

Операторы

Оператор - описание действий, изменяющих состояние программных объектов (например, значения переменных) или управляющих ходом выполнения программы.

Простые операторы:

- оператор присваивания
- оператор перехода
- пустой оператор
- оператор процедуры, оператор вызова функции

Сложные операторы:

- составной оператор
- выбирающие операторы
- операторы цикла.

Составной оператор

Составной оператор - это последовательность любого количества любых операторов, которая начинается служебным словом **begin** и заканчивается словом **end**

begin

<оператор1>;

<оператор2>;

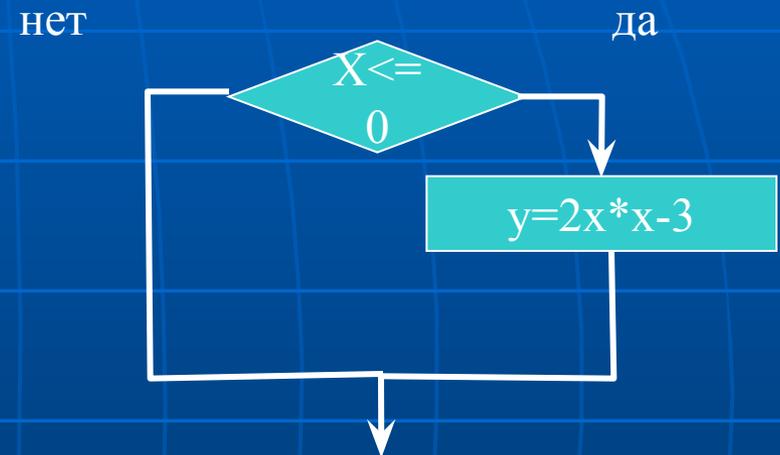
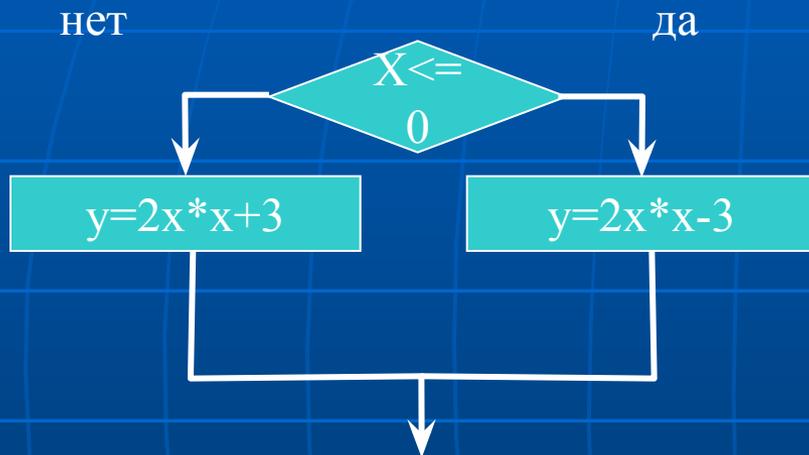
<оператор3>;

...

<операторN>

end

Условный оператор IF



```
IF <условие> THEN <Оператор 1>  
ELSE <Оператор 2>;
```

Вычисление функции

```
program l2;  
uses crt;  
const a=2;b=5;  
var  
x:integer;  
y:real;  
begin  
  clrscr;  
  write('vvedite X=');  
  readln(x);  
  if x >= 0 then  
    y := a+b;  
  if x < 0 then  
    y := a/b;  
  writeln('y=',y:3:3);  
  readkey;  
end.
```

```
program l2;  
uses crt;  
const a=2;b=5;  
var  
x:integer;  
y:real;  
begin  
  clrscr;  
  write('vvedite X=');  
  readln(x);  
  if x >= 0 then  
    y := a+b  
  else  
    y := a/b;  
  writeln('y=',y:3:3);  
  readkey;  
end.
```

сокращенная форма

Если Оператор2 - пустой, то
получается

сокращенная форма

```
if (<условие>) then  
<Оператор>
```

**При ложности условия оператор
просто пропускается.**

```
program l2;  
uses crt;  
const a=2;b=5;  
var  
x:integer;  
y:real;  
begin  
  clrscr;  
  write('vvedite X=');  
  readln(x);  
  if x >= 0 then  
    y := a + b  
  writeln('y=',y:3:3);  
  readkey;  
end.
```

Составные условия

В условных операторах

if B then P else Q

if B then P

и в операторе цикла

while B do P

в качестве условия B можно использовать не только отношения типа равенства и неравенства, но и более сложные составные условия. Все эти отношения заключаются в скобки:

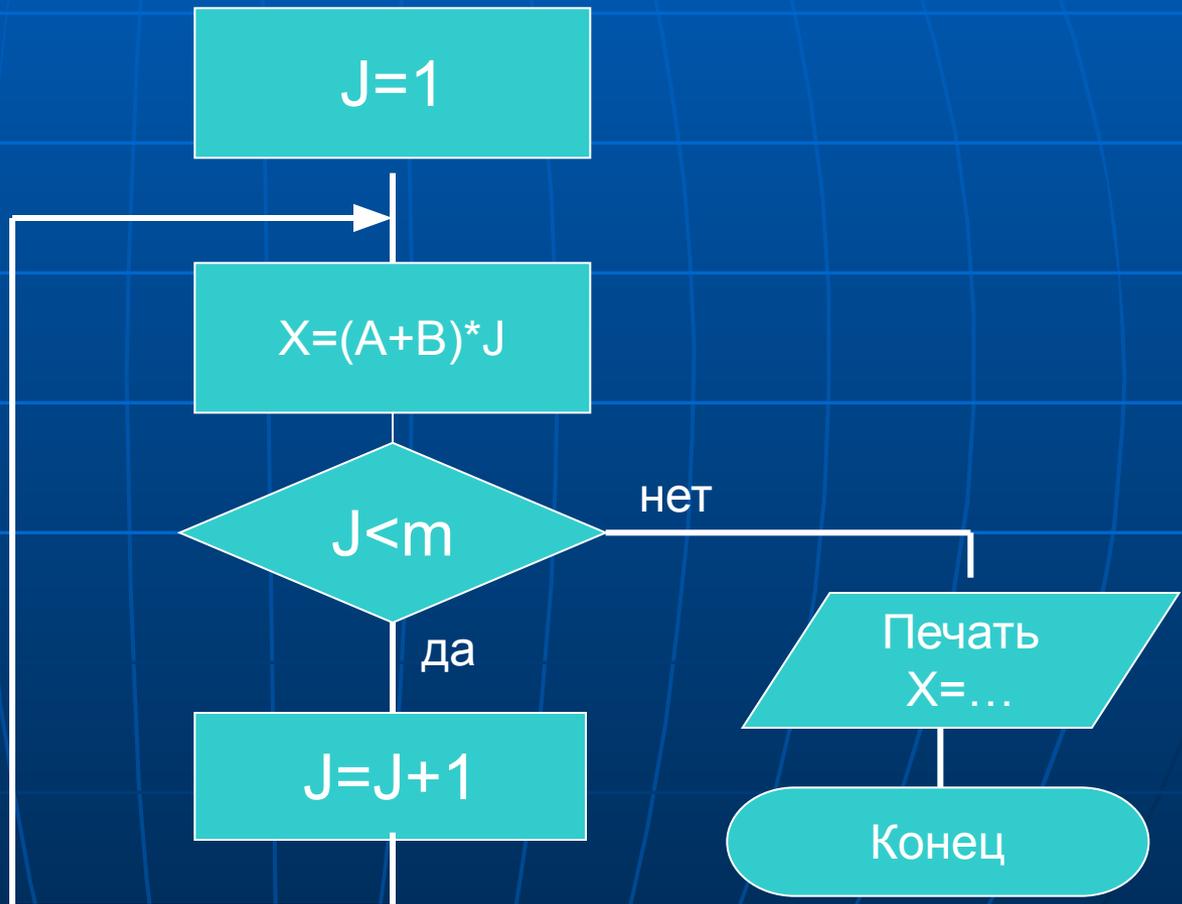
(a=b+1), (n>=0) и т.д.

Оператор Case

- **case** n_day **of**
1,2,3,4,5: day:='Рабочий день. ' ;
6: day:='Суббота!';
7: day:='Воскресенье!';
end;
- **case** n_day **of**
1..5: day:='Рабочий день.';
6: day:='Суббота!';
7: day:='Воскресенье!';
end;
- **case** n_day **of**
6: day:='Суббота!';
7: day:='Воскресенье!';
else day:='Рабочий день.';
end;

Циклические алгоритмы

- Цикл
- Итерация
- Тело цикла



Оператор цикла с предусловием (цикл "пока")

```
while <условие В> do  
    <оператор Р>
```

Оператор Р-называется
телом цикла.

Выполняется оператор цикла следующим образом: проверяется условие В, и если оно соблюдается то выполняется Р, а затем вновь проверяется условие В и т. Д. Как только на очередном шаге окажется, что условие В не соблюдается, то выполнение оператора цикла прекращается.

Табулирование функции

```
program l3;  
uses crt;  
const a=2;b=5;dx=0.5;dk=10;  
var  
y,x:real;  
begin  
  clrscr;  
  write('vvedite X=');  
  readln(x);  
  while x<dk{+dx/2} do  
    begin  
      y:=sqr(x);  
      writeln('x=',x:3:3,'    y=',y:3:3);  
      x:=x+dx;  
    end;  
  readkey;  
end.
```

Оператор цикла с постусловием

repeat

<оператор **P**>; {тело цикла }

until <условие **B**>;

Повторять тело цикла до тех пор пока не будет выполнено условие **B**.

Оператор Repeat

```
program l3;  
uses crt;  
const a=2;b=5;dx=0.5;dk=10;  
var  
y,x:real;  
begin  
  clrscr;  
  write('vvedite X=');  
  readln(x);  
  repeat  
    y:=sqr(x);  
    writeln('x=',x:3:3,'   y=',y:3:3);  
    x:=x+dx;  
  until x>dk;  
  readkey;  
end.
```

Цикл с заданным количеством повторений (цикл "для")

```
for I:=A to B do  
<оператор S>; {тело цикла}  
A <= B.
```

Здесь I – некоторая переменная
целого типа (integer), которая
называется параметром цикла.

A и **B** – выражения со значением
целого типа (integer).

- Оператор цикла выполняется так, сначала вычисляются значения выражений A и B и если $A \leq B$ то $I := A+1, A+2...$ и для каждого из этих значений выполняется оператор S .
- Если $A > B$ то выполнение цикла останавливается.

```
program l3;
uses crt;
const a=2;b=5;dx=0.5;dk=10;x0=5;
var
i:integer;
y,x:real;
begin
  clrscr;
  write('vvedite X=');
  readln(x);
  x:=x0;
  For i:=0 to 10 do
    begin
      y:=sqr(x);
      writeln('x=',x:3:3,'    y=',y:3:3);
      x:=x+dx;
    end;
  readkey;
end.
```

Вариант оператора цикла с параметром

```
for  $I := A$  downto  $B$  do  
  <оператор  $S$ >;  
   $A > B$ .
```

Здесь I принимает последовательно значения $A, A-1, A-2, \dots, B$ и для каждого из этих значений выполняется оператор S , если же $A < B$, то оператор S не выполняется ни разу.

Алгоритм умножения

- При возведении числа в степень в промежуточную ячейку записывают единицу.
- При умножение массива чисел в промежуточную ячейку записывают первый элемент массива.
- Пример: вычислить факториал числа $n!$.

```
Const
n = 5;
Var
n, I, p: integer;
Begin
    p:=1;
    for i:=1 to n do
        p:=p·i;
    // Вывод на печать n!
End.
```

Вложенные операторы цикла

- Получаются тогда, когда оператор, расположенный после do, сам является оператором цикла или сам содержит в себе оператор цикла.
- Пример. Пусть дано натуральное n и требуется вычислить сумму степеней

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

```
Const n=10;
Var i,j: integer;
    a, s, p: real ;
Begin
s:=0; {при суммировании ячейка обнуляется}
for i:=1 to n do
begin
a:=1/i;
p:=a;
for j:=2 to n do
begin
p:=p*a;
end;
s:=s+p;
end;
{ВЫВОД на печать s}
end.
```

Заполнение одномерного массива

```
Var m: Array[1..5] of Char;
```

```
  i: Integer;
```

```
Begin
```

```
  For i:=1 to 5 do
```

```
    m[i] := '*';
```

```
  For i := 1 to 10 do
```

```
    writeln( m [i] );
```

Датчик случайных чисел

- Для формирования одномерного или двухмерного массивов при программировании используется равномерный датчик случайных чисел `random(n)`. `N` – задает диапазон случайных чисел от $0 \div n-1$.
Для того, что бы датчик случайных чисел начинал работать с различных начальных значениях перед ним ставят процедуру

`randomize.`

Ввод одномерного массива с использованием датчика

Вариант 1:

```
TYPE
```

```
    VEK=ARRAY[1..10] OF REAL;
```

```
VAR A: VEK;
```

```
    X, I: INTEGER;
```

```
BEGIN
```

```
    RANDOMIZE; {каждый раз запускает датчик с  
другого числа}
```

```
    X:=10;
```

```
    FOR I:=1 TO 10 DO
```

```
        A[I]:=RANDOM(X);
```

Вариант 2:

VAR

A:ARRAY[1..10] OF REAL;

X, I: INTEGER;

BEGIN

RANDOMIZE; {каждый раз запускает датчик с другого числа}

X:=10;

FOR I:=1 TO 10 DO

A[I]:=RANDOM(X);

Сумма элементов массива

```
Var A: Array[1..10] of Integer;  
    s, i: Integer;  
Begin  
    Randomize;  
    For I := 1 to 10 do  
        A[i] := Random(100);  
    S := 0;  
    For I := 1 to 10 do  
        S := S + A[i];  
    writeln('сумма элементов равна',S);  
End;
```

Определение наименьшего
(наибольшего) среди чисел

Пример. Заданы n чисел
 $k^2 \cdot \sin(n+k/n)$, ($k=1,2,\dots,n$),
определить наименьшее
(наибольшее) значение.

Const $n=10$;

Var min, p : real; k : integer;

Begin

$\text{min} := \sin(n+1/n)$; {присваивается
 первое число массива }

 for $k := 2$ to n do

 begin

$p := \text{sqr}(k) \cdot \sin(n+k/n)$;

 If $p < \text{min}$ then $\text{min} := p$ {если знак отношения
 $>$, то находится наибольшее значение}

 end;

 Writeln('минимальный элемент', min);

end.

Заполнение матрицы

- `Var m: Array[1..5,1..5] of Char;`
 `I, j : Integer;`
Begin
 For i:=1 to 5 do
 For j:=1 to 5 do
 IF i=j then m[I, j] := '+'
 else m[I, j] := '*';

Вывод матрицы

```
For i := 1 to 10 do  
  begin  
    For j := 1 to 10 do  
      write( M [i, j]:3 );  
    writeln;  
  end;
```

Транспонирование матриц

```
FOR i := 1 to 2 do  
  FOR J := 1 to 3 DO  
    At[j, i] := a[i, j];
```

Ввод двумерного массива с использованием датчика

Вариант 1:

TYPE

MAS=ARRAY[1..5,1..5] OF REAL;

VAR

M: MAS;

I, J: INTEGER;

BEGIN

RANDOMIZE;

FOR I:=1 TO 5 DO

FOR J:=1 TO 5 DO

M[I,J]:=RANDOM(5);

Вариант 2.

VAR

M:ARRAY[1..5,1..5] OF REAL;

I, J: INTEGER;

BEGIN

RANDOMIZE;

FOR I:=1 TO 5 DO

FOR J:=1 TO 5 DO

M[I,J]:=RANDOM(5);

Работа с матрицей

```
program l2;  
uses crt;  
var  
mas: array[1..5,1..5] of integer;  
i,j:integer;  
begin  
  clrscr;  
  Randomize;  
  For i:=1 to 5 do  
    begin  
      for j:=1 to 5 do  
        begin  
          mas[i,j]:=random(10)+10;  
          write(mas[i,j]:3);  
        end;  
        writeln;  
      end;  
    end;  
end;
```

```
writeln;  
writeln;
```

```
For i:=1 to 5 do  
begin  
  For j:=1 to 5 do  
  begin  
    if i=j  
      then mas[i,j]:=0;  
    write(mas[i,j]:3);  
  end;  
  writeln;
```

- end;
- readkey;
- end.

Упорядочивание (сортировка) массива

Упорядочить массив X_1, X_2, \dots, X_n – это значит расположить все числа массива в возрастающем порядке т. е.

$$X_1 < X_2 < \dots < X_n.$$

На первом месте должен быть наименьший элемент, на втором месте – наименьший из всех остальных элементов и т. д. Для этого вводим индекс K по которому будем искать наименьший элемент из X_i, X_{i+1}, \dots, X_n .

Как только наименьший элемент занял свое место, он сразу выводится из массива.

Для перестановки $X[i]$ с $X[k]$ привлекается дополнительная переменная V

$$V := X[i]; X[i] := X[k];$$
$$X[k] := V;$$

Алгоритм называется

алгоритмом сортировки выбором.

```
CONST N=5;
```

```
TYPE
```

```
    POR=ARRAY[1..N] OF INTEGER;
```

```
VAR
```

```
    X:POR;
```

```
    V:INTEGER;
```

```
    I,J,K:INTEGER;
```

```
BEGIN
```

```
  FOR I:=1 TO N DO
```

```
    X[I]:=RANDOM(10);
```

```
  FOR I:=1 TO N DO
```

```
    BEGIN
```

```
      K:=I;
```

```
      FOR J:=I+1 TO N DO
```

```
        IF X[J]<X[K] THEN K:=J;
```

```
        V:=X[I]; X[I]:=X[K]; X[K]:=V;
```

```
      end;
```

```
      Вывод на экран
```

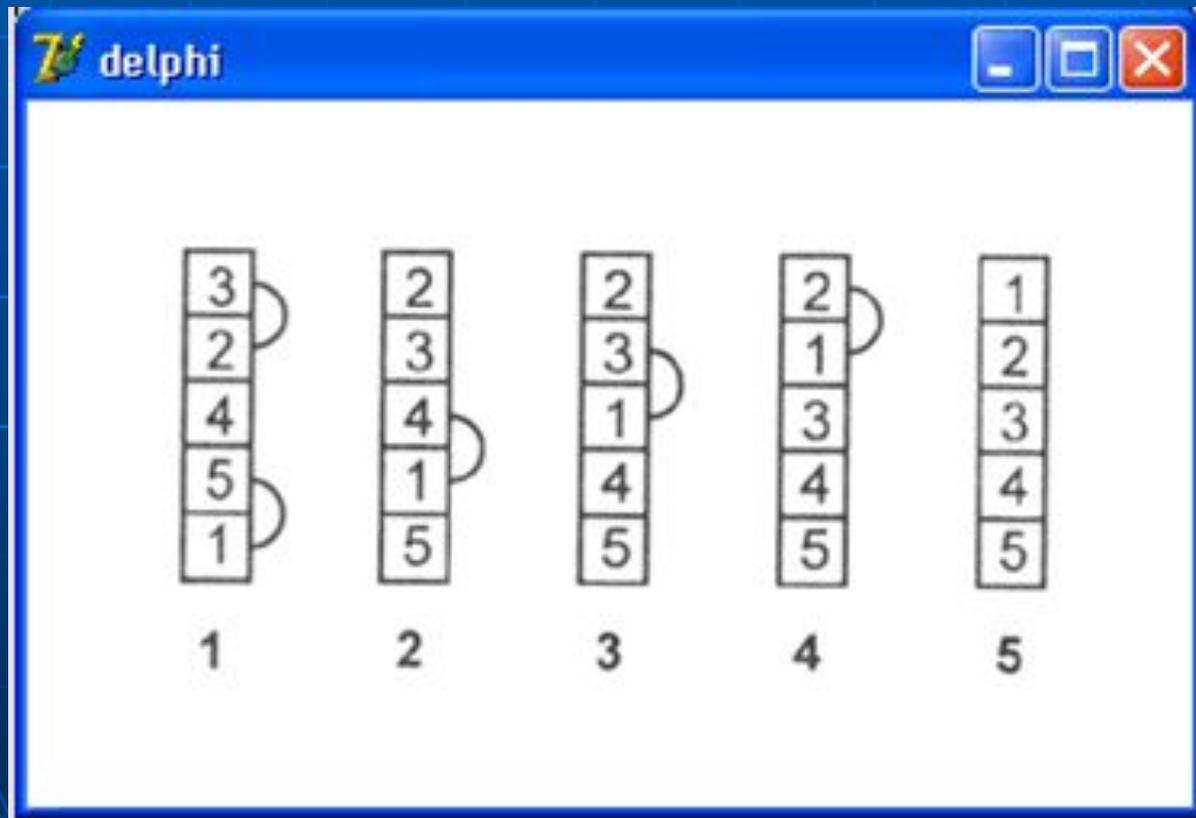
```
end;
```

```
END.
```

Сортировка методом обмена

- В основе алгоритма лежит обмен соседних элементов массива. Каждый элемент массива, начиная с первого, сравнивается со следующим, и если он больше следующего, то элементы меняются местами. Таким образом, элементы с меньшим значением продвигаются к началу массива (всплывают), а элементы с большим значением — к концу массива (тонут). Поэтому данный метод сортировки обменом иногда называют методом "пузырька". Этот процесс повторяется столько раз, сколько элементов в массиве, минус единица.

На рис. цифрой 1 обозначено исходное состояние массива и перестановки на первом проходе, цифрой 2 — состояние после перестановок на первом проходе и перестановки на втором проходе, и т. д.

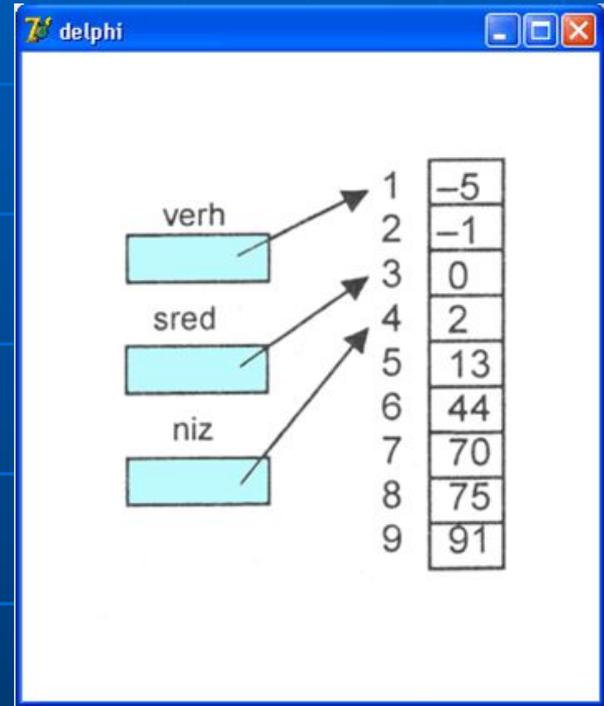
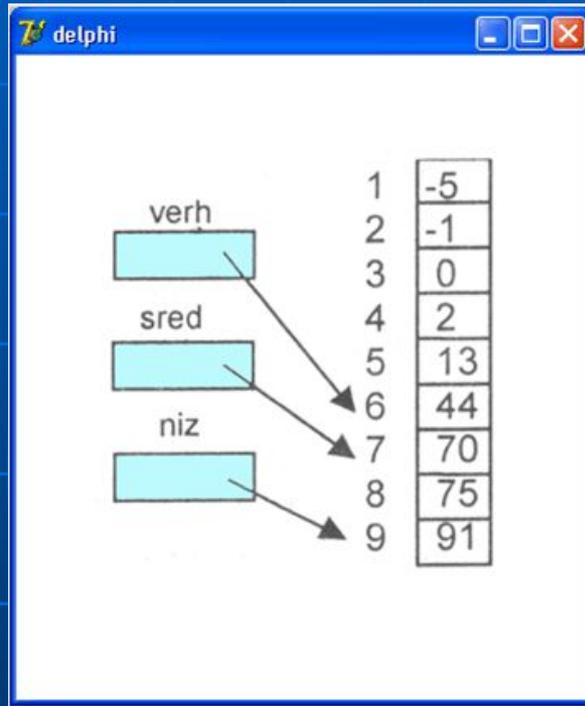
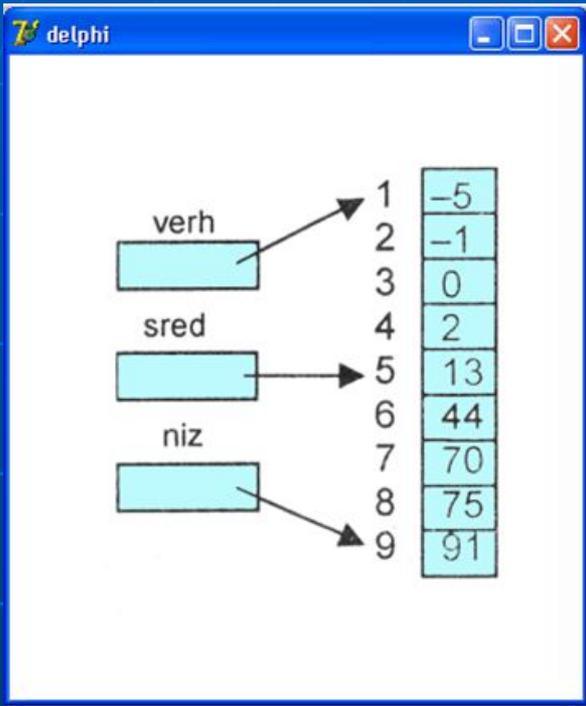


- var
- Form2: TForm2;
- Mas: array[1..10]of integer;
- i,k,buf,z:integer;
- flag:boolean;
- implementation
- {\$R *.dfm}
- procedure TForm2.Button1Click(Sender: TObject);
- begin
- randomize;
- z:=0;
- for i:=1 to 10 do
- begin
- Mas[i]:=random(10);
- memo1.Lines.Add(inttostr(Mas[i]))
- end;

- repeat
- flag:=false; // Пусть в текущем цикле нет обменов
- for k:=1 to 9 do
- if Mas[k] > Mas[k+1] then
- begin // обменяем k-й и k+1-й элементы
- buf := Mas[k]; Mas[k] := Mas[k+1];
- Mas[k+1] := buf;
- flag := TRUE;
- end;
- z:=z+1;
- until not flag ;
- for i:=1 to 10 do
- Memo2.Lines.Add(inttostr(Mas[i]));
- label1.Caption:='Массив отсортирован за'+
- #13 +inttostr(z) + ' шагов'
- end;

Метод бинарного поиска

- Метод (алгоритм) бинарного поиска реализуется следующим образом:
 1. Сначала образец сравнивается со средним (по номеру) элементом массива.
- Если образец равен среднему элементу, то задача решена.
- Если образец больше среднего элемента, то это значит, что искомый элемент расположен ниже среднего элемента (между элементами с номерами $sred+1$ и niz), и за новое значение $verb$ принимается $sred+1$, а значение niz не меняется.
- Если образец меньше среднего элемента, то это значит, что искомый элемент расположен выше среднего элемента (между элементами с номерами $verb$ и $sred-1$), и за новое значение niz принимается $sred-1$, а значение $verb$ не меняется.



- var
- Form1: TForm1;
- i,ver,niz,sred,obr:integer;
- Mas: array [0..10] of integer;
- Flag:boolean;

- implementation

- {\$R *.dfm}

- procedure TForm1.Button1Click(Sender: TObject);
- begin
- memo1.Clear;
- flag:=false;
- niz:=10;
- ver:=1;
- if edit1.Text=""
- then showmessage('введите число')
- else
- begin
- obr:=strtoint(edit1.Text);

- for i:=1 to 10 do
- begin
- Mas[i]:=i-1;
- memo1.Lines.Add(inttostr(Mas[i])) ;
- end;
- i:=0;
- repeat
- i:=i+1;
- sred:=trunc((niz-ver)/2+ver);
- if Mas[sred]=obr then
- flag:=true
- else if obr<Mas[sred] then niz:=sred-1
- else ver:=sred+1;
-
- until (ver>niz) or flag;
- if flag then label1.Caption:='совпадение с номером '+
- inttostr(sred)+ #13+
- 'на '+#9+ inttostr(i)+#9 + 'шаге'
- else
- label1.Caption:='такого числа в массиве нет ';
- end;
- end;
- end.

Подпрограммы. Функции и процедуры

Функция, выполняя некоторые действия, вычисляет **единственное значение**, которое является основным результатом ее работы.

Отработав, функция должна вернуть этот результат вызвавшей ее программе.

Процедура просто выполняет какие-то действия, **не возвращая никакого значения**. Именно эти действия являются главным результатом ее работы.

При этом процедура может изменить, если необходимо, значения некоторых объектов программы, к которым она имеет доступ.

Структура описания функции:

```
function <имя> (<СписокФормПарам>) :<типРезульт>;  
    <Разделы описаний>  
begin  
    <Операторы>  
    <имя> := <выражение> //ОБЯЗАТЕЛЬНЫЙ ОПЕРАТОР!!  
end;
```

Структура описания процедуры:

```
procedure <имя> (<СписФормПарам>) ;  
  < Разделы описаний >  
begin  
  < Операторы >  
end;
```

ФОРМАЛЬНЫЕ параметры -
это не сами данные,
передаваемые в
подпрограмму, а только их
описание, которое
содержит информацию для
подпрограммы о
характеристиках этих
данных и о действиях над
ними.

ФАКТИЧЕСКИЕ параметры -
данные, фактически
передаваемые подпрограмме
при ее вызове. Эти данные
должны быть описаны в
вызывающей программе.

*Порядок перечисления и
другие характеристики
формальных и фактических
параметров должны
соответствовать друг другу.*

ЛОКАЛЬНЫЕ параметры - это данные, которые описаны внутри самой подпрограммы. Эти параметры "недолговечные", они "живут" только во время работы подпрограммы. При начале работы подпрограммы они как бы "создаются" (в соответствии со смыслом описания), а при окончании работы "уничтожаются".

ПРИМЕРЫ:

```
1 function LatinChar(Litera: char): boolean;  
  begin  
  LatinChar := ((Litera >= 'A') and (Litera <='Z'))  
  or  
    ((Litera >= 'a') and (Litera <='z'))  
  end;
```

```
2. function kolich_cifr(chislo : integer):  
   integer;  
   var  
       i: integer;  
begin  
       i := 0;  
       while ((chislo div 10) <> 0) do  
       begin  
           chislo := chislo div 10;  
           i := i + 1  
       end;  
       kolich_cifr := i + 1  
end;
```

Пример процедуры:

```
procedure Obmen (var Znach1, Znach2:integer);  
  var  
    Znach: integer;  
begin  
  Znach := Znach1;  
  Znach1 := Znach2;  
  Znach2 := Znach  
end;  
var  
  Number1, Number2: integer;  
begin  
  .....  
  Number1 := 5;  
  Number2 := 9;  
  Obmen (Number1, Number2);
```

Операции над целыми числами

Операция **DIV** (**division**-деление);

Операция **MOD** (**modulus**-мера);

Эти операции имеют по два целых аргумента (операнда). Если **a** и **b** неотрицательны и **b** ≠ 0, то **a div b** – это частное от деления.

$$17 \text{ div } 3 = 5; 3 \text{ div } 5 = 0.$$

Оператор выбора

Оператор выбора позволяет выбрать одно из нескольких возможных продолжений программы.

Параметр, по которому осуществляется выбор, служит

КЛЮЧ ВЫБОРА -

это выражение любого порядкового типа, кроме типов REAL и STRING.

Структура оператора выбора:

CASE < ключ выбора > OF < список выбора >

ELSE < оператор > END.

Здесь CASE – случай, OF – из
ELSE – иначе, END –
зарезервированные слова.

< список выбора > - одна или более
конструкций вида:

< константа выбора > : < оператор >

<константа выбора> - это константа того же типа, что выражение <ключ выбора>.

<оператор> - произвольный оператор Паскаля.

Оператор выбора работает следующим образом. Вначале вычисляется значение выражения <ключ выбора>, а затем в последовательности операторов <список выбора> отыскивается такой, которому предшествует константа, равная вычисленному значению.

Найденный оператор выполняется, после чего оператор выбора завершат свою работу. Если в списке выбора не будет найдена константа, соответствующая вычисленному значению ключа выбора, управление передается оператору, стоящему за словом ELSE.

Пусть задано описание переменной I – как переменная целого типа, то оператор выбора запишется как

```
CASE I OF
```

```
1: Y:=SIN(X);
```

```
2: Y:=COS(X);
```

```
3: Y:=EXP(X);
```

```
4: Y:=LN(X);
```

```
END;
```

При выполнении этой программы могут возникать ошибки, если значение переменной I не равно 1,2,3,4, то программа завершается аварийно. Для предотвращения подобной ситуации обычно используют совместно условный оператор и оператор выбора:

```
IF (I >= 1) AND (I <= 4)
```

```
THEN
```

```
    CASE I OF
```

```
        1: Y := SIN(X);
```

```
        2: Y := COS(X);
```

```
        3: Y := EXP(X);
```

```
        4: Y := LN(X);
```

```
    END;
```

Все константы выбора внутри одного оператора выбора обязательно должны быть различными, поскольку в противном случае возникает неоднозначность в выборе исполняемого оператора.

Описание объектов

Классы и объекты

- Объекты объединяют в единое целое данные и средства действий над ними. Подобно переменным, объекты, используемые в программе, должны быть описаны. Для описания объектов используются классы.

- Класс – это средство описания типа объекта, поэтому он помещается разделе описания типов **type**.
Описав в программе один раз класс, в дальнейшем можно **создавать** необходимое количество экземпляров этого класса - объектов. Основными свойствами классов являются инкапсуляция, наследование и полиморфизм. Эти три понятия являются основными для ООП.

Инкапсуляция

- скрывание данных и методов внутри использующего их класса. Это означает, что данные и методы описываемого класса доступны для использования только ему.

Наследование

- это возможность порождения новых классов от уже описанных. В этом случае данные и методы родительского класса автоматически включаются в порожденный класс и нет необходимости их описывать повторно. Исходный класс будем называть предком, а порожденный от него класс-наследник назовем потомком.

Полиморфизм

- ЭТО ВОЗМОЖНОСТЬ ИСПОЛЬЗОВАТЬ ОДИНАКОВЫЕ ИМЕНА ДЛЯ МЕТОДОВ РАЗНЫХ КЛАССОВ С ОБЩИМ ПРЕДКОМ, ИМЕЮЩИХ ОДИНАКОВЫЙ СМЫСЛ, НО ПО РАЗНОМУ ВЫПОЛНЯЮЩИХСЯ.

Структура описания класса

- Описание класса напоминает описание записей, в которых наряду с описаниями данных существуют и описания методов. Ниже приведена структура описания класса.

- `<имя класса> = class (<имя наследуемого класса>)`
- *// Для классов, описываемых в среде Delphi, здесь*
- *// помещаются описания компонентов Delphi и заголовки*
- *// методов-обработчиков событий*

- **protected**
- *//Здесь помещаются описания элементов класса, которые*
- *// доступны напрямую в пределах данного модуля, а также*
- *// в классах-наследниках в других модулях*

- **private**
- *// Здесь помещаются описания элементов класса, которые*
- *// доступны напрямую только в пределах данного модуля*

- **public**
- *// Здесь помещаются описания элементов класса, которые*
- *// доступны напрямую в пределах любого модуля программы*
- **end;**

Например, описание класса для выделения разрядов целого числа может иметь вид (помещается в секцию **Interface** модуля):

- **Interface**
- **type**
- TRazriadyCelogo = **class**
- **private**
- Celoe : integer;
- Razriady : **array** [1..10] **of** integer;
- NomerRazriada : 1..10;

- **public**
- **procedure** PoluchitCeloe(Chislo: integer);
- **procedure** VydelitRazriady;
- **function** ZnachenieRazriada (N:integer)
:integer;
- **end;**
- Теперь можно описать объект
(переменную) этого типа:
- **var**
- RazriadCelogo : TRazriadyCelogo;

- Полное описание объявленных в классе процедур помещается в секцию **Implementation** модуля:
- **Implementation**
- **procedure**
TRazriadyCelogo.PoluchitCeloe(Chislo:
integer);
- **begin**
- Celoe := Chislo;
- **end;**

- **procedure** TRazriadyCelogo.VydelitRazriady;
- **var**
- i, CelChislo : integer;
- **begin**
- CelChislo := Celoe;
- i := 1;
- **while** ((CelChislo **div** 10) <> 0) **and** (i < 10) **do**
- **begin**
- Razriady[i] := CelChislo **mod** 10;
- CelChislo := CelChislo **div** 10;
- i := i + 1;
- **end;**
- Razriady[i] := CelChislo
- **end;**

- **function** ZnachenieRazriada
(N:integer) :integer;
- **begin**
- ZnachenieRazriada := Razriady[N]
- **end;**