Белорусский национальный технический университет

Факультет информационных технологий и робототехники

Кафедра "Электропривод и автоматизация промышленных установок и технологических комплексов"

СОГЛАСОВАНО	СОГЛАСОВАНО
Заведующий кафедрой	Декан факультета
_____	_____
«03» сентября 2014г.	«02» октября 2014г.

ЭЛЕКТРОННЫЙ УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

«Информатика»

Для специальности 1-53 01 05 «Автоматизированные электроприводы»

Составители: Александровский Сергей Владимирович
Миронович Артём Викторович

Рассмотрено и утверждено
На заседании Совета ФИТР «02» октября 2014г.
протокол № 1

Перечень материалов

1. Конспект лекций по дисциплине
2. Лабораторный практикум по дисциплине
3. Методические указания по выполнению курсовой работы
4. Типовые задания к экзамену и зачёту
5. Типовая учебная программа по дисциплине

Пояснительная записка

Цели ЭУМК: Цель электронного учебно-методического комплекса по дисциплине «Информатика» заключается в приобретении студентами знаний современных средств вычислительной техники, умений использовать аппаратные и программные средства вычислительной техники для решения общих и специализированных задач, а так же навыков работы с компьютером на уровне пользователя.

Особенности структурирования и подачи учебного материала: Весь курс Информатики разделён на две части. В первой части рассматриваются основы функционирования компьютера, аппаратные и программные средства вычислительной техники, общие сведения о компьютерных сетях. Вторая часть курса полностью посвящена алгоритмизации и программированию на языке Турбо-Паскаль.

Рекомендации по организации работы с ЭУМК: При работе с УМК в первую очередь изучается теоретическая часть с подробным рассмотрением примеров, представленных в конспекте лекций. Для закрепления теоретических знаний проводятся лабораторные занятия в соответствии с предложенной тематикой. В ходе самостоятельной работы студенты выполняют курсовую работу по теме «Разработка алгоритма и Паскаль - программы по вычислению сложной функции». Контроль полученных знаний и умений осуществляется в виде письменного зачёта (1-я часть) и экзамена (2-я часть).

СОДЕРЖАНИЕ

Министерство образования Республики Беларусь
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Электропривод и автоматизация промышленных установок и технологических комплексов»

А.В. Миронович

ИНФОРМАТИКА

Конспект лекций
для студентов специальности
1-53 01 05 «Автоматизированные электроприводы»

В 2-х частях

Часть 1-я

Основы вычислительной техники

Учебное электронное издание

Минск 2010

Библиографическое описание:
Миронович, А. В. Информатика [Электронный ресурс]. В 2 ч. Ч.1. Основы вычислительной техники : [конспект лекций для специальности 1-53 01 05 "Автоматизированные электроприводы"] / А. В. Миронович ; Белорусский национальный технический университет, Кафедра "Электропривод и автоматизация промышленных установок и технологических комплексов" . - Электрон. дан.. - БНТУ, 2010.
УДК: 004(075.8)
Дата публикации: 2010
URI: http://rep.bntu.by/handle/data/1190
Дата добавления: 2012-02-13

ИНФОРМАТИКА

Конспект лекций
для студентов специальности
1-53 01 05 «**Автоматизированные электроприводы**»

В 2-х частях

Часть 2-я

Основы программирования

РАЗДЕЛ I. ОСНОВЫ АЛГОРИТМИЗАЦИИ

Тема 1. Методика программирования и решения инженерных задач на ЭВМ

Методику решение инженерных задач любой сложности можно представить в виде последовательности шести этапов.

На первом этапе условие задачи записывается в виде последовательности формул или уравнений, необходимых для решения задачи.

На втором этапе выбирается такой метод решения задачи, который сведет поиск результата к выполнению последовательности элементарных математических операций (сложение, вычитание, умножение, деление). Для большинства практических задач разработаны различные математические методы решения дифференциальных уравнений, например, Эйлера, Рунге-Кутта и т.д. Выбор того или иного метода осуществляется на некоторых критериях, например, минимальное время вычислений, заданная точность вычислений, интегральные или квадратичные критерии. Более подробное изучение методов и критериев оценки будет на старших курсах обучения.

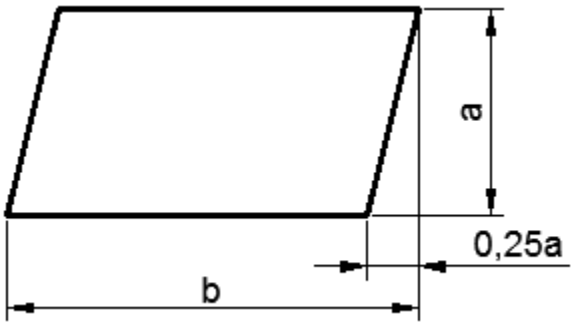
На третьем этапе на основании выбранного метода разрабатывается алгоритм – общая схема решения задачи. В настоящее время существует несколько определений алгоритма, одним из них является следующее. Алгоритмом называется предписание, определяющее содержание и последовательность операций, преобразующих исходные данные в искомый результат. В инженерной деятельности широкое распространение получил схемный способ описания алгоритма, при котором алгоритм представляется в виде блоков, каждый из которых выполняет определенное действие, и направленных связей между ними. Часто схема алгоритма называется еще блок-схемой. Размеры блоков, их форма и назначение определяются по ГОСТ19.701-90.

На четвертом этапе, на основании блок-схемы алгоритма, составляется программа решения задачи на определенном алгоритмическом языке, в нашем случае Turbo Pascal 7.0.

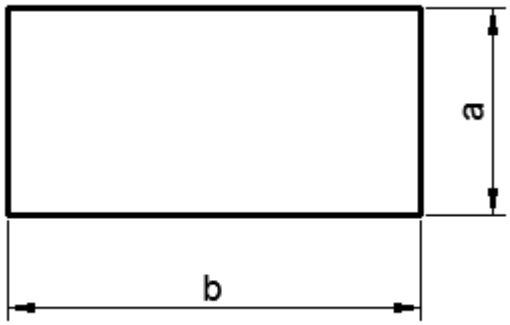
На пятом этапе вводят программу в память ЭВМ, транслируют, редактируют, проверяют правильность ввода и написания.

На шестом этапе вводят исходные данные, выполняем вычисления по программе и получаем результат.

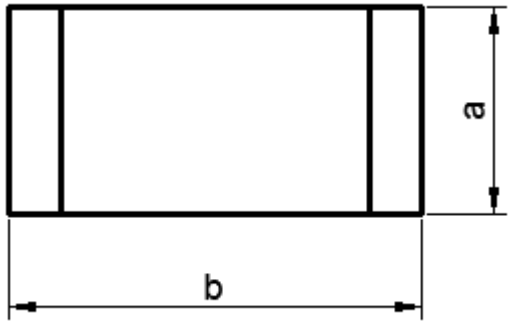
Приведем условные обозначения чаще используемых символов (блоков) и некоторые правила выполнения схем алгоритмов из ГОСТ 19.701-90.



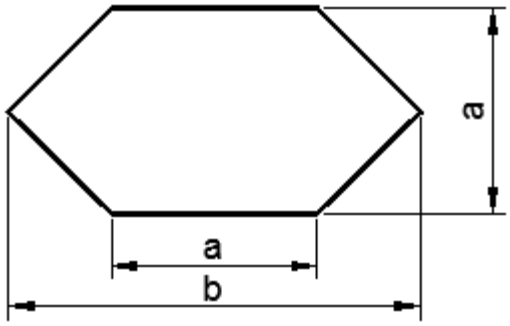
1. Данные. Символ отображает данные, носитель данных не определен (символы данных во многих случаях представляют способы ввода/вывода данных).



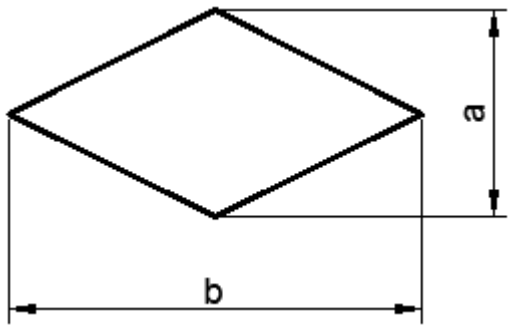
2. Процесс. Символ отображает обработку данных (вычисления по формулам).



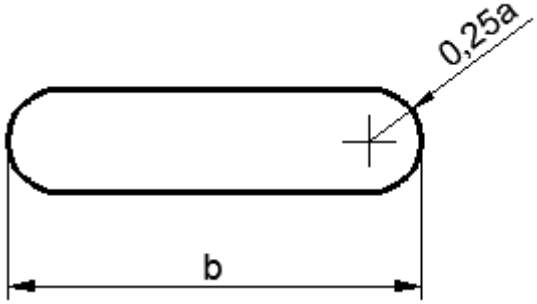
3. Предопределенный процесс. Символ отображает подпрограмму, модуль.



4. Подготовка. Символ отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию (начало цикла с заданным числом повторения).



5. Решение. Символ отображает выбор направлений выполнения алгоритма в зависимости от некоторых условий.



6. Терминатор. Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы), прерывание (остановка, пуск).

Блоки в схеме алгоритма должны быть расположены равномерно, быть по возможности одного размера; не должны изменяться углы и другие параметры, влияющие на соответствующую форму символов (размер *a* выбирается из ряда 10, 15, 20, ... мм, а размер *b* = 1,5*a* или 2*a* согласно ГОСТ 19.003-80).

Тема 2. Структура Pascal-программы

Программой называется последовательность операторов и других элементов языка, построенных в соответствии с определенными правилами и предназначенных для решения определенной задачи.
Структура программы на языке Паскаль имеет следующий вид:

{*Директивы компилятору*}

Program *Name*;

Раздел описаний;

Begin

Раздел операторов;
(Тело программы)

end.

Для обеспечения независимости разрабатываемой программы от конкретных настроек компилятора, можно использовать набор директив компилятору:
{$B+, $N+, … }
Например: $B+ - установка полной схемы вычислений логических выражений («+» - включение, «-» - выключение); $D+ - создание отладочной информации в процессе компиляции; $I+ - автоматический контроль правильности операций ввода-вывода; $S+ - автоматический контроль переполнения стека; $N+ - подключение математического сопроцессора для работы с вещественными переменными.

Имя программы *Name* должно начинаться с буквы латинского алфавита, не должно содержать более 80 символов, не должно содержать букв кириллицы.
Раздел описания программы содержит описания модулей (uses), меток (label), констант (const), переменных (var), типов (type), функций (function) и процедур (procedure).
Переменная – это ячейка (или несколько) оперативной памяти компьютера. Содержимое ее может изменяться в ходе выполнения программы. Вид информации в ячейке, операции над значением которой и множество допустимых значений определяются типом переменной.
Раздел описания переменный содержит предложения описания переменных, которое имеет следующий вид:
Var v1, v2, … , vn : type_of_var;
v1, v2, … , vn – список переменных, в котором имена переменных разделяются запятыми.
type_of_var – задает тип переменным данного списка.
Если в программе используются переменные различных типов, то в предложении var приводятся списки переменных каждого типа:

```
Var   v11, v12, … , v1n :   type_1;
      v21, v22, … , v2n :   type_2;
```

Например:

```
Var   a, b, c: intrger; (целочисленный тип)
      d, e, f: real; (вещественный тип)
      g: char; (символьный тип)
      h: string; (строковый тип)
      k: boolean; (логический тип)
```

Константа отличается от переменной тем, что ее значение фиксированно и не может быть изменено в ходе выполнения программы.
Описание констант производится в следующей форме:

```
const v1 = val_1;
      v2 = val_2;
```

где v1, v2 – имена констант;
val_1, val_2 – значения этих констант.
Например:

```
const d = 3.4e-5;
      b = 188;
```

В теле программы располагаются операторы, выполняющие определенные действия и разделяемые точкой с запятой. Операторы определяют действия, которые должна выполнить программа. Операторы в программе могут размещаться как на отдельных строках, так и по несколько в строке. Одним из основных операторов является оператор присваивания «:=», в правой части которого записывается вычисляемое выражение, а в левой переменная, которой будет присвоен результат вычислений выражения. Например:

```
y:=5*a+3*sin(2*a-1);
x:=(3-a)/(2*b+1);
```

Составной оператор имеет формат:

```
begin
    s1;
……
    sn;
end;
```

Составной оператор трактуется как один и используется в тех случаях, когда согласно формальным правилам языка разрешается использовать лишь один оператор, а в действительности требуется несколько.

Тема 3. Форматный ввод-вывод данных

При работе с любой программой необходимо осуществлять ввод и вывод данных. В качестве стандартных устройств с которых производится ввод-вывод данных являются клавиатура и экран монитора. Для ввода данных с клавиатуры применяются процедуры Read и Readln в качестве параметров которых указываются имена переменных через запятую, значения которых необходимо ввести.

Например:

```
Read (a,b,b,d);
```

В данном случае будут вводится 4 переменные a, b, c, d, при этом после ввода значения последней курсор останется на той же строке. При вводе численные значения переменных разделяются пробелом.

```
Readln(a1,b1,c1,d1);
```

В данном случае будут вводится 4 переменные a1, b1, c1, d1, при этом после ввода значения последней курсор перейдет на следующую строку.

Вывод информации на экран монитора осуществляется с помощью операторов Write и Writeln. Параметрами данных процедур могут быть текстовые сообщения (строки заключенные в кавычки), и/или имена переменных разделенные запятыми, содержимое которых выводится на экран монитора.

Например если переменная I целого типа, а R вещественного типа, тогда:

```
I:=10;
Write('I=',I);
```

В данном случае на экране будет отображаться

```
I=10
```

При этом после вывода последнего значения курсор останется на той же строке.

```
R:=15.2;
Write('R=',R,'km');
```

В данном случае на экране будет отображаться:

```
R=_1.5200000000E+01km
```

При этом после вывода последнего значения курсор перейдет на следующую строку.

Как видно из примеров формат вывода переменных целого и вещественного типов отличается. В случае вывода переменной вещественного типа используется вывод с плавающей запятой и под содержимое переменной отводится 17 позиций, включая знак числа и степени, разделительную точку и символ десятичной степени E.

Пользователь может задать свой формат вывода данных на экран. Для этого используются параметры в процедурах вывода в следующем виде:

```
E
E:m
E:m:n
```

где E – переменная, значение которой выводится на экран.

m, n – выражения типа integer, необязательные параметры, указывающие соответственно ширину выводимого поля и количество дробных цифр.

Конструкция вида E:m:n может использоваться только для вещественных чисел. Для остальных типов употребляется конструкция вида E:m.

Если выводимое данное имеет меньше знаков, чем m то оно дополняется слева пробелами. Если больше, то выводится столько знаков, сколько необходимо для корректного представления результата.

Если параметры m и n опущены, то подразумевается их некоторые, зависящие от реализации, значения.

Если для данных вещественного типа отсутствует параметр n, то выводимое данное представляется с плавающей запятой и показателем степени. В противном случае при выводе используется представление числа с фиксированной точкой, причем после точке запишется n цифр (общая длина поля – m символов).

Рассмотрим несколько примеров.

```
I:=135;
write(I:5);
_ _ 1 3 5
R:=-19.274;
write(R:10);
-1.927E+01
write(R:10:2);
_ _ _ _ -19.27
write(R:10:0);
_ _ _ _ _ _ _ -19
```

Тема 4. Линейные алгоритмы и их программирование.

Линейный алгоритм – это такой алгоритм, в котором все блоки выполняются последовательно один за другим в естественном порядке. При решении задачи следует выбирать наилучший вариант, в смысле некоторого критерия (точность, затраты времени, объем программы, интегральный критерий).

Рассмотрим линейный алгоритм и его программирование на классическом примере.

Задача. Необходимо разработать линейный алгоритм и программу для вычисления высот ha, hb, hc треугольника ABC с заданными длинами его сторон a, b, c.

Нахождение высот можно производить исходя из площади треугольника на основании выражения:

$$S = \frac{1}{2}a \cdot h_a = \frac{1}{2}b \cdot h_b \frac{1}{2}c \cdot h_c.$$

Вычисление площади треугольника будем производить по формуле Герона:

$$S = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}.$$

где p – полупериметр:

$$p = \frac{1}{2}(a+b+c).$$

Для решения данной задачи разработаем алгоритм, который показан на рисунке 4.1.

По алгоритму разработаем Паскаль-программу, которая имеет следующий вид:

```
Program Primer;
var a,b,c: integer;
    p,s,ha,hb,hc: real;
begin
writeln('Введите исходные данные:');
readln(a,b,c);
writeln('a=',a);
writeln('b=',b);
writeln('c=',c);
p:=0.5*(a+b+c);
s:=2*sqrt(p*(p-a)*(p-b)*(p-c));
ha:=s/a;
hb:=s/b;
hc:=s/c;
writeln('ha=',ha);
writeln('hb=',hb);
writeln('hc=',hc);
readln;
end.
```

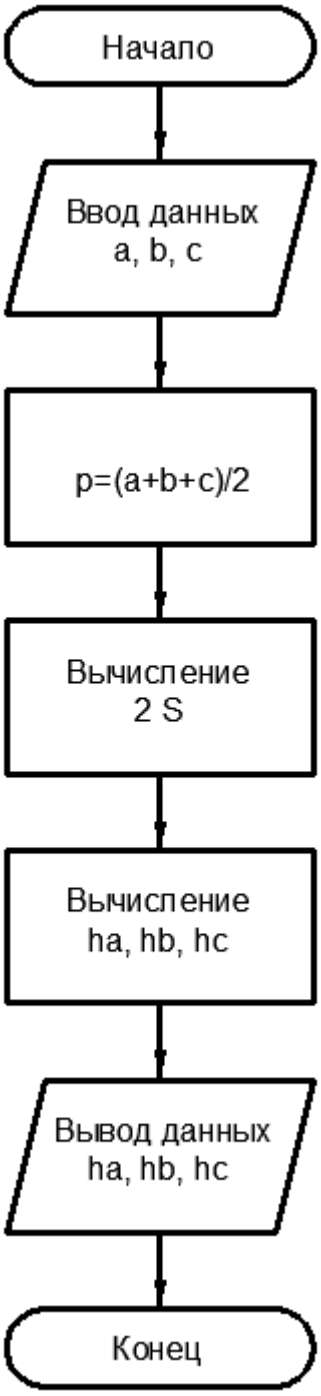


Рисунок 1.1. – Блок-схема линейного алгоритма решения задачи

Тема 5. Разветвляющиеся алгоритмы и их программирование.

Разветвляющимся алгоритмом называется такой алгоритм, в котором вычислительный процесс идет по нескольким направлениям в зависимости от выполнения некоторых логических условий.

Рассмотрим разветвляющийся алгоритм и его программирование на примере.

Задача. Необходимо вычислить корни квадратного уравнения $a x^2 + b x + c = 0$ с вещественными коэффициентами a, b, c.

Как известно, вначале необходимо вычислить дискриминант квадратного уравнения:

$D = b^2 - 4ac.$

В зависимости от значения дискриминанта имеем три ветви алгоритма:

1) если D>0, то корни уравнения вещественные и определяются по выражению:

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}.$$

2) если D=0, корни вещественные и совпадают:

$$x_{1,2} = -\frac{b}{2a}.$$

3) если D<0, то корни комплексно-сопряженные:

$x_{1,2} = x_1 \pm jx_2.$

$$x_1 = -\frac{b}{2a}.$$

$$x_2 = \frac{\sqrt{D}}{2a}.$$

Для определения номера варианта решения введем переменную k.

Разработаем алгоритм решения задачи, который показан на рисунке 1.2.

Для программирования разветвляющихся алгоритмов в TurboPascal используется так называемый структурный оператор. Структурный оператор – это оператор, состоящий из специальных зарезервированных слов, логического выражения и других операторов. Такие операторы явно или косвенно содержат логические выражения.

Самым распространенным для программирования разветвляющихся алгоритмов является оператор if, формат которого имеет следующий вид:

- сокращенная форма:

if <exp> then <st>;

где exp – логическое выражение, оно может принимать одно из двух значений *true* или *false*. В качестве логического выражения может быть условие (операции отношения), которое может либо выполняться, либо не выполняться.

st – оператор, который будет выполняться если условие exp принимает значение истина, в противном случае будет выполняться следующий за if оператор.

- полная форма:

if <exp> then <st1> else <st2>;

где exp – логическое выражение.

st1 – оператор, который будет выполняться если условие exp принимает значение истина.

st2 – оператор, который будет выполняться если условие exp принимает значение ложь.

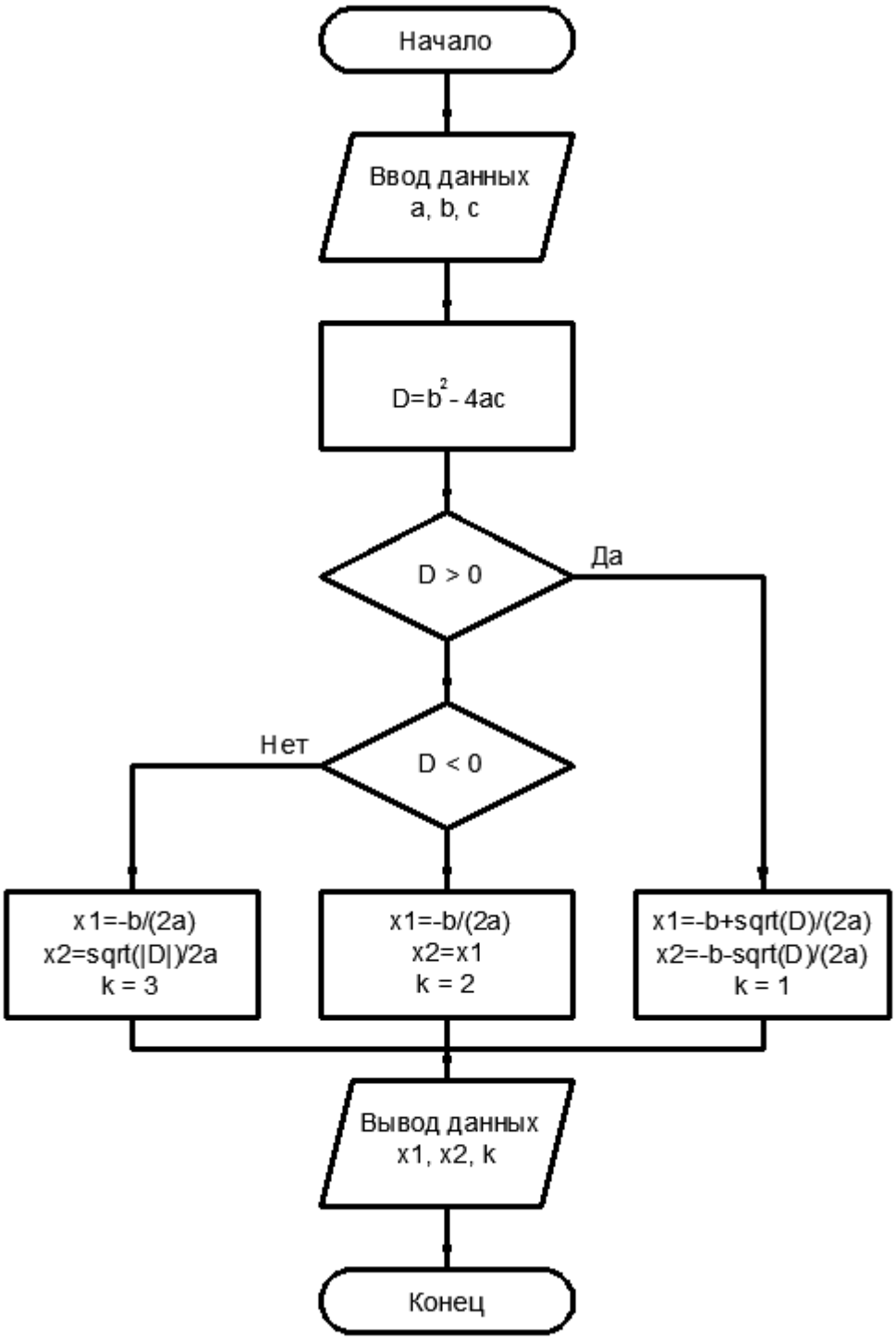
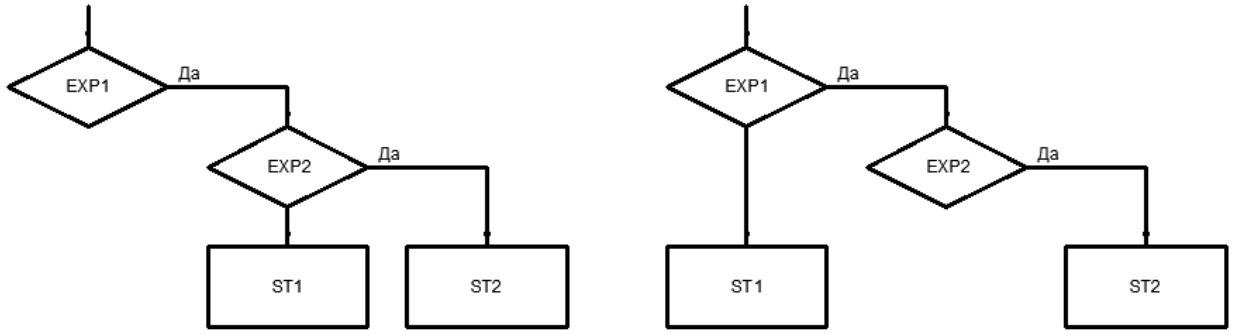


Рисунок 1.2. – Разветвляющийся алгоритм решения задачи.

По правилам языка после слова then (else) должен идти один оператор. Если необходимо несколько, то используется составной оператор:

```
if <exp> then
   begin
   <st1>;
   .....
   <stn>;
```

```
   end;
```
структурные операторы можно вкладывать друг в друга:
```
if <exp> then
    if <exp> then
        if <exp> then
            if <exp> then <st>;
```
При использовании во вложении полную форму оператора if следует помнить, что каждому else соответствует ***ближайший*** if.
Рассмотрим два варианта алгоритмов показанных на рисунке 1.3.



а) б)

Рисунок 1.3. – Вложенный алгоритм

Рисунку 3а будет соответствовать следующая запись
```
if <exp1> then begin
                if <exp2> then <st2> else <st1>;
                end;
```
Рисунку 3б будет соответствовать следующая запись
```
if <exp1> then begin
                if <exp2> then <st2>;
                end
else <st1>;
```
Применительно к нашей задаче программа будет иметь вид:
```
Program Primer;
var a,b,c,D,x1,x2:real
    k:byte;
begin
writeln('Введите исходные данные:');
readln(a,b,c);
D:=sqr(b)-4*a*c;
if D>0 then
  begin
  x1:=(-b+sqrt(D))/(2*a);
  x2:=(-b-sqrt(D))/(2*a);
  k:=1;
  end
```

```
else
  if D<0 then
    begin
    x1:=-b/(2*a);
    x2:=sqrt(abs(D))/(2*a);
    k:=3;
    end
  else
    begin
    x1:=-b/(2*a);
    x2:=x1;
    k:=2;
    end;
writeln('x1=',x1,'x2=',x2,'k=',k);
end.
```

Тема 6. Разветвляющиеся структуры с селектором

Разновидностью разветвляющихся алгоритмов являются структуры в которых имеется три и более альтернативы. В этом случае определение направления выполнения осуществляется селектором. Пример такой структуры показан на рисунке 1.4.

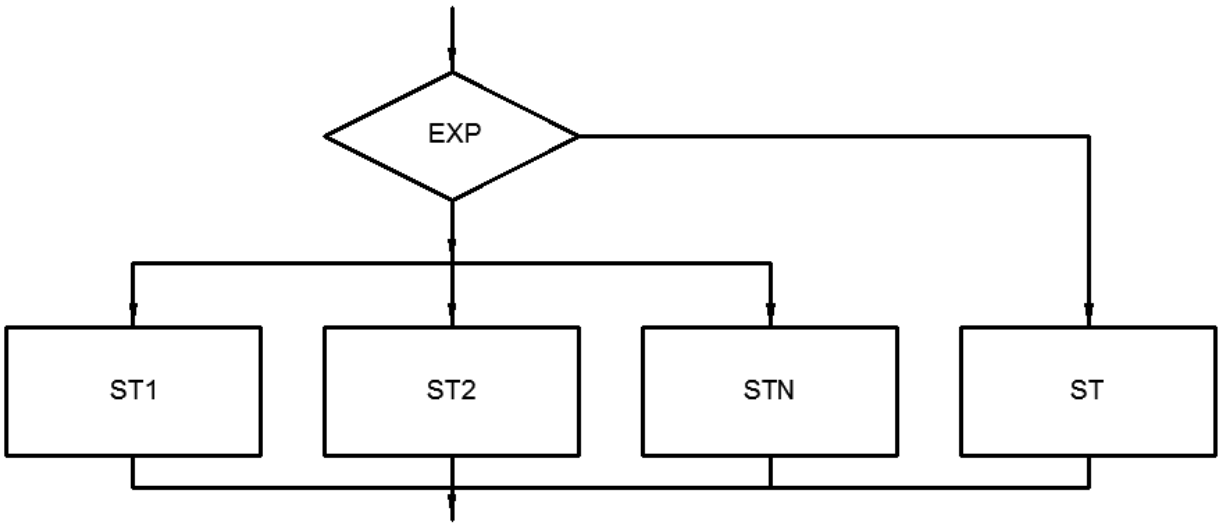


Рисунок 1.4. – Разветвляющаяся структура с селектором

Программирование структур с селектором производится с помощью структурного оператора CASE, который имеет следующий формат:

```
Case <exp> of
<val1>: <st1>;
<val2>: <st2>;
.....
```

<valn>: <stn>
else <st>;
end;
где exp – селектор, принимает одно из значений val1 .. valn.
При выполнении данного оператора вначале вычисляется значение селектора exp, а затем выбирается тот список значений, которому принадлежит полученное значение и выполняется соответствующий оператор.
Ветвь else ... работает в том случае, если ни одно из условий не выполняется. Данная ветвь является не обязательной. Если она отсутствует и условие не выполняется, то весь оператор рассматривается как пустой.
Пример. Необходимо отсортировать случайные числа по их делимости на число 3. В данном примере рассмотрим ряд возможностей TurboPascal.
В турбо Паскале имеется встроенный генератор случайных чисел с так называемым нормальным распределением. Для этого необходимо сначала вызвать процедуру инициализации датчика случайных чисел Randomize и лишь затем вызвать функцию генерации случайных чисел Random (n).
Если функция random вызывается без параметров, то генерируется случайное число в интервале от 0 до 1.
Если в место n указано целое положительное число, то результат случайное целое число типа word в диапазоне от 0 до n-1.
b:=random+0.1; b=[0;1]+0,1
k:=random(10)*2; k=[0;9]*2
Определить делимость числа можно с помощью операций целочисленного деления:
a div b – возвращается целая часть от результата деления;
10 div 3 => 2.
a mod b – возвращается остаток от результата деления;
10 mod 3 => 1.
Возвращаясь к нашему примеру часть программы с оператором case будет иметь вид:

```
Randomize;
N:=random(100);
case N mod 3 of
0: m:=m+1;
1: n:=n+1;
2: k:=k+1;
end;
```

Тема 7. Циклические алгоритмы.

Циклическим называется вычислительный процесс, в котором решение задачи или ее части сводится к многократному вычислению по одним и тем же формулам при различных значениях входящих в них некоторых переменных.

Многократно повторяющиеся участки такого вычислительного процесса называются циклами.

Различают циклы с заданным (известным) и неизвестным числом повторений. Первые так же называют арифметическими циклами. Ко вторым относятся так называемые итерационные циклы, характеризующиеся последовательным приближением к искомому значению результата с заданной точностью. Например, при вычислении суммы членов сходящегося бесконечного ряда чисел.

Переменную, изменяющуюся в цикле, называют параметром или переменной цикла.

7.1 Циклы с неизвестным числом повторений. Оператор цикла с предусловием

Структура алгоритма цикла с предусловием имеет вид как показано на рисунке 1.5.

В такой структуре вначале вычисляется значение логического выражения «exp». Если его значение «истина» (TRUE), то выполняется тело цикла. Как только логическое выражение принимает значение «ЛОЖЬ» (FALSE) происходит выход из цикла. Если выражение «exp» принимает значение «ЛОЖЬ» при первой же проверке, то оператор «st» не выполняется вообще.

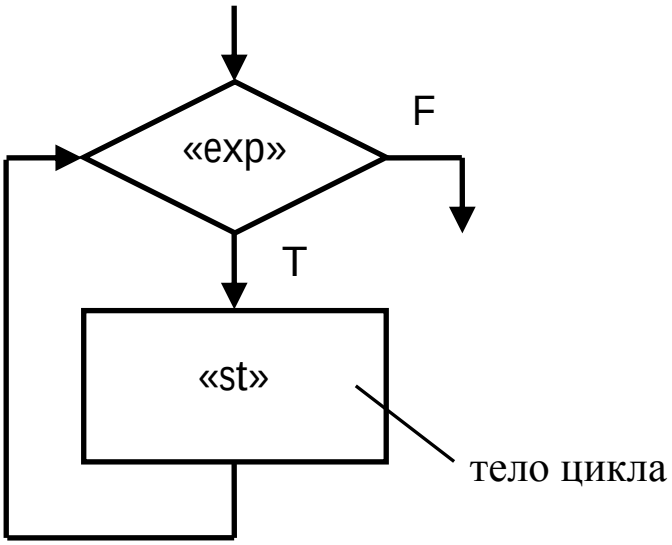


Рисунок 1.5. – Структура цикла с предусловием

Программируется цикл с предусловием с помощью структурного оператора:
WHILE <exp> do <st>;
где <exp> – логическое условие:
<st> – оператор.

По правилам языка оператор <st> - одиночный. Если необходимо использовать несколько операторов в теле цикла, то нужно использовать составной оператор (операторные скобки begin…end).

Отметим частный случай:

While TRUE do <st>;

Здесь оператор <st> будет выполняться бесконечно.

Рассмотрим пример: необходимо составить алгоритм и программу по вычислению произведения четных чисел от нуля до N, где N – вводится с клавиатуры.

Разработанный алгоритм имеет вид, как показано на рисунке 1.6.

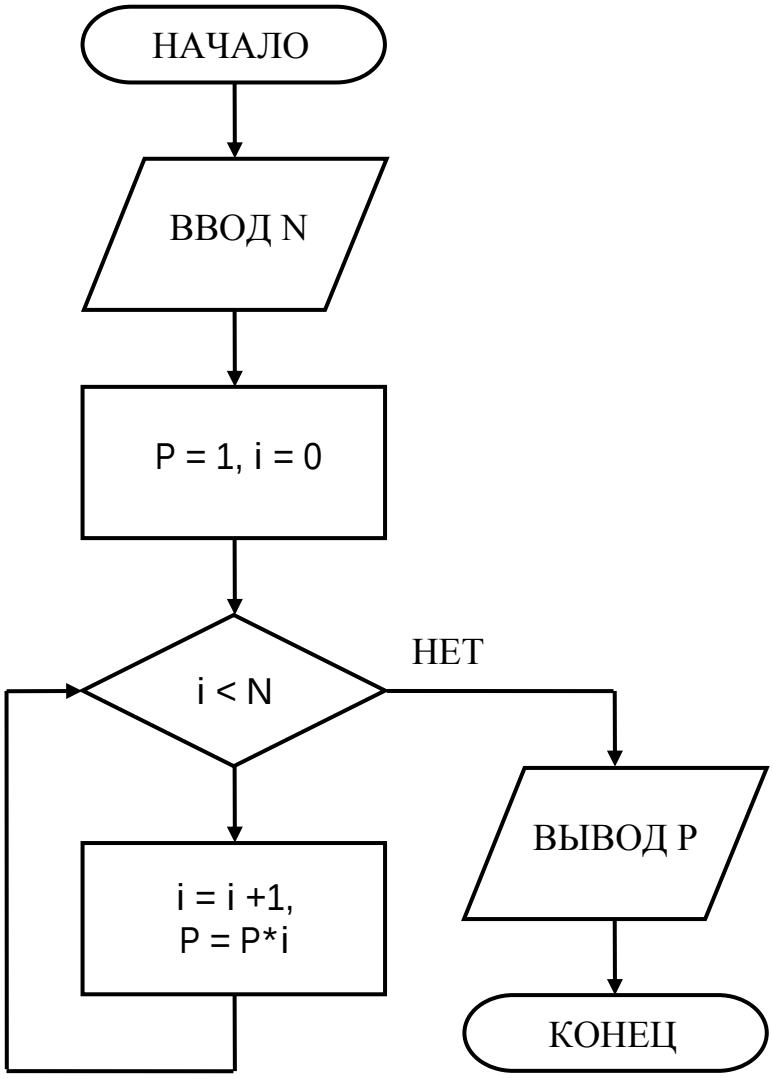


Рисунок 1.6. – Алгоритм решения задачи

Программа для нашего примера:

```
Program PR;
var i, p, N: integer;
begin
writeln (‘Vvedite chislo N’);
```

```
wrire (‘N=’);
read (N);
p:=1; i:=0;
while i < N do
        begin
        i:=i+2;
        p:=p*i;
        end;
writeln (‘p=’, p);
end.
```

7.2 Циклы с неизвестным числом повторений Оператор цикла с постусловием

Структура алгоритма цикла с постусловием имеет вид как показано на рисунке 1.7.

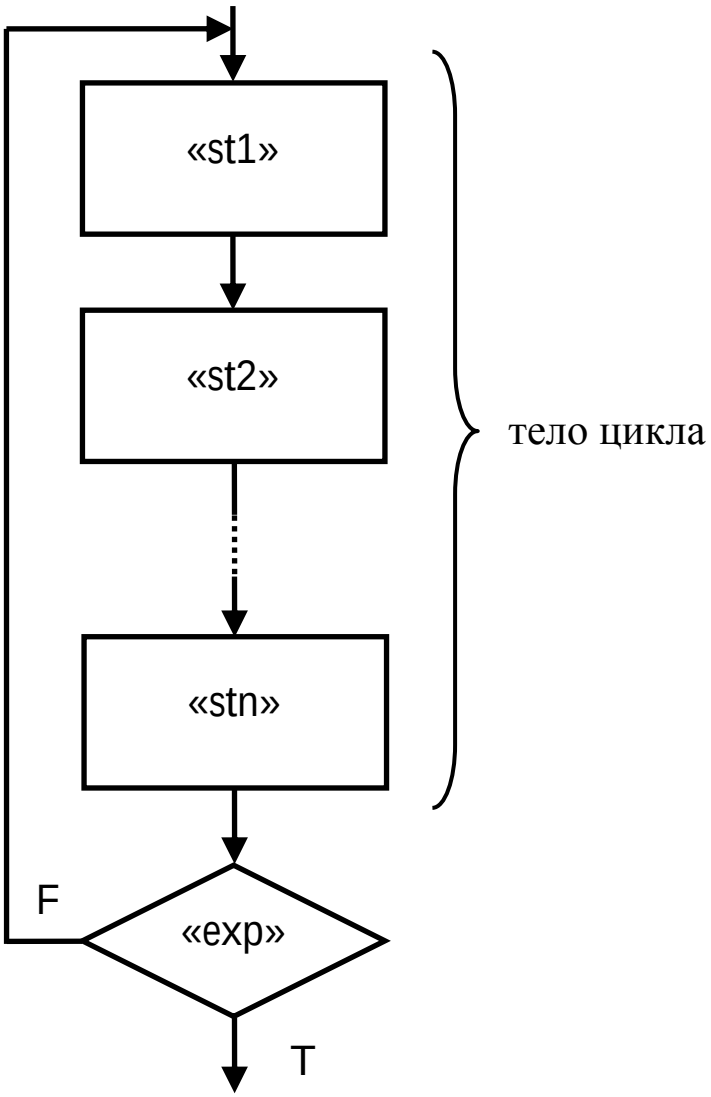


Рисунок 1.7. – Структура цикла с постусловием

В такой структуре вначале выполняются операторы st1, st2,…, stn, а затем производится вычисление и проверка условия логического выражения «exp». Выполнение тела цикла будет происходить до тех пор, пока условие <exp> не примет значение «истина» (TRUE).

Программируется цикл с постусловием с помощью структурного оператора:

```
REPEAT
  <st1>;
  <st2>;
… … …
  <stn>;
UNTIL <exp>;
```

где <exp> – логическое условие:

<sti> – операторы.

Отметим особенности структурного оператора Repeat:

1. Тело цикла выполняется, по крайней мере, один раз:
2. Тело цикла выполняется до тех пор, пока значение <exp> принимает значение «ЛОЖЬ»;
3. В теле цикла может находиться несколько операторов, при этом операторные скобки begin…end не требуются.

Отметим частный случай:

```
Repeat
<st>;
Until False;
```

Здесь оператор <st> будет выполняться бесконечно.

Для примера возьмем ту же задачу, что и для цикла предусловием.

Разработанный алгоритм будет иметь как показано на рисунке 1.8.

Программа для нашего примера:

```
Program PR;
var i, p, N: integer;
begin
writeln (‘Vvedite chislo N’);
wrire (‘N=’);
read (N);
p:=1; i:=0;
repeat
        i:=i+2;
        p:=p*i;
until i > N;
writeln (‘p=’, p);
end.
```

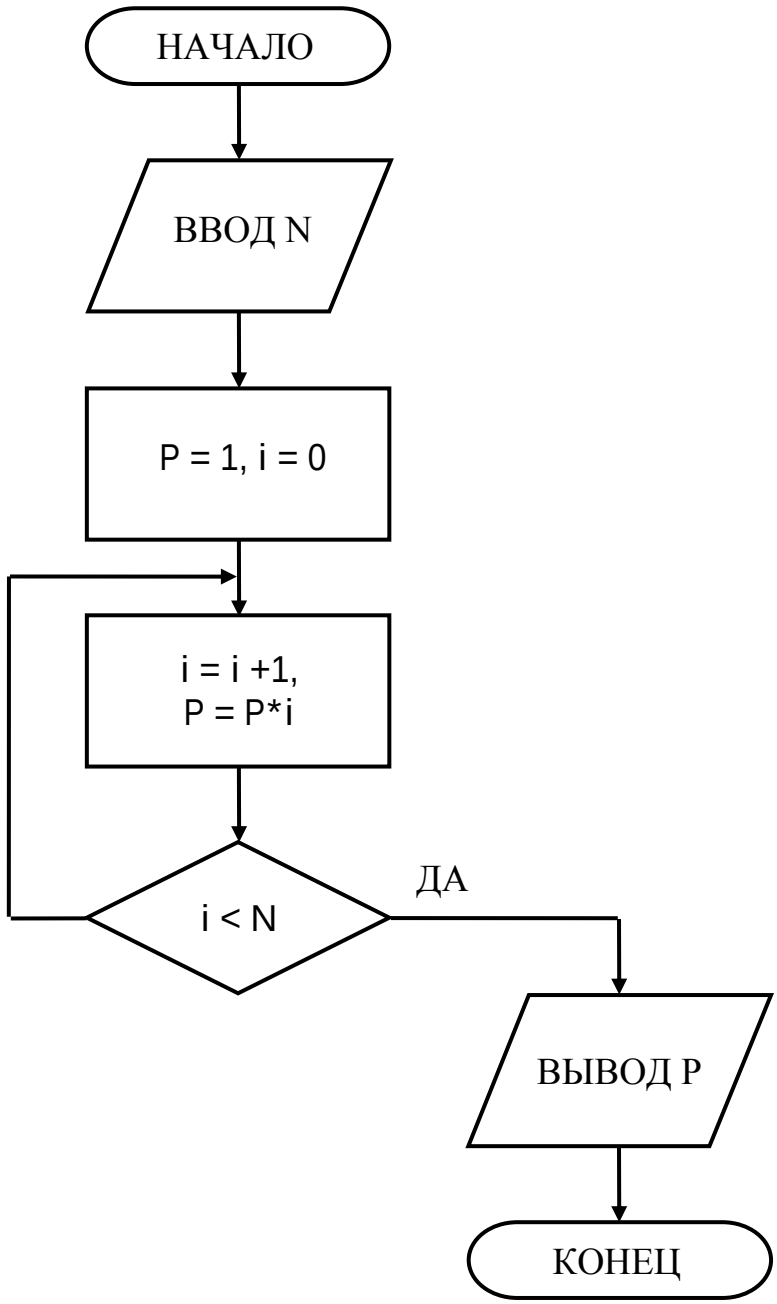


Рисунок 1.8 – Алгоритм решения задачи

7.3 Циклы с известным числом повторений

Структура алгоритма цикла с известным числом повторов имеет вид как показано на рисунке 1.9.

Здесь i – управляющая переменная цикла является произвольным идентификатором, который объявляется как переменная любого скалярного типа (целый, символьный, булевский, перечисляемый).

P – оператор (тело цикла).

B1 и B2 – выражение (в общем случае).

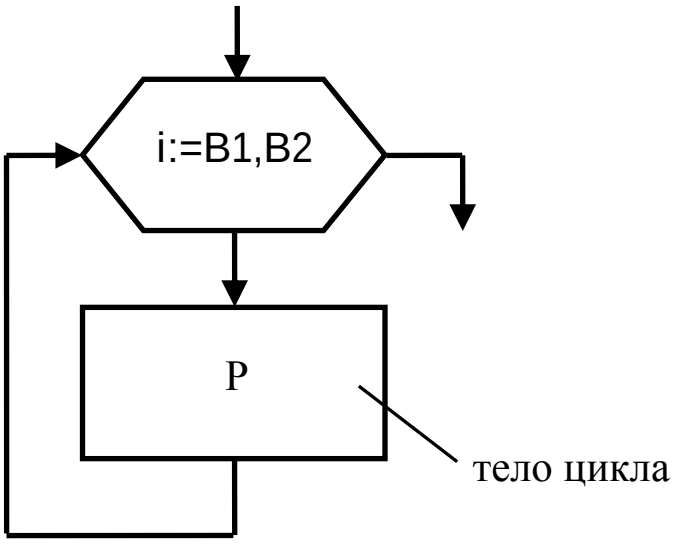


Рисунок 1.9 – Структура цикла с известным числом повторений

В TurboPascal программирование цикла со счетчиком осуществляется с помощью структурных операторов:

FOR … TO … DO … ;

или

FOR … DOWNTO … DO … ;

Первый из этих операторов имеет формат:

FOR i:= <B1> TO <B2> DO <P> ;

Второй:

FOR i:=<B1> DOWNTO <B2> DO <P> ;

Оператор FOR работает следующим образом: сначала вычисляется значение выражения <B1>, затем вычисляется значение выражения <B2>. Далее управляющая переменная i последовательно пробегает все значения от <B1> до <B2>. Если B1 > B2, то цикл не будет выполняться вовсе. Значения <B1> и <B2> остаются неизменными в ходе выполнения всего цикла.

В теле цикла FOR следует избегать операторов изменяющих значение управляющей переменной i. Это не приведет к ошибкам компиляции, но из-за этого могут быть ошибки вычислений.

Вариант FOR … DOWNTO … DO … ; цикла аналогичен вышеописанному, за исключением того, что в нем управляющая переменная на каждом шаге выполнения не увеличивается, а уменьшается на единицу. В этом случае должно быть B1 < B2.

Пример: необходимо вычислить сумму натуральных чисел от 1 до 30.

Разработанный алгоритм будет иметь вид как показано на рисунке 1.10.

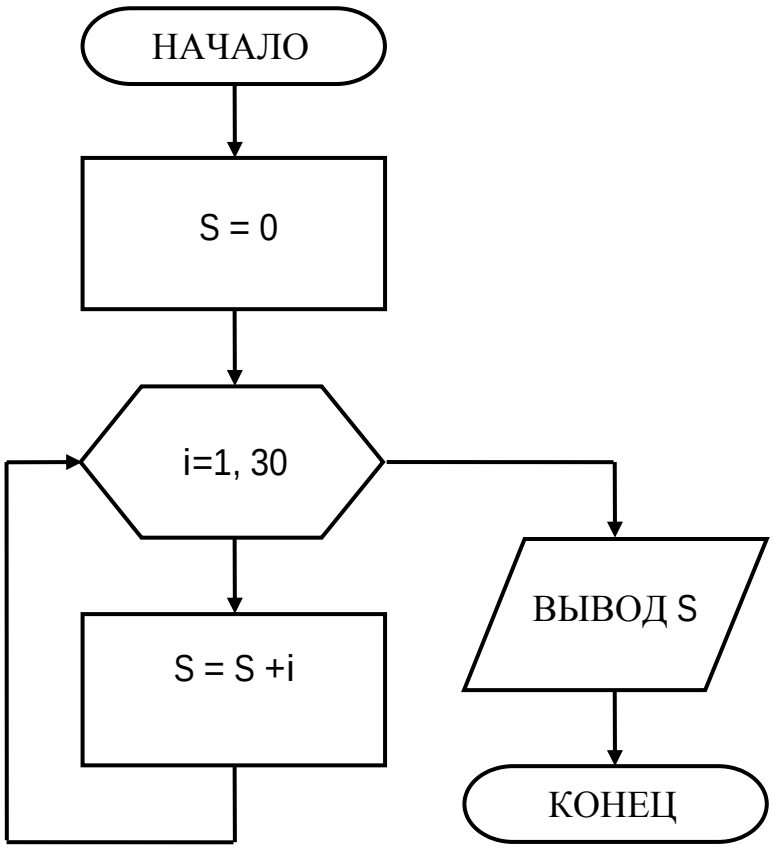


Рисунок 1.10 – Алгоритм решения задачи

Программа:

```
Program PR;
var i, s: integer;
begin
s:=0;
for i:=1 to 30 do s:=s+i;
writeln (‘s=’, s);
end.
```

РАЗДЕЛ II. ИСПОЛЬЗОВАНИЕ ПОДПРОГРАММ

В современном программировании важным моментом является принцип модульности. В модульной программе отдельные ее части, предназначенные для решения частных задач, организованы в подпрограммы. Преимущества такого принципа заключаются в том, что:

во-первых, один и тот же фрагмент может использоваться многократно;

во-вторых, такие программы легко читать, тестировать и отлаживать.

В TurboPascal модульность обеспечивается за счет применения:

а) подпрограмм-функций;

б) подпрограмм-процедур;

в) модулей.

Тема 1. Подпрограмма-функция

Описание подпрограмм-функций должно располагаться в разделе описаний программы. Описание функции начинается с зарезервированного слова Function. Заголовок подпрограммы-функции имеет вид:

Function <Name> (argument_list): type_of_result;

Здесь type_of_result – идентификатор типа результата, он описывает тип значения, носителем которого является идентификатор (имя) функции <Name>.

argument_list – список параметров. Содержит перечисление идентификаторов переменных-параметров функции. Эти параметры используются для передачи данных в подпрограмму-функцию. После имени переменной или группы переменных следует двоеточие и идентификатор типа переменных этой группы. Список параметров подпрограммы-функции может отсутствовать.

Внутренняя структура подпрограммы-функции аналогична структуре программы, т.е. она содержит раздел описаний, а затем после слова «begin» идут операторы (тело подпрограммы). В разделе описаний могут быть описания других функций.

В теле подпрограммы-функции должен присутствовать оператор присвоения, в левой части которого находится имя функции, а в правой – выражение.

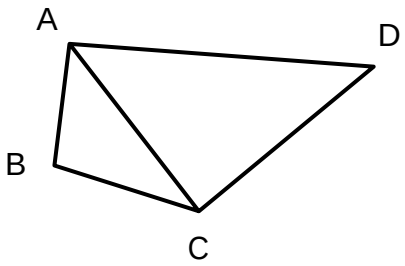
Обращение к подпрограмме-функции производится путем указания имени со списком параметров в составе какого-либо выражения. Имя функции в вызывающей программе может появиться только в правой части выражения, т.к. функция – это операнд.

Параметры представляют собой идентификаторы переменных и служат для обмена значениями между подпрограммой и вызывающей ее программной единицей. При этом в описании подпрограммы, поскольку она включается в текст

программы один раз, имена параметров выбираются определенным образом и безотносительно к именам переменных, используемых в различных частях программы. Такие параметры, имена которых указаны в заголовке подпрограммы, называются формальными.

При каждом новом обращении к подпрограмме в нее могут передаваться значения разных переменных. Такие параметры, имена которых подставляются в оператор вызова подпрограммы при фактическом обращении к ней, называются фактическими параметрами.

Пример: требуется составить программу вычисления площади выпуклого четырехугольника ABCD.



Данный четырехугольник можно разбить на два треугольника ABC и ACD. Таким образом, суть программы – вычисление площадей треугольников ABC и ACD и суммирование их.

Вычисление площадей треугольников выполним с помощью подпрограммы. Площадь будем вычислять по формуле Герона:

$S = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$.

Разработанный алгоритм будет иметь вид как показано на рисунке 2.1.

Программа:

```
Program PR;
Function STR (a,b,c: integer): real;
var p,s: real;
begin
p:=(a+b+c)/2;
s:=sqrt(p*(p-a)*(p-b)*(p-c));
STR:=s;
end;
var AB, BC, CD, DA, AC: integer;
Sabc, Sacd, Sabcd: real;
begin
writeln (‘Vvedite AB, BC, CD, DA, AC’);
readln (AB, BC, CD, DA, AC);
Sabc:=STR (AB, BC, AC);
Sacd:=STR (CD, DA, AC);
Sabcd:=Sabc+Sacd;
writeln (‘Sabcd=’,Sabcd);
end.
```

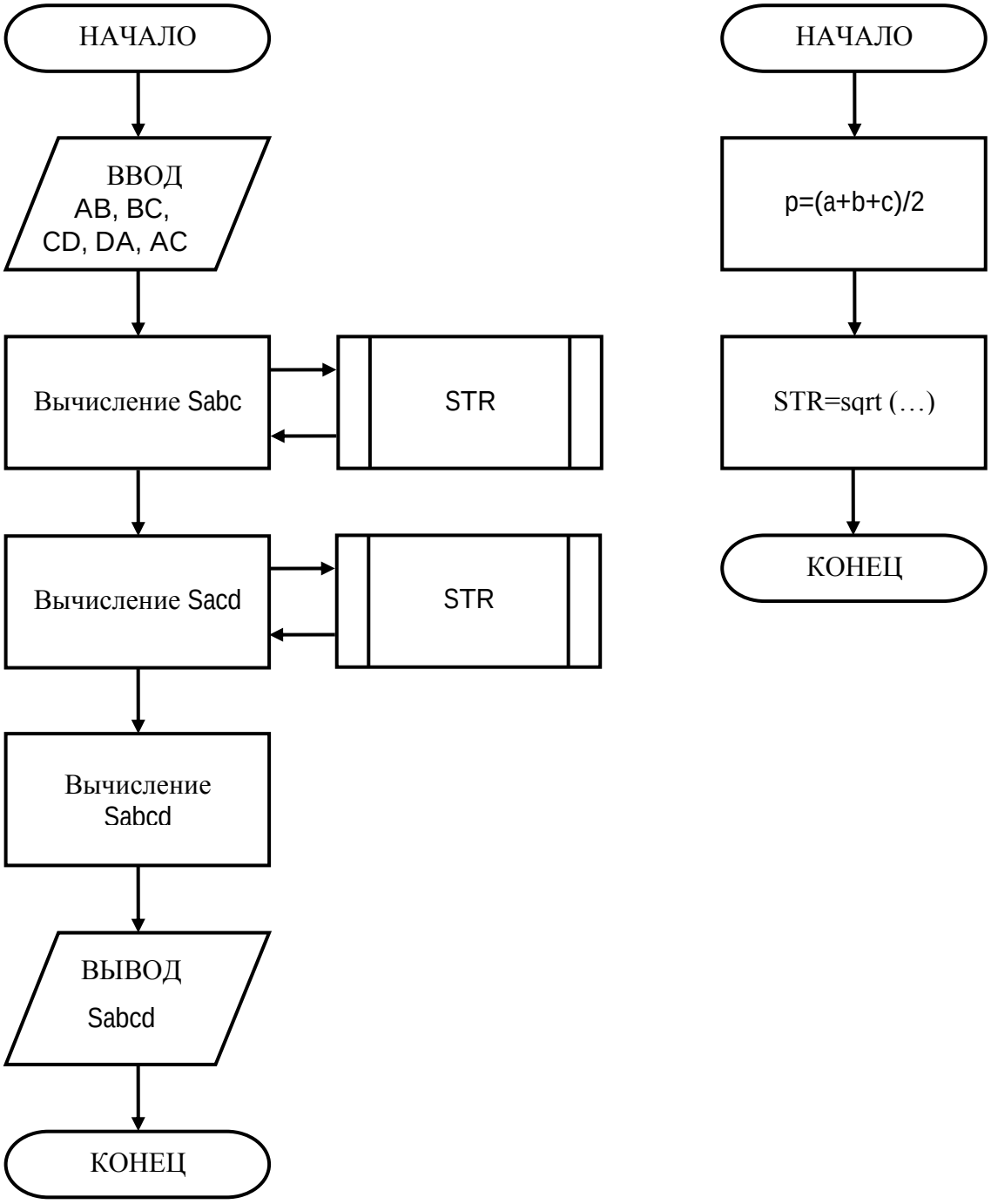


Рисунок 2.1. – Алгоритм решения задачи

Тема 2. Подпрограмма-процедура

Возвращаясь к подпрограмме–функции, можно отметить, что функция при вызове возвращает единственное значение, носителем которого является имя функции. Это не всегда отвечает нашим потребностям, т.к. зачастую результат работы подпрограммы – это набор значений, а иногда результат ее выполнения не сводится к вычислениям.

Подпрограммы-процедуры могут выполнять все виды действий, включая вычисление одного и более результатов.

Процедура, так же как и функция, - это фрагмент программы (или модуля), начинающийся заголовком Procedure, который имеет разделы описаний и операторов и заканчивающийся словом end. Заголовок процедуры имеет следующий вид:

```
Procedure <Name> (list_of_parametrs);
```

Здесь list_of_parametrs – список параметров процедуры.

Обмен данными между процедурой и вызывающей программной единицей производится с помощью параметров, имя процедуры не является носителем результата, поэтому тип процедуры в заголовке не описывается.

Вызов процедуры осуществляется указанием ее имени со списком фактических параметров, т.е. имя процедуры – это оператор.

Пример: Требуется составить программу вычисления площади выпуклого четырехугольника ABCD.

Алгоритм будет аналогичен предыдущему (рисунок 2.1). Программа будет иметь следующий вид:

```
Program ABCD1;
Procedure STR (var S: real; a, b, c: integer);
var p: real;
begin
p:=(a+b+c)/2;
S:=sqrt((p-a)*(p-b)*(p-c));
end;
var Sabc, Sacd, Sabcd: real;
AB, BC, CD, DA, AC: integer;
begin
writeln (‘Vvedite AB, BC, CD, DA, AC’);
readln (AB, BC, CD, DA, AC);
STR (Sabc, AB, BC, AC);
STR (Sacd, CD, DA, AC);
Sabcd:=Sabc+Sacd;
writeln (‘Sabcd=’,Sabcd);
end.
```

Тема 3. Область видимости идентификаторов

Как уже известно, функции и процедуры имеют собственные разделы описаний. Описанные в подпрограмме переменные являются локальными и действуют только внутри данной подпрограммы. Между ними и объектами вызывающей их программы, которые имеют такое же имя, нет никакой связи. Они полностью независимы.

В подпрограмме можно так же использовать идентификаторы, описанные только в вызывающей программе. Смысл и значения этих идентификаторов и там и там одинаков. Такие идентификаторы называют глобальными. Использование таких переменных следует избегать, т.к. подпрограмма в этом случае становится менее универсальной, а также возрастает риск ошибок, вызванных «несанкционированным» изменением глобальной переменной в теле подпрограммы.

Область действия описания конкретного идентификатора называется областью видимости.

В Турбо Паскале имеется три типа параметров:

1) Формальный параметр-значение – при вызове подпрограммы получает свое начальное значение путем копирования соответствующего ему фактического параметра. При изменении формального параметра-значения фактический параметр не меняется.

Фактическое значение параметра-значения должно быть выражением, а его тип должен быть совместим по присвоению с типом формального параметра-значения. В заголовке подпрограммы перед списком параметров-значений отсутствует ключевое слово var. Пример:

```
Function Pr1 (a, b, c: real): Boolean;
```

2) Параметр-переменная – используется в тех случаях, когда значение должно передаваться из подпрограммы вызывающей программе. Соответствующий параметр в операторе вызова подпрограммы должен быть ссылкой на переменную. При вызове подпрограммы формальный параметр-переменная замещается фактической переменной. Если изменяется формальный параметр, то изменяется и фактический параметр.

Перед списком параметров-переменных необходимо записать ключевое слово var. Пример:

```
Function Pr2 (var a, b, c: real): Boolean;
```

3) Нетипизированные параметры. Если формальный параметр является нетипизированным параметром-переменной, то соответствующий ему фактический параметр может представлять собой любую ссылку на переменную, независимо от ее типа. В заголовке подпрограммы группа параметров, перед которыми стоит ключевое слово var и за которыми не следует указателя типа, является списком нетипизированных параметров-переменных. Пример:

```
Function Pr1 ( var a, b, c): Boolean;
```

Тема 4. Модули

Библиотечный модуль содержит подпрограммы, которые могут использоваться в различных программах. Подпрограмму можно записать в модуль, если она используется часто, т.е. подпрограмму можно записать и отладить один раз, а затем многократно использовать.

Доступ к процедурам и функциям модуля обеспечивает оператор использования USES, в котором указывается имя модуля. Данный оператор указывается в разделе описаний.

Модуль хранится в файле с таким же именем, как и у модуля. Модуль – это программная единица, текст которой компилируется независимо, т.е. автономно. Он содержит определения констант, типов данных, переменных, процедур и функций, доступных для использования в вызывающей этот модуль программах.

Структура модуля:

1 Unit - заголовок;

2 Interfase - интерфейсная часть;

3 Implementation - реализационная часть;

4 Begin - инициализационная часть

end

В (2) описываются константы, типы данных, переменные, процедуры и функции, доступные в этом модуле при использовании внешними программами. Процедуры и функции, предназначенные для общего использования, описываются в интерфейсной части, написанием их заголовка с указанием типов параметров.

В (3) содержатся тексты процедур и функций. Описаны в (3) переменные, константы, типы недоступны из вне.

В (4) содержатся операторы, выполняемые первыми в основном блоке вызвавшей модуль программы.

Основные модули языка Турбо Паскаль:

System – основной модуль, содержащий основные процедуры и функции. Его не нужно описывать в разделе описаний.

<ClrScr – очистка экрана>.

Graph – модуль содержащий процедуры для работы в графическом режиме.

DOS – содержит процедуры, обеспечивающие возможность выполнения некоторых системных операций. Например, прямой доступ к различным устройствам, например к дискам.

Можно создать свой модуль, а затем его использовать.

Например: Создадим модуль, в котором будут храниться описания гиперболических функций:

$$sh(x)=\frac{e^x-e^{-x}}{2},\ ch(x)=\frac{e^x+e^{-x}}{2},\ th(x)=\frac{sh(x)}{ch(x)},$$

Модуль:

```
Unit hyper;
Interfase
  Function sh(x: real): real;
  Function ch(x: real): real;
```

```
  Function th (x; real): real;
Implementation
var t: real;
function sh (x: real): real;
begin
t:=exp(x);
sh:=0.5*(t-1/t);
end;
function ch (x: real): real;
begin
t:=exp(x);
ch:=0.5*(t+1/t);
end;
function th (x: real): real;
begin
t:=exp(2*x);
th:=(t-1)/(t+1);
end;
end.
```

После компиляции модуля hyper.pas будет создан файл hyper.tpu.

Пример использования нового модуля:

```
Program hyp;
Uses crt, hyper;
Var a: real;
Begin
Clrscr;
Write (‘a=’);
Readln (a);
Writeln (‘sh(’,a,’)=’,sh(a));
Writeln (‘ch(’,a,’)=’,ch(a));
Writeln (‘th(’,a,’)=’,th(a));
Readln;
End.
```

РАЗДЕЛ III. ТИПЫ ДАННЫХ

Любые данные, т.е. константы, переменные, значения функций или выражения, в Турбо Паскале характеризуются своими типами. Тип определяет множество допустимых значений, которые может иметь тот или иной объект, а также множество допустимых операций, которые применимы к нему. Кроме того, тип определяет также и формат внутреннего представления данных в памяти ПК. Турбо Паскаль характеризуется следующей структурой типов данных:

1 простой:
  1.1 порядковый:
    1.1.1 целые;
    1.1.2 символьный;
    1.1.3 логический;
  1.2 вещественный;
2 структурный:
  2.1 массив;
  2.2 множество;
  2.3 запись;
  2.4 файл;
3 строковый;
4 ссылочный;
5 процедурный:
  5.1 процедура;
  5.2 функция;
6 объектный.

Тема 1. Порядковые типы данных

Порядковые типы называются так потому, что их допустимые значения представляют собой множество, состоящее из конечного числа элементов. В этом множестве есть первый и последний элементы. Кроме того, каждый элемент порядкового типа имеет предшествующий ему и следующий за ним элементы (за исключением первого и последнего). Элементы порядкового типа можно пронумеровать, расположив их в определенном порядке, например по возрастанию.

Таблица 1.1. Порядковые типы данных

Название	Идентификатор	Длина	Диапазон значений
Короткий целый	Shortint	8 бит	-127…127
Целый	Integer	16 бит	-32768…32767

Длинный целый	Longint	32 бит	-2147483648 … 2147483647
Байтовый	Byte	8 бит	0…255
Слово	Word	16 бит	0…65535
Логический	Boolean	8 бит	False, True
Символьный	Char	8 бит	Символы кода ASCII

Функции для величин порядкового типа:
High(x) – получение максимального значения величины данного типа;
Low(x) – получение минимального значения величины данного типа;
Ord(x) – определение порядкового номера значения величины x;
Pred(x) – определение предыдущего значения величины x;
Succ(x) – определение последующего значения величины x.
Например:
Ord(false) = 0,
Ord(true) = 1,
Ord(A) = 65,
Ord(Z) = 90,
Ord(a) = 97,
Ord(z) = 122.

1.1 Символьный тип

Символьный тип данных обозначается зарезервированным словом Char. Допустимые значения данного типа принадлежат расширенному набору символов кода ASCII (американский стандартный код для обмена информацией). Каждому символу и некоторым управляющим инструкциям соответствует свой числовой код, принимающий значения от 0 до 127, т.е. в двоичном представлении код использует 7 разрядов. Например: «A» – 65 , «$» – 36, «?» – 63, «a» – 97 и т.д.
Таблица 1.2. Коды символов стандарта ASCII

Кодировка символов в соответствии со стандартом ASCII							
Код	Символ	Код	Символ	Код	Символ	Код	Символ
0	NUL	32	BL	64	®	96	'
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	АСК	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	k	107	k
12	FF	44	,	76	L	108	i

13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	0
Код	Символ	Код	Символ	Код	Символ	Код	Символ
16	DEL	48	0	80	p	112	P
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	V
23	ETB	55	7	87	w	119	w
24	CAN	56	8	88	X	120	X
25	EM	57	9	89	У	121	У
26	SUB	58	:	90	z	122	z
27	ESC	59	/	91	[	123	{
28	FS	60	<	92	\	124	I
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	—	127	n

Расширенная таблица ASCII использует двоичных разрядов и состоит из двух частей:
0-127 – стандартные символы;
128-255 – специальные символы и буквы национальных алфавитов (в том числе русского).
Символьная константа задается указанием символа между апострофами: 'A'.
Переменная типа Char хранит один символ. Если переменная S описана как переменная типа Char:
var S: Char;
то допустимы следующие операторы присваивания:
S:=’D’;
S:=’&’;
S:=’ ’;
Значение символьной переменной можно задать с помощью функции Chr. Функция Chr устанавливает соответствие между однобайтным целым значением кода и символом.
Обратной ей является функция Ord, которая возвращает код символа элемента:
Chr(58) = : , Ord (:) = 58.
Кроме того, имеется возможность задавать значения указанием непосредственно числового значения ASCII-кода. В этом случае перед числом, обозначающим код символа ASCII, поставить знак «#»:
S:=#97;
Символьные переменные можно сравнивать друг с другом посредством отношений <, >, <=, => и т.д. При сравнении символьных значений сравниваются их коды.

Код ASCII кроме обычных символов содержит специальные управляющие символы, которые . Они представляют собой команды, вывод которых на стандартное выводное устройство приводит к выполнению определенных действий. К управляющему символу можно обратиться по него ASCII-коду или Ctrl-последовательности.
Допустим переменная c: char, тогда записи:
```
c:=chr(13);
c:=#13;      (# – означает код символа)
c:=^M;       (Ctrl+M)
```
присваивают символьной переменной одно и то же значение (CR – возврат каретки).
#7 – звуковой сигнал динамика.
#8 – горизонтальная табуляция; при выводе на экран смещает курсор в позицию, кратную 8, плюс 1 (9, 17, 25 и т.д.).
#12 – прогон страницы.

1.2 Целые типы

Целыми типы со знаком являются: ShortInt, Integer, LongInt, и без знака – Byte и Word. При использовании процедур и функций с целочисленными параметрами следует руководствоваться «вложенностью» типов, т.е. везде, где может использоваться Word, допускается использование Byte (но не наоборот), в Longint «входит» Integer, который в свою очередь, включаетв себя Shortint. Однобайтовые типы ShortInt и Byte целесообразно использовать в больших массивах, если их элементы принимают значения из ограниченного диапазона (для экономии памяти). Например:
```
A: array [1..1000] of shortint;
B: array [1..1000] of integer;
```
Если мы сравним их размеры:
```
SizeOf (A) = 1000;
SizeOf (B) = 2000;
```
Основными бинарными операциями над целыми числами являются: сложение (+), вычитание (-), умножение (*), деление (/).
При действиях с целыми числами тип результата будет соответствовать типу операндов, а если операнды относятся к различным целым типам, – типу того операнда, который имеет максимальный диапазон значений. Возможное переполнение результата никак не контролируется, что может привести к ошибкам при вычислениях, например:
```
var
a : Integer;
x, y : Real;
begin
a := 32767; {Максимально возможное значение типа Integer}
x := a + 2; {Переполнение при вычислении этого выражения}
```

```
y := LongInt (a)+2; {Переполнения нет после приведения переменной к более мощному типу}
WriteLn (x:10:0, y:10:0);
end.
```

В результате прогона программы получим:

```
-32767      32769
```

Основные функции, выполняемые над целыми числами:

a div b – целочисленное деление, возвращает целую часть от деления:
8 div 5 = 1.

a mod b – целочисленное деление, возвращает остаток от деления:
8 mod 5 = 3.

(a – не отрицательное, b – положительное).

abc(x) – унарная операция (функция), возвращает модуль числа x.

sqr(x) – унарная операция (функция), возвращает квадрат числа.

inc(x) – операция инкрементирования (по умолчанию прибавление единицы).

dec(x) – операция декрементирования (по умолчанию вычитание единицы).

odd(x) – булева функция, которая возвращает значение TRUE, если аргумент нечетное значение.

Над целочисленными переменными также могут быть применены операции сдвига и логические операции.

Операции сдвига: shl (левый сдвиг) и shr (правый сдвиг).

В тех случаях, когда операции сдвига применяются к беззнаковым целым, дают тот же результат, что умножение (shl) и целочисленное деление (shr) на степень двойки. Например:

3 shl 3 => 24: $00000011_2 \rightarrow 00011000_2 = 24$.

10 shr 2 => 2: $00001010_2 \rightarrow 00000010_2 = 2$.

«Лишние» биты отбрасываются, а освободившиеся места заполняются нулями.

Применяя операции сдвига к целым со знаком, следует иметь в виду, что старший бит, является битом знака. Например, любой правый сдвиг присваивает этому биту нулевое значение, что означает установку знака «+». При левом сдвиге знак результата может оказаться произвольным.

Для операций сдвига применяются быстрые алгоритмы вычислений низкого уровня, поэтому их можно использовать для замены операций умножения или деления на степени двойки.

В Турбо Паскале существуют четыре логических операций, которые применимы к целым типам: and, xor, not, or.

Таблица 1.3. Таблица истинности логических операций

a	b	a and b	a or b	a xor b	not a	not b
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	1	0	0	0

Пример:
a := 24;
b := 18;
c := a or b;
24_{10} => 0 0 0 1 1 0 0 0 $_2$
18_{10} => 0 0 0 1 0 0 1 0 $_2$
0 0 0 1 1 0 1 0_2 => c = 26_{10}.

1.3 Логический тип

Логический или булевский тип имеет обозначение boolean и принимает одно из двух возможных значений – «истина» (TRUE или «1») и «ложь» (FALSE или «0»). Эти значения считаются упорядоченными, так что False < True. Под этот тип отводится один байт памяти.
Булевыми операциями в Турбо Паскале являются and, or, xor, not.
Логические выражения могут включать в себя операции отношения и булевы операции. Булевыми операциями в Турбо Паскале являются and, or, xor, not. В свою очередь логические выражения могут использоваться в разветвляющихся и циклических структурах.
Пример использования логических переменных.
Допустим, в разделе описаний имеется следующая запись:
var c, d: boolean;
Тогда в разделе операторов допустимо следующее выражение:
c := d or (a = b);
Если a = b или d = TRUE, то c = TRUE.

Тема 2. Вещественные типы

В Турбо Паскале имеется несколько типов вещественных данных. Для работы со всеми вещественными типами, кроме Real, требуется математической сопроцессор 80x87, для подключения которого необходимо предварительно перед текстом программы выдать директиву компилятору {$N+}.
В отличие от порядковых, вещественные типа представляют вещественные числа, которые имеют как целые, так и дробные части, а множество вещественных чисел даже из ограниченного диапазона пронумеровать не возможно. При этом компьютер может оперировать лишь конечным набором чисел. Это связано с его конечной разрядностью, т.е. количеством двоичных разрядов, отводимых под хранение данных. Но количество допустимых значений вещественных чисел весьма велико, поэтому их нельзя отнести к порядковым.

Таблица 1.4. Вещественные типы данных

Тип	Размер	Диапазон
Real	6 байт, 11-12 знач. цифр	$-1{,}7*10^{38}\ldots-2{,}9*10^{-39}$

		$2,9*10^{-39}…1,7*10^{38}$
Single	4 байта с одинарной точностью, 7-8 знач. цифр	$-3,4*10^{38}…-1,5*10^{-45}$ $1,5*10^{-45}…3,4*10^{38}$
Double	8 байт с двойной точностью, 15-16 знач. цифр	$-1,7*10^{308}…-5,0*10^{-324}$ $5,0*10^{-324}…1,7*10^{308}$
Extended	10 байт с повышенной точностью, 19-20 знач. цифр	$-1,1*10^{4932}…-1,9*10^{-4951}$ $1,9*10^{-4951}…1,1*10^{4932}$
Comp	8-ми байтовый сложный тип	$-2^{63}+1…2^{63}-1$

Основными операциями над вещественными числами являются бинарные операции сложение (+), вычитание (-), умножение (*), деление (/), а также функции:

abs - модуль аргумента,
sqr – возведение в квадрат,
sqrt - корень квадратный,
exp - экспонента,
ln - натуральный логарифм,
sin – синус угла,
cos – косинус угла,
arctan - арктангенс.

Аргумент тригонометрических функций берется в радианах, например:

```
x := arctan(l); {x=Pi/4}
y := cos(Pi/3); {y=0.5}
```

Для вычисления логарифма с основанием a используется соотношение:
$\log_a(x) = \ln(x) / \ln(a)$.
Для вычисления других тригонометрических функций следует использовать известные соотношения:
tan(x) = sin(x) / cos(x),
ctg(x) = cos(x) / sin(x),
csc(x) = l / sin(x),
sec(x) = l / cos(x),
$\arcsin(x) = \arctan(x / (l - x^2))^{1/2}$,
arccos(x)=pi/2 – arcsin(x),
arcctg(x)=pi/2 – arctan(x).
В языке Паскаль нет операции возведения в степень. Поэтому эту операцию заменяют другими с применением стандартных функций exp и ln:
$x^a = \exp(a * \ln(x))$.
Любую из вышеуказанных функций можно применять и к аргументам целого типа. В этом случае аргумент автоматически преобразован в вещественный тип.
Кроме того, в Паскале есть встроенная константа – pi ($\pi \approx 3,14$), но нет константы основания натурального логарифма $e \approx 2,71$.
Для преобразования данных можно использовать функции:
Round – преобразует вещественное значение в ближайшее к нему значение типа LongInt.

Trunc – отсекает дробную часть вещественного числа, оставляя длинное целое LongInt.
Int и Frac – это соответственно целая (отсекаемая по направлению к нулю) и дробная части вещественного числа, также имеющие вещественный тип.
Например:
Round (3,8) = 4; Trunc (3,8) = 3;
Int (3,8) = 3.0E+00; Frac (3,8) = 8.0E-01.
Двоичное представление вещественных чисел соответствует стандарту IEEE, в котором определяется формат хранения вещественных значений с различной точностью. Например, в компьютерах с процессором INTEL вещественный тип Extended состоит из четырех полей:

Биты	79	78…64	63	62…0
Назнач.	Бит знака (s)	Порядок со смещением (порядок двойки) (e)	Бит потери значимости (j)	Дробная часть (f)

Тема 3. Строковый тип

Переменные строкового типа описываются зарезервированным словом string. Допустимыми значениями переменных строкового типа являются строки символов. В каком-то смысле строковая переменная является массивом символов, но в отличие от обычного массива фиксированной длины, длина строковой переменной может меняться. Пример описания строковой переменной:
var p: string [4];
где число в квадратных скобках указывает максимальную длину строковой переменной, т.е.
p:=’abc’; – верно,
p:=’abcdef’; – неверно.
Максимальная длина строки может составлять 255 символов. Эта длина задается по умолчанию, если, например, дано описание:
var p: string;
Если мы имеем описание:
var p: string [50];
то компьютер резервирует под эту переменную 51 байт памяти, из них 50 байт хранят коды символов и 1 байт содержит фактическую длину строки.
Как уже известно, расширенный код ASCII может содержать до 255 символов, т.е. под каждый символ отводится по одному байту памяти, а еще один байт содержит фактическую длину строки.
Например:
P:=’abcdef’;
P[0]=chr(6)
P[1]=’a’

……
P[6]=’f’.
Операции над строками:
Объединение строк – конкатенация. Обозначается знаком «+». Например, a1 и a2 – переменные строкового типа, тогда:
a1:='abc';
a2:=a1+’cde’;
Тогда результат a2=’abccde’.
Данная операция не равнозначна сложению, т.к. перемена мест слагаемых изменяет результат:
a2:=’cde’+a1;
Результат a2=’cdeabc’.
Строковая константа в тексте программы должна размещаться в пределах одной строки. Чтобы задать длинное строковое значение, занимающее несколько строк, можно воспользоваться конкатенацией:
H:=’ab……xyz’+’01……>+’;
Операцию слияния строк a1 и a2 выполняет и функция Concat:
a3 := Concat (a1, a2);
Процедуры и функции работы со строками:
Length (st) – определяет длину строковой переменной.
Delete (st, s_p, n) - удаляет n символов из строки st, начиная с позиции s_p.
Например:
st:=’012abcde3456789’;
delete (st,4,5);
Результат st:=’0123456789’.
Insert (inst, st, s_p) – вставляет строку inst в строку st начиная с позиции s_p.
Copy (st, s_p, n) – копирует n символов строки st начиная с позиции s_p.
Pos (subst, st) – определяет начальную позицию подстроки subst в строке st.
Pos ('pump', ‘axial pump’) = 7.
Str (x, st) – процедура, которая преобразует число x любого вещественного или целого типов в строку символов st так, как это делает процедура writeln перед выводом.

Тема 4. Массивы

Массивы относятся к структурным типам данных. Под массивом данных понимается упорядоченная совокупность конечного числа данных одного типа под одним именем. Имена массивов образуются так же, как и имена простых переменных. Элементами массива данных могут быть данные любого типа, включая структурированные типы. Число элементов массива фиксируется при описании и в процессе выполнения программы не меняется. Доступ к каждому отдельному элементу массива осуществляется путем указания имени массива и конкретных индексов этого элемента в квадратных скобках. Индексы могут

представлять собой выражения любого скалярного типа, кроме вещественного. Тип индекса определяет границы изменения его значений. Если задан один индекс, массив называется одномерным, если задано два индекса – двумерным, если n индексов – n-мерным.

Возможны два способа описания массивов:

1) Type <*имя типа*> = array [t1, t2,…, tN] of <*тип данных*>;
   var <*имя массива*>: <*имя типа*>;
2) var <*имя массива*> : array [t1, t2,…, tN] of <*тип данных*>;

Здесь t1, t2,…, tN – типы индексов массива (количество индексов N определяет размерность массива).

Пример: пусть в программе необходимо описать следующий двумерный массив M:

```
[ 1 2 5 ]
[ 6 7 2 ]
```

Описание этого массива в соответствии с первым способом выглядит так:

```
Type Mas = array [1..2, 1..3] of integer;
var M: Mas;
```

Для второго способа имеем:

```
var M: array [1..2, 1..3] of integer;
```

или

```
var M: array [1..2] of array [1..3] of integer;
```

В самой программе, задав конкретные значения индексов, можно выбрать определенный элемент массива:

```
N := M [1, 3 ] ;     {т.е. N будет присвоено значение 5}.
```

Для описания массива можно использовать предварительно определенные константы, например:

```
const G1 = 4; G2 = 6;
var Mas: array [1..G1, 1..G2] of real;
```

Элементы массива располагаются в памяти последовательно. Элементы одномерного массива с меньшими значениями индекса хранятся в более низких адресах памяти. Многомерные массивы располагаются таким образом, что самый правый индекс возрастает самым первым. Например, если имеется массив

```
A : array [1..2, 1..3] of integer;
```

то в памяти элементы массива будут размещены по возрастанию адресов в такой последовательности:

A[1, 1], A[1, 2], A[1, 3], A[2, 1], A[2, 2], A[2, 3].

Для работы с массивом как единым целым используется имя массива без указания индексов. Массив может участвовать только в операциях отношения «равно», «не равно» и в операторе присваивания. При этом участвующие в этих действиях массивы должны быть одинаковыми по структуре, т.е. иметь одинаковые типы индексов и количество элементов. Например,

```
var A, B : array [1..20] of real;
```

Выражение **A = B** дает результат True, если значения каждого элемента массива A равны значениям соответствующих элементов массива B. Выражение **A <> B** дает

результат True, если хотя бы один элемент массива A по значению не равен соответствующему элементу массива B.

По оператору **A := B** все значения элементов массива B присваиваются соответствующим элементам массива A, при этом значения элементов массива B остаются неизменными.

Индексированные элементы массива называются индексированными переменными и могут быть использованы точно так же, как и обычные (простые) переменные. Например, они могут находиться в выражениях в качестве операндов; использоваться в операторах повтора; входить в качестве параметров в операторы ввода и вывода данных; им можно присваивать любые значения, соответствующие их типу.

Рассмотрим типичные ситуации, возникающие при работе с массивами данных. Пусть имеем три массива и четыре вспомогательные переменные:

```
Var A, D : array [1..4] of real;
B : array [1..10, 1..15] of integer;
I, J, K : integer; S : real;
```

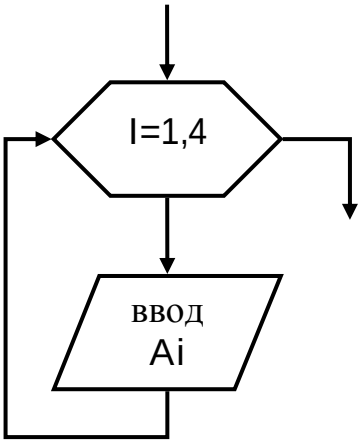
Инициализация массива заключается в присваивании каждому элементу массива одного и того же значения, соответствующего базовому типу массива. Наиболее эффективно эта операция выполняется с помощью оператора цикла for:

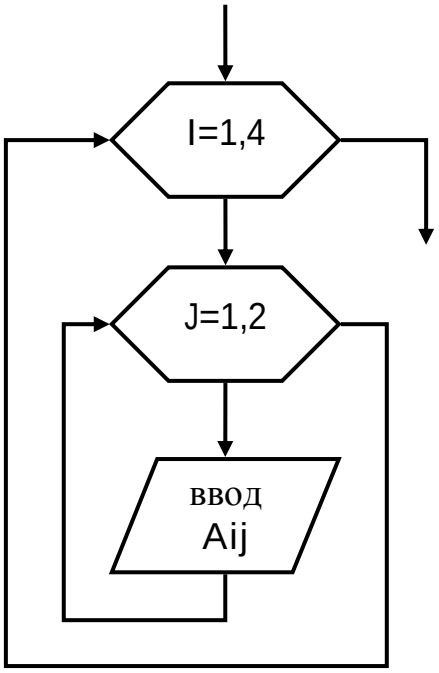
```
For I : = 1 to 4 do A[I] : = 0;
```

Для инициализации двумерного массива используется вложенный оператор for:

```
for I : = 1 to 10 do
            for J : = 1 to 15 do
            B[I, J] : = 0 ;
```

Паскаль не имеет средств ввода/вывода элементов массива сразу, поэтому вод/вывод их значений производится поэлементно, чаще всего с помощью оператора Read или ReadLn с использованием оператора организации цикла for, например (рисунок 1.1).





```
For I : = 1 to 4 do
ReadLn (A[I]);
```

а)

```
For I : = 1 to 10 do
 for J : = 1 to 15 do
 ReadLn (B[I, J]);
```

б)

Рисунок 1.1 – Алгоритм ввода массива данных а) одномерного и б) двумерного

В этих примерах использовался оператор ReadLn, поэтому каждое вводимое значение элементов массива будет набираться с новой строки.

Можно ввести и значения отдельных элементов, а не всего массива, например: Read (A[3]), B[6, 9]); – оба значения набираются на одной строке экрана, начиная с текущей позиции расположения курсора.

Вывод значений элементов массива выполняется аналогичным образом, но используются операторы Write или WtiteLn.

Тема 5. Множество

Множество является представителем группы структурных типов. Тип «множество» задает интервал значений, который является множеством всех подмножеств базового типа. Множество описывается в разделе переменных с помощью зарезервированного слова Set. Базовым типом множества может быть любой скалярный (конечный) тип, состоящий не более чем из 256 элементов. Значит, базовый тип множества не может быть Integer, LongInt, Word.

Пример описания:

```
var a: set of byte;
```

Если переменная типа множество описана как

```
var a: set of 2..5;
```

то она может принимать значение из следующих множеств ((2, 3, 4, 5), (2, 3), (2, 3, 4), …, (3, 4, 5), …).

Константы множественного типа записываются с помощью квадратных скобок и списка элементов. Пример:

```
const   a = [ 'A'..'Z', 'a'..'z' ];
        d = [ 0..9 ];
```

Отсутствие списка в квадратных скобках означает пустое множество. Для определения принадлежности элемента к множеству используется зарезервированное слово in. Пример:

```
If c in a then … else … ;
```

Множества паскаля обладают свойствами математических множеств. Над ними можно выполнять следующие операции:

S1 и S2 – константы и переменные множественного типа.

1) Объединение S1 + S2.

Объединением S1 и S2 называется множество, состоящее из элементов, принадлежащих S1 и S2 или обоим.

2) Пересечение S1*S2.

Пересечение множеств состоит из элементов, одновременно принадлежащих обоим множествам.

3) Разность S1-S2.

Разностью множеств называется множество, элементы которого принадлежат первому множеству, но не принадлежат второму.

К множествам применяются операции отношения: =, <>, <=, >=, <, >.

Для добавления в множество какого-либо элемента необходимо либо прибавить к нему множество из одного элемента, либо использовать процедуру:

Include (a, ch);

Пример: Фрагмент программы по формированию множества заглавных букв латинского алфавита с нечетными кодами [A, C, E, Y].

```
var s: set of ‘A’..’Z’;
    c: char;
begin
s:=[];
c:=’A’;
repeat
s:=s+[c];
inc(c);
inc(c);
until c>=’Z’;
```

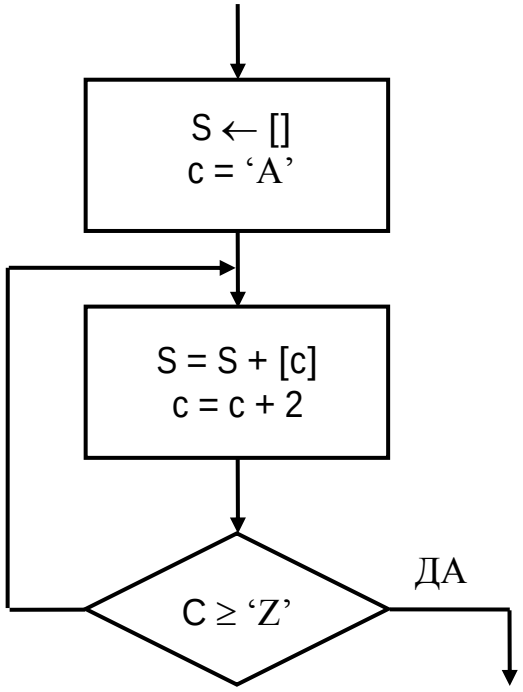


Рисунок 1.2 – Фрагмент алгоритма решения задачи

Есть и обратная процедура:

Exclude (s, c);

где s – множество, c – исключаемый элемент.

Тема 6. Старшинство операций

Порядок выполнения операций определяется их приоритетом. Есть 4 группы:

Операция	Приоритет
not	1
*, /, div, mod, and, shl, shr	2
+, -, or, xor	3
=, <>, <, >, <=, >=, in	4

1) операнд, находящийся между двумя операциями с разными приоритетами относится к операции с большим приоритетом.
2) операнд, находящийся между двумя операциями с равным приоритетом относится к операции, которая левее его.
3) выражение, заключенное в скобки, перед использованием вычисляется как отдельный операнд.
4) операции с равным приоритетом выполняются слева направо, если этот порядок не изменен с помощью круглых скобок.

Тема.7 Метки

Метки – это своеобразные элементы языка Паскаль, которые служат для изменения последовательности выполнения операторов в зависимости от определенных условий.
Метки описываются в разделе описаний с помощью зарезервированного слова Label. Например:

```
Label 1,2,3;
```

В разделе операторов метки используются следующим образом:

```
If <условие> then goto 1;
……..
1: <оператор>;
```

Пример: Вычислить выражение:

$$y = \frac{x}{3 - \frac{x}{4}}.$$

Пусть x вводится с клавиатуры. Необходимо предусмотреть возможность равенства нулю знаменателя. При равенстве знаменателя нулю программа должна вернуться к вводу x.

```
Program Pr;
Label 1;
var x, y, a: real;
begin
1: write (‘Введите x=’);
readln (x);
```

```
a:=3-x/4;
if a=0 then
begin
writeln (‘x введено неверно’);
goto 1;
end
else y:=x/a;
writeln (‘y=’,y);
end.
```

Тема 8. Записи

Запись – это такой тип данных, который содержит определенное число элементов (полей). Поля могут быть разных типов. Различают записи с фиксированными частями и записи с вариантами. Описание типа «запись» находится в разделе описаний и имеет вид:

```
type <имя> = record
<описание переменных>;
end;
```

В разделе <описание переменных> приводится идентификатор каждого поля и его тип.

Рассмотрим на примере. Допустим, нам дан моном: $C \cdot x^k \cdot y^l \cdot z^m$.

Удобнее всего в памяти компьютера его хранить, используя тип «запись». Полями записи будут вещественные коэффициент С и показатели степеней.

```
const
max=3;
type
t=record
C: real;
d: array[1..max] of word;
end;
```

Теперь, если мы имеем переменную M типа t, то к ее коэффициенту C можно обратиться следующим образом:

```
X:=M.C;
```

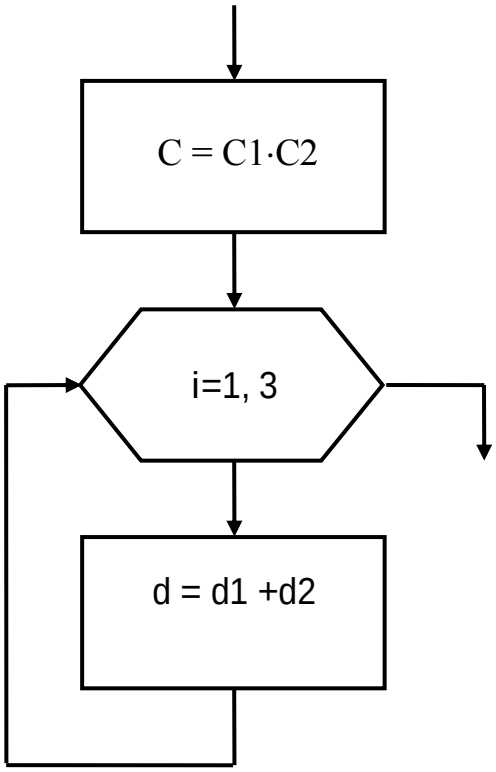
а еще это можно сделать с помощью зарезервированного слова with:

```
with M do X:=C;
```

т.е. во втором случае после слова do мы уже работаем без указания переменной типа t.

Пример программирования для перемножения двух мономов:

$M_1 \cdot M_2 = (C1 \cdot x^k \cdot y^l \cdot z^m) \cdot (C2 \cdot x^p \cdot y^q \cdot z^r)$.



Перемножение коэффициентов

Сложение соответствующих степеней

Рисунок 1.3 – Фрагмент алгоритма решения задачи

```
Program monom;
type t=record
      c: real;
      d: array [1..3] of word;
end;
var M1, M2, M3: t;
k: word;
begin with M3 do begin
c:=M1.C*M2.C;
for k:=1 to 3 do
d[k]:=M1.d[k]+M2.d[k];
end;
end.
```

С помощью типа запись очень удобно работать с комплексными числами, представляя действительную и мнимую части как поля.

Записи с вариантами в различных ситуациях могут иметь разную структуру. Записи с вариантными частями имеет следующее описание:

type rv=record
v1, v2, … vn: integer; - описание фиксированной части
case n: word of
val_1: <список полей - описание переменных>;
val_2: <список полей - описание переменных>;
…….

val_7: <список полей - описание переменных>;
end;
Переменная часть должна следовать после фиксированной части, если таковая имеется.
Ели есть описание:
var M: rv;
то обращение к полям производится следующим образом:
M.v1 – обращение к фиксированному полю;
M.n:=val_n; – обращение к вариантной части;
M.a:=8; – обращение к вариантной части;
Типы могут быть различными, например скалярные.

Тема 9. Скалярные типы

Скалярными часто называют перечисляемые типы и отрезки типов. Перечисляемые типы определяют упорядоченные наборы значений. Эти наборы определяются перечислением идентификаторов. Пример:
type digits=(red, orange, … violet);
где (red, orange, … violet) – возможные значения данного типа.
Каждое из них имеет порядковый номер. Первое значение имеет порядковый номер «0». К ним применима функция ord:
ord (orange) = 1,
которая возвращает номер положения в списке.
Кроме того применимы функции:
Pred (green) = yellow – предшествующее значение;
Succ (green) = blue – последующее значение.
Справедливы также соотношения: green < blue.
Отрезок типа можно описать:
type nt=0..6;
Отрезок представляет собой диапазон главного типа, например byte. Возможные значения типа nt: 0, 1, 2, 3, 4, 5, 6.

Тема 10. Типы, определяемые пользователем

Раздел описания типов начинается с зарезервированного слова type. Пример:
type t1=type_1;
t2=type_2;
t1, t2 – идентификаторы типа, определяемого пользователя.
type_1, type_2 – соответствующие базовые типы.
Пример:
type ind=1..50;
Arr100=array [0..100] of word;

```
    Arr100by8=array[1..8] of Arr100;
……
r=record
……
end;
```

ind – интервал целочисленных значений от 0 до 50.
Arr100 – элемент массива типа word.
Arr100by8 – двумерный массив 100x8.
Далее типы можно использовать в разделе описаний.

Тема 11. Совместимость типов

В выражениях допускается использовать только совместимые типы. Типы являются совместимыми, если выполняется хотя бы одно из следующих условий:
- оба типа одинаковы;
- оба типа вещественные;
- оба типа целочисленные;
- один тип является поддиапазоном другого;
- оба типа являются отрезками одного и того же базового типа;
- оба типа являются множественными с совместимыми базовыми типами;
- один тип «Pointer», другой – любой ссылочный.

Но есть еще понятие совместимости по присвоению. Согласно стандарту ISO7185 допустимо присваивание T1:=T2, если выполняется одно из следующих условий:
- T1 и T2 имеют тождественные типы и ни один из них не является файловым или структурным, содержащим файловый на оном из своих уровней;
- T1 и T2 совместимые порядковые, а значения T2 попадают в диапазон возможных значений T1;
- T1 и T2 – вещественные типы, а значения T2 попадает в диапазон возможных значений T1;
- T1 является вещественным, а значение T2 целочисленным;
- T1 и T2 оба строковые типы;
- T1 является строковым типом, а T2 является символьным;
- T1 и T2 совместимые множественные типы, а все члены значений T2 попадают в диапазон возможных значений T1;
- T1 и T2 оба совместимые типы указателей.

Тема 12. Динамические переменные. Указатели

До этого мы работали со статическими переменными. Они размещаются в памяти компьютера при запуске и находятся там постоянно во время ее выполнения. В результате память используется неэффективно. Указатели позволяют создавать переменные во время выполнения программы, т.е. динамически. При необходимости можно размещать в памяти новые переменные и освобождать

память, когда необходимость в них отпадает. В этом случае память используется более гибко.

Обращение к динамическим переменным производится не по имени, а по их адресу в памяти. Адрес динамической переменной хранится в переменной ссылочного типа, которая называется указателем. Указатель занимает двойное слово (сегмент и смещение). Описание ссылочного типа имеет вид:

type A=^<type_1>;

type_1 – идентификатор базового типа.

Пример:

```
type byte_ptr=^byte;
var a, b, c: byte_ptr;
var a1, b1, c1: ^byte;
```

Обе строки описания переменных эквивалентны.

byte – базовый тип;

byte_ptr – указатель на byte.

Этот тип удобно использовать, если мы имеем дело с матрицами, векторами, полиномами, если их размер заранее не известен, но определяется в ходе выполнения программы.

Для обращения к переменной по указателю, нужно его разыменовать, т.е. поставить справа от идентификатора переменной ссылочного типа символ «^», т.е.:

```
m:=a^;
```

Рассмотрим пример программы, которая размещает в памяти динамические переменные, сообщает о размере свободной памяти после размещения и о том какое место в памяти зарезервируется под переменную.

```
Program pointer;
const max=1000;
type vector_p=^vector;
      vector=array [1..max] of LongInt;
var p: vector_p;
    i, j: LongInt;
    k: word;
begin
i:=MemAvail;
writeln (‘Перед размещением свободно’, i:8, ‘байт’);
New (p);
j:=MemAvail;
writeln (‘После размещения свободно’, j:8, ‘байт’);
k:=i-j;
writeln (‘Размещено’, k:8, ‘байт’);
for k:=1 to max do v^[k]:=sqr(k)+1;
Dispose (p);
p:=nil;
i:=MemAvail;
```

```
writeln (‘После освбождения свободно’, i:8, ‘байт’);
writeln ('Нажмите ввод');
readln;
end.
```

Некоторые процедуры для работы с динамическими перемеными:

New (p) – отводит память под переменную p^, где p^ - имя массива.

Dispose (p) – обратна New, она освобождает память, отводимую под динамическую переменную, и делает доступной для последующего использования.

Процедуру следует всегда использовать, если необходимость в динамической переменной отпадает. Если динамическая переменная уже уничтожена, то к ней нельзя обращаться (могут быть проблемы с ОС).

FreeMem (p, size) – освобождает адресуемый указателем p фрагмент памяти размером size.

GetMem (p, size) – резервирует блок памяти размером size. Адрес начала блока присваивается указателю p.

Функции:

Addr (x): pointer – возвращает адрес аргумента (эквивалент @x).

Seg (x): word – возвращает численное значение сегмента адреса переменной x.

Ofs (x): word – возвращает численное значение смещения адреса переменной x.

Ptr (seg, ofs):pointer – возвращает указатель с адресом $Seg:$Ofs.

nil – встроенная константа, она применяется к указателю и означает, что он ни на что не указывает.

13.1 Динамические структуры данных

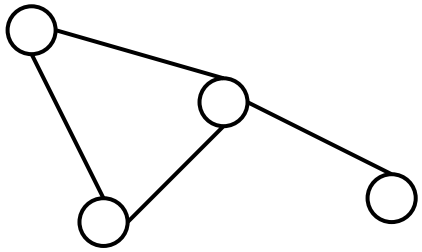
Мы уже рассматривали структурированные данные (массив, множество, запись). Их характерная особенность – они статические, т.е. размещаются в памяти компьютера при компиляции и остаются там на протяжении выполнения программы.

Компоненты динамических структур определяются и задаются в ходе выполнения программы.

Основные разновидности динамических структур:

- списки;
- деревья;
- графы.

Любая динамическая структура состоит из узлов и связей между ними.

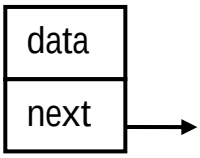


Узел представляет собой запись минимум с двумя полями:

1) поле данных: переменные, массив, множество, запись;
2) поле указателя: указатель на тип узла динамической структуры.

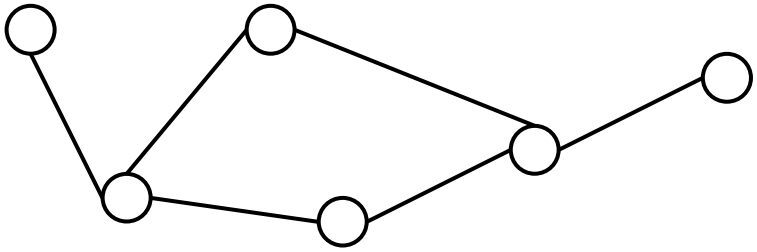
Описание:

```
type
     node_ptr=^node;
     node=record
     data: real;
     next: node_ptr;
end;
```



Граф

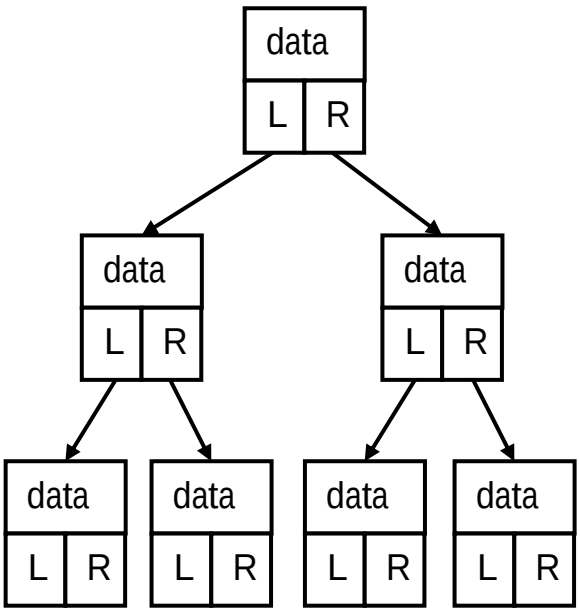
Наиболее общая динамическая структура:



Здесь любая вершина (узел) связана с произвольным количеством других вершин. Структура эта достаточно сложна и не наглядна, поэтому мы не будем не ней останавливаться.

Дерево

Бинарным деревом называется структура, в которой каждый узел ссылается на два других нижестоящих.

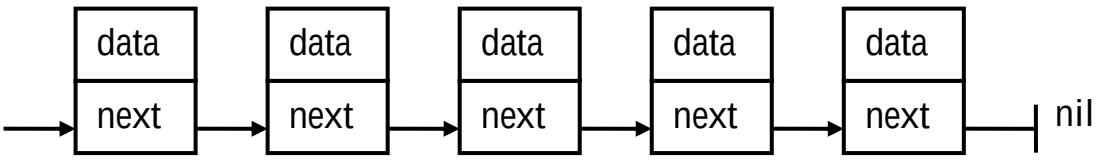


Узел, на который не ссылается ни один другой узел, называется корнем.
Узел, который не ссылается ни на один другой, называется листом дерева.
Описание узла дерева:

```
type
    node_ptr=^node;
    node=record
    data: integer;
    L, R: node_ptr;
end;
```

12.2 Связные списки

Связный список представляет собой цепочку записей (узлов), в которой каждая запись содержи, кроме основных данных, ссылку на следующую запись в цепочке. Возглавляет цепочку списка «корневой» указатель, который указывает на первую запись в списке.
Односвязный список имеет следующую структуру:



Указателю последнего узла присваивают значение nil, которое говорит о конце списка.
Структура списка со связями является динамической, т.е. может меняться в ходе выполнения программы.

Односвязный список в принципе схож с массивом (динамическим), но у них есть различия. Для списка и динамического массива отводится ровно столько памяти, сколько нужно. Главные различия проявляются при работе с элементами списка и массива.
Допустим, нам необходимо добавить новый элемент в массив и список.
Массив: нужно сдвинуть вправо все элементы, начиная с «b».
Для добавления элемента «c» в связной список необходимы две операции:
a.next:=@c;
c.next:=@b;
Т.е. необходимо изменить указатель, а сами узлы перемещаться не будут.
Для удаления элемента в массиве необходимо сдвигать все элементы правее «b» влево. Чем больше элементов, тем больше время выполнения программы.
В случае со списком необходимо сослаться на узел следующий за удаляемым:
a.next:=@b;
disposr (@c);
Однако в случае, когда необходимо получить доступ к отдельному элементу:
- в массиве достаточно указать его с помощью индекса;
- в связном списке придется перебирать все элементы.
Вывод: списки лучше использовать в тех случаях, когда необходимо часто добавлять и удалять элементы структуры. В случае если необходимо часто получать доступ к элементу, то лучше использовать массив.
В программировании различают две разновидности связных списков:
- стек;
- очередь.
Стеком называется последовательность, добавление элементов в которую происходит с одного конца, называемого вершиной стека. Т.е. извлекать элементы из стека можно лишь в последовательности обратной добавлению (Принцип: «первый на входе – последний на выходе»).
Если стек реализован в виде односвязного списка, то вершиной его может быть либо голова, либо хвост списка. Обычно вершиной является голова, т.к. из нее удобнее извлекать элементы.
Добавление и извлечение элемента в/из стек.
Допустим, мы создаем модуль для работы со стеком. Запишем тексты процедур для добавления и извлечения элементов в/из стек: stack_in, stack_out – соответственно. Для всего модуля у нас есть общее описание типа:

```
type
    node1_ptr=^node1;
    node1=record
    data: real;
    next: node1_ptr;
end;
```

1. Добавление элемента в стек.

```
Procedure  Stack_in (var top: node1_ptr; val: real;);
var a: node1_ptr;
```

```
begin
New (a);
a^.data:=val;
a^.next:=top;
top:=a;
end;
```
Параметрами данной процедуры являются указатель на вершину стека (top) и значение (val), присваиваемое потом data.

2. Извлечение элемента из стека.

В данном случае необходимо предусмотреть проверку наличия в стеке элементов, ибо при попытке извлечения элемента из пустого стека возникает ошибка.

Здесь b – переменная, в которую запишется значение извлекаемого элемента.

```
Procedure  Stack_out (var top: node1_ptr; b: real;);
var a: node1_ptr;
begin
if top <> nil then
    begin
    b:=top^.data;
    a:=top;
    top:=top^.next;
    Dispose (a);
    end;
end;
```

Очередь – это последовательность, добавление элементов в которую происходит с одного конца, называемого головой, а извлечение из другого конца, называемого хвостом (Принцип: «первый на входе – первый на выходе»).

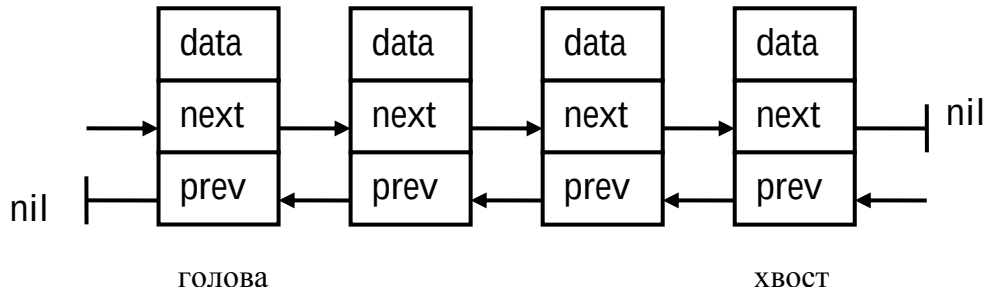
Очередь достаточно сложно реализовать с помощью односвязного списка. Поэтому используется другая структура – двусвязный список.

Список называется двусвязным, если каждый его элемент содержит указатели на следующий и предыдущий элементы списка.

Пример списка:

```
type
     node2_ptr=^node2;
     node2=record
     data: real;
     next, prev: node2_ptr;
end;
```

Двусвязный список имеет следующую структуру:



Добавление элемента в двусвязный список:

```
c.next:=@b;
c.prev:=@a;
a.next:=@c;
b.prev:=@c;
```

Удаление элемента из двусвязного списка:

```
a.next:=@b;
b.prev:=@a;
Dispose (@c);
```

Т.о. двусвязный список является наиболее удобным способом работы с очередью.

Добавление и извлечение в/из очередь.

Допустим, мы создаем модуль для работы с очередью. Запишем тексты процедур для добавления и извлечения элементов в/из очередь: node2_in, node2_out – соответственно. Для всего модуля у нас есть общее описание типа:

```
type
    node2_ptr=^node2;
    node2=record
    data: real;
    next, prev: node2_ptr;
end;
```

1. Добавление элемента в очередь.

```
Procedure node2_in (var head, tail: node1_ptr; val: real;);
var a: node2_ptr;
begin
New (a);
a^.data:=val;
a^.next:=head;
if head=nil then tail:=a
else head^.prev:=a;
a^.ptev:=nil;
head:=a;
end;
```

Здесь head и tail – указатели на голову и хвост очереди.

2. Извлечение элемента из очереди.

В данном случае необходимо предусмотреть проверку наличия в очереди элементов, ибо при попытке извлечения элемента из пустой очереди возникает ошибка.

```
Procedure node2_out (var head, tail: node2_ptr; b: real;);
var a: node2_ptr;
begin
if tail <> nil then
    begin
    b:=tail^.data;
    a:=tail;
    if tail^.prev=nil then nead:=nil;
    tail:=tail^.prev;
    if tail <> nil then tail^.next:=nil;
    Dispose (a);
    end;
end;
```

РАЗДЕЛ IV. РАБОТА С ФАЙЛАМИ

Файлом называется именованная область на внешнем носителе, в которой хранится определенная информация. Любой файл имеет три характерные особенности: у файла есть имя, что позволяет программе работать с несколькими файлами; он содержит компоненты одного типа; длина файла не оговаривается и ограничивается емкостью накопителя.
Мы уже знакомились со стандартными процедурами ввода/вывода write, read.
При выполнении программы на Паскале автоматически открываются две файловые переменные Input и Output.
Input – обеспечивает ввод символов с клавиатуры.
Output – обеспечивает вывод элементов на экран.
Имя файла в процедурах Read и Write не указывается, если работа ведется со стандартным файлом. Но с помощью этих процедур можно связываться и с другими устройствами ввода-вывода (например, последовательный порт, дискета). К каждому из этих устройств можно обратиться путем определения соответствующих файлов. При этом используются обычные имена, принятые в MS-DOS:
D:\TP7.0\TP\<Имя файла> - файл на жестком диске.
A:\<Имя файла> - файл на дискете.
PRN – принтер.
COM1 – последовательный порт.
В Паскале есть три класса файлов:
- текстовые;
- типизированные;
- нетипизированные.

Тема 1. Текстовые файлы

Текстовый файл – это файл, состоящий из символов, организованных в строки.
Такой файл создается обычным текстовым редактором.
Для записи информации в файл используется т.н. файловая переменная, которая в случае с текстовым файлом будет иметь тип Text. Текстовый файл трактуется в Turbo Pascal как совокупность строк переменной длины. Доступ к каждой строке возможен лишь последовательно, начиная с первой. При создании текстового файла в конце каждой записи (строки) ставится специальный символ возврата каретки CR (код #13) и перевода строки LF (код #10). Заканчивается текстовый файл признаком конца файла (код #26). Эти символы не отображаются на экране программами просмотра текстовых файлов.
Прежде чем записывать информацию в файл, необходимо связать файловую переменную с файлом на диске. Для этого используется стандартная процедура

модуля System – ASSIGN. Эта процедура лишь связывает файловую переменную с файлом на диске (уже существующем или нет).
Для открытия существующего файла следует воспользоваться процедурами модуля System:
Rewrite – открывает файл для записи, а если файл уже существует, то удаляет его содержимое;
Reset – открывает файл только для чтения.
Непосредственно запись информации в текстовый файл или чтение информации из него осуществляется с помощью уже известных процедур write, writeln, read, readln. В этом случае первой в списке параметров необходимо указать файловую переменную, тогда запись будет осуществляться в связанный с ней файл:
Writeln (out_file, a1, a2, …, an);
out_file – переменная типа Text.
Точно также как и при выводе на экран при записи файл возможны форматы. Например, если a – выражение целого, булевского или строкового типа, то
Write (out_file, a:n);
Это означает запись «a» в правые позиции поля размером n позиций. Но если значение «a» занимает больше позиций, то значение n игнорируется.
Если «a» вещественного типа, то допустим такой формат:
Write (out_file, a:n:m);
Это означает запись «a» в правые n позиций с m десятичными разрядами. Если ширина поля вывода не указана, то соответствующий параметр выводится вслед за предыдущим без какого-либо их разделения.
Чтение из файла выполняется аналогично, но только с использованием процедур read и readln. При чтении по процедуре readln после считывания последней переменной оставшаяся часть строки до символа конец строки пропускается, и новое обращение к процедурам read и readln начинается первого символа новой строки.
Переменные строкового типа вводятся при помощи процедуры readln (для перевода на новую строку). Если количество символов во входном потоке данных больше максимальной длины строки, то «лишние» символы до конца строки отбрасываются, а новое обращение к процедуре read возвращает пустую строку.
Рассмотрим пример записи в файл значений функции:

$$f(x) = \frac{x}{x+1}.$$

Программа:

```
Program File_demo;
var    x: real;
       k: integer;
       o_f: text;
Function Fun (x: real): real;
begin
Fun:=x/(x+1);
end;
```

```
begin
assign (o_f, ‘d:\...\function.txt’);
rewrite (o_f);
x:=0;
writeln (o_f, ‘Таблица значений функции F(x)’);
writeln (o_f);
writeln (o_f, ‘x’:9, ‘F(x)’:19);
writeln(o_f);
for k:=1 to 50 do
begin
writeln (o_f, x:9:3, Fun(x):19:9);
x:=x+1;
if k mod 10 = 9 then writekn (o_f);
end;
close (o_f);
end.
```

Тема 2. Типизированные файлы

Типизированные файлы используются для хранения однородной по типу информации. Если в случае с текстовым файлом мы оперировали файловой переменной, то здесь в разделе описаний мы описываем весь файл:
var f: file of <тип файла>;
где <тип файла> - любой тип за исключением файлового.
Для работы с типизированными фалами применяются те же процедуры и функции, что и при работе с текстовыми, за исключением:
- типизированный файл, открытый процедурой Reset доступен еще и для записи;
- в процедурах read и write каждый из параметров должен быть переменной описанного типа, не допускаются выражения и константы;
- процедуры readln и writeln неприменимы.
Если мы в выше описанном примере запишем результаты в типизированный файл типа, например, real, мы получим не упорядоченный в столбики набор значений, а хаотичный разброс чисел без заголовков.
Но зачастую использование типизированных файлов экономит память компьютера.

Тема 3. Нетипизированные файлы

Нетипизированные файлы применяются для более эффективного выполнения операций ввода/вывода из внешних файлов.
При работе с этими файлами можно использовать быстрые дисковые операции низкого уровня.

Нетипизированные файлы дают возможность прямого доступа к любому файлу на диске независимо от его структуры и типа.
Описание файловой переменной:
var type_file: file;
Среди параметров процедур Reset и Rewrite кроме файловой переменной имеется необязательный второй параметр:
Reset (type_file, n);
n – описывает размер индивидуальной записи (в байтах). Если параметр n отсутствует, то по умолчанию берется n=128.

10.4 Основные процедуры и функции для работы с файлами

Процедуры:
Assign (…) – связывает внешний файл с файловой переменной;
Block Read (…) – считывает из нетипизированного файла одну или несколько записей;
Block Write (…) – записывает в типизированнй файл одну или несколько записей;
Close (…) – закрывает открытый файл;
Erase (…) – стирает внешний файл;
Read (…) – считывает одно или несколько значений из файла;
Readln (…) – считывает одно или несколько значений из файла, но выполняет пропуск до начала следующей строки текстового файла;
Reset (…) – открывает существующий файл только для чтения;
Rewrite (…) – открывает существующий файл для перезаписи;
Write (…) – запись в файл;
Writeln (…) – запись в файл с записью в текстовый файл признака конца строки.
Функции:
Eof (…): boolean – возвращает True, если достигнут конец файла или файл пуст;
File Pos (…): longint – возвращает текущую позицию для файла (для текстовых файлов не используется);
File Size (…): longint – возвращает размер файла (для текстовых файлов не используется).

Раздел V. ГРАФИЧЕСКИЙ РЕЖИМ В TURBO-PASCAL

Тема 1. Графический режим работы

Монитор компьютера с точки зрения вывода изображения может работать в одном из двух режимов текстовом или графическом. Различие между текстовым и графическим режимами работы монитора заключается в возможностях управления выводом визуальной информации. В текстовом режиме минимальным объектом, отображаемым на экране, является символ, алфавитно-цифровой или какой-либо иной. В обычных условиях экран монитора, работающий в текстовом режиме, может содержать не более 80 символов по горизонтали и 25 символов по вертикали, т.е. всего 2000 визуальных объектов. Но для работы с изображениями текстовый режим дисплея не подходит.

В графическом режиме минимальным объектом является так называемый пиксел (от английского pixel, возникшего в результате объединения слов «рисунок» (picture) и элемент (element)). Пиксел имеет меньшие размеры по сравнению с символом, его геометрические размеры определяются разрешением монитора. Разрешение монитора задается в виде rx×ry, где rx – количество пикселов на экране по горизонтали, ry – количество пикселов по вертикали. На практике используются стандартные разрешения такие как: 640×480, 800×600, 1024×768, 1280×1024 и т.д. Геометрические размеры пиксела определяют степень детализации изображения, его качество.

Тема 2. Графические координаты

Любое изображение формируется из достаточно простых геометрических фигур – отрезки прямых, окружности и т.д. Из геометрии известно, что положение геометрического объекта и его форма задаются координатами его точек. Следовательно, что бы запрограммировать графический вывод, надо задать координаты графического объекта.

Графические координаты задают положение точки на экране дисплея. Поскольку минимальным элементом, к которому имеет доступ программист, является пиксел, то в качестве графических координат используются порядковые номера пикселов. Допустимый диапазон изменения графических координат составляет [0, *rx*-1] для x-координат и [0, *ry*-1] для y-координаты. Точкой отсчета является верхний левый угол экрана. Значения x-координаты отсчитываются слева направо, а y-координаты – сверху вниз. Последнее отличает графические координаты от обычных декартовых координат, принятых в математике.

Для правильного отображения графика на экране необходимо учесть различия между декартовой и графической системами координат. Выделим три таких различия:

1. Графические координаты принимают только целочисленные значения.
2. Графические координаты принимают значения, ограниченные как снизу (нулевым значением), так и сверху (значением разрешения).
3. Графическая координата *y* отсчитывается сверху вниз.

Геометрические декартовы координаты точки (x,y) для отображения ее на экране следует пересчитать в графические (xg, yg) по формулам:

$$xg = \lfloor sx \times x \rfloor + dx,$$

$$yg = ry - \lfloor sy \times y \rfloor - dy.$$

где $\lfloor x \rfloor$ – целая часть *x*, sx и sy – масштабные множители, выбираемые из условия

$$rx = \lfloor sx \times x_{MAX} \rfloor + 1,$$

$$ry = \lfloor sx \times y_{MAX} \rfloor + 1.$$

Здесь x_{max} и y_{max} – максимальные значения геометрических координат. Пересчет координаты y по такой же формуле, что и для x, привел бы к зеркально отраженному относительно горизонтальной линии изображению. Слагаемые dx и dy обеспечивают смещение изображения относительно левого верхнего угла экрана. Например, изображение будет смещено в центр экрана при:

$$dx = \lfloor rx / 2 \rfloor,$$

$$dy = \lfloor ry / 2 \rfloor.$$

Тема 3. Переключение между текстовым и графическим режимами

Работу в графическом режиме поддерживает видеоадаптер. Работой видеоадаптера управляет специальная программа, которая называется драйвером. Драйвер хранится в отдельном файле на диске и содержит как исполняемый код, так и необходимые ему для работы данные. Признак файла с драйвером – расширение .bgi имени файла.

Переключение в графический режим и работа в нем реализованы в Турбо Паскале в виде набора процедур, находящихся в специальном модуле graph.tpu. Этот модуль должен явно подключаться к программе с помощью оператора использования *uses*. В модуль graph входит примерно 50 процедур.

При работе в графическом режиме следует иметь в виду, что большинство видеоадаптеров могут работать в нескольких графических режимах. Эти режимы различаются, прежде всего, разрешением и набором допустимых цветов.

Для перевода в графический режим программа должна определить тип видеоадаптера. Это можно сделать, явно указав в программе тип видеоадаптера или дав программе возможность самостоятельно определить значение соответствующего параметра. Для этого необходимо ввести переменную целого типа, пусть ее идентификатор будет gd. При явном определении видео адаптера в программе должен присутствовать оператор присвоения:

```
gd := value;
```

где value – это либо некоторое число, либо встроенная константа (некоторые возможные значения приведены в табл. 3.1).

Таблица 3.1 - Встроенные константы видео адаптеров

Константа	Значение
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMono	5
HercMono	7
ATT400	8
VGA	9
PC3270	10

При автоматическом распознавании видеоадаптера в правой части оператора присвоения используется константа Detect или нулевое значение.
Второе, что должна сделать программа, - задать определенный графический режим. Для этого следует ввести еще одну переменную целого типа, назовем ее gm, и присвоить ей значение (некоторые допустимые значения приведены в табл. 3.2).
Переключение в графический режим работы дисплея выполняется вызовом процедуры InitGraph из модуля graph:
InitGraph (gd, gm, ‘c:\tp\bgi’);

Таблица 3.2 - Графические режимы

Константа	Значение	Описание графического режима
EGALo	0	640×200, 16 цветов, 4 страницы
EGAHi	1	640×350, 16 цветов, 2 страницы
EGA64Lo	0	640×200, 16 цветов, 1 страница
EGA64Hi	1	640×350, 4 цвета, 1 страница
VGALo	0	640×200, 16 цветов, 4 страницы
VGAMed	1	640×350, 16 цветов, 2 страницы
VGAHi	2	640×480, 16 цветов, 1 страница

Первый параметр в этой процедуре задает тип видеоадаптера, второй определяет режим, а третий представляет собой строку с указанием расположения драйвера на диске. Пустая строка означает, что графический драйвер находится в том же каталоге, что и программа. Процедура InitGraph переводит систему в графический режим, а затем возвращает управление вызывающей программе.

Завершение работы в графическом режиме производится с помощью процедуры CloseGraph, которая выгружает драйвер из памяти и восстанавливает предыдущий видеорежим.
С помощью процедур RestoreCrtMode и SetGraphMode можно переключать между текстовым и графическим режимами, не закрывая графический режим.

Тема 4. Основные операции рисования

1. Построение отрезков прямых.
Line (x1, y1, x2, y2) – выводит на экран отрезок прямой, задаваемый начальной точкой с координатами (x1, y1) и конечной точкой отрезка с координатами (x2, y2). Координаты точек задаются в пикселях.
LineRel (x, y) – построение прямолинейного отрезка, начинающегося из текущей точки A (xk, yk) и заканчивающийся в точке с координатами B (xk + x, yk + y). Для перевода пера в требуемую точку A можно использовать эту же процедуру, но предварительно установив цвет линии такой, как и цвет фона изображения.
Для изменения параметров линий используется процедура SetLineStyle, которая имеет следующий формат:
SetLineStyle (s: word; p: word; th: word);
где s – задает стиль линии;
p – задает шаблон линии;
th – задает толщину линии.
Параметры процедуры SetLineStyle задаются либо числами, либо соответствующими константами.
2. Построение прямоугольников.
Rectangle (x1, y1, x2, y2) – процедура построения прямоугольника. Параметры x1 и y1 задают графические координаты верхнего левого угла прямоугольника, а параметры x2 и y2 – координаты нижнего правого угла.
Bar (x1, y1, x2, y2) – процедура построения прямоугольника с закрашенной внутренне областью. Параметры x1 и y1 задают графические координаты верхнего левого угла прямоугольника, а параметры x2 и y2 – координаты нижнего правого угла. Стиль и цвет заполнения прямоугольника может быть выбран при помощи процедуры SetFillStyle(Style, Color). Первый параметр этой процедуры задает стиль заполнения, а второй – цвет. Примеры допустимых стилей представлены в таблице 3.3.

Таблица 3.3 - Стили заполнения геометрических фигур

Константа	Код	Описание
EmptyFill	0	Сплошное заполнение цветом фона
SolidFill	1	Сплошное заполнение данным цветом
LineFill	2	Заполнение горизонтальными линиями
LtSlashFill	3	Диагональное заполнение (///)
SlashFill	4	Диагональное заполнение толстыми линиями (///)

BkSlashFill	5	Обратное диагональное заполнение толстыми линиями (\\\)
LtBkSlashFill	6	Обратное диагональное заполнение (\\\)
HatchFill	7	Клетчатое заполнение
XhatchFill	8	Косое клетчатое заполнение
InterleaveFill	9	Чередующееся линейное заполнение
WideDotFill	10	Редко расположенные точки
CloseDotFill	11	Часто расположенные точки

3. Построение окружности и эллипса.

Circle (x, y, r) – процедура построения окружности с центром в точке (x, y) и радиусом r. Радиус задается в пикселях.

Arc (x, y, BA, EA, r) – процедура построения дуги окружности с центром в точке (x, y) радиусом r. Параметры BA, EA задают соответственно начальный и конечный углы дуги. Углы описываются против часовой стрелки и указываются в градусах. Нулевой угол соответствует горизонтальному направлению вектора слева направо. Если задать значения начального угла 0 и конечного – 359°, то будет выведена полная окружность.

Ellipse (x, y, BA, EA, rx, ry) – процедура построения эллипсной дуги с центром в точке (x, y). Параметры BA, EA задают соответственно начальный и конечный углы дуги. Углы описываются против часовой стрелки и указываются в градусах. Нулевой угол соответствует горизонтальному направлению вектора слева направо. Если задать значения начального угла 0 и конечного – 359°, то будет выведен полный эллипс. Параметры rx, ry задают соответственно горизонтальный и вертикальный радиусы эллипса в пикселях.

FillEllipse (x, y, rx, ry) – процедура построения эллипса с закрашенной внутренней областью с центром в точке (x, y). Параметры rx, ry задают соответственно горизонтальный и вертикальный радиусы эллипса в пикселях. Стиль и цвет заполнения эллипса может быть выбран при помощи процедуры SetFillStyle(Style, Color).

4. Дополнительные процедуры и функции модуля Graph.

GetMaxX и GetMaxY – функции, которые возвращают наибольший номер пикселя по горизонтали и по вертикали соответственно.

GetPixel (x, y) – функция, которая определяет цвет пикселя в точке с заданными графическими координатами.

PutPixel (x, y, color) – процедура, которая выводит пиксель в заданной точке (x, y) и с указанным цветом color.

SetBkColor (color) – процедура, которая устанавливает цвет фона.

SetColor (color) – процедура, которая устанавливает цвет графического объекта.

Параметр color указанных выше процедур является либо цифровой код цвета, либо встроенная константа, обозначающая цвет (таблица 9.5).

5. Вывод текста в графическом режиме

Во время работы в графическом режиме часто требуется вывести текст, например, название графического объекта. Для этого необходима поддержка работы со шрифтами. Шрифт представляет собой набор символов, используемых для

отображения текстовой информации. В Турбо Паскале используются два вида шрифтов, различающихся своим внутренним форматом – растровый (один) и векторный (несколько). Растровый символ задается с помощью матрицы элементов изображения этого символа. Матрица имеет размер 8x8 пикселов. Векторный шрифт задается набором векторов, которые указывают, как графической системе как рисовать символ, что позволяет увеличивать размер текста без ухудшения качества изображения. По умолчанию, а также при неверной ссылке на векторный шрифт текст будет выводиться растровым шрифтом DefaultFont.

Таблица 3.4

Цвет	Константа	Код
Черный	Black	0
Синий	Blue	1
Зеленый	Green	2
Бирюзовый	Cyan	3
Красный	Red	4
Розовый	Magenta	5
Коричневый	Brown	6
Светло-серый	LightGrey	7
Темно-серый	DarkGrey	8
Светло-синий	LightBlue	9
Светло-зеленый	LightGreen	10
Светло-бирюзовый	LightCyan	11
Светло-красный	LightRed	12
Светло-розовый	LightMagenta	13
Желтый	Yellow	14
Белый	White	15

Для вывода текста служит процедура OutTextXY:

```
OutTextXY (x, y, ' <text> ');
```

Первые два параметра задают графические координаты начала текста (верхний левый угол текста), а третий параметр собственно текст или строковые / символьные константы / переменные. Для вывода на экран численных значений следует вначале преобразовать их в строку символов при помощи процедуры Str:

```
Str (p1, p2);
```

Здесь параметр p1 задает имя численной переменной, а параметр p2 – имя строковой переменной.

Следует обратить внимание, что в графическом режиме операторы вывода Write и WriteLn могут некорректно работать.

Способ вывода текста можно задать, вызвав предварительно процедуру SetTextStyle:

```
SetTextStyle (font, dir, size);
```

Здесь font задает шрифт, dir – направление текста (0 – горизонтальное, 1 - вертикальное), size – размер.

Процедура SetTextJustify (hor, ver) устанавливает выравнивание в горизонтальном (первый параметр) и вертикальном (торой параметр) направлениях. Типы выравниваний приведены в таблице 3.5.

Таблица 3.5 - Типы выравниваний текста

Горизонтальное			Вертикальное		
Константа	Код	Выравнивание	Константа	Код	Выравнивание
LefText	0	Влево	BottomText	0	Вниз
CenterText	1	По центру	CenterText	1	По центру
RightText	2	Вправо	TopText	2	Вверх

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вальвачев, А.Н., Крисевич, В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС. – Мн.: Выш. шк., 1989. – 223 с.
2. Паскаль для персональных компьютеров / Ю.С. Бородич [и др.]. – Мн.: Выш. шк., 1991. – 365 с.
3. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Госстандарт, 1990. – 28 с.
4. Офицеров, Д.В., Старых, В.А. Программирование в интегрированной среде Турбо-Паскаль. – Мн.: Беларусь, 1992. – 240 с.

Министерство образования Республики Беларусь
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Электропривод и автоматизация промышленных установок и технологических комплексов»

Информатика

**Лабораторный практикум
для студентов специальности 1-53 01 05 «Автоматизированные электроприводы»**

Учебное электронное издание

Минск БНТУ 2009

Библиографическое описание:
Информатика [Электронный ресурс] : лабораторный практикум для студентов специальности 1-53 01 05 "Автоматизированные электроприводы" / сост. Ковальчук И. Г. ; Белорусский национальный технический университет, Кафедра "Электропривод и автоматизация промышленных установок и технологических комплексов".- БНТУ, 2009.
УДК: 004 (076.5)
Дата публикации: 2009
URI: http://rep.bntu.by/handle/data/797
Дата добавления: 2011-12-16

Лабораторный практикум Часть 2
Основы программирования

МОДУЛЬ М1 – «АЛГОРИТМИЗАЦИЯ ЗАДАЧ»

Лабораторная работа № 1
Алгоритмизация задач

Цель работы: *Ознакомление с методикой программирования и решения инженерных задач на персональном компьютере; изучение типовых алгоритмов, встречающихся при решении инженерных задач; приобретение практических навыков по разработке алгоритмов линейных, разветвляющихся и циклических вычислительных процессов в виде блок-схем (согласно ГОСТу 19.701-90).*

Постановка задачи

По варианту условия, определяемому номером бригады (из двух-трех студентов), разработать блок-схемы алгоритмов решения следующих задач (табл. 1.1):

Таблица 1.1

№ варианта	Условия		
	Линейная структура	Разветвляющаяся структура	Циклическая структура
1	2	3	4
1.	Вычислить объём усечённого конуса: $V = \frac{1}{3}\pi h\left(R^2 + r^2 + Rr\right)$	Вычислить значение: $\omega_1 = \begin{cases} e_1^{2t} + \cos(xt_2) & \text{при } t_1 < \sqrt{t_2 + x}; \\ \ln(xt_1)\cdot\sqrt{\sin t_2} & \text{при } t_1 = \sqrt{t_2 + x}; \\ xt_1 + \operatorname{tg} t_2 & \text{при } t_1 > \sqrt{t_2 + x}. \end{cases}$	Вычислить значения: $\beta_i = \frac{ax_i^2 + e_i^{2z}}{t\cdot\sqrt{\sin z_i}}$; $i = \overline{1,n};\quad n \le 4$
2.	Вычислить значение функции: $y = ae^{-bx} + \sin(\omega t)$	Вычислить значение: $\beta_2 = \begin{cases} 3q + \sqrt{\lvert x^3\cdot\sin z\rvert} & \text{при } \sin z < q; \\ \sqrt{aq} + e^{\alpha z} & \text{при } \sin z = q; \\ \lvert\alpha^5\rvert\cdot\ln\sqrt{\ln\cos z} & \text{при } \sin z > q. \end{cases}$	Вычислить значения: $\alpha_i = \frac{z_i + \sqrt[3]{q^2}}{b\cdot\ln^2\omega_i}$; $i = \overline{1,n};\quad n \le 5$

1	2	3	4
3.	Вычислить площадь треугольника: $S=\sqrt{p(p-a)(p-b)(p-c)}$ где $p=\frac{a+b+c}{2}$	Вычислить значение: $\gamma_1=\begin{cases}\alpha^3+\sqrt{\cos y^2} & \text{при } y<\ln\beta;\\ \lg(x+\omega)^2+\lvert y^3\rvert & \text{при } y=\ln\beta;\\ \sqrt{\operatorname{tg} y^5+x\omega} & \text{при } y>\ln\beta.\end{cases}$	Вычислить значения: $\gamma_i=\frac{3q_i\sqrt{y_i^2+\cos z}}{2\operatorname{tg}^2 b+q_i}$; $i=\overline{1,n};\quad n\le 5$
4.	Вычислить площадь правильного n-угольника: $S=\frac{1}{2}nR^2\sin\alpha$	Вычислить значение: $\tau_5=\begin{cases}e_1^{2T}+\cos(xr_2) & \text{при } r_1<r_2x;\\ \ln(xr_1)^2+\sin r_2 & \text{при } r_1=r_2x;\\ xr_1+\operatorname{tg} r_2 & \text{при } r_1>r_2x.\end{cases}$	Вычислить значения: $\omega_i=\frac{\arcsin(z_i^2)+z_1}{\cos z_1+\lvert z_1^5\rvert}$; $i=\overline{1,n};\quad n\le 5$
5.	Вычислить площадь сектора с углом α^0: $S_c=\pi R^2\frac{\alpha}{360}$	Вычислить значение: $\sigma_6=\begin{cases}3\lvert q+y^3+\sin z\rvert & \text{при } \sin z<q;\\ \sqrt{aq}+e^{\alpha z} & \text{при } \sin z=q;\\ \lvert\beta^3\rvert\ln\cos z & \text{при } \sin z>q.\end{cases}$	Вычислить значения: $\omega_i=\frac{z_i+\sin q^2}{t\cdot\ln^2\alpha_i}$; $i=\overline{1,n};\quad n\le 5$
6.	Вычислить площадь поверхности цилиндра: $S_п=2\pi R(H+R)$	Вычислить значение: $R_7=\begin{cases}\alpha^3+\cos y^3 & \text{при } y<\ln\alpha;\\ \lg(x+\omega)^2+\lvert y^5\rvert & \text{при } y=\ln\alpha;\\ \operatorname{tg} y^3+e^{x\omega} & \text{при } y>\ln\alpha.\end{cases}$	Вычислить значения: $t_i=\frac{3g_ix_i^2+\cos z}{2\operatorname{tg}^2(t+g_i)}$; $i=\overline{1,n};\quad n\le 5$
7.	Вычислить длину хорды сегмента с центральным углом α: $L=2R\cdot\sin\frac{\alpha}{2}$	Вычислить значение: $f_8=\begin{cases}\beta^3+\sin z^2 & \text{при } z<\ln\beta;\\ \lg(q+\omega)^2+\lvert z^7\rvert & \text{при } z=\ln\beta;\\ \operatorname{tg} z^3+e^{q\omega} & \text{при } z>\ln\beta.\end{cases}$	Вычислить значения: $t_i=\frac{x_iy_i^2+\ln d}{2\operatorname{tg}^2(t+x_i)}$; $i=\overline{1,n};\quad n\le 5$
8.	Вычислить: $y=x^a+e^{\sin(\omega t)}$	Вычислить значение: $k_8=\begin{cases}\ln\lvert f\rvert+\lvert g\rvert & \text{при } \lvert f\cdot g\rvert>0;\\ e^{f+g} & \text{при } \lvert f\cdot g\rvert<0;\\ f+g & \text{при } \lvert f\cdot g\rvert=0.\end{cases}$	Вычислить значения: $t_i=\frac{z_i+\sin^2 g}{2\operatorname{tg}(z_i+g)}$; $i=\overline{1,n};\quad n\le 5$

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическим материалом (см. Приложение 1) .
2. Построение в отчете по лабораторной работе блок-схем алгоритмов согласно варианту задания.

Контрольные вопросы

1. Дать определение алгоритма.
2. Что Вы понимаете под термином «алгоритмизация задачи»?
3. Может ли задача иметь несколько алгоритмов решения?

4. На каких принципах основано построение схем алгоритмов?

5. Каковы особенности построения схем алгоритмов линейных, разветвляющихся и циклических вычислительных процессов?

Содержание отчета

Отчет по выполненной работе оформляется на основании варианта задания и должен содержать следующие сведения:

1. Номер и наименование лабораторной работы.
2. Цель работы.
3. Постановку задачи.
4. Решение задачи в виде блок-схем алгоритмов согласно варианту задания.
5. Выводы по работе.

Приложение 1 (к модулю М1)

Теоретические сведения к лабораторной работе № 1- «Алгоритмизация задач»

1. МЕТОДИКА ПРОГРАММИРОВАНИЯ И РЕШЕНИЯ ИНЖЕНЕРНЫХ ЗАДАЧ НА ПК

Методика подготовки (программирования) и решения инженерных задач на ПК включает в себя выполнение следующих основных этапов:

1. Математическую формулировку задачи.
2. Выбор метода вычисления.
3. Разработку схемы алгоритма решения задачи.
4. Написание программы на алгоритмическом языке.
5. Ввод текста программы в память ПК и отладку ее.
6. Ввод исходных данных и выполнение вычислений по программе.

На ***первом этапе*** условие задачи записывается в виде последовательности формул, которые в дальнейшем будут использоваться в вычислениях.

На ***втором этапе*** выбирается такой метод решения математически сформулированной задачи, который позволяет свести поиск результата к выполнению последовательности элементарных математических операций. Для большинства практических задач разработаны численные методы, обеспечивающие приближенное решение с требуемой точностью.

На ***третьем этапе*** на основе выбранного метода разрабатывается схема алгоритма решения задачи (см. пункты 2 и 3).

На ***четвертом этапе,*** используя схему алгоритма, составляют программу решения задачи на определенном алгоритмическом языке в виде последовательности соответствующих операторов.

На ***пятом этапе*** с помощью системы программирования вводят программу в память ПК, транслируют, редактируют, проверяют правильность ее написания и ввода.

На ***шестом этапе*** вводят исходные данные и выполняют вычисления по программе.

2. РАЗРАБОТКА АЛГОРИТМА РЕШАЕМОЙ ЗАДАЧИ

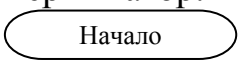
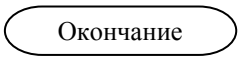
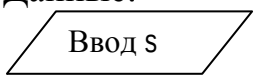
Под ***алгоритмом*** понимается определенная последовательность предписаний (инструкций, действий, операций) по переработке исходных и промежуточных данных в результат решения задачи, т.е. алгоритм представляет собой общую схему решения конкретной задачи. При

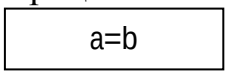
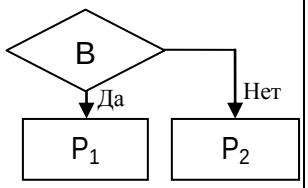
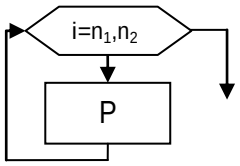
разработке алгоритмов сложных задач выделяют составные части, различные по назначению. Каждая часть может представлять собой отдельный алгоритм. Иногда этот алгоритм можно разбить на ряд еще более простых алгоритмов и т.д. Глубина разработки алгоритма (детализация) зависит от наличия разработанных ранее стандартных алгоритмов решения типовых задач и от условий решаемой задачи. В итоге алгоритм решения задачи сводится к определенной последовательности таких действий: начало решения, ввод данных, вычисления по формулам, обращение к подпрограммам (другим мини-программам), проверка условий, влияющих на ход вычислительного процесса, организация повторяющихся участков вычислений, вывод данных (исходных, промежуточных, результатов решения задачи), конец решения.

Из различных способов описания алгоритмов (словесный, операторный, схемный) наиболее распространен (из-за большей наглядности, облегчающий затем написание по алгоритму самой программы на каком-нибудь алгоритмическом языке) ***схемный способ,*** при котором алгоритм представляется в виде ***символов*** (блоков), выполняющих определенные действия, и связей между ними с помощью ***линий,*** указывающих очередность выполнения этих символов при вычислениях. Форму, размеры, наименование символов и выполняемые ими функции, а также правила построения схем алгоритмов определяет ГОСТ 19.701-90 [5]. В табл. 1.2 приведены основные, чаще употребляемые символы схем алгоритмов и форма их записи (программирование в общем формате) на алгоритмическом языке Паскаль.

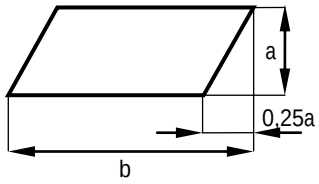
Таблица 1.2

Некоторые символы и форма их записи на Паскале

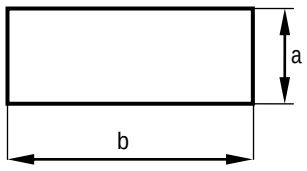
Название и форма обозначения символа	Выполняемая функция	Общая форма записи символа на Паскале	Примечание
1	2	3	4
Терминатор:  или 	Начало программы Конец программы	Заголовок программы, описательные разделы и операторные скобки begin..end. раздела операторов	
Данные:  или 	Ввод данных Вывод данных	Read (s) или Readln (s) Write (s) или Writeln (s)	s – список вводимых или выводимых переменных

1	2	3	4
Процесс 	Выполнение вычислений по формуле	a:=b	a – имя переменной b – арифметическая или логическая операция
Решение: 	Выбор дальнейших действий в зависимости от значения B	if b then p1 или if b then p1 else p2	B – логическое выражение; P_1 и P_2 – операторы: простые, составные или вложенные
Подготовка: 	Начало цикла с известным числом повторений и шагом изменения параметра цикла i, равным 1.	for i:=n1 to n2 do p; или for i:=n1 downto n2 do p; (если n1> n2)	i – параметр цикла; n_1 и n_2 – начальное и конечное значения i; P – тело цикла (простой, составной или вложенный оператор)

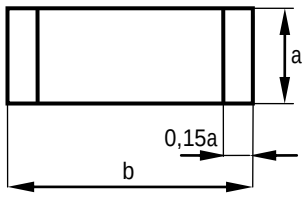
Приведем условные обозначения чаще используемых символов (блоков) и некоторые правила выполнения схем алгоритмов из ГОСТа 19.701-90.



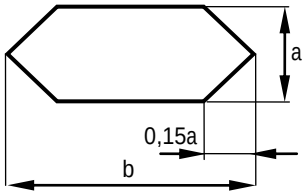
1. ***Данные*** Символ отображает данные, носитель данных не определен (символы данных во многих случаях представляют способы ввода/вывода данных).



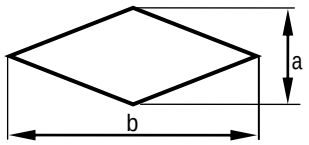
2. ***Процесс***. Символ отображает обработку данных (вычисления по формулам).



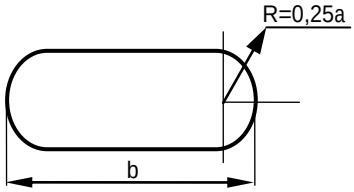
3. ***Предопределенный процесс.*** Символ отображает подпрограмму, модуль.



4. ***Подготовка***. Символ отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию (начало цикла с заданным числом повторения).

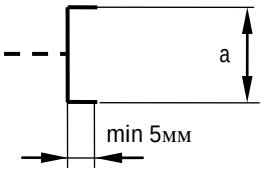


5. ***Решение***. Символ отображает выбор направлений выполнения алгоритма в зависимости от некоторых условий.



6. ***Терминатор***. Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы), прерывание (остановка, пуск).

7. ***Линия***. Поток данных или управления.



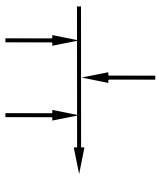
8. ***Комментарий***. Символ отображает связь между элементом схемы и пояснением (используется, если текст внутри символа не помещается).

Символы (блоки) в схеме алгоритма должны быть расположены равномерно, быть по возможности одного размера; *не должны изменяться углы и другие параметры, влияющие на соответствующую форму символов* (размер *а* выбирается из ряда 10, 15, 20, ... мм, а размер b = 1,5*а* или 2*а* согласно ГОСТ 19.003-80).

Помещаемый внутри символа текст записывается слева направо и сверху вниз независимо от направления потока выполнения символов. Текст должен быть минимальным и достаточным для понимания выполняемых функций данного символа; если объем текста превышает размеры символа, то следует использовать символ комментария.

Для большей ясности на линиях потока в схеме алгоритма используются стрелки. Направление линий потока (данных или управления) слева направо и сверху вниз считается ***стандартным*** и стрелкой может не обозначаться. Линии, отличные от стандартного направления или имеющие изломы, должны обязательно обозначаться стрелками.

В схемах алгоритмов следует избегать пересечения линий. Пересекающиеся линии не имеют логической связи между собой, поэтому изменение направления в точках пересечения не допускается. Две или более входящих линий могут объединяться в одну исходящую линию, при этом место объединения должно быть смещено:



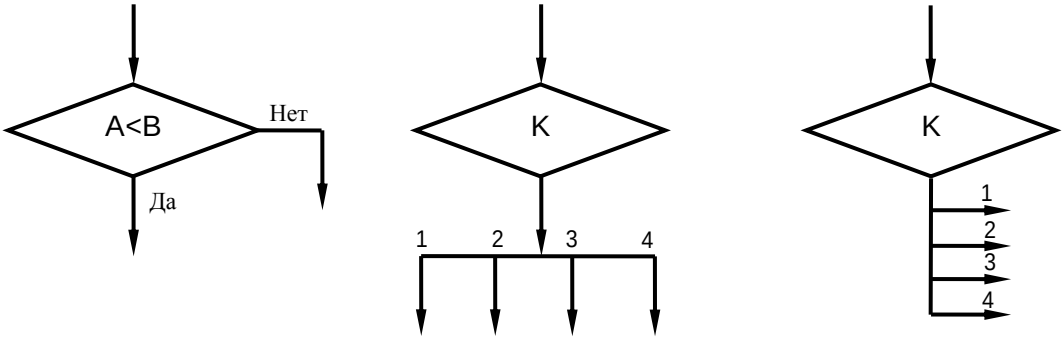
Линии в схемах алгоритмов должны подходить к символу либо слева, либо сверху, а исходить либо справа, либо снизу (по центру символа).

Несколько выходов из символа можно показывать:

-несколькими линиями от данного символа к другим символам;

-одной линией от данного символа, которая затем разветвляется на соответствующее число линий. Каждый выход из символа должен сопровождаться соответствующим значением условия разветвления.

Пример.



Символы в схеме алгоритма могут помечаться идентификаторами, например, для ссылки на них в других частях документации. Идентификатор располагается слева над символом.

При разработке схемы алгоритма и написании Паскаль-программы целесообразно использовать некоторые простые приемы, позволяющие уменьшить затраты времени на вычисления (при решении задачи по программе):

- вместо операции возведения в целую степень (для низких степеней) использовать операцию умножения (например, $a^3 = a \bullet a \bullet a$);
- выражение, вычисляемое в программе многократно с одними и теми же данными, необходимо вычислять один раз, присваивая его значение промежуточной переменной, используемой затем в дальнейших вычислениях как известной;
- в качестве границ параметров цикла использовать не выражения, а промежуточные переменные, вычисляемые до начала выполнения цикла;
- все повторяющиеся внутри цикла вычисления с одинаковыми данными выносить за пределы цикла, выполняя их один раз до начала цикла.

Рассмотрим в следующем разделе некоторые характерные приемы алгоритмизации типовых задач на конкретных примерах.

2. СТАНДАРТНЫЕ СХЕМЫ АЛГОРИТМОВ

2.1. Линейный алгоритм

Линейным называется алгоритм, в котором все блоки выполняются последовательно один за другим. При разработке алгоритма решаемой задачи следует выбирать наилучший вариант в смысле некоторого критерия, в качестве которого используют либо оценку точности решения задачи, либо затраты времени на решение, либо некоторые интегральные критерии.

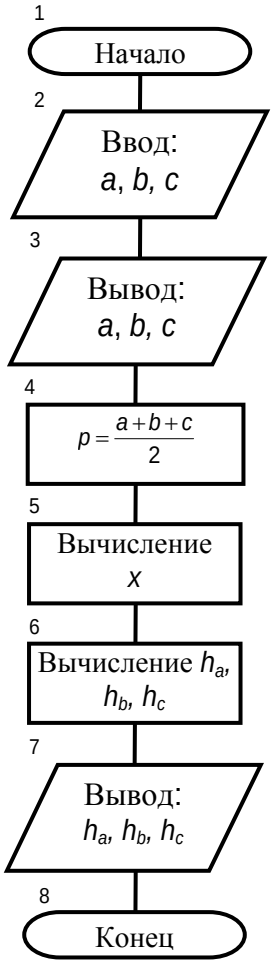


Рис. 1.1

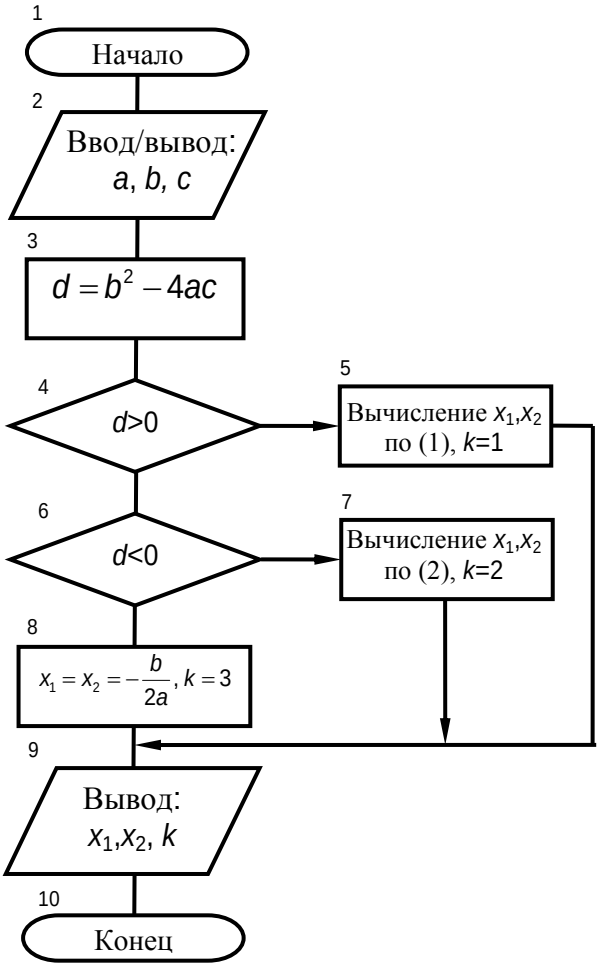


Рис. 1.2

Пример.

Разработаем линейный алгоритм для определения высот: h_a, h_b , h_c треугольника по заданным длинам его сторон *a, b, c:*

$$h_a = 2/a\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)}\ ;$$

$$h_b = 2/b\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)}\ ;$$

$$h_c = 2/c\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)}\ ,$$

где $p = (a + b + c)/2$.

При решении данной задачи, с целью уменьшения затрат на вычисления, определять высоты будем не по приведенным выше формулам, а с использованием промежуточной переменной:

$$x = 2\sqrt{p\cdot(p-a)\cdot(p-b)\cdot(p-c)}\ .$$

Тогда $h_a = x/a;\ h_b = x/b;\ h_c = x/c$ и блок-схема алгоритма решения данной задачи будет иметь вид, представленный на рис. 1.1.

3.2. Разветвляющийся алгоритм

На практике часто, в зависимости от исходных данных или от результатов промежуточных вычислений, бывает необходимо организовать дальнейшие вычисления по одним или другим формулам, т.е. вычислительный процесс в зависимости от выполнения некоторых логических условий должен идти по нескольким направлениям (ветвям). Алгоритм такого вычислительного процесса называют ***разветвляющимся***.

Пример.

Разработаем разветвляющийся алгоритм вычисления корней квадратного уравнения

$ax^2 + b\,x + c = 0$ с вещественными коэффициентами $a,\ b,\ c$.

Алгоритм решения данной задачи (рис. 1.2) содержит три ветви (в зависимости от значения дискриминанта $d = b^2 - 4ac$):

1. Если $d > 0$, то корни вещественные и определяются по формулам:

$$x_1 = (-b + \sqrt{d}\ ) / (2a);\ x_2 = (-b - \sqrt{d}\ ) / (2a). \qquad (1)$$

2. Если $d < 0$, то корни комплексно-сопряженные вида $x_1 \pm jx_2$, а их вещественная (x_1) и мнимая (x_2) части вычисляются по формулам:

$$x_1 = -\,b/(2a);\ x_2 = \sqrt{-d}\,/(2a). \qquad (2)$$

3. Если $d = 0$, то корни вещественные и равные между собой: $x_1 = x_2 = -\,b/(2a)$.

В данном алгоритме вид корней при выводе результатов вычисления определяется значением переменной ***к:*** если ***к*** = 1 , то корни вещественные и разные; если ***к*** = 2, то корни комплексно-сопряженные; если ***к*** = 3, то корни вещественные и равные между собой.

3.3. Циклические алгоритмы

Циклическим алгоритмом называют вычислительный процесс, в котором решение задачи или ее части сводится к многократному вычислению по одним и тем же формулам при различных значениях входящих в них некоторых переменных. Многократно повторяющиеся участки такого вычислительного процесса называются ***циклами***.

Различают циклы с заданным (известным) и неизвестным числом повторений. К последним относятся так называемые ***итерационные циклы,*** характеризующиеся последовательным приближением к искомому значению результата с заданной точностью (например, при вычислении суммы членов сходящегося бесконечного ряда чисел).

Переменную, изменяющуюся в цикле, называют ***параметром цикла.***

В одном цикле могут быть один или несколько параметров. *Цикл с несколькими одновременно изменяющимися параметрами* организуется по схеме построения цикла с одним каким-либо параметром, а для остальных параметров перед началом цикла задаются их начальные значения, а внутри цикла вычисляются их текущие значения для очередных повторений цикла.

Пример 1.

Составим схему алгоритма по вычислению функции S, равной сумме членов y_i, т.е.

$$S = \sum_{i=1}^{20} y_i, \quad y_i = \frac{x_i^2}{i}; \quad \textbf{\textit{x}}_i\textbf{- элементы массива (}\textbf{\textit{x}}_1\textbf{ , }\textbf{\textit{x}}_2\textbf{,..., }\textbf{\textit{x}}_{20}\textbf{).}$$

Эта задача относится к циклическому вычислительному процессу с заданным числом повторения цикла по накапливанию суммы членов y_i .

Схема алгоритма решения данного примера представлена на рис. 1.3, в котором блок 3 задает начальное значение суммы S перед циклом, а блок 5 вычисляет значение слагаемых y_i и накапливает искомую сумму.

Примечание. Алгоритм по вычислению функции *P,* равной

произведению членов y_i , т.е.

$$P = \prod_{i=1}^{20} \frac{x_i^2}{i}$$

аналогичен алгоритму на рис. 3 с той лишь разницей, что для накапливания произведения будет использоваться в блоке 5 формула $P = P \bullet y_i$, а начальное значение произведения *P* в блоке 3 должно быть равно единице.

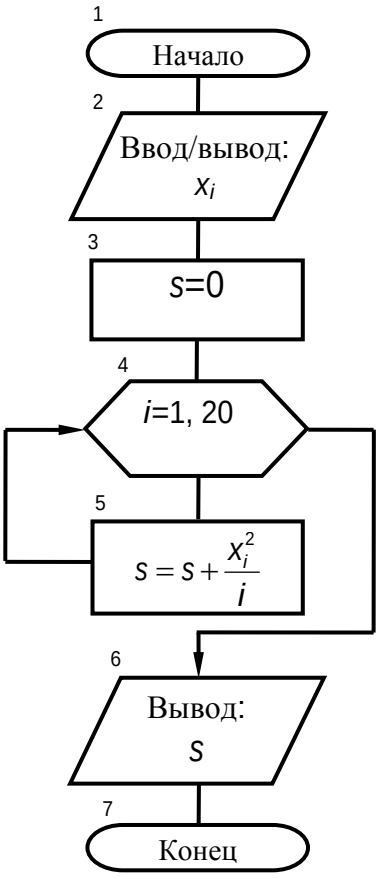


Рис. 1.3

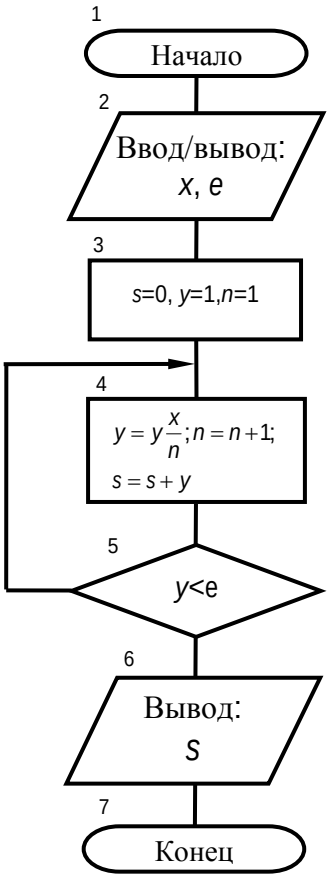


Рис. 1.4

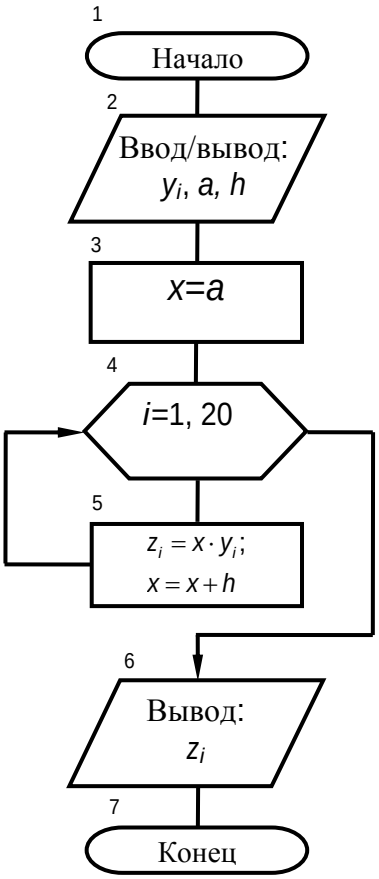


Рис. 1.5

.

Пример 2.

Разработаем алгоритм по вычислению суммы членов сходящегося бесконечного ряда чисел:

$$\mathbf{S= x + x^2/2! + x^3/3! + ...} = \sum_{n=1}^{\infty} \frac{x_n}{n!}$$

с точностью до члена ряда, меньшего ***e.***

Здесь имеет место итерационный цикл, так как заранее не известно, при каком ***n*** выполняется условие $(\mathbf{x}^n/n!) < \boldsymbol{e}$.

Сравнивая два соседних члена ряда, видим, что $y_n / y_{n-1} = x / n$.

Поэтому для уменьшения затрат времени на вычисление текущего члена ряда целесообразно в цикле использовать *рекуррентную* формулу: $y_n = y_{n-1} \bullet x / n$, т.е. формулу, использующую при вычислении члена y_n уже вычисленное значение предыдущего члена y_{n-1} . А чтобы использовать эту формулу для вычисления первого члена ряда $y_1 = y_0 \bullet x / 1$, необходимо положить начальное значение $y_0 = 1$. Тогда схема алгоритма решения данного примера будет иметь вид, представленный на рис. 1.4.

Пример 3.

Разработаем алгоритм по вычислению функции $z_i = x \cdot y_1$, если x изменяется одновременно с i от начального значения a с шагом h, а y_t - элементы массива $(y_1, y_2, \ldots, y_{20})$.

Здесь в цикле, повторяемом 20 раз, изменяются одновременно 2 параметра: простая переменная x и i - индекс массива y. Схема алгоритма решения данной задачи представлена на рис. 1.5, где блок 4 задает закон изменения параметра i , блок 3 (перед циклом) - начальное значение параметра x, а блок 5 (внутри цикла) вычисляет (кроме z_i) новое значение параметра x для очередного выполнения цикла.

Пример 4.

Разработаем алгоритм по вычислению суммы положительных элементов каждой строки матрицы $a\ (10 \bullet 15)$.

Для вычисления суммы элементов одной строки заданной матрицы необходимо организовать цикл с целью перебора всех элементов строки, поэтому параметром этого цикла следует выбрать номер столбца $к$. Перед циклом нужно задать начальное значение суммы $S_t = 0$.

Для вычисления суммы положительных элементов каждой строки матрицы необходимо организовать другой цикл (с параметром номера строки i), охватывающий первый цикл. Здесь мы имеем алгоритм со структурой вложенного цикла. ***Вложенным*** называется цикл, содержащий внутри себя один или несколько других циклов, при этом цикл, охватывающий другие циклы, называется ***внешним***, а остальные циклы - ***внутренними.*** Параметры этих циклов изменяются не одновременно, так как при одном каком-либо значении параметра внешнего цикла параметр внутреннего цикла принимает по очереди все свои значения. Схема алгоритма данного примера приведена на рис. 1.6.

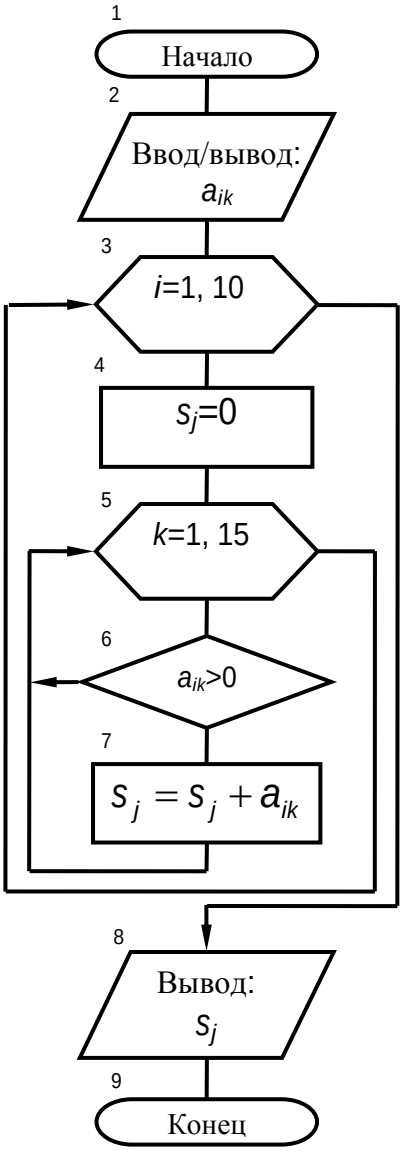


Рис. 1.6

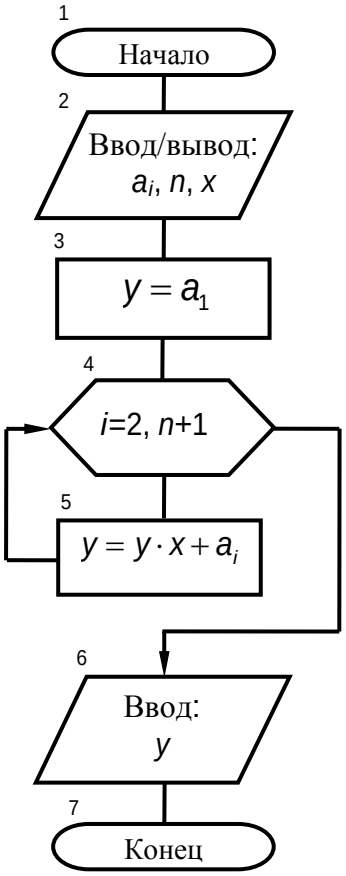


Рис. 1.7

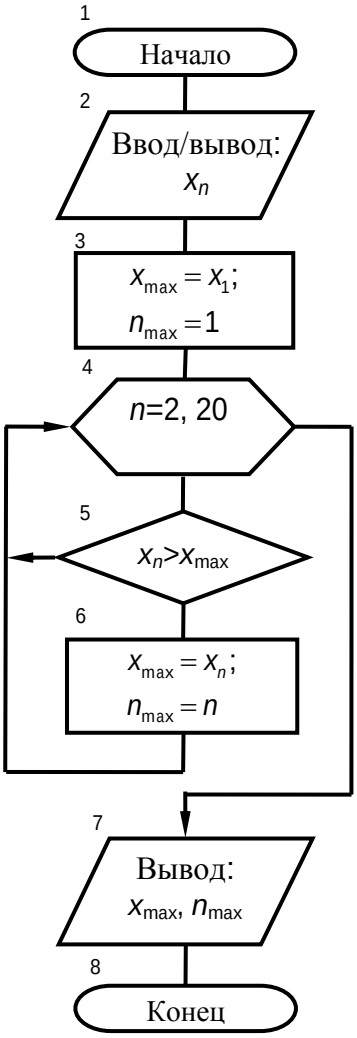


Рис. 1.8

3.4. Вычисление полинома

Для вычисления полинома n-й степени

$$y = a_1x^n + a_2x^{n-1} + \ldots + a_nx + a_{n+1}$$

удобно использовать формулу Горнера:

$$y = ((a_1x + a_2)x + \ldots + a_n)x + a_{n+1}.$$

Если выражение в самых внутренних скобках обозначить y_i , то значение выражения в следующих скобках можно вычислять по рекуррентной формуле: $y_{i+1} = y_i x + a_{i+1}$, а значение полинома y получим после повторения этого процесса в цикле n раз. Начальное значение y_i целесообразно взять равным a_1 , а цикл начать с i = 2. Тогда алгоритм для вычисления полинома *n-й* степени по формуле Горнера можно представить в виде схемы рис. 1.7. Все коэффициенты полинома сводятся в массив из n+1 элементов, при этом следует иметь в виду, что если полином не содержит членов с

некоторыми степенями x, то в массиве коэффициентов a_i на соответствующих местах необходимо помещать коэффициенты, равные нулю.

3.5. Нахождение наибольшего или наименьшего значения функции

Нахождение наибольшего (или наименьшего) значения функции $y = f[x]$ выполняется по циклу, в котором вычисляется текущее значение функции и сравнивается с наибольшим (или с наименьшим) из всех предыдущих значений этой функции. Если текущее значение функции оказывается больше из наибольшего (или меньше из наименьшего) из предыдущих значений, то его считают новым наибольшим (или наименьшим) значением.

При этом в качестве начального значения y_{max} необходимо взять очень малое значение (например, число $-1 \cdot 10^{10}$), а в качестве начального значения y_{min} - очень большое число (например, $+1 \cdot 10^{10}$).

Пример.

Разработаем алгоритм по нахождению наибольшего элемента массива x_i (x_1, x_2 ,..., x_{20}) и его порядкового номера.

Здесь нет необходимости вычислять сравниваемые значения, так как они уже имеются в массиве x_i . Поэтому в качестве начальных значений x_{max} и номера n_{max} примем $x_{max} = x_1$ и $n_{max} = 1$ и сравнение будем производить по ц иклу, начиная со второго элемента массива x_i.

Схема алгоритма решения данного примера представлена на рис. 1.8.

Литература

1. Вальвачев, А.Н., Крисевич, В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС. – Мн.: Выш. шк., 1989. – 223 с.
2. Паскаль для персональных компьютеров / Ю.С. Бородич [и др.]. – Мн.: Выш. шк., 1991. – 365 с.
3. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Госстандарт, 1990. – 28 с.
4. Офицеров, Д.В., Старых, В.А. Программирование в интегрированной среде Турбо-Паскаль. – Мн.: Беларусь, 1992. – 240 с.
5. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем.

   Условные обозначения и правила выполнения. – М.: Госстандарт,1990.-28с.

МОДУЛЬ М2 – «БАЗОВЫЕ ЭЛЕМЕНТЫ АЛГОРИТМИЧЕСКОГО ЯЗЫКА ПАСКАЛЬ »

Лабораторная работа № 2
Запись чисел и переменных на языке Паскаль

Цель работы: *Приобретение практических навыков записи на языке Паскаль чисел и переменных.*

Постановка задачи

Записать на Паскале по варианту условия, определяемому номером бригады, следующие данные (табл. 2.1 и табл. 2.2):

Таблица 2.1

Константы

№ пп	Варианты							
	1	2	3	4	5	6	7	8
1	12,3	13,6	21,7	8,5	9,8	-6,7	7,34	8,19
2	-0,95	0,75	-0,85	-0,53	-0,79	0,24	-0,94	-0,37
3	5	6	3	4	5	6	7	8
4	4,0	2,0	3,0	4,0	5,0	6,0	7,0	8,0
5	$1{,}3\cdot10^{5}$	$1{,}1\cdot10^{3}$	$1{,}4\cdot10^{2}$	$1{,}4\cdot10^{3}$	$1{,}7\cdot10^{5}$	$1{,}1\cdot10^{3}$	$1{,}5\cdot10^{4}$	$1{,}2\cdot10^{5}$
6	$-1{,}7\cdot10^{4}$	$-1{,}5\cdot10^{4}$	$-1{,}8\cdot10^{3}$	$-1{,}5\cdot10^{2}$	$-1{,}8\cdot10^{4}$	$-1{,}2\cdot10^{4}$	$-1{,}7\cdot10^{5}$	$-1{,}9\cdot10^{3}$
7	$-2{,}2\cdot10^{-3}$	$-0{,}7\cdot10^{-5}$	$-1{,}6\cdot10^{-4}$	$-1{,}7\cdot10^{-4}$	$-1{,}9\cdot10^{-3}$	$-1{,}3\cdot10^{-5}$	$-1{,}8\cdot10^{-6}$	$-2{,}7\cdot10^{-2}$

Таблица 2.2

Переменные

№ пп	Варианты							
	1	2	3	4	5	6	7	8
1	α	ω	β	φ	π	ω	Δ	ρ
2	β_{12}	α_{13}	γ_{15}	β_{01}	φ_{05}	ρ_{06}	τ_{07}	ω_{06}
3	q_{zi}	r_s	v_r	q_z	p_v	r_z	q_ω	t_z
4	nos_1	st_{02}	les_3	bar_4	rab_5	vod_6	var_7	nok_8
5	$z_{i(i=1,\bar{n})}$	$\nu_{i(i=1,\bar{n})}$	$\omega_{k(k=1,\bar{n})}$	$\tau_{i(i=1,\bar{n})}$	$\sigma_{i(i=1,\bar{n})}$	$\eta_{k(k=1,\bar{n})}$	$\mu_{k(k=1,\bar{n})}$	$\delta_{k(k=1,\bar{n})}$
6	$\omega_{ij(i=1,\bar{n})}$ $(j=1,\bar{m})$	$z_{ij(i=1,\bar{n})}$ $(j=1,\bar{m})$	$\varepsilon_{ik(i=1,\bar{n})}$ $(k=1,\bar{m})$	$\delta_{ij(i=1,\bar{n})}$ $(j=1,\bar{m})$	$\psi_{ij(i=1,\bar{n})}$ $(j=1,\bar{m})$	$\theta_{ik(i=1,\bar{n})}$ $(k=1,\bar{m})$	$\lambda_{ik(i=1,\bar{n})}$ $(k=1,\bar{m})$	$\beta_{ik(i=1,\bar{n})}$ $(k=1,\bar{m})$
7	astra (симв. пер.)	bntu (симв. пер.)	bgpa (симв. пер.)	bpi (симв. пер.)	Bar (симв. пер.)	tost (симв. пер.)	fitr (симв. пер.)	post (симв. пер.)

Константы, переменные и их запись на языке Паскаль удобнее представить в виде следующих таблиц (табл. 2.3):

Таблица 2.3

Константы

№ п/п	Обычная запись	Паскаль	Тип
1	17,5	17.5	Real
…			

Переменные

№ п/п	Обычная запись	Паскаль
1	a	Alf a1-
…		

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическими сведениями (см. Приложение 2, с. 4-7).
2. Оформление в отчете по лабораторной работе ответов на вопросы согласно варианту задания.

Контрольные вопросы

1. Данные каких типов на языке Паскаль Вам известны?
2. В чем отличие понятий «константа» и «переменная»?
3. Какие формы записи констант на Паскале Вам известны?
4. Что такое идентификаторы? Каковы правила их записи на Паскале?

Содержание отчета

Отчет по выполненной работе должен содержать следующие сведения:

1. Номер и наименование лабораторной работы.
2. Цель работы.
3. Постановку задачи.
4. Ответы на вопросы задания.
5. Выводы по работе.

Лабораторная работа № 3
Запись математических выражений на языке паскаль

Цель работы: *Приобретение практических навыков записи на языке Паскаль произвольных математических выражений.*

Постановка задачи

Записать на Паскале по варианту условия, определяемому номером бригады, следующие выражения (табл. 3.1):

Таблица 3.1

Варианты заданий

№ вариантов	№ п/п	выражения
1	2	3
1	1	$y_1 = \frac{3{,}3x_1 + 25e_1^2}{\left\|x_1^7\right\| \cdot \sqrt{a_1 + z_1}}$
	2	$z_1 = \frac{a_1 + b_1}{e_1^x + \sin\left(x_1^2\right)} + \ln^2 \cdot x_1^2$
	3	$y_1 = a_1 + \frac{b_1 \cdot x_1}{e^{\alpha_1 \beta_1}}$
	4	$\gamma_1 = 3\sqrt{\cos^2\left(\alpha_1 + \rho_1\right) + \arcsin\beta_1}$
	5	$q_1 = \frac{1}{2\pi}\sqrt{\operatorname{arctg}\frac{\alpha_1}{\beta_1}}$
2	1	$y_2 = \frac{3{,}8y_1 + 65e^{x_1^2}}{\left\|y_1^7\right\| \cdot \sqrt{a_1 + z_1}}$
	2	$z_1 = \frac{c_1 + a_1}{e_2^y + \sin\left(x_2^2\right)} + \ln^2 \cdot x_2^3$
	3	$y_2 = a_2 + \frac{d_2 \cdot w_2}{e^{\alpha_2 \beta_2}}$
	4	$w_2 = 3\sqrt{\cos^2\left(\alpha_2 + \rho_2\right) + \arccos\varepsilon_2}$
	5	$\gamma_2 = \frac{1}{2\pi}\sqrt[5]{\sin^3 \omega_2^3}$
3	1	$y_3 = \frac{7{,}3x_3^2 + 29x_3}{\left\|x_3^5\right\| \cdot \sqrt{a_3 + z_3}}$
	2	$z_3 = \frac{a_3 + b_3}{e^{\beta} + \sin\left(x_3^2\right)} + \ln^2 \cdot x_3^2$
	3	$y_3 = a_3 + \frac{b_3 \cdot x_3}{e^{\alpha_3 \beta_3}}$
	4	$\gamma_3 = 3\sqrt{\cos^2\left(\alpha_3 + \rho_3\right) + \operatorname{arctgs}_3}$
	5	$t_3 = \arcsin\varphi_3 + \sqrt{\left\|\gamma_3^5\right\|}$
4	1	$y_4 = \frac{3{,}8y_4^2 + 65e^{x_4^2}}{\left\|y_4^7\right\| \cdot \sqrt{\beta_4 + z_4}}$
	2	$z_4 = \frac{c_4 + a_4}{e_4^y + \sin\left(x_4^2\right)} + \ln^2 \cdot x_4^2$
	3	$y_4 = a_4 + \frac{d_4 \cdot w_4}{e^{\alpha_4 \beta_4}}$
	4	$w_4 = 3\sqrt{\operatorname{tg}^2\left(\alpha_4 + \rho_4\right) + \arccos\left(\varphi_2\right)}$

1	2	3
	5	$\rho_4 = \arcsin\frac{\alpha_4}{\beta_4\sqrt{\lvert\sigma_4\rvert}}$
5	1	$y_5 = \frac{3{,}8y_5^2 + 65e^{x_5^2}}{\lvert\varepsilon_5^2\rvert \cdot \sqrt{\alpha_5 + \beta_5}}$
	2	$z_5 = \frac{c_5 + a_5}{e_5^y + \sin\left(x_5^2\right)} + \ln^2 \cdot x_5^2$
	3	$y_5 = a_5 + \frac{d_5 \cdot w_5}{e^{\alpha_5\beta_5}}$
	4	$w_5 = 3\sqrt{\cos^2\left(\alpha_5 + \rho_5\right) + \operatorname{arctg}\left(\varphi_5\right)}$
	5	$t_5 = \arcsin\frac{x_5}{\beta_5} + e^z$
6	1	$y_6 = \frac{3{,}8\,f_6^{\,2} + 65e^{x_6^2}}{\lvert z_6^5\rvert \cdot \sqrt{\alpha + \varepsilon}}$
	2	$z_6 = \frac{c_6 + a_6}{e_6^y + \sin\left(x_6^2\right)} + \ln^2 \cdot x_6^3$
	3	$y_6 = a_6 + \frac{d_6 \cdot w_6}{e^{\alpha_6\beta_6}}$
	4	$w_6 = 3\sqrt{\cos^2\left(\alpha_6 + \rho_6\right) + \arcsin\left(\alpha_6\right)}$
	5	$r_6 = \arcsin\frac{\gamma_6}{\beta_6} + e^{\sqrt{\lvert\ln(x)\rvert}}$
7	1	$y_7 = \frac{13{,}8w_7^2 + 65x_7^3}{\lvert\varepsilon_7^5\rvert \cdot \sqrt{\alpha + \lambda}}$
	2	$z_7 = \frac{c_7 + a_7}{e_7^y + \sin\left(x_7^2\right)} + \ln^2 \cdot x_7$
	3	$y_7 = a_7 + \frac{d_7 \cdot \sin\left(w_2\right)}{e^{\alpha_7\beta_7}}$
	4	$w_7 = 3\sqrt{\cos^2\left(\alpha_7 + \rho_7\right) + \operatorname{arctg}\left(w_7\right)}$
	5	$q_7 = \arccos\frac{\gamma_7}{\beta_7} + e^{2h}$
8	1	$z_8 = \frac{9{,}7y_2^2 + 2{,}6x_8^2}{\lvert\delta_8^7\rvert \cdot \sqrt{a_8 + z_8}}$
	2	$z_8 = \frac{c_1 + a_1}{e_8^y + \sin\left(x_8^2\right)} + \ln^2 \cdot x_8^3$
	3	$y_8 = tg\left(a_8\right) + \frac{z_8 \cdot w_8}{e^{\alpha_8\beta_8}}$
	4	$w_8 = 3\sqrt{\sin^2\left(\alpha_8 + \rho_8\right) + \operatorname{arctg}\left(\varphi_8\right)}$

	5	$q_8 = \arccos\frac{x_8}{\beta_8} + e^{\sqrt{t}}$

Содержание лабораторной работы

Лабораторная работа включает:

4. Ознакомление с теоретическими сведениями (см. Приложение 2, с. 4-7).
5. Оформление в отчете по лабораторной работе ответов на вопросы согласно варианту задания.

Контрольные вопросы

1. Что представляет собой математическое выражение на языке Паскаль?
2. Что такое операнд? Какими знаками связаны между собой операнды в выражениях?
3. Какие математические операции на Паскале Вам известны?
4. Что такое стандартные функции Паскаля? Каковы правила их записи?
5. Каковы правила записи выражений на Паскале?
6. Что такое приоритет выполнения операций в выражениях Паскаля?

Содержание отчета

Отчет по выполненной работе должен содержать следующие сведения:

6. Номер и наименование лабораторной работы.
7. Цель работы.
8. Постановку задачи.
9. Ответы на вопросы задания.
10. Выводы по работе.

Приложение 2 (к модулю М2)

Теоретические сведения к лабораторным работам № 2- «Запись чисел и переменных на языке Паскаль» и № 3 - «Запись математических выражений на языке Паскаль»

1. Вводные сведения о Паскале и системе программирования Турбо-Паскаль

Алгоритмический язык Паскаль является одним из популярных и широко распространённых языков программирования инженерных задач. Впервые его описание было опубликовано в 1971г. создателем этого языка Никлаусом Виртом - профессором Цюрихского технологического института (Швейцария). Название язык получил в честь французского математика и философа Блеза Паскаля (1623-1662). Первый транслятор с языка Паскаль был разработан в 1973г. Этот язык лёгок в изучении и удобен для программирования (набор его операторов относительно мал), является наиболее совершенным по сравнению с Бейсиком, Фортраном и др. языками и в 1982г. утверждён в качестве международного стандарта.

В начале 80-х годов появилась первая версия языка Паскаль как составная часть системы программирования для ПК. В настоящее время одной из наиболее популярных систем программирования для ПК является Турбо-Паскаль (ТП), представляющая собой интегрированную среду и включающая: экранный редактор, компилятор,

редактор связей и отладчик. Интегрированность среды проявляется не только в единой концепции построения её составляющих частей, но и в связи их друг с другом. Так, при возникновении ошибки трансляции система ТП автоматически переходит в режим экранного редактирования и ставит курсор в точку возникновения ошибки. Аналогичные действия выполняются отладчиком при возникновении ошибки во время выполнения программы.

Система программирования ТП создана американской фирмой Borland International. В первой версии этой системы был объединён очень быстрый компилятор с редактором текста. В 1985г. на рынке ПК появилась версия 3.0 системы программирования ТП с компилятором стандартного Паскаля, которая благодаря простоте её использования получила широкое применение. В пакете ТП версии 4.0 было устранено большинство подвергавшихся критике ограничений компилятора и была повышена производительность системы программирования, что дало возможность разработки в этой системе крупных программных продуктов. В ТП версии 5.0 в среду программирования был встроен интегрированный отладчик, позволяющий повысить производительность труда программистов. В версии ТП 5.5 улучшены технические характеристики: наряду с внутренними улучшениями реализована концепция объектно-ориентированного программирования. В ТП версии 6.0 чисто теоретическая концепция объектно-ориентировочного программирования реализована практически с полным набором объектов, которые могут использоваться для решения прикладных задач. Кроме того, реализация системы меню приведена в соответствие со стандартом SAA (Turbo Vision); реализован текстовый редактор, встроенный в IDE (интегрированную инструментальную оболочку). В 1992 г. фирмой Borland International разработана была очередная версия 7.0 системы программирования ТП, в которой унаследованы преимущества предыдущей версии и произведены некоторые изменения и улучшения: появилась возможность выделять определённым цветом различные элементы исходного текста программы (ключевые слова, идентификаторы, числа и т.д.), улучшен компилятор (коды программ стали более эффективными), улучшен интерфейс пользователя и др.

2. Базовые элементы языка Паскаль

Алфавит и словарь. При записи алгоритма решаемой задачи на языке Паскаль используется конечный набор знаков (символов), образующих ***алфавит*** этого языка.

Он состоит из *прописных и строчных* ***букв*** *латинского алфавита*

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z ,

знака подчёркивания _

цифр 0 1 2 3 4 5 6 7 8 9

и *специальных символов*

+ плюс	{} фигурные скобки
- минус	. точка
* звёздочка	, запятая
/ дробная черта	: двоеточие
= равно	; точка с запятой
> больше	' апостроф
< меньше	# номер
[] квадратные скобки	$ номер денежной единицы
() круглые скобки	^ тильда

_ пробел

Комбинации специальных символов образуют *составные символы:*

:= присвоить <= меньше или равно

<> не равно >= больше или равно

.. диапазон значений (..) альтернатива квадратных скобок

(* *) альтернатива фигурных скобок.

В комментариях к программе и в символьных константах могут использоваться любые другие знаки, например буквы русского алфавита. Комментарий представляется следующей конструкцией:

{Любой текст} и может быть помещен в любом месте программы.

Неделимые последовательности знаков алфавита образуют *слова,* которые несут определённый смысл в программе и отделяются друг от друга *разделителями* (пробелом, символом конца строки, комментарием). Слова делятся на ключевые (зарезервированные) слова, стандартные идентификаторы и идентификаторы пользователя.

Ключевые слова имеют фиксированное написание и определённый смысл.

Идентификатором *называется последовательность букв и цифр, начинающаяся с буквы или знака подчёркивания.* Идентификаторы применяются для обозначения имён переменных, констант, функций и процедур (подпрограмм).

Стандартные идентификаторы - это обозначения заранее определённых разработчиками языка Паскаль типов данных, констант, процедур и функций. Например, стандартный идентификатор sin(x) вызывает функцию по вычислению синуса заданного угла **x.**

Идентификаторы пользователя применяются для обозначения меток, констант, переменных, процедур и функций, которые задаются самим пользователем. Например, дату удобнее обозначать идентификатором Data, чем просто буквой D или каким-либо другим символом. Существуют *общие правила написания идентификаторов пользователя:*

- идентификатор должен начинаться только с буквы или знака подчёркивания, исключение, составляют метки, которые могут начинаться также и с цифры;
- для обозначения идентификаторов используются только буквы, цифры и знак подчёркивания;
- между двумя идентификаторами должен быть, по крайней мере, один пробел;
- идентификаторы могут быть любой длины (до 126 символов), но сравнение их между собой производится по первым 63 символам;
- при написании идентификаторов можно использовать как прописные, так и строчные буквы, так как компилятор не делает различий между ними. Поэтому на практике для более простого чтения и понимания целесообразно, например, написать *NomerOtdela* (вместо *nomerotdela),* выделив прописными буквами каждую из двух смысловых частей этого идентификатора.

Константы и переменные. При решении любой задачи по программе используются конкретные данные, над которыми выполняются определённые действия для получения результата решения. В программе каждый элемент данных является или константой, или переменной.

Константами *называются элементы данных, значения которых заданы перед решением задачи и которые в процессе выполнения программы не изменяются. Формат их описания:*

Const идентификатор = значение константы;

Имеется ряд стандартных констант, к значениям которых можно обращаться без предварительного описания, например:

Maxint = 32767;

True = истинно;

False = ложно;

Pi =3,14159…

Переменная - *это именованный объект, который в процессе выполнения программы может принимать различные значения.*

Переменные имеют следующий формат описания:

Var идентификаторы: *тип;*

В Турбо-Паскале используются три вида констант:

* числовые (целые или вещественные);

* логические (или булевские);

* символьные и строковые.

Целые константы *— это положительные или отрицательные целые числа (без десятичной точки).* Знак «+» в положительных числах можно опускать.

Турбо-Паскаль позволяет использовать также шестнадцатеричные целые значения. При использовании шестнадцатеричной константы перед ее значением указывается знак доллара $. Например, $27 определяет десятичное число 39.

Вещественные константы могут быть представлены в двух формах: с фиксированной и плавающей точкой. *Константы с фиксированной точкой* - это числа, содержащие точку, разделяющую целую и дробную части. *Константы с плавающей точкой* - это числа, представленные с десятичным порядком m**Ep** (без пробелов), где m - мантисса (как целые, так и вещественные числа с фиксированной точкой); **E** - признак записи числа с десятичным порядком; **p** - порядок числа (только целые числа).

Пример.

Значение константы	Целая константа	Константа с фиксированной точкой	Константа с плавающей точкой
-257	-257	-257.0	-2.57E2
16,4	-	16.4	1.64E1
0,0032	-	0.0032	0.32E-2

Константы с фиксированной точкой обязательно должны содержать как целую, так и дробную часть: 2.0; 0.5; -13.0.

Логические константы - *это константы, которые принимают только два значения: True (истинно) или False (ложно).*

Символьные константы - *это какой-либо один символ, заключенный в апострофы: 'A', '1'.*

Строковые константы - *это ряд символов, заключенных в апострофы: '+9CL', 'A и B'.*

Кроме констант и переменных существуют ещё так называемые ***типизированные константы,*** которые являются промежуточными данными между константами и переменными. Слово *константа* означает, что типизированная константа описывается в разделе Const, а слово *типизированная* - что должен указываться ***тип,*** как у переменных. Формат описания их следующий;

Const идентификатор: ***тип*** = значение;

Типы данных. Константы, переменные, функции, выражения, которые используются в программе, относятся к определённому типу. ***Тип*** - это множество значений элементов программы и совокупность операций над ними. Например,

значения 1 и 5 относятся к целочисленному типу, над ними можно выполнять различные арифметические операции.

В Паскале типы данных делятся на *скалярные* (стандартные и пользовательские) и *структурированные* (содержащие различные комбинации скалярных типов).

К стандартным скалярным типам данных относятся: целочисленные, байтовые, вещественные, символьные и булевские типы.

Целочисленный тип данных включает все целые числа в диапазоне от -32768 до +32767. Для размещения в памяти значения переменной целочисленного типа требуется 2 байта. Формат описания целочисленных переменных следующий:

Var идентификаторы: integer;

Байтовый тип данных аналогичен целочисленному, но охватывает меньший диапазон значений: от нуля до 255. Переменная байтового типа занимает в памяти 1 байт и имеет следующий формат описания:

Var идентификаторы: byte;

Вещественный тип данных включает положительные и отрицательные числа от $1 * 10^{-38}$ до $1 * 10^{+38}$, при этом мантисса может содержать до 11 значащих цифр. Данные этого типа могут записываться с фиксированной (целая часть числа от дробной отделяется точкой) и плавающей точкой в виде **mE±p,** где m - мантисса, представляющая собой целое или дробное число с десятичной точкой; **E** означает «десять в степени»; **p** - порядок в виде двухзначного целого числа.

Переменная вещественного типа в памяти занимает 6 байт и имеет следующий формат описания:

Var идентификаторы: rea1;

Переменная *булевского типа* в памяти занимает 1 байт, может принимать одно из двух значений (констант): True (истина) или False (ложь) и имеет следующий формат описания:

Var идентификаторы: *boolean;*

Константы и переменные ***символьного (литерного) типа*** принимают одно из значений кодовой таблицы ПК. Переменная этого типа в памяти занимает 1 байт. Конкретные значения переменных и констант символьного типа записываются в апострофах. Например, 'A' представляет собой букву A, ';' - точку с запятой. Формат описания переменной символьного типа следующий:

Var идентификаторы: char;

К скалярным пользовательским типам данных относятся перечисляемый и интервальный типы. Данные этих типов в памяти занимают по 1 байту.

Перечисляемый тип данных задаётся перечислением всех значений в круглых скобках через запятую. Форматы описания их следующие:

1. **Type** имя типа = (значение 1,... , значение n);
   Var идентификаторы: имя типа;

или

2. Var идентификаторы: (значение 1, ... , значение n);

Пример:

```
Type Gaz = (C,O,N,F);
     MetalI = (Fe,Co,Na,Cu,Zn);
Var Gl,G2:Gaz;
  MetI,Met2:MetalI;
Ses : (Winter, Spring, Summer, Autumn);
```

В этом примере явно описаны 2 типа данных пользователя - Gaz и MetalI (перечислены некоторые газы и металлы периодической таблицы Д.И. Менделеева). Переменные Gl, G2 и MetI, Met2 могут принимать только одно из перечисленных значений. Третий тип перечисления (Ses) анонимный, так как не имеет имени типа. Он

задаётся перечислением значений переменных в разделе описаний Var (т.е. записан по второму формату).

Интервальный тип данных задаётся двумя граничными константами диапазона значений для данной переменной. Обе константы должны быть одного из стандартных типов, кроме вещественного, при этом значение первой константы должно быть меньше значения второй. Формат описания:

Type имя типа = константа 1 .. константа 2;
Var идентификаторы: имя типа;

Пример:

```
Type Dni=I ..31;
Var RabDni, VolnDni: Dni;
```

В этом примере переменные RabDni и VolnDni имеют интервальный тип Dni и могут принимать любые значения из диапазона 1 ..31. Выход из диапазона вызывает программное прерывание.

Границы диапазона можно задавать не значениями констант, а их именами:

```
Const Min= I; Max=31;
Type Dni=Min .. Max;
Var RabDni, VolnDni: Dni;
```

Структурированные типы данных (строки, массивы, записи, множества, файлы и указатели) представляют собой упорядоченные определённым образом совокупности скалярных переменных и характеризуются типом своих элементов. Приведём лишь краткую характеристику этих данных (данные типа массивов, которые часто используются в вычислительных алгоритмах инженерных задач, более подробно рассмотрены в лабораторной работе № 9).

Строка - *это последовательность символов, заключённая в апострофы.*

Массив - *это данные, состоящие из фиксированного количества элементов, имеющих один и тот же тип и объединённых под общим именем.*

Множество - *это набор выбранных по какому-то признаку (или группе признаков) объектов, которые можно рассматривать как единое целое.*

Запись - *это данные, состоящие из фиксированного числа элементов разного типа.*

Файл - *это данные, состоящие из последовательности элементов одного типа и одной длины.* Чаще всего элементами файла являются записи.

Указатель - *это структурированный тип данных, состоящий из неограниченного множества указывающих на однотипные элементы значений.* Используется при работе с динамическими структурами данных.

Стандартные арифметические функции. Они реализуют наиболее часто встречающиеся математические действия и операции. Приведём их описание и примеры использования, обозначив через **x** целочисленные и вещественные типы.

Abs(x) - вычисление абсолютной величины выражения **x**. Тип результата совпадает с типом параметра **x**.

```
Abs(4-6)=2
```

ArcTan(x) - вычисление угла, тангенс которого равен **x**; значение угла представляется в радианах в диапазоне от -Pi/2 до Pi/2. Результат имеет вещественный тип.

```
ArcTan(I)=Pi/4
```

Sin(x), Cos (x) - вычисление синуса или косинуса выражения **x**; **x** - в радианах; результат имеет вещественный тип.

```
Cos(60*Pi/180)=0.5
```

Exp(x) - вычисление экспоненты **x**, т.е. значение e в степени **x** (**e** - основание натурального логарифма, равное 2,718282). Результат имеет вещественный тип.

```
Exp(1)=2.718282
```

Frac(x) - вычисление дробной части выражения **x**; результат имеет вещественный тип.

Frac(0.25*11)=0.75

Int(x) - вычисление целой части **x**.

Int (123.448)=123

Int(-345.555)=-345

Sqr(x) - возведение в квадрат значения **x**. Тип результата совпадает с типом аргумента **x**.

Sqr(5)=25

Sqrt(x) - вычисление квадратного корня из **x**. Тип результата вещественный.

Sqrt(25)=5.0

Ln(x) - вычисление натурального логарифма по основанию **e**. Результат имеет вещественный тип.

Ln(3)=1.0986

Для вычисления логарифма с основанием **a** используется соотношение:

$\log_a(x)=\ln(x)/\ln(a)$

Для вычисления других тригонометрических функций следует использовать известные соотношения:

Tan(x)=sin(x)/cos(x) Ctg(x)=cos(x)/sin(x)

Csc(x)=l/sin(x) Sc(x)=l/cos(x)

$\text{ArcSin}(x)=\text{ArcTan}(x/(l-x^2))^{1/2}$

ArcCos(x)=Pi/2-ArcSin(x)

ArcCtg(x)=Pi/2-ArcTan(x)

В языке Паскаль нет операции возведения в степень. Поэтому эту операцию заменяют другими с применением стандартных функций Exp и Ln:

$X^a=\text{Exp}(a*\text{Ln}(x))$

Вычисление выражения $(-x)^n$, если n - целочисленная константа, организуют по циклу путём умножения (-x) само на себя n раз.

Выражения, операнды и операции. ***Выражение*** задаёт порядок выполнения действий над элементами данных и состоит из *операндов* (констант, переменных, обращений к функциям), *знаков операций* и *круглых скобок*. ***Операции*** определяют действия, которые необходимо выполнить над операндами. В простейшем случае выражение может состоять из одного операнда. Круглые скобки ставятся так же, как в обычных арифметических выражениях для управления порядком выполнения операций. Их использование также вполне приемлемо и полезно для более чёткого и понятного визуального определения порядка вычислений.

Операции подразделяются на арифметические, отношения, логические, строковые и др. Выражения соответственно бывают арифметические, отношения, логические, строковые и др. в зависимости от того, какого типа операнды и операции в них используются.

Операции могут быть *унарными* (относятся к одному операнду и записывается перед ним) и *бинарными* (выражают действия над двумя операндами и записываются между ними). Например, в выражении -A символ "-" является унарной операцией, а в выражении A-B - бинарной.

Под ***арифметическим выражением*** понимают совокупность констант, переменных и функций, объединённых знаками арифметических операций и круглыми скобками. Результатом вычисления арифметического выражения всегда является число. Арифметические операции языка Паскаль представлены в табл. 3.2.

Таблица 3.2

Арифметические операции

Знак	Операция	Выражение
+	Сложение	A+B
-	Вычитание	A-B
*	Умножение	A*B
/	Деление	A/B
Div	Целочисленное деление	A div B
Mod	Деление с остатком	A mod B
And	Арифметические «И»	A and B
Shr	Целочисленный сдвиг вправо	A shr B
Shl	Целочисленный сдвиг влево	A shl B
Or	Арифметическое «ИЛИ»	A or B
Xor	Исключающее «ИЛИ»	A xor B
-	Изменение знака	-A

Операции сложения (+), вычитания (-), умножения (*) и деления (/) выполняется так же, как и в обычных арифметических выражениях.

Целочисленное деление (div) отличается от обычной операции деления тем, что результатом является целая часть частного, при этом дробная часть отбрасывается. Результат целочисленного деления всегда равен нулю, если делимое меньше делителя.

Примеры.	Выражение	Результат
	10 div 3	3
	2 div 3	0

Деление по модулю (mod) восстанавливает остаток, полученный при выполнении целочисленного деления.

Примеры.	Выражение	Результат
	10 mod 3	1
	2 mod 3	2

В выражениях для *арифметических операций* and, shr, shl, or, xor операнды записываются в десятичной форме, а при выполнении операций предварительно переводятся в двоичную систему счисления. Результаты поразрядного вычисления затем представляются в десятичной форме.

Примеры.	Выражение	Результат
	12 and 22	4
	2 shl 7	256
	160 shr 2	40
	12 or 22	30
	12 xor 22	26

Операция (унарная) изменения знака восстанавливает значение операнда с противоположным знаком.

Примеры.	Выражение	Результат
	-(-256)	256
	-(+39)	-39

При написании арифметических выражений рекомендуется соблюдать следующие ограничения и правила:

1. Запрещено последовательное написание двух операций, поэтому запись A/-B запрещается, A/(-B) разрешается.
2. Приоритет операций в порядке убывания следующий:

1) *, /, div, mod, and, shl; shr;

2) +, -, or, hor.

3. Часть выражения, заключённая в круглые скобки, выполняется в первую очередь.

4. Операции одинакового приоритета в выражении выполняются последовательно слева направо.

Выражение отношения состоит из двух или более арифметических выражений, соединённых операциями отношения. Оно определяет истинность или ложность результата, который имеет логический (булевский) тип и принимает одно из двух значений: True (истина) или False (ложь). Операции отношения на Паскале записывается так: = (равно), <> (не равно), > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), in (принадлежность). Все операции являются бинарными.

При объединении в одном выражении арифметических операций и операций отношения первыми всегда выполняются арифметические операции. *Сравниваемые данные должны быть одинакового типа.*

Логическое выражение образуется из операндов логического типа и логических операций: not (логическое отрицание), and (логическое умножение, И), or (логическое сложение, ИЛИ), **хог** (исключающее ИЛИ). При этом операндами могут быть: логические константы, логические переменные, выражения отношения.

Старшинство логических операций в порядке убывания следующее: 1) not, 2) and, 3)or, xor. Приоритет логических операций выше операций отношения (в других алгоритмических языках наоборот).

При вычислении логического выражения, содержащего различные операции (арифметические, логические и отношения), выполнение каждой операции осуществляется с учётом её приоритета (табл.3.3).

Таблица 3.3

Порядок выполнения операций

Операция	Приоритет	Вид операции
Not	Первый (высший)	Унарная операция
*, /, div, mod, and, shl, shr	Второй	Операции типа умножения
+, - or, xor	Третий	Операции типа сложения
=, <>, >, <, <=, >=, in	Четвёртый (низший)	Операции отношения

Для выполнения операций не по старшинству применяются круглые скобки. Например, в выражении **(A<C)** and **(B**=D) операции отношения будут выполняться раньше, чем логическая операция and.

Результатом вычисления логического выражения является константа логического типа.

Выражения 1-5 (см. табл. 3.1) программируются с помощью операторов присваивания. Общий вид оператора присваивания следующий:

Идентификатор переменной := выражение;

Здесь *идентификатор переменной* – имя переменной, текущее значение которой заменяется новым значением, определяемым данным *выражением*.

Пример.

```
Y := Sqrt (x)+1;
b := M and N;
```

В операторе присваивания *идентификатор переменной* и *выражение* должны иметь один и тот же тип, кроме одного исключения: идентификатору переменной типа real разрешается присваивать выражение типа integer.

Литература

5. Вальвачев, А.Н., Крисевич, В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС. – Мн.: Выш. шк., 1989. – 223 с.
6. Паскаль для персональных компьютеров / Ю.С. Бородич [и др.]. – Мн.: Выш. шк., 1991. – 365 с.
7. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Госстандарт, 1990. – 28 с.
8. Офицеров, Д.В., Старых, В.А. Программирование в интегрированной среде Турбо-Паскаль. – Мн.: Беларусь, 1992. – 240 с.

МОДУЛЬ М3 – «ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ»

Лабораторная работа № 4
Ввод-вывод данных на языке паскаль

Цель работы: *Приобретение практических навыков организации ввода-вывода данных на языке Паскаль.*

Постановка задачи

Осуществить ввод-вывод данных на Паскале по варианту условия, определяемому номером бригады ,(табл. 4.1). При этом предусмотреть для данных колонки Ввода-вывода использование операторов Read-Write, а для данных колонки Вывода - использование операторов **Присваивания** и Writeln.

Таблица 4.1

Варианты заданий

№ вариантов	Условия	
	Ввода/вывода	Вывода
1	2	3
1	$\alpha_1 = 1{,}5;$ $t_1 = 1{,}3 \cdot 10^{-5};$ $r_1 = -3;$ $\beta_1 = -36{,}2$	$z_1 = 16{,}3;$ $x_1 = -2{,}1 \cdot 10^{-3};$ $y_1 = 5;$ $\omega_1 = 0{,}75$
2	$\varepsilon_2 = 7 \cdot 10^{-4};$ $z_2 = 1{,}62;$ $c_2 = 6;$ $d_2 = -1{,}362$	$\omega_2 = 1{,}3 \cdot 10^{-2};$ $f_2 = -22{,}1;$ $g_2 = 10;$ $h_2 = 0{,}935$
3	$\beta_3 = 9;$ $\omega_3 = 1{,}63 \cdot 10^{-7};$ $a_3 = -0{,}3;$ $\alpha_3 = 12{,}62$	$\omega_3 = -3{,}63;$ $p_3 = 3{,}1 \cdot 10^{4};$ $z_3 = 0{,}3;$ $y_3 = 9$
4	$\delta_4 = 11{,}39;$ $q_4 = 2{,}5 \cdot 10^{-3};$ $f_4 = 8;$ $z_4 = -0{,}762$	$q_4 = 19;$ $b_4 = 0{,}4 \cdot 10^{4};$ $c_4 = -465;$ $n_4 = 195$

1	2	3
5	$\gamma_5 = 1{,}1 \cdot 10^{-5}$; $t_5 = 12{,}5$; $\omega_5 = 0{,}52$; $n_5 = 15$	$\delta_5 = 1{,}89$; $d_5 = 0{,}5 \cdot 10^5$; $f_5 = -79$; $m_5 = 137$
6	$\omega_6 = 13{,}8$; $h_6 = 6{,}2 \cdot 10^{-5}$; $k_6 = 2003$; $\alpha_6 = -0{,}75$	$t_6 = 19{,}8$; $\beta_6 = 7{,}9 \cdot 10^{-3}$; $j_6 = -13$; $m_6 = 15$
7	$\beta_7 = 6{,}79$; $z_7 = 5{,}1 \cdot 10^{-5}$; $x_7 = -0{,}863$; $l_7 = 12$	$p_7 = -8{,}91$; $\omega_7 = -27$; $\alpha_7 = 8{,}1 \cdot 10^{-6}$; $\gamma_7 = 0{,}21$
8	$\beta_8 = 11$; $\omega_8 = 1{,}33 \cdot 10^{-6}$; $a_8 = -0{,}5$; $\alpha_8 = 13{,}56$	$\omega_8 = -3{,}63$; $p_8 = 2{,}1 \cdot 10^{-3}$; $z_8 = 0{,}73$; $y_8 = 19$

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическими сведениями (см. Приложение 3).
2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:

1) номер и название работы;
2) цель работы;
3) постановку задачи;
4) схему алгоритма;
5) таблицу идентификаторов;
6) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:

1. Набрать на клавиатуре текст Паскаль-программы .
2. Произвести компиляцию исходной программы.
3. В случае обнаружения ошибок отредактировать программу с ее последующей повторной компиляцией.
2. Запустить программу после сообщения об ее успешной компиляции.
3. Ввести исходные данные для получения окончательного результата.

4. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

1.Что Вы понимаете под вводом данных на Паскале?

2.В какой форме осуществляется ввод данных на Паскале?

3.Что Вы понимаете под выводом данных на Паскале?

4.В каких форматах можно осуществить вывод данных на Паскале?
5.Какими операторами можно организовать ввод данных на Паскале?
6.Каковы правила записи операторов ввода-вывода данных на Паскале?

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:
1. Экспериментальные результаты в виде распечатки текста отлаженной Паскаль-программы и результатов счета.
2. Выводы по работе.

Лабораторная работа № 5
Программирование линейных вычислительных процессов

Цель работы: *Приобретение практических навыков составления Паскаль-программ решения задач линейных вычислительных процессов.*

Постановка задачи

Разработать блок-схемы алгоритмов и составить Паскаль-программы решения задач по варианту условия, определяемому номером бригады (табл. 5.1 и 5.2).

Таблица 5.1

Варианты заданий

№ вариантов	Математические выражения	Исходные данные
1	2	3
1	$\alpha_1 = \frac{ax^2 + \sin^2 z}{\sqrt{1+e^y}};$	$a = 15{,}2;\quad x = 0{,}89;$ $z = 31{,}8;\quad y = 1{,}25.$
2	$t_2 = \frac{\beta^2 + \sqrt{\lvert q \rvert}}{\cos^2 x + \beta \ln x};$	$\beta = 0{,}85;\quad q = 10{,}2;$ $x = 2{,}675.$
3	$q_3 = \frac{\sin^2(z+a)^3}{t\sqrt[3]{e^{a+2q}}};$	$z = 0{,}764;\quad a = 1{,}27;$ $t = 12{,}5;\quad q = 0{,}9.$
4	$z_4 = \frac{3x^2 - \sqrt{\cos(y^3)}}{\ln^2(y+\gamma)};$	$x = 2{,}61;\quad y = 1{,}13;$ $\gamma = 0{,}84.$
5	$\lambda_5 = \frac{4q\sqrt{\lvert x + \sin(z^3) \rvert}}{3\ln^2(q+x)};$	$q = 7{,}6;\quad x = -0{,}78;$ $z = 4{,}67.$
6	$\delta_6 = \frac{a\sqrt[3]{x + \ln^2 y}}{\lvert t^3 \rvert};$	$a = 1{,}8;\quad x = 0{,}729;$ $y = 6{,}3;\quad t = -1{,}5.$
7	$\varphi_7 = \frac{\rho^3 + e^{2\beta t}}{13{,}2\sqrt{\ln(\alpha + t)}}$	$\rho = 0{,}875;\quad \alpha = 1{,}8;$ $t = 7{,}9;\quad \beta = 1{,}1.$
8	$\omega_8 = \frac{\lvert \alpha^3 \rvert + \sqrt[3]{\sin^2 z}}{e^{\alpha t}\sqrt{x}}$	$\alpha = 2{,}65;\quad z = 1{,}7;$ $x = 15{,}4;\quad t = 0{,}76.$

Таблица 5.2

Варианты заданий

№ вариантов	Логические выражения	Арифметические выражения	Примечания
1	B1 = (a < 1) ^ (b > 7)	c = d or f	1. Значениями исходных данных задаться самостоятельно; 2. Символ ^ означает логическую операцию «И», а символ v – логическую операцию «ИЛИ»
2	B2 = (a >= 10) v (c1 < 15)	c = d shl 2	
3	B3 = (s <= (t + 1)) ^ (g < 5)	c = d and f	
4	B4 = m v (f > (k + 1))	c = d xor k	
5	B5 = (c > b1) ^ (f < k)	c = d shr 3	
6	B6 = (x < y) v (g > 2)	c = d and k	
7	B7 = (2a = x1) ^ (z1 < 8)	c = d xor k1	
8	B8 = ((k + 1) = c1) ^ x1	c = k1 and f	

Содержание лабораторной работы

Лабораторная работа включает:
1. Ознакомление с теоретическими сведениями (см. Приложение 3).
2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:
   1) номер и название работы;
   2) цель работы;
   3) постановку задачи;
   4) блок-схему алгоритма решения задачи;
   5) таблицу идентификаторов;
   6) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:
1. Набрать на клавиатуре текст Паскаль-программы .
2. Произвести компиляцию исходной программы.
3. В случае обнаружения ошибок отредактировать программу с ее последующей повторной компиляцией.
4. Запустить программу после сообщения об ее успешной компиляции.
5. Ввести исходные данные для получения окончательного результата.
6. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

1. Что Вы понимаете под термином «линейный вычислительный процесс»?
2. Как строится схема алгоритма линейного вычислительного процесса?
3. Каково назначение таблицы идентификаторов?
4. С чего начинается написание Паскаль-программы?
5. Чем заканчивается текст Паскаль-программы?
6. В каких форматах можно осуществить вывод данных на Паскале?
7. Что означает выражение «естественный порядок выполнения операторов»?

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:

1. Экспериментальные результаты в виде распечатки текста отлаженной Паскаль-программы и результатов счета.
2. Выводы по работе.

Приложение 3 (к модулю М3)

Теоретические сведения к лабораторным работам:

№ 4- «ВВОД-ВЫВОД ДАННЫХ НА ЯЗЫКЕ ПАСКАЛЬ» и №5- «Программирование линейных вычислительных процессов»

1. Структура и общие правила написания программы на Паскале

Программа реализует алгоритм решения задачи и представляет собой последовательность действий над определёнными данными с помощью математических операций. При разработке программы следует руководствоваться *основными принципами структурного программирования:*

- не используйте сложных методов там, где можно обойтись простыми;
- без крайней необходимости не используйте оператор перехода goto;
- исключайте переходы извне внутрь рассматриваемой разветвляющейся структуры;
- циклические структуры задавайте в явном виде, избегая операторов goto;
- большие программы разбивайте на логически завершённые сегменты (процедуры и функции);
- выбирайте имена констант, переменных, процедур, функций по смыслу, с учётом их назначения.

Программа на языке Паскаль состоит из строк. Набор текста программы осуществляется с помощью встроенного редактора текстов системы программирования Турбо-Паскаль. Существуют различные схемы написания программ на Паскале, которые отличаются количеством отступов слева в каждой строке и различным использованием прописных букв. Строка может начинаться с любой колонки, т.е. величина отступа от левой границы для каждой строки устанавливается самим программистом с целью получения наиболее ясного текста программы. Количество операторов в строке произвольно. Один оператор может записываться на нескольких строках. Такое разбиение является условным из соображения удобства и чёткости, так как никаких знаков переноса в Паскале не используется.

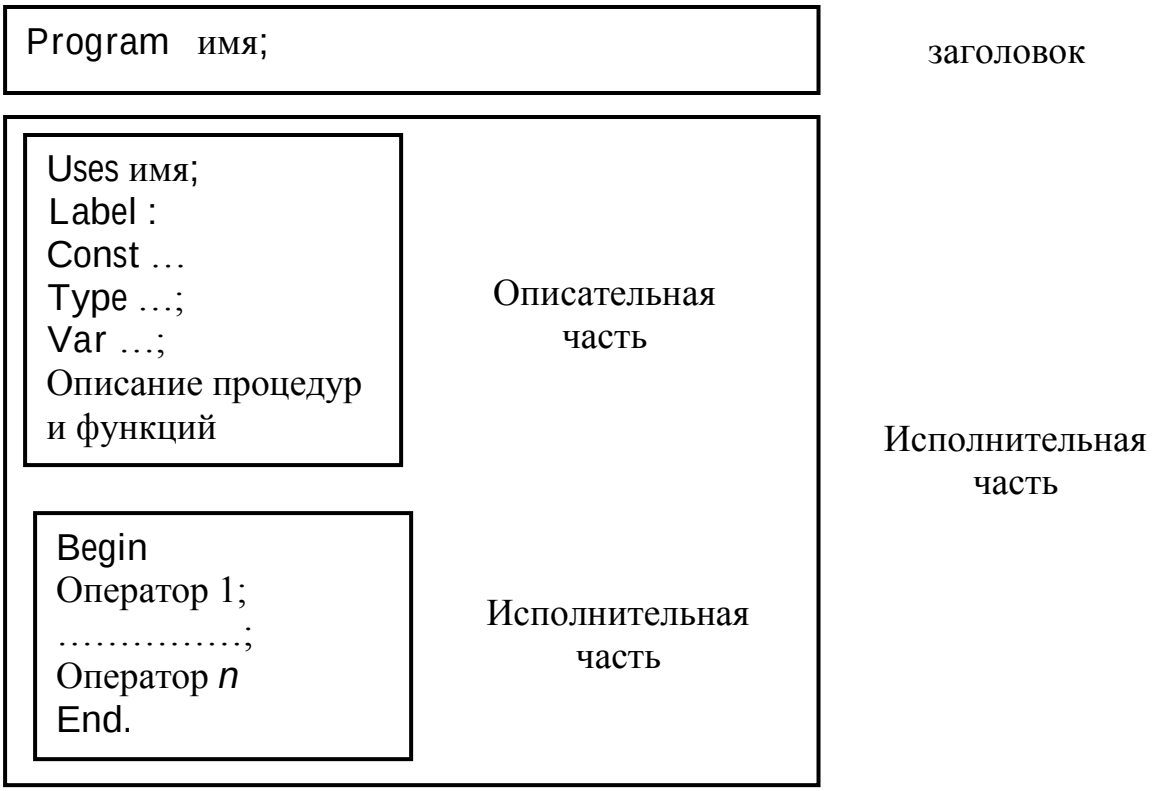
Синтаксически программа состоит из необязательного заголовка и программного блока. Заголовок в общем случае состоит из ключевого слова Program и имени программы.

Программный блок может содержать в себе другие блоки. Блок, который не входит ни в какой другой блок, называется *глобальным*. Другие блоки, находящиеся в глобальном блоке, называются *локальными*. ***Глобальный блок*** - это основная программа, локальные блоки - это процедуры и функции. Отдельные элементы программы (типы, переменные, константы и др.) соответственно называются глобальными или локальными и областью действия их являются: блок, в котором они описаны, и все вложенные в него блоки. Блочная структура обеспечивает структуризацию программ. В идеальном случае программа на Паскале состоит из подпрограмм (процедур и функций), которые вызываются для выполнения из раздела операторов основной программы.

Программный блок состоит из двух частей: описательной и исполнительной. Описательная часть в общем случае включает в себя 6 разделов: список имён подключаемых модулей (он определяется ключевым словом Uses), описание меток, описание констант, определение типов данных, описание переменных, описание процедур и функций. Исполнительная часть (её ещё называют разделом операторов) начинается ключевым словом Begin (начало), далее следуют операторы, записанные согласно алгоритму решаемой задачи и отделенные друг от друга точкой с запятой.

Завершается исполнительная часть программного блока ключевым словом End (конец) с точкой. Слова Begin и End являются аналогом открывающей и закрывающей скобок в обычных арифметических выражениях, поэтому их называют ещё **операторными скобками.**

Структуру программы на Паскале в общем случае можно представить следующем образом:



В программе любой описательный раздел может отсутствовать. Разделы описания могут следовать в любом порядке (кроме Uses, который всегда располагается после заголовка программы). Главное, чтобы все описания элементов были бы сделаны до того, как они будут использоваться.

При компиляции программы процессор ПК рассматривает содержащиеся перед операторами описания переменных и отводит в памяти соответствующие места для размещения каждой из переменных. При выполнении программы во время вычисления значения выражения производятся обращения за значениями переменных в отведённые для них места памяти, а полученное новое значение для переменной помещается в закреплённое за данной переменной место в памяти, а предыдущее значение этой переменной стирается.

Раздел Uses. Он состоит из ключевого слова Uses и списка имён подключаемых стандартных и пользовательских модулей.

Пример: Uses Crt, Dos, MyLib;

Раздел описания меток. Перед любым оператором можно поставить метку, состоящую из имени и следующего за ним двоеточия. Именем метки может служить идентификатор или число. Все метки должны быть описаны. Раздел описания меток начинается ключевым словом Label (метка), за которым следуют имена меток, разделённые запятыми. За последним именем ставится точка с запятой. Например:

Label Blok, Ml, 5, 15;

Раздел описания констант. В этом разделе производится присвоение идентификаторам констант постоянных значений. Раздел начинается ключевым словом Const, за которым следуют выражения, присваивающие идентификаторам (через =) постоянные числовые или строковые значения. Эти выражение отделяются друг от друга точкой с запятой.

Пример. Const **A=50; B2='Блок1';**

Раздел описания типов данных. Тип данных может быть описан либо в разделе описания переменных, либо определяться идентификатором типа. Раздел описания типов данных начинается ключевым словом **Type,** за которым следует определение типов, разделяемых точкой с запятой.

Пример. Type Matr = array [1..10] of real;
Dni = 1..31; LatBukva = ('a'..'2');

Раздел описания переменных. Каждая встречающаяся в программе переменная должна быть описана. Раздел описания переменных начинается ключевым словом Var, затем через запятые перечисляются имена переменных и через двоеточие указывают их тип, а после типа ставится точка с запятой.

Пример. Var A,B,C:integer; Res,Sum:real;
Ll, Vhod:boolean;

Раздел описания процедур и функций. В этом разделе размещаются тела подпрограмм. ***Подпрограммой*** *называется программная единица, имеющая имя, по которому она может быть вызвана из других частей программы.* В Паскале роль подпрограмм выполняют процедуры и функции, которые подразделяются на стандартные и определённые пользователем. Стандартные процедуры и функции являются частью языка Паскаль и могут вызываться без предварительного описания. А процедуры и функции пользователя должны описываться обязательно. В общем случае подпрограмма имеет ту же структуру, что и основная программа. При описании подпрограмм их заголовки начинаются ключевыми словами: Procedure или Function. Более подробное рассмотрение описания процедур и функций пользователя приведено позже (в лабораторной работе № 8).

Комментарии. ***Комментарий*** *— это пояснительный тест, который можно записать в любом месте программы, где размещён пробел.* Текст комментария ограничивается символами {} или (**) и может использовать любые комбинации латинских и русских букв, цифр и других символов алфавита языка Паскаль. Комментарии игнорируются компилятором.

В процессе отладки программы часто требуется временно исключить выполнение какой-либо части программы. Это удобно выполнить путём заключения

временно этой части программы в символы {} или (**), которые после отладки программы можно убрать, и программа будет выполняться в полном объёме.

2.Программирование линейных алгоритмов

Линейный алгоритм использует такие символы-блоки (см. рис.1): *терминатор* (начало, конец), *данные* (ввод/вывод данных), *процесс* (блоки 4-6). Рассмотрим программирование данных символов-блоков на языке Паскаль.

Блок 1 (начало). Этот блок включает заголовок программы на Паскале, все описательные разделы и операторную скобку Begin раздела операторов.

Блоки ввода/вывода данных. Эти блоки программируются с помощью операторов ввода/вывода данных: Read, ReadLn, Write, WriteLn.

Операторы Read и ReadLn обеспечивают ввод данных с клавиатуры для последующей их обработки программой. Их форматы:

Read (xl,x2, ...,xn); ReadLn(xl,x2,... ,xn); ,

где xl, x2, ..., xn - вводимые переменные. Значения переменных xl,x2, ...,xn при вводе набираются на клавиатуре минимум через один пробел (но не через запятую) и высвечиваются на экране. После набора данных для одного оператора Read или ReadLn нажимается клавиша <Enter>. Значения вводимых переменных должны соответствовать этим переменным по очерёдности и типам. Если соответствие по типам будет нарушено, то возникнет ошибка ввода и появится сообщение об этом. Если соответствие будет нарушено по очерёдности, то будут неверными результаты вычислений по программе.

Если в программе имеется несколько операторов Read, данные для них можно набирать в одной строке потоком, так как после считывания значений переменных по первому оператору Read курсор остаётся в той же строке вводимых данных.

Оператор ввода ReadLn аналогичен оператору Read, за исключением того, что после считывания значения последней переменной одного оператора ReadLn курсор перейдёт на начало следующей строки и данные для очередного оператора ReadLn должны набираться с начала новой строки.

Операторы Read и ReadLn требуют *обязательного* ввода данных. Если вы их не введёте, а просто нажмёте клавишу <Enter>, то работа оператора ввода не закончится и он будет ожидать ввода конкретной информации, а вы не сможете перейти к выполнению следующего оператора программы.

Значения переменных xl,x2,...,xn в операторах Read и ReadLn при вводе можно набирать и в нескольких строках, нажимая клавишу <Enter> после набора каждой строки данных, так как в данном случае после каждого нажатия клавиши <Enter> курсор переходит в начало следующей строки.

Оператор ReadLn;, записанный в программе без списка водимых переменных, приводит к остановке её дальнейшего выполнения до тех пор, пока вы не нажмёте клавишу <Enter>. Поэтому можно использовать оператор ReadLn; для просмотра выводимых результатов вычисления по программе, например:

… … …

WriteLn ('x=',x,'y=',y);

```
Write (‘Нажмите Enter’);
ReadLn;
End.
```

В Турбо-Паскале допускается вводить значения следующих данных: целых, вещественных, символьных и строковых переменных.

С помощью оператора ввода нельзя ввести:

1) значение логической переменной;
2) значение переменной перечисляемого типа;
3) значения структурированных переменных (массивов, множеств, записей).

Пример 1.

```
Var A, B, C: real;
I, K: integer;
… … …
ReadLn (A, B, C);
Read (I, K);
```

В данном примере значения переменных вводятся в следующем порядке:

```
0.5 6.25 -7.1E-1
1 5
```

или

```
0.5
6.25
-7.1E-1
1
5
```

Но нельзя все числа записать в одной строке, так как используется оператор ReadLn. После выполнения операции ввода переменным будут присвоены такие значения: A = 0,5; B = 6,25; C = -0,71; I=1; K = 5.

Пример 2. Пусть имеются переменные следующих типов: R: real; Cl, C2, C3 : char, которым необходимо присвоить соответственно значения: 1,5; 'A' ; 'B' ; 'C. Для этого используется оператор Read (R, Cl, C2, C3); . При вводе значения переменных можно расположить следующим образом:

1.5ABC или **1.5EOABC** (без апострофов), но нельзя после 1.5 поместить пробел, так как он воспримется как значение символьной константы.

Оператор вывода данных имеет формы записи:

1) Write (список переменных) – выводит последовательно значения переменных из списка;
2) WriteLn(список переменных) – то же, что и оператор Write, но после вывода значения последней переменной из списка осуществляется переход на новую строку;
3) WriteLn ; -- осуществляет переход на новую строку (без вывода данных).

Транслятор по умолчанию отводит определенное число позиций для выводимых величин каждого типа. Все элементы вывода печатаются в строку в заданном порядке, при этом пробелы между ними автоматически не ставятся. Их при желании необходимо учитывать самим при программировании операторов вывода.

Пример. Пусть в результате выполнения программы переменные получили такие значения: I=-5, R=3.52, C= '+' , B= True.

Выведем их на печать:

```
Program Pr;
Var I: integer;
R:real;
C:char;
B:Boolean;
Write (‘Пример‘); WriteLn;
WriteLn (’ I=’, I,’ R=’, R’);
WriteLn (‘ C=’, C );
WriteLn (‘ B=’, B );

End.
```

В результате выводимые значения примут вид:

Пример.

```
I=-5 R= 5.2000000000 E+00
C=+
B= True.
```

Форматный вывод данных. В ТП предусмотрен вывод данных с форматами. В общем случае формат имеет вид:

P: M,

где **P**- имя переменной,

M – целая константа, указывающая на число позиций для выводимой величины **P**.

Для вещественных переменных формат может быть задан и в таком виде:

P: M: N,

где **M** –общее число позиций для выводимой переменной **P**, включая знак числа, целую часть, точку и дробную часть;

N – число позиций дробной части. Если парметры M и N опущены, то вещественная переменная выводиться в виде константы с плавающей точкой.

Пример. Используем форматный вывод переменных из предыдущего примера:

```
WriteLn (‘ I=’, I:3, ’ R=’,R:5:2);
Write (‘ C=’, C:2, ’ B=’,B:6);
```

В результате получим:

```
I=  5_R=  3.52
C= +  B=   True.
```

Необходимо помнить, что все символы (включая пробелы, запятые), заключенные между открывающим и закрывающим апострофом в списке переменных операторов Write и WriteLn, выводятся на экран как элементы текста. *Если при выводе значения некоторой переменной выводимое число не будет помещаться в указанный формат, то часть значения переменной, расположенная перед десятичной точкой, будет выведена на экран полностью.* При этом число позиций, предназначенных для вывода дробной части числа остается равным указанной в формате величин (дробная часть, не укладывающаяся в заданное число позиций, округляется; округление не изменяет самого значения переменной, а касается только процесса вывода этого значения).

Если в выводимом числе дробная часть отсутствует, оно выводится в экспоненциальной форме с достаточным для точного изображения числом позиций.

Но если же вы хотите выдать значение вещественного числа без дробной части (и без экспоненты), то необходимо указать следующий формат:

WriteLn (P :M :O);

Использование форматного вывода позволяет корректно оформлять различного рода таблицы.

Ввод данных с клавиатуры можно осуществлять и по запросам. В этом случае необходимо запрограммировать соответствующие запросы на ввод данных, используя операторы Write, а ввод численных значений - по операторам Read или ReadLn. Например, запрограммируем ввод переменных x = 37,5; y = $-0{,}7*10^2$; z = -2,73 по следующим запросам:

Введите значение x = 37.5

1-й запрос ответ пользователя на запрос (набор на клавиатуре);

Введите значение y = -0.7E+02

2-й запрос ответ пользователя на запрос;

Введите значение z = -2.73

3-й запрос ответ пользователя на запрос.

Тогда на Паскале такой ввод данных будет иметь вид:

```
Write ('Введите значение x = ');
ReadLn (x);
Write ('Введите значение y = ');
ReadLn (y);
Write ('Введите значение z = ');
ReadLn (z);
```

Блоки 4-6 (рис.1) линейного алгоритма программируются с помощью *операторов присваивания.* Общий вид оператора присваивания следующий:

Идентификатор переменной := выражение;

Оператор присваивания предписывает выполнить *выражение*, записанное в его правой части, и результат вычисления по этому выражению присвоить (:=) *переменной*, идентификатор которой расположен в левой части оператора. Допустимо присваивание любых типов данных, кроме *файловых*.

Пример.

```
Y :=Sqrt (x)+1;
b :=M and N;
```

В операторе присваивания идентификатор переменной и выражение должны иметь один и тот же тип, кроме одного исключения: *идентификатору переменной типа* real *разрешается присваивать выражение типа* integer.

Блок 8 (рис.1) линейного алгоритма программируется оператором End . (с точкой).

3. Интегрированная среда программирования Турбо-Паскаль

Интегрированная среда программирования Турбо-Паскаль (в дальнейшем ТР) включает в себя: экранный редактор, компилятор, редактор связей и отладчик. ТП позволяет набирать тексты программ с использованием внутреннего редактора текстов, компилировать их, выполнять программы и проводить их отладку. Управление всеми этими функциями возможно и в режиме *меню*, и с помощью

соответствующих *функциональных клавиш*. Так, для выбора необходимой функции нужно подвести курсор к требуемой команде и нажать клавишу *ввода* (или *нажать выделенную в команде заглавную букву*).

При возникновении ошибки трансляции ТП автоматически переходит в режим экранного редактирования и ставит курсор в точку возникновения ошибки. Аналогичные действия выполняются и отладчиком при возникновении ошибки во время выполнения программы.

Запуск системы программирования Турбо-Паскаль осуществляется командой Turbo , после выполнения которой на экране появляется *главное меню* системы.

Для выхода из ТР можно нажать клавиши «Alt + X».

ТП использует следующие основные расширения файлов:

com и *exe* – выполнимые файлы (программы, готовые для выполнения);

pas – файл с исходным текстом программы на Паскале;
bak – резервная копия *pas*- файла;
tp1 - файл, содержащий стандартные модули ТР.

Для работы с ТР обязательными являются два файла: *Turb.exe* (компилятор с интегрированной средой программирования) и *Turbo.tp1* (библиотека стандартных модулей).

Главное меню (*первая строка экрана*) содержит команды: *File* (файл), *Edit* (редактор), *Run* (выполнение), *Compile* (компилирование), *Options* (опции), *Debug* (отладка), *Break/Watch* (прерывание/просмотр). Все они, кроме *Edit* , имеют собственные подменю, а некоторые – и несколько вложенных подменю.

Для входа в главное меню можно нажать клавишу F10, для выхода из него - Esc .

Команда *File* содержит функции, управляющие работой с файлами: *Load* - загрузка файла с диска и переход в режим экранного редактирования; *New* - удаление текущей программы из памяти и очистка экрана; *Save* - сохранение на диске текущего редактируемого файла и продолжение редактирования и др.

Команда *Edit* активизирует экранный редактор (эта команда не имеет собственного меню).

Команда *Run* объединяет функции и команды, управляющие трассировкой и выполнением программы. В этот режим входят следующие функции:

Run - запуск программы на выполнение (при необходимости выполняется трансляция программы). По завершении работы программы происходит возврат в ТР. Синоним этой функции – *Ctrl+F9* ;

Go to cursor - выполнение программы (без трассировки) от текущей строки (в режиме отладки текущая строка выделяется голубым цветом) до строки, в которой находится курсор. Синоним – *F4*;

Trace into - покомандное выполнение (трассировка) программы. Синоним *F7*;

Step over - пооператорное выполнение программы. Синоним *F8*;

User screen - показ результатов выполнения программы, выведенных на экран. Для возврата достаточно нажать любую клавишу. Синоним - *Alt +F5*;

Команда *Compile* содержит команды для управления процессом трансляции программы, например *Compile* - трансляция программы (синоним *Alt +F9*) и др.

Команда *Options* обеспечивает управление режимами ТР.

Команда *Debug* позволяет определять и изменять значения переменных и используется при отладке программы.

Команда *Break/Watch* позволяет управлять точками прерывания и переменными в окне просмотра (в этом окне отражаются текущие значения

заданных переменных и выражений; для переключения в окно просмотра из окна редактирования служит клавиша F6).

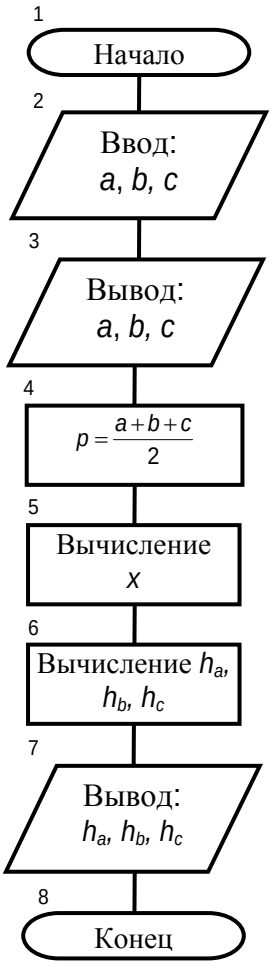


Рис. 5.1

Литература

5. Вальвачев, А.Н., Крисевич, В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС. – Мн.: Выш. шк., 1989. – 223 с.
6. Паскаль для персональных компьютеров / Ю.С. Бородич [и др.]. – Мн.: Выш. шк., 1991. – 365 с.
7. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Госстандарт, 1990. – 28 с.
8. Офицеров, Д.В., Старых, В.А. Программирование в интегрированной среде Турбо-Паскаль. – Мн.: Беларусь, 1992. – 240 с.

МОДУЛЬ М4 – «ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ»

Лабораторная работа № 6
Программирование разветвляющихся вычислительных процессов с использованием условного оператора IF

Цель работы: *Приобретение практических навыков составления программ решения задач разветвляющейся вычислительной структуры с использованием условного оператора IF .*

Постановка задачи

Разработать блок-схему алгоритма и составить Паскаль-программу решения задачи по варианту условия, определяемому номером бригады (табл. 6.1) .

Таблица 6.1

Варианты заданий

№ вариантов	Математические выражения	Исходные данные
1	Y = sin(5k + 3m· | k|); n=1, если k<m; Y = cos(5k + 3m·| k|); n=2, если k>m; Y = k + 5m ; n=3, если k=m	k, m
2	H = arctg(x + |y|); n=1, если x<y; H = arctg(|x| + y); n=2, если x>y; H = $(x + y)^2$; n=3, если x=y	x, y
3	A =$(x + y)^2$ + $\sqrt{x \cdot y}$; n=1, если x·y>0; A =$(x + y)^2$ + $\sqrt{\lvert x \cdot y \rvert}$; n=2, если x·y>0; A =$(x + y)^2$ + 1 ; n=3, если x·y>0;	x, y
4	K = ln(|f| + |g|); n=1, если (f·g)>0; K = e^{f+g} ; n=2, если (f·g)<0; K = f + g; n=3, если (f·g)=0;	f, g
5	L = $3k^3 + 3p^2$; n=1, если k>|p|; L = |k – p| ; n=2, если k<|p|; L = $(k – p)^2$; n=3, если k=|p|;	k, p
6	C = $x^2 + y^2$ + sin(x); n=1, если x – y = 0; C = $(x - y)^2$ + cos(x); n=2, если x – y > 0; C = $(y – x)^2$ + tg(x); n=3, если x – y < 0;	x, y
7	Y = a + b; n=1, если c = 0; Y = a + b + c; n=2, если c > 0; Y = (a + b) · c; n=3, если c< 0;	a, b, c
8	Y = tg(2x) + z; n=1, если z > 0; Y = $5x^4 + 3x^3 – 2x^2 + 1{,}5$ + ln|z| ; n=2, если z < 0; Y = sin(x); n=3, если z = 0;	x, z

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическими сведениями (см. Приложение 4).

2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:

1) номер и название работы;
2) цель работы;
3) постановку задачи;
4) блок-схему алгоритма;
5) таблицу идентификаторов;
6) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:

1. Набрать на клавиатуре текст Паскаль-программы .

2. Произвести компиляцию исходной программы.

3. В случае обнаружения ошибок отредактировать программу с ее последующей повторной компиляцией.

2. Запустить программу после сообщения об ее успешной компиляции.
3. Ввести исходные данные для получения окончательного результата, при этом исходные данные подобрать так, чтобы результаты получались по всем ветвям алгоритма.

4. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

1. Что Вы понимаете под термином «разветвляющаяся вычислительная структура»?
2. Как строится схема алгоритма разветвляющейся вычислительной структуры?
3. Какой символ осуществляет проверку некоторых условий?
4. Какой оператор Паскаля соответствует этому символу?
5. От чего зависит количество ветвей в алгоритме?
6. Какие структуры оператора If на Паскале Вам известны? Как они выполняются?

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:

1. Экспериментальные результаты в виде распечатки текста отлаженной Паскаль-программы и результатов счета (по всем ветвям алгоритма).

2. Выводы по работе.

Лабораторная работа № 7
Программирование разветвляющихся вычислительных процессов с использованием оператора выбора CASE

Цель работы: *Приобретение практических навыков составления программ решения задач разветвляющейся вычислительной структуры с использованием оператора выбора CASE.*

Постановка задачи

Построить блок-схему алгоритма и составить Паскаль-программу решения задачи с использованием оператора выбора Case по варианту условия, определяемому номером бригады

(табл. 6.1). При этом сначала определяйте значение селектора n в зависимости от логического условия, а затем определяйте значение первого математического выражения по оператору выбора *CASE* (с селектором n).

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическими сведениями (см. Приложение 4).
2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:

7) номер и название работы;
8) цель работы;
9) постановку задачи;
10) блок-схему алгоритма;
11) таблицу идентификаторов;
12) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:

5. Набрать на клавиатуре текст Паскаль-программы .
2. Произвести компиляцию исходной программы.
3. В случае обнаружения ошибок отредактировать программу с ее последующей повторной компиляцией.
6. Запустить программу после сообщения об ее успешной компиляции.
7. Ввести исходные данные для получения окончательного результата, при этом исходные данные подобрать так, чтобы результаты получались по всем ветвям алгоритма.
8. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

1. Какова структура оператора выбора *CASE* ?
2. Что такое селектор? Как он задается?
3. Какова последовательность работы оператора *CASE* ?

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:

1. Экспериментальные результаты в виде распечатки текста отлаженной Паскаль-программы и результатов счета (по всем ветвям алгоритма).
2. Выводы по работе.

Приложение 4(к модулю М4)

Теоретические сведения к лабораторным работам:
№ 6 (Программирование разветвляющихся алгоритмов с использованием условного оператора If *) и № 7 (Программирование разветвляющихся алгоритмов с использованием оператора выбора* Case *)*

Программирование разветвляющихся алгоритмов

В разветвляющихся алгоритмах кроме блоков, применяемых в линейных алгоритмах (программирование которых мы рассмотрели в Приложении 3 для рис.1), применяются управляющие блоки «решение», определяющие нужную ветвь дальнейших вычислений в зависимости от выполнения или невыполнения конкретных условий. Так как при этом нарушается естественный порядок выполнения блоков алгоритма, то при программировании разветвляющихся алгоритмов могут использоваться условные (If или Case) и безусловные (Goto) операторы управления. Рассмотрим сначала эти операторы.

Оператор безусловного перехода Goto означает « перейти к » и применяется в случаях, когда после выполнения некоторого оператора необходимо выполнять дальше не следующий по порядку оператор, а какой-либо другой, помеченный меткой.

Формат написания оператора безусловного перехода следующий:

Goto Метка;

В соответствии с принципами структурного программирования этот оператор следует применять как можно реже, так как его частое употребление усложняет понимание логики программы.

Условный оператор If (*если*). Он может использоваться в двух формах:

1. If B then P1 else P2 ;

Здесь ключевые слова If, then , else означают соответственно: *если, то, иначе;* **B** - любое логическое (булевское) выражение; P1, P2 - операторы, которые могут быть простыми, составными или условными.

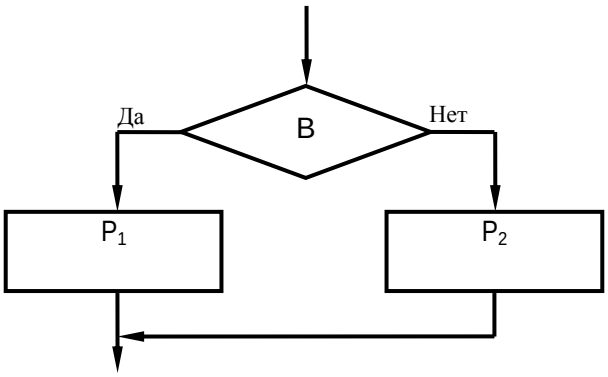


Рис. 7.1

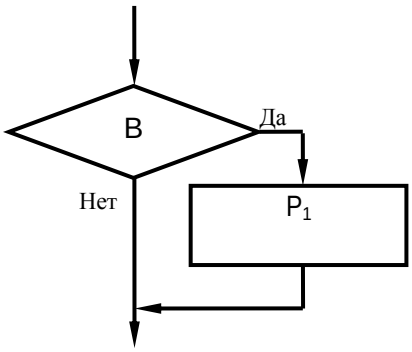


Рис. 7.2

Управление по этому оператору осуществляется следующим образом (рис. 7.1): если выражение **B** *истинно,* то выполняется оператор **P1**, а оператор **P2** пропускается; если **B** *ложно,* то пропускается оператор **P1**, а оператор **P2** выполняется.

2. If B then P1;

Управление по этому оператору осуществляется следующим образом (рис. 7.2): если выражение B *истинно,* то выполняется оператор **P1** и далее оператор, записанный после оператора If; если **B** *ложно,* то оператор **P1** пропускается (не выполняется).

Один оператор If может входить в состав другого оператора If. В таком случае говорят о вложенности операторов If:

If Bl then If B2 then P1 else P2 ;

При вложенности операторов If каждое else соответствует тому then , которое непосредственно ему предшествует (рис. 7.3). Конструкций со степенью вложения более 2-3 стараются избегать (из-за сложности их анализа при отладке программы).

Простыми операторами *называются такие операторы, которые не содержат в себе никаких других операторов.* К ним относятся операторы присваивания, безусловного перехода, вызова процедуры (этот оператор рассмотрен позже - в Приложении 7) и пустой оператор.

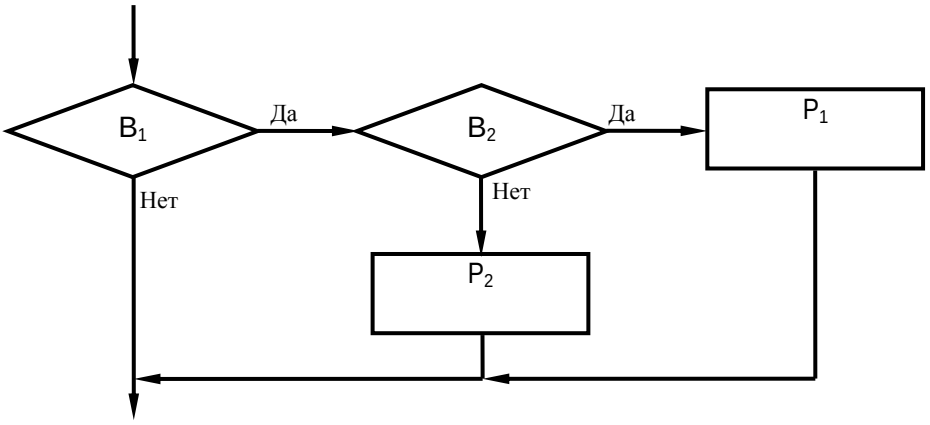


Рис. 7.3

Пустой оператор *не содержит в себе никаких символов и не выполняет никаких действий.* Он может быть расположен в любом месте программы, где синтаксис языка допускает наличие оператора. Как и все другие операторы, пустой оператор может быть помечен меткой. Чаще всего пустой оператор используется для организации выхода из середины программы или составного оператора:

```
Begin ... ..
Goto Ml; {Переход в конец программы}
... .
M1:     {Пустой оператор с меткой Ml}
End.
```

Составной оператор - это группа из произвольного числа операторов, отделенных друг от друга точкой с запятой и ограниченных операторными скобками begin и end . Составной оператор воспринимается как единое целое и может находиться в любом месте программы (чаще всего он используется в условных операторах и операторах повтора).

Оператор выбора Case. Этот оператор является обобщением оператора If и позволяет сделать выбор из произвольного числа вариантов. Он состоит из *выражения - селектора* и *последовательности операторов*, каждому из которых предшествует *список констант выбора* (список может состоять и из одной константы). Как и оператор If, оператор Case может использоваться в двух формах: со словом else , имеющим тот же смысл, как и в операторе If, и без него. Их форматы записи следующие:

1.
```
Case K of
S I : P I ;
S2: P2;
...
Sn: Pn
```

```
    else Pn+1
    end;
2.  Case K of
    SI:PI                                                    :
    S2: P2;
    ...
    Sn: Pn
    end;
```

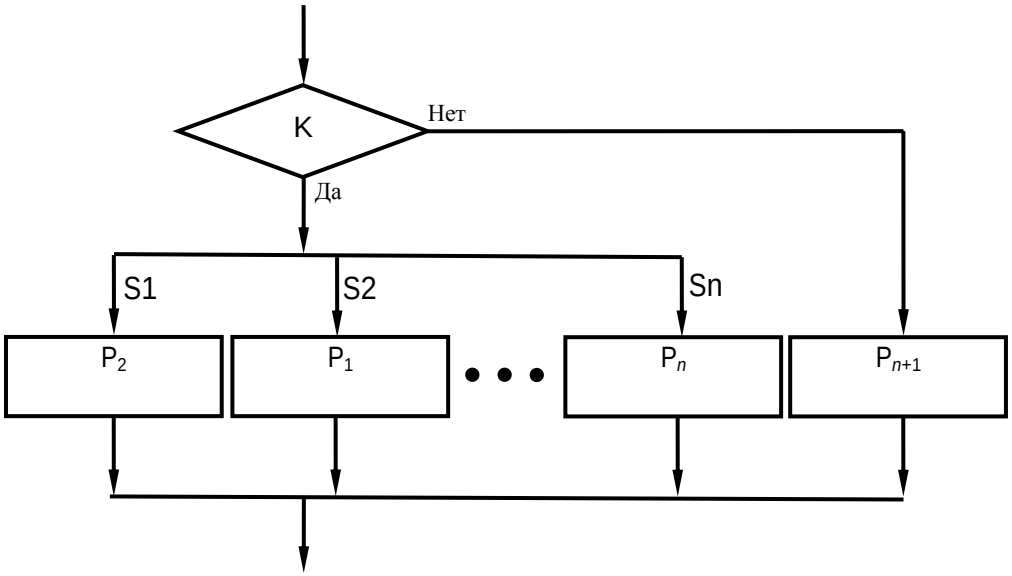


Рис. 7.4

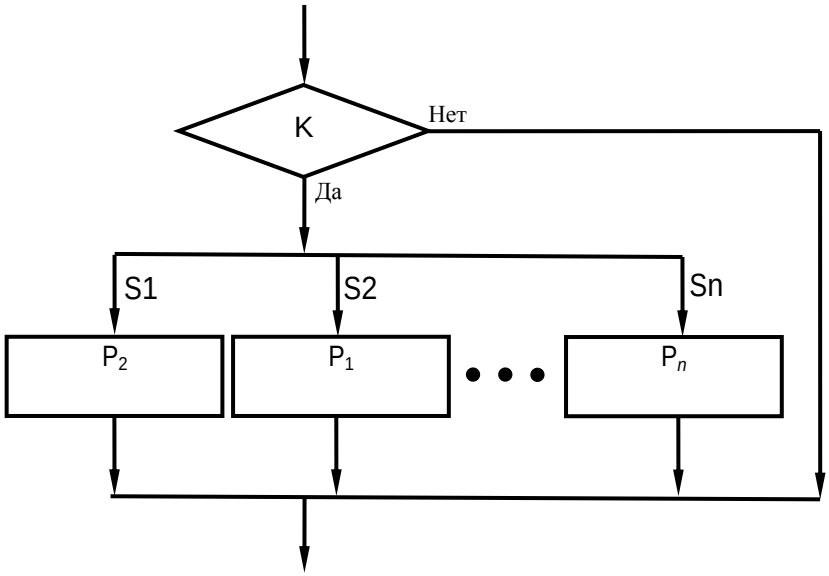


Рис. 7.5

Здесь **К** - *выражение-селектор*, которое может быть одним из скалярных типов (кроме real), т.е. целого, логического, символьного или пользовательских типов.

Оператор Case работает следующим образом (рис. 7.4): сначала выполняется тот из операторов Pi, константа выбора которого равна текущему значению *выражения-селектора* **К**. Если ни одна из констант Si не равна текущему значению **К**, то выполняется оператор, стоящий за словом else. Во втором случае (рис. 7.5), когда

слово else в операторе Case отсутствует, если ни одна из констант Si не равна текущему значению **К**, то выполняется первый оператор, за границей end; (оператор Case в этом случае пропускается).

Список констант выбора состоит из произвольного количества значений или диапазонов. Тип констант должен совпадать с типом выражения-селектора **К**.

Пример. Составим программу разветвляющегося алгоритма (рис. 7.6) по вычислению функции

$$D = \begin{cases} Ln(ab)^2, & ab<0 \\ Ln(ab), & ab>0 \\ (a+b), & ab=0. \end{cases}$$

Пусть исходные переменные *a* и *b* вещественного типа. Запрограммируем, например, разветвляющуюся структуру с блоком «решение» 3 с помощью оператора If, а - с блоком «решение» 5 с помощью оператора Case . Тогда программа будет иметь вид:

```
Program RazAlg;
 Var a, b, d :real;
 Begin
  ReadLn (a, b);
  If a*b<0 then
   D := Ln (sqr (a*b))
   else Case a*b>0 of
        True: D:=Ln(a*b)
        else D:= a+b
        end;
  WriteLn(' D = ',D);
   End.
```

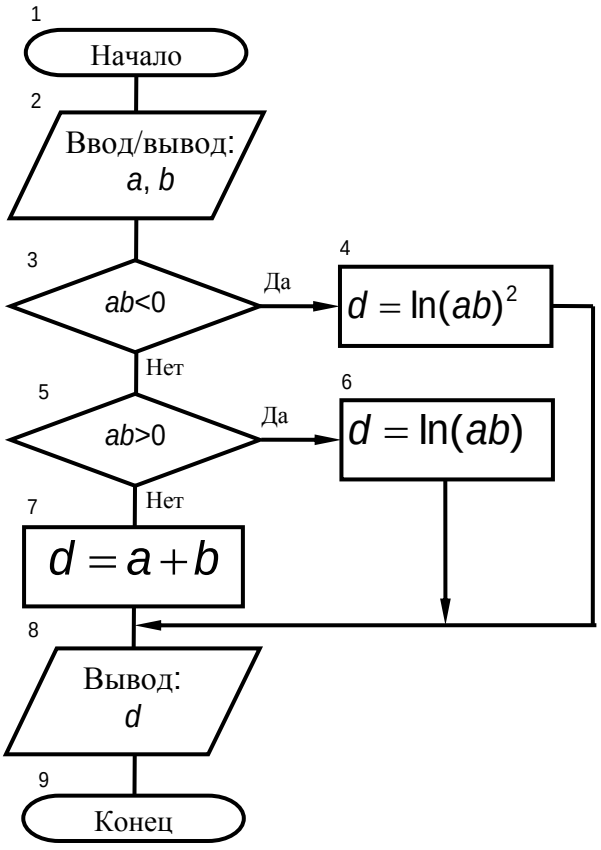


Рис. 7.6

Литература

9. Вальвачев, А.Н., Крисевич, В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС. – Мн.: Выш. шк., 1989. – 223 с.
10. Паскаль для персональных компьютеров / Ю.С. Бородич [и др.]. – Мн.: Выш. шк., 1991. – 365 с.
11. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Госстандарт, 1990. – 28 с.
12. Офицеров, Д.В., Старых, В.А. Программирование в интегрированной среде Турбо-Паскаль. – Мн.: Беларусь, 1992. – 240 с.

МОДУЛЬ М5 – «ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ»

Лабораторная работа № 8
Программирование циклических вычислительных процессов с использованием оператора цикла FOR

Цель работы: Приобретение практических навыков составления программ решения задач, содержащих циклические вычислительные структуры, с использованием оператора цикла *FOR* .

Постановка задачи

Разработать блок-схему алгоритма и составить Паскаль-программу решения задачи с использованием оператора цикла *FOR* по варианту условия, определяемому номером бригады (табл. 8.1) .

Таблица 8.1

Варианты заданий

№ вариантов	Математические выражения	Исходные данные
1	$Y = \sin(x) + \sin^2(x) + \ldots + \sin^{15}(x)$	x
2	$H = 1/a + 1/a^2 + \ldots + 1/a^{25}$	a
3	$A = 1 - x + x^2/2! - x^3/3! + \ldots + x^{12}/12!$	x
4	$K = \cos(x) + \cos(x^2) + \ldots + \cos(x^{30})$	x
5	$L = x + x^3/3! + x^5/5! + \ldots x^{15}/15!$	x
6	$C = \sum_{i=1}^{10} (x/i! + \sqrt{\lvert x \rvert})$	x
7	$S = \prod_{i=1}^{15} (x/i)$	x
8	$M = \sum_{i=1}^{10} (x/i^2)$	x

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическими сведениями (см. Приложение 4).

2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:

13) номер и название работы;
14) цель работы;
15) постановку задачи;
16) блок-схему алгоритма;
17) таблицу идентификаторов;
18) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:

9. Набрать на клавиатуре текст Паскаль-программы .

2. Произвести компиляцию исходной программы.

3. В случае обнаружения ошибок отредактировать программу с ее последующей повторной компиляцией.

10. Запустить программу после сообщения об ее успешной компиляции.

11. Ввести исходные данные для получения окончательного результата.

12. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

7. Какова структура оператора *FOR ?* Как он работает *?*

8. Как записывается оператор *FOR* , если он охватывает группу операторов ?

9. Какими операторами можно запрограммировать циклический вычислительный процесс с известным числом повторений цикла ?

10.Как программируются циклические вычислительные процессы с неизвестным числом повторений цикла ?

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:

1. Экспериментальные результаты в виде распечатки (или запись с экрана дисплея) текста отлаженной Паскаль-программы и результатов счета .

2. Выводы по работе.

Лабораторная работа № 9
Программирование циклических вычислительных процессов с использованием оператора цикла с предусловием WHILE

Цель работы: Приобретение практических навыков составления программ решения задач, содержащих циклические вычислительные структуры, с использованием оператора цикла *While* .

Постановка задачи

Разработать блок-схему алгоритма и составить Паскаль-программу решения задачи с использованием оператора цикла *While* по варианту условия, определяемому номером бригады (табл. 9.1) .

Таблица 9.1

Варианты заданий

№ вариантов	Математические выражения	Изменяемые параметры	Исходные данные
1	$S = \frac{ax^2 + \sin^2 z}{\sqrt{1+e^y}}$	Параметр x изменяется от $x = x_н = 1$ до $x = x_к = 4,5$ с шагом h = 0,5	a, z, y –константы, значения которых задать самостоятельно
2	$M = \frac{\beta^2 + \sqrt{\lvert q \rvert}}{\cos^2 x + \beta \ln y}$	Параметр x изменяется от $x = x_н = 1$ до $x = x_к = 5$ с шагом h = 0,5	β, q, y – константы, значения которых задать самостоятельно
3	$W = \frac{\sin^2 (z+a)^3}{t\sqrt[3]{e^{a+2q}}}$	Параметр z изменяется от $z = z_н = 0,3$ до $z = z_к = 1$ с шагом h = 0,1	a, z, y –константы, значения которых задать самостоятельно
4	$K = \frac{3x^2 - \sqrt{\cos(q^3)}}{\ln^2 (y+\alpha)t}$	Параметр x изменяется от $x = x_н = 0,2$ до $x = x_к = 1,5$ с шагом h = 0,1	α, q, t, y – константы, значения которых задать самостоятельно
5	$L = \frac{4\delta\sqrt{\lvert x + \sin(z^3) \rVert}}{3\ln^2 (q+x)}$	Параметр z изменяется от $z = z_н = 0,3$ до $z = z_к = 1,5$ с шагом h = 0,1	δ, x, q – константы, значения которых задать самостоятельно
6	$C = \frac{a\sqrt[3]{x + \ln^2 y}}{\lvert t^3 \rvert}$	Параметр y изменяется от $y = y_н = 0,2$ до $y = y_к = 1,5$ с шагом h = 0,2	a, x, t – константы, значения которых задать самостоятельно
7	$N = \frac{p^3 + e^{2\beta t}}{13,2\sqrt{\ln(\alpha + t)}}$	Параметр t изменяется от $t = t_н = 1$ до $t = t_к = 7$ с шагом h = 1	p, β, α – константы, значения которых задать самостоятельно
8	$P = \frac{\lvert \alpha^3 + \sqrt[3]{\sin^2 z} \rvert}{\sqrt{x}e^{\alpha t}}$	Параметр z изменяется от $z = z_н = 0,5$ до $z = z_к = 4,5$ с шагом h = 0,5	α, x, t – константы, значения которых задать самостоятельно

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическими сведениями (см. Приложение 5).
2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:

19) номер и название работы;
20) цель работы;
21) постановку задачи;
22) блок-схему алгоритма;
23) таблицу идентификаторов;
24) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:

13. Набрать на клавиатуре текст Паскаль-программы .
2. Произвести компиляцию исходной программы.
3. В случае обнаружения ошибок отредактировать программу с ее последующей повторной компиляцией.

14. Запустить программу после сообщения об ее успешной компиляции.
15. Ввести исходные данные для получения окончательного результата .
16. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

4. Какие типы циклов встречаются в циклических вычислительных процессах ?
5. Какова структура оператора цикла *While* ? Как он работает ?
6. Как осуществляется в операторе *While* выход из цикла?

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:

1. Экспериментальные результаты в виде распечатки текста отлаженной Паскаль-программы и результатов счета .
2. Выводы по работе.

Лабораторная работа № 10
Программирование циклических вычислительных процессов с использованием оператора цикла с постусловием REPEAT

Цель работы: Приобретение практических навыков составления программ решения задач, содержащих циклические вычислительные структуры, с использованием оператора цикла *Repeat*

Постановка задачи

Разработать блок-схему алгоритма и составить Паскаль-программу решения задачи с использованием оператора цикла *Repeat* по варианту условия, определяемому номером бригады (табл. 9.1) .

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическими сведениями (см. Приложение 5).
2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:

25) номер и название работы;
26) цель работы;
27) постановку задачи;
28) блок-схему алгоритма;
29) таблицу идентификаторов;
30) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:

17. Набрать на клавиатуре текст Паскаль-программы .
2. Произвести компиляцию исходной программы.
3. В случае обнаружения ошибок отредактировать программу с ее последующей повторной компиляцией.

18. Запустить программу после сообщения об ее успешной компиляции.
19. Ввести исходные данные для получения окончательного результата .
20. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

1. Какова структура оператора цикла *Repeat* ? Как он работает ?
2. Чем отличаютс я между собой операторы цикла *Repeat* и *While*?
3. Какие служебные слова в операторе *Repeat* обозначают границы тела цикла ?

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:

1. Экспериментальные результаты в виде распечатки текста отлаженной Паскаль-программы и результатов счета .
2. Выводы по работе.

Приложение 5 *(к модулю М5)*

Теоретические сведения к лабораторным работам:

№ 8 - «ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ С ИСПОЛЬЗОВАНИЕМ ОПЕРАТОРА ЦИКЛА *FOR* », *№ 9* - «ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ С ИСПОЛЬЗОВАНИЕМ ОПЕРАТОРА ЦИКЛА С ПРЕДУСЛОВИЕМ *While*», *№ 10* – «ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ С ИСПОЛЬЗОВАНИЕМ ОПЕРАТОРА ЦИКЛА С ПОСТУСЛОВИЕМ *Repeat*»

Программирование циклических алгоритмов

При программировании циклических алгоритмов используются операторы повтора: For (для), Repeat (повторять) и While (пока).

Оператор повтора For состоит из заголовка и тела цикла. Тело цикла представляет собой один или несколько операторов (согласно алгоритму), которые могут выполняться более одного раза. Оператор For используется обычно тогда, когда количество повторов известно заранее. Он может представляться в двух формах:

1. For I:= nl to n2 do P;
2. For I:= nl downto n2 do P; ,

где nl, n2 - *выражения*, определяющие начальное и конечное значения параметра цикла I;

For ... do - заголовок цикла;

P - тело цикла, которое может быть простым или составным оператором.

Оператор For обеспечивает выполнение цикла до тех пор, пока не будет перебраны все значения параметра цикла I от начального до конечного значения включительно (рис.10.1). Параметр цикла, его начальное и конечное значения должны принадлежать к одному и тому же типу, при этом допустим любой скалярный тип, кроме вещественного. Если используется тип integer, byte и интервальный, то значения параметра цикла последовательно увеличиваются (при For ... to) или уменьшаются (при For ... downto) на единицу при каждом повторе.

В теле цикла нельзя использовать операторы, изменяющие значение параметра цикла. После нормального (если не используется преждевременный выход из цикла с помощью оператора goto) завершения оператора значение параметра цикла равно его конечному значению.

В теле оператора For могут находиться другие операторы For (при вложенных циклах, рис. 10.2) :

Оператор повтора Repeat записывается в виде :
Repeat P1; …; Pn until B ; ,
где операторы P1; ... ; Pn - тело цикла, B - выражение булевского типа.

Этот оператор означает: *повторять* (Repeat) выполнять тело цикла *пока не* (until) станет выражение B истинным, после чего осуществляется выход из цикла (рис. 10.3).

Оператор Repeat имеет такие характерные особенности : тело цикла выполнятся по крайней мере один раз; оно выполняется до тех пор, пока условие B равно False; в

теле цикла может находиться несколько операторов, при этом операторные скобки begin и end не используются; по крайней мере один оператор из операторов тела цикла должен влиять на значение условия B, а иначе цикл может повторяться бесконечно.

Пример. Программа вычисления суммы четных чисел в интервале (0 - 10) включительно согласно алгоритму рис. 10.4, составленная с использованием оператора Repeat, имеет вид:

```
Program CklAlg ;
Var I, S : integer ;
Begin
   I : = 0 ;  S : = 0 ;
    Repeat  S : = S + I ;
            I : = I + 2
            Until ( I > 10 ) ;
     Writeln ( ‘ S = ‘ , S )
End .
```

Оператор While похож на оператор Repeat , но проверка логического условия **B** выполнения тела цикла осуществляется в самом начале оператора. Он записывается в виде:

```
While  B  do P ;
```

и означает: *пока* (While) выражение B истинно, то необходимо *выполнять* (do) тело цикла P (рис. 10.5).

Перед каждым выполнением тела цикла вычисляется значение выражения B. Если результат равен True , происходит выход из цикла и переход к первому после цикла оператору. Если перед первым выполнением цикла значение выражения B равно False , то тело цикла не выполняется ни разу и происходит сразу переход к следующему после цикла оператору.

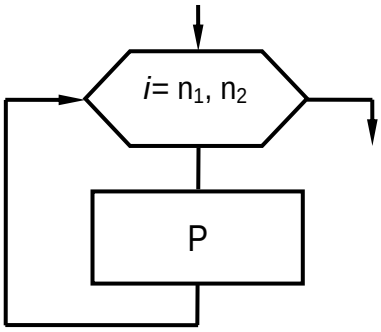


Рис. 10.1

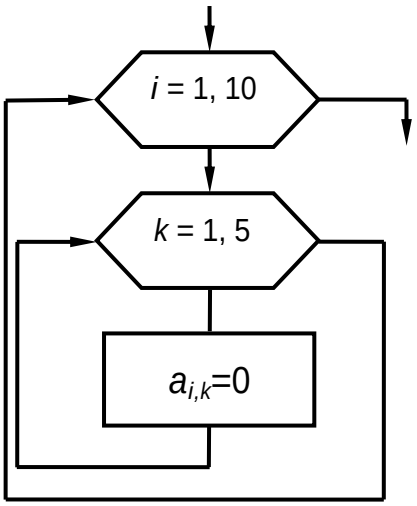


Рис. 10.2

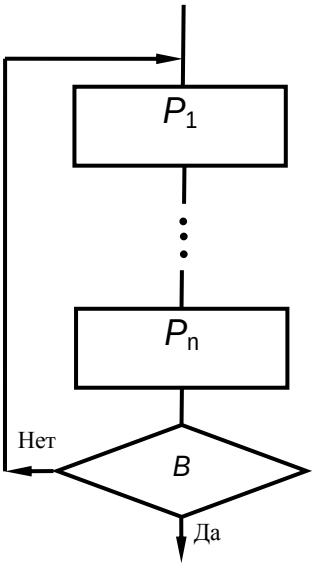


Рис. 10.3

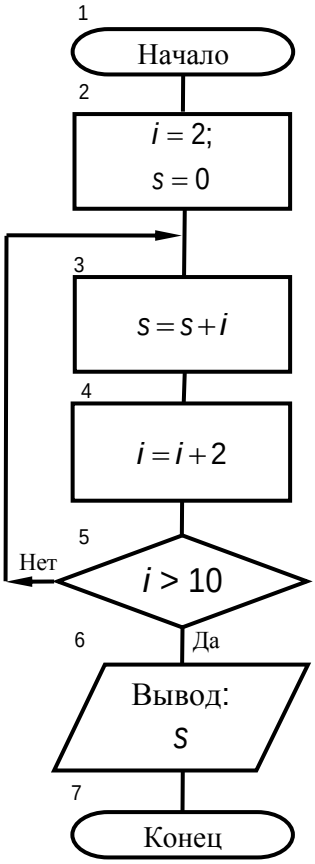


Рис. 10.4

Как и в операторе Repeat, в теле цикла должен присутствовать оператор, изменяющий переменную, входящую в условие B, иначе цикл может оказаться бесконечным.

Оператор While, как и операторы For и Repeat, может быть вложенным.

Пример. Программа, в которой с помощью оператора вычисляется n! согласно алгоритму рис. 13, имеет вид:

```
Program Fakt ;
Var n , I , p : integer ;
 Begin
    ReadIn (n) ;
     P : = 1 ; I : = 0 ;
     While I , n do
         Begin
            I : = I + 1 ;
            P : = p * I
         End ;
     Write ( ‘ n! = ‘ , p )
    End .
```

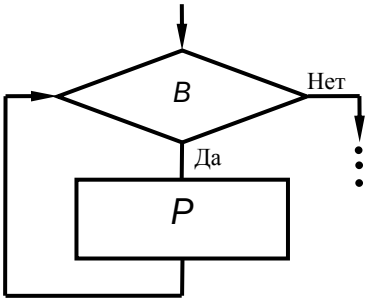


Рис. 10.5

Литература

13. Вальвачев, А.Н., Крисевич, В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС. – Мн.: Выш. шк., 1989. – 223 с.
14. Паскаль для персональных компьютеров / Ю.С. Бородич [и др.]. – Мн.: Выш. шк., 1991. – 365 с.
15. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Госстандарт, 1990. – 28 с.
16. Офицеров, Д.В., Старых, В.А. Программирование в интегрированной среде Турбо-Паскаль. – Мн.: Беларусь, 1992. – 240 с.

МОДУЛЬ М6 – «ПРОГРАММИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ С ИСПОЛЬЗОВАНИЕМ МАССИВОВ ДАННЫХ »

Лабораторная работа № 11
Программирование вычислительных процессов с использованием обработки одномерных массивов данных

Цель работы: *Приобретение практических навыков составления программ решения задач, содержащих обработку одномерных массивов данных.*

Постановка задачи

Разработать блок-схему алгоритма и составить Паскаль-программу решения задачи с использованием обработки одномерных массивов данных по варианту условия, определяемому номером бригады. Числовые значения переменных задать самостоятельно, представив их в виде таблицы сразу после условия задачи.

Варианты заданий

1. В массиве А(п) определить наибольший элемент, вывести его значение и номер, затем отсортировать массив по возрастанию.
2. В массиве А(п) определить наименьший элемент, вывести его значение и номер, затем отсортировать массив по убыванию.
3. В массиве А(п) определить сумму и количество положительных элементов и определить наименьший элемент среди положительных. Вывести его значение и номер.
4. В массиве А(п) определить сумму и количество отрицательных элементов и определить наибольший элемент среди отрицательных. Вывести его значение и номер.
5. Определить цифры введенного с клавиатуры натурального числа п и вывести их виде одномерного массива.
6. В массиве целых чисел А(п) подсчитать число элементов, находящихся в отрезке (1,10), а также найти их сумму и произведение. Определить значение и номер максимального элемента из данного отрезка.
7. В массиве С(10) найти два элемента, имеющие наибольшие значения по сравнению с остальными и вывести их номер.
8. В массиве С(15) поменять местами первый и максимальный по значению элементы.

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическими сведениями (см. Приложение 6).

2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:

31) номер и название работы;
32) цель работы;
33) постановку задачи;
34) блок-схему алгоритма;
35) таблицу идентификаторов;
36) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:

21. Набрать на клавиатуре текст Паскаль-программы .

2. Произвести компиляцию исходной программы.

3. В случае обнаружения ошибок отредактировать программу с ее по- следующей повторной компиляцией.

4. Запустить программу после сообщения об ее успешной компиляции.

5. Ввести исходные данные для получения окончательного результата.

6. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

1. Как описывается одномерный массив?
2. К каким типам данных могут относится элементы массива?
3. К каким типам данных могут относится индексы элемента массива?
4. Какие действия допустимы над массивами данных?
5. Какие действия допустимы над отдельными элементами массива?

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:

1. Экспериментальные результаты в виде распечатки (или запись с экрана дисплея) текста отлаженной Паскаль-программы и результатов счета.

2. Выводы по работе.

Лабораторная работа № 12
Программирование вычислительных процессов с использованием обработки двумерных массивов данных

Цель работы: *Приобретение практических навыков составления программ решения задач, содержащих обработку двумерных массивов данных .*

Постановка задачи

Разработать блок-схему алгоритма и составить Паскаль-программу решения задачи с использованием обработки двумерных массивов данных по варианту условия, определяемому номером бригады. Числовые значения переменных задать самостоятельно, представив их в виде таблицы сразу после условия задачи.

Варианты заданий

1. Вычислить сумму элементов каждой строки матрицы A(4*3). Результаты занести в одномерный массив B. Вывести матрицу A и массив B.
2. Дана матрица A(п*п). Заменить элементы главной диагонали нулями, а элементы, стоящие над главной диагональю - единицами.
3. Дана матрица A(n*m). В каждой строке найти минимальный элемент, затем среди этих п чисел найти минимальное. Вывести индексы элементов с найденными значениями.
4. Из матрицы целых чисел A(5*5) составить массив B, элементы которого больше среднего арифметического элементов матрицы A.
5. Для заданной квадратной матрицы сформировать одномерный массив из ее диагональных элементов, умноженных на среднее значение всех элементов матрицы. Вывести исходную матрицу и результирующий массив.
6. Дана матрица A(3*4). Найти среднее арифметическое наибольшего и наименьшего значений ее элементов.
7. Дана матрица A(п*п). Найти произведение элементов главной диагонали и сумму элементов, стоящих ниже главной диагонали.
8. Дана матрица A(6*6). Заменить нулями все ее элементы, расположенные на главной диагонали и выше ее.

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическими сведениями (см. Приложение 6).
2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:

1) номер и название работы;
2) цель работы;
3) постановку задачи;
4) блок-схему алгоритма;

5) таблицу идентификаторов;
6) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:
1. Набрать на клавиатуре текст Паскаль-программы .
2. Произвести компиляцию исходной программы.
3. В случае обнаружения ошибок отредактировать программу с ее последующей повторной компиляцией.
4. Запустить программу после сообщения об ее успешной компиляции.
5. Ввести исходные данные для получения окончательного результата.
6. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

1. Как описывается двухмерный массив?
2. Как вывести двухмерный массив в виде матрицы?
3. В чем заключается инициализация массива?
4. Какими способами можно выполнять копирование двухмерных массивов?

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:

1. Экспериментальные результаты в виде распечатки текста отлаженной Паскаль-программы и результатов счета .
2. Выводы по работе.

Приложение 6 ***(к модулю М6)***

Теоретические сведения к лабораторным работам:

№ 11 - «ПРОГРАММИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ С ИСПОЛЬЗОВАНИЕМ ОБРАБОТКИ ОДНОМЕРНЫХ МАССИВОВ ДАННЫХ », *№ 12* - «ПРОГРАММИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ С ИСПОЛЬЗОВАНИЕМ ОБРАБОТКИ ДВУМЕРНЫХ МАССИВОВ ДАННЫХ»

1. Массивы данных и их описание

Под **массивом данных** понимается упорядоченная совокупность конечного числа данных одного типа под одним именем. Имена массивов образуются так же, как и имена простых переменных. Элементами массива данных могут быть данные любого типа, включая структурированные типы. Число элементов массива фиксируется при описании и в процессе выполнения программы не меняется. Доступ к каждому отдельному элементу массива осуществляется путем указания имени массива и конкретных индексов этого элемента в *квадратных* скобках. Индексы могут представлять собой выражения любого

скалярного типа, кроме вещественного. Тип индекса определяет границы изменения его значений. Если задан один индекс, массив называется одномерным, если задано два индекса - двумерным, если n индексов - n-мерным.

Возможны два способа описания массивов:

1) Type *имя типа* = array [t1, t2,…, tN] of *тип данных* ;
   var *имя массива* : *имя типа*;

2) var *имя массива* : array [t1, t2,…, tN] of *тип данных* ;

Здесь t1, t2,…, tN – типы индексов массива (количество индексов N определяет размерность массива.

Пример. Пусть в программе необходимо описать следующий двумерный массив M :

[1 2 5]
[6 7 2]

Описание этого массива в соответствии с первым способом выглядит так:

```
Type Mas = array [1.. 2, 1.. 3] of integer ;
var M : Mas;
```

Для второго способа имеем:

```
var M : array [1.. 2, 1.. 3] of integer ;
```

В самой программе, задав конкретные значения индексов, можно выбрать определенный элемент массива:

N : = M [1, 3] ; {т.е. N будет присвоено значение 5} .

Для описания массива можно использовать предварительно определенные константы, например:

```
Comst G1 = 4; G2 = 6 ;
Var Mas : array [1..G1, 1..G2] of real;
```

Элементы массива располагаются в памяти последовательно. Элементы одномерного массива с меньшими значениями индекса хранятся в более низких адресах памяти. Многомерные массивы располагаются таким образом, что самый правый индекс возрастает самым первым. Например, если имеется массив

A : array [1.. 2, 1.. 3] of integer ; ,

то в памяти элементы массива будут размещены по возрастанию адресов в такой последовательности:

A[1, 1], A[1, 2], A[1, 3], A[2, 1], A[2, 2], A[2, 3].

2. Действия над массивами

Для работы с массивом как *единым целым* используется имя массива без указания индексов. Массив может участвовать только в операциях отношения «равно», «не равно» и в операторе присваивания. При этом участвующие в этих действиях массивы должны быть одинаковыми по структуре, т.е. иметь одинаковые типы индексов и количество элементов. Например,

```
Var A, B : array [1..20] of real;
```

Выражение **A = B** дает результат *True* , если значения каждого элемента массива A равны значениям соответствующих элементов массива B. Выражение **A<> B** дает результат *True* , если хотя бы один элемент массива A по значению не равен соответствующему элементу массива B.

По оператору **A : = B** все значения элементов массива B присваиваются соответствующим элементам массива A, при этом значения элементов массива B остаются неизменными.

3. Действия над элементами массива

Индексированные элементы массива называются индексированными переменными и могут быть использованы точно так же, как и обычные (простые) переменные. Например, они могут находиться в выражениях в качестве операндов; использоваться в операторах повтора; входить в качестве параметров в операторы ввода и вывода данных; им можно присваивать любые значения, соответствующие их типу.

Рассмотрим *типичные ситуации*, возникающие при работе с массивами данных. Пусть имеем три массива и четыре вспомогательные переменные:

```
Var A, D : array [1..4] of real;
    B : array [1..10, 1..15] of integer;
    I, J, K : integer;  S : real;
```

Инициализация массива заключается в присваивании каждому элементу массива одного и того же значения, соответствующего базовому типу массива. Наиболее эффективно эта операция выполняется с помощью оператора цикла for (рис. 12.1):

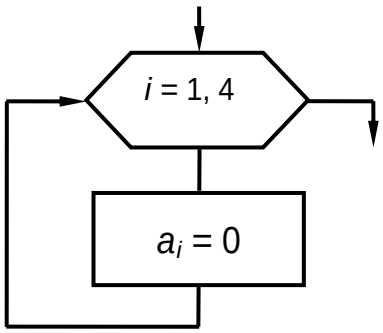


Рис. 12.1

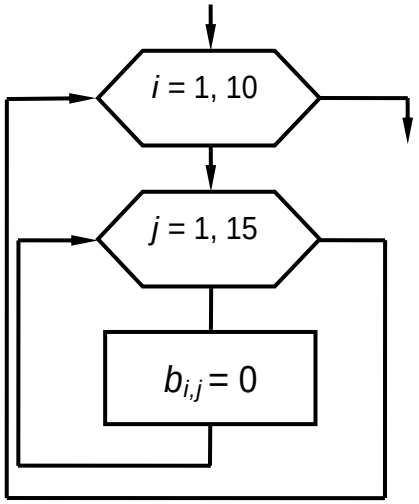


Рис. 12.2

```
For I : = 1 to 4 do A[I] : = 0 ;
```

Для инициализации двумерного массива используется вложенный оператор for (рис. 12.2):

```
for I : = 1 to 10 do
For J : = 1 to 15 do
    B[I, J] : = 0 ;
```

Паскаль не имеет средств ***ввода/вывода элементов массива*** сразу, поэтому вод/вывод их значений производится поэлементно, чаще всего с помощью оператора Read или ReadLn с использованием оператора организации цикла for, например (рис. 12.3, 12.4):

```
For I : = 1 to 4 do
  ReadLn (A[I]);
```

```
For I : = 1 to 10 do
  for J : = 1 to 15 do
    ReadLn (B[I, J]);
```

В этих примерах использовался оператор ReadLn, поэтому каждое вводимое значение элементов массива будет набираться с новой строки.

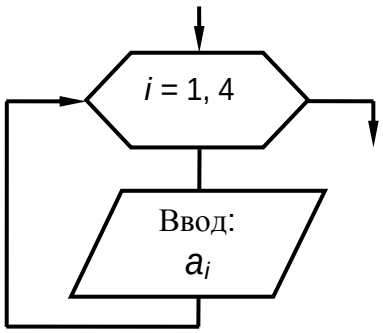


Рис. 12.3

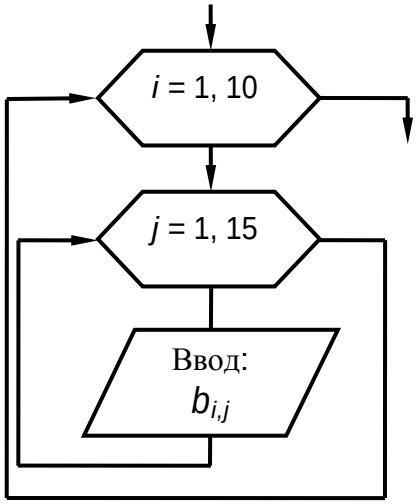


Рис. 12.4

Можно ввести и значения отдельных элементов, а не всего массива, например: Read (A[3]), B[6, 9]); - оба значения набираются на одной строке экрана, начиная с текущей позиции расположения курсора.

Вывод значений элементов массива выполняется аналогичным образом, но используются операторы Write или WtiteLn.

Копированием массивов называется присваивание значений всех элементов одного массива всем соответствующим элементам другого массива. Копирование можно выполнить одним оператором присваивания, например A : = D; или с помощью оператора for :

```
For I : = 1 to 4 do A[I] : = D[I];
```

В обоих случаях значения элементов массива D не изменяются. Очевидно, что оба массива должны быть одинаковыми по структуре.

Иногда требуется осуществить **поиск** *в массиве каких-либо элементов, удовлетворяющих некоторым известным (заданным) условиям*. Пусть, например, надо определить, сколько элементов массива А имеет нулевое значение. Для ответа на этот вопрос введем дополнительную переменную К и определим ее значение по алгоритму рис. 12.5:

```
K : = 0;
For I : = 1 to 4 do
   If A[I] = 0 then
   K : = K + 1;
```

После выполнения данного цикла переменная К будет иметь значение, равное числу элементов массива А с нулевыми значениями.

Перестановка значений элементов массива осуществляется с помощью дополнительной переменной того же типа, что и базовый тип массива. Например, требуется поменять значения первого и четвертого элементов массива А:

```
Vs : = A[4]; { Vs – вспомогательная переменная }
A[4] : = A[1];
A[1] : = Vs;
```

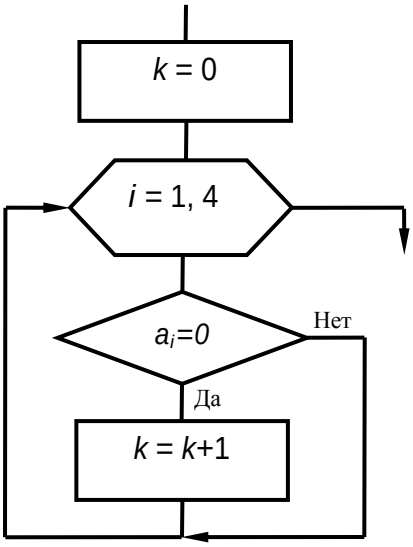


Рис. 12.5

Литература

17. Вальвачев, А.Н., Крисевич, В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС. – Мн.: Выш. шк., 1989. – 223 с.
18. Паскаль для персональных компьютеров / Ю.С. Бородич [и др.]. – Мн.: Выш. шк., 1991. – 365 с.
19. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Госстандарт, 1990. – 28 с.
20. Офицеров, Д.В., Старых, В.А. Программирование в интегрированной среде Турбо-Паскаль. – Мн.: Беларусь, 1992. – 240 с.

МОДУЛЬ М7 – «ПРОГРАММИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ С ИСПОЛЬЗОВАНИЕМ ПОДПРОГРАММ »

Лабораторная работа № 13
Программирование вычислительных процессов с использованием подпрограммы FUNCTION

Цель работы: *Приобретение практических навыков составления программ решения зада с использованием подпрограммы-функции FUNCTION .*

Постановка задачи

Разработать блок-схему алгоритма и составить Паскаль-программу решения задачи с использованием подпрограммы-функции *FUNCTION* по варианту условия, определяемому номером бригады. Числовые значения переменных задать самостоятельно, представив их в виде таблицы сразу после условия задачи.

Варианты заданий

1.Даны числа s и t . Вычислить

$Y = h(s, t)+\sin(h^2(s-t, st))$, *где* $h(a,b)= a+b^2+b+a^2 - (a-b)^2$.

2. Даны действительные числа s и t . Вычислить

$Y = f(t, -2s, 2.17) + f(1.2,t,s-t)$, *где* $f(a,b,c) = \frac{2a-b-\sin c}{5+|c|}$.

3.Вычислить функцию $y = \sin t + \sum_{i=1}^{3} a_i$, где вычисление суммы $\sum_{i=1}^{n} b_i$ организовать по подпрограмме.

4.Даны вещественные числа x, y, z. Вычислить

$y = f(x+y, y^2, z -1) - f(x^2, y^2, z^2)$, *если* $f(a, b, c) = \frac{\sin a-\cos b}{a+b-c}$.

5. Даны числа **a,** b, **c**. Вычислить

У = (max(a, a+b) + max(a, b + c)) / (1 + max(a+b*c, 1.5)) , где max(x, y) оформить в виде подпрограммы.

6. Даны действительные числа s и t . Вычислить

$Y = g(1.2, 5)+g(t, s) - g(2s-l, s-t)$, *где* $g(a, b) = (a^2 + b^2)/(a^2 +2ab + 3b^2 +4)$.

7. Вычислить функцию $z = a+b/a + \prod_{i=1}^{3} c_i$, *где* вычисление произведения $\prod_{i=1}^{m} f_i$ организовать по подпрограмме.

7. Даны числа x и y. Вычислить

$$Y = f(x, z) - f(x+z, z^2), \quad \textit{если} \quad f(a, b)=tg^3(a/2 + \pi/4) - e^b.$$

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическими сведениями (см. Приложение 7).
2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:

37) номер и название работы;
38) цель работы;
39) постановку задачи;
40) блок-схему алгоритма;
41) таблицу идентификаторов;
42) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:

22. Набрать на клавиатуре текст Паскаль-программы .
2. Произвести компиляцию исходной программы.
3. В случае обнаружения ошибок отредактировать программу с ее по- следующей повторной компиляцией.
4. Запустить программу после сообщения об ее успешной компиляции.
5. Ввести исходные данные для получения окончательного результата.
6. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

1. В чем различие между подпрограммами *Function* и *Procedure* ?

23. Как оформляется подпрограмма *Function* ?
24. Как осуществляется обращение к подпрограмме *Function* ?
25. Сколько выходных параметров может возвращать подпрограмма *Function* ?
26. Какие действия допустимы над массивами данных?

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:

1. Экспериментальные результаты в виде распечатки (или запись с экрана дисплея) текста отлаженной Паскаль-программы и результатов счета.
2. Выводы по работе.

Лабораторная работа № 14
Программирование вычислительных процессов с использованием подпрограммы PROCEDURE

Цель работы: *Приобретение практических навыков составления программ решения задач, содержащих обработку двумерных массивов данных .*

Постановка задачи

Разработать блок-схему алгоритма и составить Паскаль-программу решения задачи с использованием подпрограммы *Procedure* по варианту условия, определяемому номером бригады. Числовые значения переменных задать самостоятельно, представив их в виде таблицы сразу после условия задачи.

Варианты заданий

1. Ввести с клавиатуры три одномерных массива, в каждом из них определить наименьший элемент и определить среднее арифметическое этих элементов. Определение минимального элемента организовать в подпрограмме *Procedure* с использованием параметров переменных.

2. Ввести с клавиатуры три одномерных массива, в каждом из них определить наибольший элемент и определить среднее арифметическое этих элементов. Определение максимального элемента организовать в подпрограмме *Procedure* с использованием параметров переменных.

3. Ввести три одномерных массива и определить среднее арифметическое и количество положительных элементов каждого массива. Посчитать сумму средних арифметических и общее количество положительных элементов. Определение среднего арифметического положительных элементов и их количество организовать в подпрограмме *Procedure* с использованием
параметров переменных.

4. Аналогично первому, только определить наименьшие элементы среди положительных.

5. Аналогично первому, только определить наибольшие элементы среди отрицательных.

6. Ввести с клавиатуры три двухмерных массива, определить сумму

элементов каждой строки массивов, записать эти суммы в одномерные массивы и вывести на экран напротив каждой строки. Определение суммы организовать с помощью подпрограммы *Procedure* с использованием
параметров переменных.

7. Аналогично шестому, только определить произведение элементов столбцов.

8. Аналогично шестому, только в каждой строке определить максимальный и минимальный элементы и поменять их местами. Вывести исходные массивы, значения и позиции минимальных и максимальных элементов и преобразованные массивы.

Содержание лабораторной работы

Лабораторная работа включает:

1. Ознакомление с теоретическими сведениями (см. Приложение 7).
2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:

1) номер и название работы;
2) цель работы;
3) постановку задачи;
4) блок-схему алгоритма;
5) таблицу идентификаторов;
6) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:

1. Набрать на клавиатуре текст Паскаль-программы .
2. Произвести компиляцию исходной программы.
3. В случае обнаружения ошибок отредактировать программу с ее последующей повторной компиляцией.
4. Запустить программу после сообщения об ее успешной компиляции.
5. Ввести исходные данные для получения окончательного результата.
6. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

5. Как оформляется подпрограмма *Procedure* ?
6. Что такое фактические и формальные параметры?
7. Как осуществляется обращение к подпрограмме *Procedure* ?
8. Сколько выходных параметров может возвращать подпрограмма *Procedure* ?

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:

1. Экспериментальные результаты в виде распечатки текста отлаженной Паскаль-программы и результатов счета .
2. Выводы по работе.

Приложение 7 ***(к модулю М7)***

Теоретические сведения к лабораторным работам:

№ 13 - **«ПРОГРАММИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ С ИСПОЛЬЗОВАНИЕМ ПОДПРОГРАММЫ** *FUNCTION* », ***№ 14*** - **«ПРОГРАММИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ С ИСПОЛЬЗОВАНИЕМ ПОДПРОГРАММЫ** *Procedure* »

Программирование алгоритмов с использованием подпрограмм

В 1957 г. была описана М.Уилксом концепция подпрограмм, которая получила широкое распространение практически во всех языках программирования. ***Подпрограммой*** *называется именованная логически законченная группа операторов языка, которую можно вызывать по имени многократно из разных мест основной программы.* В Паскале для организации подпрограмм используются процедуры и функции, которые подразделяются на две группы: стандартные (встроенные) и определенные пользователем. Стандартные процедуры и функции (например, рассмотренные раньше арифметические функции) являются частью языка и могут вызываться по имени без предварительного описания в программном блоке, а процедуры и функции пользователя разрабатываются самим программистом в соответствии с синтаксисом языка и предварительное описание их обязательно. Процедуры и функции пользователя по структуре похожи с основной программой.

Процедура пользователя. Описание процедуры включает заголовок и тело процедуры (рис. 14.1). Заголовок состоит из ключевого слова Procedure, имени процедуры и необязательного заключенного в круглые скобки списка формальных параметров с указанием их типа. Тело процедуры представляет собой локальный блок, по структуре аналогичный программе.

```
Procedure имя (формальные параметры);
Разделы описаний
  Begin
Раздел операторов
  End;
```

Рис. 14.1 Структура процедуры пользователя.

Для обращения к процедуре из программы используется оператор вызова процедуры, который состоит из имени процедуры и списка фактических параметров, отделенных друг от друга запятыми и заключенные в круглые скобки. При выполнении процедуры формальные параметры заменяются на фактические. Количество, порядок расположения и тип формальных параметров соответствуют количеству, порядку расположения и типу фактических параметров. Если процедура возвращает в программу какие-то значения, то соответствующие переменные в заголовке процедуры должны быть описаны как параметры-переменные с использованием слова Var, например

```
Procedure W1 (Var X1, X2 : integer; R1, R2 :integer);
```

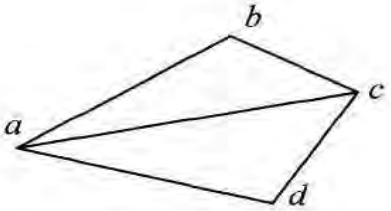


Рис. 14.2 - Выпуклый четырёхугольник

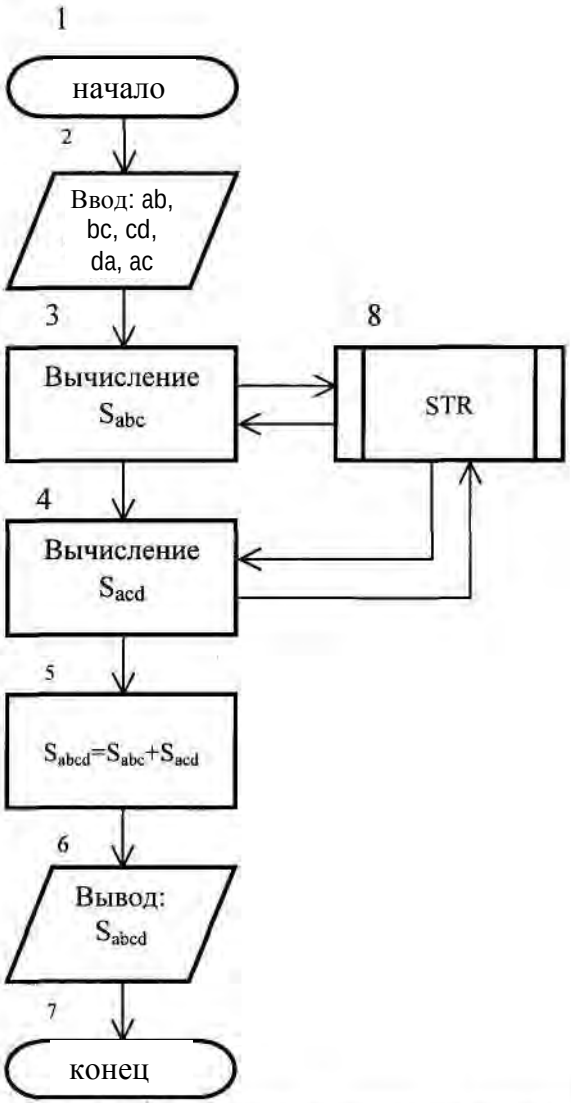


Рис. 14.3 – Схема алгоритма основной программы

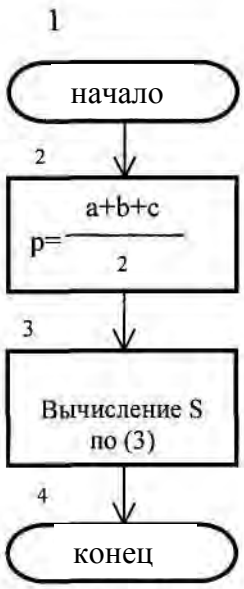


Рис. 17

Пример. Пусть требуется составить программу вычисления площади (Sabcd) выпуклого четырехугольника (рис. 14.2), заданного длинами четырех сторон ab, bc, cd, da и диагонали **ac** .

Диагональ делит выпуклый четырехугольник на два треугольника, к которым применима формула Герона по вычислению площади треугольника S :

$$S = p(p-a)(p-b)(p-c), \qquad (3)$$

где a, b, **c** - длины сторон треугольника;

p - полупериметр треугольника.

Решим поставленную задачу путем вычисления площади Sabcd как сумму площадей треугольников Sabc и Sacd согласно алгоритму рис. 14.3, в котором вычисление площади треугольника по формуле Герона оформим в процедуру с именем STR согласно алгоритму рис. 14.4.

Тогда один из вариантов программы на Паскале будет иметь вид:

```
Program Proc ;
 Var AB, BC, CD, DA, AC, Sabc, Sacd, Sabcd : real;
Procedure STR (Var S : real; a, b, c : real);
 Var p : real;
 begin
  P : = (a + b + c) / 2 ;
  S: = Sqrt(p*(p-a)*(p-b)*(p-c))
 end ;

 Begin ReadLn ( AB, BC, CD, DA, AC);
    STRK ( Sabc, AB, BC, AC ) ;
    STRK ( Sacd, AC, CD, DA ) ;
    Sabcd : = Sabc + SacB ;
    Write (' Sabcd = ', Sabcd )
 End.
```

Функция пользователя. Она состоит (рис. 18) из заголовка и тела функции. Заголовок содержит ключевое слово Funktion , имя функции, заключенный в круглые скобки необязательный список формальных параметров с указанием их типа, а после скобок через двоеточие тип возвращаемого функцией значения.

Например:

```
Funktion W2 (X, Y, T : integer): real;
```

Тело функции представляет собой локальный блок, по структуре аналогичный программному блоку (рис. 14.5).

```
Funktion имя (формальные параметры ): тип результата ;
  Разделы описаний
begin
  Раздел операторов
end ;
```

Рис. 14.5. Структура функции пользователя.

В разделе операторов должен находиться, по крайней мере, один оператор, присваивающий имени функции пользователя значение.

Обращение к функции из основной программы осуществляется по имени с указанием списка фактических параметров, которые должны соответствовать формальным

параметрам и иметь тот же тип. Имя функции, встречающееся в выражениях основной программы, называется указателем функции или обращением к функции.

Функция имеет следующие отличия от *процедуры:*

1) она возвращает результат своей работы (значение) в точку вызова;
2) возвращает результат через имя функции;
3) при этом возвращает только один результат;
4) имя функции входит в выражение как операнд (этим именем она вызывается), а *имя процедуры не может входить в выражение в качестве операнда; процедура вызывается отдельным оператором, может возвращать несколько результатов, при этом возвращает результаты через формальные параметры, а не через имя.*

Пример. Решим задачу предыдущего примера, оформив вычисление площади треугольника по формуле Герона с помощью функции пользователя с именем STR , а не процедуры. Тогда программа на Паскале будет иметь вид:

```
Program Funk ;
 Var AB, BC, CD, DA, AC, Sabc, Sacd, Sabcd : real;
  Function STR ( a, b, c : real): real;
   Var S, p : real;
   begin p : =  (a + b + c ) / 2 ;
         S : = Sqrt ( p * ( p - a ) * ( p - и ) * ( з - c ) ) ж
        STR: = S ;
   end ;
 Begin ReadLn (AB, BC, CD, DA, AC);
       Sabcd : = STR ( AB, BC, AC ) + STR ( AC, CD, DA ) ;
       WriteLn ( ' Sabcd = ', Sabcd )
 End.
```

Оператор Sabcd : = STR (AB, BC, AC) + STR (AC, CD, DA) ; можно также записать и в виде трех операторов присваивания (согласно блокам 3-5 алгоритма рис. 15):

```
Sabc : = STR (AB, BC, AC) ;
Sacd : = STR (AC, CD, DA);
Sabcd : = Sabc + Sacd ;
```

Литература

21. Вальвачев, А.Н., Крисевич, В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС. – Мн.: Выш. шк., 1989. – 223 с.
22. Паскаль для персональных компьютеров / Ю.С. Бородич [и др.]. – Мн.: Выш. шк., 1991. – 365 с.
23. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Госстандарт, 1990. – 28 с.
24. Офицеров, Д.В., Старых, В.А. Программирование в интегрированной среде Турбо-Паскаль. – Мн.: Беларусь, 1992. – 240 с.

МОДУЛЬ М8 – «ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ С ИСПОЛЬЗОВАНИЕМ ФАЙЛОВ»

Лабораторная работа № 15
Программирование алгоритмов с использованием файлов

Цель работы: Приобретение практических навыков по программированию алгоритмов с использованием файлов, по применению стандартных процедур и функций при чтении данных из файла и при сохранении результатов вычислений по программе в файл.

Постановка задачи

Разработать блок-схему алгоритма и составить Паскаль-программу решения задачи с использованием обращений к файлам по варианту условия, определяемому номером бригады .

Варианты заданий

Произвести считывание одномерного массива из файла Data.dat, расположенного на дискете (если нет такого файла, то создайте его в любом каталоге). С данной информацией произвести заданные по вариантам действия, результаты преобразований вывести на экран и записать в текстовый файл на жёстком диске.

1. Найти два элемента, имеющие наибольшие значения и пе реставить их в начало массива.
2. Найти два элемента, имеющие наименьшие значения и пе реставить их в конец массива.
3. Найти максимальный и минимальный элементы массива и поменять их местами.
4. Отсортировать элементы массива по возрастанию.
5. Отсортировать элементы массива по убыванию.
6. Определить элемент массива, имеющий значение, наибо лее близкое к среднему арифметическому элементов масси ва и поменять его местами со средним элементом.
7. Найти первый максимальный элемент и переставить его в начало массива.
8. Найти первый минимальный элемент и переставить его в конец массива.

Содержание лабораторной работы

Лабораторная работа включает:
1. Ознакомление с теоретическими сведениями (см. Приложение 8).
2. Оформление отчета по лабораторной работе, который должен содержать следующие пункты:

43) номер и название работы;
44) цель работы;
45) постановку задачи;
46) блок-схему алгоритма;
47) таблицу идентификаторов;
48) текст исходной Паскаль-программы.

Порядок выполнения работы

Последовательность выполнения работы следующая:

27. Набрать на клавиатуре текст Паскаль-программы .
2. Произвести компиляцию исходной программы.
3. В случае обнаружения ошибок отредактировать программу с ее последующей повторной компиляцией.

28. Запустить программу после сообщения об ее успешной компиляции.
29. Ввести исходные данные для получения окончательного результата.
30. Распечатать текст Паскаль-программы и результаты.

Контрольные вопросы

Содержание отчета

Отчет по выполненной работе (кроме протокола) должен содержать следующие сведения:

1. Экспериментальные результаты в виде распечатки (или запись с экрана дисплея) текста отлаженной Паскаль-программы и результатов счета .
2. Выводы по работе.

Приложение 8 *(к модулю М8)*

Теоретические сведения к лабораторной работе

№ 15 - «ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ С ИСПОЛЬЗОВАНИЕМ ФАЙЛОВ»

Файлы

Под **файлом** понимается любой набор элементов одного и того же типа. Число элементов, называемое **длиной файла**, не фиксируется (в этом состоит основное отличие файла от массива). Если файл не содержит ни одного элемента (*его длина равна нулю*), то он называется **пустым**.

Файлы подразделяются по методу доступа к их элементам на файлы последовательного доступа и прямого доступа. В файлах *последовательного доступа* каждый элемент становится доступным только после перебора всех предыдущих элементов. Файлы *прямого доступа* позволяют обращаться к каждому элементу непосредственно по его порядковому номеру в файле.

Файлы по отношению к программе могут быть внутренними и внешними. *Внутренние файлы* – это такие файлы, которые создаются,

используются и существуют только во время работы данной программы (например, исходные данные в стандартном файле ввода *Input* или результаты вычислений в файле вывода *Output* . Файлы, которые существуют вне программы, называются *внешними* (например, файлы на магнитных дисках).

Внешние файлы должны быть описаны в разделе описаний программы по формату:

1) Type *имя типа* = file of *базовый тип* ;
   Var *имя файла* : *имя типа* ;

или 2) Var *имя файла* : file of *базовый тип* ;

В качестве *базового типа* элементов файла можно использовать любой тип данных (как *простой*, так и *сложный*), за исключением типа file.

Пример: Type Mas = array (1 .. 10) of real ;
Var F1 : file of integer ;
F2 : file of Mas ; {каждый элемент файла - массив}

Доступ к элементам файла осуществляется через *указатель файла* (буферную переменную). При чтении или записи этот указатель перемещается к следующему элементу и делает его доступным для обработки. В каждый момент доступен для записи (или для чтения) только тот элемент файла, на который установлен указатель.

Стандартные процедуры для работы с файлами

Assign (F, Name); - связывает имя файловой переменной F в программе с именем внешнего файла на диске Name . Здесь Name - выражение строкового типа вида

'*диск* : \ *имя каталога* \ *имя подкаталога* \ … \ *имя файла*'

Если в параметре Name имена диска и подкаталогов не указаны, то выбирается текущий диск и текущий каталог. Процедура *Assign* используется до начала работы с файлом (до его открытия процедурой *Reset* или *Rewrite*).

Reset (F); - открытие файла *для чтения*.

Rewrite (F); - открытие файла *для записи*.

Read (F, x1, x2, … , xN); - считывает в переменную x1 один элемент файла F (или несколько Элементов в переменные x1, x2, …, xN), начиная чтение с элемента, на который установлен указатель

Write (F, x1, x2, … , xN); - записывает одно (x1) или более (x1, … , xN) значений переменных в файл F, начиная с той позиции, на которую установлен указатель.

Close (F); - закрывает файл .

Seek (F, N); - осуществляет прямой доступ к элементам файла F . Здесь N - целая положительная константа, соответствующая передаваемому номеру элемента в файле (если N = 0, то первый элемент файла, N = 1 – второй

элемент и т.д.). Процедура *Seek* лишь перемещает указатель файла к элементу с номером N.

Eof - логическая функция, тестирующая конец файла (возвращает *True*, если указатель стоит в конце файла).

В Турбо Паскале можно открыть файл для **чтения**, для **записи информации**, а также для **чтения и записи** одновременно.

Чтение файла. Под *чтением файла* понимается ввод данных из внешнего файла (его еще называют *входным файлом*) в оперативную память ПК. Для этого в программе необходимо выполнить следующие действия:

1) открыть файл для чтения (*Reset*);
2) ввести данные файла в программу (*Read*);
3) закрыть файл для чтения (*Close*).

Запись в файл. Под *записью в файл* понимается вывод результатов программы из оперативной памяти ПК на диск, т.е. создание нового файла на внешнем устройстве (его еще называют *выходным файлом*). Для этого необходимо в прграмме выполнить такие действия:

1) открыть файл для записи (*Rewrite*);
2) вывести данные из программы в файл (*Write*);
3) закрыть файл для записи (*Close*).

Пример. Составим программу формирования файла F ,состоящего из целых чисел 1, 2. 3. 4. 5.

```
Program Pr;
 Var F : file of integer;
       I : integer;
 Begin Assign (F, 'F');
 Rewrite (F);
 For I := 1 to 5 do Write (F, I);
 Close (F);
 End.
```

Запись и чтение файла. Пусть необходимо сначала создать некоторый файл, а затем найти в нем определенный элемент. Для этого надо выполнить следующие действия:

1) открыть файл для записи (*Rewrite*);
2) записать данные в файл (*Write*);
3) закрыть файл (*Close*);
4) открыть файл для чтения (*Reset*);
5) читать файл, пока не будет найден нужный элемент (Read);
6) закрыть файл. (*Замечание*: пункт 3) выполнять не обязательно).

Текстовые файлы

Особым типом файлов являются текстовые файлы. Эти файлы содержат некоторый текст, который состоит из обычных символов. Символы текстового файла разбиты на строки, подобно тому, как они записываются на бумаге.

Описание текстового файла имеет следующий вид:

Имя файла : text ;

Здесь *text* - стандартный идентификатор, точно такой же, как real, char и т.д. Отличие текстового файла (типа *text* от файла типа *char* (или *string*) заключается в том, что текстовый файл состоит из последовательности строк различной длины, каждая из которых содержит величины типа *char* и заканчивается специальным символом конца строки. Для определения конца строки используется логическая функция Eoln (F); . Она принимает значение *True* , если достигнут конец строки, и значение *False* - в противном случае.

Для работы с текстовыми файламис наряду с вышерассмотренными процедурами и функциями (кроме *Seek*) используются еще такие стандартные процедуры:

Append (F); - открытие уже существующего текстового файла для добавления данных в конец файла;

Writeln (F); - завершение текущей строки текстового файла (при его записи);

Readln (F); - переход к началу следующей строки текстового файла (при его чтении);

Writeln (F, x1, x2, … , xN); - запись в текстовый файл F значений переменных x1, x2, …, xN с завершением текущей строки;

Readln (F, x1, x2, … , xN); - чтение N символов файла с переходом к новой строке.

Рассмотрим программирование алгоритмов с использованием файлов на следующих конкретных примерах.

Пример 1. Введите с клавиатуры текст и сохраните его в виде файла "Proba.txt" .

```
program  zadacha7_I;
  var f:text;
     s:string;
Begin
  writeln('Введите текст');
  readlh(s);
  assign(f,'proba.txt'); {связываем файловую переменную f с именем файла}
  rewrite(f); {открываем файл для записи, файл при этом создаётся}
  writeln(f,s);  {выводим информацию в файл}
  close(f);      {закрываем файл)
```

```
End.
```

Примечание: Выйдите из Pascal, найдите файл Proba.txt, просмотрите его содержимое. Загрузите Pascal.

Пример 2. Введите с клавиатуры текст и допишите его в существующий файл 'Proba.txt' .

```
program zadacha7_2;
  var f:text;
     s: string;
Begin
   writeln('Введите текст');
   readln(s);
   assign(f,'proba.txt');  {связываем, файловую переменную f с именем файла}
    append(f); {открываем файл для дополнения}
    writeln(f,s); {выводим информацию в файл}
    close(f); {закрываем файл}
End.
```

Примечание: Выйдите из Pascal и просмотрите содержимое файла. Загрузите Pascal.

Пример 3. Выведите на экран содержимое файла 'Proba.txt'.

```
program zadacha7_3:
  var p:text;
     t:string;
Begin
   assign(p,'proba.txt);   {связываем файловую переменную f с именем файла}
    reset (p);    {открываем файл для чтения}
    while not eof(p)  do     {пока не конец файла}
          begin
       readln(p,t);    {считываем из фата}
       writeln(t);      {выводим информацию на экран}
     end;
close(p);       {закрываем файл)
End.
```

Пример 4. Введите несколько строк в файл процедурами Write, а затем процедурами Writeln. Просмотрите файл и разберитесь, в чём отличие между этими процедурами.

Пример 5. Сохраните в файле 'cisla.in' N натуральных чисел.

```
program zadacha7_5;
```

```
 var      i,n,a: integer;
                f:text;
Begin
   assign(f,'cisla.in');
   rewrite(f);
   writeln ('Введите n'); readln (n);
   for i:=i to n do
     begin
       writeln('Введите ',i,' число');
       readln(a);
       writeln(f,a);
     end;
close(i);
End.
```

Пример 6. Найти сумму чисел, находящихся в файле "cisla.in".

```
program zadaclia7_6;
var     s,i,n,a: integer:
            f:text;
Begin
   assign(f,'cisla.in');
   reset(f);
s:=0;
while not eof(f) do
  begin
    readln(f,a);
    s:=s+a;
  end;
  writeln('Сумма =',s);
close(f);
End.
```

Пример 7. Создать файл 'cisla.nat', в который записать более 10 чисел. Десять первых чисел считать из файла в массив. Вывести массив на экран.

```
program zadacha7_7;
  const n=10;
  var   k,i,a:integer;
          f:text;
          m: array [l..n] of integer;
Begin
assign(f,'cisla.nat');
rewrite(f);                    {открываем файл для ЗАПИСИ}
writeln('Введите кол-во чисел > 10');
```

```
read ln (k); for i:=l to к do
     begin
     writeln ('Введите ',i,' число');
      readln(a);
      writeln(f,a);                    {сохраняем в файл}
   end;
close(f); reset (f);    {закрываем файл и открываем его для ЧТЕНИЯ}
i:-1;
while not eof(f) and (i<=l 0) do        {пока не конец файла и i<10 }
begin
      readln(f,a);                    {считываем из файла}
      m[i]:=a                         {сохраняем в массив}
      inc(i);
  end;
close(f);                             {закрываем файл}
for i:=1 to 10 do
     writc(m[i],'');
End.
```

Пример 8. Из файла 'cisla.nat' перенести в файл 'chet.cis' все чётные числа. Вывести содержимое файла 'chet.cis' на экран.

```
program zadacha7_8;
var     a: integer;
    f1,12:text;
 Begin
assign(f1,'cisla.nat');
assign(f2,'chet.cis');
reset (f1);                  {открываем файл для чтения}
rewrite(f2);                 {открываем файл для записи}
while not eof(f1) do      (пока не коней файла!
begin
readln(fl,a);                {считываем из файла/1}
if a mod 2 =0 then writeln(f2,a);     /если чётное, то сохраняем в файл/2}
end;
close(fl);close(f2);         {закрываем файл}
                    {выводим содержимое фата 'chet.cis'}
reset (12);                   {открываем файл д./я чтения}
while not eof(f2) do begin
readln(f2,a):                 {считываем из фа/па/2}
writeln(a); end;
close(12); End.
```

Литература

1. Вальвачев, А.Н., Крисевич, В.С. Программирование на языке Паскаль для персональных ЭВМ ЕС. – Мн.: Выш. шк., 1989. – 223 с.
2. Паскаль для персональных компьютеров / Ю.С. Бородич [и др.]. – Мн.: Выш. шк., 1991. – 365 с.
3. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – М.: Госстандарт, 1990. – 28 с.
4. Офицеров, Д.В., Старых, В.А. Программирование в интегрированной среде Турбо-Паскаль. – Мн.: Беларусь, 1992. – 240 с.

Методические указания по выполнению курсовой работы по дисциплине «Информатика»

Каждому студенту выдаётся индивидуальное задание по курсовому проектированию, в котором математически сформулирована задача. Студент должен разработать схемы алгоритмов основной программы и подпрограмм, а затем реализовать их на алгоритмическом языке, например Турбо-Паскаль.

Стандартное задание на проектирование: *Вычислить заданную функцию и вывести результаты расчётов в виде таблицы и графика (y=f(j)). Осуществить вывод результатов в файл*

Письменная часть работы оформляется в виде расчётно-пояснительной записки, включающей в себя:

Титульный лист;

Лист задания на проектирование;

Содержание расчётно-пояснительной записки;

Введение

1.1.Анализ заданной функции и разработка схемы алгоритма по ее вычислению

1.2.Программирование отдельных блоков и структур алгоритмов.

1.3.Полная программа в соответствии с разработанным алгоритмом.

1.4.Анализ полученных результатов расчета.

Заключение

Список использованных источников

Расчётно-пояснительная записка предоставляется на листах формата А4 (20 – 30 листов). Текст должен быть либо написан чётким разборчивым почерком, либо набран на компьютере (Times New Roman, размер - 14 пт, интервал – 1,5 строки). Схемы алгоритмов должны быть выполнены аккуратно в соответствии с ГОСТ 19.701-90. Кроме схем алгоритмов, графическая часть включает в себя график искомой функции, построенный программными средствами Турбо-Паскаля.

Кроме того, для проверки работоспособности программы преподавателем студент должен предоставить в электронном виде файл с расширением *.pas, содержащий текст программы.

Предлагаемое в курсовой работе задание является частным случаем инженерной задачи.

Методику решение инженерных задач любой сложности можно представить в виде последовательности шести этапов.

На первом этапе условие задачи записывается в виде последовательности формул или уравнений, необходимых для решения задачи.

На втором этапе выбирается такой метод решения задачи, который сведет поиск результата к выполнению последовательности элементарных математических операций (сложение, вычитание, умножение, деление). Для

большинства практических задач разработаны различные математические методы решения дифференциальных уравнений, например, Эйлера, Рунге-Кутта и т.д. Выбор того или иного метода осуществляется на некоторых критериях, например, минимальное время вычислений, заданная точность вычислений, интегральные или квадратичные критерии. Более подробное изучение методов и критериев оценки будет на старших курсах обучения.

На третьем этапе на основании выбранного метода разрабатывается алгоритм – общая схема решения задачи. В настоящее время существует несколько определений алгоритма, одним из них является следующее. Алгоритмом называется предписание, определяющее содержание и последовательность операций, преобразующих исходные данные в искомый результат. В инженерной деятельности широкое распространение получил схемный способ описания алгоритма, при котором алгоритм представляется в виде блоков, каждый из которых выполняет определенное действие, и направленных связей между ними. Часто схема алгоритма называется еще блок-схемой. Размеры блоков, их форма и назначение определяются по ГОСТ19.701-90.

На четвертом этапе, на основании блок-схемы алгоритма, составляется программа решения задачи на определенном алгоритмическом языке, в нашем случае Турбо-Паскаль.

На пятом этапе вводят программу в память ЭВМ, транслируют, редактируют, проверяют правильность ввода и написания.

На шестом этапе вводят исходные данные, выполняем вычисления по программе и получаем результат.

Методика разработки алгоритмов на основании имеющейся математической задачи, а также реализация различных алгоритмических структур на базе языка Турбо-Паскаль подробно изложена в «Конспекте лекций» часть 2-я (теоретический раздел) и во второй части лабораторного практикума (практический раздел).

Примерный перечень контрольных вопросов и заданий для самостоятельной работы

1. Информация. Меры информации. Информационные технологии.
2. История развития вычислительной техники.
3. Системы счисления. Перевод целых чисел из одной системы счисления в другую.
4. Перевод дробных чисел из одной системы счисления в другую.
5. Способы представления информации в ЦВМ.
6. Основные типы современных компьютеров.
7. Принцип работы ЦВМ. Структура фон Неймана. Структура современных ПК.
8. Микропроцессоры. Характеристики. Охлаждение процессоров.
9. Материнская плата. Контроллеры. Порты. Шины.
10. Память вычислительной машины. Внутренняя память компьютера.
11. Дисковая память. НГМД, НЖМД.
12. Организация хранения информации на магнитных дисках.
13. Принцип записи и чтения информации с помощью лазера. Накопители на оптических дисках.
14. Блок питания и корпус ПК. Полная внутренняя структура ПК.
15. Периферийные устройства. Общая классификация.
16. Клавиатура. Манипулятор.
17. Общая классификация мониторов. Электронно-лучевые мониторы.
18. Мониторы на плоских панелях.
19. Общая классификация печатающих устройств. Матричные принтеры.
20. Струйные и электрографические принтеры.
21. Плоттеры. Принцип действия. Характеристики.
22. Сканеры. Принцип действия. Характеристики.
23. Дигитайзеры. Принцип действия. Характеристики.
24. Каналы связи. Классификация Характеристики.
25. Аналоговые модемы. Виды модуляции.
26. Модемы для цифровых каналов связи.
27. Программное обеспечение информационных технологий. Виды программного обеспечения.
28. Операционная система. Основные семейства ОС.
29. Файл. Файловая система. Атрибуты файлов.
30. Каталоги. Логические дисководы.
31. ОС Windows. Элементы интерфейса. Работа с файлами и программами. Файловые менеджеры. Сжатие информации. Дефрагментация. Антивирусные программы.
32. Компьютерная графика. Основные виды графической информации.
33. Системы управления базами данных
34. Компьютерные презентации
35. Компьютерные сети. Общие сведения. Топологии сетей. Способы адресации.

36. Разновидности компьютерных сетей. Всемирная сеть *Internet*. *Web*-технологии.
37. Методика программирования инженерных задач на ЦВМ
38. Алгоритмический язык Турбо-Паскаль. Структура Паскаль-программы. Основные элементы
39. Линейные структуры. Разветвляющиеся структуры и их программирование
40. Циклические структуры
41. Арифметические циклы и их программирование
42. Циклы с предусловием и их программирование
43. Циклы с постусловием и их программирование
44. Подпрограммы-процедуры
45. Подпрограммы-функции
46. Формальные и фактические параметры подпрограмм. Область видимости идентификаторов. Параметры-переменные, параметры-значения и нетипизированные параметры
47. Модуль. Структура модуля
48. Массивы
49. Текстовые файлы. Организация работы с текстовыми файлами
50. Типизированные и нетипизированные файлы. Организация работы с типизированными и нетипизированными файлами
51. Графика в Турбо-Паскале. Графические координаты. Инициализация графического режима
52. Основные процедуры и функции работы с графикой: построение изображений, управление графическим курсором, вывод текста
53. Порядковые типы языка Турбо-Паскаль
54. Вещественные типы языка Турбо-Паскаль
55. Строковый тип данных
56. Тип данных «Запись»
57. Тип данных «Множество»
58. Динамические переменные. Работа с динамическими переменными
59. Динамические структуры данных. Графы и деревья
60. Динамические структуры данных. Связные списки
61. Динамические структуры данных. Стек. Добавление элемента в стек
62. Динамические структуры данных. Стек. Извлечение элемента из стека
63. Динамические структуры данных. Очередь. Добавление элемента в очередь
64. Динамические структуры данных. Очередь. Извлечение элемента из очереди
65. Обработка структурных данных: сортировка элементов массива.
66. Решение прикладных задач: построение графика производной и интегральной кривой.
67.

Примерный перечень задач для самостоятельной подготовки к экзамену

1. Рассчитать по формуле Горнера:
$y = b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0$,
b_i - массив случайных чисел из диапазона (-12;12)
x – вводится с клавиатуры.
2. Вычислить:
$$z_i = \begin{cases} a^{2i} + 10, & i \le 4 \\ \sqrt{2i^2 + 4}, & i > 4 \end{cases}$$
i=1,2, …, 15.
3. Сформировать множество чётных чисел, кратных пяти в диапазоне от 1 до 80 и проверить, входит ли туда случайное число b из того же диапазона.
4. Вычислить: $y_i = 3 \cdot sh(4 - x_i)$,
Параметр x_i изменяется от 0,2 до 1,5 с шагом h = 0,3. Вычисление гиперболического синуса sh(t) организовать по подпрограмме-функции.
5. Дан массив:
$$a_i = 2\sin\frac{i}{2} + 3\cos\frac{i}{3}$$
$i = 1,2,\ldots,20$
Отсортировать элементы массива по убыванию.
6. Рассчитать среднее эквивалентное значение элементов массива ai, удовлетворяющих условию $20 > a_i > 2$. Элементы массива считываются из файла A:\data2.dat
i=1, 2, …, 40.
7. Записать в виде одномерного массива цифры введённого с клавиатуры натурального числа M.
8. Дан одномерный массив, состоящий из 12 элементов, вводимых с клавиатуры. Отсортировать массив по убыванию. Найти разность между максимальным и минимальным элементами.
9. Вычислить:
$$y_j = \begin{cases} aj^2 - bj - c, & j \le a \\ bj^2 + cj + 1, & j > a \end{cases}$$
a,b,c - случайные целые числа из диапазона (3;20), $j = 1,2,3,\ldots,30$.
10. Записать в текстовый файл значения аргумента и функции $y = 2\sin^2 x + 3$;
аргумент x изменяется от 0 до 1 с шагом 0,02.
11. Найти методом рекурсии корень уравнения: $y = 2x^2 - 4x - 1$ на промежутке: $x \in (0;6)$.
12. Рассчитать сумму элементов массива a_i, считываемого из файла A:\data1.dat
i=1, 2, …, 20.
13. Разработать алгоритм и программу вычисления:
$Y = AX + B$,
где $A(6 \times 6)$ - массив случайных чисел из диапазона (-40;40);
$X(6 \times 1)$ - вектор, вводимый с клавиатуры;
$B(6 \times 1)$ - вектор случайных чисел из диапазона (-10;10).
14. Записать в типизированный файл значения функции $y = j^2 + 8$, кратные трем
j=1, 2, …, 20.

15. Разработать алгоритм и программу вычисления:

$Z = A^2 + B$,

где $A(4\times 4)$ - массив $a_{ij} = \sin(\frac{i}{10}) - \cos(\frac{j}{20})$;

$B(4\times 4)$ - массив $b_{ij} = \sin(\frac{i^2 + j^2}{10})$

Вычисление A^2 выполнить с помощью подпрограммы-процедуры.

16. Вычислить

$$z = \begin{cases} \lg(2a+5), & 20 > a \geq 10 \\ a^3 + 2b, & a \geq 20 \text{ или } 10 > a \end{cases}$$

a и b случайные числа из диапазона (3,5; 30,5).

17. В матрице $B(6\times 6)$ случайных чисел из диапазона (-30;30) заменить единицами все положительные элементы.

18. Определить N, за которое y достигнет значения числа A, вводимого с клавиатуры:

$$y = \sum_{i=1}^{N} \frac{2\log_3 i}{i}.$$

19. Ввести с клавиатуры три одномерных массива, в каждом из них определить наименьший элемент среди положительных и определить среднее квадратичное этих элементов. Определение минимального элемента организовать в подпрограмме-процедуре.

20. Определить среднее квадратичное элементов массива a_i. Массив вводится с клавиатуры. Число элементов массива – 12.

21. Вычислить:

$$z_i = \cos\frac{i}{20} + \sin^2\frac{i}{40}$$

$i = 1,2,3,\ldots,30.$

22. Вычислить с использованием подпрограммы-процедуры:

$$L = \frac{\max(a,b,c,d)}{\min(a,b,c,d)} + \max(a-b, c-d)$$

a, b, c, d – случайные числа, вводимые с клавиатуры.

23. Сформировать множество чётных чисел, кратных трём в диапазоне от 1 до 100 и проверить, входит ли туда случайное число b из того же диапазона.

24. Дан двумерный массив:

$$b_{ij} = \begin{cases} \lg 2i - j, & i < j \\ a^{-j} + 2i, & i \geq j \end{cases}$$

$i = 1,2,3,\ldots,10$

$j = 1,2,3,\ldots,10$

Определить значение и номер минимального элемента массива.

25. Вычислить:

$$y = \frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \frac{1}{3\cdot 4} + \ldots$$

суммирование вести до слагаемого, меньшего числа ε, вводимого с клавиатуры.

26. Рассчитать среднее квадратичное значение элементов массива a_i, считываемого из файла A:\data2.dat

i=1, 2, …, 40.

27. Вычислить:

$y = \sum \frac{1}{i!}$, где I = 1, 2, 3, …

суммирование вести до слагаемого, меньшего числа ε, вводимого с клавиатуры.

28. Разработать алгоритм и программу представления в двоичном виде десятичного числа, введённого с клавиатуры.

29. Сформировать множество целых чисел, кратных трём в диапазоне от 12 до 64 и проверить, входит ли туда число a, вводимое с клавиатуры.

30. Определить среднее арифметическое элементов массива ai с чётными номерами. Массив вводится с клавиатуры. Число элементов массива – 24.

31. Дан двумерный массив: $c_{ij} = 3i - j^2$, $i = 1,2,3,\ldots 5$, $j = 1,2,3,\ldots 10$

Найти разность между максимальным и минимальным значением элементов массива

32. Дан двумерный массив: $c_{ij} = 4i^2 - j^3$, $i = 1,2,3,\ldots 10$, $j = 1,2,3,\ldots 8$

Найти разность квадратов между максимальным и минимальным значением элементов массива.

33. Сформировать множество, состоящее из натуральных чисел в диапазоне от 1 до 200, кратных трём и пяти, а затем исключить из этого множества все чётные числа.

34. Вычислить:

$$y = \begin{cases} ax^2 + bx + c, & a \neq 0 \\ bx^2 - cx + 1, & a = 0 \end{cases}$$

a, b, c - случайные числа из диапазона (-5;5).

35. Вычислить:

$$y = \begin{cases} \ln(|f| + |g|), & f + g > 10 \\ ch(f + g), & f + g < 10 \end{cases}$$

Вычисление гиперболического косинуса ch(x) выполнить по подпрограмме-функции.

36. N – случайное число из диапазона (10;70). Преобразовать число N в зависимости от остатка от его деления на 3:

0 - $N = N^2$;

1 - $N = 2 \cdot N$;

2 - $N = 2 \cdot N + 1$.

Министерство образования Республики Беларусь

Учебно-методическое объединение вузов Республики Беларусь по образованию в области автоматизации технологических процессов, производств и управления

УТВЕРЖДАЮ
Первый заместитель Министра образования
Республики Беларусь

_____А.И. Жук

_____

Регистрационный № ТД-_____/тип.

ИНФОРМАТИКА

Типовая учебная программа для высших учебных заведений по специальности 1-53 01 05 Автоматизированные электроприводы

СОГЛАСОВАНО
Председатель учебно-методического
объединения вузов
Республики Беларусь по образованию
в области автоматизации технологических
процессов, производств и управления

_____Г.Н.Здор

_____

СОГЛАСОВАНО
Начальник Управления высшего и
среднего специального
образования
Министерства образования
Республики Беларусь

_____Ю.И. Миксюк

_____

Проректор по учебной и
воспитательной работе
Государственного
учреждения образования
«Республиканский институт
высшей школы»

_____ В.И. Шупляк

_____

Эксперт-нормоконтролер

_____

_____

инск 2010

СОСТАВИТЕЛИ:
Г. И. Гульков, доцент кафедры «Электропривод и автоматизация промышленных установок и технологических комплексов» Белорусского национального технического университета, кандидат технических наук, доцент;
А. В. Миронович, ассистент кафедры «Электропривод и автоматизация промышленных установок и технологических комплексов» Белорусского национального технического университета

РЕЦЕНЗЕНТЫ:
Кафедра электропривода и автоматизации промышленных установок Государственного учреждения профессионального образования «Белорусско-Российский университет»
(протокол № 6 от 12 марта 2010г.);

Кузьмицкий И. Ф. заведующий кафедрой автоматизации производственных процессов и электротехники Учреждения образования «Белорусский государственный технологический университет», кандидат технических наук, доцент.

РЕКОМЕНДОВАНА К УТВЕРЖДЕНИЮ В КАЧЕСТВЕ ТИПОВОЙ:

Кафедрой «Электропривод и автоматизация промышленных установок и технологических комплексов» Белорусского национального технического университета
(протокол №____ от _____ 200_г.);

Научно-методической комиссией Белорусского национального технического университета
(протокол №____ от _____ 200_г.);

Учебно-методическим объединением вузов Республики Беларусь по образованию в области автоматизации технологических процессов, производств и управления
(протокол №____ от _____ 200_г.)

Ответственный за редакцию: *Гульков Г. И.*
Ответственный за выпуск:

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Типовая учебная программа «Информатика» разработана в соответствии с требованием образовательного стандарта по специальности 1-53 01 05 «Автоматизированные электроприводы». Целью изучения данной дисциплины является подготовка будущих специалистов к квалифицированной работе с вычислительной техникой.

На сегодняшний день в сфере автоматизации технологических процессов, производств и управления вычислительная техника и информационные технологии играют очень важную роль. Цифровые вычислительные машины (ЦВМ) выполняют функции как вспомогательных средств инженера (подготовка документации, моделирование динамических систем), так и непосредственного управления технологическими процессами. В частности, для специалистов в области автоматизированных электроприводов изучение дисциплины «Информатика» способствует решению трёх основных задач:

- совершенствование навыков работы с современными компьютерами и периферийными устройствами;
- ознакомление с современным программным обеспечением, позволяющим решать как технические задачи, так и задачи из других сфер деятельности;
- обучение основам алгоритмизации и программирования для развития у студентов логического и математического мышления.

Дисциплина «Информатика» базируется на математической подготовке студентов, обеспечиваемой курсом «Математика».

Основы, заложенные при изучении «Информатики», позволят в дальнейшем студенту, а после – инженеру, решать большинство предложенных задач с меньшими затратами сил и времени за счёт использования вычислительной техники.

В результате освоения дисциплины «Информатика» студент должен:

знать:

- архитектуру компьютеров и структуру операционных систем;
- теорию алгоритмов, структур данных, языков программирования;
- структурное и объектно-ориентированное программирование;
- технологии баз данных, Web-технологии, компьютерные сети.

уметь:

- работать с пакетами Corel Draw, Photoshop и др. компьютерными программами;
- программировать на языках Visual Basic for Applications, C++ (visual)
- применять языки для работы с базами данных и в компьютерных сетях.

Методы (технологии) обучения

Основными методами обучения, отвечающими целям изучения дисциплины, являются:

– элементы проблемного обучения (проблемное изложение), реализуемые на лекционных занятиях;

– элементы учебно-исследовательской деятельности, реализация творческого подхода на лабораторных работах и при самостоятельной работе;

– коммуникативные технологии (дискуссия, учебные дебаты, мозговой штурм и другие формы и методы), реализуемые на конференциях;

– проектные технологии, используемые при проектировании конкретного объекта, реализуемые при выполнении курсовой работы.

Организация самостоятельной работы студентов

При изучении дисциплины рекомендуется использовать следующие формы самостоятельной работы:

– контролируемая самостоятельная работа в виде решения индивидуальных задач в аудитории во время проведения лабораторных занятий под контролем преподавателя в соответствии с расписанием;

– управляемая самостоятельная работа, в том числе в виде выполнения индивидуальных расчетных заданий с консультациями преподавателя;

– подготовка курсовой работы по индивидуальным заданиям, в том числе разноуровневым заданиям.

Диагностика компетенций студента

Промежуточный контроль знаний студента осуществляется на лабораторных занятиях путём проведения защиты отчётов по лабораторным работам. В ходе защиты кроме проверки правильности оформления отчёта студенту предоставляется задание, которое он должен выполнить при помощи компьютера. Таким образом проверяется как уровень знаний студента, так и его навыки работы с вычислительной техникой. Оценкой при защите отчёта по лабораторной работе является «зачёт» или «незачёт».

В общем, для оценки достижений студента используется следующий диагностический инструментарий:

- защита отчётов по лабораторным работам;
- защита курсовой работы;
- выступление студента на студенческой научно-технической конференции (СНТК) с докладом;
- сдача экзамена.

Изучение дисциплины «Информатика» рассчитано на 373 часа, в том числе — 186 часов аудиторных занятий. Примерное распределение аудиторных часов по видам занятий:

лекции — 84 часа;

лабораторные работы — 102 часа.

ПРИМЕРНЫЙ ТЕМАТИЧЕСКИЙ ПЛАН

Наименование раздела и темы	Лекции (часы)	Лабораторные занятия (часы)	Всего аудиторных часов
1	2	3	4
Раздел I. Вводная информация	4		4
Тема 1. Общие понятия	2		2
Тема 2. История развития вычислительной техники	1		1
Тема 3. Типы современных компьютеров	1		1
Раздел II. Принцип действия ЦВМ	6		6
Тема 4. Системы счисления	3		3
Тема 5. Представление информации в ЦВМ	2		2
Тема 6. Принцип фон Неймана. Структура современных ЦВМ.	1		1
Раздел III. Аппаратные средства ЦВМ	12		12
Тема 7. Микропроцессор	1		1
Тема 8. Элементы интерфейса в ПК	1		1
Тема 9. Память ЦВМ	2		2
Тема 10. Устройства ввода информации	3,5		3,5
Тема 11. Указующие устройства	0,5		0,5
Тема 12. Устройства вывода информации	4		4
Раздел IV. Программное обеспечение вычислительной техники	9		9
Тема 13. Общая классификация программного обеспечения	1		1
Тема 14. Структура операционных систем	4		4
Тема 15. Системные операции	4		4
Раздел V. Прикладное программное обеспечение	9	45	54
Тема 16. Программы обработки текстовых документов	1	9	10
Тема 17. Программы обработки табличных данных	1	10	11
Тема 18. Компьютерная графика	3	12	15
Тема 19. Системы обработки баз данных	2	12	14
Тема 20. Программы подготовки компьютерных презентаций	1	2	3
Тема 21. Основы компьютерного моделирования	1		1
Раздел VI. Компьютерные сети	10	6	16
Тема 22. Основы построения компьютерных сетей	3		3
Тема 23. Аппаратные средства компьютерных сетей	2		2
Тема 24. Разновидности компьютерных сетей	3		3

1	2	3	4
Тема 25. Web-технологии	1	4	5
Тема 26. Защита информации в компьютерных сетях	1	2	3
Раздел VII. Основы алгоритмизации	7	11	18
Тема 27. Этапы решения задач на компьютере	1	1	2
Тема 28. Линейные алгоритмы	1	2	3
Тема 29. Разветвляющиеся алгоритмы	2	4	6
Тема 30. Циклические алгоритмы	3	4	7
Раздел VIII. Использование подпрограмм	2	4	6
Тема 31. Подпрограммы-функции	1	2	3
Тема 32. Подпрограммы-процедуры	1	2	3
Раздел IX. Массивы	2	8	10
Тема 33. Одномерные массивы	1	4	5
Тема 34. Многомерные массивы	1	4	5
Раздел X. Основные типы данных	5	10	15
Тема 35. Простые типы	1	2	3
Тема 36. Структурированные типы	4	8	12
Раздел XI. Программирование графики	4	8	12
Тема 37. Изображение простейших графических объектов	1	2	3
Тема 38. Построение графиков математических функций	3	6	9
Раздел XII. Работа с динамическими переменными	4	6	10
Тема 39. Динамические переменные	2	3	5
Тема 40. Динамические структуры	2	3	5
Раздел XIII. Решение прикладных задач с помощью программирования	2	4	6
Тема 41. Нахождение экстремумов функций	1	2	3
Тема 42. Сортировка данных	1	2	3
Раздел XIV. Инструментальное программное обеспечение	8		8
Тема 43. Эволюция и классификация языков программирования	2		2
Тема 44. Обзор основных языков программирования	6		6
ВСЕГО:	84	102	186

СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

РАЗДЕЛ I ВВОДНАЯ ИНФОРМАЦИЯ

Тема 1. Общие понятия

Рассматриваются такие понятия как «информация», «информатика», «информационные технологии». Приводятся основные меры и показатели качества информации

Тема 2. История развития вычислительной техники

Изучение основных этапов развития вычислительной техники от механических счётных машин до современных ЦВМ. Сравнение основных характеристик вычислительной техники разных поколений

Тема 3. Типы современных компьютеров

Краткий обзор существующих на сегодняшний день видов вычислительных машин. Их классификация по принципу действия, платформе, выполняемым функциям

РАЗДЕЛ II. ПРИНЦИП ДЕЙСТВИЯ ЦВМ

Тема 4. Системы счисления

Классификация и обзор существующих систем счисления. Правила перевода целых и дробных чисел из одной системы счисления в другую. Арифметические операции в различных системах. Основы алгебры логики.

Тема 5. Представление информации в ЦВМ.

Представление информации в ЦВМ. Формы представления информации: естественная и с плавающей точкой. Единицы информации

Тема 6. Принцип фон Неймана. Структура современных ЦВМ.

Краткое описание принципа Дж. Фон Неймана. Структура ЦВМ, предложенная фон Нейманом. Принцип программного управления. Производится сравнение структуры современных ЦВМ со структурой фон Неймана. Обзор основных компонентов современных ЦВМ.

РАЗДЕЛ III. АППАРАТНЫЕ СРЕДСТВА ЦВМ

Тема 7. Микропроцессор

Рассматривается упрощённо структура микропроцессора, его основные характеристики и взаимодействие с другими устройствами.

Тема 8. Элементы интерфейса в ПК

Рассматриваются такое понятие, как «интерфейс». Изучаются устройства: материнская плата, шина, адаптер, контроллер, порт.

Тема 9. Память ЦВМ

Понятие «память». Принцип организации памяти в цифровых устройствах. Основные виды внутренней памяти ПК: оперативная, сверхоперативная, постоянная, полупостоянная. Основные характеристики устройств памяти. Общая классификация внешней памяти ПК. Принцип записи и чтения информации магнитным и оптическим способом. Дисковые накопители: *FDD*, *HDD*, *CD*, *DVD*. Основные характеристики устройств дисковой памяти

Тема 10. Устройства ввода информации

Рассматриваются основные устройства ввода информации в ПК: клавиатура, сканер, графический планшет. Приводятся их основные характеристики.

Тема 11. Указующие устройства

Описываются такие устройства как манипулятор-мышь, трекбол и сенсорная панель

Тема 12. Устройства вывода информации

Рассматриваются основные устройства вывода информации: мониторы, принтеры, плоттеры. Приводится их классификация и основные характеристики

РАЗДЕЛ IV. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Тема 13. Общая классификация программного обеспечения

Общая классификация программного обеспечения. Базовое, прикладное, инструментальное программное обеспечение.

Тема 14. Структура операционных систем

Операционная система. Семейства операционных систем. Структуры наиболее известных операционных систем. Разбиение дискового пространства. Файл. Виды файлов, их расширение и атрибуты. Директории. Логические дисководы.

Тема 15. Системные операции

Общая классификация системных операций. Более подробное рассмотрение файловых менеджеров и утилит. Программы по архивации, дефрагментации, борьбе с компьютерными вирусами.

РАЗДЕЛ V.ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Тема 16. Программы обработки текстовых документов

Текстовые редакторы. Интерфейс. Создание, открытие, сохранение документов. Форматирование. Редактирование. Работа с таблицами, формулами, графикой.

Тема 17. Программы обработки табличных данных

Электронные таблицы. Интерфейс. Ввод и редактирование данных. Форматирование. Адресация ячеек. Вычисления в таблицах. Построение диаграмм.

Тема 18. Компьютерная графика

Растровая и векторная компьютерная графика. Основные программы просмотра, редактирования и создания графической информации: *Corel Draw*, *Photoshop*, *Paint*, *AutoCAD.*

Тема 19. Системы обработки баз данных

Понятие СУБД (Система управления базами данных). Объекты баз данных Основные операции с данными в СУБД.

Тема 20. Программы подготовки компьютерных презентаций

Понятие мультимедийной презентации. Программы подготовки мультимедийных презентаций.

Тема 21. Основы компьютерного моделирования

Компьютерное моделирование. Математическая и имитационная модели объекта. Программы компьютерного моделирования.

РАЗДЕЛ VI. КОМПЬЮТЕРНЫЕ СЕТИ

Тема 22. Основы построения компьютерных сетей

Краткая история развития компьютерных сетей. Взаимодействие компьютеров между собой. Топология компьютерных связей. Адресация в компьютерных сетях.

Тема 23. Аппаратные средства компьютерных сетей

Здесь изучается оборудование, позволяющее организовать сетевую работу: модемы, кабели, повторители, коммутаторы, маршрутизаторы, концентраторы и др.

Тема 24. Разновидности компьютерных сетей

Производится классификация компьютерных сетей, рассматриваются некоторые, наиболее часто используемые. Особое внимание уделяется Всемирной компьютерной сети *Internet*.

Тема 25. Web**-технологии**

Знакомство с *Web*-технологиями. Основные средства обмена информацией в сети *Internet* с помощью *Web*-технологий. Гипертекстовые технологии. Электронная почта.

Тема 26. Защита информации в компьютерных сетях

Защита информации от несанкционированного доступа. Брандмауэры. Электронная цифровая подпись.

РАЗДЕЛ VII. ОСНОВЫ АЛГОРИТМИЗАЦИИ

Тема 27. Этапы решения задач на компьютере

Этапы разработки алгоритма. Виды алгоритмов. Изображение блок-схем алгоритмов в соответствии с ГОСТ.

Тема 28. Линейные алгоритмы

Структура линейного алгоритма. Пример разработки линейного алгоритма и его программирование.

Тема 29. Разветвляющиеся алгоритмы

Разновидности разветвляющихся структур. Структурные операторы программирования разветвляющихся алгоритмов Примеры задач с разветвляющимися структурами.

Тема 30. Циклические алгоритмы

Циклические структуры с заранее известным числом повторений, с предусловием и постусловием. Программирование структур разветвляющихся алгоритмов. Примеры разработки циклических алгоритмов и их программирование.

РАЗДЕЛ VIII. ИСПОЛЬЗОВАНИЕ ПОДПРОГРАММ

Тема 31. Подпрограммы-функции

Описание подпрограммы-функции. Вызов подпрограммы-функции. Примеры использования функций. Некоторые стандартные функции.

Тема 32. Подпрограммы-процедуры

Описание подпрограммы-процедуры. Вызов подпрограммы-процедуры. Примеры использования процедур. Некоторые стандартные процедуры.

РАЗДЕЛ IX. МАССИВЫ

Тема 33. Одномерные массивы

Описание одномерного массива. Обращение к элементу массива. Использование одномерных массивов для представления векторов.

Тема 34. Многомерные массивы

Описание многомерного массива. Обращение к элементу многомерного массива. Использование двумерных массивов для представления матриц.

РАЗДЕЛ X. ОСНОВНЫЕ ТИПЫ ДАННЫХ

Тема 35. Простые типы

Целочисленные и вещественные типы данных. Логический тип данных и операции с логическими переменными. Примеры решения задач.

Тема 36. Структурированные типы

Множество. Операции с переменными множественного типа. Записи. Обращение к полям записей. Работа с записями. Запись/чтение информации в файл/из файла. Примеры решения задач.

РАЗДЕЛ XI. ПРОГРАММИРОВАНИЕГРАФИКИ

Тема 37. Изображение простейших графических объектов

Текстовый и графический режимы монитора. Построение линий, окружностей, прямоугольников. Установка цветов.

Тема 38. Построение графиков математических функций

Изображение координатных осей. Нанесение чисел на график. Перенос значений математической функции на экран. Автоматическое масштабирование координатных осей в зависимости от значений, принимаемых функцией. Примеры решения задач.

РАЗДЕЛ XII. РАБОТА С ДИНАМИЧЕСКИМИ ПЕРЕМЕННЫМИ

Тема 39. Динамические переменные

Понятие динамической переменной. Создание и удаление динамической переменной. Указатель. Примеры решения задач.

Тема 40. Динамические структуры

Понятие динамической структуры. Дерево. Граф. Списки со связями. Стек. Очередь. Примеры решения задач.

РАЗДЕЛ XI||. РЕШЕНИЕ ПРИКЛАДНЫХ ЗАДАЧ С ПОМОЩЬЮ ПРОГРАММИРОВАНИЯ

Тема 41. Нахождение экстремумов функций

Поиск максимальных и минимальных элементов массива с помощью циклических алгоритмов. Примеры решения задач.

Тема 42. Сортировка данных

Алгоритмы сортировки массивов. Использование динамических структур для сортировки массивов данных. Примеры решения задач.

РАЗДЕЛ XIV. ИНСТРУМЕНТАЛЬНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Тема 43. Эволюция и классификация языков программирования

История появления и развития языков программирования. Классификация инструментального программного обеспечения. Интегрированные среды программирования. Основы объектно-ориентированного программирования.

Тема 44. Обзор основных языков программирования

Вкратце рассматриваются особенности наиболее известных языков программирования. *Turbo Pascal*, *Delphi*, *Visual Basic for Applications*, *C*, *C++*.

ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

Примерный перечень тем лабораторных работ

1. Форматирование и редактирование текстовых документов
2. Знакомство электронными таблицами. Ввод и форматирование данных
3. Простейшие вычисления электронных таблицах
4. Основные приёмы работы с компьютерной графикой
5. Ознакомление с приёмами обработки баз данных
6. Подготовка компьютерных презентаций
7. основные приёмы работы с использованием *Web*-технологий
8. Программирование математических выражений
9. Ввод/вывод данных при программировании
10. Программирование линейных вычислительных процессов
11. Программирование разветвляющихся вычислительных процессов с использованием условного оператора
12. Программирование разветвляющихся вычислительных процессов с использованием операторов выбора
13. Программирование циклических вычислительных процессов с использованием счётчика цикла
14. Программирование циклических вычислительных процессов с использованием оператора цикла с предусловием
15. Программирование циклических вычислительных процессов с использованием оператора цикла с постусловием
16. Программирование вычислительных процессов с использованием обработки одномерных массивов данных
17. Программирование вычислительных процессов с использованием обработки двумерных массивов данных
18. Программирование вычислительных процессов с использованием подпрограмм-функций
19. Программирование вычислительных процессов с использованием подпрограмм-процедур
20. Программирование алгоритмов с использованием файлов
21. Программирование графики
22. Программирование динамических структур

Примерный перечень тем курсовых работ

1. Разработка алгоритма и программы по вычислению сложной математической функции и построение графика функции
2. Решение математических задач итерационными методами
3. Решение дифференциальных уравнений численными методами
4. Обработка массивов данных с помощью динамических структур

Примерный перечень контрольных вопросов и заданий для самостоятельной работы

68. Информация. Меры информации. Информационные технологии.
69. История развития вычислительной техники.
70. Системы счисления. Перевод целых чисел из одной системы счисления в другую.
71. Перевод дробных чисел из одной системы счисления в другую.
72. Способы представления информации в ЦВМ.
73. Основные типы современных компьютеров.
74. Принцип работы ЦВМ. Структура фон Неймана. Структура современных ПК.
75. Микропроцессоры. Характеристики. Охлаждение процессоров.
76. Материнская плата. Контроллеры. Порты. Шины.
77. Память вычислительной машины. Внутренняя память компьютера.
78. Дисковая память. НГМД, НЖМД.
79. Организация хранения информации на магнитных дисках.
80. Принцип записи и чтения информации с помощью лазера. Накопители на оптических дисках.
81. Блок питания и корпус ПК. Полная внутренняя структура ПК.
82. Периферийные устройства. Общая классификация.
83. Клавиатура. Манипулятор.
84. Общая классификация мониторов. Электронно-лучевые мониторы.
85. Мониторы на плоских панелях.
86. Общая классификация печатающих устройств. Матричные принтеры.
87. Струйные и электрографические принтеры.
88. Плоттеры. Принцип действия. Характеристики.
89. Сканеры. Принцип действия. Характеристики.
90. Дигитайзеры. Принцип действия. Характеристики.
91. Каналы связи. Классификация Характеристики.
92. Аналоговые модемы. Виды модуляции.
93. Модемы для цифровых каналов связи.
94. Программное обеспечение информационных технологий. Виды программного обеспечения.
95. Операционная система. Основные семейства ОС.
96. Файл. Файловая система. Атрибуты файлов.
97. Каталоги. Логические дисководы.
98. ОС Windows. Элементы интерфейса. Работа с файлами и программами. Файловые менеджеры. Сжатие информации. Дефрагментация. Антивирусные программы.
99. Компьютерная графика. Основные виды графической информации.
100. Системы управления базами данных
101. Компьютерные презентации
102. Компьютерные сети. Общие сведения. Топологии сетей. Способы адресации.

103. Разновидности компьютерных сетей. Всемирная сеть *Internet*. *Web*-технологии.
104. Методика программирования инженерных задач на ЦВМ
105. Линейные структуры.
106. Разветвляющиеся структуры.
107. Циклические структуры
108. Арифметические циклы и их программирование
109. Циклы с предусловием и их программирование
110. Циклы с постусловием и их программирование
111. Подпрограммы-процедуры
112. Подпрограммы-функции
113. Формальные и фактические параметры подпрограмм. Область видимости идентификаторов. Параметры-переменные, параметры-значения.
114. Массивы
115. Графика. Графические координаты.
116. Порядковые типы
117. Вещественные типы
118. Строковый тип данных
119. Тип данных «Запись»
120. Тип данных «Множество»
121. Динамические переменные. Работа с динамическими переменными
122. Динамические структуры данных. Графы и деревья
123. Динамические структуры данных. Связные списки
124. Динамические структуры данных. Стек.
125. Динамические структуры данных. Очередь.
126. Решение прикладных задач с помощью программирования.
127. Разновидности языков программирования.

Основная литература

1. Фигурнов В. Э. IBM PC для пользователя. Краткий курс / В. Э. Фигурнов. – 7-е изд. – М.: ИНФРА-М. 2003. – 480 с.
2. Бройдо В. Л., Ильина О. П. Архитектура ЭВМ и систем. Учебник для вузов. / В. Л. Бройдо, О. П. Ильина – СПб.: Питер, 2006. – 718 с.
3. Леонтьев В. П. Новейшая энциклопедия персонального компьютера / В. П. Леонтьев. – 5-е изд., перераб. и доп. – М.: ОЛМА-ПРЕСС. 2003. – 957 с.
4. Симонович С. В. Общая информатика: Учебное пособие / С. В. Симонович, Г. А. Евсеев, А. Г. Алексеев. - М.: АСТ- ПРЕСС КНИГА, 2004. - 592с.
5. Информатика. Базовый курс: Под ред. Симановича С. В. - 2-е изд. - СПб.: Питер, 2006. - 640с.
6. Веретенникова Е.Г. и др. Информатика: Учебное пособие / Веретенникова Е.Г. и др.; Е. Г. Веретенникова, С. М. Патрушина, Н. Г. Савельева . - Ростов н/Д.: Изд-кий центр "Март", 2002. - 416с.
7. Острейковский В. А. Информатика: Учебник / В. А. Острейковский. - 3-е изд., стер. - М.: Высшая школа, 2005. - 511с.

8. Олифер В. Г. Компьютерные сети. Принципы, технологии, протоколы: Учебник для ВУЗов. 3-е изд. – СПб.: Питер, 2009. – 958 с.

Дополнительная литература

1. Стоцкий Ю. Самоучитель Office XP. / Ю. Стоцкий. – СПб.: Питер, 2005. – 571с.
2. Конев Ф. Б. Информатика для инженеров: Учебное пособие / Ф. Б. Конев. - М.: Высшая школа, 2004. - 272с.
3. Рыжиков Ю.И. Информатика: Лекции и практикум / Ю. И. Рыжиков. - СПб.: Корона принт, 2000. - 256с.
4. Каймин В.А. Информатика: Учебник / В. А. Каймин. - М.: ИНФРА-М, 2000. - 232с. - (Серия "Высшее образование").
5. Панько И. Л. Практикум по прикладной информатике: Учебное пособие / И. Л. Панько, И. Г. Кузенкова, А. В. Суворов. - Мн.: Изд-во"Пачатковая школа", 2004. - 276с.
6. Могилев А.В. и др. Информатика: Учебное пособие / Могилев А.В. и др.; Под ред. Е.К. Хеннера; А.В. Могилев, Н.И. Пак, Е.К. Хеннер. - М.: Академия, 2000. - 816с.
7. Воройский Ф.С. Информатика. Новый систематизированный толковый словарь-справочник: Вводный курс по информатике и вычислительной технике в терминах / Ф. С. Воройский. - 2-е изд., перераб. и доп. - М.: "Издательство Либерея", 2001. - 536с.
8. Аляев Ю. А. Практикум по алгоритмизации программированию на языке Паскаль / Ю. А. Аляев, В. П. Гладков, О. А. Козлов. – М.: Финансы и статистика, 2004. – 528 с.