Язык C++

Структура программы

Простейшая программы на языке С состоит всего из 12 символов, но заслуживает внимательного рассмотрения.

```
void main()
{

}
```

```
#include <stdio.h>

int main()
{
   printf("Privet!\n");
   return 0;
}
```

```
include "stdafx.h"


int _tmain(int argc, _TCHAR* argv[])
{   printf("Privet!\n");
    return 0;
}
```

Основные понятия языка

Программы на языке C++ может использовать лишь символами, предусмотренными алфавитом этого языка.

-Буквы латинского алфавита

-Арабские цифры

-Специальные символы

Программа состоит из инструкций включающих в себя слова и выражения

Основные понятия языка

Слова в языке C++ делятся на:

- **Ключевые слова**
- **Стандартные идентификаторы**
- **Идентификаторы пользователя**

Основные понятия языка

Ключевые слова являются частью языка, имеют определенное написание и несут определенный смысл.

Например:

IF FOR WHILE DO

INT FLOAT CHAR

Основные понятия языка

Стандартные идентификаторы предназначены для вызова стандартных функций, хранящихся в библиотеках языка.

Например:

sin(x) exp(x) pow(x)

abs(x) cos(x) tan(x)

sqrt(x) log10(x)

Основные понятия языка

Идентификаторы пользователя служат для обозначения процедур, переменных и констант.

Правила:

1) **не содержит пробелов**
2) **Состоит из букв, цифр и символов подчеркивания, но начинается с буквы**

Переменная – это именованная область памяти, предназначенная для хранения значений, которые могут изменяться в процессе работы программы

```
int f,x=0;
float Y; char z;
```

Константа – это так же именованная область для хранения значений, которые не изменяться в процессе работы программы

```
const  int x=4;
```

Константы и переменные используются в выражениях

ВЫРАЖЕНИЯ

Выражение задает порядок выполнения действий над элементами данных и состоит из операндов (переменных, констант, вызовов функций), круглых скобок и знаков операций.

Операции определяют действия, выполняемые над данными.

Скобки ставятся для управления порядком выполнения действий.

Арифметические операции

1)* /

2)+ -

Операции сравнения

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

Логические операции

! -отрицание, && - И, || - ИЛИ

Стандартные функции

Имя(параметры)

Функция	Вызов
$sinx$	sin(X)
$cosx$	cos(X)
tgx	tan(X)
e^x	exp(X)
$\|x\|$	abs(X)
$\sqrt{x}$	sqrt(X)
lnx	log(x)

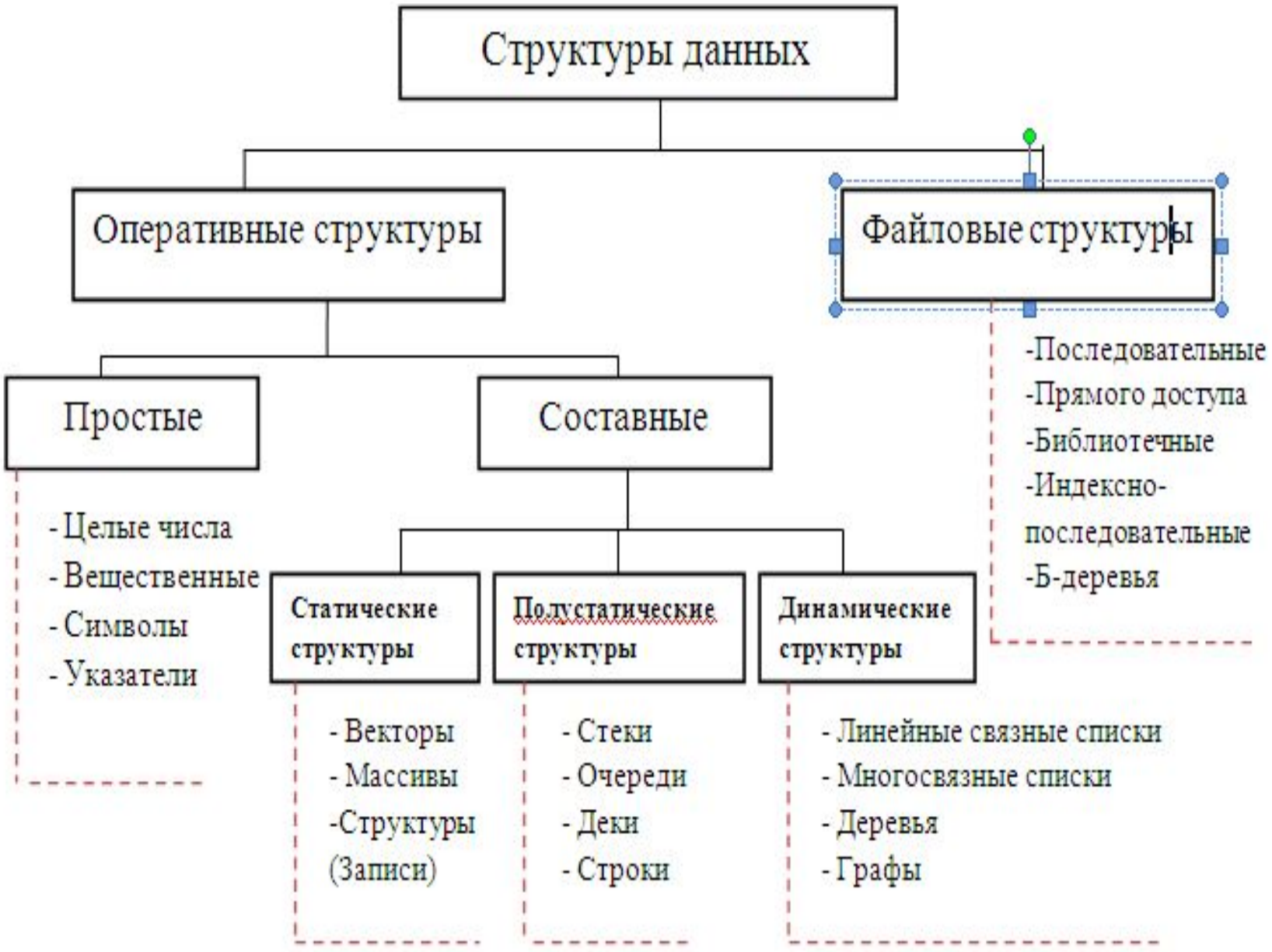
Написать выражение на языке С++

$$\frac{|x-y|}{(1+2x)^2} - e^{\sqrt{1+\mu}}$$

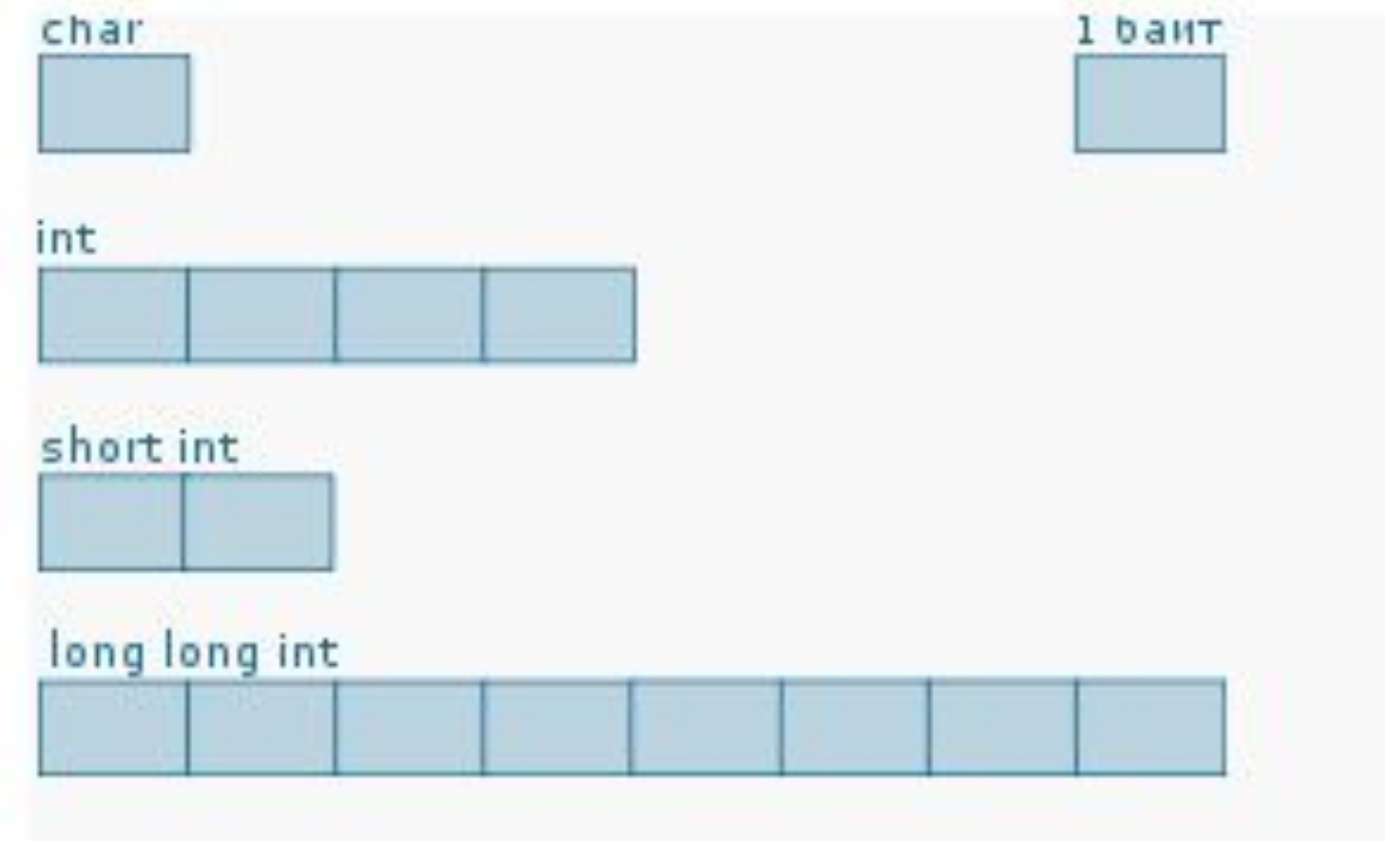
abs(x-y)/pow(1+2*x,2)- exp(sqrt(1+м))

Типа данных

Объявить тип переменной означает установить границы значений этой переменной, которые определяются этим типом и установить допустимые операции для этого типа.


Структуры данных
Оперативные структуры
Файловые структуры
Простые
Составные
- Целые числа
- Вещественные
- Символы
- Указатели
Статические структуры
Полустатические структуры
Динамические структуры
- Векторы
- Массивы
-Структуры (Записи)
- Стеки
- Очереди
- Деки
- Строки
- Линейные связные списки
- Многосвязные списки
- Деревья
- Графы
-Последовательные
-Прямого доступа
-Библиотечные
-Индексно-последовательные
-Б-деревья

Целочисленные типы данных



Целочисленные типы данных

char – целочисленный знаковый тип данных размером в 1 байт

int – целочисленный знаковый тип данных размером в 4 байта.

short int – сокращённый вариант int, имеет размер 2 байта

long long int – самый большой из встроенных типов данных, имеет размер 8 байт и позволяет хранить значения от -2^{63} до $2^{63}-1$.

К каждому из этих типов можно добавить модификатор **unsigned**, что поднимет верхнюю границу диапазона значений в 2 раза, а нижнюю сделает равной 0.

```
int a;
unsigned char b=49;
long long int d, e, f=35416545, g;
```

Вещественные типы данных

В языке Си есть три типа чисел с плавающей запятой:

float – вещественное одинарной точности.

double – вещественное двойной точности.

long double – вещественное расширенной точности.

```
#include "stdafx.h"
#include <iostream>

int _tmain(int argc, _TCHAR* argv[])
{float x; double y=3.141592;
x=1.0*rand()/10;
std::cout << x<<“  “<<y<< std::endl;
    return 0;
}
```

ОПЕРАТОРЫ ЯЗЫКА С++

1) Оператор присваивания

Оператор присваивания служит для вычисления выражения и записи результата в память компьютера.

Общий вид записи оператора

переменная=выражение;

Знак = читается как «присвоить». Конец любого оператора на языке С++ фиксируется точкой с запятой.

Странные операторы присваивания

В программировании часто используются несколько странные операторы присваивания, например:

```
i = i + 1;
```

В языке C++ определены специальные операторы быстрого увеличения (уменьшения) на единицу ***инкремент и декремент***

(***инкремент***)

i++; (постфиксная форма)

++i; (префиксная форма)

что равносильно оператору присваивания i = i + 1;

(***декремент***)

--i; i--; *равносильно оператору* i=i-1;

Сокращенная запись оператора присваивания:

s+=x; (s=s+x;) p*=n;(p=p*n;)

2)Комментарии

Комментарии используются для документирования текста программы и не выполняют никаких действий.

Комментарий может размещаться, как в отдельной строке, так и в конце любой.

Комментарий может начинаться с // или заключаться в /* … */

Пример линейной программы

Составить программу вычисления площади треугольника по формуле:

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

Где $p = \frac{a+b+c}{2}$ - полупериметр; a, b, c - стороны треугольника.

Исходные данные: $a = 1$; $b = 2$; $c = 0{,}5$.

```
#include "stdafx.h"
#include<math.h>
int  main()
{
float a, b, c, p, S;// описание переменных
printf("Введите исходные данные  ");
scanf("%f %f %f",&a, &b,&c); /* ввод с
клавиатуры значений для a, b и c */
p = (a + b + c) / 2; // вычисление полупериметра
S=sqrt(p*(p - a)*(p -b)*(p - c)); // выч-е  площади
printf("Площадь треугольника S =%5.2f", S);
return 0;
}
```

3) Составной оператор

Если возникла необходимость объединить несколько операторов в одно целое, используется составной оператор:

```
{
  оператор 1;
    . . .
  оператор N;
}
```

4) Операторы цикла

1) оператор For

Формат:

```
for( i = ml;    i<= m2;  i=i+шаг)
{     <тело цикла>;     }
```

i - параметр, управляющий работой цикла;

ml, m2 - выражения, определяющие соответственно начальное и конечное значения параметра цикла.

Правила использования оператора for

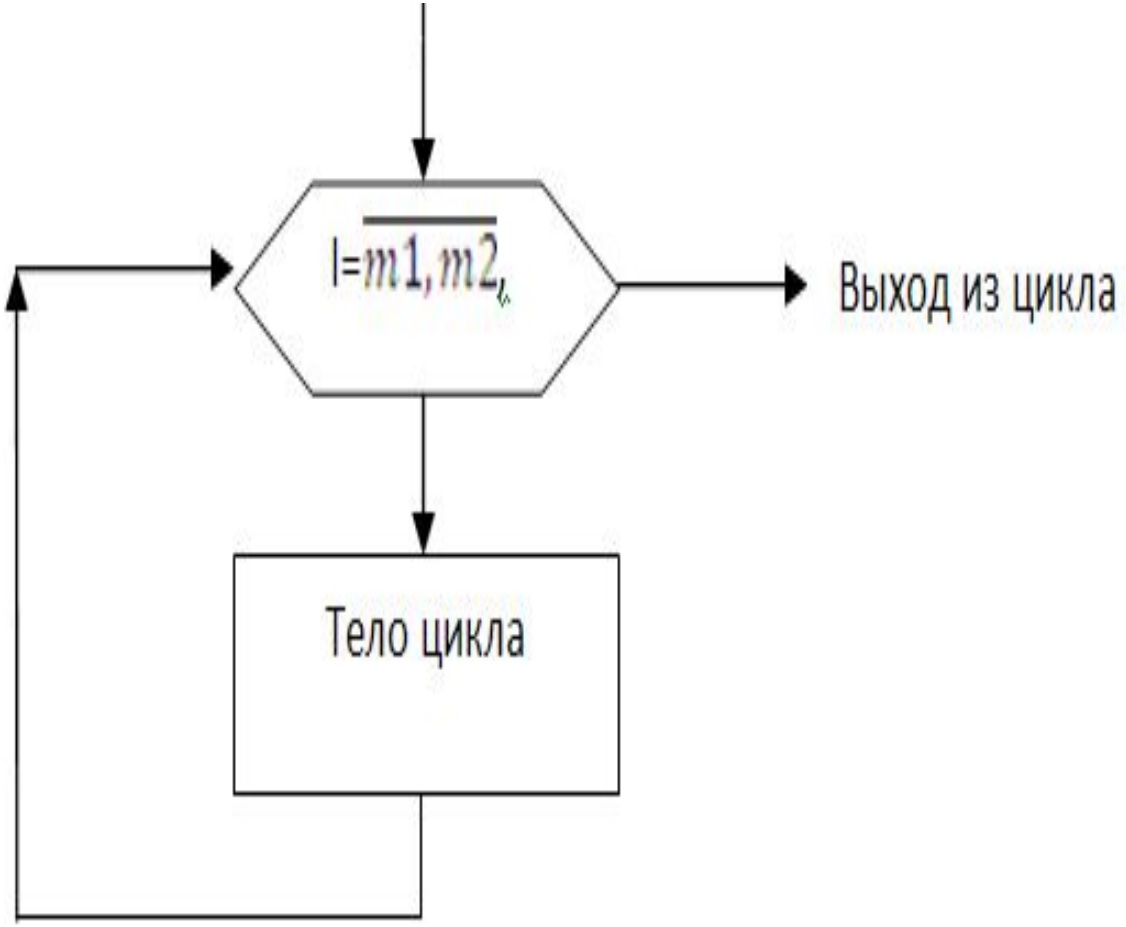
1)Параметр цикла i, а также его значения ml и m2 могут быть любого типа.

2)Параметр i, а также значения ml и m2 не должны переопределяться (менять значения) в теле цикла.

3)При завершении работы оператора for параметр i становится неопределенным и переменную i можно использовать в других целях.

4)Тело цикла может не выполниться ни разу, если m1>m2 для цикла for... с положительным шагом, или m1<m2 для отрицательного шага.

Графическая интерпретация оператора цикла for...



Пример 1. Алгоритм расчета значений функции с одной переменной.

Вычислить таблицу значений функции:

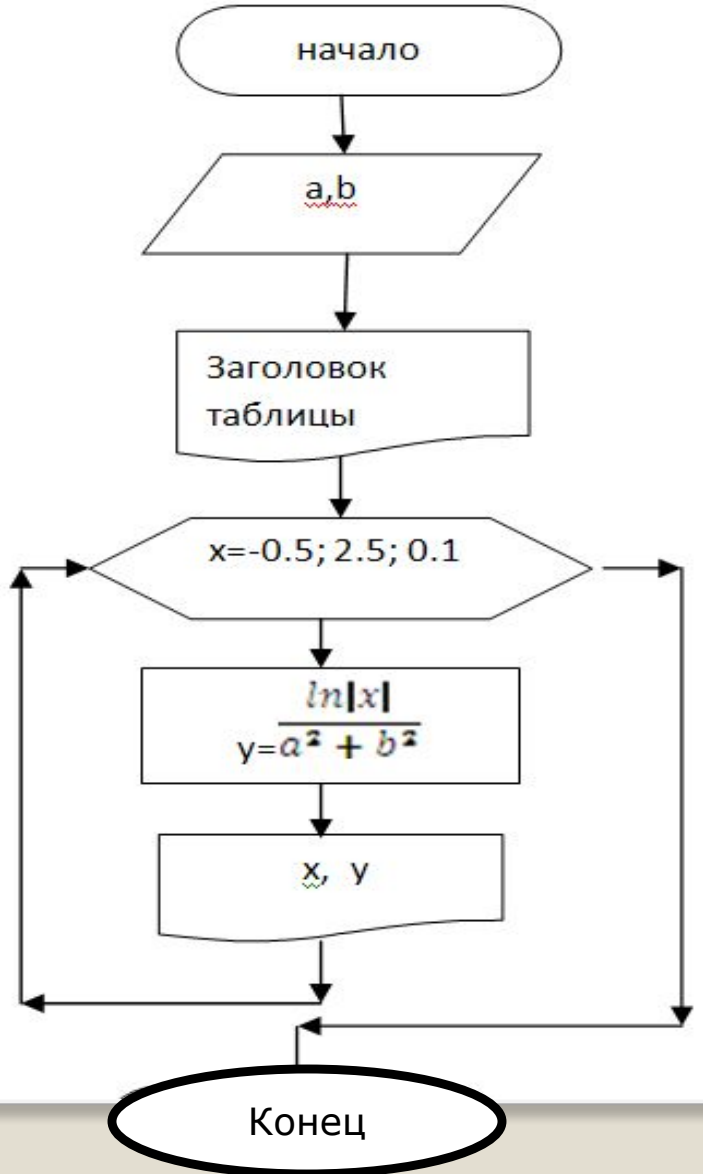
$$y(x) = \frac{\ln|x|}{a^2 + b^2}$$

для всех x, изменяющихся в интервале $[-0.5\ ,\ 2.5]$ с шагом $\Delta x = 0.1$,
a, b - заданные вещественные числа.

В данной задаче переменная x является управляющей переменной цикла.

***Пример* 3.** Решить предыдущую задачу табулирования функции с использованием оператора цикла *for*.

Схема алгоритма



Программа

```
#include "stdafx.h"
#include<math.h>
int  main()
{
float a, b, x, y;
printf("введите a и b  ");
scanf("%f%f",&a,&b);
printf(" x        y(x)\n");
for( x=-0.5; x<=2.5; x=x+0.1)
{
  y=log(fabs(x))/(a*a+b*b);
  printf("%8.1f %8.1f\n",x,y);
}
return 0;
}
```

2)Оператор While

Оператор цикла с предусловием

Формат:

While (условие)

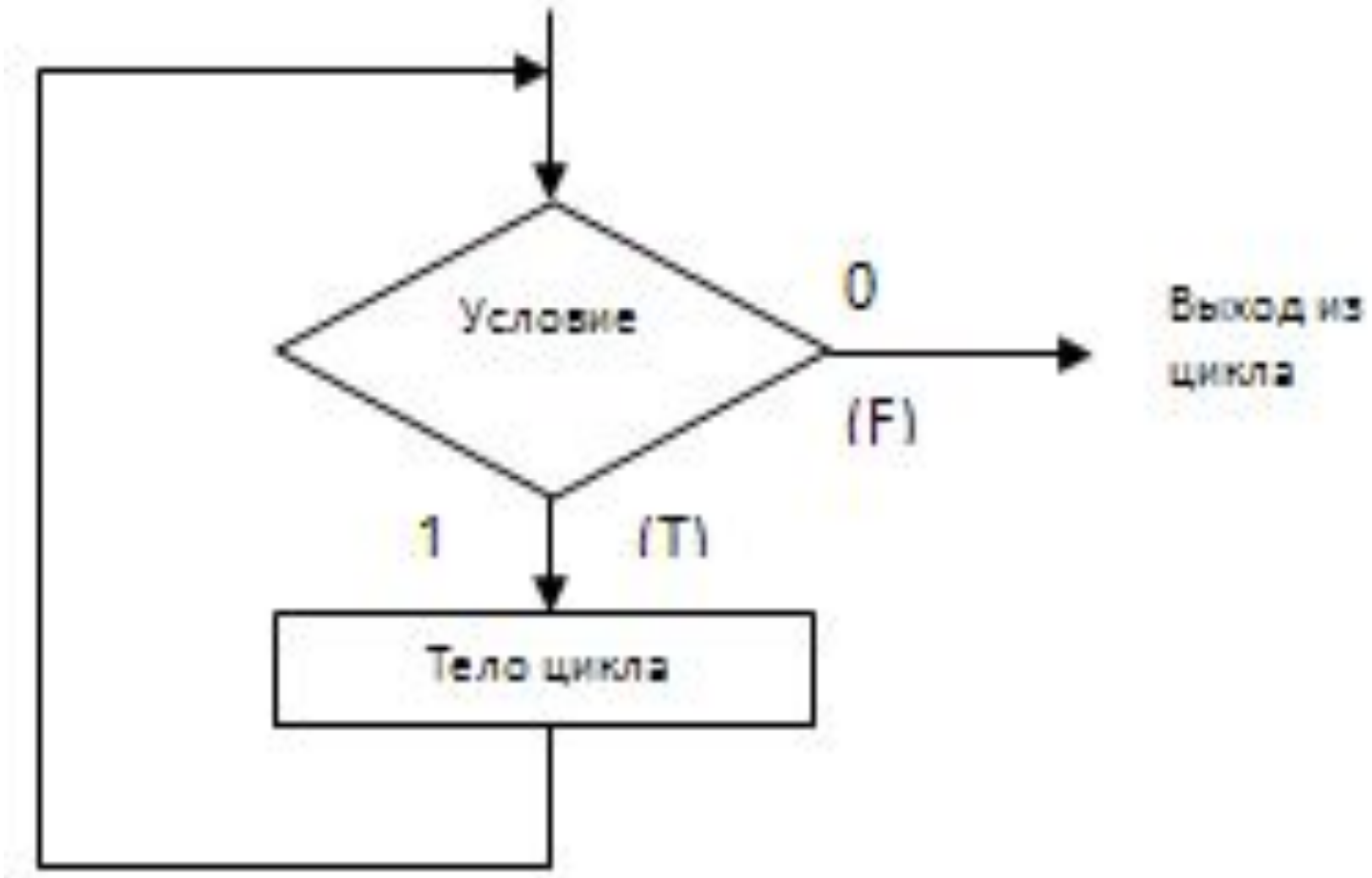
{

Операторы тела цикла

}

Замечание. Если тело цикла состоит из нескольких операторов, то их обязательно заключают в операторные скобки {...}.

Графическая интерпретация оператора



Пример 1. Алгоритм расчета значений функции с одной переменной.

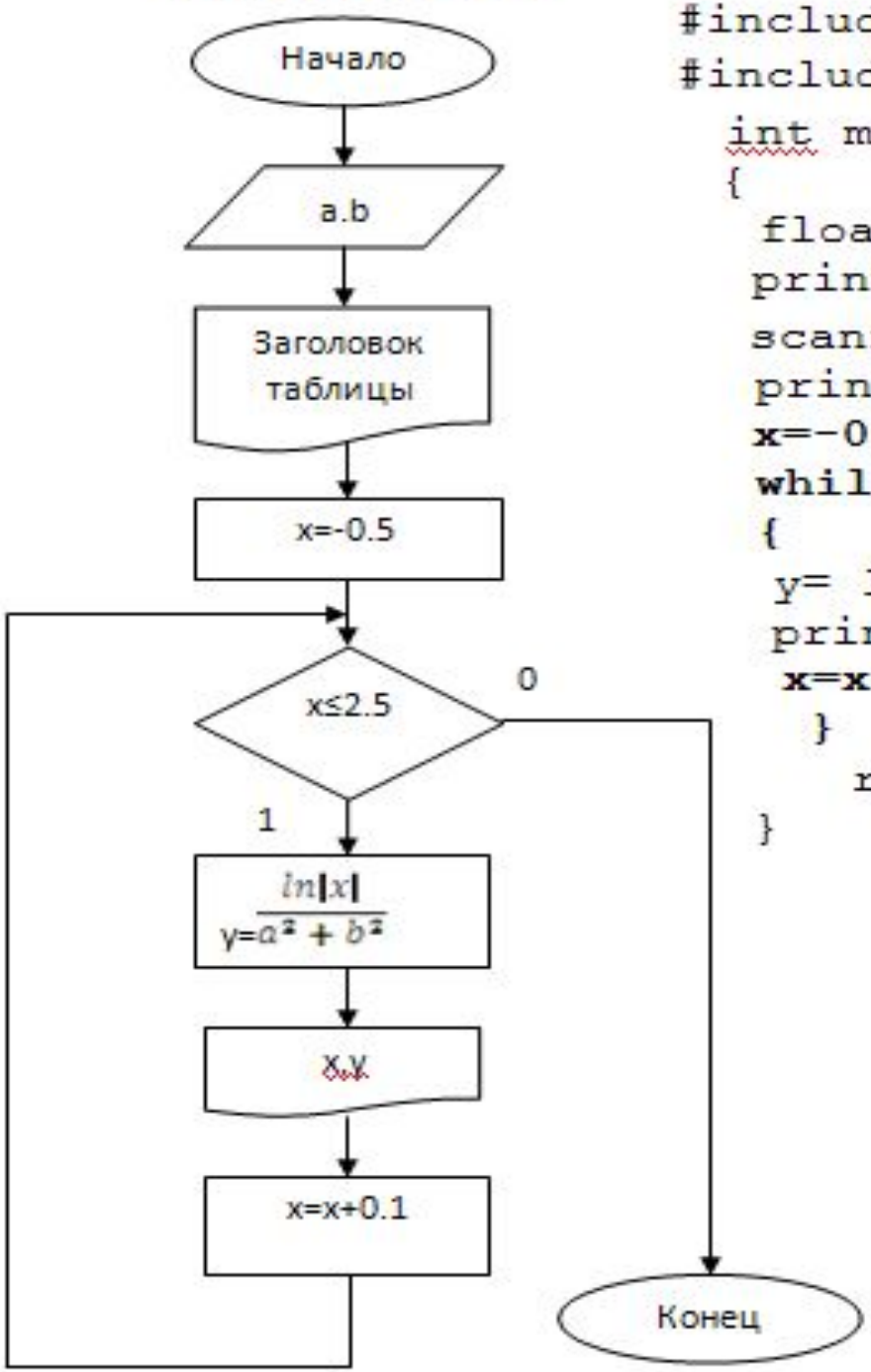
Вычислить таблицу значений функции:

$$y(x) = \frac{\ln|x|}{a^2 + b^2}$$

для всех x, изменяющихся в интервале $[-0.5\ ,\ 2.5]$ с шагом $\Delta x = 0.1$,
a, b - заданные вещественные числа.

В данной задаче переменная x является управляющей переменной цикла.

Схема алгоритма



Программа

```
#include "stdafx.h"
#include<math.h>
  int main()
  {
   float  a, b, x, y;
   printf("Введите a и b\n");
   scanf("%f%f",&a,&b);
   printf(" x      y(x)\n");
   x=-0.5; //нач. установка
   while(x <= 2.5)
   {
    y= log(fabs(x))/(a*a +b*b);
    printf("%8.1f   %8.1f",x,y);
    x=x + 0.1;
     }
       return 0;
  }
```

3)Оператор цикла do...while

Оператор цикла с постусловием

Формат

```
do
 {
<тело цикла>
   }
        while (логическое выражение);
```

Графическая интерпретация оператора

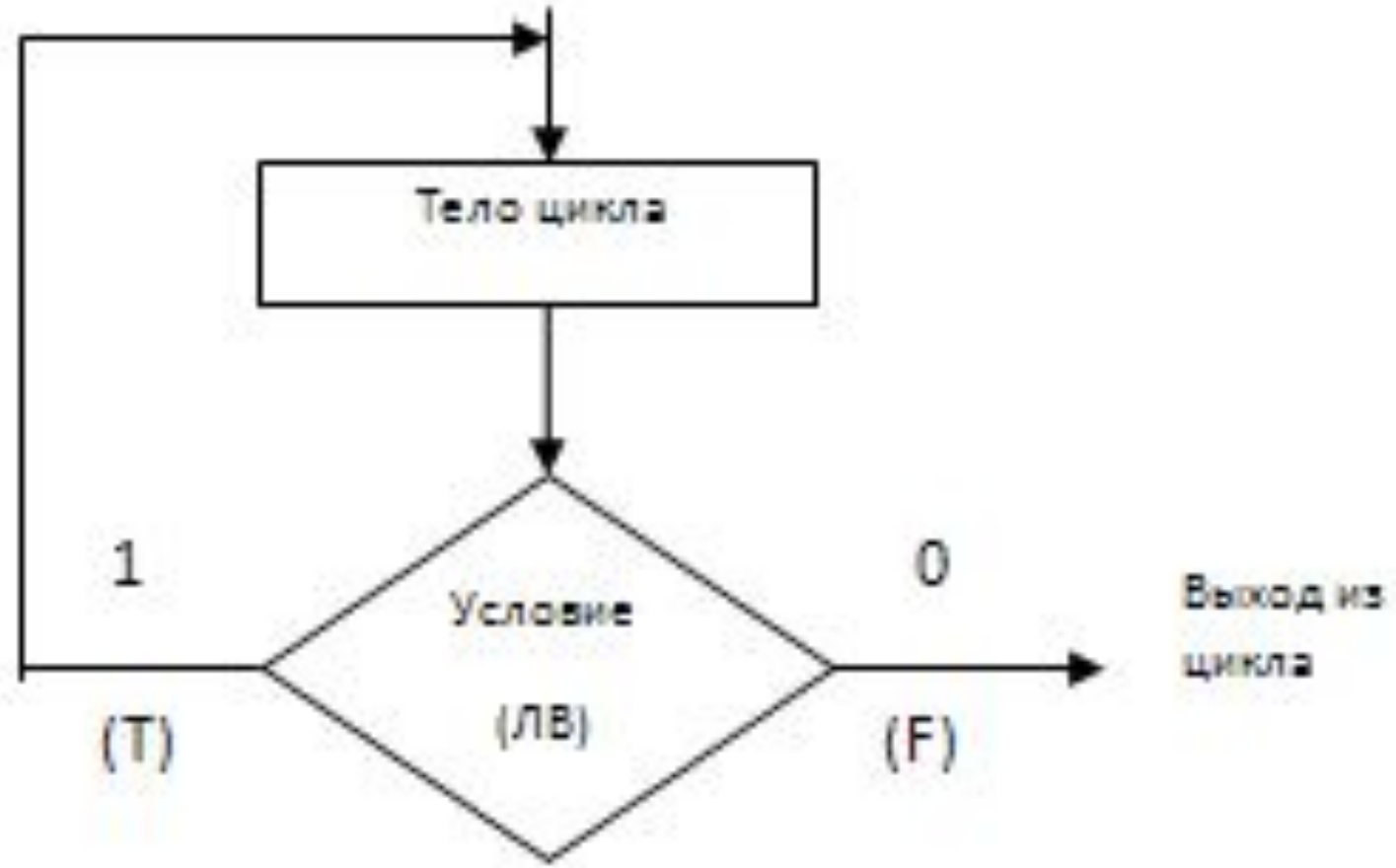
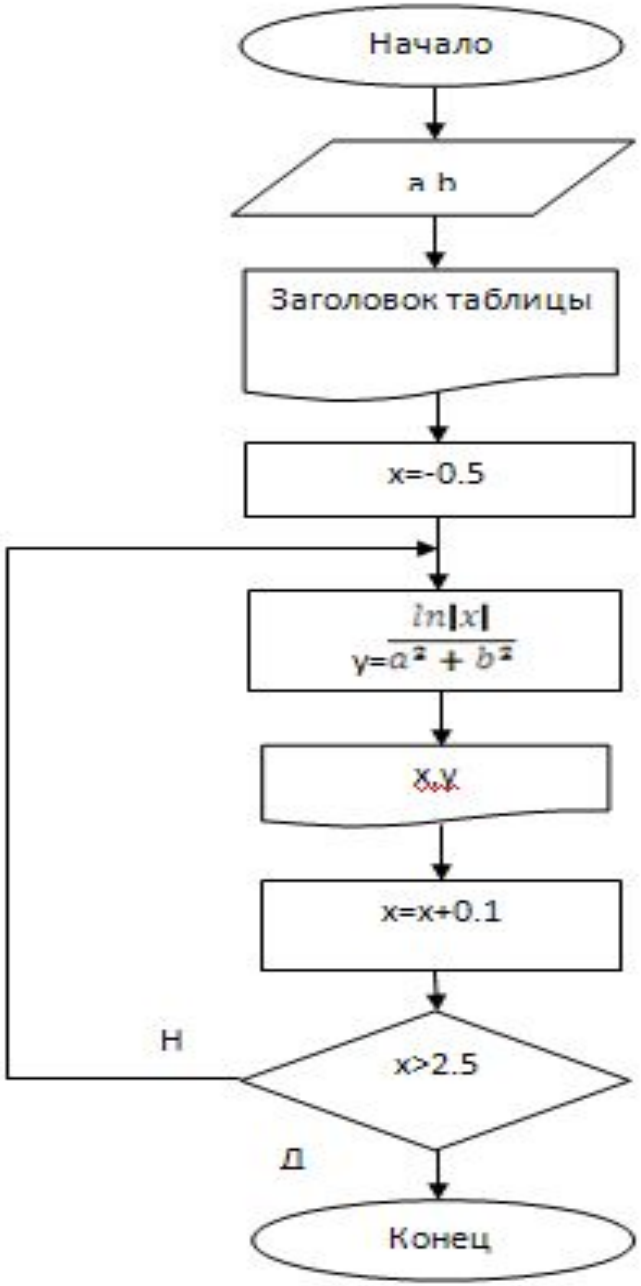
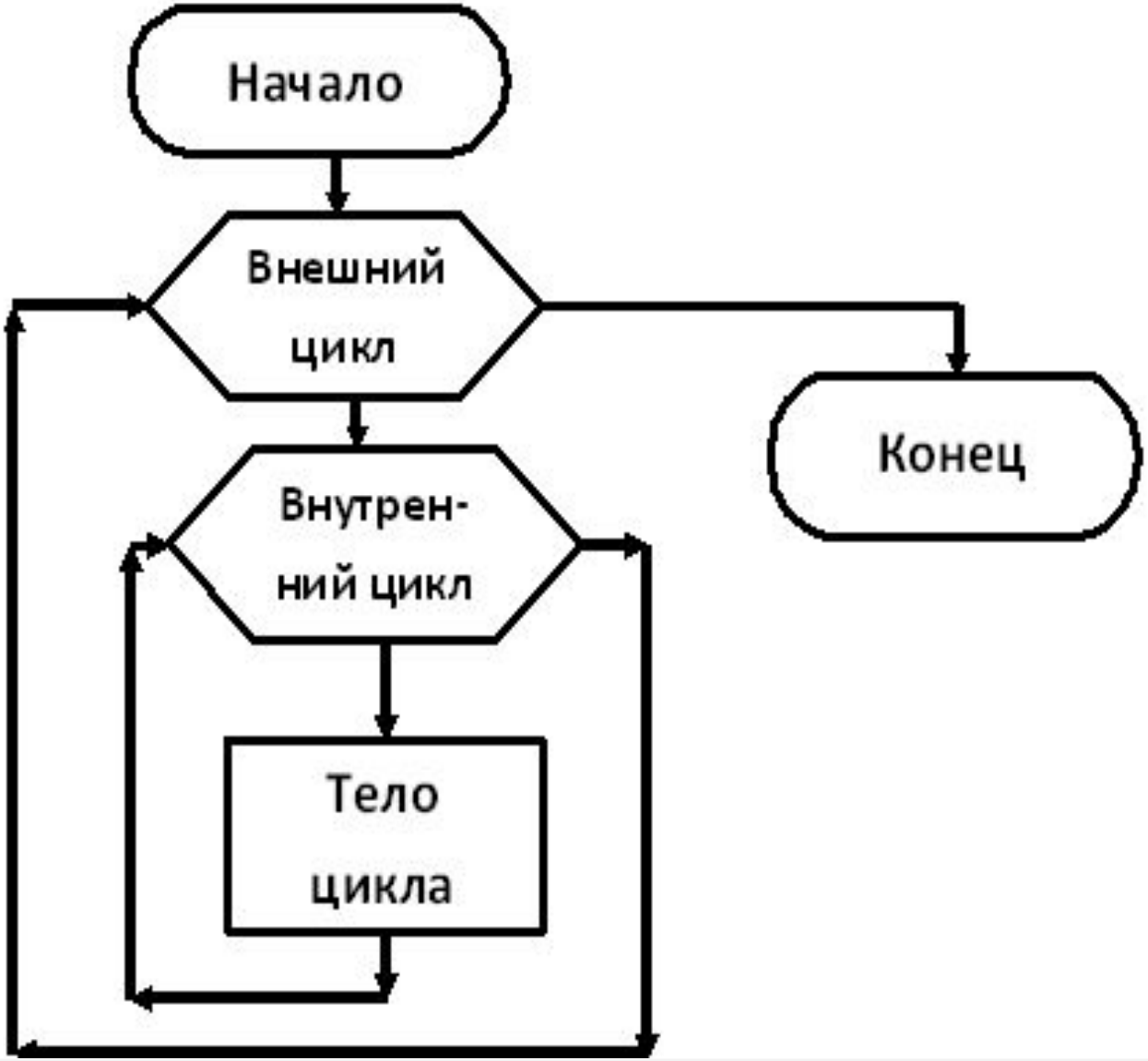


Схема алгоритма **Программа**




```
#include "stdafx.h"
#include<math.h>
 int  main()
 {
 float   a, b, x, y;
 printf("Введите a и b ");
 scanf("%f%f",&a,&b);
 printf(" x        y(x)\n");
 x=-0.5; //нач. установка
 do
 {
  y=log(fabs(x))/(a*a+b*b);
  printf("%8.1f    %8.1f\n",x,y);
  x= x+ 0.1;
 } while( x<= 2.5);
 return 0;
 }
```

Вложенные циклы



```
int _tmain(int argc, _TCHAR* argv[])
{float x, y, z;            // описание переменных
printf("Расчет функции двух переменных\n");
x= -l; //x -  параметра  внешнего цикла
 while (x<=1)   // запуск внешнего цикла
{
   printf("x=%6.1f   \n", x); //вывод значения x
  for( y=0; y<=1; y=y+0.1)
  {   z=sin(x) + cos(y);    // вычисление функции
printf("y=  %6.1f     z=%6.1f\n", y, z);
   }    x=x + 0.5;
 }return 0;}
```

5) Условные операторы

На языке C++ различают два вида условных операторов: короткий и полный.

Короткий условный оператор

Общий вид записи

if (логическое выражение) P;

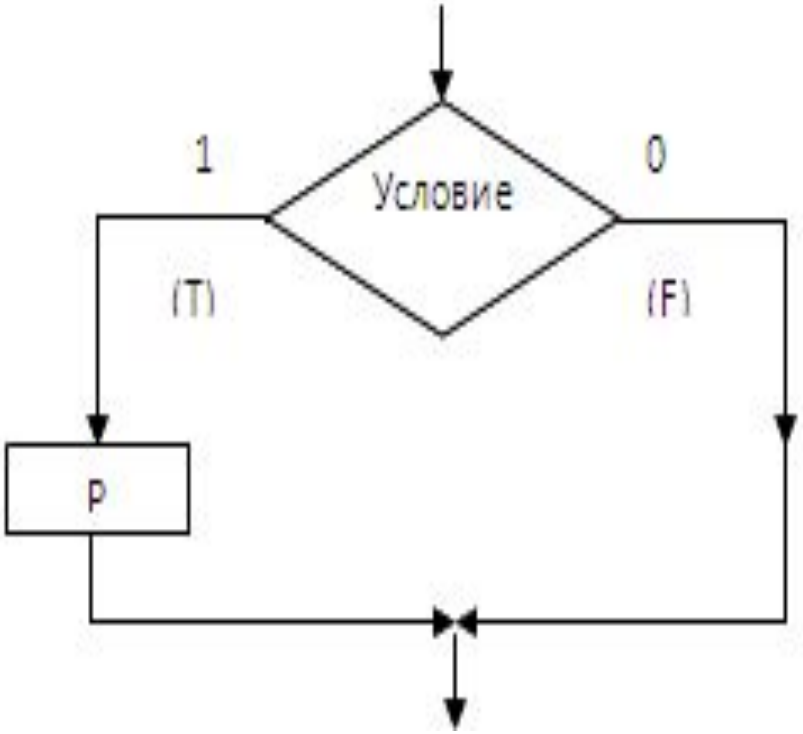
где P - любой оператор языка.

Работа оператора

Сначала вычисляется логическое выражение (ЛВ), и если оно имеет значение TRUE, то выполняется оператор Р, стоящий за логическим выражением. В противном случае оператор Р игнорируется.

Графическая интерпретация оператора

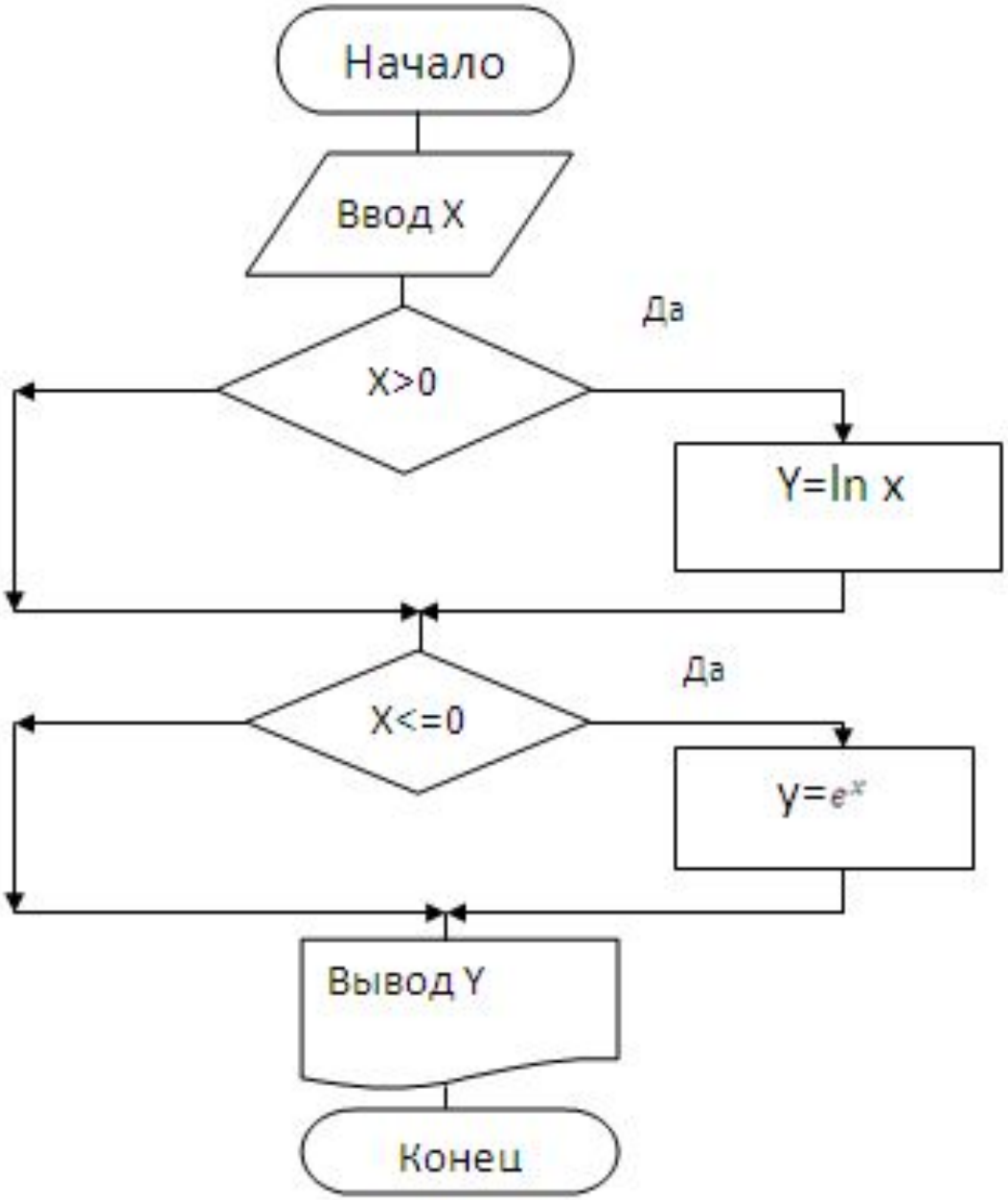
В схемах алгоритма короткому условному оператору соответствует структура **ЕСЛИ—ТО.**



Пример 1.

Вычислить значение переменной *y* заданной двумя интервальными выражениями:

$$y = \begin{cases} \ln x, если \quad x > 0, \\ e^x, если \quad x \le 0. \end{cases}$$


Начало
Ввод X
Да
X>0
Y=ln x
Да
X<=0
y=e^x
Вывод Y
Конец

```
int main()
{   setlocale (0,"Rus");
    float x,y;
    cout <<"Введите X:";
    cin>>x;
    if(x>0) y=log(x);
    if (x<=0) y=exp(x);
    printf("Y=%6.2f\n",y);
return 0;
}
```

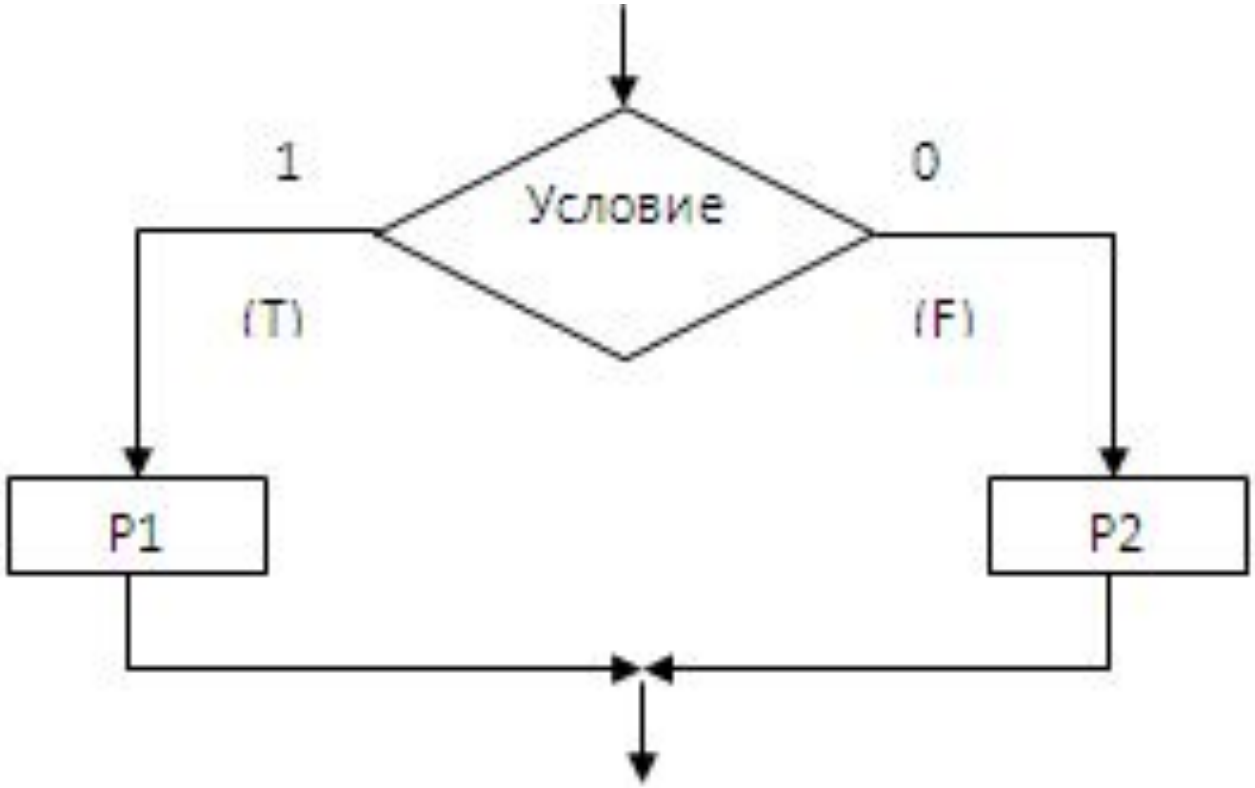
Полный условный оператор

Общий вид записи

```
if (логическое выражение)
    P1;
else
    P2;
```

где P1, P2 - любые операторы или даже группы операторов.

Графическая интерпретация оператора



Работа оператора

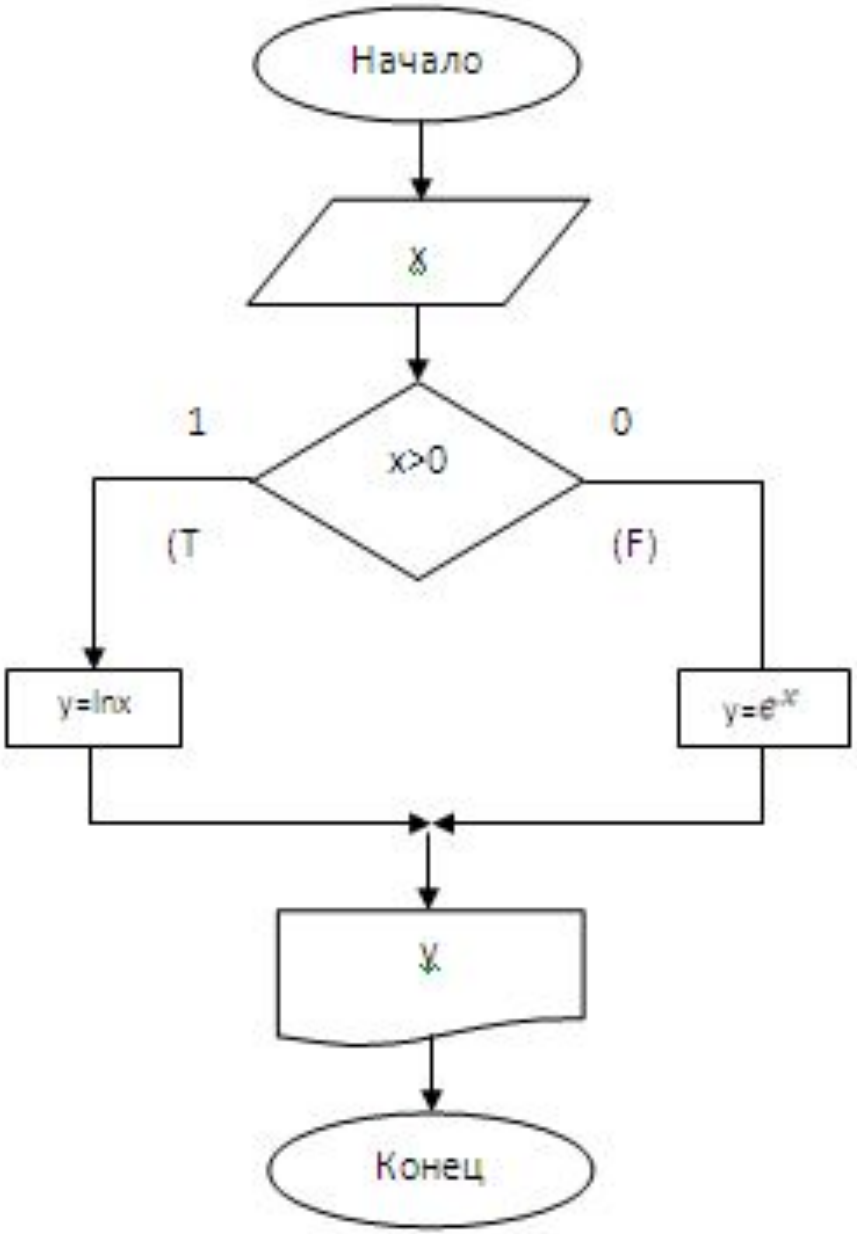
Вычисляется логическое выражение, и если оно имеет значение ИСТИНА(не ноль), то выполняется оператор P1, стоящий после логического выражения. В противном случае (ЛОЖЬ(ноль)) оператор P1 пропускается, а выполняется оператор P2, стоящий после служебного слова else.

```
if (логическое выражение)
  {
    оператор 1;
    ……….
   оператор n;
 }
else
 {
   оператор 1;
   ……….
   оператор m;
}
```

Пример 1.

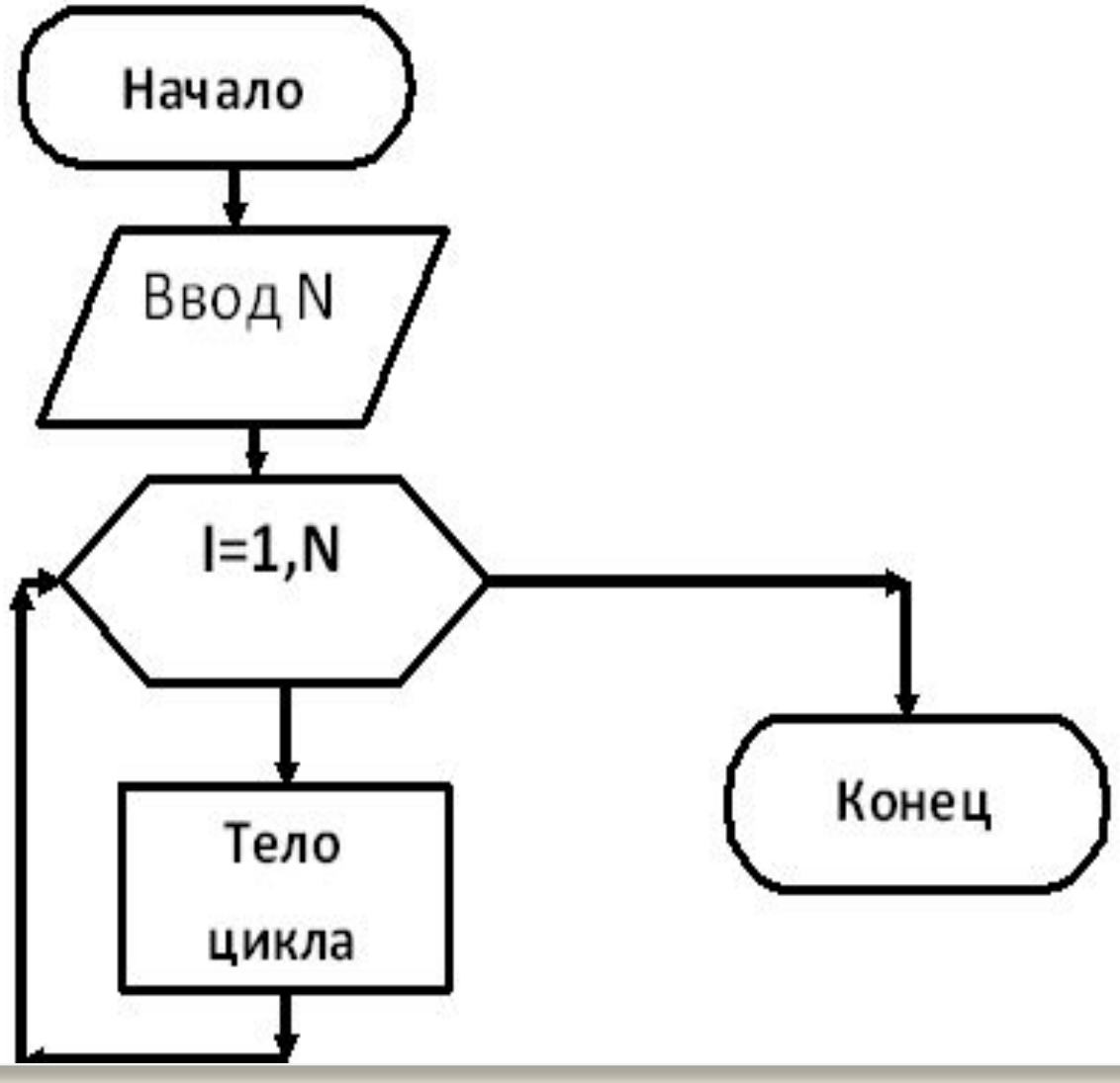
Вычислить значение переменной *y* заданной двумя интервальными выражениями:

$$y = \begin{cases} \ln x, если \quad x > 0, \\ e^x, если \quad x \le 0. \end{cases}$$


Начало
X
1
0
x>0
(T
(F)
y=lnx
y=e^-x
Y
Конец

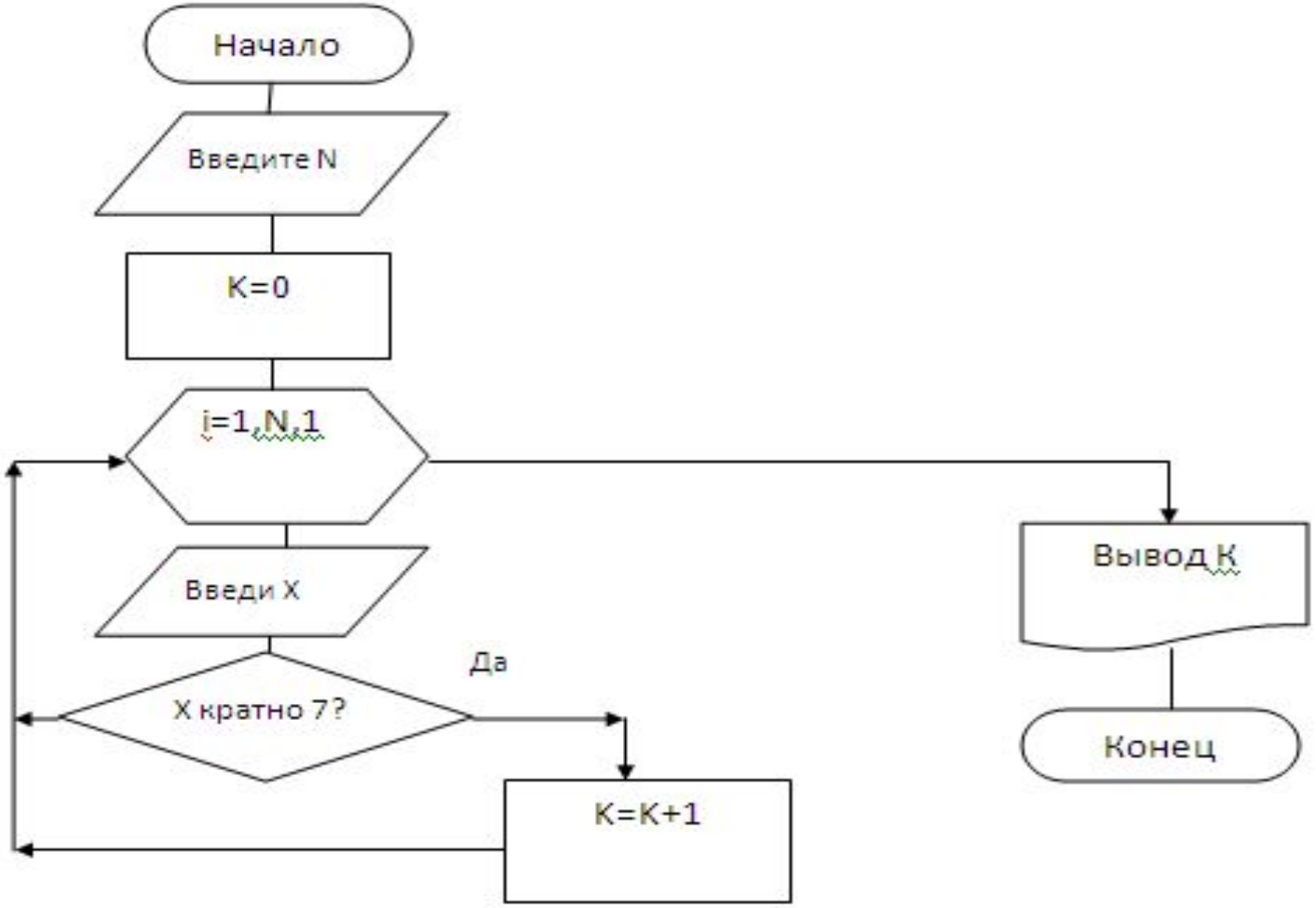
```
int main()
{   setlocale (0,"Rus");
    float x,y;
    cout <<"Введите X:";
    cin>>x;
    if(x>0)
        y=log(x);
    else
        y=exp(x);
    printf("Y=%6.2f\n",y);
    return 0;
}
```

Обработка последовательностей



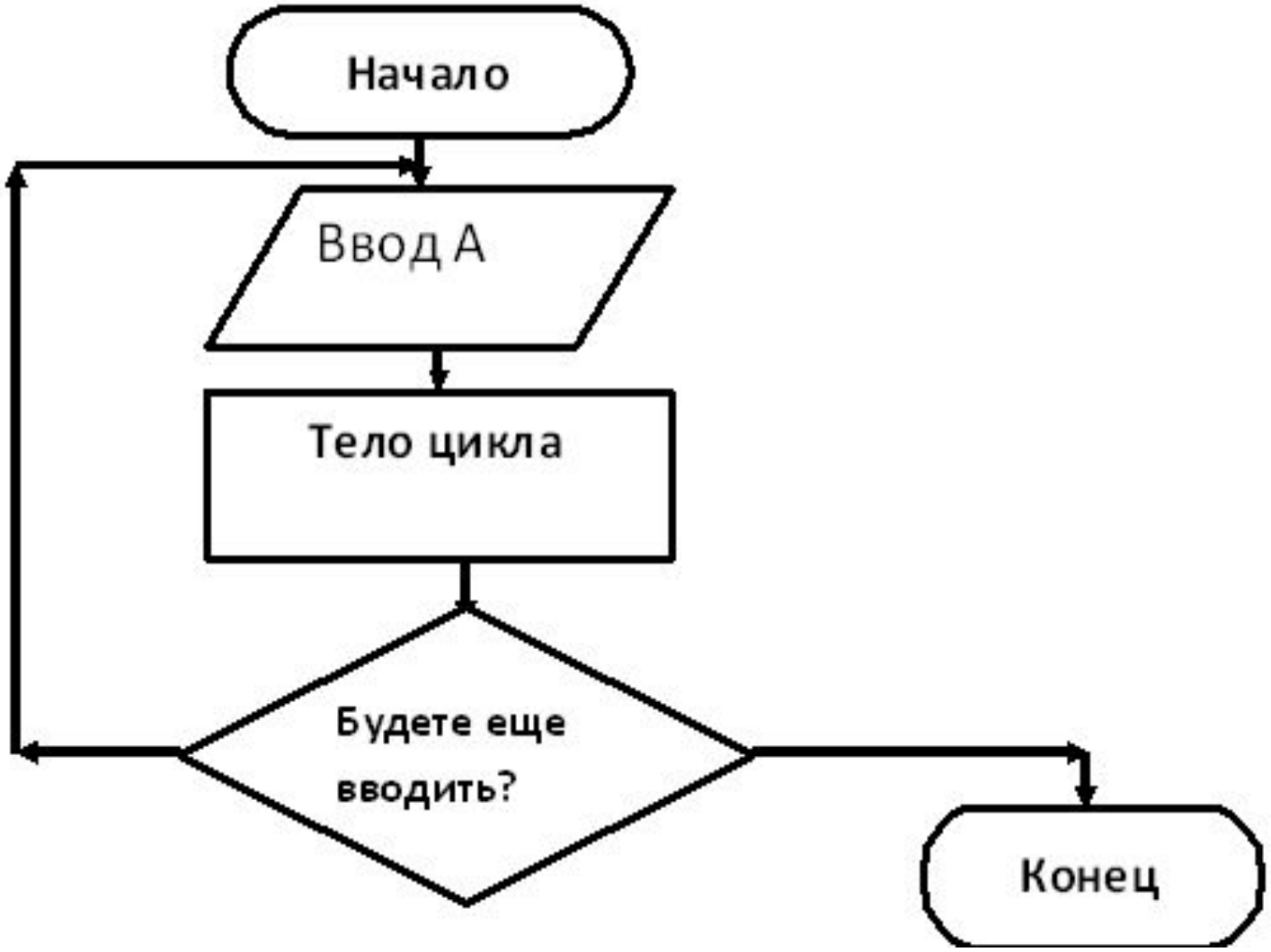
Пример 2.

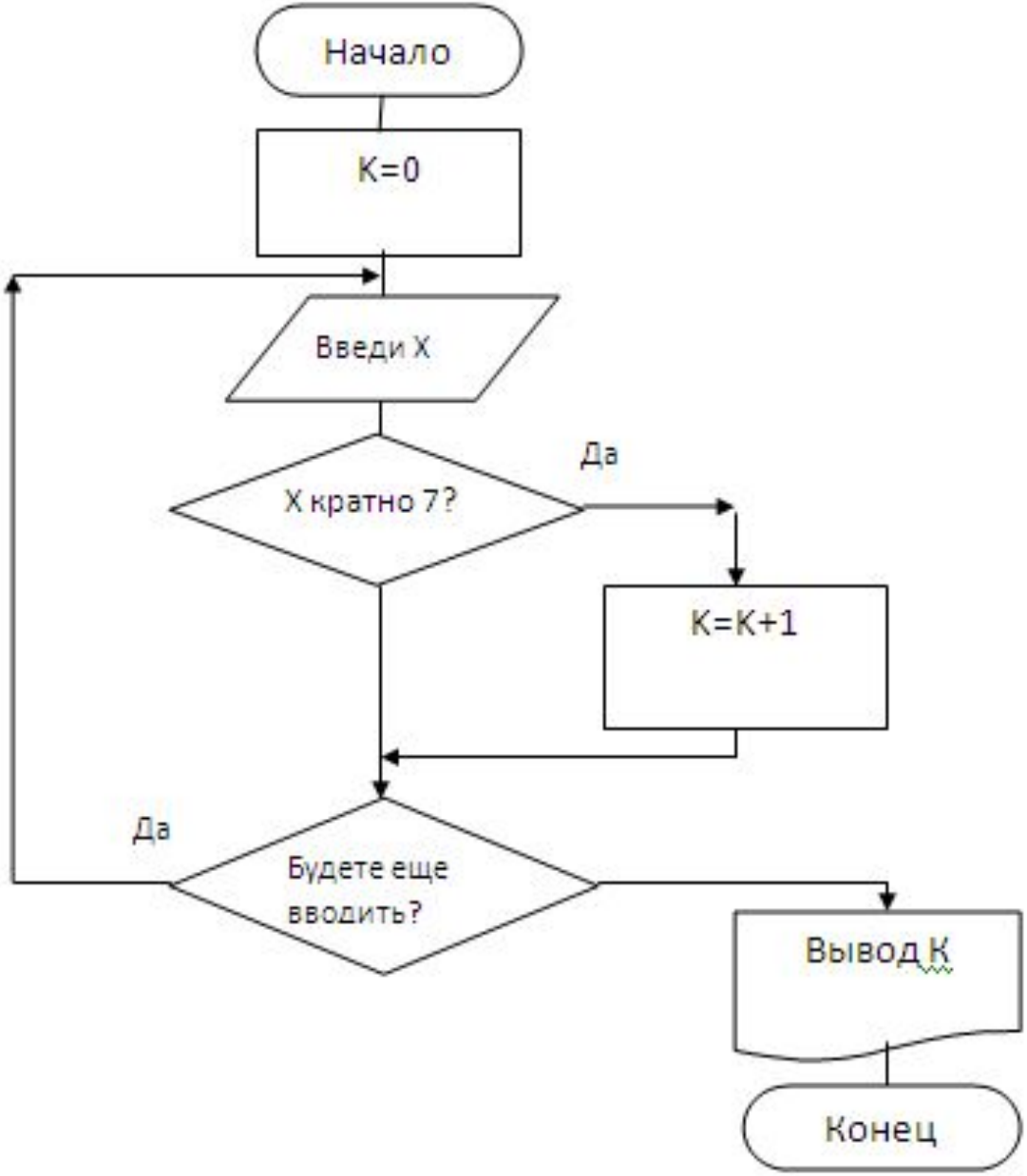
В последовательности чисел, вводимой с клавиатуры вычислить количество чисел кратных 7.



```
int main()
{   setlocale (0,"Rus");
    int N,x,k,i;
    cout <<"Сколько чисел будете вводить:";
    cin>>N;
    k=0;
    cout<<"Введите "<<N<<“элементов последовательности:";
    for(i=1;i<=N;i++)
     {
        cin>>x;
        if(x%7==0)  k++;
     }
   printf("K=%i\n",k);
   return 0;
}
```

Обработка последовательностей




Начало
K=0
Введи X
X кратно 7?
Да
K=K+1
Будете еще вводить?
Да
Вывод K
Конец

```
int main()
{   setlocale (0,"Rus");
    char N; int x,k,i;
    k=0;
    do
     {   cout<<"Введите элемент последовательности: ";
     cin>>x;
      if(x%7==0)  k++;
      cout<<"Будете еще вводить?(Y/N):";
      cin>>N;
      }while(N=='Y');
     printf("K=%i\n",k);
     return 0;
}
```

Оператор выбора

При большем числе вложений условий рекомендуется использовать оператор выбора switch-case.

switch <выражение>

{

case константа выбора 1: оператор 1; break;

……

case константа выбора n: оператор n; break;

default: оператор n+1;

}

Константы выбора - возможные значения селектора.

default – осуществляет обработку непредусмотренного значения выражения. Наличие этой метки в операторе switch необязательно.

Работа оператора

По вычисленному значению выражения выбирается для исполнения case-оператор, содержащий константу выбора, равную значению селектора. После выполнения выбранного case-оператора управление передается на конец оператора case. Следующим в программе выполняется оператор, стоящий за оператором выбора switch.

МАССИВЫ

Массив представляет собой упорядоченное множество однотипных элементов.

A[0] A[1] A[2] A[3] A[4]

При обработке массива необходимо:

1) Описать массив
2) Инициализировать массив исходными данными
3) Обработать в соответствии с алгоритмом
4) Вывести результаты

1) Описание массива

При описании массива необходимо указать:

- способ объединения элементов в структуру (одномерный, двухмерный и т. д.);
- число элементов;
- тип элементов.

<тип элементов> имя [число элементов];

Доступ к каждому элементу массива осуществляется с помощью индексов. Индексы задают порядковый номер элемента, к которому осуществляется доступ. В языке C++ первый элемент массива имеет индекс ноль.

Число индексов определяет структуру массива: если используется один индекс, то такой массив называется одномерным, если два индекса - двухмерным, и т.д.

Одномерные массивы

float A[5];

Компилятор отводит под массив память размером (sizeof(тип)*размер) байтов.

При описании массива можно задать начальные значения его элементов:

int dat[4]={5,8,-2,11};

float kom[]={3.5,6,-1.1};

2) Инициализация массива

-это задание начальных значений.

Инициализировать массив можно:

- При описании:int dat[4]={5,8,-2,11};

-вводом: (для этого организуется цикл ввода по индексу)

```
float A[5];
 printf("Введите массив A\n");
 for( i = 0;  i<=4; i++)
    scanf("%f",&A[i]);
```

Обработка одномерных массивов

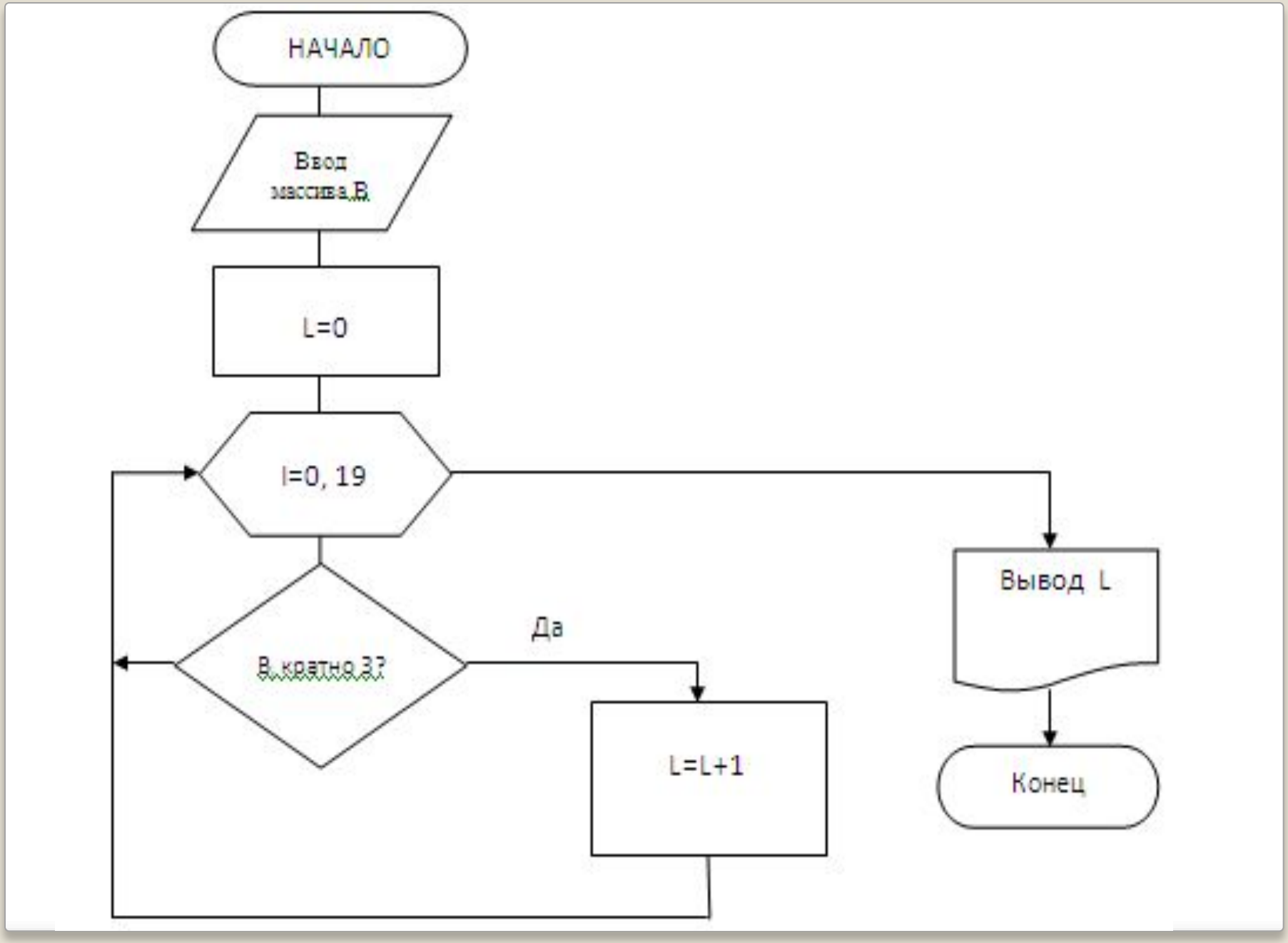
При решении задач обработки массивов используют типовые алгоритмы.

Просмотр массива осуществляется в цикле.

Задача 1.

Дан целочисленный массив: $B = \{b_i\}; i =$.

Определить количество элементов массива, которые делятся на 3 без остатка.


НАЧАЛО
Ввод массива B
L=0
I=0, 19
B кратно 3?
Да
L=L+1
Вывод L
Конец

```
int main()
{ int  B[20] ;              /* описание массива B*/
   int  i, L;
   printf("Введите массив B\n");
    for( i=0;  i<20;  i++)
     scanf("%d", &B[i]);    // ввод данных
     L=0;
     for( i =0; i<20;  i++)
          if (B[i] % 3== 0) //проверка на кратность 3
           L++;
   printf("Кол-во=%d\n", L);
  return 0; }
```

Дано целое число n и массив вещественных чисел:

$$X = \{x_i\}, \quad i = \overline{1, n}.$$

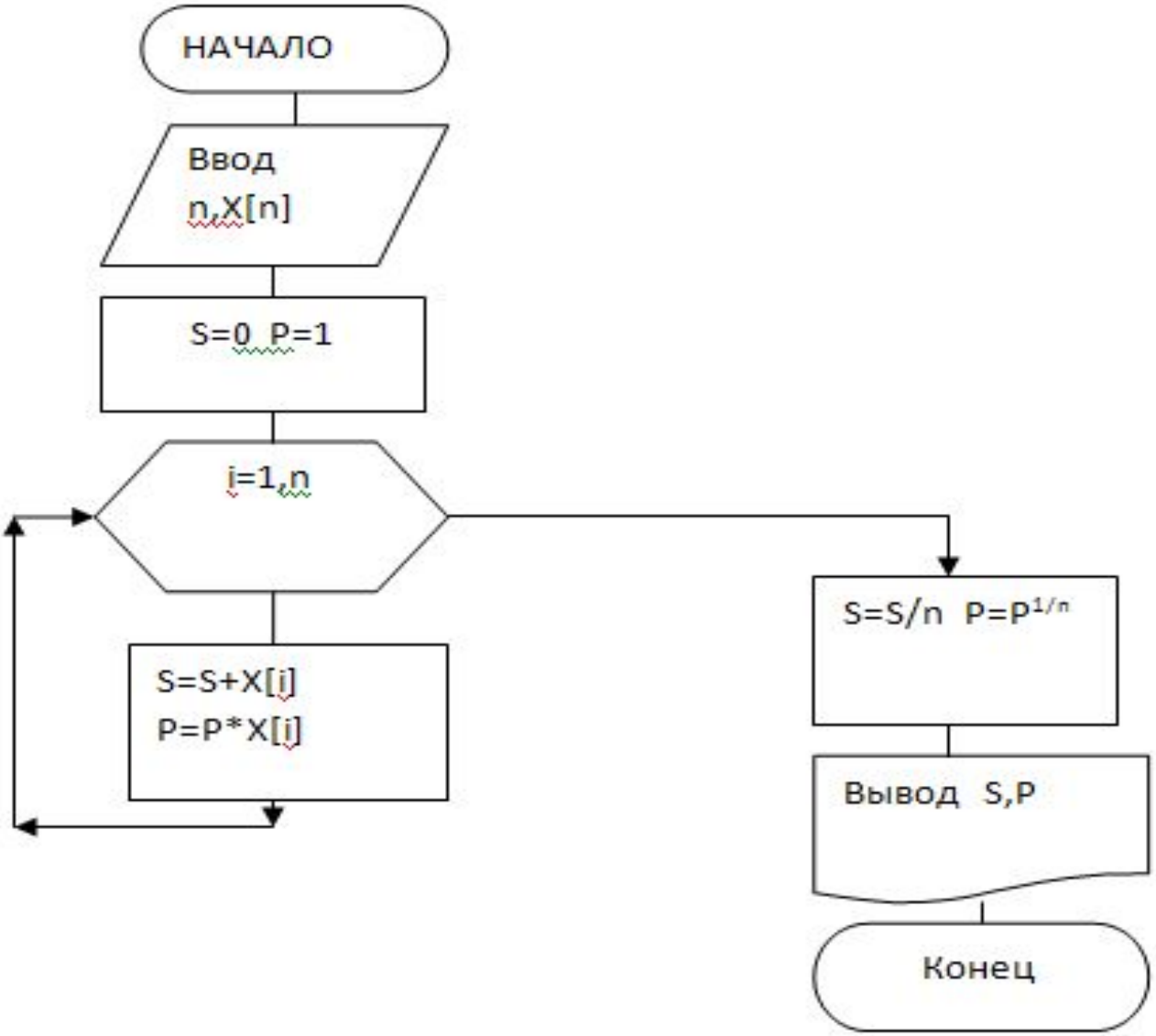
.

Вычислить среднее арифметическое и среднее геометрическое чисел массива, используя формулы:

; .

$$S = \frac{1}{n}\sum_{i=1}^{n} x_i \; ; \qquad P = \sqrt[n]{\prod_{i=1}^{n} x_i} \; .$$

Схема алгоритма:



```
#include "stdafx.h"
#include <math.h>
int main()
{
  float   X [100];          //описание массива X
  float z;
  int   n;
  int i;
  float   S=0, P=1;          //начальные значения S и P
  printf("Введите размер массива n= ");
  scanf("%d",&n);
  printf("Введите массив X\n");
  for( i = 0; i<n; i++)
  scanf("%f",&X[i]);
```

```
for( i=0;  i<n; i++)
  {
   S = S + X[i];   // вычисление суммы элементов
   P =P*X[i];      // вычисление произведения
  }
 S = S/n;          /* вычисление среднего значения X */
 z=1.0/n;
 P=pow(P,z);       /*  вычисление среднего
                     геометрического X */
 printf("S=%6.2f\n", S);
 printf("P=%10.6f\n",P);
return 0;
}
```

Страницы пособия 81-88

Указатели и массивы

Указатель - это переменная, содержащая адрес области памяти.

Присваивая указателю то или иное допустимое значение, можно обеспечить доступ к данным через этот указатель.

Формат описания:

Тип *имя;

Например:

int *x;

char *y;

x – это указатель на ячейку, в которой хранится целое значение, а y – указатель на однобайтовую ячейку, предназначенную для хранения символа.

Двумя наиболее важными операциями, связанными с указателями, являются операция обращения по адресу * и определение адреса &.

Операция обращения по адресу предназначена для записи или считывания значения, размещенного по адресу, содержащемуся в переменной-указателе.

Например:

```
int *x;
. . .
*x=5;
```

Операция определения адреса & возвращает адрес памяти своего операнда. Операндом должна быть переменная.

Например:

```
int *x;
int a=5;
x=&a;
```

Над указателями можно выполнять арифметические операции сложения и вычитания.

Если x – указатель на целое, то операция x++; увеличивает значение адреса, хранящегося в переменной-указателе на число равное размеру ячейки целого типа, т.е. на 4 байта; теперь оно является адресом следующей ячейки целого типа.

Инициализация указателей

Указатели обычно используют при работе с динамической памятью.

В C++ используется 2 способа работы с динамической памятью:

1) Использование функция malloc() и free();
2) Использование операций new и delete.

При определении указателя надо выполнить его инициализацию.

Существует 4 способа правильного задания начального значения для указателя:

1) Присваивание указателю адреса существующего объекта:

```
int a=5;
int *p=&a;
```

2) Присваивание указателю адреса области в явном виде:

```
char *p=(char*)0xB8000000;
```

3) Присваивание указателю безопасного нулевого адреса.

```
int *x=0;
int *y=NULL;
```

4) Выделение участка динамической памяти и присваивание ее адреса указателю:

- с помощью операции new:

  int *x=new int;

- с помощью функция malloc() :

  int *x=(int*)malloc(sizeof(int));

```
#include <malloc.h>
  int main()
 {   int *x, *w;
     int  y;
     x=(int*)malloc(sizeof(int));
     *x=16;
      y=-15;
     w=&y; …
```

Указатели могут обеспечить простой способ ссылок на массив.

Имя массива является указателем, ссылающимся на адрес первого элемента массива.

Например:

```
int  Ar[6];
printf ("адрес Ar=%x\n",Ar);
printf ("адрес Ar=%x\n",&Ar[0]);
```

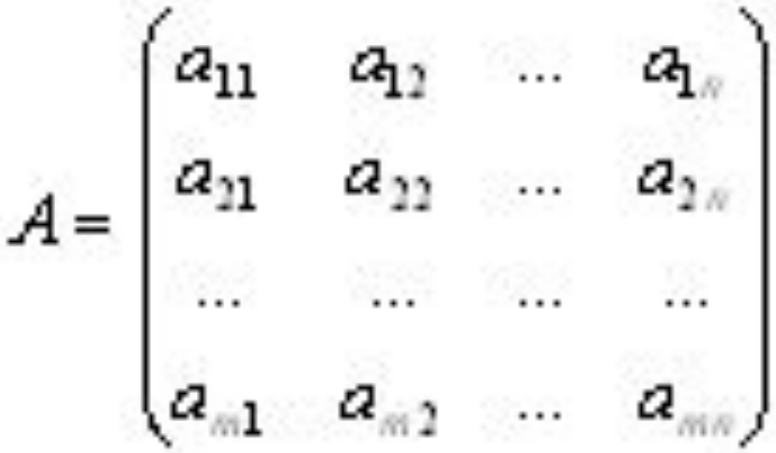
В приведенном фрагменте обе функции printf выводят адрес массива Ar, т.к. выражения Ar и &Ar[0] эквивалентны.

Используя указатели, можно организовать работу с динамическими массивами.

```
int  main( )
{ int n;
   cout << "vvedi razmer massiva:";
   cin >> n;
   int *M=new int[n];
   for(int i=0;i<n;i++)
   {       M[i]=rand()/1000;
           cout<< *(M+i)<<" ";
   }
    . . .
}
```

Двухмерные массивы

Двухмерные массивы в математике представляются матрицей:



или сокращенно можно записать: $A =$, где m – число строк; n - число столбцов; i,j - индексы (номера) строки и столбца, на пересечении которых находится элемент a_{ij}.

Описание двухмерного массива

Описание матрицы задается структурным типом вида:

<тип элементов><имя> [m][n] ;

где m – количество строк;

n –количество столбцов матрицы.

Например:

float A [3][5];

Обращение к отдельным элементам матрицы осуществляется по имени переменной с двумя индексами, причем индексы, как и для одномерного массива начинаются с нуля.

Например:

A[i][j] A[2][3] A[2*n][k+1]

При инициализации многомерного массива он представляется как массив из массивов.

При этом левую размерность можно не указывать:

```
int matr2[][2]={{1,1},{0,2},{1,0}};
```

А можно значения задавать общим списком

```
int matr1[3][2]={ 1, 1, 0, 2, 1, 0};
```

Ввод-вывод двухмерного массива

Для поэлементного ввода и вывода матрицы используется двойной цикл for....

```
int main()
{      int M[2][3] ;
       int  i, j;
  printf("Введите матрицу M\n");
  for( i = 0; i<2; i++)
       for( j = 0; j< 3; j++)
            scanf("%i",&M[i][ j]);
```

Пример организовать вывод матрицы M на экран.

```
for ( i = 0; i<2; i++)
  {
        for ( j = 0; j< 3; j++)
        cout<<m[i][j]<<"\t";
        cout<<"\n";
 }
```

Вид матрицы на экране будет следующим:

1 2 3

4 5 6

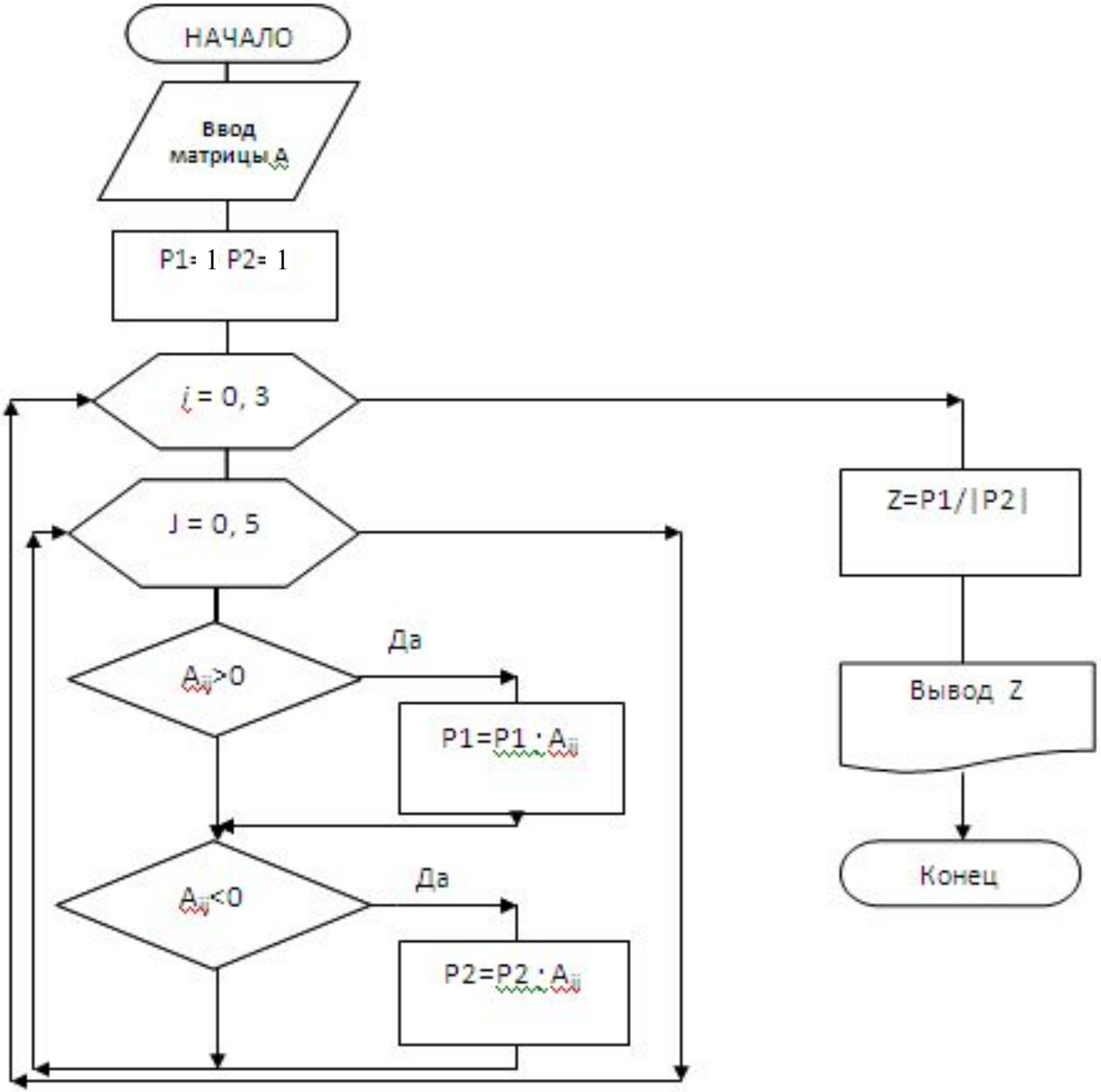
Обработка матриц

Базовыми алгоритмами обработки матриц являются те же алгоритмы, которые используются при обработке одномерных массивов.

Для просмотра всех элементов матрицы организуются два цикла: по строкам и по столбцам.

Пример 1:

Дана матрица вещественных чисел $A = \{a_{ij}\}_{4x6}$. Вычислить значение $Z = P1/|P2|$, где $P1$ и $P2$ – произведения положительных и отрицательных элементов матрицы соответственно.


НАЧАЛО
Ввод матрицы A
P1= 1 P2= 1
i = 0, 3
J = 0, 5
Aij>0
Да
P1=P1 · Aij
Aij<0
Да
P2=P2 · Aij
Z=P1/|P2|
Вывод Z
Конец

```
void main()
 {  float A [4][6] ;    int i;     int j;
    float P1;    float P2;     float Z;
    printf("Введите матрицу A\n");
    for ( i =0; i<4; i++)
        for ( j = 0;j<6; j++)
           scanf("%f",&A[i][j]);
    P1=1; P2=1;           // установка начальных значений
    for ( i = 0; i<4; i++)
       for ( j =0; j<6 ;j++)
       { if (A[i][j]>0) P1= P1*A[i][j];  // произведение   +
         if ( A[i][j]<0) P2 = P2*A[i][j];// произведение   -
       }     Z=P1/abs(P2);
 printf("Z=%10.2f  \n",Z);
 }
```

Пример 2:

Дана матрица вещественных чисел $C = \{c_{ij}\}_{8x4}$. Вычислить среднее арифметическое каждого столбца. Результат оформить в виде одномерного массива $S = \{s_j\}; j = \overline{1,4}$.

```
 void main()
{  float  C[8][4];
   float S[4];
   int  i, j;
   printf("Введите матрицу C:\n");
   for(i=0; i<8;  i++)
        for (j= 0; j<4;  j++)
          scanf("%f",&C[i][j]);
```

```
for (j= 0; j<4;  j++)
     {
      S[j]=0; //начальная установка  элемента  массива S
      for(i=0; i<8;  i++)
        S[j]= S[j] + C[i][j]; //накопление суммы j-го столбца
     S[j]=S[j]/8; //вычисление среднего значения j столбца
      }
for (j= 0; j<4;  j++)
     printf("%8.2f",S[j]); // вывод всех сумм
 printf("\n");
}
```

При обработке матрицы часто возникает необходимость просмотра не всей матрицы, а ее фрагмента.

Эта задача решается путем управления индексами в циклах.

M:

5	-9	8	10
0	4	-2	6
11	7	2	-1

int M[3][4];

```
for(i=1;i<3;i++)
    for(j=1;j<3;j++)
<обработка фрагмента>
```

Работа с треугольными фрагментами:

5	-9	8
0	4	-2
11	7	2

```
int M[3][3];
. . .
for(i=0;i<3;i++)
    for(j=0;j<=i;j++)
<обработка фрагмента>
```

С помощью указателей можно создать динамический многомерный массив.

Например:

```
int nstr, nstb;
cout << «Введите кол-во строк и столбцов:";
cin>> nstr>>nstb;
int **a=new int *[nstr];
for(int i=0;i<nstr;i++)
   a[i]=new int [nstb];
```

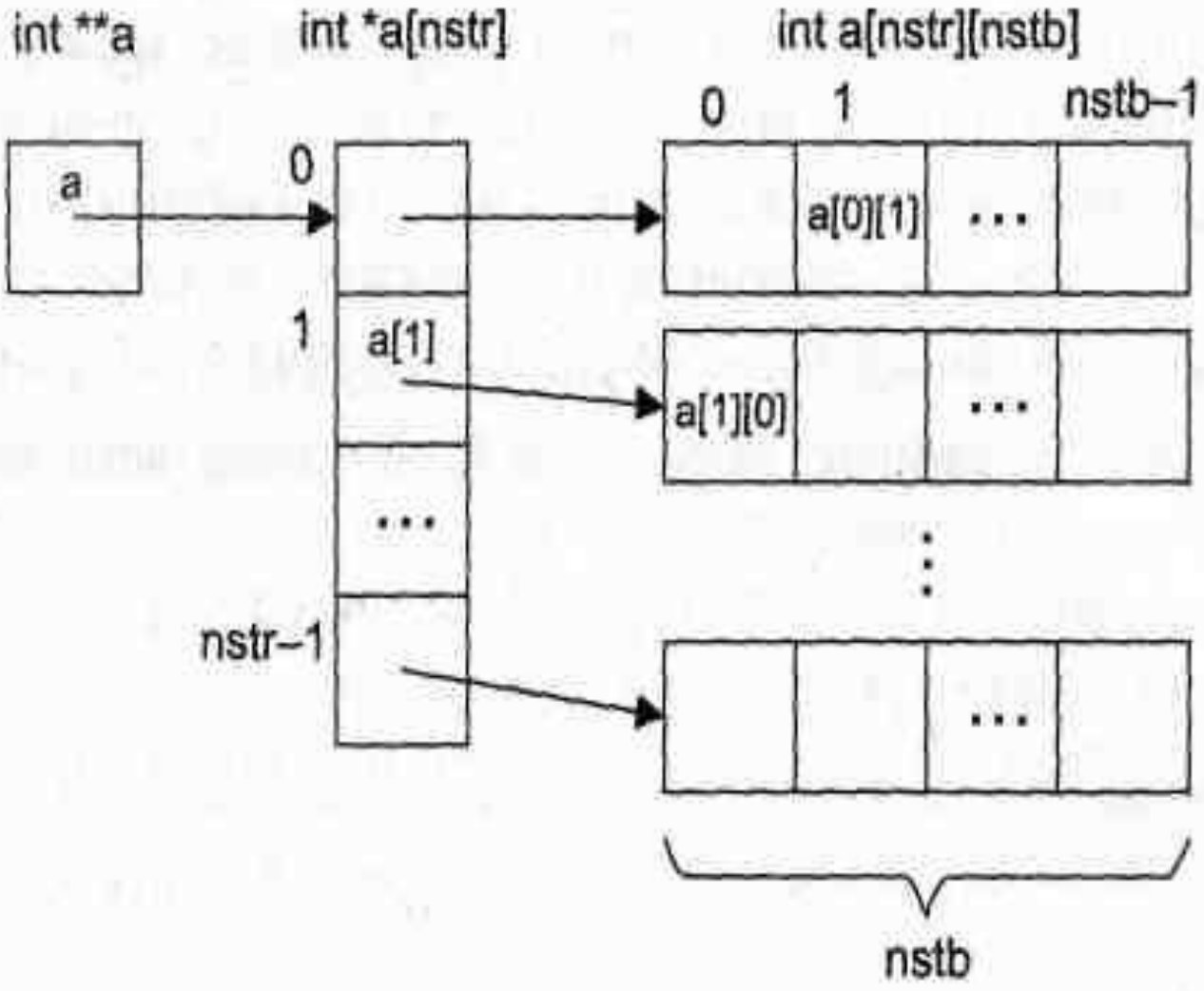


Рис. 1.10. Выделение памяти под двумерный массив

Пример 1:

Дана целочисленная матрица $A = \{a_{ij}\}_{4x4}$.

Упорядочить главную диагональ матрицы.

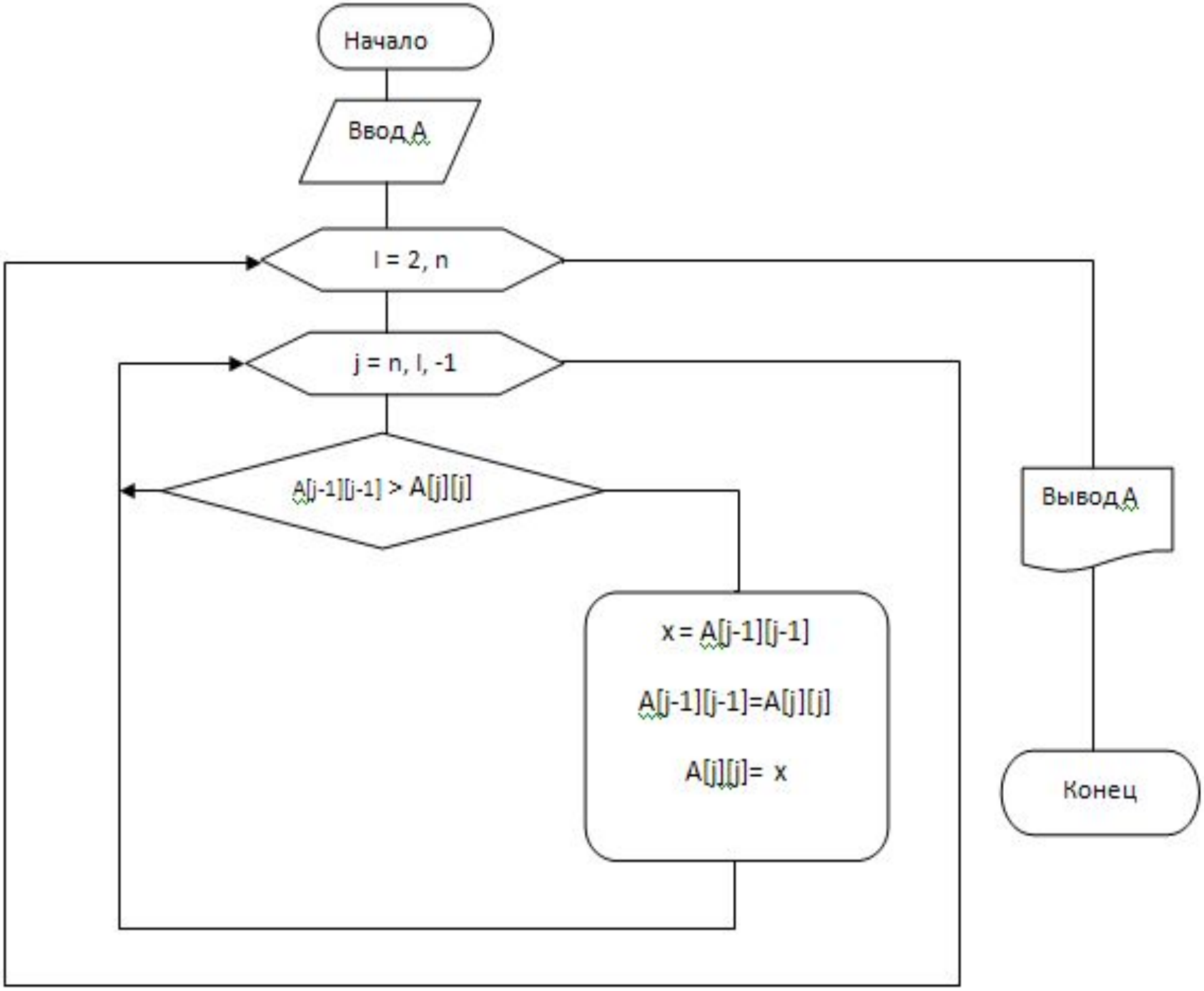
Решение:

Элементы главной диагонали имеют одинаковые индексы: A[0][0],A[1][1], A[2][2]…A[i][i] .

Для упорядочивания элементов можно использовать любой типовой метод сортировки (например, метод пузырька).

Просматривается массив с конца и сравнивается каждый j-й элемент с j-1. Если j-й элемент меньше j-1, то элементы меняются местами.

Действия повторяются n-1 раз.


Начало
Ввод A
i = 2, n
j = n, i, -1
A[j-1][j-1] > A[j][j]
x = A[j-1][j-1]
A[j-1][j-1]=A[j][j]
A[j][j]= x
Вывод A
Конец

```
//ввод матрицы
for(i=1;i<n;i++)
  { for(j=n-1;j>=i;j--)
      if(A[j][j]<A[j-1][j-1])
          {
            x=A[j][j];
           A[j][j]=A[j-1][j-1];
          A[j-1][j-1]=x;
          }
   }
//вывод матрицы
```

Текстовые данные

В языке C++ текстовая информация представляется двумя типами данных: с помощью символов и строк - массивов символов.

Символьная переменная объявляется с помощью ключевого слова char, например:

char cr; char x= ‘!’ ;

Для ввода используются функции:cin-поточный ввод, scanf() – форматированный ввод, getchar() или getch() – специальные функции для ввода символа.

```
char a,b,c;
   printf("Введите исходные данные");
   cin>>a>>b>>c;
```

СТРОКИ

Строка представляет собой массив символов, заканчивающийся нуль-символом.(\0)

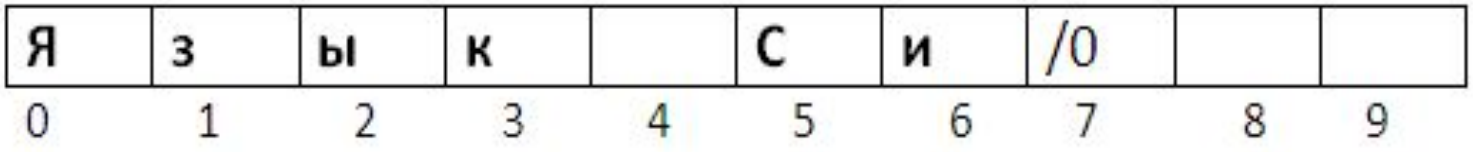
По положению нуль-символа определяется фактическая длина строки.

char st[30];

В квадратных скобках указывается максимальное число символов в строке st.

Начальное значение строки можно задать при ее объявлении следующим образом:

char s[10] = "Язык Си";



При вводе строк обычно используются функции cin и scanf().

```
char fam[20];
  printf ("Введите фамилию студента");
  scanf("%s", fam);
```

или

```
  cin>>fam;
```

Для ввода текста содержащего пробелы следует использовать специальную функцию gets().

```
  printf(“Введите фамилию студента”);
 gets(fam);
```

Вывод строк осуществляется с помощью функции printf(), cout и специальной функции puts().

```
printf(“ %20s”,fam);
cout<< fam;
puts(fam);
```

Обработка строковых данных

К любому символу строки можно обратиться как к элементу одномерного массива, например, запись st[i] определяет i-ый символ в строке st.

Для просмотра строк необходимо организовать цикл по порядковым номерам символов.

Например: Дано предложение. Определите количество слов в нем.

```
int main()
{ char slova[120];
    int i, n, k=1;
  printf("Введите предложение\n");
  gets(slova);
  n= strlen(slova);//функция strlen() возвращает длину строки
  for(i=0;i<n; i++)
        if(slova[i]==' ')k++; //поиск и подсчет пробела
     printf("k=%d\n",k);
      return 0;
}
```

Стандартные функции обработки строк

Определение длины строки: strlen(str)

Сравнение строк: strcmp(str1,str2) – сравнивает строки и возвращает 0, если они одинаковы; результат отрицателен, если str1<str2 и положителен, если str1>str2.

Сцепление строк: strcat(str1,str2)-склеивает строки в порядке их перечисления.

Копирование строк: strcpy(str1,str2) – копирует строку str2 в строку str1.

Поиск адреса символа в строке: strchr(st, ch) - результатом выполнения поиска является адрес найденного символа в строке st, иначе возвращается нулевой адрес.

Пример. В заданной фамилии определить порядковый номер символа ‘n’.

```
#include "stdafx.h"
#include<string.h>
int main()
{ char fam[] = "Ivanov";
   char faml[20];
   char  a='n';    char *p=0;
    p=strchr(fam,a);
    if(p)
         printf("|%s|%d\n", fam, p-fam);
    else
     printf("нет такого символа в фамилии!\n");
return 0;}
```

Пример2.

Исходным текстом является предложение, заканчивающееся точкой. Слова в предложении отделяются друг от друга одним пробелом. Определить самое длинное слово в предложении.

```
int main()
{ char slovo[12],x[120];    int i,m=0,n,k=0;
gets(x);                    // ввод строки x
for(i=0; i<strlen(x); i++) //цикл до конца строки x
if(x[i]!=' ')  k++;         // считаем символы до пробела
else
    { if (k>m){ m=k;n=i;}//поиск мах значения счетчика к
    k=0;  }
     k=0;
     for(i=n-m;i<n;i++)//выбор самого длинного слова
     slovo[k++]=x[i];
     slovo[k]=0;
 printf("%s   \n%s\n",slovo,x); //вывод слова и всей
строки
return 0;      }
```

Структуры данных

Очень часто при обработке информации приходится работать с блоками данных, в которых присутствуют разные типы данных.

Структура - это тип данных, который может включать в себя несколько **полей** – элементов разных типов.

В общем случае при работе со структурами следует выделить четыре момента:

- объявление и определение типа структуры,
- объявление структурной переменной,
- инициализация структурной переменной,
- использование структурной переменной.

Структура состоит из фиксированного числа элементов, называемых полями. Например, структурой можно считать строку экзаменационной ведомости:

Андреева С.В. 4 5 5

Данная структура состоит из четырех полей: одно поле - строка (ФИО студента) и три числовых поля (оценки студента по предметам).

Описание типа структуры делается так:

```
struct Имя
{ <тип> <имя 1-го поля>;
  <тип> <имя 2-го поля>;
        ……
  <тип> <имя последнего поля>;
};
```

Например, задание типа записи строки экзаменационной ведомости выглядит так:

```
struct student
{ char  fam[20];
int mathematics, informatics, history;
};
```

Тогда при описании переменных можно использовать этот тип:

```
struct student   X;
```

Здесь X - переменная типа структура;

struct student - тип;

fam, mathematics, informatics, history - поля структуры.

Чтобы упростить обращение к структурному типу, можно воспользоваться директивой #define.

Например, для предыдущей структуры:

```
#define stud struct student
stud
{
    char  fam[20];
    int mathematics, informatics, history;
   };
```

Теперь идентификатор stud заменит в любом месте программы громоздкий тип struct student.

Теперь описании переменной типа структура будет выглядеть так:

```
stud   X;
```

В более поздних версиях языка С ключевое слово typedef позволяет создать синоним типа, который удобно использовать для объявления переменных структурного типа. Например:

```
typedef struct student
{
char  fam[20];
    int mathematics, informatics, history;
} STUD;
```

Идентификатор STUD представляет собой синоним типа struct student. С помощью синонима STUD можно объявить переменную:

```
STUD X;
```

Для обращения к отдельным полям переменной типа структура используется составное имя:

<имя переменной>.<имя поля>

Например, для переменной X обращения к полям записываются следующим образом:

X.fam, X. mathematics, X. informatics, X. history.

Структурную переменную можно инициализировать явно при объявлении:

STUD X={"Андреева С.В.", 4, 5, 5};

Над структурами возможны следующие операции:

- присваивание значений структурной переменной;
- ввод и вывод значений переменных структурного типа;
- сравнение полей переменных структурного типа.

Операция присваивания применима, как к отдельным полям переменной структурного типа, так и к переменным в целом.

```
#include "stdafx.h"
#include <string.h>
 typedef struct student  // описание структуры
{ char  fam[20];
    int mathematics, informatics, history;
} STUD;
 int  main()
 { STUD X;          //описание переменной
    strcpy(X.fam, "Андреева С.В. ");
    X. mathematics=4;     X. informatics=5;
    X. history=5;
    printf("\n %s  %d   %d   %d", X.fam,
X.mathematics, X.informatics,X.history);/*вывод
информации
   . . .
}
```

Для структурного типа возможно присваивание значений одной структурной переменной другой структурной переменной, при этом обе переменные должны иметь один и тот же тип.

```
STUD X, Y;
. . .
Y=X;        // копирование информации из X в Y
. . .
```

Работа со структурной переменной обычно сводится к работе с отдельными полями структуры. Такие операции, как ввод с клавиатуры, сравнение полей и вывод на экран применимы только к отдельным полям.

Пример задачи с использованием структурированных данных

Рассмотрим пример программы, в которой вводится информация об абонентах сети: ФИО, телефон и возраст. В программе выбираются абоненты моложе 25 лет и их список выводится в алфавитном порядке.

```
 #include "stdafx.h"
#include<conio.h>
#include<stdlib.h>
typedef struct abon  //описание структуры
{ char f[10],i[10],o[10];
  long tel;
  int voz;
}ABON;
```

```
const int n=5;
int i,k,j;
 int main()
 { ABON z[n],y[n];    //описание массивов структур
   ABON x;
for (i=0; i<n; i++)//ввод в цикле исходной информации о пяти абонентах
 {printf("Введите ФИО абонента:");
 scanf("%s%s%s",z[i].f, z[i].i, z[i].o);
 printf("введите его телефон и возраст:");
 scanf("%ld%d",&z[i].tel,&z[i].voz);
}
printf("-----------------------------------------\n");
```

```
printf("| Фамилия |  Имя   | Отчество| Телефон | Возраст |\n”);
printf("-----------------------------------------------------\n");
for (i=0;i<n;i++) //вывод в цикле информации
 printf("|%9s|%8s|%9s|%7ld  |  %5d   |\n", z[i].f,z[i].i,z[i].o,
z[i].tel,z[i].voz);
printf("-----------------------------------------------------\n");
for (i=0;i<n;i++)
  {if(z[i].voz<25) // поиск абонента  моложе 25 лет
   y[k++]=z[i];  }
for(i=1;i<k;i++)//сортировка списка абонентов моложе 25 лет
    for(j=k-1;j>=i;j--)
  if(strcmp(y[j],y[j-1])<0)
      {x=y[j];
      y[j]=y[j-1];
      y[j-1]=x;}
```

```
printf("mologe 25\n");
printf("-----------------------------------------------\n");
printf("| Фамилия |  Имя   |Отчество| Телефон |  Возраст |\n");
printf("-----------------------------------------------\n");
for (i=0;i<k;i++) // вывод отсортированного списка
 {printf("|%9s|%8s|%9s|%7ld  |  %5d   |\n", y[i].f,y[i].i,
y[i].o, y[i].tel,y[i].voz);
 }
printf("-----------------------------------------------\n");
return 0;
   }
```

ФУНКЦИИ

Любая программа на языке высокого уровня может быть разбита на ряд логически завершенных программных единиц - подпрограмм. Такое разделение вызвано двумя причинами.

1) ***Экономия памяти.***
2) ***Структурирование программы***

В языке Си существует один вид подпрограмм, который называется функция. Каждая программа имеет главную функцию (main), которая служит точкой входа в программу и может иметь произвольное число функций.

Ниже схематично приведена структура программы, в которой описана подпрограмма-функция.

```
    #include "stdafx.h "
Тип   имя_функции(тип параметр1, тип параметр2,...)
    {
       тело функции
    }
……….
main()  //начало главной функции
  {
      Обращение к подпрограмме
      ……..
       }   //конец главной функции
```

Функция – это автономная часть программы, реализующая определенный алгоритм и допускающая обращение к ней из различных частей программы.

Общий вид описания функции

```
Тип   Имя(список формальных параметров)
{   Описание локальных переменных;
    Операторы тела функции;
    return  результат;
}
```

Тип указываемый в заголовке функции определяет тип результата ее работы, который будет возвращаться в точку вызова.

Для возврата значения в теле функции должен быть оператор *return*. В дальнейшем будем называть такую функцию *типизированной*.
Если функция не должна возвращать результат, то она считается *нетипизированной*, что задается ключевым словом *void,* стоящим на месте типа. В этом случае оператор *return* в функции не требуется.

```
void   Имя(список формальных параметров)
{
    Описание локальных переменных;
    Операторы тела функции;
}
```

Список формальных параметров обеспечивает передачу исходных данных в функцию.

Параметры, указанные в заголовке функции, называются формальными, а параметры, указываемые при ее вызове – фактическими.

Рассмотрим пример функции для вычисления максимального значения из двух заданных.

```
int max(int a, int b)
  {  int c;
     if (a>b)  c=a;
     else        c=b;
     return c;
}
```

Обращение к *типизированной* функции не является специальным оператором, а включается в состав выражения. Результат выполнения функции возвращается в точку вызова через имя функции.

Приведем пример вызова приведенной выше функции:

```
void main()
{    int x,y,z;
     printf("Введите x  и y:");
     scanf("%d%d",&x,&y);
     z=max(x,y);
     printf("max=%d\n",z); }
```

При каждом вызове функции происходит замена формальных параметров **(int a, int b)** на фактические **(x,y)**. Вычисленный результат возвращается в выражение. Далее вычисляется значение **z** и выводится на экран.

Формальные и фактические параметры должны быть согласованы друг с другом по количеству, типу и порядку следования.

Для того чтобы функция могла быть вызвана, необходимо, чтобы до ее вызова о ней было известно компилятору.

Это значит, что либо мы текст функции должны поместить до main(), либо перед main() записывается прототип функции. Прототип функции по форме аналогичен заголовку функции, в конце которого ставится ";".

```
#include "stdafx.h"
int max(int   , int ) ;  //прототип функции
void main()
{. . .
 z=max(x,y);
. . . }
int max(int a, int b) // заголовок функции
  { int c;
    if (a>b)   c=a;  else  c=b;
    return c;}
```

Механизм передачи параметров

В языке C++ существует два механизма передачи параметров в функции: по значению и по адресу.

Параметры, передаваемые по значению, играют роль входных параметров. Для них в памяти компьютера выделяются ячейки, в которые передаются копии значений параметров. При выполнении функции значения формальных параметров могут измениться, однако соответствующие им фактические параметры останутся без изменения.

При передаче параметров по адресу все действия в процедуре выполняются непосредственно над фактическим параметром, а не его копией.

Рассмотрим два примера, иллюстрирующих механизмы передачи параметров

Пример 1

```
#include "stdafx.h"
 void  Z (int  y)
 {
    y=1;
  }
  void main()
  { int x;
     x=0;
     Z(x);
    printf("x=%d", x);
  }
```

Пример 2

```
#include "stdafx.h"
  void  Z (int  *y)
   {
     *y=1;
   }
  void main()
   { int x;
      x=0;
      Z(&x);
      printf("x=%d", x);
   }
```

Массивы, так же как и простые переменные, можно передавать в функции в качестве параметров. Так как имя массива – это адрес, то передача массива происходит всегда по адресу.

Обратим внимание, что в заголовке функции размер массива рекомендуется указать отдельно. Тогда функцию можно использовать для работы с массивом разной длины. Нельзя объявлять массив-параметр как *A[N]*, а только как *A[]* или **A*.

Перегрузка функций в C++

Цель перегрузки состоит в том, чтобы функция с одним именем по разному выполнялась и возвращала разные значения при обращении к ней с различными типами и различным числом фактических параметров.

Для обеспечения перегрузки функций необходимо для каждого имени функции определить сколько различных функций с ним связано.

Приведем пример перегруженной функции для поиска максимума (для 4 разных типов данных)

```
int max_element ( int n, int a[ ])
// находит максимальный элемент для массива типа int
{ int max=a[0];
for ( i=1; i<n; i++)
if (a[i]>max) max=a[i];
return max;
}
long max_element ( int n, long a[ ])
// находит максимальный элемент для массива типа long
{ long max=a[0];
for ( i=1; i<n; i++)
if (a[i]>max) max=a[i];
return max;
}
```

```
double max_element ( int n, double a[ ])
//находит максимальный элемент массива типа double
{ double max=a[0];
for ( i=1; i<n; i++)
if (a[i]>max) max=a[i];
return max;
}
float max_element ( int n, float a[ ])
// находит максимальный элемент массива типа float
{ float max=a[0];
for ( i=1; i<n; i++)
if (a[i]>max) max=a[i];
return max;
}
```

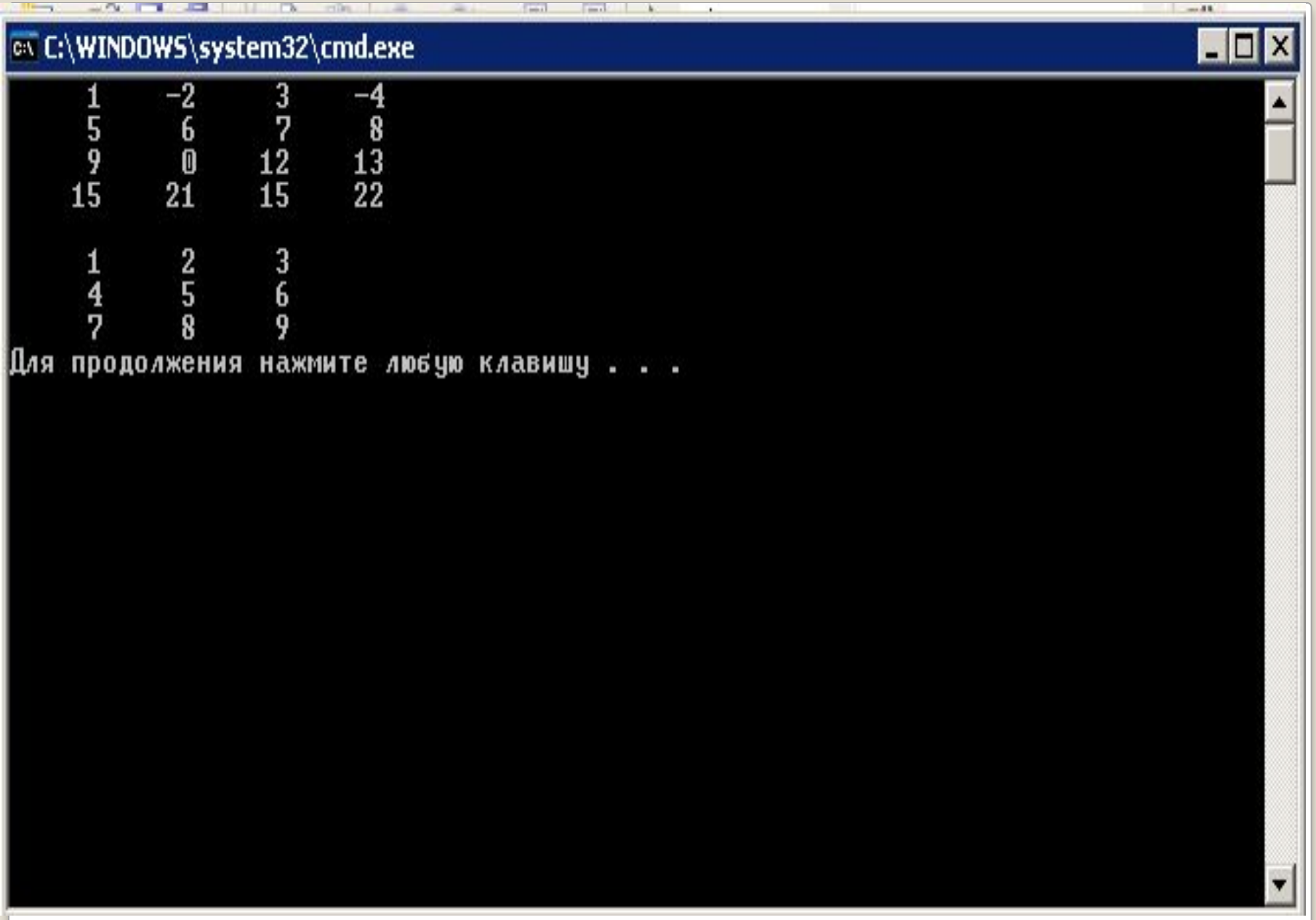
```
void main ( )
{
int x[]={10, 20, 30, 40, 50, 60};
long y[]={12L, 44L, 22L, 37L,30L};
. . . . . .
int m1=max_element(6, x );
long m2=max_element(5, y);
. . . . .
}
```

Если в функцию передаётся двумерный массив, то описание соответствующего аргумента функции должно содержать количество столбцов; количество строк - несущественно, поскольку фактически передаётся указатель. Например: int X[ ][5], или X [5][5].

```
#define k   4
void vivod(int n, int m, int A[][k])
{for (int i=0;i<n;i++)
  {
   for(int j=0;j<m;j++)
     printf("%6d",A[i][j]);
   printf("\n");
  }
}
```

Если при передаче многомерных массивов все размерности неизвестны, то можно передавать адрес первого элемента, а внутри функции интерпретировать массив как одномерный, а индексы пересчитывать в программе.

```
void vivod(int n, int m, int *A)
{for (int i=0;i<n;i++)
  {
   for(int j=0;j<m;j++)
     printf("%6d",A[i*m+j]);
   printf("\n");
  }
}
```


C:\WINDOWS\system32\cmd.exe
1 -2 3 -4
5 6 7 8
9 0 12 13
15 21 15 22
1 2 3
4 5 6
7 8 9
Для продолжения нажмите любую клавишу . . .

Функции с переменным числом параметров

В Си допустимы функции, у которых при компиляции не фиксируется число параметров.

Каждая функция с переменным числом параметров должна иметь хотя бы один обязательный параметр.

<тип> <имя_функции>(явные параметры, . . .)

Как правило количество параметров определяется значением обязательного параметра.

Для доступа к списку параметров используется указатель *p типа int. Он устанавливается на начало списка параметров в памяти.

```
int sum (int k, . . .)
{
int *p = &k; //настроили указатель на параметр k
int s=0;
for ( ; k!=0;k--)
s+=*(++p);
return s;
}
void main( )
{
cout<<”\nСумма(2,4,6)= ”<<sum(2,4,6);
cout<<”\nСумма(4,1,2,3,4)= ”<<sum(4,1,2,3,4);
}
```

Рекурсия

В теле функции известны все объекты, описанные во внешнем блоке, т.е. все глобальные переменные и имя самой функции.

```
#include <stdafx.h>
const int n=5; int x;//описание глобальных объектов
int sum (int k, . . .)
{int  p; // описание локальных объектов
 …
return s;}
void main( )
{int a; // описание локальных объектов
…
}
```

Таким образом, внутри любой функции можно вызывать любую доступную функцию, в том числе и саму себя. ***Ситуация, когда функция вызывает саму себя, называется рекурсия.***

Рекурсия возможна благодаря тому, что при вызове функции создаются новые экземпляры локальных переменных, которые сохраняются во внутреннем стеке машины.

Стек функционирует по принципу LIFO - Last In – First Out (последний вошел – первый вышел). Переменные помещаются в стек одна за другой и выбираются из стека в обратном порядке.

Обязательным элементом всякого рекурсивного процесса является утверждение, определяющее условие завершения рекурсии. Оно называется опорным условием рекурсии.

Если опорное условие выполняется, то может быть задано некоторое фиксированное значение, заведомо достижимое в ходе вычисления. Это позволит организовать своевременную остановку рекурсивного процесса.

Рассмотрим пример вычисления факториала 5.

5!=1x2x3x4x5, где 1x2x3x4- это 4!

Т.е 5!=(4!)x5

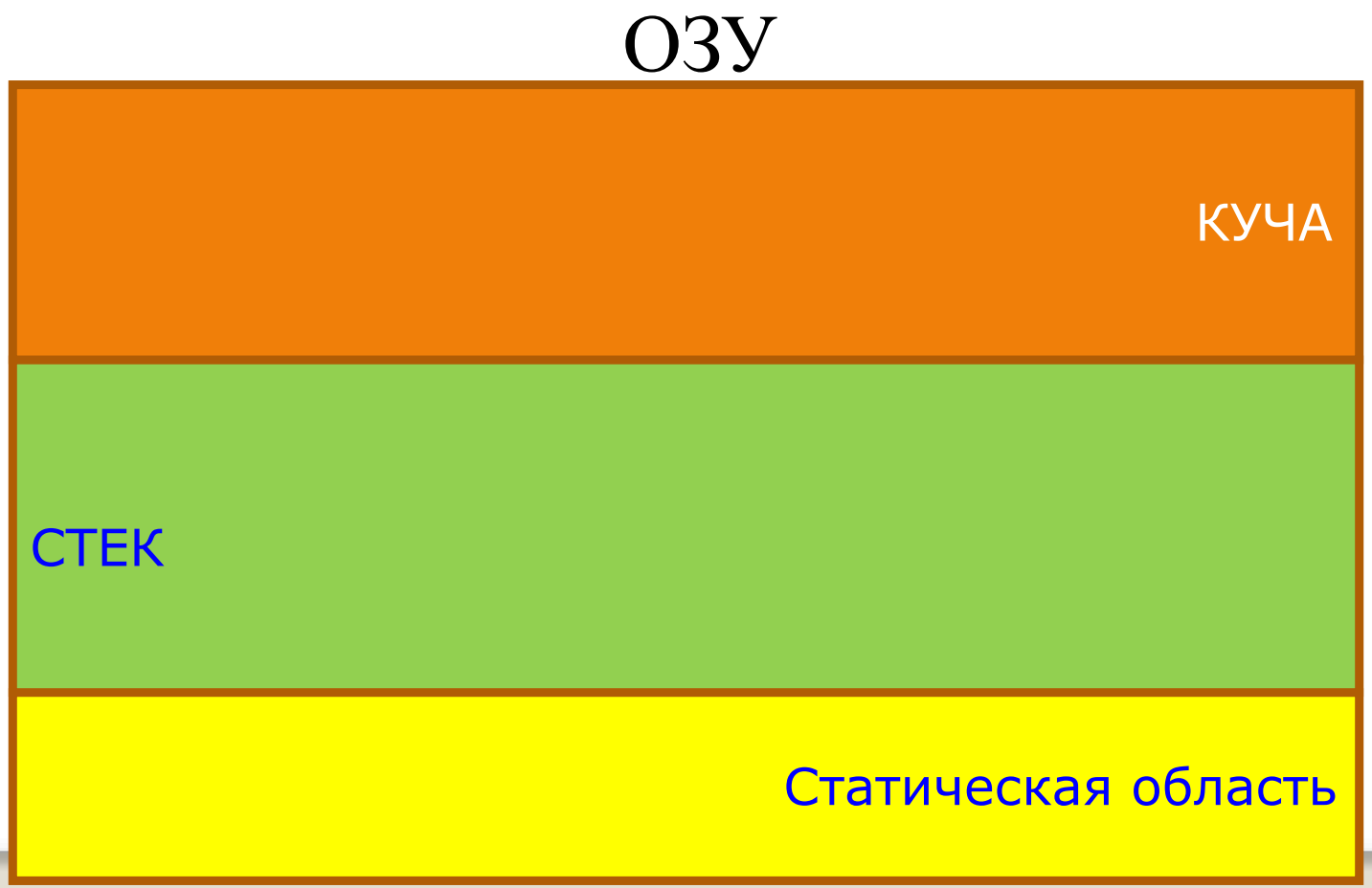
Факториал нуля равен 1. Отсюда формула вычисления N-факториала:

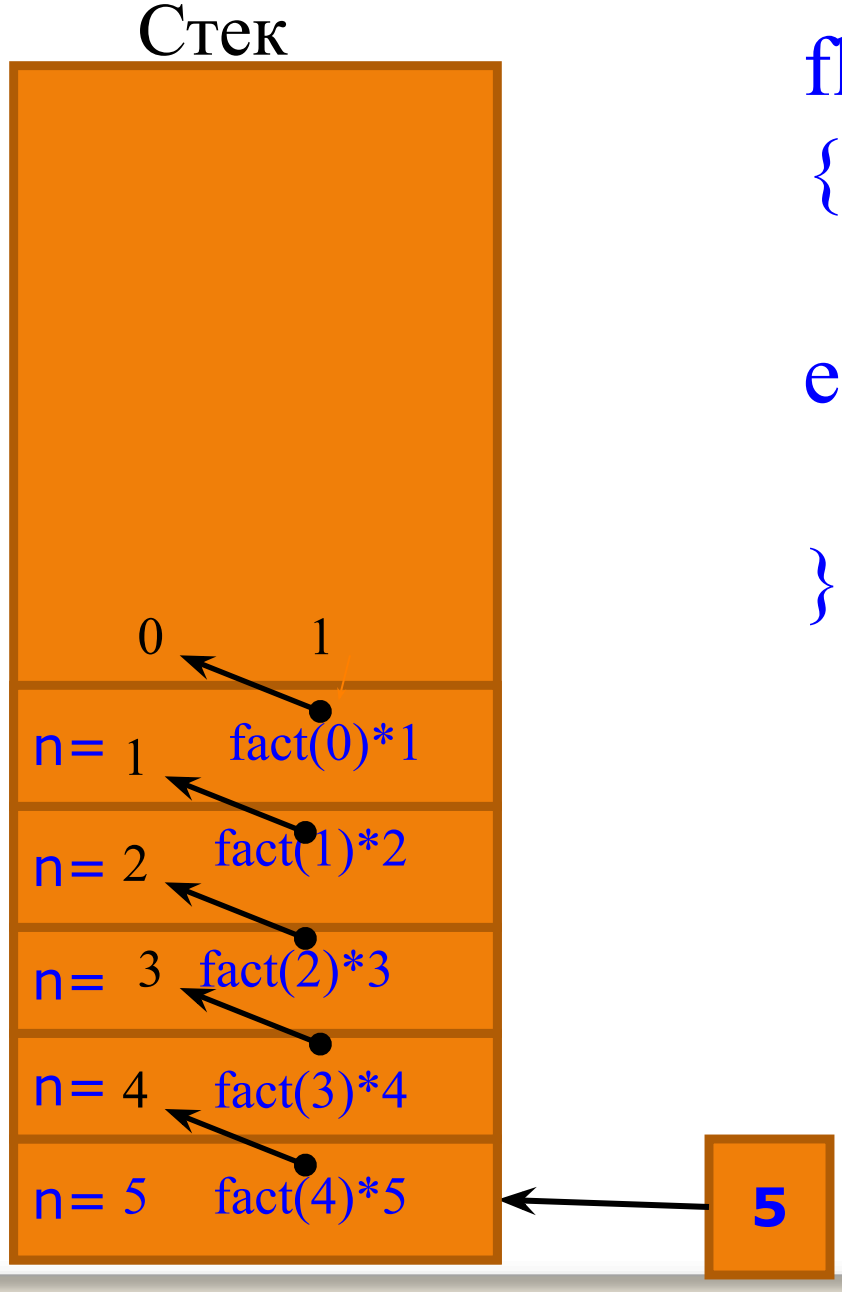
$$N! = \begin{cases} 1, & \text{если } N = 0 \\ (N-1)! \cdot N, & \text{если } N > 0 \end{cases}$$

Реализуем выч-е факториала в виде функции:

```
#include "stdafx.h"
float fact(int n)
{  if (n==0)
        return 1;
else
       return (fact(n-1)*n);
}
```

```
int main()
{
    printf("факториал 5=%f\n",fact(5));
    return 0; }
```






```
float fact(int n)
{  if (n==0)
     return 1;
else
    return (fact(n-1)*n);
}
```

Пример: Вычислить кол-во нулей в массиве A[10]

```
# include “stdafx.h”
const int n=10;
Int kol(int i,int *A)
{ if (i==n) return 0;
   else {
          if(A[i]==0) return kol(i+1,A)+1;
          else return kol(i+1,A);
         }
}
int main()
{ int y[]={1,0,2,5,4,0,1,3,0,4,3};
  int x=kol(0,y);
```

СПАСИБО за ВНИМАНИЕ