Pascal

• • •

ETH Pascal, Turbo Pascal, Borland Pascal, Delphi, Object Pascal, PascalABC.NET

Реализации языка Pascal

- **ETH Pascal** – Никлаус Вирт, **1970 г.**
- **Turbo (Borland) Pascal** - Андерс Хейлсберг, **1983 г.**
- **Delphi (Object Pascal)** - Андерс Хейлсберг, **1996 г.**
- **Free Pascal / Lazarus**
- **Pascal ABC.NET** - разработка ведется коллективом кафедры алгебры и дискретной математики факультета математики, механики и компьютерных наук ЮФУ, **2006 г.**

Алфавит и лексика

Общая структура программы

Алфавит и лексика

Алфавит

- a, b, ..., z и A, B, ..., Z, символ подчеркивания "_"
- цифры от 0 до 9
- + - * / = < > [ ] . , ( ) : ; ^ @ { } $ # ‘

Лексический состав языка

- ключевые слова;
- идентификаторы;
- изображения (неименованные константы);
- знаки операций;
- разделители;
- комментарии;
- директивы компилятора.

Ключевые слова

(служебные или зарезервированные слова)

```
and  array  as  auto  begin case  class const constructor destructor div
do downto  else  end  event  except  extensionmethod file finalization
   finally for foreach function  goto if implementation in inherited
initialization interface is label  lock  mod  nil  not  of  operator or
procedure program property raise record repeat sealed  set sequence shl
 shr sizeof  template  then  to try type typeof until uses  using var
                    where  while  with  xor
```

- используются для оформления конструкций языка;
- не могут использоваться как идентификаторы.

Идентификаторы

Правила построения:

- первым символом должна быть буква;
- не должны совпадать с ключевыми словами;

```
x, a2, m_1
```

```
1x, begin, ж
```

```
mass, alpha, button, number
```

так лучше, чем

```
m, a, b, n
```

Общая структура программы

```
program <имя программы>;
uses <список подключаемых модулей>;
label <список меток>;
const <список констант>;
type <описание типов>;
var <описание переменных>;
<описание процедур>;
<описание функций>;
begin
   <операторы>;
end.
```

```
program Example;
var
  a,b : integer;
  x : real;
begin
  readln(a,b);
  x := a/b;
  writeln(x);
end.
```

Комментарии

```
{ Это
комментарий }
(* Это
тоже комментарий *)

var  n : integer; // Количество итераций

sum := 0;
for k := 1 to 100 do begin
     read(x);
//   if x<0 then x := 0;
     sum := sum + x;
end;
```

Типы данных

Классификация, характеристики, изображения

Что определяет тип данных?


Типы данных
Простые
Порядковые
Целочисленные
Логический
Символьный
Вещественные
Структурные
Строковый
Массив
Запись
Множество
Класс
Файловые
Процедурные

Целочисленные типы / Integer Types

Тип	Диапазон значений	Размер, байт
byte	0 … 255	1
shortint	-128 … 127	1
word	0 … 65535	2
integer	-32768 ... 32767	2
longint	-2147483648 ... 2147483647	4
int64	-9223372036854775808 … 9223372036854775807	8

```
123   -12
```

десятичный формат

~~`10.0`~~

```
$01AF   $FFFF   $1A
```

шестнадцатеричный формат

`+` `-` `*` `div` `mod` `>` `<` `=` `<>`

Вещественные типы / Floating Types

Тип	Диапазон значений	Количество значащих цифр	Размер, байт
real	$-1.7 \cdot 10^{308} .. 1.7 \cdot 10^{308}$	15-16	**8**
single	$-3.4 \cdot 10^{38} .. 3.4 \cdot 10^{38}$	7-8	**4**
double	$-1.7 \cdot 10^{308} .. 1.7 \cdot 10^{308}$	15-16	**8**
extended	$-1.1 \cdot 10^{4932} .. 1.1 \cdot 10^{4932}$	19-20	**10**

```
123.0    0.0012    -34.781
```

формат с фиксированной точкой

```
1.23E2    1.2E-3    -3.4781E1
```

формат с плавающей точкой

+ **-** ***** **/** **>** **<** **=** **<>**

Вещественные типы / Floating Types

Значащие цифры: 23.1240 0.06546 -0.00041

Формат с плавающей точкой:

$$M \cdot 10^p$$

<мантисса>E<показатель степени>

<мантисса>e<показатель степени>

real, double:

0.0000000000000000e+000

16 разрядов

0.0000053186294 → 5.3186294000000000e-006

531862.94 → 5.3186294000000000e+006

Логический тип / Boolean Type

Идентификатор типа: **Boolean**

TRUE (истина)

FALSE (ложь)

`not` `or` `and` `xor`

`>` `<` `=` `<>`

логические операции

Значения *логического* типа являются результатом вычислений **условных** и **логических выражений**.

Символьный тип / Character Type

Идентификатор типа: **Char**

```
'a', 'b', 'ф', 'ж', '+', '2', '@', '[', '. '
```

или **#<ASCII/Unicode код>**

```
#65, #13, #27
```

`>` `<` `=` `<>`

```
'a'<'b' // True
```

PascalABC.NET: тип char занимает 2 байта и хранит Unicode-символ. Символы реализуются типом System.Char платформы .NET.

Строковый тип / String Type

Идентификатор типа: **String**

```
'This is a string', 'Это строка'
```

`+` конкатенация (слияние, склеивание)

`>` `<` `=` `<>`

```
'Зима' + ' ' + 'идет' // 'Зима идет'
```

PascalABC.NET

```
'Число =' + 23 // 'Число равно 23'
```

Сравнение строк на неравенство осуществляется **лексикографически**: s1 < s2 если для первого несовпадающего символа с номером i s1[i]<s2[i] или все символы строк совпадают, но s1 короче s2.

Интервальный тип / Interval Type

<имя типа> = <константа 1> .. <константа 2>

```
month = 1..12
```

```
latins = 'a'..'z'
```

Интервальные типы используются, например,

при описании индексов массивов.

Разделы описаний

Описание констант, переменных и типов

Описание констант

Синтаксис:

```
const
  <Идентификатор> = <Значение>;
  <Идентификатор> = <Выражение>;
```

Примеры:

```
const
  Min = 0;
  Max = 100;
  e = 2.7;
  SpecChar = '\';
  HelpStr = 'Нажмите клавишу F1';
  OK = True;
```

Описание констант

Примеры:

```
const
  Min = 0;
  Max = 100;
  e = 2.7;
  SpecChar = '\';
  HelpStr = 'Нажмите клавишу F1';
  OK = True;
  // теперь используем введенные выше константы
  Interval = Max - Min + 1;
  e2 = e*e;
  BigHelpStr = HelpStr + ' для подсказки';
```

Описание переменных

Синтаксис:

```
var
  <Идентификатор> = <Тип>;
  <Идентификатор1, Идентификатор2> = <Тип>;
```

Примеры:

```
var
  X : real;                  //вещественная переменная
  i, j, k : integer;         //три целочисленных
переменных
  M : array[1..5] of byte;   //целочисленный массив
```

Описание переменных

PascalABC.NET, Object Pascal: **инициализация при описании**

```
var
  n : integer = 100;
  v : real = 12.63;
```

PascalABC.NET: **автоопределение типа**

```
var
  n := 1;     //переменная n получит целочисленный тип
  v := 1.0;   //переменная v получит вещественный тип
```

PascalABC.NET: **внутриблочное описание**

```
for var i := 1 to 10 do print(i);
        //переменная i описана внутри оператора цикла
```

Описание типов

Синтаксис:

```
type
  <Идентификатор> = <Описание типа>;
```

Примеры:

```
type
  vector = array[1..10] of real;
  month = 1..12;
  pinteger = ^integer;
  IntFunc = function(x: integer) : integer;
```

Эквивалентность и совместимость типов

Типы эквивалентны, если:

- T1 и T2 — одно и то же имя типа.
- Тип T2 описан с использованием типа T1 равенством вида type T2=T1; или последовательностью подобного вида равенств.

Примеры:

```
type
   TVector = array[1..10] of integer;
var
   a1, b1 : TVector;
   a2 : array[1..10] of integer;
   b2 : array[1..10] of integer;
```

Эквивалентность и совместимость типов

Типы совместимы, если:

- T1 и T2 эквивалентны;
- T1 и T2 принадлежат к целым типам;
- T1 и T2 принадлежат к вещественным типам;
- один из типов - поддиапазон другого или оба - поддиапазоны некоторого типа;
- T1 и T2 - множества с совместимыми базовыми типами.

Типы совместимы по присваиванию, если:

- T1 и T2 совместимы;
- T1 - вещественного типа, T2 – целого;
- T1 - строкового типа, T2 - символьного;
- …

Совместимость по присваиванию

```
var
   a : integer;
   x : real;
   c : char;
   s : string;
begin
  a := 10; ✓
  a := 10.0; □
  x := 10.0; ✓
  x := a; ✓
  a := 10 div 2; ✓
  a := 10/2; □
  c := 'a'; ✓
  c := 'ab'; □
  s := 10; □
  s := '10'; ✓
end.
```

Выражения

Арифметические, логические, строковые

Выражения: основные понятия

Выражение – это формальное правило для вычисления некоторого **значения**.

Выражения: основные понятия

Выражения
(по типу значения)

- Арифметические
- Логические
- Строковые

Результатом вычисления арифметического выражения является число, тип которого зависит от типов операндов, составляющих это выражение, и вида выполняемых операций.

Арифметические выражения

Символ операции	Операция	Тип операндов	Тип результата
+	сложение	целочисленный вещественный	целочисленный вещественный
–	вычитание	целочисленный вещественный	целочисленный вещественный
*	умножение	целочисленный вещественный	целочисленный вещественный
/	деление	целочисленный вещественный	вещественный вещественный
div	целочисленное деление	целочисленный	целочисленный
mod	остаток от деления	целочисленный	целочисленный

- Если все операнды целочисленные, то операции сложения, вычитания, умножения дают значение также целочисленного типа. **Операция деления "/" всегда дает вещественный тип результата**.
- Если в выражение присутствуют целочисленные операнды, имеющие различные типы (например, byte и integer), то значение выражения будет иметь целочисленный тип с наибольшим диапазоном (в данном случае integer).
- Если среди операндов хоть один имеет вещественный тип, то значение выражения будет также вещественным.

Базовые математические функции

Функция	Тип аргумента	Тип значения	Назначение
abs(x)	целый вещественный	целый вещественный	модуль числа x
sin(x)	целый вещественный	вещественный	синус x радиан
cos(x)	целый вещественный	вещественный	косинус x радиан
arctan(x)	целый вещественный	вещественный	главное значение арктангенса x
sqr(x)	целый вещественный	целый вещественный	квадрат числа x
sqrt(x)	целый вещественный	вещественный	квадратный корень из x
exp(x)	целый вещественный	вещественный	число e в степени x
ln(x)	целый вещественный	вещественный	натуральный логарифм x
trunc(c)	целый вещественный	целый	целая часть x
frac(x)	целый вещественный	целый	дробная часть x
int(x)	целый вещественный	вещественный	целая часть x
round(x)	целый вещественный	целый	округленное до ближайшего целого x
random	–	вещественный	случайное число в интервале от 0 до 1
random(x)	целый	целый	случайное число в интервале от 0 до x
power(x,y)	целый вещественный	целый вещественный	возведение x в степень y

Арифметические выражения

$$\frac{1}{1+|\sin x|}$$

```
1/(1+abs(sin(x)))
```

$$\frac{a+b^2}{\cos(a+b)-\sqrt{\pi}}$$

```
(a+sqr(b))/(cos(a+b)-sqrt(pi))
```

$$\frac{2s^3}{ln(1+s^5)-e^{-s^4}}$$

```
2*sqr(s)*s/(ln(1+power(s,5))-exp(-power(s,4)))
```

Так лучше →

```
s3 := s*s*s; s4 := s3*s; s5 := s4*s;

2*s3/(ln(1+s5)-exp(-s4))
```

Выражения: основные понятия

Выражения
(по типу значения)

- Арифметические
- Логические
- Строковые

Результатом вычисления логического выражения является одно из двух логических значений (истина или ложь)

Логические выражения

Логические выражения могут состоять из:

- логических констант;
- переменных логического типа boolean;
- условных выражений;
- логических операций;
- круглых скобок.

***Условное выражение (условие)* –** совокупность переменных и констант простых типов, объединенных знаками операций сравнения (отношения). Часто условное выражение также называется логическим выражением.

Символ операции	Операция
=	равно
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно
<>	не равно

```
x<2
ch='Y'
a+b<>0
sin(x)>=0.5
```

Логические выражения

Логические операции, применяются только к операндам логического типа и вырабатывают результат также логического типа. Всего имеется четыре логических операции:

- **not** – логическое отрицание (инверсия);
- **and** – логическое умножение (конъюнкция);
- **or** – логическое сложение (дизъюнкция);
- **xor** – исключающее сложение (строгая дизъюнкция).

Таблица истинности

A	B	not A	A and B	A or B	A xor B
F	F	T	F	F	F
F	T	T	F	T	T
T	F	F	F	T	T
T	T	F	T	T	F

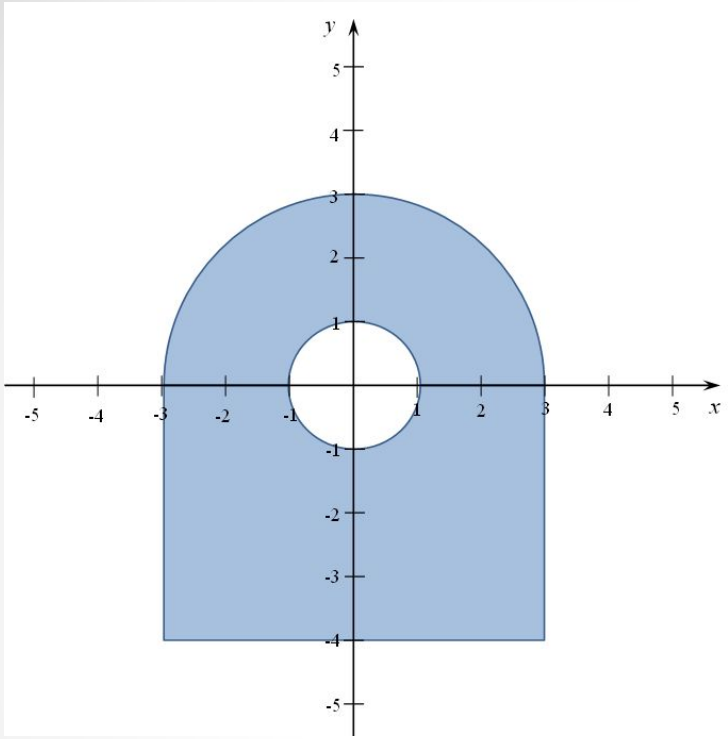
Логические выражения

```
not (L1 and L2)
(x>=0) and (x<=10)
(ch='y') or (ch='Y')
(x+1>0) and (x+1<10) or (y>0) and (y<2)
```

Приоритет выполнения операций:

- вычисление функций;
- унарный минус;
- арифметическое умножение, деление;
- арифметическое сложение, вычитание;
- логическое отрицание;
- логическое умножение;
- логическое сложение, исключающее сложение;
- операции сравнения;

Использование логических выражений



$x^2+y^2 \le 9$ - круг радиусом 3 (множество **A**);

$\begin{cases} -3 \le x \le 3, \\ -4 \le y \le 0. \end{cases}$ - прямоугольник (множество **B**);

$x^2+y^2 \ge 1$ - вся плоскость за исключением точек внутри круга радиусом 1 (множество **C**).

$$S = (A \cup B) \cap C$$

На языке алгебры логики:

$LA = x^2+y^2 \le 9$

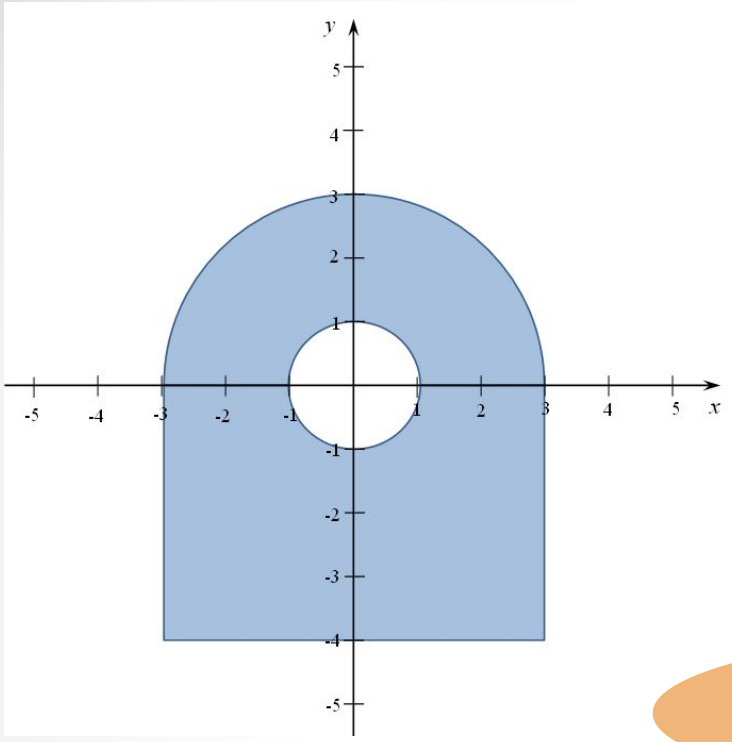
$LB = (-3 \le x \le 3) \wedge (-4 \le y \le 0)$

$LC = x^2+y^2 \ge 1$

$LS = (LA \vee LB) \wedge LC$

```
LA := sqr(x)+sqr(y)<=9;
LB := (x>=-3) and (x<=3) and
      (y>=-4) and (y<=0);
LC := sqr(x)+sqr(y)>=1;
LS := (LA or LB) and LC;
```

Использование логических выражений




```
LA := sqr(x)+sqr(y)<=9;
LB := (x>=-3) and (x<=3) and
      (y>=-4) and (y<=0);
LC := sqr(x)+sqr(y)>=1;
LS := (LA or LB) and LC;
```

Можно и без использования промежуточных переменных

```
LS := ((sqr(x)+sqr(y)<=9) or (x>=-3) and (x<=3) and (y>=-4)
      and (y<=0)) and (sqr(x)+sqr(y)>=1);
```

Использование логических выражений

```
LS := ((sqr(x)+sqr(y)<=9) or (x>=-3) and (x<=3) and (y>=-4)
      and (y<=0)) and (sqr(x)+sqr(y)>=1);
```

Так лучше с точки зрения скорости вычислений.

Выражения с операциями and и or вычисляются **по короткой схеме**:

- в выражении **x and y** если x ложно, то все выражение ложно, и y не вычисляется;
- в выражении **x or y** если x истинно, то все выражение истинно, и y не вычисляется.

Организация ввода и вывода данных

Read, Write, Print и т.д.

Вывод на экран

```
Write(<список вывода>)  или  Writeln (<список вывода>)
```

Элементы списка вывода – ***переменные, константы и выражения***

- «Старый» Pascal – **только простые типы**
- PascalABC.NET – **простые и структурные типы**

Примеры вывода на экран:

```
Write('2+2 = ');
Writeln(2+2);
var x := 2.5;
writeln(x);
Writeln('Значение x = ', x);
Writeln('sin(',x,') = ', sin(x));
```

```
2+2 = 4
2.5
Значение x = 2.5
sin(2.5) = 0.598472144103957
```

Вывод на экран (PascalABC.NET)

Дополнительные процедуры:

```
Print(<список вывода>)  и  Println (<список вывода>)
```

Аналогичны процедурам Write и Writeln, но дополнительно выводят на экран символ "пробел" между значениями элементов списка вывода.

Удобно использовать для вывода отладочной информации.

Примеры вывода на экран:

```
Print('Значение x=', x);
Print(x, sin(x))
```

```
Writeln('Значение x= ', x);
Writeln(x, ' ', sin(x))
```

Ввод с клавиатуры

```
Read(<список ввода>)  или  Readln (<список ввода>)
```

Элементы списка ввода – ***только переменные***

```
var
  n: integer;
  a, b, c : real;
begin
  Read( n );          // ввод значений переменной n
  Read( a, b, c );    // ввод значений переменных a, b и c
end.
```

```
10 [Enter]
0.5  12.5 -2.6 [Enter]
```

```
10 [Enter]
0.5 [Enter]
12.5 [Enter]
-2.6 [Enter]
```

Ввод с клавиатуры

Хорошее правило: выводить на экран поясняющий текст

```
var
  n: integer;
  a, b, c : real;
begin
  Write('Введите значение n: ');
  Read( n );
  Write('Введите значение переменных a, b и c: ')
  Read( a, b, c );
end.
```

```
Введите значение n: 10[Enter]
Введите значение переменных a, b и c: 0.5  1.5 -2.6[Enter]
```

Ввод с клавиатуры (PascalABC.NET)

Вывод поясняющего текста и ввод значения вместе – **очень удобно!**

```
var
  n : integer;
  x : real;
begin
  n := ReadInteger('Введите значение n:');
  x := ReadReal('Введите значение x:');
  ...
end.
```

а так еще лучше:

```
begin
  var n := ReadInteger('Введите значение n:');
  var x := ReadReal('Введите значение x:');
  ...
end.
```

Операторы

if, case, for, while, repeat

Простой и составной оператор

Оператор в программе – это единое и неделимое предложение, выполняющее какое-либо ***действие***.

```
a := 10; b := a*5; //операторы присваивания
Write( a, b );     //оператор вызова процедуры Write
```

Составной оператор (блок) – это последовательность операторов, перед которой стоит слово begin, а после – end.

Слова begin и end - *операторные скобки*.

```
s:=0; p:=1;
for i:=1 to 10 do begin       //составной оператор образует
                    p:=p*i;   //тело цикла, состоящее из
                    s:=s+p    //двух операторов присваивания
                  end;
```

Оператор присваивания

Синтаксис:

```
<переменная> := <выражение>
```

```
a := 12.5;      //переменной a присваивается значение
                //константы 12.5
x := a;     //переменной x присваивается значение
                 //переменной a
z := x + 1;     //переменной z присваивается значение
                 //выражения
```

```
z := z + 5;
```

Условный оператор

Синтаксис:

полная форма

```
if <условие> then <оператор 1> else <оператор 2>
```

сокращенная форма

```
if <условие> then <оператор>
```

допускается вложенность

```
if <условие 1> then
  if <условие 2> then <оператор 1>
                 else <оператор 2>
```

```
if a<b then min := a
       else min := b;
```

Условный оператор

Вычисление кусочно-заданной функции:

$$f(x)=\begin{cases}1+2x+ln(x^2-1), & x<-1;\\ -2, & x=-1;\\ \frac{\sin x}{x+1}, & x>-1.\end{cases}$$

```
program example;
var
  f : real;
begin
  Writeln('Вычисление значения кусочно-заданной функции.');
  var x := ReadReal('Введите значение аргумента x =');


  if x<-1 then f := 1+2*x+ln(sqr(x)-1)
          else if x>-1 then f := sin(x)/(x+1)
                       else f := -2;


  Writeln('Значение функции f = ', f)
end.
```

Оператор множественного выбора

Синтаксис:

```
case <ключ> of
    <набор значений 1> : <оператор 1>;
    <набор значений 2> : <оператор 2>;
              . . .
    <набор значений n> : <оператор n>;
    else <альтернативный оператор>
end
```

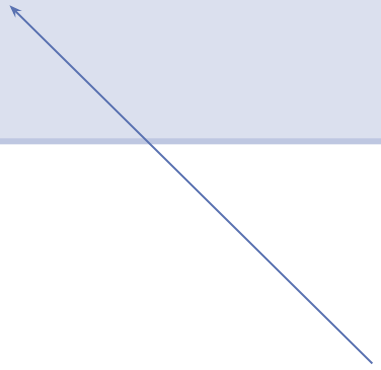
```
case DayOfWeek of
  1..5 : writeln('Будний день');
  6,7  : writeln('Выходной день');
end;
```

Оператор множественного выбора

Ключ – это переменная или выражение **только порядкового типа** (например, целочисленного или символьного)

Наборы значений – не должны пересекаться

```
case i of
  2,5  : write(1);
  4..6 : write(2);
end;
```



Ошибка!

Оператор цикла с параметром **for**

Синтаксис:

первая форма

```
for переменная := <нач.значени> to <кон.значение> do оператор
```

вторая форма

```
for переменная := <нач.значени> downto <кон.значение> do оператор
```

В **PascalABC.NET** переменную-параметр цикла можно (**и нужно!**) описывать непосредственно в заголовке:

```
for var i := 1 to 10 do write(i);
```

Переменная-параметр цикла может иметь **любой порядковый тип**. При этом начальное и конечное значения должны быть **совместимы по присваиванию** с переменной-параметром цикла.

Оператор цикла с параметром

«Старый» Pascal:

```
var
 i, j : integer;
begin
  writeln('Таблица умножения');
  for i := 1 to 10 do begin
    for j := 1 to 10 do
                write(i*j, ' ');
    writeln;
  end;
end.
```

PascalABC.NET:

```
begin
  writeln('Таблица умножения');
  for var i := 1 to 10 do begin
    for var j := 1 to 10 do
          print(i*j);
    writeln;
  end;
end.
```

Оператор цикла **foreach** (PascalABC.NET)

Синтаксис:

```
foreach переменная in контейнер do оператор
```

```
foreach переменная : тип in контейнер do оператор
```

```
foreach var переменная in контейнер do оператор
```

Примеры:

```
var
  students: set of string := ['Иванов','Петров','Сидоров'];
  numbers: array of integer := (3,4,5);
begin
  foreach s : string in students do print(s);
  writeln;
  foreach var x in numbers do print(x);
end.
```

Операторы циклов **while** и **repeat**

Оператор цикла с предусловием:

```
while условие do оператор
```

```
while x<>0 do begin
  print(x mod 10);
  x := x div 10;
end;
```

Оператор цикла с постусловием:

```
repeat
  <оператор 1>
  <оператор 2>
  ...
  <оператор n>
until условие
```

Обычно оператор repeat используют в ситуациях, где условие нельзя проверить, не выполнив тело цикла. Например:

```
repeat
  read(x);
until x<>0;
```

Циклические алгоритмы

Основные понятия, особенности организации, примеры

Основные понятия

- **Цикл** - многократное повторение заданной последовательности действий.
- **Тело цикла** - повторяемая последовательность действий.
- **Итерация** - единичное выполнение тела цикла
(от лат. ***iteratio*** – повторение).
- **Условие выхода** – логическое выражение определяющее, будет ли в очередной раз выполняться итерация, или цикл завершится.

Виды и структура циклов

Виды:

- детерминированные;
- недетерминированные (итерационные).

Структура:

- подготовка (инициализация) цикла;
- выполнение тела цикла;
- модификация параметров;
- проверка условия окончания (или продолжения) цикла.

Типы структур:

- цикл с пред условием;
- цикл с пост условием;
- цикл с параметром.

Циклические алгоритмы

Большая часть времени исполнения программы приходится на циклы, поэтому:

- Можно сделать без цикла – сделай без цикла
- Можно вынести действия за цикл – вынеси за цикл
- Можно сделать без вложенного цикла – очень хорошо
- Можно поменять местами внешний и вложенный цикл – сделай так, чтобы внешний цикл содержал меньше итераций, чем вложенный

Есть и другие способы оптимизации, например

- распараллеливание циклов.

Вычисление сумм и произведений

```
//вычисление суммы и произведения
var
   x, S, P : real;
      n, i : integer;
begin
   x := ReadReal('x=');
   n := ReadInteger('n=');
   S := 0;
   P := 1;
   for i:=1 to n do begin
      S := S + sin(x*i);
      P := P * (i+x)
   end;
   S := S/n;
   Writeln('S = ', S, 'P = ', P)
end.
```

$$S = \frac{1}{n}\sum_{i=1}^{n} sin(ix)$$

$$P = \prod_{i=1}^{n} (i + x)$$

Вычисление сумм и произведений

```
//вычисление суммы факториалов (решение 1)
var
  n, i, j, S, f : integer;
begin
  n := ReadInteger('n=');
  S := 0;
  for i:=1 to n do begin
     f := 1;
     for j:=1 to i do f:=f*j;
     S := S+f;
  end;
  Writeln('S = ', S)
end.
```

$$S = \sum_{i=1}^{n} i! = 1 + 1 \cdot 2 + 1 \cdot 2 \cdot 3 + \cdots + 1 \cdot 2 \cdot 3 \times \cdots \times n$$

Вычисление сумм и произведений

```
//вычисление суммы факториалов (решение 2)
var
  n, i, S, f : integer;
begin
  n := ReadInteger('n=');
  S := 0; f := 1;
  for i:=1 to n do begin
     f := f*i;
     S := S+f;
  end;
  Writeln('S = ', S)
end.
```

$$S = \sum_{i=1}^{n} i! = 1 + 1 \cdot 2 + 1 \cdot 2 \cdot 3 + \cdots + 1 \cdot 2 \cdot 3 \times \cdots \times n$$

Табулирование функции

```
//табулирование функции (решение 1)
var
  x, y, a, b, h : real;
begin
  //ввод a, b, h
  x := a;
  while x <= b+h/2 do begin
     y := ...;
     writeln(x,' ',y);
     x := x + h;
  end;
end.
```

Табулирование функции

```
//табулирование функции (решение 2)
var
  x, y, a, b, h : real;
begin
  //ввод a, b, h
  var n := trunc((b-a)/h);
  for var i:=0 to n do begin
     x := a+i*h;
     y := ...;
     writeln(x,' ',y);
  end;
end.
```

Защищенный ввод

```
//ввод с «защитой»
var
 x : integer;
begin
  repeat
    Write('Введите число в интервале [0..10]');
    Readln(x);
    if (x<0) or (x>10) then Writeln('Ошибочка!');
  until (x>=0) and (x<=10);
  . . .
end.
```

Вычисление значений функций

Как вычисляются функции?

sin(x), **cos**(x), $\mathbf{e}^{x}$, **ln**(x) и т.д.

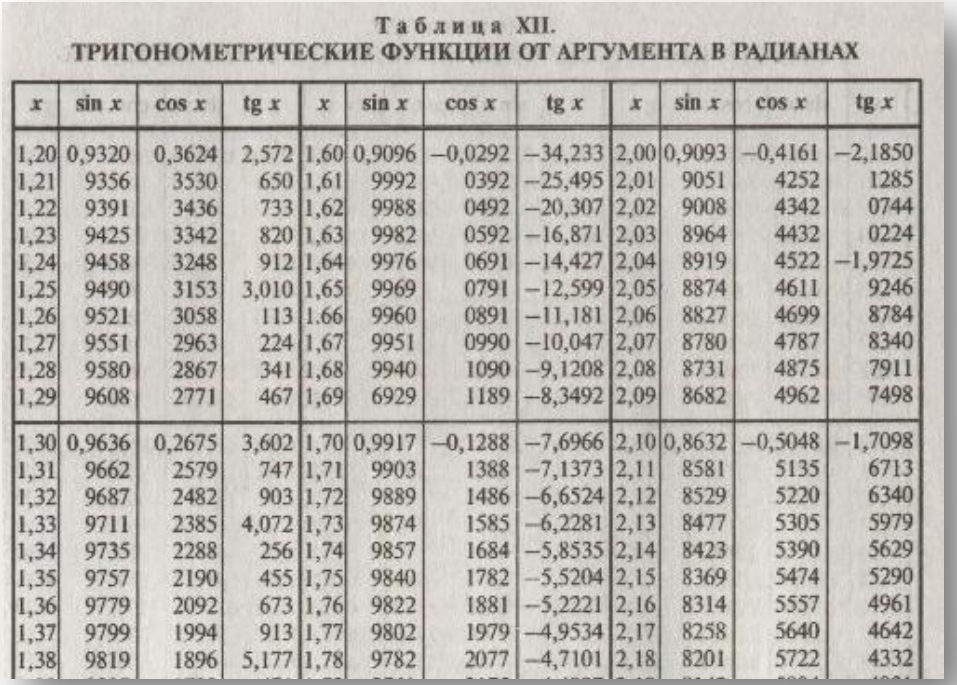


Таблица XII.
ТРИГОНОМЕТРИЧЕСКИЕ ФУНКЦИИ ОТ АРГУМЕНТА В РАДИАНАХ

x	sin x	cos x	tg x	x	sin x	cos x	tg x	x	sin x	cos x	tg x
1,20	0,9320	0,3624	2,572	1,60	0,9096	−0,0292	−34,233	2,00	0,9093	−0,4161	−2,1850
1,21	9356	3530	650	1,61	9992	0392	−25,495	2,01	9051	4252	1285
1,22	9391	3436	733	1,62	9988	0492	−20,307	2,02	9008	4342	0744
1,23	9425	3342	820	1,63	9982	0592	−16,871	2,03	8964	4432	0224
1,24	9458	3248	912	1,64	9976	0691	−14,427	2,04	8919	4522	−1,9725
1,25	9490	3153	3,010	1,65	9969	0791	−12,599	2,05	8874	4611	9246
1,26	9521	3058	113	1,66	9960	0891	−11,181	2,06	8827	4699	8784
1,27	9551	2963	224	1,67	9951	0990	−10,047	2,07	8780	4787	8340
1,28	9580	2867	341	1,68	9940	1090	−9,1208	2,08	8731	4875	7911
1,29	9608	2771	467	1,69	6929	1189	−8,3492	2,09	8682	4962	7498
1,30	0,9636	0,2675	3,602	1,70	0,9917	−0,1288	−7,6966	2,10	0,8632	−0,5048	−1,7098
1,31	9662	2579	747	1,71	9903	1388	−7,1373	2,11	8581	5135	6713
1,32	9687	2482	903	1,72	9889	1486	−6,6524	2,12	8529	5220	6340
1,33	9711	2385	4,072	1,73	9874	1585	−6,2281	2,13	8477	5305	5979
1,34	9735	2288	256	1,74	9857	1684	−5,8535	2,14	8423	5390	5629
1,35	9757	2190	455	1,75	9840	1782	−5,5204	2,15	8369	5474	5290
1,36	9779	2092	673	1,76	9822	1881	−5,2221	2,16	8314	5557	4961
1,37	9799	1994	913	1,77	9802	1979	−4,9534	2,17	8258	5640	4642
1,38	9819	1896	5,177	1,78	9782	2077	−4,7101	2,18	8201	5722	4332

Вычисление значений функций

Как вычисляются функции?

sin(x), **cos**(x), $\mathbf{e}^{x}$, **ln**(x) и т.д.

Представление функции в виде степенного ряда:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Вычисление значений функций

Степенной ряд Тейлора – замечательная вещь!

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f'''(x_0)}{3!}(x - x_0)^3 + \cdots$$

Если x_0 = 0, то ряд называют также рядом Маклорена:

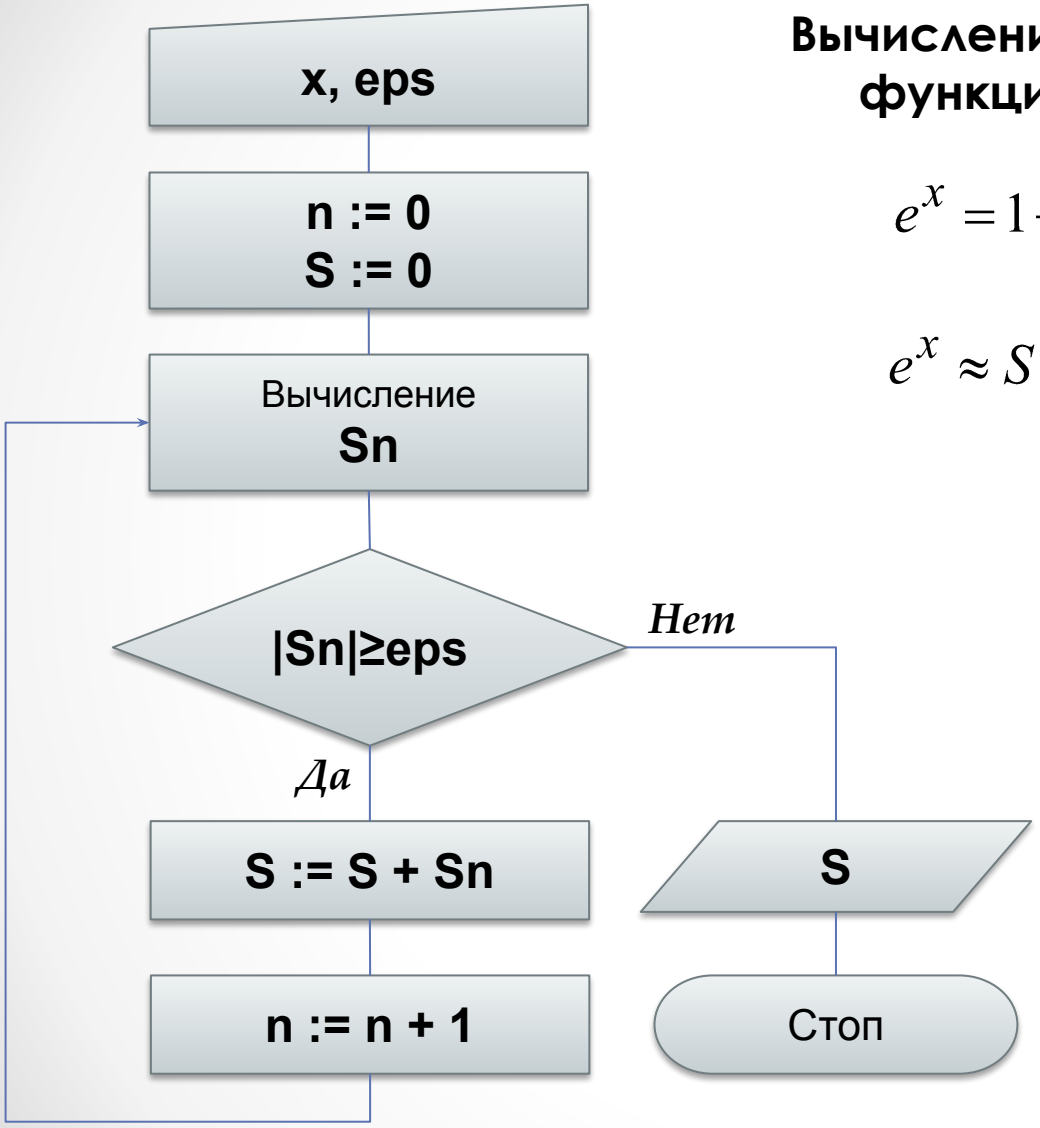
$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \cdots$$

Например, для функции $f(x) = e^x$:

$$f(0) = f'(0) = f''(0) = f'''(0) = \cdots = 1$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

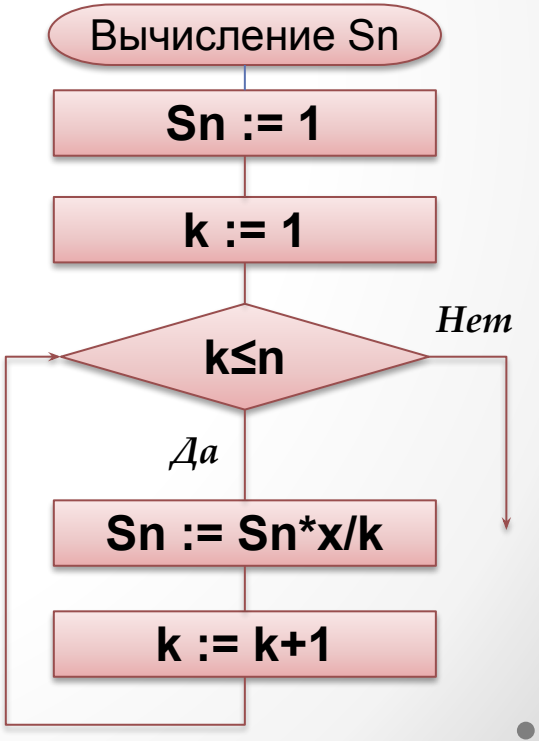
Вычисление значений функций



Вычисление приближенного значения функции с заданной точностью

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

$$e^x \approx S = S_0 + S_1 + S_2 + \ldots = \sum_{n=0} S_n$$



Вычисление значений функций

Вычисление приближенного значения функции с заданной точностью

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

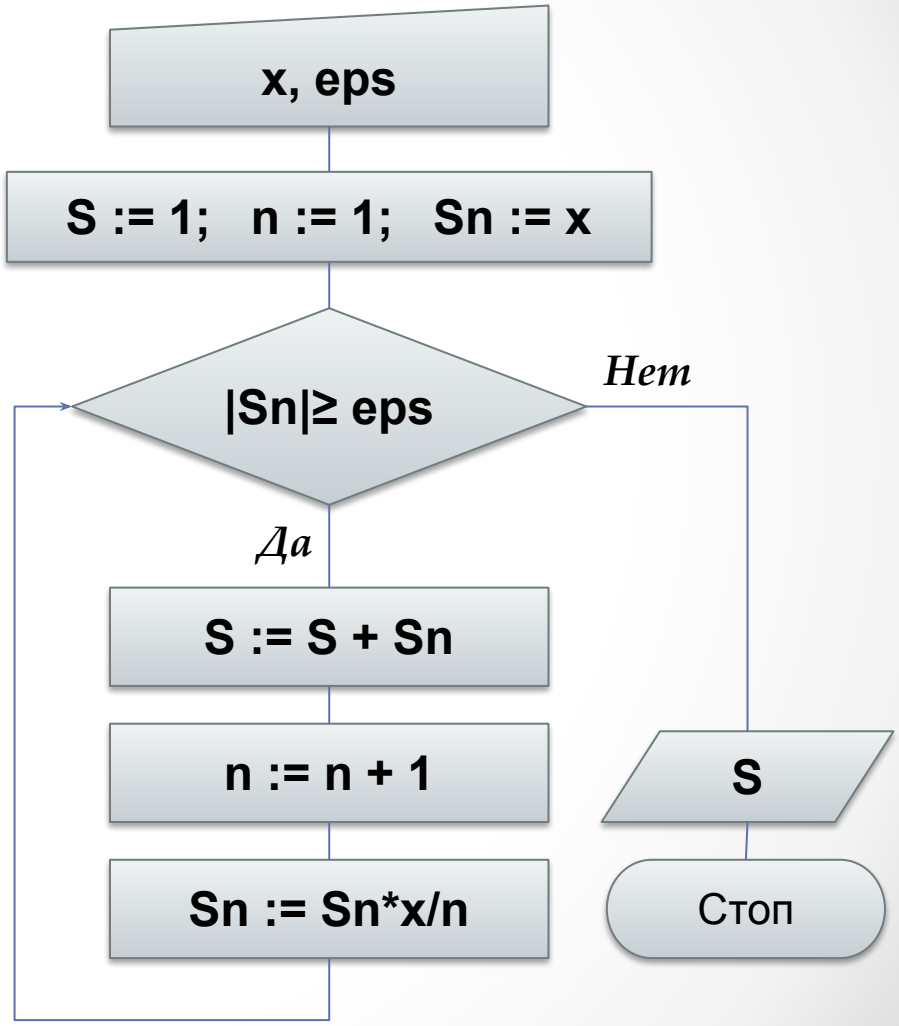
$$e^x \approx S = S_0 + S_1 + S_2 + \ldots = \sum_{n=0} S_n$$

Рекуррентное соотношение:

$$S_1 = S_0 \cdot x, \quad S_2 = S_1 \cdot \frac{x}{2},$$

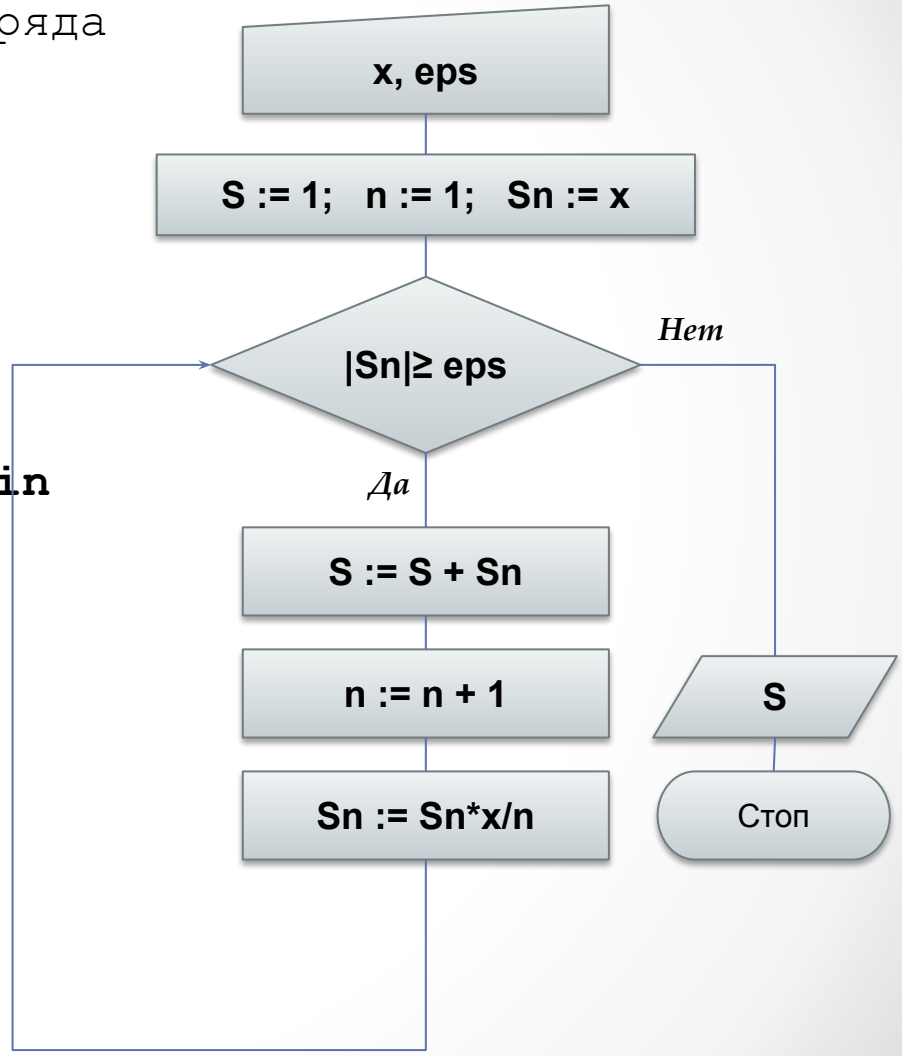
$$S_3 = S_2 \cdot \frac{x}{3}, \quad \text{и т.д.}$$

$$S_n = S_{n-1} \cdot \frac{x}{n}$$



Вычисление значений функций

```
//вычисление частичной суммы ряда
const
 eps = 1e-8;
var
  n, S, Sn, x : integer;
begin
  x := ReadReal('x=');
  S := 1; Sn := x; n := 1;
  while abs(Sn >= eps) do begin
     S := S+Sn;
     n := n+1;
     Sn := Sn*x/n;
  end;
  Writeln('S = ', S)
end.
```



Структуры данных

Структуры данных

Физическая память компьютера – пронумерованная последовательность ячеек (с прямым доступом)

Обрабатываемые данные:

- числа
- матрицы
- текст
- таблицы
- и др.

ТИПЫ ДАННЫХ

Структуры данных

Структура данных — это множество элементов данных и связей между ними.

Структура данных (в программировании) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных.

Основные операции:

- добавление,
- поиск,
- изменение
- удаления.

Структуры данных

Структуры данных (логические):

- вектор;
- матрица;
- множество;
- список;
- таблица;
- очередь;
- стек;
- и др.

Простые типы данных:

- числовой;
- символьный;
- логический;
- указатель.

Структурные типы данных:

- массив;
- множество;
- запись;
- и др.

Массивы

Массив — упорядоченная структура однотипных данных (элементов).

Свойства:

- **однотипность элементов**;
- **упорядоченность** - в памяти компьютеры элементы массива хранятся последовательно друг за другом; доступ к любому элемент возможен путем указания его порядкового ***номера (индекса)***.

Характеристики:

- **Размерность** – количество используемых индексов (одномерные, двумерные и т. д.)
- **Размер** – общее количество элементов;
- **Форма**

Массивы

По времени выделения памяти:

- статические;
- динамические.

Статические массивы

array[*тип индекса*] **of** *базовый тип*

```
var
  a : array[1..10] of real;
  b : array[0..9] of real;
  g : array[1..3] of array[1..5] of real;
  s : array[1..3,1..5] of real;
```

Массивы

Динамические массивы

1. Описание динамического массива

array of тип элементов //одномерный массив

array [,] of тип элементов //двумерный массив

2. Выделение памяти под динамический массив

```
var
  a: array of integer;
  b: array [,] of real;
begin
  a := new integer[5];
  b := new real[4,3];
end.
```

Массивы: ввод-вывод элементов

Ввод элементов одномерного статического массива:

```
const n = 10
var
  A : array[1..n] of integer;
begin
   for var i:=1 to n do begin
     write('Введите значение элемента A[',i,']: ');
     readln(A[i]);
   end;
   //вывод массива в строку
   writeln('Введенный массив:');
   for i:=1 to n do write(A[i],' ');
end.
```

Массивы: ввод-вывод элементов

Ввод элементов одномерного динамического массива:

```
var
  A : array of integer;
begin
   var n := ReadInteger('Введите количество элементов:');
   A := new integer[n];
   for var i:=0 to n-1 do begin
     write('Введите значение элемента A[',i+1,']: ');
     readln(A[i]);
   end;
   //вывод массива в строку
   writeln('Введенный массив:');
   for i:=0 to n-1 do write(A[i],' ');
end.
```

Массивы: ввод-вывод элементов

Вывод элементов одномерного массива:

«Старый» Паскаль:

```
for i:=0 to n-1 do write(A[i],' ');
```

PascalABC.NET:

```
write(A);
```

Массивы: поиск элементов

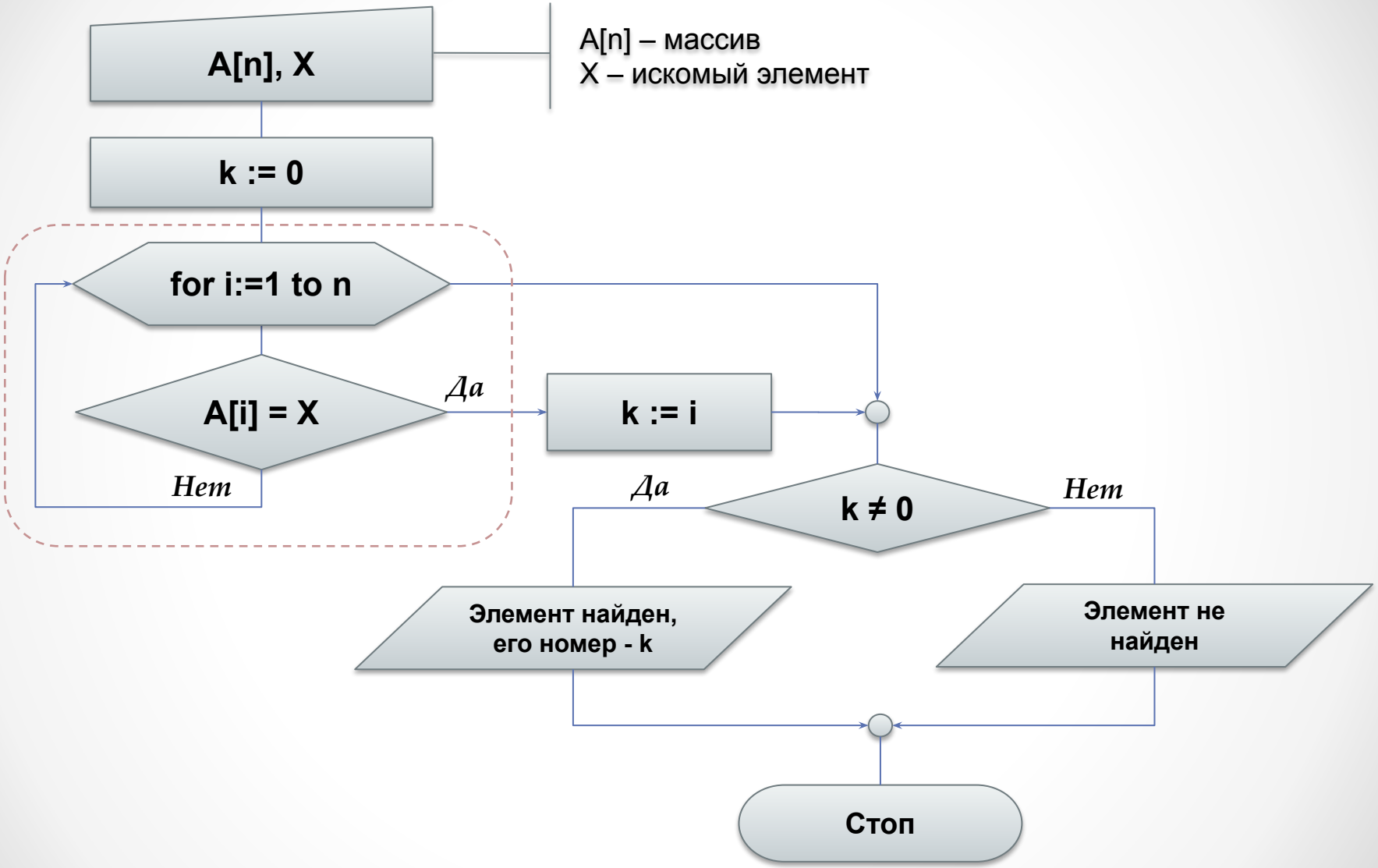
Линейный поиск – поиск элементов в неупорядоченном массиве.

Суть алгоритма: перебираем последовательно элементы; каждый элемент проверяем на соответствие критерию поиска.

В худшем случае придется перебрать все элементы.

Сложность алгоритма (число элементарных действий) растет линейно с ростом числа элементов.

Линейный поиск



Массивы: поиск элементов

Поиск заданного элемента в одномерном массиве:

```
const n = ...;
var
   A : array[1..n] of integer;
   k, i, x : integer;
begin
   //ввод элементов массива A
   //ввод значения искомого элемента x
   k := 0;
   for i:=1 to n do
      if A[i]=x then begin
          k := i;
          break;
      end;
   if k<>0 then writeln('Найден элемент с индексом: ',k)
           else writeln('Заданный элемент не найден');
end.
```

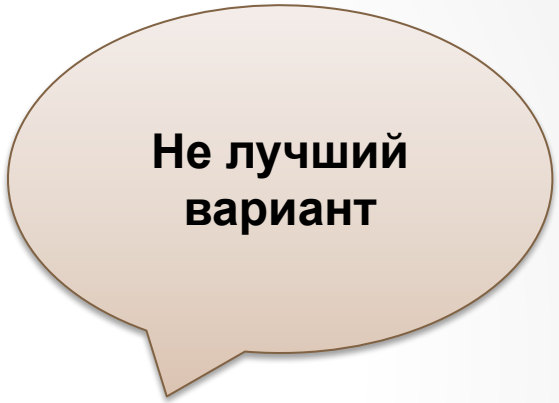
Поиск заданного элемента в двумерном массиве:

```
const n = ...; m = ...; //размер массива
var           A : array[1..n, 1..m] of integer;
    row, col, x : integer;
label
  exit_loop;
begin
   //ввод элементов массива A
   //ввод значения искомого элемента x
   var found := false;
   for var i:=1 to n do
    for var j:=1 to n do
      if A[i,j]=x then begin
          row := i; col := j; found := true;
          goto exit_loop;
      end;
   exit_loop:
   if found then writeln('Найден элемент с индексами: ',row, '  ', col)
            else writeln('Заданный элемент не найден');
end.
```

Массивы: поиск элементов

Поиск минимального и максимального элементов в одномерном массиве:

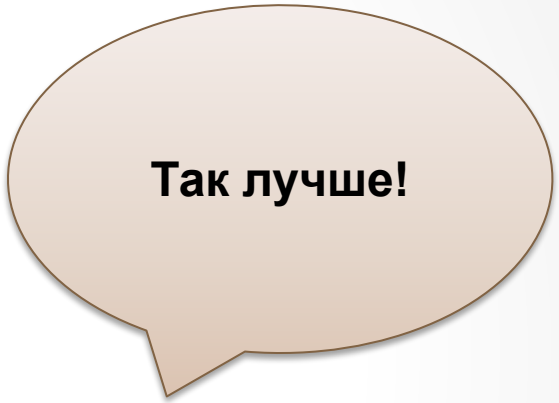
```
const n = ...;
var
   A : array[1..n] of integer;
   i, Amin, Amax: integer;
begin
   //ввод элементов массива A
   Amin := A[1]; Amax := A[1];
   for i:=2 to n do begin
      if A[i]<Amin then Amin := A[i];
      if A[i]>Amax then Amax := A[i];
   end;
  //вывод значений Amin, Amax
end.
```



Массивы: поиск элементов

Поиск минимального и максимального элементов в одномерном массиве:

```
const n = ...;
var
   A : array[1..n] of integer;
   i, min, max: integer;
begin
   //ввод элементов массива A
   min := 1; max := 1;
   for i:=2 to n do begin
      if A[i]<A[min] then min := i;
      if A[i]>A[max] then max := i;
   end;
  //вывод значений min, max и A[min], A[max]
end.
```



Массивы: поиск элементов

Двоичный поиск в упорядоченном массиве

(искомый элемент – 16)

шаг \ i	1	2	3	4	5	6	7	8	9	a, b, c
1	1	4	9	16	25	36	49	64	81	a=1 b=9 c=5
2	1	4	9	16	25	36	49	64	81	a=1 b=4 c=2
3	1	4	9	16	25	36	49	64	81	a=3 b=4 c=3
4	1	4	9	16	25	36	49	64	81	a=4 b=4 c=4

Двоичный поиск в упорядоченном массиве

```
const  n = ;
var
   A : array[1..n] of integer;
   i, first, last, mid : integer;
begin
   //ввод элементов массива A
   //ввод значения искомого элемента x
   first:=1;  last:=n;
   var found := false;
   while (first<=last) and not found do begin
     mid := first+(last-first) div 2;
     if x<A[mid] then last := mid-1
       else if x>A[mid] then first := mid+1
                        else found := true;
   end;
   if found then writeln('Элемент найден! Его номер ', mid)
            else writeln('Элемент не найден!');
 end.
```

Перестановка элементов

A □□ B ?

Обычно так:

```
t := A;
A := B;
B := t;
```

Но можно и так (только для чисел):

```
A := A-B;
B := A+B;
A := B-A;
```

Массивы: инверсия

Изменение порядка следования элементов на обратный

```
const n = ...;
var
   A : array[1..n] of integer;
   i, t : integer;
begin
   //ввод элементов массива A
   for i:=1 to n div 2 do begin
      t := A[i];
      A[i] := A[n-i+1];
      A[n-i+1] := t;
   end;
   //вывод элементов массива A
end.
```

Массивы: сортировка

Упорядочивание элементов по возрастанию

```
const n = ...;
var
   A : array[1..n] of integer;
   i, j, t : integer;
begin
   //ввод элементов массива A
   for i:=1 to n-1 do
    for j:=n-1 downto i do
       if A[j+1]<A[j] then begin
          t := A[j];
          A[j] := A[j+1];
          A[j+1] := t;
       end;
   //вывод элементов массива A
end.
```

Пузырьковая сортировка (Bubble sort)

Массивы: сортировка

Упорядочивание элементов по возрастанию

```
const n = ...;
var
   A : array[1..n] of integer;
   i, j, imax, t : integer;
begin
   //ввод элементов массива A
   for i:=1 to n-1 do begin
    imax := i;
    for j:=imax+1 to n do if A[j]>A[imax] then imax := j;
    if i<>imax then begin
          t := A[i];
          A[i] := A[imax];
          A[imax] := t;
    end;
   end;
   //вывод элементов массива A
end.
```

Сортировка выбором (Selection sort)

Подпрограммы

Решаемые задачи и принципы организации
Виды подпрограмм и общая структура
Параметры подпрограмм

Подпрограммы: общие сведения

ПОДПРОГРАММА (англ. SUBROUTINE)

Что такое подпрограмма?

Именованная часть программы, оформленная в виде специальной синтаксической конструкции и выполняющая определенные действия.

Зачем она нужна?

- Уменьшает объем исходного кода
- Структурирует исходный код
- Упрощает модификацию программы
- Повышает устойчивость к возникновению ошибок при модификации программы

Пример первый – без подпрограммы*

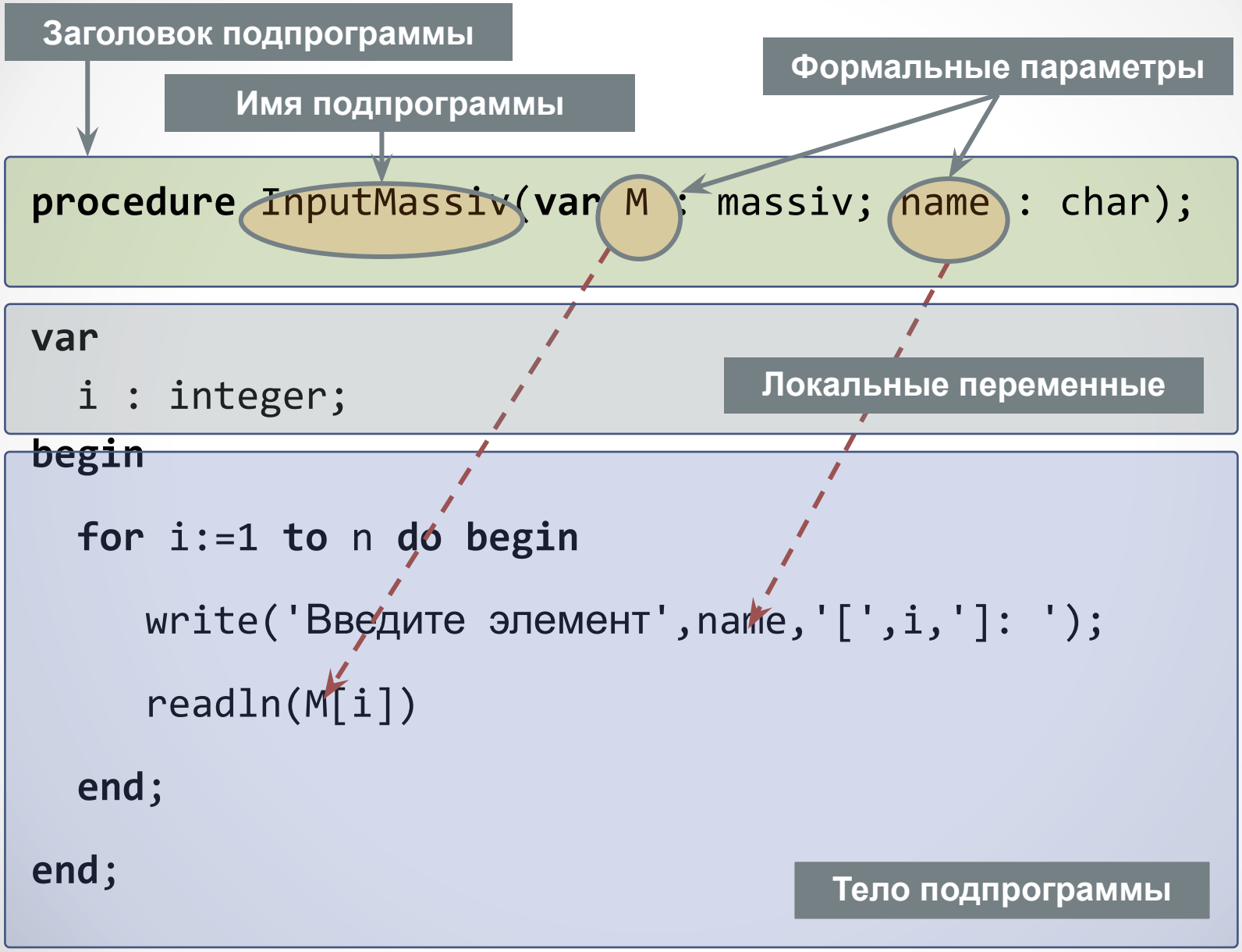
```
const n = ...;
var
   A, B, C : array[1..n] of integer;
   i : integer;
begin
   //ввод элементов массива A
   for i:=1 to n do begin
     write('Введите элемент A[',i, ']: ');
     readln(A[i])
   end;
   //ввод элементов массива B
   for i:=1 to n do begin
     write('Введите элемент B[',i, ']: ');
     readln(B[i])
   end;
   //ввод элементов массива C
   for i:=1 to n do begin
     write('Введите элемент C[',i, ']: ');
     readln(C[i])
   end;
   . . .
end.
```

* не считая стандартных подпрограмм write и readln

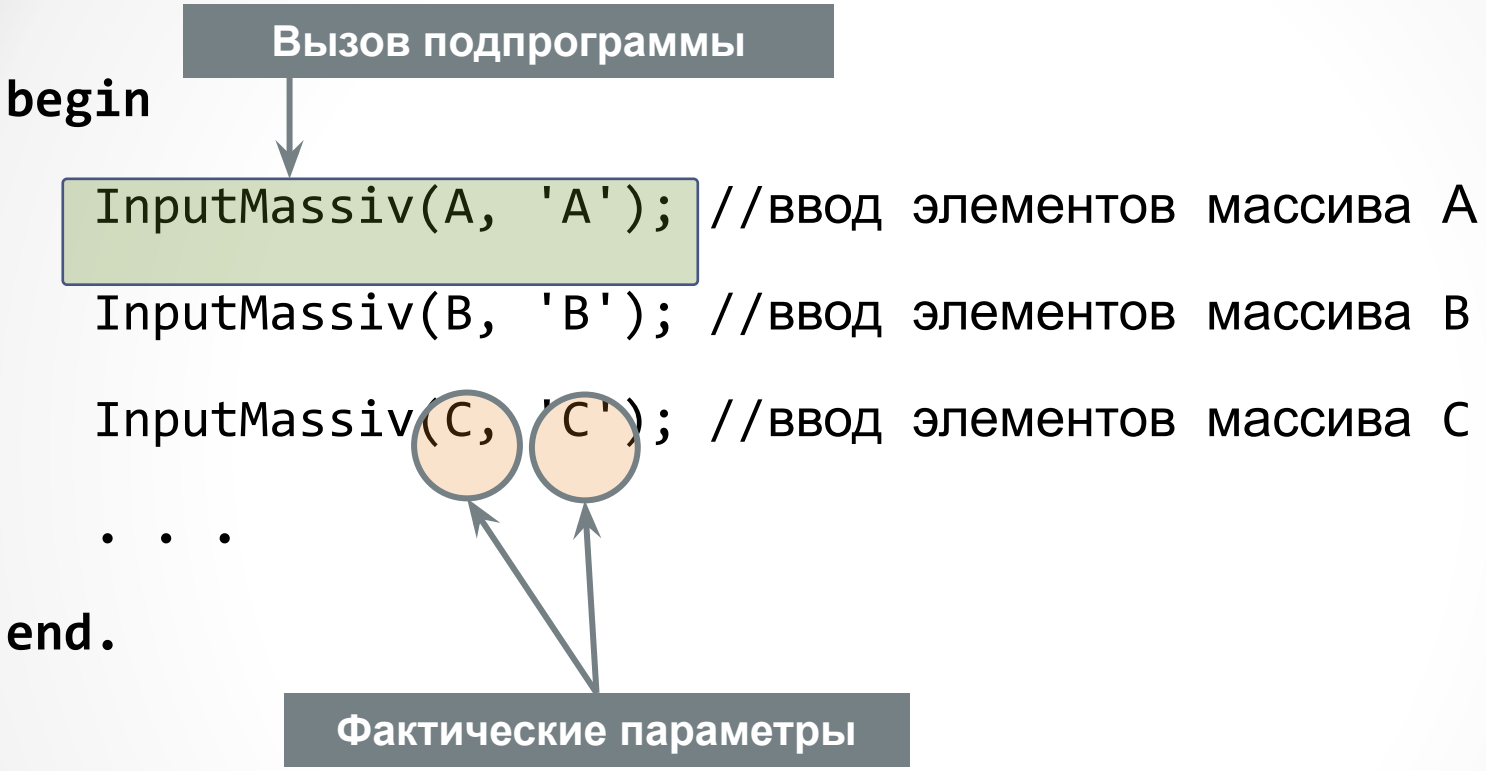
Пример второй – с подпрограммой

```
const n = ...;
type massiv = array[1..n] of integer;
var A, B, C : massiv;

procedure InputMassiv(var M : massiv; name : char);
var i : integer;
begin
  for i:=1 to n do begin
     write('Введите элемент', name, '[', i, ']: ');
     readln(M[i])
  end;
end;

begin
   InputMassiv(A, 'A'); //ввод элементов массива A
   InputMassiv(B, 'B'); //ввод элементов массива B
   InputMassiv(C, 'C'); //ввод элементов массива C
   . . .
end.
```

Подпрограммы: основные понятия



Подпрограммы: основные понятия



Подпрограммы: принципы организации

Что оформлять в виде подпрограммы?

- Повторяемые фрагменты кода
- Логически завершенные фрагменты кода
- То, что может быть полезно в других программах

При решении сложной задачи выделяем подзадачи

- подзадача - подпрограмма

Виды подпрограмм, общая структура

Процедура

```
procedure Имя(список формальных параметров);
var локальные переменные
begin
  тело процедуры
end;
```

Функция

```
function Имя(список формальных параметров) : Тип;
var локальные переменные
begin
  тело функции
  result := Возвращаемое значение
end;
```

Вызов процедуры и функции

Процедура

Вызываем там, где может быть ОПЕРАТОР

```
writeln('Привет мир!');
for i:=1 to n do writeln(i);
```

Функция

Вызываем там, где может быть ЗНАЧЕНИЕ

```
x := sin(0.5);
for i:=1 to n do s:=s + sqr(i);
```

Процедура или функция?

1. Однозначного ответа нет
2. Если что-то вычисляем и результатом является одно значение, то функция
3. Если что-то делаем не обязательно связанное с вычислениями, то процедура
4. Если что-то вычисляем и результатом являются несколько значений (разного типа), то процедура
   (если не вводить новый тип данных)

В некоторых языках вообще нет явного деления подпрограмм на процедуры и функции… (например, в C, C++, C#)

Параметры подпрограмм

- Зачем они нужны?
- Какие они бывают?
- Каковы механизмы передачи параметров?
- Как лучше передавать массивы?
- Как одной подпрограмме передать другую подпрограмму в качестве параметра?

Параметры подпрограмм

Зачем они нужны?

Параметры служат для обмена информацией (например, данными) между основной программой и подпрограммой.

Какие они бывают?

- Параметры, указываемые при описании подпрограммы, называются **формальными**.
- Параметры, указываемые при вызове подпрограммы, называются **фактическими**.
- Если формальный параметр описан с ключевым словом **var** или **const**, то его называют ***параметром-переменной*** и говорят, что он передается ***по ссылке***.
- Если же параметр описан без слов **var** или **const**, то его называют ***параметром-значением*** и говорят, что он передается ***по значению***.

Параметры-значения

- Если параметр передается по значению, то при вызове подпрограммы значения фактических параметров присваиваются соответствующим формальным параметрам.
- Типы фактических параметров-значений должны быть **совместимы по присваиванию** с типами соответствующих формальных параметров.

Например:

```
procedure PrintSquare(i: integer);
begin
  writeln(i*i);
end;
```

При вызове PrintSquare(2*a+b) значение 2*a+b будет вычислено и присвоено переменной i, после чего выполнится тело процедуры

Параметры-переменные

- Если параметр передается **по ссылке**, то при вызове подпрограммы фактический параметр заменяет собой в теле процедуры соответствующий ему формальный параметр.
- Любые изменения формального параметра-переменной внутри процедуры **приводят к соответствующим изменениям фактического параметра**.
- Фактические параметры-переменные должны быть **переменными**, а их типы должны быть **эквивалентны** типам соответствующих формальных параметров..

Например:

```
procedure Mult2(var a: integer);
begin
  a := a*2;
end;
```

После вызова Mult2(x) значение x увеличится в 2 раза.

Параметры подпрограмм

Способ передачи	Фактический параметр
По значению	любое выражение, тип которого совпадает с типом формального параметра или неявно к нему приводится
По ссылке	только переменная, тип которой в точности совпадает с типом формального параметра

Параметры подпрограмм

Механизмы передачи параметров

При передачи параметра **по значению** в подпрограмму передается **значение фактического параметра**.

При передаче параметра **по ссылке** в подпрограмму передается **адрес фактического параметра**.

Если параметр занимает много памяти (массив, запись, строка), то обычно он передается **по ссылке**.

Так как динамические массивы являются ссылками (сами по себе) то их можно передавать **по значению**.

Подпрограмма как параметр

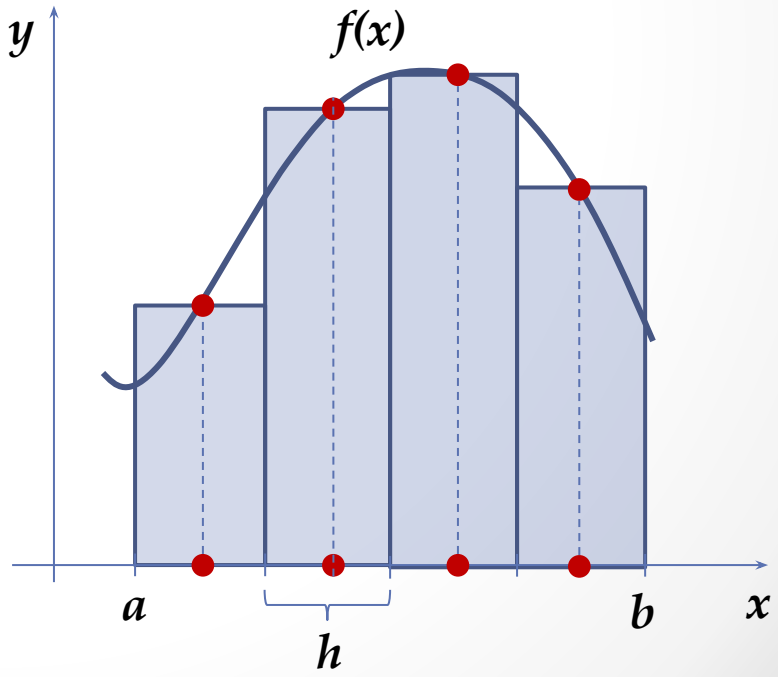
Нужно в тех случаях когда одна подпрограмма выступает параметром другой подпрограммы.

Например, в задаче о приближенном вычислении определенного интеграла:

$$\int_a^b f(x)dx \approx h\sum_{i=1}^{n} f(x_i)$$

$$h = \frac{b-a}{n}$$

$$x_i = a + h(i - 1/2)$$



Решение первое: без использования подпрограмм

```
var
  a, b, S, h, x : real;
  n, i : integer;
begin
  a := 0; b := 1; n := 100;
  h := (b-a)/n;  S := 0;
  for i:=1 to n do begin //вычисление суммы
    x := a + h*(i-0.5);
    S := S + sin(sqr(x))*x;
  end;
  S := S*h;
  write( S );
end.
```

$$\int_0^1 \sin(x^2)\, x\, dx$$

```
function f( x : real ) : real;
begin
  f := sin(sqr(x))*x; //подынтегральная функция
end;
```

```
function Integral(a, b : real; n : integer) : real;
var S, h, x : real;
begin
  h := (b-a)/n;  S := 0;
  for var i:=1 to n do begin //вычисление суммы
    x := a+h*(i-0.5);
    S := S+f(x);
  end;
  Integral := S*h;
end;
```

```
begin
   write( Integral(0,1,100) );
end.
```

Решение второе: с подпрограммами

Функциональный (процедурный) тип

Функциональный тип – тип, задающий интерфейс абстрактной функции.

Синтаксис:

```
ИмяТипа = function( Список параметров ) : Тип;
```

Пример функционального типа:

```
type
  FType = function ( x : real ) : real;
```

```
type
  FType = function( x : real ) : real;

function Integral(a, b : real; n : integer; f : FType) : real;
var S, h, x : real;
begin
  h := (b-a)/n;  S := 0;
  for var i:=1 to n do begin //вычисление суммы
    x := a+h*(i-0.5);
    S := S+f(x);
  end;
  Integral := S*h;
end;

function f1( x : real ) : real;
begin
  f1 := . . .; //подынтегральная функция 1
end;
function f2( x : real ) : real;
begin
  f2 := . . .; //подынтегральная функция 2
end;

begin
  write( Integral(0,10,100,f1) );
  write( Integral(0,10,100,f2) );
end.
```

Решение третье: передаем функцию как параметр

Рекурсивный вызов подпрограмм

Организация повторяющихся действий (вычислений):

- Нерекурсивное (итерационное) – с использованием ***циклов***
- Рекурсивное

Рекурсия в программировании – обращение функции к самой себе.

Рекурсия в математике:

а) один из способов определения функций

$$n! = \begin{cases} n \cdot (n-1)!, & n > 0 \\ 1, & n = 0 \end{cases}$$

Конечная рекурсивная функция

$$e = 2 + \cfrac{2}{2 + \cfrac{3}{3 + \cfrac{4}{4 + \dots}}} = 2 + f(2) \qquad f(n) = \frac{n}{n + f(n+1)}$$

Бесконечная рекурсивная функция

б) рекуррентные соотношения при вычислении рядов

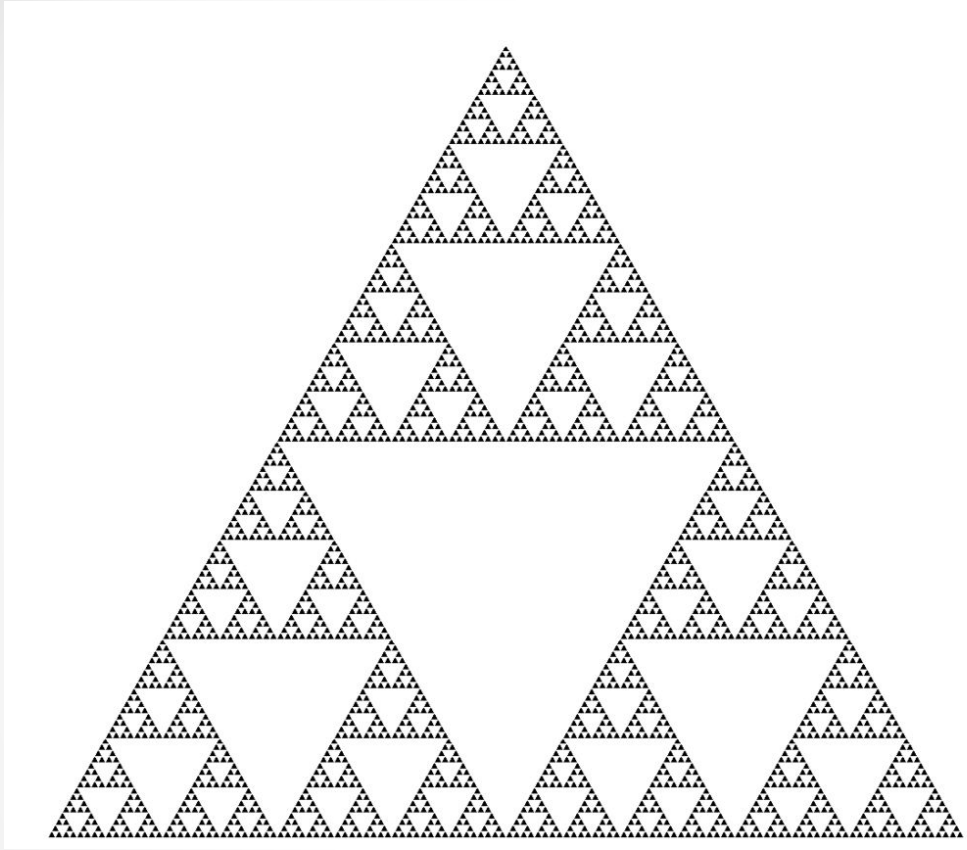
Рекурсивный вызов подпрограмм

Рекурсивное вычисление факториала

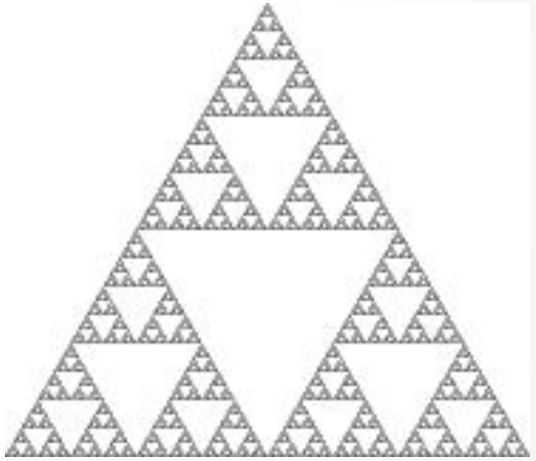
```
function f( n : integer ) : integer;
begin
  if n = 0 then f := 1 else f := n*f(n-1);
end;
begin
   write( f(5) );
end.
```

Нерекурсивное вычисление факториала

```
function f( n : integer ) : integer;
begin
  var p := 1;
  for var i:=1 to n do p := p*i;
  f := p;
end;
begin
   write( f(5) );
end.
```



Построение треугольника Серпинского



Треугольник Серпинского – геометрический фрактал

```
program FractalSierpinky;
uses GraphABC;
const iter = 5;
procedure triangle(x1, y1, x2, y2, x3, y3: real);
begin
  Line(Round(x1), Round(y1), Round(x2), Round(y2));
  Line(Round(x2), Round(y2), Round(x3), Round(y3));
  Line(Round(x3), Round(y3), Round(x1), Round(y1));
end;
procedure draw(x1, y1, x2, y2, x3, y3: real; n: integer);
var x1n, y1n, x2n, y2n, x3n, y3n : real;
begin
 if n>0 then begin
  x1n := (x1 + x2) / 2;  y1n := (y1 + y2) / 2;
  x2n := (x2 + x3) / 2;  y2n := (y2 + y3) / 2;
  x3n := (x3 + x1) / 2;  y3n := (y3 + y1) / 2;
  triangle(x1n, y1n, x2n, y2n, x3n, y3n);
  draw(x1, y1, x1n, y1n, x3n, y3n, n - 1); //рекурсивный вызов процедуры draw
  draw(x2, y2, x1n, y1n, x2n, y2n, n - 1);
  draw(x3, y3, x2n, y2n, x3n, y3n, n - 1);
 end;
end;
begin
  triangle(320,10,600,470,40,470);
  draw(320,10,600,470,40,470,iter);
end.
```