Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Пермский национальный исследовательский политехнический университет»

Кафедра «Информационные технологии и автоматизированные системы»

Полякова О.А.

«Информатика 1».
«Информатика 2».

Теоретические материалы и методические указания для выполнения лабораторных работ

4-5 ЗЕ
7 ЗЕ

Часть 1

Пермь 2017

Оглавление

Введение

Алгоритмический язык высокого уровня Паскаль был разработан в конце 60-х годов профессором Н.Виртом. Он был создан специально для обучения программированию. К основным достоинствам языка Паскаль следует отнести гибкость и надежность, простоту и ясность конструкций, возможность удовлетворения требованиям структурного программирования, наличия набора структурированных типов данных: массивов, записей, записей с вариантами, файлов, множеств, возможность построения новых типов данных.

На базе стандартного Паскаля фирма Borland разработала семейство Паскаль-систем, называемых Турбо Паскалем. Турбо Паскаль пользуется широкой популярностью среди массовых пользователей и профессиональных программистов. Это объясняется наличием очень удобной интегрированной среды и тем, что в его основе лежит мощный язык программирования, представляющий собой расширенную версию языка Паскаль.

За последние годы фирма Borland разработала и выпустила на рынок шесть модификаций этой системы. Каждая из них представляет собой усовершенствование предыдущей. Непрерывное совершенствование системы Турбо Паскаля породило в конце концов очень мощную по своим возможностям систему программирования, отвечающую самым взыскательным требованиям. С помощью Турбо Паскаля можно создавать многие программы — от программ, предназначенных для решения простейших вычислительных задач, до сложных современных систем управления базами данных и операционных систем.

И вместе с тем Турбо Паскаль остается простым в изучении, что позволяет начинающему программисту на его основе изучить методы и способы эффективного программирования.

Данное пособие состоит из 8 разделов. Каждый раздел включает в себя краткие теоретические сведения и лабораторные работы по соответствующей теме. Пособие “Практикум по программированию на языке Турбо Паскаль” можно использовать при изучении курсов “Информатика”, “Алгоритмические языки программирования”, “Основы программирования” и т.п. для студентов электротехнического (специальности АСУ, ЭВТ, КРЭС и др.) и гуманитарного (специальность ЭУП) факультетов.

В подготовке данного пособия активное участие принимали инженеры Лобанова Е.В. и ПавленкоТ.М.

1.Стандартные типы данных

1.1.Структура программы

Программа на языке Турбо Паскаль состоит из заголовка и собственно программы, называемой блоком. Блок состоит из разделов. Максимальное количество разделов шесть. Разделы располагаются в следующем порядке:

1. Раздел меток;
2. Раздел констант;
3. Раздел типов;
4. Раздел переменных;
5. Раздел процедур и функций;
6. Раздел операторов.

Раздел операторов заключается в операторные скобки begin ... end. В нем указывается последовательность действий, которые должны выполняться ЭВМ. Все остальные разделы носят описательный характер.

Любой раздел, кроме последнего, может отсутствовать. Разделителем между разделами и операторами служит точка с запятой. В конце программы должна стоять точка.

В любое место программы могут быть включены комментарии. При этом смысл программы не меняется. Комментарии включаются в фигурные скобки.

1.2.Описание стандартных типов данных

Программа, написанная на языке TP, оперирует некоторыми объектами, называемыми данными. Каждый элемент данных в программе является либо константой, либо переменной. Для каждой переменной задается тип, определяющий возможное значение переменной и операции, которые могут над ней выполняться. Тип переменной задается в разделе переменных.

Язык TP дает возможность строить сложные типы данных, которые основываются на следующих элементарных:

1) целый “integer”;
2) вещественный “real”;
3) символьный “char”;
4) булевский “boolean”;
5) перечислимые.

Целый тип

Обеспечивает задание целых чисел. Существует несколько видов целых типов: byte, shortint, integer, longint.

Встроенные процедуры и функции, применимые к целым типам.

Обращение	Тип результата	Действие
abs(x)	x	Возвращает модуль x
chr(b)	Char	Возвращает символ по его коду
dec(vx[,i])	процедура	Уменьшает значение vx на i, при отсутствии i- на 1
inc(vx[,i])	-/-	Увеличивает значение vx на i, при отсутствии i- на 1
odd(l)	boolean	Возвращает TRUE, если аргумент - нечетное число, FALSE - если четное
random(w)	как у параметра	Возвращает псевдослучайное число, равномерно распределенное на интервале 0<=x<w
sqr(x)	-/-	Возвращает квадрат аргумента
exp(x)	real	e^x
sqrt(x)	real	Возвращает квадратный корень из x
sin(x)	-/-	sin x
cos(x)	-/-	cos x
ln(x)	-/-	ln x
arctan(x)	-/-	arctg x
succ(x)	как у параметра	Возвращает следующее целое число, т.е. x+1
pred(x)	-/-	Возвращает предыдущее целое число, т.е. x-1

x - выражение любого из типов.

b, l, i, w - выражения соответствующих типов: byte,

longint, integer, word.

vx - переменная типа x.

Арифметические операции: +(сложение), -(вычитание), *(умножение), /(деление), DIV(деление нацело), MOD(вычисление остатка от целочисленного деления).

Операции отношения: =(равно), <>(не равно), <(меньше), >(больше), <=(меньше или равно), >=(больше или равно).

Вещественный тип

Запись вещественного числа в TP возможна:

в виде числа с фиксированной точкой: 12.3, 0.67;

в экспоненциальной форме: 1.4 E-8 ($1.4*10^{-8}$), 9.7 E3 ($9.7*10^{3}$).

Над переменными этого типа определены арифметические операции: +(сложение), -(вычитание), *(умножение), /(деление), а также операции отношения (см. выше).

Встроенные процедуры и функции.

Обращение	Тип параметра	Тип результата	Действие
abs(x)	real, integer	x	Возвращает модуль x
random(x)	integer	integer	Возвращает псевдослучайное число, равномерно распределенное на интервале 0<=i<x
sqr(x)	real, integer	тип аргумента	Возвращает квадрат аргумента
exp(x)	real	real	e^x
sqrt(x)	real	real	Возвращает квадратный корень из x
sin(x)	-/-	-/-	sin x
cos(x)	-/-	-/-	cos x
ln(x)	-/-	-/-	ln x
arctg(x)	-/-	-/-	arctg x
frac(x)	-/-	-/-	Дробная часть числа
int(x)	-/-	-/-	Целая часть числа
pi	---	real	Пи=3.14159265...
trunc(x)	real	integer	Отбрасывание дробной части
round(x)	real	integer	Округление до ближайшего целого
randomize	---	---	Инициация датчика псевдослучайных чисел
random	---	real	Возвращает псевдослучайное число, равномерно распределенное на интервале 0<=x<1

Символьный тип

Значениями символьного типа являются элементы конечного и упорядоченного набора знаков.

Символ, заключенный в апострофы, обозначает константу символьного типа, например: ‘5’, ‘d’.

Над переменными символьного типа определены следующие функции:

1) функции преобразования:

ORD (s) - дает порядковый номер символа s в упорядоченном множестве символов: ORD(‘5’)=53.

CHR (i) - дает символ, стоящий под номером i в упорядоченном множестве символов: CHR(66)=‘B’.

2) операции отношения (см. выше):

если C1 и C2 - символьные переменные, то C1>C2 истинно только тогда, когда ORD(C1)>ORD(C2).

3) стандартные функции:

PRED (s) - возвращает предыдущий символ.

SUCC (s) - возвращает следующий символ.

Булевский тип

Переменные булевского типа могут принимать только два значения: TRUE и FALSE.

Над ними определены:

1) логические операции:

AND(и или конъюнкция), OR(или или дизъюнкция),

NOT(не или отрицание).

2) операции отношения (см. выше):

причем TRUE>FALSE.

Перечисляемый тип

Задается перечислением тех значений, которые может получать переменная этого типа. Нумерация в списке значений начинается с 0, т.е. первое значение имеет номер - 0, а второе - 1 и т.д.

Пример: type tree=(birch, oak, pine);

или

var tr=(birch, oak, pine);

1.3.Выражения

Значения выражений вычисляются с учетом расставленных скобок и старшинства операций.

Ниже приведены операции в порядке убывания их приоритета, причем операции в одной строке имеют одинаковый приоритет:

NOT

*, /, MOD, DIV, AND

+, -, OR

<, >, <=, >=, <>, =

Операции одного и того же старшинства выполняются слева направо в порядке их появления в выражении. Выражения в круглых скобках вычисляются в первую очередь.

Лабораторная работа №1

Вычисление выражений с использованием стандартных функций

Цель задания:

1. Изучение порядка действий при вычислении выражений.
2. Приобретение навыков в записи выражений на языке TP и использование стандартных функций.

Постановка задачи

1. Для задания (а) найти значение функции Y(X) при заданном X. Затем возвести полученное значение в квадрат, т.е. найти $Y1=Y^2(X)$, и вычислить абсолютное значение Y1.
2. Для задания (б) записать выражение, зависящее от координат точки X1 и Y1 и принимающее значение TRUE, если точка принадлежит заштрихованной области, и FALSE, если не принадлежит. Для исследуемой точки вычислить полученное выражение.
3. Результаты всех вычислений вывести на печать.

Содержание отчета

1. Постановка задачи для конкретного варианта.
2. Описание используемых стандартных функций.
3. Текст программы.
4. Распечатка результатов выполнения программы.

Образец выполнения задания.

Лабораторная работа № 1.

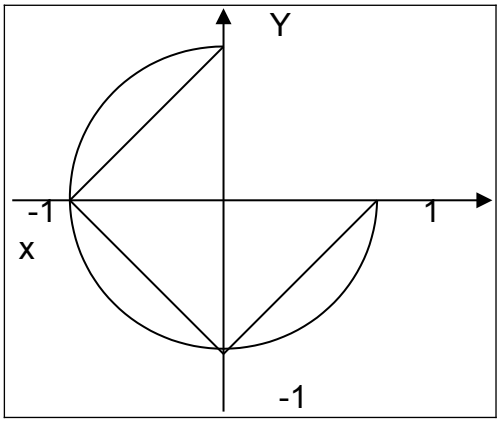
Вычисление выражений с использованием стандартных функций.

Постановка задачи

1. Найти значение функции $Y(X) = 1+ x \cos^2(x) + \sin^3(x)$ при X= 2.346. Затем возвести полученное значение в квадрат, т.е. найти $Y1=Y^2(X)$, и вычислить абсолютное значение ABS(Y1).

2. Записать выражение, зависящее от координат точки M(-0.8; 0.9) и принимающее значение TRUE, если точка принадлежит заштрихованной области, и FALSE, если не принадлежит. Для исследуемой точки вычислить полученное выражение.

3.



4. Результаты всех вычислений вывести на печать.

Описание используемых стандартных функций.

Стандартные функции, возвращают вещественный результат при вещественном или целом аргументе:

- Cos(r), вычисляет косинус аргумента r.
- Sin(r), вычисляет синус аргумента r.
- Abs(r), вычисляет абсолютную величену аргумента r.

Текст программы № 15.а

```
program lab1{ вариант №15.a};
const x=2.346;
var y:real;
begin
writeln('Вычислим значение функции Y=1+cos^2(x)+sin^3(x)  при x=2.346');
y:=1+x*cos(x)*cos(x)+sin(x)*sin(x)*sin(x);
writeln('Y=',y);
writeln('Y^2=',y*y);
writeln('ABC(Y^2)',abs(y*y));
end.
```

Текст программы № 15.б

```
program lab1{  вариант №15.b};
const x0=-0.8;
        y0=0.9;
        r=1;
var pro:boolean;
begin
pro:=(x0*x0+y0*y0<=r*r) and ((x0+1<=y0) or (abs(x0)-1>=y0));
writeln('Точка с координатами M(',x0:0:1,'.',y0:0:1,')');
if pro then writeln('Принадлежит заштрихованной области.')
else writeln('Не принадлежит заштрихованной области.');
end.
```

Распечатка результатов выполнения программы.

Программы № 15.а

```
Y=2.5135058366E+00
Y^2=6.3177115909E+00
ABS(Y^2)=6.3177115909E+00
```

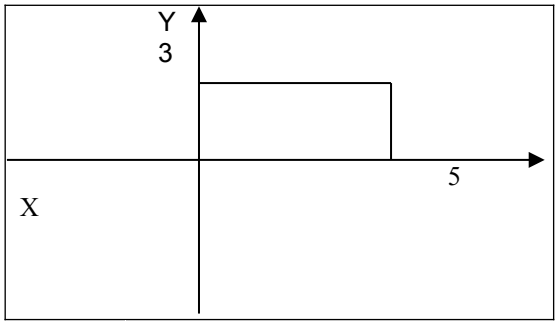
Программы № 15.б

```
Точка с координатами M(-0.8,0.9)
Не принадлежит заштрихованной области.
```

Варианты заданий

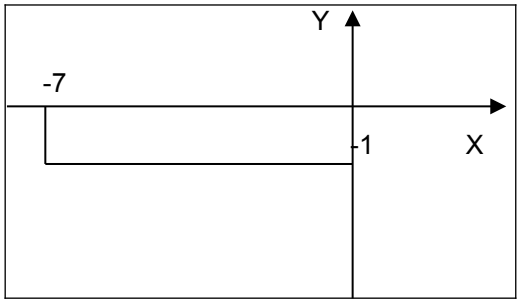
1) а) $Y=\sin(x)+x^3+\frac{1}{x^2+1}$ при x = 5.137

б) Координаты исследуемой точки: (3; 2) Область (I четверть)



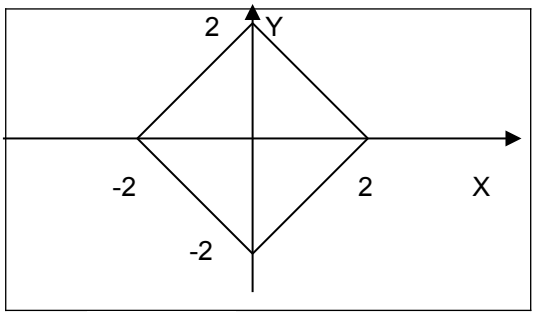
2) а) $Y= x+\frac{1}{x^3 - x} - 2$ при x = 0.675

б) Координаты исследуемой точки: (1.5; 0.5) Область (III четверть)



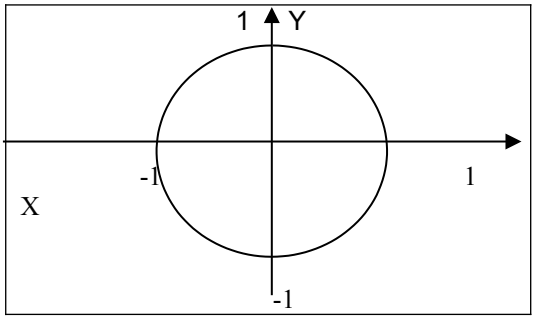
3) а) $Y=x^4 - \cos \arcsin x$ при x=0.051

б) Координаты исследуемой точки: (0.1; 0.3) Область (квадрат):



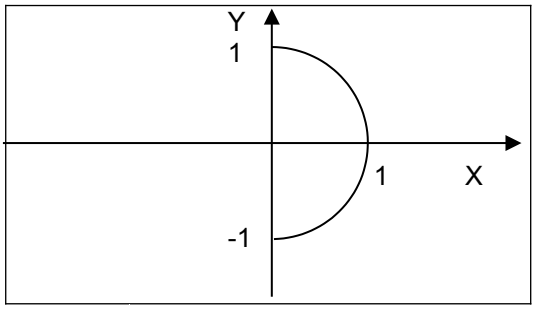
4) а) Y= $\sqrt[3]{x - x^2} + x^5$ при x = 7.873

б) Координаты исследуемой точки: (0.6; 2.5) Область (окружность):



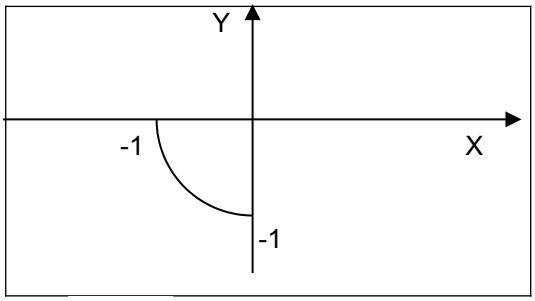
5) а) Y= tg x - (5 - x^4) при x = -3.777

б) Координаты исследуемой точки: (-0.7; 0.2) Область (I и IV четверть):



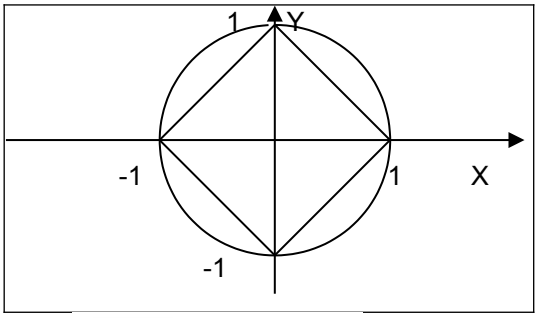
6) а) Y=25x^5 - $\sqrt{x^2 + x}$ при x = 25.144

б) Координаты исследуемой точки: (-0.3; -0.5) Область (III четверть):



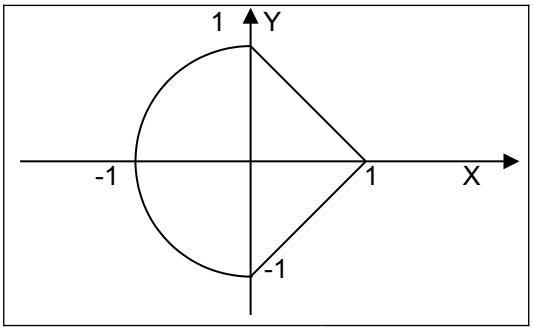
7) а) Y= $\sqrt[5]{x^3 + x^4}$ + ctg arctg x^2 при x = -5.113

б) Координаты исследуемой точки: (2.5; 3) Область (между окружностью и квадратом):



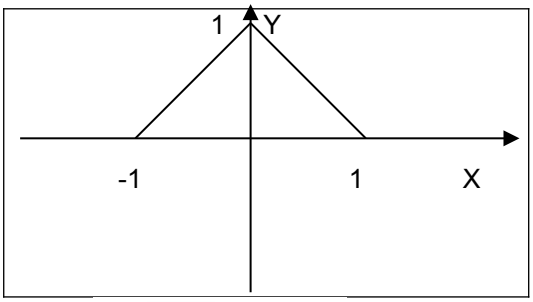
8) а) Y= $\sqrt{|x^3 - 1|} - 7\cos\sqrt[3]{x^4 + x}$ при x = 10.237

б) Координаты исследуемой точки: (-1; -5) Область (вся область определения):



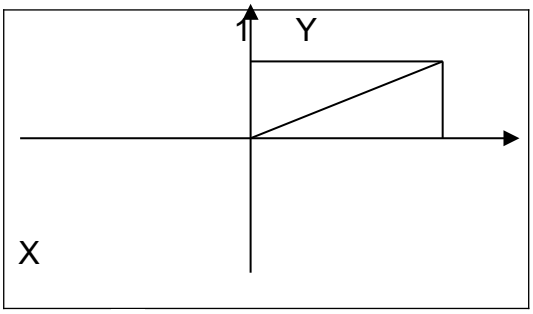
9) а) Y= sin $x^3 + x^4 + \sqrt[5]{x^2 + x^3}$ при x = 1.031

б) Координаты исследуемой точки: (-5; 5) Область (I и II четверть):



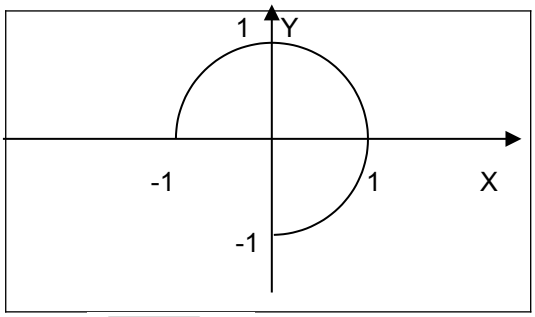
10) а) Y= $x^5\sqrt{|x - 1|} + |25 - x^5|$ при x = 11.131

б) Координаты исследуемой точки: (-5; 7) Область (I четверть, ниже диагонали прямоугольника)



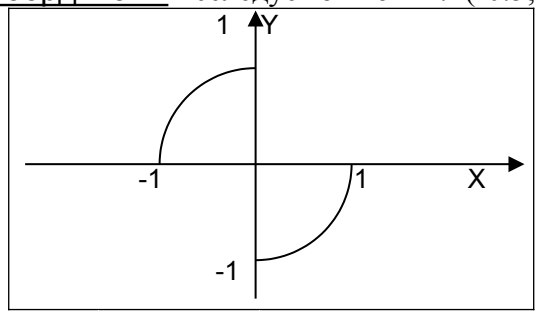
11) а) Y= 2^{-x} x cos(x) +1 при x = 34.211

б) Координаты исследуемой точки: (1; -3) Область (I, II, IV четверти):



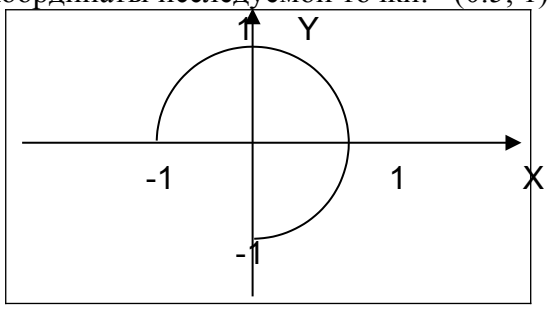
12) а) Y= $\sqrt{x+\sqrt[4]{|x|}+|x|}$ при x = -12.333

б) Координаты исследуемой точки: (-0.5; 0.5) Область (II, IV четверти):



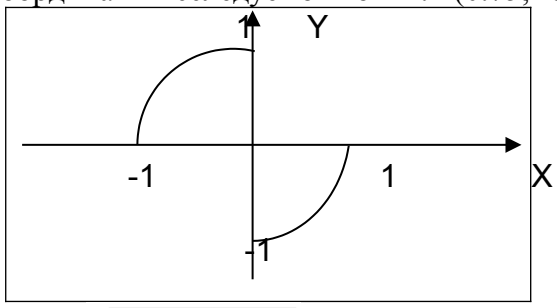
13) а) Y = $\sqrt[3]{e^x+tgx}+\frac{1}{x}$ при x = -3.449

б) Координаты исследуемой точки: (0.5; 1) Область (вся область определения):



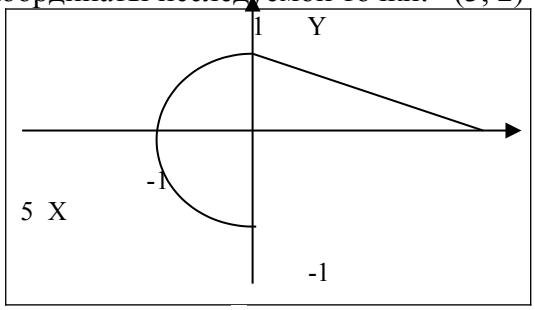
14) а) Y= $\sqrt[4]{|x+1|}+\frac{1}{x^2}$ при x = -45.276

б) Координаты исследуемой точки: (0.75; -0.75) Область (вся область определения):



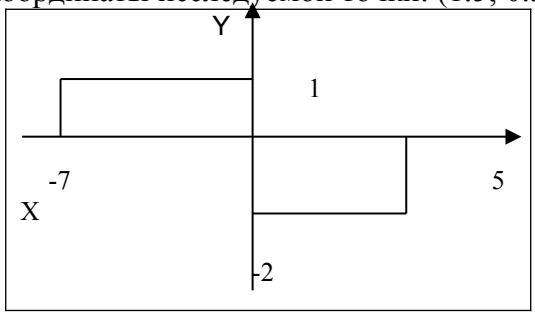
15) а) Y= $\sqrt{\sin(x)+|x^2+x|}$ при x = 3.778

б) Координаты исследуемой точки: (3; 2) Область (I, II, III четверти):



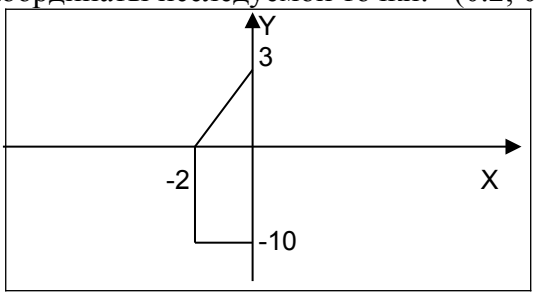
16) а) Y= arcsin x+x^2 при x = -0.671

б) Координаты исследуемой точки: (1.5; 0.5) Область (II, IV четверти):



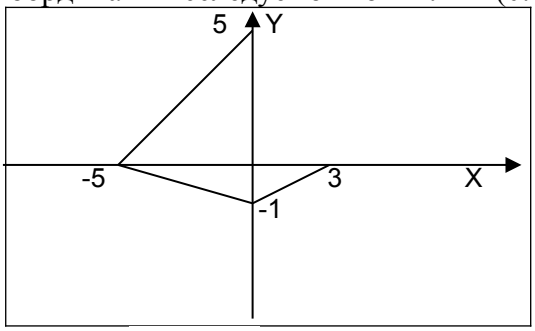
17) а) Y= cos arctg x при x = -0.692

б) Координаты исследуемой точки: (0.2; 0.9) Область (II, III четверти):



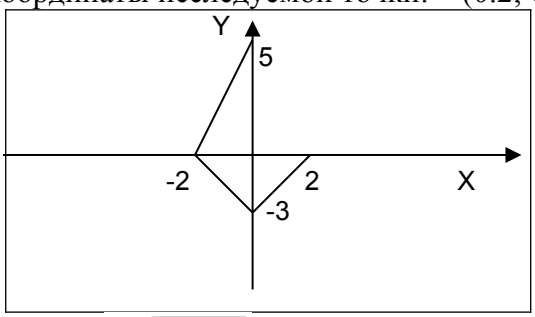
18) а) Y= 7 arcctg x^2 при x = 0.276

б) Координаты исследуемой точки: (0.75; -0.3) Область (II, III, IV четверти):



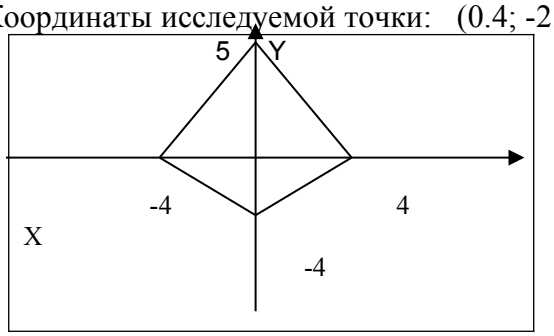
19) а) Y= $5x^{3}\sqrt[5]{\frac{1}{x^2}+\frac{1}{x^3}}$ при x = 28.954

б) Координаты исследуемой точки: (0.2; 0.45) Область (вся область определения):



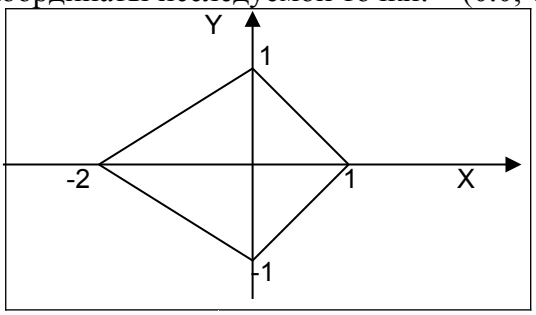
20) а) Y= $2^{-x}\sqrt{x+\sqrt[4]{|x|}}$ при x = 4.741

б) Координаты исследуемой точки: (0.4; -2.5) Область (I, III, IV четверти):



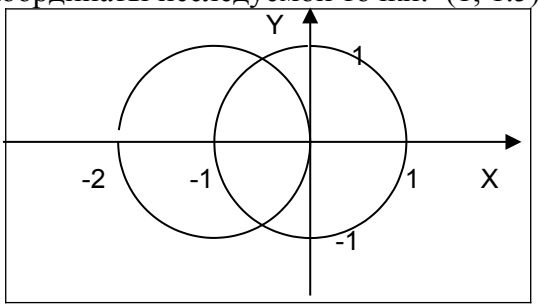
21) а) Y= $\sqrt[3]{e^x - \sin x}$ при x = 2.312

б) Координаты исследуемой точки: (0.0; 0.0) Область (I, II, III четверти):



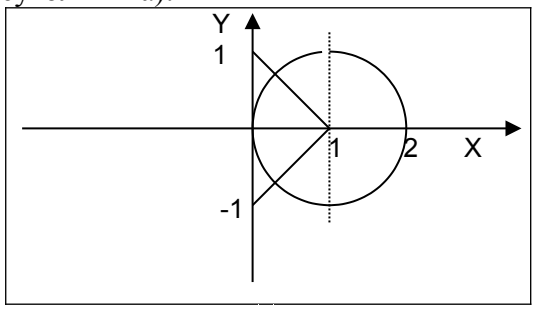
22) а) Y= 1+ $\frac{1}{x}+\frac{1}{x^2}$ при x = -0.387

б) Координаты исследуемой точки: (1; 1.5) Область (пересечение окружностей):



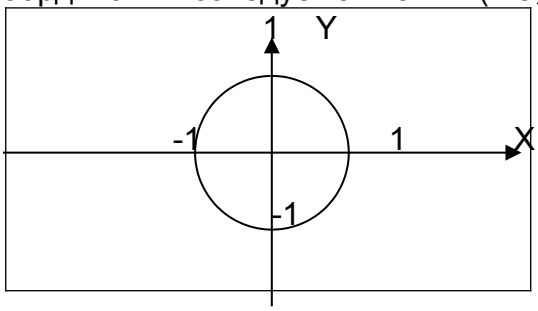
23) а) Y= ch$|x+1|$ при x = 4.352

б) Координаты исследуемой точки: (-0.5; 0.9) Область (I, IV четверти, пересечение окружности и треугольника):



24) а) Y = arcsin x + x^2 при x = 0.112

б) Координаты исследуемой точки: (1.5; 0.0) Область (окружность, но не треугольник):



2. Операторы языка.

2.1. Составной и пустой операторы.

Составной оператор — это последовательность произвольных операторов программы, заключенная в операторные скобки — зарезервированные слова BEGIN ... END [17]. Характер входящих операторов в данный момент может быть различным. Также TP допускает любую глубину вложенности составных операторов.

2.2.Условный оператор.

Условный оператор имеет следующий вид:

IF <условие> THEN <оператор1> ELSE <оператор2>, где IF, THEN, ELSE — зарезервированные слова;

<условие> — выражение логического типа;

<оператор1>, <оператор2> — операторы языка TP.

Выполнение этого оператора начинается с проверки условия. Если оно истинно (=TRUE), то выполняется <оператор1>, иначе (значение выражения = FALSE) выполняется <оператор2>. Часть ELSE может быть опущена, тогда оператор выполняется только при значении выражения =TRUE, иначе он пропускается. Операторы повторений.

2.3.Операторы повторений.

Счетный оператор цикла (вариант 1):

FOR <пар_цикла>:= <нач_значение> TO <кон_значение> DO <операторы>, где FOR, TO, DO — зарезервированные слова;

<пар_цикла> — параметр цикла (переменная типа integer);

<нач_значение> — начальное значение <пар_цик>;

<кон_значение> — конечное значение <пар_цик>;

<операторы> — произвольные операторы языка TP.

Выполнение оператора начинается с вычисления начального значения. Выполняется присваивание <параметр цикла>:=<начальное значение>.

Затем циклически выполняется:

1. Если параметр цикла меньше или равен конечному значению, то переход к п.2, иначе оператор заканчивает работу.
2. Выполнение операторов после слова DO.
3. <пар_цик> увеличивается на 1.

Счетный оператор цикла (вариант 2):

FOR <пар_цикла>:= <нач_значение> DOWNTO <кон_значение> DO <операторы>, где FOR, TO, DO — зарезервированные слова;

<пар_цикла> — параметр цикла (переменная типа integer);

<нач_значение> — начальное значение <пар_цик>;

<кон_значение> — конечное значение <пар_цик>;

<операторы> — произвольные операторы языка TP.

Выполнение оператора начинается с вычисления начального значения. Выполняется присваивание <параметр цикла>:=<начальное значение>.

Затем циклически выполняется:

4. Если параметр цикла меньше или равен конечному значению, то переход к п.2, иначе оператор заканчивает работу.
5. Выполнение операторов после слова DO.
6. <пар_цик> уменьшается на 1.

Оператор цикла с предусловием:

WHILE <условие> DO <операторы>, где WHILE, DO — зарезервированные слова (пока выполняется условие, делать);

<условие> — выражение логического типа;
<операторы> — операторы языка TP.

Пока значение выражения <условие> равно TRUE, выполняются <операторы>. Если условие FALSE, то выполнение оператора заканчивается.

Здесь оператор выполняется до тех пор, пока логическое выражение TRUE. Для того чтобы цикл завершился, оператор в теле цикла должен в некоторый момент изменить значение логического условия. Оператор может быть любым оператором Паскаля, в том числе оператором цикла. В последнем случае цикл называется вложенным. Если тело цикла содержит более одного оператора, то он оформляется как составной оператор.

Оператор цикла с постусловием:

REPEAT <тело цикла> UNTIL <условие>, где REPEAT, UNTIL — зарезервированные слова (повторять до тех пор, пока не будет выполнено условие);
<тело цикла> — операторы языка TP;
<условие> — выражение логического типа.

Оператор выполняется хотя бы один раз. Затем проверяется условие. Если его значение FALSE, то <тело цикла> повторяется, иначе оператор заканчивает работу.

Здесь оператор выполняется до тех пор, пока логическое выражение FALSE. В теле цикла можно указывать несколько любых операторов Паскаля без дополнительных операторных скобок. Для выхода из цикл необходимо, чтобы операторы тела цикла изменили значение логического условия на TRUE. Тело цикла этого оператора выполняется хотя бы один раз.

2.4.Оператор выбора

CASE <кл_выб> OF <сп_выб> ELSE <оператор> END, где CASE, OF, ELSE, END — зарезервированные слова (случай, из, иначе, конец);
<кл_выб> — ключ выбора (выражение любого порядкового типа);
<оператор> — оператор языка TP;
<сп_выб> — список выбора: одна или несколько конструкций вида:
<константа_выбора>: <оператор>;
<константа_ выбора> — константа того же типа, что
и <кл_выб>;
<оператор> — произвольный оператор языка TP.

Часть ELSE может отсутствовать. Оператор начинается с вычисления значения <кл_выб>. Затем в списке выбора отыскивается значение константы, равное вычисленному значению. Если поиск удачен, то выполняются соответствующие операторы, после чего оператор завершает работу. Если в результате поиска нужное значение не было найдено, то выполняется оператор после слова ELSE. В случае отсутствия части ELSE, оператор ничего не выполняет и заканчивает работу.

2.5.Практические задания.

Лабораторная работа № 2

Решение уравнений и неравенств с использованием условного оператора.

Цель задания:

1. Получение навыков в использовании условного оператора в программе.
2. Знакомство с задачами, для решения которых используются условные операторы.

Постановка задачи:

1. Составить программу решения уравнения (системы уравнений, неравенства, системы неравенств).
2. Напечатать исходные данные и результаты.

Содержание отчета:

1. Постановка задачи.
2. Текст программы.
3. Распечатка исходных данных и результатов выполнения программы.
4. Для результатов должен быть напечатан соответствующий текст.

Образец выполнения задания.

Лабораторная работа № 2, вариант № 8.

Решение уравнений и неравенств с использованием условного оператора.

Постановка задачи.

Составить программу решения системы неравенств: $\begin{cases} a_1 x + b_1 < 0 \\ a_2 x + b_2 < 0 \end{cases}$

Напечатать исходные данные и результаты.

Текст программы.

```
program lab2{ вариант № 8};
var n,a1,a2,b1,b2:integer;
    r1,r2:real;
begin
 writeln('Решим систему неравенств:');
 writeln(' -');
 writeln('| A1X+B1<0');
 writeln('| A2X+B2<0');
 writeln(' -');
 write('Введите a1=');
 readln(a1);
 write(' Введите a2=');
 readln(a2);
 write(' Введите b1=');
 readln(b1);
 write(' Введите b2=');
 readln(b2);
 writeln(' -');
 writeln('| ',a1,'X+(',b1,')<0');
 writeln('| ',a2,'X+(',b2,')<0');
 writeln(' -');
 r1:=((-b1)/a1);
 r2:=((-b2)/a2);
 if (r1<0) and (r2<0)
```

```
  then case r1<r2 of
          false:writeln('Ответ: X<',r1:3:2);
          true:writeln('Ответ: X<',r2:3:2);
          end
  else case r1>r2 of
          false:writeln('Ответ: X<',r1:3:2);
          true:writeln('Ответ: X<',r2:3:2);
          end;
end.
```

Распечатка исходных данных и результатов выполнения программы.

Для результатов должен быть напечатан соответствующий текст.

Рещить систему неравенств: $\begin{cases} a_1x + b_1 < 0 \\ a_2x + b_2 < 0 \end{cases}$

При a1=2, a2=4, b1=-1, b2=6,

При заданных параметрах систма неравенств имеет вид: $\begin{cases} 2x - 1 < 0 \\ 4x + 6 < 0 \end{cases}$
Ответ: X<-1.50 .

Варианты заданий

1) $ax^2 + b = 0$ 2) $ax^2 + bx + c = 0$ 3) $ax^2 + b > 0$

4) $ax^2 + b \le 0$ 5) $ax^2 + bx + c > 0$ 6) $ax^2 + bx + c \le 0$

7) $\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases}$ 8) $\begin{cases} a_1x + b_1 \ge 0 \\ a_2x + b_2 \ge 0 \end{cases}$ 9) $\begin{cases} a_1x + b_1 > 0 \\ a_2x + b_2 < 0 \end{cases}$

10) $\begin{cases} a_1x + b_1y + c_1 < 0 \\ a_2x + b_2y + c_2 < 0 \end{cases}$ 11) $\begin{cases} a_1x + b_1y + c_1 > 0 \\ a_2x + b_2y + c_2 > 0 \end{cases}$ 12) $\begin{cases} a_1x + b_1y + c_1 \ge 0 \\ a_2x + b_2y + c_2 < 0 \end{cases}$

13) $\begin{cases} a_1x + b_1y + c_1 \le 0 \\ a_2x + b_2y + c_2 \ge 0 \end{cases}$

Лабораторная работа № 3.

Построение таблиц функций.

Цель задания:

Получение навыков в использовании оператора цикла с параметром.

Постановка задачи:

Составить программу вычисления значений функции F(x) на отрезке [A, B] в точках Xi = A + iH, где H = (B - A)/M, M — заданное целое число. Значение шага H должно вычисляться один раз.

Содержание отчета:

1. Постановка задачи.
2. Текст программы.

3. Таблица результатов.

Образец выполнения задания.

Лабораторная работа № 3, вариант № 8.

Построение таблиц функций.

Постановка задачи.

Составить программу вычисления значений функции arctg(x) на отрезке [A, B] в точках Xi = A + iH, где H = (B - A)/M, M — заданное целое число. Значение шага H должно вычисляться один раз.

При A=2, B=7, M=15.

Текст программы.

```
program lab3{ вариант № 8};
var h,r:real;
    n:integer;
begin
 h:=(7-2)/15;
 r:=2;
 for n:=1 to 16 do
  begin
  writeln('arctg(',r:5:4,')=',(arctan(r)):5:4);
  r:=r+h;
  end;
end.
```

Таблица результатов

```
arctg(2.0000)=1.1071
arctg(2.3333)=1.1659
arctg(2.6667)=1.2120
arctg(3.0000)=1.2490
arctg(3.3333)=1.2793
arctg(3.6667)=1.3045
arctg(4.0000)=1.3258
arctg(4.3333)=1.3440
arctg(4.6667)=1.3597
arctg(5.0000)=1.3734
arctg(5.3333)=1.3854
arctg(5.6667)=1.3961
arctg(6.0000)=1.4056
arctg(6.3333)=1.4142
arctg(6.6667)=1.4219
arctg(7.0000)=1.4289
```

Варианты заданий

Номер вар.	F(x)	A	B	M

1	x - sin(x)	0	п/2	10
2	sin(x)	п/4	п/2	15
3	cos(x)	п/3	2п/3	20
4	tg(x)	0	п/4	10
5	ctg(x)	п/4	п/2	15
6	arcsin(x)	0	1	20
7	arccos(x)	0.5	1	10
8	sin(x) - cos(x)	0	п/2	20
9	x sin(x)	0	3п	10
10	sin(1/x)	п/8	2/п	15
11	cos(1/x)	п/4	4/п	20
12	$\sin(x^2)$	п/6	2п/3	10
13	$\cos(x^2)$	п/3	3п/2	15
14	sin(x) + tg(x)	0	п/4	20
15	cos(x) + ctg(x)	п/4	п/2	10
16	tg(x/2)	0	2п/3	15
17	tg(x/2) + cos(x)	п/2	п	20
18	ctg(x/3) + sin(x)	п/4	п/2	10
19	sin(x/4)/4	п/2	п	15
20	arcctg(x)	1	5	20
21	tg(x/4) +sin(x/2)	-1	1	10
23	x^2tg(x/2)	5	8	15
24	tg(x)/x	-5	5	20
25	tg(x) + ctg(x/2)	0	3	10

Лабораторная работа № 4.

Организация циклов в программе.

Рекуррентной называется формула, связывающая значения p+1 соседних членов uk, uk-1, …, uk-p (k>=p+1) некоторой последовательности {un}(n=1, 2, …): uk=F(k, uk-1, …, uk-p). Рекуррентная формула позволяет шаг за шагом определить любой член последовательности, если известны p первых её членов u1, u2, …, up, где p называют порядком формулы. Рассмотрим пример.
Рассмотрим задачу нахождения n-го члена рекуррентной последовательности на примере чисел Фибоначчи. Каждое число Фибоначчи равно сумме двух предыдущих. В частности:
U3 = u2+u1 = 1+1 = 2;
U4 = u3+u2 = 1+2 = 3 и т.д.
Отсюда следует, что для получения очередного числа достаточно хранить два предыдущих. Таким образом, в программе постоянно используются три соседних числа Фибоначчи. Для их хранения достаточно ввести три переменных: A хранит uk, B хранит uk-1, C хранит uk-2. Для вычисления следующего числа Фибоначчи необходимо провести сдвиг, т.е. переписать содержимое B в C, а содержимое A в B. Исходя из этого, получим фрагмент

```
{фрагмент}
c:=1;{значение первого числа известно}
b;=1;{значение второго числа тоже известно}
k:=3;{начинаем вычисление с третьего числа}
while k<=n do {цикл, пока не найдено n-е число}
```

```
        begin
                a:=b+c;{вычисляем следующее число как сумму двух предыдущих}
                c:=b;{сдвигаем b в c для нахождения следующего числа}
                b:=a;{сдвигаем a в b для нахождения следующего числа}
                k:=k+1;{увеличиваем счетчик найденных чисел}
        end;
write(a);{выводим найденное число}
```

Цель задания:

1. Получение навыков в выборе и использовании операторов цикла.
2. Знакомство с итерационными процессами.

Постановка задачи:

Используя оператор цикла, найти сумму элементов, указанных в конкретном варианте. Результат напечатать, снабдив соответствующим заголовком.

Содержание отчета:

1. Постановка задачи.
2. Текст программы.
3. Результат решения конкретного варианта.

Методические указания:

При определении суммы членов ряда следует использовать рекуррентную формулу для получения следующего члена ряда.
Факториалом целого числа называют произведение
1*2*3*…*n = n!

n! = n*(n-1)

Например, требуется найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{2(n!)^2}{(3(2n)!)!}$.

Для получения рекуррентной формулы вычислим отношение:

$$\frac{a_{n+1}}{a_n} = \frac{2((n+1)!)^2 \cdot 3(2n)!}{3(2n+2)! \cdot 2(n!)^2} = \frac{n+1}{2(2n+1)},$$

откуда:

$$a_{n+1} = a_n \cdot \frac{(n+1)}{2(2n+1)}.$$

При составлении программы считать, что точность достигнута, если $a_n < \varepsilon$

Образец выполнения задания.

Лабораторная работа № 4, вариант № 8.

Организация циклов в программе.

Постановка задачи:

Используя оператор цикла, найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{n!}{n^n}$. Результат напечатать.

При определении суммы членов ряда следует использовать рекуррентную формулу, в нашем случае рекуррентная формула имеет вид: $a_{n+1} = a_n * \left(\frac{n}{n+1} \right)^n$

При составлении программы считать, что точность достигнута, если $a_n < 10^{-4}$

Текст программы:

```
program lab4{вариант № 11};
var sum,an:real;
    n:integer;
begin
clrscr;
 sum:=0;
 an:=1;
 n:=1;
 while an>0.0001 do
  begin
  sum:=sum+an;
  n:=n+1;
  an:=an*(exp(n*(ln(n/(n+1)))));
  end;
 writeln('Сумма ',n,' элементов равна =',sum:7:6);
end.
```

Результат решения конкретного варианта:

Сумма 12 элементов равна =1.759641

Варианты заданий

1) Найти сумму целых положительных чисел, кратных 3 и меньших 200.
2) Найти сумму целых положительных четных чисел, меньших 100.
3) Найти сумму целых положительных нечетных чисел, меньших 200.
4) Найти сумму целых положительных чисел, больших 20, меньших 100 и кратных 3.
5) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{(-1)^{n-1}}{n^n}$

6) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{1}{2^n} + \frac{1}{3^n}$

7) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{1}{((3n-2)(3n+1))}$

8) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{(2n-1)}{2^n}$

9) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{10^n}{n!}$

10) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{n!}{(2n)!}$

11) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{2^n n!}{n^n}$

12) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{3^n n!}{(3n)!}$

13) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{n!}{3n^n}$

14) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{(n!)^2}{2^{n^2}}$

15) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \lg(n!)e^{-n\sqrt{n}}$

16) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = 10^{-n}(n-1)!$

17) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{n^3}{(3n-3)!}$

18) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = \frac{n}{(n-1)^2}$

19) Найти сумму ряда с точностью $\varepsilon=10^{-4}$, общий член которого $a_n = e^n \cdot 100^{-n^2}$

20) Найти сумму 13 членов ряда, в котором $a_n = \frac{\ln(n!)}{n^2}$

21) Найти сумму 15 членов ряда, в котором $a_n = \frac{n^{\ln n}}{(\ln n)^n}$

22) Найти сумму 10 членов ряда, в котором $a_n = \frac{n!}{n^{\sqrt{n}}}$

23) Найти сумму 9 членов ряда, в котором $a_n = e^{-\sqrt{n}}$

24) Найти сумму 7 членов ряда, в котором $a_n = n^2 e^{-\sqrt{n}}$

3.Численные методы.

Довольно часто на практике приходится решать уравнения вида:

$$F(x) = 0, \qquad (1)$$

где функция F(x) определена и непрерывна на некотором конечном или бесконечном интервале $\alpha < x < \beta$.

Всякое значение $\bar{x}$ такое, что $F(x) \equiv 0$, называется корнем уравнения, а нахождение этого значения и есть решение уравнения.

На практике в большинстве случаев найти точное решение возникшей математической задачи не удается. Поэтому важное значение приобрели численные методы, позволяющие найти приближенное значение корня. Под численными методами подразумеваются методы решения задач, сводящиеся к арифметическим и некоторым логическим действиям над числами, т.е. к тем действиям, которые выполняет ЭВМ. [5]

Существует множество численных методов. Рассмотрим только три из них:

1. **метод итераций;**
2. **метод Ньютона;**
3. **метод половинного деления.**

3.1.Метод итераций

Этот метод заключается в замене уравнения (1) эквивалентным ему уравнением вида

$$x = \varphi(x) \qquad (2)$$

После этого строится итерационный процесс

$$x^{(n+1)} = \varphi(x^{(n)}) \qquad (3)$$

При некотором заданном значении $x^{(0)}$. Для приведения выражения (1) к требуемому виду (2) можно воспользоваться простейшим приёмом

$$f(x) = f(x) + x - x = 0,$$
$$x = x + f(x) = \varphi(x).$$

Если в выражении (2) положить $\varphi(x) = x + \tau * f(x)$, можно получить стандартный вид итерационного процесса для поиска корней нелинейного уравнения:

$$\frac{x^{(n+1)} - x^{(n)}}{\tau^{(n+1)}} - f\left(x^{(n)}\right) = 0.$$

Иначе можно получить уравнение (2) следующим способом: левую и правую часть уравнения (1) умножить на произвольную константу λ и прибавить к левой и правой части x, т.е. получаем уравнение вида:

$$x = x + \lambda F(x), \qquad (4)$$

где $f(x) = x + \lambda F(x)$.

На заданном отрезке [a; b] выберем точку x_0 – нулевое приближение – и найдем:

$x_1 = f(x_0)$,

потом найдем:

$x_2 = f(x_1)$,

и т.д.

Таким образом, процесс нахождения корня уравнения сводится к последовательному вычислению чисел:

$$x_n = f(x_{n-1}) \qquad n = 1,2,3..... .$$

Этот процесс называется **<u>методом итераций</u>**.

Если на отрезке [a; b] выполнено условие:

$$|f'(x)| \le q < 1,$$

то процесс итераций сходится, т.е.

$$\lim_{n \to \infty} x_n = \bar{x}.$$

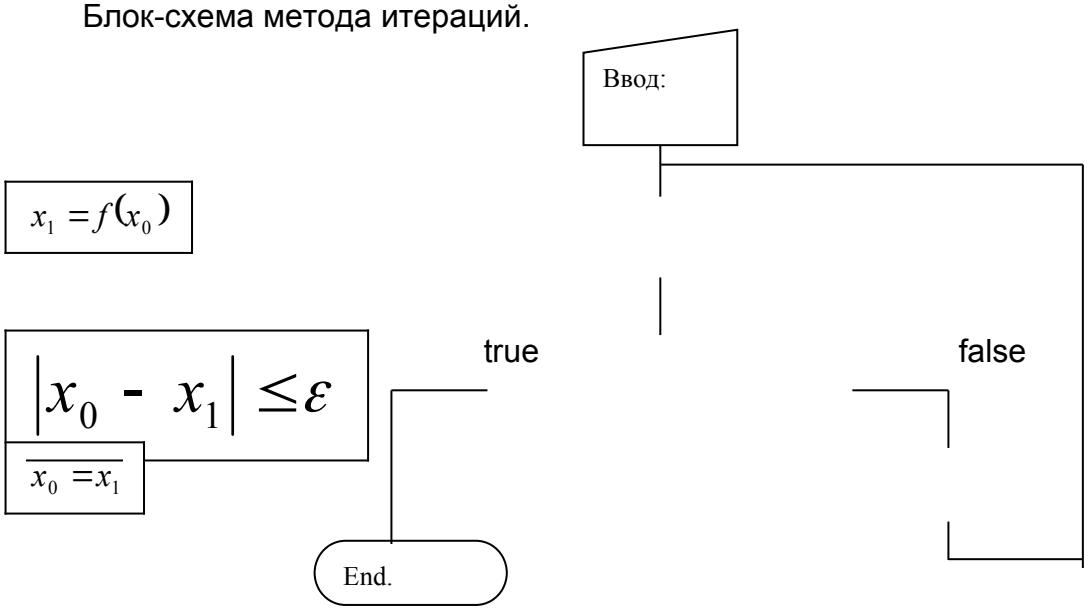
Процесс итераций продолжается до тех пор, пока

$$|x_n - x_{n-1}| \le \varepsilon,$$

где ε – заданная абсолютная погрешность корня x. При этом будет выполняться:

$$|\bar{x} - x_n| \le \varepsilon.$$

Блок-схема метода итераций.



<u>3.2.Метод Ньютона</u>

Для поиска корней уравнения (1) в окрестности решения $\tilde{x}$ выберем точку x и разложим функцию $f(x)$ в ряд Тейлора возле этой точки:

$$f(\tilde{x}) = f(x) + f'(x)(\tilde{x} - x) + \ldots$$

Отсюда следует приближённое равенство

$$f(\tilde{x}) \approx f(x) + f'(x)(\tilde{x} - x)$$

Которое с учётом

$$f(\tilde{x}) = 0$$

Позволяет получить выражение

$$\tilde{x} \approx x - \frac{f(x)}{f'(x)}$$

Приводящее к итерационному процессу следующего вида:

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})}$$

Выберем на отрезке[a; b] произвольную точку x_0 – нулевое приближение. Затем найдем:

$$x_1 = x_0 - \frac{F(x_0)}{F'(x_0)},$$

потом $$x_2 = x_1 - \frac{F(x_1)}{F'(x_1)}.$$

Таким образом, процесс нахождения корня уравнения сводится к вычислению чисел x_n по формуле:

$$x_n = x_{n-1} - \frac{F(x_{n-1})}{F'(x_{n-1})} \qquad n = 1,2,3..... .$$

Процесс вычисления продолжается до тех пор, пока не будет выполнено условие: $|x_n - x_{n-1}| < \varepsilon$.

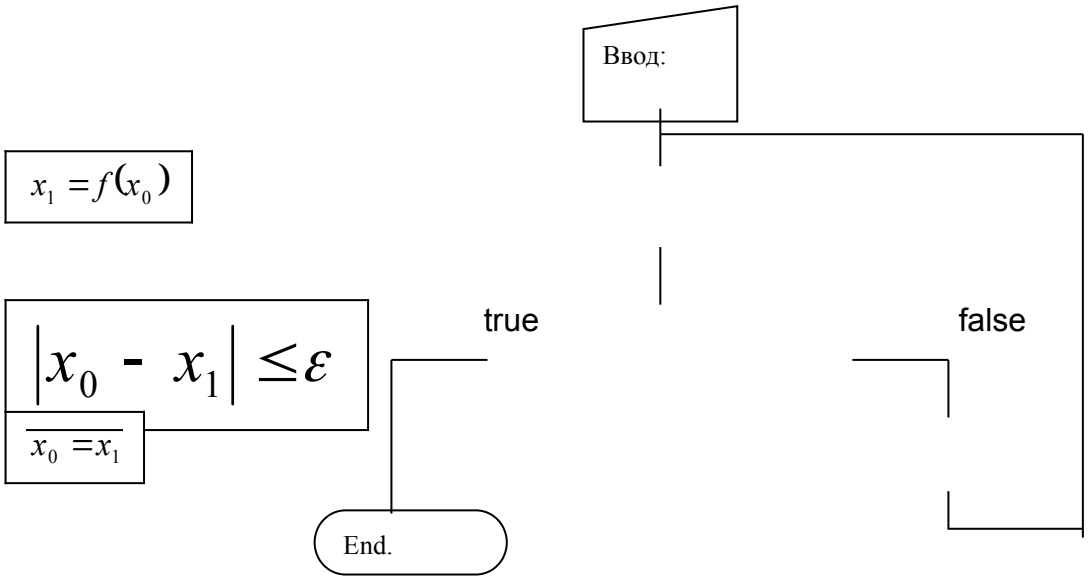
Очевидно, что метод Ньютона можно рассматривать как вариант метода простых итераций, при условии $\varphi(x) = x - \frac{f(x)}{f'(x)}$.

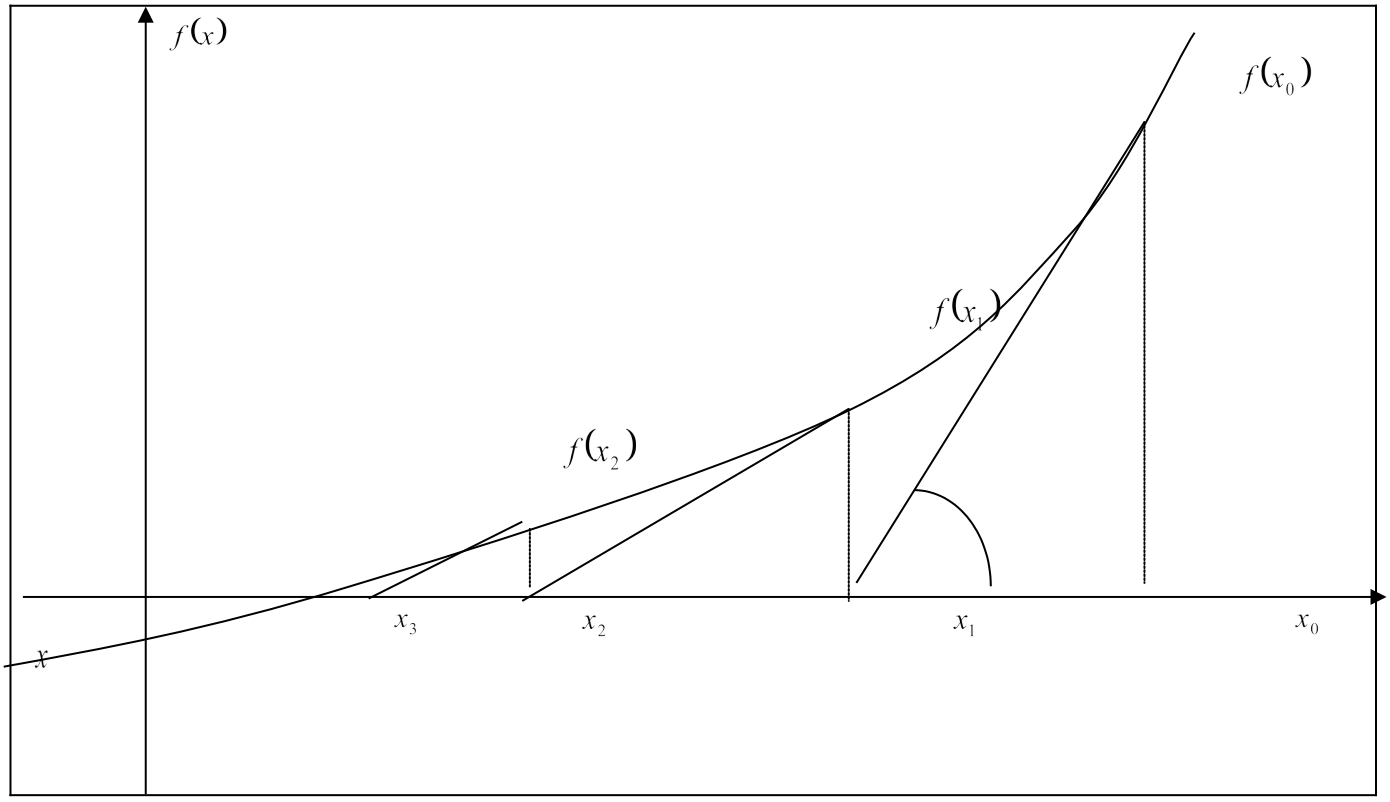
Геометрическая иллюстрация итерационного процесса метода Ньютона приведена на рисунке, из которого понятно, что каждое следующее приближение может быть определено из геометрических построений:

$$x^{(n+1)} - x^{(n)} = \frac{f(x^{(n)})}{tg\alpha} = \frac{f(x^{(n)})}{f'(x^{(n)})}$$

Этот процесс называется **<u>методом Ньютона</u>**.

Блок-схема метода Ньютона.





Геометрический смысл процедуры Ньютона

Пример. Требуется определить корни уравнения $f(x) = x^2 - a = 0$.
Согласно рассмотренному методу Ньютона строится итерационная процедура

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})}$$

Поскольку

$$f'(x^{n}) = 2x^{(n)}$$

$$x^{(n+1)} = x^{(n)} - \frac{(x^{(n)})^2 - a}{2x^{(n)}} = x^{(n)} - \frac{1}{2}x^{(n)} + \frac{a}{2x^{(n)}}.$$

Таким образом, применение процедуры метода Ньютона к заданному уравнению приводит к вычислительному процессу

$$x^{(n+1)} = \frac{1}{2}\left(x^{(n)} + \frac{2}{x^{(n)}}\right)$$

Для a=2 ‘точное’ решение $x = \sqrt{2} \approx 1,4142135624$.Результаты расчётов приведены в таблице 3.2.

Номер итерации	Приближения решения	Приближения решения
1	2,0	-10,0
2	1,5	-5,1
3	1,416666667	-2,746078431
4	1,414215686	-1,737194874
5	1,414213562	-1,444238095
6	1,4142135624	-1,414525655
7	1,4142135624	-1,414213597
8	1,4142135624	-1,4142135624

Последовательность получения приближённого решения уравнения $x^2 - 2 = 0$ методом Ньютона.

3.3. Метод половинного деления.

Пусть уравнение F(x) = 0 имеет один корень на отрезке [a;b]. Функция F(x) непрерывна на отрезке [a; b].

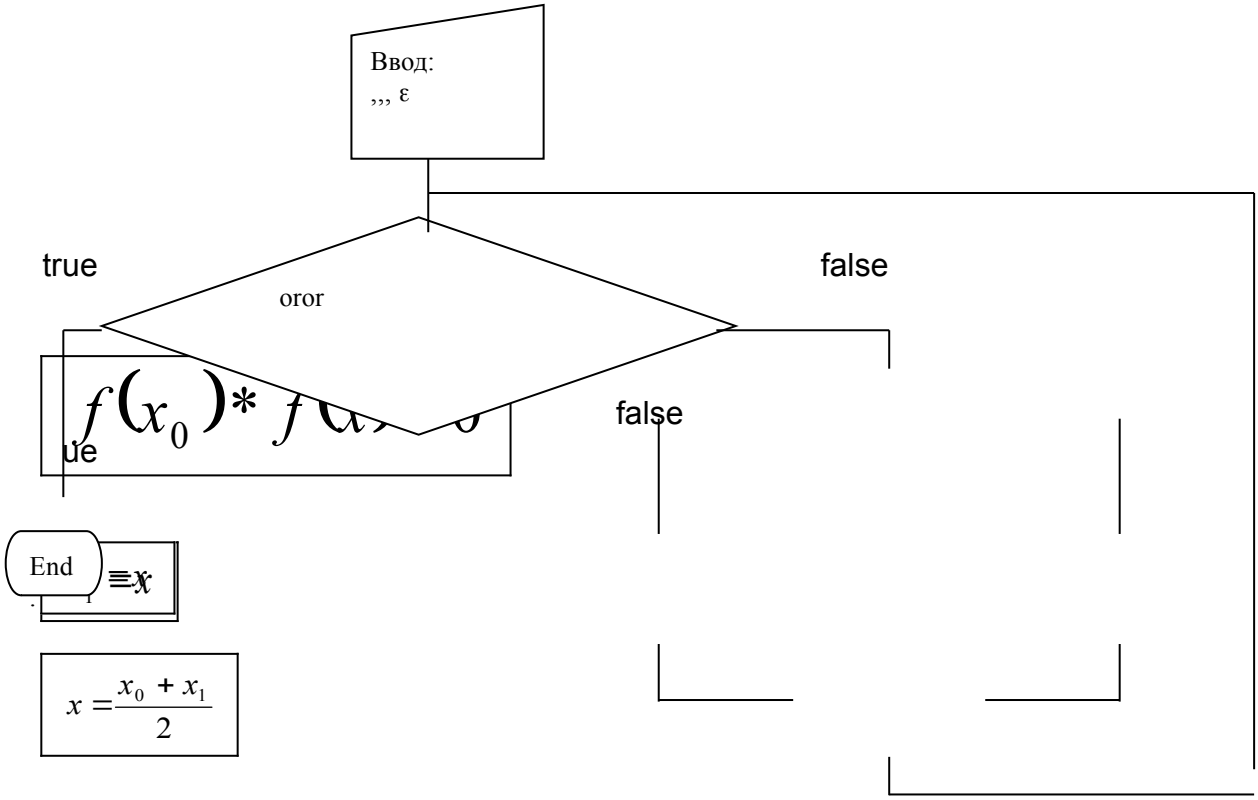
Метод половинного деления заключается в следующем:

Сначала выбираем начальное приближение, деля отрезок пополам, т.е. $x_0 = (a+b)/2$.

Если F(x)=0, то x_0 является корнем уравнения. Если $F(x) \neq 0$, то выбираем тот из отрезков, на концах которого функция имеет противоположные знаки. Полученный отрезок снова делим пополам и выполняем действия сначала и т.д.

Процесс деления отрезка продолжаем до тех пор, пока длина отрезка, на концах которого функция имеет противоположные знаки, не будет меньше заданного числа ε.

Блок-схема метода половинного деления.



Теорема математического анализа метода половинного деления.

Согласно тому что функция, непрерывна в замкнутом интервале и принимающая на концах этого интервала значения разных знаков, хотя бы один раз обращается в нудь внутри интервала.

Пусть функция $f(x)$ непрерывна на отрезке [x_0, x_1]. Процедура метода заключается в последовательном сокращении длинны отрезка для локализации корня уравнения. Первоначально проверяются значения заданной функции на концах отрезка.

В случае, если

$$f(x_0) * f(x_1) = 0,$$

Один из концов отрезка является корнем уравнения.

Пусть на концах отрезка значения функции имеют разные знаки, то есть имеет место соотношение $f(x_0) * f(x_1) < 0$,

Вычисляется значение аргумента в середине отрезка, $x_2 = \frac{x_0 + x_1}{2}$, и вычисляется значение функции $f(x_2)$ в этой точке.

Далее сравниваются знаки функций в точке x_2 например, в левой точке x_0 отрезка.

Если имеет место соотношение $f(x_0)*f(x_1)<0$ (рис.3.1), то корень следует искать на отрезке $\lfloor x_0, x_2 \rfloor$. В противном случае-корень разыскивается на отрезке $\lfloor x_2, x_1 \rfloor$, в результате выполненной операции исходный отрезок сократился вдвое.

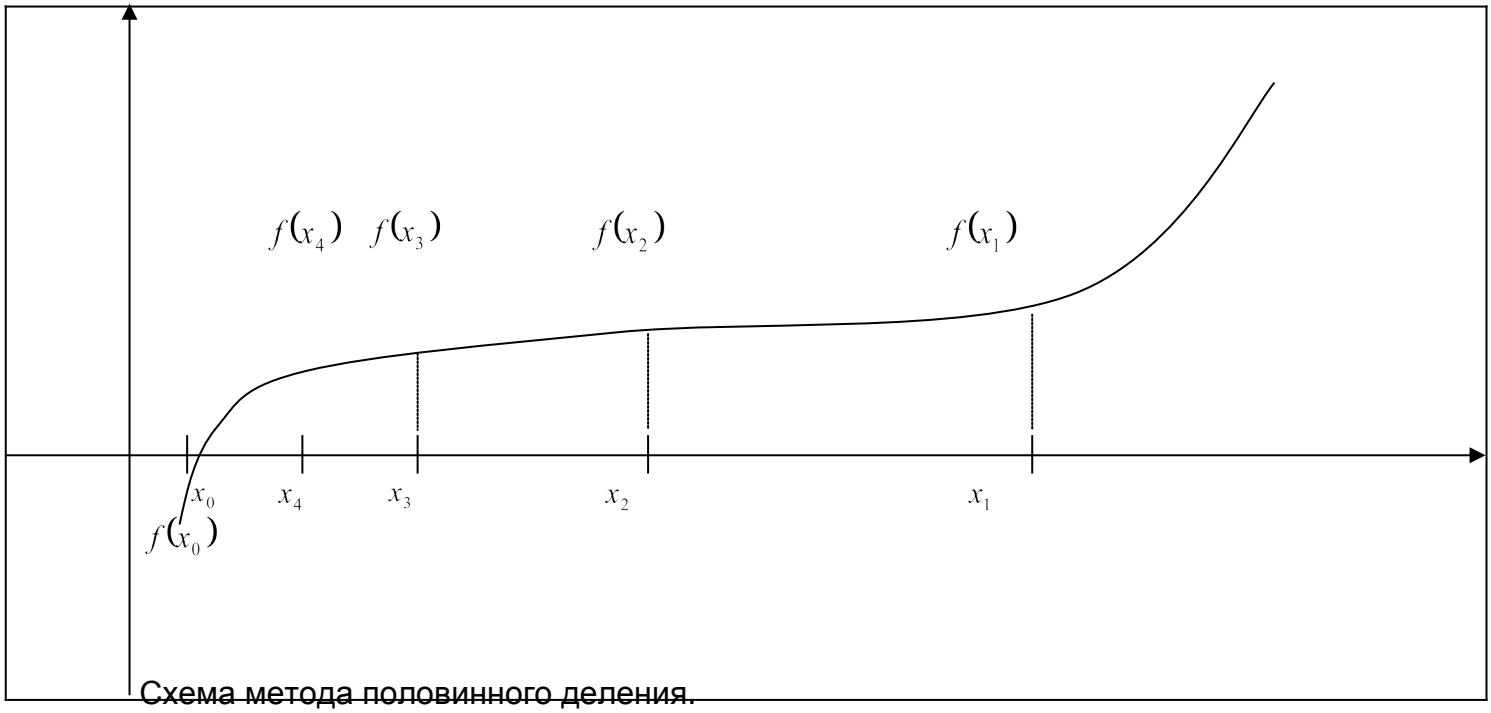


Схема метода половинного деления.

Далее, в зависимости от ситуации, отрезок вновь делится пополам,

$$x_3 = \begin{cases} \frac{x_2 + x_0}{2}, f(x_2)*f(x_0)<0 \\ \frac{x_2 + x_1}{2}, f(x_2)*f(x_0)>0 \end{cases}$$

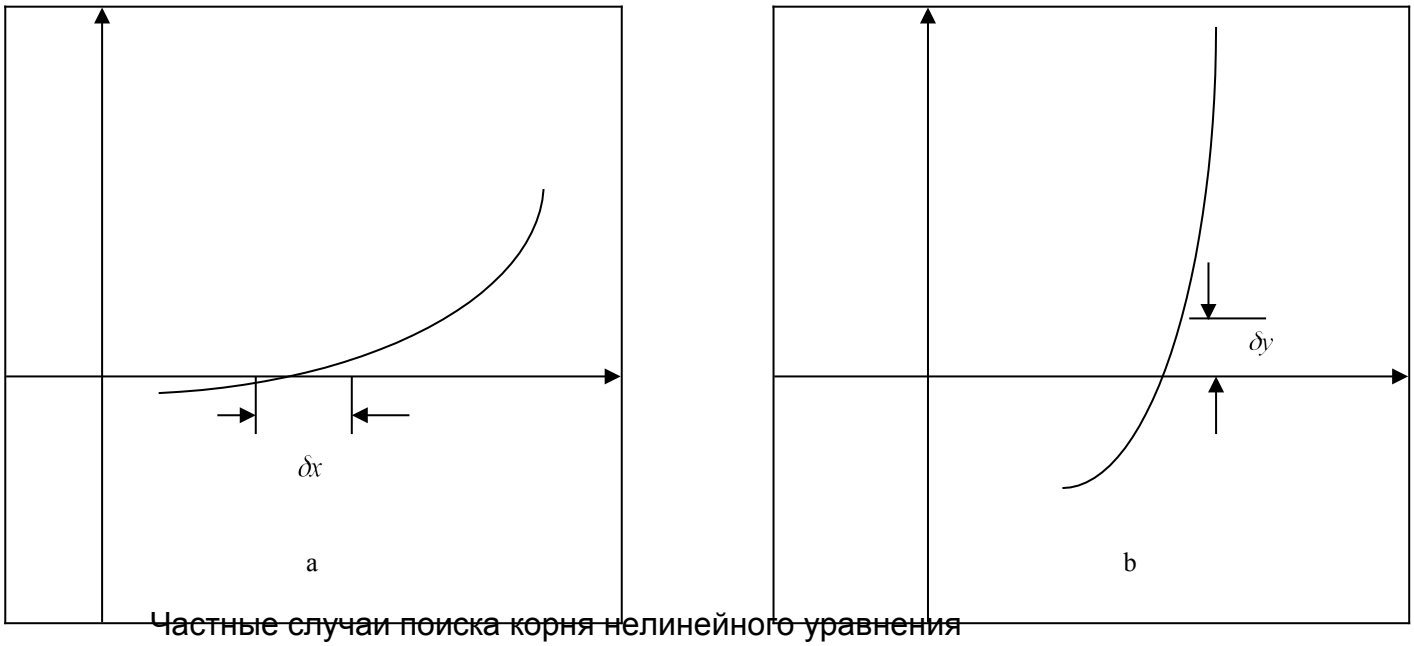
И так далее.

Для прекращения вычислительной процедуры применяют различные критерии:
-если функция достаточно ‘пологая’, имеет смысл использовать условие (рис. а).

$$\left| x_{k+1} - x_k \right| < \delta_x;$$

-если функция ‘круто’ меняет своё значение, целесообразно применять условие(рис. b).

$$\left| y_{k+1} \right| < \delta_y$$



Частные случаи поиска корня нелинейного уравнения

В случае, если заранее неизвестен характер ‘поведения’ функции, имеет смысл использовать одновременно оба условия для прекращения итерационного процесса.

Лабораторная работа № 5

Решение нелинейных уравнений.

Цель работы:

1. Получение практических навыков в организации итерационных циклов.
2. Знакомство с методами решения алгебраических и трансцендентных уравнений.
3. Получение навыков составления блок-схемы алгоритма и определения данных.

Постановка задачи:

1. Для конкретного варианта найти корень уравнения F(x) = 0 с точностью $\varepsilon=10^{-4}$, используя один из численных методов. На печать вывести вычисленное значение корня и для сравнения точное значение корня.
2. Значения a, b, x_0, ε вводятся с клавиатуры. Должен быть предусмотрен контроль вводимых значений.
3. В программе необходимо предусмотреть подсчет и вывод на печать числа итераций, за которое удается найти значение корня с заданной точностью.

Содержание отчета :

1. Постановка задачи для конкретного варианта и исходные данные.
2. Описание и блок-схема метода решения.
3. Текст программы.

Распечатка результатов работы программы в следующем виде:

РЕШЕНИЕ УРАВНЕНИЯ
ТОЧНОЕ ЗНАЧЕНИЕ КОРНЯ .....

ВЫЧИСЛЕННОЕ ЗНАЧЕНИЕ КОРНЯ .....….
ЧИСЛО ИТЕРАЦИЙ .....

Образец выполнения задания.

Лабораторная работа № 5, вариант № 3.

Решение нелинейных уравнений методом итераций.

Постановка задачи для конкретного варианта и исходные данные:

1. Найти корень уравнения : $x + \sqrt{x} + \sqrt[3]{x} - 2{,}5 = 0$ с точностью $\varepsilon=10^{-4}$, корень уравнения находится на отрезке (0.4, 1), используя метод итераций. На печать вывести вычисленное значение корня и для сравнения точное значение корня, точное значение корня x=0.7376.

Значения :
Xo – примерное значение корня,
ε - точность нахождения корня, вводятся с клавиатуры.
Должен быть предусмотрен контроль вводимых значений.

2. В программе необходимо предусмотреть подсчет и вывод на печать числа итераций, за которое удается найти значение корня с заданной точностью.

Описание и блок-схема метода решения:

Описание метода итераций:

Пусть уравнение $x + \sqrt{x} + \sqrt[3]{x} - 2{,}5 = 0$ имеет один корень на отрезке [a;b]. Функция F(x) непрерывна на отрезке [a; b].

Этот метод заключается в замене уравнения $x + \sqrt{x} + \sqrt[3]{x} - 2{,}5 = 0$ эквивалентным ему уравнением вида $x = 2.5 - \sqrt{x} - \sqrt[3]{x}$

$$f(x) = 2.5 - \sqrt{x} - \sqrt[3]{x}$$

После этого строится итерационный процесс:

На заданном отрезке [a; b] выберем точку x_0 – нулевое приближение – и найдем:

$x_1 = f(x_0)$,

потом найдем:

$x_2 = f(x_1)$,

и т.д.

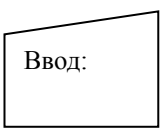
Таким образом, процесс нахождения корня уравнения сводится к последовательному вычислению чисел:

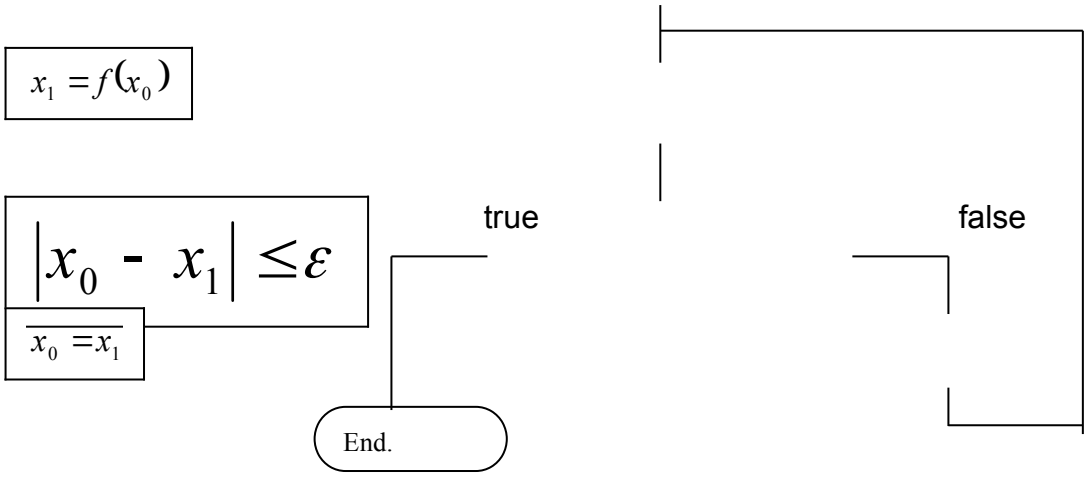
$x_n = f(x_{n-1})$ n = 1,2,3..... .

Процесс итераций продолжается до тех пор, пока $|x_0 - x_1| \le \varepsilon$

где ε – заданная абсолютная погрешность корня x.

Блок-схема метода итераций:





Текст программы.

```
program lab5{ вариант № 3};
uses crt;
var x0,x1,a,b,e:real;
   iteraz:integer;
function fun(x:real):real;
        begin
        fun:=2.5-sqrt(x)-exp((1/3)*(ln(x)));
        end;
begin
 clrscr;
 write('Введите приближённое значение X=');
 readln(x1);
 write('Введите точность e=');
 readln(e);
 iteraz:=0;
 repeat
  iteraz:=iteraz+1;
  x0:=x1;
  x1:=fun(x0);
 until (abs(x1-x0)<=e);
 writeln('Решение уравнения:');
 writeln('Точное значение корня..... ……0.7376');
 writeln('Вычисленное значение корня…',x1:6:5);
 writeln('Число итераций..……..... ',iteraz);
 writeln('Программа закончена, нажмите Enter.');
 readln;
end.
```

Распечатка результатов работы программы в следующем виде:

```
Решение уравнения:
Точное значение корня..... ……...0.7376
Вычисленное значение корня.. …0.73767
```

Число итераций..... ..989

Лабораторная работа № 5, вариант № 3.

Решение нелинейных уравнений методом Ньютона.

Постановка задачи для конкретного варианта и исходные данные:

1. Найти корень уравнения : $x + \sqrt{x} + \sqrt[3]{x} - 2{,}5 = 0$ с точностью $\varepsilon=10^{-4}$, корень уравнения находится на отрезке (0.4, 1), используя метод Ньютона. На печать вывести вычисленное значение корня и для сравнения точное значение корня, точное значение корня x=0.7376.

Значения :
Xo – примерное значение корня,
ε - точность нахождения корня, вводятся с клавиатуры.
Должен быть предусмотрен контроль вводимых значений.

2. В программе необходимо предусмотреть подсчет и вывод на печать числа итераций, за которое удается найти значение корня с заданной точностью.

Описание и блок-схема метода решения:

Описание метода Ньютона:

Пусть уравнение $x + \sqrt{x} + \sqrt[3]{x} - 2{,}5 = 0$ имеет один корень на отрезке [a;b]. Функция F(x) непрерывна на отрезке [a; b].

$$f(x) = x + \sqrt{x} + \sqrt[3]{x} - 2{,}5$$

$$f'(x) = 1 + \frac{1}{2\sqrt{x}} + \frac{1}{3\sqrt[3]{x}}$$

Приводящее к итерационному процессу следующего вида:

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})}$$

Выберем на отрезке[a; b] произвольную точку x_0 – нулевое приближение. Затем найдем:

$$x_1 = x_0 - \frac{F(x_0)}{F'(x_0)},$$

потом $$x_2 = x_1 - \frac{F(x_1)}{F'(x_1)}.$$

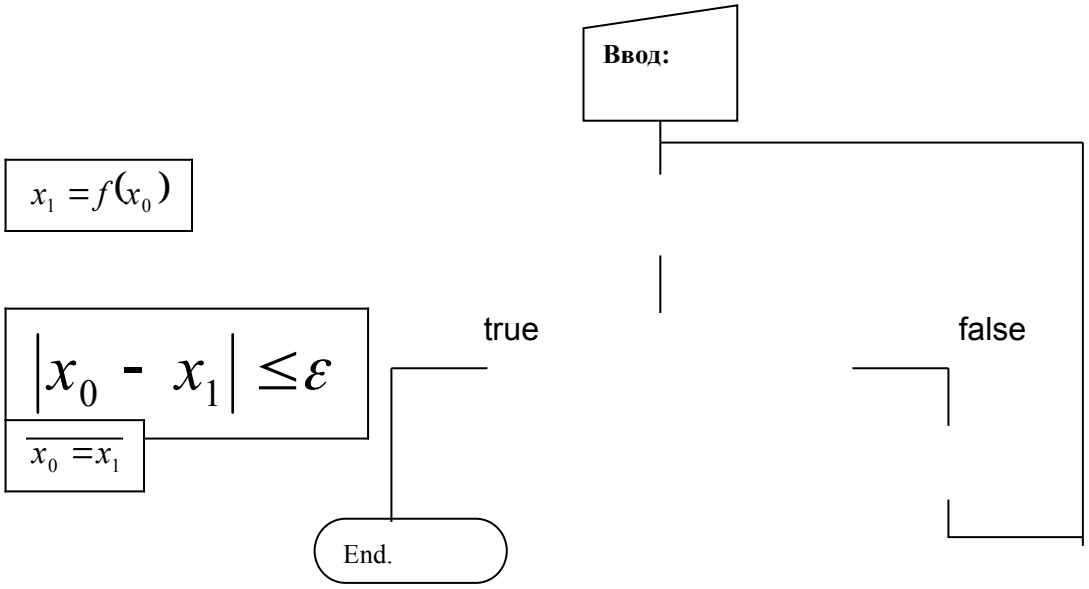
Таким образом, процесс нахождения корня уравнения сводится к вычислению чисел x_n по формуле:

$$x_n = x_{n-1} - \frac{F(x_{n-1})}{F'(x_{n-1})} \qquad n = 1,2,3.....$$

Процесс вычисления продолжается до тех пор, пока не будет выполнено условие:

$$|x_0 - x_1| \le \varepsilon$$

Блок-схема метода Ньютона:



Текст программы.

```
program lab5{ вариант № 3};
uses crt;
var x0,x1,a,b,e:real;
   iteraz:integer;
function fun(x:real):real;
        begin
        fun:=(x+sqrt(x)+exp((1/3)*(ln(x)))-2.5)/(1+1/(2*sqrt(x))+1/(3*exp((1/3)*(ln(x)))));
        end;
begin
 clrscr;
 write('Введите приближённое значение корня X=');
 readln(x1);
 write('Введите точность e=');
 readln(e);
 iteraz:=0;
 repeat
  iteraz:=iteraz+1;
  x0:=x1;
  x1:=x0-fun(x0);
 until (abs(x1-x0)<=e);
writeln('Решение уравнения:');
 writeln('Точное значение корня..... ……0.7376');
 writeln('Вычисленное значение корня…',x1:6:5);
 writeln('Число итераций..……..... ',iteraz);
 writeln('Программа закончена, нажмите Enter.');
 readln;
end.
```

Распечатка результатов работы программы в следующем виде:

Решение уравнения:
Точное значение корня..... ……...0.7376
Вычисленное значение корня.. …0.73762
Число итераций.....…… ..3

Лабораторная работа № 5, вариант № 3.

Решение нелинейных уравнений методом половинного деления.

Постановка задачи для конкретного варианта и исходные данные:

1. Найти корень уравнения : $x + \sqrt{x} + \sqrt[3]{x} - 2{,}5 = 0$ с точностью $\varepsilon=10^{-4}$, корень уравнения находится на отрезке (0.4, 1), используя методов половинного деления. На печать вывести вычисленное значение корня и для сравнения точное значение корня, точное значение корня x=0.7376.

Значения :
(a, b) – отрезок на котором находится корень уравнения,
Xo – примерное значение корня,
ε - точность нахождения корня,
вводятся с клавиатуры.
Должен быть предусмотрен контроль вводимых значений.

2. В программе необходимо предусмотреть подсчет и вывод на печать числа итераций, за которое удается найти значение корня с заданной точностью.

Описание и блок-схема метода решения:

Описание метода половинного деления:

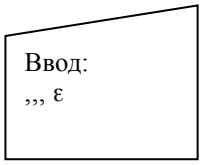
Пусть уравнение $x + \sqrt{x} + \sqrt[3]{x} - 2{,}5 = 0$ имеет один корень на отрезке [a;b]. Функция F(x) непрерывна на отрезке [a; b].

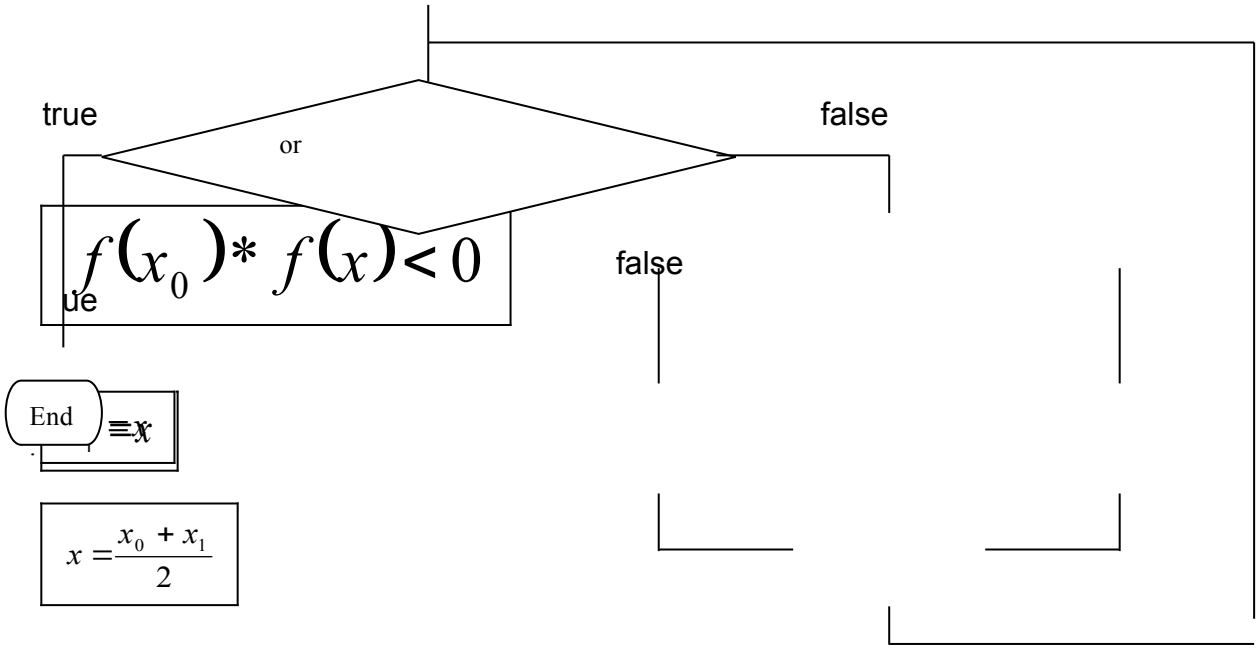
Метод половинного деления заключается в следующем:

Сначала выбираем начальное приближение, деля отрезок пополам, т.е. $x_0 = (a+b)/2$. Если F(x)=0, то x_0 является корнем уравнения. Если $F(x) \neq 0$, то выбираем тот из отрезков, на концах которого функция имеет противоположные знаки. Полученный отрезок снова делим пополам и выполняем действия сначала и т.д.

Процесс деления отрезка продолжаем до тех пор, пока длина отрезка, на концах которого функция имеет противоположные знаки, не будет меньше заданного числа ε.

Блок-схема метода половинного деления:





Текст программы.

```
program lab5{ вариант № 3};
uses crt;
var x,a,b,e:real;
    iteraz:integer;
function fun(x:real):real;
        begin
        fun:=x+sqrt(x)+exp((1/3)*(ln(x)))-2.5;
        end;
begin
 repeat
  clrscr;
  writeln('Корень уравнения находиться на интервале [a,b]');
  write('Введите [a=');
  readln(a);
  write('Введите [b=');
  readln(b);
  write('Введите приближённое значение корня X=');
  readln(x);
  write('Введите точность e=');
  readln(e);
 until (b-a>e) or (x>a) or (x<b) or (a<>0);
 iteraz:=0;
 while (fun(x)<>0) and (abs(a-b)>e) do
 begin
  iteraz:=iteraz+1;
  if (fun(a)*fun(x))<0
     then b:=x
     else a:=x;
  x:=((a+b)/2);
 end;
 writeln('Решение уравнения:');
 writeln('Точное значение корня.....……..0.7376’);
```

```
 writeln('Вычисленное значение корня.. ',x:6:5);
 writeln('Число итераций..... ',iteraz);
 writeln('Программа закончена, нажмите Enter.');
 readln;
end.
```

Распечатка результатов работы программы в следующем виде:

```
Решение уравнения:
Точное значение корня..... ……..0.73760
Вычисленное значение корня.. …0.73764
Число итераций..... ..14
```

Варианты заданий.

N^0 вар.	Уравнение	Отрезок, содержащий корень	Метод	Точное значение корня
1	$3\sin\sqrt{x} + 0{,}35x - 3{,}8 = 0$	[2;3]	Итераций	2,2985
2	$0{,}25x^3 + x - 1{,}2502 = 0$	[0;2]	Ньютона	1,0001
3	$x - \frac{1}{3 + \sin 3{,}6x} = 0$	[0;0,85]	Итераций	0,2624
4	$0{,}1x^2 - x\ln x = 0$	[1;2]	Ньютона	1,1183
5	$tgx - \frac{1}{3}tg^3x + \frac{1}{5}tg^5x - \frac{1}{3} = 0$	[0;8]	Половинного деления	0,3333
6	$\arccos x - \sqrt{1 - 0{,}3x^3} = 0$	[0;1]	Итераций	0,5629
7	$3x - 4\ln x - 5 = 0$	[2;4]	Ньютона	3,2300
8	$\cos\frac{2}{x} - 2\sin\frac{1}{x} + \frac{1}{x} = 0$	[1;2]	Половинного деления	1,8756
9	$\sqrt{1 - 0{,}4x^2} - \arcsin x = 0$	[0;1]	Итераций	0,7672
10	$e^x - e^{-x} - 2 = 0$	[0;1]	Ньютона	0,8814
11	$\sin(\ln x) - \cos(\ln x) + 2\ln x = 0$	[1;3]	Половинного деления	1,3749
12	$x - 2 + \sin\frac{1}{x} = 0$	[1,2;2]	Итераций	1,3077
13	$e^x + \ln x - 10x = 0$	[3;4]	Ньютона	3,5265
14	$\cos x - e^{\frac{-x^2}{2}} + x - 1 = 0$	[1;2]	Половинного деления	1,0804
15	$1 - x + \sin x - \ln(1 + x) = 0$	[0;1,5]	Итераций	1,1474
16	$3x - 14 + e^x - e^{-x} = 0$	[1;3]	Ньютона	2,0692
17	$\sqrt{1 - x} - tgx = 0$	[0;1]	Половинного деления	0,5768
18	$x + \cos(x^{0{,}52} + 2) = 0$	[0,5;1]	Итераций	0,9892
19	$3\ln^2 x + 6\ln x - 5 = 0$	[1;3]	Ньютона	1,8832
20	$\sin x^2 + \cos x^2 - 10x = 0$	[0;1]	Половинного деления	0,1010
21	$x^2 - \ln(1 + x) - 3 = 0$	[2;3]	Итераций	2,0267

22	$2x \sin x - \cos x = 0$	[0,4;1]	Ньютона	0,6533
23	$e^x + \sqrt{1 + e^{2x}} - 2 = 0$	[-1;0]	Половинного деления	-0,2877
24	ln x -x + 1,8 = 0	[2;3]	Итераций	2,8459

4. Случайные числа.

Мы не можем заранее сказать, какая сторона монеты или игрального кубика окажется сверху, какую карту мы вытащим из колоды. Говорят, что результат такого эксперимента является случайным числом. Повторяя эксперимент много раз, можно получить последовательность случайных чисел. Интересно, что невозможно предвидеть значение нового члена ряда – он не зависит от предыдущих членов.

Случайные числа широко используются в задачах, моделирующих жизненные ситуации (игры, эпидемии заболеваний и др.). В Паскале реализован датчик или генератор псевдослучайных чисел. Числа, вырабатываемые при помощи приведенных ниже функций, называют псевдослучайными («похожими» на случайные), поскольку это модель случайных чисел, иногда весьма приближенная.

Randomize – инициирует датчик.

Random – генерирует случайные действительные числа из диапазона [0;1]

Random (x) – генерирует любые случайные целые числа из диапазона [0;x].

Random (x) + random – генерирует случайное действительное число из диапазона [0;x];

Random (x) + random + y – генерирует случайное действительное число из диапазона [y; x + y];

Random (x) + random – y – генерирует случайное действительное число из диапазона [-y; x – y];

2* x* Random – x – генерирует случайное действительное число из диапазона [-x; x].

Метод Монте-Карло (метод статистических испытаний)

Случайные числа широко используются для приближенного вычисления площади с помощью метода Монте-Карло. Суть метода очень проста. Пусть есть некоторая фигура, площадь которой необходимо вычислить. Размещаем эту фигуру внутри стандартного квадрата со сторонами, параллельными осям. Пусть про любую точку квадрата можно узнать попадает ли эта точка внутрь фигуры или нет. Тогда площадь может быть вычислена следующим образом: поделив количество точек, попавших внутрь фигуры, на количество всех точек, попавших в квадрат, можно узнать, какую часть площади квадрата занимает фигура, домножив это отношение на площадь квадрата, получим площадь фигуры. Ясно, что число точек, попавших внутрь фигуры тем больше, чем больше фигура, а точность решения будет пропорциональна количеству точек в квадрате. Пара случайных чисел в этом методе может быть рассмотрена как координаты точки на плоскости.

Пример: Определить площадь круга с радиусом R=1.

Решение:

```
program task;
var  i,n, m: integer;  {n- общее кол-во точек, m- кол-во точек внутри круга}
       x, y: real;     {координаты точки}
begin
   wriiteln (‘задайте количество точек’);
   readin (n);
   Randomize;
   m: = 0;
```

```
    for i: = 1 to n do
    begin
    x: = 2* random – 1; y: = 2* random – 1;
    if sqr (x) + sqr (y) < = 1 then m: - m + 1;
    end;
writeln ('площадь круга равна ‘, 4* m/n);
readln;
End.
```

Результаты выполнения программы:

```
Задайте количество точек: 8

Площадь круга равна : 2.5000000000E+00
```

5. Массивы.

5.1. Процедуры и функции.

Процедуры и функции представляют собой важный инструмент TP, позволяющий писать хорошо структурированные программы.

Процедура представляет собой отдельную часть программы, которая имеет своё имя.

Процедуры и функции представляют собой относительно самостоятельные фрагменты программы, оформленные определенным образом и снабженные именем.

Описать программу - это значит указать ее заголовок и тело. В заголовке объявляют имя подпрограммы и формальные параметры. Для функции еще указывается тип возвращаемого ею результата. За заголовком следует тело подпрограммы, которое состоит из раздела описаний и раздела исполняемых операторов.

Заголовок процедуры имеет вид:

PROCEDURE <имя>[(< сп. ф. п. >)];

Заголовок функции:

FUNCTION<имя> [(< сп. ф. п.>)]: <тип>,

где <имя> - имя подпрограммы;
<сп. ф. п.> - список формальных параметров;
<тип> - тип возвращаемого функцией результата;

Список формальных параметров необязателен и может отсутствовать. Если же он есть, то в нем должны быть перечислены их имена и тип:

PROCEDURE SB (a : real; b : integer; c : char);

Операторы тела подпрограммы рассматривают список формальных параметров как своеобразное расширение раздела описаний: все переменные из этого списка могут использоваться в любых выражениях внутри подпрограммы.

Вызов и выполнение осуществляется при помощи оператора процедуры:

<имя процедуры>(<список фактических параметров>);

Между формальными и фактическими параметрами должно быть полное соответствие, т.е. должно быть одинаковое количество, порядок их следования должен быть один и тот же, тип каждого фактического параметра должен совпадать с типом соответствующего ему формального параметра. Определение формального параметра тем или иным способом существенно только для вызывающей программы: если формальный параметр объявлен как параметр-переменная, то при вызове подпрограммы ему должен соответствовать фактический параметр в виде переменной нужного типа; если формальный параметр

объявлен как параметр-значение, то при вызове ему может соответствовать произвольное выражение. Контроль над неукоснительным соблюдением этого правила осуществляется компилятором языка Turbo Pascal.

Расмотрим взаимодействие основной программы с процедурами и функциями.

```
Program
Const
   MWSt = 14.0;
Var
   Value, cost:real;
Procedure Input_number (var Input_value:real);
{данная процедура решает задачу ввода стоимости товара после предварительного
запроса }
Begin
   Write('Ведите, велечину стоимости товара: ')
   Readln (Input_value);
End;
Prodcedure Calulation_cost (Cost,Value:real);
{данная процедура позволяет вычислить велечину цены ,учитывая при этом велечину
налога}
Begin
   Cost:=(1.0+MWSt/100.0)*Value;
End;
Procedure Output_result (Input_value:real);
{данная процедура позволяет вывести на экран результаты расчетов}
Begin
   Writeln;{Оператор Writeln заданный без параметров,}
   Writeln;{Переводит курсор на начало следующей строки}
   Writeln('Расчетная цена с учетом',MWSt:5:2,'% налога','составляет: ',Cost:7:2,' $.');
End;
Begin          {Основной блок программы}
   Input_number (Value);
   Calculation_cost (Cost,Value);
   Output_result (Cost);
End.
```

На примере сложения двух чисел проиллюстрируем возможности ТП 7.0 по оформлению программ при помощи процедур и функций.

```
Program
{Программа демонстрирует различия между процедурами и функциями.}
Uses  Crt;
Var
   a,b,Sum_numbers : integer;
Prosedure  Summing_up (Var sum : integer; a,b : integer);
Begin
   Sum := a + b ;
End;
Function  Sum(a ,b : integer) : integer;
Begin
   Sum := a + b ;
End;
Begin
   Clrscr;
```

```
    a:= 12;
    b:= 15;
{Сумма чисел с использованием процедуры}
Summing_up  (Sum_numbers,   a,   b);
Writeln('Сумма чисел равна: ',Sum_numbers);
{Сумма чисел с использовнием функции}
Sum_numbers := Sum(a , b);
Writeln('Сумма чисел равна:  ',Sum_numbers);
Writeln('Сумма чисел равна:  ', Sum(a, b));
End.
```

5.2. Одномерные массивы.

5.2.1. Описание массивов.

Массив – это формальное объединение нескольких однотипных объектов (чисел, символов, строк и т.п.), рассматриваемое как единое целое.

Массив описывается следующим образом:

<имя_типа> = ARRAY [<сп_инд_типов>] OF <тип>,

где ARRAY, OF – зарезервированные слова (массив, из);

<имя_типа> -- правильный идентификатор;

<сп_инд_типов> -- тип-диапазон, с помощью которого компилятор определяет число элементов массива.

<тип> -- любой тип TP, в том числе и другой массив.

Определить переменную как массив можно непосредственно при описании этой переменной, без предварительного описания типа массива, например:

```
var  f : array [1..12] of integer.
```

5.2.2. Классы задач по обработке массивов.

Перечисленные ниже классы задач относятся как к одномерным, так и к двумерным массивам.

Классы задач:

1. Однотипная обработка всех или указанных элементов массива.
2. Задачи, в результате решений которых изменяется структура (порядок следования элементов) массива.
3. Обработка нескольких массивов одновременно. Сюда же относятся задачи, когда по-разному обрабатываются подмассивы одного и того же массива.
4. Поисковые задачи для массивов.
5. Сортировка массивов.

5.2.2.1. Однотипная обработка всех или указанных элементов массивов.

Решение задач первого класса сводится к установлению того, как обрабатывается каждый элемент массива или указанные элементы. Затем выбирается подходящая схема перебора элементов, в которую вставляются операторы обработки элементов массива.

Пример: Найти максимальный (минимальный) элемент массива

Решение:

```
const n = 30;
var a: array [1..n] of integer;
  i, max: integer;
begin
  for i: = 1 to n do
  read (a[i]);
  max: = a[1];
  for i: = 2 to n do
  if a [i] > max then max: = a [i];
  writeln (‘максимальный элемент массива равен ‘ , max)
end.
```

5.2.2.2. Задачи, в результате решения которых изменяется структура массива.

Особенностью задач этого класса является изменение порядка следования элементов массива. Для этого часто приходится менять местами элементы массива. Для чего необходимо ввести дополнительную переменную, для того, чтобы не потерять старое значение элемента массива. Это выполняется с помощью приведенных ниже операторов.

Пример: Поменять местами пары соседних элементов, т.е. первый и второй, третий и четвертый и т.д. (n-1)-ый и n-ый

Решение:

```
i: = 2
while i < n do
begin
    r: = a[i];
    a[i]: = a[j];
    a[j]: = r;
    i: = i + 2
End.
```

5.2.2.3. Обработка нескольких массивов одновременно.

Если обрабатываются несколько массивов одновременно, то для каждого массива нужно выбрать подходящую схему перебора, завести свой индекс, следить , чтобы индекс не вышел за границы массива. В некоторых частных случаях для обработки нескольких массивов бывает достаточно одного индекса, потому что элементы массива обрабатываются «синхронно», то есть, зная индекс элемента одного массива, можно вычислить по некоторой формуле индекс соответствующего ему элемента другого массива. Если такой формулы установить не удается, то говорят, что массивы обрабатываютя «асинхронно».

Пример: Дан массив целых чисел. Необходимо сформировать второй массив, содержащий четные элементы первого массива, при этом расположить элементы во втором массиве:

а) на тех же позициях, что и в первом;

б) сдвинуть к началу массива.

Решение:

Вариант 1:

```
const nn = 30;
var a, b: array [1..n] of integer;
   i, n: integer;
begin
   write (‘задайте количество элементов массива’);
   readin (n);
   for i: = 1 to n do
   begin
            read (a[i]);
            if a[i] mod 2 = 0 then b[i]: = a[i];
   End;
   for i: = 1 to n do
   write (b[i],”);
End.
```

Вариант 2.

```
const nn = 30;
var a, b: array [1..n] of integer;
   i, k, n: integer;
begin
   write (‘задайте количество элементов массива’);
   readln (n);
   for i: = 1 to n do
   read (a[i]);
   k: = 0;    {в массиве b нет ещё элементов}
   for i: = 1 to n do
   if a[i] mod 2 = 0 then begin
            k: = k + 1;
                     b[k]: = a[i];
            end;
   for i: = 1 to k do
   write (b[i], “);
End.
```

5.2.2.4. Поисковые задачи для массивов.

Часто в программировании возникает задача отыскать первый элемент, совпадающий с заданным x. Для решения этой задачи могут быть предложены следующие варианты:

- линейный поиск;

поиск с барьером;

дихотомический поиск.

Линейный поиск

Алгоритмом такого поиска был рассмотрен при решении типовых задач на построение циклических алгоритмов. Напомним, что особенностью такого поиска является две причины прекращения поиска:

Элемент найден (это программируется с помощью логической переменной)

Просмотрены все элементы, но заданный элемент так и не нашли

(i > n)

```
const n = 20; {количество элементов в массиве}
var a: array [1..n] of integer;   {исходный массив}
```

```
    i, x: integer;
    f: boolean;
begin
    write (‘задайте искомый элемент’);
    readln (x);
    writeln (‘задайте элементы массива’);
    for i: = 1 to n do
            readln (a[i]);
            f: = false;   {элемент ещё не найден}
            i: = 1;
            while (i < = n) and not f do
            if a[i] = x then f: = true    {нашли}
            else i: = i + 1         {переходим к следующему элементу}
            if f then writeln (‘ нашли элемент с номером‘, i)
            else writeln (‘такого элемента нет’)
End.
```

Поиск с барьером

Применяется широко распространенный прием фиктивного элемента, или барьера, расположенного в конце массива. Использование барьера позволяет упростить условие окончания цикла, т.к. заранее ясно, что хотя бы один элемент, равный a, в массиве есть. При этом необходимо увеличить размер массива на 1.

```
const n = 20; {количество элементов в массиве}
var a: array [1..n + 1] of integer; {исходный массив}
    i, x: integer;
begin
    write (‘задайте искомый элемент’);
    readln (x);
    writeln (‘задайте элементы массива’);
    for i: = 1 to n do
            readln (a[i]);
            a[n + 1]: = x;  {установлен барьер, равный x, в          конце}
            i: = 1;             {массива}
            while a[i] <> x do
            i: = i + 1;    {переходим к следующему элементу}
            if i <> n + 1 then writeln (‘нашли элемент с номером’, i)
            else writeln (‘такого элемента нет’):
End.
```

Дихотомический поиск (поиск элемента в упорядоченном массиве)

Алгоритм дихотомического поиска достаточно прост. Делим массив пополам и определяем в какой из частей может находиться искомый элемент. Поскольку массив упорядочен, то сравнивая искомый элемент со средним элементом массива, легко определить интересующую нас половину. Затем выбранную половину опять делим пополам и определяем в какой половине находится искомый элемент. Этот процесс продолжаем до тех пор, пока не будет найден искомый элемент, либо левая граница нового отрезка не станет больше правой. В последнем случае можно сделать вывод о том, что искомого элемента в массиве нет.

```
const n = 20; {количество элементов в массиве}
var a: array [1..n] of integer; {исходный массив}
    i, x, k, m: integer;
    left, right, mid: integer: {левая, правая граница и середина}
```

```
    f: boolean;                 {отрезка}
begin
    write ('задайте искомый элемент');
    readln (k);
    for i: = 1 to n do
            readln (a [i]);
            {упорядочивание массива по возрастанию}
    for i: = 1 to n – 1 do
            begin
            m: = i;
    for j: = i + 1 to  n do
            if a[j] < a[m] then m: =j;
            x: = a[i];
            a[i]: = a[m]; a[m]: = x
    end;
    {поиск элемента}
    f: = false;  {элемент не найден}
    left: = 1; right: = n;
    repeat {поиск элемента в части массива от элемента [left] до
             элемента [rigth]}
    mid: = (left + right) div 2;
    if k < a[mid] then right: = mid – 1 {элемент в левой части}
    else if k > a[mid] then left: = mid + 1 {элемент в правой части}
            else f: = true;  {нашли}
            else f: = true;   {нашли}
    until f or (left > rigth);
    if f then writeln ('элемент с номером', mid: 2, 'совпадает с
                                исковым', k)
    else writeln ('не нашли');
End.
```

5.2.2.5. Сортировка массивов.

5.2.2.5.1. Сортировка вставкой

Массив разделяется на две части: отсортированную и неотсортированную. Элементы из неотсортированной части поочередно выбираются и вставляются в отсортированную часть так, чтобы не нарушить в ней упорядочность элементов.

В начале работы алгоритмы в качестве отсортированной части массива принимают только один первый элемент, а в качестве неотсортированной части – все остальные элементы.

Таким образом, алгоритм будет состоять из n – 1-го прохода (n – размерность массива), каждый из которых будет включать четыре действия:

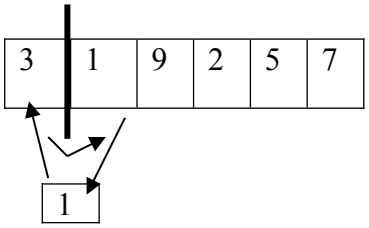
Взятие очередного i-го неотсортированного элемента и сохранение его дополнительной переменной.

Поиск позиции j в отсортированной части массива, в которой присутствие взятого элемента не нарушит упорядоченности элементов.

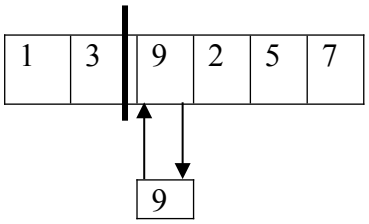
Сдвиг элементов массива от i – 1-го до j-го вправо, чтобы освободить найденную позицию вставки.

Вставка взятого элемента в найденную j-ю позицию.

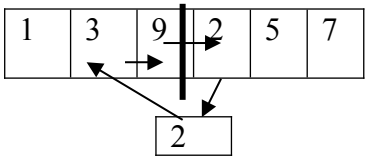
1 шаг:



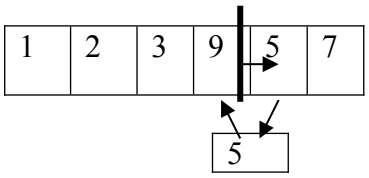
2 шаг:



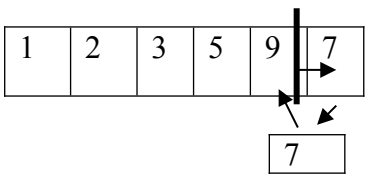
3 шаг:



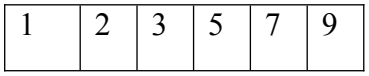
4 шаг:



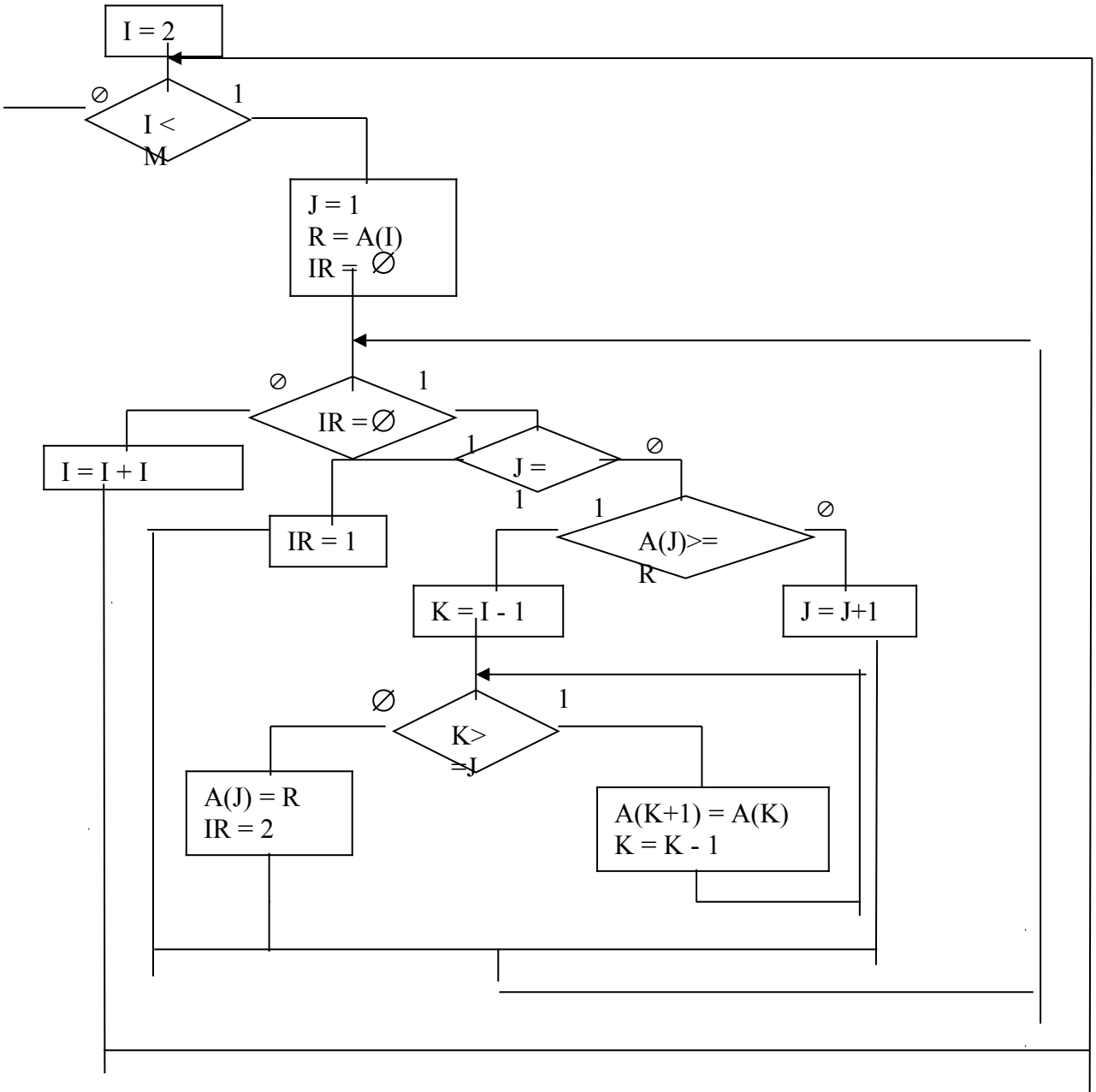
5 шаг:



Результат:



БЛОК – СХЕМА



Программа, реализующая рассмотренный алгоритм, будет иметь следующий вид.

```
Program Insertion Sort;
Uses Crt;
Const
        n =10; {длина массива}
type
        TVector  = array  [1…n]  of  Real;
Var
        Vector  :  TVector;
         B        :   Real;
```

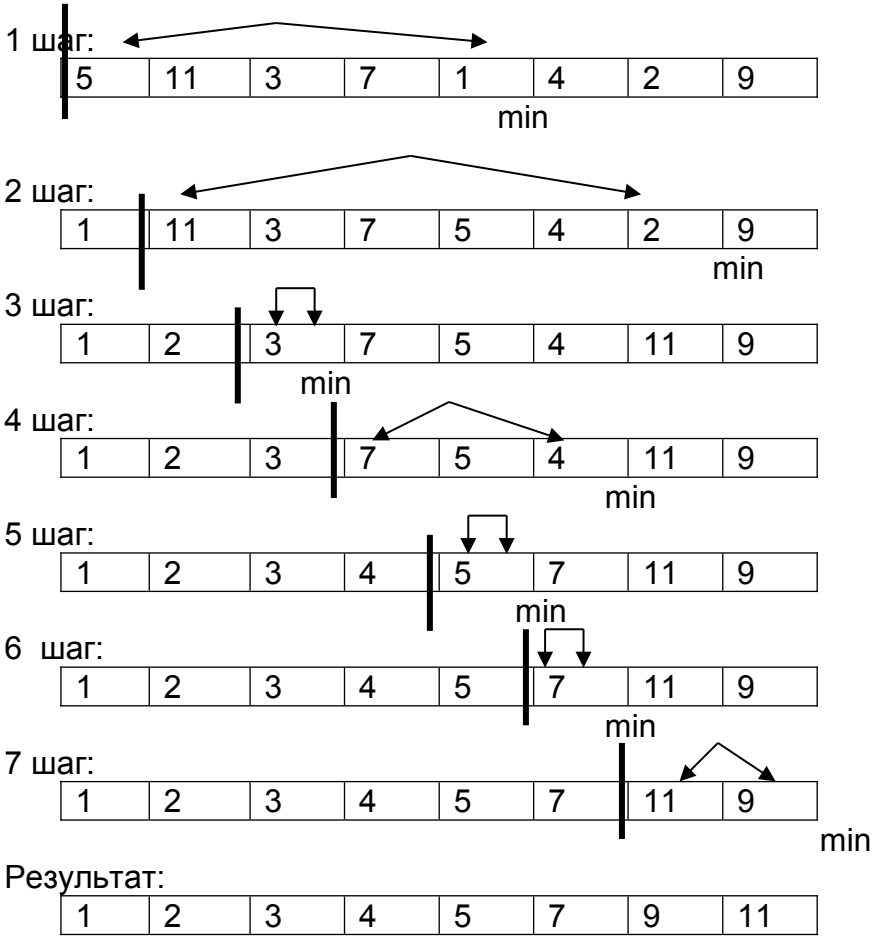
```
        i,  j, K :   Integer;
Begin
        ClrScr;
        Writeln (‘Введите элементы массива:’);
        For  i:=1 to n do Read (Vector [i]); Readln;
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
        for   i:=2 to n do
begin
        B:=Vector [i]; {взятие неотсортированного элемента}
                          {цикл поиска позиции вставки}
        j:=1
        While (B>Vector [j] )  do
        j:= j+1 {после окончания цикла индекс  j фиксирует позицию}
                {вставк}                                              {цикл сдвига
элемента для освобождения позиции  вставки}
        for K:= i-1   downto j do
        Vector  [K+1]: = Vector  [K];
        {Вставка взятого элемента на  найденную позицию}
        Vector [j]: = B;
         End;
{ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
      Writein (‘Отсортированный  массив:’);
       For i: = 1 to n do write    (‘vector [i] : 8 : 2);
       Writeln;
End.
```

Результат работы :

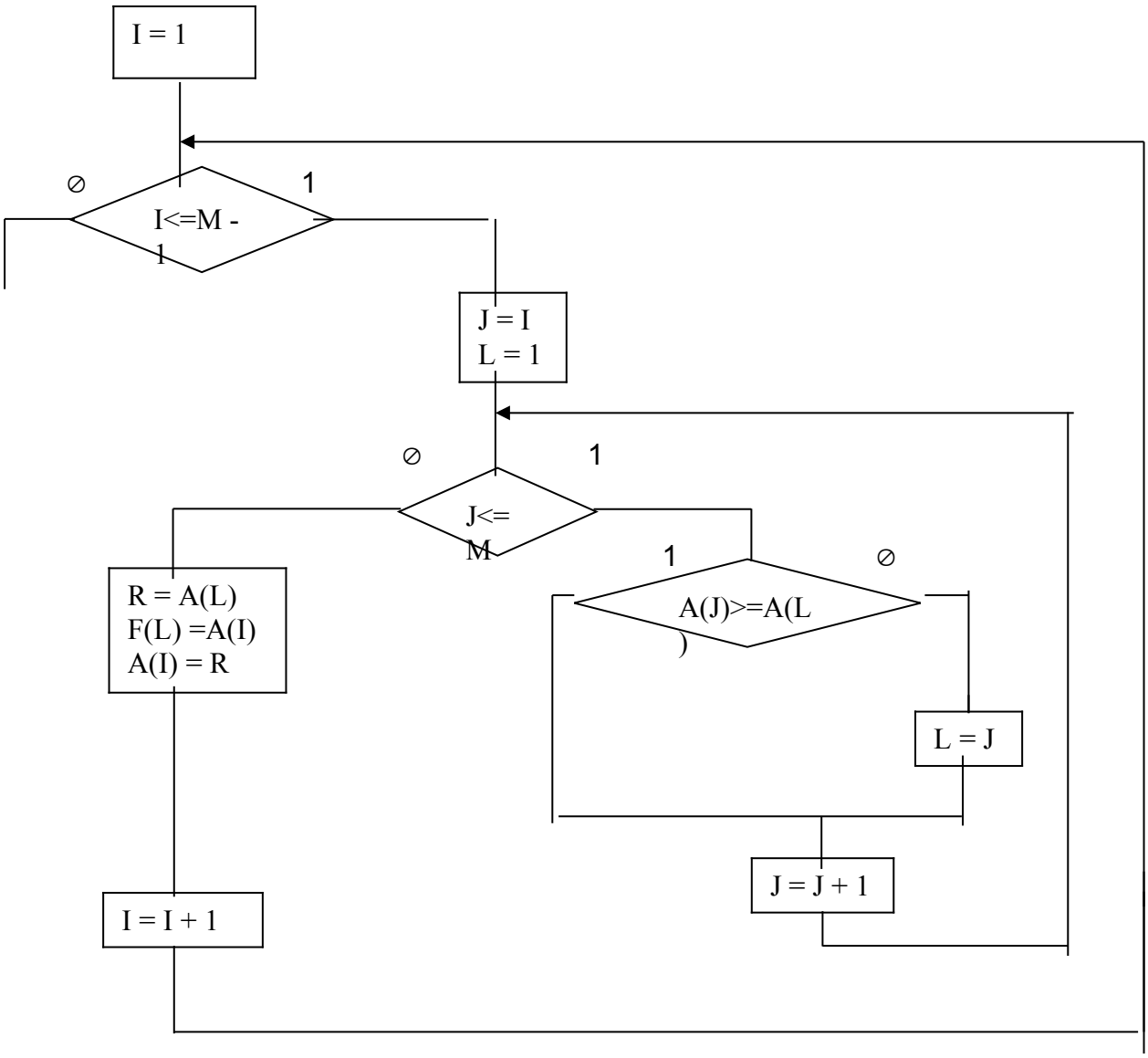
```
Введите элементы массива :
  7654098321
Отсортированный массив :
   0123456789
```

5.2.2.5.2. Сортировка выбором

Находим (выбираем) в массиве элемент с минимальным значением на интервале от 1-го элемента до n-го (последнего) элемента и меняем его местами с первым элементом. На втором шаге находим элемент с минимальным значением на интервале от 2-го до n-го элемента и меняем его местами со вторым элементом. И так далее для всех элементов до n-1-го.


1 шаг:
5 11 3 7 1 4 2 9
min
2 шаг:
1 11 3 7 5 4 2 9
min
3 шаг:
1 2 3 7 5 4 11 9
min
4 шаг:
1 2 3 7 5 4 11 9
min
5 шаг:
1 2 3 4 5 7 11 9
min
6 шаг:
1 2 3 4 5 7 11 9
min
7 шаг:
1 2 3 4 5 7 11 9
min
Результат:
1 2 3 4 5 7 9 11

БЛОК – СХЕМА



Программа реализующая метод выбора, будет иметь следующий вид:

```
Program Selection Sort;
Uses Crt;
const
        n=20;  { длина массива}
typc
        TVector = array [1…n] of Real;
Var
        Vector : TVector;
        Min     :  Real;
        Imin, S: Integer;
        i         : Integer;
Begin
        Clr Srt:
       Writeln (‘введите элементы массива:’);
for i: = 1 to n do Read (Vector [i]);  Readln;
{ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
for S:=1 to n-1  do
```

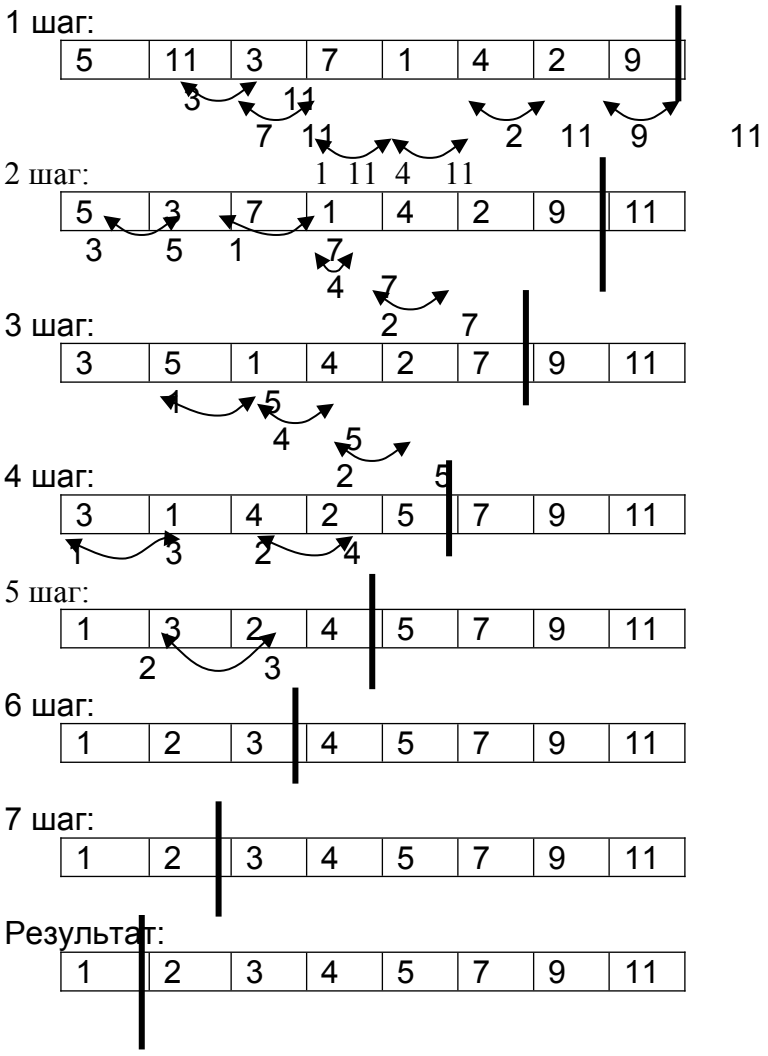
```
begin
{поиск минимального элемента в  диапазоне от }
     {S-го элемента до n-го}
Min: = Vector [S];
I min: = S;
For i: = S+1 to n do
           If   Vector  [i] < Min  then
           begin
           Min: = Vector [i];
           I min: = I;
End;
 {обмен местами минимального и  S-го элемента}
    Vector [Imin]: = Vector [S];
     Vector [S]: = Min;
      End;
 { - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
    Writeln ('Отсортированный массив:');
    For i: = 1 to n do Write (Vector [i]: 8 : 2);
    Writeln;
End.
```

Результат работы :

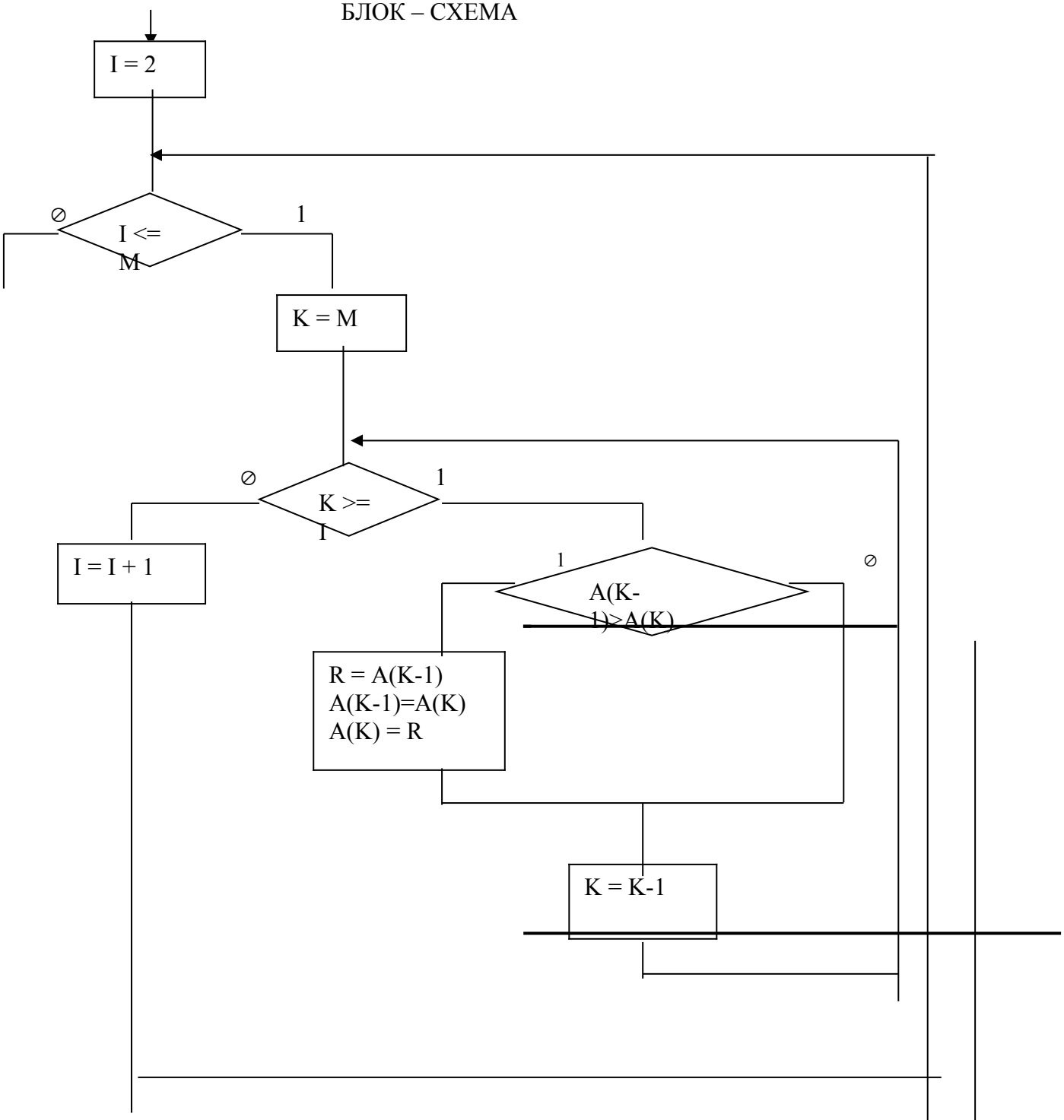
```
Введите элементы массива :
  7654098321
Отсортированный массив :
   0123456789
```

5.2.2.5.3. Сортировка обменом («пузырьковая сортировка»)

Слева направо поочередно сравниваются два соседних элемента, и если их взаиморасположение не соответствует заданному условию упорядоченности, то они меняются. Далее берутся два следующих соседних элемента и так далее до конца массива. После одного такого прохода на последний n-ой позиции массива будет стоять максимальный элемент («всплыл первый пузырек»). Поскольку максимальный элемент уже стоит на своей последней позиции, то второй проход обменов выполняется до n-1 элемента. И так далее. Всего требуется n-1 проход.


1 шаг:
5 11 3 7 1 4 2 9
3 11
7 11 2 11 9 11
2 шаг:
1 11 4 11
5 3 7 1 4 2 9 11
3 5 1 7
4 7
2 7
3 шаг:
3 5 1 4 2 7 9 11
1 5
4 5
2 5
4 шаг:
3 1 4 2 5 7 9 11
1 3 2 4
5 шаг:
1 3 2 4 5 7 9 11
2 3
6 шаг:
1 2 3 4 5 7 9 11
7 шаг:
1 2 3 4 5 7 9 11
Результат:
1 2 3 4 5 7 9 11

БЛОК – СХЕМА



Программа реализирующая метод обмена («пузырька»), будет иметь следующий вид:

```
Program Duddle Sort;
Uses  Crt;
Const
       N = 20;    { длина массива}
Type
       TVector = array [1…n] of Real;
Var
   Vector : Tvector;
        B       :  Real;
        i, K    : Integer;
```

```
begin
        ClrScr;
        Writeln ('введите элементы массива:');
        for i: = 1 to n do Read (Vector [i]); Readln;
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
        for K: = n  downto  2 do
{"всплывание" очередного максимального элемента}
{на К-ю позицию}
         for i: = 1 to K-1 do
         if  Vector [i] > Vector [i+1] then
 begin
         B: = Vector [i];
         Vector [i]: = Vector [i+1];
         Vector [i+1]: = B;
         End;
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - }
         Writeln  ('отсортированный массив:');
         For i:  = 1 to n  do Write ('Vector [i]: 8 : 2');
         Writeln;
End.
```

Результат работы:

```
Введите элементы массива:
  7654098321
Отсортированный массив:
   0123456789
```

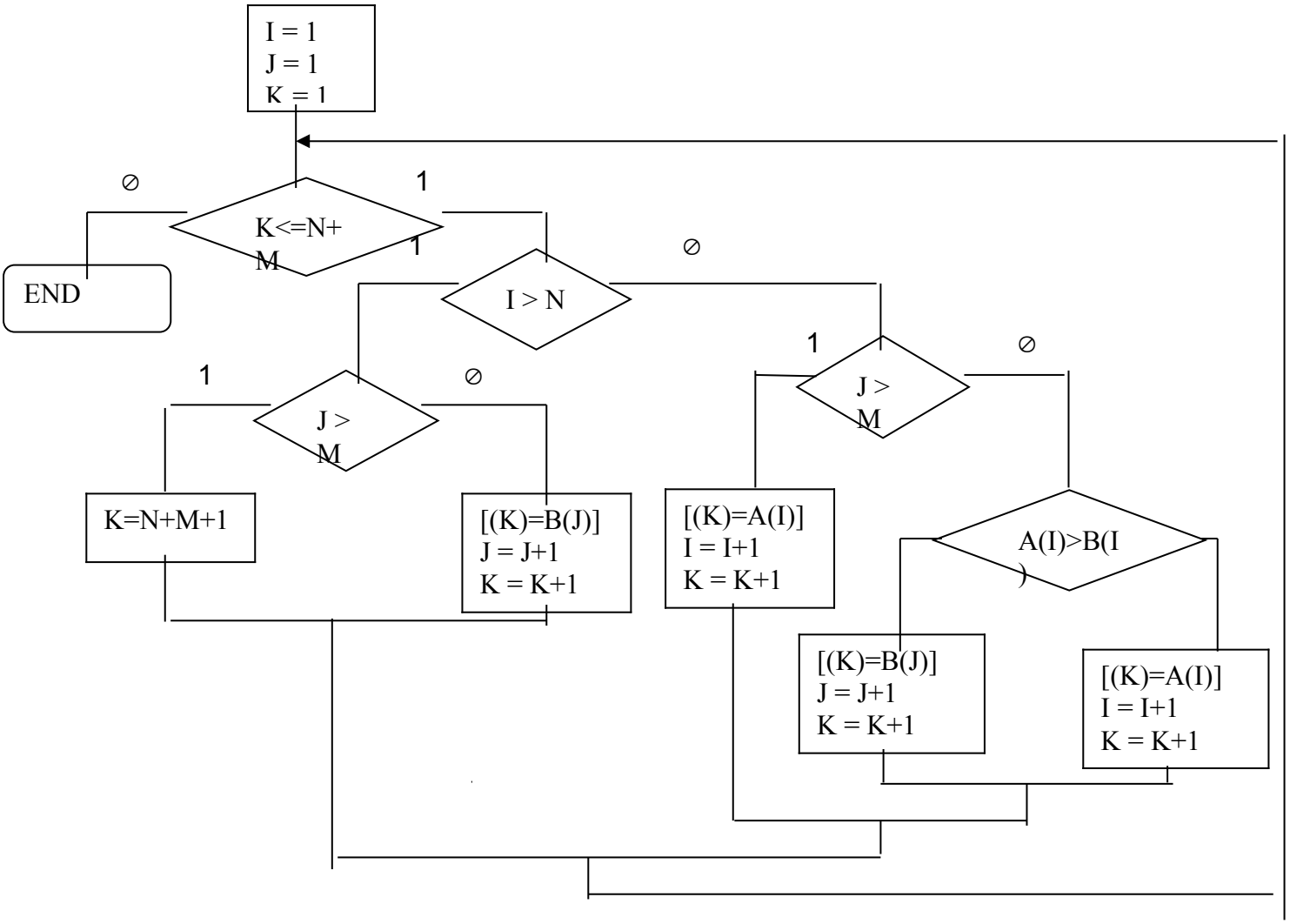
5.2.2.5.4. Сортировка фон Неймана (слиянием)

Заданы два упорядоченных по возрастанию элементов одномерных массива: a размерности n и b размерности m. Требуется получить третий массив с размерности n+m, который содержал бы все элементы исходных массивов, упорядоченных по возрастанию.

Алгоритм решения этой задачи известен как «сортировка фон Неймана» или сортировка слиянием. Идея алгоритма состоит в следующем. Сначала анализируются первые элементы обоих массивов. Меньший элемент переписывается в новый массив. Оставшийся элемент последовательно сравнивается с элементами из другого массива. В новый массив после каждого сравнения попадает меньший элемент. Процесс продолжается до исчерпания элементов одного из массивов. Затем остаток другого массива дописывается в новый массив. Полученный новый массив упорядочен таким же образом, как исходные.

Программа, реализующая метод Фон-Неймана, имеет следующий вид:

БЛОК – СХЕМА




```
program confluence;
   const n = 10;
         m = 20;
                    l = n + m;
           var     x: array [1..n] of integer;
                    y: array [1..m] of integer;
           z: array [1..l] of integer;
            k, i, j: integer;
begin
           for i: = 1 to n do
           read (x[i]);  {формирование первого упорядоченного массива}
           for i: = 1 to m do
           read (y[i]); {формирование второго упорядоченного массива}
I:= 1; j:=1; l:=1 {i-индекс первого массива, j-индекс              второго
                          массива, l-индекс результирующего массива}
```

```
   while (i<=n) and (j<=m) do {пока не закончились элементы в
одном из массивов}
            if x[i] < y[i] then {переписываем элементы из первого массива}
            begin
                z[k]: = x[i];
                i: = i + 1; k: = k + 1;
            end
            else   {переписываем элемент из второго массива}
            begin
                z[k]: = y[j];
                j: = j + 1; k: = k + 1;
            end;
      while i < = n do {если не закончились элементы в первом массиве,}
      begin     {переписываем их в результирующий массив}
      z[k]: = x[i]; i: = i + 1; k: = k+1;
      end;
      while j < = m do {если не закончились элементы во втором массиве}
           begin              {переписываем их в результирующий массив}
      z[k]: = y[j]; j: = j + 1; k: = k+ 1;
      end;
      for i: = 1 to l do {вывод на экран упорядоченного массива,}
      writeln (z[i]);      {полученного слиянием}
   End.
```

Результаты работы:

```
0123456789
0123456789
00112233445566778899
```

5.2.2.5.5. Шейкер-сортировка

Шейкер — сортировка является модификацией сортировки методом пузырька. Здесь, как и в методе пузырька проводят попарное сравнение элементов и обменов в паре. Если имеется необходимость, но первый проход осуществляют снизу вверх, второй – сверху вниз и т.д. Иными словами меняется направление прохода.

```
   program sheyker;
   uses crt;
   const n=30;
   var a:array[1..n]of integer;
      x,j,i,u:integer;
      h:boolean;
   begin
    clrscr;
    writeln('Массив до сортировки:');
    for i:=1 to n do
    begin
     a[i]:=round(random*999);
     write(a[i],'-');
    end;
    writeln;
```

```
writeln('Массив после сортировки:');
h:=true;u:=n;i:=2;
repeat
if h then begin
        for j:=u downto i do
        if a[j-1]>a[j] then begin
                        x:=a[j-1];
                        a[j-1]:=a[j];
                        a[j]:=x;
                        h:=false;
                       end;
        u:=u-1;
        end
    else begin
        for j:=i to u do
        if a[j+1]<a[j] then begin
                        x:=a[j+1];
                        a[j+1]:=a[j];
                        a[j]:=x;
                        h:=true;
                       end;
        i:=i+1;
        end;
until u=i;
for i:=1 to n do
 write(a[i],'-');
readln;
end.
```

Результаты выполнения программы:

```
Массив до сортировки:
0-31-860-202-273-671-318-162-372-425-82-474-70-840-60-293-916-368-774-328-697-84
3-717-306-162-329-466-246-825-279-
Массив после сортировки:
0-31-60-70-82-162-162-202-246-273-279-293-306-318-328-329-368-372-425-466-474-67
1-697-717-774-825-840-843-860-916-
```

5.3. Двумерные массивы.

5.3.1. Описание двумерных массивов.

Двумерный массив можно рассматривать как одномерный массив, каждый элемент которого сам является одномерным массивом. Поэтому для работы с элементами двумерного массива нужно организовать два цикла. Каждый из них отвечает за перебор значений соответствующего индекса. Для двумерного массива можно использовать те же схемы перебора, что и для одномерного, но комбинаций здесь будет в два раза больше.

Рассмотрим один из способов ввода элементов двумерного массива. Будем использовать схему перебора по одному от начала массива к концу. Считаем что массив имеет размерность n * m

For i : = 1 to n do {перебираем строки двумерного массива}

```
For j: = 1 to m do {перебираем столбцы двумерного массива}
        Read (a [i,j]).
```

Транспонирование двумерного массива, значит переставить местами его стоки и столбцы. Например для исходного массива:

```
2  3  получить:   1  4  7
4  5  6                 2 5  8
7  8  9                 3  6  9
```

Из приведенного примера хорошо видно, что диагональные элементы в результате обмена остаются на своих местах, обмениваются местами элементы, расположенные симметрично относительно главной диагонали.

```
For i: = 1 to n do{перебираем все строки массива}
For j: = 1 to i-1 do {перебираем элементы до главной диагонали}
   Begin r: = a [i, j]; a [i, j]: = a [j, i]; a [j,i]: = r ; end.
```

Нахождение максимального (минимального) элемента двумерного массива. Эта задача совпадает с решением задачи для одномерного массива. Отличие заключается в необходимости для двумерного массива вложенных циклов перебора. Фрагмент программы приведен ниже:

```
i max: =1; j max: = 1;{предлагаем максимальный первый элемент}
for i: = 1 to n do
   for j: = 1 to n do
        if a [i max, j max] < a [i, j]
        then begin i max: = i; jmax = j;    end.
```

5.3.2. Сортировка двумерных массивов

Отсортировать элементы двумерного массива по элементам второй строки.

```
Исходный массив.                    Результат
 1  2  3  4  5                      5  2  4  3  1
 9  3  7  3  1                      1  3  3  7  9
 6  7  8  9  1                      1  7  9  8  6
```

От сортировки одномерного массива этот случай отличается только тем, что переставлять нужно не два сравниваемых элемента, а два столбца:

```
for i: = 1  to n – 1  do
   for j: = i+1 to n do
        if a [2, i] > a [2, j]
        then  for k: = 1 to n do
                begin r: = a [k, i]; a [k, i]; = a[k, j]; a [k, j]: =r
                end.

                    {Сортировка массивов}
     { 1. Вставками }
     { 2. Обменом }
     { 3. Выбором }
     { 4. Фон Неймана (Слияние двух отсортированных массивов)}

   program sor1;
   uses crt;
   const n=4;
   var mas:array[1..n,1..n]of integer;
      buf,l,i,j,a,c,nextmas:integer;
      quit:boolean;
```

```
procedure print;
begin
 for i:=1 to n do
  begin
  for j:=1 to n do
   write(' ',mas[i,j],' ');
  writeln;
  end;
 writeln;writeln('Ќ ¦¬ЁвЃ Enter ¤«п Їа®¤®«¦ЃЁп !');
 readln;
end;
procedure next;
begin
 j:=j+1;
 if j=n+1 then begin j:=1;i:=i+1;end;
end;
procedure last;
begin
 j:=j-1;
 if j=0 then begin j:=n;i:=i-1;end;
end;
begin
 clrscr;
 for i:=1 to n do
  for j:=1 to n do
   mas[i,j]:=round(random*(9));
 print;
 i:=1;j:=1;
 for l:=2 to (n*n) do
 begin
  next;buf:=mas[i,j];a:=i;c:=j;
  last;quit:=true;
  while (buf<mas[i,j])and(quit) do
  begin
   nextmas:=mas[i,j];
   mas[i,j]:=buf;
   next;mas[i,j]:=nextmas;last;last;
   if j=0 then quit:=false;
  end;
  i:=a;j:=c;
 end;
 print;
end.
```

Результаты работы:

```
0  0  8  2
 2  6  3  1
 3  4  1  4
 1  8  1  3

Нажмите Enter для продолжения !
```

```
 0  0  1  1
 1  1  2  2
 3  3  3  4
 4  6  8  8

Нажмите Enter для продолжения !
```

```
program sor2;
uses crt;
const n=4;
var mas:array[1..n,1..n]of integer;
    buf,l,units,i,j,nextmas:integer;
    quit:boolean;
procedure print;
begin
 for i:=1 to n do
  begin
  for j:=1 to n do
   write(' ',mas[i,j],' ');
  writeln;
  end;
 writeln;writeln('Ќ ¦¬ЁвЃ Enter ¤«п Їa®¤®«¦ЃЁп !');
 readln;
end;
procedure next;
begin
 j:=j+1;
 if j=n+1 then begin j:=1;i:=i+1;end;
end;
procedure last;
begin
 j:=j-1;
 if j=0 then begin j:=n;i:=i-1;end;
end;
begin
 clrscr;
 units:=n*n;
 for i:=1 to n do
  for j:=1 to n do
   mas[i,j]:=round(random*9);
 print;
 repeat
  i:=1;j:=1;
  quit:=true;
  units:=units-1;
  for l:=1 to units do
  begin
  next;nextmas:=mas[i,j];last;
   if mas[i,j]>nextmas
    then begin buf:=mas[i,j];mas[i,j]:=nextmas;next;mas[i,j]:=buf;quit:=false;end
```

```
    else next;
  end;
 until quit or (units=1);
 print;
end.
```

Результаты работы:

```
0  0  8  2
 2  6  3  1
 3  4  1  4
 1  8  1  3

Нажмите Enter для продолжения !

 0  0  1  1
 1  1  2  2
 3  3  3  4
 4  6  8  8

Нажмите Enter для продолжения !
```

```
program sor3;
uses crt;
const n=4;
var mas:array[1..n,1..n]of integer;
    min,buf,l,l1,i,j,i1,j1,a,c:integer;
procedure print;
begin
 for i:=1 to n do
  begin
  for j:=1 to n do
   write(' ',mas[i,j],' ');
  writeln;
  end;
 writeln;writeln('Ќ ¦¬ЁвЃ Enter ¤«п Їа®¤®«¦ЃЁп !');
 readln;
end;
begin
 clrscr;
 for i:=1 to n do
  for j:=1 to n do
   mas[i,j]:=round(random*9);
 print;
 a:=1;c:=1;i:=1;j:=1;
 for l:=1 to (n*n) do
 begin
  min:=mas[a,c];i:=a;j:=c;i1:=a;j1:=c;
  for l1:=1 to (n*n+1)-l do
  begin
   if min>mas[i1,j1] then begin min:=mas[i1,j1];i:=i1;j:=j1;end;
```

```
   j1:=j1+1;if j1=(n+1) then begin j1:=1;i1:=i1+1;end;
  end;
  buf:=mas[i,j];
  mas[i,j]:=mas[a,c];
  mas[a,c]:=buf;
  c:=c+1;if c=(n+1) then begin c:=1;a:=a+1;end;
 end;
 print;
end.
```

Результаты работы:

```
0  0  8  2
 2  6  3  1
 3  4  1  4
 1  8  1  3

Нажмите Enter для продолжения !

 0  0  1  1
 1  1  2  2
 3  3  3  4
 4  6  8  8

Нажмите Enter для продолжения !
```

```
program sor4;
uses crt;
const n=4;
type mas=array[1..n,1..n]of integer;
var mas1,mas2:mas;
    mas3:array[1..2*n,1..n]of integer;
    buf,i,j,i1,j1,a,c:integer;
procedure next(t:integer);
begin
 if t=1 then begin
             j:=j+1;if j=n+1 then begin j:=1;i:=i+1;end;
            end;
 if t=2 then begin
             j1:=j1+1;if j1=n+1 then begin j1:=1;i1:=i1+1;end;
            end;
 if t=3 then begin
             c:=c+1;if c=n+1 then begin c:=1;a:=a+1;end;
            end;
end;
procedure print(pr:mas);
begin
 for a:=1 to n do
  begin
  for c:=1 to n do
   write(' ',pr[a,c],' ');
```

```
  writeln;
  end;
 writeln;writeln('Ќ ¦¬ЁвЃ Enter ¤«п Їa®¤®«¦ЃЁп !');
 readln;
end;
begin
 clrscr;
 buf:=10;
 for a:=1 to n do
  for c:=1 to n do
  begin
   mas1[a,c]:=buf;
   buf:=buf+1;
   mas2[a,c]:=buf;
   buf:=buf+1;
  end;
 print(mas1);print(mas2);
 a:=1;c:=1;i:=1;j:=1;i1:=1;j1:=1;
 while (i<>n+1)and(i1<>n+1) do
  begin
   if mas1[i,j]<mas2[i1,j1] then begin mas3[a,c]:=mas1[i,j];next(1);end
                            else begin mas3[a,c]:=mas2[i1,j1];next(2);end;
   next(3);
  end;
 if i=n+1 then while not(i1=n+1)do
                begin
                 mas3[a,c]:=mas2[i1,j1];
                 next(3);next(2);
                end;
 if i1=n+1 then while not(i=n+1)do
                  begin
                   mas3[a,c]:=mas2[i,j];
                   next(3);next(1);
                  end;
 for a:=1 to n*2 do
  begin
  for c:=1 to n do
   write(' ',mas3[a,c],' ');
  writeln;
  end;
 writeln;writeln('Ќ ¦¬ЁвЃ Enter ¤«п Їa®¤®«¦ЃЁп !');
 readln;
end.
```

Результаты работы:

```
10  12  14  16
 18  20  22  24
 26  28  30  32
 34  36  38  40

Нажмите Enter для продолжения !
```

```
 11  13  15  17
 19  21  23  25
 27  29  31  33
 35  37  39  41

Нажмите Enter для продолжения !

 10  11  12  13
 14  15  16  17
 18  19  20  21
 22  23  24  25
 26  27  28  29
 30  31  32  33
 34  35  36  37
 38  39  40  41

Нажмите Enter для продолжения !
```

Лабораторная работа № 4.

Работа с массивами чисел.

Цель задания:

1. Получение практических навыков при работе с массивами.
2. Знакомство с алгоритмами упорядочения.

Постановка задачи:

1. Написать программу, которая для введенного числа N формирует двумерный массив заданного в варианте вида. В задании дан массив для N = 4.
2. Выполнить конкретное задание. Программа должна работать с массивами для любого N.

Содержание отчета:

1. Постановка задачи для конкретного варианта.
2. Текст программы.
3. Результаты тестов.
4. Распечатка результатов работы программы.

Образец выполнения задания.

Лабораторная работа № 4.

Работа с массивами чисел.

Постановка задачи:

Написать программу, которая для введенного числа N формирует двумерный массив заданного в варианте вида. В задании дан массив для N = 4.

Выполнить конкретное задание. Программа должна работать с массивами для любого N.

Вид массива: 

Полученный массив напечатать.

Найти количество чисел, входящих в массив больше одного раза. Результат напечатать.

Печать и задание оформить в виде процедур.

Текст программы:

```
program 2D_Array;
uses  crt;
label sle;
const n=4;
      par=(n*4);
type matriz=array[1..n,1..n] of integer;
var i,j,i1,j1,kol,kolo,l,k:integer;
    mas:matriz;
    im:array[1..par] of integer;
procedure vvod(var vmas:matriz);
        begin
         clrscr;
          writeln('Вводим элементы массива , размерность [',n,',',n,']');
          for i:=1 to n do
           for j:=1 to n do
            begin
             write('Введите элемент [',i,',',j,']=');
             readln(vmas[i,j]);
            end;
        end;
procedure print(vmas:matriz);
        begin
         writeln('Ваш массив имеет вид.');
            for i:=1 to n do
             begin
              for j:=1 to n do
              begin
              write(' ',vmas[i,j],' ');
              end;
              writeln;
             end;
        end;
begin
 vvod(mas);
 print(mas);
 for i:=1 to par do im[i]:=0;
 k:=1;
 kolo:=0;
 for i:=1 to n do
  for j:=1 to n do
   begin
```

```
    for l:=1 to par do if im[l]=mas[i,j] then goto sle;
    kol:=0;
    for i1:=1 to n do for j1:=1 to n do if mas[i,j]=mas[i1,j1] then kol:=kol+1;
    if kol>1 then
     begin
      kolo:=kolo+kol;
      im[k]:=mas[i,j];
      k:=k+1;
     end;
    sle:end;
  writeln('Количество чисел, входящих в массив больше одного раза(без нулей) =',kolo);
  writeln('Программа закончена , нажмите Enter.');
  readln;
end.
```

Результаты работы:

```
Ваш массив имеет вид.
 0  0  0  4
 0  0  3  4
 0  2  3  4
 1  2  3  4
Количество чисел, входящих в массив больше одного раза (без нулей) =9
Программа закончена, нажмите Enter.
```

Варианты заданий.

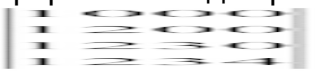
1) Вид массива: 

Полученный массив напечатать.

Найти суммы чисел в первом и последнем столбцах и сравнить их.

Вывести результаты на печать с соответствующим сообщением.

Задания и печать оформить в виде процедур (функций).

2) Вид массива: 

Полученный массив напечатать.

Сравнить сумму произведений элементов строк и произведение сумм элементов столбцов и напечатать с соответствующими сообщениями.

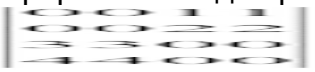
Печать и задания оформить в виде процедур (функций).

3) Вид массива: 

Полученный массив напечатать.

Найти максимальный элемент среди элементов, сумма индексов которых нечетна. Результат вывести на печать.

Задания и печать оформить в виде процедур или функций.

4) Вид массива: 

Полученный массив напечатать.

Упорядочить массив по убыванию элементов в каждом столбце. Результат напечатать.

Печать и задания оформить в виде процедур.

5) Вид массива: 

Полученный массив напечатать.

Упорядочить массив по возрастанию элементов в каждой строке.

Печать и задание оформить в виде процедур (функций).

6) Вид массива: 

Полученный массив напечатать.

Найти произведение индексов элементов в первой строке, а затем в последней, и сравнить их. Результаты напечатать с соответствующими заголовками.

Печать и задание оформить в виде процедур.

7) Вид массива: 

Полученный массив напечатать.

Найти количество чисел, входящих в массив только один раз. Результат напечатать.

Печать и задание оформить в виде процедур (функций).

8) Вид массива: 

Полученный массив напечатать.

Найти сумму максимального и минимального элементов. Полученное значение увеличить на 10. Результат напечатать.

Печать и задание оформить в виде процедур.

9) Вид массива: 

Полученный массив напечатать.

В каждой строке найти наибольший элемент, затем получить их сумму. Результат напечатать.

Печать и задание оформить в виде процедур.

10) Вид массива: 

Полученный массив напечатать.

Выяснить, есть ли в массиве хотя бы одна пара совпадающих по величине чисел. Результат напечатать.

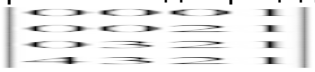
Печать и задание оформить в виде процедур.

11)Вид массива: 

Полученный массив напечатать.

В каждом столбце найти наименьший элемент, затем найти их произведение. Результат напечатать.

Печать и задание оформить в виде процедур (функций).

12) Вид массива: 

Полученный массив напечатать.

Переставить элементы массива так, чтобы все нули оказались в конце. Преобразованный массив напечатать.

Печать и задание оформить в виде процедур.

13) Вид массива: 

Полученный массив напечатать.

Найти произведение последних элементов строк. Результат напечатать.

Печать и задание оформить в виде процедур (функций).

14) Вид массива: 

Полученный массив напечатать.

Найти сумму элементов, расположенных ниже главной диагонали, и количество элементов, расположенных выше главной диагонали. Результаты напечатать.

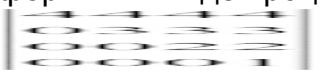
Печать и задание оформить в виде процедур (функций).

15) Вид массива: 

Полученный массив напечатать.

Перевернуть строку, содержащую максимальное число нулей, и столбец, содержащий минимальное число нулей. Преобразованный массив напечатать.

Печать и задание оформить в виде процедур.

16) Вид массива: 

Полученный массив напечатать.

Найти сумму элементов, расположенных на главной диагонали в строках, начинающихся с нуля. Результат напечатать.

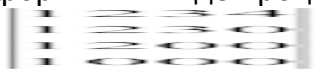
Печать и задание оформить в виде процедур.

17) Вид массива: 

Полученный массив напечатать.

Упорядочить элементы, находящиеся выше главной диагонали, по убыванию. Преобразованный массив напечатать.

Печать и задание оформить в виде процедур.

18) Вид массива: 

Полученный массив напечатать.

Перевернуть все четные строки массива. Преобразованный массив напечатать.

Печать и задание оформить в виде процедур.

19) Вид массива: 

Полученный массив напечатать.

Переставить элементы массива так, чтобы нули в каждой строке стояли в конце. Преобразованный массив напечатать.

Печать и задание оформить в виде процедур.

20) Вид массива: 

Полученный массив напечатать.

Перевернуть все нечетные столбцы массива. Преобразованный массив напечатать.

Печать и задание оформить в виде процедур.

21) Вид массива: 

Полученный массив напечатать.

Найти строку, содержащую минимальный элемент, и упорядочить ее по убыванию элементов. Результат напечатать.

Печать и задание оформить в виде процедур.

22) Вид массива: 

Полученный массив напечатать.

Найти третий по величине элемент массива. Результат напечатать.

Печать и задание оформить в виде процедур (функций).

23) Вид массива: 

Полученный массив напечатать.

Поставить последнюю строку массива на первое место, не меняя порядок остальных. Результат напечатать.

Печать и задание оформить в виде процедур.

24) Вид массива: 

Полученный массив напечатать.

Поставить последний столбец на первое место, не меняя порядок остальных. Результат напечатать.

Печать и задание оформить в виде процедур.

6. Обработка строк.

Для обработки текстовых данных используется строковой тип данных **String**. Строка – это последовательность символов из кодовой таблицы, заключённая в апострофы. Каждая строка характеризуется своей текущей длинною (количеством символов, имеющихся в строке в данный момент), порядком расположения символов. По умолчанию для строк определена максимальная длинна, равная 255 символам, но её можно изменить, указав при описании: **string [число символов]**. Если строка длиннее максимальной длинны, то не поместившиеся символы отбрасываются

Символьным константам можно присваивать имена.

Const st=’строка’;

Символьные переменные описываются в разделе описания переменных с описателем **String**.

```
Var  st1,st2:string[10];
          st3:string;
begin
  st1:=’строка’;
end.
```

К любому символу в строке можно обратится точно так же, как к элементу одномерного массива, т.е. указав имя строки и индекс символа в этой строке.

Например:**st[3]=’р’**. При этом для элемента строки разрешены те же операции и функции, что и для типа **char**.

Строки выводятся и вводятся с помощью операторов read, readln, write, writeln без организации циклов.

Над строками выполняется операция сцепления, которая позволяет соединить две или более строк в одну без разделителей.

Пример:
```
st1:=’Кро’;
st2:=’ил’;
st3:=st1+’код’+st2;
```
Результат:**st3=’Крокодил’;**

Над строками выполняются операции сравнения:

= , < , > , <= , >= , <>. Строки сравниваются посимвольно слева направо до первого результата или до исчерпания символов строки.

Например: ’азбука’=’азбука’, т.к. все символы поэлементно совпадают.

Например: ’школа’<’школьник’. Результат сравнения (true) , т.к.

‘ш’=’ш’, ’к’=’к’, ’о’=’о’, ’л’=’л’, ’а’<’ь’ (символ ‘а’ расположен в кодовой таблице раньше символа ‘ь’).

6.1. Функции обработки строк.

Сцепление – **concat(строка1, строка2,….)**, Аналогична операции сцепления.

Пример: Исходные данные: a=’код’, b=’ил’.

Оператор: **s:=concat(‘кро’,a,b)**.
Результат: s=’крокодил’.

Копировать – **copy(строка, число1, число2)**. Из указанной строки выделяется подстрока, начиная с позиции, заданной числом1, длиной, заданной числом2.
Пример: Исходные данные: s=’крокодил’.
Оператор: **b:=copy(s,2,3)**.
Результат: b=’рок’.

Позиция - **Pos(строка1, строка2**). Отыскивает первое вхождение строки1 в строке2 и возвращает номер начальной позиции вхождения или ноль, если строка1 не входит в строку2.
Пример: Исходные данные: s=’крокодил’.
Оператор: **i:=pos(‘око’,s)**.
Результат: i=3.
Оператор: **i:=pos(‘я’,’крокодил’)**.
Результат: i=0.

Длина - **length(строка)**. Возвращает длину строки – аргумента.
Пример: Исходные данные: s=’крокодил’.
Оператор: **j:=length(s)**.
Результат: j=8.

6.2. Процедуры обработки строк.

Вставить – **insert(строка1, строка2, число)**. Вставляет строку1 в строку2, начиная с позиции, заданной числом. Если в результате получается строка длины больше максимальной, то она усекается справа.
Пример: Исходные данные: S=’крокодил’.
Оператор: **d:=copy(s,3,3).**
Результат: d=’око’.
Оператор: **insert(‘h’,d,3).**
Результат: d=’окно’.

Удалить – **delete(строка, число1, число2)**. Удаляет из строки подстроку, начиная с позиции, заданной числом1, длиною, заданной числом2. Если число1 больше размера строки, то подстрока не удаляется. Если число2 больше имевшегося количества, то удаляются символы до конца строки.
Пример: Исходные данные: S=’крокодил’.
Оператор: **delete(s,4,3).**
Результат: =’кроил’.
Оператор: **delete(s,1,1).**
Результат: s=’роил’.

Преобразовать число в строку – **str(число[:M[:N]],строка)**. Преобразует число в строку. M задаёт общее количество символов, получаемых в строке, N – для вещественных чисел (типа real) задаёт количество цифр в дробной части.
Пример:
Оператор: **str(123,s).**
Результат: s=’123’.

Преобразовать строку в число – **val(строка, число, код).** Преобразует строку символов во внутреннее представления числа. Код указывает номер неправильного символа или равен 0 в случае успешного преобразования.

Пример:

Оператор: **val('+12.3',v,k).**

Результат: v=12.3, k=0 {преобразование прошло успешно}

Оператор: **val('23+5',v,k).**

Результат: v=неправильно, k=3 {ошибка при попытке преобразовать третий символ}

Лабораторная работа № 7.

Обработка строк.

Цель задания:

1. Получение практических навыков в работе со строками.
2. Знакомство с задачами, для решения которых используются строковые типы данных и, функции и процедуры по их обработке.

Постановка задачи:

1. Для решения конкретного варианта составить программу.
2. Вывести на печать результат выполнения программы.

Содержание отчета:

1. Постановка задачи для конкретного варианта.
2. Текст программы.
3. Результаты выполнения программы.

Образец выполнения задания.

Лабораторная работа № 7, вариант № 8.

Обработка строк.

Постановка задачи для конкретного варианта:

Задана строка, состоящая из слов, разделённых одним или несколькими пробелами. Удалить повторные вхождения каждого слова.

Выделяем слова, переписываем их в первую строку двумерного массива, во вторую записываем '0' для уникального слова и '1'- для повторяющегося слова. Затем формируем строку, состоящую из элементов первой строки массива, у которых во второй строке записано '0' и распечатываем строку.

Текст программы:

```
program lab7{вариант № 8};
const nn=10;
```

```
type mas=array [1..2,1..nn] of string;
var a:mas;
     n:integer;
     s,ss:string;                          {исходная и вспомогательная строки}
     i, j, k:integer;
begin
 write('Введите строку : ');readln(s);
 s:=s+' ';
 j:=0; ss:=' ';
 for i:=1 to length(s)-1 do
  if (s[i]<>' ')and(s[i+1]=' ')            {выделение слов}
  then begin
           ss:=ss+s[i];
            j:=j+1;a[1, j]:=ss;a[2, j]:='0'; {в первую строку}
           ss:=' ';                         {записываем слово}
         end;                               {'0' во второй строке означает, что слово
встретилось впервые}
  else if s[i]<>' ' then ss:=ss+s[i];
 for i:=1 to j-1 do
   for k:=i+1 to j do
     if (a[2,i]<>'1')and(a[2,k]<>'1')and(a[1,i]=a[1,k]) then a[2,k]='1';  {нашли совпавшие
слова}
 s:=' ';
 for i:=1 to j do
   if a[2,i]<>'1' then s:=s+a[1,i]+' ';
 writeln('Результат : ',s);
end.
```

Результаты выполнения программы:

```
Введите строку: жили были жили
Результат:  жили  были
```

Варианты заданий.

1. Задано предложение, состоящее из слов, разделённым одним или несколькими пробелами. Упорядочить слова предложения в алфавитном порядке.

2. Задано предложение, состоящее из слов, разделённым одним или несколькими пробелами. Найти самое длинное слово в предложении.

3. Задано предложение, состоящее из слов, разделённым одним или несколькими пробелами. Подсчитать количество гласных русских букв в предложении.

4. Задано предложение, состоящее из слов, разделённым одним или несколькими пробелами. Вывести на экран все слова, преобразовав каждое при этом следующим образом: первую букву слова заменить последней.

5. Задано предложение, состоящее из слов, разделённым одним или несколькими пробелами. Выяснить какая буква встречается чаще всего.

7. Комбинированные типы. Оператор присоединения

7.1. Записи

Комбинированный тип характеризует объекты, называемые записями. Запись – это сложная переменная с несколькими компонентами. В отличие от массивов компоненты записи (поля) могут иметь разные типы и доступ к ним осуществляется не по индексу, а по имени поля. При определении комбинированного типа задаются имя и тип каждого поля.

Тип имеет следующую структуру:

type <имя_типа> = RECORD <сп_полей> END;

где RECORD, END – зарезервированные слова (запись, конец);

<имя_типа> -- правильный идентификатор;

<сп_полей> -- список полей; представляет собой последовательность разделов записи, между которыми ставится точка с запятой.

К каждому компоненту записи можно обратиться, используя имя переменной типа записи и имя поля, разделенных точкой.

Пример:
```
type     СотрФирм = RECORD
                      ФИО: array [1..10] of char;
                      Оклад: integer;
                      Адрес: RECORD
                                Улица : array [1..10] of char;
                                Ндома,Нкв : integer;
                             END;
                    END;
var сотф : СотрФирм;
           или
var сотф :RECORD
            ФИО: array [1..10] of char;
           Оклад: integer;
            Адрес: RECORD
                      Улица : array [1..10] of char;
                       Ндома,Нкв : integer;
                   END;
         END;

Обращение :   сотф.Оклад:=1344;
              сотф.Адрес.Ндома:=12;
             сотф.Адрес.Нкв:=34;
```

7.2. Оператор присоединения

Приведенные операторы присваивания при обращении к записям можно записать компактно, если использовать оператор присоединения, имеющий вид:

WITH <список перем_записей, полей> DO <оператор>.

Имена переменных-записей и полей, указанные в заголовке оператора присоединения, можно опускать при обращении к компонентам записей в области действия оператора WITH:

Пример: Обращение: with сотф do Оклад:=1344;

или

```
with сотф  do begin
                Оклад:=1344;
                Адрес.Ндом:=12;
                Адрес.Нкв:=34;
              end;
```

При определении того или иного комбинированного типа имена отдельных полей могут совпадать с именами переменных. Путаницы при использовании этих переменных и соответствующих полей записи не происходит в силу того, что в частичной переменной-записи указывается и имя собственной переменной-записи. Однако при использовании оператора присоединения может возникнуть недоразумение, связанное с тем, что внутри него имена переменных-записей опускаются. Возникает вопрос: что обозначает имя внутри оператора присоединения, если и у соответствующей переменной-записи присутствует поле с таким именем, и в разделе переменных введена в употребление переменная с таким же именем? В языке Паскаль этот конфликт решается так: предпочтение отдаётся именам полей записи, т.е. считается, что внутри оператора присоединения соответствующий идентификатор обозначает имя поля, а не имя переменной.

Пусть, например, в разделах описания типов и описания переменных введены в употребление следующие комбинированные типы и переменные:

```
Type студ=record
               Фам,Имя,Отч:array [1..16] of char;
               Пол: (муж,жен);
               Группа:101..520;
               Стип:boolean;
             End;
     сотр=record
               Фам,Имя,Отч:array [1..16] of char;
               Пол: муж..жен;
               Должность:(мнс,нс,снс,асс,доц,проф);
               Зарплата:integer;
             End;
  Var  X:Студ;
       Y:Сотр;
```

Тогда в следующем фрагменте программы, использующем оператор присоединения:

```
With X,Y do begin
              Пол:=муж;
              Имя:='Александр';
              Стипендия:=true;
              Группа:=108;
            End;
```

Поля Пол и Имя относятся к переменной Y типа Сотр, так как эта переменная в списке переменных-записей заголовка оператора присоединения фигурирует после переменной X типа Студ, имеющей одноимённые поля Пол и Имя. Кроме того, в этом фрагменте имя Степендия в теле оператора присоединения трактуется как имя поля переменной X, а вне его как имя переменной целого типа.

Приведём ещё один пример, иллюстрирующий трактовку оператора присоединения. Пусть имеются описания переменных:

```
Var   R1:record  A,B,C:integer; end;
      R2:record     A,D:integer;
                      B:record C,E:integer;end;
                    end;
```

Тогда оператор присоединения

```
With R1,B,R2  do
Begin  A:=1; B:=2; C:=3; D:=4; E:=5; end;
```

эквивалентен составному оператору

```
begin  R1.A:=1; R1.B:=2; R1.C:=3; R2.D:=4; R2.B.E:=5; end;
```

Рекомендуется внимательно проанализировать каждый оператор присваивания и чётко понять, почему именно такие частичные переменные фигурируют в составном операторе, эквивалентном оператору присоединения.

Лабораторная работа № 8.

Работа с комбинированными типами данных.

Цель задания:

1. Получение навыков в организации ввода/вывода значений комбинированного типа данных.
2. Получение практических навыков программирования задач с использованием записей.

Постановка задачи:

1. Существует некоторая фирма, которая образована в 1991 году. Составить список сотрудников этой фирмы, содержащей 20 человек. Список должен содержать следующие сведения для каждого сотрудника: ФИО, дату рождения, год поступления в фирму, оклад, адрес (улица, номер дома и квартиры). Информацию о каждом сотруднике оформить в виде записи. Записи объединить в массив.
2. Составить программу, которая выполняет ввод и печать информации для конкретного варианта. Ввод и печать оформить в виде процедур.

Содержание отчета:

4. Постановка задачи для конкретного варианта.
5. Исходные данные.
6. Текст программы.
7. Результаты выполнения программы.

Образец выполнения задания.

Лабораторная работа № 8, вариант № 8.

Работа с комбинированными типами данных.

Постановка задачи для конкретного варианта:

1. Существует некоторая фирма, которая образована в 1991 году. Составить список сотрудников этой фирмы, содержащей 20 человек. Список должен содержать следующие сведения для каждого сотрудника: ФИО, дату рождения, год поступления в фирму, оклад, адрес (улица, номер дома и квартиры). Информацию о каждом сотруднике оформить в виде записи. Записи объединить в массив.
2. Составить программу, которая выполняет ввод и печать списка сотрудников, фамилии которых начинаются с буквы Т, и их даты рождения. Ввод и печать оформить в виде процедур.

Исходные данные:

Анисимов Петр Иванович Родился 23.1.1960 В фирме с 1991 года Зарплата 15000 рублей Проживает по адресу: Улица Ленина 12-45	Синилов Сергей Анатольевич Родился 14.5.1964 В фирме с1991 года Зарплата 14500 рублей Проживает по адресу: Улица Мира 67-19	Шорапов Евгений Владимирович Родился 28.2.1969 В фирме с 1991 года Зрплата 14000 Проживает по адресу: Улица Левченко 84-37
Бажин Никита Андреевич Родился 3.9.1963 В фирме с 1991года Зарплата 13500 рублей Проживает по адресу: Улица Вагонная 94-36	Созинов Алексей Петрович Родился 13.12.1964 В фирме с 1991года Зарплата 13000 рублей Проживает по адресу: Улица Куйбешева 68-83	Малышев Василий Владимирович Родился 18.6.1968 В фирме с 1991года Зарплата 12500 рублей Проживает по адресу: Улица Охотников 8-3
Мельникова Лариса Анатольевна Родилась 11.2.1959 В фирме с 1991 года Зарплата 12000 рублей Проживает по адресу: Улица Кирова 83-56	Тихонов Сергей Генадьевич Родился 30.3.1967 В фирме с 1991года Зарплата 11500 рублей Проживает по адресу: Улица Автозаводская 42-88	Еговцев Иван Артурович Родился 18.9.1968 В фирме с 1991года Зарплата 11000 рублей Проживает по адресу: Улица Дзержинского 23-69
Ползунова Елена Андреевна Родилась 15.10.1962 В фирме с 1991 года Зарплата 10500 рублей Проживает по адресу: Улица Дружбы 28-75	Михайлов Артем Егоровну Родился 2.11.1970 В фирме с 1992 года Зарплата 10000 рублей Проживает по адресу: Улица Невская 13-46	Смирнов Никита Владимирович Родился 3.8.1968 В фирме с 1992 года Зарплата 9500 рублей Проживает по адресу: Улица Болотная 59-38
Токарев Надежда Александровна Родилась 4.7.1970 В фирме с 1992 года Зарплата 9000 рублей Проживает по адресу: Улица Кочегаров 75-63	Маслова Нина Михайловна Родилась 7.3.1966 В фирме с 1993 года Зарплата 8500 рублей Проживает по адресу: Улица Васнецова 49-92	Молчановский Ильнар Ирекович Родился 9.8.1969 В фирме с 1993 года Зарплата 8000 рублей Проживает по адресу: Улица Лебедева 34-81

Корягина Нина Плахова Родилась 10.2.1970 В фирме с 1994 года Зарплата 7500 рублей Проживает по адресу: Улица Калинина 24-12	Егорова Пелагея Луповна Родилась 12.1.1971; В фирме с 1996 года Зарплата 7000 рублей Проживает по адресу: Улица Нефтяников 47-38	Гаспер Валентина Александровна Родилась 16.11.1972 В фирме с 1998 года Зарплата 6500 рублей Проживает по адресу: Улица Докучаево 75-94
Теплоухов Юрий Леонидович Родился 25.5.1978 В фирме с 2000 года Зарплата 6000 рублей Проживает по адресу: Улица Заречная 28-47	Кирьянов Антон Алексеевич Родился 28.8.1968 В фирме с 1993 года Зарплата 12700 рублеу Проживает по адресу: Улица Кислотная 26-14	

Текст программы:

```
program lab8{ вариант № 8};
type man=record
        fio:record fameli,name,och:string[15];end;
        date:record day,mes,god:integer;end;
        godpos:integer;
        many:integer;
        adres:record ul:string[15];
                dom,kv:integer;end;
        end;
var sot:array [1..20] of man;
    n:integer;
    symb:string[1];
procedure vvod;
begin
  for n:=1 to 20 do
  begin
    writeln('Вводим данные на сотрудника номер:',n);
    write('Фамилия');
    readln(sot[n].fio.fameli);
    write('Имя');
    readln(sot[n].fio.name);
    write('Отчество');
    readln(sot[n].fio.och);
    writeln('Дата рождения');
    write('День');
    readln(sot[n].date.day);
    write('Месяц');
    readln(sot[n].date.mes);
    write('Год');
    readln(sot[n].date.god);
    write('Год поступления в фирму');
    readln(sot[n].godpos);
```

```
    write('Заработная плата');
    readln(sot[n].many);
    writeln('Адрес проживания');
    write('Улица');
    readln(sot[n].adres.ul);
    write('Номер дома');
    readln(sot[n].adres.dom);
    write('Номер квартиры');
    readln(sot[n].adres.kv);
  end;
end;
procedure list(n:integer);
        begin
            writeln('-------------------------------------------------------------------------------');
            write(sot[n].fio.fameli,' ',sot[n].fio.name,' ',sot[n].fio.och);
             writeln('  Дата рождения ',sot[n].date.day,'/',sot[n].date.mes,'/',sot[n].date.god);
            writeln;
        end;
begin
vvod; {процедура ввода исходных данных}
writeln('Распечатать список сотрудников, фамилии которых начинаются с буквы T');
writeln(' и их даты рождения.');
writeln;
for n:=1 to 20 do begin
                    symb:=copy(sot[n].fio.fameli,1,1);
                    if symb='T' then list(n);
                  end;
end.
```

Результаты выполнения программы:

```
Распечатать список сотрудников, фамилии которых начинаются с буквы T
 и их даты рождения.
-------------------------------------------------------------------------------
Тихонов Сергей Геннадьевич     Дата рождения 30/3/1967
-------------------------------------------------------------------------------
Токарева Надежда Александровна     Дата рождения 4/7/1970
-------------------------------------------------------------------------------
Теплоухов Юрий Леонидович     Дата рождения 25/5/1978
-------------------------------------------------------------------------------
Конец данных , нажмите Enter.
```

Варианты заданий.

1. Распечатать анкетные данные сотрудников, имеющих срок службы больше 5 лет.
2. Распечатать анкетные данные сотрудников, которым больше 25 лет.
3. Распечатать анкетные данные сотрудников, у которых улица начинается с буквы С.
4. Распечатать ФИО сотрудников, у которых улица начинается с буквы М, и номер дома больше 5.

5. Распечатать анкетные данные сотрудников, месяц рождения которых больше 6 и фамилия начинается с буквы Л.
6. Распечатать анкетные данные сотрудников, упорядоченные по ФИО и Улицам.
7. Распечатать список сотрудников, фамилии которых начинаются с буквы А, и их оклад.
8. Распечатать список сотрудников, фамилии которых начинаются с буквы В и Г, и год их поступления в фирму.
9. Распечатать фамилии и даты рождения сотрудников, имеющих номер квартиры меньше 35.
10. Распечатать список сотрудников, упорядоченный по адресам (т.е. по улице, номерам дома и квартиры).
11. Упорядочить список сотрудников по году поступления в фирму и распечатать его.
12. Вычислить средний оклад сотрудников и распечатать список сотрудников, имеющих оклад выше среднего.
13. Вычислить средний оклад сотрудников и распечатать список сотрудников, имеющих оклад ниже среднего.
14. Вычислить средний оклад сотрудников и распечатать список сотрудников, имеющих оклад, равный среднему.
15. Упорядочить список сотрудников фирмы по году рождения и распечатать его.
16. Распечатать список сотрудников, упорядоченный по алфавиту.
17. Распечатать список сотрудников, упорядоченный по месяцу рождения.
18. Распечатать список сотрудников, фамилии которых начинаются с буквы Р, упорядоченный по году рождения.
19. Распечатать анкетные данные сотрудников, упорядоченные по дате рождения.
20. Распечатать ФИО и адреса сотрудников, которые проработали в фирме меньше года.
21. Распечатать ФИО сотрудников, оклад которых выше среднего в 1.2-1.5 раза.
22. Распечатать ФИО сотрудников, в названии улиц которых есть буква А.
23. Распечатать ФИО и номера домов сотрудников, у которых номера квартир начинаются с 3, 4, 5, 6, 7, 8.
24. 25) Распечатать ФИО и дату рождения сотрудников, у которых номера домов меньше 20, а номера квартир больше 5.

8. Множественные типы данных.

8.1. Множества.

Множества - это наборы однотипных логически связанных друг с другом объектов.

Количество элементов, входящих в множество, может меняться в пределах от 0 до 256 (множество, не содержащие элементов, называется пустым). Именно непостоянством количества своих элементов множества отличаются от массивов и записей.

Два множества являются эквивалентными тогда и только тогда, когда все элементы одинаковы причем порядок следования элементов в множестве безразличен.

Описание типа множества имеет вид:

<имя типа>=SET OF <базовый тип>,

где <имя типа> - правильный идентификатор;

SET,OF - зарезервированные слова;

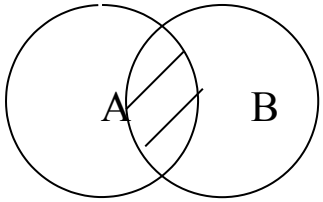
<базовый тип> - базовый тип элементов множества, в качестве которого может использоваться любой порядковый тип, кроме WORD, LONGINT, INTEGER.

Для задания множества используется так называемый конструктор множества: список спецификаций элементов множества, отделяемых друг от друга запятыми, список обрамляется квадратными скобками.

Операции над множествами:

- Пересечение множеств. Результатом операции пересечения двух множеств A*B будет множество C, состоящее только из тех элементов которые принадлежат, как множеству A, так и множеству B.

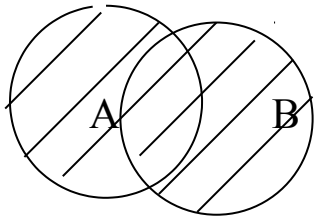
Пример: [1,2,3,4]*[3,4,5,6] результат [3,4]



C = A * B
C: = A * B

+ - Объединение множеств. Результатом операции объединения множеств A+B будет множество C , включающее как все элементы множества A, так и все элементы множества B.

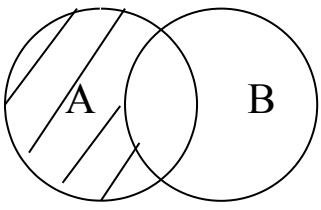
Пример: [1,2,3,4]+[3,4,5,6] результат [1,2,3,4,5,6]



C = A ∪ B
C: = A + B

- Разность множеств. Результатом операции разности двух множеств A-B, будет множество C, состоящее только из тех элементов множества A, которые не входят в множество B.

Пример: [1,2,3,4]-[3,4,5,6] результат [1,2]



C = A \ B
C: = A - B

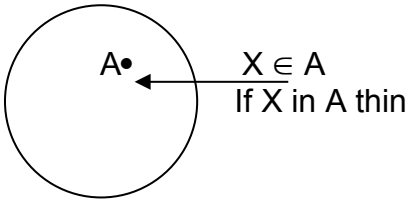
Результатом операции сравнения A=B будет TRUE, а операции

A< >B будет FALSE, только тогда, когда A и B содержат одни и те же элементы.

Результатом операции сравнения A<=B будет TRUE, если множество A является подмножеством множества B.

Результатом операции сравнения A>=B будет TRUE, если множество A включает в себя все элементы множества B.

Результатом операции принадлежности X in A будет TRUE, если значение X какого либо порядкового типа T является элементом множества A того же типа T.



Пример операций принадлежности и объединения множеств.

```
PROGRAM   Dem_Set_Type;
USES  Crt;
TYPE  SetType =set of char;
{объявление отдельного типа для передачи параметров в My function}
VAR    Myset, Set1, Set2  :Settype;
         CH :char;
FUNCTION  Myfunction (Var set:  Set type) : char;


VAR  Flag: char;
BEGIN
      REPEAT
    Flag:=UpCase(ReadKey);
    UNTIL  Flag IN Var Set;
Writeln(‘ Правильно!!!’);
    Myfunction := Flag;
END;
BEGIN
    MySet:= [‘Y’,’N’];
    Writeln(‘Допускаются ответы только Y или N’);
    CH:= Myfunction (myset);
    Set1:=[‘K’];
    Myset := Set1 + Myset; {Слияние двух множеств}
    Writeln(‘Допускается также К’);
    CH :=Myfunction (Myset);
    REPEAT
    UNTIL  KeyPressed;  {Ожидается нажатие клавиши для
                                    Возврата в ИИО}
END.
```

Лабораторная работа № 9.

Работа с множественными типами данных.

Цель задания:

Получение навыков в задании переменных множественного типа и выполнении простейших операций над ними.

Знакомство с задачами, в которых целесообразно использовать переменные множественных типов.

Постановка задачи:

1. Ознакомиться с конечным и упорядоченным множеством символов.
2. Составить программу для одного из вариантов.

Содержание отчета:

1. Постановка задачи.
2. Текст программы.
3. Выводы.

Образец выполнения работы.

Лабораторная работа № 9, вариант № 3.

Работа с множественными типами данных.

Постановка задачи:

Ознакомиться с конечным и упорядоченным множеством символов.
Составить программу для одного из вариантов.

Методические указания:

Программа должна правильно работать для произвольного набора символов.

Вариант задания:

Дана непустая последовательность символов. Требуется построить и напечатать множество, элементами которого являются встречающиеся в последовательности:
19. знаки ‘%’ ,’!’ ,’?’ ,’$’, ’#’ и цифры от ‘1’ до ‘5’.

Текст программы:

```
Uses crt;
const
      Length = 255;
var
      m1,m2 : array [1..Length] of Char;
  i,a :  Integer;
Begin
      ClrScr;
      Randomize;
  For i:=1 to Length do
      m1[i]:=Chr(Random(255));
  a:=1;
  For i:=1 to Length do
```

```
      Begin
          Case m1[i] of
           '%': Begin m2[a] := m1[i] Inc(a); End;
           '!': Begin m2[a] := m1[i] Inc(a); End;
           '?': Begin m2[a] := m1[i] Inc(a); End;
           '$': Begin m2[a] := m1[i] Inc(a); End;
           '#': Begin m2[a] := m1[i] Inc(a); End;
           '1': Begin m2[a] := m1[i] Inc(a); End;
           '2': Begin m2[a] := m1[i] Inc(a); End;
           '3': Begin m2[a] := m1[i] Inc(a); End;
           '4': Begin m2[a] := m1[i] Inc(a); End;
  end;

      End;
      For i:=1 to Length do
        Write(m2[i],' ');
    ReadLn;
  End.
```

Результаты работы:

```
% 2 $ ! 5 5 5 5
```

Методические указания:

Программа должна правильно работать для произвольного набора символов.

Варианты заданий.

Дана непустая последовательность символов. Требуется построить и напечатать множество, элементами которого являются встречающиеся в последовательности:

1. цифры от ‘0’ до ‘9’.
2. буквы от ‘A’ до ‘F’ и от ‘X’ до ‘Z’.
3. буквы от ‘G’ до ‘N’ и цифры от ‘0’ до ‘9’.
4. знаки препинания.
5. буквы от ‘A’ до ‘Z’ и цифры от ‘0’ до ‘5’.
6. буквы от ‘T’ до ‘X’ и знаки препинания.
7. цифры от ‘5’ до ‘9’ и знаки арифметических операций.
8. знаки арифметических операций и знаки препинания.
9. цифры и знаки арифметических операций.
10. знаки препинания и буквы от ‘E’ до ‘N’.
11. знаки операций отношений.
12. цифры от ‘3’ до ‘9’, буквы от ‘A’ до ‘F’ и знаки препинания.
13. знаки арифметических операций и операций отношения.
14. буквы от ‘F’ до ‘M’ и знаки арифметических операций.
15. знаки препинания и операций отношения.
16. цифры от ‘6’ до ‘9’ и знаки операций отношения.
17. знаки арифметических операций и цифры от ‘2’ до ‘8’.

18. знаки ‘%’ ,’!’ ,’?’ ,’$’, ’#’, ’@’, ’&’ ,’*’.
19. цифры от ‘3’ до ‘7’ и знаки препинания.
20. знаки операций отношения и буквы от ‘A’ до ‘F’.
21. цифры от ‘4’ до ‘9’ , буквы от ‘G’ до ‘M’ и знаки ‘%’ ,’!’ ,’?’.
22. цифры от ‘4’ до ‘9’ и операции отношения.
23. цифры от ‘0’ до ‘8’ и знаки ‘&’,’#’,’@’.
24. знаки арифметических операций, цифры ‘2’и ‘5’, буквы ‘C’ до ‘H’.

Лабораторная работа № 10.

Операции над множествами.

Цель задания:

1. Получение навыков в организации ввода/вывода значений множественных типов.
2. Получение практических навыков в выполнении операций над множествами.

Постановка задачи:

Задан список объектов, включающий в зависимости от варианта названия ЭВМ или видов спорта. Известно, что в каждом институте имеется определенный набор вычислительных машин, а учащиеся каждой группы занимаются определенными видами спорта. Необходимо задать конкретные наборы ЭВМ (перечни видов спорта) для каждого института (каждой группы). Количество институтов(групп) указано в варианте.

Введя исходные данные, необходимо построить и распечатать множество, удовлетворяющее указанному в варианте условию.

Содержание отчета:

1. Постановка задачи для конкретного варианта.
2. Инструкция пользования программой.
3. Текст программы и результаты ее выполнения.
4. Выводы.

Образец выполнения работы.

Лабораторная работа № 10.

Операции над множествами.

Постановка задачи:

Задан список объектов, включающий в зависимости от варианта названия ЭВМ или видов спорта. Известно, что в каждом институте имеется определенный набор вычислительных машин, а учащиеся каждой группы занимаются определенными видами спорта. Необходимо задать конкретные наборы ЭВМ (перечни видов спорта) для каждого института (каждой группы). Количество институтов(групп) указано в варианте.

Введя исходные данные, необходимо построить и распечатать множество, удовлетворяющее указанному в варианте условию.

Варианты задания:

требуется построить и распечатать три множества : первое множество должно включать в себя ЭВМ, , имеющиеся во всех институтах; второе - ЭВМ, имеющиеся хотя бы в одном институте; третье - ЭВМ, которых нет ни в одном ин ституте(N=4).

Текст программы:

```
Program Sets;
Uses Crt;
Type
  Comps = (i386, i486, Apple, Pentium, Acer, Macintosh);
  TComps = set of Comps;
Const
  All_comps : TComps = [i386, i486, Apple, Pentium, Acer, Macintosh];
  Inst_1 : TComps = [i386,Acer, Pentium];
  Inst_2 : TComps = [macintosh, Pentium];
  Inst_3 : TComps = [Apple, Pentium ];
  Inst_4 : TComps = [Pentium, Acer, i486];
Var
   InAll, NoOne, InOne, All_Comps_In, NotInst_1,
          NotInst_2, NotInst_3, NotInst_4 : TComps;
  Flag : String;
Procedure OutPut(s : TComps);
Begin
  If i386 in s then Write('i386 ');
  If i486 in s then Write('i486 ');
  If Pentium in s then Write('Pentium ');
  If Apple in s then Write('Apple ');
  If Acer in s then Write('Acer ');
  If Macintosh in s then Write('Macintosh ');
End;
Begin
   ClrScr;
  All_Comps_In := Inst_1 + Inst_2 + Inst_3 + Inst_4;
  NoOne := All_Comps - All_Comps_In;
   Write('Comps not met in all VUZ:  ');
  OutPut(NoOne); WriteLn;
  Write('Comps met in only one VUZ: ');          OutPut(All_Comps_In-Inst_1-Inst_2-Inst_3);
  OutPut(All_Comps_In-Inst_2-Inst_3-Inst_4);
  OutPut(All_Comps_In-Inst_3-Inst_4-Inst_1);
  OutPut(All_Comps_In-Inst_2-Inst_4-Inst_1);
WriteLn;
  Write('Comps met in every VUZ: ');
  NotInst_1 := All_Comps_In-Inst_1;
  NotInst_2 := All_Comps_In-Inst_2;
  NotInst_3 := All_Comps_In-Inst_3;
NotInst_4 := All_Comps_In-Inst_4;
  OutPut(All_Comps_In-(NotInst_1 + NotInst_2 + NotInst_3 + NotInst_4));
  While not KeyPressed Do;
```

End.

Результаты программы:

```
Comps not met in all VUZ:
Comps met in only one VUZ: i486 i386 Macintosh Apple
Comps met in every VUZ: Pentium
```

Варианты заданий.

Задано множество вычислительных машин, которыми может быть обеспечен институт: IBM-386, IBM-486, Pentium, Macintosh, APPLE, ACER. Известен набор машин, имеющихся в каждом институте. Количество институтов (N) указано в варианте:

1) требуется построить и распечатать множество, включающее в себя вычислительные машины:

- которыми обеспечены все институты (N=10).
- которые имеют хотя бы один институт.
- которых нет ни водном институте.

2) требуется построить и распечатать два множества:

- первое множество должно включать в себя ЭВМ, имеющиеся во всех институтах второе - ЭВМ, имеющиеся хотя бы в одном институте(N=5).

- первое множество должно включать в себя ЭВМ, имеющиеся в одном институте; второе - ЭВМ, которых нет ни в одном институте(N=5).

- первое множество должно включать в себя ЭВМ, которых нет ни в одном институте; второе - ЭВМ, имеющиеся во всех институтах(N=5).

3) требуется построить и распечатать три множества :

- первое множество должно включать в себя ЭВМ, , имеющиеся во всех институтах;
- второе - ЭВМ, имеющиеся хотя бы в одном институте;
- третье - ЭВМ, которых нет ни в одном институте(N=4).