer;
       Size:integer;
      end;
```

12. В секцию инициализации (после слова *Initialization*) поместите оператор связывания файла и откройте файл для создания и записи.

13. В секцию **Finalization** поместите оператор закрытия файла.

Занесенный код выглядит так:

```
implementation
{$R *.dfm}
var Book:TBook;
  F:File of TBook;
```

```
Initialization
{$i-}
AssignFile(f,'Catalog.dat');
try Reset(f) except Rewrite(f);
end;
{$i+}
Finalization
CloseFile(F);
end.
```

14. Чтобы разместить справочное сообщение из свойства **Hint** текущего компонента в строке статуса, создайте новый метод формы с заголовком procedure *ShowLongHint (Sender:Tobject);*

15. Поместите этот заголовок в описание класса **Tform1** (добавьте новый метод).

16. В раздел реализаций модуля (после слова **Implementation**) поместите реализацию (описание метода *ShowLongHint*), добавив к имени процедуры имя класса, которому принадлежит метод:

```
procedure Tform1.ShowLongHint (Sender:Tobject);
begin
  SbHint.SimpleText := Application.Hint
end;
```

17. В обработчик события **OnCreate** формы добавьте следующий оператор, определяющий свойство **OnHint** объекта программы (объекта класса **TApplication**):

```
Application.OnHint:=ShowLongHint;
```

18. В обработчик события первого пункта меню поместите операторы, выполняющие следующие действия:

- отобразить контейнер ввода (компонент **GroupBox1**):

```
GroupBox1.Show;
```

- установить фокус ввода в первую строку ввода (в компонент **edAutor**).

19. В обработчике события кнопки **Добавить**:

- сохраните введенные данные в файле:

```
with Book do
begin
  for i:=1 to Length(edAutor.Text) do
    Autor[i] := edAutor.Text[i];
  for i:=1 to Length(edTitle.Text) do
    Title[i] := edTitle.Text[i];
  for i:=1 to Length (cbCity.text) do
     Publ[i] := cbCity.Text[i];
 try
  Year := StrToInt(edYear.Text);
```

```
  except Year := 1900; end;
  try
    Size:=StrToInt(edSize.Text);
  except Size := 0; end;
end;
write(F,Book);
```

– очистите строки ввода и поместите фокус ввода в первую строку:

```
edAutor.Text := '';
edTitle.Text := '';
cbCity.Text := '';
edYear.Text := IntToStr(1990);
edSize.Text := '';
edAutor.SetFocus;
```

20. В обработчике события кнопки **Закрыть** скройте контейнер ввода: *GroupBox1.Hide;*

21. В обработчик события каждой подопции пункта меню **Вывод** поместите очистку компонента **Memo**, ввод из файла и вывод в **Memo** информации, соответствующей выбранной подопции. Перед выводом первой записи установите указатель файла **F** на начальный компонент *Seek(F,0);*

Покажем это на примере подпункта меню **Вывод** «После 1990 г.»:

```
procedure TForm1.mnLast90Click(Sender: TObject);
begin
Memo1.Lines.Clear;
Seek(F,0);
while not eof(F) do
begin
   read(F,Book);
   if Book.Year>1990 then  Memo1.Lines.Add(Book.Title);
end;
end;
```

Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Что такое запись?
2. Чем работа с типизированным файлом отличается от работы с текстовым файлом?
3. Для чего используется строка состояния?

Лабораторная работа 2.13. Разработка графических приложений

Рисование простейших геометрических фигур с помощью объекта ***Shape.***

1. Разместите на форме компонент **Shape** (стр. **Additional**).
2. В инспекторе объектов определите значения свойства **Brush** (Кисть) — цвет **Color** и тип **Style** заливки и значения свойства **Pen** (Перо) — цвет **Color**, ширину **Width** и стиль **Style** линии контура.
4. С помощью свойства **Shape** выберите тип фигуры (фигура будет занимать все пространство компонента **Shape**).
5. Постройте несколько разных фигур на форме.
6. Свяжите прорисовку новой фигуры (например, **shCircle**) с каким-нибудь событием (с кнопкой, списком выбора и др.), разместив соответствующий объект на форме.
7. Создайте обработчик события этого компонента, поместив в него следующие операторы:

```
shCircle.Brush.Color := clred; //цвет заливки
shCircle.Shape := stCircle; //нарисовать круг
```

и др.

8. Запустите программу и нарисуйте фигуры.

Рисование с помощью объекта ***Canvas***

1. Разместите на форме, панели или любом другом контейнере окно с канвой для рисования (компонент **PaintBox** (стр. **System**). Установите его размеры **Width** — *300*, **Height** — *200* и разместите в центре.
2. В обработчике события **OnPaint** компонента **PaintBox** объявите переменные *x* и *y* типа *integer*, а в тело обработчика поместите следующие операторы:

```
with PaintBox1.Canvas do
begin
   Brush.Color:=clYellow; //цвет заливки
   Pen.Color:=clBlue; //цвет контура
//  чертим эллипс внутри компонента
  Ellipse(0,0,width div 2,height div 2);
  Font.Name:='Arial'; //определяем тип шрифта для надписи
  Font.Size:=Height div 10; //задаем размер шрифта
  Font.Style:=[fsBold,fsItalic]; //определяем стиль (начертание)
  Font.Color:=clWhite; //задаем цвет символов
// определяем положение надписи
  x:=(width div 2-TextWidth('Delphi')) div 2;
   y:=(height div 2-TextHeight ('Delphi')) div 2;
  TextOut(x,y,'Delphi'); //выводим надпись
```

end
end;

Для отображения других элементов см. методы класса **TCanvas**.

3. Сверните окно, а затем раскройте вновь. Обратите внимание на то, что изображение не исчезло. Это связано с тем, что в отличие от других событий событие **OnPaint** отображает объект, а не рисует.

Отображение растровых изображений

Задание. Создайте программу для отображения и сохранения под другим именем одного из трех типов растровых изображений. поддерживаемых **Delphi**: растровой картинки (**.bmp)**, пиктограммы (**.ico)** и метафайла (.**wmf**)).

1. На форме разместите меню с пунктами *Файл (mnFile)* (подопции *Открыть (mnOpen)* и *Сохранить* как *mnSaveAs)* и *Выход (mnExit)*, а также компоненты **OpenDialog** и **SaveDialog** (стр. **Dialog**).

2. Добавьте компонент **Image** (стр. **Additional**) для размещения на форме в контейнере **Picture** компонента **Image1** растрового изображения.

3. В обработчике события **OnClick** пункта меню **Открыть**:

– выберите имя **FileName** файла с помощью компонента **OpenDialog**;

– загрузите и разместите в контейнере **Picture** растровое изображение с помощью следующего свойства контейнера **Picture**:

Image1.Picture.LoadFromFile(OpenDialog.FileName);

– поместите имя файла в свойство **Caption** формы:

Caption:=Lowercase (OpenDialog1.(Filename);

4. Создайте обработчик события **OnClick** пункта меню **Сохранить как**. В нем:

– назначьте сохраненному файлу его первоначальное имя (для сохранения под первоначальном именем, но в другом месте на диске):

SaveDialog1.fileName:=Caption;

– сохраните растровое изображение в файле:

Image1.Picture.SaveToFile(FileName);

– поместите краткое имя файла в свойство **Caption** формы.

5. Создайте обработчик события пункта меню **Выход**.

Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Какие геометрические фигуры можно вывести на экран с помощью компонента Shape?

2. Какие геометрические фигуры можно вывести на экран с помощью компонента PaintBox?

3. Как вывести на экран графическое изображение из файла?

Лабораторная работа 2.14. Разработка программы с графической заставкой

Создание заставки программы

1. Начните новый проект. Свойствам формы **Name** присвоить *MainForm*, **Caption** — *Splashin Demo*. Файл модуля — *main*, файл проекта — *splashin*.

2. На **MainForm** разместите экземпляр компонента **Button**: **Name** — *ExitButton*; **Caption** — *Exit*. Создайте обработчик события **OnClick** кнопки **ExitButton** (можно дважды щелкнуть по кнопке) и вставьте *Close*;

3. Выберите **File** — **New** — **Other**. В окне **New items** выберите на вкладке **New** шаблон **Form**.

4. Вы добавили в приложение вторую форму. **Name** — *SplashForm*, **Caption** — удалите. **BorderStyle** — *bsNone*.

5. Сохраните проект. Файл модуля — **splash.**

6. Свойству **Enabled** формы **SplashForm** присвойте значение *false*, чтобы пользователь не управлял окном во время работы приложения с помощью клавиатуры и мыши.

7. Измените размеры окна **SplashForm**. Поскольку данное окно не имеет рамки, разместите на форме фаску (компонент **Bevel** на странице **Additional**). Это позволит обозначить границы окна. **Align** — *alClient*, **Shape** — *bsFrame*, **Style** — *bsRaised*.

8. Разместите на **SplashForm** экземпляры компонентов **Image** и **Label**. Не следует применять кнопки или другие интерактивные управляющие элементы, так как выводить и удалять заставку должно само приложение.

9. Теперь преобразуйте исходный код проекта так, чтобы заставка отображалась до появления главного окна. Выполните **Project** — **View Source**, между *begin и end* в **Splashin.dpr** вставить:

```
SplashForm:= TsplashForm.Create(Application);
{Метод создает объект формы заставки}
SplashForm.Show;
SplashForm.Update;
Application. CreateForm(TmainForm, mainForm);
SplashForm.Hide;
SplashForm.Free;
Application.Run;
```

10. Чтобы заставка оставалась на экране несколько секунд, выберите форму **Mainform**. Создайте обработчик события **OnCreate**. Вставить переменную CurrentTime типа Longint и операторы:

CurrentTime:= GetTickCount div 1000;

while ((GetTickCount div 1000) <(CurrentTime +4)) do;

11. Нажать **F9**.

Закрытие окна

1. Разместите на форме экземпляр компонента Button. Дважды щелкните на объекте **Button1** и между ключевыми словами **begin** и **end** созданного обработчика события введите *Close;*.

2. Запустите приложение и выйдите из него, щелкнув на объекте **Button**. Окно закроется.

3. В окне **Object Inspector** перейдите на вкладку **Events** и щелкните на поле справа от события **OnCloseQuery**. Введите между ключевыми словами **begin** и **end** созданного обработчика события следующий код:

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose:
Boolean);
begin
  If MessageDlg   ('End the program?',
  MtConfirmation, [mbYes, mbNo], 0) = mrYes
  Then Canclose := True
  Else CanClose :=False;
end;
```

4. Запустите приложение. Если теперь щелкнуть на кнопке, появится окно подтверждения. Для завершения работы выберите **Yes**, для аннулирования вызова *Close* — **No**.

Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Что такое заставка?
2. Как определяется время отображения заставки?
3. Как сделать, чтобы окно проекта отображалось поверх других окон?

Лабораторная работа 2.15. Разработка расчетной программы и построение графиков

Создание форм с помощью шаблонов

1. Создайте новый проект. Присвойте главной форме имя ***fmMain*** (свойство **Name**).

2. Вызовите команду **File** — **New** — **Other**. В появившемся окне **New Items** на вкладке **Forms** выберите шаблон формы **AboutBox**, отображающий в диалоговом окне данные об авторских правах.

3. Введите имя компонента *fmAbout* (свойство **Name**) и заголовок «*О программе*» (свойство **Caption**).

4. Сохраните проект в файле *ExmAbout.dpr*, модуль главной формы — в файле *Main.pas*, а модуль дополнительной формы — в файле *About.pas*.

5. Добавьте на форму **fmMain** меню с пунктом «*О программе*».

6. В обработчик пункта меню «О Программе» поместите оператор

fmAbout.ShowModal;

7. В результате выполнения этого пункта меню должно появится диалоговое модальное окно. Для продолжения работы программы модальное окно надо закрыть. Немодальные диалоговые окна не мешают нормальному функционированию меню и других программ.

8. Запустите программу.

Построение графика с помощью компонента Chart

Задание. Создайте программу построения двух функций sin(x) и cos(x) на интервале от 0 до 2π.

Измените в форме ***fmAbout*** свойства **Caption** объектов **Label**: введите имя вашей программы; информацию об авторских правах и любой другой текст.

Добавьте в меню формы ***fmMain*** пункты ***График*** (***Синус, Косинус***) и ***Выход***.

Разместите на всем свободном пространстве главной формы компонент **Chart** (стр. **Additional**).

Откройте диалоговое окно **Editing Chart1**, выполнив двойнойщелчок по компоненту.

В окне **Editing Chart1** выберите вкладку **Chart**, а затем вкладку **Series** и сделайте щелчок по кнопке **Add**.

В окне **TeeChart Gallery** выберите вкладку **Standard**, затем выберите тип диаграммы **Line**, сделайте щелчок по кнопке **OK**.

В окне **Editing Chart1** ознакомьтесь с параметрами настройки компонента **TChart** на вкладках General, Axis, Titles, Legend, Panel, Paging, Walls, 3D.

Задайте название графика «*Синус X*». Для этого в окне **Editing Chart1** выберите вкладку **Chart**, а затем вкладку **Legend**, в списке **Legend Style** выберите значение **Series Name**.

На вкладке **Titles** диалогового окна **Editing Chart1** введите заголовок диаграммы *«Тригонометрические функции»*.

В обработчике события **OnActivate** главной формы скройте объект ***Chart1***.

В обработчике события **OnClick** пункта меню ***График — Синус*** выполните следующие действия:

- покажите объект ***Chart1***;
- определите шаг изменения аргумента ($2\pi./n$, где n — количество точек, выводимых на график);
- в цикле **while** или **repeat** пока не будет достигнут конец интервала изменяйте значения аргумента **x**; здесь же для каждой функции поместите оператор обращения к методу **addxy**, который добавит к объекту **Series** объекта **Char1** очередную точку графика с координатами x,y:

chart1.series[i].addxy(x,y,'',clTeeColor);

В этом операторе:

i — номер графика (серии) 0,1,…;

x — значение аргумента;

y — значение функции или выражение, вычисляющее это значение;

'' — текст, выводимый вдоль нижней оси; в данном случае текст будет определяться установленными параметрами компонента **Chart**;

clTeeColor — цвет графика (в данном случае определяется автоматически).

Постройте график «*Косинус X*» самостоятельно.

Создайте обработчик события пункта меню **Выход**.

Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Как построить график по заданной формуле?
2. Какие способы построения графиков доступны в Delphi?
3. Какие компоненты используются для построения графиков?

Лабораторная работа 2.16. Разработка справочной системы

Справочная система для Windows-приложения в формате hlp-файла представляет собой разновидность гипертекста.

Разработка справочной системы для Windows-приложения состоит из двух основных этапов — планирования справочной системы и кодирования справочной информации, и включает следующие шаги:

– определение назначения справочной информации;

– определение содержания тематических разделов справочной системы;
– определение связей между тематическими разделами;
– определение перечня ключевых слов (терминов);
– определение внешнего вида тематических разделов справочной системы;
– подготовка (ввод) справочной информации по каждому разделу в виде обычного текста;
– подготовка графических изображений;
– кодирование справочной информации в специальном формате (RTF);
– создание файла проекта справочной системы;
– трансляция файлов справочной системы в двоичный формат (.hlp);
– проверка корректности формирования справочной системы с помощью программы Winhelp.exe;
– добавление в текст прикладной программы операторов, обеспечивающих вызов справочной информации из соответствующего раздела.

Составные части справочной системы:
– текст, составляющий содержание тематического раздела;
– термины (отдельные слова или фразы), которые выделяются шрифтом зеленого цвета и подчеркиванием;
– рисунки (bitmaps).

Для подготовки файла справочной системы в формате RTF можно использовать любой текстовый редактор, поддерживающий работу с данным форматом, например Microsoft Word. Каждый тематический раздел должен размещаться на отдельной странице. Для разметки файла справки (в формате RTF) используются специальные символы и сноски (к страницам).

Кодирование элементов справочной системы

Код	*Элемент*
Звездочка ‘*’ и соответствующая сноска	Метка условной компиляции
Знак доллара ‘$’ и соответствующая сноска	Заголовок тематического раздела
Знак фунта ‘#’ и соответствующая сноска	Контекстная строка (символьная константа, однозначно сопоставляемая контекстному числу, идентифицирующему раздел)
Заглавная буква K и соответствующая сноска	Ключевые слова (используются для поиска темы)

Знак «плюс» ‘+’ и соответствующая сноска	Последовательность просмотра
Перечеркнутый текст или текст с двойным подчеркиванием	Перекрестная ссылка
Подчеркнутый текст	Определение (термин, при выборе которого появляется дополнительное окно с детальным описанием термина)
Невидимый (скрытый) текст	Контекстная строка, идентифицирующая раздел, к которому происходит переход при выборе перекрестной ссылки или появляющийся в дополнительном окне при выборе определения. Должен следовать сразу за описанием термина

Специальные символы сносок (‘*’, ‘#’, ‘$’, ‘+’, ‘K’) должны перечисляться в начале раздела. Символ ‘*’, если используется, должен идти первым, порядок остальных произвольный. Для вставки специального символа следует:

– поместить курсор в начало страницы с описанием тематического раздела;

– вставить специальный символ как знак сноски;

– задать необходимый текст сноски (заголовок раздела, ключевые слова и т.д.).

Графические изображения можно помещать в текст раздела справочной системы, а можно хранить в отдельном файле, а в текст раздела вставлять одну из кодовых строк: {bmc filename.bmp}, {bml filename.bmp}, {bmr filename.bmp} — рисунок при этом размещается в центре (слева или справа соответственно).

Для трансляции файлов справочной системы в двоичный формат в состав инструментальных средств разработки (пакетов Borland Delphi, Borland C++ Builder, Microsoft Visual C++) входит программа Help Compiler (файл hwc.exe). Путь к программе Help Compiler в случае установки по умолчанию: C:\Program Files\Borland\Delphi7\Help\Tools.

Задание 1.

1. Создайте на рабочем диске папку с именем HELPDemo.
2. С помощью программы MS Word подготовьте файл справочной системы, включающий не менее трех разделов, задайте необходимые сноски и сохраните его в формате RTF в созданной папке.
4. В программе Help Compiler создайте файл проекта для подготовленного файла справки и оттранслируйте его в двоичный формат.

5. С помощью программы Winhelp.exe (winhlp32.exe) проверьте, правильно ли сформирован двоичный файл справочной системы.

Вызов справочной системы из приложения.

Для того чтобы программно вызвать справочную систему, нужно использовать функцию WinHelp.

Краткое описание функции WinHelp:

BOOL WinHelp(HWND hWndMain, LPCTSTR lpszHelp, UINT uCommand, DWORD dwData);

Параметр hWndMain — указатель на окно приложения, вызывающего справку. Параметр lpszHelp задает путь к файлу справки. Параметр uCommand определяет тип справки. Параметр dwData определяет дополнительные данные (номер темы). В случае успешного вызова функция возвращает ненулевое значение, в случае неудачи – нулевое.

Пример использования функции WinHelp.

```
. . .
const
   hlpItem = 100;
   hlpFile = 'helpfile.hlp';
. . .
  WinHelp(0, hlpFile, HELP_CONTEXT, hlpItem);
. . .
```

Задание 2.

1. С помощью Delphi создайте Windows-приложение и подключите к нему подготовленный файл справки.
2. Проверьте корректность работы справочной системы с Windows-приложением.

Контрольные вопросы

1. Как организована и из каких элементов состоит справочная система Windows-приложения?
2. Как создать справочную систему средствами Delphi?
3. Как вызвать нужный раздел справочной системы из приложения на Delphi?

Лабораторная работа 2.17. Использование компонента ActiveX для навигации в гипертекстовой среде

1. Создайте на рабочем диске папку **web.**
2. Загрузите **Delphi** и создайте новый проект.

3. Поместите на форму кнопку и компонент WebBrowser (стр. **Internet**).

4. Загрузите из сети Интернет или подготовьте самостоятельно гипертекстовый документ (файл с расширением .htm).

5. Задайте в разделе описания констант путь к этому документу. Например,

const p = 'D:\mydoc.htm';

6. В обработчике **OnClick** кнопки введите следующий код:

WebBrowser1.Navigate(p);

7. Запустите программу. Сделайте щелчок по кнопке и убедитесь в том, что подготовленный гипертекстовый документ загружен в окно web-браузера на форме вашего приложения.

8. Внесите в программу изменения, позволяющие задавать путь к гипертекстовым документам с помощью клавиатуры (добавьте строку ввода).

9. Запустите программу и убедитесь в ее работоспособности.

10. Преобразуйте разработанную в предыдущей лабораторной работе справочную систему в форму гипертекста (на языке HTML).

11. Запустите программу и проверьте ее работу.

Контрольные вопросы

1. Опишите основные отличия гипертекста на языке HTML от гипертекстовой справочной системы Windows (hlp-файлов)?

2. Какой компонент Delphi используется для навигации в среде гипертекста?

3. Какие поля и методы компонента WebBrowser используются для навигации?

3. ПРИЕМЫ РАБОТЫ С БАЗАМИ ДАННЫХ

Лабораторная работа 3.1. Использование BDE

В лабораторной работе изучается технология доступа к таблицам базы данных с помощью компонентов BDE. Используется демонстрационная база данных DBDEMOS, содержащая сведения о поставщиках (customer.db), заказах (orders.db) и т.д.

1. Создайте на рабочем диске папку с именем table.

2. Запустите Borland Delphi и создайте новый проект.

3. Поместите на форму компонент TTable (страница **BDE** Палитры компонентов), компонент TDataSource (страница **Data Access** Палитры компонентов), компоненты TDBGrid и TDBNavigator (страница **Data Control** Палитры компонентов). Задайте значения свойств компонентов следующим образом:

Table1.Name := OrdersTable;

Обратите внимание, что после изменения значения свойства **Name** название компонента меняется.

```
OrdersTable.DatabaseName := DBDEMOS;
OrdersTable.TableName := orders.db;
DataSource1.Name := OrdersSource;
OrdersSource.DataSet := OrdersTable;
DBGrid.Name := OrdersGrid;
OrdersGrid.DataSource := OrdersSource;
DBNavigator.Name := DBNav;
DBNav.DataSource = OrdersSource;
DBNav.ShowHint := True;
```

4. Сохраните проект в созданной ранее папке, задав для модуля имя unit1, для файла проекта — имя bdedemo.

5. Установите значение **true** для свойства Active компонента OrdersTable. После этого в компоненте OrdersGrid отобразится содержимое таблицы, определенное в свойстве OrdersTable.TableName, т.е. orders.db.

6. Запустите приложение на выполнение и изучите возможности компонента DBNavigator (как осуществляется навигация по таблице с помощью компонента DBNavigator, как можно добавить/удалить запись, отменить сделанные изменения).

7. Измените имена столбцов таблицы, отображаемых на экране с помощью компонента OrdersGrid, задав им русские названия (имена полей в таблице базы данных при этом не изменятся). Для этого выберите свойство Columns этого компонента. В окне **Editing OrdersGrid.Columns** щелкните по кнопке  (**Add All Fields**). Затем

сделайте щелчок по строке “2 — SaleDate”. В окне Object Inspector раскройте свойство **Title** и выберите свойство **Caption**. Вместо значения *«SaleDate»* задайте значение *«Дата продажи»*. Аналогично измените заголовки еще 2–3 столбцов таблицы на экране.

8. Запустите приложение на выполнение и проанализируйте его работу.

9. Для построения диаграммы по данным таблицы базы данных поместите на форму компонент TDBChart (страница **Data Control** Палитры компонентов). Задайте значения свойств компонента следующим образом:

DBChart1.Name := DataChart;

DataChart.ShowHint := True;

10. Сделайте щелчок правой кнопкой на диаграмме и из контекстного меню выберите команду **Edit Chart**.

11. Для добавления на диаграмму нового ряда (серии) данных в окне **Editing DataChart** на вкладке **Chart** выберите страницу **Series** и щелкните по кнопке **Add**.

12. В окне **TeeChart Gallery** на странице **Standard** выберите тип диаграммы *Line*, снимите флажок **3D** и щелкните по кнопке OK (см. рис. 3.1).

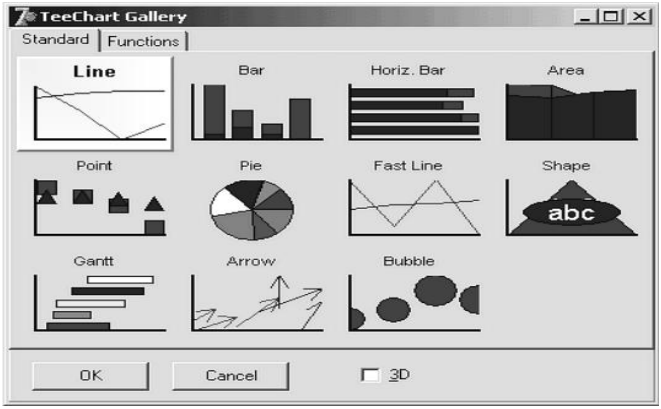

Рис. 3.1. Выбор типа диаграммы

13. Закройте окно **Editing DataChart**, щелкнув по кнопке **Close**.

14. Для того чтобы связать компонент Series1 с данными из таблицы базы данных, выберите из контекстного меню команду **Edit Chart**. В окне **Editing DataChart** выберите вкладку **Series**. Выберите из списка строку *Series1*. Перейдите на страницу **DataSource** и в следующем списке выберите строку *DataSet* (см. рис. 3.2).

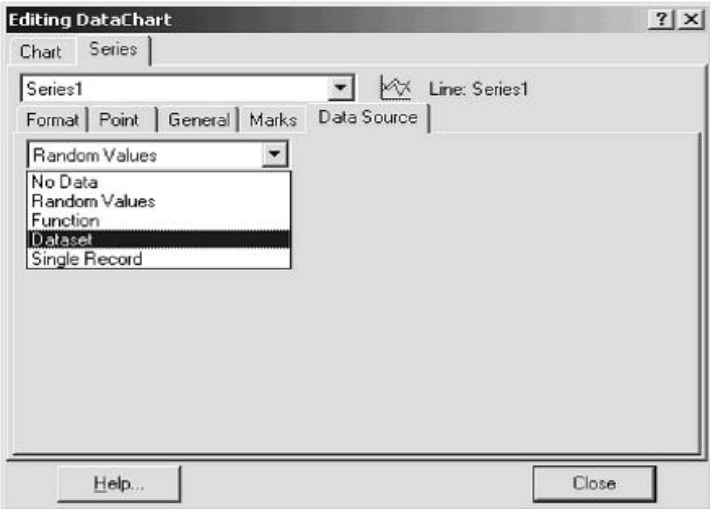

Рис. 3.2. Выбор источника данных для диаграммы

15. После этого для элемента DataSet выберите из списка значение OrdersTable, для элемента X — значение SaleDate, для элемента Y — значение AmountPaid (см. рис. 3.3). SaleDate и AmountPaid — это имена полей таблицы orders.db. Данные из этих полей таблицы будут отображаться на диаграмме.

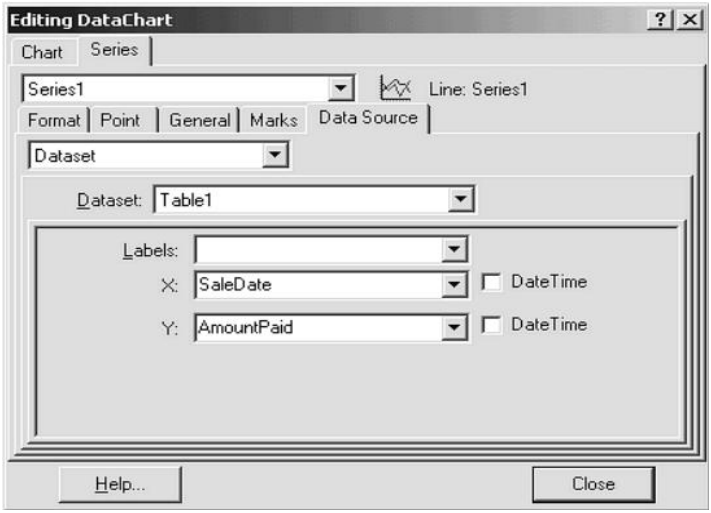

Рис. 3.3. Выбор источников данных для осей диаграммы

16. Запустите приложение на выполнение и проанализируйте его работу.

Задание для самостоятельной работы

1. Добавьте на диаграмму данные еще 1–2 полей таблицы orders.db.
2. Измените тип диаграммы на столбиковую (Bar).
3. Задайте для диаграммы заголовок «Объем продаж».
4. Создайте новую папку. С помощью программы Database Desktop (меню **Tools — Database Desktop** программы Borland Delphi) создайте таблицу базы данных Paradox 7 (меню **File — New — Table** программы Database Desktop). Структура таблицы приведена на рис. 3.4. Сохраните базу данных в созданной папке в файле с именем tree.db.
5. С помощью программы Database Desktop создайте псевдоним для созданной в п. 3 базы данных (**Tools — Alias manager**, кнопка **New**):

Database alias: LABDB
Driver type: STANDARD (драйвер для базы данных Paradox)
Path: <в папке, созданной в п. 4>

6. Создайте новый проект, подключитесь к созданной базе данных и добавьте в таблицу 3–4 записи произвольного содержания.

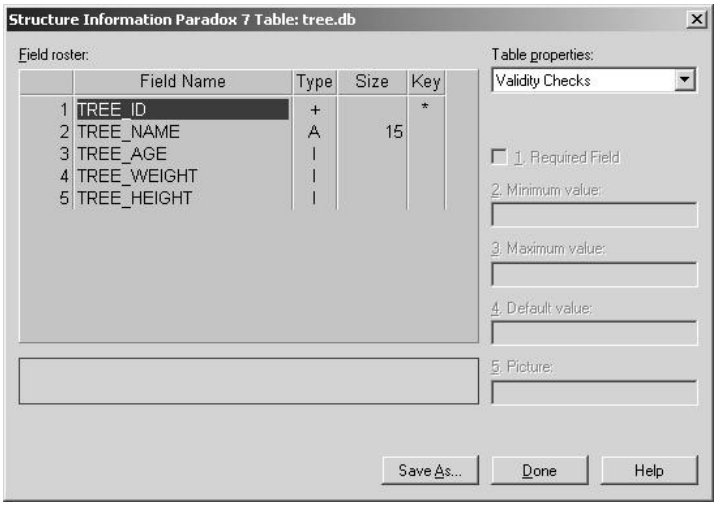

Рис. 3.4. Структура таблицы Paradox

7. Отобразите на диаграмме данные полей TREE_AGE, TREE_WIEGHT, TREE_HEIGHT таблицы tree.db.

Контрольные вопросы

1. Что такое псевдоним базы данных и для чего он используется?
2. Для чего предназначена программа Database Desktop?
3. Каким образом можно изменить заголовок диаграммы?
4. Для чего предназначено свойство Alignment компонента OrdersGrid.Columns[1]?
5. Каким образом можно отобразить на диаграмме данные таблицы БД?

Лабораторная работа 3.2. Синхронный просмотр данных в главной и подчиненной таблицах

В лабораторной работе изучается использование компонентов BDE для организации синхронного просмотра данных в главной и подчиненной таблицах БД, а также фильтрация записей таблиц на основе заданных условий. Используется демонстрационная база данных DBDEMOS. В качестве главной используется таблица поставщиков (customer.db), в качестве подчиненной — таблица заказов (orders.db).

1. Создайте на рабочем диске папку с именем bde_table2.
2. Запустите Borland Delphi и создайте новый проект.
3. Поместите на форму компонент TTable (страница **BDE** Палитры компонентов), компонент TDataSource (страница **Data Access** Палитры компонентов), компоненты TDBGrid и TDBNavigator (страница **Data Control** Палитры компонентов). Задайте значения свойств компонентов следующим образом:

Table1.Name := CustTable.

Обратите внимание, что после изменения значения свойства Name название компонента меняется.

```
CustTable.DatabaseName := DBDEMOS;
CustTable.TableName := customer.db;
DataSource1.Name := CustSource;
CustSource.DataSet: = CustTable;
DBGrid.Name := CustGrid;
CustGrid.DataSource := CustSource;
DBNavigator.Name := DBNav;
DBNav.DataSource = CustSource;
DBNav.ShowHint := True;
```

4. Сохраните проект в созданной ранее папке, задав для модуля имя unit1, для файла проекта — имя bdedemo2.

5. Установите значение *true* для свойства **Active** компонента CustTable. После этого в компоненте CustGrid отобразится содержимое таблицы, определенное в свойстве CustTable.TableName, т.е. customer.db.

6. Выполните построение проекта и запустите его на выполнение.

7. Поместите на форму компоненты для доступа к таблице заказов orders.db. При этом таблица customer.db будет выступать в качестве главной таблицы (master), а таблица orders.db — в качестве подчиненной (detail). Между главной и подчиненной таблицей можно установить связь через поля, имеющие одинаковые характеристики. Поля, по которым устанавливается связь между таблицами, должны иметь индексы.

Поместите на форму компоненты TTable, TDataSource и TDBGrid. Задайте значения свойств компонентов следующим образом:

Table2.Name := OrderTable;
OrderTable.DatabaseName := DBDEMOS;
OrderTable.TableName := orders.db;
OrderTable.MasterSource := CustSource;
OrderTable.IndexName := CustNo;
OrderTable.MasterFields := CustNo; (в окне **Field Link Designer** выберите поле CustNo из списка **Master Fields**)
DataSource1.Name := OrderSource;
OrderDataSource.DataSet := OrderTable;
DBGrid1.Name := OrderGrid;
OrderGrid.DataSource := OrderSource;
OrderTable.Active := true;

8. Запустите приложение на выполнение и проанализируйте его работу. Обратите внимание, что при перемещении по записям в таблице поставщиков в таблице заказов отображаются записи, соответствующие номеру поставщика (CustNo).

9. Поместите на форму компонент TDBLookupListBox. Задайте значения свойств этого компонента следующим образом:

DBLookupListBox1.DataSource := CustSource;
DBLookupListBox1.DataField := CustNo;
DBLookupListBox1.ListSource := OrderSource;
DBLookupListBox1.KeyField := CustNo;
DBLookupListBox1.ListField := AmountPaid;

Компонент OrderGrid сделайте невидимым (Visible := false).

10. Запустите приложение на выполнение и проанализируйте его работу.

11. Для того чтобы вывести не все записи таблицы customer.db, а только записи, удовлетворяющие определенным условиям, задайте для свойства **Filtered** компонента CustTable значение *true*, а для свойства **Filter —** условие отбора: "*CustNo>6000*".

12. При фильтрации записей можно использовать логические условия, например **CustTable.Filter** := "*CustNo>6000 AND Country='US'* ".

13. Запустите приложение и убедитесь, что выводятся только нужные записи.

Задание для самостоятельной работы

1. Разработайте структуру двух таблиц для хранения данных о книгах в библиотеке и об авторах этих книг. Структура таблиц должна быть разработана таким образом, чтобы эти таблицы можно было использовать в качестве главной и подчиненной.

2. Создайте новую папку. С помощью программы Database Desktop (меню **Tools — Database Desktop** программы Borland Delphi) создайте базу данных Paradox 7, включающую две таблицы, разработанные в п. 1, и сохраните ее в созданной папке. Добавьте в таблицы 3–4 записи произвольного содержания.

3. С помощью программы Database Desktop создайте псевдоним для созданной в п. 2 базы данных (**Tools — Alias manager**, кнопка **New**):

Database alias: LIBDB

Driver type: STANDARD (драйвер для базы данных Paradox)

Path: <в папке, созданной в п. 2>

4. Создайте новый проект и организуйте синхронный просмотр записей созданных таблиц.

Контрольные вопросы

1. Что такое набор данных (dataset)?

2. В чем различие компонентов TDBLookupListBox и TDBComboBox?

3. Каким образом можно выполнить фильтрацию записей по заданному условию?

4. Какие условия должны быть выполнены для того, чтобы две таблицы базы данных можно было использовать в качестве главной и подчиненной?

5. Как организовать в программе синхронный просмотр данных в главной и подчиненной таблицах?

Лабораторная работа 3.3. Использование модуля данных

В лабораторной работе изучается использование модуля данных для размещения невизуальных компонентов BDE и обработка исключительных ситуаций при работе с базами данных.

1. Создайте на рабочем диске папку с именем dbmdemo.

2. Запустите Borland Delphi и создайте новый проект. Задайте для формы имя DBMDEMO. Сохраните проект в созданной в п. 1 папке: модуль — под именем mainunit, проект — под именем dbmdemo.

3. Создайте модуль данных (**File — New — Data Module**) для размещения невизуальных компонентов для работы с базами данных. Задайте для модуля данных имя DemoDM. Сохраните файл модуля данных в созданной в п. 1 папке под именем dataunit.

4. Поместите в модуль данных компонент TTable и компонент TDataSource. Задайте значения свойств компонентов следующим образом:

Table1.DatabaseName := DBDEMOS;
Table1.TableName := orders.db;
DataSource1.DataSet := Table1;

5. Поместите на форму компонент TDBGrid. Задайте свойству **DataSource** компонента DBGrid1 значение *DemoDM.DataSource1*.

6. Поместите на форму компонент TButton. Задайте свойству **Caption** компонента Button1 значение «*Подключиться к базе данных*». В метод-обработчик **OnClick** поместите следующий код:

```
try
    DemoDM.Table1.Open;
except
    on EDatabaseError do
    begin
MessageBox(0, 'Ошибка открытия базы данных', 'Error', MB_OK);
        exit;
    end;
end;
Button1.Enabled := False;
```

7. Поскольку класс для обработки исключительных ситуаций при работе с базами данных EDatabaseError описан в модуле DB, нужно добавить этот модуль в секцию uses модуля mainunit. Откомпилируйте проект, чтобы убедиться в отсутствии ошибок.

8. Поместите на форму DBDemo еще один компонент TButton. Задайте свойству **Caption** компонента Button2 значение *«Завершить работу»*, свойству **Enabled** — значение *«False»*. В метод-обработчик **OnClick** поместите следующий код:

```
try
```

```
    DemoDM.Table1.Close;
  except
    on EDatabaseError do
    begin
  MessageBox(0, 'Ошибка закрытия базы данных', 'Error', MB_OK);
        exit;
     end;
end;
  Button1.Enabled := True;
  Button2.Enabled :=False;
```

9. Для того чтобы проверить, как прикладная программа обрабатывает исключительные ситуации, нужно выполнить команды меню **Tools** — **Debugger Options** и на вкладке **Language Exceptions** снять флажок **Stop on Delphi Exceptions** (см. рис. 3.5).

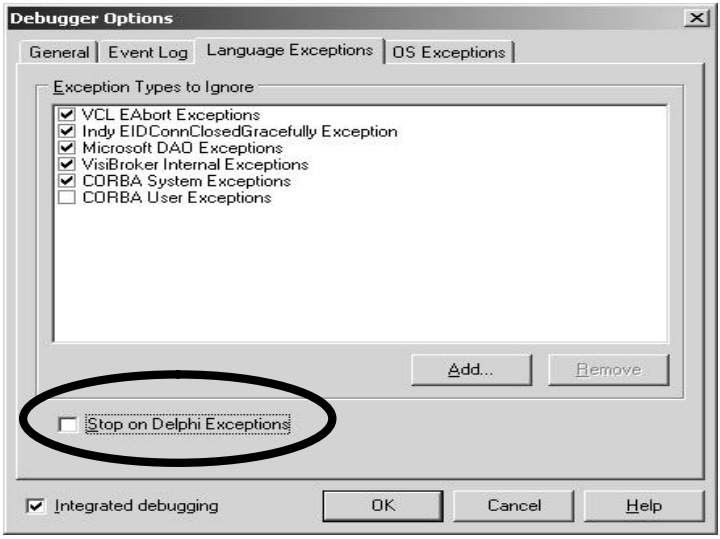

Рис. 3.5. Отключение обработки исключительных ситуаций

10. Далее, задайте для свойства **DatabaseName** компонента **Table1** имя несуществующей базы данных, например выберите значение «Файлы DBase».

11. Выполните построение проекта, запустите его на выполнение и проанализируйте его работу.

12. Выполните задания для самостоятельной работы.

Задание для самостоятельной работы

1. Измените код процедур TForm1.Button1Click и TForm1.Button2Click таким образом, чтобы получать более подробную информацию об исключительной ситуации (указание: используйте свойство Message класса EDatabaseError).

2. Добавьте в приложение компонент TDBEngineErrorDlg и используйте его методы (например, ShowException) в процедурах обработки исключительных ситуаций.

Контрольные вопросы

1. Для чего предназначен компонент TTable?
2. Почему следует использовать модули данных в приложениях, работающих с базами данных?
3. Какие еще классы VCL для обработки исключительных ситуаций можно использовать в приложениях баз данных?
4. В чем отличие классов EDatabaseError и EDBEngineError?
5. Для чего нужно свойство Errors класса EDBEngineError?

Лабораторная работа 3.4. Использование компонентов Rave Reports для формирования отчетов

В лабораторной работе изучается использование компонентов Rave Reports для формирования отчетов в приложениях баз данных.

1. Создайте на рабочем диске папку с именем dbdemo.

2. Запустите Borland Delphi и создайте новый проект. Задайте для формы имя DBRDEMO. Сохраните проект в созданной в п. 1 папке: модуль — под именем mainunit, проект — под именем dbrdemo.

3. Создайте модуль данных (**File — New — Data Module**) для размещения невизуальных компонентов для работы с базами данных. Задайте для модуля данных имя DataM. Сохраните файл модуля данных в созданной в п. 1 папке под именем dataunit.

4. Поместите в модуль данных компоненты BDE, необходимые для работы с двумя таблицами аналогично п. 3–7 лабораторной работы 3.2.

Проверьте, чтобы свойству **Active** компонента CustTable было присвоено значение *True*.

5. Поместите в на главную форму компонент RvDataSetConnection (страница **Rave** Палитры компонентов).

6. Для свойства **DataSet** компонента RvDataSetConnection1 задайте значение *«DataM.CustTable»*.

7. Вызовите утилиту для разработки отчетов Rave Reports (**Tools — Rave Designer**).

8. Выполните команды **File — New Data Object** утилиты Rave Reports.

9. В окне **Data Connections** выберите элемент **Direct Data View** и щелкните по кнопке **Next**.

10. На следующем шаге в окне **Data Connections** выберите (единственный) элемент **RvDataSetConnection1 (DT)** и щелкните по кнопке **Finish**.

11. В правой части окна утилиты Rave Reports, в которой отображается дерево проекта отчета, раскройте узел **Data View Dictionary**, а затем — узел **DataView1**. Обратите внимание, что поле данных таблицы CustTable, имя которой указано в качестве значения поля DataSet компонента **RvDataSetConnection1**, отображается в узле DataView1.

12. Выполните команды **Tools — Report Wizards — Simple Table** утилиты Rave Reports для запуска мастера формирования отчета в пошаговом режиме.

13. В окне **Simple Table** выберите элемент **DataView1** и щелкните по кнопке **Next**.

14. Далее в окне **Simple Table** отметьте флажками поля *CustNo, Company, State, LastInvoiceDate* для включения в отчет (для включения в отчет всех полей нужно щелкнуть по кнопке **All**). Переместите поле *LastInvoiceDate* на третью позицию сверху и щелкните по кнопке **Next**.

15. На следующем шаге работы мастера в строке **Report Title** задайте имя отчета: «Отчет о клиентах».

16. На следующем шаге работы мастера выберите шрифты для использования в отчете: в качестве шрифта заголовка отчета (**Title Font**) задайте MS Sans Serif, жирный, размер 24; в качестве шрифта заголовка раздела (**Caption Font**) задайте шрифт MS Sans Serif, жирный курсив, размер 14; в качестве шрифта текста отчета (**Body Font**) задайте MS Serif, обычный, 10. Щелкните по кнопке **Generate** для формирования отчета.

17. Выполните команды **Zoom — 50%** и проанализируйте размещение на странице элементов отчета.

18. Сохраните проект, созданный с помощью утилиты Rave Reports, в той же папке, что проект Delphi (**File — Save As**). Задайте для проекта отчета имя rpt1.

19. Для проверки корректности формирования отчета выполните команды **File — Execute Report** утилиты Rave Reports. Выберите режим предварительного просмотра (**Preview**), посмотрите сформированный отчет, а затем закройте окно предварительного просмотра (**File — Exit**).

20. Не завершая работу с утилитой Rave Reports, сверните (минимизируйте) окно этой программы и перейдите в проект приложения в среде Borland Delphi.

21. Добавьте на главную форму проекта компонент TRvProject (страница **Rave** Палитры компонентов).

22. В качестве значения свойства **ProjectFile** компонента RvProject1 задайте имя файла проекта (rpt1.rav).

23. Добавьте на главную форму проекта компонент TButton. В обработчик OnClick компонента Button1 поместите код для формирования отчета:

RvProject1.Execute;

24. Запустите приложение и сгенерируйте отчет.

25. Выполните задания для самостоятельной работы.

Задание для самостоятельной работы

1. Сделайте так, чтобы в отчете в качестве фонового рисунка использовался файл с каким-либо растровым изображением (указание: на старнице **Standard** утилиты Rave Reports выберите компонент **Bitmap component**).

2. Сделайте так, чтобы по умолчанию отчет выводился не в режиме предварительного просмотра, а на принтер (указание: используйте компонент TRvSystem).

Контрольные вопросы

1. Опишите процедуру подготовки отчета с использованием утилиты Rave Reports.

2. Каким образом можно использовать в приложении компоненты Rave Reports?

3. Каково назначение компонента TRaveDirectDataView?

4. Каким образом можно изменить внешний вид (дизайн) отчета средствами утилиты Rave Reports?

5. Для чего можно использовать компонент TRvSystem?

Лабораторная работа 3.5. Использование вычисляемых полей

В лабораторной работе изучается использование вычисляемых полей и способы поиска данных в таблицах базы данных.

1. Создайте на рабочем диске папку с именем dbsdemo.

2. Запустите Borland Delphi и создайте новый проект. Задайте для формы имя DBSDEMO. Сохраните проект в созданной в п. 1 папке: модуль — под именем mainunit, проект — под именем dbsdemo.

3. Создайте модуль данных (**File — New — Data Module**) для размещения невизуальных компонентов для работы с базами данных. Задайте для модуля данных имя MD. Сохраните файл модуля данных в созданной в п. 1 папке под именем dataunit.

4. Поместите в модуль данных компоненты TTable и TDataSource. Задайте значения свойств компонентов следующим образом:

Tab1e1.Name := OrderTable;
OrderTable.DatabaseName := DBDEMOS;
OrderTable.TableName := orders.db;
OrderTable.Active := True;
DataSource1.Name := OrderDataSource;
OrderDataSource.DataSet := OrderTable;

5. Поместите на форму компонент TDBGrid. Задайте значения свойств компонента следующим образом:

DBGrid1.Name := OrderGrid;
OrderGrid.DataSource := MD.OrderDataSource;
OrderGrid.Options.dgEditing := False;

6. Запустите приложение и проверьте корректность его работы.

7. Добавьте в таблицу еще одно поле, содержащее значение поля AmountPaid, переведенное из рублей в доллары. Такое поле называется вычисляемым. Для этого сделайте двойной щелчок по компоненту OrderTable в модуле MD для вызова редактора полей таблицы базы данных Field Editor.

8. В окне **MD.OrderTable** сделайте щелчок правой кнопкой и выберите команду **New Field**.

9. В окне **New Field** в строке **Name** задайте имя поля *«AMDollars»*, в строке **Type** задайте тип поля *«Float»*. Для элемента **Field Type** выберите тип *Calculated* (см. рис. 3.6) и щелкните по кнопке OK.

10. Добавьте столбец для нового поля в таблицу, отображаемую на экране с помощью компонента OrderGrid. Для этого сделайте щелчок правой кнопкой мыши на компоненте OrderGrid и выберите команду **Columns Editor**. В окне **Editing OrderGrid.Columns** также сделайте щелчок правой кнопкой и выберите команду **Add**, после чего в эту таблицу добавится новое поле. С помощью Инспектора объектов для свойства **FieldName** этого поля выберите из списка значение *«AMDollars»*.

11. С помощью утилиты **Object TreeView** найдите поле **AMDollars** в таблице OrderTable и создайте обработчик события OnGetText для этого поля.

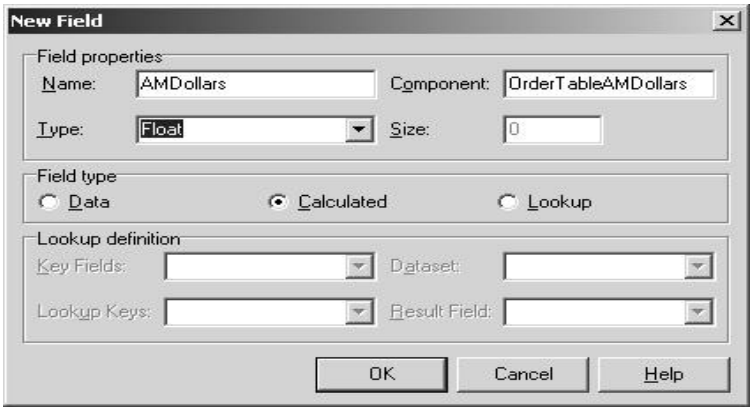

Рис. 3.6. Добавление нового поля

12. Поместите в обработчик события **OnGetText** для вычисляемого поля AMDollars код для перевода из рублей в доллары (по условному курсу 30 рублей за 1 доллар):

Text:=FloatToStr(OrderTable.FieldByName('AmountPaid').AsFloat/30);

13. Запустите приложение и проверьте корректность его работы.

14. Найдите все записи таблицы, в поле OrderNo которых содержится значение «1005». Для этого поместите на форму компоненты TLabel, TEdit и TButton и используйте методы Locate и Lookup компонента TTable. С помощью свойства Caption компонента Label1 опишите объект поиска, например «Значение поля OrderNo», компонент Edit1 используйте для ввода искомого значения поля OrderNo, в обработчик OnClick компонента Button1 поместите следующий код:

if MD.OrderTable.Locate ('OrderNo', VarArrayOf([‘1005’]), [loPartialKey]) then ShowMessage('Запись найдена')
else ShowMessage('Запись не найдена');

15. Аналогично используйте для поиска метод Lookup компонента TTable.

```
Var v : Variant;
. . .
v:=MD.OrderTable.Lookup('OrderNo',VarArrayOf([‘1005’]), ‘OrderNo’);
if not (VarType(v) in [VarNull]) then ShowMessage('Запись найдена')
  else ShowMessage('Запись не найдена');
. . .
```

16. Запустите приложение и проверьте корректность его работы.

17. Выполните задания для самостоятельной работы.

Задание для самостоятельной работы

1. Сделайте так, чтобы дробная часть поля AMDollars отображалась не полностью, а только в виде двух знаков после запятой.

2. Найдите какую-либо запись таблицы с помощью индексного поиска (указание: используйте метод FindKey компонента TTable).

3. С помощью методов SetKey, EditKey, GotoToKey, SetRange компонента TTable выполните поиск в выбранном диапазоне записей таблицы.

Контрольные вопросы

1. Для чего используется компонент TField?

2. Что такое вычисляемое поле?

3. Для чего используется свойство DisplayFormat компонента TField?

4. В чем отличие методов FindKey, Locate и Lookup компонента TTable?

5. Что такое вариантные массивы и как их использовать?

6. Чем отличается поиск в диапазоне от использования фильтров?

Лабораторная работа 3.6. Программное формирование запросов

В лабораторной работе изучается использование компонентов TDataSource и TQuery для формирования запросов.

1. Создайте на рабочем диске папку с именем dbdemo.

2. Запустите Borland Delphi и создайте новый проект. Задайте для формы имя DBDEMO. Поместите на форму компонент DBGrid и два компонента TButton. Задайте значения свойств компонентов следующим образом:

DBGrid1.Name := DataGrid;
Button1.Name := OpenBtn;
Button1.Caption := «Подключиться к базе данных»;
Button2.Name := CloseBtn;
Button2.Caption := «Отключиться от базы данных»;

3. Сохраните проект в созданной в п. 1 папке: модуль — под именем mainunit, проект — под именем dbqdemo.

4. Создайте модуль данных (**File — New — Data Module**) для размещения невизуальных компонентов для работы с базами данных. Задайте для модуля данных имя DM. Сохраните файл модуля данных в созданной в п. 1 папке под именем dataunit.

5. Поместите в модуль данных компонент TTable и компонент TDataSource. Задайте значения свойств компонентов следующим образом:

Table1.Name := DataTable;
DataTable.DatabaseName := DBDEMOS;
DataTable TableName := vendors.db;
DataSource1.Name := TDS;
TDS.DataSet := DataTable;
DataGrid.DataSource := DM.TDS;

6. Установите свойство **Active** таблицы в *«True»*, запустите программу и проверьте корректность ее работы.

7. Поместите в модуль данных компонент TQuery (страница **BDE** Палитры компонентов) и еще один компонент DataSource. Задайте значения свойств компонентов следующим образом:

Query1.Name := VendorsQuery;
VendorsQuery.DatabaseName = DBDEMOS;
VendorsQuery.DataSource := TDS;
// текст запроса SQL
*VendorsQuery.SQL := 'select * from vendors';*
DataSource2.Name := QDS;
QDS.DataSet := VendorsQuery;

8. Измените источник данных для компонента DataGrid:

DataGrid.DataSource := DM.QDS;

и установите значение свойства Active таблицы в «*False*».

9. В метод-обработчик **OnClick** компонента OpenBtn поместите следующий код для выполнения запроса:

```
. . .
DM.VendorsQuery.SQL.Close;
DM.VendorsQuery.SQL.Clear;
DM.VendorsQuery.SQL.Add('select * from vendors');
  try
    DM.VendorsQuery.Open;
except
     MessageBox(0,'Ошибка запроса','Error',MB_OK);
  end;
. . .
```

10. В метод-обработчик **OnClick** компонента CloseBtn поместите следующий код для закрытия базы данных:

```
  try
    DM.DataTable.Close;
except
     MessageBox(0,'Ошибка закрытия базы данных','Error',MB_OK);
  end;
```

11. Выполните построение проекта, запустите его на выполнение и проверьте корректность работы.

12. Внесите изменение в текст запроса SQL, чтобы результатом выполнения запроса было множество записей, значение поля Country которых равно «Canada» (‘select * from vendors where Country = “Canada”’).

13. Запустите приложение и проверьте корректность его работы.

14. Внесите изменение в текст запроса SQL, чтобы результатом выполнения запроса стала попытка удаления записей, значение поля VendorNo которых равно заданному значению (например, ‘delete from vendors where VendorNo = “ 2674”’). Код метода-обработчика **OnClick** компонента OpenBtn будет выглядеть так:

```
. . .
DM.VendorsQuery.SQL.Clear;
DM.VendorsQuery.SQL.Add('delete from vendors where VendorNo =
"2674"');
try
   DM.VendorsQuery.ExecSql;
except
   MessageBox(0, ’Ошибка запроса’,'Error',MB_OK);
end;
. . .
```

15. Запустите приложение и проверьте корректность его работы. Удалось ли удалить запись и почему?

16. Выполните задания для самостоятельной работы.

Задание для самостоятельной работы

1. Сделайте так, чтобы текст запроса SQL пользователь мог вводить с клавиатуры.

2. Реализуйте запрос с параметром (параметром является значение поля VendorNo таблицы vendors.db). *Указание*: используйте свойство **Params** компонента Query и компонент DBComboBox, а в качестве источника данных для компонента DBComboBox используйте результат соответствующего запроса.

Контрольные вопросы

1. Для чего предназначен компонент TQuery? Чем он отличается от компонента TTable?

2 Что такое SQL и для чего он предназначен? Какие операторы SQL вы знаете?

3. Каким образом можно динамически изменять текст запроса SQL?

4. Что такое параметрический запрос?

5. В чем различие между методами ExecSQL и Open компонента TQuery?

6. Каким образом с помощью SQL можно создавать таблицы базы данных?

Лабораторная работа 3.7. Многомерное представления данных

В лабораторной работе изучаются компоненты VCL для многомерного представления данных. Используется демонстрационная база данных DBDEMOS, содержащая сведения о поставщиках, заказах, сотрудниках, товарах и т.д.

1. Создайте на рабочем диске папку с именем DCube.
2. С помощью программы BDE Administrator или Database Desktop проверьте наличие псевдонима (alias) базы данных DBDEMOS. Эта база данных содержит связанные таблицы заказчиков (customer.db) и их заказов (orders.db).
3. Запустите Borland Delphi.
4. Выберите на палитре компонентов страницу **Decision Cube** с компонентами для многомерного представления данных.
5. Создайте новое приложение и поместите на форму следующие компоненты: TDecisionCube, TDecisionQuery, TDecisionSource, TDecisionPivot, TDecisionGrid.
6. Задайте значения свойств данных компонентов следующим образом:

```
DecisionCube1.DataSet := DecisionQuery1;
DecisionCube1.ShowProgressDialog := true;
DecisionQuery1.DatabaseName := DBDEMOS;
DecisionSource1.DecisionCube := DecisionCube1;
DecisionPivot1.DecisionSource := DecisionSource1;
DecisionPivot1.ButtonAutoSize := false;
DecisionGrid1.DecisionSource := DecisionSource1;
```

7. Выберите компонент DecisionQuery1 и с помощью правой кнопки мыши вызовите контекстное меню. В контекстном меню выберите пункт **Decision Query Editor**. В окне **Decision Query Editor** щелкните по кнопке **SQL Builder,** чтобы создать SQL-запрос к двум таблицам.
8. В поле **Table** окна **SQL Builder** выберите из списка сначала имя файла базы данных заказчиков (customer.db), а затем — заказов (orders.db).
9. Отметьте в таблице Customer поля CustNo, Company, City, а в таблице Orders — поля OrderNo, SaleDate, AmountPaid. Выбранные поля попадут в поле **OutputFields** на вкладке **Grouping**. Выделите шесть выбранных полей и, щелкнув по кнопке Add, перенесите их в поле Grouped on.

10. Закройте окно **SQL Builder** и вернитесь в окно **Decision Query Editor**. Для этого в окне **SQL Builder** выберите команды **File — Exit** и ответьте *Yes* на вопрос о сохранении изменений в запросе.

11. В окне **Decision Query Editor** в списке **List of Available Fields** выделите шесть выбранных полей и перенесите их в список **Dimensions**.

12. Затем сформируйте выражения для агрегатных функций в поле **Summaries**. Для этого выберите в списке **List of Available Fields** поле Orders.OrderNo и перенесите его в поле **Summaries** с использованием функции суммирования (sum). Затем повторите данную процедуру для поля Orders.AmountPaid с использованием функций sum и count. Таким образом, в поле **Summaries** должно быть следующее:

SUM(Orders.OrderNo)

SUM(Orders.AmountPaid)

COUNT(Orders. AmountPaid)

13. В окне **Decision Query Editor** перейдите на вкладку **SQL Query** и посмотрите текст сформированного запроса на языке SQL. Щелкните по кнопке OK.

14. Установите значение *true* свойства **Active** компонента DecisionQuery1, чтобы увидеть результаты выполнения запроса на стадии проектирования программы. При появлении сообщений о недостатке памяти откройте контекстное меню для компонента DecisionCube1 и выберите команду **Decision Cube Editor.** В окне **Decision Cube Editor** на вкладке **Dimension Settings** при необходимости можно модифицировать параметры агрегирования запроса, а на вкладке **Memory Control** — параметры куба решений. При необходимости удалите из списка **Summaries** окна **Decision Query Editor** две последних строки. В случае успеха в ячейках компонента DecisionGrid1 отобразится результат выполнения запроса.

15. Запустите программу и проверьте корректность ее функционирования.

16. Выполните задание 1 для самостоятельной работы.

17. Добавьте к проекту еще одну форму и поместите на нее компонент **TDecisionGraph**.

18. Задайте для этого компонента следующие значения свойств:

DecisionGraph1.DecisionSource := Form1.DecisionSource1;

19. Сделайте так, чтобы форму с компонентов DecisionGraph1 можно было вызвать из основной формы (поместите на основную форму соответствующую кнопку или меню).

20. Запустите программу еще раз и проверьте корректность ее функционирования.

21. Выполните задание 2 для самостоятельной работы.

22. Сохраните проект в созданной папке под именем DecisionCubeProject.

Задание для самостоятельной работы

1. С помощью компонента DecisionPivot1 измените способы группировки данных и посмотрите, как изменяется вид данных в компоненте DecisionGrid1.

2. С помощью компонента DecisionPivot1 измените способы группировки данных и посмотрите, как изменяется вид диаграммы (компонент DecisionGraph1).

Контрольные вопросы

1. Что такое «многомерные данные»?

2. Какие компоненты VCL можно использовать для многомерного представления данных?

3. В чем особенность запросов SQL при работе с многомерными данными?

4. Какие агрегатные функции можно использовать при формировании запросов SQL с помощью **Decision Query Editor**?

5. Для чего предназначен компонент TDecisonPivot? В чем сходство и различие компонентов TDecisonPivot и TDBNavigator?

Лабораторная работа 3.8. Компоненты ADO

В лабораторной работе изучается технология доступа к данным с помощью компонентов ADO (Microsoft ActiveX Data Objects).

1. Создайте на рабочем диске папку с именем ADODemo.

2. Запустите Borland Delphi.

3. Выберите на палитре компонентов страницу **ADO**.

4. Создайте новое приложение (**File — New Application**) и поместите на форму компоненты TADOConnection и TADOTable.

5. Задайте значения свойств компонента TADOConnection следующим образом:

ADOConnection1.Provider := MicrosoftJet.OLEDB.4.0
ADOConnection1.LoginPrompt := false

6. Щелкните по строке **ConnectionString**. После этого появится окно **Form1.ADOConnection1 Connection String** (Form1 — имя формы приложения), с помощью которого можно установить связь с базой данных.

7. В окне **Form1.ADOConnection1 Connection String** выберите переключатель **Use Data Link File** и щелкните по кнопке **Browse**. В окне выбора файлов выберите файл BCDEMOS.udl. Запишите путь к

этому файлу, который появится в строке редактирования в окне **Form1.ADOConnection1 Connection String**. Щелкните по кнопке **OK**.

8. Задайте значения свойств компонента TADOTable следующим образом:

ADOTable1.Connection := ADOConnection1
ADOTable1.TableName := customer

9. Для проверки задайте свойству **Connected** компонента ADOConnection1 значение true, свойству **Active** компонента ADOTable1 значение true.

10. Для отображения данных таблицы customer на экране поместите на форму компоненты DBGrid и DataSource. Задайте значения свойств компонента DataSource следующим образом:

DataSource1.DataSet := ADOTable1
DataSource1.Enabled := true
DBGrid1.DataSource := DataSource1
DBGrid1.Enabled := true

11. Выведите на экран значение свойства **ConnectionString** компонента **ADOConnection1** (для этого можно использовать, например, компонент TMemo).

12. Поместите на форму компонент **TADOQuery** и задайте значения его свойств:

ADOQuery1.DataSource := DataSource1
ADOQuery1.ConnectionString := ...\BCDEMOS.udl
// текст запроса SQL
ADOQuery1.SQL := «select company from customer»
ADOQuery1.Active := true

13. Поместите на форму компонент DataSource2 и для свойства **DataSet** задайте значение ADOQuery1.

14. Измените значение свойства **DataSource** компонента DBGrid1: вместо DataSource1 укажите DataSource2.

15. Откомпилируйте и запустите на выполнение созданный проект.

16. Сохраните проект в созданной при выполнении п. 1 папке под именем ADODemo.

17. Выполните задание для самостоятельной работы.

Задание для самостоятельной работы

1. Используя компоненты ADODataSet и DataSource3, отобразите на экране содержимое записей поля City таблицы customer. *Указание*: измените значение свойства **CommandText** компонента ADODataSet1, значение свойства **DataSet** компонента DataSource3, значение свойства **DataSource** компонента DBGrid1.

2. С помощью компонента **ADOCommand** выполните удаление и добавление одной записи в таблице customer. *Указание*: используйте свойства **CommandType**, **CommandText, Parameters**, метод **Execute()** компонента ADOCommand1.

Контрольные вопросы

1. Что такое провайдеры ADO?
2. В чем отличие доступа к данным с помощью технологии ADO и с помощью BDE?
3. Какие параметры задаются в свойстве **ConnectionString** компонента **ADOConnection**?
4. Для чего используется файл с расширением .udl?
5. Для чего используется свойство **Parameters** компонента ADOCommand?
6. Для чего используется свойство **DefaultDatabase** компонента **ADOConnection**?

Лабораторная работа 3.9. Использование компонентов InterBase

В лабораторной работе изучается технология работы с сервером баз данных Borland Interbase с помощью компонентов InterBase. Использование компонентов InterBase имеет следующие преимущества: повышение скорости работы за счет непосредственного использования API InterBase, улучшенное управление транзакциями (компонент TIBTransaction), получение сведений о состоянии БД без прямого обращения к ее системным таблицам (компонент TIBDatabaseInfo), отслеживание состояния запросов к БД (компонент TIBSQLMonitor).

1. Создайте на рабочем диске папку с именем ibdemo.
2. Проверьте, установлен ли сервер баз данных Borland Interbase. Если установлен, запустите IBConsole (**Пуск — Программы — Interbase—IBConsole**) и проверьте наличие демонстрационной базы данных Employee.gdb (путь по умолчанию "..\Program Files \ Borland \ InterBase \ examples \ Database\ EMPLOYEE.GDB"). Для подключения используйте параметры по умолчанию: user name=sysdba, password=masterkey.
3. Запустите Borland Delphi.
4. Выберите на палитре компонентов страницу **Interbase** и поместите на форму компонент IBDatabase. Задайте значения свойств компонента IBDatabase следующим образом:

IBDatabase1.LoginPrompt := false.

Затем поместите на форму компонент **IBTransaction**. Компонент IBTransaction инкапсулирует средства управления транзакцией для соединения с сервером InterBase. Задайте значения свойств компонента IBTransaction следующим образом:

IBTransaction1.DefaultDatabase := IBDatabase1

5. Затем поместите на форму компонент IBTable. Задайте значения свойств компонента IBTable следующим образом:

IBTable1.Database := IBDatabase1
IBTable1.DefaultTransaction := IBTransaction1
IBTable1.TableName := Employee
IBTable1.Active := True

6. Затем поместите на форму компонент DataSource. Задайте значения свойств компонента DataSource следующим образом:

DataSource1.DataSet := IBTable1

7. Поместите на форму компонент DBGrid для управления отображением данных. Задайте значения свойств компонента DBGrid1 следующим образом:

DBGrid1.DataSource := DataSource1

8. Откомпилируйте проект, чтобы убедиться в отсутствии ошибок, и запустите его на выполнение.

9. Выполните запрос к таблице базы данных. Для этого поместите на форму компонент IBQuery. Задайте значения свойств компонента IBQuery следующим образом:

IBQuery1.Database := IBDatabase1
*IBQuery1.SQL :="select * from EMPLOYEE" // текст запроса SQL*
IBQuery1.Active := true

Затем поместите на форму компонент DataSource. Задайте значения свойств компонента DataSource2 следующим образом:

DataSource2.DataSet := IBQuery1

Измените значение свойства DataSource компонента DBGrid1, чтобы отобразить результаты запроса SQL:

DBGrid1.DataSource := DataSource2

10. Откомпилируйте проект, чтобы убедиться в отсутствии ошибок, и запустите его на выполнение.

11. Поместите на форму компонент IBDataSet. Задайте значения свойств компонента IBDataSet1 следующим образом:

IBDataSet1.Database := IBDatabase1
*IBDataSet1.SelectSQL :="select * from EMPLOYEE"*
IBDataSet1.Active := true

Затем поместите на форму компонент DataSource. Задайте значения свойств компонента DataSource3 следующим образом:

DataSource3.DataSet := IBDataSet1

Измените значение свойства DataSource компонента DBGrid1, чтобы отобразить результаты запроса SQL:

DBGrid1.DataSource := DataSource3

12. Откомпилируйте проект, чтобы убедиться в отсутствии ошибок, и запустите его на выполнение.

13. Для получения информации о базе данных Interbase поместите на форму компоненты IBDatabaseInfo и Memo. Задайте значения свойств компонента IBDatabaseInfo следующим образом:

IBDatabaseInfo1.Database := IBDatabase1

В метод **OnActivate()** компонента Form1 добавьте следующий код:

```
Memo1.Lines.Clear();
Memo1.Lines.Add( IBDatabaseInfo1.Version );
Memo1.Lines.Add( IBDatabaseInfo1.DBFileName );
Memo1.Lines.Add(IBDatabaseInfo1→PageSize);
```

14. Откомпилируйте проект, чтобы убедиться в отсутствии ошибок, запустите его на выполнение и прочитайте информацию о базе данных.

15. Сохраните проект в созданной при выполнении п. 1 папке под именем ibdemo.

Задание для самостоятельной работы

1. Измените значение поля **SQL** компонента Query1 (текст запроса SQL) таким образом, чтобы извлечь из таблицы данные о сотрудниках, у которых значение поля SALARY превышает 100 тысяч.

2. Используя поля **DeleteSQL, InsertSQL, ModifySQL, SelectSQL** компонента IBDataSet1:

а) извлеките из таблицы данные о сотрудниках, у которых значение поля SALARY превышает 500 тысяч;

б) удалите из таблицы запись, у которой значение поля EMP_NO равно 110;

в) добавьте в таблицу новую запись, в поле LAST_NAME внесите свою фамилию, а остальные поля заполните произвольными данными.

Контрольные вопросы

1. Для чего используются компоненты InterBase?

2. Для чего предназначен компонент IBTransaction?

3. Какие различия существуют между компонентами классов TTable и TIBTable, TQuery и TIBQuery?

4. Какие различия имеются между компонентами TIBDataSet и TIBSQL?

5. Каким образом клиентское приложение может получать уведомления о событиях в базе данных сервера Interbase? О состоянии базы данных? О выполнении сервером запросов к базе данных?

Лабораторная работа 3.10. Использование компонентов dbExpress

В лабораторной работе изучаются возможности компонентов dbExpress для доступа к базам данных в архитектуре «клиент-сервер» (на примере Interbase). Доступ к серверам баз данных на основе технологии dbExpress осуществляется с помощью специализированных драйверов, реализованных в виде динамических библиотек.

1. Создайте на рабочем диске папку с именем dbexpressdemo.

2. Проверьте, установлен ли сервер баз данных Borland Interbase. Если установлен, запустите IBConsole и проверьте наличие демонстрационной базы данных Employee.gdb (путь по умолчанию "..\Program Files\Borland\InterBase\examples\Database\EMPLOYEE.GDB").

Запишите путь к этой базе данных.

3. Запустите Borland Delphi.

4. Выберите на Палитре компонентов страницу **dbExpress** и поместите на форму компонент SQLConnection. Задайте значения свойств компонента SQLConnection1 следующим образом:

SQLConnection1.ConnectionName := IBConnection
SQLConnection1.LoginPrompt := false

Сделайте щелчок правой кнопкой мыши по компоненту SQLConnection1 и из контекстного меню выберите команду **Edit Connection Properties**. В поле **Database** списка **Connection Settings** скопируйте путь к файлу базы данных. После этого установите значение true для поля **Connected** компонента SQLConnection1.

5. Поместите на форму компонент SQLTable. Задайте значения свойств компонента SQLTable1 следующим образом:

SQLTable1.SQLConnection := SQLConnection1
SQLTable1.TableName := EMPLOYEE_PROJECT
SQLTable1.Active :=true

6. Для отображения данных из таблицы EMPLOYEE_PROJECT поместите на форму два компонента TLabel и два компонента TEdit, а также компоненты TDataSource и TDBNavigator. В качестве значений полей **Caption** компонентов Label задайте имена полей таблицы (EMP_NO и PROJ_ID, соответственно). В качестве значения поля **DataSet** компонента DataSource1 задайте *SQLTable1*. В качестве значения поля **DataSource** компонента DBNavigator1 задайте *Data-*

Source1. Поскольку в технологии dbExpress используются однонаправленные курсоры, для компонента DBNavigator1 нужно запретить команду возврата на предыдущую запись и команду перехода на последнюю запись.

7. Для того чтобы полям **Text** компонентов TEdit присвоить значения полей таблицы EMPLOYEE_PROJECT, поместите следующий код в обработчик события AfterScroll компонента SQLTable1:

```
Edit1.Text := SQLTable1.FieldByName("EMP_NO").AsInteger;
Edit2.Text := SQLTable1.FieldByName("PROJ_ID").AsString;
```

8. Откомпилируйте проект, чтобы убедиться в отсутствии ошибок, и запустите его на выполнение. Используя кнопки навигации, просмотрите содержимое таблицы EMPLOYEE_PROJECT.

9. Для редактирования данных на стороне клиента поместите на форму компонент SQLClientDataSet, а также компоненты DataSource и DBGrid. Задайте значения свойств компонента SQLClientDataSet1 следующим образом:

```
SQLClientDataSet1.DBConnection := SQLConnection1
SQLClientDataSet1.CommandType := ctQuery
SQLClientDataSet1.CommandText := "select *  from EMPLOY-
EE_PROJECT"
SQLClientDataSet1.Active := true
```

В качестве значения поля **DataSet** компонента DataSource2 задайте *SQLClientDataSet1*. В качестве значения поля **DataSource** компонента DBGrid1 задайте *DataSource2*.

Для того чтобы внесенные пользователем изменения сохранялись, в обработчик события AfterPost компонента SQLClientDataSet1 добавьте следующий код:

```
SQLClientDataSet1.ApplyUpdates(-1);
```

10. Откомпилируйте проект, чтобы убедиться в отсутствии ошибок, и запустите его на выполнение. Обратите внимание, что при попытке сохранить внесенные в компонент DBGrid1 изменения возникает исключительная ситуация.

11. Сохраните проект в созданной при выполнении п. 1 папке под именем dbexpressdemo.

Задание для самостоятельной работы

1. Добавьте необходимый код в процедуру

```
TForm1.SQLClientDataSet1AfterPost(TDataSet *DataSet)
```

для корректной обработки исключительных ситуаций.

2. Проведите отладку соединения с сервером баз данных с помощью компонента SQLMonitor. *Указание*: задайте значения свойств компонента SQLMonitor1 следующим образом:

```
SQLMonitor1.SQLConnection := SQLConnection1
```

SQLMonitor1.FileName := log.txt
SQLMonitor1.Active :=true
SQLMonitor1.AutoSave :=true
Просмотрите содержимое файла log.txt.

Контрольные вопросы

1. В чем состоят преимущества и недостатки технологии dbExpress?
2. Для чего предназначен компонент TSQLConnection?
3. Какие различия существуют между компонентами TTable и TSQLTable, TQuery и TSQLQuery?
4. Каким образом можно организовать редактирование данных при использовании технологии dbExpress?
5. Для чего предназначен компонент TSQLMonitor?

Лабораторная работа 3.11. Отображение содержимого xml-файла в виде таблицы

В работе рассматривается создание приложения, позволяющего отображать данные, хранящиеся в формате XML. В качестве источника данных используется файл biolife.xml, входящий в состав дистрибутива Borland Delphi.

1. Создайте на рабочем диске папку с именем xmldemo.
2. Запустите Borland Delphi.
3. Для работы с источником данных в формате XML нужно с помощью утилиты XML Mapper создать трансформационные файлы. Для запуска утилиты XML Mapper выполните команды **Tools — XML Mapper**.
4. В окне **XML Mapping Tools** выберите пункты меню **File — Open** или соответствующую кнопку на панели инструментов. Откройте файл biolife.xml (...\Common Files\Borland Shared\Data) и сохраните его в папке xmldemo (**File — Save — XML Document**).
5. Убедитесь, что в левой части окна утилиты XML Mapper выбрана вкладка **Document View** и в ней отображается древовидная структура документа biolife.xml. Сделайте щелчок правой кнопкой мыши на верхнем узле (DATAPACKET) документа XML и в контекстном меню последовательно выберите пункты **Select All** и **Create Datapacket from XML**.
6. Убедитесь, что в левой части окна утилиты XML Mapper выбрана вкладка **Mapping** и щелкните по кнопке **Create and Test Transformation** (см. рис. 3.7).

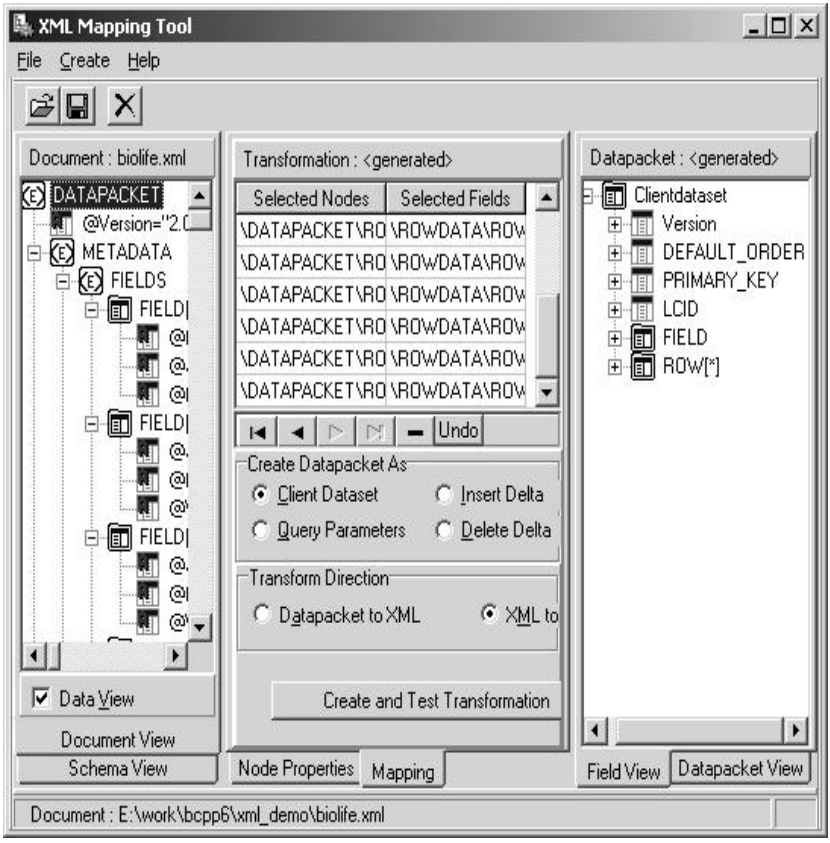

Рис. 3.7. Окно утилиты XML Mapper

7. Сохраните в созданной папке транформационный файл. Для этого выполните команды **File — Save — Transformation** и для трансформационного файла задайте имя ToBiolifeDp.xtr.

8. Сохраните в созданной папке файл пакета данных (это также файл в формате XML, который используется для обмена данными между клиентским набором данных и компонентом-провайдером). Для этого выполните команды **File—Save—Datapacket** и для файла пакета данных задайте имя packet.xml.

9. Завершите работу с утилитой XML Mapper и создайте новый проект. Поместите на форму следующие компоненты: XMLTransformProvider, ClientDataSet, DataSource (страница **Data Access** Палитры компонентов) и компонент DBGrid для отображения содержимого XML-файла (страница **Data Controls** Палитры компонентов).

10. Задайте значения свойств компонентов следующим образом:
XMLTransformProvider1.XMLDataFile := <...>\biolife.xml
XMLTransformProvider1.TransformationRead.TransformationFile := <...>\ToBiolifeDp.xtr
ClientDataSet1.ProviderName := XMLTransformProvider1
ClientDataSet1.FileName := <...>\packet.xml
DataSource1.DataSet := ClientDataSet1
DBGrid1.DataSource := DataSource1

11. Сохраните проект в созданной папке под именем xmldemo.

Затем установите свойство ClientDataSet1.Active := true. После этого в таблице в качестве заголовков столбцов отобразятся поля файла — пакета данных.

12. Для того чтобы в таблице DBGrid отобразились записи XML-файла biolife.xml, установите свойство ClientDataSet1.Active := false, а свойство ClientDataSet1.FileName := <. ..>\biolife.xml. После этого снова установите свойство ClientDataSet1.Active := true. Найдите наименования отображаемых столбцов данных в схеме документа (в левой части окна утилиты XML Mapper выберите вкладку **Schema View**, а на ней выберите вкладку **XML — Schema**).

13. Выполните задание для самостоятельной работы.

Задание для самостоятельной работы

1. С помощью соответствующих компонентов VCL отобразите содержимое файла biolife.xml в виде дерева. *Указание*: используйте компоненты TreeView на странице Win32, XMLDocument на странице **Internet** и XMLTransformClient.

Контрольные вопросы

1. Что такое XML? В чем состоят различия между XML и HTML?
2. Что такое DTD, XDR, XML Schema?
3. Для чего предназначена утилита XML Mapper?
4. Что такое трансформационный файл и для чего он используется?
5. Какие компоненты VCL используются для обработки данных в формате XML и каково их назначение?

Лабораторная работа 3.12. Отображение содержимого xml-файла в виде дерева

В лабораторной работе демонстрируется создание приложения, позволяющего отображать данные, хранящиеся в формате XML, в виде древовидной структуры. В качестве источника данных используется файл biolife.xml, входящий в состав дистрибутива Borland Delphi.

1. Создайте на рабочем диске папку с именем lab12.
2. Запустите Borland Delphi и создайте новое приложение.
3. Сохраните проект в созданной в п. 1 папке. Скопируйте в эту же папку и файл biolife.xml (...\Common Files\Borland Shared\Data).
4. Поместите на форму компонент XMLDocument (страница **Internet** Палитры компонентов), компонент TButton, компонент ImageList и компонент TreeView для отображения содержимого XML-файла (страница **Win32** Палитры компонентов).
5. Задайте значения свойств компонентов следующим образом:

```
Button1.Caption := 'Open XMLtree';
XMLDocument1.FileName := '…\biolife.xml';
```

6. В обработчик OnClick компонента **Button1** поместите следующий код:

```
function DOMNode2TreeViewNode( ADOMNode : IDOMNode; Parent-
Node : TTreeNode ):boolean;
var i: Integer;
   ThisNode : TTreeNode;
   TVNodes : TTreeNodes;
begin
 if not Assigned( ADOMNode ) then Exit;
  TVNodes := TreeView1.items;
  ThisNode := TVNodes.AddChild(ParentNode, ADOM-
Node.NodeName);
  if not Assigned( ThisNode ) then Abort;
  ThisNode.ImageIndex := 0;
  if Assigned( ADOMNode.attributes ) then
 for i := 0 to ADOMNode.attributes.length -1  do
    with TVNodes.AddChild( ThisNode, ADOM-
Node.Attributes.item[i].NodeName ) do
   begin
     ImageIndex := 1;
     SelectedIndex := ImageIndex;
   end;
   ThisNode.SelectedIndex := ThisNode.ImageIndex;
 if Assigned( ADOMNode.ChildNodes ) then
```

```
   for i := 0 to ADOMNode.ChildNodes.length -1  do
     DOMNode2TreeViewNode( ADOMNode.ChildNodes[i], ThisNode );
end;
begin   {   процедура OnClick  }
try
   XMLDocument1.LoadFromFile('biolife.xml');
   TreeView1.Items.Clear;
   try
     Screen.Cursor := crHourGlass;
     DOMNode2TreeViewNode( XMLDocument1.DOMDocument, nil );
   finally
     Screen.Cursor := crDefault;
   end;
   except  on E:EDOMParseError do
    ShowMessage( 'Parser error. Message:'#13#10 + E.Reason );
end;
end;
```

7. Обратите внимание, что функция DOMNode2TreeViewNode, используемая для построения дерева XML-документа, вызывается рекурсивно. Сохраните проект в созданной папке под именем xmlt-demo.

8. Запустите приложение и проанализируйте его работу. Примерный вид дерева XML-документа приведен на рис. 3.8.

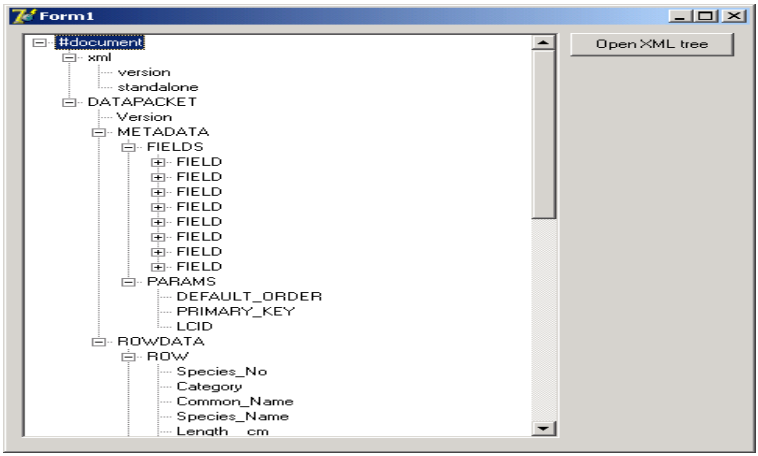

Рис. 3.8. Дерево XML-документа

9. Выполните задание для самостоятельной работы.

Задание для самостоятельной работы

1. Используя результаты лабораторных работ 3.11 и 3.12, с помощью соответствующих компонентов VCL сохраните в виде документа XML данные таблицы country.db и отобразите содержимое этого документа в виде дерева.

Контрольные вопросы

1. Что такое объектная модель документа (DOM)?
2. Как происходит разбор XML-документа?
3. Что такое правильный XML-документ? Что такое состоятельный XML-документ?
4. Для чего предназначен компонент XMLDocument?
5. Какие анализаторы XML-документов можно использовать в прикладной программе?

4. ЗАДАЧИ И УПРАЖНЕНИЯ

4.1. Арифметические операции и стандартные арифметические функции

4.1.1. Вычислить процент от числа.

4.1.2. Поменять местами значения целых переменных *x* и *y*, используя только две переменные.

4.1.3. По формуле Герона определить площадь треугольника:

$$S = \sqrt{P(P-a)(P-b)(P-c)},$$

где *a*, *b*, *c* — стороны треугольника, *P* — полупериметр.

4.1.4. Даны числа *x*, *y*, z. Вычислить *A* и *B*, если

a) $A = \dfrac{3+e^{y-1}}{1+x^2|y-\mathrm{tgz}|}$,

b) $B = 1 + |y-x| + \dfrac{(y-x)^2}{2} + \dfrac{(y-x)^3}{3}$.

4.1.5. Даны действительные числа *x* и *y*. Не пользуясь никакими другими операциями, кроме умножения, сложения и вычитания, вычислить:

$$3x^2y^2 - 2xy^2 - 7xy^2 - 4y^2 + 15xy + 2x^2 - 3x + 10y + 6.$$

Разрешается использовать не более восьми операций умножения и восьми операций сложений и вычитаний.

4.1.6. Вычислить дробную часть среднего геометрического трех заданных чисел.

4.1.7. Определить число, полученное выписыванием в обратном порядке цифр заданного трехзначного числа (см. код программы в разделе 5).

4.1.8. Ввести год и определить, к какому столетию он относится.

4.1.9. По введенной дате первого воскресного дня определить количество учебных дней в месяце.

4.1.10. Вычислить функции:

a) $Y = \lg(x^2 + 3x + 0{,}1)$;

b) $Y = \sqrt{\sin^2(x^2 + 2x + 1)}$;

c) $Y = 5\cos 3x + 6\sqrt{x^2 - 5x + 4}$;

d) $Y = \sqrt{\dfrac{x+\sqrt{x^2-1}}{2}}$;

e) $Y = \sqrt[3]{x} + \sqrt{x^2}$;

f) $Y = -4\cot 2x + \sqrt{3};$

g) $Y = x\sqrt{5 \cdot \sin\frac{\pi x}{3}}.$

4.2. Разветвление и простейшие циклы

4.2.1. Дано действительное число *x*. Вычислить *f(x)*, если

$$f(x) = \begin{cases} 0, x \le 0 \\ x^2 - x, 0 \le x \le 1 \\ x^2 - sin\pi x^2, x > 1 \end{cases}$$

4.2.2. Даны действительные числа *x*, *y*, *z*. Получить *max(x,y,z)* и *min(x,y,z)*. Рассмотреть все варианты для *x*, *y*, *z*.

4.2.3. Напечатать таблицу истинности для логических операций not A, A and B, A or B, A xor B, используя соответствующие операции:

A	B	not A	A and B	A or B	A xor B
true	**true**				
true	**false**				
false	**true**				
false	**false**				

4.2.4. Записать условный оператор, который эквивалентен оператору присваивания

x:=a or b and c

(все переменные логические) и в котором не используются логические операции. Например, оператору

x:= not a

эквивалентен оператор

if a then x:= false else x:= true.

4.2.5. Дано целое число. Необходимо определить его остаток по модулю 12 и вывести на экран название соответствующего месяца.

4.2.6. Определить ASCII-коды букв своего имени.

4.2.7. Даны действительное число a, натуральное число n. Вычислить:

a(a + 1)…(a + n – 1).

4.2.8. Даны числа из разных цифр. Напечатать все четырехзначные натуральные числа, в десятичной записи которых нет двух одинаковых цифр.

4.2.9. Используя одно условное выражение, определить, является ли введенное натуральное число *N* полным квадратом.

4.2.10. В натуральном числе *N* определить число цифр и поменять местами первую и последнюю цифры.

4.2.11. Вывести на экран таблицу кодов всех строчных и прописных букв латинского алфавита, расположив их в четыре столбца.

4.2.12. Среди *N* псевдослучайных величин, равномерно распределенных в интервале *[0, 10]*, определить количество чисел, равных заранее введенному *M (M < 10)*.

4.3. Циклические процессы

4.3.1. Вычислить:

a) $y - 1! + 2! + 3! + \cdots + n!$;

b) $A = \sum_{j=1}^{m} B_j$ и $C = \prod_{j=1}^{m} B_j$.

4.3.2. Дано действительное число *x*. Вычислить:

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!} - \frac{x^{15}}{15!} + \frac{x^{17}}{17!}$$

4.3.3. Вычислить наибольший общий делитель двух натуральных чисел *A* и *B*.

4.3.4. Определить все простые числа в диапазоне от 1 до *N*.

4.3.5. Найти корень уравнения $f(x) = 0$ на отрезке [*a*,*b*] с точностью $eps((b-a) \le eps$. На этом отрезке $f(x)$ непрерывна и имеет единственный нуль:

№ варианта	$f(x)$	*a*	*b*	*eps*
1	$x^3 - x^2 - 2$	1	2	0.01
2	$\sin x$	3	4	0.001
3	$\ln x$	0	10	0.1

4.3.6. Приближенно вычислить интеграл от функции $f(x)$ в пределах от *a* и *b* по формуле прямоугольников:

$$\int_h^a f(x)dx \approx h[f(x_1) + f(x_2) + K + f(x_n)$$

где $h = \frac{b-a}{n}$;

$$x_{ii} = a + ih - \frac{h}{2}, n > 10.$$

№ варианта	$f(x)$	*a*	*b*
1	$\ln(2+\sin x)$	0	π
2	$\ln\cos x$	0	$\pi/2$
3	$\exp\cos x$	0	2π

4.4. Массивы

4.4.1. Составить программу вычисления многочлена:

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + K + a_n x^n$$

по схеме Горнера:

$$P_n(x) = \left(K\left((a_n x + a_{n-1})x + a_{n-2}\right)\right)x + K + a_1\Big)x + a_0.$$

4.4.2. Дан одномерный массив символов S(50). Вывести на экран те элементы массива *S*, индексы которых являются:

– по формулам: степенями двойки (1, 2, 4, 8, 16, …);

– полными квадратами (1, 4, 9, 16, 25, …);

– числами Фибоначчи (1, 2, 3, 5, 8, 13, …), т.е. числами, определяемыми по формулам:

$$F_0 = F_1 = 1;$$
$$F_n = F_{n-1} + F_{n-2}, \quad n = 2, 3, \ldots$$

4.4.3. Задан числовой массив *A(n)*. Найти длину самой длинной последовательности идущих подряд элементов массива, равных нулю.

4.4.4. Дана квадратная матрица целых чисел *A* порядка *N*. Определить, является ли она симметричной относительно главной диагонали.

4.4.5. В матрице поменять местами первую и последнюю строку.

4.4.6. Найти сумму диагональных элементов матрицы порядка *N*.

4.4.7. Ввести *M* элементов массива *A*. Учесть, что $M < 15$. Найти и напечатать сумму S_p положительных и сумму S_o отрицательных элементов массива. Преобразовать и напечатать элементы массива по формуле:

$$A_i = (A_i - S_o) / (S_p - S_o).$$

4.4.8. Определить, является ли заданная целая матрица четвертого порядка магическим квадратом, т.е. такой, в которой суммы элементов во всех строках и столбцах одинаковы.

4.4.9. Упорядочить по убыванию четные элементы матрицы $A(m \times m)$, которые стоят ниже главной диагонали, и разместить их в одном векторе.

4.4.10. Найти в одномерном массиве элемент, для которого произведение суммы всех элементов, стоящих справа, на сумму всех элементов, стоящих слева, максимально.

4.4.11. Векторы *X* и *Y* упорядочены по возрастанию. Объединить элементы этих векторов так, чтобы элементы нового вектора также были упорядочены по возрастанию.

4.4.12. Переписать одномерный массив *A* в массив *B*, отбросив максимальные по модулю элементы.

4.4.13. Построить булевский вектор *B*, *i*-ая компонента которого имеет значение «истина», если в *i*-ой строке матрицы нет нулевых элементов, и значение «ложь» в противном случае, если дана матрица $C(n \times m)$.

4.4.14. В двумерном массиве определить элемент, наименее отличающийся от среднего значения элементов массива (см. код программы в разделе 5).

4.4.15. Переставить столбцы матрицы в обратном порядке, не используя дополнительный массив.

4.4.16. Умножить матрицу на вектор.

4.4.17. Умножить матрицу на матрицу.

4.4.18. Найти суммы диагональных элементов матрицы.

4.4.19. Дана действительная матрица $M \times N$. Найти сумму наибольших значений элементов ее строк.

4.4.20. В данной действительной матрице порядка N найти сумму элементов строки, в которой расположен элемент с наименьшим значением. Предполагается, что такой элемент единственный.

4.4.21. Дана целочисленная матрица порядка 4. Найти наименьшее из значений элементов столбца, который обладает наибольшей суммой модулей элементов. Если таких столбцов несколько, то взять первый из них

4.4.22. Даны целые числа $a_1, \ldots, a_{10}$ и квадратная матрица порядка *N*. Заменить нулями в матрице те элементы с четной суммой индексов, для которых имеются равные среди $a_1, \ldots, a_{10}$.

4.4.23. Дана действительная матрица $M \times N$. Найти среднее арифметическое каждого из столбцов, имеющих четные номера.

4.4.24. Дана действительная матрица $M \times N$, все элементы которой различны. В каждой строке выбирается элемент с наименьшим значением, затем среди этих чисел выбирается наибольшее.

4.4.25. Вычислить определитель квадратной матрицы произвольной размерности. Все исходные данные должны вводиться с клавиатуры.

4.4.26. Решить систему линейных уравнений методом Крамера.

4.4.27. Вычислить обратную матрицу.

4.4.28. Дана квадратная матрица $C(m \times n)$. Построить вектор *B*, *i*-ая компонента которого имеет значение «истина», если в *i*-ой строке матрицы *C* нет нулевых компонент, и значение «ложь» в противном случае.

4.5. Строки, множества, перечислимый тип.

4.5.1. Вывести буквы заданного слова в алфавитном порядке.

4.5.2. Распечатать в ряд порядковый номер каждой буквы в алфавите.

4.5.3. Составить программу, которая принимает с клавиатуры символ и выдает его на экран, преобразуя английские и русские строчные буквы в прописные, т.е. в буквы верхнего регистра.

4.5.4. Во введенном тексте первую букву *'a'* заменить ее кодом, вторую — числом, введенным с клавиатуры. Встретившиеся в тексте символы-числа преобразовать в число.

4.5.5. Если в тексте слово содержит менее шести букв, то дополнить его символами подчеркивания, а если больше шести, то отбросить лишние символы.

4.5.6. Инвертировать все слова предложения (переставить буквы слов в обратном порядке).

4.5.7. Проверить соответствие круглых и квадратных скобок в записи выражения на языке Паскаль.

4.5.8. Подсчитать количество символов между парами скобок комментариев {} и (* *). Количество комментариев неизвестно. Учесть возможность нарушения парности скобок.

4.5.9. Определить количество слов в заданном предложении. В конце предложения ставится точка. Слова могут отделяться символами «:», «–», «;», «,», « ». Определить самое длинное и самое короткое слово в предложении. Вывести на экран короткое слово заглавными буквами, а длинное — в обратном порядке.

4.5.10. Подсчитать количество гласных и согласных букв в заданном слове.

4.5.11. Получить множество, элементы которого — случайные числа из некоторого диапазона чисел. Вывести на экран все элементы этого множества.

4.5.12. Используя следующие перечислимые типы:

TYPE

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

Months = (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec);

определить сезон, на который приходится месяц.

4.6. Процедуры и функции

4.6.1. Дано действительное число *y*. Получить

$$Z = \frac{1{,}7t(y)+2t(1+y)}{6-t(y^2-1)},$$

где

$$t(x) = \frac{\sum_{k=0}^{10} x^{2k}}{\sum_{k=0}^{10}(2k)!}.$$

Для вычисления *t(x)* использовать подпрограмму-функцию.

4.6.2. Даны две квадратные вещественные матрицы порядка *N*. Используя функцию определения следа матрицы (суммы элементов главной диагонали), вывести на экран ту из них, у которой след наименьший. При составлении программы считать, что такая матрица единственная.

4.6.3. Написать процедуру умножения матриц *A* и *B*.

4.6.4. Даны три одномерных массива. Используя только одну функцию определения минимального или максимального элемента одномерного массива, найти минимальное значение из максимальных элементов заданных массивов. Рассмотреть все случаи.

4.6.5. Используя функцию определения слов-полиндромов, т.е. таких, которые одинаково читаются как слева направо, так и справа налево, подсчитать их количество в словаре из *n* слов и вывести их на экран (см. рис. 5.3).

4.6.6. Составить функцию расчета наименьшего простого числа, большего заданного *N*. Используя функцию, определить *π(n)*, т.е. количество простых чисел, меньших *n*.

4.6.7. Дано натуральное число *N*. Вычислить *N!*. Для вычисления факториала использовать рекурсивную процедуру (но не функцию).

4.6.8. Описать рекурсивную функцию *root(f,a,b,eps)*, которая методом деления отрезка пополам находит с точностью *eps* корень уравнения *f(x)=0* на отрезке [*a,b*]. При составлении программы считать, что *eps>0, a<b, f(a)·f(b)<0* и, что функция *f(x)* непрерывна и монотонна на отрезке [*a,b*].

4.6.9. Написать подпрограмму, вычисляющую для любого одномерного массива сумму элементов, кратных пяти. В программе обработать не менее двух массивов различной размерности.

4.6.10. Составьте функцию для определения максимального элемента матрицы. Используя эту функцию, определите для каждой из заданных матриц *A* и *B* значение максимального элемента и, если оно больше заданной величины *H*, извлеките квадратный корень из

значения каждого элемента первой строки матрицы. В противном случае выведите сообщение: «Значение не предельно».

4.7. Задачи на построение графиков

4.7.1. Нарисовать посередине окна круг.

4.7.2. Нарисовать по диагонали окна 5 окружностей разного цвета.

4.7.3. Нарисовать посередине окна прямоугольник и провести в нем диагонали.

4.7.4. Вывести на экран целочисленную матрицу (разрядность чисел от 1 до 3) в окне с рамкой зеленого цвета. Максимальный элемент выделить окружностью красного цвета.

4.7.5. Разработать программу, позволяющую выбирать из меню построение различных линий, окружности, круга, эллипса, сектора, прямоугольника, параллелепипеда, многоугольника. Над фигурами сделать соответствующие надписи различными шрифтами (см. код программы в разделе 5).

4.7.6. Построить графики тригонометрических функций:

– $Y = \sin x$;
– $Y = \tan x$;
– $Y = \cos(x - 1) + |x|$.

4.7.7. С учетом всех реквизитов построить график функции (см. рис. 5.4):

– $Y = |x^2 - 3|$
– $Y = \frac{1}{x}$;
– $Y = x^5$.

4.7.8. Графически проиллюстрировать типовые кривые второго порядка. Предусмотреть возможность демонстрации влияния основных параметров, входящих в канонические уравнения кривых, на вид кривых и их положение в системе координат.

4.7.9. Графически проиллюстрировать операции над множествами (объединение множеств, пересечение множеств и т.д.).

4.7.10. Разработать программу построения графика функции, выражающей плотность вероятности распределенной случайной величины.

4.7.11. Разработать программу построения графика функции, выражающей плотность вероятности нормально распределенной случайной величины.

4.8. Тип-запись или комбинированный тип

4.8.1. Данные торговой организации о заказах хранятся следующим образом:

```
TYPE
Zakaz = Record
                Komu: Stroka;
                Skolko: Real;
End;
```

Выдать на экран данные о максимальном заказе.

4.8.2. Данные о клиентах банка организованы следующим образом:

```
TYPE
Klient = Record
                Fio: Stroka;
                Summa: Integer;
End;
```

Удалить из списка всех, у кого сумма равна нулю.

4.8.3. Все клиенты банка должны быть расположены в списке по убыванию суммы вклада. Определить, не нарушена ли эта последовательность.

4.8.4. Описать типы данных следующим образом:

```
TYPE
        Stroka = String [20];
        Gitel = Record
                Fam, Gorod: Stroka;
                Dom: 1..99;
                Kvartira: 1..999;
                End;
        Spisok = Array [1..15] of Gitel;
```

Создать процедуру IRONISUD(C), которая выводит на экран фамилии двух (любых) жителей из списка C, живущих в разных городах по одинаковому адресу.

4.8.5 Используя запись «Государство», содержащую название страны, столицу, государственный язык, численность населения и площадь территории, организовать поиск и вывод на экран сведений о тех странах, численность которых превышает ранее введенное значение, а площадь территории находится в заданных пределах.

4.8.6. Создать массив, состоящий из записей:

```
TYPE
    Variant = (V1,V2);
```

```
Zap = Record
        Numer:Integer;
        Case V: Variant of
        V1: (Key1: Integer);
        V2: (Key2: Real);
End;
```

а) значения в полях Key1 и Key2 должны быть получены случайным образом;

б) массив должен содержать не более 100 записей;

в) полученный массив отсортировать по возрастанию или по убыванию значений Key1;

г) осуществить бинарный поиск целого числа;

д) найти количество чисел, попадающих в интервал от 0,5 до 1,0 в поле Key2.

4.8.7. Используя запись с вариантами, в которой хранятся значения основных параметров геометрических фигур (для треугольника — длины трех сторон; для прямоугольника — длины двух сторон, для круга — радиус; координаты *X*, *Y* их положения на плоскости), нарисовать эти фигуры, поместив в центр каждой из них значения предварительно вычисленных площадей.

4.9. Файловый тип

4.9.1. Создать типизированный файл KURS1, содержащий сведения о студентах первого курса:

```
TYPE
    Ekzamen = (MatAn, LinAlg, Programm);
    Student = Record
        Fio: Record;
            Fam, Im, Ot: String[40];
        End;
        Ocenki: Array [Ekzamen] of 2..5;
        Gruppa: 411..414;
        End;
```

Написать программу, которая заносит в текстовый файл KURS2 сведения только о тех студентах, которые успешно сдали все экзамены, и выводит на экран сведения о студентах, имеющих хотя бы одну задолженность: печатает их фамилии и инициалы, номера их групп и количество несданных экзаменов.

СОДЕРЖАНИЕ