Язык программирования Pascal

Оглавление:

Рекомендуемая литература:

1. TurboPascal 7.0. учебное пособие. Фаронов В.В.
2. Основы алгоритмизации. Методические указания к лабораторным работам по информатике. 2007. Костенко В. Г.,Кузнецова В.В.

- Язык программирования PASCAL создан профессором Виртом, директором Института информатики Швейцарской высшей политехнической школы, и назван в честь великого французского математика и философа Блеза Паскаля- первого в мире создателя счетно-решающей машины.
- Язык PASCAL был разработан для обучения учащихся практике современного программирования. И считается наиболее желательным на первых шагах в этой области.

Синтаксис языка - правила построения языковых конструкций.

Алфавит языка *Pascal* включает:

а) буквы английского алфавита от *A* до *Z* и от *a* до z.

б) Арабские цифры от 0 до 9.

в) Специальные символы:

Знаки операций + – / *

Знаки пунктуации

. , : ; ( ) [ ] { } $ @ # ^ ' = < >

г) Символ подчёркивания _ , считающийся буквой.

Комментарии

В текст программы можно вставлять комментарии, облегчающие понимание используемых переменных, блоков, действий и т.п. Заключаются между символами *{* и *}* или *(** и **)*

Пример: { Это комментарий }

(* Это тоже комментарий *)

Для экспоненциальной формы записи чисел используется буква «*E*» или «*e*».

Например: 9E-6 означает $9 \cdot 10^{-6}$, а 24.337E+6 или 24.337E6 обозначают $24.337 \cdot 10^{6}$.

Атрибут длины строки символов выражается действительным количеством символов между апострофами.

Например:

'Строка символов'

' ' ' '

';'

'' {пустая строка}

' ' {пробел}

Идентификаторы

Идентификатор – это любая последовательность латинских букв, цифр и символа подчеркивания, но всегда начинающаяся с буквы.

Применяются для *определенных пользователем* имен констант, переменных, типов, процедур, функций, модулей, программ, меток, полей в записях. Имеют произвольную длину, но только первые 63 символа являются значимыми. Строчные и прописные символы тождественны.

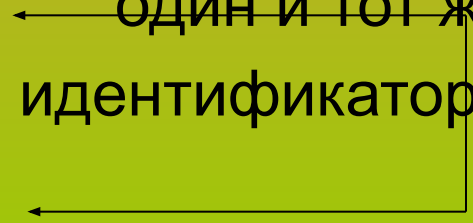
Примеры:

Com53All30

My_Ident

Name1

name1



Операторы

Программа состоит из операторов – единиц действий языка. Могут быть *выполняемые* и *невыполняемые* операторы. Выполняемые – производят вычисления или управляют процессом вычислений. Невыполняемые содержат сведения о структуре и организации данных и их свойствах. В конце оператора ставится *;*. Максимальная длина строки – 126 символов. Почти все операторы начинаются *ключевым словом.*

Простые и составные операторы

Операторы делятся на *простые* и *составные.*

Простые операторы описаны ранее.

Составной оператор (блок) – группа операторов, ограничен-ная конструкциями *Begin* и *End*.

Точка с запятой не может быть до *Begin* и перед *End*.

После *End* можеь быть:

1. пробел, если следующий оператор *End* или слово *Else*
2. точка с запятой, если следующим является выполняемый оператор

Составной оператор используется для ограничения:

1) Раздела операторов программ, процедур, функций
2) Групп операторов в условных операторах, операторах варианта(выбора) и цикла, где он рассматривается как один оператор

Ключевые (зарезервированные) слова

- Ключевые слова – это идентификаторы, включающие служебные слова - операторы и стандартные функции (например, begin, end, div и т. д.)
- Ключевые слова можно использовать только по своему прямому назначению и их нельзя переопределять.
- Операторы языка описывают некоторые алгоритмические действия, необходимые для решения задач.
- Стандартные функции это функции (подпрограммы) встроенные в язык.

Структура программы на языке *Pascal*

Программа состоит из трёх блоков:

а) *Заголовок программы*

б) *Раздел описаний*

в) *Тело программы*

Схема программы со всеми возможными разделами:

Program Name (Input,Output); {Заголовок программы}
Uses {Описание используемых модулей}
Label {Описание меток}
Const {Описание констант} **Раздел описаний**
Type {Описание типов}
Var {Описание переменных}
Procedure {Описание процедур}
Function {Описание функций}
Begin
Оператор 1;
Оператор 2; {Раздел операторов ... (тело программы)}
…
Оператор n;
End.

Описание заголовка заканчивается символом «;»
Например:

Program *MyProgram (Input, Output);*

Program *MyProgram;*

После служебного слова *uses* перечисляются модули. Например:

Uses *Crt, Graph;*

За служебным словом *Label* перечисляются идентификаторы меток. Например:

Label *M1, M2, M3;*

После служебного слова *Const* перечисляют константы. Например:

```
Const
A = 12;          {целочисленная константа A}
B: Real = 23.05; {типизированная константа B}
S='Строка';      {строковая константа}
```

Типы данных описываются после служебного слова *Type*. Например:

```
Type
Color = (Red, Green, Blue); {Перечисляемый тип}
Alfafit = 'A'..'Z'              {Тип диапазон (интервальный)}
MassivReal=array[1..100] of Real; {Массив из ста
                                   элементов типа Real}
MassivChar=array[0..19] of Char;   {Массив из 20
                                   элементов типа Char}
```

Раздел описания переменных начинается служебным словом *Var.* Например:

```
Var X,Y,Z: real;
    I,J,K: integer;
    Digit: 0..9;
    C: Color;
    Done, Error: boolean;
    Operator: (plus, minus, times);
    Matrix: array[1..10,1..10] of Real;
```

Типы данных, используемые в *Pascal*

Концепция типа языка *Pascal* имеет следующие основные свойства:

- Любой тип данных определяет множество значений, к которому принадлежит константа, которые может принимать переменная или выражение, или вырабатывать операция или функция;

- Тип значения, задаваемого константой, переменной или выражением, можно определить по их виду или описанию;

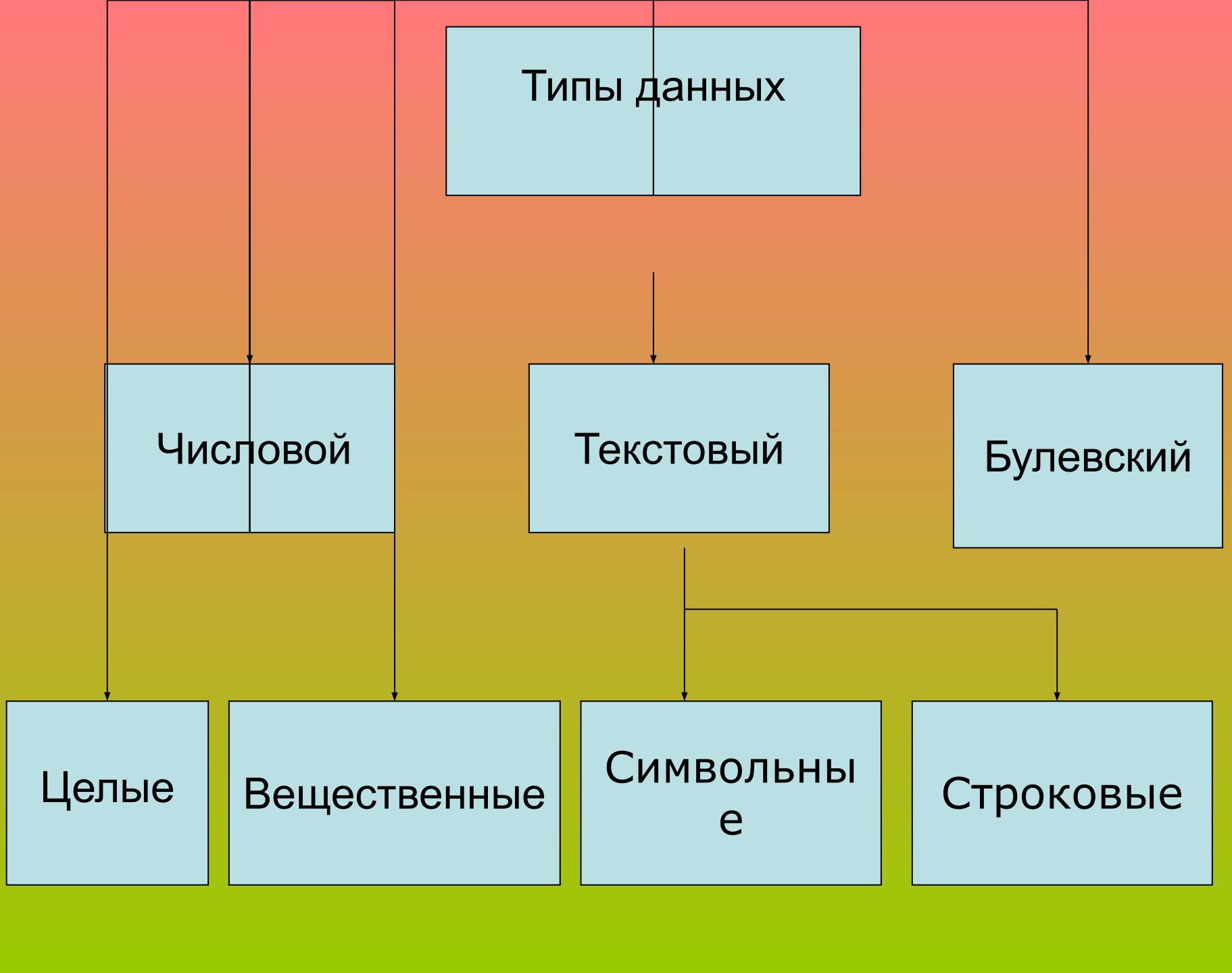
- Каждая операция или функция требует аргументов фиксированного типа и выдает результат фиксированного типа.

Тип определяет:

- Возможные значения переменных, констант, функций, выражений, принадлежащих к данному типу;
- Внутреннюю форму представления данных в ЭВМ;
- Операции и функции, которые могут выполняться над величинами, принадлежащими к данному типу.

В *Pascal* все типы данных разделятся на следующие группы:

- Скалярные;
- Ссылочные;
- Структурированные процедурные и объектные;


Типы данных
Числовой
Текстовый
Булевский
Целые
Вещественные
Символьные
Строковые

Целый тип данных

- *Byte (длинной в байт).* Диапазон: 0..255. Занимает 1 байт.
- *Word (длиной в слово беззнаковый).* Диапазон: 0..65535. Занимает 2 байта.
- *Shortint (короткое целое).* Диапазон: -128…127. Занимает 1 байт.
- *Integer (целое).* Диапазон: -32768..32767. Занимает 2 байта.
- *Longint (длинное целое).* Диапазон: -2147483648…2147483647. Занимает 4 байта.

К данными целого типа применимы следующие операции:

а) сравнения («=» равенство, «<>» неравенство, «<» меньше, «<=» меньше либо равно, «>» больше, «>=» больше либо равно);

б) сложение (+);

в) одноместный (унарный) плюс (+);

г) вычитание (–);

д) одноместный (унарный) минус (–);

е) умножение (*);

ж) деление нацело (получение частного) (*DIV*);

з) получение остатка от деления на цело (*MOD*);

и) логический сдвиг влево (*ShL*);

к) логический сдвиг вправо (*ShR*).

Вещественный тип

ТИП	РАЗМЕР (в бит/байт)	ДИАПАЗОН
Single	32/4	1.5E-45..3.4E38
Real	48/6	2.9E-39..1.7E38
Double	64/8	5.0E-324..1.7E308
Extended	80/10	3.4E-4932..1.1E4932

- ***чаще всего используется тип Real***

Логический тип

- Данные типа *Boolean* могут принимать два значения: *True (Истина)* и *False (Ложь)*. Над данными типа *Boolean* допустимы следующие операции:
- - сравнения (=, <>, <, <=, >, >=);
- - *And* (логическое И);
- - *Or* (логическое ИЛИ);
- - *Xor* (логическое исключающее ИЛИ);
- - *Not* (логическое отрицание).

Над данными вещественных типов допустимы следующие операции:

- сложение (+);
- одноместный (унарный) плюс (+);
- вычитание (–);
- одноместный (унарный) минус (–);
- умножение (*);
- деление (получение частного) (/);

Текстовый тип данных

Два вида: *символьный* и *строковый*

- ***Char*** . Диапазон: 1 символ (в соответствии с внутренним кодом от 0 до 255). Занимает 1 байт.
- ***String*** . Диапазон: до 255 символов. Занимает (n+1) байта, где n – количество символов.

  Заключаются в кавычки - ' ‘.

  Для них разрешены две функции преобразования:

Ord(C) ***Chr(K)***.

Функция ***Ord(C)*** возвращает кодировку символа (**с**).

Функция ***Chr(K)*** по коду (**к**) возвращает значение символа.

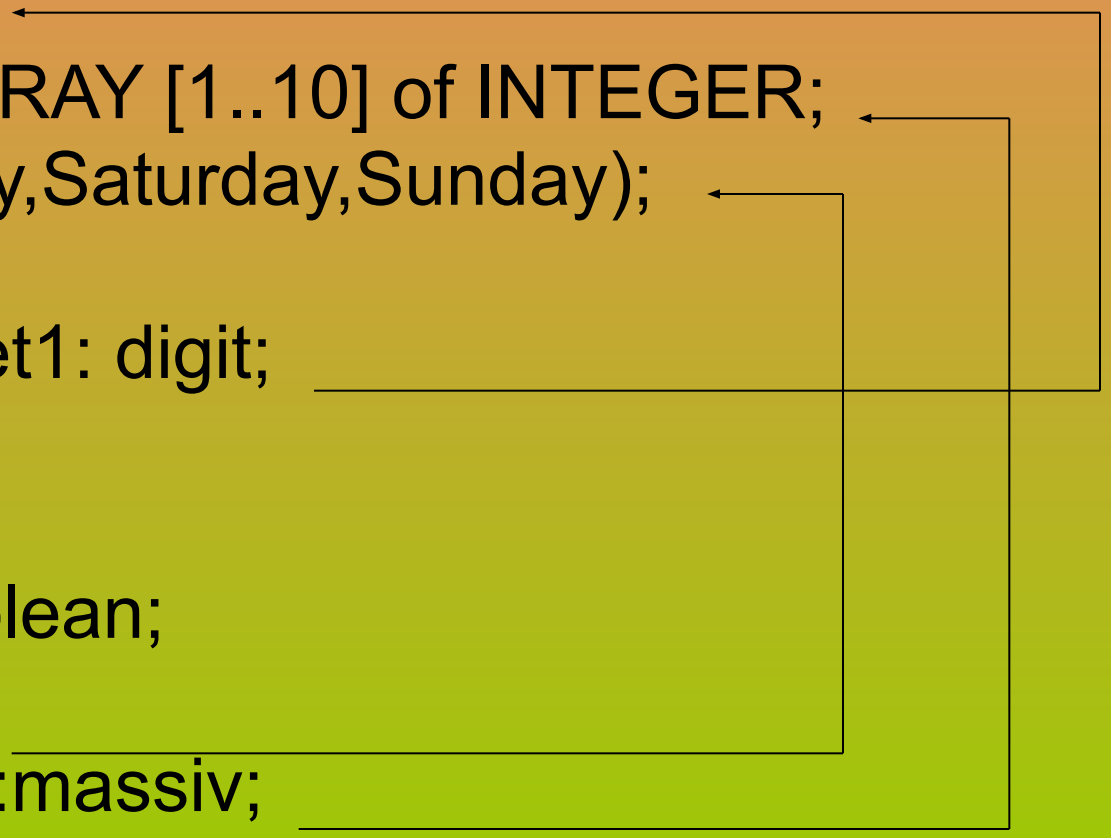
Пример:

```
VAR MyChar, B: char;
   MyString: string[12];        {переменная строкового
            типа длиной 12 символов}     …
MyChar:=‘A’; B:=‘Z’; MyString:=‘FK-the best!’;
```

Совмещенные объявления типов

Типы переменных можно определять и в разделе типа и в разделе описания переменных.

```
Type
  Stroka = STRING[10];
  digit = 0..9;
  massiv = ARRAY [1..10] of INTEGER;
  Days=(Friday,Saturday,Sunday);
Var
  MySet, Myset1: digit;
  A, B, C:real;
  D, E:integer;
  MyLogic:boolean;
  aDay:Days;
  T2,T3,T4,T5:massiv;
```



Стандартные функции языка *Pascal*

При описании стандартных функций будет использоваться следующий синтаксис:
<Имя_функции>(<аргумент:тип_аргумента>):<тип_возвращаемого_значения>.

Abs (X :Real): Real – возвращает абсолютное значение аргумента (|*X*|);

Abs (X :Integer): Integer – возвращает абсолютное значение аргумента (|*X*|);

ArcTan (X:Real): Real – возвращает арктангенс аргумента (*arctg X*).

Chr (A :Byte): Char – возвращает символ, код которого равен *A*.

Cos (X :Real): Real – возвращает косинус аргумента (*cos X*);

Exp (X :Real): Real – возвращает экспоненту аргумента (e^x);

Frac (X :Real): Real – возвращает дробную часть аргумента;

Int(X:Real):Real – возвращает целую часть аргумента;

Ln (X :Real): Real – возвращает натуральный логарифм аргумента (*ln X*);
Odd (A:Integer):Boolean – возвращает *True*, если *A* нечетно.
Ord (A :Char): Byte – возвращает порядковый номер символа *A*;
Round (X:Real): Integer – возвращает результат округления аргумента до ближайшего целого;
Random (A:Integer): Integer – возвращает случайное число из интервала [*0,A*];
Sqr (X:Real): Real – возвращает квадрат аргумента (*X2*);
Sqr (X:Integer): Integer – возвращает квадрат аргумента (*X2*);
Sqrt (X:Real): Real – возвращает квадратный корень аргумента( );
Sin (X:Real): Real – возвращает синус аргумента;
Trunc (X:Real): Integer – отбрасывает дробную часть действительного аргумента;
UpCase (A :Char):Char – превращает строчные буквы латинского алфавита в соответствующие им прописные.

Для вычисления значений других функций следует пользоваться тождествами:

$$\arcsin x = arctg\frac{x}{\sqrt{1-x\cdot x}};$$

$$\arccos x = \frac{\pi}{2 - arctg\dfrac{x}{\sqrt{1-x\cdot x}}};$$

$$arcctg\, x = \frac{\pi}{2-\arctan x};$$

$$\log_b a = \frac{\ln a}{\ln b};$$

$$a^x = e^{x\cdot \ln a}\ (a>0).$$

Построение арифметических выражений

- При построении арифметических выражений используются унарные и бинарные арифметические операции.
- Порядок выполнения арифметических операций:
  1. Умножение и деление
  2. Сложение и вычитание

Для изменения порядка выполнения арифметических операций, выражения, которые необходимо выполнить в первую очередь, записываются в круглых скобках.

Математическая запись:

$$x = \frac{a^2 + \sqrt{a + \cos^2 b}}{\log_c b^2 - tg\frac{a}{b+\pi}} \Big/ \left(\frac{\cos a + |\sin b|}{e^{b-a}} + 1 \right)$$

Версия на языке Pascal: (все операторы пишутся друг за другом!)

```
X := ( Sqr ( a ) + Sqrt ( a + Sqr ( cos ( b ) ) ) )
/ ( Ln ( Sqr ( b ) ) / Ln ( c ) – Sin ( a / ( b + Pi ) ) / Cos ( a / ( b + Pi ) ) )
/ ( ( Cos ( a ) + Abs ( Sin ( b ) ) ) / Exp ( b – a ) + 1 );
```

- Оператор « :=»

Оператор « := » называется оператором присваивания. Он предписывает выполнить выражение, заданное в его правой части, и присвоить результат переменной, имя которой указано в левой его части. Переменная и выражение должны быть совместимы по типу. Его синтаксис:

```
Y:=2;
X:='Строка';
P:=V1;
Summa:=V2;
```

Где:

X, Y, P, Summa – имена переменных, описанных в разделе описания переменных;

2, 'Строка' – значения заданные явно (своими изображениями);

V1,V2 - выражения, значения которых нужно вычислить.

Встроенные константы

- Встроенными константами в *Pascal* называются константы, значения которых предопределены.

Например:

MaxInt : Integer - возвращает число *32767*, наибольшее значение типа *Integer*;
PI : Real – возвращает число *π= 3.14159265358*.

Элементы структурного программирования

- Структурированная программа – это программа, составленная из фиксированного множества базовых конструкций.

Типы базовых конструкций программы:

1) ***следование*** - конструкция, представляющая собой последовательное выполнение двух или более операций.

2) ***ветвление*** - конструкция, состоящая из развилки, двух операций и слияния.

3) ***цикл*** - конструкция, имеющая линии управления, ведущие к предыдущим операциям или развилкам.

В языке *Pascal* количество базовых конструкций увеличено до шести, это:

- следование;
- ветвление;
- цикл с предусловием;
- цикл с постусловием;
- цикл с параметром;
- выбор.

Программирование алгоритмов линейной структуры

Ввод-вывод данных

Процедуры вывода:

- *Write(<параметры>)*, *WriteLn(<параметры>)*

Процедуры ввода:

- *Read(<параметры>)*, *ReadLn(<параметры>)*

Операторы с *Ln* отличаются тем, что после вывода(ввода) последней переменной курсор переводится в начало новой строки.

Примеры:

Write (A,B,4); Вывод значений *A, B* и значения 4

Write (A+B); Вывод результата сложения значений двух переменных *A* и *B*

Write ('Строка'); Вывод строкового изображения «Строка» на экран

Write ('Строка', B); Вывод строкового изображения «Строка» на экран и значения переменной *B*.

Допускается использование операторов без параметров:

ReadLn; - останавливает выполнение программы до нажатия клавиши *Enter*

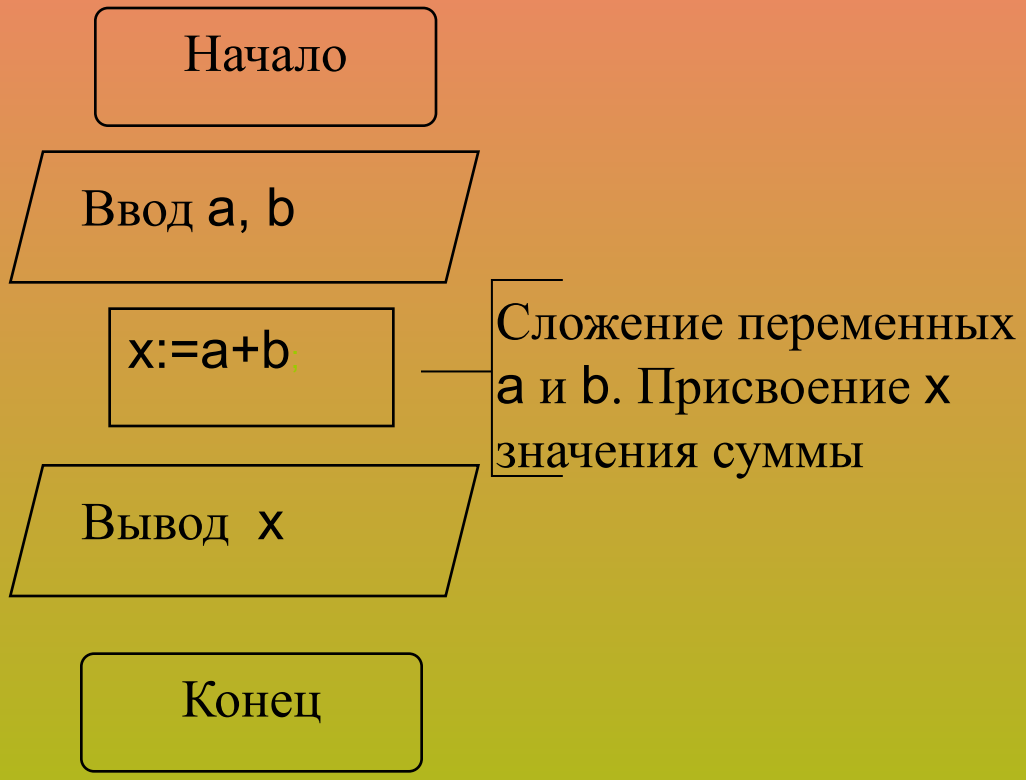
WriteLn; - осуществляет пропуск строки, в которой находится курсор, и переводит его в начало новой строки.

Операторы вывода допускают использование указания о ширине поля, отводимого под значение в явном виде:

WriteLn(Y:5:3);

где *5* – количество позиций, отведенных под запись значения переменной *Y*, а *3* – количество позиций, отведенное под запись дробной части

Линейные вычислительные процессы (следование)



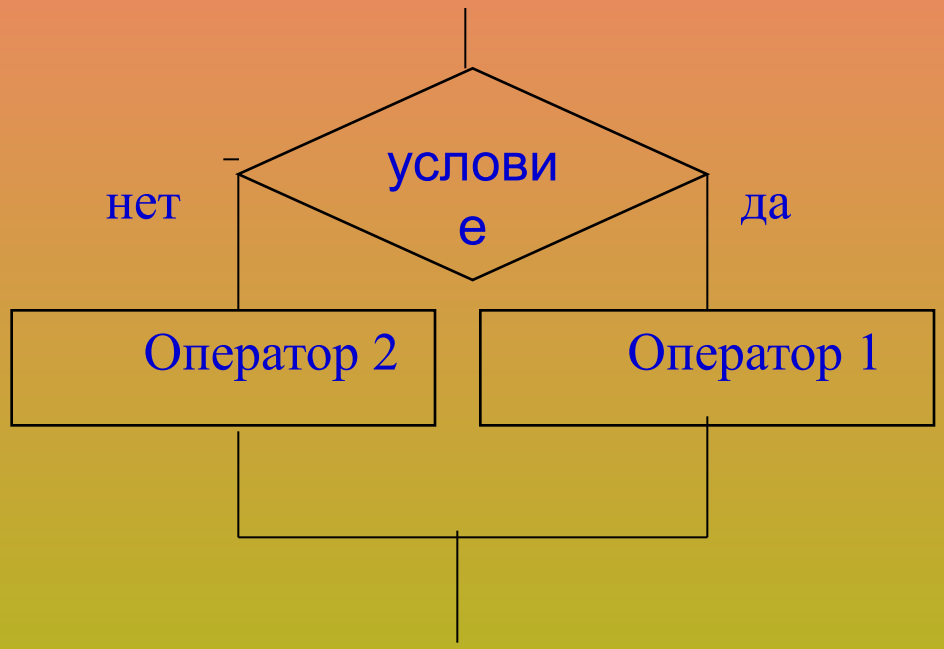
Программа на языке Pascal линейного алгоритма сложения двух чисел

```
PROGRAM Example;
var    a, b, x : Byte;
BEGIN
Write (’Введите значения переменных a и b
   типа Byte’);
Read (a,b);
x:=a+b;       {Сложение}
WriteLn('a + b =',x);
END.
```

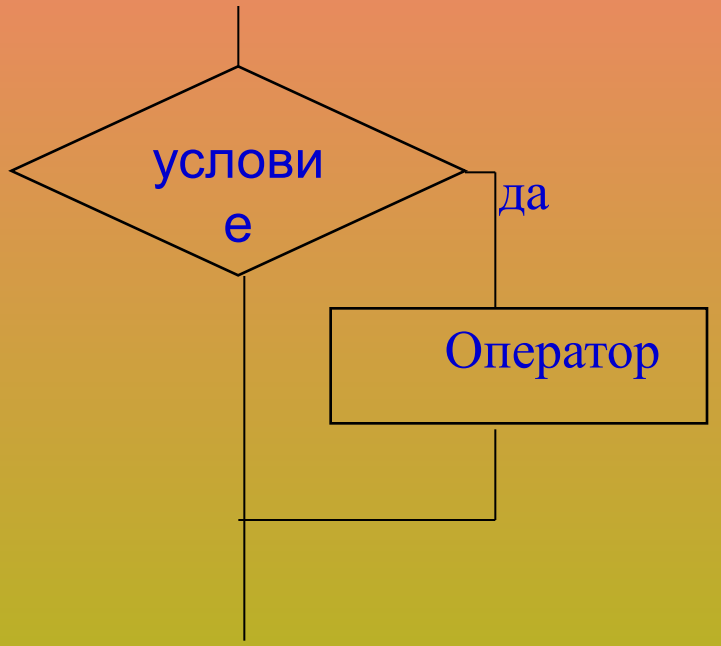
Алгоритмы и программы ветвящейся структуры. Условный оператор *If*.

Оператор выбора *Case*

Алгоритмы и программы ветвящейся структуры



а)



б)

Логические операции

Логическая операция конъюнкция (AND)

Значение операндов		Результат операции
A	***B***	***A and B***
True	***True***	***True***
True	***False***	***False***
False	***True***	***False***
False	***False***	***False***

Логическая операция дизъюнкция (OR)

Значение операндов		Результат операции
A	***B***	***A or B***
True	***True***	***True***
True	***False***	***True***
False	***True***	***True***
False	***False***	***False***

Логическая операция исключающее ИЛИ (XOR)

Значение операндов		Результат операции
A	***B***	***A xor B***
True	***True***	***True***
True	***False***	***True***
False	***True***	***True***
False	***False***	***False***

Логическая операция отрицания (NOT)

Значение операндов	Результат операции
A	***not A***
True	***False***
False	***True***

Например:

(A<=B) and (B>C) or (A<>D). При значении переменных *A=10, B=15, C=20, D=25* значение всего выражения равно *True*.

(A<=B) or (B>C) xor (A<>D). При значении переменных *A=10, B=15, C=20, D=25* значение всего выражения равно *False*.

not (A<=B) or (B>C). При значении переменных *A=10, B=15, C=20* значение всего выражения равно *False.*

(A<=B) or not (B>C). При значении переменных *A=10, B=15, C=20* значение всего выражения равно *True.*

Порядок выполнения логических операций:

а) Not;
*б) And, *, Div, Mod, /;*
в) Or, Xor, +, – ;

Условный оператор *IF*

Позволяет произвести развилку алгоритма, в которой осуществляется выбор одной из двух альтернативных ветвей, в зависимости от некоторого условия. В качестве условия выбора используется значение логического выражения.

Синтаксис оператора *IF*:

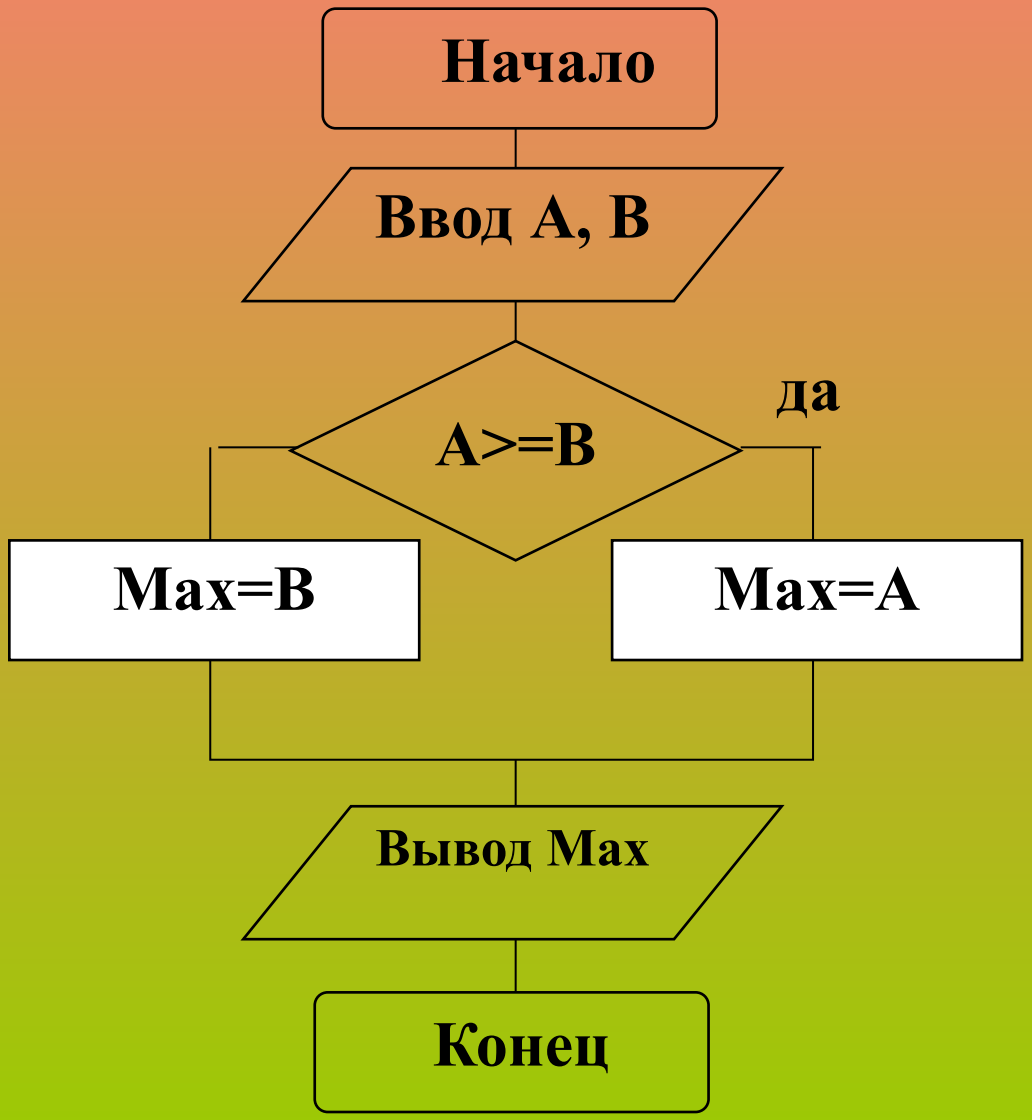
If <логическое выражение> then <оператор1>
{else <оператор2>}

В результате вычисления выражения получается логическое значение типа *Boolean*. Если результатом является значение *True*, то выполняется оператор1, в противном случае (*False*) – оператор2 .

Примеры составления алгоритмов и программ с использованием условного оператора *If*

Пример:

Найти максимальное из двух целых чисел

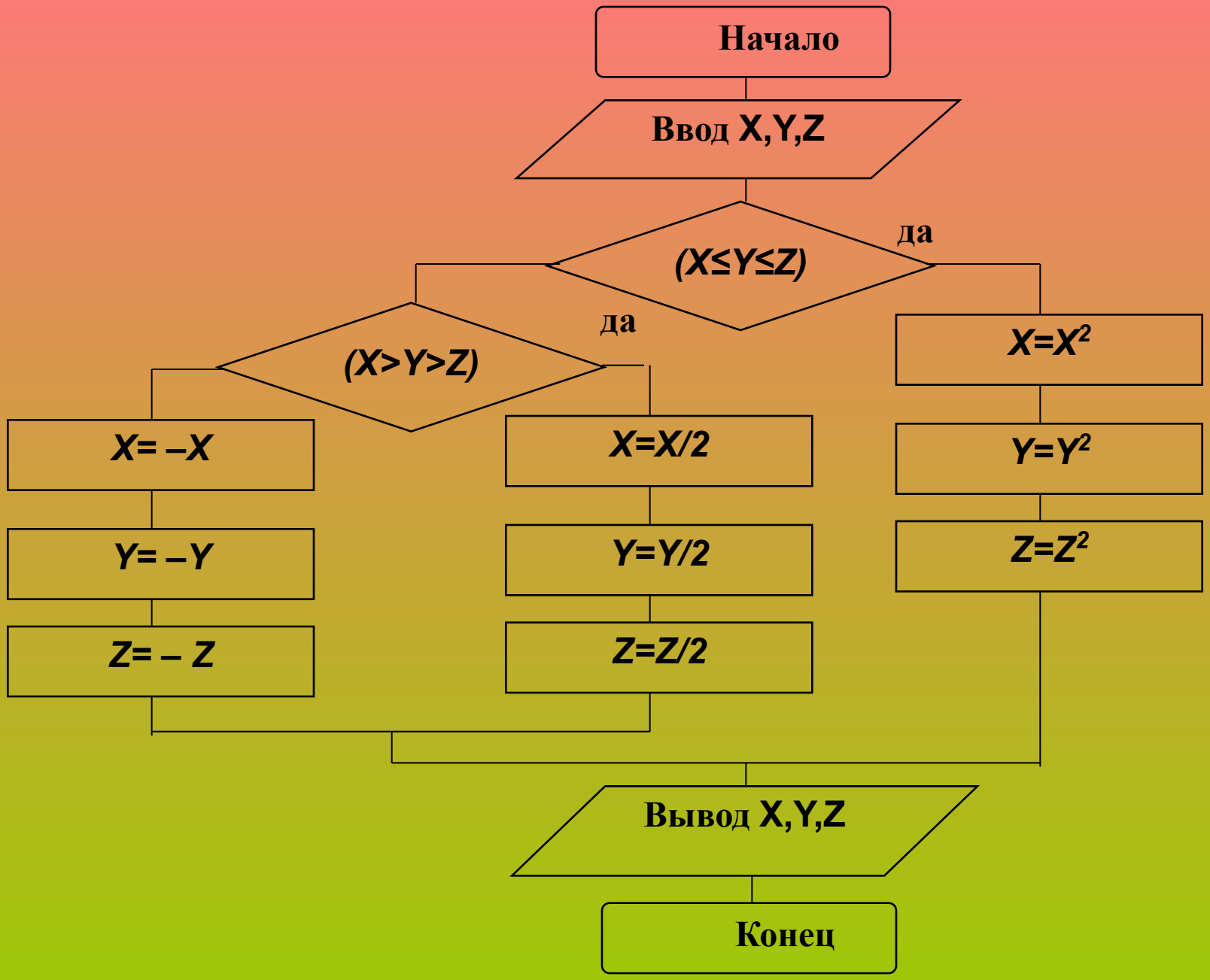


```
Program  Example;      {заголовок программы}
Var  A, B  : Integer;      {описание переменных}
       Max : Integer;

Begin                  {начало программы}
    {вывод на экран сообщения}
     Write ('Введите значение A = ');
    {ввод с клавиатуры переменной A}
    ReadLn (A);
    Write ('Введите значение B = '); ReadLn (B);
    {блок проверки условия}
     If  A>=B   then  Max:=A  else  Max:=B;
    {вывод на экран результата }
    WriteLn (‘Большее из двух целых чисел A и B: ',Max);
End.                   {конец программы}
```

Определить принадлежит ли вводимое с клавиатуры значение *A* интервалу [0..9].

```
Program  Example;
Var  A  : Real;
 Max : Integer;
 Str   : String;

Begin
     Write ('Введите значение A = '); ReadLn (A);
     If  (A>=0) and (A<=9)   then  Str:='принадлежит'
                                 else  Str:='не принадлежит';
      WriteLn ('Значение A  ',Str,'  отрезку  [0..9]')
End.
```

Задача: Даны значения X,Y,Z. Выяснить, если ***(X≤Y≤Z)*** тогда значения переменных X,Y,Z нужно возвести в квадрат, если (***X>Y>Z)*** значения X,Y,Z нужно разделить на 2, в противном случае присвоить отрицательное значение.


Начало
Ввод X,Y,Z
(X≤Y≤Z)
да
(X>Y>Z)
да
X=X²
X= –X
X=X/2
Y=Y²
Y= –Y
Y=Y/2
Z=Z²
Z= – Z
Z=Z/2
Вывод X,Y,Z
Конец

```
Program Example;
Var X,Y,Z:Real;
Begin
    Write ('Введите значение X,Y,Z');
    ReadLn (X,Y,Z);
    If (X<=Y)and(Y<=Z) then
        begin
            X:=Sqr(X);  Y:=Sqr(Y); Z:=Sqr(Z)
        end
    else
        If (X>Y) and (Y>Z) then
            begin
                X:=X/2;  Y:=Y/2; Z:=Z/2
            end
        else
            begin
                X:= -X; Y:= -Y Z:= -Z
        end;
    WriteLn(´X=´, X, ´   Y=´, Y, ´    Z=´, Z)
End.
```

Оператор варианта *Case*

Производит развилку алгоритма на произвольное количество ветвей. Из этого множества выбирается единственная ветвь, отвечающая одному из заданных условий, либо ни одной, если ни одно из условий не выполняется.

Синтаксис:

Case <селектор> of

<константа выбора1> : <оператор1>;

…

< константа выбораN > : <операторN>;

[*else* <оператор>;]

end;

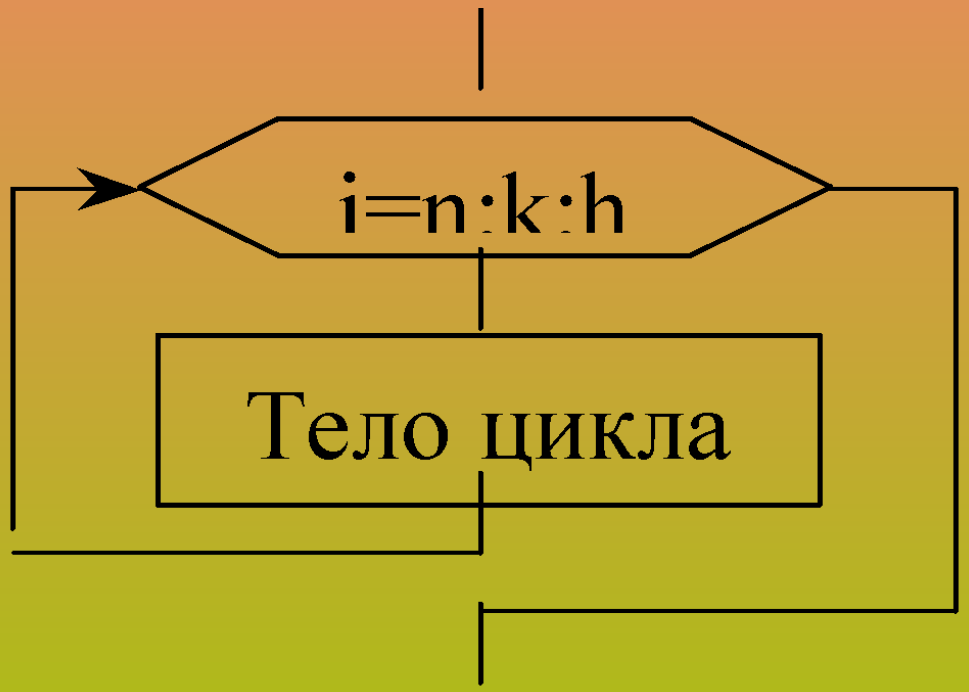
Селектор должен иметь порядковый тип!

Примеры оператора варианта:

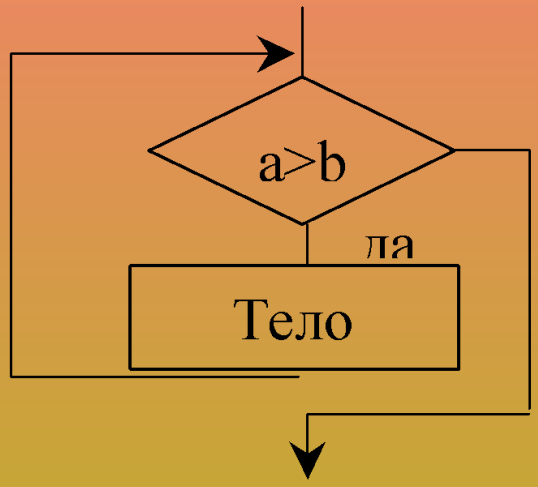
```
Case Operat of
        plus:  X := X+Y;
       minus:  X := X-Y;
       times:  X := X*Y;
End;

Case I of
       0, 2, 4, 6, 8: Writeln('Четная цифра');
       1, 3, 5, 7, 9: Writeln('Нечетная цифра');
       10..100: Writeln('Между 10 и 100');
 else
       Writeln(‘Цифра вне диапазона 0-100');
End;
```

Алгоритмы и программы циклической структуры

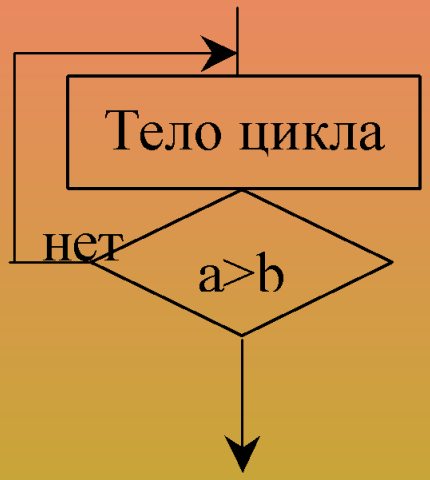
Понятие цикла. Разновидности циклов



Блок-схема алгоритма
цикла с параметром



Блок-схема алгоритма цикла с предусловием



Блок-схема алгоритма цикла с постусловием

Цикл с параметром *FOR*

Синтаксис оператора *For* выглядит следующим образом:

For i:=n to k do <*оператор*>; - с возрастанием параметра

For i:=n downto k do <*оператор*>; - с убыванием параметра

где *i* – параметр цикла;

n – начальное значение параметра цикла;

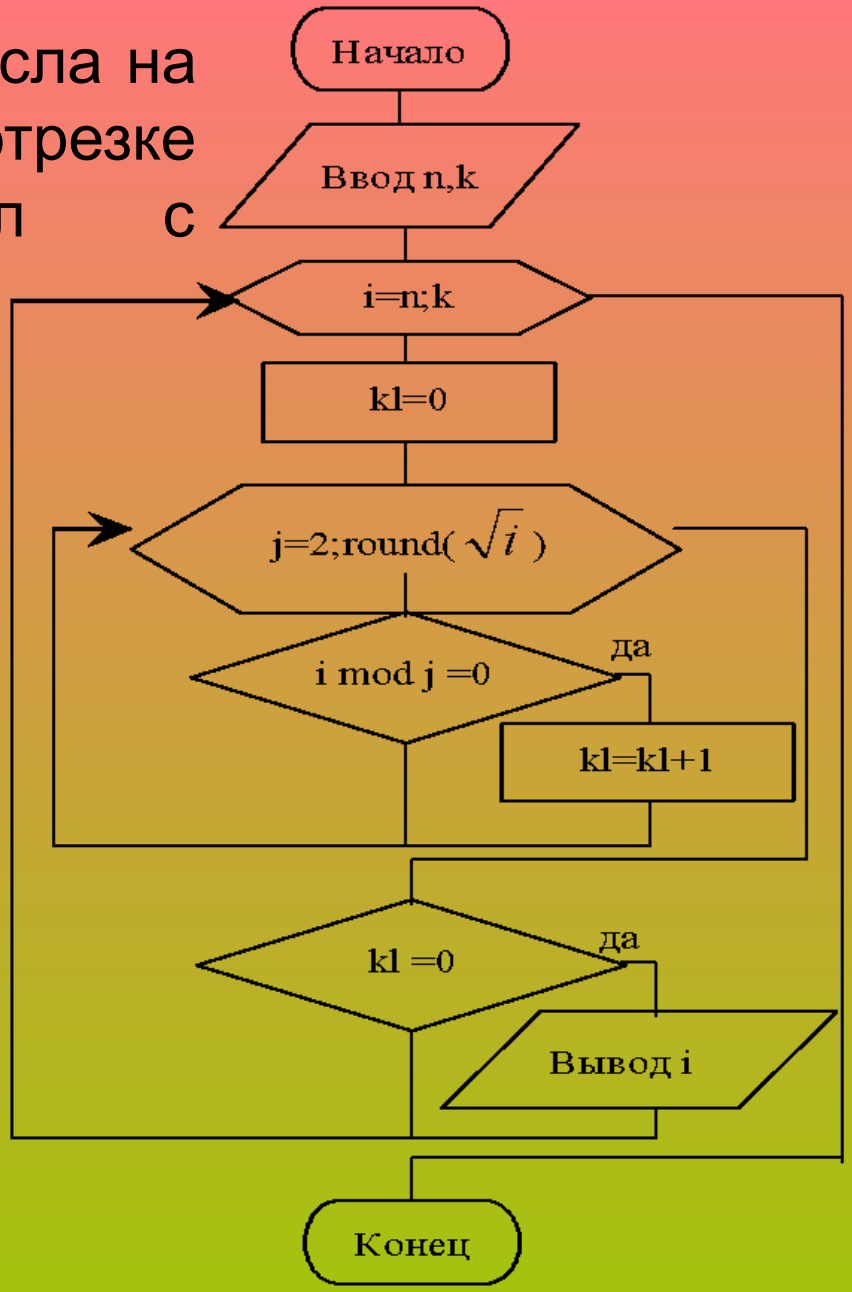
k – конечное значение параметра цикла;

<*оператор*> – оператор, являющийся телом цикла;

Например:

```
For i:=n downto k do
begin
  <оператор 1>;
        . . .
  <оператор N>;
end;
```

Найти все простые числа на заданном отрезке (использовать цикл с параметром)



Блок-схема алгоритма решения задачи

```
Program  Example (Input, Output);
   var  n  : Integer;
  k  : Integer;
  i,j: Integer;
  kl : Integer;
Begin
   Write ('Введите нижнюю границу  отрезка – '); ReadLn (n);
   Write ('Введите верхнюю границу отрезка – '); ReadLn (k);
   WriteLn ('Все простые числа из отрезка [',n,',',k,']');
   For i:=n to k  do
      begin
        kl:=0;
           For j:=2 to Round (Sqrt(i))  do
              If  (i MOD j)=0  then  kl:=kl+1;
              If  kl=0  then  Write (i,' ')
      end
End.
```

Цикл с предусловием *While*

Синтаксис его выглядит следующим образом:
While <*условие*> *do* <*оператор*>;

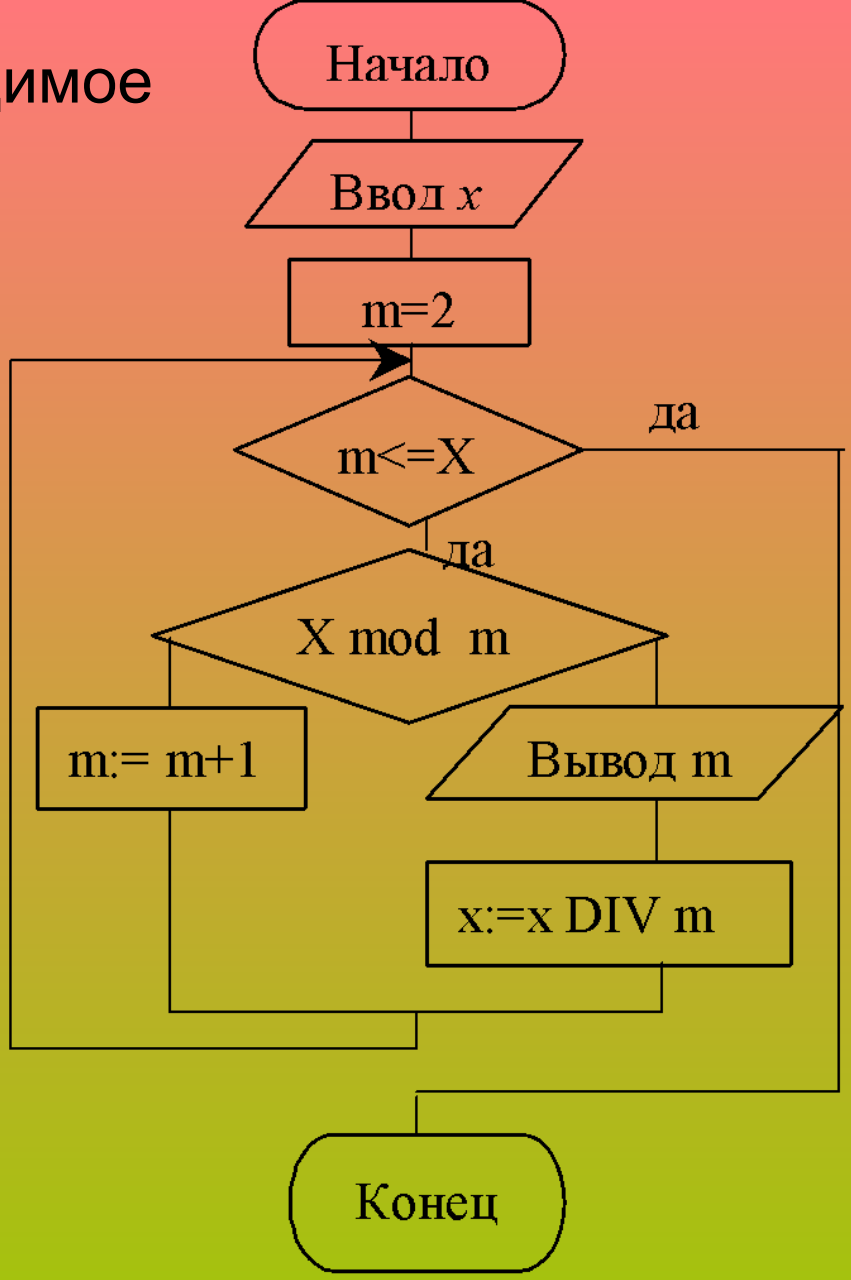
Условие должно быть булевского типа.

Вычисление его производится до того, как внутренний оператор будет выполнен.

Внутренний оператор выполняется до тех пор, пока условие будет иметь значение *True*. Если выражение с самого начала принимает значение *False*, то цикл не выполняется ни разу.

Разложить целое число, вводимое с клавиатуры, на простые множители.

```
Program  Example;
     var  x,m: Integer;
  Begin
     Write ('Введите целое
число... '); ReadLn (x);
     WriteLn ('Разложение
числа ',x,' на простые
множители');
     m:=2;
     While  m<=x  do
        If  (x mod m)=0
     then  begin  Write (' * ',m);
                  x:=x DIV m  end
           else  m:=m+1
  End.
```



Блок-схема алгоритма решения задачи

Цикл с постусловием *Repeat*

Синтаксис цикла с постусловием выглядит следующим образом:

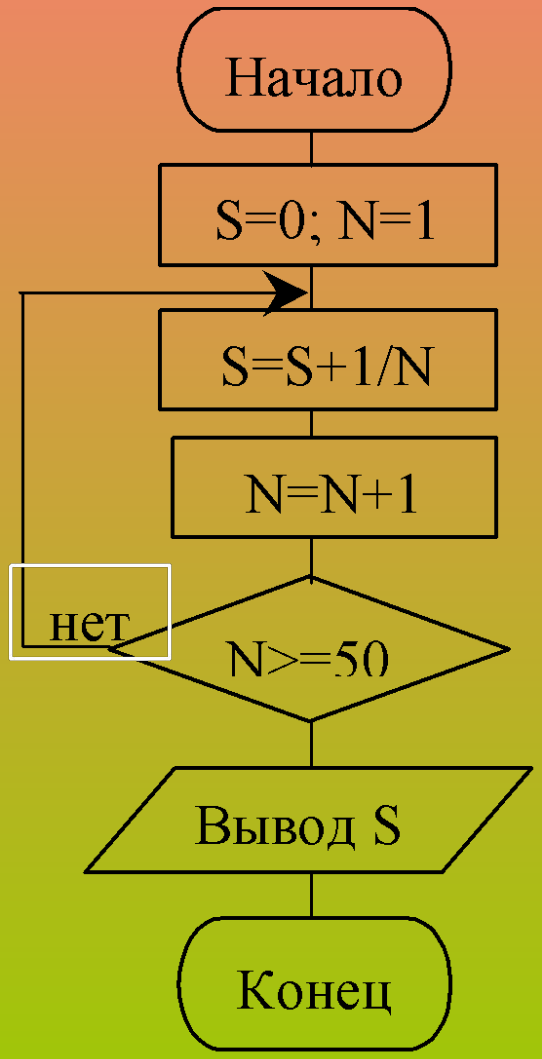
Repeat
  <оператор 1>;
  <оператор 1>;
    …
  <оператор 1>;
Until <условие>;

Условие должно быть булевского типа.
Цикл повторяется, пока условие имеет значение *False.*

Вычислить значение суммы

$$S = \sum_{n=1}^{50} \frac{1}{n}.$$

```
Program Example;
var
   N: Integer;
   S: Real;
Begin
   S:=0;
   N:=1;
   Repeat
      S:=S+1/N;
      N:=N+1
  Until N>50;
  WriteLn ('Результат суммирования... ',S)
End.
```



Регулярные типы данных. Массивы

Понятие регулярного типа

Массив - ограниченная упорядоченная совокупность однотипных величин.

Для обозначения отдельных компонент используется конструкция, называемая переменной с индексом или с индексами:
A[5] S[k+1] B[3,5].

Пример описания одномерного массива:

```
Type
  Massiv = array [1..20] of Real;
Var
   A, B: Massiv;
       C: array [10..30] of Integer;
```

Пример описание двумерного массива:

```
Type
 Matrix = array [1..20, 1..10] of Real;
Var
    X, Y: Matrix;
        Z: array [1..10, 1..10] of Integer;
```

Инициализация элементов массива

Для ввода или вывода массива в список ввода или вывода помещается переменная с индексом, а операторы ввода или вывода выполняются в цикле, изменяя при каждой итерации значение индекса.

Инициализация массивов осуществляется:

Первый способ:

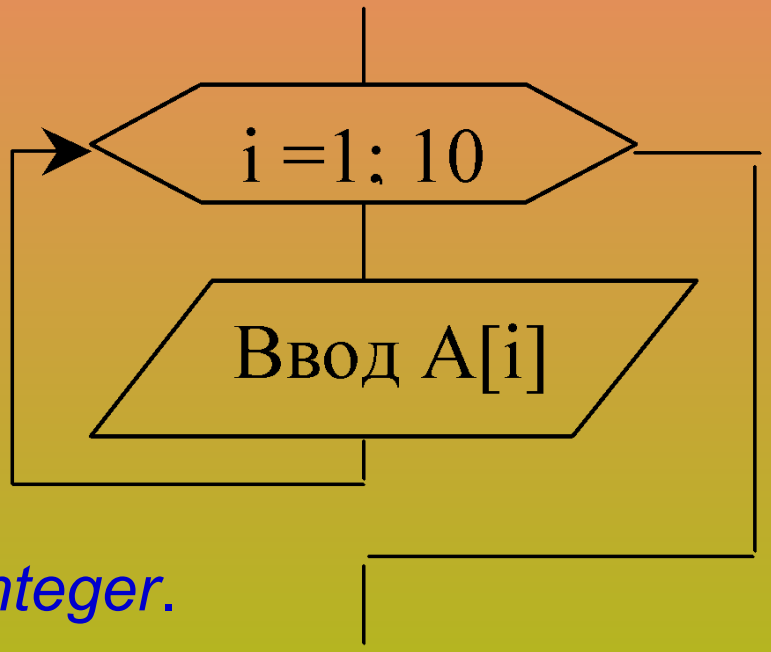
```
type Mass= Array[1..10] of Real;
const
    K: Mass= ( 0, 2.1, 4, 5.65, 6.1, 6.7, 7.2, 8, 8.7, 9.3 );
```

При инициализации двумерных массивов значения компонент каждого из входящих в него одномерных массивов записывается в скобках:

```
type Mass3x2= Array[1..3,1..2] of Integer;
const
   L: Mass3x2= ( (1, 2)
                 (3, 4)
                 (5, 6) );
```

Второй способ инициализации - использование разновидности процедуры FillChar:

FillChar(var V; NBytes: Word; B: {Byte|Char});



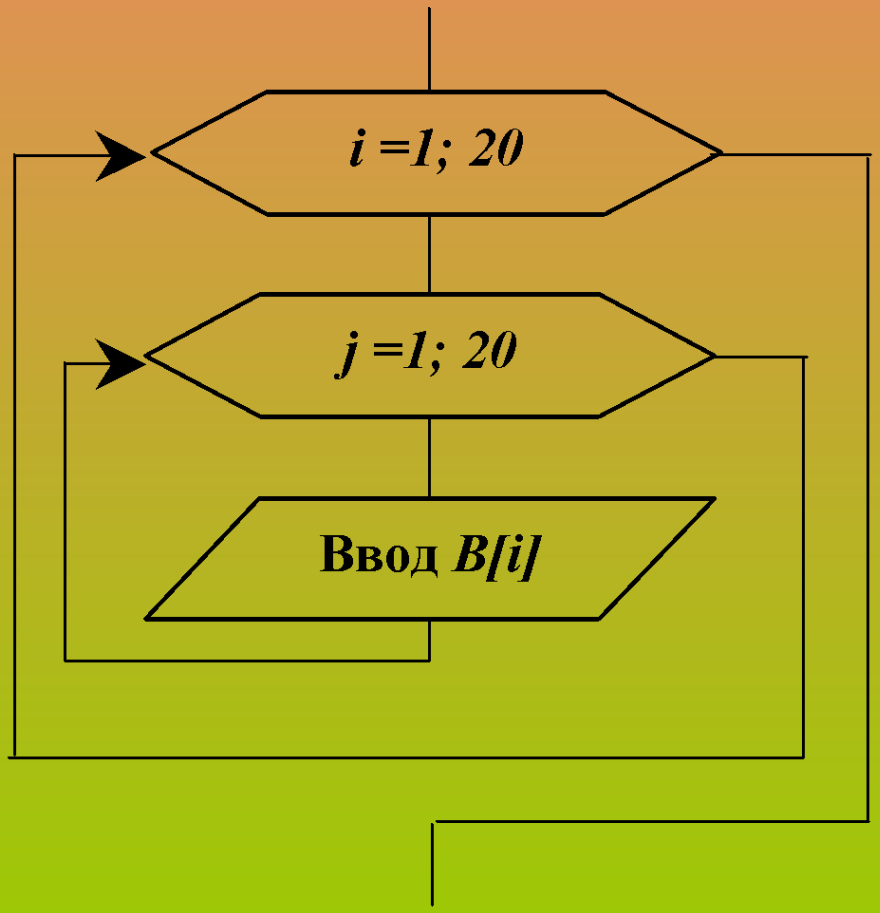

```
A:array [1..10] of Integer.
For i:=1 to 10 do
begin
    Write('Введите A[', i,']');
    Read(A[i])
end;
```

Инициализирование массива случайными значениями:

```
Randomize;
For i:=1 to 10 do
    A[i]:=Random(100);
```

Ввод элементов двумерного массива:

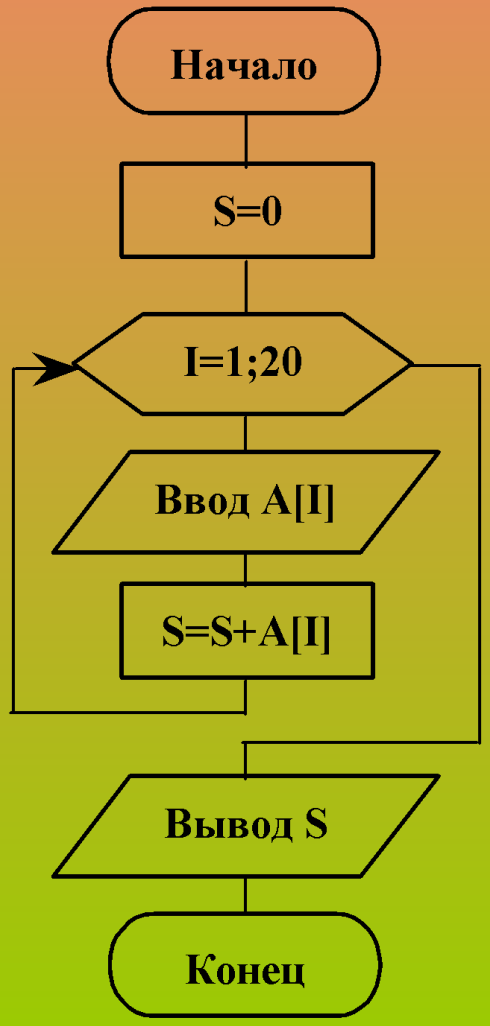
```
    B:array [1..20,1..20] of
Real.
    For i:=1 to 20 do
    For i:=1 to 20 do
    begin
        Write('Введите B[', i,']');
        Read(B[i])
    end;
```



Нахождение суммы и произведения массива

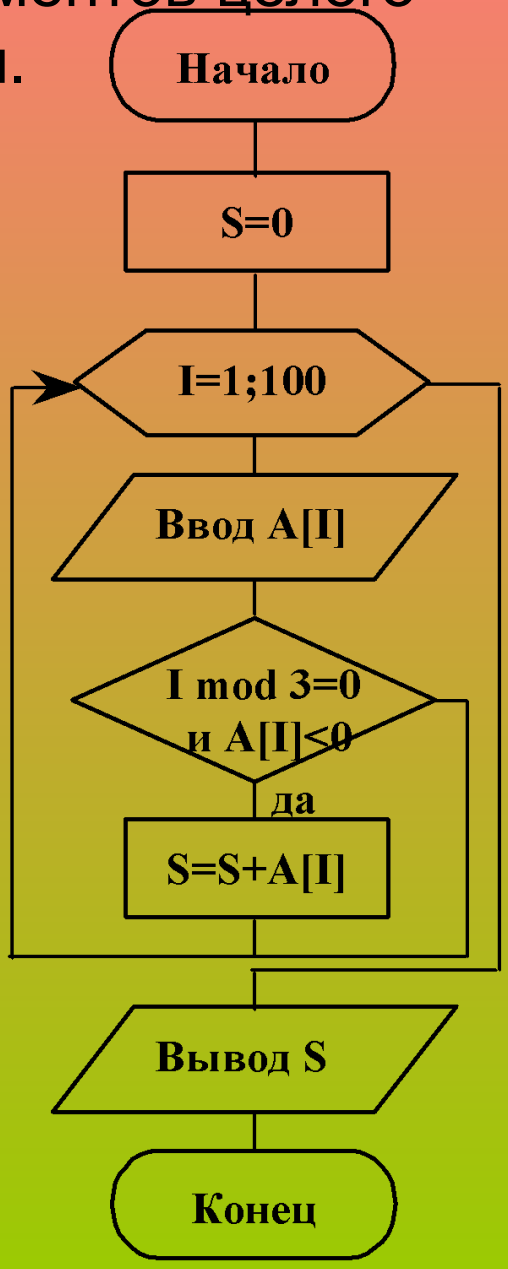
Пример: Необходимо найти сумму элементов одномерного массива состоящего из 20 элементов действительного типа.

```
Program Example;
var A:array [1..20] of Real;
      S: Real;
      I:Integer;
Begin
    S:=0; For I:=1 to 20 do begin
    Write('Введите A[', I,']');
    Read(A[I]);
    S:=S+A[I];
    end;
   WriteLn ('Сумма... ',S)
End.
```



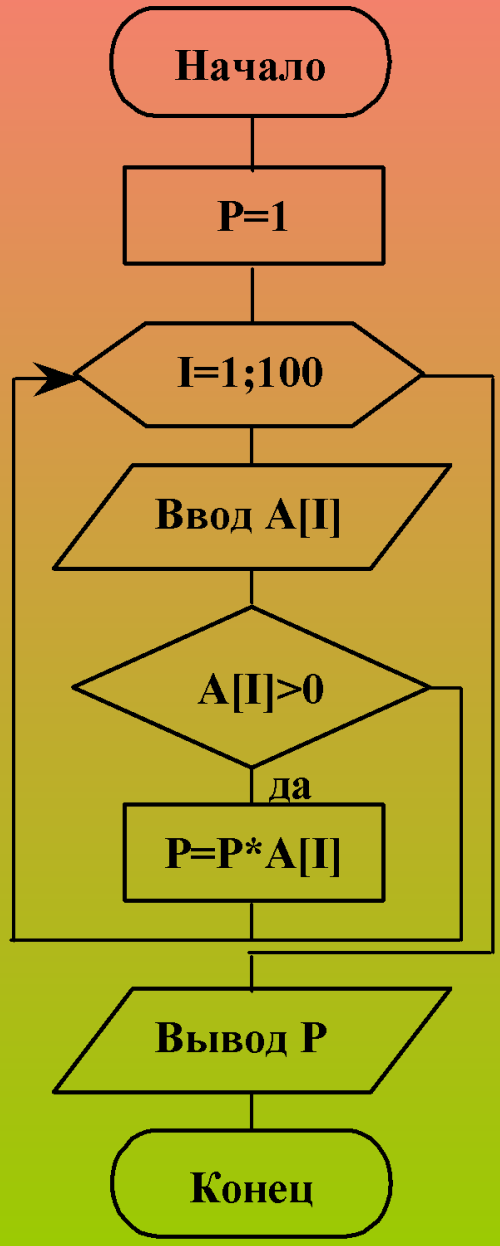
Пример: Найти сумму отрицательных элементов одномерного массива состоящего из 100 элементов целого типа, порядковый номер которых кратен трём.

```
Program Example;
var
   A:array [1..100] of Integer;
   S,I:Integer;
Begin
    Randomize;
   S:=0;
   For I:=1 to 100 do
   begin
      A[I]:=Random(100) – Random(50);
      If (I mod 3=0) and (A[I]<0) then
S:=S+A[I]
    end;
   WriteLn ('Результат суммирования...
',S)
End.
```



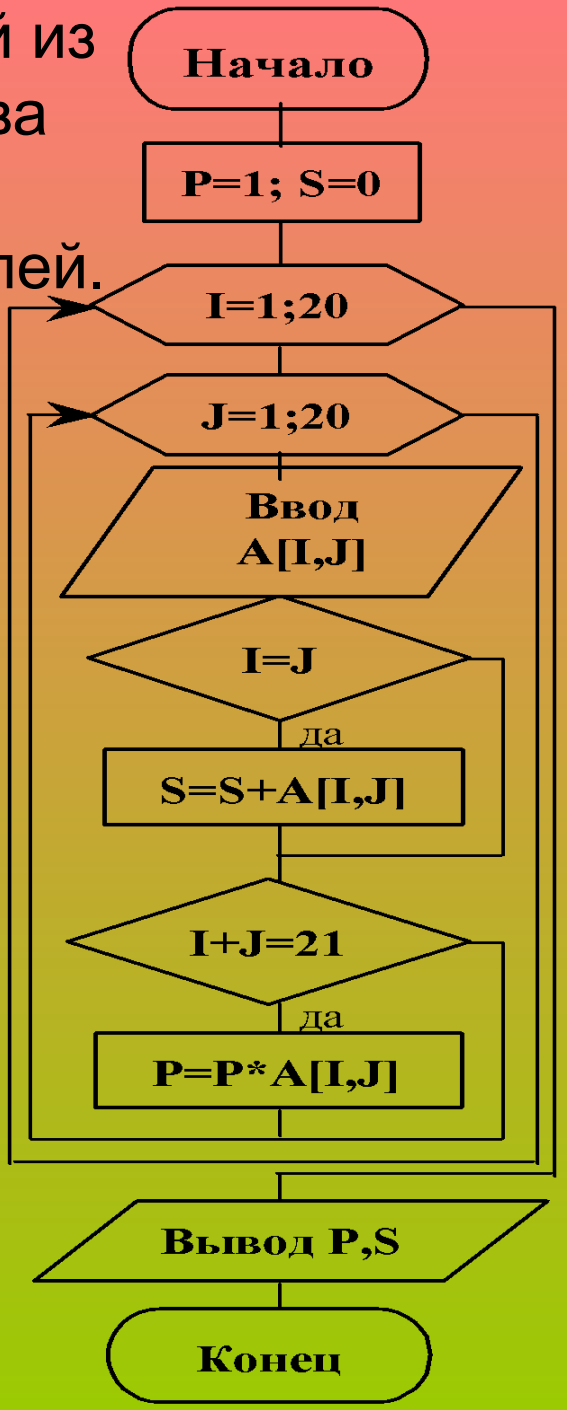
Пример: Найти произведение положительных элементов одномерного массива состоящего из 100 элементов целого типа.

```
Program Example;
 Var
    A:array [1..100] of Integer;
    S,I:Integer;
Begin
     Randomize;
    P:=1;
    For I:=1 to 100 do
    begin
        A[I]:=Random(100) – Random(50);
        If A[I]>0 then  P:=P*A[I]
     end;
   WriteLn ('Результат произведения...
',P)
End.
```



Пример: Дан двумерный массив, состоящий из элементов целого типа. Размерность массива 20×20. Найти сумму элементов главной и произведение элементов побочной диагоналей.

```
Program Example;
Var A:array [1..20,1..20] of Integer;
    S,P,I,J:Integer;
Begin
     Randomize; P:=1; S:=0;
    For I:=1 to 20 do
    For J:=1 to 20 do begin
        A[I, J]:=Random(100) – Random(50);
        If I=J then  S:=S+A[I, J];
        If I+J=20 then  P:=P*A[I, J];
     end;
     WriteLn ('Результат произведения...
',P);
     WriteLn ('Результат суммы... ',S)
End.
```



Алгоритмы сортировки массивов

Сортировка информации – это одна из стандартных функций, возникающих в процессе решения задач.

Сортировка данных - процесс изменения порядка расположения элементов в некоторых упорядоченных структурах данных.

Для любой пары чисел определены отношения «больше» или «меньше».

Сортировка данных используется для упорядоченной совокупности данных быстро и легко решается задача, как поиск и отбор информации по заданному условию.

Постановка задачи сортировки и методы её решения

Для сортировки данных требуется:

1) определить понятие порядка для элементов массива
2) определить понятия «больше» и «меньше» для каждой пары элементов.

Различают два вида сортировки данных:

- сортировка данных, расположенных в оперативной памяти компьютера (внутренняя сортировка);
- сортировка данных, расположенных на внешних запоминающих устройствах (внешняя сортировка).

Имеется одномерный массив чисел, состоящий из *n* элементов: *X[n].* Переставить элементы массива так, чтобы их значения располагались в порядке возрастания. Другими словами, для любой пары элементов *X[i]* и *X[i+1]* выполняется неравенство вида:
X[i] <= X[i+1].

В этой задаче определяется структура данных для внутренней сортировки (одномерный массив) и порядок упорядочения элементов. Результатом решения задачи должна быть программа сортировки массива одним или несколькими методами.

Наиболее известными являются следующие:

- метод сортировки обменами («пузырьковая» сортировка);
- метод сортировки вставками;
- метод сортировки выбором элемента;
- метод разделения (алгоритм «быстрой» сортировки метод Хоора);
- метод «пирамиды» (метод Уильямса-Флойда).

Главным показателем качества алгоритма внутренней сортировки является скорость сортировки.

Алгоритм сортировки обменами («пузырьковая» сортировка)

Суть алгоритма состоит в последовательном просмотре массива от конца к началу или от начала к концу и сравнении каждой пары элементов между собой. При этом «неправильное» расположение элементов устраняется путем их перестановки.

```
Procedure Puzirek;
Var i, j: Integer;
        y:Integer;
Begin
   For i := 2 to n do
   For j := n downto i do
      If X[j-1] > X[j] then  begin   y:=X[j-1]; X[j-1]:=X[j]; X[j]:=y
end;
End;
```

Алгоритм сортировки вставками

Метод сортировки вставками заключается в переборе всех элементов массива от первого до последнего и вставке каждого очередного элемента на место среди предшествующих ему элементов, упорядоченных ранее таким же способом.

```
Procedure Vstavka;
Var Z, Y, i, j: Integer;
Begin
    For i := 2 to n do
    For j := 1 to i-1 do
        If X[j] > X[i] then
        begin
Z := X[i];
For Y := i downto j+1 do X[Y] := X[Y-1];
X[j] := Z
        end
End;
```

Алгоритм сортировки выбором элемента

В массиве необходимо найти элемент с минимальным значением и поменять его местами с первым элементом массива. После этого элемент с минимальным значением отыскивается среди всех элементов, кроме первого, и меняется значениями со вторым элементом массива и т.д.

```
Procedure Vibor;
Var r, i, j: Integer;
Begin
    For i := 1 to n-1 do
    begin
        r := i;
        For j := i+1 to n do If a[r] > a[j] then r := j;
       Y:=a[r];  a[r]:=a[i]; a[i]:=Y;
     end
End;
```

Алгоритм быстрой сортировки (метод Хоора)

Метод основан на разделении массива на два непустых непересекающихся подмножества элементов. При этом в одной части массива должны оказаться все элементы, которые не превосходят по значению некоторый предварительно выбранный элемент массива, а в другой части - все элементы, не меньшие его. Аналогично следует поступить с каждой из полученных групп (при этом элемент для разделения можно выбирать просто из «середины» группы).

```
Procedure QuickSort(Left,Right:Integer);
Var I,J,Y,W,L:Integer;
Begin
    I:=Left; J:=Right; Y:=X[(Left+Right) div 2];
  Repeat
      While X[I]>Y do Inc(I);
      While Y>X[J] do Dec(J);
      If I<=J then
      begin
          W:=X[I]; X[I]:=X[J]; X[J]:=W;
          Inc(I);
          Dec(J);
      end;
  Until   I>J;
  If Left<J then QuickSort(Left,J);
  If I<Right then QuickSort(I,Right);
End;
```

Алгоритм пирамиды (метод Уильямса-Флойда)

Основан на специальном представлении массива в форме бинарного дерева, обладающего особыми свойствами и называемого «пирамидой».

Процесс сортировки состоит из двух этапов:
1) массив преобразуется к виду «пирамиды»
2) осуществляется сортировка «пирамиды».

Элементы массива, являющегося «пирамидой», обладают дополнительными свойствами:

1. Любой элемент пирамиды *X[i]* не меньше, чем его элементы-потомки *X[2i]* и *X[2i+1]* (соответственно первый элемент обладает максимальным значением):

X[i] >= X[2i],

X[i] >= X[2i+1]

2. Любая подпоследовательность элементов вида *X[n\2+1]*, *X[n\2+2], ... X[n]* также является пирамидой, обладающей свойствами 1 и 2. После преобразования массива к форме пирамиды сортировка осуществляется следующим образом.

Подпрограммы в *Turbo Pascal*

Понятие подпрограммы. Разновидности подпрограмм в *Turbo Pascal*

Подпрограмма – последовательность операторов, которые определены и записаны только в одном месте программы, однако их можно вызвать для выполнения из одной или нескольких точек программы.

Различают два вида подпрограмм:

1) процедуры
2) функции.

Процедура и функция – это именованная последовательность описаний и операторов

Процедура – это независимая именованная часть программы, которую можно вызвать по имени для выполнения определённой в ней последовательности действий.

Функция отличается от процедуры тем, что возвращает результат указанного при её описании типа.

Описание, определение и вызов процедур

Синтаксис заголовка процедуры:

```
Procedure <Name>(<Список формальных параметров>);
… {Раздел описаний}
Begin
  …{Раздел операторов процедуры}
End;
Procedure MyProc (A,B,C: Real; var X1,X2: Real);
Begin
  WriteLn('A=',A, '  B=', B, 'C=', C);
X1:=A+B;
X2:=A*B-C
End;
```

Формальные параметры – это переменные, посредством которых передаются данные из места вызова процедуры в её тело, либо из процедуры в места вызова.

Вызов процедуры производится указанием имени процедуры и списком фактических параметров:

Name(<Список фактических параметров>);

MyProc(K, L+M, 12, Y1, Y2);

Фактические параметры – это переменные (или значения заданные явно), которые передаются в процедуры на место формальных параметров.

Описание, определение и вызов функций

Функция состоит из заголовка, раздела описаний и раздела операторов.

Function <Name>(<Список формальных параметров>):<Type>;
… {Раздел описаний}
Begin
…{Раздел операторов процедуры}
Name:=<выражение соответствующего типа>;
…
End;

Пример: разработать функцию, определяющую по двум катетам гипотенузу прямоугольного треугольника.

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

Вызов функции из основной программы может выглядеть следующим образом:

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

или

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

Передача параметров в подпрограммы

Параметры процедур и функций могут быть следующих видов:

а) параметры-значения(без ключевого слова),

б) параметры-переменные(var),

в) параметры-константы(const)

г) не типизированные параметры(или var или const но без типа).

Передача параметров по значению

Формальный параметр-значение обрабатывается, как локальная по отношению к процедуре или функции переменная, за исключением того, что он получает свое начальное значение из соответствующего фактического параметра при активизации процедуры или функции.

Если параметр имеет строковый тип, то формальный параметр будет иметь атрибут размера, равный 255.

Передача параметров по ссылке

При передаче параметров по ссылке в формальный параметр передаётся адрес соответствующего фактического параметра.

При вызове процедур параметры-переменные обрабатываются так: формальный параметр получает адрес ячейки памяти, в которой хранится соответствующий фактический параметр.

Параметр-переменная используется, когда значение должно передаваться из процедуры или функции вызывающей программе.

Параметры-константы

Формальные параметры-константы работают аналогично локальной переменной, доступной только по чтению, которая получает свое значение при активизации процедуры или функции от соответствующего фактического параметра.

Присваивания формальному параметру-константе не допускаются.

Не типизированные параметры

Когда формальный параметр является не типизированным параметром-переменной, то соответствующий фактический параметр может представлять собой любую ссылку на переменную или константу, независимо от ее типа. Не типизированный параметр, описанный с ключевым словом *Var,* может модифицироваться, а не типизированный параметр, описанный с ключевым словом *Const,* доступен только для чтения.

Не типизированные параметры-переменные:

```
Function Equ(Var s,d; size: Word): Boolean;
Type     Bytes = array[0..MaxInt] of Byte;
Var      N: Integer;
Begin
   N := 0;
   While(N<size) and (Bytes(d)[N]<>Bytes(s)[N] do
Inc(N);
        Equ := N = size;
End;
```

Эта функция может использоваться для сравнения любых двух переменных любого размера. Например, с помощью описаний:

```
Type
    Vekt_Ar = array[1..10] of Integer;
    Pset = record
            x,y: integer;
    End;
Var
    Vector1, Vector2: Vekt_Ar;
      N: integer;
      P: Pset;
```

и вызовов функций:

```
Equ(Vector1,Vector2,SizeOf(Vekt_Ar))
Equ(Vector1,Vector2,SizeOf(integer)*N)
Equ(Vector1[1],Vector1[6],SizeOf(intege
r)*5)
Equ(Vector1[1],P,4)
```

Открытые параметры

Открытые параметры позволяют передавать одной и той же процедуре или функции строки и массивы различных размеров.

Открытые строковые параметры могут описываться двумя способами:

с помощью идентификатора *OpenString;*

с помощью ключевого слова *String в состоянии {$P+}.*

Параметр *S* процедуры *AssignString* – это открытый строковый параметр:

```
Procedure AssignString(var S: OpenString);
Begin
    S := '0123456789ABCDEF';
End;
```

Так как *S - это открытый строковый параметр, AssignString можно передавать переменные любого строкового типа:*

```
Var
    Str1: string[10];
    Str2: string[20];
Begin
    AssignString(Str1);          { Str1 := '0123456789' }
    AssignString(Str2);          { Str2 := '0123456789ABCDEF' }
End;
```

Использование открытых параметров.

```
Procedure FillStr(var S: OpenString; Ch: Char);
Begin
    S[0] := Chr(High(S));          { задает длину строки }
    FillChar(S[1], High(S), Ch);  { устанавливает число
символов }
End;
```

Использование открытых параметров массивов

```
Procedure Clear(var A: array of Real);
Var
   I: Word;
Begin
     for I := 0 to High(A) do A[I] := 0;
End;
Function Sum(const A: array of Real): Real;
Var
    I: Word;
    S: Real;
Begin
     S := 0;
     for I := 0 to High(A) do S := S + A[I];
       Sum := S;
End;
```

Когда типом элементов открытого параметра-массива является *Char, фактический параметр может быть строковой константой.* Например, с учетом предыдущего описания:

```
Procedure PringStr(Const S: array of Char);
Var
   I: Integer;
 Begin
     for I := 0 to High(S) do
         if S[I] <> #0 then Write(S[I]) else Break;
End;
```

и допустимы следующие операторы процедур:

```
PrintStr('Hello word');
PrintStr('A');
```

Передача имен процедур и функций в качестве параметров

Описание процедурных и функциональных типов производится в разделе описания типов:

```
Type
    FType = Function (Z: Real): Real;
    PType = Procedure (A,B,C: Real; Var X,Y: Real);
```

Для передачи имени функции (или процедуры) в качестве фактического параметра в процедуру или функцию необходимо придерживаться следующие правил:

- в секции *Type описать пользовательский тип как функцию (или процедуру с соответствующими параметрами);*
- в секции *Var объявить переменную с типом функции;*
- процедуры, передаваемые в качестве параметров, должны быть обязательно откомпилированы с опцией *{$F+};*
- связать переменную типа функция с пользовательской процедурой или функцией;
- передать в процедуру значение этой переменной (можно передавать и непосредственно имя процедуры или функции).

Составить программу для вычисления определенного интеграла методом Симпсона.

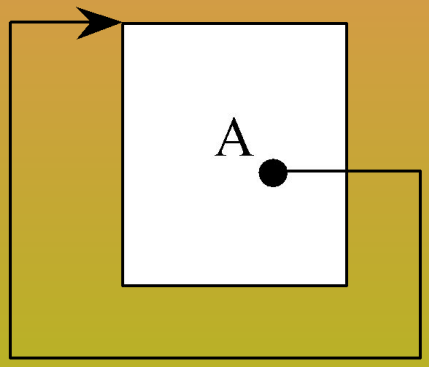
$$\int_{t_n}^{t_k} \frac{2t}{\sqrt{1-\sin(2t)}}\,dt$$

Значение определенного интеграла по формуле Симпсона вычисляется по формуле:

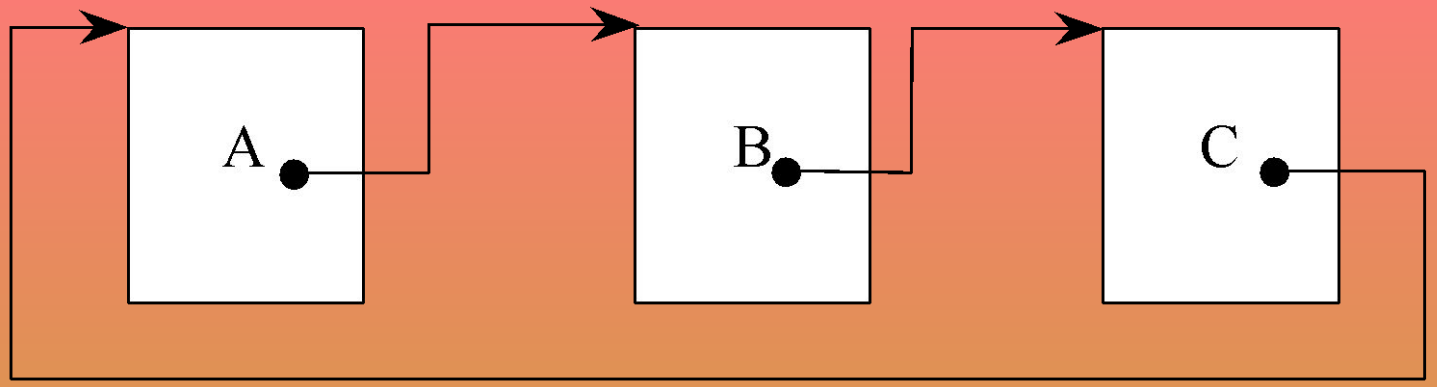
*IS=2*h/3*(0,5*F(A)+2*F(A+h)+F(A+2*h)+2*F(A+3*h)+...+2*F(B-h)+0,5*F(B)),*

Рекурсивные процедуры и функции

Рекурсия - способ организации подпрограммы, при котором она прямо или косвенно вызывает сама себя.



Прямая рекурсия



Косвенная рекурсия

Вызов рекурсивной подпрограммы ничем не отличается от вызова обычной подпрограммы.

Использование рекурсивных процедур. Программа реверса строки

```
Program Reverse_String;
Procedure Reverse;
Var
   ch:Char;
Begin
     If not EoLn then
     begin
          Read(ch);
          Reverse;
          Write(ch);
     end;
End;
Begin
    Reverse;
End.
```