§1. Начало программирования на языке Паскаль.

Программа - алгоритм, записанный на языке программирования, служащий для выполнения каких-либо действий.

Транслятор.

Допустим, что написана программа решения задачи на каком-то языке программирования, но компьютер понимает только команды, переведенные в двоичный код. Этот перевод можно осуществить с помощью *трансляторов*.

Транслятор - *программа, переводящая текст алгоритма, записанного на каком-то языке программирования, на язык машинных кодов.*

Различают два вида трансляторов: *компиляторы* и *интерпретаторы.* Они различаются с точки зрения выполнения работы.

Компилятор читает всю программу целиком, делает ее перевод и создает законченный вариант программы на машинном языке, который затем загружается в компьютер и выполняется.

Интерпретатор переводит и выполняет программу строка за строкой (как синхронный переводчик).

Компилятор осуществляет синтаксический контроль программы и при обнаружении ошибок выдает диагностические сообщения. Если ошибок нет, результатом компиляции является программа на языке машинных команд (ее называют объектной).

Краткое знакомство.

Язык Паскаль появился в 1984 году. Он разработан Н.Виртом в 1968-1970 г.г., получил широкое распространение благодаря наглядности программ и легкости при изучении.

Turbo Pascal 7.0 позволяет :

а) создавать тексты программ;
б) компилировать их(находить и исправлять ошибки);
г) компоновать программы из отдельных частей;
д) использовать модули библиотек;
е) отлаживать и выполнять программы.

Система ТР состоит из множества файлов, основные из которых :

* TURBO.EXE, который содержит готовую к работе диалоговую систему программирования (в нее входят текстовый редактор, компилятор, компоновщик, загрузчик);
* TURBO.TPL - основная библиотека ТР;
* TURBO.HLP - справочная служба.

Для загрузки программы *Turbo Pascal 7.0* надо :

1. Зайти в каталог ТР7 на диске.
2. Найти и запустить файл **turbo.exe**

После загрузки системы экран разделен на три части :

1. Главное меню.
2. Основное, или рабочее, окно.
3. Строка, в которой указывается назначение основных функциональных клавиш.

Переход из основного окна в главное меню осуществляется посредством клавиши **[F10]**

Функциональные клавиши Турбо Паскаль.

F1 - помощь;
F2 - сохранение редактируемого текста на диске;
F3 - загрузка текста с диска в окно редактирования;
F9 - компилировать программу, но не выполнять ее;
F10 - переход к верхнему меню;
CTRL-F9 - выполнить прогон программы (компилировать ее и выполнить);
ALT-F5 - просмотр результатов выполнения программы;
Esc - переход из главного меню в окно редактирования.
ALT-X - выход из системы Турбо Паскаль.

§2. Алфавит языка Паскаль. Переменные. Типы переменных

Алфавит языка Паскаль.

Алфавит языка - *это конечный набор символов, которые используются при написании любой конструкции на языке.*

Алфавит языка Паскаль можно разбить на 3 группы :

1. буквы строчные и прописные буквы латинского и русского алфавита;

2. цифры - арабские 0..9

3. специальные символы - (+,-,*,/,=,<,>,<>,<=,>=,.:;{}' := ( )), служебные слова : and, begin,const,div,do,else,for, и т.д.
и указывается тип переменных списка.

Переменная - это область памяти, названная собственным именем, которая может менять свое значение в процессе выполнения программы. Переменная характеризуется именем, типом и значением.

Константа - не изменяет своего значения в процессе выполнения программы, она может быть задана явно своим значением или обозначена именем.

Имя (идентификатор) переменной или константы задается латинскими буквами и арабскими цифрами. В качестве идентификатора нельзя использовать служебные слова. Идентификатор должен быть уникальным, т.е. в данном блоке программы один идентификатор не может быть использован для обозначения более чем одного объекта.
Например : Zap, gor,X, p1, summa, a28,rar_1, proba, x1,y1,max,min и т.д.

Типы переменных

Тип		Диапазон значений
Целый	ShortInt	-128...127
	Integer	-32768 ...32767

	Longint	-2147483648 ... 2147483647
	Byte	0 255
	Word	0 ... 65535
Вещественный	Real	$2{,}9\ 10^{-39}$..... $1{,}7\ 10^{38}$ 11..12 значащих цифр
	Single	7..8 значащих цифр
	Double	15..16 значащих цифр
	Extended	19..20 значащих цифр
	Comp	19..20 значащих цифр
Логический	Boolean	TRUE FALSE
Символьный	Char	1 символ
Строковый	String	Массив символов, по умолчанию длина 256 символов.
Массивы	array	
Записи	record	
Множества	set	
Файлы	file	
Указатели	pointer	

§3. Операторы присваивания. Оператор ввода-вывода. Арифметические операции. Стандартные функции

Оператор присваивания (:=).

переменная := { переменная
арифметическое выражение
функция
число или константа

Значение выражения из правой части оператора присваивания заменяет текущее значение переменной из левой части.

!!! Тип значения выражения должен совпадать с типом переменной

ПРИМЕР : a=2; a | 2 |

a=a+2; a | 4 |

Стандартные математические функции

abs(x)	Абсолютное значение x, т.е. x
exp(x)	Значением функции является e в степени x.
sin(x) и cos(x)	Значение синуса или косинуса x, x должен задаваться в радианах.
arctan(x)	Арктангенс x.
ln(x)	Натуральный логарифм x (x>0)
sqr(x)	Квадрат x.
sqrt(x)	Квадратный корень из x.
random(x)	Случайное число из диапазона 0<=...< x
Pi	Значение пи.
odd(x)	Значение функции true, если x нечетен, и false в противном случае.

inc(x,n)	Значением является x увеличенное на n.
dec(x,n)	Значением является x уменьшенное на n.
int(x)	Целая часть числа x.
frac(x)	Дробная часть числа x.
trunc(x)	Целая часть в форме longint.
round(x)	Значение x округленное до следующего целого числа.

Правила записи арифметических выражений.

1. Все данные, входящие в арифметическое выражение, должны быть одного типа. Допускается использовать вместе данные целого и действительного типа.
2. Записывать все составные части в одну строку без подстрочных и надстрочных индексов.
3. Использовать скобки одного типа - круглые. ([{ и другие скобки применять запрещается)
4. Нельзя записывать подряд 2 знака арифметических операций.
5. Вычисления выполняются слева направо в соответствии со старшинством операций:
   1) вычисление функций;
   2) * / DIV (деление нацело)
      MOD (получение остатка от деления)
   3) + -

Правила записи стандартных функций.

1. Имя функции записывается латинскими буквами.
2. Аргумент функции записывается в круглых скобках после имени функции.
3. Аргументом функции может быть : константа, переменная или арифметическое выражение.

Например :

$ax^2 + bx + c$ записывают так a*x*x +b*x + c

$\sqrt{b^2 - 4ac}$ записывают так sqrt(b*b – 4*a*c)

$\frac{a + c - 2b}{3 - x}$ записывают так (a + c – 2*b)/(3-x)

Рассмотрим примеры использования арифметических действий :

Правильно :

```
VAR  a,b : integer;
       r,s : integer;
.....................
   r:=a div b;       {r=3 при a=7, b=2}
   r:= a mod b;    {r=1 при a=7,b=2}
   s:=a*b;
   s:=a div b;
```

Неправильно :

```
VAR a,b : integer;
        r : integer;
```

r:=a/b; {если результат объявлен как целочисленный, нельзя использовать наклонную черту деления}

```
VAR  a,b : real;
        r : integer;
        .................
```

r:=a div b; { нельзя использовать операцию div для вещественных чисел}

r:=a mod b; { операция mod используется только по отношению к целым числам}

```
VAR  a,b : integer;
        r : real;
      ...................
```

r:=a div b; {r должно быть целым}

Слева по отношению к оператору присваивания должен стоять более широкий тип.

```
VAR   a : integer;
      b : real;
      c : real;
        .....
      c:=a+b    {правильно}
      a:=c+b;   {неправильно}
```

Оператор ввода .

Read(<список переменных>**);**
Readln(<список переменных>**);**

После выполнения данного оператора программа останавливается и ждет ввода данных с клавиатуры. Если вводятся числовые данные, то их можно ввести через пробел друг за другом, а можно каждый отдельно нажимая в конце ввода Enter. Переменный в <списке переменных> разделяются запятой.

Ln за словом Read означает, что после ввода следует перевести курсор на следующую строку.

Вводить с клавиатуры можно только значения переменных, но не выражения.

```
READLN(f,b,a);
read(s);
readln(w);
```

Оператор вывода .

Write(‘комментарии**’, <**переменные**>);**
Writeln(‘ком - рии**’, <**переменные**>);**

Переменный в <списке переменных> разделяются запятой.

Для вывода целых и действительных чисел можно указывать форматы в операторе Write. Формат указывается через двоеточие после переменной.

Write(y:5:2) на вывод значения y отведено 5 позиций, из них 2 на дробную часть.

Writeln('y=',y:8:3) - восемь позиций, на дробную часть 3.

Writeln('M=',M:4) - переменная M целого типа, на нее отведено 4 позиции.

§4. Структура программы.

(* комментарии *)
PROGRAM <имя программы>**;**
USES - модули;
{раздел описаний }
LABEL - описание меток**;**
CONST - описание констант**;**
TYPE - описание типов**;**
VAR - описание переменных**;**
PROCEDURE - описание процедур;
FINCTION - описание функций**;**
{начало раздела операторов}
BEGIN
операторы ввода, вывода и обработки данных
END. {конец раздела операторов, конец программы}

Среди разделов описаний есть раздел описания переменных. Он начинается со служебного слова **Var**, после которого идет последовательность объявлений переменных, разделенных точкой с запятой. В каждом объявлении перечисляются через запятую имена

переменных одного типа, после каждого списка имен переменных ставится двоеточие **Var f,a : integer;**

g,y1,y2 : real;
s,s1,s2 : char;

Раздел описания констант начинается со слова **CONST.**

Пример :

Const n=25; r=38;
m=5.15;

Первая программа.

Задача 1 : Написать программу, которая вводит значения двух любых чисел, выводит на экран сумму, разность, произведение и частное от деления этих чисел. Ввод каждого числа произвести с отдельной строке. Каждый результат также нужно поместить на отдельную строку, пояснив в комментарии, чему соответствует выводимое число.

```
program exampl_1;
    var a,b : integer;
begin
    writeln('Введите первое число’);
    readln(a);
    writeln('Введите второе число’);
    readln(b);
    writeln('A+B=  ', a+b:5);
    writeln('A-B=  ', a-b:5);
    writeln('A*B=  ', a*b:7);
    writeln('A div B=  ', a div b:5);
    writeln('Нажмите <Enter>');
    readln;
  end.
```

Сохранение программы.

Для того чтобы сохранить программу, необходимо либо**,** либо :

- нажать Enter, в вертикальном меню выбрать команду **Save as...** и нажать клавишу Enter;
- в появившемся окне ввести имя файла и нажать Enter;

Компиляция программы

или

- выйти в верхнее меню и выбрать команду **Compile** и нажать клавишу Enter;

или

- нажать клавишу **ALT-F9.**

Запуск программы на выполнение.

или

- выйти в верхнее меню и выбрать команду **RUN** и нажать клавишу Enter;

или

- нажать клавишу **CTRL-F9.**

Алгоритм работы с программой

1. Написать программу на бумаге.
2. Запустить Турво Паскаль.
3. Набрать программу при помощи клавиатуры.
4. Сохранить программу на диске.
5. Запустить программу на компиляцию.
6. Запустить программу на выполнение.

Наша программа есть пример линейного алгоритма.

Линейные алгоритмы описывают решение задач с последовательным выполнением действий. Обычно такие действия идут в следующем порядке :

- ***ввод исходных данных*** (может отсутствовать, тогда данные задаются внутри программы),
- ***последовательные команды*** - обычно вычислительного характера;
- ***вывод результатов*** (должен присутствовать обязательно).

Задача 2 : Написать программу определения цифр трехзначного числа.

```
program examp2_1;
   var a, z1,z2,z3, rez : integer;
begin
   writeln('Введите  число');  readln(a);
   z1:=a Mod 10;
   writeln('Цифра единиц числа - 'z1);
   z2:=(a Div 10) Mod 10;
   writeln('Цифра десятков числа - 'z2);
   z3:=a Div 100;
   writeln('Цифра сотен  числа - 'z3);
   rez:=z3*100+z2*10+z1;
   writeln('Это тоже число - 'rez);
end.
```

Эксперименты с программой.

1. Введите в качестве исходного данного число 3476589. Убедитесь, что у вас получается неправдоподобный результат.
2. Вместо числа введите какой-нибудь символ. убедитесь, что компьютер выдает сообщение об ошибке “Error 106: Invalid numeric format”.
3. Добавьте лишний знак апострофа в операторе Writeln. Убедитесь, что программа не проходит компиляцию, а система сообщает об ошибке “Error 8: String constant” exceeds line”.
4. Измените программу examp12_1 для нахождения цифр двузначного числа. Сохраните ее под именем examp12_2.
5. Измените программу examp12_2 для нахождения цифр четырехзначного числа. Сохраните ее под именем examp12_3.
   Решите следующую задачу : Дано двузначное число.
   Определить :а) сумму и произведение цифр числа;

б) число, образованное перестановкой цифр исходного числа.

§5. Разветвляющие алгоритмы.

Составной оператор

Составной оператор представляет собой последовательность операторов, выполняемых в том порядке, в котором они записаны в программе. Он имеет следующий вид :

begin
оператор; оператор;...оператор;
end;

Количество слов BEGIN совпадает с количеством слов END. Для того чтобы избежать ошибки при составлении программы, пишите каждую пару BEGIN-END с одной позиции.

Разветвляющий алгоритм - *это алгоритм, в котором в зависимости от выполнения или невыполнения некоторого условия выполняется либо одна, либо другая последовательность действий.*

Оператор if или условный оператор

Условный оператор может записываться в полной и неполной форме.

Полная форма условного оператора :

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

Обратите внимание на запись служебных слов и операторов. Каждое слово ELSE пишется под своим словом IF. Для большей наглядности и «читаемости» текста программы операторы, следующие за словом THEN и ELSE можно писать на следующей строке.

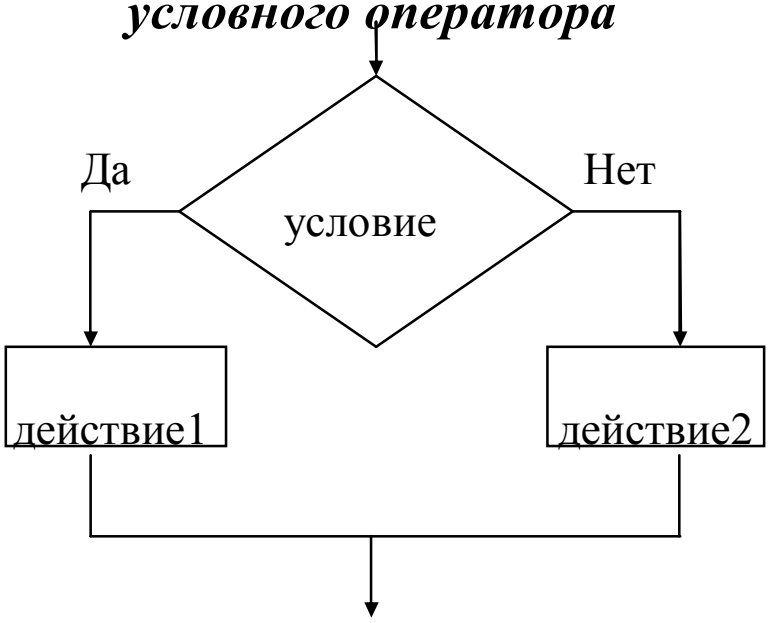
Внимание !!!! Перед словом ELSE нельзя ставить точку с запятой.

Условие в операторе IF..THEN...ELSE может быть *простым* или *сложным. Простое условие* представляет собой логическое выражение, состоящее из двух выражений одинакового типа, соединенных знаком операции отношения (>,<,>=,<>,<=). *Сложное условие* состоит из простых, соединенных знаками логических операций : OR(или) NOT(не) AND (и). При этом каждое условие ограничено с обеих сторон круглыми скобками.

Выполнение оператора IF... THEN... ELSE

1. Проверяется : выполняется или нет условие, стоящее после слова IF.
2. Если условие выполняется, т.е. значение его «истина», то выполняется оператор, стоящий после слова THEN, а затем управление передается оператору программы, записанному после оператора IF ...THEN...ELSE.
3. Если условие оператора IF не выполняется (т.е. оно имеет значение “ложь”), то выполняется оператор, стоящий после слова ELSE, а затем начинает выполняться следующий после IF...THEN...ELSE оператор.

Блок-схема полного условного оператора



Задача 1: ***Составить программу вывода на экран большего из двух чисел.***

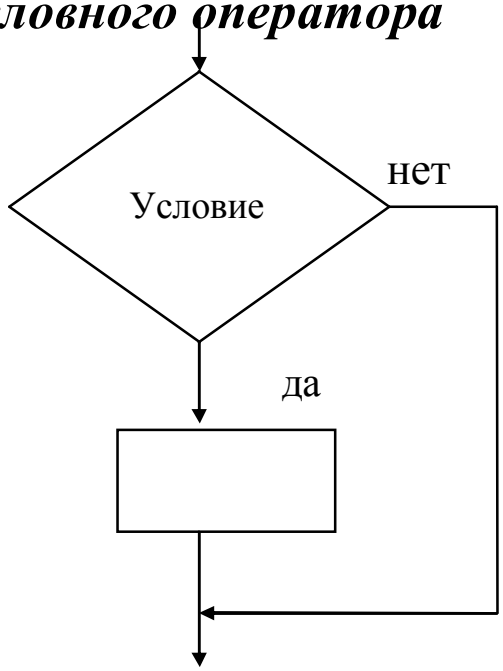
```
PROGRAM prim1;
USES Crt;
var x,y : real;
begin
    ClrScr; (* очищение экрана *)
     Writeln('Введите два числа');
     Readln(x,y);
     IF x>y THEN Writeln(x)
     ELSE  Writeln(y);
end.
```

Неполная форма условного оператора :

if<*условие*> ***then*** <*оператор*>

Действие этого оператора отличается от предыдущего том, что в случае невыполнения условия начинает выполняться следующий после IF...THEN...оператор программы.

Блок-схема неполного условного оператора



Задача 2 : ***Написать программу поиска наибольшего из двух чисел. Найденное число должно быть помещено в переменную MAX.***

```
PROGRAM prim2;
USES Crt;
var x,y ,max : real;
begin
    ClrScr; (* очищение экрана *)
     Writeln('Введите два числа');
     Readln(x,y);
     max:=x;
     IF y>max THEN  max:=y;
     Writeln(max);
end.
```

Решая всевозможные задачи часто необходимо выполнять группу операторов либо за словом THEN , либо за словом ELSE. Для того чтобы записать группу действий необходимо использовать составной оператор BEGIN END.

1. *If <условие> then*
   begin
   onepamop1;
   onepamop2;
   .....
   onepamop n;
   end
   else onepamop;

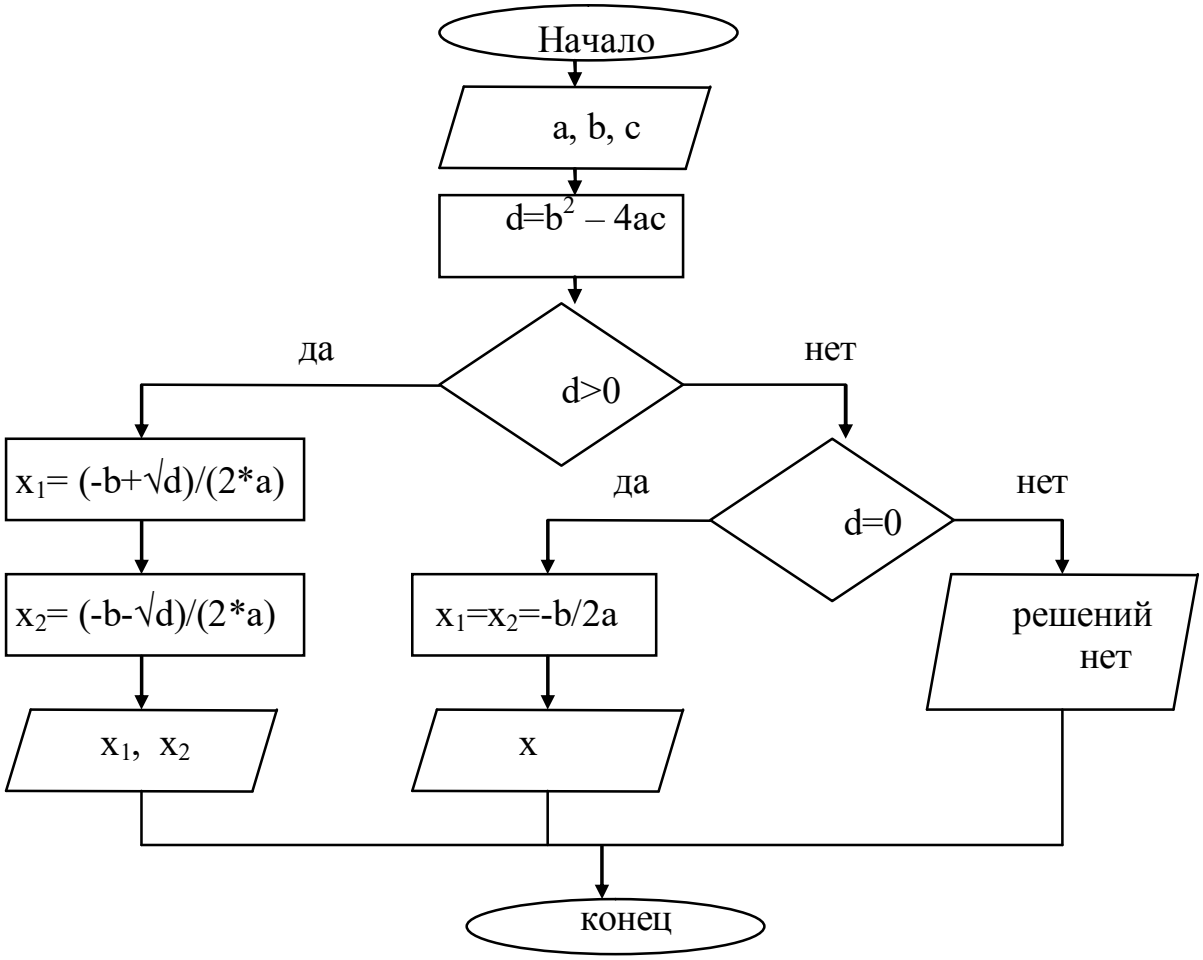
2. *If <условие> then onepamop*
   else begin
   onepamop1;
   onepamop2;
   .....
   onepamop n;
   end;

3. *If <условие> then*
begin
оператор1;
оператор2;
.....
оператор n;
end
else
begin
оператор1;
оператор2;
.....
оператор n;
end

Задача 3. Решить квадратное уравнение вида
$ax^2 + bx + c = 0$
План решения задачи :
1. задать коэффициенты a,b,c.
2. вычислить значение дискриминанта.
3. Проверить значения дискриминанта и, в зависимости от полученного ответа, выполнить то или иное действие.

```
program kvyr;
var a,b,c : integer;
     x1,x2,d : real
begin
writeln('введите коэф-ты квадратного уравнения');
  read(a,b,c);
  d:=b*b-4*a*c;
  if d>=0 then begin
                    x1:=(-b+sqrt(d))/(2*a);
                    x2:=(-b-sqrt(d))/(2*a);
                    writeln('корни  x1=,x1,' x2',x2)
                end
```

```
    else if d=0  then writeln('x1=x2=',-b/(2*a))
         else writeln('решений нет');
end.
```



§6. Оператор цикла с параметром

Циклический алгоритм

Алгоритмом *называется понятное и точное предписание (указание) исполнителю совершить определенную последовательность действий для достижения поставленной цели для решения поставленной задачи.*

Цикл - *процесс многократного повторения каких-либо действий.*

Действия, повторяющиеся в цикле, называются ***телом цикла.***

Различают 3 вида циклов :

1) **Цикл с предусловием «Пока»**
2) **Цикл с постусловием «До»**
3) **Цикл с параметром «Для»**

Цикл с параметром «Для»

Возьмем очень простой пример - таблицу умножения: число 2 умножаем на числа от 1 до 10 и печатаем ответ. Какие действия повторяются ? Повторяются действия умножения и вывод значения произведения на экран. При этом заметим, что второй сомножитель увеличивается при каждом действии на единицу, и операция умножения повторяется заданное количество раз. Как мы видим, в этом случае процесс вычислений носит циклический характер, причем число повторений цикла известно к началу его выполнения, управление циклом осуществляется с помощью переменной (счетчик цикла), которая в этом циклическом процессе принимает последовательное значение от заданного начального значения до заданного конечного значения.

Значение таблицы умножения на 2 можно записать так :
2 x i = a, где i изменяется от 1 до 10, a - от 2 до 20.

Оператор цикла с параметром применяют тогда, когда заранее известно число повторений одной и той же последовательности операторов. Начальные и конечные значения параметра цикла могут быть представлены константами, переменными или арифметическими выражениями.

FOR K:=M1 TO M2 DO
оператор1

```
FOR K:=M1 TO M2 DO
 BEGIN
        оператор1;
       .....
        оператор n;
 END;
FOR K:=M1 DOWNTO M2 DO
 BEGIN
```

операторы циклической части

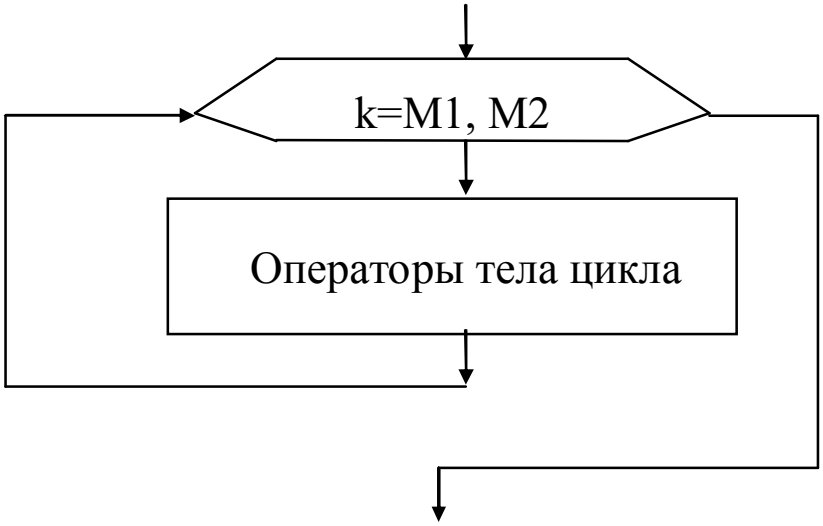
END;

ГДЕ ***К - параметр цикла, М1 и М2 - начальное и конечное значение параметра цикла. DOWNTO изменяет параметр с шагом -1.***

Внимание !

1) Счетчик изменяется на единицу при каждом следующем исполнении оператора цикла.
2) Не забывайте, что счетчику присваивается начальное значение, т.е. К=**начальное** значение.
3) Если начальное значение совпадает с конечным значением, то операторы цикла (еще говорят «тело цикла») выполняются один раз.
4) Если начальное значение счетчика больше конечного значения, то тело цикла не выполняется ни разу.
5) При выходе из цикла значение счетчика совпадает с конечным значением, т.е. К=М2

Блок-схема цикла «FOR»



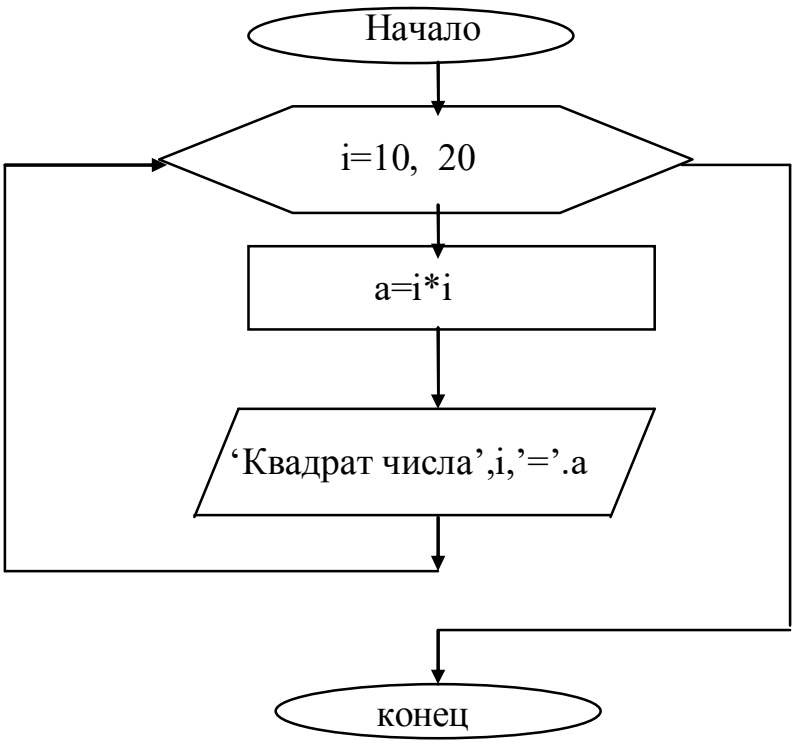
Пример 1: Вывести на экран квадраты натуральных чисел от 10 до 20.

```
PROGRAM N1;
VAR  i: integer; {счетчик цикла}
       a: integer;
BEGIN
    FOR i:=10 TO 20 DO
```

```
        BEGIN
        a:=i*i;
        writeln('квадрат числа',i,'=',a);
    END
END.
```

Или

```
FOR i:=10 TO 20 DO writeln('квадрат числа',i,'=',i*i);
```



Пример 2. Вывести на экран натуральные числа от 40 до 10.

```
PROGRAM N2;
VAR  i: integer; {счетчик цикла}
BEGIN
    FOR i:=40 DOWNTO 10 DO
        write(i,' ')
END.
```

§7. Базовые циклические алгоритмы

1. Алгоритм вычисления суммы чисел.

Вычислить сумму 12 целых чисел, введенных с клавиатуры.

```
PROGRAM summa;
VAR  a: integer; {введенное число}
       i: integer; {счетчик цикла}
      S: integer; {сумма}
BEGIN
     S:=0;
     FOR i:=1 TO 12 DO
       BEGIN
          writeln('задайте значение очередного слагаемого ');
          readln(a);
          S:=S+a
       END;
       writeln('Сумма =',S);
END.
```

2. Подсчет количества чисел удовлетворяющих заданному условию.

Подсчитать количество отрицательных чисел из 10, введенных с клавиатуры.

```
PROGRAM kol;
VAR  a: integer; {входная переменная}
       i: integer; {счетчик цикла}
      K : integer; {счетчик отрицательных чисел}
BEGIN
     K:=0;
     FOR i:=1 TO 10 DO
       BEGIN
          write('задайте значение переменной a: ');
          readln(a);
          IF a<0 THEN  K:=K+a
       END;
      IF K=0 THEN writeln('Отрицательных чисел нет»)
```

```
        ELSE writeln('Количество отрицательных чисел',K:4);
END.
```

3. Вычислить сумму чисел, кратных 3, из десяти чисел введенных с клавиатуры.

```
PROGRAM summa;
VAR  a: integer; {введенное число}
       i: integer; {счетчик цикла}
      S: integer; {сумма}
      K: integer; {счетчик чисел кратных 3}
BEGIN
    S:=0;  K:=0;
    FOR i:=1 TO 10 DO
      BEGIN
        write('задайте значение очередного слагаемого ')
        readln(a);
        IF a mod 3=0 THEN
                        begin
                          K:=K+1;
                           S:=S+a;
                end
      END;
   IF K=0 THEN   writeln('чисел кратных 3 нет')
    ELSE writeln('сумма чисел кратных 3 =', S:6)
END.
```

4. Алгоритм вычисления n!.

n!=1*2*3*...*n. 0!=1

```
PROGRAM N;
VAR  F: integer;
       i: integer;
BEGIN
    F:=1;
    FOR i:=1 TO 15 DO
        F:=F*i;
    writeln('15! =',F)
END.
```

5. Алгоритм вычисления a^n.

```
PROGRAM step;
VAR  a: integer; {основание}
       n:integer; {показатель степени}
       i: integer; {счетчик цикла}
      W: integer; {результат}
BEGIN
    write('задайте основание a');   readln(a);
    write('введите показатель n');   readln(n);
    W:=1;
    FOR i:=1 TO n DO
      W:=W*a;
    writeln(a,'в степени ',n,' равно ',W:6)
END.
```

Вычислить : 1! + 2! + 3! + ... + n!, при n=5

```
.....
s:=0; F:=1;
For i:=1 to 5 do
 begin
    f:=f*i;
    s:=s+f;
end;
```

§8. Цикл с предусловием WHILE

Цикл с предусловием **WHILE** используется тогда, когда число повторений оператора цикла заранее не известно, а задается некоторое условие продолжение цикла.

Форма записи:

WHILE *<лог выр-ние / условие>* ***DO***
BEGIN *операторы циклической части программы*
END

WHILE (пока) **DO** (выполнять)

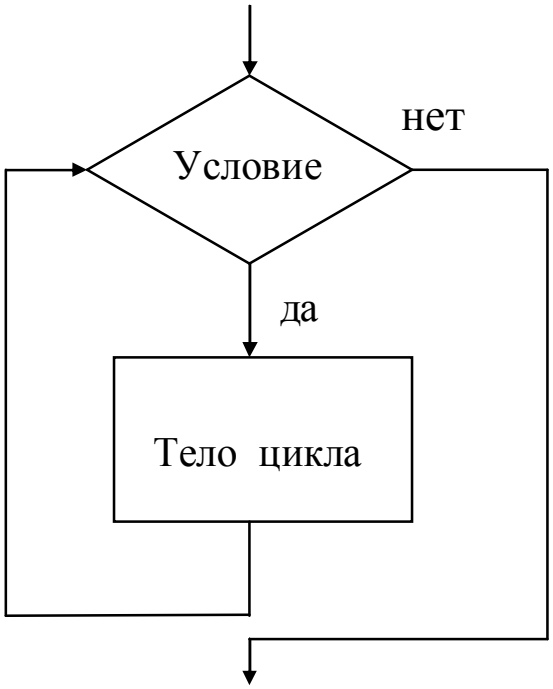
Сначала выполняется значения условия . Пока оно истинно выполняются операторы циклической части . Когда только оно становится ложным, происходит выход из цикла. Если условие ложно то цикл не выполняется ни разу. Возможен случай , когда в циклической части стоит оператор перехода (EXIT, GOTO) передающий управление за пределы цикла. В такой ситуации цикл может завершиться до его естественного окончания.

Данная конструкция может работать, как конструкция цикла со счетчиком, но при этом необходимо изменять переменную, являющуюся счетчиком, с помощью оператора присваивания. В конструкции WHILE вы можете выбирать любой необходимый вам шаг.

Пример :

```
a:=1;
n:=1;
WHILE 2-a<=3-n+1  DO
  BEGIN
     a:=a+2;
     n:=n+1
  END;
```

Блок-схема цикла «Пока»



Задача 1. Найти количество чисел, сумма которых превысит 100.

```
PROGRAM N;
var  k:integer;
     s:integer;
begin
     s:=0;   k:=0;
    while  s<=100  do
      begin
          k:=k+1;
          s:=s+k
      end;
    writeln('количество слагаемых=',k);
end.
```

Задача 2. Вычислить сумму четных чисел на отрезке от 10 до 30.

```
PROGRAM N;
var  k:integer; {очередное слагаемое}
     s:integer;   {сумма}
```

```
    begin
     s:=0;   k:=10;
    while  k<=30  do
      begin
          s:=s+k;
          k:=k+2
      end;
    writeln('сумма четных чисел от 10 до 30=',s);
end.
```

Задача 3. Напечатать в виде таблицы значения функции $y=4x^2 - 2x + 3$, для значений x, изменяющегося от -4 до 2 с шагом 0.5

```
PROGRAM N;
var  x:integer;
     y:real;
begin
     x:=-4;
    while  x<=2  do
      begin
          y=4*x*x-2*x+3;
          x:=x+0.5
      end;
    writeln('при x=',x,'  y=',y:8:3);
end.
```

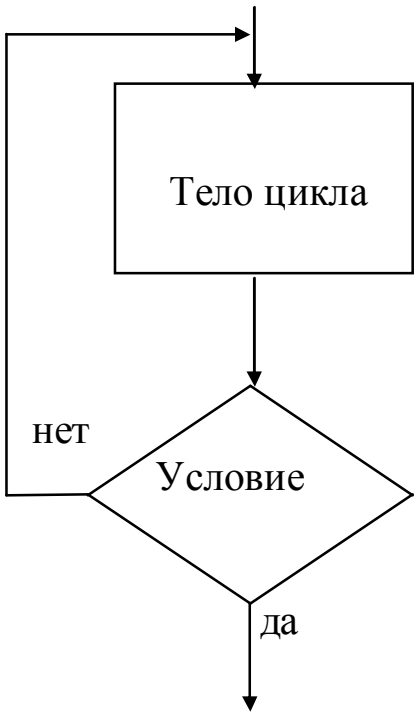
§9. Оператор цикла с постусловием Repeat

Этот оператор отличается от цикла с предусловием WHILE тем, что проверка условия производится после очередного выполнения тела цикла. Это обеспечивает выполнение цикла хотя бы один раз.

Форма записи

REPEAT	***Повторить***
операторы	
циклической части	
программы	
UNTIL *<логическое выражение>*	***До тех пор***

Блок-схема цикла «До»



Операторы циклической части выполняются повторно до тех пор пока значение логического выражения ложно.

Условием прекращения циклических вычислений является истинное значение логического выражения

Служебное слово REPEAT открывает тело цикла затем выполняются последовательно операторы тела цикла и проверяется условие выхода из него.

Запомните !

1. Тело цикла выполняется хотя бы один раз в любом случае.
2. В ходе работы цикла мы должны прийти к истинности условия.

Задача 1. Вычислить сумму квадратов натуральных чисел до тех пор, пока квадрат очередного числа не превысит 81.

```
Program n;
var  k:integer; {слагаемое}
     s:integer; {сумма}
begin
  k:=1;   s:=0;
  REPEAT
       s:=s+k*k;
       k=k+1
  Until k>81;
  writeln('сумма равна=',s)
end.
```

Задача 2. Вычислите значение функции $Y=X^2$, при x=8; 6; 4 ;2

```
X=8;
REPEAT
 Y:=X*X;  writeln(x:3, y:5); X:=X-2
UNTIL  X=0
```

Задача 3. С клавиатуры вводятся последовательно целые числа, и вычисляется их сумма до первого встречного отрицательного числа (не включая его самого).

```
Program ex3;
Var  a: integer; {вводимое число}
     s: integer; {сумма}
Begin
     s:=0;
     a:=0; {мы вынуждены выполнять этот искусственный шаг}
     REPEAT
       s:=s+a;
       readln(a)
```

```
    UNTIL  a<0;
    Writeln(s);
End.
```

Задача 4. Найти наибольший общий делитель (НОД) двух чисел. НОД - это наибольшее целое число, которое делит нацело оба числа.

Решение : Даны числа a и b, то для чисел a и b выполняется равенство НОД(a, b)=НОД(b, r), где r - остаток от деления a на b .

r= a mod b=a- (a div b)*b

Например, пусть a=48, b=18.

A	b	Результаты
48	18	-
48 mod 18 = 12	18	НОД(48,18)=НОД(12,18)
12	18 mod 12 = 6	НОД(12,18)=НОД(12,6)
12 mod 6 = 0	6	НОД(12,6)=НОД(0,6)
0	6	НОД(0,6)=6

```
Program NOD;
  var a,b:longint;
begin
   writeln('введите два числа');
   readln(a,b);
   REPEAT
      if a>b then a:=a mod b  else b:=b mod a
   UNTIL (a=0) or (b=0);
    writeln('НОД=', a+b);
    readln;
end.
```

Выбор конструкции цикла зависит от конкретного алгоритма. Для каждого задания необходимо продумать, какая конструкция является наиболее оптимальной.

§10. Эксперимент с программой. Лабораторная работа

Задача1 : Дано натуральное число n (n<=9999). Определить, является ли оно палиндромом («перевертышем»), с учетом четырех цифр. Например, палиндромами являются числа : 2222, 6116, 0440.

Итак, у нас четырехзначное число, поэтому переменная оператора For изменяется от 1 до 4. В переменной m хранится «остаток» числа, сначала он равен введенному числу. В переменной с именем r формируем значение числа - «перевертыша». Основными операциями являются : r:=10*r + m Mod 10 m:=m Div 10.

Результат трассировки приведен в таблице.

i	m	r
-	3994	0
1	399	0*10+3994 mod 10= 0+4=4
2	39	4*10+399 mod 10=40+9=49
3	3	49*10+39 mod 10=490+9=499
4	0	499*10+3 mod 10=4990+3=4993

```
Program ex1;
Var n,m,r,i : Integer;
Begin
   writeln('Введите целое число, меньшее    10000');
   readln(n);
  m:=n; r:=0;
 For i:=1 to 4 do
    begin
     r:=r*10+m mod 10;  m:=m Div 10;
   end;
  if r=n Then Writeln('Да');
          Else Writeln('Нет');
  readln;
end.
```

1) Измените программу так, чтобы можно было обрабатывать целые числа из диапазона Longint.

2) Замените цикл for, в программе, на циклы While и Repeat ... Until

Задача 2 : Дано натуральное число n . Требуется подсчитать количество цифр данного числа.

Решение: Подсчет цифр начнем с последней цифры числа. Число делим на 10, убирая последнюю цифру и т.д.

Пусть m - это число, л- счетчик цифр.

```
Program ex2;
var m,n : Longint;
        k : Integer;
begin
  writeln('введите целое число');
  readln(n);
  m:=n; k:=0;
  While  m<>0 Do
     begin  Inc(k); m:=m Div 10     end;
   writeln ('В числе',n,' - ',k,'  цифр  ');
end.
```

Модифицируя программу ex2, решите следующие задачи :

- найдите сумму цифр числа;
- найдите первую цифру числа;
- поменяйте порядок цифр числа на обратный;
- найдите количество четных цифр числа;
- найдите наибольшую цифру числа;
- найдите сумму цифр числа, больших 5;
- ответьте на вопрос, сколько раз данная цифра встречается в числе.

Задача 3 : Натуральное число p называется простым, если оно делится только на 1 и на себя. По соглашению 1 не считают простым числом. Начало последовательности простых чисел имеет вид : 2,3,5,7,11,13, 17, 19, 23, ...

В программе ex3 определяется, является ли данное число простым. Мы ищем делители числа n в интервале от 2 до n div 2, хотя можно было бы ограничиться интервалом от 2 до целой части √n

```
program ex3;
Var i, n :Longint;
Begin
  Writeln('Введите натуральное число');
  Readln(n);
   i:=1;
    Repeat
       inc(i)
    Until (i>n div 2) or (n mod i = 0);
    if i> n div 2 Then Writeln('число ',n,'простое');
   Else Writeln('Число ',i,'- первый делитель
                              числа', n, ' , больший 1');
  end.
```

Эту задачу можно решить и с использованием оператора While. Сделайте это. Затем измените программу так, чтобы в ней осуществлялся вывод всех делителей числа n

Подсказка. Логическое выражение в операторе Repeat....Until упростится, а в нем останется только условие i>n div 2, а в теле цикла появится оператор
if n mod i = 0 Then Writeln(....,i)

§11. Оператор варианта выбора

Case <порядковая переменная> ***of***
<константа 1>**:** <оператор 1>;
<константа 2>**:** <оператор 2>;
.....
<константа n>**:** <оператор n>;
[Else <оператор >; ***]***
end;

Если порядковая переменная равна одной из перечисленных констант, то выполняется соответствующий оператор. Затем

управление передается за пределы оператора выбора. Если значение переменной не совпадает ни с одной константой, то выполняется оператор, стоящий после Else, если он есть, либо управление передается на оператор, следующий за End.

Задача 1: Составьте программу, в которой определяется, какой буквой - гласной или согласной - является введенный символ английского алфавита.

Решение :

Разделим все символы на три группы :

- гласные буквы английского алфавита;
- согласные буквы английского алфавита;
- символы, не являющиеся буквами английского алфавита.

```
Program ex4;
var ch : char;
begin
  Writeln('Введите символ');
  Readln(ch);
  Case ch of
  'a','e','i','o','u' : writeln('Это гласная');
  'a'...'z' : Writeln('это согласная');
 else writeln('это не английский алфавит');
 end;
end.
```

Задача 2. Мастям игральных карт условно присвоены следующие порядковые номера: «пики»-1, «трефы»-2, «бубны»-3, «червы»-4. По заданному номеру масти m ($1 \le m \le 4$) определить название соответствующей масти.

```
Program mast;
var m:integer;
begin
   write('введите номер масти'); readln(m);
   case m of
     1 : writeln('пики');
     2 : writeln('трефы');
     3 : writeln('бубны');
     4 : writeln('червы')
```

```
      else writeln('вы ввели неправильный номер');
   end;
end.
```

§12. Типы определенные пользователем

Раздел описания типов

В языке Паскаль все данные, используемые программой, должны принадлежать к какому-либо заранее известному типу данных.

Тип данных определяет:

- формат представления данных в памяти ЭВМ;
- множество допустимых значений;
- множество допустимых операций.

Все простые типы можно разделить на стандартные и пользовательские. Стандартные типы данных рассматривались в задании1.

Пользовательские типы объявляются в разделе описания типов, который открывается словом **Type.**

```
Type
week=(sunday, monday, tuesday, wednesday, thursday,
      friday,  satufday);
work_week=monday..friday;
day=1..31;
```

Обратите внимание на то, что при объявлении пользовательских типов между их именем и конструкцией, определяющей тип, ставится знак равенства.

Перечислимый тип данных

Этот тип данных получил название перечисляемого, потому что он задается в виде перечисления некоторых значений. Эти значения образуют упорядоченное множество и являются константами. Для объявления переменной список возможных значений, разделенных запятой, указывается в круглых скобках.

Например,

Var month(january, february, march, april, may, june, july, august, september, october, november, december);

Порядок элементов перечисляемого типа определяется порядком их следования в описании. Левый имеет минимальное значение.

В любом порядковом типе для каждого значения, кроме первого, существует предшествующее значение, и для каждого значения, кроме последнего, существует последующее значение. В языке Паскаль имеются стандартные функции, которые позволяют определять предшествующее и последующее значения для заданного значения:

функция Pred(x) определяет значение, предшествующее x;

функция Ord(x) возвращает порядковый номер величины x;

функция Succ(x) определяет значение, следующее за x.

К переменным перечисляемого типа можно применять операции сравнения. Так, например, february<november.

Задача 3.Вычислить значения :

а) Ord(august);

б) Ord(succ(september));

в) Pred(Pred(december)).

Задача 4. Имеются описания :

```
var x,y : (winter, spring, summer, autumn);
          t : (cold, warm);
```

а) Допустимы ли присваивания :x:=spring; t:=warm: t:=hot; y:=x; y:=t;

б) Вычислить значения выражений :

```
spring<summer;
autumn<winter;
Succ(spring);
Pred(autumn);
Ord(spring);
winter<=summer;
spring<>warm;
Pred(spring);
Pred(cold);
Pred(autumn) + Ord(cold);
```

в) Допустим ли оператор цикла с заголовком : For x:=spring To autumn Do

Задача 5. Даны описания следующих переменных :

```
VAR m, m1: (january, february, march, april, may, june, july, august,
                  september, october, november, december);
```

k: 1..maxint; n: 1..12;

Присвоить переменной m1:

а) название месяца, следующего за месяцем m;

б) название k-го месяца после месяца n.

Задача 6. Имеются описания :

VAR d: '0'..'9'; k: 0..9; n: integer;

а) Какие значения может принимать переменная d ? Каков ее базовый тип? Допустимы ли присваивания : d:='7'; d:='a'; d:=7 ?

б) Какие значения может принимать переменная k ? Каков ее базовый тип ? Допустимы ли присваивания : k:=5; k:=10; k:=-0: k:='5' ?

в) Есть ли ошибки в операторе :

```
if k+n>7*k  then k:=abs(n) mod 10
            else d:=chr(k+Ord('0')  ?
```

§13. Вложенные циклы

Для решения задачи достаточно часто требуется использовать две и более циклические конструкции, одна из которых расположена внутри другой (других). Такие конструкции называют вложенными циклами. Какие именно циклы при этом используются, роли не играет, они могут быть организованы посредством любых рассмотренных ранее операторов (For, While, Repeat ... Until).

Задача : Сколько можно купить быков, коров и телят, если бык стоит 10 рублей, корова - 5 рублей, теленок - полтинник (0,5 рубля), при условии, что на 100 рублей надо купить 100 голов скота.

Решение : Обозначим через b- количество быков, k - количество коров, t - количество телят. После этого можно записать два уравнения:

10b+5k+0,5t=100 и b+k+t=100

Преобразуем их :

20b+10k+t=200 и b+k+t=100

На 100 рублей можно купить :

- не более 10 быков, т.е. 0<=b<=10;
- не более 20 коров, т.е. 0<=k<=20;
- не более 200 телят, т.е. 0<=t<=200;

Таким образом получаем :

```
Program skot;
 Var b,k,t : integer;
 Begin
   For b:=0 to 10 do
    For k:=0 to 20 do
      For t:=0 to 200 do
       if (20*b+10*k+t=200) and (b+k+t=100) then
        writeln('быков ',b,' коров ',k,' телят ',t);
 end.
```

Условие будет проверятся 11*21*201=46431 раз. Но задачу можно сократить на один цикл если количество телят вычислять по формуле : t=100-(b+k)

§14. Одномерные массивы

Работа с элементами

Одномерный массив - это фиксированное количество элементов одного типа, объединенных одним именем. Каждый элемент имеет свой номер- индекс. Обращение к элементам массива осуществляется с помощью указания имени массива и номера элементов.

Пусть для решения требуется массив из 30 целых чисел.

Var mas: array [1..30] of integer;

Способы задания значений элементов массива

1) **Оператор присваивания.**

```
A[1]:=3;
A[2]:=4;
.....
```

или

```
B[1]:='Иванов';
B[2]:='Петров';
.....
```

Такой способ задания значений используется, если массив небольшой.

2) Оператором ввода с клавиатуры.

```
For i:=1 to 10 do
  begin
    Writeln('введите ',i,'-ый элемент массива');
     Readln(a[i]);
{в качестве индекса используется параметр цикла }
   end;
```

Этот способ задания значений также используется для небольших массивов.

3) Заполнение массива с использованием генератора случайных чисел.

```
.......
   RANDOMIZE;
   For i:=1 to 10 do
     a[i]:=Random(x);
…....
```

{массив заполнится целыми числами в диапазоне от 0 до X}

4) Заполнение по формуле.

Каждый элемент массива должен быть рассчитан по формуле (например a_i=sin i - cos i)

```
for i:=1 to 10 do
     a[i]:=sin(i)-cos(i);
```

Вывод массива.

```
For i:=1 to 10 do  writeln(a[i]);
```

Вывод пятого элемента массива записывается так : write(a[5]);

Алгоритмы работы с массивами

1) Сумма элементов массива.

```
Program symma;
  var a: array [1..10] of integer;
        i, s : integer;
begin
  s:=0;
  for i:=1 to 10 do
   begin
     readln(a[i]);
     s:=s+a[i];
   end;
  writeln (‘Сумма= ’,s);
end.
```

2) Сумма положительных чисел.

```
Program symma2;
  var a: array [1..10] of integer;
        i, s : integer;
begin
  s:=0;
  for i:=1 to 10 do
   begin
     readln(a[i]);
     if a[i]>0 then s:=s+a[i];
   end;
  writeln (‘Сумма положительных чисел = ’,s);
end.
```

3) Сумма и количество положительных чисел.

```
Program symma-kol;
  var a: array [1..10] of integer;
        i, s,k : integer;
begin
  s:=0; k:=0;
```

```
        for i:=1 to 10 do
    if a[i]>0 then begin k:=k+1;  s:=s+a[i];  end;
   writeln (‘Сумма  ’,s, ‘ количество’, k);
end.
```

4) Поиск заданного элемента в массиве.

Найти элементы массива большие числа 5.

```
Program elem;
  var a: array [1..10] of integer;
        i : integer;
begin
      for i:=1 to 10 do
     if a[i]>5 then writeln(a[i]);
 end.
```

5) Поиск наибольшего (наименьшего) элемента в массиве.

```
Program mm;
  var a: array [1..10] of integer;
        i, max, min : integer;
begin
for i:=1 to 10 do readln(a[i]);
max:=a[1];  min:=a[1];
for i:=1 to 10 do
   begin
    if a[i]>max Then max:=a[i];
    if a[i]<min Then min:=a[i];
   end;
  writeln (‘ max=’,max,’   min=’, min);
end.
```

6) Упорядочение массива

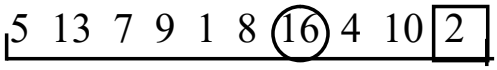
Упорядочение (сортировка) *- это расположение значений элементов массива в порядке возрастания или убывания их значений.*

Сортировка простым выбором.

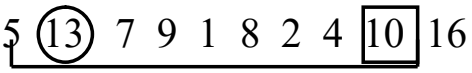
Рассмотрим идею этого метода на примере. Пусть исходный массив А состоит из 10 элементов : 5 13 7 9 1 8 16 4 10 2.
После сортировки массив : 1 2 4 5 7 8 9 10 13 16

Максимальный элемент текущей части массива заключен в кружок, а элемент, с которым происходит обмен, в квадратик. Скобкой помечена рассматриваемая часть массива.

1 шаг Рассмотрим весь массив и найдем в нем максимальный элемент - 16. Поменяем его местами с последним элементом - с числом 2.



2 шаг. Рассмотрим часть массива, исключая последний элемент. Максимальный элемент этой части - 13, он стоит на втором месте. Поменяем его местами с последним элементом этой части - с числом 10.



И т.д.

Фрагмент программы :

```
for i:=n downto 2 do
 begin {цикл по длине рассматриваемой части массива}
  maxi:=i;
  for j:=1 to i-1 do if a[j]>a[maxi] then maxi:=j;
  if maxi<>i then begin {перестановка элементов}
                        k:=a[i]; a[i]:=a[maxi]; a[maxi]:=k
                  end;
end;
```

«Пузырьковый метод»

Массив просматривают слева направо. Каждый предыдущий элемент сравнивается с последующим. Если предыдущий элемент больше последующего, то предыдущий и последующий элементы меняются местами.

```
Program sort1;
  const n=10;
  label metka;
  var a: array [1..n] of integer;
        i, c, k : integer;
begin
 for i:=1 to n do readln(a[i]);
```

```
      metka: k=0; {счетчик перестанокок}
    for i:=1 to n-1 do
     if a[i]>a[i+1]  then  begin c:=a[i];
                                  a[i]:=a[i+1];
                                  a[i+1]:=c;
                                   k:=1;
                           end;
     if   k=1 then goto metka
             else writeln('упорядочен');
     for i:=1 to n do writeln(a[i]);
   end.
```

Тоже самое, но с использованием вложенных циклов

```
   Program sort2;
     const n=10;
     var a: array [1..n] of integer;
          i, k, c : integer;
   begin
    for i:=1 to n do readln(a[i]);
    for i:=1 to n-1 do
    for k:=1 to n-i do
     if a[k]>a[k+1]  then  begin c:=a[k];
                                  a[k]:=a[k+1];
                                  a[k+1]:=c;
                           end;
     for i:=1 to n do writeln(a[i]);
end.
```

Эксперименты с программой

Дана программа замены отрицательных элементов массива на их модули:

```
   program mass;
   const n=30;
   type mas=array[1..n] of integer;
```

```
var a : mas;
    i : integer;
 begin
  for i:=1 to n do  {ввод массива}
     begin write('введите ',i, '—ый элемент массива); readln(a[i]);
end;
  for i:=1 to n do   writeln('a[',i,']=',a[i]); {вывод массива}
  for i:=1 to n do
     if a[i]<0 then a[i]:=abs(a[i]);
   for i:=1 to n do writeln('a[',i,']=',a[i]); {вывод нового массива}
end.
```

Измените программу так, чтобы она выполняла :

а) прибавить к каждому элементу массива число 25;

б) если элемент четный, то прибавить к нему первый, если нечетный - последний элемент массива. Первый и последний элементы не изменять.

в) найти значение максимального по модулю элемента массива;

г) найти среднее арифметическое значение четных элементов.

§15. Обработка символьных массивов.

Данные символьного типа.

Данные типа CHAR и STRING позволяют представлять в программах тексты и производить над ними некоторые операции, например, исправлять орфографические ошибки, вставлять и удалять

отдельные буквы и слова. Кроме того, они дают возможность обрабатывать различные ведомости, документы и т.д.

Значением строковой величины может быть любая цепочка символов.

Строка - *это последовательность символов кодовой таблицы персонального компьютера.*

Кол-во символов в строке (длина строки) может изменяться от 0 до 255.

Константа строкового типа *- это любая цепочка символов языка Паскаль, заключенная в апострофы.*

Описание строковых переменных

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

ПРИМЕР :
```
const adres=’ул.Королева, 5’;

var  s:string;     d : char;
     st1, st2 : string [30]
```

Значения строковым переменным *задаются либо оператором присваивания, либо оператором readln с клавиатуры.*

Операции над строковыми переменными

а) *Сравнения* (<>,<,>, >=,<=,=)
```
d:=’мама’;    p:=’папа’
d>p
```

б) *конкатенация* (сложение)
```
s:=’Д’+’Артаньян’;
writeln(s);
```

Задание : определить правильность написания операторов :
```
var A,B,C,D,G,Z : STRING;
                  v : integer;
```

```
begin
   A:= «Информатика»;        ошибка
   B:=Формат;                ошибка
   C:='';                    пустая строка
   D:=25 ;                   ошибка
   G='25';
   Z:='формат;               ошибка
   V:='пример';              ошибка
```

в) ***функции обработки***

1. **Delete(str,poz,n)** - удаление из строки str, начиная с позиции poz, n символов.

Пример :
str:='оператор';
Delete(str,2,2)
результат 'оратор'

2. **Insert(str1,str2,poz)** - вставка строки str1 в строку str2, начиная с позиции poz.

Пример :
str1:='ка';
str2:='Тропинка';
Insert(str1,str2,6)
результат 'Тропиканка'

3. **Length(st)** - вычисляет текущую длину(количество символов) строки.

Пример :
str:='оператор';
Length(str)
результат 8

4. **Copy(st,poz,n)** - из строки st, начиная с позиции poz, берутся n символов;

Пример :
str:='стройка';
Copy(str,2,6)

результат ‘тройка’

5. Pos(str1,str2) - поиск подстроки str2 в строке str1 и возвращает номер позиции, с которой начинается строка str2, если подстрока не найдена, то возвращается 0.

Пример :
str1:=’абракадабра’; str2:=’брак’;
pos(str1,str2)
результат 2

6. Str(v,s) - заданное число v преобразуется в строку s;

7. Val(s,v,c) - если строка s состоит из цифр, они преобразуются в некоторое числовое значение и передаются переменной v.

8. Concat(s1,s2,....,sn) -строки s1,s2,....,sn записываются одна за другой.

Задача 1 : Установить : какие типы должны иметь переменные в приведенном фрагменте программы и какие значения они примут после выполнения операций.

```
s:=’ситуация’;
st:=’ya’;
a:=Length(s+’’+st);
n:=Pos(st,s);
insert(st,s,6);
delete(s,4,2);
st:=copy(s,1,3)+copy(s,7,1);
```

Задача 2. Написать программу, которая из двух строковых констант «КЛАССШКОЛА» и «АБВГД№0123456789» выводит на экран 2 строки, содержащие номер школы и литеру класса, определяет длину полученных строк.

НАПРИМЕР : ШКОЛА №7
КЛАСС 9Б

```
program z2;
const a=’КЛАССШКОЛА’;
      b=’АБВГД№0123456789’;
```

```
var s1,s2 : string;
begin
  s1:=copy(a,6,5)+''+ copy(b,6,1)+ copy(b,14,1)
  s2:= copy(a,1,5)+''+ copy(b,length(b),1)+  copy(b,2,1)
  writeln(s1,s2);
end.
```

Символы имеют коды от 0 до 255.
ORD(w) - возвращает код символа w.
CHR(i) - определяет символ с кодом i.

Задача 1. Дана строка символов. Вывести символы строки в столбец.

```
Program xx;
var st:string; i:integer;
begin
for i:=1 to length(st) do
   writeln(st[i]);
end.
```

Задача 2. Вывести символы и соответствующие им коды. Переменная k используется в качестве счетчика для организации последовательного вывода по 15 символов.

```
Program kod;
var i, k : integer;
begin
 writeln('Вывод порядковых номеров (кодов) символов - значение
              переменной i и самих символов');
 for i:=1 to 255 do
   begin
     write(i:4,' символ',chr(i));   inc(k);
      if k=15 then begin writeln; k:=0 end
   end;
end.
```

Задача 3: Вывести символы в виде :
A

```
    BB
  CCC
  ...
  WWW...WWW (23 раза)
  program vv;
  var  i:char;  j:integer;
  begin
    for i:='A' to 'W' do
      begin
        for j:=1 to Ord(i)-Ord('A')+1  do  write(i);
       writeln;
      end;
  end.
```

Задача 4. Удалить среднюю букву строки при нечетной длине и две средние буквы при четной длине строки.

```
  Program rr;
  var st:string; k:integer;
  begin  k:=length(st);
    if k mod 2=1 then delete(st, k div 2+1,1)
    else delete(st, k div 2,2)
  end.
```

§16. Двумерные массивы

Работа с элементами

В математике часто используют многомерные массивы (двумерные, трехмерные и т.д.). Мы рассмотрим *двумерные* массивы, иначе называемые *матрицами.*

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

Данная матрица имеет размер 3 на 4, т.е. она состоит из трех строк и четырех столбцов. Если всю матрицу обозначить одним именем,

например A, то каждый элемент матрицы будет иметь два индекса - A[i,j]

Здесь первый индекс i обозначает номер строки (i=1,2,3), второй индекс j - номер столбца (j=1,2,3,4).

Такую матрицу можно описать следующим образом :

1 способ :С использованием типа

```
Type T=array [1..3,1..4] of integer;
Var  A: T;
```

2 способ :

```
Var  A: array [1..3,1..4] of integer;
```

При решении задач с использованием двумерных массивов во всех случаях (кроме некоторых частных) организуются вложенный циклы.

Перемещение по строке :

```
for i:=1 to m do  {внешний цикл, изменяется номер строки}
 .....
   for j:=1 to n do {внутренний цикл, изменяется номер столбца}
     .........
```

Перемещение по столбцу :

```
for j:=1 to n do  {внешний цикл, изменяется номер столбца}
 .....
   for i:=1 to m do {внутренний цикл, изменяется номер строки}
```

Перечислим базовые алгоритмы и рассмотрим каждый из них.

1. Заполнение двумерного массива :
   а) по строке; б) по столбцу
2. Печать в виде таблицы.
3. Вычисление суммы элементов каждой строки и каждого столбца.
4. Поиск максимального (минимального) элементов каждой строки (столбца) и их индексов.
5. Сумма элементов массива.
6. Максимальный (минимальный) элемент массива.

Ввод (заполнение) элементов двумерного массива (матрицы)

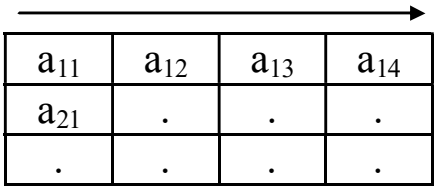
```
For i:=1 to n do
```

```
    For j:=1 to m do
    Readln(A[i,j]);
```

1. Заполнение двумерного массива по строке.

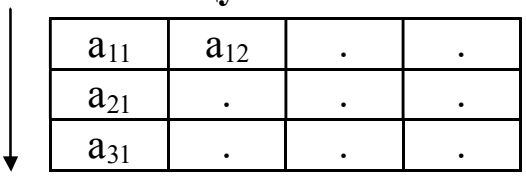
Массив содержит 3 строки и 4 столбца, т.е. 3x4=12 элементов

```
For i:=1 to 3 do
  For j:=1 to 4 do
    a[i,j]:=random(100);
```



2. Заполнение двумерного массива по столбцу :

```
For j:=1 to 4 do
  For i:=1 to 3 do
    a[i,j]:=random(100);
```



3. Печать содержимого на экран :

```
For i:=1 to n do
 begin
  For j:=1 to m do
    Write (A[i,j]); {Вывод элементов одной
                     строки матрицы}
     Writeln; { переход на следующую строку экрана}
 end;
```

где

A - имя массива;
i - индекс строки;
j - индекс столбца;
n - количество элементов в строке;
m - количество элементов в столбце.

4. Вычисление суммы элементов каждой строки, столбца.

Дана квадратная матрица NxN, содержащая вещественные числа. Найти сумму элементов первого столбца.

```
Program  pr2;
```

```
CONST  N=3;
TYPE  MAS=array [1..N,1..N] of real;
Var a: MAS;
       i: 1..3;
       j : 1..3;
       s:real;
BEGIN
Writeln('Введите элементы массива');
For i:=1 to n do
   For j:=1 to n do
      Readln(a[i,j]);
{Вывод значений массива}
For i:=1 to n do
  begin
   For j:=1 to n do
     Write (a[i,j]:5:1);
      Writeln;
  end;
s:=0; j=1;
For i:=1 to n do
    s:=s+a[i,j];
Writeln('Сумма элементов первого столбца = ',s:5:2);
end.
```

5.Вычисление суммы элементов всего двумерного массива.

```
.....
S:=0;
for i:=1 to m do
   for j:=0 to n do
     S:=S+a[i,j];
.............
```

6. Задача поиска максимального (минимального) элемента и его индексов.

Ищем максимальный элемент каждой строки :

```
For i:=0 to m do
 begin
```

```
        max:=a[i,1];
    ind_L:=i; {сохраняем номер строки}
    ind_C:=1; {заносим номер 1 - первый столбец}
    for j:=1 to n do
     if a[i,j]>max then begin
                          max:=a[i,j];
                          ind_C:=j {сохраняем номер j-ого столбца}
                        end;
    writeln('max строки ',i,'=',max)
  end;
```

Ищем минимальный элемент каждого столбца :

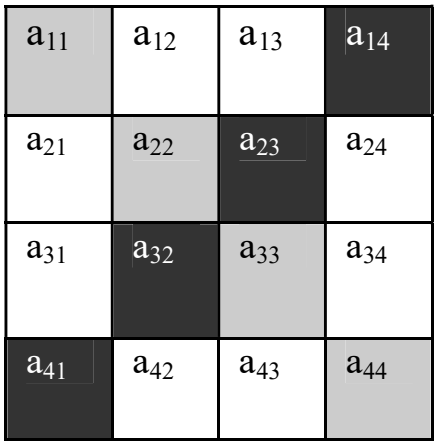
```
  For j:=0 to n do { перемещаемся по столбцу}
   begin
    min:=a[1,j];
    ind_L:=1; {сохраняем номер строки}
    ind_C:=j;  {сохраняем номер  столбца}
    for i:=1 to m do
     if a[i,j]<min then begin
                          min:=a[i,j];
                          ind_L:=i {сохраняем номер j-ой строки}
                        end;
    writeln('min ',,j,' столбца',min)
  end;
```

7. Алгоритм поиска минимального элемента и его индексов для всего массива.

```
  Min:=a[1,1];
  ind_L:=1;
  ind_C:=1;
  for i:=1 to m do
   for j:=1 to n do
    if a[i,j]<min then begin
                         min:=a[i,j]; ind_L:=i; ind_C:=j;
                       end;
```

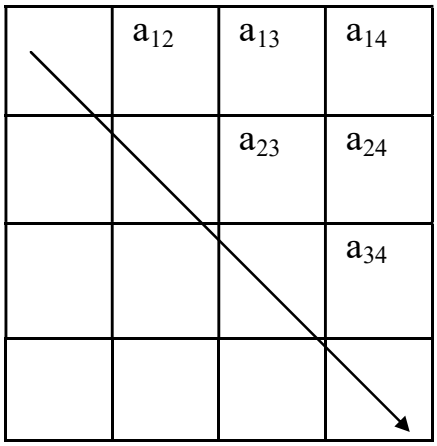
8. Квадратные матрицы.

```
  Type  mas4x4=array[1..4,1..4] of integer;
  var   a: mas4x4;
```

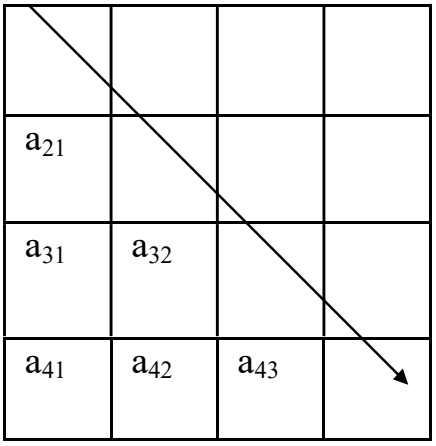


Главная диагональ - элементы a_{11}, a_{22}, a_{33}, a_{44} (индексы элементов, расположенных на главной диагонали **(i=j)**

Побочная диагональ - элементы a_{41}, a_{32}, a_{23}, a_{14} (сумма индексов элементов на 1 больше размерности строки (или столбца), т.е. i+j=4=1 или i+j=n+1. На рисунке главная диагональ закрашена сплошным серым цветом, побочная - черным.



Элементы, расположенные над главной диагональю, Для индексов элементов, расположенных над главной диагональю выполняется отношение **i<j;**



Элементы, расположенные под главной диагональю, Для индексов элементов, расположенных под главной диагональю выполняется отношение **i>j;**

Примеры :

1) Найти сумму элементов главной диагонали :

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

2) Найти минимальный элемент побочной диагонали :

```
min:=a[1,n];
for i:=1 to n do
if a[i,n+1-i]<min then min:=a[i,n+1-i];
```

Примеры решения задач

Задача 1. Дан массив действительных чисел, состоящий из 3 строк и 5 столбцов. Вычислить произведение всех элементов массива.

```
Program  pr1;
CONST  N=3; M=5;
TYPE  MAS=array [1..N,1..M] of real;
Var b: MAS;
       i: 1..N;
       j : 1..M;
       p:real;
BEGIN
Writeln('Введите элементы массива');
For i:=1 to n do
   For j:=1 to m do
      Readln(b[i,j]);
{Вывод значений массива}
For i:=1 to n do
  begin
   For j:=1 to m do
     Write (b[i,j]); {Вывод элементов одной
                          строки матрицы}
      Writeln; { переход на следующую строку
                   экрана}
  end;
p:=1;
For i:=1 to n do
   For j:=1 to m do
    p:=p*b[i,j];
Writeln('Произведение = ',p:7:2);
end.
```

Задача 2. Дан двумерный массив A[N,M]
Сформировать массив B[N,M], где

$$B[I,J] = \begin{cases} SQR(A[I.J]), \text{ если } I\text{- НЕЧЕТНОЕ;} \\ SQRT(A[I.J]), \text{ если } I\text{- ЧЕТНОЕ;} \end{cases}$$

```
Program  pr3;
CONST  N=3; M=5;
TYPE  MAS=array [1..N,1..M] of real;
Var a,b :  MAS;
       i: 1..N;
       j : 1..M;
BEGIN
Writeln('Введите элементы массива');
For i:=1 to N do
   For j:=1 to M do
      Readln(a[i,j]);
{Вывод значений массива}
For i:=1 to N do
  begin
   For j:=1 to M do
     Write (a[i,j]:5:1);
      Writeln;
  end;
For i:=1 to N do
if  i/2= int(i/2) Then For j:=1 to M do
                          b[i,j]:=sqrt (a[i,j])
                  Else For j:=1 to M do
                          b[i,j]:=sqr (a[i,j])
For i:=1 to N do
  begin
   For j:=1 to M do
     Write (b[i,j]:7:2);
      Writeln;
  end;
end.
```

§17. Подпрограммы

В практике программирования часто встречаются задачи, в которых по ходу выполнения программы приходится производить одни и те же действия или вычисления, но при различных исходных данных. Чтобы исключить повторение одинаковых операторов и сделать тем самым программу проще и понятнее, можно выделить эти повторяющиеся действия в самостоятельную часть программы, которая может быть использована многократно по мере необходимости.

Автономная часть программы, реализующая определенный алгоритм и допускающая обращение к ней из различных частей общей программы, называется ***подпрограммой.***

Подпрограммы оформляются в виде замкнутых участков программы, имеющих четко обозначенные вход и выход.

Обращение к подпрограмме осуществляется из основной программы через заголовок подпрограммы. При вызове подпрограммы работа основной программы приостанавливается, и начинает выполняться вызванная подпрограмма. Она обрабатывает данные, переданные ей из основной программы, или просто выполняет заданную последовательность действий. По завершении выполнения подпрограммы основная программа продолжает выполнятся с того места, где прервалось ее действие.

Передача данных из основной программы в подпрограмму (входные данные) и возврат результата выполнения подпрограммы осуществляется с помощью параметров.

Параметры - *это данные, которые передаются вызываемой подпрограмме и используются последней в качестве входной и (или) выходной информации.*

Использование подпрограмм позволяет реализовать один из самых прогрессивных методов программирования - структурное программирование.

Процедура в Паскале и ее формат.

Любая программа может содержать несколько процедур и функций. Процедуры и функции объявляются в разделе описания вслед за пределом описания переменных.

Процедура - *это независимая часть программы, которую можно вызывать по имени для выполнения определенных действий.*

Структура процедуры имеет вид :

```
Procedure  имя(список формальных параметров);
  (*  раздел описаний *)
begin
   (* раздел операторов *)
end;
```

Первая строка описания называется ***заголовком процедуры***, а раздел операторов называется ***телом процедуры.***

В заголовке указывается служебное слово PROCEDURE, за которым следуют имя процедуры и список формальных параметров, заключенные в круглые скобки (если такие имеются). В списке перечисляются имена формальных параметров и их тип. Имя параметра отделяется от типа двоеточием, а параметры друг от друга - точкой с запятой. Если несколько формальных параметров имеют одинаковый тип, тогда их можно перечислить через запятую, а затем указать тип.

Тело процедуры заключается в операторные скобки BEGIN и END, причем после END ставится точка с запятой.

Раздел описаний процедуры подобен программе и состоит из разделов меток, констант, типов, переменных и , в свою очередь, процедур и функций.

Процедура вызывается по ее имени :

имя(список фактических параметров);

Формальные параметры - *параметры, определенные в заголовке процедуры.*

Фактические параметры - *выражения, задающие конкретные значения при обращении к процедуре.*

При обращении к процедуре ее формальные параметры замещаются фактическими, переданными из основной программы.

Фактические параметры - это параметры, которые передаются процедуре при ее вызове.

Количество и тип формальных и фактических параметров должны в точности совпадать.

Формальные параметры описываются в заголовке процедуры и определяют тип и место подстановки фактических параметров. Формальные параметры делятся на два вида: параметры-переменные и параметры-значения.

Параметры-переменные отличаются тем, что передними стоит служебное слово **Var**. Они используются тогда, когда необходимо, чтобы изменения значений формальных параметров в теле процедуры приводили к изменению соответствующих фактических параметров.

Параметры-значения отличаются тем, что перед ними слово **Var** не ставится. Внутри процедуры можно производить любые действия с параметрами-значениями, но все изменения никак не отражаются на значениях соответствующих фактических параметров, то есть какими они были до вызова процедуры, такими же и останутся после завершения ее работы.

Все переменные программы делятся на глобальные и локальные. *Глобальные переменные* объявляются в разделе описаний основной программы. *Локальные переменные* объявляются в процедурах и функциях. Таким образом, локальные переменные «живут» только во время работы подпрограммы.

Пример. Составить программу для вычисления a^n : целые числа a и n (n>=0) вводятся с клавиатуры. (составить процедуру для вычисления степени целого числа).

```
Program ex;
var a, n : integer;
```

```
      s: longint;
Procedure Degree(x,y : integer; var st : longint);
var i : integer;
begin
  st:=1;
  for i:=1 to y do st:=st*x;
end;
{ начало основной программы}
begin
  writeln('введите два числа - основание и показатель степени');
  readln(a,n);
  Degree(a,n,s); { обращение к процедуре }
  writeln('Результат ',s);
end.
```

Процедура названа именем Degree. В скобках записан список формальных параметров, то есть перечислены переменные с указанием их типа. Используем три параметра: первый - основание степени, то есть число, которое надо возвести в степень; второй - показатель степени, третий - результат. Первые два формальных параметра - параметры значения, третий - параметр-переменная, и перед ним указано слово **var.** Все они описаны как целые (x и y - переменные типа integer, st - Longint, так как степенная функция быстро возрастает).

После заголовка процедуры идут разделы описаний. В нашем примере имеется только раздел описания переменных, в котором описывается одна переменная i (счетчик цикла).

Далее идет тело процедуры. Оно начинается служебным словом **Begin** и заканчивается служебным словом **End,** после которого стоит точка с запятой (в конце программы после последнего End ставится точка). В теле процедуры вычисляется степень числа x с помощью цикла For.

В программе процедуры и функции описываются после раздела описания переменных программы, но до начала ее основной части, то есть до **Begin** , начинающего эту часть программы.

!!!! Процедура вызывается как оператор, состоящий из имени процедуры. В круглых скобках записываются фактические параметры.

В нашем примере формальные параметры x, y и st принимают значения фактических параметров a, n и s соответственно. После завершения работы процедуры переменные a и n сохранят те же значения, что и при вызове, а s получит новое значение.

Пример 2 : Используя процедуру для вычисления степени числа, найти значение выражения : $y=a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$

```
program ex2;
var a: array[0..4] of integer;
      i, x: integer;
      y,s: longint;
 Procedure Degree(xx,n : integer; var st : longint);
 var i : integer;
 begin
   st:=1;
   for i:=1 to n do st:=st*xx;
 end;
{ начало основной программы}
begin
 write('введите значение переменной x '); readln(x);
 writeln('введите массив коэффициентов');
  for i:=0 to 4 do begin write('a[',i,']='); readln(a[i]); end;
 y:=a[0];
for i:=1 to 4 do
begin
 Degree(x,i,s); y:=y+a[i]*s;
end;
writeln('y=',y);
end.
```

Пример 3. Просуммировать различные части массива.

```
Program sumir;
var a: array [1..100] of integer; sa1, sa2,sa3 : integer;
      n,l,t : integer;
procedure summa(a:array [1..100] of integer; k,m :integer; var
s:integer);
 var i:integer;
```

```
 begin
    s:=0;
   for i:=k to m do s:=s+a[i];
 end;
BEGIN
  for t:=1 to 100 do
   begin
    write('введите очередной элемент массива ');
    readln(a[i]);
   end;
 summa(a,10,20,sa1);
 summa(a, n , l , sa2);
 summa(a, n,n+3,sa3);
end.
```

Функции

Заголовок функции состоит из слова **Function**, за которым указывается имя функции, затем в круглых скобках записывается список формальных параметров, далее ставится двоеточие и указывается тип результата функции.

В теле функции обязательно должен быть хотя бы один оператор присваивания, в левой части которого стоит имя функции, а в правой - ее значение. Иначе значение функции не будет определено.

Таким образом, общий вид описания функции следующий :

Function *Имя[(список формальных параметров)]:Тип результата*

описательная часть

Begin

тело функции, в которой обязательно должно быть присваивание Имя_функции:=значение

End;

Пример 1 Составить программу, подсчитывающую число сочетаний без повторения из n элементов по k. Число сочетания без повторения вычисляется по формуле $C_n^k = \frac{n!}{k!(n-k)!}$

Обозначим через n и k переменные для хранения введенных чисел; C - переменную для хранения результата.

Воспользуемся функцией для вычисления факториала числа n. (n!=1*2*..*n)

```
program sochet;
var n,k : integer;
     a1,a2,a3,c : lohgint;
Function factorial(n:integer):longint;
  var i: integer;
       rez : longint;
 begin
   rez:=1;
   for i:=1 to n do rez:=rez*i;
   factorial:=rez;
  end;
begin
   writeln(‘ ввод n и k :’); readln(n,k);
   a1:=factorial(n); { вычисление n!}
   a2:=factorial(k); { вычисление k!}
   a3:=factorial(n-k); {вычисление (n-k)!}
   c:=a1 div (a2*a3); { результат}
  writeln(‘результат=’,c) ;
end.
```

Первая строка в описании функции - это ее заголовок. Служебное слово **Function** (функция) указывает на то, что именем factorial названа функция. В скобках записан список формальных параметров функции, состоящий из одной переменной целого типа. Далее в заголовке указан тип значения функции. В данном примере результат функции factorial - длинное целое число.

За заголовком функции следует описательная часть функции, которая, как у программы, может состоять из разделов описания

переменных, констант, типов. В данном примере есть переменные i (счетчик цикла) rez (для накопления значения факториала).

Далее идет раздел операторов (тело функции).

В тексте программы описания функций всегда следуют за разделом описания переменных и до начала основной части, как и описания процедур.

Пусть n=5, k=3. Когда в программе встречается оператор a1:=factorial(n), выполняются следующие действия:

- выделяется память для переменных, описанных в функции factorial;
- формальному параметру присваивается значение фактического: n:=n (n=5);
- выполняется функция, вычисляется факториал числа 5;
- значение функции передается в место обращения к этой функции, то есть присваивается переменной a1;
- в операторах a2:=factorial(k) a3:=factorial(n-k) еще дважды вызывается функция factorial с параметрами k=3 n-k=2.

Функция - это самостоятельная часть программы, имеющая собственные переменные, которым отводится *отдельное* место в памяти ЭВМ. Этим объясняется тот факт, что переменные с одинаковыми именами, используемые в функции и в основной программе, являются разными (переменная n основной программы и параметр n функции). При выполнении программы машина «не путает» имена этих переменных, т.к. области их действия не совпадают.

Пример 2 : Написать функцию, подсчитывающую количество цифр натурального числа. Используя ее, определить, в каком из двух данных чисел больше цифр.

```
Program chisla;
Var  n1, n2 : longint;
       k1, k2 : byte;
Function kol(x : longint): byte;
 var k: byte;
 begin
  k:=0;
  While x<>0 do
    begin
```

```
            Inc(k);
          x:=x div 10;
       end;
    kol:=k;
    end;
BEGIN
   writeln('Введите два числа'); readln(n1, n2);
   k1:=kol(n1);
   k2:=kol(n2);
   if k1=k2 Then writeln('одинаковое количество цифр')
    else  if  k1>k2 Then Writeln('в первом числе цифр больше')
           else writeln('во втором числе цифр больше')
END.
```

§18. Примеры рекурсивного программирования

Алгоритм, который в процессе работы обращается сам к себе, называется *рекурсивным.* Для иллюстрации понятия рекурсия часто приводят пример с телевизором, на экране которого изображен этот же телевизор, на экране второго - опять телевизор и так далее.

Можно разбить все рекурсивные задачи на 4 вида.

I. Задачи с рекурсивной формулировкой

Некоторые объекты являются рекурсивными по определению, поэтому рекурсивные алгоритмы их получения буквально повторяют соответствующие определения.

Пример : Вычисление факториала натурального числа.

$$N! = \begin{cases} 1 & \text{при } N=1 \\ N(N-1)! & \text{При } N>1 \end{cases}$$

```
Function factorial(n:integer): longint;
begin
  if n=1 Then factorial:=1
   else factorial:=n*factorial(n-1);
```

end;

Функция вызывается 5 раз. N=1 - это условие окончания рекурсии.

Задача 2. Написать рекурсивную функцию вычисления значений функции Аккермана для неотрицательных чисел n и m, вводимых с клавиатуры.

$$A(n,m) = \begin{cases} m+1, \text{ если } n=0 \\ A(n-1,1), \text{ если } n\neq 0,\ m=0 \\ A(n-1,A(n,m-1)), \text{ если } n>0,\ m\geq 0 \end{cases}$$

Задача 3. Найти первые N чисел Фибоначчи. Каждое число Фибоначчи, кроме первых двух, равно сумме двух предыдущих чисел, а первые два равны 1 (1, 1, 2, 3, 5, 8, 13, 21...)

$$\Phi(n) = \begin{cases} 1, \text{ если } n=1 \text{ или } n=2 \\ \Phi(n-1) + \Phi(n-2), \text{ если } n>2 \end{cases}$$

Задача 4. Найти сумму первых N членов арифметической (геометрической) прогрессии.

II. Задачи, из постановки которых можно извлечь рекурсию

В формулировках некоторых задач рекурсия не присутствует в явном виде, но их можно свети к рекурсивным.

Пример 1. Сложение двух чисел. Пусть надо сложить два целых числа a и b, а можно только прибавлять или вычитать 1. Тогда:

если b=0, то a+b=a

если b>0, то a+b=(a+1)+(b-1)

если b<0, то a+b=(a-1)+(b-1)

Это есть рекурсивное определение операций сложения двух чисел.

```
Function Sum(a, b : integer) : integer;
 begin
   if b=0 Then Sum:=a
    else  if b>0  Then Sum:=Sum(a+1, b-1)
            else Sum:=Sum(a-1, b+1);
 end;
```

Пример 2. Найти НОД двух натуральных чисел.(2 способ)
Имеются два натуральных числа a и b.
Если a=b, то НОД(a,b)=a.
Если a>b, то НОД(a,b)=НОД(a-b,b).
Если a<b, то НОД(a,b)=НОД(a,b-a).

```
Function NOD(a,b : integer) : integer;
 begin
   if a=b  Then  NOD:=a
    else  if a>b Then NOD:=NOD(a-b,b)
            else NOD:=NOD(a,b-a)   ;
 end;
```

a	b	Примечание
123	36	Так как a>b, a:=a-b
87	36	a:=a-b
51	36	a:=a-b
15	36	Так как b>a, b:=b-a
15	21	b:=b-a
15	6	a:=a-b
9	6	a:=a-b
3	6	b:=b-a
3	3	Так как a=b, НОД:=a

Пример 3. Перевести натуральное число из десятичной системы счисления в двоичную.

```
Function Rec(n: integer);
 begin
   if n>1 Then Rec(n div 2);
   Write(n Mod 2);
 end;
```

III. Задачи, которые можно решить как частный случай обобщенной

Если нельзя извлечь рекурсию из постановки задачи, то можно расширить задачу, обобщить ее (например, введя дополнительный параметр). Удачное обобщение может помочь увидеть рекурсию. После этого возвращаемся к частному случаю и решаем исходную задачу.

Пример 1 Определить, является ли заданное натуральное число простым.

Данную задачу можно обобщить, например, так: определить, верно ли, что заданное натуральное число N не делится ни на одно число, большее или равное M (2≤M≤N), но меньше N.

Соответствующая функция должна принимать значение «истина» в двух случаях:

- если M=N;
- если N не делится на M и функция принимает значение «истина» для чисел M+1 и N.

```
Function Simple(m,n : integer): boolean;
begin
  if m=n  Then Simple:=True
   else  Simple:=(n Mod m <> 0)  And  Simple(m+1,n);
end;
```

IV. Задачи, в которых можно использовать характеристику или свойство функции

Пример 1. Для заданного натурального числа *N≥1* определить натуральное число *a*, для которого выполняется неравенство :

$2^{a-1} \le N < 2^a$.

a(N) = 1 , если N=1

a(N)=a(N div 2) + 1, если N>1

Рассмотрим пример. Пусть N=34.

$2^{a-1} \le 34 < 2^a$, прибавим 1 и переходим к 34 div *2*

$2^{a-1} \le 17 < 2^a$ +1

$2^{a-1} \le 8 < 2^a$ +1

$2^{a-1} \le 4 < 2^a$ +1

$2^{a-1} \le 2 < 2^a$ +1

$2^{a-1} \le 1 < 2^a$; получим a=1

А теперь возвращаемся назад, к последней единице прибавляем все предыдущие. Таким образом, получается 6.

```
Function A(N : integer) : integer;
 begin
   if N=1  then  a:=1  else  a:=a (N div 2) +1;
 end;
```

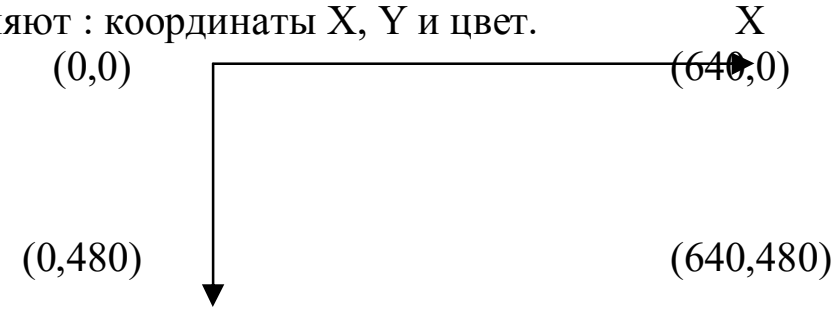
§19. Графика

В *графическом режиме* экран рассматривается как последовательность точек (пикселов), из которых строится изображение.

Количество пикселов в строке и количество строк на экране характеризуют его *разрешающую способность.*

640x480 - в строке 640 пикселов , а строк всего 480.

Пиксел определяют : координаты X, Y и цвет.



Для работы в графическом режиме разработана библиотека GRAPH, содержащая множество графических процедур и набор драйверов.

Драйвер находится в файле с расширением .BGI

Для того, чтобы в программе можно было использовать процедуры модуля GRAPH, его надо подключить к программе, использовав раздел описаний модулей:

USES GRAPH;

С момента подключения модуля GRAPH программисту доступны все находящиеся в ней подпрограммы.

В первую очередь вызывается процедура InitGraph, которая устанавливает один из возможных графических режимов.

Формат процедуры :

InitGraph(Driver, Mode,'c:\tp7\bgi');

В программе нужно описать переменные Driver и Mode :

var Driver, Mode : integer;

В разделе операторов, перед тем как написать первый графический оператор, следует выполнить :

Driver:=Detect; InitGraph(Driver, Mode,'c:\tp7\bgi');

Перед окончанием программы следует закрыть видеорежим с помощью процедуры CloseGraph.

Установка цвета и стиля заполнения

SetColor(<** константа определяющая цвет**>:word); **-** установка цвета графического изображения.

SetBkColor(<**константа определяющая цвет фона**>:word); **-** установка цвета фона;

SetFillStyle(<**константа стиля заполнения**>:word; **<**константа цвета заполнения**>:word);*** **-** установка способа закраски.

Таблица цветов

Константа		**Цвет**
Имя	**Значение**	
Black	0	Черный
Blue	1	Синий
Green	2	Зеленый
Cyan	3	Бирюзовый
Red	4	красный
Magenta	5	малиновый
Brown	6	коричневый
LightGray	7	светло-серый
DarkGray	8	темно-серый
LightBlue	9	ярко-голубой

LightGreen	10	ярко-зеленый
LightCyan	11	ярко-бирюзовый
LightRed	12	ярко-красный
LightMagenta	13	ярко-малиновый
Yellow	14	желтый
White	15	белый

Таблица констант для стандартных стилей заполнения.

Константа		**Стиль заполнения**
Имя	**Значение**	
EmptyFill	0	заполнение цветом фона
SolidFill	1	заполнение текущим цветом
LineFill	2	Заполнение символами ---
LtslashFill	3	заполнение символами // нормальной толщ.
SlashFill	4	заполнение символами // удвоенной толщ.
BkslashFill	5	заполнение символами \\ удвоенной толщ.
LtbkSlashFill	6	заполнение символами \\ нормальной толщ.
HatchFill	7	заполнение вертикально-горизонтальной штриховкой тонкими линиями
XhatchFill	8	заполнение штриховкой крест-накрест по диагонали «редкими» тонкими линиями
InterLeaveFil	9	заполнение штриховкой крест-накрест по диагонали «частыми» тонкими линиями
WideDotFill	10	заполнение «редкими» точками
CloseDotFill	11	заполнение «частыми» точками

Процедуры создания графических примитивов

1. Текущий указатель.

При построении изображения иногда надо указать точку начала вывода. В текстовом режимах эту точку указывает курсор, который присутствует на экране (его можно убрать). В графическом режимах видимого курсора нет, но есть невидимый текущий указатель. Для перемещения текущего указателя по экрану дисплея служит процедура ***MoveTo(x,y)*** - перемешает указатель в точку с координатами (x,y).

MoveRel(dx, dy) - перемещает указатель на dx точек по горизонтали и dy точек по вертикали от предыдущей позиции.

Примеры

```
1.   ..........
 MoveTo(200,100);
 MoveRel(5,10); {указатель переместится в точку (205,110)}
..........
```

Чтобы определить максимальное значение координат X Y для установленного видеорежима, используют функции

GetMaxX : integer; максимум по X

GetMaxy : integer; максимум по Y

Установить указатель в центр экрана.

```
  .........
 var  Xcentr, Ycentr : integer;
   ..........
 begin
   ..........
  Xcentr:=GetMaxX  div  2;
  Ycentr:=GetMaxy  div  2;
  MoveTo(Xcentr, Ycentr);
   ..............
```

2. Вывод точки

PutPixel(x,y : integer; <*цвет точки****>);***

где x,y координаты точки.

3. Вывод отрезка
Line(x1,y1,x2,y2);
(x1,y1) - координаты начала отрезка
(x2,y2) - координаты конца отрезка

!!! Обратите внимание на то, что в процедуре не задается цвет. В этом и аналогичных случаях цвет определяется процедурой SetColor().

LineTo(x,y) **-** строит отрезок из точки текущего положения указателя в точку с координатами (x,y).

LineRel(dx,dy) - строит отрезок из точки текущего положения указателя в точку с координатами (x+dx, y+dy)

4. Построение прямоугольника
Rectangle(x1,y1,x2,y2:integer);

Bar(x1,y1,x2,y2:integer) **-** рисует прямоугольник и закрашивает его цветом и стилем, определенным в процедуре SetFillStyle().

5. Построение дуг, окружностей, эллипсов.
Circle(x,y,<радиус> : word); **-** окружность указанного радиуса
Ellipse(x,y:integer; <нач_угол>,<кон_угол> ***:word; xR,yR : word)*** **-** построение эллиптических дуг.
X,Y - координаты центра,
xR, yR - длина горизонтальной и вертикальной полуосей в пикселах.
Угол отсчитывается против часовой стрелки и указывается в градусах. Дуга эллипса вычерчивается от заданного начального угла до конечного угла. Если значение начального угла 0, а конечного 360 - будет построен полный эллипс.

6. Построение закрашенного эллипса:
FillEllipse(x,y:integer; xR,yR);
X,Y - координаты центра,

xR, yR - длина горизонтальной и вертикальной полуосей в пикселах.

Стиль заполнения области внутри эллипса устанавливается процедурой SetFillStyle(), а самого эллипса - SetColor().

7. Заполнение внутренней или внешней области замкнутой фигуры.

FloodFill(x,y:integer; <*цвет границы области****>);***

Стиль задан SetFillStyle()

X,Y - координаты точки внутри (или вне) замкнутой области.

Задача. Построить в центре экрана синий прямоугольник, закрасив его линиями вида \\ темно-серого цвета. Фон экрана сделать белым.

```
Program graph1;
uses Graph;
var Driver, Mode : integer;
begin
Driver:=Detect;                  {инициализация графического}
InitGraph(Driver, Mode,'')       {    режима}
SetBkColor(15);                  { установка цвета фона - белый}
SetColor(1);                     {установка текущего цвета - синего}
Cleardevice;            {очистка экрана установленным цветом фона}
SetFillStyle(5,8);       {установка стиля заполнения}
Rectangle(290,290, GetMaxX-290, GetMaxy-290);   {прямоугольник}
FloodFill(301,230,1);                    {заполнение прямоугольника
выбранным
                                  стилем}
ReadLn;
CloseGraph;
end.
```

Задача 2. Построить график y=sinx по оси y в диапазоне -100,100 y=100*sinx, 0<x<2pi, длина шага Dx=pi/100.

```
Program sinus;
uses crt, graph;
var  gt, gr : integer;
     xm,ym,i : integer;
```

```
            dx,x1,y1,x2,y2:real;
    begin
      gt:=detect;  initgraph(gt,gr,'');
      xm:=getmaxx div 2;
      ym:=getmaxy div 2;
    setcolor(3);
    line(0,ym-100,0,yn+100);      line(0,ym.getmaxx,ym);        {построение
осей}
    outtextxy(100,ym+50,'pi');  {вывод текста рядом с осями}
    moveto(0,ym);
    dx:=oi/100;
    x1:=0;
    setcolor(5);
     while x1<=200 do begin
                          y1:=100*sin(x1*dx);
                          x2:=x1+dx;
                          y2:=100*sin(x2*dx);
            line(round(x1),ym+round(y1),round(x2), ym+round(y2));
                          x1:=x2;
                          y1:=y2
                          end;
    readln;
    closegraph;
    end.
```

Материалы для дополнительного чтения.

§20. Файловый тип данных

В задачах, которые мы рассматривали, данные поступали с клавиатуры, а результаты выводились на экран дисплея. Поэтому ни исходные данные, ни результаты не сохранялись. Приходилось заново вводить данные всякий раз, когда запускали программу. А если их очень много ? Тогда удобно оформить исходные данные и результаты в виде файлов, которые можно хранить на диске точно так же, как и программы.

Файл - это область памяти на внешнем носителе, в которой хранится некоторая информация. В языке Паскаль файл представляет собой последовательность элементов одного типа. Мы будем работать только с ***файлами последовательного доступа.*** В таких файлах, чтобы получить доступ к элементу, необходимо ***последовательно*** просмотреть все предыдущие.

Объявление файловой переменной в разделе описания переменных имеет вид :

var <имя файла>**:** File of <тип элементов>;

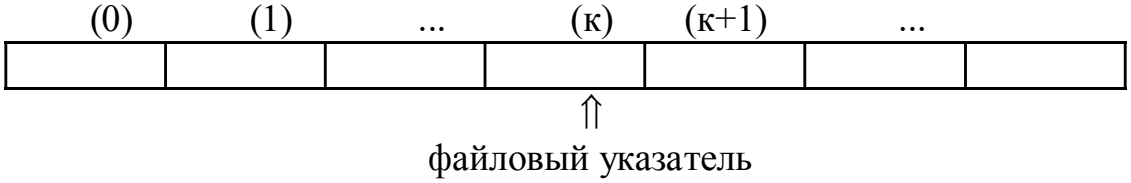
Например:

```
var  Ft : File of integer;
     M : File of char;

Type File_integer=File of integer
     File_char=File of char;
Var  F1: File_integer;
     F2: File_char;
```

Так как в описании указывается тип элементов, такие файлы называются ***типизированными.*** Все элементы файла пронумерованы начиная с нуля.

С каждым файлом связан так называемый ***файловый указатель***. Это неявно описанная переменная, которая указывает на некоторый элемент файла.



Все операции производятся с элементом, который определен файловым указателем.

Связь переменной файлового типа с файлом на диске.

Процедура ***Assign(***<*имя файловой пер-ой*>***,'***<*имяфайла на диске*>***');***

Например:

Assign(F1,'A:INT.DAT');

После установления такого соответствия все операции, выполняемые над переменной F1, будут выполнятся над файлом, хранящимся на диске A и имеющим имя INT.DAT

Файл в каждый момент времени может находиться в одном из двух состояний: либо он открыт только для записи, либо только для чтения.

Чтение из файла.

Под чтением из файла понимается пересылка данных из внешнего файла, находящегося на диске, в оперативную память.

Для чтения из файла необходимо открыть для чтения посредством процедуры

Reset(<*имя файловой переменной*>***);***

Собственно чтение данных из файла выполняется процедурой

Read(<*имя файловой переменной*>***,***<*имя переменной*>***);***

Переменная должна иметь тот же тип, что и компоненты файла. Отметим, что если оператор ввода имеет вид Read(<имя переменной>), то данные вводятся с клавиатурой, а если Read(<имя файловой переменной>,<имя переменной>); то данные вводятся из файла, хранящегося на диске.

Закрытие файла

После того как данные из файла прочитаны, его необходимо закрыть посредством процедуры

Close(< имя файловой переменной>)

Общая схема чтения данных из файла, таким образом, следующая:

Reset(<имя файловой переменной>);

.....
Read(<имя файловой переменной>,<имя переменной>);
.....
Close(<имя файловой переменной>);

Признак конца файла

Так как число элементов файла не известно заранее, необходимо уметь определять, что файл кончился. Для этого используется логическая функция ***Eof(<имя файловой переменной>)*** (Eof - End Of File). Она принимает истинное значение (True), если достигнут конец файла, и ложное (False) - в противном случае.

Пример Прочитаем из файла целые числа и выведем их на экран:

```
Assign(F1,'A:INT.DAT');
Reset(F1);
While Not Eof(F1) do
 begin
   read(f1,n); { считываем очередное число из файла}
   write(n,' '); { выводим его на экран}
 end;
Close(F1);
```

Запись в файл

Под записью в файл понимается вывод результатов программы из оперативной памяти ЭВМ в файл на диске.

Для записи в файл необходимо открыть файл для записи посредством процедуры

Rewrite(< Имя файловой переменной >);

Собственно запись данных в файл выполняется процедурой :

Write(<имя файловой переменной>,<значение>);

Общая схема записи данных в файл, таким образом, следующая:

Rewrite(<>);
.....
Write(<имя файловой переменной>,<значение>);
.....

Close(<имя файловой переменной>);

Прямой доступ к элементам файла

Несмотря на то, что в стандартном Паскале имеются лишь файлы последовательного доступа, Турбо Паскаль содержит процедуры и функции для более эффективной работы с файлами. В частности, имеется возможность осуществлять прямой доступ к элементам файла.

Установка указателя.

Процедура ***Seek(<имя файловой переменной>,N)*** устанавливает файловый указатель на N-й элемент. Например, Seek(F1,3). (на 4 элемент)

Определение номера элемента

Функция ***FilePos(<имя файловой переменной>)*** возвращает номер элемента, на который «смотрит» файловый указатель.

Определение количества элементов в файле

Функция ***FileSize(<имя файловой переменной>)*** возвращает количество элементов в файле.

Удаление и переименование файлов

Erase(<имя файловой переменной>) процедура удаления файла.

Rename(<имя файловой переменной>,'<новое имя на диске>') переименование файла.

Пример : В файле DAT1.DAT записаны целые числа. Вычислить сумму элементов файла и результат вместе с исходными данными записать в файл DAN2.DAT

```
Program WW;
Var f1,f2 : file of integer;
      s, n : integer;
begin
  Assign(f1,'DAT1.DAT');
  Reset(F1);
  Assign(f2,'DAT2.DAT');
  Rewrite(f2);
  s:=0;
  While Not Eof(f1) do  { проверка на конец файла}
   begin
```

```
    read(f1,n); {чтение элемента из файла F1}
    write(f2,n);  { запись элемента в файл F2}
   s:=s+n;
 end;
write(f2,s); {запись суммы элементов в конец файла F2}
write('Результат находится в файле DAT2.DAT');
Close(f1);
Close(f2);
end.
```

§21. Текстовые файлы

Текстовые файлы состоят из символьных строк. Строки могут иметь различную длину, и в конце каждой строки стоит признак конца строки. Для описания текстовых файлов используется служебное слово Text:

Var A: Text;

Для обработки текстовых файлов используются те же процедуры и функции, что и для обработки обычных типизированных файлов. Для связывания файловой переменной с файлом на диске употребляется процедура Assign. Текстовые файлы могут быть открыть для чтения процедурой Reset или для записи процедурой Rewrite.

Для чтения данных применяется процедура Read. Если необходимо после чтения данных перейти на следующую строку, то используется процедура Readln. Если необходимо просто перейти к следующей строке, то можно использовать процедуру Readln(<имя файловой переменной текстового файла>); которая устанавливает файловый указатель на первый элемент следующей строки.

Процедура Write записывает данные в текущую строку. Если надо записать данные и перейти к следующей строке, то можно использовать процедуру Writeln. Если требуется только перейти для записи на новую строку, то применяется процедура Writeln(<имя файловой переменной текстового файла>); которая записывает в файл признак конца строки и устанавливает файловый указатель на начало следующей строки.

Так как в строках может быть разное количество символов, имеется логическая функция ***Eoln(<имя файловой переменной текстового файла>)***, которая принимает значение True, если достигнут конец строки.

Процедура ***Append(<имя файловой переменной текстового файла>).*** Она открывает файл для «дозаписи», помещая файловый указатель в конец файла.

Пример : Дан текстовый файл, содержащий только целые числа, в каждой строке может быть несколько чисел, которые разделяются пробелами. Вывести на экран все числа с учетом разбиения на строки и подсчитать количество элементов в каждой строке.

Решение: Пусть в файле содержится следующая информация:

-32 16 0 8 7
4 5 9 13 11 -5 -8
6 -8 0 -12
5 4 3 2 1 12
1 2

Этот файл можно создать в среде Турбо Паскаль следующим образом:

- создайте новый файл посредством команды **New** меню **File;**
- запишите все числа, разделяя их пробелами, и разбейте на строки, как указано в задании;
- сохраните файл, например, под именем INT1.DAT
- теперь напишем программу

```
program rrr;
var f : text;
   x, k: integer;
begin
  Assign(f,'int1.dat');
  Reset(f);
  While Not Eof(f) do {пока не достигнут конец файла}
     begin
       k:=0;
       While Not Eoln(f) do  {пока не достигнут конец строки}
       begin
         read(f,x);  {считываем очередное число}
         write(x,' ');  {выводим его на экран}
         Inc(k); {увеличиваем счетчик}
       end;
     writeln('в строке', k, '  элементов');
     Readln(f)   {переходим к следующей строке файла}
  end;
Close(f);
end.
```

Пример. Записать двумерный массив вещественных чисел 5x4 в тестовый файл.

```
Program mas;
const m=5; n=4;
Var fil : text;
      a: real;
      s: char;
     i,j : integer;
begin
   Assign(fil,'massiv.txt');
   Rewrite(fil);
   for i:=1 to m do
    begin
     for j:=1 to n do
       begin
          a:=random(100);
          write(fil,a:5:3,' '); {число записывается в файл в указанном формате, за ним пробел}
        end;
          writeln(fil);  {переход в файле на новую строку}
      end;
    Close(fil);
{Чтение файла и вывод матрицы на экран по строкам}
Reset(fil);   {открытие уже имеющегося файла}
 while not Eof(fil) do
   begin
      while not Eoln(fil) do
        begin
          read(fil,a); {чтение числа}
          write(a:5:3);
          read(fil,s);  { чтение пробела после числа}
           write(s);
        end;
      writeln;
      readln(fil);
  end;
Close(fil);
end.
```

Пример. Дан текстовый файл f. Переписать в файл g все компоненты исходного файла f в обратном порядке.

```
program tofile;
var f, g : text;
```

```
        n, i, j : integer;
    s : string;
    x : array [1..32000] of char;
begin
  assign(f,'f.txt');   assign(g,'g.txt');
   rewrite(g);  rewrite(f);
   writeln('Введите число строк в создаваемом вами файле ');
   readln(n);
   writeln('вводите строки, после введения каждой нажмите Enter');
   for i:=1 to n do begin readln(s); write(f,s);  end;
    reset(f);
   i:=0;
   writeln('Исходный файл :');
    while(not eof(f)) and (i<32000) do
      begin i:=i+1; read(f,x[i]); write(x[i]); end;
    writeln;
   writeln('Измененный файл :');
    for j:=i downto 1 do
     begin write(g,x[j]); write(x[j]);  end;
    writeln;
  close(f);  close(g);
end.
```

Задача. Дан текстовый файл. Вставить в начало каждой строки ее номер и записать преобразованные строки в новый файл.

Задача. Даны два текстовых файла. Записать в третий файл только те строки, которые есть и в первом, и во втором файлах.

§22. Множества

Множество в Паскале представляет собой набор различных элементов одного (базового) типа.

Базовый тип - это совокупность всех возможных элементов множества. Всего в базовом типе должно быть не более 256 различных элементов. Значение переменной множественного типа может содержать любое количество различных элементов базового типа - от нуля элементов (пустое множество) до всех возможных значений базового типа

Множества, используемые в программе, могут быть описаны либо в разделе **Type:**

Type <имя типа> = Set Of <тип элементов>;
Var <имя множества> : <имя типа>;

Либо непосредственно в разделе описания переменных **Var:**

Var <имя множества> : Set Of <тип элементов>;

Пример.

```
Type mnog_Char=Set Of Char;
Var  mn1 : Set Of Char;
     mn2 : mnog_Char;
     mn3 : Set Of 'A'..'Z';
      s1 : Set Of Byte;
      s2 : Set Of 1000..1200;
```

Здесь mn1 и mn2 - это множества символов; так как различных символов всего 256, то тип Char можно использовать в качестве базового;

mn3 - множество больших латинских букв;

s1 - множество целых чисел (от 0 до 255); так как тип Byte содержит только целые числа от 0 до 255, его тоже можно использовать в качестве базового типа элементов;

s2 - множество целых чисел от 1000 до 1200.

Формирование (конструирование) множеств. В программе элементы множества задаются в квадратных скобках, через запятую. Если элементы идут подряд друг за другом, то можно использовать диапазон.

Пример

```
Type digit = Set Of 1..5;
Var  s : digit;
```

Переменная s может принимать значения, состоящие из любой совокупности целых чисел от 1 до 5;

[ ] - пустое множество;

[1], [2], [3], [4], [5] - одноэлементные множества;

[1,2], [1,3], ..., [2,4], [4,5] - двухэлементные множества (пара любых элементов);

[1,2,3], [1,2,4], ... , [3,4,5] - трехэлементные (тройка элементов);

[1,2,3,4], [1,2,3,5], [1,2,4,5], [1,3,4,5], [2,3,4,5] - четырехэлементные;

[1,2,3,4,5] - полное множество (взяты все элементы базового типа).

Операции над множествами

Объединение двух данных множеств называется множество элементов, принадлежащих хотя бы одному из этих множеств. Знак операции объединения множеств - «+».

Примеры

1) [‘A’, ‘F’] + [‘B’, ‘D’] = [‘A’, ‘F’, ‘B’, ‘D’];
2) [1..3, 5, 7, 11] + [3..8, 10, 12, 15..20] = [1..8, 10..12, 15..20]

Пусть S1:=[1..5, 9], а S2:=[3..7, 12]. Тогда если S:=S1 + S2, то S=[1..7, 9, 12].

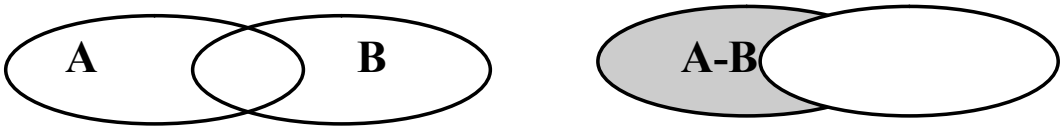
Пусть A1:=[‘a’..’z’]; A1:=A1 = [‘A’]. Тогда A1=[‘A’, ‘a’..’z’].

Пересечением двух множеств называется множество элементов, принадлежащих одновременно и первому, и второму множеству. Знак операции пересечения - «*».

Примеры.

1) [‘A’, ‘F’] * [‘B’, ‘D’] = [ ] , так как общих элементов нет;
2) [1..3, 5, 7, 11] * [3..8, 10, 12, 15..20] = [3, 5, 7];
3) если S1:=[1..5, 9] и S2:=[3..7, 12], а S:=S1 * S2, то S=[3..5].

Разностью двух множеств называется множество, состоящее из тех элементов первого множества, которые не являются элементами второго. Знак операций вычитания множеств - «—»



Примеры.

[‘A’, ‘F’] - [‘B’, ‘D’] = [‘A’, ’F’] , так как общих элементов нет;
[1..3, 5, 7, 11] - [3..8, 10, 12, 15..20] = [1, 2, 11];
если S1:=[1..5, 9] и S2:=[3..7, 12], а S:=S1 - S2, то S=[1, 2, 9];
A1:=[‘A’..’Z’]; A1:=A1 - [‘A’]. Тогда A1=[‘B’..’Z’].

Операция определения принадлежности элемента множеству

Эта логическая операция обозначается служебным словом ***in.*** Результат операции имеет значение ***true,*** если элемент входит в множество, и ***false*** в противном случае.

Примеры.

1. Выражение 5 in [3..7] имеет значение true, так как 5 ∈ [3;7]
2. Выражение ‘a’ in [‘A’..’Z’] имеет значение false, так как маленькой латинской буквы «a» нет среди больших латинских букв.

Сравнение множеств

Для сравнения множеств используются операции отношения:

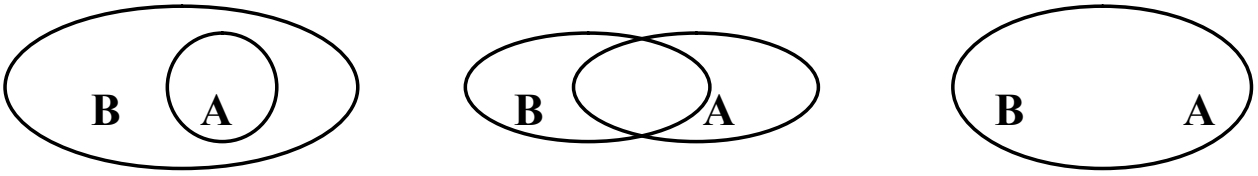
= - проверка на равенство (совпадение) двух множеств;

<> - проверка на неравенство двух множеств;

<=, < - проверка на вхождение первого множества во второе множество;

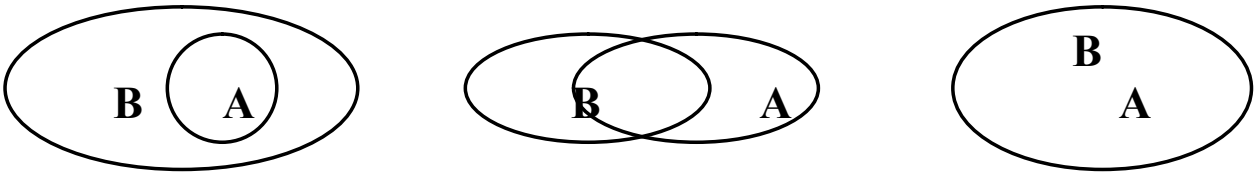
>=, > - проверка на вхождение второго множества в первое множество.

Первое множество меньше или равно второму (A≤B)



true **false** **true**

Первое множество меньше второго (A<B)



true **false** **false**

Пример. Составить программу выделения следующих множеств из множества целых чисел от 1 до 30 :

- множества чисел, кратных 2;

- множества чисел, кратных 3;
- множества чисел, кратных 6;
- множества чисел, кратных 2 или 3;

Вопросы для обсуждения :

1. Сколько множеств надо описать? (Каков тип их элементов? (четыре множества с элементами типа Byte).

2. Каково начальное значение множеств ? (начальное значение множеств - пустое множество).

3. Как формируются множества? (Первые два формируются перебором всех чисел данного промежутка и отбором подходящих, а третье и четвертое получаются из первых двух путем применения операции пересечения или объединения).

4. Как осуществить вывод сформированных множеств? (Вывод множеств производится ТОЛЬКО поэлементно, поэтому удобно составить процедуру и передавать в нее множество, элементы которого и будут выводить на экран. Для этого в разделе типов надо создать соответствующий тип и использовать его в дальнейшем).

```
Program mnog;
const   n=30;
type   mn=Set Of 1..n;
var  n2, n3, n6, n23 : mn;
        k : integer;
procedure print(m: mn);
 var i: integer;
  begin
     for i:=1 to n do   if i In m  then write(i:3);
     writeln;
  end;
BEGIN
 n2=[ ]; n3=[ ];
 for k:=1 to n do
   begin
     {если число делится на 2, то заносим его в n2 }
    if k Mod 2 = 0  Then  n2:=n2+[k];
     {если число делится на 3, то заносим его в n3 }
  if k Mod 3 = 0  Then  n3:=n3+[k];
 end;
```

{числа, кратные 6, - это те, которые кратны и 2 и 3, поэтому это пересечение двух множеств, а числа, кратные 2 или 3, - это объединение этих же множеств}

```
n6:=n2*n3;  n23:=n2+n3;
writeln('числа, кратные 2');   print(n2);
writeln('числа, кратные 3');   print(n3);
writeln('числа, кратные 6');   print(n6);
writeln('числа, кратные 2 или 3');   print(n23);
END.
```

Пример 2. Если взять то общее, что есть у боба с ложкой, добавить кота и поместить в тепло, то получится муравей. Так ли это ?

```
program bob;
var  y1, y2, y3, y4, x : Set Of Char;
        s : Char;
BEGIN
  y1:=['б', 'о', 'б'];    y2:=['л', 'о', 'ж', 'к', 'а'];
  y3:=['к', 'о', 'т'];     y4:=['т', 'е', 'п', 'л', 'о'];
  x:=(y1*y2) + y3 - y4;
 writeln('множество x');
 for s:='a'  to  'я'  do  if s In x  then write(s); writeln;
 {проверка: состоит ли муравей из кота}
 if y3<=x  then  write('муравей состоит из кота')
 else write('муравей не состоит из кота');
END.
```

Пример 3. Дан ребус МУХА + МУХА = СЛОН

Каждой букве соответствует некоторая цифра, разным буквам соответствуют разные цифры. Необходимо заменить буквы цифрами так, чтобы получить верное равенство.

Для решения этой задачи применим метод перебора с возвратом. Используем множество S1 для хранения цифр слова МУХА, причем будем заносить в него цифры последовательно, учитывая уже внесенные цифры. Начальное значение S1 - пустое множество. После выбора всех цифр первого слова формируем соответствующее число и находим число, соответствующее слову СЛОН. Выделяем цифры СЛОНа (множество S2), и если слова состоят из различных цифр (то есть пересечение S1 и S2 пустое) и все цифры СЛОНа разные, то выводим решение на экран. Далее удаляем из множества S1 последнюю внесенную цифру и пытаемся выбрать еще одно ее значение. Таким образом, мы перебираем все возможные варианты и выводим на экран только те, которые удовлетворяют равенству.

Заметим, что букве «М» в слове МУХА может соответствовать цифра от 1 до 4, а букве «А» в этом же слове не может соответствовать 0.

```
      Program муха;
  type mn=Set Of  0..9;
  var m, y, x, a : 0..9;  {цифры числа МУХА}
      n1, n2 : integer;  {числа МУХА  и СЛОН}
      a1, a2, a3, a4 : 0..9;   {цифры числа СЛОН}
      S1, S2 :mn;
  procedure  Print(x,y:integer); {вывод решения в виде ребуса }
   begin
     writeln(x:5);
     writeln('+');
     writeln(x:5);
     writeln('____');
     writeln(y:5);
   end;
  BEGIN
    s1:=[ ];  s2:= [ ];
    for  m:=1 to 4 do
     begin
       s1:=s1+[m];   { заносим первую использованную цифру}
       for  y:=0 to 9 do  {если это цифра не была еще взята, то добавляем ее
во множество цифр числа МУХА и выбираем цифру для следующей буквы }
          if Not (y In s1)
          then  begin s1:=s1+[y];
                      for x:=0 to 9 do
                       if Not (x In s1)
                         then begin  s1:=s1+x;
                                for a:=1 to 9 do
                                 if Not (a In s1)
                                  then begin s1:=s1+[a];
                                             n1:=1000*m+100*y+10*x+a;
                                             n2:=2*n1;
                                             a1:=n2 div 1000;
                                             a2:=n2 div 100 mod 10;
                                             a3:=n2 div 10 mod 10;
                                             a4:=n2 mod 10;
                                             s2:=[a1,a2,a3,a4];
                                       if (s1*s2=[]) and ([a1]*[a2]*[a3]*a[4]=[])
                                        then Print(n1,n2);
                                        s1:=s1-[a];
     end;
  s1:=s1-[x];
```

```
end;
s1:=s1-[y];
end;
s1:=s1-[m];
end;
END.
```

§23. Комбинированный тип данных (записи)

При использовании массивов основное ограничение заключается в том, что все элементы должны иметь один и тот же тип. Но при решении многих задач возникает необходимость хранить и обрабатывать совокупность данных различного типа.

Пусть известны фамилии и оценки (в баллах) по пяти дисциплинам каждого из двадцати пяти учеников класса. Требуется вычислить среднюю оценку каждого из учеников и выбрать человека, имеющего максимальный средний балл.

В данном случае фамилия может быть представлена символьной строкой, оценки - это целые числа, а средний балл должен быть представлен вещественным числом. В Паскале для описания объектов, содержащих данные разных типов, используются записи.

Запись - это структурированный тип, описывающий набор данных разных типов. Составляющие запись объекты называются ее полями. Каждое поле имеет уникальное (в пределах записи) имя. Чтобы описать запись, необходимо указать ее имя, имена объектов, составляющих запись, и их типы.

Общий вид описания записи :

Type *<имя записи>* = ***Record***
<поле 1> : <тип 1>;
<поле 2> : <тип 2>
<поле n> : <тип n>
End;

Применительно к рассматриваемой задаче запись можно описать так :

```
Type
  pupil = Record
  fam : string[15];          {поле фамилии}
  b1,b2,b3,b4,b5 : 2..5;     {поля баллов}
  sb : Real                  {средний бал}
```

End;

Чтобы хранить в памяти ЭВМ информацию обо всех 25 учениках класса, необходимо ввести массив klass - массив записей.

Var klass : array[1..25] Of pupil;

Доступ к полям записи можно осуществить двумя способами:

1. ***С указанием имени переменной и имени поля.*** Например, klass[2].fam, klass[3].sb, klass[1].b4

Ввод фамилий и оценок учащихся, т.е. элементов массива klass, можно записать так:

```
for i:=1 to 25 do
 begin
   readln(klass[i].fam);
   readln(klass[i].b1);
   readln(klass[i].b2);
   readln(klass[i].b3);
   readln(klass[i].b4);
   readln(klass[i].b5);
end;
```

2. ***С использованием оператора присоединения.***

Имеется возможность осуществлять доступ к полям записи таким образом, как если бы они были простыми переменными. Общий вид оператора присоединения:

With *<имя записи>* ***Do*** *<оператор>****;***

Внутри оператора присоединения к компонентам записи можно обращаться только с помощью имени соответствующего поля.

Пример

```
for i:=1 to 25 do
With klass[i] do
  begin
    readln(fam);
    readln(b1,b2,b3,b4,b5);
  end;
```

Программа для решения рассматриваемой задачи может быть записана следующим образом:

```
program zap;
Type pupil = Record
     fam : string[15];          {поле фамилии}
```

```
    b1,b2,b3,b4,b5 : 2..5;    {поля баллов}
    sb : Real                 {средний бал}
 End;
 Var  klass : array[1..25] Of pupil;
        p: pupl;
        i,m : integer;
       sbmax : real;
begin
  for i:=1 to 25 do
  with klass[i] do
   begin
    writeln('Введите фамилию и пять оценок');
    readln(fam);
    readln(b1,b2,b3,b4,b5);
   end;
  for i:=1 to m do   {вычисление среднего балла}
   with klass[i] do sb:=(b1+b2+b3+b4+b5)/5;
   sbmax:=0;
   { поиск максимального среднего балла}
   for i:=1 to m do
    if klass[i].sb>=sbmax  then  sbmax:=klass[i].sb;
   for i:=1 to m  do  {печать результатов}
    if klass[i].sb=sbmax
      then  with klass[i] do writeln(fam:20,' - ', sb:6:3);
 end.
```

Пример. Определите дату завтрашнего дня.

Чтобы определить дату завтрашнего дня, надо знать не только дату сегодняшнего дня, но и количество дней в данном месяце (так как если это последний день месяца, то завтра будет первый день следующего), кроме того, надо знать, какой год - високосный или нет.

Пусть дата вводится в формате *число - месяц - год* следующим образом:

1 2 1997

Опишем запись для хранения даты таким образом:

```
Type year=1500..2000;
   month=1..12;
    day=1..31;
    data = Record
     y : year;
     m : month;
     d : day;
```

```
                end;
```

Заметим, что :

• если дата соответствует не последнему дню месяца, то год и месяц не изменяются, а число увеличивается на 1;

• если дата соответствует последнему дню месяца, то :

а) если месяц не декабрь, то год не изменяется, месяц увеличивается на 1, а число устанавливается в 1;

б) если месяц - декабрь, то год увеличивается на 1, а месяц и число устанавливаются в 1.

```
Program datanext;
Type year=1500..2000;
   month=1..12;
    day=1..31;
    data = Record
      y : year;
      m : month;
      d : day;
           end;
Var dat, next : data;
Function leap(yy:year):boolean; {функция определяет
                                          високосный ли год}
 begin
  leap:=(yy Mod 4=0) And (yy Mod 400 <>0);
 end;
Function Dmonth(mm:month; yy : year) : day;  {функция определения
                                 количества дней данного месяца в данном году}
 begin
   case  mm  of
    1,3,5,7,8,10,12: Dmonth:=31;
     4,6,9,11 : Dmonth:=30;
     2 : if leap(yy) then Dmonth:=29  else Dmonth:=28;
    end;
  end;
procedure Tomorrow(td : data; var nd : data); {опр-ние завтрашней даты}
 begin   {если это не последний день месяца}
  if   td.d<> Dmonth(td.m, td.y)     then
    with  nd  do
    begin
       d:=td.d+1;
       m:=td.m;
       y:=td.y;
```

```
      end;
    else     {если это последний день месяца}
      if  td.m=12  then    {если это декабрь}
      with  nd do
       begin
         d:=1;
         m:=1;
         y:=td.y+1;
       end;
      else  { если  это не декабрь}
       with  nd  do
       begin
          d:=1;
          m:=td.m+1;
          y:=td.y;
       end;
  end;
BEGIN
  writeln('Введите сегодняшнее число, месяц и год');
  readln(dat.d, dat.m, dat.y);
  Tomorrow(dat,next);
  writeln('завтра будет');
  writeln(next.d, '.', next.m, '.', next.y);
END.
```

§24. Приближенные вычисления.

Задача 1. Вычислить приближенное значение суммы

$1 + x/1! + x^2/2! + x^3/3! + ...$

Считаем, что нужное приближение получено, если вычислена сумма нескольких слагаемых, и очередное слагаемое оказалось по модулю меньше, чем данное малое положительное число ε.

Для решения задачи мы должны выполнить следующие действия:

1) получить очередное слагаемое (назовем его a);

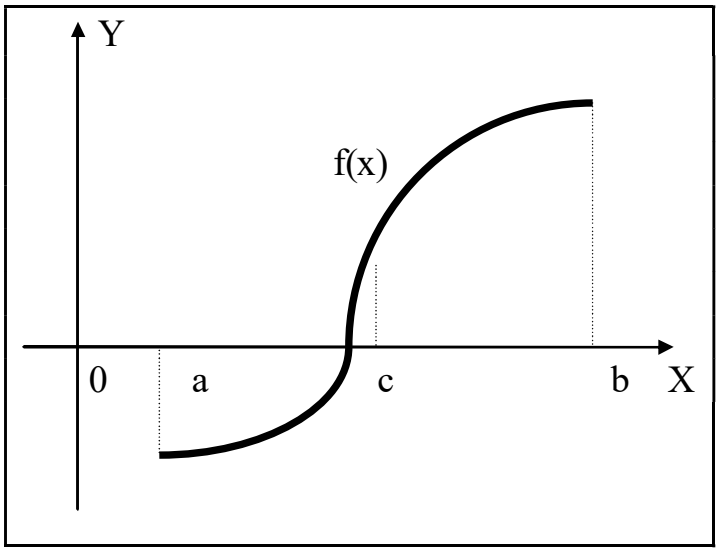
2) сравнить абсолютную величину слагаемого a с ε и, в зависимости от результата сравнения, либо продолжить, либо прекратить вычисления.

```
Program p;
var  a: real;
```

```
        x : real;
      ε : real;
      i : integer;
      S : integer;
begin
  readln(x, ε);
  i:=1;
  a:=1;
  S:=0;
  WHILE a>ε DO
    begin
      S:=S+a;  i:=i+1;   a:=a*x/i;
    end;
   writeln('приближенное значение суммы=',S:4:2)
end.
```

Задача 2. Рассмотрим часто встречающуюся задачу приближенного решения уравнения *f(x)=0*, где *f(x)* - заданная функция. Решить уравнение - значит найти такое значение x^*, при котором $f(x^*)=0$. Поиск решения осуществляется на интервале *[a;b],* причем *f(a)<0, a f(b)>0.* Используем метод деления пополам. Находим точку *c=(a+b)/2* - середина отрезка. Если *f(c)>0,* то границу *b* изменяем на значение *c*, а если *f(c)<0*, то изменяем *a*. *Процесс продолжаем, пока длина интервала не будет меньше заданной точности.*

Пусть $f(x)=x^2-2$, т.е. мы попытаемся найти значение квадратного корня из 2.

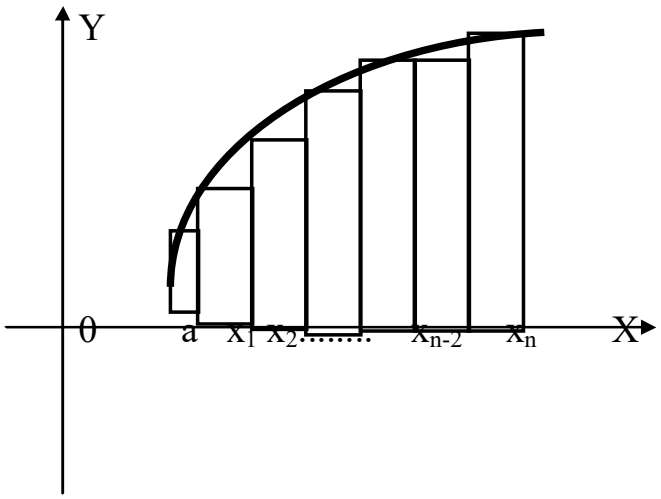



```
program popolam;
const eps=1.0E-3;
var a,b,c:real;
Function eg(x,y:real):boolean;
 begin
   Eg:=Abs(x-y)<eps
 end;
Function F(x:real):real;
 begin
  F:=Sgr(x)-2
 end;
BEGIN
 Writeln('введите интервал'); readln(a,b);
```

```
if F(a)*F(b)>0  then  writeln('на этом интервале мы не можем
                                найти решение уравнения')
 else  begin
        while Not Eg(a,b) do
          begin  c:=(a+b)/2;
           if F(c)>0  then  b:=c  else  a:=c
          end;
         writeln('Результат ',a)
        end;
   readln
end.
```

Можно с помощью этой программы решить уравнения :
$x^2-6x+5=0$ $x-\cos x=0$ $x-\ln x-2=0$ $2x^3-9x^2-60x+1=0$

Задача 3. Пусть непрерывная положительная на отрезке [a,b] (a<b) функция. Вычислим площадь фигуры, ограниченную графиком функции *f(x)*, осью x и прямыми x=a и x=b. Для этого разобьем отрезок [a,b] на n равных отрезков одинаковой длины $\Delta x=(b-a)/n$ $a=x_0<x_1<...<x_i<x_{i+1}<....<x_n=b$



На каждом из отрезков $[x_i,x_{i+1}]$ как на основании, построим прямоугольник с высотой $f(x_i)$. Площадь прямоугольников мы умеем находить. Сумма площадей всех прямоугольников даст приближенное значение площади фигуры:

$S_{приб}=(b-a)/n*(f(x_0)+....f(x_{n-1}))$

Естественно, что чем мельче будет разбиение, тем точнее мы подсчитаем площадь фигуры.

Для отладки программы возьмем функцию $f(x^2)$ на отрезке [1,2]. Площадь фигуры равна 2,333....

```
program trapez;
const eps=1.0E-3;
var a,b,s,dx :real;
     i,n : integer;
Function eg(x,y:real):boolean;
 begin
```

```
        Eg:=Abs(x-y)<eps
 end;
Function F(x:real):real;
 begin
  F:=Sgr(x)
 end;
BEGIN
 Writeln('введите интервал и число частей для разбивки'); readln(a,b,n);
  x:=a;  dx:=(b-a)/n;  s:=0;
For i:=1 to n-1 do
 begin
    s:=s+F(x);
    x:=x+dx;
  end;
writeln('Результат = ',(b-a)/n*s);
readln;
end.
```

Вычислить площадь криволинейных трапеций для следующих функций: *f(x)=1/(1+x) [0,1]* *f(x)=1/x [1,3]*
f(x)=sinx [0,π/2]

Задача 4. Для вычисления элементарных функций в математике широко распространено представление этих функций в виде некоторых бесконечных сумм. Не вдаваясь в обоснование таких представлений, приведем некоторые их них :

$e^x=1 + x + x^2/2! + x^3/3! + ... + x^n/n! + ...$

$sinx=x - x^3/3! + x^5/5! - x^7/7! +...+ (-1)^n x^{2n-1}/(2n+1)! +....$

$cosx= 1 - x^2/2! + x^4/4! - x^6/6! +...+ (-1)^n x^{2n}/(2n)! +....$

$ln(1+x)=x - x^2/2 + x^3/3 - x^4/4 + ...+ (-1)^{n+1} x^n/n + ... (-1<x<=1)$

В каждом из разложений точность представления функции будет, вообще говоря, тем выше, чем больше взято слагаемых в сумме. Причем значения самих слагаемых с ростом n стремятся к нулю. Для вычисления значений функции с некоторой заданной точностью ε поступают следующим образом. Вычисляют и суммируют слагаемые до тех тор, пока очередное слагаемое не станет по абсолютной величине меньше ε или абсолютное значение разности между соседними слагаемыми не станет меньше ε. Полученную сумму и принимают за приближенное значение функции.

Вычисли функцию sinx.

```
Program rad;
```

```
 const  eps=1.0E-3;
 var  x,sn,ss : real;
       p, n : integer;
  {p - используется для чередования знака слагаемого}
Function Eg(x,y;real):boolean;
begin
   Eg:=Abs(x-y)<eps
 end;
Function F(n:integer;var x:real):real;
 var i:integer;
     s:longint;
 begin
  s:=1;
   for i:=2 to n do s:=s*i;
   x:=x*sqr(x);  F:=x/s
 end;
BEGIN
  writeln('Введите значение x');
  readln(x);
  ss:=0; sn:=x;  n:=1;  p:=1;
   Repeat
     ss:=sn;  {предыдущее значение слагаемого}
   { новое значение слагаемого}
    n:=n+2; p:=-1*p;  sn:=ss+p*F(n,x)
    Until  Eq(ss,sn)  or  (n>=12);
Writeln('Результат=',sn); readln
end.
```

Задача 5. Метод Монте-Карло для приближенного вычисления площади.

Пусть есть какая-нибудь фигура на плоскости, расположенная внутри стандартного квадрата со сторонами параллельными координатным осям.

Пусть про любую точку квадрата мы можем быстро узнать, попадает ли эта точка внутрь фигуры или нет.

Для выбора точек используют ***случайные числа Random(x)***

Если вызвать функцию много раз подряд, то множество полученных чисел будет равномерно распределено по отрезку [0,a]

```
Program ss;
var n: integer; {количество точек}
    a : integer;  {длина стороны квадрата}
    m, i : integer;
```

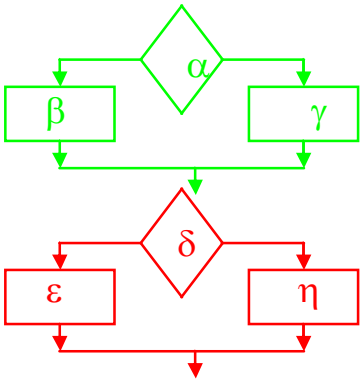
100

```
            x,y : real;
begin
  writeln('введите кол-во точек и сторону квадрата');
   readln(n,a);
m:=0;
 for i:=1 to n do
   begin
   x:=Rondom(a); y:=Random(a);
    if точка(x,y) внутри квадрата  then m:=m+1
   end;
S:=(m/n)*a*a
writeln('Результат ',s);
end.
```

§25. Основы структурного программирования

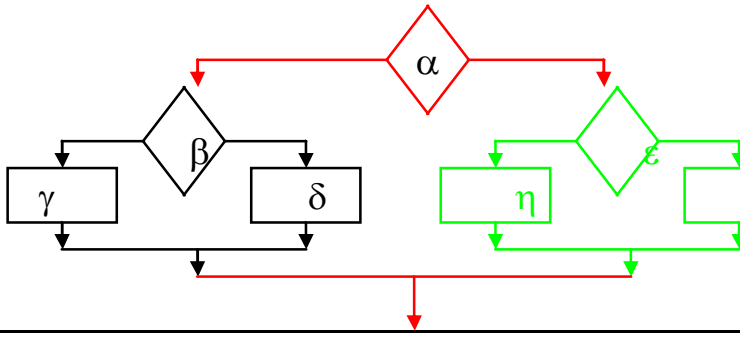
1. Следование ветвлений

if α then β else γ;
if δ then ε else η;



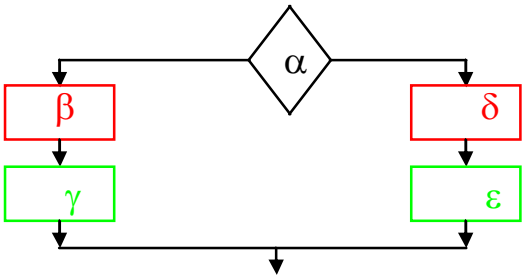
2. Вложение ветвлений :

if α then if β then γ else δ
else if ε then η else s



3. Вложение следования в ветвление

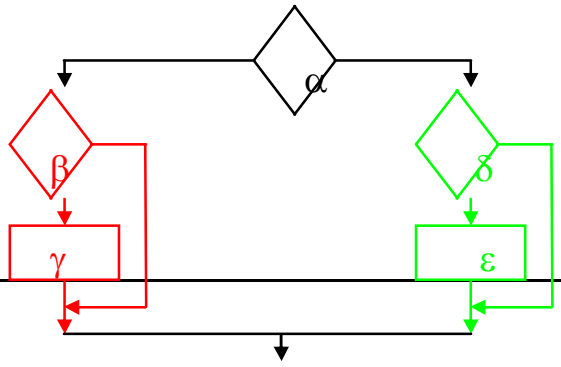
if α then begin β; γ; end
else begin δ; ε; end;



4. Вложение обхода в ветвление

if α then begin if β then γ end
else if δ then ε

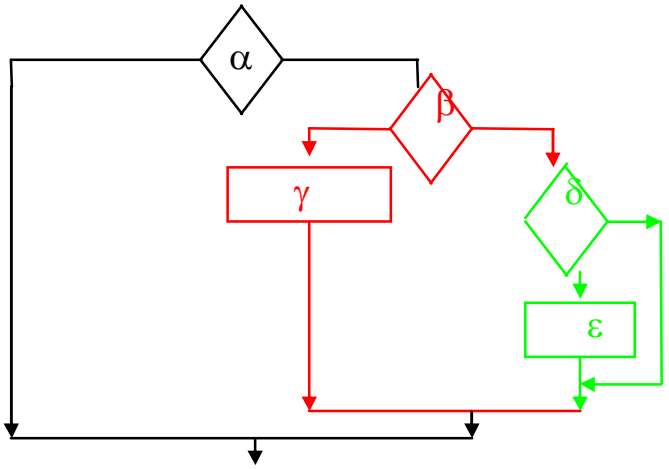
begin - end писать обязательно,
иначе смысл программы будет



другим.

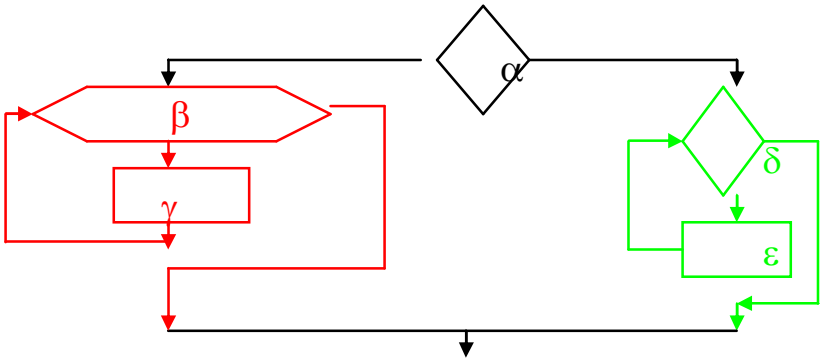
Если begin end не ставить то получим

if α then if β then γ
else if δ then ε



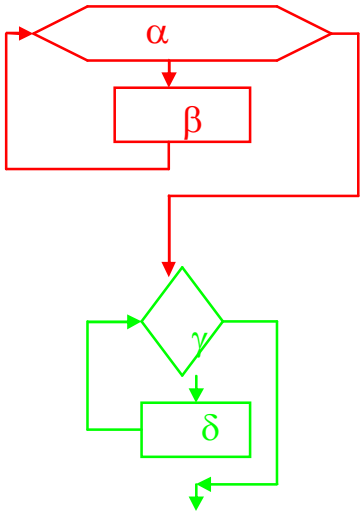
5. Вложение циклов в ветвление.

If α then for β do γ
else while δ do begin
ε
end



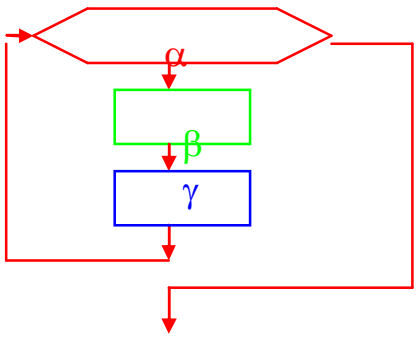
6. Следование циклов

```
for  α  do
  begin
    β
  end;
while γ  do
 begin
    δ
 end;
```



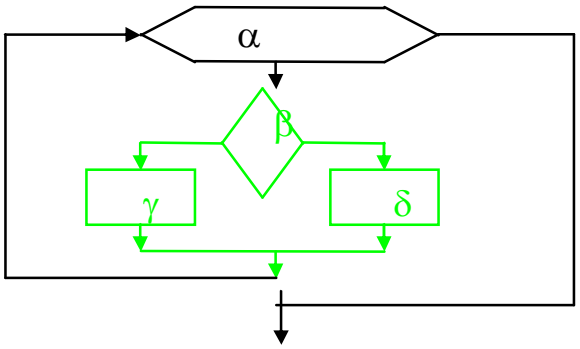
7. Вложение следования в цикл.

```
For  α  do
 begin
    β
    γ
 end;
```



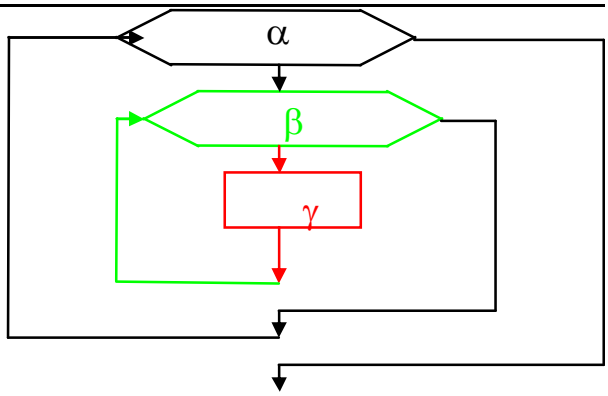
8. Вложение ветвлений в цикл

```
for α do
  if β then γ else δ
```



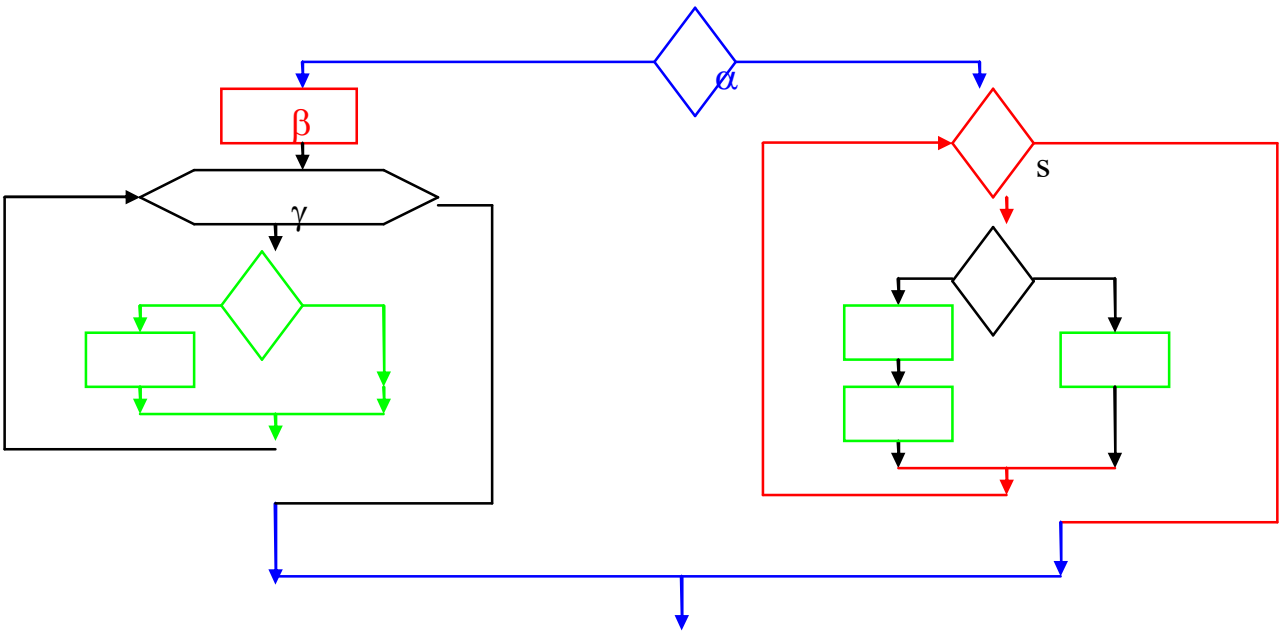
9. Вложение цикла в цикл

```
for α do
 for β do
  begin
     γ
  end
```



10. Комплексный пример

```
if α
 then beginβ; for γ do if δ then begin ε end
 else while s do begin if i then l; m
                              else k;
                 end;
```



Содержание