МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего профессионального образования
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ
"МИСиС"
НОВОТРОИЦКИЙ ФИЛИАЛ

Кафедра Прикладной информатики и управляющих систем автоматики

В.И. Юдина

ИНФОРМАТИКА

часть I

Лабораторный практикум

Новотроицк, 2011 г

УДК 681.3.06
ББК 32.973.26-018.1
Ю16

Рецензенты:

Профессор кафедры вычислительной техники и прикладной математики ГОУ ВПО "Магнитогорский государственный технический университет им. Г.И. Носова", доцент, к.т.н.
И.М. Ячиков

Доцент кафедры прикладной информатики и управляющих систем автоматики ФГАОУ ВПО "Национальный исследовательский технологический университет "МИСиС" Новотроицкий филиал, к.т.н.
С.Н. Басков

Информатика. Часть I / В.И. Юдина: Лабораторный практикум. – Новотроицк: НФ НИТУ «МИСиС», 2011. – 77 с.

Методические указания предназначены для приобретения практических навыков алгоритмизации задач и программирования на языке Pascal в рамках курса "Информатика". На простых примерах излагаются основные правила программирования на языке Pascal с использованием разных типов данных.

Методические указания предназначены для студентов всех направлений

Рекомендовано Методическим советом НФ НИТУ «МИСиС»

Содержание

ВВЕДЕНИЕ

Лабораторный практикум (ЛП) по дисциплине "Информатика" (ч.I) включает 8 лабораторных работ (ЛР).

Порядок выполнения практикума.

1. ЛР выполняются в компьютерном классе.
2. По каждой работе составляется отчет.
3. Производится защита ЛР на консультации.

Инструкции к ЛР имеют единую структуру.

1. Теоретическое введение
2. Учебный пример
3. Задания для самостоятельной работы
4. Контрольные вопросы

Теоретическое введение содержит краткое описание необходимых для выполнения работы сведений по соответствующей теме курса. Содержимое данного раздела может быть использовано для ответа на Контрольные вопросы.

Учебный пример. Данный раздел содержит: блок-схему и исходный текст программы, служащие практическим примером к теме работы, а также подробный разбор выполняемых программой действий. Блоки и соответствующие этим блокам фрагменты программы обозначены одинаковыми цифрами.

Задания для самостоятельной работы. Самостоятельная часть ЛР выполняется по вариантам. № варианта соответствует № студента в списке группы в Журнале учета лабораторных работ.

Контрольные вопросы. На данные вопросы студент должен уметь ответить при защите ЛР. При подготовке к защите следует использовать текст настоящих Методических указаний, лекции и учебники.

ОБЩИЕ ТРЕБОВАНИЯ

1 Условия допуска студента к лабораторным работам

1. Студент должен пройти инструктаж по технике безопасности и правилам поведения в компьютерном классе.

2. На занятии студент должен иметь настоящие Методические указания и тетрадь для отчетов по ЛР.

2 Порядок выполнения работы

Выполнение каждой ЛР включает 3 этапа.

1. Домашняя подготовка:

- изучить теоретическое введение к лабораторной работе, при необходимости воспользоваться конспектом лекции по соответствующей теме и/или учебником;
- разобрать учебный пример по блок-схеме (понять назначение каждого блока и логику алгоритма) и по исходному тексту программы (понять назначение и характер действий, выполняемых каждой структурной единицей программы и их логические взаимосвязи);
- установить соответствие структурных единиц блок-схемы и исходного текста программы;
- продумать алгоритм решения задачи по своему варианту.

2. Работа в лаборатории:

- набрать на компьютере и отладить программу учебного примера на предлагаемом тестовом варианте;
- пользуясь учебным примером, составить блок-схему программы из раздела "Самостоятельная работа" по своему варианту; текст программы набрать на компьютере, отладить на тестовых данных.

3. Оформление отчета и подготовка ответов на контрольные вопросы.

3 Сохранение исходных текстов программ

Тексты программ, созданных в ЛП, сохраняются каждым студентом в личной папке. Личная папка создается на общем ресурсе, указываемом преподавателем.

Необходимо также иметь копию личной папки на дискете или флэш-диске.

4 Отчет о работе

По каждой ЛР в тетради для отчетов должен быть составлен отчет. В отчет включаются только задания из раздела "Самостоятельная работа" (см. Приложение А).

Структура отчета:

1 № ЛР
2 Тема работы
3 Цель работы
4 № варианта
5 Задание для самостоятельной работы
6 Блок-схема алгоритма

7 Исходный текст программы
8 Тест
9 Полное имя файла, содержащего исходный текст программы.

Отчет следует оформлять четко и аккуратно. Несоблюдение данного условия, а также структуры отчета, является основанием для недопуска работы к защите.

5 Защита работы

5.1. Процесс защиты работы:
- предъявить программу на компьютере и результаты ее работы;
- предъявить отчет;
- объяснить алгоритм работы программы;
- устно ответить на контрольные вопросы.

5.2. Условия зачета ЛР. ЛР считается зачтенной при наличии:
- правильно работающей программы;
- отчета, составленного в соответствии с рекомендациями;
- верных ответов на Контрольные вопросы.

Отметка о зачете проставляется в Журнале ЛР и в тетради для отчетов

5.3. Порядок защиты лабораторных работ.

Защита каждой работы производится на консультации, непосредственно следующей за занятием, на котором выполнялась работа. ЛР может быть защищена на занятии после ее выполнения при наличии отчета.

На последней консультации производится защита отдельных работ, не зачтенных в срок.

ЛАБОРАТОРНАЯ РАБОТА № 1

ТЕМА: РАБОТА В ИНТЕГРИРОВАННОЙ СРЕДЕ РАЗРАБОТКИ TURBO DELPHI

Цель работы: изучить основные возможности консольного режима интегрированной среды разработки Turbo Delphi и научиться создавать простейшие программы на языке Pascal.

1 Теоретическое введение

1.1 Интегрированная среда разработки Turbo Delphi (ИСР)

ИСР служит для организации взаимодействия с программистом и включает в себя ряд окон, содержащих различные управляющие элементы. С помощью средств интегрированной среды разработчик может удобно проектировать интерфейсную часть приложения, а также писать программный код и связывать его с управляющими элементами. При этом вся работа по созданию приложения, включая отладку, происходит в интегрированной среде разработки.

Delphi является однодокументной средой и позволяет одновременно работать с одним приложением (проектом приложения). Название проекта выводится в строке заголовка главного окна в верхней части экрана.

Turbo Delphi является приложением операционной системы Windows, поэтому пользовательский интерфейс данного приложения (приемы взаимодействия пользователя с программой) ничем не отличается от интерфейса прочих окон Windows.

Запуск приложения: **ПУСК - Программы - Borland Developer Studio 2006- Turbo Delphi.**

Заголовок окна содержит имя приложения - Turbo Delphi.

Главное окно Delphi (рисунок 1) включает:

- главное меню;
- панели инструментов.

Рисунок 1 - Главное меню и панели инструментов Turbo Delphi

Главное меню содержит обширный набор команд для доступа к функциям Delphi. Команды, используемые в настоящем практикуме, представлены в таблице 1.

Панели инструментов находятся под главным меню в левой части главного окна и содержат ряд кнопок для вызова наиболее часто используемых команд главного меню, например, *File-Open* (Файл-Открыть) или *Run-Run* (Выполнение-Выполнить).

Вызвать многие команды главного меню можно также с помощью комбинаций клавиш, указываемых справа от названия соответствующей команды. Например,

команду *Run-Run* (Выполнение-Выполнить) можно вызвать с помощью клавиши <F9>.

Таблица 1- Способы выполнения наиболее распространенных действий в ИСР Turbo Delphi

<table>
<tr><th rowspan="2">№ п/п</th><th rowspan="2">Сущность действия</th><th colspan="2">Способ действия</th></tr>
<tr><th>Команды</th><th>Горячие клавиши или кнопки на панелях инструментов</th></tr>
<tr><td>1.</td><td>Создание нового проекта (исходного текста новой программы)</td><td>File-New-Other-Console Application-OK (Файл-Новый-Другие-Консольное приложение)</td><td>New items- Console Application-OK на панели Standard</td></tr>
<tr><td rowspan="2">2.</td><td rowspan="2">Первоначальное сохранение текста программы на диске</td><td>File-Save All (Файл-Сохранить все)</td><td>Save All на панели Standard</td></tr>
<tr><td colspan="2">открыть папку, в которой будет сохранен проект, задать нужное имя проекта, нажать кнопку Сохранить</td></tr>
<tr><td>3.</td><td>Повторное сохранение существующего на диске текста программы с прежним именем</td><td>File-Save (Файл-Сохранить)</td><td>Кнопка Save на панели Standard или CTRL+S</td></tr>
<tr><td rowspan="2">4.</td><td rowspan="2">Сохранение существующего на диске текста программы с другим именем</td><td colspan="2">File-Save As (Файл-Сохранить как)</td></tr>
<tr><td colspan="2">открыть папку, в которой будет сохранен проект, задать нужное имя проекта, нажать кнопку Сохранить</td></tr>
<tr><td>5.</td><td>Завершение работы с текущим проектом</td><td>File-Close (Файл-Закрыть)</td><td></td></tr>
<tr><td rowspan="2">6.</td><td rowspan="2">Открытие существующего (ранее сохраненного на диске текста программы) проекта</td><td>File-Open (Файл-Открыть)</td><td>Кнопка Open на панели Standard</td></tr>
<tr><td colspan="2">открыть папку, в которой сохранен проект, выбрать нужное имя проекта, нажать кнопку Открыть</td></tr>
<tr><td>7.</td><td>Компиляция программы</td><td>Project - Compile Project (Проект - Компиляция проекта)</td><td>CTRL + F9</td></tr>
<tr><td>8.</td><td>Выполнение программы</td><td>Run-Run (Выполнение - Выполнить)</td><td>F9 или кнопка Run на панели Debug (отладка)</td></tr>
<tr><td>9.</td><td>Завершение приложения при его зацикливании (зависании)</td><td>Run - Program Reset (Выполнение - Остановить программу)</td><td>CTRL + F2 или кнопка Program Reset на панели Debug (отладка)</td></tr>
</table>

1.2 Структура программы на языке Pascal

Программа на языке Pascal состоит из трех частей: *заголовок программы, раздел описаний* и *раздел операторов* (рисунок 2).

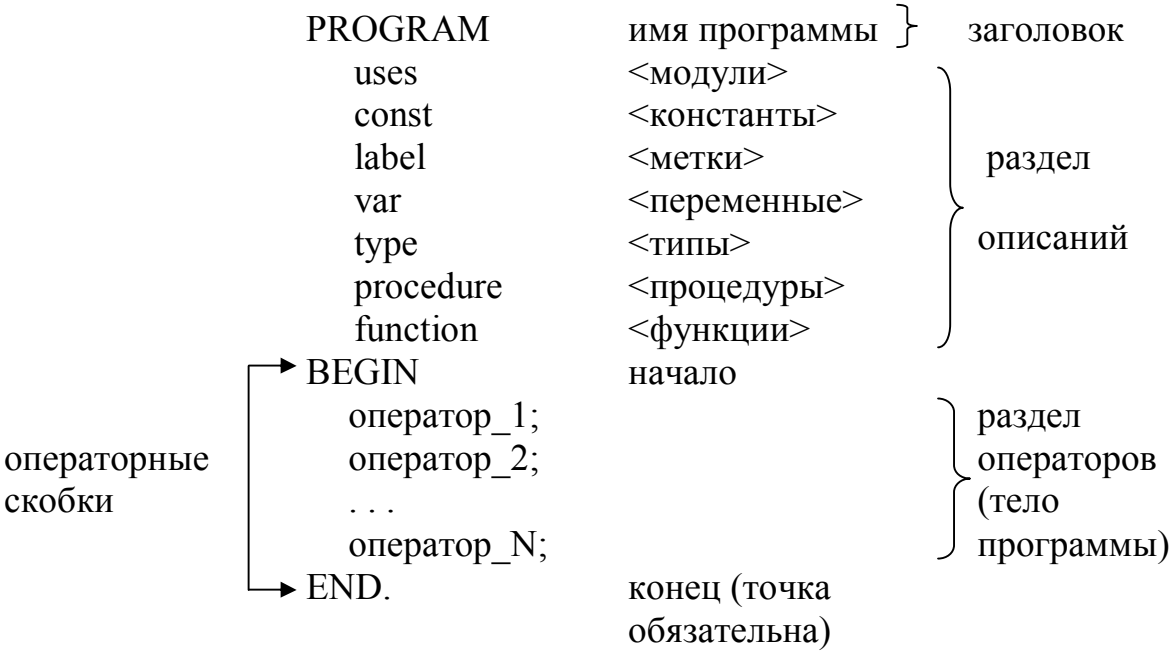


Рисунок 2 – Структурные элементы Pascal-программы

Заголовок содержит служебное слово PROGRAM и имя программы, задаваемое программистом, заканчивается заголовок точкой с запятой (;).

Раздел описаний предназначен для объявления всех используемых в программе данных и их характеристик (имена, типы и др.). Объявление означает резервирование места в памяти для соответствующих констант, переменных, и т.д. в соответствии с их типом. Необъявленные структурные элементы невозможно использовать в программе.

Раздел операторов (тело программы) заключается в операторные скобки BEGIN … END и содержит операторы, необходимые для выполнения последовательности действий для решения поставленной задачи.

Элементы раздела описаний, заключенные в треугольные скобки < >, могут отсутствовать, т.е. каждый из них вставляется, только если он используется в теле программы. Разделителем между разделами описательной части и операторами служит точка с запятой (;). В конце программы должна стоять точка. Язык Турбо-Паскаль является языком свободного формата, что позволяет размещать в строке как один, так и несколько операторов.

1.3 Порядок разработки программ

В общем случае решение задачи на ЭВМ можно разбить на следующие этапы:

- постановка задачи;
- создание математической модели;

- разработка алгоритма в виде блок-схемы;
- составление исходного текста (кода) программы на языке программирования;
- трансляция программы (перевод исходного кода с языка программирования в машинный код - создание исполнимой программы);
- отладка и выполнение программы с тестовыми данными;
- анализ результатов.

Тестовыми называются данные, позволяющие проверить правильность работы алгоритма. В качестве тестовых следует использовать такие данные, чтобы легко было "вручную" подсчитать ожидаемый результат.

1.4 Встроенные процедуры и функции языка Pascal

Встроенные (стандартные) процедуры и функции являются частью языка и могут вызываться по имени без предварительного описания (они описаны в библиотечных модулях языка, которые подключаются автоматически при запуске ИСР). Например, *abs, sqrt, ln, sin* - функции (возвращают результат), *readln, write* – процедуры (не возвращают результат). Их наличие существенно облегчает разработку прикладных программ. Запись математических функций производится аналогично их записи в математике.

Вызов встроенной функции:

имя_функции (аргумент)

Функция возвращает результат в точку вызова. Например:

sqr(x) – возведет в квадрат значение x и возвратит в точку вызова вычисленное значение квадрата числа;

y:= sqr(x); при x = 10 y:=sqr(10); y:=100.

Стандартные функции могут быть использованы в выражении:

*y:=a+b*sin (c);*

В свою очередь, выражение может быть использовано в качестве аргумента:

cos(a-d/2)

Некоторые математические функции и примеры их использования представлены в Приложении В.

1.5 Процедуры ввода

Ввод/вывод связан с обменом информацией между оперативной памятью и внешними устройствами. Для реализации указанных функций в языке Паскаль имеются встроенные процедуры *read, readln, write, writeln*, использующие стандартные файлы ввода/вывода. Стандартным файлом ввода является клавиатура, а вывода – экран.

Для ввода данных с клавиатуры применяются процедуры:

read (<список ввода>);
readln (<список ввода>);

Список ввода – это последовательность из одной или более переменных. При вводе числовых переменных процедура *read* вначале выделяет подстроку во

входном потоке по следующему правилу: все ведущие пробелы, символы табуляции и маркеры конца строки пропускаются, выделяется первый значащий символ, признаком конца подстроки является любой из вышеперечисленных символов или символ конец файла. Выделенная таким образом подстрока рассматривается как символьное представление числовой константы, которое преобразуется в соответствии с типом переменной, и полученное значение присваивается переменной. Если значащих символов в строке нет, а список ввода еще не исчерпан, то автоматически осуществляется переход к новой строке. Процедура *readln* идентична процедуре *read* за исключением того, что после считывания последней переменной, оставшаяся часть строки пропускается, так что следующее обращение к *read* или *readln* начнется с первого символа новой строки. Рассмотрим следующий фрагмент программы:

var a,b:real;
begin ... read(a,b); ... end.

Пусть, требуется ввести числа *1,3; 13,3*. Информация набирается в виде:
1.3 13.3

на одной или нескольких строках. Если набираем на одной строке, то между числами вводятся пробелы, после последнего числа - [Enter]. Если на нескольких строках - то клавиша [Enter] нажимается после каждого числа. При вводе чисел с дробной частью в качестве разделителя используется точка.

В результате выполнения процедуры *read* переменной *a* присвоится значение *1.3*, переменной *b* – значение *13.3*.

1.6 Процедуры вывода

Для вывода данных на экран используются процедуры:

writeln(<список вывода>);
write(<список вывода>).

Где *writeln* и *write* - операторы вызова процедуры;

(*<список вывода>*)- фактические параметры, передаваемые процедуре при ее вызове.

Список вывода - это переменные, константы, текстовые сообщения. Отличие процедуры *writeln* от *write* состоит в том, что в процедуре *write* курсор остается на той же строке экрана за последним выведенным символом, а в *writeln* курсор переходит на начало следующей строки (*writeln* состоит из букв двух английских слов: write - писать и line - строка).

Предположим, что переменные *a,b,c* описаны как целые и имеют соответственно значения 132, 25, -37. После выполнения процедур

writeln(a,b);write(c);

a и *b* будут напечатаны на одной строке, а *c* – на другой:

132 25
-37

Кроме того, процедура вывода предоставляет возможность определить формат вывода, т.е. указать в выводимом числе общее количество позиций и сколько из них после запятой (последнее только для чисел с плавающей точкой):

writeln(a:3,b:5,c:4);

В этом случае при печати под значение переменной *a* отводится 3 позиции, *b* – 5 позиций, под *c* – 4 позиции, т.е. будет выведено 132 25 -37.

Если количество литер в представлении выводимого значения меньше, чем указано в процедуре, то оно слева дополняется пробелами. Если количество указанных позиций недостаточно, то происходит автоматическое увеличение поля до необходимых размеров:

writeln('a=',a:1,'b=',b:2,'c=',c:3);

В последнем примере используется возможность вывода строк символов, при этом будет напечатано:

a=132b=25c= -37

Пусть переменные d, e, f описаны как вещественные и имеют соответственно значения 13,13; 123,45; -987,654. Результатом работы оператора

writeln('d=',d:5:2,'e=',e:8:3,'f=',f:8:2);

будет строка

d=13.13e= 123.450f= -987.65

Если для чисел с плавающей точкой указывается только количество позиций в числе, без указания числа позиций после запятой, то в этом случае числа выводятся в экспоненциальной форме, занимая указанное количество позиций. Если длина поля не указывается совсем, то под каждое число отводится стандартная длина поля и числа печатаются в экспоненциальной форме:

writeln('d=',d:10,' e=',e:9,' f=',f);

d= 1.313E+01 e= 1.23E+02 f=-9.8765000000E+02

Употребление *writeln* без параметров приводит к выводу пустой строки, и переводу курсора в следующую строку.

Встроенные процедуры не нуждаются в предварительном описании и доступны любой программе.

2 Учебные примеры

2.1 Создание папки для сохранения результатов работы

Тексты программ, относящихся к каждой ЛР, должны быть сохранены в отдельных папках личной папки студента: Лаб1, Лаб2, …, Лаб8.

Создать иерархию папок для сохранения результатов выполнения ЛР:

Личная папка создается на общем ресурсе, указанном преподавателем.

Имя папки - **Ваша_Фамилия_Имя.**

В личной папке создать папки **Лаб1, Лаб2, . . ., Лаб8.**

Все программы, созданные в настоящей лабораторной работе, должны быть сохранены в папке **Лаб1.**

2.2 Запуск приложения **Turbo Delphi** (см. п. 1.1 настоящего Руководства)

2.3 Создание нового проекта (см. Таблица 1, п.1).

В центре рабочей области будет выведено окно, в заголовке которого содержится имя файла по умолчанию (т.е. так определено разработчиком ИСР) -

Project1.bdsproj.

В окне содержится текст:

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;                                  } (1)
begin
  { TODO-User-Console Main: Insert code here}    (2)
end.
```

Часть текста (1), обведенную фигурной скобкой, не следует изменять. Раздел операторов Ваших программ следует записывать в части (2), т.е. между begin и end, предварительно удалив строку

{ TODO - User – Console Main: Insert code here }.

Итак, удалите названную строку и запишите текст Вашей первой программы:

```
writeln ('Ya izuchayu Pascal');
readln;
```

В данной программе процедуре *writeln* передается фактический параметр - текстовая строка **'Ya izuchayu Pascal'.** Текстовые строки заключаются в апострофы (' ') и выводятся на экран в неизменном виде. **(**Буквы кириллицы в ИСР Turbo Delphi отображаются некорректно, поэтому использовать их даже в текстовых строках не рекомендуется).

readln - процедура, позволяющая осуществить ввод в программу (переменной, символа и т.д.) в ходе ее выполнения. В данном случае никакого ввода не предполагается. Использование процедуры имеет служебное значение и предназначено для задержки окна, содержащего результаты выполнения программы (иначе сразу после вывода результатов оно будет автоматически закрыто).

Важно точно написать текст, в том числе пробелы, скобки, апострофы.

2.4 Сохранить текст программы в **Вашей** папке **Лаб1** (см. Таблица 1, п. 2). Имя не изменять - **Project1**.

2.5 Выполнить программу (см. Таблица 1, п. 8).

В отдельном окне выводится результат выполнения программы: текстовая строка Ya izuchayu Pascal

Просмотреть результат выполнения, представить на проверку преподавателю. Закрыть окно выполнения.

2.6 Закрыть текущий проект (см. Таблица 1, п. 5).

2.7 Создать новый проект (см. Таблица 1, п. 1).

Процедура *writeln*, как мы видели, позволяет вывести на экран содержимое, заключенное в скобки. Причем это может быть не только текстовая строка, но и, например, пробелы или результат вычисления арифметического или иного выражения.

Между *begin* и *end* напишите следующий фрагмент:

writeln ('100 + 120 = ', 100+120);
readln;

Сохранить текст в файле **Project2** (см. Таблица 1, п. 2).
Выполнить программу (см. Таблица 1, п. 8).
В отдельном окне выводится результат выполнения программы:
100 + 120 = 220,
где 100 + 120 = - текстовая строка,
220 - результат вычисления выражения 100+120.
Просмотреть результат выполнения, представить на проверку преподавателю. Закрыть окно выполнения.
Закрыть текущий проект (см. Таблица 1, п. 5).

2.8 Создать новый проект (см. Таблица 1, п. 1).
Между *begin* и *end* напишите следующий фрагмент:
writeln (cos (2));
readln;

Сохранить текст в файле **Project3** (см. Таблица 1, п. 2).
Выполнить программу (см. Таблица 1, п. 8).
В отдельном окне выводится результат выполнения программы:
- 4.16146836547142E-0001 - результат вычисления косинуса 2-х радиан.
В данной программе использована встроенная функция *cos(x).*
Просмотреть результат выполнения, представить на проверку преподавателю. Закрыть окно выполнения.
Закрыть текущий проект (см. Таблица 1, п. 5).

3 Задания для самостоятельной работы

Задания данного раздела необходимо описать в отчете по ЛР.
Тексты программ сохранять в файлах **Project4 - Project7.**

Задание 1. Вычислить значение выражения:
$\sqrt{(10^2 + 15^2)}$
Результат вывести в формате: всего 4 знака, после запятой - 2.

Задание 2. Написать программу, которая напечатает слово "MISiS" в 5-й строке экрана.

Задание 3. Вычислить и вывести примерно в середине строки значение выражения $10^2 + 7^2$.

Задание 4. Используя алгоритм действий, описанный в п. 2.7, процедуру writeln, а также встроенные процедуры и функции языка Pascal, приведенные в Приложении 3, вычислить значение выражения.
Каждый студент выполняет одно из заданий из таблицы 2 по варианту. № варианта = № студента в Журнале лабораторных работ.

Таблица 2 – Задания к лабораторной работе №1

Вариант	Выражение	Параметры
1	sin(x)	x = π; 1; -1
2	abs(x)	x = 5; -10; 0
3	cos(x)	x = 1; -1, π
4	sqr(x)	x= 12; 5; 9
5	sqrt(x)	x = 81; 144; 225
6	odd(x)	x = 15; 22; 0
7	frac(x)	x = 20,567
8	trunk(x)	x = 20,567
9	round(x)	x = 3,7; 3,2; -0,78
10	pi	
11	int(x)	x = 3,7; 3,2; -0,78
12	inc(x)	x = 8; 10; 12

4 Контрольные вопросы

1. Способы выполнения команд ИСР.
2. Способы создания нового проекта.
3. Как выполняется сохранение исходного текста программы на диске?
4. Как выполняется завершение работы с текущим проектом?
5. Как открыть проект, ранее сохраненный на диске?
6. Какова структура программы на языке Pascal?
7. Какие встроенные процедуры и функции языка Pascal вы знаете?
8. Назначение процедуры writeln.
9. Формат процедуры writeln.
10. Объясните действие, реализуемое каждым оператором, во всех программах.

ЛАБОРАТОРНАЯ РАБОТА № 2

ТЕМА: **ПРОГРАММИРОВАНИЕ АЛГОРИТМА РАЗВЕТВЛЯЮЩЕЙСЯ СТРУКТУРЫ**

Цель работы: овладение практическими навыками разработки, программирования алгоритма разветвляющихся структур, получение дальнейших навыков по отладке и тестированию программы.

1 Теоретическое введение

1.1 Разветвляющиеся программы

Разветвляющийся алгоритм – это алгоритм, содержащий хотя бы одно условие; он позволяет, в зависимости от условий, выполнять команды, содержащиеся в ветвях алгоритма. К *разветвляющимся программам* приводят задачи, в которых, в зависимости от некоторого условия, вычисления производятся тем или иным путем. Пусть необходимо вычислить значение *y* по формуле:

$$y = \begin{cases} x^3 + 3, \text{если } x < 0, \\ x * \sin x, \text{если } x \geq 0. \end{cases}$$

Пример программы и блок-схема алгоритма приведены на рисунке 3.

```
Program Pr1;
  var   x,y:real;
  begin
    writeln('vvedite x:');
    readln(x);
    if x<0
      then
          y:=x*x*x+3
      else
          y:=x*sin(x);
     writeln(y);
end.
```

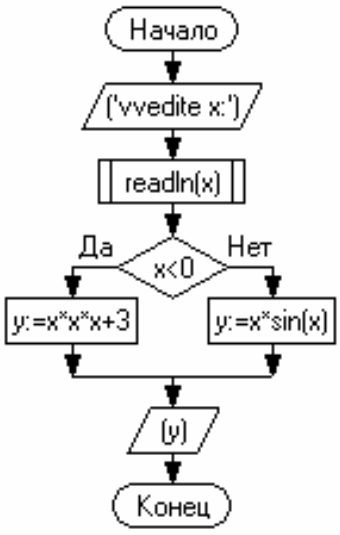


Рисунок 3 - Исходный текст программы вычисления функции *y* и графическая схема алгоритма

В этой программе встречается условный оператор *if*, он служит для выбора формулы вычисления *y* в зависимости от введенного значения *x*.

1.2 Условный оператор

Условный оператор - это оператор, который выполняет команду или группу команд в зависимости от определенного условия. Условный оператор служит для ветвлений в программе и имеет полную и сокращенную форму записи.

Синтаксис (формат) полной формы:

IF *<условие>* ***THEN*** *<оператор-1>* ***ELSE*** *<оператор-2>;*

где *if, then, else* – ключевые слова (перев. с *англ.* если, то, иначе соответственно);

<условие> – логическое выражение типа сравнения (например, *a>b, c<=d, f=1*);

<оператор-1> и *<оператор-2>* – любой оператор Турбо-Паскаля.

Оператор работает следующим образом:

если *<условие>* истинно (true), то выполняется *<оператор-1>*(ветвь "да") и управление передается на следующий за условным оператор;

если *<условие>* ложно (false), то выполняется *<оператор-2>* (ветвь "нет") и управление передается на следующий за условным оператор.

Таким образом, всегда выполняется один из двух операторов: либо из ветви *then*, либо из ветви *else*. Фрагмент программы, содержащий полную форму условного оператора, и блок-схема алгоритма данного фрагмента приведены на рисунке 4.

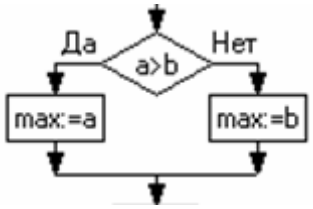
Фрагмент программы	Блок-схема
if a>b then *max:=a* *else* *max:=b;*	

Рисунок 4 - Полная форма оператора IF

Синтаксис (формат) сокращенной формы (в ней отсутствует ветвь *else)*:

IF *<условие>* ***THEN*** *<оператор-1>;*

Оператор работает следующим образом:

если *<условие>* истинно (true), то выполняется *<оператор-1>* и управление передается на следующий за условным оператор;

если *<условие>* ложно (false), то управление сразу передается на следующий за условным оператор.

Таким образом, в зависимости от условия <оператор-1> либо выполняется, либо не выполняется. Фрагмент программы, содержащий полную форму условного оператора, и блок-схема алгоритма данного фрагмента приведены на рисунке 5.

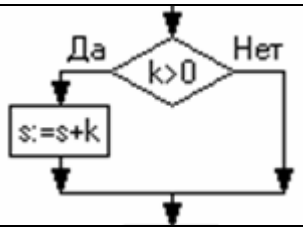
Фрагмент программы	Блок-схема
if k>0 then *s:=s+k;*	

Рисунок 5 - Сокращенная форма оператора IF

1.3 Вложенные условные операторы IF

Оператор *if* является *вложенным*, если он вложен, т.е. находится внутри другого оператора *if*.

if *условие-1* ***then*** *оператор-1*
else if *условие-2* ***then*** *оператор-2*
else *if условие-n* ***then*** *оператор-N*
else *оператор-X;*

В таком варианте оператор *if* может содержать произвольное количество предложений *else if* и только одно предложение *else*. *Оператор-1* выполняется, когда значение выражения *условие-1* равно *true*; *оператор-2* выполняется, когда *условие-1* ложно, а *условие-2* истинно. И, наконец, *оператор-N* выполняется, когда все предыдущие условия ложны, а *условие-N* истинно. *Оператор-X* выполняется, только если все условия (от *условие-1* до *условие-N*) ложны. Другими словами, каждое *else* соответствует тому *if*, которое ему непосредственно предшествует.

Фрагмент программы, содержащий вложенный условный оператор, и блок-схема алгоритма данного фрагмента приведены на рисунке 6.

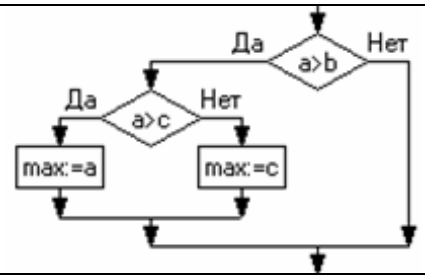
Фрагмент программы	Блок-схема
if a>b then *if a>c then* *max:=a* *else max:=c;*	

Рисунок 6 - Вложенный условный оператор

1.4 Составной оператор

Когда необходимо добиться того, чтобы последовательность операторов работала как единый оператор, можно помещать эту последовательность между ключевыми словами *begin* и *end*. Такая конструкция называется *составным оператором*, или операторными скобками: *begin* открывает скобку, *end* – закрывает. Фрагмент программы, содержащий составной оператор, и блок-схема алгоритма данного фрагмента приведены на рисунке 7.

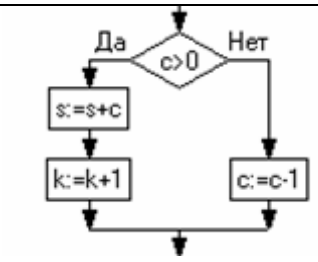
Фрагмент программы	Блок-схема
if c>0 then *begin* *s:=s+c;* *k:=k+1* *end* *else* *c:=c-1;*	

Рисунок 7 - Составной оператор

2 Учебный пример

Составить программу для вычисления значения функции *S*:

$$s = \begin{cases} cos\ at & \text{если } at<=1 \\ a\ sin\ at & \text{если } at>1 \end{cases}$$

Исходный текст программы вычисления функции *s* (Program lab_2) и блок-схема алгоритма представлены на рисунке 8.

```
Program lab_2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
Var t,s: real;
Const a = 0.2;
Begin
Writeln ('vvedite  t');
Read (t);
If a*t<=1
        then s:=cos(a*t)
        else s:=a*sin(a*t);
Writeln ('a = ', a:8:3, 't = ', t:8:3);
Writeln ('s = ', s:8:4);
read (t);
end.
```

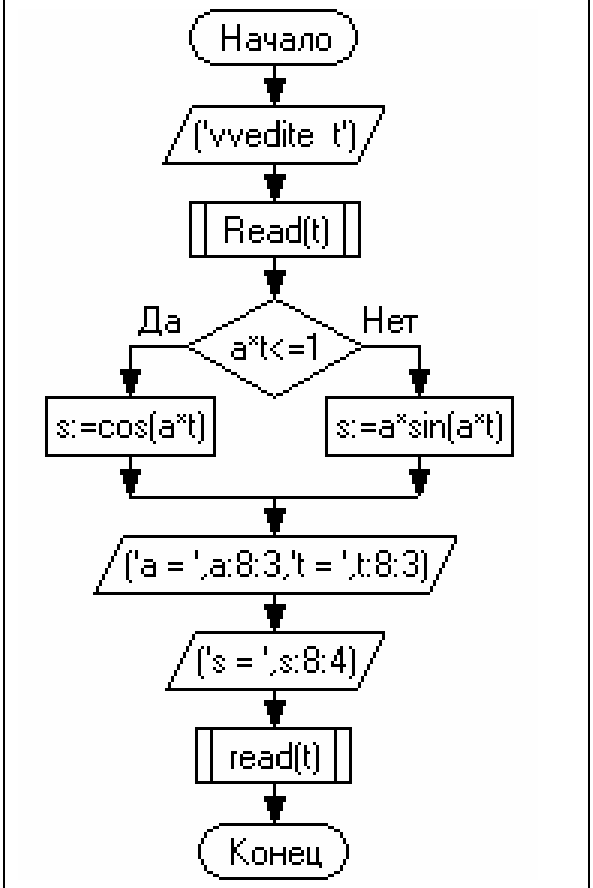


Рисунок 8 - Исходный текст программы вычисления функции *s* и блок-схема алгоритма

ТЕСТ:

1. t = 1 - это значение вводится при первом выполнении программы,
   s = 0,9801 - это значение должно получиться (вычисляется cos(0.2*1) - по ветви then)
2. t = 10 - это значение вводится при втором выполнении программы,
   s = 0,1819 - это значение должно получиться (вычисляется sin (1*2)(-0.2*1) - по ветви else).

Подробное описание работы программы, включающее комментарии к каждой строке, представлено в таблице 3.

Таблица 3 - Описание работы программы вычисления функции *s*

Строка программы	Комментарий
Var t,s: real;	Объявление переменных *t* и *s*, используемых в программе
Const a = 0.2;	Объявление константы *a*, используемой в программе
Begin	Начало программы
Writeln ('vvedite t');	Выводит текстовую строку, заключенную в апострофы
Read (t);	Позволяет пользователю ввести значение переменной *t* (взять из раздела ТЕСТ)
If a*t<=1 then s:=cos(a*t)	Проверка условия: если введенное значение *t* таково, что произведение *a*t* не больше 1 (условие истинно), то вычисляется выражение, стоящее справа от оператора присваивания и результат вычисления присваивается переменной *s*
else s:=a*sin(a*t);	*Иначе* (условие ложно) - вычисляется выражение, стоящее справа от оператора присваивания и результат вычисления присваивается переменной *s*
Writeln ('a = ', a:6:3, 't = ', t:6:3);	Выводит текстовую строку **a =** , рядом - значение *a*, заданное константой, в формате: 6 знакомест, из них 3 после запятой; далее аналогично для *t*
Writeln ('s = ', s:8:4);	Выводит текстовую строку **s=**, рядом - значение переменной *s*, вычисленное выше
read (t);	Имеет служебное значение, позволяя задержать окно с результатом
End.	Конец программы

3 Задания для самостоятельной работы

1. Составить программу обработки алгоритма разветвляющейся структуры (таблица 4); исходные данные - *a, b, c* - объявить как константы.
2. Набрать ее текст, сохранить в **Вашей** папке **Лаб2** в файле **Project2**.
3. Выполнить программу на компьютере.
4. Проверить правильность работы программы на двух тестовых вариантах в соответствии с числом ветвей вычислительного процесса.
5. Составить отчет (см. Приложение Б).

Таблица 4 - Задания к лабораторной работе № 2

Вариант	Функция	Условие	Константы
1	$Y = \begin{cases} ax^2 \ln x \\ x \cos bx \end{cases}$	x<2 x>=2	a=- 0,2 b=2
2	$Q = \begin{cases} \pi x^2 - 7/x^2 \\ \lg(x) + 7\sqrt{x}) \end{cases}$	x<1,3 x>=1,3	
3	$W = \begin{cases} ax^2 + bx + c \\ (a + bx)/\sqrt{x^2 + 1} \end{cases}$	x<1,2 x>=1,2	a=2,8 b=-0,3 c=4
4	$Q = \begin{cases} \pi x^2 - 7/x^2 \\ \lg(x) + 7\sqrt{\lvert x + a \rvert}) \end{cases}$	x<1,4 x>=1,4	a=1,65
5	$Y = \begin{cases} (x - 2)^2 + 6 \\ 3tgx \end{cases}$	x<2 x>=2	a=2,3
6	$W = \begin{cases} x\sqrt{a - x} \\ (x-a) \cos ax \end{cases}$	x<a x>=a	a=2,5
7	$Q = \begin{cases} bx - \lg bx \\ bx + \lg bx \end{cases}$	bx<1 bx>=1	b=1,5
8	$F = \begin{cases} \lg(x +1) \\ \sin^2 \sqrt{\lvert ax \rvert} \end{cases}$	x>1 x<=1	a=20,3
9	$Y = \begin{cases} \sin x \lg x \\ \cos^2 x \end{cases}$	x>3,5 x<=3,5	-
10	$S = \begin{cases} (a + b)^2/\cos x \\ (a + b)/(x + 1) \end{cases}$	x<2,8 x>=2,8	a=2,6 b=-0,39
11	$Y = \begin{cases} a \lg x + \sqrt{\lvert x \rvert} \\ 2a \cos x + 3x^2 \end{cases}$	x>1 x<=1	a=0,9
12	$W = \begin{cases} a/i + bi^2 \\ ai + bi^3 \end{cases}$	i<=6 i>6	a=2,1 b=1,8

4 Контрольные вопросы

1. Когда применяется алгоритм разветвляющейся структуры?
2. Какие виды оператора IF существуют?
3. Когда применяется полная форма оператора IF?
4. Каков формат полной формы оператора IF? Его блок-схема?
5. Опишите процесс выполнения условного оператора полной формы. Приведите пример.
6. Когда применяется сокращенная форма оператора IF?
7. Каков формат сокращенной формы оператора IF? Его блок-схема?
8. Опишите процесс выполнения условного оператора сокращенной формы. Приведите пример.
9. Дайте определение и приведите пример вложенного условного оператора
10. Зачем необходимо при отладке программы тестировать все ветви алгоритма?
11. Для чего используется составной оператор?

ЛАБОРАТОРНАЯ РАБОТА № 3

ТЕМА: **ПРОГРАММИРОВАНИЕ АЛГОРИТМА ЦИКЛИЧЕСКОЙ СТРУКТУРЫ**

Цель работы: овладение практическими навыками разработки, программирования вычислительного процесса циклической структуры, получение дальнейших навыков по отладке и тестированию программы.

1 Теоретическое введение

1.1 Понятие циклического алгоритма

Циклический алгоритм - это такой алгоритм, в котором некоторая группа действий выполняется некоторое число раз.

Циклом в программирование называют повторение одних и тех же действий (шагов).

Переменная, изменяющаяся с некоторым шагом, называется *переменной цикла*. Переменная цикла определяется своим *начальным значением*, *конечным значением* и *шагом изменения*.

Например, если переменная цикла изменяется следующим образом:

i = 10, 20, 30, . . , 90,

тогда она однозначно определяется своим начальным значением *in* = 10, конечным значением *ik* = 90 и шагом *di* = 10.

Последовательность действий, которые повторяются в цикле, называют *телом цикла*. Количество повторений цикла определяется условием окончания цикла.

Если условие окончания цикла расположено перед телом цикла, то такой цикл называется *циклом с предусловием*.

Если условие окончания цикла расположено после тела цикла, то такой цикл называется *циклом с постусловием*.

При написании условных циклических алгоритмов следует помнить следующее. Во-первых, чтобы цикл имел шанс когда-нибудь закончиться, содержимое его тела должно обязательно влиять на условие цикла. Во-вторых, условие должно состоять из корректных выражений и значений, определенных еще до первого выполнения тела цикла.

Существует 3 типа алгоритмов *циклической структуры*:

1. WHILE - цикл с предусловием ;
2. REPEAT - цикл с постусловием;
3. FOR - цикл со счетчиком (с параметром или с заданным числом повторений).

Если число повторений оператора (составного оператора) заранее неизвестно, а задано лишь условие его повторения (или окончания), используются операторы *while*, *repeat*. Оператор *for* используется, если число повторений заранее известно.

Для всех операторов цикла характерны следующие особенности:

- повторяющиеся вычисления записываются только один раз;
- вход в цикл возможен только через его начало;
- возможен досрочный выход из цикла.

1.2 Оператор цикла с предусловием

Синтаксис (формат) оператора:

WHILE *<условие>* ***DO*** *<тело цикла>;*

где:

while, do – ключевые слова (перев. с англ. *пока* и *делать*);

<условие> – логическое выражение типа сравнения, которое может принимать значение *True* или *False*, используемое для выхода из цикла.

<тело цикла> – простой или составной оператор.

Блок-схема алгоритма циклической структуры с предусловием WHILE . .. DO представлена на рисунке 9.

Выполнение оператора цикла с предусловием

1. Перед каждым выполнением тела цикла вычисляется значение логического выражения.

2. Если выражение принимает значение *True*, то выполняется тело цикла, затем происходит переход на начало цикла.

3. Если значение выражения равно *False*, происходит выход из цикла и переход к оператору программы, следующему за оператором цикла.

4. Если перед первым выполнением цикла значение выражения равно *False*, тело цикла не выполняется ни разу и происходит переход к оператору, следующему за оператором цикла.

В теле цикла обязательно должен быть оператор, изменяющий значение переменной цикла, которая определяет условие выхода из цикла, иначе цикл получится бесконечным, например:

While n = 1 Do Write (' Бесконечный цикл ');

Очевидно, что результат выражения n=1 всегда равен *True*, так как значение переменной цикла n не изменяется в теле цикла, состоящем из оператора *Write*, поэтому цикл будет выполняться бесконечно.

Пример 1. Программа вычисления суммы нечетных чисел в интервале от 0 до 100, использующая оператор цикла с предусловием.

```
Program While-Demo;
Var i, in, ik, h, Sum: Integer;
Begin
Sum : = 0;
{Определение параметров переменной цикла i}
 in: = 1; it =100; h:=2; i:=in;
{Цикл с предусловием}
While i<=ik Do {Начало тела цикла}
Begin
Sum : = Sum + i;
  i:=i+h; {Изменение переменной цикла i на величину шага}
End; {Конец тела цикла}
```

Writeln (' Сумма нечетных чисел равна', Sum:4);
End.
Результат выполнения программы
Сумма нечетных чисел равна 2500

1.3 Оператор цикла с постусловием

Синтаксис (формат) оператора:

REPEAT *<тело цикла>* ***UNTIL*** *<условие>;*

где:

repeat, until – ключевые слова (перев. с англ. *повторять, до тех пор пока*);

<тело цикла> – любые операторы;

<условие> – логическое выражение типа сравнения, которое может принимать значение *True* или *False*, используемое для выхода из цикла.

Блок-схема алгоритма циклической структуры с постусловием REPEAT… UNTIL представлена на рисунке 10.

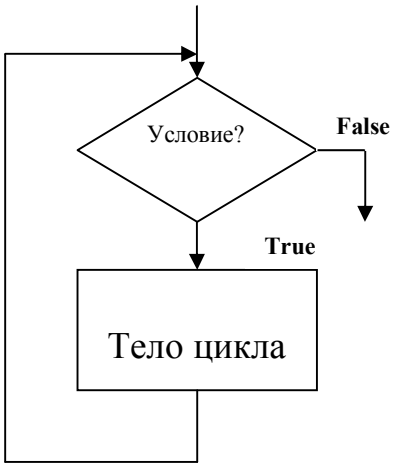


Рисунок 9 - Блок-схема цикла WHILE . .. DO

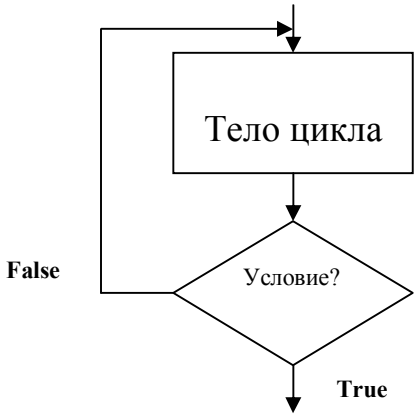


Рисунок 10 - Блок-схема цикла REPEAT …UNTIL

Выполнение оператора цикла с постусловием

1. Выполняется тело цикла.
2. Вычисляется значение логического выражения.
3. Если значение выражения равно *False*, то тело цикла выполняется еще раз.
4. Если значение выражения равно *True* , то происходит выход из цикла и переход к оператору, следующему за оператором цикла.

Отличительные особенности оператора цикла *Repeat*:

- оператор цикла с постусловием выполняется, по крайней мере, один раз, независимо от значения выражения;
- тело цикла выполняется, пока значение выражения равно *False;*
- в теле может находиться произвольное количество операторов без

операторных скобок *Begin ... End*.

По крайней мере, один из операторов тела цикла должен влиять на значение выражения в условии окончания цикла, иначе цикл будет выполняться бесконечно (рисунок 11).

{Цикл имеет завершение}	{Бесконечный цикл}
D:=1;S:=0; *Repeat* *S:=S + D;* *D:=D+1;* *Until D> 100;*	*D:=1;S:=0;* *Repeat* *S:=S+D;* *Until D> 100;*

Рисунок 11 - Конечный и бесконечный циклы

Так как условие окончания цикла проверяется после выполнения тела цикла, то тело цикла выполняется, по крайней мере, один раз. Поэтому важно четко проследить правильность вхождения в цикл и выхода из него. При неправильной организации выхода может возникнуть ошибка переполнения, так как наращивание какой-либо величины в теле цикла может продолжаться до бесконечности.

Пример 2. Программа вычисления суммы четных чисел в интервале 0 до 100, использующая оператор цикла с постусловием.

```
Program Repeat-Demo;
Var i, in, ik, h, Sum: Integer;
Begin
 Sum : =0;
{Определение параметров переменной цикла i}
in: = 0; ik:= 100; h:=2; i: =in;
{Цикл с постусловием}
Repeat {Начало тела цикла}
Sum : = Sum +i;
{Изменение переменной цикла i на величину шага}
i: =i+h;
Until i > ik; {Конец тела цикла}
Writeln (' Сумма четных чисел равна', Sum:4);
End.
```

Результат выполнения программы:

Сумма четных чисел равна 2550

Нетрудно догадаться, что описанные циклы взаимозаменяемы и обладают некоторыми отличиями:

- в цикле с предусловием условие проверяется до тела цикла, в цикле с постусловием – после тела цикла;
- в цикле с постусловием тело цикла выполняется хотя бы один раз, в цикле с

предусловием тело цикла может не выполняться ни разу;

- в цикле с предусловием проверяется условие продолжения цикла, в цикле с постусловием – условие выхода из цикла.

1.4 Оператор цикла со счетчиком

В случаях, когда число повторений может быть заранее известно, для организации циклической обработки информации применяется оператор цикла со счетчиком FOR. Часто этот оператор называют оператором цикла с параметром, так как число повторений задается переменной, называемой параметром цикла, или управляющей переменной.

Синтаксис оператора цикла со счетчиком следующий:

1. ***FOR*** *<i - параметр цикла>:= n1* ***TO*** *n2* ***DO*** *<оператор>;*

2. ***FOR*** *<i - параметр цикла>:= n1* ***DOWNTO*** *n2* ***DO*** *<оператор>;*

где:

for, to (downto), do – ключевые слова (перев. с англ. *для, к, выполнить* соответственно);

<i - параметр цикла> - это переменная, которая изменяется внутри цикла по определенному закону и влияет на его окончание;

n1 и *n2* - начальное и конечное значения параметра цикла. В первом случае *n1<n2*, шаг изменения параметра цикла равен 1. Во втором случае *n1> n2*, шаг изменения параметра цикла равен (-1);

FOR.. .DO - заголовок цикла;

<оператор> - тело цикла.

Тело цикла может быть простым или составным оператором. Оператор *FOR* обеспечивает выполнение тела цикла до тех пор, пока не будут перебраны все значения параметра цикла от начального до конечного.

Заголовок оператора повтора *FOR* определяет:

- диапазон изменения значений управляющей переменной (параметра цикла) и одновременно число повторений оператора, содержащегося в теле цикла;
- направление изменения значения параметра цикла (возрастание на 1 - *TO* или убывание на (-1) - *DOWNTO*).

Выполнение оператора цикла с параметром

1. Перед первым выполнением тела цикла вычисляется значение параметров цикла *n1* и *n2* (в общем случае они могут быть заданы арифметическими выражениями).
2. Параметру цикла присваивается значение выражения *n1*.
3. Если полученное значение параметра цикла не превышает конечного значения *n2,* то выполняется тело цикла.
4. Выполняется возврат на начало оператора цикла.
5. Автоматически значение параметра цикла увеличивается на величину шага.
6. Если полученное значение параметра цикла не превышает конечного значения *n2,* то еще раз выполняется тело цикла.
7. Если полученное значение параметра цикла превысило конечное

значение *n2*, то выполняется переход к оператору, следующему за оператором цикла.

8. Если перед первым выполнением оператора, параметр цикла превышает конечное значение *n2,* то тело цикла не выполняется ни разу и происходит переход к следующему оператору программы.

На использование параметра цикла *FOR* налагаются следующие ограничения:

- в качестве параметра должна использоваться простая переменная, описанная в текущем блоке (локальная);
- управляющая переменная должна иметь дискретный тип (порядковый);
- начальные и конечные значения диапазона должны иметь тип, совместимый с типом управляющей переменной. При этом допустим любой простой тип, кроме вещественного.
- в теле цикла запрещается явное изменение значения управляющей переменной (например, оператором присваивания).

Примеры выполнения оператора цикла с параметром содержатся в таблице 5.

Таблица 5 - Описание и результаты выполнения оператора цикла FOR

Оператор	Результат
For n: = 10 To 15 Do Write (n:3);	*10 11 12 13 14 15*
For n:= 15 DownTo 10 Do Write (n:3)	*15 14 13 12 11 10*

После нормального завершения оператора цикла значение параметра цикла равно конечному значению. Если оператор цикла не выполнялся ни разу, значение параметра цикла не определено.

Не допускается изменение параметра цикла в теле цикла, так как он изменяется автоматически только на величину *±1*. Однако это не является большим недостатком, так как произвольный шаг изменения переменной цикла можно задать при использовании операторов цикла с предусловием или с постусловием.

Пример 3. Программа вычисления суммы чисел в интервале 0 до 100, использующая оператор цикла с параметром.

```
Program FOR-Demo;
Var i, n, k, Sum: Integer;
Begin
Sum:= 0;
{Определение параметров переменной цикла i}
 n:= 0; k:= 100;
   {Цикл с параметром}
 For i:=n To k Do {Начало тела цикла}
 Sum:= Sum + i; {Конец тела цикла}
   Writeln (' Summa chisel ravna ', Sum:4);
End.
```

Результат выполнения программы

Сумма чисел равна 5050

2 Учебный пример

Протабулировать функцию *S(x)* на отрезке [0,1; 2,2] с шагом 0,3. Константа *a*=1,3. Вывод значений *x* и *S* выполнить в виде таблицы (см.Тест на рисунке 13)

$$S=\begin{cases} cos\ ax, если\ ax <= 1 \\ sin\ ax,\ если\ ax > 1 \end{cases}$$

Исходный текст программы вычисления функции *S* (Program lab_3) и блок-схема алгоритма представлены на рисунке 13.

```
Program lab_3;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var s, x, x0, xk, dx:real;
const a=1.3;
begin                                {1}
writeln ('vvod x0, xk, dx');         {2}
read (x0, xk, dx);                   {3}
writeln ('x0 =', x0:6:3);            {4}
writeln ('xk =', xk:6:3);
writeln ('dx =', dx:6:3);
 writeln ('tablica funkcii');        {5}
 writeln ('    x          s(x)');
 x:=x0;                              {6}
 repeat
   if a*x<=1 then                    {7}
   s:=cos (a*x)                      {8}
     else s:=sin (a*x);              {9}
     writeln (x:6:3,' ', s:6:3);     {10}
     x:=x+dx;                        {11}
 until x>xk;                         {12}
 read(x);                            {13}
end.                                 {14}
```

ТЕСТ:

```
x0 = 0.1
xk = 2.2
dx = 0.3
tablica funkcii
  x       S(x)
0.1      0.992
0.4      0.868
. . .    . . .
2.2      0.278
```

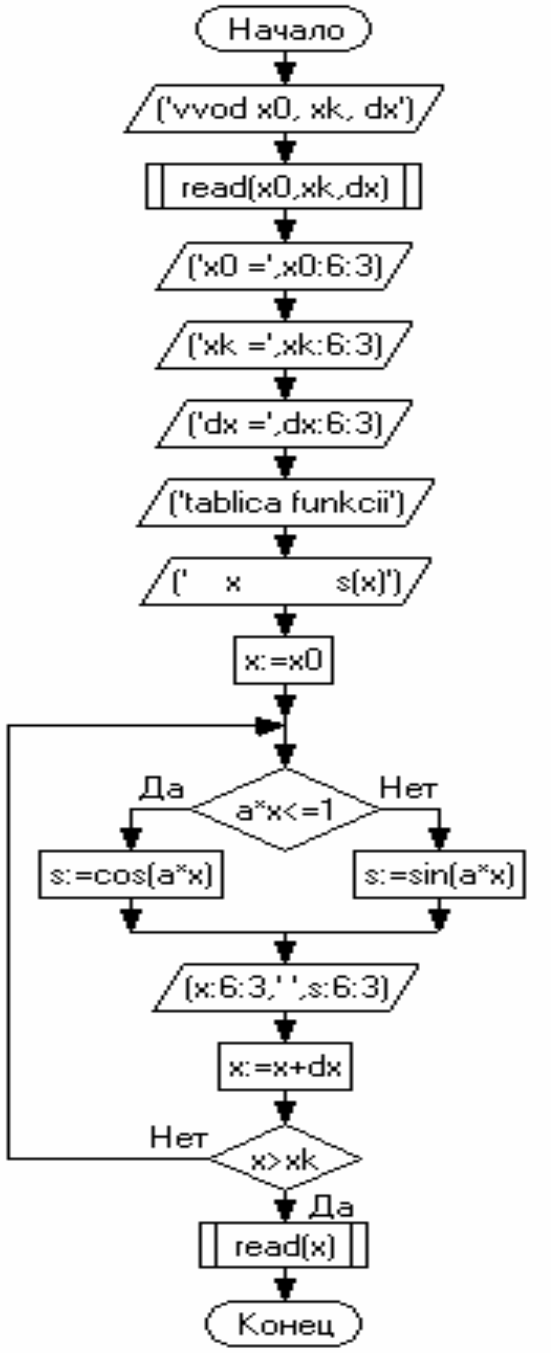


Рисунок 13 - Исходный текст программы вычисления функции *S* и блок-схема алгоритма

В программе объявлены переменные:
s - вычисляемая функция;
x - аргумент функции;
x0 - начало отрезка;
xk - конец отрезка;
dx - шаг табуляции.

Блоки 2 и 3 выполняют ввод переменных *x0, xk, dx*; блоки 4 и 5 - вывод введенных переменных и заголовка результата вычислений. Блок 6 задает начальное значение параметра цикла *x*, т.е. осуществляет подготовку цикла. Собственно вычисление значений функции в цикле выполняется в блоках 7, 8 и 9: в блоке 7 происходит проверка условия - если оно истинно (*Да*), то выполняется блок 8, если ложно (*Нет*) - блок 9. Далее блок 10 выводит текущее значение аргумента *x* и вычисленное значение функции *s* на печать. В блоке 11 выполняется приращение аргумента на величину шага *dx*. Блок 12 выполняет проверку условия окончания цикла: если текущее значение аргумента *x* не достигает конца отрезка (ветвь *Нет*), то цикл повторяется в очередной раз; в противном случае происходит выход из цикла и окончание программы. (Блок 13 выполняет вспомогательную функцию - задержка окна выполнения программы.)

3 Задания для самостоятельной работы

Протабулировать функцию согласно условиям, заданным в таблице 6.

Таблица 6 - Задания к лабораторной работе №3

Вари-ант	Функция	Условие	Диапазон изменения функции [*x0*; *xk*]	Шаг изменения функции (*dx*)	Констан-ты
1	$Y = \begin{cases} ax^2 \ln x \\ x \cos bx \end{cases}$	x<2 x>=2	[0; 2]	0,2	a=- 0,2 b=2
2	$Q = \begin{cases} \pi x^2 - 7/x^2 \\ \lg(x) + 7\sqrt{x}\ ) \end{cases}$	x<1,3 x>=1,3	[0,8; 2]	0,1	
3	$W = \begin{cases} ax^2 + bx + c \\ (a + bx)/\sqrt{x^2+1} \end{cases}$	x<1,2 x>=1,2	[1; 2]	0,05	a=2,8 b=-0,3 c=4
4	$Q = \begin{cases} \pi x^2 - 7/x^2 \\ \lg(x) + 7\sqrt{\lvert x + a \rvert}\ ) \end{cases}$	x<1,4 x>=1,4	[0,7; 2]	0,1	a=1,65
5	$Y = \begin{cases} (x - 2)^2 + 6 \\ 3tgx \end{cases}$	x<2 x>=2	[0,2; 2,8]	0,2	a=2,3
6	$W = \begin{cases} x\sqrt{a - x} \\ (x-a)\cos ax \end{cases}$	x<a x>=a	[1; 5]	0,5	a=2,5

Продолжение таблицы 6

7	$Q = \begin{cases} bx - \lg bx \\ bx + \lg bx \end{cases}$	bx<1 bx>=1	[0,1; 1]	0,1	b=1,5
8	$F = \begin{cases} \lg(x+1) \\ \sin^2 \sqrt{\lvert ax \rvert} \end{cases}$	x>1 x<=1	[2; 5]	0,25	a=20,3
9	$Y = \begin{cases} \sin x \lg x \\ \cos^2 x \end{cases}$	x>3,5 x<=3,5	[0; 7]	0,5	-
10	$S = \begin{cases} (a+b)^2/\cos x \\ (a+b)/(x+1) \end{cases}$	x<2,8 x>=2,8	[0,8; 2]	0,1	a=2,6 b=-0,39
11	$Y = \begin{cases} a \lg x + \sqrt{\lvert x \rvert} \\ 2a \cos x + 3x^2 \end{cases}$	x>1 x<=1	[0,5; 2]	0,1	a=0,9
12	$W = \begin{cases} a/i + bi^2 \\ ai + bi^3 \end{cases}$	i<=6 i>6	[1; 10]	1	a=2,1 b=1,8

4 Контрольные вопросы

1. Какой алгоритм называется циклическим?
2. Что такое переменная цикла?
3. Какими параметрами характеризуется переменная цикла?
4. Что такое тело цикла?
5. Где может располагаться условие окончания цикла?
6. Какие операторы используются для организации цикла?
7. Формат и блок-схема оператора WHILE, как он работает?
8. Формат и блок-схема оператора REPEAT, как он работает?
9. Каковы отличительные особенности оператора цикла с постусловием?
10. Различие между операторами REPEAT и WHILE.
11. В каких случаях получается бесконечный цикл?
12. В каком случае используется оператор FOR?
13. Формат и блок-схема оператора FOR, как он работает?
14. Какие ограничения накладываются на оператор FOR?
15. Выражения какого типа используются в операторах цикла?

ЛАБОРАТОРНАЯ РАБОТА № 4

ТЕМА: **ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ. ОБРАБОТКА ОДНОМЕРНЫХ МАССИВОВ**

Цель работы: овладение практическими навыками объявления, ввода и вывода одномерных массивов, совершенствование приемов программирования циклических структур, получение дальнейших навыков по отладке и тестированию программы.

1 Теоретическое введение

1.1 Понятие массива

Наряду с простыми типами (*byte, integer, real*) в языке Pascal используются также структурированные данные, самыми простыми из которых является массивы. Массивы удобно использовать для работы с множеством однотипных данных (целочисленными значениями, строками и датами). Например, можно создать массив для хранения списка студентов, обучающихся в одной группе. Вместо того, чтобы задавать переменные для каждого студента, например Студент1, Студент2 и т.д., достаточно создать один массив, где каждой фамилии из списка будет присвоен порядковый номер.

Массив – структурированный тип данных, состоящий из фиксированного числа элементов одного типа, упорядоченных по номерам (рисунок 14). Массив на рисунке состоит из 6-ти элементов, каждый из которых сохраняет число вещественного типа. Элементы в массиве пронумерованы от 1 до 6. Такого рода массив, представляющий собой список данных одного и того же типа, называют простым или одномерным.

№ элемента массива (индекс)	1	2	3	4	5	6
Элемент массива	12,1	0,13	-1,5	0	21,9	-3,7

Рисунок 14 - Одномерный массив вещественных чисел

Именно так, как показано на рисунке, и располагаются элементы массива в памяти ЭВМ, т.е. они упорядочены по возрастанию порядковых номеров (индексов). Для доступа к данным, хранящимся в определенном массиве, необходимо указать имя массива (идентификатор) и порядковый номер этого элемента.

1.2 Описание (объявление) одномерного массива и обращение к элементам массива

Одномерные массивы имеют аналогию с таким понятием в математике, как вектор. Описание переменной типа массив задается следующим образом:

имя_массива: ***ARRAY*** *[тип индекса]* ***OF*** *тип_элемента_массива;*

где:

array и *of* – ключевые слова (пер. с англ. *массив*, *из)*.

Для типа индексов обычно используют тип-диапазон, который определяет границы изменения индексов. Так как память выделяется под массив во время компиляции программы, то границы изменения индексов должны быть константами, или константными выражениями, т.е. они должны быть явно определены в программе. Чаще всего в реальных задачах индексы изменяются от 1, и в этом случае индекс элемента совпадает с его порядковым номером.

Элементами массива могут быть как простые переменные любых типов, так и переменные составных типов (массивов, строк, записей).

Ниже приведены примеры описания одномерных массивов:

```
var
f :array [1..10] of integer;        массив f из 10 целых чисел
mas: array[0..100] of char;         массив mas из 101 символов
b: array[-5..5] of real;            массив b из 11 вещественных чисел
```

Размерность массива – величина произвольная, однако суммарная длина внутреннего представления любого массива не может быть больше 64 Кбайт. Следует учесть, что резервирование объема памяти для массива выполняется при компиляции программы. Поэтому, если количество элементов массива заранее неизвестно или оно может изменяться, то необходимо объявлять массив максимально необходимой размерности.

Описать массив можно и в разделе *type*. Процедура объявления состоит в следующем: ввести новый тип данных, а потом описать переменные нового типа:

```
type
    имя_типа = array[тип_индекса] of тип_элемента_массива;
```

Тип_элемента_массива – это любой ранее определенный тип данных, например:

```
type massiv = array[0..12] of real;
     abc = array[-3..6] of integer;
var  x, y: massiv;   z: abc;
```

Для описания массива можно использовать предварительно определенные константы:

```
const   n=10; m=12;
var    a: array[1..n] of real;   b: array[0..m] of byte;
```

Константы должны быть определены до использования, так как массив не может быть переменной длины.

Массив можно объявить и в разделе констант, явно перечислив его элементы:

```
Const
mas3: array [1. . 3] of integer = (2, 4, 8)
```

Обращение к массиву в целом осуществляется по его имени. Для обращения к отдельному элементу массива указывается имя массива и индекс - целое число,

следующее за именем массива в квадратных скобках:

f[1]:=0	1-й элемент массива *f*
mas[100]:='a'	100-й элемент массива *mas*
b[3]:=1.13	3-й элемент массива *b*

При обработке массивов возникают такие задачи, как ввод элементов массива, нахождение суммы, произведения, среднего и т.д., поиск некоторого элемента в массиве, сортировка элементов массива, вывод элементов массива.

1.3 Ввод-вывод элементов массива

Паскаль не имеет специальных средств ввода-вывода всего массива, поэтому необходимо последовательно вводить 1-й, 2-й, 3-й и т.д. его элементы и аналогичным образом поступить при выводе. Следовательно, для ввода-вывода необходимо организовать цикл, в котором практически все операции с массивами необходимо проводить поэлементно. Для формирования и обработки элементов массива удобно использовать циклы *for* и *while*. Кроме этого, массивы можно формировать с помощью датчика случайных чисел, используя процедуру *random*.

1. Ввод элементов массива с использованием цикла с предусловием (*while...do*). Фрагмент программы и блок-схема алгоритма данного фрагмента приведены на рисунке 15.

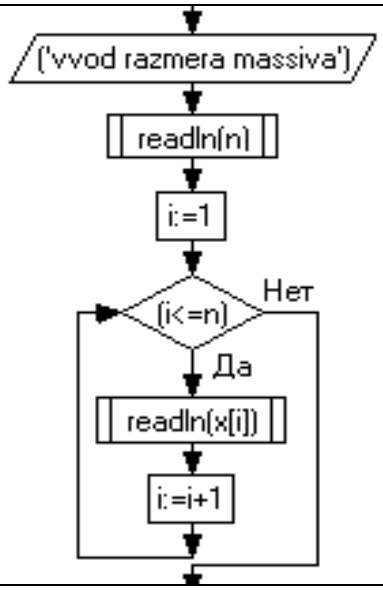
Фрагмент программы	Блок-схема
var x: array [1..100] of real; *i, n: integer;* *begin* *writeln ('vvod razmera massiva');* *readln(n);* *i:=1;* *while (i<=n) do* *begin* *readln(x[i]);* *i:=i+1;* *end;*	

Рисунок 15 - Ввод элементов массива с использованием цикла *while...do*.

2. Ввод элементов массива с использованием блока модификации (с помощью цикла *for*). Фрагмент программы и блок-схема алгоритма данного фрагмента приведены на рисунке 16.

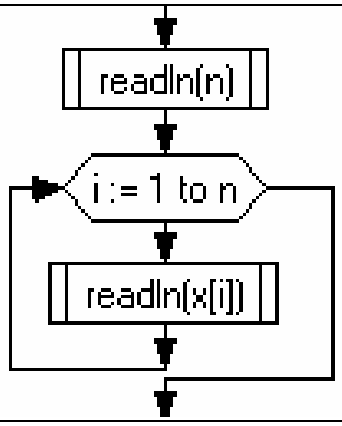
Фрагмент программы	Блок-схема
var *x: array [1..100] of real;* *i,n: integer;* *begin* *readln(n);* *for i:=1 to n do* *readln(x[i]);*	

Рисунок 16 - Ввод элементов массива с использованием цикла *for*.

3. Ввод элементов массива с использованием процедуры *random*. Фрагмент программы и блок-схема алгоритма данного фрагмента приведены на рисунке 17.

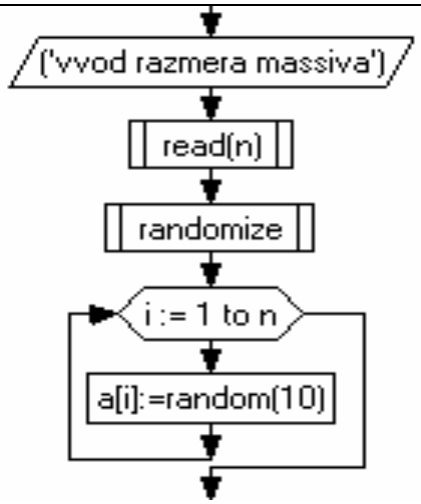
Фрагмент программы	Блок-схема
type mas = array[1..n1] of integer; *var a:mas;* *i, n:integer;* *begin* *writeln('vvod razmera massiva');* *read(n);* *randomize;* *for i:=1 to n do* *a[i]:=random(10);*	

Рисунок 17 - Ввод элементов массива с использованием процедуры *random*.

В данном примере процедура *random* формирует массив *mas* случайных целых чисел из диапазона 1..10. Процедура *randomize* предназначена для инициализации процедуры *random*.

2 Учебный пример

Поиск максимального элемента и его номера в массиве.

Дан массив *X* целых чисел, состоящий из *n* (n<20) элементов. Найти максимальный элемент и номер, под которым он хранится в массиве. Проверить правильность работы программы при n = 3 и следующих элементах массива: (10, 5, 2).

Исходный текст программы (Program lab_4) и блок-схема алгоритма представлены на рисунке 18.

```
 Program lab_4;
{$APPTYPE CONSOLE}
 uses
 SysUtils;
 Var n, i, nmax, xmax:integer;
       x: array[1..15] of integer;
 Begin                                    {1}
 Writeln('vvod razmera massiva n');       {2}
 Readln(n) ;
 Writeln('vvod elementov massiva');
 For i:=1 to n do                         {3}
   read(x[i]);                            {4}
   xmax:=x[1] ;                           {5}
   nmax:=1;                               {5}
 For i:=2 to n do                         {6}
  If x[i]>xmax then                       {7}
   Begin
            xmax:=x[i];                   {8}
            nmax:=i;                      {8}
   End;
 Writeln ('xmax=', xmax:8, '    ');       {9}
 Writeln ('nmax=', nmax);                 {10}
 read (i);
 End.                                     {11}
```

ТЕСТ:

n = 3
Массив: (10, 5, 2).
Результат:
xmax=10
nmax=1

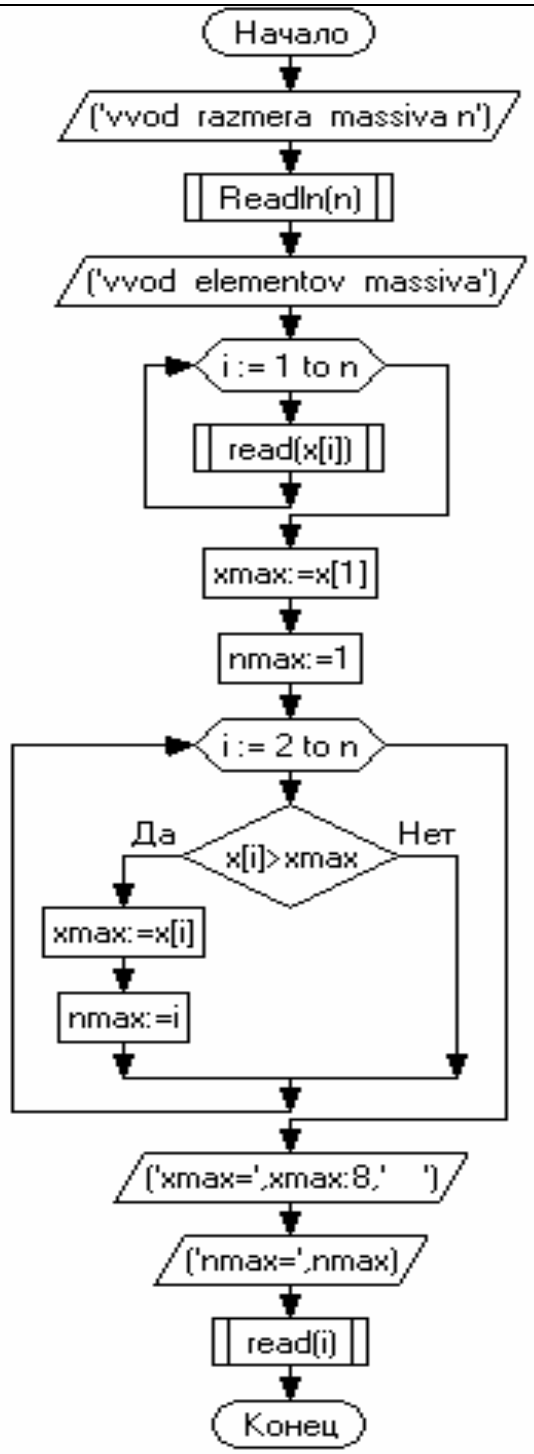


Рисунок 18 - Исходный текст программы и блок-схема алгоритма обработки одномерного массива

Алгоритм решения задачи следующий. Пусть в переменной с именем *XMax* хранится значение максимального элемента массива, а в переменной с именем *Nmax* - его номер.

Блок 2 осуществляет ввод значения *n* - количество элементов в массиве. Блоки 3,4 осуществляют ввод массива X, состоящего из *n* элементов. Предположим, что первый элемент массива является максимальным, и запишем его в переменную

Xmax, а в *Nmax* - его номер (то есть 1). Блок 5 задает начальное значения *Xmax* и *Nmax*. Затем все элементы, начиная со второго, сравниваем в цикле с максимальным. Блок 6 организует цикл, перебирающий элементы массива, начиная со второго и кончая *n*-м. Блок 7 сравнивает текущий элемент массива с максимальным и, если текущий элемент массива оказывается больше максимального, то блок 8 записывают его в переменную *XMax*, а текущее значение индекса *i* - в переменную *Nmax*. Блок 9 выводит на экран значение максимального элемента, блок 10 - его номер в массиве.

3 Задания для самостоятельной работы

Выполнить на ЭВМ программу обработки одномерного массива в соответствии с индивидуальным заданием (таблица 7).

Проверить правильность выполнения программы на тестовых данных.

Таблица 7 - Задания к лабораторной работе № 4

Вариант	Массив	Действия	Условия и ограничения
1	X(100)	Вычислить сумму и количество элементов массива X	$0 \le X \le 1$
2	A(80)	Вычислить среднее арифметическое значение элемента массива A	$A_i<0$
3	X(70)	Подсчитать количество элементов массива X, удовлетворяющих условию	$-1 \le X_i \le 1$
4	B(50)	Определить максимальный элемент массива B и его порядковый номер	$X_i >0$
5	C(40)	Вычислить минимальный элемент массива C и его порядковый номер	$X_i < 0$
6	D(80)	Вычислить сумму положительных элементов массива X	
7	Y(20)	Вычислить среднее геометрическое элементов массива Y	$Y_i > 0$
8	Z(30)	Подсчитать количество положительных элементов массива Z	
9	N(50)	Определить сумму элементов массива N, кратных трем	
10	X(N)	Вычислить сумму элементов массива X, удовлетворяющих условию	$X_i >0, N \le 30$
11	X(N)	Переписать в массив Y положительные элементы массива X	$X_i >0, N \le 30$
12	X(N)	Переписать в массив Y отрицательные элементы массива X	$N \le 40$

4 Контрольные вопросы

1. Что представляет собой массив как структура данных?
2. Что такое индекс?
3. Как определяется положение элемента в одномерном массиве?
4. Формат объявления массива.
5. Формат обращения к массиву и элементам массива.
6. Виды циклических алгоритмов ввода-вывода массива
7. Блок-схемы ввода-вывода одномерного массива
8. Какие типы данных используются для индекса массива?
9. Какие типы данных могут быть элементами массива?
10. Как элементы массива располагаются в памяти ЭВМ?

ЛАБОРАТОРНАЯ РАБОТА № 5

ТЕМА: **ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ ОБРАБОТКИ МАТРИЦ**

Цель работы: овладение практическими навыками объявления, ввода и вывода двумерных массивов, овладение навыками алгоритмизации и программирования вложенных циклов при решении типичных задач обработки матриц, совершенствование приемов программирования циклических структур и навыков по отладке и тестированию программы.

1 Теоретическое введение

1.1 Понятие матрицы

Если возникает необходимость хранения данных в виде таблиц, в формате строк и столбцов, то необходимо использовать многомерные массивы. На рисунке 19 приведен пример массива, состоящего из четырех строк и четырех столбцов. Это двумерный массив. Строки в нем можно считать первым измерением, а столбцы – вторым. Двумерные массивы имеют аналогию с таким понятием в математике, как матрица. Для доступа к данным, хранящимся в этом массиве, необходимо указать его имя и два *индекса*, первый из которых должен соответствовать *номеру строки*, а второй – *номеру столбца*, в котором хранится необходимый элемент.

		Номера столбцов			
		1	2	3	4
Номера строк	1	3,5	7,8	1,3	0,6
	2	-1,4	0,3	0	12,1
	3	-5,7	-0,78	5,0	6,9
	4	45,1	124,0	-24,7	0,96

Рисунок 19 - Двумерный массив вещественных чисел

Двумерный массив располагается в непрерывной области памяти построчно, т.е. сначала размещаются все элементы 1-й строки, затем - 2-й и т.д.

1.2 Описание (объявление) двумерного массива и обращение к элементам массива

В языке Pascal двумерный массив - это массив, элементами которого являются одномерные массивы. Формат объявления двумерного массива:

*b: **ARRAY**[1. .n] **OF ARRAY**[1. .m] **OF** integer;*

С другой стороны двумерный массив можно описать и так:

*b: **ARRAY**[1. .n, 1. .m] **OF** integer;*

Чаще пользуются вторым вариантом описанием, оно является более кратким, но менее наглядным. В описании n – количество строк, m – количество столбцов матрицы. При изменении второго индекса на единицу мы передвигаемся вдоль строки, а при изменении первого индекса на единицу передвигаемся вертикально

вдоль столбца. Обычно в качестве идентификатора номера строки используют символ *i*, а столбца – *j*. Тогда к элементу массива, описанному выше, можно обратиться по имени *b[i,j]*.

Описать двумерный массив, также как и одномерный, можно и в разделе TYPE:

type x = array[0. . 5, 1 . .10] of real;
y = array[1. .6, 1. .5] of integer;
var a,b: x; z: y;

Для описания массива можно использовать предварительно определенные константы:

const n=10; m=12;
var a: array[1. .n, 1. . m] of real; b: array[0..m] of byte;

Массив можно объявить и в разделе констант, явно перечислив его элементы:

Const
mas3: array [1. . 3, 1. . 2] of integer =((2,4,8), (1,3,7))

В таблице 8 приведены примеры описания двумерных массивов и обращения к их элементам.

Таблица 8 - Описание двумерных массивов и обращение к их элементам

Описание массива	Обращение к элементу массива
mas:array[0..4,1..5] of char - матрица из 4 строк, 5 столбцов символов	*mas[1,1]:='a'* - элемент, находящийся в 1-й строке, в 1-м столбце
b:array[1..5,1..6] of integer - матрица из 5 строк, 6 столбцов целых чисел	*b[5,6]:=7* - элемент, находящийся в 5-й строке, в 6-м столбце
asd:array[1..3, 1..10] of real - матрица из 3 строк, 10 столбцов вещественных чисел	*asd[i,8]:=1,2* - элемент, находящийся в i-й строке, в 8-м столбце

При обработке двумерных массивов возникают такие же задачи, как и при обработке одномерных массивов: ввод элементов массива, нахождение суммы, произведения, среднего и т.д., поиск некоторого элемента в массиве, сортировка элементов массива, вывод элементов массива. Для реализации указанных алгоритмов также используются циклические алгоритмы, но, поскольку число измерений массива больше 1, то рационально использовать вложенные циклы.

1.3 Вложенные циклы

В теле оператора цикла могут находиться другие операторы цикла. Такая структура циклов называется вложенными циклами. Цикл, который не входит ни в какой другой цикл, называется *внешним*. Если во внешнем цикле находятся другие циклы, то они называются *внутренними* циклами.

При использовании вложенных циклов необходимо составлять программу таким образом, чтобы внутренний цикл полностью укладывался в циклическую часть внешнего цикла.

Пример вложенных циклов:

```
For i: = 1 To m Do
Begin                                   внутренний цикл      внешний цикл
  For j: = 1 To n Do Write(C[i,j]:5);
Writeln;
 End;
```

Следует отметить, что внутренний цикл выполняется полностью для каждого значения переменной внешнего цикла. Например, если *i* - переменная внешнего цикла, которая изменяется от 1 до *m* с шагом 1, а *j* -переменная внутреннего цикла, которая изменяется от 1 до *n* с шагом 1, тогда:

для *i = 1* переменная *j* принимает значения *j = 1, 2, 3, ..., n;*
для *i = 2* переменная *j* принимает значения *j = 1, 2, 3, ..., n;*
для *i = 3* переменная *j* принимает значения *j = 1, 2, 3, ..., n;*
. . .
для *i = m* переменная *j* принимает значения *j = 1, 2, 3, ..., n;*

Пример 1. Программа вычисления количества положительных элементов заданной матрицы С размера *m* x *n.*

```
Program Demo-For;
Var C: array[1..100,1..100] of integer;
    i,j,n,m,k: Integer;
Begin
Readln(m,n); {Ввод матрицы}
{Внешний цикл}
For i: = 1 To m Do
{Внутренний цикл}
  Begin
    For j:= 1 To n Do Read(C[i,j]); {Конец внутреннего цикла}
Readln;
End; {Конец внешнего цикла}
K: = 0;
{Суммирование числа положительных элементов матрицы}
{Внешний цикл}
For i: = 1 To m Do
{Внутренний цикл}
For j := 1 To n Do
If C[i,j] > 0 then K := K + 1;
{Конец внутреннего и внешнего циклов}
Writeln('Kolichestvo',K:4);
End.
```

2 Учебный пример

Задача обработки матрицы, состоящая из трех частей:
1) ввод матрицы с клавиатуры;

2) суммирование элементов матрицы;

3) вывод матрицы на экран.

Исходный текст программы (Program lab_5) и блок-схема алгоритма представлены на рисунке 20.

```
Program lab_5;
{$APPTYPE CONSOLE}
uses
  SysUtils;

Const s=10; st=20;
Var i, j, n,m:integer;
    sum:real;
    a: array[1..s, 1..st]   of  integer;
Begin                                          {1}
Writeln('vvod razmera matrici - n, m');        {2}
Readln(n,m) ;
sum:=0;                                        {3}
Writeln('vvod elementov matrici');
For  i:=1  to n do                             {4}
   for j:=1 to m do                            {5}
     begin
         read (a[i,j]);                        {6}
         sum:=sum+a[i,j];                      {7}
     end;
Writeln;
For  i:=1  to n do                             {8}
  begin
for j:=1 to m do                               {9}
write(a[i,j]:5);                               {10}
  Writeln;                                     {11}
  end;
 Writeln;
 writeln ('sum = ', sum:8:2);                  {12}
read (i);
End.                                           {13}
```

ТЕСТ:

n = 3 m = 3

Матрица:

1 2 3

4 5 6

7 8 9

Результат: sum = 45.00

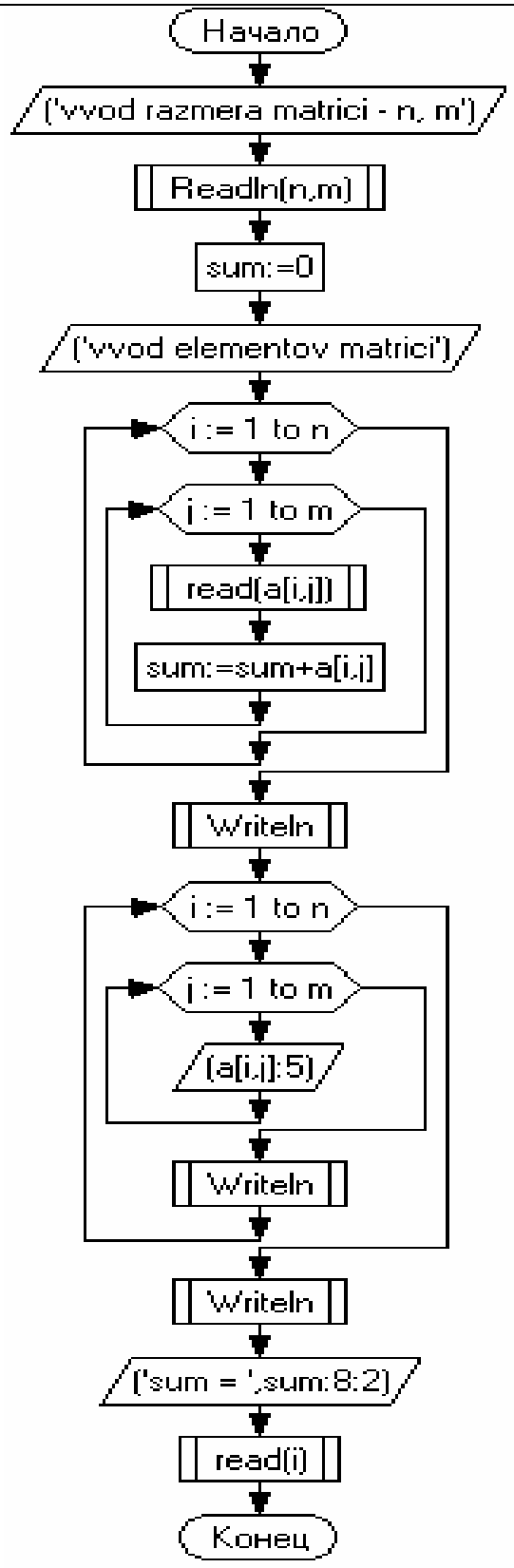


Рисунок 20 - Исходный текст программы и блок-схема алгоритма обработки матрицы

Анализируя предложенную программу, можно заметить, что для решения задачи два раза используются вложенные циклы: 1-й (блоки 4-7) - реализует ввод и одновременно суммирование, 2-й (блоки 8-11) - вывод на экран.

Так как матрица располагается в непрерывной области памяти построчно, более рационально будет и обрабатывать элементы построчно. В программе во вложенных циклах для каждого значения индекса *i* индекс *j* изменяется от *1* до *m*, т.е. индекс *j* изменяется чаще. Таким образом, обрабатываются элементы матрицы построчно. При выводе элементов матрицы на экран для каждого значения *i* в теле цикла выполняются два оператора вывода: первый оператор вывода (10) выводит на экран в строчку элементы одной строки, а второй оператор вывода (11) переводит курсор на новую строку, что как раз и обеспечивает вывод матрицы в виде прямоугольника (в т.н. общепринятом виде).

3 Задания для самостоятельной работы

Составить и выполнить на компьютере программу обработки матрицы в соответствии с индивидуальным заданием (таблица 9). Проверить правильность выполнения программы на матрице из 3-х строк и 3-х столбцов.

Программа должна иметь следующую структуру:

1. Ввод матрицы (размерность 3 строки, 3 столбца);
2. Печать исходной матрицы в виде прямоугольной таблицы;
3. Выполнение операций, реализующих "Действие";
4. печать результата (варианты 1 - 4, 9)
   если матрица подвергалась преобразованию - печать результирующей матрицы в виде прямоугольной таблицы (варианты 5 - 8, 10 - 12).

Таблица 9 - Задания к лабораторной работе № 5

Вариант	Матрицы	Действие	Условия и ограничения
1	2	3	4
1	A(10,15)	Вычислить сумму элементов каждой строки матрицы, удовлетворяющих условию	$a_i > 0$
2	A(N,M)	Вычислить число элементов каждой строки матрицы, удовлетворяющих условию.	$a_i > 0$ $N < 20$ $M < 15$
3	B(N,N)	Вычислить сумму и число элементов, находящихся под главной диагональю и на ней.	$N < 12$
4	C(N,N)	Вычислить сумму и число элементов матрицы, находящихся под главной диагональю.	$N < 12$
5	D(K,K)	Записать на место отрицательных элементов матрицы нули.	$K < 10$

Продолжение таблицы 9

1	2	3	4
6	D(10,10)	Записать на место отрицательных элементов матрицы нули, а на место положительных - единицы.	-
7	F(N,M)	Найти в каждой строке матрицы максимальный элемент и поместить его на место первого элемента строки.	$N \le 20$ $M \le 15$
8	F(10,8)	Найти в каждой строке матрицы минимальный элемент и поместить его на место последнего элемента строки.	
9	N(10,10)	Для целочисленной матрицы найти для каждой строки число элементов, кратных пяти.	
10	N(10,10)	В каждой строке матрицы положительные элементы заменить первым элементом строки.	-
11	P(N,N)	В каждой строке первый элемент поменять местами с элементом главной диагонали.	$N \le 15$
12	R(K,N)	В каждой строке первый элемент заменить элементом с главной диагонали.	$K \le 20$ $N \le 15$

4 Контрольные вопросы

1. Что представляет собой двумерный массив как структура данных?
2. Как определяется положение элемента в двумерном массиве?
3. Формат объявления двумерного массива.
4. Формат обращения к массиву и элементам массива.
5. В чем состоит особенность циклических алгоритмов обработки двумерных массивов?
6. Блок-схемы ввода-вывода двумерного массива
7. Какие типы данных используются для индекса массива?
8. Какие типы данных могут быть элементами массива?
9. Как элементы матрицы располагаются в памяти ЭВМ?
10. Каковы общие и отличительные черты одномерных и двумерных массивов?
11. Что такое вложенные циклы?
12. Что такое внутренний и внешний цикл?
13. Каково расположение внутреннего цикла относительно внешнего?

ЛАБОРАТОРНАЯ РАБОТА № 6

ТЕМА: **ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ПОДПРОГРАММ (ПРОЦЕДУРЫ И ФУНКЦИИ)**

Цель работы: овладение практическими навыками алгоритмизации и программирования задач с использованием подпрограмм и функций пользователя различных видов, приобретение навыков написания подпрограмм и обращения к ним, выбора параметров подпрограмм.

1 Теоретическое введение

1.1 Понятие подпрограммы

Программы с использованием подпрограмм позволяют реализовать один из самых прогрессивных методов программирования - структурное программирование.

Подпрограммы решают следующие важные задачи:

- избавляют от необходимости многократно повторять в тексте программы аналогичные фрагменты;
- улучшают структуру программы, облегчая ее понимание;
- облегчают отладку больших программ;
- позволяют использовать части одной программы в другой;
- позволяют экономить память, т.к. память для хранения переменных, используемых в подпрограммах, выделяется только на время работы подпрограммы
- повышают устойчивость к ошибкам программирования и непредвиденным последствиям при модификациях программы.

Подпрограммы могут быть стандартными, т.е. определенными системой, и собственными, т.е. определенными программистом.

Стандартная подпрограмма (процедура или функция) - подпрограмма, включенная в библиотеку программ, доступ к которой обеспечивается средствами языка программирования. Вызывается подпрограмма по имени с заданием фактических параметров (*cos(15), abs(-5))*.

Из набора стандартных процедур и функций по обработке одного типа информации составляются модули. Каждый модуль имеет своё имя. Доступ к процедурам и функциям модуля осуществляется при подключении этого модуля (предложение *Uses*).

В стандартных модулях содержится большое количество подпрограмм, но невозможно создать модуль, который бы содержал все нужные программисту подпрограммы. Поэтому большую роль играют собственные подпрограммы, которые создает программист для решения конкретной задачи.

Существует два способа объединения программ и подпрограмм пользователя:

1. Текст подпрограммы может быть приведен в разделе описания использующей их программы.
2. Подпрограммы группируются в отдельных файлах, имеющих специальную структуру - модулях. Для того чтобы основная программа могла использовать модуль, он должен быть подключен к основной программе.

Первый способ применяется тогда, когда программа в целом не очень большая, а ее подпрограммы, скорее всего, не будут использоваться в других программах.

Структура текста подпрограммы соответствует структуре текста основной программы за двумя исключениями:

1. Подпрограмма начинается с заголовка, содержащего имя подпрограммы, передаваемые в нее и возвращаемые от нее параметры, т.е. запись заголовка подпрограммы отличается от заголовка программы;

2. Подпрограмма заканчивается не точкой, а точкой с запятой.

Подпрограмма - это особым образом оформленный фрагмент программы, имеющий собственное имя. Упоминание этого имени в тексте программы приводит к активизации подпрограммы и называется ее вызовом. При вызове подпрограммы выполнение основной программы приостанавливается, и управление передается в подпрограмму, где выполняются команды, заданные в ней. Подпрограмма завершается, если выполнены все ее операторы до завершающего слова End или по специальной команде выхода из подпрограммы Exit. По окончании работы подпрограммы управление возвращается основной программе и выполняются операторы, стоящие непосредственно за оператором вызова процедуры (рисунок 21).

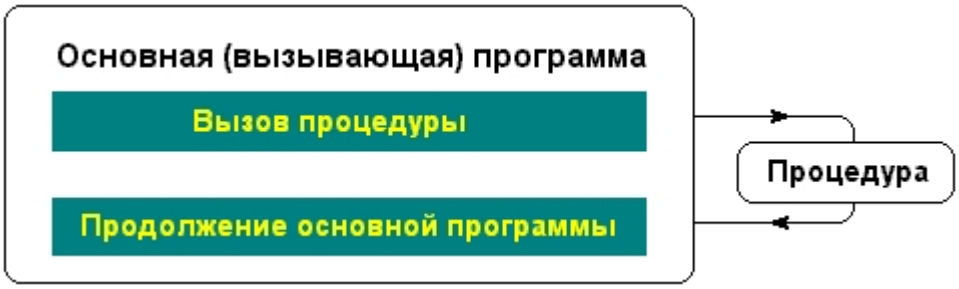


Рисунок 21 - Схема взаимодействия вызывающей программы и процедуры

Для обмена информацией между процедурами, функциями и другими блоками программы существует механизм *входных* и *выходных параметров*.

Входными параметрами называют величины, передающиеся из вызывающего блока в подпрограмму (исходные данные для подпрограммы), а *выходными* - передающиеся из подпрограммы в вызывающий блок (результаты работы подпрограммы).

В языке Pascal определяются два типа подпрограмм - *процедуры* (Procedure) и *функции* (Function).

Если подпрограмма должна возвращать в вызывающую программу несколько значений, либо возвращать результаты структурированного типа, либо вообще не иметь параметров, то такая подпрограмма оформляется в виде *процедуры*.

Если подпрограмма возвращает в вызывающую программу единственный результат скалярного типа, то она оформляется в виде *функции*. При этом в теле функции значение результата должно быть присвоено имени функции.

Основное различие между процедурой и функцией состоит в том, что процедура только выполняет какую-либо законченную последовательность действий, не возвращая результата работы в основную программу, а функция и выполняет действия, и возвращает результат.

Любая программа может содержать несколько процедур и функций. Подпрограмма должна быть описана до того, как будет использована в программе или другой подпрограмме. Процедуры и функции объявляются в разделе описания вслед за разделом переменных.

Общая структура программы, содержащей подпрограммы, выглядит так:

```
Program hh;
    Label; {описание меток}
    Const; {описание констант}
    Type; {описание типов}
    Var; {описание переменных}
    Procedure; {описание процедур}
    Function; {описание функций}
Begin
 . . .
    вызов процедуры
      . . .
    вызов функции
      . . .
    end.
```

1.2 Описание подпрограммы

Структура процедуры или функции очень похожа на структуру главной процедуры, она также содержит раздел описаний и раздел операторов; раздел операторов начинается с *BEGIN* и заканчивается *END*; (но не END. - как у главной процедуры). Все процедуры и функции записываются в разделе описаний какой-либо другой процедуры или функции, в том числе и главной процедуры.

1.2.1 Описание процедуры

Любая процедура оформляется аналогично программе и содержит заголовок, разделы описаний и операторов.

Структура процедуры:

```
Procedure <Name>(<Список формальных параметров>);    {Заголовок}
          раздел описаний
  BEGIN
          раздел операторов
  END;
```

Заголовок:

Procedure - служебное слово;

Name - произвольный идентификатор, определяющий имя процедуры;

Формальные параметры - это переменные, посредством которых передаются

данные из места вызова процедуры в её тело, либо из процедуры в места вызова. Список формальных параметров может отсутствовать, при этом символ ";" ставится сразу за именем процедуры и данные из места вызова процедуры в её тело не передаются. Если список формальных параметров имеется, то он записывается в круглых скобках после имени процедуры и имеет вид:

([VAR] имя, ... имя:тип;)

где:

имя - имена параметров, каждый параметр может использоваться внутри процедуры как обычная переменная соответствующего типа;

тип - имя типа, но не описание пользовательского типа; скажем, описание параметра в виде x:1..5 неверно, но, если выше описан соответствующий тип: *TYPE MyType=1..5,* то параметр можно описать в виде *x:MyType.* Ключевое слово *VAR* перед описанием параметров означает в данном случае, что все параметры до ";" или до ")" - параметры-переменные; если же *VAR* отсутствует, то параметры являются параметрами-значениями. Смысл этих понятий рассмотрим ниже.

Раздел описаний: Процедура может содержать такие же разделы описаний, что и Паскаль-программа, а именно: разделы описания модулей, меток, констант, типов, переменных, процедур и функций.

Раздел операторов процедуры: собственно тело процедуры.

Пример 1. Процедура ввода N целых чисел.
Пусть в основной программе объявлен тип
type mas=array [1..50] of integer;
Процедура может иметь вид:

```
Procedure vvmass(var A:mas; n: integer);
var i:integer;
Begin
WriteLn('vvedite', n, 'chisel');
for i:=1 to n do
read (a[i]);
End;
```

1.2.2 Вызов процедуры

Для вызова процедуры на исполнение к ней необходимо обратится. Обращение к подпрограмме - переход к выполнению подпрограммы с заданием информации, необходимой для ее выполнения и возврата.

Процедуры вызываются в других процедурах и функциях с помощью оператора вызова:

имя (список фактических параметров);

Тогда вызов процедуры, описанной выше, будет выглядеть так:

vvmas(x,k);

Данная запись означает, что вызывается процедура *vvmass* для ввода *k* целых чисел в массив *X*, при этом: *X* - массив типа *mas*, *k* - целого типа (может быть

указано просто целое число). К моменту вызова процедуры *k* должно быть определено в вызывающей программе.

Список фактических параметров задается в том и только в том случае, когда в заголовке процедуры задан список формальных параметров.
Фактические параметры - это переменные (или значения, заданные явно), которые передаются в процедуры на место формальных параметров. Если в вызываемой процедуре отсутствует список формальных параметров, то список фактических параметров тоже отсутствует.

Количество фактических параметров должно соответствовать количеству формальных параметров; соответствующие фактические и формальные параметры должны совпадать по порядку записи и по типу данных.

Параметры в списке разделяются запятыми и могут быть любыми выражениями, если соответствующий формальный параметр есть параметр-значение, или только именами переменных, если соответствующий формальный параметр есть параметр-переменная. При вызове процедуры значение соответствующих фактических параметров передается формальным параметрам, и таким образом процедура получает информацию из вызывающей процедуры или функции.

Выполнение оператора вызова процедуры состоит в том, что все формальные параметры заменяются соответствующими фактическими. После этого создается динамический экземпляр процедуры, который и выполняется. После выполнения процедуры происходит передача управления в основную программу, т.е. начинает выполняться оператор, следующий за оператором вызова процедуры.

1.2.3 Описание функции

Функция описывается аналогично процедуре. Отличительной особенностью функции является то, что она возвращает только один результат выполнения. Этот результат обозначается именем функции и возвращается (передается) в основную программу (место вызова). Функция состоит из заголовка, раздела описаний и раздела операторов.

Структура функции:

Function <Name>(<Список формальных параметров>):<Type>; {Заголовок}

раздел описаний

BEGIN

раздел операторов

Name:=<выражение соответствующего типа>;

…

END;

Заголовок:

Function - служебное слово;

Name - произвольный идентификатор, определяющий имя функции;

Формальные параметры - это переменные, посредством которых передаются данные из места вызова функции в её тело;

Type - тип возвращаемого параметра.

В отличие от процедур в разделе операторов тела функции обязательно должен быть хотя бы один оператор присвоения имени функции выражения или значения соответствующего типа. После работы функции результат присваивается имени функции.

Таким образом, алгоритм можно оформить в виде функции в том случае, если в качестве результата получается одно единственное значение.

Для вызова функции достаточно указать ее имя (с фактическими параметрами) в любом выражении, где тип результата функции будет приемлем. Имя функции можно использовать в арифметических выражениях и других командах.

Пример 2. Функция определяет по двум катетам гипотенузу прямоугольного треугольника.

```
Function Gepoten(a,b:real):real;
Begin
Gepoten:=Sqrt(Sqr(a)+Sqr(b))
End;
```

1.2.4 Вызов функции

Вызов функции из основной программы или из другой подпрограммы может выглядеть следующим образом:

```
z:=Gepoten(x, y); {z присваивается значение гипотенузы}
```

или

```
WriteLn('Значение гипотенузы', Gepoten(x, y));
```

Также как и в случае с вызовом процедуры, к моменту вызова функции параметры *x* и *y* должны быть определены.

1.3 Формальные и фактические параметры

Как упоминалось выше, формальные параметры подпрограммы показывают, с какими параметрами следует обращаться к данной подпрограмме (количество, их последовательность, типы). Они задаются в заголовке подпрограммы в виде списка, разбитого на группы, разделенные точкой с запятой.

Передача параметров в подпрограмму может осуществляться несколькими способами. Параметры процедур и функций могут быть следующих видов:

- параметры-значения;
- параметры-переменные;
- параметры-константы;
- нетипизированные параметры.

Группа параметров без предшествующего ключевого слова является списком параметров-значений.

Группа параметров, перед которыми следует ключевое слово *Const* и за которыми следует тип, является списком параметров-констант.

Группа параметров, перед которыми стоит ключевое слово *Var* и за которыми

следует тип, является списком типизированных параметров-переменных.

Группа параметров, перед которыми стоит ключевое слово *Var* или *Const* за которыми не следует тип, является списком нетипизированных параметров-переменных.

Имена формальных и фактических параметров могут совпадать. Необходимо лишь помнить, что в этом случае объект основной программы с таким именем становится недоступным для непосредственного использования подпрограммой. Тип формального параметра может быть практически любым, однако в заголовке подпрограммы нельзя вводить новый тип. Например, нельзя писать

function SredArif (X: array[1..20]of integer): integer;

Чтобы правильно записать этот заголовок, следует в основной программе ввести тип-массив, а затем использовать его в заголовке:

type ch =array [1..50] of integer;
function SredArif(X: ch) : integer;

Список параметров, задаваемый в заголовке процедуры или функции, обеспечивает связь подпрограммы с вызывающей программой. Через него в подпрограмму передаются исходные данные и возвращается результат. При этом предусмотрено два принципиально отличающихся механизма передачи параметров - *по значению* и *по ссылке*. Синтаксически эти два способа отличаются употреблением слова *Var* перед соответствующей переменной в заголовке подпрограммы. Если это слово имеется, то переменная передается по ссылке, а если нет - по значению.

Параметры-значения

При вызове по значению в подпрограмме создаются переменные в соответствии с объявлениями в заголовке подпрограммы. Эти переменные существуют только на время выполнения подпрограммы. В вызывающей программе в качестве аргумента подпрограммы может использоваться не только переменная, но и выражение. В начале выполнения подпрограммы значение этой переменной или выражения присваиваются внутренней временной переменной подпрограммы. Когда подпрограмма завершается, используемые подпрограммой переменные не сохраняют своего значения, поэтому передачу данных по значению можно использовать только для передачи данных в подпрограмму, но не для получения от нее результатов. Тип параметра-значения может быть любым, за исключением файлового.

procedure MaxNumber(a,b: integer);
function MaxNumber(a,b: real):real;

Пример 3. Функция вычисления максимального элемента в массиве целых чисел. В основной программе объявлены тип-массив, массив этого типа и переменная целого типа:

type mas=array [1..50] of integer;

var A:mas;
integer;

Функция может иметь вид:

```
function maxel(x:mas;n:byte):integer;
var m:integer;
i:byte;
begin
m:= x[1];
for i:=2 to n do
if m< x[i] then m:= x[i];
maxel:= m;
end;
```

Вызов функции maxel выглядит так:

```
max:= maxel(a,k);
```

После окончания работы функции *maxel* переменной *max* будет присвоено значение максимального элемента массива A.

Параметры-переменные

При передаче параметров переменных в подпрограмму передаются их адреса (ссылки) в порядке, объявленном в заголовке подпрограммы. При вызове по ссылке в подпрограмме память под передаваемые переменные не отводится. Основная программа передает в подпрограмму не значение переменной, а ссылку на место в памяти основной программы, где расположена некоторая переменная. Подпрограмма, производя некоторые действия с этой переменной, в действительности производит действия с переменной основной программы, поэтому после выполнения процедуры изменения, совершенные с переменными основной программы, сохраняются. Этот механизм используется для получения от подпрограммы результатов ее выполнения. Понятно, что при вызове подпрограмм с передачей данных по ссылке невозможно использовать в качестве аргументов выражения или константы, так как они не имеют адреса для передачи.

Параметр-переменная указывается в заголовке подпрограммы с указанием ключевого слова *var*. Действие слова *var* распространяется до ближайшей точки с запятой, т.е. в пределах одной группы.

Procedure vvmass(var A:mas; var max,min:real; n: integer);

Где *A*, *max,min* - параметры-переменные, *n* - параметр-значение. Тип параметров-переменных может быть любым, включая и файловый.

Параметры-константы

Если значение параметра в подпрограмме изменяться не должно, то его следует передать как параметр-константу.

Объявление параметра-константы в заголовке подпрограмм имеет вид

const <имя константы>:<тип константы>

Попытка изменить данный параметр в подпрограмме обнаруживается компилятором как ошибка. При вызове из основной программы подпрограммы с таким параметром могут использоваться те же способы, что и при передаче параметров по значению:

- константа может быть задана в явном виде;
- может быть указана переменная или выражение совместимого с константой типа.

Например,

Procedure Primer (Const x:byte).

Пример 4. Функция вычисления максимального элемента в массиве целых чисел.

```
function maxel(const x:mas;n:byte):integer;
var m:integer;
i:byte;
begin
m:= x[1];
for i:=2 to n do
if m<x[i] then m:= x[i];
maxel:= m;
end;
```

Нетипизированные параметры

В языке Turbo Pascal можно использовать параметры-переменные и параметры-константы без указания типа. Параметр считается нетипизированным, если тип формального параметра-переменной в заголовке подпрограммы не указан, при этом соответствующий ему фактический параметр может быть переменной любого типа. Нетипизированными могут быть только параметры-переменные. Нетипизированный параметр, описанный с ключевым словом *var*, может модифицироваться, а нетипизированный параметр, описанный с ключевым словом const, доступен только по чтению.

В процедуре или функции у нетипизированного параметра-переменной тип отсутствует, т. е. он несовместим с переменными всех типов, пока ему не будет присвоен определенный тип с помощью присваивания типа переменной.

Хотя нетипизированные параметры дают большую гибкость, их использование сопряжено с некоторым риском. Компилятор не может проверить допустимость операций с нетипизированными переменными.

Пример 5. Использование параметров без типа

Function Get (var pl, p2; le:byte):.real;

Где *p1, p2* - параметры-переменные без типа (вместо них можно использовать, например, любые переменные простого типа, типа-массива, типа-записи и т. д.); *le* - параметр-значение.

Следует иметь в виду, что параметр без типа внутри подпрограммы типа не имеет и его перед использованием следует привести к конкретному типу, применяя идентификатор соответствующего типа, при этом полученный результат может быть любого размера.

Пример 6. Функция вычисления максимального элемента в массиве целых чисел.

Рассмотрим другой вариант подпрограммы примера, используя в качестве первого параметра параметр-переменную без типа.

```
function maxel(var x;n:byte):integer;
type  mas=array [1..50] of integer;
var m:integer;
i:byte;
begin
m:= mas(x)[1];
for i:=2 to n do
if m<mas(x)[i] then m:= mas(x)[i];
maxel:= m;
end;
```

В этом случае в качестве первого передаваемого параметра можно использовать любой массив (и не только массив), так что подпрограмма становится более универсальной. Тем не менее, здесь необходимо передавать в качестве второго параметра фактический размер информации, что не всегда удобно.

1.4 Локальные и глобальные переменные

Все переменные, которые использует подпрограмма, могут быть либо глобальными, либо локальными.

Глобальными называются переменные, объявленные в основной программе и доступные как программе, так и всем ее подпрограммам.

Локальными называются переменные, объявленные внутри подпрограммы и доступные только ей самой. Локальные переменные могут быть описаны как в заголовке подпрограммы, так и в разделе описания переменных. Если в подпрограмме описана локальная переменная, имя которой совпадает с именем некоторой глобальной переменной, то данная глобальная переменная становится недоступной в этой подпрограмме, и при указании идентификатора переменной произойдет обращение к локальной переменой подпрограммы, а не одноименной глобальной переменной.

Память для локальных (т.е. описанных в подпрограмме) переменных выделяется на время исполнения данной подпрограммы в специальной области, называемой стеком. При завершении работы подпрограммы память освобождается, поэтому все внутренние результаты работы подпрограммы не сохраняются от одного обращения к другому.

Обмен информацией между основной программой и подпрограммой может осуществляться только с помощью глобальных переменных и с помощью параметров подпрограммы.

Подпрограмма может использовать любые глобальные переменные кроме тех, которые имеют те же имена, что и ее локальные переменные.

Рассмотрим две почти одинаковые программы, решающие задачу: присвоить глобальной переменной X некоторое значение, а затем напечатать это число через специальную процедуру (рисунок 22).

Program Variant1;	***Program Variant2;***
Var *X : real;* *Procedure writeX;* *Var* *X : real;* *Begin* *write(X)* *End;* *Begin* *X := Pi;;* *writeX* *End.*	*Var* *X : real;* *Procedure writeX;* *Begin* *write(X)* *End;* *Begin* *X := Pi;* *writeX* *End.*

Рисунок 22 - Использование глобальных и локальных переменных

В первом варианте переменная *X* переопределена в подпрограмме, таким образом, в подпрограмме имеется локальная переменная *X*, ничем не связанная с переменной *X* основной программы. Поскольку этой локальной переменной не присвоено значение, будет напечатано случайное число, содержащееся в соответствующей ячейке памяти.

Во втором варианте программы переменная с именем *X* описана только в основной программе, поэтому она, как глобальная переменная, доступна в подпрограмме. В результате будет напечатано значение числа π.

Возникает вопрос, какова роль локальных переменных, нельзя ли все переменные описать как глобальные?

Процедура должна быть, по возможности, независима от основной программы, поэтому все переменные, нужные только в пределах процедуры, должны описываться как локальные. Общение основной программы с подпрограммой должно, как правило, идти через список параметров процедуры, что придает подпрограммам необходимую гибкость. Вместе с тем, излишне большое число параметров, передаваемое в подпрограмму при ее вызове, замедляет работу программы, поэтому не следует пренебрегать использованием в подпрограммах глобальных переменных.

Локальность или глобальность являются понятиями относительными. Программа с вложенными в нее подпрограммами представляет собой иерархическое дерево. Объект, локальный по отношению к более высокому уровню иерархии, ведет себя как глобальный по отношению к подпрограммам более низкого уровня.

2 Учебный пример

Вычислить суммы элементов главных диагоналей матриц A(10,10), B(15,15), C(20,20).

Исходный текст программы - Program lab_6:

```
Program lab_6;
{$APPTYPE CONSOLE}
uses   SysUtils;
Type
   Matrica=Array[1..20,1..20] of integer;
Var a,b,c:Matrica;
   i,j:byte;
Function Sum_Gl(var a:Matrica;n:byte):integer;
var s:integer;
Begin
s:=0;
FOR i:=1 to n do
s:=s+a[i,i];
 Sum_Gl:=s;
End;
Procedure Input(Var x:Matrica;n:byte);
Begin
for i:=1 to n do begin
for j:=1 to n do begin
x[i,j]:=random(10);
WRITE(x[i,j]:3); end; writeln end;
Writeln('Summa elementov glavnoi diagonali matrici: ',
Sum_Gl(x,n));
    END;
Begin
Randomize; Writeln('Matrica A:'); INPUT(a,3);
Writeln('Matrica B:'); INPUT (b,4);
Writeln('Matrica C]:'); INPUT(c,5);
Readln; End.
```

В основной программе **Program lab_6** объявлен тип *Matrica*, который используется в процедуре INPUT, описаны функция *Sum_Gl* и процедура INPUT. Процедура INPUT вызывается из основной программы, функция *Sum_Gl* - из процедуры INPUT.

Процедура INPUT реализует ввод матриц с помощью генератора случайных чисел, вывод их на экран в форме таблиц, а также результат вычисления сумм главных диагоналей матриц. Само вычисление реализовано в процедуре *Sum_Gl*. Процедура INPUT вызывается 3 раза для 3-х разных матриц. В заголовке процедуры описаны формальные параметры:

a - двумерный массив типа *Matrica* (параметр-переменная) и

n - размерность матрицы (параметр-значение) .

При вызове процедуры из основной программы в процедуру INPUT передаются фактические параметры:

при 1-м вызове: имя массива - *a*, размерность - 3;

при 2-м вызове: имя массива - *b*, размерность - 4;

при 1-м вызове: имя массива - *c*, размерность - 5.

Функция *Sum_Gl* реализует алгоритм вычисления сумм главных диагоналей матриц. В заголовке функции описаны формальные параметры-значения:

a - двумерный массив типа *Matrica* и *n* - размерность матрицы.

При вызове процедуры происходит аналогичное замещение формальных параметров фактическими. В теле процедуры происходит присваивание вычисленного значения идентификатору имени функции (*Sum_Gl:=s),* которое возвращается в точку вызова.

Блок-схема алгоритма представлена на рисунке 23.

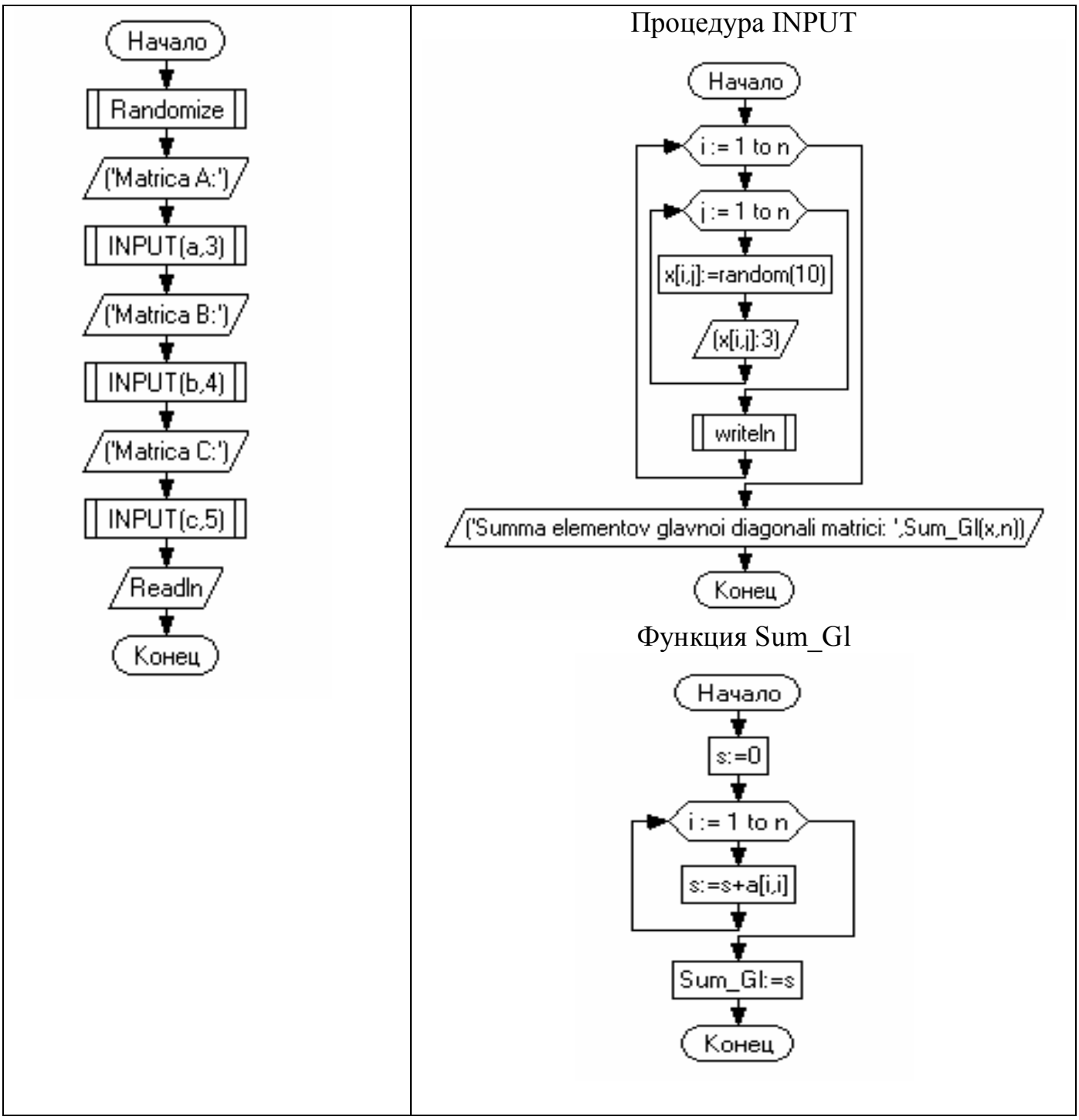


Рисунок 23 - Блок-схема алгоритма вычисления сумм элементов главных диагоналей матриц

ТЕСТ:
Matrica A:
3 0 3
6 6 5
6 6 2
Summa elementov glavnoi diagonali matrici: 11
Matrica B:
8 8 0 7
1 2 1 0
5 7 2 7
4 0 6 0
Summa elementov glavnoi diagonali matrici: 12
Matrica C:
4 7 6 0 2
0 8 0 1 6
4 1 4 0 9
9 1 4 7 3
6 7 1 9 9
Summa elementov glavnoi diagonali matrici: 32

3 Задания для самостоятельной работы

В соответствии с заданием (таблица 10) реализовать в процедуре ввод исходных матриц с помощью генератора случайных чисел, вывод матриц и результатов вычислений на экран. Реализовать вычисления в соответствующих функциях.

Таблица 10 - Задания к лабораторной работе № 6

Вариант	Характер вычислений	Матрицы
1	2	3
1	Найти средние значения для элементов массивов	X (10), Y(20), Z(15)
2	Найти максимальные элементы массивов	X (8), Y(10), Z(12)
3	Вычислить суммы элементов массивов	B1(20), B2(10), B3(12)
4	Вычислить суммы элементов под главной диагональю матриц	A(10, 10), B(15, 15)
5	Подсчитать число нулевых элементов для матриц	A(10, 10), B(15,15)
6	Найти минимальные элементы массивов	X1 (10), X2(20), X3(10)
7	Определить число положительных элементов в массивах	A1(20), A2(10), A3(12)
8	Вычислить суммы отрицательных элементов массивов	X1(8), X2(10), X3(20)
9	Найти место расположения (индексы) наибольших элементов массивов	X(10), Y(16)
10	Вычислить суммы квадратов элементов матриц	N(6,6), M(7,7), P(10,10)

Продолжение таблицы 10

1	2	3
11	Определить число отрицательных элементов в массивах	X(10), Y(8), Z(12)
12	Вычислить суммы положительных элементов матриц	A(15, 10), B(10,15)

4 Контрольные вопросы

1. Укажите целесообразность использования подпрограмм.
2. Что такое подпрограмма, каков механизм использования подпрограмм?
3. Что такое формальные и фактические параметры, как они согласуются между собой?
4. Какие типы данных могут быть формальными и фактическими параметрами?
5. В каких случаях можно и целесообразно использовать функцию?
6. Как описывается функция?
7. Как осуществляется вызов функции?
8. В каких случаях можно и целесообразно использовать процедуру?
9. Как описывается процедура?
10. Как осуществляется вызов процедуры?
11. Назовите главное отличие процедуры от функции.
12. Каковы особенности локальных и глобальных переменных?
13. Как осуществляется передача параметров-значений?
14. Как осуществляется передача параметров-переменных?

ЛАБОРАТОРНАЯ РАБОТА № 7

ТЕМА: **ОБРАБОТКА СИМВОЛЬНЫХ И СТРОКОВЫХ ДАННЫХ**

Цель работы: овладение практическими навыками алгоритмизации и программирования задач, обрабатывающих символьные и строковые данные, использования встроенных процедур и функций для обработки строк.

1 Теоретическое введение

1.1 Символы и строки символов

Наряду с числовыми данными в языке Pascal используется алфавитно-цифровая или символьная информация, которая включает в себя заглавные, строчные буквы, цифры от 0 до 9 и вспомогательные символы. Для описания символьных переменных используется тип данных *char* или *string*.

Символьный тип (*char*) - это тип данных, состоящих из одного символа (знака, буквы, кода). Все символы упорядочены в соответствии с принятым в ЭВМ коде (например ASCII). При этом порядковый номер символов называется кодом (например, код латинского символа 'A ' равен 65; символа цифры '3' равен 51). Объявление символьного типа в программе:

***var** simv: **char**;*

Значением типа *char* может быть любой символ из набора ASCII.

Если символ имеет графическое представление, то в программе он записывается заключенным в одиночные кавычки (апострофы), или соответственно своим кодом. Если символ не имеет графического представления, например, символ табуляции или символ возврата каретки, то он может быть представлен только кодом, состоящим из префикса # и ASCII-кода символа. Примеры ASCII-кодов некоторых символов приведены в таблице 11.

Таблица 11 - Примеры ASCII-кодов

Символ	Ж	S	Пробел	Перевод строки	Возврат каретки
Графическое представление	'Ж'	'S'	' '		
Код ASCII	#134	#83	#32	#10	#13

Допустимые операции над символьным типом:

- присваивание;
- сравнение: <, >, >=, <=, <>, =.

Большим считается тот символ, который имеет больший код ASCII.

Для символьных данных существуют две стандартные функции преобразования:

1) ORD (C) - принимает значение кода символа *C*, например:
   ORD('A ') = 65;
2) CHR(I) - значением функции является символ с кодом *I*, например:

CHR (65) = A;
CHR(ORD(A)) = A;

Строкой называется последовательность заданной длины, состоящая из символов.

Строковая константа есть последовательность символов, заключенная в апострофы. Например: 'это строковая константа', '272'.

Строковая переменная описывается в разделе описания переменных следующим образом:

var *<идентификатор>* : ***string****[<максимальная длина строки>];*

Например:

Var Str1: String[3];

Str2: String;

Параметр длины может и не указываться в описании. В таком случае подразумевается, что он равен максимальной величине - 255.

В данном примере максимальная длина переменной *Str1* равна 3, а переменной *Str2* - 255 символов.

Строка в языке Pascal трактуется как цепочка символов. Строка похожа на одномерный массив символов: она имеет определенную длину (не больше некоторого числа), к любому символу можно обратиться как к элементу массива по его номеру:

Str1[i] − это обращение к i-му элементу строки *Str1;*

Str1[1]:='A';

Str2[2]:='M'

Str1[3]:='D'.

Важнейшим отличием строк от символьного массива является то, что строки могут динамически менять свою длину. При этом необходимо помнить, что память выделяется по максимуму. Самый первый байт в строке имеет индекс 0 и содержит текущую длину строки. Таким образом, строка имеет две разновидности длины:

- общая длина строки, которая характеризует размер памяти, выделяемой при ее объявлении;
- текущая длина строки (меньше или равна общей длине), которая показывает количество символов в строке в конкретный момент времени выполнения программы.

Переменные типа *String* выводятся на экран посредством стандартных процедур *Write* и *Writeln* и вводятся с помощью стандартных процедур *Readln* и *Read*. То есть вводятся и выводятся не поэлементно, как массивы, а целиком. Если при вводе задать символов больше, чем максимально допустимо, то лишние символы будут проигнорированы.

1.2 Операции со строками

В языке Pascal имеется два основных способа обработки переменных типа *String*. Первый способ предполагает обработку всей строки как единого целого, т.е. единого объекта. Кроме того (это второй способ), можно рассматривать строку как составной объект, состоящий из отдельных символов, то есть элементов типа *Char*,

которые при обработке доступны каждый в отдельности.

Тип *String* и стандартный тип *Char* совместимы. Строки и символы могут употребляться в одних и тех же выражениях.

Строковые выражения строятся из строковых констант, переменных, функций и знаков операций. Над строковыми данными допустимы операции сцепления и операции отношения

Значение строковой переменной может быть присвоено с помощью оператора присваивания или процедуры ввода:

line1:='программирование';

readln(line2);

1.2.1 Сцепление (конкатенация)

Операция *сцепления (конкатенации)* (+) применяется для соединения нескольких строк в одну результирующую строку. Сцеплять можно как строковые константы, так и переменные.

Под сцеплением понимается последовательное объединение нескольких строк:

Var Str1, Str2, Str3: String[20];

Str1:='Символы'; {текущая длина строки Str1 - 7}

Str2:='и строки символов'; {текущая длина строки Str1 - 17}

Str3:=Str1+' '+Str2;

Строка *Str3* имеет значение *'Символы и строки сим'*. В данном примере максимальная длина строки *Str3* равна 20 символам, поэтому будут взяты только первые 20 символов суммы строк, а остальные рассматриваться не будут.

Паскаль позволяет выполнять операции объединения (сцепления) нескольких строк в процессе их присвоения какой-либо переменной:

Str3:= 'Символы'+'и строки'+'символов'.

В результате такой операции в переменной *Str3* будет то же самое содержимое, что и в предыдущем примере.

Сцепить строки можно также при помощи функции *Concat*, которая будет рассмотрена ниже (таблица 16).

1.2.2 Сравнение

Над значениями строкового типа определены операции сравнения с обычным смыслом (<, <=, >, >=, =, <>), при выполнении которых действуют следующие правила: более короткая строка всегда меньше длинной; а если длины сравниваемых строк равны, то происходит поэлементное сравнение символов этих строк с учетом лексикографической упорядоченности значений стандартного символьного типа *char*:

'Balkon'<'balkon'(Ord(' B')<Ord('b'));

'balkon '>'balken'(Ord('o')>Ord('e')) ;

'balkon'>'balk' (длина первой строки больше);

'кот'='кот' (равны по длине и совпадают посимвольно).

Можно использовать любые операции отношения и их комбинации в условных операторах. Их результат − это одно из двух значений: *True* или *False*.

При этом по отношению к отдельному символу строки возможны все те же операции, что и к переменной типа *Char*.

1.3 Стандартные процедуры и функции

Паскаль предоставляет в распоряжение программиста целый ряд встроенных функций и процедур, предназначенных для обработки строк. Рассмотрим наиболее важные из них.

Процедуры

1. **Delete** (S, Poz, N) - удаление N символов из строки S, начиная с позиции Poz (таблица 12).

Таблица 12 - Примеры описания и выполнения процедуры Delete

Исходное значение S	Оператор	Конечное значение S
'abcdefg'	Delete(S, 3, 2)	'abefg'
'abcdefg'	Delete(S, 2, 6)	'a'

В результате выполнения процедуры уменьшается текущая длина строки в переменной *S*.

2. **Insert** (S1, S2, Poz) - вставка строки S1 в строку S2, начиная с позиции Poz (таблица 13).

Таблица 13 - Примеры описания и выполнения процедуры **Insert**

Исходное значение S2	Оператор	Конечное значение S2
'ЭВМ PC'	Insert('IBM-', S2, 5)	'ЭВМ IBM-PC'
'Рисунок 2'	Insert('N', S2, 6)	'Рисунок N 2'

В результате выполнения процедуры первая строка не изменяется, а вторая получает новое значение.

3. **Str** (N, S) - преобразует числовое значение N в строковое и присваивает результат строке Str1, N - целые и вещественные (таблица 14).

Таблица 14 - Примеры описания и выполнения процедуры **Str**

Значение N	Оператор	Значение S
1234	Str (N, S)	'1234'
452.567	Str(N, S)	'452.567'

4. **Val** (S, N, K) - преобразует строковое значение в числовое (таблица 15). Если данная строка действительно является записью числа (целого или вещественного), то значение K=0, а N − это искомое число, иначе K будет равно номеру первого символа, с которым процедура Val "не справилась".

Таблица 15 - Примеры описания и выполнения процедуры **Val**

Исходное значение S	Оператор	Конечное значение S2
'1234'	Val(S, N, k)	n=1234, k=0;
'234.56'	Val(S, N, k)	n=234.56, k=0;
'12-45'	Val(S, N, k)	k=3, так как знак "−" в записи чисел может быть только на первом месте

Строковое выражение S преобразуется в величину целочисленного или

вещественного типа и записывается в переменной N. Если при этом ошибок не обнаруживается, то K будет равно 0. В противном случае значение K будет равно номеру позиции первого ошибочного символа и N будет неопределено. Строка S не должна содержать незначащих пробелов, переменная N может быть целой или вещественной, а переменная K - только целой.

Функции

1. **Concat** (S1, S2, …, SN) - выполняет сцепление (конкатенацию) строк S1, S2,…, SN в одну строку (таблица 16). Результат - строка.

Таблица 16 - Примеры описания и выполнения функции **Concat**

Выражение	Результат
Concat('Turbo ', 'Pascal')	'Turbo Pascal'
Concat('Процедуры', 'и', 'Функции'	'Процедуры'и 'Функции'

2. **Copy** (S, Pozition, N) - выделяет из строки S подстроку длиной N символов, начиная с позиции Pozition. Здесь N и Pozition - целочисленные выражения (таблица 17). Результат - строка.

Таблица 17 - Примеры описания и выполнения функции **Copy**

Значение S	Выражение	Результат
‘IBM PC’	Copy(S, 5, 2)	‘PC'
‘Turbo Pascal’	Copy(S, 1, 5)	‘Turbo’

3. **Length** (S) - определяет текущую длину строки S (таблица 18). Результат - значение целого типа.

Таблица 18 - Примеры описания и выполнения функции **Length**

Значение S	Выражение	Результат
'test-5'	Length(S)	6
'(A+B)*C'	Length(S)	7

4. **Pos** (S1, S2) - обнаруживает первое появление в строке S2 подстроки S1 (таблица 19). Результат - целое число, равное номеру позиции, где находится первый символ подстроки S1. Если в S2 подстроки S1 не обнаружено, то результат равен 0.

Таблица 19 - Примеры описания и выполнения функции **Pos**

Значение S2	Выражение	Результат
'abcdef'	Pos('cd', S2)	3
'abcdcdef'	Pos('cd', S2)	3
'abcdef'	Pos('k', S2)	0

5. UpCase (Key) - возвращает прописную латинскую букву для строчной буквы, передаваемой ей в параметре вызова (таблица 20). Функция не работает с кириллицей.

Таблица 20 - Примеры описания и выполнения функции **UpCase**

Значение Key	Выражение	Результат
'a'	UpCase('a');	'A'
'd'	UpCase('d');	'D';
'z'	UpCase('z');	'Z'

6. LowerCase (Key) - возвращает строчную латинскую букву для прописной буквы, передаваемой ей в параметре вызова (таблица 21). Функция не работает с

кириллицей.

Таблица 21 - Примеры описания и выполнения функции **LowerCase**

Значение Key	Выражение	Результат
'A'	LowerCase('A');	'a'
'D'	LowerCase('D');	'd';
'Z'	LowerCase('Z');	'z'

2 Учебный пример

Дана строка, состоящая из нескольких слов, между словами один пробел, в конце строки - точка. Подсчитать количество слов и вывести на экран только те из них, которые начинаются с буквы 'a'.

Исходный текст программы (Program lab_7) и блок-схема алгоритма представлены на рисунке 24.

```
Program Project_lab_7;
{$APPTYPE CONSOLE}
uses
  SysUtils;
Const n=30;
Type Myarray_Str=Array [1..n] Of  String;
Var     A: Myarray_Str;  str: String;
          k: Byte; Var i: Integer;
begin
   Writeln('vvedite  stroku');
         Readln(str);
{Разобьем предложение на отдельные слова и каждое будем хранить как элемент массива строк}
k:=1; {Пока не встретится пробел,
формируем очередное слово k, прибавляя по одной букве}
For i:=1 To Length(str)-1 Do
   If str[i]<>' ' Then a[k]:=a[k]+str[i]
Else
{Если это не последний символ, то
увеличиваем счетчик слов и начинаем
формировать очередное слово}
   If i<>Length(str)-1 Then
Begin k:=k+1; a[k]:=''; End;
Writeln('Vsego slov: ',k);
{Просматриваем все слова, если первый
символ очередного слова равен букве
'a', то выводим его}
For i:=1 To k Do If A[i][1]='a'
Then Write(A[i],' ');
Readln;
End.
```

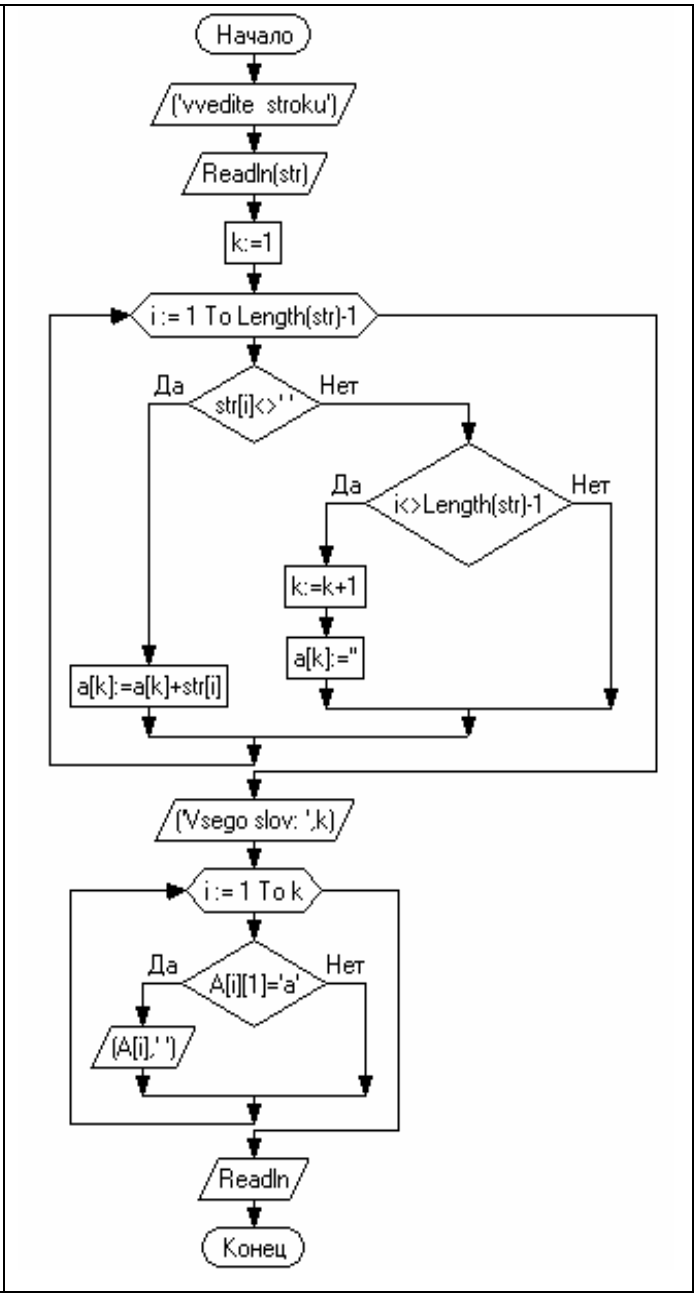


Рисунок 24 - Исходный текст программы и блок-схема алгоритма обработки строки

ТЕСТ:
Исходная строка:
address name and table about
Vsego slov: 5
address and about

3 Задания для самостоятельной работы

Символьная строка вводится с клавиатуры. Произвести в ней необходимые преобразования согласно индивидуальному заданию (таблица 22). Вывести на экран преобразованную строку.

Таблица 22 - Задания к лабораторной работе № 7

Вариант	Условие задачи
1	Удалить из строки буквы 'b', 'c', 'd'
2	Если какой-либо символ встречается в строке более одного раза, то первое вхождение этого символа оставить без изменения, второе и последующие удалить.
3	Строка содержит слова, разделенные несколькими пробелами. Оставить по одному пробелу между словами.
4	Подсчитать, сколько раз в данной строке встречается символ 's'.
5	Написать строку задом наперед.
6	Определить, содержит ли данная строка цифры, если да, то вывести их на экран
7	В строке найти цифры и удалить их
8	Строка состоит из цифр и букв. Строчные латинские буквы заменить на прописные.
9	Вывести на экран только первые символы слов.
10	Заменить все вхождения подстроки 'del' на 'Insert'.
11	Подсчитать сумму цифр, входящих в данную строку.
12	Удалить из строки цифры и вывести на экран строку без цифр.

4 Контрольные вопросы

1. Как объявить величину символьного типа, строкового типа?
2. К каким типам данных относятся строки?
3. Какова максимально возможная длина строки?
4. С величиной какого типа данных совместим по присваиванию отдельный символ строки?
5. Какие операции можно выполнять над строковыми данными?
6. Какие существуют стандартные функции для величин строкового типа?
7. Какие существуют стандартные процедуры для величин строкового типа?
8. Как осуществляется доступ к отдельному символу строки?
9. Почему значение отношения 'IBM'<>'ibm' равно TRUE?
10. Какая функция (процедура) является аналогом операции сцепления (+) при работе со строками?

ЛАБОРАТОРНАЯ РАБОТА № 8

ТЕМА: **СОСТАВНЫЕ ДАННЫЕ НЕОДНОРОДНОЙ СТРУКТУРЫ. ФИКСИРОВАННЫЕ ЗАПИСИ**

Цель работы: овладение практическими навыками алгоритмизации и программирования задач ввода и вывода записей, их обработки, использования записей как структурных элементов баз данных.

1 Теоретическое введение

1.1 Понятие записи

Отдельно взятые массив, множество или файл всегда включают элементы одинакового типа. Если необходимо логически объединить компоненты разных типов, то применяется тип *запись*.

Запись Паскаля – структурированный комбинированный тип данных, состоящий из фиксированного числа компонент (полей) разного типа.

Например, анкетные данные о студенте вуза могут быть представлены в виде информационной структуры (рисунок 25).

Рисунок 25 - Двухуровневая структура данных

Такая структура называется двухуровневым деревом. В Паскале эта информация может храниться в одной переменной типа *record* (запись).

Запись может быть объявлена в разделе *type*:

type *< имя _ типа >=**record***

<имя_поля1>: тип;

<имя_поля2>: тип;

……..

<имя_поля K >: тип

end ;

где *record* – служебное слово, а *<имя_типа>* и *<имя_поля>* - правильные идентификаторы языка. Поля записи могут иметь любой тип, кроме файла, в частности, сами могут быть записями.

Описание анкеты студента в Паскале будет выглядеть так:

type *anketa=**record***

fio: string[45];

pol: char;

dat_r: string[8];

adres: string[50];

curs: byte;

grupp: string[3];

***end**;*

Такая запись Паскаля, так же как и соответствующее ей дерево, называется двухуровневой.

После того, как определен тип записи Паскаля, можно определять переменную этого типа. Переменная определяется путем задания ее идентификатора и указания типа.

***var** student: **anketa**;*

1.2 Обращение к записи и к элементам записи

Обращение к записи в целом допускается только в операторах присваивания, где слева и справа от знака присваивания используются имена записей одинакового типа. Элементы записи называются *полями*, а обращение к ним производится через использование их имен – *идентификаторов* полей. Практически, поля записи обрабатываются точно так же, как и любые другие переменные. Но в отличие от обычной переменной имена полей должны предваряться ссылкой на идентификатор записи Паскаля и отделяться от него точкой:

<имя_записи>.<имя_поля>

Такая запись называется уточняющий или составной идентификатор, а *<имя_записи>* - префикс. Например, чтобы обратиться к полю *curs* переменной *student* , необходимо указать следующее составное имя:

student.curs:=3;

Использование полей записи Паскаля в выражениях и условиях идентично использованию обычных переменных.

1.3 Операции над записями

Единственная операция, которую можно произвести над однотипными записями Паскаля – это присваивание. Все другие операции, в том числе ввод и вывод, производятся над отдельными полями записи.

Префикс – обязательная предшествующая часть составного идентификатора для имен полей в структуре типа запись Паскаля. Если в программе необходимо последовательно обращаться к полям одной и той же записи, то префикс можно не указывать. Такая возможность реализуется при помощи *оператора присоединения*, который в общем виде выглядит так:

***with** <имя_записи> **do** <оператор>*

Он позволяет, один раз указав имя переменной типа "запись" после слова *with*, работать в пределах одного оператора (простого или составного) с именами полей как с обычными переменными, т.е. не писать громоздких составных имен.

Оператор представляет собой область действия оператора присоединения, в пределах которой можно не использовать составные имена. Например, ввод записей с использованием оператора присоединения будет выглядеть так:

```
    for I:=1 to 100 do
  with student[I] do
  begin
    write('введите сведения о', I , '-м студенте');
```

```
    write('введите фамилию, имя и отчество');readln (fio);
    write('введите дату рождения'); readln (dat_r);
    write('введите курс');  readln(curs);
    write('введите группу');readln (grupp);
  end;
```

2 Учебный пример

Сформировать и вывести на экран базу данных "Сотрудники", состоящую из N записей по 5 полей в каждой: фамилия, адрес, телефон, образование, оклад. Отобрать и вывести на экран только записи о сотрудниках с высшим образованием.

Исходный текст программы (Program lab_8) и блок-схема алгоритма представлены на рисунке 26.

```
Program lab_8;
{$APPTYPE CONSOLE}
uses  SysUtils;
 type z=record
     f:string[15];      adres:string[20];
     tel:string[6];     obraz:string[10];
     oklad:integer;       end;
var  sotr:array [1..10] of z;
     i,n:integer;
begin
write('Vvedite kol-vo zapisej  '); readln(n);
writeln('Svedeniya o sotrudnikakh:');
for i:=1 to n do begin
with sotr[i] do  begin
    write('Familija:'); readln(f);
    write('Adres:'); readln(adres);
    write('Telefon:');  readln(tel);
    write('Obrazovanie:');  readln(obraz);
    write('Oklad:'); readln(oklad);
end; end;
writeln('Spisok sotrudnikov');
writeln('Familija   Adres     Telefon
Obrazovanie Oklad');
for i:=1 to n do
   with sotr[i] do writeln(f:10, adres:10,
tel:7, obraz:10, oklad:5); writeln;
writeln('Spisok sotrudnikov s vishim
           obrazovaniem ');
writeln('Familija   Adres   Telefon
Obrazovanie Oklad');
       for i:=1 to n do begin
   with sotr[i] do if obraz='vicshee' then
   writeln(f:10, adres:10, tel:7, obraz:10,
            oklad:5);end;
  readln; end.
```

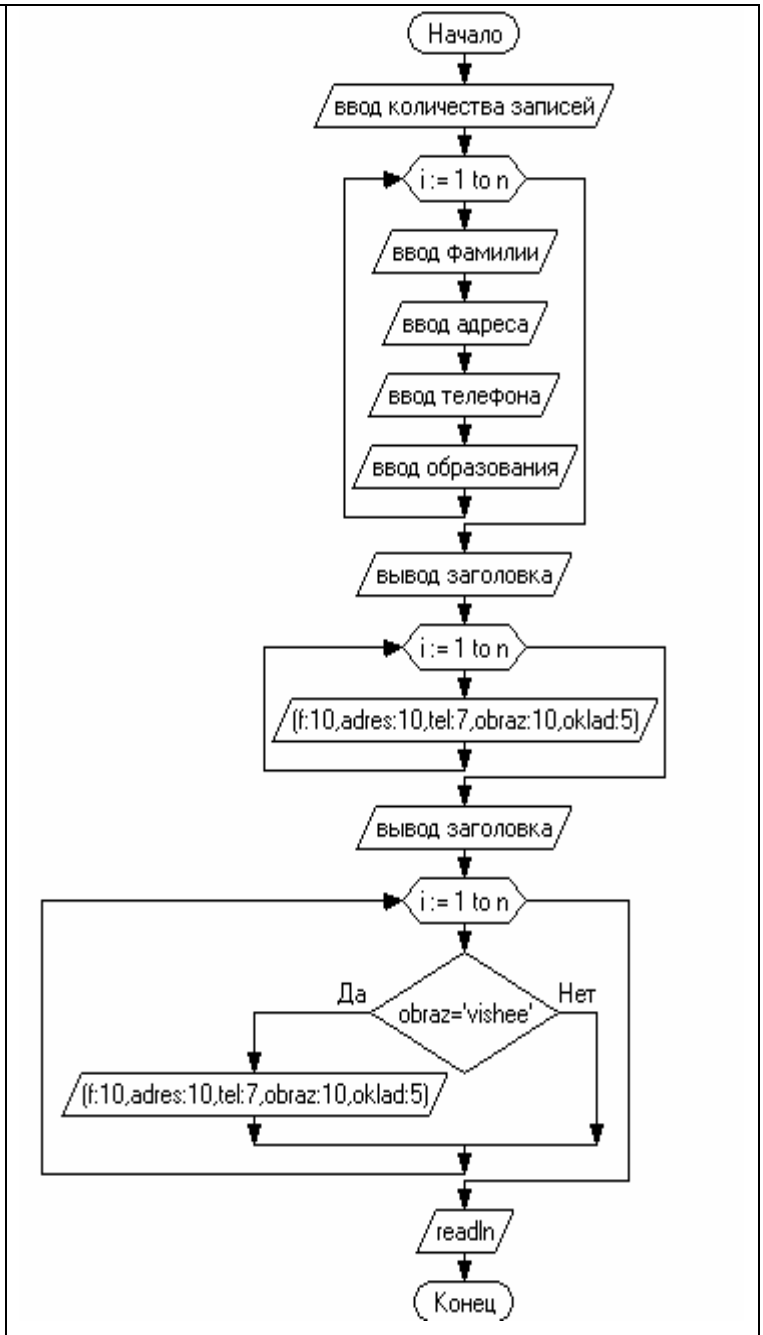


Рисунок 26 - Исходный текст программы и блок-схема алгоритма обработки записей

ТЕСТ:

```
N=3
Spisok sotrudnikov
Familija        Adres           Telefon     Obrazovanie     Oklad
Ivanov          pr.Mira, 17     111111      srednee         1000
Petrov          ul.Mira, 20     222222      vicshee         2000
Sidorov         ul.Sadovaya,1   300000      vicshee         3000
Ivanova         pr.Mira, 17     111111      srednee         4000

Spisok sotrudnikov s vishim obrazovaniem
Petrov          ul.Mira, 20     222222      vicshee         2000
Sidorov         ul.Sadovaya,1   300000      vicshee         3000
```

Ввод и вывод записей выполняется в цикле с применением оператора *With*. Введенные записи о сотрудниках хранятся в массиве *Sotr*: его размерность - *10*, тип элементов – *record (z)*. Массив читается в цикле дважды:

1-й раз - для вывода всех записей базы данных;

2-й раз - для отбора записей о сотрудниках с высшим образованием и вывода отобранных записей в отдельном списке.

3 Задания для самостоятельной работы

Разработать и вывести на экран базу данных согласно заданию (таблица 22). В базе должно быть не менее 3-х записей, содержащих сведения как соответствующие критерию отбора, так и не соответствующие ему. Отобрать и вывести на экран записи согласно критерию отбора.

Таблица 22 - Задания к лабораторной работе № 8

Вариант	База данных	Критерий отбора
1	2	3
1	"Абитуриенты" (оценки по 3-м экзаменам, средний балл)	Средний балл не ниже 4,5
2	"Отдел кадров университета" (фамилия, имя, отчество, стаж педагогической деятельности)	Стаж педагогической деятельности более 10 лет
3	"Научно-техническая библиотека" (фамилия автора, название книги, город, издательство, год выпуска, тематика)	Книги по программированию
4	"Легковые автомобили" (марка, цвет, год выпуска, цена)	Стоимость менее 500 тыс. руб.
5	"Администратор железнодорожной кассы" (№ поезда, пункты отправления и прибытия, время отправления и прибытия)	Поезда, следующие до Оренбурга
6	"Магазин по продаже ПК" (процессор, ОЗУ, винчестер, цена)	Цена менее 10 тыс. руб

Продолжение таблицы 22

1	2	3
7	"Товары на складе" (название, количество, стоимость 1 единицы товара)	Товары, суммарная стоимость которых более 20 тыс. руб.
8	"Спортсмены" (фамилия, имя, разряд, рост, вес)	Спортсмены, рост которых превышает 175 см
9	"Телефонный справочник" (фамилия, имя, отчество, адрес, № телефона)	Абоненты, № телефона которых начинается на "22"
10	"Сведения о родственниках" (фамилия, имя, отчество, дата рождения, № телефона)	Родственники, родившиеся в январе
11	"Домашняя библиотека" (фамилия, имя, отчество автора книги, название, год издания, тематика)	Книги автора "Толстой"
12	"Кондитерская" (название торта, вид (бисквит, слоеный, песочный, калорийность, срок годности, цена)	Вид торта - бисквитный

4 Контрольные вопросы

1. Чем отличается тип "запись" от других структурированных типов?
2. Как определяется тип записи?
3. Что такое поле записи?
4. Какие типы может иметь поле записи?
5. Могут ли поля записи быть разных типов?
6. Как обратиться к отдельному полю записи?
7. Какие операции возможны над данными типа "запись"?
8. Что такое "оператор присоединения"? В каких целях он используется?
9. Как заполнить массив записей?

Список рекомендуемой литературы

1. Алексеев Е.Р., Чеснокова О.В. Турбо Паскаль 7.0. Учебное пособие. - М.: НТ Пресс, 2009.- 320 с.
2. Аляев Ю.А. Козлов О.А. Алгоритмизация и языки программирования Pascal, C++, Visual Basic: Учебное-справочное пособие. - М.: Финансы и статистика, -2002.- 320 с.: ил.
3. Андреева Т.А. Программирование на языке Pascal. Учебное пособие. - М.: Бином, 2009. - 243 с.
4. Архангельский А.Я. Язык Pascal и основы программирования в Delphi. - М.: Бином, 2008. - 496 с.
5. Белецкий Я. Турбо Паскаль с графикой для персональных компьютеров/Пер. с польск. Д.И. Юренкова. - М.: Машиностроение, 1991.-320 с.
6. Бородин Ю.С. и др. Паскаль для персональных компьютеров: Справ, пособие/ Ю.С. Бородич, А.Н. Вальвачев, А.И. Кузьмич.-Мн.: Выш. шк.: БФ ГИТМП «НИКА», 1991.-365 с.
7. Быковец Н.П. Методические указания для выполнения лабораторных работ по дисциплине "Информатика". - Новотроицк, 2004 г. - 79 с.
8. Васильев П.П. Турбо Паскаль в задачах и примерах. Освой самостоятельно. Учебное пособие. -М.: Финансы и статистика, 2002.- 496 с.
9. Вирт Н. Алгоритмы и структуры данных: Пер. с англ. - М.: Мир, 1989. - 360 с: ил.
10. Гордон Я. Тонкости программирования на языке Паскаль: Учебное пособие по программированию на ПК / Я. Гордон. - М.: Бук_пресс, 2006. - 320 с.
11. Грызлов В.И., Грызлова Т.П. Турбо Паскаль 7.0. - М.: ДМК, 2000. - 416 с.
12. Епанешников А.М. Епанешников В.А. Программирование в среде TURBO PASCAL 7.0. - М.: Диалог-МИФИ, 1995.- 282 с.
13. Зеленяк О.П. Современный задачник по Турбо Паскалю. - М.: ДМК-Пресс, 2010.- 320 с.
14. Зелковиц М., Шоу А., Гэннон Дж. Принципы разработки программного обеспечения: Пер. с англ. - М.: Мир, 1982. - 386 с: ил.
15. Культин Н.Б. Программирование в Turbo Pascal 7.0 и Delphi / 2-е изд., перераб. и доп.- СПб.: БХВ - Санкт-Петербург, 2001. - 416 с: ил.
16. Культин Н.Б. Программирование на Object Pascal в Delphi 5. СПб.: БХВ - Санкт-Петербург, 1999 - 464 с: ил.
17. Культин Н.Б. Turbo Pascal в задачах и примерах. - СПб.: БХВ-Петербург, 2006.- 256 с.: ил.
18. Марченко А.И., Марченко Л.А. Программирование в среде TURBO PASCAL 7.0./ Под ред. В.П. Тарасенко. - 5-е изд., перераб. и доп.- К.: Век, 1999.- 464 с.
19. Меженный О.А. Turbo Pascal. - М.: Вильямс Диалектика, 2008.- 336 с.
20. Немнюгин С. Turbo Pascal. - СПб.: Питер, 2000.- 496 с.
21. Немнюгин С. Turbo Pascal: практикум. - СПб.: Питер, 2000.- 256 с.
22. Попов В.Б. Turbo Pascal для школьников. Учебное пособие. - М.: Финансы и статистика, 1999. - 528 с.
23. Практическое руководство по программированию: Пер. с англ. Б. Мик, П.

Хит, Н. Рашби и др.; под ред. Б. Мик, П. Хит, Н. Рашби. - М.: Радио и связь, 1986. 168 с: ил.

24. Программирование на языке Паскаль: задачник / под ред. Усковой О.Ф. - СПб.: Питер, 2003.- 336 с.: ил.

25. Рапаков Г.Г., Ржеуцкая С.Ю. Turbo Pascal для студентов и школьников. - М.: BHV, 2007.- 352 с.

26. Сухарев М. Turbo Pascal 7.0. Теория и практика программирования.- М.: Наука и техника, 2007.- 544 с.

27. Тарануха Н.А., Гринкруг Л.С. Обучение программированию: язык Pascal. - М.: Солон-пресс, 2009.- 384 с.

28. Фаронов В.В. Turbo Pascal 7.0. Начальный курс. Учебное пособие. - М.: Издательство "ОМД Групп", 2003.- 616 с.: ил.

29. Фаронов В.В. TurboPascal 7.0 Практика программирования. - М.: КноРус, 2009.- 416 с.

30. Федоренко Ю. Алгоритмы и программы на Turbo Pascal. Учебный курс.- СПб.: Питер, 2001.- 240 с. ил.

31. Федоров А. Особенности программирования на Borland Pascal,- Киев: Диалектика, 1994.- 144 с.

32. Фокс Дж. Программное обеспечение и его разработка: Пер. с англ. - М.: Мир, 1985. - 368 с: ил.

33. Шпак Ю.А. Turbo Pascal 7.0 на примерах /Под ред. Ю.С. Ковтанюка - К.:Издательство Юниор, 2003. - 496 с.: ил.

34. Энциклопедический словарь юного математика. Сост. А.П. Савин. - М.: Педагогика, 1989.- 352 с: ил.

35. Язык компьютера: Пер. с англ. / Под ред. и с предисл. В.М. Курочкина. - М.: Мир, 1989. - 240 с: ил.

Приложение А

Тетрадь для отчетов

Тетрадь для отчетов - обычная ученическая тетрадь объемом 18-24 листов. Следует подписать ее на лицевой стороне по образцу:

Тетрадь

для отчетов по лабораторным работам

по дисциплине "Информатика" (ч. I)

студента группы _____

_____

(Фамилия Имя)

На 1-й странице поместить таблицу для фиксации результатов защиты работ:

№ работы	Срок защиты (№ консультации)		Подпись преподавателя
	по плану	фактически	
1.	1		
2.	2		
3.	3		
4.	4		
5.	5		
6.	6		
7.	7		
8.	8		

Отчет по каждой ЛР должен размещаться с четной страницы, чтобы одновременно были видны блок-схема алгоритма и исходный текст программы.

Приложение Б

ОБРАЗЕЦ ОТЧЕТА

Лабораторная работа № 2

Тема: Программирование алгоритма разветвляющейся структуры

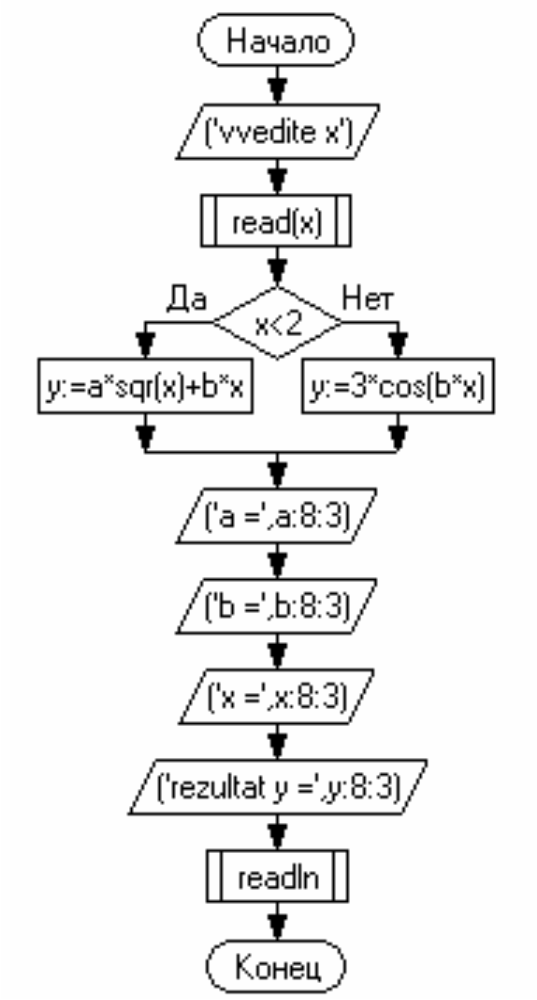
Цель работы: Овладение практическими навыками разработки и программирования вычислительного процесса разветвляющихся структур, развитие навыков по отладке и тестированию программ.

Вариант 1

Выполнить на компьютере программу обработки алгоритма разветвляющейся структуры: вычислить значение функции при заданных условиях и исходных данных

$$y = \begin{cases} ax^2 + \text{bx} & x < 2 \\ 3\cos bx & x \geq 2 \end{cases} \qquad a = -0{,}2 \qquad b = 2$$

Блок-схема алгоритма



Исходный текст программы

```
program Project2;
   var x, y: real;
const a=-0.2;
const b=2;
begin
     writeln ('vvedite x');
     read (x);
      if x<2 then y:= a*sqr(x) + b*x
               else   y:=3*cos (b*x);
      writeln ('a =', a:8:3);
      writeln ('b =',b:4);
      writeln ('x =', x:8:3);
      writeln ('rezultat  y =', y:8:3);
     readln;
end.
```

ТЕСТ:

a = -0.2	a = -0.2
b = 2	b = 2
x = 1	x = 3
y = 1.800	y = 2.881

Файл исходного текста программы:
Z:\Ivanov Ivan\Лаб1\Project2

Приложение В

Встроенные (стандартные) процедуры и функции для обработки числовых данных

Таблица В.1 - Функции

Функция	Назначение	Пример вызова	Результат
abs(*число*)	абс. значение числа	abs(-3.5)	+3.5
arctan(*тангенс -угла*)	арктангенс числа	arctan(0)	0
cos(*угол*)	косинус угла(рад.)	cos(pi)	-1
exp(*число*)	экспонента	exp(1)	2.718281828...
frac(*число*)	дробная часть числа	frac(3.5)	0.5
int(*число*)	целая часть числа	int(3.5)	3.0
ln(*число*)	нат. логарифм	ln(2.718281828)	~1.0
odd(*число*)	проверка нечетности	odd(3)	True
pi	число пи	pi	3.141592...
round (*число*)	округление до ближайшего целого	round (10.6)	10
random(*число*)	«случайное» число	random(10)	Число в [0;10]
sin(*угол*)	синус угла(рад.)	sin(pi)	0
sqr(*число*)	квадрат числа	sqr(2.0)	4.0
sqrt(*число*)	квадратный корень	sqrt(25.0)	5.0
trunk (*число*)	отсечь дробную часть	trunk (5.78)	5

Таблица В.2 - Процедуры

Процедура	Назначение	Пример вызова	Результат
inc(*число*)	увеличить на 1	inc(n)	n:= n + 1
dec(*число*)	уменьшить на 1	dec(n)	n:= n - 1

Таблица В.3 - Выражение некоторых функций через стандартные

Функция	Обращение	Реализуемое действие
tg x =	Sin (x) / Cos (x)	тангенс *x*
ctg x =	Cos (x) / Sin (x)	котангенс *x*
$\log_a x$ =	ln (x) / ln (a)	логарифм *x* по основанию *a*
x^y =	e^{ylnx}	возведение числа *x* в степень *y* (x>0)

ЮДИНА ВЕРА ИВАНОВНА

ИНФОРМАТИКА
Часть 1
Лабораторный практикум

Подписано в печать
06.12.2011
Формат 60х90 1/16 Печать офсетная Уч.-изд.л.4,81
Рег.№ 6 Тираж 100 экз.

Национальный исследовательский технологический университет «МИСиС»
Новотроицкий филиал
462359, Оренбургская обл., г. Новотроицк, ул. Фрунзе, 8.
E-mail: nfmisis@yandex.ru
Контактный тел. 8 (3537) 679729.

Отпечатано в типографии ООО «Медиа Сервис»
Оренбургская обл.,г.Орск, ул.Краматорская, 2б
Контактный тел. 8 (3537) 236161.