Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра вычислительных методов и программирования

А. А. Бурцев, А. А. Навроцкий, В. П. Шестакович

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ OBJECT PASCAL В СРЕДЕ DELPHI

Лабораторный практикум по курсам
«Программирование» и «Основы алгоритмизации и программирования»
для студентов всех специальностей заочной формы обучения

В 2-х частях

Часть 1

Минск 2006

УДК 681.3.06 (075.8)
ББК 32.973-018 я73
Б 35

Бурцев А. А.

Б 35 Основы программирования на языке Object Pascal в среде DELPHI: Лаб. практикум по курсам «Программирование» и «Основы алгоритмизации и программирования» для студ. всех спец. заоч. формы обуч.: В 2 ч. Ч. 1. / А. А. Бурцев, А. А. Навроцкий, В. П. Шестакович. – Мн.: БГУИР, 2006. – 50 с.: ил.

ISBN 985-444-971-8 (ч. 1)

В лабораторном практикуме даны краткие теоретические сведения по основам программирования на языке Object Pascal в среде DELPHI, рассмотрены простейшие алгоритмы. После каждой темы приведен набор индивидуальных заданий.

УДК 681.3.06 (075.8)
ББК 32.973-018 я 73

ISBN 985-444-971-8 (ч. 1)
ISBN 985-444-970-X

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА 1.
ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ

Цель работы: Изучить основы языка Object Pascal. Научиться составлять программы в среде DELPHI.

1.1. Базовые элементы языка Object Pascal

1.1.1. Алфавит языка

Язык Object Pascal оперирует следующим набором символов:

1. *Прописные и строчные буквы латинского алфавита* (A, B, C, ... , X, Y, Z, a, b, c, ..., x, y, z).
2. *Десятичные цифры* (0 … 9).
3. *Символ «подчеркивание»*(«_»).
4. *Специальные символы* («+», «-», «/», «*», «**{**», «**}**», «**:**», «**[**», «**]**», «**;**», «**(**», «**)**», «'», «#», «**@**», «=», « », «**$**», «>», «**.**», «^», «**<**», «**,**», а также их комбинации).
5. *Ключевые слова* (например: «Begin», «End»);
6. *Стандартные идентификаторы* (Например: «Sin», «Cos»).
7. *Идентификаторы пользователя*.

1.1.2. Синтаксис языка

Основные правила написания программ:

1. Прописные и строчные буквы компилятором не различаются (описание «mas» идентично написанию «MAS»).
2. Идентификаторы (имена) могут начинаться только с буквы и символа подчеркивания, и должны содержать буквы, цифры и символ подчеркивания. Компилятор воспринимает идентификаторы длиной не более 255 символов.
3. Каждое предложение заканчивается точкой с запятой («;»). В одной строке могут находиться несколько предложений, и наоборот, одно большое предложение может быть разбито на несколько строк.
4. Все данные, процедуры и функции должны быть объявлены в разделе объявлений до первого их использования.
5. Для повышения читабельности текста можно использовать комментарии (допускается использования русского языка). Комментарием является текст, заключенный в фигурные скобки ({ комментарий }), круглые с символом звездочка ((* комментарий *)) либо размещенные после двух символов «правый слэш» (// комментарий).

1.1.3. Основные типы данных

Тип данных определяет количество выделяемых ячеек памяти и перечень допустимых операций. Существует две основные группы типов данных: *скалярные* (простые) и *структурированные* (составные). Данные скалярного типа представляют собой одно значение, размещенное в одной или нескольких ячейках памяти. Структурированные данные (объявляются после ключевого слова

type) представляют собой объединение нескольких однотипных или неоднотипных данных скалярного типа. Все данные могут быть либо константами (объявляются после ключевого слова *const*), либо переменными (объявляются после ключевого слова *var*). Значение переменных может быть изменено в процессе выполнения программы, а значение констант – не может.

1.1.4. Оператор присваивания

Оператор присваивания записывается в виде:

<имя переменной>:=< выражение соответствующего типа>;

Переменная может быть любого типа, а выражение должно иметь тип, соответствующий типу переменной (разрешается присваивать действительной переменной значение, имеющее целый тип).

1.1.5. Целые типы данных

Используются для представления целых чисел. Характеристики основных целых типов данных приведены в табл. 1.1.

Таблица 1.1

Тип	Диапазон значений	Требуемая память, байт
Byte	0 .. 255	1
Word	0 .. 65535	2
Cardinal	0 .. 4294967295	4
Integer	-2147483648 .. 2147483647	4

Операции над целыми числами:

Наименование операции	Обозначение	Пример
Сложение	+	5 + 2 = 7
Вычитание	-	5 – 2 = 3
Умножение	*	5 * 2 = 10
Целочисленное деление	div	5 div 2 = 2
Остаток от целочисленного деления	mod	5 mod 2 = 1
Логическое равно	=	5 = 5 , результат: истина
Логическое не равно	<>	5 <> 5 , результат: ложь
Сравнение	>, >=, <, <=	

1.1.6. Действительные типы данных

Используются для представления чисел, имеющих дробную часть. Характеристики основных действительных типов данных приведены в табл. 1.2.

Таблица 1.2

Тип	Диапазон значений	Требуемая память, байт
Real	$\pm 5.0*10^{-324}$ .. $\pm 1.7*10^{308}$	8
Single	$\pm 1.5*10^{-45}$ .. $\pm 3.4*10^{38}$	4
Extended	$\pm 3.6*10^{-4932}$ .. $\pm 1.1*10^{4392}$	10

Операции над действительными числами:

Наименование операции	Обозначение	Пример
Сложение	+	5 + 1.5 = 6.5
Вычитание	-	8 – 4.2 = 3.8
Умножение	*	2 * 2,6 = 5.2
Деление	/	5 / 2 = 2.5
Логическое равно	=	5 = 5 , результат: истина
Логическое не равно	<>	5 <> 5 , результат: ложь
Сравнение	>, >=, <, <=	

1.1.7. Булевы типы данных

Используются для представления логических значений. Наиболее часто используется тип **Boolean**, который занимает 1 байт памяти и может принимать два значения: **true** (истина) или **false** (ложь). Для булевых типов данных определены логические операции **and**, **or**, **not**, **xor**. Например, если переменные b1 и b2 имеют тип Boolean, то

```
b1 := 5 > 3;   // результат: b1=true
b2 := 5 = 3;   // результат: b2=false
b1 := not b2;   // результат: b1=true
b2 := b1 and b2;   // результат: b2=true
```

1.1.8. Символьные типы данных

Используются для хранения одного символа. Наиболее часто используется тип **Char**, который занимает 1 байт памяти.

1.1.9. Арифметические вычисления

Арифметические выражения строятся из числовых констант, переменных, стандартных функций и операций над ними. В арифметическом выражении принят следующий приоритет операций:

1) вычисление значений стандартных функций;
2) умножение и деление;
3) сложение и вычитание.

Порядок выполнения операций изменяется с помощью скобок.

Для проведения арифметических вычислений используются следующие процедуры и функции (табл. 1.3).

Таблица 1.3

Описание	Обозначение	Тип аргумента	Тип значения	Модуль
1	2	3	4	5
Модуль (абсолютное значение)	Abs(x)	Целый или вещественный	Соответствует типу аргумента	System
Экспонента	Exp(x)	Вещественный	Вещественный	System

Дробная часть	Frac(x)	Вещественный	Вещественный	System
Целая часть	Int(x)	Вещественный	Вещественный	System
Натуральный логарифм	ln(x)	Вещественный	Вещественный	System
Десятичный логарифм	Log10(x)	Вещественный	Вещественный	Math
Логарифм по основанию 2	Log2(x)	Вещественный	Вещественный	Math
Логарифм по основанию N	LogN(N, x)	Вещественный	Вещественный	Math
Максимум двух чисел	Max(A, B)	Вещественный	Вещественный	Math
Минимум двух чисел	Min(A, B)	Вещественный	Вещественный	Math
Число π	PI			System
Возведение числа A в степень E	Power(A,E)	Вещественный	Вещественный	Math
Округление до ближайшего целого	Round(x)	Вещественный	Целый	System
Квадрат	Sqr(x)	Вещественный	Вещественный	System
Корень квадратный	Sqrt(x)	Вещественный	Вещественный	System
Возврат целой части	Trunc(x)	Вещественный	Целый	System
Арккосинус	ArcCos(x)	Вещественный	Вещественный	Math
Арксинус	ArcSin(x)	Вещественный	Вещественный	Math
Арстангенс	ArcTan(x)	Вещественный	Вещественный	System
Косинус	Cos(x)	Вещественный	Вещественный	System
Синус	Sin(x)	Вещественный	Вещественный	System
Тагенс	Tan(x)	Вещественный	Вещественный	Math
Косинус гиперболический	Cosh(x)	Вещественный	Вещественный	Math
Синус гиперболический	Sinh(x)	Вещественный	Вещественный	Math
Тангенс гиперболический	Tanh(x)	Вещественный	Вещественный	Math

Окончание табл. 1.3

1	2	3	4	5
Генерация псев-дослучайного числа из диапазо-на 0 ..1	Random	Вещественный	Вещественный	System

Генерация псевдослучайного числа из диапазона 0 ..R	Random(R)	Целый	Целый	System
Делает случайной генерацию псевдослучайного числа	Randomize			System
Возвращает True (истина), если x – нечетное и False (ложь), если x – четное	Odd(x)	Целый	Логический	System
Возвращает следующее за x значение	Succ(x)	Целый, символьный, перечисляемый	Соответствует типу аргумента	System
Возвращает предыдущее x значение	Pred(x)	Целый, символьный, перечисляемый	Соответствует типу аргумента	System
Возвращает символ ASCII кода x	Chr(x)	Целый	Символьный	System
Возвращает ASCII код символа x	Ord(x)	Символьный	Целый	System
Увеличивает значение x на 1	Inc(x)	Целый	Целый	System
Увеличивает значение x на n	Inc(x, n)	Целый	Целый	System
Уменьшает значение x на 1	Dec(x)	Целый	Целый	System
Уменьшает значение x на n	Dec(x, n)	Целый	Целый	System

Примечание: В тригонометрических функциях аргумент задается только в радианах. Математические функции, находящиеся в модуле *Math* могут быть использованы только после добавления его имени в раздел подключаемых модулей (**uses**).

1.2. Интегрированная среда разработчика DELPHI

Интегрированная среда разработчика представляет собой набор окон, содержащих все необходимые инструменты для проектирования, тестирования и запуска приложения. Количество, расположение, размер и вид окон может меняться программистом в зависимости от его текущих нужд. Кроме того, имеется возможность расширять среду, включая инструменты, поставляемые сторон-

ними разработчиками, что значительно повышает производительность работы. Основное окно среды Delphi имеет вид как на рис. 1.1.

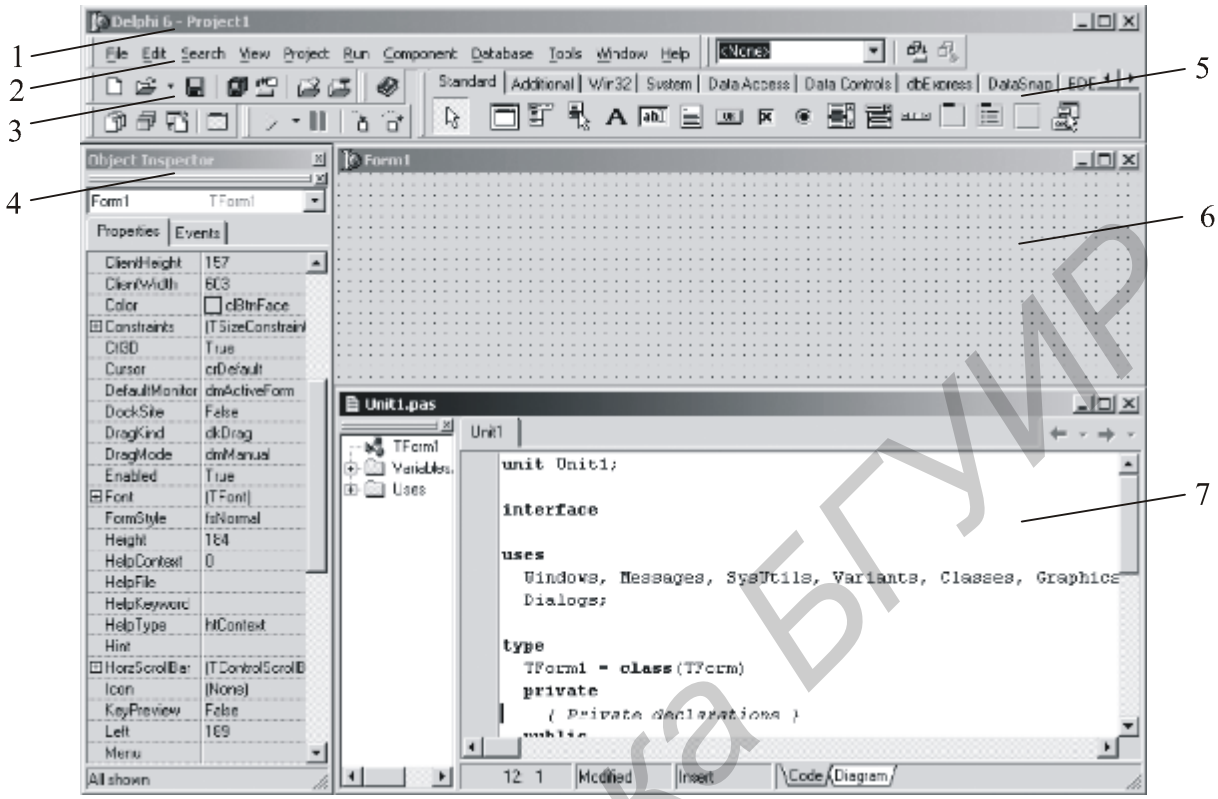

Рис.1.1

1 – главное окно; 2 – основное меню; 3 – пиктограммы основного меню;
4 - окно инспектора объектов; 5 – меню компонентов;
6- окно формы; 7 – окно текста программы

Главное окно предназначено для управления процессом создания программы. Основное меню содержит все необходимые средства для управления проектом. Левая панель содержит набор кнопок быстрого доступа, дублирующих наиболее часто используемые команды меню. Правая панель представляет собой меню библиотеки визуальных компонентов (VCL), которые представляют собой стандартные сервисные программы, помещаемые программистом на окно формы. Каждый компонент имеет определенный набор свойств (параметров), которые можно изменять как на этапе проектирования, так и во время выполнения программы.

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 и делится на две части. Слева находится имя свойства, а справа – текущее значение свойства, которое может быть изменено. Если при щелчке на свойстве появится окно выпадающего списка, то нажав на кнопку со стрелкой, направленной вниз, можно увидеть список возможных значений свойства. Если слева от имени свойства находится значок плюс, то это означает, что свойство содержит некоторое количество вложенных свойств,

доступ к которым осуществляется либо нажатием на знак плюс, либо двойным щелчком на свойстве. Страница *Events* (События) предназначена для определения реакции компонента на то или иное событие (например, нажатие определенной клавиши или щелчок «мышью» по кнопке).

Окно формы представляет собой проект Windows-окна программы. В это окно в процессе написания программы помещаются необходимые визуальные и невизуальные компоненты. При выполнении программы, помещенные визуальные компоненты будут иметь тот же вид, что и на этапе проектирования.

Окно текста программы предназначено для просмотра, написания и редактирования текста программы на языке программирования Object Pascal. При первоначальной загрузке в окне текста программы находится набор операторов обеспечивающий возможность функционирования пустой формы в качестве Windows-окна. При помещении новых компонентов на форму, текст программы автоматически дополняется описанием необходимых библиотек стандартных программ (раздел uses) и типов переменных (раздел type) (см. Листинг 1.1).

Программа в среде DELPHI составляется как описание алгоритмов, которые выполняются при возникновении того или иного события (например щелчок «мыши» на кнопке – событие OnClick, создание формы – OnCreate). 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 текста программы осуществляется с помощью клавиши **F12**.

1.3. Структура программ DELPHI

Приложение в среде DELPHI состоит из файлов с исходным текстом (расширение *pas*), файлов форм (расширение *dfm*) и файла проекта (расширение *dpr*), который связывает вместе все файлы проекта. При компиляции программы DELPHI создает файл с расширением *dcu*, содержащий в себе результат перевода в машинные коды содержимого файлов с расширениями *pas* и *dfm*. Компоновщик преобразует файлы с расширением *dcu* в единый загружаемый файл с расширением *exe*. В файлах, имеющих расширения *~df, ~dp, ~pa*, хранятся резервные копии файлов с образом формы, проекта и исходного текста соответственно.

В ***файле проекта*** (расширение *dpr*) находится информация о всех модулях, составляющих данный проект. Файл проекта автоматически создается и редактируется средой DELPHI.

Файл исходного текста (расширение *pas*) предназначен для размещения текста программы.

Модуль имеет следующую структуру:

```
unit Unit1;
interface
```

```
        // Раздел объявлений
implementation
        // Раздел реализации
begin
        // Раздел инициализации
end.
```

В разделе объявлений описываются классы, типы, переменные, заголовки процедур и функций, которые могут быть использованы другими модулями. В разделе реализации располагаются тела процедур и функций, описанных в разделе объявлений, а также типы переменных, процедуры и функции, которые будут функционировать только в пределах данного модуля. Раздел инициализации используется редко и его можно пропустить.

1.4. Пример написания программы

Задание: составить программу вычисления для заданных значений x, y арифметического выражения

$$s = \ln(y) * e^{2x} \frac{y^x}{|tg(y)|}.$$

Панель диалога программы организовать в виде, представленном на рис.1.2.

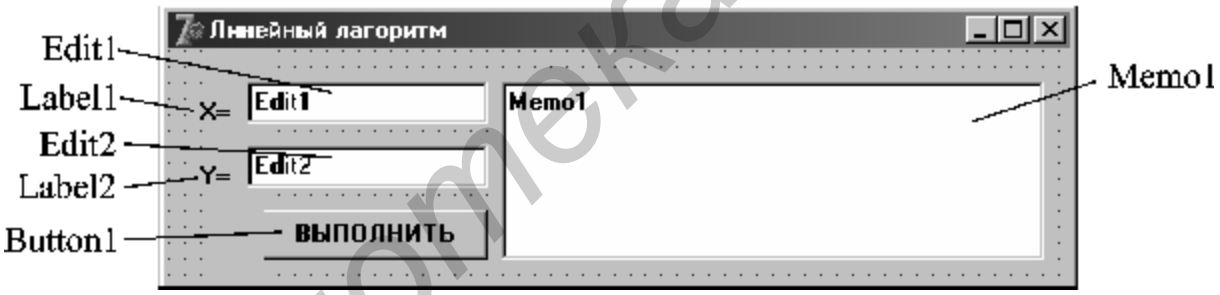

Рис. 1.2

1.4.1. Настройка формы.

Для создания нового проекта выберите в основном меню пункт File–New–Application. Новая форма в правом верхнем углу имеет кнопки управления, которые предназначены: для свертывания формы в пиктограмму, для разворачивания формы на весь экран и возвращения к исходному размеру и для закрытия формы. С помощью мыши, «захватывая» одну из кромок формы или выделенную строку заголовка отрегулируйте нужные размеры формы и ее положение на экране.

1.4.2. Изменение заголовка формы

Новая форма имеет одинаковые имя (Name) и заголовок (Caption) – Form1. Для изменения заголовка перейдите в окно инспектора объектов и щелкните кнопкой мыши на форме. На странице Properties инспектора объектов найдите свойство Caption и в правой ячейке наберите «Линейный алгоритм».

1.4.3. Размещение строки ввода (TEdit)

Для ввода данных, а так же вывода информации, которая вмещается в одну строку, используется однострочное окно редактирования (компонент **TEdit**). Доступ к отображаемой в окне информации в виде строки из символов осуществляется с помощь свойства **Text**.

Выберите в меню компонентов Standard пиктограмму  и щелкните мышью в том месте формы, где вы хотите ее поставить. Поместите два компонента TEdit в форму, в тексте программы (см. Листинг 1.1) появится две новых переменных – Edit1 и Edit2. Захватывая компоненты «мышью» отрегулируйте размеры окон и их положение.

1.4.4. Размещение надписей (TLabel)

На форме рис. 1.2 имеются две пояснительные надписи. Для нанесения таких надписей на форму используется компонент **TLabel**.

Выберите в меню компонентов Standard пиктограмму  и щелкните мышью в нужном месте формы (появится надпись Label1). Проделайте это для двух надписей, в тексте программы автоматически появятся две новых переменных типа TLabel. Для каждой надписи, щелкнув на ней мышью, отрегулируйте размер и положение на форме. В свойство Caption введите строку, например «X=».

1.4.5. Размещение многострочного окна вывода (TMemo)

Для вывода результатов работы программы в виде отчета, содержащего несколько строк текста, обычно используется текстовое окно (компонент **TMemo**). Информация, которая отображается построчно в окно типа TMemo, находится в свойстве Memo1.Lines. Новая строка добавляется методом Memo1.Lines.Add (переменная типа String). Для чистки окна во время выполнения программы используется метод Memo1.Clear.

Выберите в меню компонентов пиктограмму  и поместите компонент TMemo на форму, в тексте программы автоматически появилась новая переменная – Memo1. С помощью мыши отрегулируйте размеры и местоположение Memo1. Для отображения вертикальной и горизонтальной полос прокрутки, на странице Properties инспектора объектов установите свойство ScrollBars в положение SSBoth.

1.4.6. Написание программы обработки события создания формы (FormCreate)

После запуска программы, происходит создание спроектированной формы (событие OnCreate). Создадим подпрограмму – обработчик этого события (TForm1.FormCreate). Занесем начальные значения переменных x, y в соответствующие окна TEdit, а окно TMemo.

Для создания обработчика события FormCreate необходимо дважды щелкнуть мышью на любом свободном месте формы. На экране появится текст, в котором автоматически внесен заголовок процедуры – обработчика события создания формы: **Procedure TForm1.FormCreate(Sender:TObject)**. Между **begin** … **end** вставим текст (см. Листинг 1).

1.4.7. Написание программы обработки события нажатия кнопки (ButtonClick)

Поместите на форму кнопку (компонент **TButton**), для чего необходимо выбрать в меню компонентов Standart пиктограмму  . С помощью инспектора объектов измените заголовок (Caption) – Button1 на слово «Выполнить» или другое по вашему желанию. Отрегулируйте положение и размер кнопки.

Для создания обработчика события ButtonClick необходимо дважды щелкнуть мышью на кнопке. На экране появится текст подпрограммы, с заголовком процедуры обработчика события «щелчок мышью на кнопке»: (**Procedure TForm1.ButtonClick(Sender:TObject**);). Между **begin** … **end** вставим текст (см. Листинг 1).

Внимание! Заголовки процедур ButtonClick и FormCreate создаются средой Delphi автоматически (если набрать их вручную – программа работать не будет). При запуске программы на выполнение все функции обработки событий, у которых между begin и end не было написано текста удаляются автоматически по соответствующему запросу среды Delphi. Поэтому не надо вручную удалять ошибочно созданные обработчики.

Следует обратить внимание на то, что численные значения переменных имеют действительный тип, а компоненты TEdit и TMemo работают со строковыми переменными. Для преобразования строковой записи числа, в действительное или целое его представление и наоборот используют функции (табл. 1.4):

Таблица 1.4

StrToInt(S : string) : Integer;	Преобразует строку в целое число
StrToFloat(S:string) : extended;	Преобразует строку в действительное число
IntToStr (V : integer) : string;	Преобразует целое число в его строковое представление
FloatToStr (V : extended) : string;	Преобразует действительное число в его строковое представление
FloatToStrF (S : string; format; p,d : integer) : string;	Преобразует действительное число в его строковое представление с форматом

Правила использования параметров функции FloatToStrF (табл. 1.5):

Таблица 1.5

Значение Format	Описание
ffFixed	Формат с фиксированным положением разделителя целой и дробной частей. P задает общее количество десятичных цифр в представлении числа. d - количество цифр в дробной части.

ffNumber	Отличается от ffFixed использованием символа - разделителя тысяч при выводе больших чисел
ffCurrency	Соответствует ffNumber, но в конце строки ставится символ денежной единицы (для русифицированной версии Windows - символы «р.»).
ffExponent	Научная форма представления с множителем eXX («умножить на 10 в степени XX»). P n задает общее количество десятичных цифр мантиссы. d - количество цифр в десятичном порядке XX. Число округляется с учетом первой отбрасываемой цифры:
ffGeneral	Универсальный формат. Если число цифр в целой части числа не превышает заданной точности и значение больше $1*10^{-5}$, то используется формат с фиксированной точкой , иначе – научный.

1.4.8. Запуск и работа с программой

Запустить программу можно выбрав в главном меню пункт Run – Run, или нажав клавишу F9, или щелкнув мышью по пиктограмме ▶. При этом происходит трансляция и, если нет ошибок, компоновка программы и создание единого загружаемого файла с расширением *exe*. На экране появляется активная форма программы (рис.1.3).

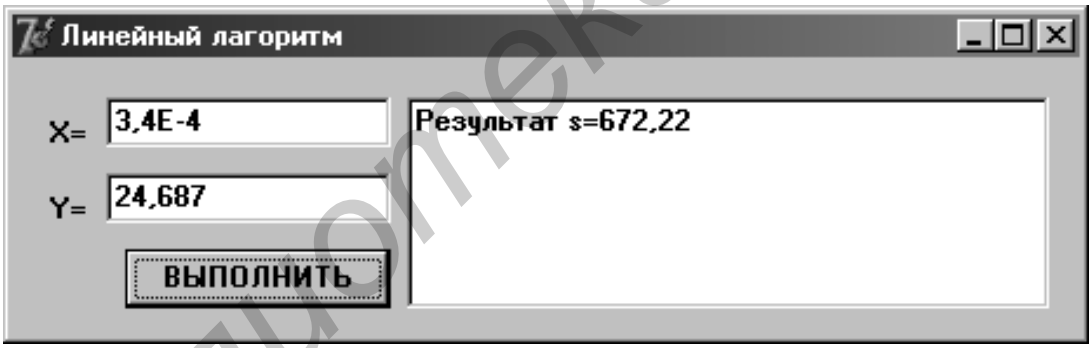

Рис.1.3

Работа с программой происходит следующим образом. Нажмите (щелкните мышью) кнопку «Выполнить». В окне Memo1 появляется результат. Измените исходные значения x, y в окнах TEdit и снова нажмите кнопку «Выполнить» - появится новые результаты. Что бы завершить работу программы нужно нажать кнопку ✕ на форме или перейти в окно DELPHI и выбрать в главном меню пункт Run – ProgramReset. Последний способ выхода из программы обычно используют в случае ее зацикливания.

В Листинге 1.1 представлен текст программы. Для наглядности, операторы, которые следует набрать выделены жирным шрифтом, остальные операторы вставляются средой Delphi автоматически.

Листинг 1.1

```
unit Unit1;
interface
uses
```

```
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, math;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Memo1: TMemo;
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit1.Text:='3,4E-4';  // Ввод начального значения X
  Edit2.Text:='24,687';  // Ввод начального значения Y
  end;

procedure TForm1.Button1Click(Sender: TObject);
 var
  x,y,a,b,s : extended;
begin
  Memo1.Clear;         // Очистка окна вывода  Memo1
   x:=StrToFloat(Edit1.Text);  // Считывание значения X
   y:=StrToFloat(Edit2.Text); // Считывание значения Y
      // Вычисление арифметического выражения
    a:=ln(y)*exp(2*x);
    b:=power(y,x)/abs(tan(y));
    s:=a*b*100;
       // Вывод результата
  Memo1.Lines.Add('Результат s='+FloatToStrF(s,ffFixed,7,2));
end;
end.
```

1.5. Индивидуальные задания

По указанию преподавателя выберите индивидуальное задание. Установите необходимое количество окон TEdit и меток TLabel. Выберите необходимые типы переменных и функции их преобразования при вводе и выводе данных. Для проверки правильности написания программы введите указанные в задании значения x, y и z – результат должен быть равен s.

1. $s=\dfrac{2\cos\left(x-\dfrac{p}{6}\right)}{0.5+\sin^2 y}\left(1+\dfrac{z^2}{3-z^2/5}\right).$

При x=14.26, y=-1.22, z=3.5×10^{-2} s=0.564849.

2. $s=\dfrac{\sqrt[3]{8+\left|x-y\right|^2+1}}{x^2+y^2+2}-e^{|x-y|}\left(tg^2 z+1\right)^x.$

При x=-4.5, y=0.75×10^{-4}, z=0.845×10^{-2} s=-55.6848.

3. $s=\dfrac{1+\sin^2(x+y)}{\left|x-\dfrac{2y}{1+x^2y^2}\right|}x^{|y|}+\cos^2\left(arctg\dfrac{1}{z}\right).$

При x=3.74×10^{-2}, y=-0.825, z=0.16×10^{2}, s=1.0553.

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

При x=0.4×10^{4}, y=-0.875, z=-0.475×10^{-3} s=1.9873.

5. $s=\ln\left(y^{-\sqrt{|x|}}\right)\left(x-\dfrac{y}{2}\right)+\sin^2 arctg(z).$

При x=-15.246, y=4.642×10^{-2}, z=20.001×10^{2} s =-182.036.

6. $s=\sqrt{10\left(\sqrt[3]{x}+x^{y+2}\right)\left(\arcsin^2 z-\left|x-y\right|\right)}.$

При x=16.55×10^{-3}, y=-2.75, z=0.15 s =-40.63069.

7. $s=5arctg(x)-\dfrac{1}{4}\arccos(x)\dfrac{x+3\left|x-y\right|+x^2}{\left|x-y\right|z+x^2}.$

При x=0.1722, y=6.33, z=3.25×10^{-4} s =-205.306.

8. $s=\dfrac{e^{|x-y|}\left|x-y\right|^{x+y}}{arctg(x)+arctg(z)}+\sqrt[3]{x^6+\ln^2 y}.$

При x=-2.235×10^{-2}, y=2.23, z=15.221 s=39.374.

9. $s=\left|x^{\frac{y}{x}}-\sqrt[3]{\dfrac{y}{x}}\right|+(y-x)\dfrac{\cos y-\dfrac{z}{(y-x)}}{1+(y-x)^2}.$

При x=1.825×10^{2}, y=18.225, z=-3.298×10^{-2} s =1.2131.

10. $s = 2^{-x}\sqrt{x+\sqrt[4]{|y|}}\sqrt[3]{e^{x-1/\sin z}}$.

При x=3.981×10^{-2}, y=-1.625×10^{3}, z=0.512 s=1.26185.

11. $s = y^{\sqrt[3]{|x|}} + \cos^3(y)\dfrac{|x-y|\left(1+\dfrac{\sin^2 z}{\sqrt{x+y}}\right)}{e^{|x-y|}+\dfrac{x}{2}}$.

При x=6.251, y=0.827, z=25.001 s=0.7121.

12. $s = 2^{(y^x)} + \left(3^x\right)^y - \dfrac{y\left(arctgz - \dfrac{p}{6}\right)}{|x| + \dfrac{1}{y^2+1}}$.

При x=3.251, y=0.325, z=0.466×10^{-4} s=4.025.

13. $s = \dfrac{\sqrt[4]{y+\sqrt[3]{x-1}}}{|x-y|\left(\sin^2 z + tgz\right)}$.

При x=17.421, y=10.365×10^{-3}, z=0.828×10^{5} s=0.33056.

14. $s = \dfrac{y^{x+1}}{\sqrt[3]{|y-2|}+3} + \dfrac{x+\dfrac{y}{2}}{2|x+y|}(x+1)^{-1/\sin z}$.

При x=12.3×10^{-1}, y=15.4, z=0.252×10^{3} s=82.8257.

15. $s = \dfrac{x^{y+1}+e^{y-1}}{1+x|y-tgz|}\left(1+|y-x|\right) + \dfrac{|y-x|^2}{2} - \dfrac{|y-x|^3}{3}$.

При x=2.444, y=0.869×10^{-2}, z=-0.13×10^{3}, s=-0.49871.

ЛАБОРАТОРНАЯ РАБОТА 2.

ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

Цель лабораторной работы: изучить способы организации ветвящихся вычислительных процессов. Написать и отладить программу разветвляющегося алгоритма.

2.1. Понятие разветвляющегося алгоритма

Алгоритм называется разветвляющимся, если он содержит несколько ветвей отличающихся друг от друга содержанием вычислений. Выход вычисли-

тельного процесса на ту или иную ветвь алгоритма определяется текущими данными.

Для программирования разветвлений в языке Паскаль имеется два условных оператора **if** и **Case .**

2.2. Оператор условия if

Этот оператор служит для разделения естественного порядка выполнения операторов программы на две ветви. Выбор пути зависит от результата проверки логического выражения или от значения булевой переменной. Конструкция использующая оператор if выглядит следующим образом:

```
If <логическое выражение> then <Оператор 1>
                          else <Оператор 2>;
```

Если значение логического выражения равно True («истина»), то выполняется «оператор 1», а «оператор 2» не выполняется. Иначе, если значение логического выражения равно False («ложь»), то выполняется «оператор 2», а «оператор 1» не выполняется. Обратите внимание на то, что в конце оператора, стоящего перед ключевым словом else точка с запятой не ставится.

Возможен сокращенный вариант конструкции:

```
If < логическое выражение > then <Оператор 1>;
```

Если значение логического выражения равно True («истина»), то выполняется «оператор 1» и далее выполняются операторы стоящие после окночания конструкции if. Иначе, если значение логического выражения равно False («ложь»), то «оператор 1» не выполняется, а выполняются операторы стоящие после окончании конструкции if.

В качестве «оператора 1» и «оператора 2» может быть использован другой оператор if. При вложенности операторов if каждое else соответствует тому then, которое непосредственно ему предшествует.

Если необходимо логическое выражение представляет собой объединение нескольких логических выражений, то каждое отдельное выражение должно быть помещено в круглые скобки.

Пример 2.1. Даны длины сторон треугольника – a, b, c. Определить является ли данный треугольник равнобедренным.

```
if (a=b) and (b=c) and (c=a) then Memo1.Lines.Add('Равнобедренный')
                          else Memo1.Lines.Add('Неравнобедренный');
```

2.3. Оператор выбора Case

Оператор **case** анализирует некоторую переменную или выражение (селектор) и в зависимости от их значения выполняет те или иные действия.

```
case <селектор> of
        <список 1>: <Оператор 1>;
        <список 2>: <Оператор 2>;
                …
        <список n>: <Оператор n>;
else <Оператор n>;
end;
```

Сначала вычисляется значение выражения-селектора, затем выполняется тот оператор, константа выбора которого равна значению выражения-селектора. Если ни одна из констант не равна значению селектора, то выполняется оператор, следующий за else. Если else отсутствует, то выполняется оператор, следующий за end. Селектор должен относиться к порядковому типу (целый, перечисляемый, булевый и др.).

Списки возможных значений могут содержать одно или несколько значений, которые разделяются запятой. Интервалы обозначаются двумя точками.

Пример 2.2. По номеру месяца вывести название сезона.

```
Case mes of
        1,2,12 : Memo1.Lines.Add('Зима');
        3..5   : Memo1.Lines.Add('Весна');
        6..8   : Memo1.Lines.Add('Лето');
        9..11  : Memo1.Lines.Add('Осень');
 else
        Memo1.Lines.Add('Месяца с таким номером нет');
 end;
```

2.4. Составной оператор

Во многих конструкциях допускается выполнение только одного оператора. Однако часто в определенном месте программы необходимо выполнить группу операторов. В Паскале имеется возможность объединить группу операторов в один ***составной оператор*** с помощью ключевых слов **begin .... end**. Такой составной оператор, например, может быть размещен после then или else.

2.5. Некоторые возможности, предоставляемые Delphi для организации разветвлений

Для организации разветвлений в Delphi обычно используются компоненты в виде кнопок – переключателей TCheckBox и TradioGroup. Состояние таких кнопок визуально отражается на форме. Кнопка типа TCheckBox, помещенная на форму, позволяет пользователю с помощью щелчка мышью на ней изменить ее состояние (включена-выключена) и соответственно изменить значение связанной с ней переменной TCheckBox.Checked булевского типа:

```
If CheckBox.Checked then <Оператор 1>
                    else <Оператор 2>;
```

Группа кнопок типа TRadioGroup, помещенная на форму, позволяет пользователю организовать селектор, передающий в программу через переменную целого типа RadioGroup.ItemIndex номер включенной кнопки (0, 1, 2, …).

2.6. Пример написания программы

Задание: Вычислить $s = \begin{cases} (x+y)^2 - \sqrt{y}, & \text{если } f(x) \rangle 0{,}5 \\ (x+y)^2 + (xy)^3, & \text{если } -0{,}5 < f(x) <= 0{,}5 \\ (x+y)^2 + \ln(y), & \text{иначе.} \end{cases}$. В качестве *f(x)* использовать по выбору cos(x), x^2 или sin(x). Панель диалога программы организовать в виде, представленном на рис. 2.1.

2.6.1. Создание формы

Создайте форму, такую же как в первом задании, скорректировав текст надписей и положение окон TEdit.

2.6.2. Работа с компонентом TRadioGroup

Выберите в меню компонентов Standard пиктограмму  и поместите ее в нужное место формы. На форме появится окаймленный линией чистый прямоугольник с заголовком RadioGroup1. Замените заголовок (Caption) на «f(x)». Дважды щелкните по правой части свойства Items мышью, появится редактор списка названий кнопок. Наберите три строки с именами: в первой строке – «Косинус», во второй – «Квадрат», в третьей – «Синус» и нажмите кнопку ОК. После этого на форме внутри окаймления появится три кнопки-переключателя с введенными надписями. Введите в свойство ItemIndex значение «0» (при выполнении программы будет выбрана первая кнопка селектора).

Обратите внимание на то, что в тексте программы появилась переменная RadioGroup1 типа TRadioGroup. При нажатии одной из кнопок группы в переменной целого типа **RadioGroup1.ItemIndex** будет находиться номер выбранной кнопки (нумерация начинается от нуля).

Форма приведена на рис. 2.1. Текст программы приведен на Листинге 2.1.

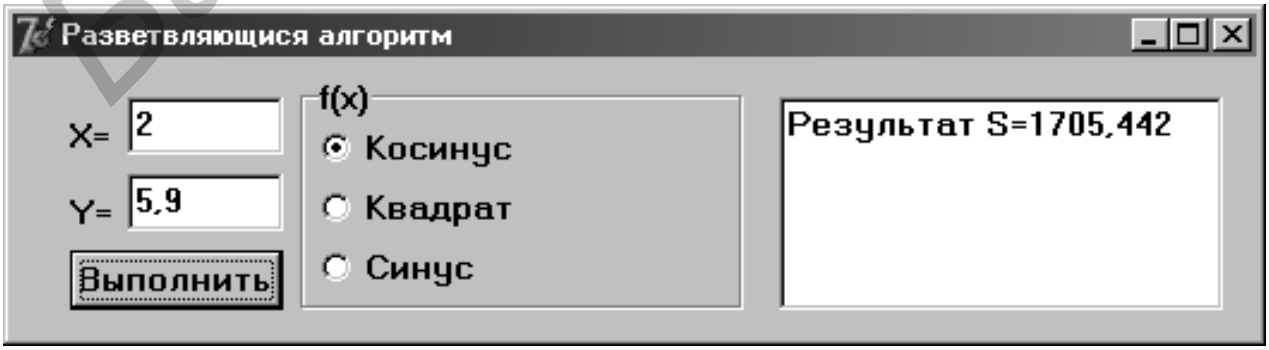

Рис. 2.1

Листинг 2.1

```
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,  Dialogs, StdCtrls, ExtCtrls, math;
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Edit2: TEdit;
    RadioGroup1: TRadioGroup;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
  Edit1.Text:='2';
  Edit2.Text:='5,9';
  Memo1.Clear;
end;

procedure TForm1.Button1Click(Sender: TObject);
 var
  x,y,a,f,s :extended;
begin
   Memo1.Clear;
 x:=StrToFloat(Edit1.Text);
 y:=StrToFloat(Edit2.Text);
  case RadioGroup1.ItemIndex of
       0 : f:=cos(x);
       1 : f:=sqr(x);
       2 : f:=sin(x);
  end;
         a:=sqr(x+y);
    if f > 0.5 then s:=a-sqrt(y)
```

```
            else
        if (f > -0.5) and (f <= 0.5) then s:=a+power(x*y,3)
                                     else s:=a+ln(y);
Memo1.Lines.Add('Результат S='+FloatToStrF(s,ffFixed,8,3));
end;
end.
```

2.7. Индивидуальные задания

По указанию преподавателя выберите индивидуальное задание. В качестве f(x) использовать по выбору: Sin(x), Cos(x) или Tg(x). Отредактируйте вид формы и текст программы, в соответствии с полученным заданием.

1. $$s=\begin{cases}\sqrt{(x+y)}-\sin(x), & f(x)>3\\ \sin^2(x)+\mathrm{tg}(y), & f(x)=3\\ (x*y)^2+1, & \text{иначе.}\end{cases}$$

2. $$s=\begin{cases}\ln(x)+\left(f(x)^2+y\right)^3, & f(x)\rangle 0\\ \ln|y|+\left(f(x)^2+y\right)^3, & f(x)\langle 0\\ \left(f(x)^2+y\right)^3, & \text{иначе.}\end{cases}$$

3. $$s=\begin{cases}\sin^2(x)+y^2+5, & f(x)=0\\ \left(x-\sqrt{y}\right)^2+\cos(y), & f(x)\rangle 0\\ (y-x)^2+tg(y), & \text{иначе.}\end{cases}$$

4. $$s=\begin{cases}f(x)^3+arctg(x*y), & f(x)\rangle y\\ y^3+arctg(x), & y\rangle f(x)\\ \left(y+\sqrt{x}\right)^3+0.5, & \text{иначе.}\end{cases}$$

5. $$s=y\begin{cases}x\sqrt{y}+2*x, & x\rangle 0\\ y/\sqrt{\sin^4(x)}, & x\langle 0\\ \sqrt{|x*\sin(y)|}+tg(y), & \text{иначе.}\end{cases}$$

6. $$s=\begin{cases}e^{x-|y|}, & 0.5\langle f(x)\langle 10\\ \sqrt{|x^2+y|}, & 0.1\langle f(x)\langle 0.5\\ 2x^3, & \text{иначе.}\end{cases}$$

7. $$s=\begin{cases}e^x+\sin(y), & 1\langle f(x)\langle 22\\ \sqrt{|x+3y|}, & 22\langle f(x)\langle 40\\ x+2\sin^2(y), & \text{иначе.}\end{cases}$$

8. $$s=\begin{cases}\sin\left(5x+tg|x|\right), & -3\langle f(x)\langle y\\ \cos\left(3x+\ln(y)\right), & y\rangle f(x)\\ (2x+y)^2, & \text{иначе.}\end{cases}$$

9. $$s=\begin{cases}2x^3+3y^2, & x\rangle|y|\\ |y-\sin(p)|, & 5\langle x\langle|y|\\ (x-y)^2+e^{xy}, & \text{иначе.}\end{cases}$$

10. $$s=\begin{cases}\ln\left(x+|y|\right), & f(x)\rangle 10\\ e^{x+y}+\sin(x), & f(x)\langle 10\\ x+3\cos(y), & \text{иначе.}\end{cases}$$

11. $$s=\begin{cases}\ln(x)+\cos(y), & f(x)\langle 3\\ \sqrt{|x^2+3\sin(y)|}, & 3\langle f(x)\langle 24\\ 2tg(y)-5x, & \text{иначе.}\end{cases}$$

12. $$s=\begin{cases}xy-2tg(y), & 1\langle f(x)\langle 22\\ tg(x)-4\ln(xy), & 22\langle f(x)\langle 40\\ 3y^2+\cos(x), & \text{иначе.}\end{cases}$$

13. $s=\begin{cases}3x^{y}+\sqrt{\sin(y)}, & f(x)\langle 10\\ \sin(x)*2tg(xy), & 10\langle f(x)\langle 12\\ \cos(x)+\ln^{2}(y), & \text{иначе.}\end{cases}$

14. $s=\begin{cases}\sqrt[3]{e^{xy}+\ln(y)}, & 1\langle f(x)\langle 22\\ 12*\left|\sin(x)-12y\right|, & 22\langle f(x)\langle 40\\ 4x^{2}+\cos^{2}(y), & \text{иначе.}\end{cases}$

15. $s=\begin{cases}e^{2x}-3\cos^{2}(y), & 10\langle f(x)\\ \ln(xy)*tg(x), & 2\langle f(x)\langle 10\\ x^{y}+\ln(y), & \text{иначе.}\end{cases}$

ЛАБОРАТОРНАЯ РАБОТА 3.

ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ

Цель лабораторной работы: изучить способы организации циклических вычислительных процессов.

3.1. Операторы для организации циклов

Цикл – многократное выполнение некоторой последовательности операторов. Количество повторений определяется *управляющей переменной (счетчиком)*, которая должна изменять свое значение внутри цикла. Совокупность операторов, находящихся внутри цикла называют *телом цикла*.

3.1.1. Организация цикла с помощью оператора for

Оператор

```
for <счетчик>:=<начальное значение> to <конечное значение> do
                                                    begin
                                                        <операторы>
                                                    end;
```

организует повторение последовательности операторов при изменении значения счетчика от начального до конечного, с шагом «единица». Вначале счетчику присваивается начальное значение. После каждого выполнения тела цикла значение счетчика увеличивается на единицу. При достижении счетчиком конечного значения операторы выполняются в последний раз, и управление передается первому оператору стоящему после цикла.

Вторая форма оператора

```
for <счетчик>:=<начальное значение> downto <конечное значение> do
                                                    begin
                                                        <операторы>
                                                    end;
```

организует повторение последовательности операторов при изменении значений счетчика от начального до конечного, с шагом « минус единица».

Если начальное и конечное значения равны, то тело цикла выполняется только один раз. Если начальное значение превышает конечное (для первой формы оператора) или начальное значение меньше конечного (для второй формы оператора), то тела цикла не выполняется ни разу.

Значение счетчика внутри цикла изменять нельзя.

Как правило, с помощью оператора for организуют циклы с заранее известным количеством повторений.

Пример 3.1. Вычислить N! (1*2*3*…*N).

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

Переменная целого типа *i* изменяется от 1 до *n* с шагом 1. Все значения *i* перемножаются для получения факториала числа *n*.

Пример 3.2. Найти минимум функции $y(x)=\sin^2(x)+4*\cos(x)$ на интервале [-1; 1] с точностью ε=0,01.

```
n:=Round((b-a)/eps)+1;          // Расчет количества шагов
   x:=a;                        // Первая точка интервала
 min:=sqr(sin(x))+4*cos(x);     // В качестве минимума принимается
                                // значение функции в начальной точке интервала
 for i:=1 to n do begin
    x:=a+eps;                   // Переход к следующей точке
      y:=sqr(sin(x))+4*cos(x); // Расчет значения функции
   if y<min then min:=y;        // Проверка, является ли
                                // текущее значение y минимальным
              end;
```

Минимум с точностью ε находится путем перебора всех значений от *a* до *b* с шагом равным ε. В начале, в качестве минимума принимается значение функции в первой точке интервала. После этого перебираются все точки интервала с заданным шагом. Если текущее значение *y* меньше минимального, то оно принимается за минимум. Так как шаг перебора параметра *x* в данном примере не кратен 1, то использование оператора for нежелательно.

3.1.2. Организация цикла с помощью оператора repeat

Оператор

```
repeat
      <операторы>
until <условие>;
```

организует повторение операторов, помещенных между ключевыми словами repeat и until до тех пор, пока условие не примет значение истина (true). Так как условие проверяется после выполнения операторов цикла, то тело цикла будет выполняться хотя бы один раз.

Пример 3.3. Вычислить N! (1*2*3*…*N).

```
i:=1; f:=1;
if n>1 then
 repeat
   i:=i+1;
     f:=f*i;
 until (i>=N);
```

Такой способ вычисления факториала менее удобен, чем с использованием оператора for. Для того, чтобы при n<2 не выполнялось тело цикла введена проверка.

Пример 3.4. Найти минимум функции $y(x)=\sin^2(x)+4*\cos(x)$ на интервале [-1; 1] с точностью ε=0,01.

```
x:=a;
min:=sqr(sin(a))+4*cos(a);
repeat
 x:=x+eps;
  y:=sqr(sin(x))+4*cos(x);
     if y<min then min:=y;
 until(x>=b);
```

Данный алгоритм более удобен, чем рассмотренный в примере 3.2.

3.1.3. Организация цикла с помощью оператора while

Оператор

```
While<условие>do begin
                    <операторы>
                end;
```

организует повторение операторов, помещенных между begin и end, до тех пор, пока условие на примет значение ложь (false).

В отличии от оператора repeat, если условие будет равно false тело цикла не будет выполнено ни разу.

Пример 3.5. Вычислить N! (1*2*3*…*N).

```
while (i<N) do begin
                i:=i+1;
                 f:=f*i;
                 end;
```

Такой способ вычисления факториала менее удобен, чем с использованием оператора for.

Пример 3.6. Найти минимум функции $y(x)=\sin^2(x)+4*\cos(x)$ на интервале [-1; 1] с точностью ε=0,01.

```
x:=a;
 min:=sqr(sin(a))+4*cos(a);
```

```
while (x<b) do begin
 x:=x+eps;
    y:=sqr(sin(x))+4*cos(x);
 if y<min then min:=y;
                  end;
```

Данный алгоритм более удобен, чем рассмотренный в примере 3.2.

3.2. Организация вложенных циклов

Циклы могут быть вложены друг в друга.

Пример 3.7. Вычислить таблицу значений функции $y(x)=\sum_{i=1}^{10}\frac{i\cdot\sin(i)}{x^2}$ на интервале $[a; b]$ с шагом h.

```
   x:=a;
repeat
     y:=0;
 for i:=1 to 10 do  y:=i*sin(i)/sqr(x);
Memo1.Lines.Add(' При x='+FloatToStrF(x,ffFixed,4,1)+
                  ' y='+FloatToStrF(y,ffFixed,8,3));
 x:=x+h;
until(x > b+h);
```

3.3. Организация прерывания цикла

Иногда необходимо прервать выполнение тела цикла до выполнения условия окончания. Можно использовать следующие процедуры и функции:

Continue – прерывает выполнение текущей итерации, и передает управление следующей итерации.

Break – прерывает тело любого цикла и передает управление следующему за циклом оператору.

Exit – прерывает не только тело цикла, но и прекращает выполнение процедуры или функции, в которой размещен этот цикл.

Abort – прерывает не только тело цикла, но и прекращает выполнение блока, в которой размещен этот цикл. Данная процедура генерирует «молчаливое» исключение, не связанное с сообщением об ошибке.

3.4. Алгоритмы, использующие рекуррентную последовательность

Последовательность чисел $a_0, a_1, a_2, \ldots, a_n$ называется рекуррентной, если каждый последующий ее член выражается через один или несколько предыдущих. Если для получения следующего значения последовательности требуется только один предыдущий элемент $a_k=\varphi(a_{k-1})$, то это последовательность первого порядка. Для нахождения всех членов такой последовательности достаточно задать начальный элемент a_0.

Пример 3.8. Вычислить сумму $s(x)=\sum_{k=1}^{100}a^k=\sum_{k=1}^{100}(-1)^k\frac{x^k}{k!}$.

Вначале составления алгоритма необходимо получить рекуррентную формулу. Для получения формулы рассмотрим значение слагаемого при различных значениях k: при $k=1;\ a_1=-1\frac{x}{1}$; при $k=2;\ a_2=1\frac{x\cdot x}{1\cdot 2}$; при $k=3;\ a_3=-1\frac{x\cdot x\cdot x}{1\cdot 2\cdot 3}$ и т.д. Видно, что на каждом шаге слагаемое дополнительно умножается на $-1\frac{x}{k}$. Исходя из этого формула рекуррентной последовательности будет иметь вид $a_k=-a_{k-1}\frac{x}{k}$. Полученная формула позволяет избавиться от многократного вычисления факториала и возведения в степень. Текст программы представлен ниже.

```
s:=0; // Начальное значение суммы
   a:=1; // Начальное значение для вычисление очередного
      //  члена рекуррентной последовательности
for k:=1 to 100 do begin
   a:=-a*x/k; // Вычисление члена рекуррентной последовательности
   s:=s+a; // Суммирование всех слагаемых
            end;
```

Пример 3.9. Вычислить сумму $s(x)=\sum_{k=0}^{100}a^k=\sum_{k=0}^{100}(-1)^k\frac{x^{2k}}{(2k)!}\sin(x)$.

Анализируя формулу можно сделать вывод, что получить рекуррентную зависимость для sin(x) достаточно сложно, поэтому будем считать функцию sin(x) нерекуррентной частью и рассчитывать отдельно. Получим рекуррентную зависимость для оставшийся формулы $\sum_{k=0}^{100}(-1)^k\frac{x^{2k}}{(2k)!}$. Рассмотрим значение слагаемого при различных значениях k: при $k=0;\ a_1=1\frac{1}{1}$; при $k=1;\ a_1=-1\frac{x^2}{1*2}$; при $k=2;\ a_2=1\frac{x^2*x^2}{1*2*3*4}$; при $k=3;\ a_3=-1\frac{x^2*x^2*x^2}{1*2*3*4*5*6}$ и т.д. Видно, что на каждом шаге слагаемое дополнительно умножается на $-1\frac{x^2}{(2k-1)*(2k)}$. Исходя из этого формула рекуррентной последовательности будет иметь вид $a_k=-a_{k-1}\frac{x^2}{(2k-1)*(2k)}$. Текст программы представлен ниже.

```
s:=sin(x);
 a:=1;
for k:=1 to 100 do begin
```

```
        a:=-a*sqr(x)/(2*k*(2*k-1));
            s:=s+a*sin(x);
                end;
```

При расчете удобно начинать расчет не с нулевого элемента, а с первого. Значение суммы в нулевом элементе рассчитывается до начала цикла (s:=sin(x)). Нерекуррентная часть добавляется при суммировании (s:=s+a*sin(x)).

3.5. Пример написания программы

Задание: написать и отладить программу, которая выводит таблицу значений функции $y(x) = ch(x)$ и ее разложения в ряд $s(x) = \sum_{k=0}^{n} \frac{x^{2k+1}}{(2k+1)!}$ для x изменяющихся в интервале от x_N до x_K с шагом h.

Панель диалога представлена на рис. 3.1.

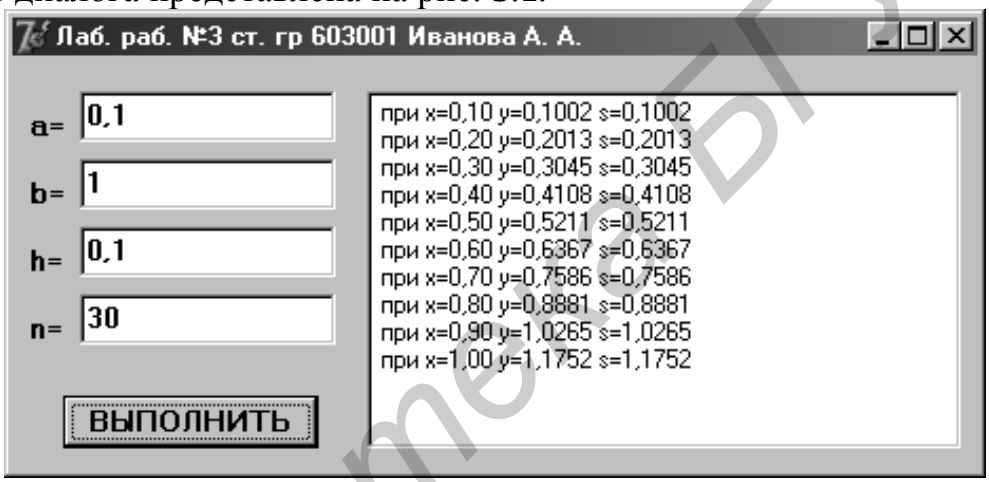

Рис. 3.1

Текст программы приведен на Листинге 3.1.

Листинг 3.1

```
unit Unit3;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, Math;
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
```

```
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
Memo1.Clear;
Edit1.Text:='0,1';
Edit2.Text:='1';
Edit3.Text:='0,1';
Edit4.Text:='30';
end;

procedure TForm1.Button1Click(Sender: TObject);
 var k, n : integer;
     a,b,h,s,x,y : extended;
begin
Memo1.Clear;
  a:=StrToFloat(Edit1.Text);
  b:=StrToFloat(Edit2.Text);
  h:=StrToFloat(Edit3.Text);
  n:=StrToInt(Edit4.Text);

x:=a;
   repeat
     s:=x;
     a:=x;
    for k:=1 to n do begin
     a:=-a*sqr(x)/(2*k*(2*k-1));
     s:=s+a*sin(x);
                      end;
     y:=sinh(x);
       Memo1.Lines.Add(' при x='+FloatToStrF(x,ffFixed,4,2)+' y='+
        FloatToStrF(y,ffFixed,8,4)+' s='+FloatToStrF(y,ffFixed,8,4));
   x:=x+h;
   until (x>b+h/2); // Добавляется h/2 для вывода последнего элемента
end;
end.
```

3.6. Индивидуальные задания

По указанию преподавателя выберите вариант задачи. Нарисуйте схему алгоритма. Спроектируйте панель диалога и напишите текст программы.

Вывести на экран таблицу значений функции *Y(x)* и ее разложения в ряд *S(x)* для x изменяющихся от x_n до x_k с заданным количеством шагов n ($h = \frac{x_k - x_n}{M}$). Близость значений *S(x)* и *Y(x)* во всем диапазоне значений x указывает на правильность вычисления *S(x)* и *Y(x).*

После написания программы и исправления ошибок трансляции изучите средства отладки программ (см. прилож.), для чего установите курсор на первый оператор и нажмите клавишу F4. После этого, нажимая клавишу F7, выполните пошагово программу и проследите, как меняются все переменные в процессе выполнения.

Таблица 3.1

№	x_n	x_k	S(x)	n	Y(x)
1	2	3	4	5	6
1	0.1	1	$\sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{(2n+1)!}$	100	$\sin x$
2	0.1	1	$\sum_{n=0}^{\infty}\frac{x^{2n}}{(2n)!}$	180	$\frac{e^x + e^{-x}}{2}$
3	0.1	1	$\sum_{n=0}^{\infty}\frac{\cos n\frac{p}{4}}{n!}x^n$	250	$e^{x\cos\frac{p}{4}}\cos(x\sin\frac{p}{4})$
4	0.1	1	$\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n}}{(2n)!}$	200	$\cos x$
5	0.1	0.7	$\sum_{n=0}^{\infty}\frac{2n+1}{n!}x^{2n}$	300	$(1+2x^2)e^{x^2}$
6	0.1	1	$\sum_{n=0}^{\infty}\frac{x^{2n+1}}{(2n+1)!}$	100	$\frac{e^x - e^{-x}}{2}$
7	0.2	1	$\sum_{n=0}^{\infty}\frac{(\ln 9)^n}{n!}x^n$	250	9^x
8	0.1	0.7	$\sum_{n=0}^{\infty}\frac{(2x)^n}{n!}$	300	e^{2x}

9	0.3	1	$\sum_{n=0}^{\infty}\frac{n^2+1}{n!}\left(\frac{x}{2}\right)^n$	260	$\left(\frac{x^2}{4}+\frac{x}{2}+1\right)e^{\frac{x}{2}}$
10	0.1	0.5	$\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n+1}}{2n+1}$	60	$arctgx$

Окончание табл. 3.1

1	2	3	4	5	6
11	0.2	1	$\sum_{n=0}^{\infty}(-1)^n\frac{2n^2+1}{(2n)!}x^{2n}$	190	$\left(1-\frac{x^2}{2}\right)\cos x-\frac{x}{2}\sin x$
12	0.1	1	$\sum_{n=1}^{\infty}(-1)^n\frac{(2x)^{2n}}{(2n)!}$	220	$2(\cos^2 x-1)$
13	-2	-0.1	$\sum_{n=0}^{\infty}(-1)^n\frac{x^{2n}}{(2n+1)!}$	300	$\frac{\sin(x)}{x}$
14	0.2	0.8	$\sum_{n=1}^{\infty}\frac{n^2}{(2n+1)!}x^n$	250	$\frac{1}{4}\left(\frac{x+1}{\sqrt{x}}sh\sqrt{x}-ch\sqrt{x}\right)$
15	0.1	0.8	$\sum_{n=1}^{\infty}(-1)^{n+1}\frac{x^{2n}}{2n(2n-1)}$	120	$xarctgx-\ln\sqrt{1+x^2}$

ЛАБОРАТОРНАЯ РАБОТА 4.

ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ МАССИВОВ

Цель лабораторной работы: изучить свойства компонента TStringGrid. Написать программу с использованием массивов.

4.1. Работа с массивами

Массив – структура однотипных данных, каждый элемент которой хранится в отдельной ячейке, доступ к которой осуществляется по ее номеру. Массив характеризуется: имением массива, типом хранимых данных, размером и размерностью.

Каждый элемент массива обозначается именем, за которым в квадратных скобках следует один или несколько индексов, разделенных запятыми, например: a[1], bb[I], c12[I,j*2], q[1,1,I*j-1]. В качестве индекса можно использовать любые порядковые типы за исключением LongInt.

Массив объявляется с помощью ключевого слова array:

```
type
  <имя типа массива> = array [<перечисляемый тип>] of <тип элемента массива>;
var
  <имя массива> : array [<перечисляемый тип>] of <тип элемента массива>;
```

Примеры описания массивов:

```
type
  TMas=array[1..10] of integer; // Описание типа одномерного массива;
var
  A : TMas;                     // Массив типа TMas;
  B : array[1..10] of extended;// Одномерный массив действительных чисел;
  D : array[1..10,1..10] of char;   // Двумерный массив символьного типа.
```

Элементы массивов могут использоваться в выражениях так же, как и обычные переменные. Доступ к каждому элементу массива осуществляется по индексу заключенному в квадратные скобки, например:

```
A[j]:=5;
B[k]:=B[i]+A[j*2+k];
D[i,j]:= 2*A[i];
```

причем значение индексов не должно выходить за границы диапазона ($m \le j \le n$) указанного в описании. Чаще всего используются индексы целого типа, хотя возможны индексы порядкового типа (например Char).

Нумерация элементов в одномерных массивах ведется слева направо, а в двумерных массивах – первый индекс – слева направо, а второй индекс – сверху вниз. Например:

Одномерный массив	Двумерный массив
A : array[1..5] of integer	A : array[1..3, 1..4] of integer
A[1], A[2], A[3], A[4], A[5]	A[1,1], A[1,2], A[1,3], A[1,4] A[2,1], A[2,2], A[2,3], A[2,4] A[3,1], A[3,2], A[3,3], A[3,4]

4.2. Операции над массивами

Однотипные массивы могут участвовать в операциях отношения «равно», «не равно» и в операторе присваивания. Например, если массивы объявлены

```
type
    TMas1 = array[1..10] of integer;
var
    A, B : Tmas;
    C : array[1..10] of integer;
```

то, допустимы следующие операции:

выражение *A* = B	принимает значения *true* если все элементы массива A и массива B одинаковы и *false* в противном случае;
выражение *A* <> B	принимает значение *true* если массивы A и B отличаются хотя бы в одном элементе;
A := B	элементам массива A присваиваются значения всех соответствующих элементов массива B.

Следует обратить внимание на то, что вышеперечисленные операции определены только для однотипных массивов. В данном примере массивы A и B являются однотипными, а массив C не является однотипным с массивами A и B, несмотря на то, что все эти массивы имеют одинаковую структуру. При попытке присваивания *A* := C компилятор выдаст сообщение о несовместимости типов.

4.3. Примеры часто встречающихся алгоритмов работы с массивами

Пример 4.1. Вычислить сумму n элементов одномерного массива a ($S = \sum_{i=1}^{n} a_i$).

```
S:=0;
for i:=1 to n do S:=S+A[i];
```

Пример 4.2. Вычислить сумму модулей всех элементов матрицы B ($S = \sum_{i=1}^{m}\sum_{j=1}^{n} |b_{ij}|$).

```
S:=0;
for i:=1 to m do
  for j:=1 to n do S:=S+(B[i,j]);
```

Пример 4.3. Вычислить сумму диагональных элементов квадратной матрицы B ($S = \sum_{i=1}^{n} b_{ii}$).

```
S:=0;
for i:=1 to n do S:=S+B[i,i];
```

Пример 4.4. Найти значение и номер максимального элемента одномерного массива.

```
max:=A[1]; // За максимум принимается первый элемент массива
nmax:=1;   // Номер максимального элемента массива равен 1
 for i:=1 to n do
    if A[i]>max then  begin // Если A[i] больше максимального, то …
          max:=A[i]; // за максимум принимается A[i]
          nmax:=i;  // номер максимального элемента равен i
                    end;
```

Пример 4.5. Транспонировать одномерный массив $\vec{a}$.

```
Repeat
      r:=a[i];
      a[i]:=a[j];
      a[j]:=r;
  i:=i+1;
  j:=j-1;
Until i>=j;
```

Для обмена данными между ячейками используется вспомогательная переменная r.

Пример 4.6. Транспонировать двумерный массив относительно главной диагонали.

В квадратной матрице A размером n×n элементы главной диагонали имеют одинаковые индексы ($a_{i,i}$, i=1, n). Каждому элементу стоящему выше главной диагонали ($a_{i,j}$, 1≤ i ≤ n-1, i+1≤ j ≤ n) соответствует симметричный ему элемент стоящий ниже главной диагонали (a_{ji} , 1≤ i ≤ n-1, i+1≤ j ≤ n).

```
for i:=1 to n-1 do
for j:=i+1 to n do
      begin
        r:=a[i,j];
        a[i,j]:=a[j,i];
        a[j,i]:=r;
      end;
```

Пример 4.7. В матрице A размера m×n переставить местами строки с номерами $к_1$ и $к_2$.

```
for j:=1 to m do
      begin
        r:=a[k1,j];
```

```
    a[k1,j]:=a[k2,j];
    a[k2,j]:=r;
  end;
```

Пример 4.8. Отсортировать одномерный массив A в порядке возрастания элементов. Использовать метод пузырька.

```
for i:=2 to n do
for j:=n downto i do
  if a[j-1]>a[j] then
      begin
        r:=a[j];
        a[j]:=a[j-1];
        a[j-1]:=r;
      end;
```

4.4. Компонент TStringGrid

Delphi имеется возможность организовать ввод/вывод двухмерных и одномерных массивов с отображением их на форме. Для этого используют специальный компонент ***TStringGrid*** (находится в меню *Additional*), который предназначен для отображения информации в виде двухмерной (одномерной) таблицы каждая ячейка которой представляет собой окно однострочного редактора (аналогично окну TEdit). Доступ к информации осуществляется с помощью свойства Cells[ACol : Integer; ARow : Integer] : String, где ACol, ARow – индексы элемента двумерного массива. Свойства ColCount и RowCount устанавливают количество столбцов и строк в таблице, а свойства FixedCols и FixedRows задают количество столбцов и строк фиксированной зоны. Фиксированная зона выделена другим цветом, и в нее запрещен ввод информации с клавиатуры.

4.5. Пример написания программы

Задание: Задан двухмерный массив целых чисел A, размером m на n. Найти количество четных положительных элементов массива, и произведение отрицательных элементов.

4.5.1. Настройка компонента TStringGrid

Для установки компонента TStringGrid на форму необходимо на странице Additional меню компонентов щелкнуть мышью по пиктограмме  . После этого щелкните мышью в нужном месте формы. Захватывая кромки компонента, отрегулируйте его размер. В инспекторе объектов значения свойств ColCount и RowCount установите 5 (пять строк и пять столбцов), а FixedCols и FixedRows установите 1 (один столбец и одна строка с фиксированной зоной). По умолчанию в компонент TStringGrid запрещен ввод информации с клавиатуры, поэтому в инспекторе объектов необходимо раскрыть раздел Options (нажав на знак «+», стоящий слева от Options) и установить свойство goEditing в положение True.

Панель диалога приведена на рис. 4.1.

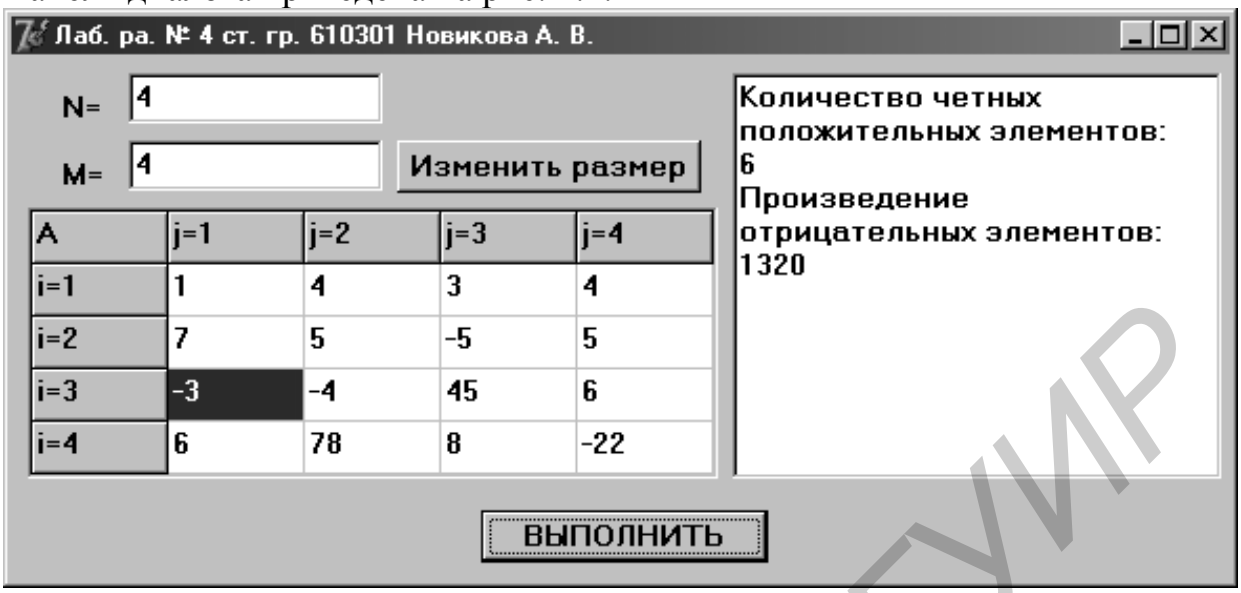

Рис. 4.1

Текст программы приведен на Листинге 4.1.

Листинг 4.1

```
unit Unit4;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, Grids;
type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    Label1: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Label2: TLabel;
    Button1: TButton;
    Memo1: TMemo;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

type
  TMas = array[1..10,1..10] of integer; // Объявление типа двухмерный
                                        // массив размерном 10 x 10
var
```

```
  Form1: TForm1;
  A : TMas;  // Объявление двухмерного массива
  n, m : integer;
  i, j : integer;
implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
 begin
  Memo1.Clear;
        n:=4;  // Число строк в массиве
       m:=4;  // Число столбцов в массиве
   Edit1.Text:='4';
   Edit2.Text:='4';
       StringGrid1.Cells[0,0]:='A';
       for i:=1 to n do StringGrid1.Cells[0,i]:='i='+IntToStr(i);
       for j:=1 to m do StringGrid1.Cells[j,0]:='j='+IntToStr(j);
end;

procedure TForm1.Button1Click(Sender: TObject); // Изменить размер
 begin
       n:=StrToInt(Edit1.Text);   // Число строк в массиве
       m:=StrToInt(Edit2.Text);  // Число столбцов в массиве
  {Задание числа строк и столбцов в таблице}
       StringGrid1.RowCount:=N+1;
       StringGrid1.ColCount:=M+1;
       for i:=1 to n do StringGrid1.Cells[0,i]:='i='+IntToStr(i);
       for j:=1 to m do StringGrid1.Cells[j,0]:='j='+IntToStr(j);
end;

procedure TForm1.Button2Click(Sender: TObject);
 var ne, p : integer;
begin
   Memo1.Clear;
   {Заполнение массива A элементами из таблицы StringGrid1}
  for i:=1 to n do
  for j:=1 to m do
    A[i,j]:=StrToInt(StringGrid1.Cells[j,i]);
   {Расчет}
    ne:=0;
    p:=1;
  for i:=1 to n do
  for j:=1 to m do
   if (A[i,j]>0) and not odd(A[i,j]) then ne:=ne+1
                          else
          if (A[i,j]<0) then p:=p*A[i,j];
```

```
    { Вывод результата }
  Memo1.Lines.Add('Количество четных положительных элементов: '
          +IntToStr(ne));
  Memo1.Lines.Add('Произведение отрицательных элементов: '
          +IntToStr(p));
  end;
  end.
```

4.6. Индивидуальные задания

По указанию преподавателя выберите вариант задачи. Во всех заданиях переменные вводить и выводить с помощью компонента TEdit, массивы – с помощью компонента TStringGrid, в котором 0-й столбец и 0-ю строку использовать для отображения индексов массивов. Вычисления выполнять, после нажатия кнопки типа TButton.

1. Задан двухмерный массив целых чисел A размером M на N. Найти сумму элементов, расположенных ниже главной диагонали.
2. Задан двухмерный массив целых чисел A размером M на N. Найти произведение элементов, расположенных выше главной диагонали.
3. Задан двухмерный массив целых чисел A размером M на N. Найти сумму элементов, расположенных выше побочной диагонали.
4. Задан двухмерный массив целых чисел A размером M на N. Найти произведение элементов, расположенных ниже побочной диагонали.
5. Задан двухмерный массив целых чисел A размером M на N. Найти сумму элементов, расположенных ниже главной диагонали.
6. Задан двухмерный массив целых чисел A размером M на N. Найти количество элементов, расположенных выше главной диагонали.
7. Задан двухмерный массив целых чисел A размером M на N. Найти минимальный элемент, расположенный ниже главной диагонали.
8. Задан двухмерный массив целых чисел A размером M на N. Найти максимальный элемент, расположенный выше главной диагонали.
9. Задан двухмерный массив целых чисел A размером M на N. Найти максимальный элемент, расположенный выше побочной диагонали.
10. Задан двухмерный массив целых чисел A размером M на N. Найти минимальный элемент, расположенный ниже побочной диагонали.
11. Задан двухмерный массив целых чисел A размером N на M. Найти максимальный элемент и поменять его местами с элементом A[1,1].
12. Задан двухмерный массив целых чисел A размером N на M . Найти минимальный элемент и поменять его с элементом A[N,M].
13. Задан двухмерный массив целых чисел A размером N на M, состоящий из нулей и единиц. Найти количество нулей и единиц в этом массиве.
14. Задан двухмерный массив целых чисел A размером N на M. Найти число элементов A[i,j] > T и сумму этих элементов.
15. Задан двухмерный массив целых чисел A размером N на M. Найти число элементов A[i,j] < T и произведение этих элементов.

ЛАБОРАТОРНАЯ РАБОТА 5.

ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ПОДПРОГРАММ И МОДУЛЕЙ

Цель лабораторной работы: изучить возможности DELPHI для написания подпрограмм и создания модулей. Составить и отладить программу, использующую внешний модуль UNIT с подпрограммой.

5.1. Описание подпрограмм

Подпрограмма – это последовательность операторов, оформленная таким образом, что ее можно вызвать по имени из любого места программы. При вызове подпрограммы в нее передаются определенные данные, а из нее получают результат вычислений.

В языке Паскаль имеется два типа подпрограмм, различающихся способом обмена данными: процедуры (Procedure) и функции (Function).

Подпрограмма-функция описывается следующим образом.

```
Function <Имя функции>[(<формальные параметры>)]: <тип результата>;
        <описание типов, констант, переменных, вложенных процедур и функций>
      begin
            <операторы>
[Result:=<результат соответствующего типа>]
end;
```

Первая строка, содержащая имя функции называется заголовком. Параметры, которые описываются в заголовке функции называются формальными параметрами. Результат можно передать в основную программу используя ключевое слово Result или имя функции.

Пример 5.1. Составить программу для вычисления суммы квадратов натуральных чисел от 1 до n и оформить ее в виде функции.

```
Function Lux (N : Integer) : Integer;
  Var s, I : Integer;
Begin
     S:=0;
     for i:=1 to N do S:=S + Sqr(i);
   Result:=S;
End;
```

Вызов функции в основной программе может иметь вид

```
      …
w:=Lux (10);
```

где, w - переменная типа Integer.

Для преждевременного выхода из функции используется процедура Exit либо Abort (молчаливое исключение).

Рекомендуется оформлять подпрограмму в виде функции в том случае, когда результатом является значение одной переменной.

Подпрограмма-процедура описывается следующим образом.

```
Procedure <Имя процедуры>[(<формальные параметры)];
  <описание типов, констант, переменных, вложенных процедур и функций>
begin
    <операторы>
end;
```

Обмен данными с основной программой осуществляется с использованием формальных параметров.

Пример 5.2. Составить программу для вычисления суммы квадратов натуральных чисел от 1 до n и оформить ее в виде процедуры.

```
Procedure Lux (N:Integer; Var Sum:Integer);
    Var i:Integer;
Begin
    Sum:=0;
     For i:=1 To N Do Sum:=Sum + Sqr(i);
End;
```

Вызов процедуры в основной программе имеет вид

```
Lux ( 10,s );
```

где, s - переменная типа Integer.

При использовании в качестве параметров процедур данных сложного типа (массивы, множества, записи) в основной программе необходимо предварительно описать имя типа этих данных, которые потом указываются в списке формальных параметров процедуры.

5.2. Передача данных через формальные параметры

Формальные параметры могут быть трех основных разновидностей:

Параметры-значения, параметры - переменные, параметры - константы.

Параметры-значения описываются следующим образом:

```
<Имя процедуры> (a:Тип 1; c, d, e : Тип 2; ...)
```

В момент обращения к процедуре компилятор создает переменные с именами, указанными как формальные параметры и копирует в их значения соответствующих фактических параметров. Никакой связи между формальными и фактическими параметрами не создается. Поэтому любые формальных параметров не приводят к изменению фактических параметров. Фактические параметры таким образом защищены от случайного изменения.

Достоинством передачи данных с помощью формальных параметров является то, что в качестве фактического параметра можно использовать не только переменные, но и константы и арифметические выражения. Например:

```
PAST(a,5,(2*x+3))
```

К недостаткам такой передачи относятся дополнительные затраты памяти (например большой массив) и затраты времени на копирования значений.

Параметры – переменные описываются следующим образом:

Имя процедуры(**Var** a, b : Тип1; **Var** c, d, e : Тип2; ...)

При обращении к процедуре происходит не копирование фактических параметров а передача адреса ячейки памяти в которой находится фактическая переменная. Таким образом внутри процедуры происходит работа не с формальным параметром, а со ссылкой на фактический параметр. Любые изменения формальных параметров внутри процедуры ведут к изменению фактических параметров в основной программе. Используя такие параметры осуществляют передачу результатов выполнения процедуры в основную программу. Достоинством использования параметров-переменных является малая занимаемая память (только под указатель) и экономия времени (не надо перезаписывать данные). Недостаток: при обращении нельзя использовать константы и арифметические выражения.

Параметры – константы описываются следующим образом:

<Имя процедуры> (Const a,b:Тип1;Const c,d,e:Тип2;...)

В этом случае фактическим параметром может быть *переменная, константа или выражение*. Для такого формального параметра новой ячейки не отводится, а при вызове подпрограммы в неё передается адрес ячейки фактического параметра, но внутри запрещены все его изменения.

Кроме рассмотренных трех основных типов формальных параметров можно использовать выходные параметры (out), которые не возможно изменять внутри программы и нетипизированные, которые используются для передачи данных. Параметры со значение по умолчанию позволяют передавать не все необходимые данные. Если данные не были переданы, то подставляется значение по умолчанию.

Кроме использования формальных параметров можно использовать глобальные параметры (видимые и в процедуре и в основной программе).

5.3. Процедурные типы

Язык Object Pascal позволяет объявлять типы и переменные процедурного типа, например

```
type  TPt1 = function(x:integer):integer;
var  Pt1 : Tpt1;
     Pt2 : procedure(var s:extended);
```

Использование процедурных типов позволяет использовать имя подпрограммы в качестве формального параметра.

5.4. Область видимости переменных

Все константы, типы, переменные, процедуры и функции могут использоваться только в определенных местах программы. Часть программы, в которой определен данный конкретный элемент называется областью видимости. При этом часто пользуются понятием блок, который состоит из объявления элемента и составного оператора:

```
        <объявление>
begin
        <операторы>
end;
```

Элементы называются локальными, если область видимости ограничена только блоком, в котором элемент объявлен. Если элемент объявлен и интерфейсной части модуля, то он будет виден во всех модулях, подключивших данный модуль, поэтому такой элемент является глобальным.

Если элемент объявлен в головном файле программы или в подпрограмме, то область его видимости начинается от точки объявления и заканчивается концом текущего блока.

Если элемент объявлен в интерфейсной части модуля, то область видимости будет включает текущий модуль, а так же все модули, которые подключили данный модуль.

Если элемент объявлен в разделе инициализации, то область видимости включает все подпрограммы, расположенные в данном разделе

5.5. Использование модулей

Наборы подпрограмм которые могут быть использованы при разработке целого ряда программ удобно оформлять в виде отдельных тематических библиотек. В разные библиотеки обычно собираются подпрограммы алгоритмов решения задач по определенной теме, например: вычисления всевозможных арифметических функций, обработка массивов и матриц, решение уравнений и др. Для организации таких библиотек в Паскале введены модули.

Модуль – автономно компилируемая программная единица, включающая в себя процедуры, функции, а также различные разделы описаний. Структура модуля представлена в п.1.3 и содержит следующие основные части: заголовок, интерфейсная часть, исполняемая, инициирующая и завершающая (последние две части могут отсутствовать).

Заголовок состоит из зарезервированного слова Unit и следующего за ним имени модуля, которое должно совпадать с именем дискового файла. Использование имени модуля в разделе Uses основной программы приводит к установлению связи модуля с основной программой.

Интерфейсная часть расположена между ключевыми словами interface и implementation и содержит объявление тех конструкций и разделов описаний модуля, которые должны быть доступны другим программам.

Исполняемая часть начинается ключевым словом implementation и содержит описание процедур и функций, объявленных в интерфейсной части. Она может также содержать разделы описаний вспомогательных типов, констант, переменных, процедур и функций, которые будут использоваться только в исполняемой части и не будут доступны внешним программам.

Инициирующая часть начинается ключевым словом initialization и содержит операторы, которые исполняются перед началом выполнения основной программы (может отсутствовать).

Завершающая часть начинается ключевым словом finalization и выполняется в момент окончания работы программы (может отсутствовать).

Заголовок модуля состоит из слова *Unit* и следующего за ним имени модуля, которое служит для связи с другими модулями и основной программой. Имя модуля должно совпадать с именем файла на диске, в который помещается исходный текст модуля. Подключение модулей к разрабатываемой программной единице осуществляется с помощью оператора

Uses имя М1,имя М2,..., имя Пм;

который должен стоять вначале раздела описаний т.е. сразу после заголовков *Program*, *Interface* или *Implementation.*

После подключения модуля в разрабатываемой программе становятся доступными все конструкции, описанные в интерфейсной части модуля.

5.6. Пример написания программы

Задание: Дана квадратная целочисленная матрица N-го порядка. Найти минимальный из элементов, лежащих ниже главной диагонали и максимальный из элементов, лежащих выше побочной диагонали.

5.6.1. Создание модуля

В среде Delphi модули могут создаваться как со своей формой, так и без нее. Для создания нового модуля без своей формы необходимо в меню File выбрать New – Unit. В результате будет создан файл с заголовком Unit Unit2. Имя модуля можно изменить на другое, отвечающее внутреннему содержанию модуля, например Unit MyMath. Для этого необходимо сохранить модуль с новым именем (например MyMath.pas). Следует обратить внимание на то, что имя файла должно совпадать с именем модуля.

5.6.2. Подключение модуля

Для того чтобы подключить модуль к проекту, необходимо в меню Project выбрать опцию Add to Project… и выбрать файл, содержащий модуль. После этого в разделе Uses добавить имя подключаемого модуля – MyMath. Теперь в проекте можно использовать функции, содержащиеся в модуле.

Панель диалога будет иметь вид как и в прошлой задаче(см. рис. 4.1).

Тексты модуля (Листинг 5.1) и вызывающей программы (Листинг 5.2) приведены ниже.

Листинг 5.1

```
unit MyMath;
interface
 type TMas = array[1..10, 1..10] of integer;
 Procedure Matric(A : TMas; n,m : integer; var min, max : integer);
implementation
 Procedure Matric(A : TMas; n,m : integer; var min, max : integer);
  var i,j : integer;
  begin
```

```
   min:=A[2,1];
    for i:=2 to n do
     for j:=1 to i-1 do
      if (A[i,j]<min) then min:=A[i,j];

    max:=A[1,1];
    for i:=1 to n-1 do
     for j:=1 to n-i do
      if (A[i,j]>max) then max:=A[i,j];
 end;
end.
```

Листинг 5.2

```
unit Unit5;
 interface
 uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, Grids, MyMath;
type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    Label1: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Label2: TLabel;
    Button1: TButton;
    Memo1: TMemo;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  A : TMas;  // Объявление двухмерного массива
  n,m : integer;
  i,j : integer;
implementation
{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject);
begin
Memo1.Clear;
  n:=4;  // Число строк в массиве
  m:=4;  // Число столбцов в массиве
```

```
 Edit1.Text:='4';
 Edit2.Text:='4';
  StringGrid1.Cells[0,0]:='A';
 for i:=1 to n do StringGrid1.Cells[0,i]:='i='+IntToStr(i);
 for j:=1 to m do StringGrid1.Cells[j,0]:='j='+IntToStr(j);
end;

procedure TForm1.Button1Click(Sender: TObject);  // Изменить размер
begin
 n:=StrToInt(Edit1.Text);  // Число строк в массиве
 m:=StrToInt(Edit2.Text);  // Число столбцов в массиве
  {Задание числа строк и столбцов в таблице}
   StringGrid1.RowCount:=N+1;
   StringGrid1.ColCount:=M+1;
 for i:=1 to n  do StringGrid1.Cells[0,i]:='i='+IntToStr(i);
 for j:=1 to m do StringGrid1.Cells[j,0]:='j='+IntToStr(j);
end;

procedure TForm1.Button2Click(Sender: TObject);
 var ne, p, min, max : integer;
begin
   Memo1.Clear;
   {Заполнение массива A элементами из таблицы StringGrid1}
  for i:=1 to n do
   for j:=1 to m do
     A[i,j]:=StrToInt(StringGrid1.Cells[j,i]);
   {Расчет}
    Matric(A,n,m,min,max);
     { Вывод результата }
Memo1.Lines.Add('min = '+IntToStr(min));
Memo1.Lines.Add('max = '+IntToStr(max));
end;
end.
```

5.7. Индивидуальные задания

По указанию преподавателя выберите вариант задачи. Составьте программу, оформив вычисления в виде подпрограммы (процедуры или функции). В головной программе произвести ввод исходных данных, вызов подпрограммы и вывод результатов.

1. В вещественной квадратной матрице N-го порядка найти максимальный и минимальный элементы. Переставить строки, в которых они находятся. Если они находятся в одной строке, выдать об этом сообщение.

2. Дана вещественная матрица размером NxM. Переставляя ее строки и столбцы, добиться того, чтобы наибольший элемент (один из них) оказался в левом верхнем углу.

3. Задана одномерная матрица N-го порядка, содержащая нули и целые числа. Заменить нули полусуммой последующего и предыдущего чисел. Если нуль является первым или последним числом матрицы, то его соответственно заменить последующим или предыдущим числом.

4. Определить, является ли заданная матрица N-го порядка магическим квадратом, т.е. такой, в которой сумма элементов во всех строках и столбцах одинакова.

5. Дана целочисленная матрица размером NxM. Найти сумму наименьших элементов ее нечетных строк и наибольших элементов ее четных строк.

6. Дана действительная квадратная матрица N-го порядка. Рассмотрим те элементы, которые расположены в строках, начинающихся с отрицательного элемента. Найти сумму тех из них, которые расположены ниже главной диагонали матрицы.

7. Дана вещественная квадратная матрица N-го порядка. Получить целочисленную квадратную матрицу, в которой элемент равен 1, если соответствующий ему элемент исходной матрицы больше элемента, расположенного на главной диагонали, и равен 0 в противном случае.

8. Дана квадратная целочисленная матрица N-го порядка. Упорядочить элементы в строках по возрастанию.

9. Дана действительная квадратная матрица N-го порядка. Рассмотрим те элементы, которые расположены в строках, начинающихся с отрицательного элемента. Найти сумму тех из них, которые расположены на главной диагонали матрицы.

10. Дана квадратная целочисленная матрица N-го порядка. Упорядочить элементы в столбцах по убыванию.

11. Дана квадратная целочисленная матрица N-го порядка. Найти минимальный элемент среди положительных и максимальный среди отрицательных и их координаты.

12. Дана квадратная целочисленная матрица N-го порядка. Найти суммы элементов, расположенных на линиях, параллельных главной диагонали матрицы и находящихся выше нее.

13. Задана матрица размером NxM. Определить K - количество «особых» элементов матрицы, считая, что элемент «особый», если он больше суммы остальных элементов своего столбца.

14. Дана квадратная целочисленная матрица N-го порядка. Найти сумму элементов тех строк матрицы, у которых на главной диагонали расположены отрицательные элементы.

15. Определить, является ли заданная квадратная матрица n-го порядка симметричной относительно главной диагонали.

Приложение

Средства отладки программ в DELPHI

Практически в каждой вновь написанной программе после запуска обнаруживаются ошибки.

Ошибки первого уровня (ошибки компиляции) связаны с неправильной записью операторов (орфографические, синтаксические). При обнаружении ошибки компилятор DELPHI останавливается напротив первого оператора, 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 характер ошибки. Для быстрого перехода к интересующей ошибке необходимо дважды щелкнуть на строке с ее описанием. Для получения более полной информации о характере ошибки необходимо обратится к HELP нажатием клавиши F1. Следует обратить внимание на то, что одна ошибка может повлечь за собой другие, которые исчезнут при ее исправлении. Поэтому следует исправлять ошибки последовательно, сверху вниз и, после исправления каждой ошибки компилировать программу снова.

Ошибки второго уровня (ошибки выполнения) связаны с ошибками выбранного алгоритма решения или с неправильной программной реализацией алгоритма. Эти ошибки проявляются в том, что результат расчета оказывается неверным либо происходит переполнение, деление на ноль и др. Поэтому перед использованием отлаженной программы ее надо протестировать, т.е. сделать просчеты при таких комбинациях исходных данных, для которых заранее 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) или щелкнуть на серой полосе слева от оператора для обозначения точки прерывания (появится красная точка) и нажать клавишу F9. Выполнение программы будет остановлено на указанной строке. Для просмотра текущих значений можно поместить на нужную переменную курсор (на экране будет высвечено ее значение), либо нажать Ctrl-F7 (окно оценки и модификации) или Ctrl-F5 (окно наблюдения) и в появившимся диалоговом окне указать интересующую переменную. 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 нажать <Run> меню Run или F9.

ЛИТЕРАТУРА

1. Архангельский А.Я. Программирование в Delphi 7. – М.: ЗАО «Издательство БИНОМ», 2003.

2. Фаронов В.В. Delphi 6. Учебный курс. - М.: Издатель Молгачева С.В., 2001.

3. Гленн Дж. Брукшир. Введение в компьютерные науки. – М., СПб, Киев. – «Вильямс», 2001.

4. Синицын А. К., Колосов С. В., Навроцкий А. А.и др. Программирование алгоритмов в среде Delphi: Лаб. практикум. Ч. 1. – Мн., БГУИР, 2004.

5. Колосов С. В. Программирование в Delphi: Учеб. пособие – Мн., БГУИР, 2005.

Св. план 2006,

Учебное издание

Бурцев Анатолий Александрович,
Навроцкий Анатолий Александрович,
Шестакович Вячеслав Павлович

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ OBJECT PASCAL В СРЕДЕ DELPHI

Лабораторный практикум по курсам
«Программирование» и «Основы алгоритмизации и программирования»
для студентов всех специальностей заочной формы обучения

В 2-х частях

Часть 1

Ответственный за выпуск А. А. Навроцкий

Подписано в печать 08.06.2006. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 3,14.
Уч.-изд. л. 2,7. Тираж 350 экз. Заказ 71.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.220013, Минск, П. Бровки, 6