МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БАРАНОВИЧСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНФОРМАТИКА

**Методические указания и задания
к лабораторным работам для студентов II курса
специальностей 1-40 01 02 Информационные системы
и технологии, 1-36 01 03 Технологическое оборудование
машиностроительного производства, 1-36 01 01 Технология
машиностроения, 1-53 01 01 Автоматизация технологических
процессов и производств инженерного факультета**

Часть 2

**Барановичи
БарГУ
2007**

УДК 004(075.8)
ББК 32.81я73

Составители:
О. И. Наранович, С. Г. Скобля, Т. В. Шляхтич

Рецензенты:
Д. А. Ционенко, кандидат физико-математических наук;
Т. Р. Якубович, кандидат физико-математических наук

Рекомендованы к изданию методической комиссией инженерного факультета (протокол от 01.10.2006 № 2)

Информатика [Текст] : методические указания и задания к лабораторным работам для студентов II курса специальностей 1-40 01 02 Информационные системы и технологии, 1-36 01 03 Технологическое оборудование машиностроительного производства, 1-36 01 01 Технология машиностроения, 1-53 01 01 Автоматизация технологических процессов и производств инженерного факультета дневной формы обучения : в 2 ч. / сост. О. И. Наранович, С. Г. Скобля, Т. В. Шляхтич. – Барановичи : БарГУ, 2007.– Ч. 2.– 92 с.– 120 экз.

Методические указания и задания к лабораторным работам посвящены изучению раздела «Программирование в среде Delphi» дисциплины «Информатика» и содержат семь лабораторных работ с теоретическим материалом и примерами программ.

Настоящее издание предназначено для студентов II курса инженерного факультета дневной и заочной форм обучения, а также для слушателей факультета повышения квалификации и переподготовки кадров в сфере экономики и образования.

УДК 004(075.8)
ББК 32.81я73

ЛАБОРАТОРНАЯ РАБОТА 1

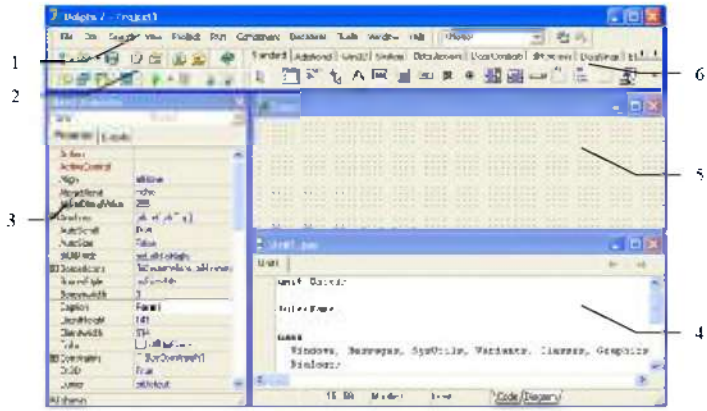
ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ. ЗНАКОМСТВО СО СРЕДОЙ РАЗРАБОТКИ DELPHI

Цели:

- познакомиться со средой разработки Delphi;
- научиться создавать простейшие приложения.

1.1 Интегрированная среда разработки Delphi

Интегрированная среда разработки (далее IDE – Integrated Development Environment) Delphi представлена несколькими одновременно раскрытыми окнами, количество, расположение, размер и вид которых может изменяться программистом в зависимости от его текущих нужд, что может значительно повысить производительность работы в среде. Основные элементы интерфейса среды представлены на рисунке 1.



1 – основное меню; *2* – панель инструментов; *3* – окно инспектора объектов; *4* – редактор кода программы; *5* – окно формы; *6* – палитра компонентов

Рисунок 1 – Основные элементы интерфейса среды

Основное меню **IDE** содержит следующие команды: **File**, **Edit**, **Search**, **View**, **Project**, **Run**, **Component**, **Database**, **Tools**, **Window**, **Help** (приложение А).

Палитра компонентов содержит множество компонентов, которые подразделяются на несколько групп, каждая из которых размещена на своей странице палитры компонентов (приложение Б).

Окно инспектора объектов (вызывается с помощью клавиши **F11**) предназначено для изменения свойств выбранных компонентов и состоит из двух страниц. Страница Properties (Свойства) предназначена для изменения необходимых свойств компонента, страница Events (События) – для определения реакции компонента или формы на то или иное событие (например, щелчок «мыши» на кнопке – событие OnClick, создание формы – OnCreate).

Окно формы представляет собой проект Windows-окна программы, на котором в процессе написания программы размещаются необходимые компоненты.

Редактор кода программы предназначен для просмотра, написания и редактирования текста программы. В системе Delphi используется язык программирования Object Pascal. При первоначальной загрузке в окне текста программы находится текст, содержащий минимальный набор операторов для нормального функционирования пустой формы в качестве Windows-окна. При помещении некоторого компонента в окно формы текст программы автоматически дополняется описанием необходимых для его работы библиотек стандартных программ (раздел **Uses**) и типов переменных (раздел **Type**).

Программа в среде Delphi составляется как описание алгоритмов, которые будут выполняться, если возникает определенное событие, связанное с формой или каким-либо из размещенных на ней компонентов. Для каждого обрабатываемого события с помощью страницы Events инспектора объектов в тексте программы организуется процедура (procedure), между ключевыми словами **begin** и **end** которой программист записывает на языке Object Pascal требуемый алгоритм.

Переключение между окном формы и окном редактора кода осуществляется с помощью клавиши **F12**.

1.2 Структура проекта Delphi

Проект программы в Delphi состоит из файла проекта (файл с расширением .dpr), одного или нескольких файлов исходного текста (с расширением .pas), файлов с описанием окон формы (с расширением .dfm) и еще нескольких вспомогательных файлов.

В *файле проекта* находится информация о модулях, составляющих данный проект. Файл проекта автоматически создается и редактируется средой Delphi и не предназначен для редактирования.

Файл исходного текста – программный модуль (Unit), предназначенный для размещения в нем программистом текстов программ на языке Pascal.

Внимание

Ни в коем случае нельзя изменять имя модуля вручную. Delphi требует, чтобы имя модуля совпадало с именем файла, в котором он хранится на диске, поэтому для того, чтобы переименовать модуль, нужно сохранить его в файле с новым именем, воспользовавшись командой меню **File-Save As**.

В разделе объявлений описываются типы, переменные, заголовки процедур и функции, которые могут быть использованы другими модулями через операторы подключения библиотек (**Uses**). В разделе реализации располагаются тела процедур и функций, описанных в разделе объявлений, а также типы переменных, процедуры и функции, которые активизируются только в пределах данного модуля. Раздел инициализации используется редко, поэтому его можно пропустить. Модуль имеет следующую структуру:

```
unit Unit 1;
interface    { Раздел объявлений
implementation    { Раздел реализации
begin    { Раздел инициализации
end.
```

При компиляции программы Delphi создаст файл с расширением .dcu, имеющий в себе результат перевода в машинные коды содержимого файлов с расширением .pas и .dfm. Компоновщик преобразует

файлы с расширением .dcu в единый загружаемый файл с расширением .exe. В файлах, имеющих расширение .~df, .~dp, .~pa, хранятся резервные копии файлов с образом формы, проекта и исходного текста соответственно.

Задание 1. Создание и сохранение проекта

1. Запустите Delphi;
2. Выберите команду **File – Save Project As** и в появившемся окне войдите в свою или создайте новую папку, введите имя сохраняемого модуля (например, MyProgram), а в следующем окне – имя сохраняемого проекта, т. е. будущей программы (например, MyPrg);
3. Закройте проект, выбрав команду **File – Close All**;
4. Создайте новый проект приложения, выбрав команду **File – New**, а затем **Application**;
5. Закройте новый проект без сохранения и выйдите из среды разработки.

Задание 2. Создание простого приложения

Создадим приложение, которое будет суммировать два числа и выводить ответ в отдельном окошке сообщения (messagebox-е).

1. При создании нового приложения по умолчанию создается форма, которая имеет название Form1. При этом в модуле в разделе описания типов **(Type)** объявляется класс формы:

```
Type
  TForm1 = class (TForm);
```

2. По условию задания приложение должно суммировать два числа, заданных пользователем. В среде Delphi для ввода текста служит однострочный редактор текста **Edit** (рис. 2), который находится на закладке Standard палитры компонентов (основные общие свойства, методы и события компонентов указаны в приложении В, а уникальные свойства – в приложении Г).

Рисунок 2 **Edit**

Для добавления его на форму дважды щелкните на данном компоненте. Элемент **Edit** появится на форме приложения. По умолчанию имя компонента будет **Edit1**. При этом Delphi сама добавляет в описание класса формы соответствующие поля:

```
Type
  TForm1 = class(TForm)
  Edit1: TEdit;
```

Таким же образом расположите на форме и второй компонент **Edit**;

3. Для того, чтобы пользователь знал, в какой из однострочных редакторов вводить значение необходимой переменной, редакторы следует подписать. Для размещения на форме надписей, которые пользователь не может редактировать, служит компонент **Label** (рис. 3), находящийся на закладке Standard.

Разместите на форме два компонента **Label**. По умолчанию они будут иметь следующие имена: **Label1**, **Label2**. В описание класса формы добавятся также строки:

Рисунок 3 – **Label**

```
Label1: TLabel;
Label2: TLabel;
```

4. На форме также необходимо разместить две кнопки: первая будет отвечать за расчет суммы введенных чисел и вывод результата, вторая – за выход из программы. В среде Delphi кнопке соответствует компонент **Button** (рис. 4), который находится на закладке Standard палитры компонентов.

Расположите на форме два компонента **Button**. По умолчанию они буду иметь имена **Button1** и **Button2** соответственно. В описание класса формы добавятся следующие строки:

Рисунок 4 **Button**

```
Button1: TLabel;
Button2: TLabel;
```

5. Расположите компоненты на форме ровно, а размер формы уменьшите так, чтобы на ней не было неиспользуемого пространства (рис. 5).

Щелкните на компоненте **Label1** для его выделения и в инспекторе объектов измените его свойство Caption (которое имеет значение **Label1**) на *Число a*. Аналогично измените свойство Caption компонента **Label2** на *Число b*.

Рисунок 5 – Расположение компонентов **Button**

Измените в инспекторе объектов значение свойства Text компонентов **Edit1** и **Edit2** на значения 10 и 5 соответственно (так задаются значения по умолчанию для различных величин, которые вводятся с помощью этих компонентов).

Измените свойства **Caption** компонентов **Button1** и **Button2** на **Вычислить** и **Выйти** соответственно, а это же свойство формы на MyPrg. Форма должна принять вид, представленный на рисунке 6.

Рисунок 6 Расположение компонентов после изменения

6. Создадим обработчик события нажатия кнопки. При нажатии на кнопку **Вычислить** программа должна производить необходимые вычисления и выводить результат в информационное окно. Для создания обработчика события нажатия этой кнопки нужно выделить компонент **Button1**, в инспекторе объектов

перейти на закладку Events и дважды щелкнуть в строке напротив названия события OnClick. В модуле программы появится заготовка процедуры обработчика события:

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

В описании класса формы добавится поле с заголовком процедуры:

```
procedure Button1Click (Sender: TObject);
```

Приведем эту процедуру к следующему виду:

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,y:integer;
begin
    a:=StrToInt(Edit1.Text);
    b:= StrToInt(Edit2.Text);
    y:=a+b;
    MessageDlg ('Функция y='+IntToStr(y), mtInformation, [mbOK], 0);
end;
```

Здесь после слова **var** перечисляются имена переменных величин и указывается их тип. Так мы используем переменные *a*, *b* и *y*, которые имеют целый тип.

Функция **StrToInt**() выполняет преобразование строки (в данном случае хранящейся в поле **Edit**) в целое число (данное преобразование возможно, если строка состоит только из цифр).

Функция **IntToStr**() выполняет обратное преобразование, т. е. преобразует целое число в строку.

Команда **MessageDlg**('Функция y='+IntToStr(y), mtInformation, [mbOK], 0) выводит окошко сообщения со строкой **Функция y=** и получившееся в результате вычислений число (значение *y*).

Аналогичным образом создайте обработчик события нажатия кнопки **Button2** (**Выйти**). В тексте процедуры введите команду **Application.Terminate**, выполняющую завершение работы приложения. Процедура должна иметь следующий вид:

```
procedure TForm1.Button2Click (Sender: TObject);
begin
    Application.Terminate;
end;
```

Полный код модуля должен иметь вид:

```
unit Myproject;
interface;
uses
```

```
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls;
Type
    TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
    Button2: TButton;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;
var
    Form1: TForm1;
implementation
    {$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
    var a,b,y:integer;
begin
    a:=StrToInt(Edit1.Text);
    b:= StrToInt(Edit2.Text);
    y:=a+b;
    MessageDlg('Функция y='+IntToStr(y), mtInformation, [mbOK], 0);
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    Application.Terminate;
end;
end.
```

1.3 Запуск программы на выполнение

Запустить программу можно, выбрав команду **Run** в меню **Run** или нажав клавишу **F9**. При этом происходит компиляция и, если нет ошибок, компоновка программы и создание единого загружаемого файла с расширением .exe. На экране при этом появляется главная форма программы.

Поэкспериментируйте с программой, обратите внимание на сообщения, выдаваемые ею.

Задание 3. Задание для самостоятельного выполнения

Создайте приложение, вычисляющее значение функции, приведенной в таблице, в соответствии с предложенным преподавателем вариантом.

Таблица – Варианты индивидуальных заданий

Вариант	Функция	Значения параметров
1	$f = \frac{ax^2 + \cos x^3}{2ab - \ln a} + \sqrt{2{,}5bx}$	$a = 2;\quad b = 3;\quad x = 4$
2	$f = \frac{x}{3b^2 - tg^3 x}\lvert x\rvert - 2$	$b = 4;\quad x = -20$
3	$f = 11ab + \sin^3 x \frac{2ax}{(b-x)^2}$	$a = 5;\quad b = 6;\quad x = 2$
4	$f = \frac{e^x}{x^3 - \cos^2 x}\sqrt{9x^2} - 7$	$x = 10$
5	$f = \frac{x^2 + 6\cos x^3}{2 - \ln 3a} + \sqrt[3]{x^2}$	$a = 5;\quad x = -4{,}5$
6	$f = \frac{d + 2c^{-3}}{12\cos x^{-2} + tg(dx)}\ln x - \cos^2 x$	$c = 2;\quad d = -2;\quad x = 32$
7	$f = \frac{y^2 + x^3}{\sin^2 y - \cos x^3}\sqrt{3{,}5ayx}$	$a = 20;\quad x = 30;\quad y = 40$
8	$f = \frac{1}{5x - tg^2 z} + \frac{\sqrt{5ax}}{z^{-\frac{4}{3}}}$	$a = 2;\quad z = 3;\quad x = 5$
9	$f = \frac{\frac{1}{x^2+1} + \sqrt{2{,}5bx}}{2ax - b^{\frac{2}{3}}}$	$a = 2{,}4;\quad b = 3;\quad x = 7{,}5$
10	$f = 2\sqrt{\frac{2xy}{(x+y)^2} + \frac{1}{\sqrt{x^3}}} + tgx^2$	$x = 2{,}4;\quad y = 7{,}1$
11	$f = \sin^2 x\left(\cos y^3 ctg\frac{x}{y}\right)^{-\frac{3}{2}}$	$x = 1{,}2;\quad y = 2$

Окончание табл.

Вариант	Функция	Значения параметров
12	$f = 3\sqrt[3]{x^2} + \lvert 2x \rvert \dfrac{5x^4 - 2ab}{tg^2 x^{-2}}$	$a = 3{,}2;\quad b = 5;\quad x = 2{,}3$

Задание 4. Создание отчета

В качестве отчета представьте преподавателю на дискете два работающих приложения (как результаты выполнения заданий 2 и 3) и отчет, оформленный в MS Word, содержащий тему и цель работы, формулировки заданий, формы приложений, тексты модулей, результаты тестирования приложений, выводы по работе.

Контрольные вопросы

1. Опишите основные элементы интерфейса среды разработки Delphi.
2. Каким образом создается, сохраняется, открывается и компилируется проект нового приложения?
3. С помощью каких функций осуществляется преобразование целых чисел в строки и наоборот?
4. Как формируется интерфейс программы?
5. Как создать обработчик нажатия кнопки?

ЛАБОРАТОРНАЯ РАБОТА 2

ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

Цели:

– научиться пользоваться компонентами для организации переключений (**CheckBox**, **RadioGroup**);

– написать и отладить программу разветвляющегося алгоритма.

2.1 Операторы if и case языка Pascal

Для программирования разветвляющихся алгоритмов в языке Pascal используются специальные переменные типа **boolean**, которые могут принимать только два значения – **true** и **false** (да и нет), а также операторы **if** и **case**.

Оператор if

Условный оператор **if** изменяет естественный (последовательный) порядок выполнения операторов программы и имеет следующий общий вид:

if < условие > **then** < оператор1 >
else < оператор2 >;

Условие – это выражение булевского типа, оно может быть простым или сложным. Сложные условия образуются с помощью логических операций и операций отношения. Операторы могут быть и составными.

Внимание

Перед словом **else** точка с запятой не ставится.

Логика работы оператора такова: выполнить оператор1, если условие истинно, и оператор2, если условие ложно.

Например:

```
var bl: Boolean;
    x,y,u: integer;
begin
    bl:=x>y;
    if bl then u:=x-y
    else u:=x-y;
end.
```

Оператор case

Оператор ветвления **case** является обобщением оператора **if** и позволяет сделать выбор из произвольного числа имеющихся вариантов. Он состоит из выражения, называемого *селектором*, и альтернативных операторов, каждому из которых предшествует свой *список допустимых значений селектора*:

```
case < селектор > of
     < список значений селектора1 > : < оператор1 >;
     < список значений селектора2 > : < оператор2 >;
     ...
     < список значений селектораN > : < операторN >;
    else < оператор N+1 >
end;
```

Селектор должен иметь перечисляемый тип данных. Вещественные и строковые типы в качестве селектора запрещены.

Список значения селектора может состоять из произвольного количества значений, отделенных друг от друга запятыми. Все значения селектора должны быть уникальными, иначе компилятор сообщит об ошибке.

Например:

```
Var In: integer;
begin
    case In of
        0: u:=x+y;
        1: u:=x-y;
        2: u:=x y;
        else: u:=0
    end;
end.
```

В данном примере в зависимости от значения In вычисляется значение переменной u. Если In = 0, то u:=x+y, если In=1, то u:=x-y, если In=2, то u:=xoy и, наконец, u=0 при любых значениях In, отличных от 0,1 и 2.

2.2 Кнопки-переключатели и многострочный редактор текста в Delphi

При создании программ в Delphi для организации разветвлений часто используются компоненты в виде кнопок-переключателей. Состояние такой кнопки (включено-выключено) визуально отражается на форме. На форме, изображенной на приведенном ниже рисунке, представлены кнопки-переключатели двух типов: **TCheckBox** и **TRadioGroup**.

Рисунок 1 – **CheckBox**

Компонент **CheckBox** (рис. 1) организует кнопку независимого переключателя, с помощью которой пользователь может указать свое решение (типа да/нет). Переключатель имеет два состояния: *включен* или *выключен*. Текущее состояние определяется значением свойства **Checked**. Если оно равно **True**, то переключатель включен, иначе – выключен. Если переключатель имеет три состояния (свойство **AllowGrayed** равно **True**), то вместо свойства **Checked** используется свойство **State**.

Рисунок 2 – **RadioGroup**

Компонент **RadioGroup** (рис. 2) организует группу кнопок – зависимых переключателей. При нажатии одной из кнопок группы все остальные кнопки отключаются. Номер активного зависимого переключателя хранится в значении свойства **ItemIndex**. Нумерация кнопок начинается с нуля. Их количество и подписи к ним определяются свойством **Items**. Расположение зависимых переключателей, которые отображает данный компонент, подбирается автоматически с учетом заданного в свойстве **Columns** количества колонок.

Рисунок 3 – **Memo**

Для вывода результатов работы программы часто используется текстовое окно, которое представлено компонентом **Memo** (многострочный редактор текста) (рис. 3). Данный компонент хранит не одну строку текста, а множество строк, доступ к которым обеспечивает свойство **Lines**, представляющее собой объект класса **TStrings**. С помощью свойства **Lines** строки можно добавлять, вставлять, удалять и т. д. Ввод

текста в процессе разработки программы осуществляется в редакторе строк, который вызывается щелчком мыши на многоточии в поле значения свойства **Lines**.

2.3 Пример разработки программы

Постановка задачи. Ввести три числа – x, y, z. Вычислить по усмотрению u = sin(x), или u = cos(x), или u = tg(x). Найти по желанию максимальное из трех чисел: max(u,y,z), или max(|u|,|y|,|z|).

Создадим форму, представленную на рисунке 4.

Изменим свойства компонентов в соответствии с таблицей 1.

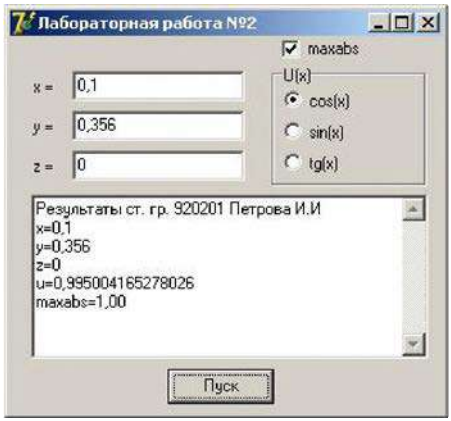


Рисунок 4 Форма приложения

Таблица 1 – Свойства компонентов и их значения

Компонент	Свойство	Значение	Новое значение
Форма Form1	Name	Form1	frmMain
Кнопка Button1	Name	Button1	btnStart
Многострочный редактор Memo1	Name	Memo1	mmResult
CheckBox CheckBox1	Name	CheckBox1	chbMaxABS
CheckBox CheckBox1	Caption	CheckBox1	maxabs
RadioGroup RadioGroup1	Name	RadioGroup1	rgUx
RadioGroup RadioGroup1	Caption	RadioGroup1	U(x)

Окончание табл. 1.

Компонент	Свойство	Значение	Новое значение
RadioGroup RadioGroup1	Items	< пусто >	cos(x) sin(x) tg(x)

Для того, чтобы изменить свойство **Items** компонента **RadioGroup**, необходимо дважды щелкнуть в строке рядом с названием свойства **Items** мышью. Появится строчный редактор списка заголовков кнопок. Наберем три строки с именами: в первой строке – cos(x), во второй – sin(x), в третьей – tg(x), нажмем **ОК**.

После этого на форме внутри окаймления появится три кнопки-переключателя с введенными надписями.

2.4 Создание процедур-обработчиков событий

1. Создайте процедуру-обработчик создания формы. Для этого выделите форму, перейдите в **Инспектор Объектов**, выберите закладку Events. Найдите событие OnCreate, нажмите два раза мышкой по его правой части. В тексте модуля появится заготовка процедуры-обработчика события. Приведите ее к следующему виду:

```
procedure TfrmMain.FormCreate(Sender: TObject);
begin
    edX.Text := '0,1';
    edY.Text := '0,356';
    edZ.Text := '0';
    mmResult.Clear;
    mmResult.Lines.Add('Результаты ст. гр. 920201 Петрова И.И');
end;
```

2. При нажатии на кнопку программа должна производить необходимые вычисления, для этого создайте процедуру-обработчик нажатия на кнопку (см. лабораторная работа 1). Измените текст процедуры следующим образом:

```
procedure TfrmMain.btnStartClick(Sender: TObject);
var x,y,z,u,ma: extended;
begin  // Ввод исходных данных и их вывод в окно Memo1
    x:= StrToFloat(edX.Text);
    mmResult.Lines.Add('x='+edX.Text);
    y:= StrToFloat(edY.Text);
    mmResult.Lines.Add('y='+edY.Text);
```

```
        z:= StrToFloat(edZ.Text);
        mmResult.Lines.Add('z='+edZ.Text);
    case rgUx.ItemIndex of  // Проверка номера нажатой кнопки и выбор соответствующей ей функции
        0: u:=cos(x);
        1: u:=sin(x);
        2: u:=sin(x)/cos(x);
    end;
        mmResult.Lines.Add('u='+FloatToStr(u));
    if chbMaxABS.Checked then begin  // Проверка состояния кнопки CheckBox1
        u:=abs(u);
        y:=abs(y);
        z:=abs(z);
    end;
    if u>y then ma:=u else ma:=y;  // Нахождение максимального из трех чисел
    if z>ma then ma:=z;
    if chbMaxABS.Checked then
        mmResult.Lines.Add('maxabs='+FloatToStrF(ma,ffFixed,8,2))
    else
        mmResult.Lines.Add('max='+FloatToStrF(ma,ffGeneral,8,2))
end;
```

Запустите программу и проверьте ее работоспособность.

Задание 1. Задание для самостоятельного выполнения

Разработайте приложение для вычисления функции, приведенной в таблице 2 в соответствии с предложенным преподавателем вариантом. В качестве f(x) использовать по выбору: $\sin(x)$, x^2, e^x.

Таблица 2 – Варианты индивидуальных заданий

Вариант	Задание	Вариант	Задание
1	$a=\begin{cases}(f(x)+y)^2-\sqrt{f(x)y}, xy>0\\(f(x)+y)^2+\sqrt{\lvert f(x)y\rvert}, xy<0\\(f(x)+y)^2+1, xy=0\end{cases}$	8	$j=\begin{cases}\sin(5f(x)+3m\lvert f(x)\rvert), -1<m<x\\\cos(3f(x)+5m\lvert f(x)\rvert), x>m\\(f(x)+m)^2, x=m\end{cases}$
2	$b=\begin{cases}\ln(f(x))+(f(x)^2+y)^3, x/y>0\\\ln\lvert f(x)/y\rvert+(f(x)+y)^3, x/y<0\\(f(x)^2+y)^3, x=0\\0, y=0\end{cases}$	9	$l=\begin{cases}2f(x)^3+3p^2, x>\lvert p\rvert\\\lvert f(x)-p\rvert, 3<x<\lvert p\rvert\\(f(x)-p)^2, x=\lvert p\rvert\end{cases}$

Окончание табл. 2.

Вариант	Задание	Вариант	Задание
3	$c=\begin{cases}f(x)^2+y^2+\sin(y), x-y=0\\(f(x)-y)^2+\cos(y), x-y>0\\(y-f(x))^2+tg(y), x-y<0\end{cases}$	10	$k=\begin{cases}\ln(\lvert f(x)\rvert+\lvert q\rvert), \lvert xq\rvert>10\\e^{f(x)+q}, \lvert xq\rvert<10\\f(x)+q, \lvert xq\rvert=10\end{cases}$
4	$d=\begin{cases}(f(x)-y)^3+arctg(f(x)), x>y\\(y-f(x))^3+arctg(f(x)), y>x\\(y+f(x))^3+0.5, y=x\end{cases}$	11	$m=\frac{\max(f(x), y, z)}{\min(f(x), y)}+5$
5	$e=\begin{cases}i\sqrt{f(x)}, i-нечетное, x>0\\i/2\sqrt{\lvert f(x)\rvert}, i-четное, x<0\\\sqrt{\lvert if(x)\rvert}, иначе\end{cases}$	12	$n=\frac{\min(f(x)+y, y-z)}{\max(f(x), y)}$
6	$g=\begin{cases}e^{f(x)-\lvert b\rvert}, 0.5<xb<10\\\sqrt{\lvert f(x)+b\rvert}, 0.1<xb<0.5\\2f(x)^2, иначе\end{cases}$	13	$p=\frac{\lvert\min(f(x), y)-\max(y, z)\rvert}{2}$
7	$s=\begin{cases}e^{f(x)}, 1<xb<10\\\sqrt{\lvert f(x)+4b\rvert}, 12<xb<40\\bf(x)^2, иначе\end{cases}$	14	$q=\frac{\max(f(x)+y+z, xyz)}{\min(f(x)+y+z, xyz)}$

Контрольные вопросы

1. Какие операторы используются для программирования разветвляющихся алгоритмов?
2. Назовите основные свойства объектов классов **TCheckBox**, **TRadioGroup**. Объясните их назначение.
3. Какие события обрабатываются в проекте? Объясните их назначение.
4. Объясните действие переменной **CheckBox1.Checked**.
5. Объясните действие переменной **RadioGroup1.ItemIndex**.
6. С помощью каких методов осуществляется добавление строк в поле **Memo** и его очистка?

ЛАБОРАТОРНАЯ РАБОТА 3

ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Цели:

- изучить простейшие средства отладки в среде Delphi;
- научиться программировать циклические алгоритмы.

3.1 Операторы повтора языка Pascal

Под ц и к л о м понимается многократное выполнение одних и тех же операторов при различных значениях промежуточных данных. Число повторений может быть задано в явной или неявной форме.

Для организации повторений в языке Delphi предусмотрены три различных оператора цикла:

1) *оператор цикла с постусловием*. Используется в тех случаях, когда тело цикла должно быть выполнено перед тем, как произойдет проверка условия завершения цикла, и имеет следующий вид:

```
repeat
    <оператор1>;
  ...
    <операторN>
until <условие завершения цикла>;
```

Тело цикла выполняется до тех пор, пока условие завершения цикла (выражение булевского типа) не станет истинным;

2) *оператор цикла с предусловием*. Является альтернативой оператору **repeat** и содержит условие выполнения цикла, а не условие завершения:

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

Перед каждым выполнением тела цикла происходит проверка условия. Если оно истинно, цикл выполняется и условие вычисляется заново, если ложно, происходит выход из цикла.

Внимание

Если тело цикла с предусловием содержит несколько операторов, то они должны быть заключены в операторные скобки **begin** <операторы> **end**.

Таким образом, оператор цикла с предусловием может не выполниться ни разу, а оператор цикла с постусловием всегда выполняется хотя бы один раз;

3) *оператор цикла с заданным числом повторений.* Используется в том случае, если заранее известно количество повторений цикла, и имеет следующий вид:

for <параметр цикла>:=<значение1> **to** <значение2> **do** <оператор>;

где <параметр цикла> – переменная любого порядкового типа, кроме вещественного;

<значение1> и <значение2> – выражения, определяющие соответственно начальное и конечное значения параметра цикла;

<оператор> – тело цикла.

Оператор **for** обеспечивает выполнение тела цикла до тех пор, пока не будут использованы все значения параметра цикла от начального до конечного. После каждого повтора значение параметра цикла увеличивается на единицу.

Существует модификация данного оператора, используемая в случае убывания параметра цикла с шагом 1 и имеющая вид:

for <параметр цикла>:=<значение1> **downto** <значение2> **do** <оператор>;

3.2 Средства отладки программ в Delphi

Практически каждая созданная программа содержит ошибки:

1) *первого уровня* (ошибки компиляции) связаны с неправильной записью операторов (орфографические, синтаксические), при обнаружении которых компилятор Delphi останавливается напротив первого операто-

ра, в котором обнаружена ошибка. В нижней части экрана появляется текстовое окно, содержащее сведения обо всех ошибках, найденных в проекте. Каждая строка этого окна содержит имя файла, в котором найдена ошибка, номер строки с ошибкой и ее характер. Для быстрого перехода к интересующей ошибке необходимо дважды щелкнуть на строке с ее описанием. Для получения более полной информации о характере ошибки необходимо обратиться к справочной системе нажатием клавиши **F1**. Следует уделить внимание тому, что одна ошибка может повлечь за собой другие, которые исчезнут при ее исправлении, поэтому необходимо исправлять ошибки последовательно, сверху вниз, и после исправления каждой ошибки компилировать программу снова;

2) *второго уровня* (ошибки выполнения) связаны с ошибками выбранного алгоритма решения или неправильной программной реализацией алгоритма. Указанные ошибки проявляются в том случае, если результат расчета оказывается неверным либо происходит переполнение, деление на ноль и др., поэтому перед использованием отлаженной программы ее необходимо протестировать, т. е. выполнить расчеты при таких комбинациях исходных данных, для которых известен результат. Если тестовые расчеты указывают на ошибку, то для ее поиска следует использовать встроенные средства отладки среды Delphi.

В простейшем случае для локализации места ошибки рекомендуется поступать следующим образом: в окне редактирования текста установить курсор в строке перед подозрительным участком и нажать клавишу **F4** (выполнение до курсора). Выполнение программы будет остановлено на строке, содержащей курсор. Теперь можно просмотреть значения интересующих переменных. Для этого необходимо поместить на нужную переменную курсор (на экране будет высвечено ее значение) либо нажать **Ctrl** + **F7** и в появившемся диалоговом окне указать имя интересующей переменной (с помощью данного окна можно также изменить значение переменной во время выполнения программы). Нажимая клавишу **F7** (пошаговое выполнение), можно построчно выполнять программу, контролируя изменение тех или иных переменных и правильность вычислений. Если курсор находится внутри цикла, то после нажатия **F4** расчет останавливается после одного выполнения тела цикла. Для продолжения расчетов следует выбрать команду **Run** в меню **Run**.

3.3 Пример разработки программы

Предположим, что нужно разработать программу, которая выводит таблицу значений функции

$$S(x)=\sum_{k=0}^{N}(-1)^k\frac{x^k}{k!}.$$

Причем значение аргумента x изменяется в интервале от $x1$ до $x2$ с шагом h.

Разместим на форме четыре компонента **Label**, четыре компонента **Edit**, текстовое поле **Memo** и одну кнопку. Примерный вид формы представлен на рисунке 1.

Создание процедур-обработчиков событий

1. Создайте процедуру-обработчик создания формы и приведите ее текст к следующему виду:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Memo1.Clear;
    Memo1.Lines.Add('Результаты ст.гр.555 Иванова Сидора Петровича');
end;
```

2. Создайте процедуру-обработчик нажатия кнопки:

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

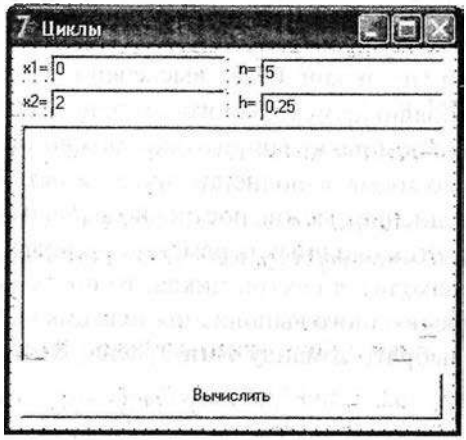


Рисунок 1 – Форма приложения

```
var x1,x2,x,h,a,s:extended;
    N,k,c:integer;
begin
    x1:=StrToFloat(Edit4.Text);
    Memo1.Lines.Add(' x1='+Edit1.Text);
    x2:=StrToFloat(Edit2.Text);
    Memo1.Lines.Add(' x2='+Edit2.Text);
    N:=StrToInt(Edit3.Text);
    Memo1.Lines.Add(' n='+Edit3.Text);
    h:=StrToFloat(Edit4.Text);
    Memo1.Lines.Add(' h='+Edit4.Text);
    c:=-1; x:=x1;
    repeat
        a:=1; S:=1;
        for k:=1 to N do begin
            a:=c*a*x/k;
            s:=s+a;
        end;
            Memo1.Lines.Add('при x='+FloatToStrF(x,ffFixed,6,2)+ '   сумма =' +
            FloatToStrF(s,ffFixed,6,2));
            x:=x+h;
    until x>x2;
end;
```

Результат выполнения программы представлен на рисунке 2.

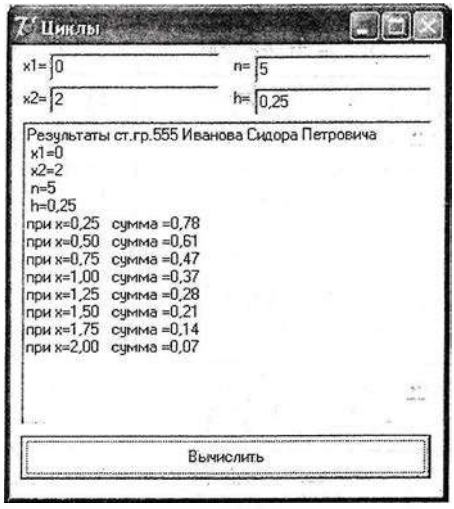


Рисунок 2 – Результат работы программы

Задание 1. Задание для самостоятельного выполнения

Выведите на экран таблицу значений функции $Y(x)$ и ее разложения в ряд $S(x)$ для x, изменяющихся от x_n до x_k с шагом $h=(x_k-x_n)/n$. Близость значений $S(x)$ и $Y(x)$ во всем диапазоне значений x указывает на правильность вычисления $S(x)$ и $Y(x)$.

Таблица – Значения функции $Y(x)$ и ее разложения в ряд $S(x)$ для x

№	x_n	x_k	$S(x)$	n	$Y(x)$
1	0,1	1	$x-\frac{x^3}{3!}+\ldots+(-1)^n\frac{x^{2n+1}}{(2n+1)!}$	16	$\sin x$
2	0,1	1	$1+\frac{x^2}{2!}+\ldots+\frac{x^{2n}}{(2n)!}$	10	$\frac{e^x+e^{-x}}{2}$
3	0,1	1	$1+\frac{\cos\frac{\pi}{4}}{1!}x+\ldots+\frac{\cos n\frac{\pi}{4}}{n!}x^n$	12	$e^{x\cos\frac{\pi}{4}}\cos(x\sin\frac{\pi}{4})$
4	0,1	1	$1-\frac{x^2}{2!}+\ldots+(-1)^n\frac{x^{2n}}{(2n)!}$	8	$\cos x$
5	0,1	1	$1+3x^2+\ldots+\frac{2n+1}{n!}x^{2n}$	14	$(1+2x^2)e^{x^2}$
6	0,1	1	$x+\frac{x^3}{3!}+\ldots+\frac{x^{2n+1}}{(2n+1)!}$	8	$\frac{e^x-e^{-x}}{2}$
7	0,1	1	$\frac{x^3}{3}-\frac{x^5}{15}+\ldots+(-1)^{n+1}\frac{x^{2n+1}}{4n^2-1}$	12	$\frac{1+x^2}{2}\operatorname{arctg}x-\frac{x}{2}$
8	0,1	1	$1+\frac{2x}{1!}+\ldots+\frac{(2x)^n}{n!}$	10	e^{2x}
9	0,1	1	$1+2\frac{x}{2}+\ldots+\frac{n^2+1}{n!}\left(\frac{x}{2}\right)^n$	14	$\left(\frac{x^2}{4}+\frac{x}{2}+1\right)e^{\frac{x}{2}}$
10	0,1	0,5	$x-\frac{x^3}{3}+\ldots+(-1)^n\frac{x^{2n+1}}{2n+1}$	15	$\operatorname{arctg}x$
11	0,1	1	$1-\frac{3}{2}x^2+..+(-1)^n\frac{2n^2+1}{(2n)!}x^{2n}$	10	$\left(1-\frac{x^2}{2}\right)\cos x-\frac{x}{2}\sin x$

Окончание табл.

№	x_n	x_k	$S(x)$	n	$Y(x)$
12	0,1	1	$-\frac{(2x)^2}{2}+\frac{(2x)^4}{24}-...+(-1)^n\frac{(2x)^{2n}}{(2n)!}$	8	$2(\cos^2 x-1)$
13	–2	– 0,1	$-(1+x)^2+\frac{(1+x)^4}{2}+...+(-1)^n\frac{(1+x)^{2n}}{n}$	16	$\ln\frac{1}{2+2x+x^2}$
14	0,2	0,8	$\frac{x}{3!}+\frac{4x^2}{5!}+...+\frac{n^2}{(2n+1)!}x^n$	12	$\frac{1}{4}\left(\frac{x+1}{\sqrt{x}}sh\sqrt{x}-ch\sqrt{x}\right)$

Контрольные вопросы

1. Какие операторы используются для программирования циклических алгоритмов?

2. Объясните принципы работы вложенных циклов.

3. Можно ли реализовать циклический алгоритм без использования операторов цикла? Ответ обоснуйте.

4. В чем различия операторов цикла с предусловием и оператора цикла с постусловием?

ЛАБОРАТОРНАЯ РАБОТА 4

ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ МАССИВОВ

Цели:

– изучить свойства и методы компонента **TStringGrid** и научиться использовать его для хранения табличных данных;

– научиться создавать приложения, обрабатывающие массивы.

4.1 Массивы в Delphi

Массив – это структурированный тип данных, состоящий из фиксированного числа элементов одного и того же типа. Для описания массива предназначено словосочетание **array of**. После слова **array** в квадратных скобках записываются границы массива, а после слова **of** – тип элементов массива:

array [границы массива] **of** <тип элемента массива>.

Тип массива или сам массив определяются соответственно в разделе описания типов **(type)** или переменных **(var)** следующим образом:

```
const N=20;
type TVector=array[1..N] of real;
var a: TVector;
    Ss: array[1..10] of integer;
    Y: array[1..5,1..10] of char;
```

Массивы, в целом, участвуют только в операциях присваивания. При этом все элементы одного массива копируются в другой.

Например:

```
var A, B : array[1..10] of integer;
begin
```

```
    A := B;
end;
```

Элементы массивов могут использоваться в выражениях так же, как и обычные переменные. Для обращения к некоторому элементу массива нужно указать имя массива и в квадратных скобках индексы элемента.

Например:

```
F:=2*a[3]+a[Ss[I]+1]*3;
A[n]:=1+sqrt(abs(a[n-1]));
```

4.2 Компонент StringGrid

При работе с массивами ввод и вывод информации на экран удобно организовывать в виде таблиц. Компонент **StringGrid** (рис. 1) предназначен для отображения информации в виде двумерной таблицы, каждая ячейка которой представляет собой окно однострочного редактора (аналогично **TEdit**). Строки хранятся в двумерном свойстве-массиве **Cells** этого компонента. Доступ к информации осуществляется с помощью свойства **Cells**[ACol, ARow:Integer]:string, где ACol, ARow – индексы (номер столбца и строки соответственно) ячейки компонента.

Рисунок 1 **StringGrid**

4.3 Пример разработки программы

Постановка задачи. Создать программу для определения вектора $\vec{Y} = A * \vec{B}$,
где A – квадратная матрица размерностью NxN,
$\vec{Y}, \vec{B}$ – одномерные массивы размерностью N.

Элементы вектора $\vec{Y}$ определяются по формуле

$$Y_i = \sum_{j=1}^{N} A_{ij} \cdot B_i .$$

Значения N вводить в компонент класса **TEdit**, A и B – в компоненты класса **TStringGrid**. Результат после нажатия кнопки класса **TButton** вывести в компонент класса **TStringGrid**.

Примерный вид формы представлен на рисунке 2 (см. с. 28).

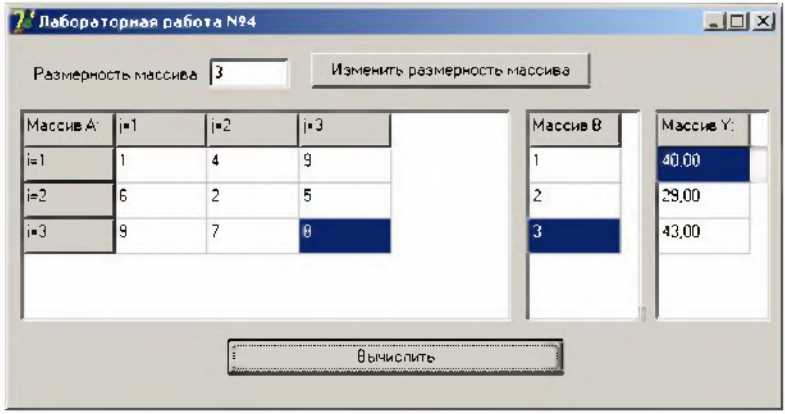


Рисунок 2 – Форма приложения

Поставьте на форму три компонента **TStringGrid** для:

- введения матрицы *A*;
- введения вектора *B*;
- вывода вектора *Y*.

В окне **Инспектора Объектов** задайте для свойств компонентов значения в соответсвии с таблицей 1.

Таблица 1 – Свойства компонентов и их значения

Компонент	Свойство	Значение	Новое значение
Form Form1	Name	Form1	frmMain
StringGrid StringGrid1	Name	StringGrid1	strgrA
StringGrid StringGrid1	ColCount		2
StringGrid StringGrid1	RowCount		2
StringGrid StringGrid1	FixedCols		1
StringGrid StringGrid1	FixedRows		1
StringGrid StringGrid1	+Option - goEditing		**true**
StringGrid StringGrid2	Name	StringGrid2	strgrB
StringGrid StringGrid2	ColCount		1
StringGrid StringGrid2	RowCount		2
StringGrid StringGrid2	FixedCols		0
StringGrid StringGrid2	FixedRows		1
StringGrid StringGrid2	+Option - goEditing		**true**

Окончание табл.

Компонент	Свойство	Значение	Новое значение
StringGrid StringGrid3	Name	StringGrid3	strgrY
StringGrid StringGrid3	ColCount		1
StringGrid StringGrid3	RowCount		2
StringGrid StringGrid3	FixedCols		0
StringGrid StringGrid3	FixedRows		1
StringGrid StringGrid3	+Option /goEditing		**false**

По умолчанию в компонент **StringGrid** запрещен ввод информации с клавиатуры, поэтому необходимо для свойства **Options goEditing** компонентов **strgrA** и **strgrB** задать значение **true**.

Также на форме разместим две кнопки **TButton**: изменить размерность массива **(btnChangeN)**, вычислить **(btnCompute)**.

Описание переменных, используемых при написании программы

В программе будут использоваться три массива: **A**, **B**, **Y**. Массив **A** – двумерный, массивы **B** и **Y** – одномерные. Для этого создадим следующие типы:

```
const
  NMax=10;
Type
  Mas1=array[1..NMax] of extended; // Объявление типа одномерного массива размерностью NMax
  Mas2=array[1..NMax,1..NMax] of extended; // Объявление типа двумерного массива размерностью NMax
```

Для определения размерности массивов будет использоваться переменная *N*.

Для того, чтобы переменные были доступны из любого места модуля, сделаем их глобальными. С этой целью в секции интерфейсных объявлений в разделе описаний переменных добавим описание этих переменных:

```
var
  A: Mas2; // Объявление двумерного массива
  B,Y: Mas1; // Объявление одномерных массивов
  N,i,j: integer;
```

Создание процедур-обработчиков событий

1. Создайте процедуру-обработчик создания формы. В данном месте программы задаем число строк и столбцов таблиц исходя из установленной размерности массива. Заполняем верхние и левые ячейки компонентов класса **TStringGrid**:

```
procedure TfrmMain.FormCreate (Sender: TObject);
begin
    N:=3; // размерность массива
    edN.Text:= FloatToStr(N);
    {Задание числа строк и столбцов в таблицах}
        strgrA.ColCount:=N+1;
        strgrA.RowCount:=N+1;
        strgrB.RowCount:=N+1;
        strgrY.RowCount:=N+1;
    {Ввод в левую верхнюю ячейку таблицы названия массива}
        strgrA.Cells[0,0]:='Массив A:';
        strgrB.Cells[0,0]:='Массив B:';
        strgrY.Cells[0,0]:='Массив Y:';
    {Заполнение верхнего и левого столбцов поясняющими подписями}
        for i:=1 to N do begin
            strgrA.Cells[0,i]:='i='+IntToStr(i);
            strgrA.Cells[i,0]:='j='+IntToStr(i);
        end;
end;
```

2. Создайте процедуру-обработчик нажатия кнопки **btnChangeN**. Текст процедуры приведите к следующему виду:

```
procedure TfrmMain.btnChangeNClick (Sender: TObject);
begin
    N:=StrToInt(edN.Text);
    {Задание числа строк и столбцов в таблицах}
        strgrA.ColCount:=N+1;
        strgrA.RowCount:=N+1;
        strgrB.RowCount:=N+1;
        strgrY.RowCount:=N+1;
    {Заполнение верхнего и левого столбцов поясняющими подписями}
        for i:=1 to N do begin
            strgrA.Cells[0,i]:='i='+IntToStr(i);
            strgrA.Cells[i,0]:='j='+IntToStr(i);
        end;
end;
```

3. Создайте процедуру-обработчик нажатия кнопки **btnCompute**.

```
procedure TfrmMain.btnComputeClick (Sender: TObject);
var s: extended;
begin
    {Заполнение массива A элементами из таблицы strgrA}
    for i:=1 to N do
        for j:=1 to N do
            A[i,j]:= StrToFloat(strgrA.Cells[i,j]);
    {Заполнение массива B элементами из таблицы strgrB}
    for i:=1 to N do
        B[i]:= StrToFloat(strgrB.Cells[0,i]);
    {Умножение массива A на массив B}
    for i:=1 to N do begin
        s:=0;
        for j:=1 to N do s:=s+A[i,j]*B[j];
        Y[i]:=s;
    {Вывод результата в таблицу strgrY}
        strgrY.Cells[0,i]:=FloatToStrF(Y[i],ffFixed,6,2);
    end;
end;
```

Проверьте работоспособность программы и правильность возвращаемых ей результатов.

Задание 1. Задание для самостоятельного выполнения

Создайте приложения для решения задач, приведенных в таблице 2, в соответствии с вариантом, предложенным преподавателем.

Таблица 2 – Варианты индивидуальных заданий

Вариант	Задание	Задание
1	1	Ввести массив $H[1..4]$ и число A. Для каждого элемента массива вычислить функцию $Z = \sqrt{\frac{H_i}{A}} + A$. Найти произведение элементов введенного массива, больших 2,5
	2	Задана матрица размером N x M. Получить массив B, присвоив его k-му элементу значение 0, если все элементы k-го столбца матрицы нулевые, и значение 1 в противном случае

Продолжение табл. 2.

Вариант	Задание	Задание
2	1	Ввести массив $A[0..5]$ и число P. Для каждого элемента массива вычислить функцию $B = e^{\frac{A_i}{P}}$. Найти сумму целых элементов введенного массива
	2	Задана матрица размером N x M. Получить массив B, присвоив его k-ому элементу значение 1, если k-я строка матрицы симметрична, и значение 0 в противном случае
3	1	Ввести массив $T[1..4]$ и число g. Для каждого элемента массива вычислить функцию $f = \sqrt[3]{T_i \cdot g}$. Найти произведение дробных элементов введенного массива
	2	Задана матрица размером N x M. Определить k – количество «особых» элементов матрицы, считая элемент «особым», если он больше суммы остальных элементов своего столбца
4	1	Ввести массив $F[0..6]$ и число C. Для каждого элемента массива вычислить функцию $G = tg\frac{F_i}{F_i + C}$. Найти сумму элементов введенного массива, кратных 3
	2	Задана матрица размером N x M. Определить k – количество «особых» элементов матрицы, считая элемент «особым», если в его строке слева от него находятся элементы, меньшие его, а справа – большие
5	1	Ввести массивы $T[1..4]$ и $C[1..4]$. Для каждого $T[i]$ и $C[i]$ вычислить функцию $d = 3{,}2T_i^2 + C_i$. Найти произведение нечетных элементов введенных массивов
	2	Задана матрица размером N x M. Упорядочить ее строки по неубыванию суммы их элементов
6	1	Ввести массивы $Z[0..3]$ и $A[0..3]$. Для каждого $Z[i]$ и $A[i]$ вычислить функцию $f = \sqrt{(Z_i + A_i)^3}$. Найти сумму дробных положительных элементов введенных массивов
	2	Задана матрица размером N x M. Упорядочить ее строки по неубыванию их наибольших элементов
7	1	Ввести массив $B[1..5]$ и число V. Для каждого элемента массива вычислить функцию $T = \sqrt[3]{3B_i} + V$. Найти произведение четных элементов введенного массива
	2	В матрице размером N x M поменять местами строку, содержащую элемент с наибольшим значением, со строкой, содержащей элемент с наименьшим значением

Продолжение табл. 2.

Вариант	Задание	Задание
8	1	Ввести массив $L[1..4]$ и число P. Для каждого элемента массива вычислить функцию $C = \cos^2\left(\frac{P}{P - L_i}\right)$. Найти сумму элементов введенного массива, меньших –5
	2	Из матрицы n-го порядка получить матрицу порядка n-1 путем удаления из исходной матрицы строки и столбца, на пересечении которых расположен элемент с наибольшим по модулю значением
9	1	Ввести массив $X[1..4]$ и число Y. Для каждого элемента массива вычислить функцию $d = \frac{X_i + Y^2}{2Y}$. Найти произведение целых элементов введенного массива
	2	Дан непустой массив из цифр. Вывести на экран цифру, наиболее часто встречающуюся в этом массиве
10	1	Ввести массив $A[1..5]$ и число Z. Для каждого элемента массива вычислить функцию $P = A_i + \sqrt{\frac{A_i}{2}}$. Найти сумму элементов введенного массива, кратных 2
	2	Задана матрица размером N x M. Получить массив B, присвоив его k-му элементу значение 1, если элементы k-ой строки матрицы упорядочены по убыванию, и значение 0 в противном случае
11	1	Ввести массив $K[1..4]$ и число A. Для каждого элемента массива вычислить функцию $r = \sin^2\left(\frac{K_i}{2}\right)A$. Найти произведение элементов введенного массива, некратных 3
	2	Задана матрица размером N x M. Определить k – количество различных элементов матрицы (повторяющиеся элементы считать один раз)
12	1	Ввести массив $Z[1..5]$ и число C. Для каждого элемента массива вычислить функцию $T = \sin^2 Z_i + C$. Найти сумму четных элементов введенного массива
	2	Элементы массива X циклически сдвинуть на k позиций влево
13	1	Ввести массив $L[1..4]$ и число P. Для каждого элемента массива вычислить функцию $C = \sin^2\left(\frac{P + L_i}{P}\right)$. Найти сумму элементов введенного массива, меньших –5
	2	Элементы массива X циклически сдвинуть на n позиций вправо

Окончание табл. 2.

Вариант	Задание	Задание
14	1	Ввести массив $F[0..6]$ и число C. Для каждого элемента массива вычислить функцию $G = \cos\frac{F_i - C}{F_i}$. Найти сумму элементов введенного массива, кратных 5
	2	Задана матрица размером N x M. Упорядочить ее строки по неубыванию их первых элементов

Контрольные вопросы

1. Что такое массив? Как описываются массивы?
2. Как получить доступ к конкретному элементу массива?
3. Опишите основные свойства компонента класса **TStringGrid**.
4. Как получить доступ к содержимому ячейки компонента класса **TStringGrid**?

ЛАБОРАТОРНАЯ РАБОТА 5

ПОСТРОЕНИЕ ГРАФИКА ФУНКЦИИ

Цель:

– научиться использовать графические возможности, предоставляемые Delphi, для отображения графической информации произвольного вида.

5.1 Формирование изображений программным способом

В любом визуальном компоненте Delphi существует специальный объект, средствами которого выполняется рисование видимых частей компонента, называемый холстом (канвой) и оформленный в виде свойства **Canvas**. Объект **Canvas** имеет пять основных свойств:

1) **Pen** – объект для рисования линий и границ геометрических фигур;
2) **Brush** – объект для заполнения фигур;
3) **Font** – объект для вывода текста;
4) **PenPos** – объект для хранения текущей позиции рисования;
5) **Pixels[x,y:Integer]** – двумерный массив, в котором хранятся цвета всех точек канвы.

Для рисования прямых, ломаных и кривых линий используются методы объекта **Canvas**, приведенные в таблице 1.

Таблица 1 – Методы объекта **Canvas** для рисования прямых, ломаных и кривых линий

Методы	Описание
MoveTo(X,Y:Integer)	Перемещает указатель текущей позиции в заданную точку
LineTo(X,Y:Integer)	Рисует прямую линию от текущей позиции рисования до заданной (X, Y) и перемещает указатель текущей позиции в точку с координатами (X, Y)

Окончание табл. 1.

Методы	Описание
Polyline(Points:array of TPoint)	Рисует ломаную линию, соединяя точки массива **Points**. Для получения элемента массива по двум координатам может использоваться функция **Point**(X, Y:Integer):TPoint
Arc(X1,Y1,X2,Y2,X3,Y3,X4,Y4:Integer)	Рисует дугу эллипса, вписанного в прямоугольник с координатами (X1, Y1) и (X2, Y2). Дуга определяется двумя радиусами эллипса, проходящими через точки (X3, Y3) и (X4, Y4), и рисуется против часовой стрелки от точки пересечения эллипса с первым радиусом до точки пересечения со вторым радиусом

Для рисования геометрических фигур предназначены методы, представленные в таблице 2.

Таблица 2 – Методы объекта **Canvas** для рисования геометрических фигур

Методы	Описание
Rectangle(X1,Y1,X2,Y2:Integer)	Рисует прямоугольник с левым верхним углом в точке (X1, Y1) и нижним правым углом в точке (X2, Y2). Прямоугольник рисуется текущими атрибутами кисти и пера
RoundRect(X1,Y1,X2,Y2,X3, Y3:Integer)	Рисует прямоугольник с закругленными углами, которые рисуются как четверти эллипса с шириной X3 и высотой Y3
Ellipse(X1,Y1,X2,Y2:Integer)	Рисует эллипс, вписанный в прямоугольник с левым верхним углом в точке (X1, Y1) и нижним правым углом в точке (X2, Y2). Эллипс рисуется текущими пером и кистью
Polygon(Points:array of TPoint)	Рисует ломаную линию, соединяя точки массива **Points**. Ломаная замыкается отрезком из последней точки в первую, и полученная линия-фигура заполняется текущей кистью

Для вывода текста служат методы объекта **Canvas**, представленные в таблице 3.

Таблица 3 – Методы объекта **Canvas** для вывода текста

Методы	Описание
TextOut(X,Y:Integer; const Text:string)	Выводит текущим шрифтом строку текста *Text* в прямоугольнике с левым верхним углом в точке (X,Y)
TextRect(Rect: TRect; X,Y:Integer; const Text:string)	Выводит текст в прямоугольнике Rect. Вывод за границы прямоугольника отсекается
TextHeight(const Text:string): Integer	Возвращает высоту (в пикселях) строки *Text* при выводе ее текущим шрифтом
TextWidth(const Text:string): Integer	Возвращает ширину (в пикселах) строки *Text* при выводе ее текущим шрифтом

5.2 Пример разработки программы

Постановка задачи. Создать приложение для построения графика функции $F(x) = 2\sin x \cdot e^{\frac{x}{5}}$.

1. Создайте новое приложение.

2. Для вычисления значения функции $F(x)$ опишем следующую функцию:

```
Function f(x:real):real;
begin
    f:=2*Sin(x)*exp(x/5);
end;
```

3. Создайте процедуру, которая будет выполнять построение графика функции на форме:

```
procedure GrOfFunc;
var
    x1,x2:real; // Границы изменения аргумента функции
    y1,y2:real; // Границы изменения значения функции
    x:real;  // Аргумент функции
    y:real;  // Значение функции в точке x
    dx:real;  // Приращение аргумента
    l,b:integer; // Левый нижний угол области вывода графика
    w,h:integer; // Ширина и высота области вывода графика
    mx,my:real; // Масштаб по осям X и Y
    x0,y0:integer; // Точка – начало координат
```

```
begin
    {Область вывода графика}
    l:=10;  // X – координата левого верхнего угла
    b:=Form1.ClientHeight-20;  // Y – координата левого верхнего угла
    h:=Form1.ClientHeight-40; //Высота
    w:=Form1.Width-40; //Ширина
    x1:=0;  // Нижняя граница диапазона аргумента
    x2:=25;  // Верхняя граница диапазона аргумента
    dx:=0.01;  // Шаг аргумента
    {Найдем максимальное и минимальное значения функции на отрезке [x1,x2]}
    y1:=f(x1); //Минимум
    y2:=f(x1); //Максимум
    x:=x1;
    repeat
        y := f(x);
        if y < y1 then y1:=y;
        if y > y2 then y2:=y;
        x:=x+dx;
    until (x>=x2);
    {Вычислим масштаб}
    my:=h/abs(y2-y1);  // Масштаб по оси Y
    mx:=w/abs(x2-x1);  //Масштаб по оси X
    {Нарисуем координатные оси}
    x0:=l;
    y0:=b-Abs(Round(y1*my));
    with form1.Canvas do begin
        MoveTo(l,b);LineTo(l,b-h);
        MoveTo(x0,y0);LineTo(x0+w,y0);
        TextOut(l+5,b-h,FloatToStrF(y2,ffGeneral,6,3));
        TextOut(l+5,b,FloatToStrF(y1,ffGeneral,6,3));
    {Построение графика}
        x:=x1;
        repeat
            y:=f(x);
            Pixels[x0+Round(x*mx),y0-Round(y*my)]:=clRed;
            x:=x+dx;
        until (x>=x2);
    end;
end;
```

4. Для определения стиля рисования рабочей области формы используется событие OnPaint. Создайте процедуру-обработчик прорисовки формы, в которой вызовите процедуру построения графика:

```
procedure TForm1.FormPaint(Sender: TObject);
begin
    GrOfFunc;
end;
```

5. Так как построенный график функции зависит от размеров формы, возникает необходимость его перерисовки при изменении размеров формы. Для этого создайте процедуру-обработчик изменения размеров формы (событие OnResize):

```
procedure TForm1.FormResize(Sender: TObject);
begin
    form1.Canvas.FillRect(Rect(0,0,ClientWidth,ClientHeight)); { Очистить форму }
    GrOfFunc; { Построить график }
end;
```

Результат выполнения программы показан на рисунке.

Задание 1. Задание для самостоятельного выполнения

Создайте приложение для построения графика функции, вид которой возьмите из лабораторной работы 3 по теме «Программирование циклических алгоритмов» в соответствии с вариантом, предложенным преподавателем.

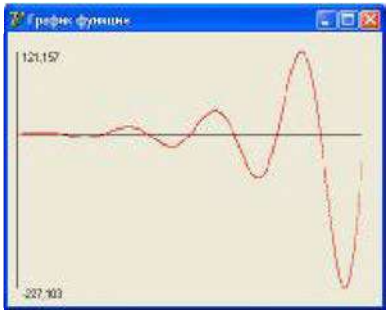


Рисунок Результат выполнения программы

Контрольные вопросы

1. Какое из свойств формы (и других компонентов) позволяет выводить графическую информацию?
2. Какие методы для вывода графической информации использовались в работе?
3. Каково назначение методов **MoveTo**, **LineTo**? Опишите формат их вызова.
4. Как изменить цвет отдельной точки канвы?

ЛАБОРАТОРНАЯ РАБОТА 6

ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ СРЕДСТВ ДЛЯ ОТОБРАЖЕНИЯ ГРАФИЧЕСКОЙ ИНФОРМАЦИИ

Цель:

– изучить возможности построения графиков с помощью компонента отображения графической информации класса **TChart**.

6.1 Проектирование диаграммы

Обычно результаты расчетов представляются в виде графиков и диаграмм. Система Delphi имеет мощный пакет стандартных программ вывода на экран и редактирования графической информации, который реализуется с помощью визуально отображаемого на форме компонента **Chart** (рис. 1) (палитра компонентов **Standard**).

Рисунок 1
Chart

Построение графика (диаграммы) производится после вычисления таблицы значений функций $y = f(x)$ на интервале $[X_{min}, X_{max}]$ с заданным шагом. Полученная таблица передается в специальный двумерный массив **Seriesk** (k – номер графика) компонента **Chart** с помощью метода **Add**. Компонент **Chart** осуществляет всю работу по построению графиков по значениям, переданным в объект **Seriesk**: строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданную таблицу в виде графиков или диаграмм различного типа. При необходимости с помощью встроенного редактора **EditingChart** компоненту **Chart** передаются данные о толщине, стиле и цвете линий, параметрах шрифта подписей, шагах разметки координатной сетки и другие настройки. В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам компонента **TChart**.

6.2 Пример разработки программы

Постановка задачи. Составить программу, отображающую графики функций **sin(*x*)** и **cos(*x*)** на интервале $[X_{min}, X_{max}]$. Предусмотреть возможность изменения разметки координатных осей, а также шага построения таблицы.

Форма приложения организуется в виде, представленном на рисунке 2.

Для ввода исходных данных используются компоненты **TEdit**:

- Xmin (edXmin);
- Xmax (edXmax);
- шаг разметки по X (edHx);
- Ymin (edYmin);
- Ymax (edYmax),
- шаг разметки по Y (edHy);
- шаг расчета таблицы (edh).

Также на форме размещаются три кнопки **TButton**:

- Разметить оси (btnMarkingAxis);
- Построить график (btnCreateChart);
- Закрыть (btnClose).

Разместим на форме компонент **TChart**.

Изменим свойство **Name** компонентов **Form** и **Chart** в соответствии с таблицей 1 (см. с. 42).

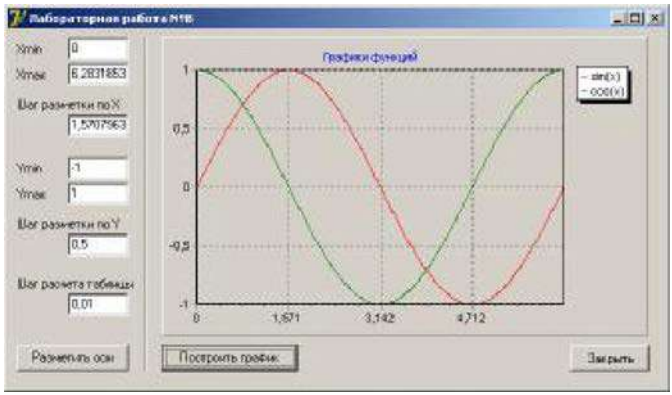


Рисунок 2 – Форма приложения

Таблица 1 – Свойства **Name** формы и диаграммы

Компонент	Свойство	Значение	Новое значение
Form Form1	Name	Form1	frmMain
Chart Chart1	Name	Chart1	chrtMy

Работа с компонентом TChart

Для изменения параметров компонента **Chart** необходимо дважды щелкнуть по нему мышью в окне формы. Появится окно редактирования **Editing chrtMy**. Для создания нового объекта **Series1** щелкнуть по кнопке **Add** на странице Series. В появившемся диалоговом окне **TeeChart Gallery** выбрать пиктограмму с надписью **Line** (график выводится в виде линии). Если нет необходимости представления графика в трехмерном виде, отключить переключатель **3D**. После нажатия на кнопку **OK** появится новая серия с названием **Series1**. Для изменения названия нажать кнопку **Title**. В появившемся однострочном редакторе набрать имя отображаемой функции – **sin(*x*)**. Аналогичным образом нужно создать объект **Series2** для функции **cos(*x*)**.

Для изменения надписи над графиком на странице Titles в редакторе набрать **Графики функций**.

Для разметки осей нужно выбрать страницу Axis и установить нужные параметры настройки осей.

Нажимая различные кнопки меню, познакомьтесь с другими возможностями **Editing Chart**.

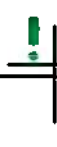
Написание текста программы

1. Определите следующие *глобальные* переменные: Xmin, Xmax, Ymin, Ymax, Hx, Hy, h : extended;

2. Создайте процедуру, которая выполняет разметку осей:

```
procedure TfrmMain.MarkingAxis;
begin
  {Отключение автоматического определения параметров нижней оси}
  chrtMy.BottomAxis.Automatic :=false;
  {Установка левой и правой нижней границы оси}
  chrtMy.BottomAxis.Minimum := Xmin;
  chrtMy.BottomAxis.Maximum := Xmax;
  {Отключение автоматического определения параметров левой оси}
  chrtMy.LeftAxis.Automatic :=false;
```

```
    {Установка нижней и верхней границы левой оси}
    chrtMy.LeftAxis.Minimum := Ymin;
    chrtMy.LeftAxis.Maximum := Ymax;
    {Установка шага разметки по нижней оси}
    chrtMy.BottomAxis.Increment :=Hx;
    chrtMy.LeftAxis.Increment := Hy;
end;
```



Внимание

Не забудьте в разделе описания типов в секции **privat** добавить поле с заголовком процедуры **procedure MarkingAxis;**

3. Создайте процедуру-обработчик создания формы. В данном месте программы устанавливаются начальные пределы и шаг разметки координатных осей:

```
procedure TfrmMain.FormCreate(Sender: TObject);
begin
    {Установка начальных параметров осей}
    Xmin :=0;
    Xmax:=2*pi;
    Ymin:=-1;
    Ymax:=1;
    Hx:=pi/2;
    Hy:=0.5;
    h:=0.01;  { Установка шага расчета таблицы
    {Вывод данных в окна однострочных редакторов}
    edXmin.Text := FloatToStr(Xmin);
    edXmax.Text := FloatToStr(Xmax);
    edYmin.Text := FloatToStr(Ymin);
    edYmax.Text := FloatToStr(Ymax);
    edHx.Text := FloatToStr(Hx);
    edHy.Text := FloatToStr(Hy);
    edh.Text := FloatToStr(h);
    {Вызов процедуры MarkingAxis}
    MarkingAxis;
end;
```

4. Создайте процедуру-обработчик нажатия кнопки **btnMarkingAxis**:

```
procedure TfrmMain.btnMarkingAxisClick(Sender: TObject);
begin
```

```
    {Чтение данных из окон однострочных редакторов}
    Xmin :=strtofloat(edXmin.Text);
    Xmax :=strtofloat(edXmax.Text);
    Ymin :=strtofloat(edYmin.Text);
    Ymax :=strtofloat(edYmax.Text);
    Hx :=strtofloat(edHx.Text);
    Hy :=strtofloat(edHy.Text);
    h:=strtofloat(edh.Text);
    {Вызов процедуры MarkingAxis}
    MarkingAxis;
end;
```

5. Создайте процедуру-обработчик нажатия кнопки **btnCreateChart**:

```
procedure TfrmMain.btnCreateChartClick(Sender: TObject);
var x,y1,y2:extended;
begin
    {Очистка графиков}
    Series1.Clear;
    Series2.Clear;
    {Устанавливаем разметки оси}
    btnMarkingAxis.Click;
    x:=Xmin; // Начальное значение по оси X
    repeat
        y1:=sin(x); // Расчет функции
        Series1.AddXY(x,y1,'',clTeeColor); // Вывод точки на график
        y2:=cos(x); // Расчет функции
        Series2.AddXY(x,y2,'',clTeeColor); // Вывод точки на график
        x:=x+h;
    until x>Xmax;
end;
```

6. Создайте процедуру-обработчик нажатия кнопки **btnClose**:

```
procedure TfrmMain.btnCloseClick(Sender: TObject);
begin
    close; // Закрываем приложение
end;
```

Задание 1. Задание для самостоятельного выполнения

Постройте графики функций в соответствии с таблицей 2.

Таблица 2 – Варианты индивидуальных заданий

Вариант	Функция	Вариант	Функция
1	$f=\begin{cases}x^2, x>0\\ x, x\le 0\end{cases}$	8	$f=\begin{cases}\ln x, x>0\\ x^2, x\le 0\end{cases}$
2	$f=\begin{cases}2+x^2, x>2.5\\ x^{-1}, x\le 2.5\end{cases}$	9	$f=\begin{cases}\sqrt{x}, x>4\\ \lvert x\rvert, x\le 4\end{cases}$
3	$f=\begin{cases}0, x=0\\ x^2+1, x\ne 0\end{cases}$	10	$f=\begin{cases}x^3, x\le 2\\ ctgx, x>0\end{cases}$
4	$f=\begin{cases}\sin x^2, x<1\\ \cos x^2, x\ge 1\end{cases}$	11	$f=\begin{cases}x^{\frac{2}{3}}, x<2.5\\ x^{\frac{3}{2}}, x\ge 2.5\end{cases}$
5	$f=\begin{cases}2x+3, x\le 0\\ x-1, x>0\end{cases}$	12	$f=\begin{cases}\sqrt{x+2}, x>0\\ \lvert x\rvert, x\le 0\end{cases}$
6	$f=\begin{cases}2tgx^2, x>3\\ 2tgx, x\le 3\end{cases}$	13	$f=\begin{cases}3+x^3, x<0\\ 3-x^3, x\ge 0\end{cases}$
7	$f=\begin{cases}\lvert x\rvert, x\\ x^{\frac{3}{2}}, x\ge 0\end{cases}$	14	$f=\begin{cases}tg3x, x<1\\ ctgx^2, x\ge 1\end{cases}$

Контрольные вопросы

1. Опишите основные свойства компонента **Chart**.
2. Графики и диаграммы каких типов можно строить с помощью компонента **Chart**?
3. Как в процессе выполнения программы изменить разметку осей компонента **Chart**?
4. Как добавить новую серию данных в компонент **Chart**?
5. Как к имеющейся серии данных компонента **Chart** добавить новую точку?
6. Как изменить свойства линии графика?

ЛАБОРАТОРНАЯ РАБОТА 7

ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ЗАПИСЕЙ И ФАЙЛОВ

Цели:

- изучить правила работы с компонентами **OpenDialog** и **SaveDialog**;
- написать программу с использованием файлов и данных типа запись.

7.1 Переменные типа запись

Запись – это структура данных, объединяющая элементы одного или различных типов, называемые полями. Записи удобны для создания структурированных баз данных с разнотипными элементами. Описание типа записи начинается словом **record** и заканчивается словом **end**. Между ними заключен список элементов, называемых полями, с указанием идентификаторов полей и типа каждого поля.

Например:

```
Type // Объявление типа запись
    TStudent = record
    Fio: string[20]; // Поле ф.и.о.
    Group: integer; // Поле номера студента группы
    Ocn: array[1..3] of integer; // Поле массива оценок
end;
```

Чтобы получить в программе реальную запись, нужно создать переменную соответствующего типа:

```
Var
    Student: TStudent; // Объявление переменной типа запись
```

Доступ к содержимому полей записи осуществляется посредством указания идентификатора переменной и идентификатора поля, разделенных точкой. Такая комбинация называется составным именем.

Например:

```
Student.Fio:= 'Иванов А.И.'; // Внесение данных в поля записи
Student. Group:=720603;
```

Доступ к полям можно осуществлять также при помощи оператора **with**, который имеет следующую структуру:

With <запись> **do** <оператор>

Например:

```
With Student do
begin
    Fio:= 'Иванов А.И.';
    Group:=720603;
end;
```

7.2 Файлы

Файл – это именованная область данных на внешнем физическом носителе. Для файла существует понятие *текущей позиции*, показывающей номер элемента, который будет прочитан или записан при очередном обращении к файлу. Для большинства файлов можно менять текущую позицию чтения-записи, выполняя прямой доступ к его элементам.

В Object Pascal различают три вида файлов в зависимости от способа их организации и доступа к элементам: *текстовые, типизированные и нетипизированные*.

Текстовой файл состоит из строк. Примером может служить файл исходного текста программы в Delphi (расширение *.pas). Для работы с текстовым файлом должна быть описана соответствующая файловая переменная: **var F: TextFile**.

Типизированные файлы имеют строго заданную их описанием структуру, когда у всех элементов фиксированный и одинаковый размер. Это свойство позволяет получить доступ к любому компоненту файла по его порядковому номеру. Элементами такого файла являются, как правило, записи. В описании файловой переменной указывается ее тип: **var F: TStudent**.

Нетипизированный файл – это файл, в котором данные не имеют определенного типа и рассматриваются как последовательность байт. Файловая переменная объявляется: **var F: File**.

7.3 Работа с текстовыми файлами

Запись и чтение текстового файла

Приступая к работе с файлом, нужно вызвать процедуру **AssignFile**, чтобы файловой переменной поставить в соответствие имя файла на диске, и создать файл с использованием процедуры **Rewrite:**

```
AssignFile(F, 'MyFile.txt');
Rewrite(F);
```

Для записи строк в файл используются процедуры **Write** и **Writeln**:

```
Writeln(F, 'Pi = ',Pi);
Writeln(F, 'Exp = ', Exp(1));
```

После работы файл должен быть закрыт с использование процедуры **CloseFile**:

```
CloseFile(F);
```

Чтение информации из текстового файла

После инициализации файловой переменной **(AssignFile)** файл открывается с помощью процедуры **Reset**. Для чтения элементов используются процедуры **Read** и **Readln**, в которых первый параметр показывает, откуда происходит ввод данных. После работы файл закрывается.

Например:

```
var F: TextFile;
    S: string;
begin
    AssignFile(F, 'MyFile.txt');
    Reset(F);
    Readln(F, S);
    CloseFile(F);
end;
```

Стандартные подпрограммы управления файлами представлены в приложении К.

7.4 Компоненты OpenDialog и SaveDialog

Диалоговые окна для выбора открываемого или сохраняемого файла организуются с помощью компонентов **OpenDialog** и **SaveDialog** (рис. 1, 2) и находятся на странице Dialogs панели компонентов. Все компоненты этой страницы являются невизуальными, т. е. во время работы программы они не отображаются, поэтому их можно разместить в любом удобном месте формы.

Рисунок 1 – **OpenDialog**

Данные компоненты являются объектно-ориентированными оболочками стандартных диалоговых окон Windows: **Open** и **Save**, имеют идентичные свойства и отличаются только внешним видом.

Рисунок 2 **SaveDialog**

После вызова компонента появляется диалоговое окно, с помощью которого выбирается имя программы и путь к ней. В случае успешного завершения диалога имя выбранного файла и маршрут поиска содержатся в свойстве **FileName**. Для фильтрации файлов, отображаемых в окне просмотра, используется свойство **Filter**, а для задания расширения файла, в случае, если оно не задано пользователем, – свойство **DefaultExt**. Если необходимо изменить заголовок диалогового окна, используется свойство **Title**.

7.5 Пример разработки программы

Постановка задачи. Написать программу, вводящую в файл или читающую из файла ведомость абитуриентов, сдавших вступительные экзамены. Каждая запись должна содержать фамилию, а также оценки по физике, математике и сочинению. Вывести список абитуриентов, отсортированный в порядке уменьшения их среднего балла, и записать эту информацию в текстовой файл.

Создайте новое приложение. Примерный вид формы представлен на рисунке 3.

Разместите на форме компоненты **OpenDialog** и **SaveDialog**. Установите фильтр типов файлов. Для этого выберите соответствующий компонент,

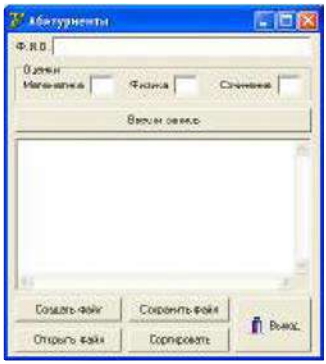


Рисунок 3 – Форма приложения

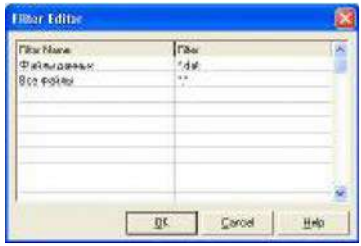


Рисунок 4 – Окно редактора фильтра

дважды щелкните в строке справа от названия свойства **Filter** инспектора объектов. Появится окно **Filter Editor** (рис. 4), в левой части которого записывается текст, характеризующий соответствующий фильтр, а в правой – маску. Для **OpenDialog1** установим значения маски как показано на рисунке 4.

Формат *.dat означает, что будут видны все файлы с расширением *. dat, а формат *.* – все файлы с любым именем и любым расширением.

Для того, чтобы файл автоматически записывался с расширением .dat, в свойстве **DefaultExt** запишем требуемое расширение – .dat.

Аналогичным образом настроим **SaveDialog1** для текстового файла (расширение .txt).

Описание переменных, используемых при написании программы

В программе будет использоваться переменная типа запись. Для этого создадим следующий тип:

```
Type
    TStudent = record
        FIO: string[40]; // Поле Ф. И. О.
        otc: array[1..3] of word; // Поле массива оценок
        sball : extended; // Поле среднего балла
    end;
```

Определим следующие глобальные переменные:

```
Var
    Fz  : file of  Tstudent; // Файл типа запись
    Ft  : TextFile; // Текстовой файл
    Stud : array[1..100] of Tstudent; // Массив записей
    nzap : integer; // Номер записи
    FileNameZ, FileNameT : string; // Имя файла
```

Создание процедур-обработчиков событий

1. Создайте процедуру-обработчик создания формы и приведите ее к следующему виду:

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

```
    Edit1.Text:='';
    Edit2.Text:='';
    Edit3.Text:='';
    Edit4.Text:='';
    Memo1.Clear;
    Button1.Hide; // Сделать невидимой кнопку Ввести запись
    nzap:=0;
end;
```

2. Создайте процедуру-обработчик нажатия кнопки **Создать файл**:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    OpenDialog1.Title :='Создать новый файл'; // Изменение заголовка окна диалога
    if OpenDialog1.Execute then // Выполнение стандартного диалога выбора имени файла
        begin
            FileNameZ:= OpenDialog1.FileName; // Возвращение имени дискового файла
            AssignFile(Fz, FileNameZ); // Связывание файловой переменной Fz с именем файла
            Rewrite(Fz); // Создание нового файла
            Button1.Show; // Сделать видимой кнопкуВвести запись
    end;
end;
```

3. Создайте процедуру-обработчик нажатия кнопки **Открыть файл**:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
if OpenDialog1.Execute then // Выполнение стандартного диалога выбора имени файла
    begin
        FileNameZ:= OpenDialog1.FileName; // Возвращение имени дискового файла
        AssignFile(Fz, FileNameZ); // Связывание файловой переменной Fz с именем файла
        Reset(Fz); // Открытие существующего файла
        while not eof(fz) do begin
            nzap:=nzap+1;
            Read(fz,stud[nzap]); // Чтение записи из файла
            with stud[nzap] do Memo1.Lines.Add(fio+'  '+IntToStr(otc[1])+'
            '+IntToStr(otc[2])+'  '+IntToStr(otc[3]));
        end;
        Button1.Show; // Сделать видимой кнопку Ввести запись
    end;
end;
```

4. Создайте процедуру-обработчик нажатия кнопки **Ввести запись**:

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

```
begin
    nzap:=nzap+1;
    with stud[nzap] do begin
        FIO:=Edit1.Text;
        otc[1]:=StrToInt(Edit2.Text);
        otc[2]:=StrToInt(Edit3.Text);
        otc[3]:=StrToInt(Edit4.Text);
        sball:=(otc[1]+otc[2]+otc[3])/3;
        Memo1.Lines.Add(fio+'  '+IntToStr(otc[1])+'  '+ IntToStr(otc[2])+'
        '+IntToStr(otc[3]));
    end;
    Write(fz,Stud[nzap]); // Запись в файл
    Edit1.Text:='';
    Edit2.Text:='';
    Edit3.Text:='';
    Edit4.Text:='';
end;
```

5. Создайте процедуру-обработчик нажатия кнопки **Сортировать**:

```
procedure TForm1.Button4Click(Sender: TObject);
var i,j : word;
    st : TStudent;
begin
    for i:=1 to nzap-1 do // Сортировка массива записей
        for j:=i+1 to nzap do
            if Stud[i].sball < Stud[j].sball then begin
               st:=Stud[i];
               Stud[i]:=Stud[j];
               Stud[j]:=st;
        end;
        Memo1.Clear;
        for i:=1 to nzap do // Вывод в окно Memo1 отсортированных записей
        with stud[i] do
        Memo1.Lines.Add(IntToStr(i)+' '+fio+' '+FloatToStrf(sball,fffixed,4,2));
end;
```

6. Создайте процедуру-обработчик нажатия на кнопки **Сохранить файл**:

```
procedure TForm1.Button5Click(Sender: TObject);
var i:word;
begin
```

```
    if SaveDialog1.Execute then   // Выполнение стандартного диалога выбора имени файла
    begin
        FileNameT:= SaveDialog1.FileName; // Возвращение имени дискового файла
        AssignFile(Ft, FileNameT); // Связывание файловой переменной Ft с именем файла
        Rewrite(Ft); // Открытие нового текстового файла
        for i:=1 to nzap do
            with stud[i] do Writeln(Ft,i:4,'. ',fio,sball:8:2); // Запись в текстовой файл
        CloseFile(Ft); // Закрытие текстового файла
    end;
end;
```

7. Создайте процедуру-обработчик нажатия кнопки **Выход**:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
    Application.Treminate;
end;
```

8. Создайте процедуру-обработчик закрытия формы:

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    CloseFile(fz);  // Закрытие файла записей при закрытии формы
end;
```

Работа с программой

После запуска программы появится ее основное окно. Кнопка **Ввести запись** видна не будет. Необходимо создать новый файл записей, нажав на кнопку **Создать файл**, или открыть ранее созданный, нажав **Открыть файл**. После этого станет видна кнопка **Ввести запись**, что даст возможность ввода записи. При нажатии на кнопку **Сортировать** произойдет сортировка ведомости по убыванию среднего балла. Затем при нажатии на кнопку **Сохранить файл** будет создан текстовой файл, содержащий отсортированную ведомость. Файл записей закрывается одновременно с программой при нажатии на кнопку **Выход** или закрытии окна программы.

Задание 1. Задание для самостоятельного выполнения

Создать программу в соответствии с заданием (табл., см. с. 54). В программе предусмотреть сохранение вводимых данных в файле и возможность чтения из ранее сохраненного файла. Результаты выводить в окно просмотра и в файл (четные варианты в текстовый, нечетные – в типизированный).

Таблица – Варианты индивидуальных заданий

Вариант	Задание
1	В магазине формируется список лиц, записавшихся на покупку товара повышенного спроса. Каждая запись этого списка содержит: порядковый номер, Ф. И. О. покупателя, его домашний адрес и дату постановки на учет. Удалить из списка все повторные записи, проверяя Ф. И. О. и домашний адрес
2	Список товаров, имеющихся на складе, включает в себя наименование товара, количество единиц, цену единицы и дату поступления товара на склад. Вывести в алфавитном порядке список товаров, хранящихся больше месяца, стоимость которых превышает 1 млн руб.
3	Для получения места в общежитии формируется список студентов, который включает Ф. И. О. обучаемого, группу, средний балл, доход на члена семьи. Общежитие, в первую очередь, предоставляется тем, у кого доход на члена семьи меньше двух минимальных зарплат, затем остальным в порядке уменьшения среднего балла. Вывести список очередности предоставления мест в общежитии
4	В справочной автовокзала хранится расписание движения автобусов. Для каждого рейса указаны его номер, тип автобуса, пункт назначения, время отправления и прибытия. Вывести информацию о рейсах, которыми можно воспользоваться для прибытия в пункт назначения раньше заданного времени
5	На междугородной АТС информация о разговорах содержит дату, время разговора, код и название города, тариф, номер телефона и номер телефона абонента. Вывести по каждому городу общее время разговоров с абонентом и сумму
6	Информация о сотрудниках фирмы включает: Ф. И. О., табельный номер, количество проработанных часов за месяц, почасовой тариф. Рабочее время свыше 144 ч считается сверхурочным и оплачивается в двойном размере. Вывести размер заработной платы каждого сотрудника фирмы за вычетом подоходного налога, который составляет 12% от суммы заработка
7	Информация об участниках спортивных соревнований содержит: наименование страны, название команды, Ф. И. О. игрока, игровой номер, возраст, рост, вес. Вывести информацию о самой молодой, рослой и легкой команде
8	Для книг, хранящихся в библиотеке, задаются: регистрационный номер книги, автор, название, год издания, издательство, количество страниц. Вывести список книг с фамилиями авторов в алфавитном порядке, изданных после заданного года
9	Различные цехи завода выпускают продукцию нескольких наименований. Сведения о выпущенной продукции включают: наименование, количество, номер цеха. Для заданного цеха вывести количество выпущенных изделий по каждому наименованию в порядке убывания

Окончание табл.

Вариант	Задание
10	Информация о сотрудниках предприятия содержит: Ф. И. О., номер отдела, должность, дату начала работы. Вывести списки сотрудников по отделам в порядке убывания стажа
11	Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит: Ф. И. О., адрес, оценки. Определить количество абитуриентов, проживающих в г. Минске и сдавших экзамены со средним баллом не ниже 4,5, вывести их фамилии в алфавитном порядке
12	В справочной аэропорта хранится расписание вылета самолетов на следующие сутки. Для каждого рейса указаны: номер рейса, тип самолета, пункт назначения, время вылета. Вывести все номера рейсов, типы самолетов и время вылета для заданного пункта назначения в порядке возрастания времени вылета
13	У администратора железнодорожных касс хранится информация о свободных местах в поездах дальнего следования на ближайшую неделю в следующем виде: дата выезда, пункт назначения, время отправления, число свободных мест. Оргкомитет международной конференции обращается к администратору с просьбой зарезервировать m мест до города N на k-й день недели со временем отправления поезда не позднее t часов вечера. Вывести время отправления или сообщение о невозможности выполнить заказ в полном объеме
14	В радиоателье хранятся квитанции о сданной в ремонт радиоаппаратуре. Каждая квитанция содержит следующую информацию: наименование группы изделий (телевизор, радиоприемник и т. п.), марку изделия, дату приемки в ремонт, состояние готовности заказа (выполнен, не выполнен). Вывести информацию о состоянии заказов на текущие сутки по группам изделий

Контрольные вопросы

1. Как описываются данные типа запись? Что такое поле записи?
2. Каким образом можно получить доступ к значению поля переменной типа запись?
3. Какие типы файлов существуют в Delphi?
4. Опишите порядок работы с файлами.
5. Опишите основные процедуры и функции для работы с файлами, использованные в работе.
6. В чем отличия в работе с текстовыми и типизированными файлами?

Рекомендуемая литература

1. *Культин, Н. Б.* Delphi в задачах и примерах / Н. Б. Культин. – СПб. : БХВ-Петербург, 2003.

2. *Культин, Н. Б.* Основы программирования в Delphi 7 / Н. Б. Культин. – СПб. : БХВ-Петербург, 2003.

3. *Синицын, А. К.* Основы алгоритмизации и программирования в среде Delphi. Базовые типы и простые алгоритмы: лаб. практикум по курсу «Основы алгоритмизации и программирования» для студентов I–II курсов всех специальностей БГУИР / А. К. Синицын, А. А. Навроцкий. – Минск : БГУИР, 2006. – 80 с. : ил.

4. *Фаронов, В. В.* Delphi. Программирование на языке высокого уровня: учеб. для вузов / В. В. Фаронов. – СПб. : Питер, 2005.

5. *Энго, Ф.* Как программировать на Delphi 3 / Ф. Энго. – Киев : DiaSoft, 1997.

ПРИЛОЖЕНИЯ

Команды основного меню

В меню **File** находятся команды для выполнения операций с проектами, модулями и файлами (табл. 1).

В меню **Edit** расположены команды (табл. 2), осуществляющие операции редактирования, работы с областью обмена данными, отмены действий и управления отображением компонентов.

Таблица 1 – Команды меню **File**

Команда		Описание
New	**Application**	Создает новый проект, состоящий из формы, модуля и файла проекта
	Form	Создает новую форму и подключает ее к проекту
	Data Module	Создает новый модуль данных и подключает его к проекту
	Other	Позволяет выбрать тип элемента из репозитория (архива, в котором хранятся заготовки для новых программ) и создать его
Open		Открывает ранее созданный проект, модуль, форму или текстовой файл
Reopen		Вызывает список ранее загружавшихся проектов и форм для выбора и повторной загрузки
Save		Сохраняет текущую форму, модуль или файл
Save As		Сохраняет текущую форму под новым именем
Save Project As		Сохраняет текущий проект под новым именем
Save All		Сохраняет все открытые файлы, проект и используемые им модули
Close		Закрывает текущую форму
Close All		Закрывает все открытые файлы
Use Unit		Добавляет имя указанного модуля в список используемых модулей (**USES**) текущего активного модуля
Print		Выводит содержимое активного файла на печать
Exit		Завершает работу Delphi

Таблица 2 – Команды меню **Edit**

Команда	Описание
Undo	Отменяет ранее выполненные действия
Redo	Восстанавливает отмененные действия
Cut	Вырезает выделенный объект и помещает его в буфер обмена данными
Copy	Копирует выделенный объект и (или) фрагмент текста программы и помещает его в буфер обмена данными
Paste	Копирует содержимое буфера обмена данными в редактор или форму
Delete	Удаляет выбранный объект или фрагмент программы
Select All	Выделяет все компоненты формы или весь текст программы
Align to Grid	Выравнивает выбранный компонент по сетке
Bring to Front	Перемещает выбранный компонент на передний план
Send to Back	Перемещает выбранный компонент под другие компоненты (на задний план)
Align	Выравнивает компоненты
Size	Изменяет размер выделенных компонентов
Scale	Изменяет размер всех компонентов в форме
Tab Order	Изменяет порядок табуляции компонентов в активной форме
Creation Order	Задает порядок создания невизуальных компонентов
Lock Controls	Запрещает перемещение компонентов внутри формы
Add To Interface	Позволяет определить новую процедуру, функцию или свойство компонента **ActiveX**

Меню **Search** предоставляет команды для поиска и замены, а также команды для поиска указанных символов и строк, содержащих ошибки, найденные компилятором (табл. 3).

Таблица 3 – Команды меню **Search**

Команда	Описание
Find	Поиск указанного фрагмента текста
Find in files	Поиск указанного текста в нескольких файлах, задаваемых в диалоговой панели
Replace	Поиск указанного фрагмента текста и замена его новым текстом
Search Again	Повторный поиск или повторная замена
Incremental Search	Поиск текста по мере его ввода
Go to Line Number	Перемещение курсора на строку с указанным номером
Find Error	Поиск ошибки времени исполнения (run-time error)
Browse Symbol	Показывает характеристики указанного символа программы по его имени

В меню **View** содержатся команды (табл. 4) для отображения различной информации и вызова **Менеджера Проектов**, **Инспектора Объектов**, **Браузера Объектов** и других информационных утилит.

В меню **Project** содержатся команды (табл. 5) для компиляции и сборки проектов, а также установки опций текущего проекта.

Таблица 4 – Команды меню **View**

Команда		Описание
Project Manager		Менеджер Проектов (Project Manager)
Object Inspector		Инспектор Объектов (Object Inspector)
Alignment Palette		Палитра выравнивания компонентов
Browser		Браузер Объектов (Object Browser)
Debug Windows	**Breakpoints**	Список точек останова (Breakpoints List)
	Call Stack	Стек вызовов (Call Stack)
	Watches	Список точек слежения за переменными (Watch List)
	Threads	Список потоков команд и их статус
	Modules	Список модулей, загружаемых при выполнении данного проекта
Component List		Список компонентов
Window List		Список открытых окон
Toggle Form/Unit		Переключает активность из окна формы в окно текста программы и обратно
Unit		Показывает окно текста программы
Forms		Показывает окно формы
Type library		Отображает содержимое библиотеки типов для компонентов **ActiveX**, серверов **ActiveX** и других **COM**-объектов
New Edit Window		Открывает новое окно с текстом текущей программы

Таблица 5 – Команды меню **Project**

Команда	Описание
Add to Project	Добавляет файл к проекту
Remove from Project	Удаляет файл из проекта
Import Type Library	Импортирует в проект библиотеку типов элементов **ActiveX**
Add To Repository	Добавляет проект в репозиторий объектов
Compile	Компилирует модули, исходный текст которых изменился после последней компиляции
Build	Компилирует все модули и создает исполняемую программу

Окончание табл. 5

Команда	Описание
Syntax Check	Проверяет синтаксическую правильность программы
Information	Отображает информацию о проекте
Web Deployment Options	Позволяет задать опции для внедрения компонента **ActiveX** или активной фирмы на Web-узел
Web Deploy	Внедряет компонент **ActiveX** или активную фирму на Web-узел
Options	Задает опции компилятора и компоновщика, управляет рабочими каталогами

В меню **Run** расположены команды (табл. 6) для отладки программ, позволяющие управлять различными функциями устроенного отладчика.

Таблица 6 – Команды меню **Run**

Команда	Описание
Run	Компилирует и выполняет программу
Parameters	Задает параметры командной строки
Register ActiveX Server	Регистрирует сервер **ActiveX** в реестре **Windows**
Unregister ActiveX Server	Удаляет информацию о ранее зарегистрированном сервере **ActiveX** в реестре **Windows**
Step Over	Пошагово выполняет программу
Trace Into	Пошагово выполняет программу с заходом в подпрограммы
Trace To Next Source Line	Пошагово выполняет программу до следующей строки исходного текста
Run To Cursor	Выполняет программу до строки в окне редактора, на которой находится курсор
Show Execution Point	Отображает оператор, на котором было прервано выполнение программы
Program Pause	Приостанавливает выполнение программы
Program Reset	Завершает выполнение программы
Add Watch	Добавляет точку слежения за переменными
Add Breakpoint	Добавляет точку останова
Evaluate/Modify	Позволяет узнать или изменить значение переменной

В меню **Component** содержатся команды (табл. 7) для создания компонентов, установки новых компонентов, импорта компонентов **ActiveX**, создания нового компонента на базе существующего и установки пакетов.

Меню **Database** содержит средства (табл. 8) для работы с базами данных.

Из меню **Tools** доступны средства настройки среды, дополнительные утилиты, входящие в состав Delphi, а также репозиторий объектов (табл. 9).

Таблица 7 – Команды меню **Component**

Команда	Описание
New Component	Вызывает окно эксперта компонентов
Install Component	Помещает компонент в существующий или новый проект
Import ActiveX Control	Импортирует компонент **ActiveX**
Create Component Template	Сохраняет компонент как шаблон для создания других компонентов
Install Package	Устанавливает пакеты, необходимые для прогона программы
Configure Palette	Вызывает диалоговую панель конфигурации палитры компонентов

Таблица 8 – Команды меню **Database**

Команда	Описание
Explore	Вызывает инструмент исследования баз данных – **Database Explorer** или **SQL Database** (в зависимости от версии Delphi)
SQL Monitor	Вызывает инструмент запросов к **БД** – **SQL Monitor**
Form Wizard	Вызывает окно эксперта форм для создания формы, отображающей наборы данных из удаленных или локальных БД

Таблица 9 – Команды меню **Tools**

Команда	Описание
Environment Options	Вызывает диалоговую панель настройки среды
Repository	Вызывает репозиторий
Configure Tools	Вызывает диалоговую панель редактирования опции **Tools**
Package Collection Editor	Вызывает окно редактора пакетов
Image Editor	Вызывает окно редактора графики
Database Desktop	Вызывает инструмент обслуживания **БД – Database Desktop**

Основные группы компонентов в палитре компонентов

Страница **Standard** содержит набор основных управляющих элементов пользовательского интерфейса (рис. 1).

Описание компонентов приведено в таблице 1.

Рисунок 1 – Страница **Standard** палитры компонентов

Таблица 1 – Описание компонентов страницы **Standard**

Компонент	Описание
MainMenu	Используется для создания на форме строки меню с выпада-ющим подменю
PopupMenu	Используется для создания локальных всплывающих меню у формы и управляющих элементов. Всплывающие меню появляются, когда пользователь щелкает правой кнопкой мыши
Label	Отображает нередактируемый текст и используется для выполнения подписей к другим управляющим элементам формы
Edit	Строка редактора, в которой пользователь может ввести некоторый текст
Memo	Простой текстовый редактор, в котором пользователь может ввести несколько строк текста
Button	Кнопка с надписью, которая получает фокус ввода. Нажатие кнопки приводит к выполнению некоторого действия
CheckBox	Независимый переключатель с двумя состояниями (Вкл./Выкл.), используется для установки режимов. О включенном состоянии переключателя свидетельствует «галочка»
RadioButton	Зависимый переключатель. Зависимые переключатели объединяются в группы и позволяют пользователю выбрать один из нескольких исключающих друг друга режимов. При включении одного переключателя тот, что был включен ранее, выключается. Включенный переключатель отмечается жирной точкой
RadioGroup	Группа зависимых переключателей

ЛАБОРАТОРНАЯ РАБОТА 3

ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Цели:

- изучить простейшие средства отладки в среде Delphi;
- научиться программировать циклические алгоритмы.

3.1 Операторы повтора языка Pascal

Под ц и к л о м понимается многократное выполнение одних и тех же операторов при различных значениях промежуточных данных. Число повторений может быть задано в явной или неявной форме.

Для организации повторений в языке Delphi предусмотрены три различных оператора цикла:

1) *оператор цикла с постусловием*. Используется в тех случаях, когда тело цикла должно быть выполнено перед тем, как произойдет проверка условия завершения цикла, и имеет следующий вид:

```
repeat
    < оператор1 >;
  ...
    < операторN >
until < условие завершения цикла >;
```

Тело цикла выполняется до тех пор, пока условие завершения цикла (выражение булевского типа) не станет истинным;

2) *оператор цикла с предусловием*. Является альтернативой оператору **repeat** и содержит условие выполнения цикла, а не условие завершения:

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

Перед каждым выполнением тела цикла происходит проверка условия. Если оно истинно, цикл выполняется и условие вычисляется заново, если ложно, происходит выход из цикла.

Внимание

Если тело цикла с предусловием содержит несколько операторов, то они должны быть заключены в операторные скобки **begin** <операторы> **end**.

Таким образом, оператор цикла с предусловием может не выполниться ни разу, а оператор цикла с постусловием всегда выполняется хотя бы один раз;

3) *оператор цикла с заданным числом повторений.* Используется в том случае, если заранее известно количество повторений цикла, и имеет следующий вид:

for <параметр цикла>:=<значение1> **to** <значение2> **do** <оператор>;

где <параметр цикла> – переменная любого порядкового типа, кроме вещественного;

<значение1> и <значение2> – выражения, определяющие соответственно начальное и конечное значения параметра цикла;

<оператор> – тело цикла.

Оператор **for** обеспечивает выполнение тела цикла до тех пор, пока не будут использованы все значения параметра цикла от начального до конечного. После каждого повтора значение параметра цикла увеличивается на единицу.

Существует модификация данного оператора, используемая в случае убывания параметра цикла с шагом 1 и имеющая вид:

for <параметр цикла>:=<значение1> **downto** <значение2> **do** <оператор>;

3.2 Средства отладки программ в Delphi

Практически каждая созданная программа содержит ошибки:

1) *первого уровня* (ошибки компиляции) связаны с неправильной записью операторов (орфографические, синтаксические), при обнаружении которых компилятор Delphi останавливается напротив первого операто-

ра, в котором обнаружена ошибка. В нижней части экрана появляется текстовое окно, содержащее сведения обо всех ошибках, найденных в проекте. Каждая строка этого окна содержит имя файла, в котором найдена ошибка, номер строки с ошибкой и ее характер. Для быстрого перехода к интересующей ошибке необходимо дважды щелкнуть на строке с ее описанием. Для получения более полной информации о характере ошибки необходимо обратиться к справочной системе нажатием клавиши **F1**. Следует уделить внимание тому, что одна ошибка может повлечь за собой другие, которые исчезнут при ее исправлении, поэтому необходимо исправлять ошибки последовательно, сверху вниз, и после исправления каждой ошибки компилировать программу снова;

2) *второго уровня* (ошибки выполнения) связаны с ошибками выбранного алгоритма решения или неправильной программной реализацией алгоритма. Указанные ошибки проявляются в том случае, если результат расчета оказывается неверным либо происходит переполнение, деление на ноль и др., поэтому перед использованием отлаженной программы ее необходимо протестировать, т. е. выполнить расчеты при таких комбинациях исходных данных, для которых известен результат. Если тестовые расчеты указывают на ошибку, то для ее поиска следует использовать встроенные средства отладки среды Delphi.

В простейшем случае для локализации места ошибки рекомендуется поступать следующим образом: в окне редактирования текста установить курсор в строке перед подозрительным участком и нажать клавишу **F4** (выполнение до курсора). Выполнение программы будет остановлено на строке, содержащей курсор. Теперь можно просмотреть значения интересующих переменных. Для этого необходимо поместить на нужную переменную курсор (на экране будет высвечено ее значение) либо нажать **Ctrl** + **F7** и в появившемся диалоговом окне указать имя интересующей переменной (с помощью данного окна можно также изменить значение переменной во время выполнения программы). Нажимая клавишу **F7** (пошаговое выполнение), можно построчно выполнять программу, контролируя изменение тех или иных переменных и правильность вычислений. Если курсор находится внутри цикла, то после нажатия **F4** расчет останавливается после одного выполнения тела цикла. Для продолжения расчетов следует выбрать команду **Run** в меню **Run**.

3.3 Пример разработки программы

Предположим, что нужно разработать программу, которая выводит таблицу значений функции

$$S(x)=\sum_{k=0}^{N}(-1)^k\frac{x^k}{k!}.$$

Причем значение аргумента x изменяется в интервале от $x1$ до $x2$ с шагом h.

Разместим на форме четыре компонента **Label**, четыре компонента **Edit**, текстовое поле **Memo** и одну кнопку. Примерный вид формы представлен на рисунке 1.

Создание процедур-обработчиков событий

1. Создайте процедуру-обработчик создания формы и приведите ее текст к следующему виду:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Memo1.Clear;
    Memo1.Lines.Add('Результаты ст.гр.555 Иванова Сидора Петровича');
end;
```

2. Создайте процедуру-обработчик нажатия кнопки:

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

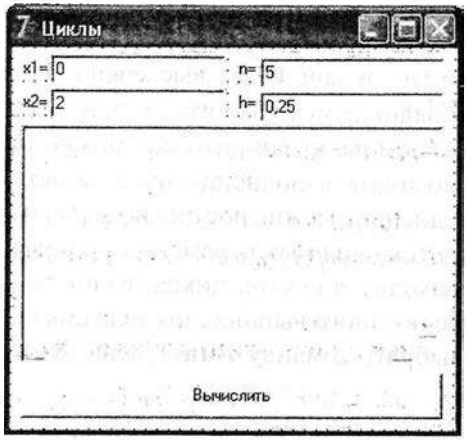


Рисунок 1 – Форма приложения

```
var x1,x2,x,h,a,s:extended;
    N,k,c:integer;
begin
    x1:=StrToFloat(Edit4.Text);
    Memo1.Lines.Add('x1='+Edit1.Text);
    x2:=StrToFloat(Edit2.Text);
    Memo1.Lines.Add('x2='+Edit2.Text);
    N:=StrToInt(Edit3.Text);
    Memo1.Lines.Add('n='+Edit3.Text);
    h:=StrToFloat(Edit4.Text);
    Memo1.Lines.Add('h='+Edit4.Text);
    c:=-1; x:=x1;
    repeat
        a:=1; S:=1;
        for k:=1 to N do begin
            a:=c*a*x/k;
            s:=s+a;
        end;
            Memo1.Lines.Add('при x='+FloatToStrF(x,ffFixed,6,2)+ '   сумма =' +
            FloatToStrF(s,ffFixed,6,2));
            x:=x+h;
    until x>x2;
end;
```

Результат выполнения программы представлен на рисунке 2.

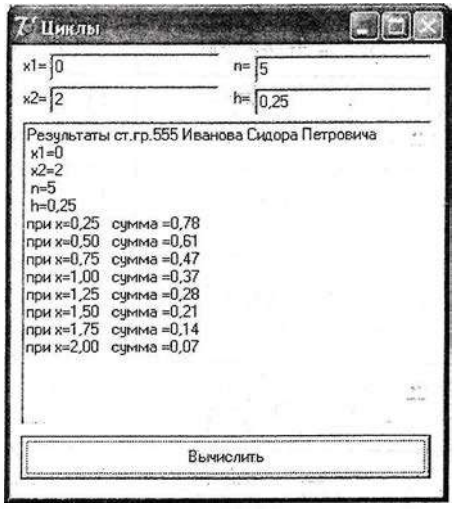


Рисунок 2 – Результат работы программы

Задание 1. Задание для самостоятельного выполнения

Выведите на экран таблицу значений функции $Y(x)$ и ее разложения в ряд $S(x)$ для x, изменяющихся от x_n до x_k с шагом $h=(x_k-x_n)/n$. Близость значений $S(x)$ и $Y(x)$ во всем диапазоне значений x указывает на правильность вычисления $S(x)$ и $Y(x)$.

Таблица – Значения функции $Y(x)$ и ее разложения в ряд $S(x)$ для x

№	x_n	x_k	$S(x)$	n	$Y(x)$
1	0,1	1	$x-\frac{x^3}{3!}+\ldots+(-1)^n\frac{x^{2n+1}}{(2n+1)!}$	16	$\sin x$
2	0,1	1	$1+\frac{x^2}{2!}+\ldots+\frac{x^{2n}}{(2n)!}$	10	$\frac{e^x+e^{-x}}{2}$
3	0,1	1	$1+\frac{\cos\frac{\pi}{4}}{1!}x+\ldots+\frac{\cos n\frac{\pi}{4}}{n!}x^n$	12	$e^{x\cos\frac{\pi}{4}}\cos(x\sin\frac{\pi}{4})$
4	0,1	1	$1-\frac{x^2}{2!}+\ldots+(-1)^n\frac{x^{2n}}{(2n)!}$	8	$\cos x$
5	0,1	1	$1+3x^2+\ldots+\frac{2n+1}{n!}x^{2n}$	14	$(1+2x^2)e^{x^2}$
6	0,1	1	$x+\frac{x^3}{3!}+\ldots+\frac{x^{2n+1}}{(2n+1)!}$	8	$\frac{e^x-e^{-x}}{2}$
7	0,1	1	$\frac{x^3}{3}-\frac{x^5}{15}+\ldots+(-1)^{n+1}\frac{x^{2n+1}}{4n^2-1}$	12	$\frac{1+x^2}{2}\operatorname{arctg}x-\frac{x}{2}$
8	0,1	1	$1+\frac{2x}{1!}+\ldots+\frac{(2x)^n}{n!}$	10	e^{2x}
9	0,1	1	$1+2\frac{x}{2}+\ldots+\frac{n^2+1}{n!}\left(\frac{x}{2}\right)^n$	14	$\left(\frac{x^2}{4}+\frac{x}{2}+1\right)e^{\frac{x}{2}}$
10	0,1	0,5	$x-\frac{x^3}{3}+\ldots+(-1)^n\frac{x^{2n+1}}{2n+1}$	15	$\operatorname{arctg}x$
11	0,1	1	$1-\frac{3}{2}x^2+..+(-1)^n\frac{2n^2+1}{(2n)!}x^{2n}$	10	$\left(1-\frac{x^2}{2}\right)\cos x-\frac{x}{2}\sin x$

Страница **Additional** содержит дополнительные управляющие элементы (рис. 2). Описание некоторых компонентов страницы **Additional** приведено в таблице 2.

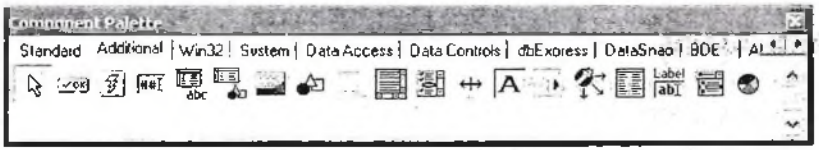


Рисунок 2 – Страница **Additional** палитры компонентов

Таблица 2 – Описание некоторых компонентов страницы **Additional**

Компонент	Описание
StringGrid	Таблица, в ячейках которой отображаются строки
Image	Отображает точечный рисунок **BMP**, значок **ICO** или метафайл **WMF**
Chart	Компонент для отображения диаграмм

Общие свойства компонентов

Многие стандартные визуальные компоненты имеют одинаковые свойства, поэтому следует рассмотреть их отдельно, чтобы впоследствии не возвращаться к этому.

Свойство Align

Задает способ выравнивания компонента внутри формы. Имеет одно из значений, представленных в таблице 1.

Свойство Color

Задает цвет фона формы или цвет компонента или графического объекта. Может иметь одно из значений, представленных в таблице 2.

Таблица 1 – Возможные значения свойства **Align**

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

Таблица 2 – Возможные значения свойства **Color**

Значение	Цвет	Значение	Цвет
clBlack	Черный (*Black*)	clSilver	Серебряный (*Silver*)
clMaroon	Темно-красный (*Maroon*)	clRed	Красный (*Red*)
clGreen	Зеленый (*Green*)	clLime	Ярко-зеленый (*Lime green*)
clOlive	Оливковый (*Olive green*)	clBlue	Голубой (*Blue*)
clNavy	Темно-синий (*Navy blue*)	clFuchsia	Сиреневый (*Fuchsia*)
clPurple	Фиолетовый (*Purple*)	clAqua	Ярко-голубой (*Aqua*)
clTeal	Сине-зеленый (*Teal*)	dWhite	Белый (*White*)
clGray	Серый (*Gray*)		

Цвета, приведенные в таблице 3, являются системными цветами Windows и зависят от используемой цветовой схемы.

Помимо перечисленных в таблице цветов значение свойства **Color** может задаваться шестнадцатеричными значениями.

Свойство Ctl3D

Позволяет задать вид компонента. Если значение этого свойства равно **false**, компонент имеет двумерный вид, если **true** – трехмерный (значение по умолчанию).

Свойство Cursor

Позволяет определить вид курсора, который он будет иметь, находясь в активной области компонента. В Delphi предопределено большое количество стандартных курсоров. Кроме того, пользователь может создавать свои собственные курсоры или использовать созданные другими.

Свойство DragCursor

Позволяет определить вид курсора, который будет отображаться, когда в компонент «перетаскивается» другой компонент. Значения этого свойства те же, что и у свойства **Cursor**.

Свойство DragMode

Позволяет определить режим поддержки протокола drag-and-drop. Возможны значения, представленные в таблице 4.

Таблица 3 – Системные цвета

Значение	Цвет	Значение	Цвет
clBackground	Текущий цвет фона окна	ciActiveBorder	Текущий цвет рамки активного окна
clActiveCaption	Текущий цвет заголовка активного окна	cllnactiveBorder	Текущий цвет рамки неактивного окна
cllnactiveCaption	Текущий цвет заголовка неактивного окна	clAppWorkSpace	Текущий цвет рабочей области окна
clMenu	Текущий цвет фона меню	clHighlight	Текущий цвет фона выделенного текста
clWindow	Текущий цвет фона Windows	clHightlightText	Текущий цвет выделенного текста
clWlndowFrame	Текущий цвет рамки окна	clBtnFace	Текущий цвет кнопки
clMenuText	Текущий цвет текста элемента меню	clBtnShadow	Текущий цвет фона кнопки
clWindowText	Текущий цвет текста внутри окна	clGrayText	Текущий цвет недоступного элемента меню
clCaptionText	Текущий цвет заголовка активного окна	clBtnText	Текущий цвет текста кнопки

Таблица 4 – Возможные значения свойства **DragMode**

Значение	Описание
dmAutomatic	Компонент можно «перетаскивать», «зацепив» мышью
dmManual	Компонент не может быть «перетащен» без вызова метода **BeginDrag**

Свойство Enabled

Если это свойство имеет значение **true**, компонент реагирует на сообщения от мыши, клавиатуры и таймера. В противном случае (значение **false**) эти сообщения игнорируются.

Свойство Enabled

Если это свойство имеет значение **true**, компонент реагирует на сообщения от мыши, клавиатуры и таймера. В противном случае (значение **false**) эти сообщения игнорируются.

Свойство Font

Многие визуальные компоненты используют шрифт по умолчанию. При создании компонента параметром свойства **Font** (класс TFont) присваиваются значения, представленные в таблице 5.

Свойство Height

Задает вертикальный размер компонента или формы.

Свойство HelpContext

Задает номер контекста справочной системы. Этот номер должен быть уникальным для каждого компонента. Если компонент активен (находится в фокусе), нажатие клавиши **F1** приводит к отображению экрана справочной системы (если такой существует для данного компонента).

Свойство Hint

Задает текст, который будет отображаться при обработке события **OnHint**, происходящего, если курсор находится в области компонента.

Таблица 5 – Значения свойств объекта **Font**, присваиваемые

Свойство	Значение
Color	clWindowText
Height	- MulDiv(10, GetDeviceCaps(DC, LOGPIXELSY), 72)
Name	System
Pitch	FpDefault
Size	10
Style	[]

Свойство Left

Задает горизонтальную координату левого угла компонента относительно формы в пикселях. Для форм это значение указывается относительно экрана.

Свойство ParentColor

Позволяет указать, каким цветом будет отображаться компонент. Если значение этого свойства равно **true**, компонент использует цвет (значение свойства **Color**) родительского компонента. Если же значение свойства **ParentColor** равно **false**, компонент использует значение собственного свойства **Color**.

Свойство ParentCtl3D

Это свойство позволяет указать, каким образом компонент будет определять, является он трехмерным или нет. Если значение этого свойства равно **true**, то вид компонента задается значением свойства **Ctl3D** его владельца, если же значение этого свойства равно **false** – то значением его собственного свойства **Ctl3D**.

Свойство ParentFont

Позволяет указать, каким образом компонент будет определять используемый им шрифт. Если значение этого свойства равно **true**, используется шрифт, заданный у владельца компонента, если же это значение равно **false**, то шрифт задается значением его собственного свойства **Font**.

Свойство PopupMenu

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

Свойство TabOrder

Задает порядок получения компонентами фокуса при нажатии клавиши **Tab**. По умолчанию этот порядок определяется размещением компонентов в форме: первый компонент имеет значение этого свойства, равное 0, второй – 1 и т. д. Для изменения этого порядка необходимо изменить значение свойства **TabOrder** определенного компонента. **TabOrder** может использоваться только совместно со свойством **TabStop**.

Свойство TabStop

Позволяет указать, может компонент получать фокус или нет. Компонент получает фокус, если значение его свойства **TabStop** равно **true**.

Свойство Tag

С помощью этого свойства можно «привязать» к любому компоненту значение типа **LongInt**.

Свойство Top

Задает вертикальную координату левого верхнего угла интерфейсного элемента относительно формы в пикселах. Для формы это значение указывается относительно экрана.

Свойство Visible

Позволяет определить, видим ли компонент на экране. Значением этого свойства управляют методы **Show** и **Hide**.

Свойство Width

Задает горизонтальный размер интерфейсного элемента или формы в пикселях.

Основные компоненты и их свойства

Рисунок 1 – **Label**

Компоненты класса **TLabel** (метки) (рис. 1) предназначены для размещения на форме различного рода текстовых надписей. Основные свойства класса **TLabel** представлены в таблице 1.

Таблица 1 – Основные свойства класса **TLabel**

Свойство	Описание
AutoSize	Указывает, будет ли метка изменять свои размеры в зависимости от помещенного в ее свойство **Caption** текста (**true** – будет)
FocusControl	Содержит имя оконного компонента, который связан с меткой (выбор компонента **Label** приводит к перемещению фокуса на связанный с ним компонент)
Layout	Определяет выравнивание текста по вертикали относительно границ метки: **tlTop** – текст располагается вверху, **tlCenter** –центрируется по вертикали, **tlBottom** – располагается внизу
ShowAccelChar	Если содержит **true**, символ **&** в тексте метки предшествует символу-акселератору
Transparent	Определяет прозрачность фона метки. Если **false**, фон закрашивается собственным цветом **Color**, в противном случае используется фон родительского компонента
WordWrap	Разрешает/запрещает разрыв строки на границе слова. Для вывода многострочных надписей задайте **AuroSize=False**, **WordWrap=True** и установите подходящие размеры метки

Рисунок 2 **Edit**

Компонент класса **TEdit** (рис. 2) представляет собой однострочный редактор текста. Основные свойства класса **TEdit** представлены в таблице 2, а основные методы – в таблице 3.

Таблица 2 – Основные свойства класса **TEdit**

Свойство	Описание
AutoSelect	Указывает, будет ли выделяться весь текст в момент получения компонентом фокуса ввода
AutoSize	Если **true** и **BorderStyle** = **bsSingle**, высота компонента автоматически меняется при изменении свойства **Font**. **Size**
BorderStyle	Определяет стиль обрамления компонента: **bsNone** – нет обрамления, **bsSingle** – компонент обрамляется одной линией

Окончание табл. 2.

Свойство	Описание
CharCase	Определяет автоматическое преобразование высоты букв: **ecNormal** – нет преобразования, ее **UpperCase** – все буквы заглавные; **ecLowerCase** – все буквы строчные. Правильно работает с кириллицей
HideSelection	Если **false**, выделение текста сохраняется при потере фокуса ввода
MaxLength	Определяет максимальную длину текстовой строки. Если имеет значение 0, длина строки не ограничена
Modified	Содержит **true**, если текст был изменен
OEMConvert	Содержит **true**, если необходимо перекодировать текст из кодировки **MS-DOS** в кодировку Windows и обратно
PasswordChar	Если символ **PasswordChar** определен, он заменяет собой любой символ текста при отображении в окне. Используется для ввода паролей
ReadOnly	Если содержит **true**, текст не может изменяться
SelLength	Содержит длину выделенной части текста
SelStart	Содержит номер первого символа выделенной части текста
SelText	Содержит выделенный текст

Таблица 3 – Основные методы

Метод	Описание
procedure Clear;	Удаляет весь текст
procedure ClearSelection;	Удаляет выделенный текст
procedure CopyToClipboard;	Копирует выделенный текст в **Clipboard**
procedure CutToClipboard;	Копирует выделенный текст в **Clipboard**, после чего удаляет выделенный текст из компонента
function GetSelTextBuf (Buffer: PChar; BufSize: Integer) : Integer;	Копирует не более **BufSize** символов выделенного текста в буфер **Buffer**
procedure PasteFromClipboard;	Заменяет выделенный текст содержимым **Clipboard**, а если нет выделенного текста, копирует содержимое **Clipboard** в позицию текстового курсора
procedure SelectAll;	Выделяет весь текст

Компоненты класса **TMemo** (рис. 3) предназначены для ввода, редактирования и (или) отображения достаточно длинного текста, содержащего большое количество строк. Большинство свойств этого компонента аналогичны свойствам класса **TEdit**. Основные свойства класса **TMemo** представлены в таблице 4.

Рисунок 3 – **Memo**

Компонент **TButton** представляет собой стандартную кнопку и широко используется для управления программами. Кнопка может содержать текст, описывающий выполняемое ею действие. Основные свойства класса **TButton** представлены в таблице 5.

Рисунок 4 – **Button**

Компоненты класса **TMemo** (рис. 5) предназначены для ввода, редактирования и (или) отображения достаточно длинного текста, содержащего большое количество строк. Большинство свойств этого компонента аналогичны свойствам класса **TEdit**. Основные свойства класса **TMemo** представлены в таблице 6 (см. с. 72).

Рисунок 5 – **CheckBox**

Таблица 4 – Основные свойства класса **TMemo**

Свойство	Описание
Lines: TStrings;	Содержит редактируемый текст. Используется для построчного доступа. Методы **Add**, **Delete**, **Insert** используются для добавления, удаления и вставки строк
ScrollBars	Определяет наличие в окне редактора полос прокрутки: **ssNone** – нет полос, **ssHorizontal** – есть горизонтальная полоса, **ssVertical** – есть вертикальная полоса, **ssBoth** – есть обе полосы
Text	Отображает содержимое свойства **Lines** в виде одной длинной строки, в которой границы отдельных строк определяются символами **EOL(CR+LF)**
WantReturns	Если содержит **true**, нажатие **Enter** вызывает переход на новую строку, в противном случае – обрабатывается системой. Для перехода на новую строку в этом случае следует нажать **Ctrl+Enter**
WantTabs	Если содержит **true**, нажатие **Tab** вызывает ввод в текст символа табуляции, в противном случае – обрабатывается системой. Для ввода символа табуляции в этом случае следует нажать **Ctrl+Tab**
WordWrap	Если равно **true**, то работает перенос слов

Таблица 5 – Основные свойства

Свойство	Описание
Cancel	Если имеет значение **true**, событие OnClick кнопки возникает при нажатии клавиши **Esc**
Default	Если имеет значение **true**, событие OnClick кнопки возникает при нажатии клавиши **Enter**
Enabled	Если имеет значение **false**, то кнопка недоступна для нажатия
ModalResult	Определяет результат, с которым было закрыто модальное окно

Примечание. В терминологии Windows модальными окнами называются такие специальные окна, которые, хоть раз появившись на экране, блокируют работу пользователя с другими окнами вплоть до своего закрытия. Если у кнопки определено свойство **ModalResult**, нажатие на нее приводит к закрытию модального окна и возвращает в программу значение **ModalResult** как результат диалога с пользователем. В Delphi определены следующие стандартные значения **ModalResult**:

- mrNone — Модальное окно не закрывается
- mrOK — Была нажата кнопка **OK**
- mrCancel — Была нажата кнопка **Cancel**
- mrAbort — Была нажата кнопка **Abort**
- mrRetry — Была нажата кнопка **Retry**
- mrIgnore — Была нажата кнопка **Ignore**
- mrYes — Была нажата кнопка **Yes**
- mrNo — Была нажата кнопка **No**
- mrAll — Была нажата кнопка **All**

Рисунок 6 – RadioGroup

Компонент класса **TRadioGroup** (рис. 6) представляет собой специальный контейнер, предназначенный для размещения зависимых переключателей класса **TRadioButton**. Каждый размещаемый в нем переключатель помещается в специальный список **Items** и доступен по индексу, что упрощает обслуживание группы. Основные свойства класса **TRadioGroup** представлены в таблице 7.

Рисунок – 7 **StringGrid**

Компонент класса **TStringGrid** (рис. 7) представляет собой разлинеенную таблицу, содержащую текстовую информацию. Основные свойства класса **TStringGrid** представлены в таблице 8.

Таблица 6 – Основные свойства класса **TMemo**

Свойство	Описание
Alignment	Определяет положение текста: **taLeftJustify** – с левой стороны компонента, **taRightJustify** – с правой стороны
AllowGrayed	Разрешает/запрещает использование неактивного состояния **cbGrayed**
Checked	Содержит выбор пользователя типа да/нет. Состояния **cbUnchecked** и **cbGrayed** отражаются как **false**
State	Содержит состояние компонента: **cbUnchecked** – нет, **cbChecked** – да, **cbGrayed** – неактивен

Таблица 7 – Основные свойства класса **TRadioGroup**

Свойство	Описание
Columns	Определяет количество столбцов-переключателей
ItemIndex	Содержит индекс выбранного переключателя
Items	Содержит список строк с заголовками элементов. Добавление/удаление элементов достигается добавлением/удалением строк списка **Items**

Таблица 8 – Основные свойства **TStringGrid**

Свойство	Описание
BorderStyle	Определяет, имеет ли таблица рамку
Cells[ACol,ARow]	Определяет содержимое ячейки с табличными координатами (**ACol**, **ARow**)
Col	Содержит номер колонки с ячейкой, имеющей фокус ввода
ColCount	Количество столбцов в таблице
DefaultColWidth	Стандартная ширина столбцов таблицы
DefaultDrawing	Определяет, отрисовываются ли ячейки таблицы автоматически
DefaultRowHeight	Стандартная высота строк таблицы
FixedColor	Цвет непрокручиваемых строк и столбцов таблицы
FixedCols	Количество зафиксированных столбцов в таблице
FixedRows	Количество зафиксированных строк в таблице
GridLineWidth	Толщина линий между ячейками таблицы
+Options	Множество флагов, задающих различные режимы работы и способы отображения таблицы
Row	Содержит номер ряда ячейки, имеющей фокус ввода
RowCount	Количество строк в таблице
ScrollBars	Управляет полосами прокрутки в таблице

Простые типы данных языка Object Pascal

Целые типы

Диапазон возможных значений целых типов зависит от их внутреннего представления, которое может занимать 1, 2 или 4 байта (табл. 1).

К целочисленным типам применимы процедуры и функции, описанные в таблице 2.

Таблица 1 – Целые типы

Название	Длина, байт	Диапазон значений
Byte	1	0...255
Shortint	1	–128...+127
Smallint	2	–32 768...+32 767
Word	2	0...65 535
Integer	4	–2 147 483 648...+2 147 483 647
Longint	4	–2 147 483 648...+2 147 483 647
Cardinal	4	0...2 147 483 647

Таблица 2 – Процедуры и функции, применяемые к аргументам целого типа

Обращение	Тип результата	Действие
abs *(x)*	*x*	Возвращает модуль *x*
chr (Byte)	Char	Возвращает символ по его коду
dec*(x[,i])*	–	Уменьшает значение *x* на *i*, а при отсутствии – *i* на 1
inc*(x[,i])*	–	Увеличивает значение *v* на *i*, а при отсутствии – *i* на 1
Hi(word)	Byte	Возвращает старший байт аргумента
Hi(integer)	Byte	Возвращает третий по счету байт
Lo(integer)	Byte	Возвращает младший байт аргумента
Lo (word)	Byte	Возвращает младший байт аргумента
Odd(LongInt)	Boolean	Возвращает **true**, если аргумент – нечетное число
Random(word)		Возвращает псевдослучайное число, равномерно распределенное в диапазоне 0...(word)

Вещественные типы

Значения вещественных типов определяют произвольное число лишь с некоторой конечной точностью, зависящей от внутреннего формата вещественного числа (табл. 3).

Для работы с вещественными типами имеются стандартные функции (табл. 4).

Таблица 3 – Вещественные типы

Название	Длина, байт	Кол-во значащих цифр	Диапазон значений	Примечание
Real	6	11...12	2,9*10–39...1,7*1039	При наличии сопроцессора использовать нежелательно, так как замедляет работу
Single	4	7. . .8	1,5*10–45…3,4*1038	–
Double	8	15...16	5,0*10–324...1,7*10308	–
Extended	10	19...20	3,4*10–4951...1,1*104932	Применяется наиболее часто
Comp	8	19...20	–263...+263–1	Дробная часть отсутствует
Currency	8	19...20	±922337203685477,5807	Длина дробной части – 4 десятичных разряда

Таблица 4 – Функции для работы с аргументами вещественного типа

Обращение	Тип параметра	Тип результата	Примечание
abs*(x)*	Вещественный	Вещественный	Модуль аргумента
ArcTan*(x)*	Вещественный	Вещественный	Арктангенс (в радианах)
Cos*(x)*	Вещественный	Вещественный	Косинус (в радианах)
Exp*(x)*	Вещественный	Вещественный	Экспонента
Frac*(x)*	Вещественный	Вещественный	Дробная часть числа
Int*(x)*	Вещественный	Вещественный	Целая часть числа
Ln*(x)*	Вещественный	Вещественный	Логарифм натуральный
Pi	–	Вещественный	π =3.141592653...
Random	–	Вещественный	Псевдослучайное число, равномерно распределенное в диапазоне 0...[1]

Окончание табл. 4.

Обращение	Тип параметра	Тип результата	Примечание
Randomize			Инициация генератора псевдослучайных чисел
Sin *(x)*	Вещественный	Вещественный	Синус (в радианах)
Sqr*(x)*	Вещественный	Вещественный	Квадрат аргумента
Sqrt*(x)*	Вещественный	Вещественный	Корень квадратный

Логические типы

К логическим относятся типы **Boolean**, **ByteBool**, **Bool**, **WordBool** и **LongBool**. В стандартном Pascal определен только тип **Boolean**, остальные логические типы введены в Object Pascal для совместимости с Windows: типы **Boolean** и **ByteBool** занимают по 1 байту каждый, **Bool** и **WordBool** – по 2 байта, **LongBool** – 4 байта. Значениями логического типа может быть одна из предварительно объявленных констант: **false** или **true**.

Символьный тип

Значением символьного типа является множество всех символов, каждому из которых приписывается целое число в диапазоне 0...255, служащее кодом внутреннего представления символа. Его возвращает функция **ord**.

Для кодировки в Windows используется код. Первая половина символов ПК с кодами 0...127 постоянна и содержит в себе служебные коды и латинский алфавит. Вторая половина символов с кодами 128...255 меняется для различных шрифтов. Символы с кодами 0...31 относятся к служебным кодам. Если эти коды используются в символьном тексте программы, они считаются пробелами.

К типу **Char** применимы операции отношения, а также встроенные функции:

- **Chr (B)** – функция типа **Char** преобразует выражение **B** типа **Byte** в символ и возвращает его своим значением;
- **UpCase (CH)** – функция типа **Char** возвращает прописную букву, если **CH** – строчная латинская буква, в противном случае возвращает сам символ **CH** (для кириллицы возвращает исходный символ).

Тип дата-время

Тип дата-время определяется идентификатором **TDateTime** и предназначен для одновременного хранения и даты, и времени. Над данными типа **TDateTime** определены те же операции, что и над вещественными числами, а в выражениях этого типа могут участвовать константы и переменные целого и вещественного типов.

Математические формулы

Язык Object Pascal имеет ограниченное количество встроенных математических функций, поэтому при необходимости использовать другие функции следует применять известные соотношения. В таблице приведены выражения наиболее часто встречающихся функций через встроенные функции языка Object Pascal.

Таблица – Формулы для вычисления некоторых функций

Функция	Соотношение	Соотношение на языке Object Pascal
$\mathrm{Log}_a(x)$	$\frac{\mathrm{Ln}(x)}{\mathrm{Ln}(a)}$	Ln(x)/Ln(a)
x^a	$e^{a*Ln(x)}$	Exp(a*Ln(x))
Tg(x)	$\frac{\mathrm{Sin}(x)}{\mathrm{Cos}(x)}$	Sin(x)/Cos(x)
Ctg(x)	$\frac{\mathrm{Cos}(x)}{\mathrm{Sin}(x)}$	Cos(x)/Sin(x)
ArcSin(x)	$\mathrm{ArcTg}\left(\sqrt{\frac{x}{1-x^2}}\right)$	ArcTan(Sqrt(x/(1 – sqr(x))))
ArcCos(x)	$\frac{\pi}{2} - \mathrm{ArcSin}(x)$	Pi/2- ArcTan(Sqrt(x/(1-sqr(x))))
ArcCtg(x)	$\frac{\pi}{2} - \mathrm{ArcTg}(x)$	Pi/2 – ArcTan(x)
Sh(x)	$\frac{e^x - e^{-x}}{2}$	(Exp(x) – Exp(–x))/2
Ch(x)	$\frac{e^x + e^{-x}}{2}$	(Exp(x)+Exp(–x))/2
Csc(x)	$\frac{1}{\sin(x)}$	1/Sin(x)
Sc(x)	$\frac{1}{\cos(x)}$	1/Cos(x)

Процедуры и функции работы со строками

В таблице 1 представлены процедуры и функции для работы со строками.

В таблице 2 представлены подпрограммы преобразования строк в другие типы.

Таблица 1 – Процедуры и функции для работы со строками

Процедуры и функции	Описание
Procedure **Delete**(St: String; Index, Count: Integers;	Удаляет Count символов из строки **St** начиная с символа с номером Index
Procedure **Insert**(SubSt: String; St, Index: Integer) ;	Вставляет подстроку **SubSt** в строку **St** начиная с символа с номером Index
Procedure **SetLength**(St: String; NewLength: Integer);	Устанавливает новую (меньшую) длину NewLength строки **St**, если NewLength больше текущей длины строки, обращение к SetLength игнорируется
Function **Concat**(S1 [, S2, ..., SN]: String): String;	Возвращает строку, представляющую собой сцепление строк-параметров S1, S2, … , SN
Function **Copy**(St: String; Index, Count: Integer): String;	Копирует из строки **St** Count символов, начиная с символа с номером Index
Function **Length**(St: String): Integer;	Возвращает текущую длину строки **St**
Function **Pos**(SubSt, St: String): Integer;	Отыскивает в строке **St** первое вхождение подстроки SubSt и возвращает номер позиции, с которой она начинается. Если подстрока не найдена, возвращается ноль

Таблица 2 – Подпрограммы преобразования строк в другие типы

Процедуры и функции	Описание
Function **StrToCurr**(St: String): Currency;	Преобразует символы строки **St** в целое число типа **Currency**. Строка не должна содержать ведущих или ведомых пробелов
Function **StrToDate**(St: String): TDateTime;	Преобразует символы строки **St** в дату. Строка должна содержать два или три числа, разделенных правильным для Windows разделителем даты (в русифицированной версии таким разделителем является «.») Первое число – день, второе – месяц, если указано третье число, оно задает год
Function **StrToDateTime**(St: String): TDateTime;	Преобразует символы строки St в дату и время. Строка должна содержать дату и время, разделенные пробелом

Окончание табл. 2.

Процедуры и функции	Описание
Function **StrToFloat**(St: String): Extended;	Преобразует символы строки **St** в вещественное число. Строка не должна содержать ведущих или ведомых пробелов
Function **StrToInt**(St: String): Integer;	Преобразует символы строки **St** в целое число. Строка не должна содержать ведущих или ведомых пробелов
Function **StrToIntDef**(St: String; Default: Integer): Integer;	Преобразует символы строки **St** в целое число. Если строка не содержит правильного представления целого числа, возвращается значение Default
Function **StrToIntRange**(St: String; Min, Max: Longint) : Longint;	Преобразует символы строки **St** в целое число и возбуждает исключение ERangeError, если число выходит из заданного диапазона Min Max
Function **StrToTime**(St: String): TDateTime;	Преобразует символы строки **St** во время
Procedure **Val**(St: String; var X; Code: Integer);	Преобразует строку символов **St** во внутреннее представление целой или вещественной переменной X, которое определяется типом этой переменной. Параметр Code содержит ноль, если преобразование прошло успешно, и тогда в X помещается результат преобразования; в противном случае он содержит номер позиции в строке **St**, где обнаружен ошибочный символ, и в этом случае содержимое X не меняется. В строке **St** могут быть ведущие и (или) ведомые пробелы

В таблице 3 представлены процедуры преобразования данных некоторых типов в строки.

Таблица 3 – Подпрограммы обратного преобразования

Процедуры и функции	Описание
Function **DateToStr**(Value: TDateTime): String;	Преобразует дату из параметра Value в строку символов
Function **DateTimeToStr**(Value: TDateTime): String;	Преобразует дату и время из параметра Value в строку символов
Procedure **DateTimeToString** (var St: String; Format: String; Value: TDataTime);	Преобразует дату и время из параметра Value в строку **St**
Function **FormatDateTime** (Format: String; Value: TDateTime): String;	Преобразует дату и время из параметра Value в строку символов
Function **FloatToStr**(Value: Extended): String;	Преобразует вещественное значение Value в строку символов

Окончание табл. 3.

Процедуры и функции	Описание
Function **FloatToStr**F(Value: Extended; Format: **T**FloatFormat; Precision, Digits: Integer) : String;	Преобразует вещественное значение Value в строку символов с учетом параметров Precision и Digits (см. пояснения ниже)
Function **FormatFloat**(Format: String; Value: Extended): String;	Преобразует вещественное значение Value в строку
Function **IntToStr**(Value: Integer) : String;	Преобразует целое значение Value в строку символов
Function **TimeToStr**(Value: **T**Date**T**ime): String;	Преобразует время из параметра Value в строку символов
Procedure **Str**(X [:width [:Decimals]]; var St: String);	Преобразует число X любого вещественного или целого типа в строку символов **St**; параметры Width и Decimals, если они присутствуют, задают формат преобразования: Width определяет общую ширину поля, выделенного под соответствующее символьное представление вещественного или целого числа X, а Decimals – количество символов в дробной части (этот параметр имеет смысл только в том случае, когда X – вещественное число)

Процедуры и функции работы с файлами

В таблице приведены процедуры и функции для работы с файлами

Таблица – Процедуры и функции для работы с файлами

Процедуры и функции	Описание
Procedure **AssignFile**(var F; FileName: string)	Связывает файловую переменную F и файл с именем FileName
Procedure **Reset**(var F[: File; RecSize: word])	Открывает существующий файл. При открытии нетипизированного файла RecSize задает размер элемента файла
Procedure **Rewrite**(var F[: File; RecSize: word])	Создает и открывает новый файл
Procedure **Append**(var F: TextFile)	Открывает текстовой файл для дописывания текста в конец файла
Procedure **Read**(F,v1[,v2,... vn])	Чтение значений переменных начиная с текущей позиции для типизированных файлов и строк для текстовых
Procedure **Write**(F,v1[,v2,... vn])	Запись значений переменных начиная с текущей позиции для типизированных файлов и строк для текстовых
Procedure **CloseFile**(F)	Закрывает ранее открытый файл
Procedure **Rename**(var F; NewName: string)	Переименовывает неоткрытый файл любого типа
Procedure **Erase**(var F)	Удаляет неоткрытый файл любого типа
Procedure **Seek**(var F; NumRec: Longint)	Для нетекстового файла устанавливает указатель на элемент с номером NumRec
Procedure **SetTextBuf**(var F: TextFile; var Buf[;Size: word])	Для текстового файла устанавливает новый буфер ввода-вывода объема Size
Procedure **Flush**(var F: TextFile)	Немедленная запись в файл содержимого буфера ввода-вывода
Procedure **Truncate**(var F)	Урезает файл, начиная с текущей позиции
Function **LoResult**: integer	Код результата последней операции ввода-вывода
Function **FilePos**(var F): longint	Для нетекстовых файлов возвращает номер текущей позиции. Отсчет ведется от нуля
Function **FileSize**(var F): longint	Для нетекстовых файлов возвращает количество компонентов в файле
Function **Eoln**(var F: TextFile): boolean	Возвращает **true**, если достигнут конец строки
Function **Eof**(var F)): boolean	Возвращает **true**, если достигнут конец файла
Function **SeekEoln**(var F: TextFile): boolean	Возвращает **true**, если пройден последний значимый символ в строке или файле, отличный от пробела или знака табуляции
Function **SeekEof**(var F: TextFile): boolean	То же, что и SeekEoln, но для всего файла

СОДЕРЖАНИЕ

Учебное издание

ИНФОРМАТИКА

Методические указания и задания к лабораторным работам для студентов II курса специальностей 1-40 01 02 Информационные системы и технологии, 1-36 01 03 Технологическое оборудование машиностроительного производства, 1-36 01 01 Технология машиностроения, 1-53 01 01 Автоматизация технологических процессов и производств инженерного факультета
Часть 2

Технический редактор *Л. Н. Щербук*
Компьютерная верстка *Е. А. Гречушкиной, А. В. Гутырчика*

Ответственный за выпуск *Е. Г. Хохол*

Подписано в печать 25.05.2007.
Формат 60x84 1/16. Бумага офсетная. Гарнитура Таймс.
Отпечатано на ризографе. Усл. печ. л. 4,9. Уч.-изд. л. 4,1.
Заказ 7. Тираж 120 экз.

ЛИ 02330/0133468 от 09.02.2005

Издатель и полиграфическое исполнение:
учреждение образования
«Барановичский государственный университет»
225404 г. Барановичи, ул. Войкова, 21