

## Организация консольного – ввода/вывода данных (т.е. в режиме чёрного экрана)

```
#include <iostream>; //директива процессора, предназначена для включения в  
исходный текст содержимое заголовочного файла, имя которого < iostream>,  
содержащий описания функций стандартной библиотеки ввода/вывода для  
работы с клавиатурой и экраном.  
  
using namespace stg; //директива означ. что все определённые ниже имена будут  
отн-ся к пространству имён std  
  
Int main() //имя функции, кот. не содержит параметров и должна возвращать  
значение типа Int  
  
{Int a,b; //объявление двух переменных типа Int - целый тип  
  
cout <<"введите два целых числа"<<endl; //оператор вывода данных на экран ,  
    << - операция помещения данных в выходной поток;  
    endl - манипулятор, переводит сообщение на новую сточку.  
  
cin >>a >>b; //оператор ввода данных с клавиатуры,  
    >> - операция для извлечения данных из выходного потока, читает значения  
из cin и сохр. их в переменных.  
  
cout >>"их сумма равна"<<a+b; //оператор вывода  
  
return 0;} //оператор вывода
```

# Стандартные типы данных

- Целые типы данных – `short`, `int`, `long` и спецификаторы (*signed*, *unsigned*);
- Вещественные типы – `float`, `double`, `long double`;
- Символьные типы – `char`, `wchar_t`;
- Логический тип – `bool`, принимающий значения (`true`-истина, `false`-ложь);

## Константы (`const`)

`a=b+2,5` //неименованная константа;

`'1L'` - целочисленный литерал (тип `long`);

`"8"` – целочисл.литерал (тип `int`);

`'f'`, – символьный литерал, `'\n'`-конец строки

Формат описания именованной константы:

[<класс памяти>]`const` <тип> <имя именованной константы>=<выражение>;

`const int l= - 124;`

`const float k1=2,345, k=1/k1`

*Класс памяти*- это спецификатор, определяющий время жизни и область видимости данного объекта.

*Выражение*, определяет значение именованной константы, т.е инициализирует её.

# Переменные

Формат описания переменных:

[<класс памяти><тип><имя>[=<выражение> | (<выражение>)];

Пример:

```
int l,j;
```

```
double x;
```

Значение переменных должно быть определено с помощью:

1. оператора присваивания: `int a;` *//описание переменной*

```
int= a; //опред.значения.переменной
```

2. оператора ввода: `int a;` *//описание переменной*

```
cin>>a; //опред.знач.переменной
```

3. инициализация – опред.значения переменной на этом этапе описания.

```
int i=100 //инициализация копией
```

```
int i (100); // прямая инициализация
```

# Управление форматом вещественных типов данных

Сущ. три аспекта оформления значения с плавающей запятой которыми можно управлять:

- *точность* (кол-во отображаемых цифр), изменяется с помощью манипулятора *setprecision*;
- *форма записи* (десятичная или экспоненциальная);
- *указание десятичной точки для значения с плавающей запятой, являющихся целыми числами*.

- |   |                             |
|---|-----------------------------|
| - #include <iostream>   | Результат работы программы: |
| - #include <iomanip>  | 1.23e+004                   |
| - using namespace std;  | 12345.7                     |
| - int main()  | 12345.6789                  |
| - { double i=12345,6789;  |                             |
| - cout << setprecision(3)<<i<<endl;   |                             |
| - cout << setprecision(6)<<i<<endl;   |                             |
| - cout << setprecision(9)<<i<<endl;   |                             |
| - return 0;}  |                             |
| - (для использования манипуляторов <i>endl</i> с аргументами требуется подключить |                             |

# Управление размещением данных на экране

## □ Используются манипуляторы:

1. **left** – выравнивает вывод по левому краю;
2. **right** – выравнивает вывод по правому краю;
3. **internal** – контролирует размещение отрицательного значения: выравнивает знак по левому краю, а значение по правому, заполняя пространство между ними пробелами;
4. **setprecision(int w)** – устанавливает max кол-во цифр в дробной части для вещественных чисел;
5. **setw(int w)** – устанавливает max ширину поля вывода;

*Пример:*

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{ cout <<“1” <<setw(10) <<“Ivanov” <<endl;
  cout <<“2” <<setw(10) <<left<<“Ivanov” <<endl;
  cout <<“3” <<setw(10) <<right<<“Ivanov” <<endl;
  return 0;}
```

*Получим:*

1. Ivanov;
2. Ivanov;
3. Ivanov;

# Задание

- С помощью данных манипуляторов запишите программу, где выравнивание отриц-го числа  $-23,4567$  будет только по правому краю.

Должно получиться:      1.  $-23,4567$

2.       $-23,5$

3. -       $23,5$

# Операции. Унарные операции

- *Операции увеличения (декремента) и уменьшения (инкремента)*

на *1(++ и --)*; записываются в двух формах:

*Префиксия* – операция записывается перед операндом и увеличивает свой операнд на 1 и возвращает изменённое значение как результат

*Постфиксия* – операция записывается после операндом, уменьшает свой операнд на 1 и возвращает изменённое значение как результат.

Пример:

```
#include <iostream>
int main()
using namespace std;
{ int x=3, y=4;
  cout << ++x << "\t" << --y << endl;
  cout << x++ << "\t" << y-- << endl;
  cout << x << "\t" << y << endl;
  return 0;}
```

# Операции отрицания (-,!)

- *(-)* - **унарный минус** – изменяет знак операнда целого или вещественного типа на противоположный;
- *(!)* – **логическое отрицание**, даёт в результате значение 0(ложь), если операнд отличен от 0(истина), если равен операнд 0 (ложь);
- *тип операнда может быть любой.*

## Пример:

```
#include <iostream>
int main()
using namespace std;
{ int x=3, y=0;
  bool f=false, v=true;
  cout <<-x<<“\t”<<!y<<endl;
  cout <<-y<<“\t”<<!y<<endl;
  cout <<v<<“\t”<<!v<<endl;
return 0;}
```



# Бинарные операции

- *Арифметические операции*: умножение. $(*)$ , деление. $(/)$ , остаток от деления. $(%)$ ; слож. $(+)$ , вычит. $(-)$

Рассмотрим операции деления и остаток от деления:

```
#include <iostream>
using namespace std;
int main()
{ cout <<100/24<<“\t”<<100/24<<endl;
  cout <<100/21<<“\t”<<100,0/24<<endl;
  cout <<21%3<<“\t”<<21%6<<“-21%8”<<endl;
  return 0;}
```

- *Операции отрицания*  $(-, !)$  унарный минус – изменяет знак операнда целого или вещест-го типа на противоположный.
- *Операции отношения*:  $(<, <=, >, >=, ==, !=)$ , меньше, меньше или равно, больше, больше или равно, равно, не равно, не равно соответственно).

Результатом операций являются значения *true, false*.

## Логические операции (&& и ||)

**И (&&)** - возвращает значение истина тогда и только тогда, когда оба операнда принимают значение истина, в противном случае операция возвращ.знач.ложь.

**ИЛИ ||** - возвращает знач.истина тогда и т.тогда, когда хотя бы один операнд принимает значение истина, в противном случае –ложь

- логические операции выполняются слева направо;
- приоритет операции && выше ||.

### Пример:

```
#include <iostream>
using namespace std;
int main()
{ cout << 'x\t y\t &&\t ||' endl;
  cout << "0\t 1\t" << (0 && 1) << '\t' << (0 || 1) endl;
  cout << '0\t 1\t' << (0 && 1) << '\t' << (0 || 1) endl;
  cout << '1\t 0\t' << (1 && 0) << '\t' << (1 || 0) endl;
  cout << '1\t 1\t' << (1 && 1) << '\t' << (1 || 1) endl;
  return 0;}
```

## Операции присваивания

- формат операция простого присваивания (=):
- **опреанд\_1 = операнд\_2**

пример:  $a=b=c=100$ , это выражение выполняется справа налево, результатом выполнения  **$c=100$** , является число 100, которое затем присвоится переменной  **$b$** , потом  **$a$** .

### □ Сложные операции присваивания:

- **(\*=)** – умножение с присв-ем,
- **(/=)** - деление с присв-ем
- **(%=)** - остаток от деления с присв-ем,
- **(+=)** –сложение с присв.,
- **(-=)** - вычит.с присв.

□ пример: к операнду **\_1** прибавляется операнд **\_2** и результат записывается в операнд **\_2**

□ т.е.  $c = c + a$ , тогда **компактная запись**  **$c += a$**

# Тернарная операция

- *Условная операция (? :)*
- Формат условной операции: **операнд\_1 ? операнд\_2 ? : операнд\_3**
- **Операнд\_1** это логическое или арифметич-ое выражение;
- Оценивается с точки зрения эквивалентности константам true и false;
- **Если** результат вычисления **операнда\_1 равен true**, то результат условной операции **будет значение операнда\_2**, иначе операнда\_3;
- Тип может различаться;
- Условная операция является сокращ. формой услов-го оператора if;

Пример:

```
#include <iostream>
int main()
using namespace std;
{ int x, y,max;
  cin >>x>>y;
  (x>y)? cout <<x: cout<<y<<endl;
  max=(x>y)? x:y;
  cout<<max<<endl;
  return 0;}
```

Результат:

для x=11 и y=9

11

11

*(x>y)? cout <<x: cout<<y<<endl; //1 нахождение наибольшего зна-*

*max=(x>y)? x:y; //2 чения для двух целых чисел;*

*cout<<max<<endl;*

*return 0;}*

# Выражения преобразования типов

- Примеры:
- $(a+0,12)/6$ ;  $x \&\& y \|\| !z$ ;
- $(t*\sin(x)-1,05e4)/((2*k+2)*(2*k+3))4$ ;
- операции выполняются в соответствии с приоритетом;
- если в одном выражении имеются неск. операций одинаково приоритета, то унарные операции выполняются- *справа*
- *налево*, остальные –*слева направо*
- *Т.е:*  $a = b + c$  значит  $a = (b = c)$ ,
- $a + b + c$  значит  $(a + b) + c$
- в выражения могут входить операнды различных типов;
- при одинаковом типе операндов, результат будет им. тот же тип;
- если разного типа операнды, то операнд с более «низким» типом будет преобразован к более «высокому» типу для сохранения значимости и точности:

Тип данных	Старшинство
har	высший
double	
float	
long	
Int	
short	
char	низший

## Неявное преобразование:

```
include <iostream>
using namespace std;
int main()
{ int a=100, b; float c=4,5, d;
  d=a/c; //1- без потери точности
  cout << "d=" <<d<<endl;
  b=a/c; //2- с потерей точности
  cout <<"b="<<b<<endl; return 0;}
```

## □ Задания:

1. Составить программу, которая для заданного значения  $x$  вычисляет значения выражения:

$x^2 + \sin(x+1)/25$ , с учётом приоритета операций в с++:  
( $\text{pow}(x,2) + \sin(x+1)/25$ ;

2. Написать программу, подсчитывающую площадь квадрата, периметр которого равен  $p$ .

Пусть дан квадрат со стороны  $a$ , тогда:  $p = 4a$ ,  $a = p/4$ .....

$$s = a^2$$

.....S=

# Операторы C++

- Программа на языке C++ состоит из последовательности операторов, каждый из них определяет значение некоторого действия;
- Все операторы разделены на 4 группы:
  - операторы следования;
  - операторы ветвления;
  - операторы цикла;
  - операторы передачи управления.

# Операторы следования

- К ним отн-ся : **оператор выражение и составной оператор.**
- **Выражение**, завершающееся точкой с запятой, рассматривается как оператор (вычислении значения выражения или выполнении законченного действия);
  - `++i;` //оператор инкремента
  - `x+=y;` //оператор сложение с присваиванием
  - `f(a, b)` //вызов функции
  - `x= max (a, b) + a*b;` //вычисление сложного выражения
- Частным случаем оператора выражения является **пустой оператор ;** (используется, когда по синтаксису оператор требуется, а по смыслу — нет)
- **Составной оператор** или **блок** представляет собой последовательность операторов, заключенных в фигурные скобки.
- **Блок обладает собственной областью видимости:** объявленные внутри блока имена доступны только внутри блока;
- Составные операторы применяются в случае, когда правила языка



# Операторы ветвления

- К ним относятся : *условный оператор if* и *оператор выбора switch*, они позволяют изменить порядок выполнения операторов в программе.

## *Условный оператор if*

- **if** используется для разветвления процесса обработки данных на два направления.
- if имеет формы: *сокращенную* или *полную*.

- **Формат сокращенного оператора if:**      **if (B) S;**

*B* –логич. или арифметич. выражение, истинность которого проверяется; *S* - один оператор: простой или составной.

- При выполнении сокращенной формы оператора if сначала вычисляется выражение *B*, затем проводится анализ его результата: если *B* истинно, то выполняется оператор *S*; если *B* ложно, то оператор *S* пропускается.
- С помощью сокращенной формы оператора If можно либо выполнить оператор *S*, либо пропустить его.

- **Формат полного оператора if:**      **if (B) S1 ; else S2;**

*S1, S2*- один оператор: простой или составной.

- При выполнении полной формы оператора *if* сначала вычисляется выражение *B*, затем анализируется его результат: если *B* истинно, то выполняется оператор *S1* а оператор *S2* пропускается; если *B* ложно, то выполняется оператор *S2*, а *S1* - пропускается.
- С помощью полной формы оператора *if* можно выбрать одно из двух альтернативных действий процесса обработки данных.

## Примеры записи условного оператора *if*.

- `if (a > 0) x=y;` // сокращенная форма с простым оператором
- `if (++i) {x=y; y=2*z;}` // сокращенная форма с составным оператором
- `if (a > 0 || b<0) x=y; eise x=z;` //полная форма с простым оператором
- `if (i+j-1) { x= 0; y= 1;} else {x=1; y:=0;}` //полная форма с составными операт

- Операторы S1 и S2 могут являться операторами *if*, такие операторы наз. *вложенные*;
- Ключевое слово *else* связывается с ближайшим предыдущим словом *if*, которое еще не связано ни с одним *else*.

### Примеры алгоритмов с использованием вложенных условных операторов:

#### Пример1 Уровни вложенности *If*

```
□ if(A<B)
□   if (C < D)
□     if(E<F)X=Q;
□   else X = R;
□     else X=Z;
□   else X = Y;
```

#### Пример2 Уровни вложенности *if*

```
□ if (A<B)
□   if (C<D) X=Y    2
□   else X=Z:
□     else          1
□     if (E<F) X=R;
□     else X=Q;    2
```

# □ Оператор выбора *switch*

*предназначен для разветвления процесса вычислений на несколько направлений.*

## □ **Формат оператора:**

`switch (<выражение>)`

`{case <константное_выражение_1>: [<оператор 1>]`

`case <константное_выражение_2>: [<оператор 2>]`

`.....`

`case <константное_выражение_n>: [<оператор n>]`

`[default: <оператор> ]}`

□ **Выражение**, стоящее за ключевым словом *switch*, должно иметь арифметич. тип или тип указатель.

□ Все константные выражения должны иметь разные значения, но совпадать с типом выражения, стоящим после *switch*.

□ **Ключевое слово** *case* и расположенное после него константное выражение называют также **меткой** *case*.

- Выполнение оператора начинается с вычисления выражения, расположенного за ключевым словом *switch*.
- Полученный результат сравнивается с меткой *case*.
- Если результат выражения соответствует метке *case*, то выполняется оператор, стоящий после этой метки.
- Затем последовательно выполняются все операторы до конца оператора *switch*, если только их выполнение не будет прервано с помощью оператора передачи управления *break*
- При использовании оператора *break* происходит выход из *switch* и управление переходит к первому после него оператору.
- Если совпадения выражения ни с одной меткой *case* не произошло, то выполняется оператор, стоящий после слова *default*, а при его отсутствии управление передается следующему за *switch* оператору.

- Пример. Известен порядковый номер дня недели. Вывести на экран его название.

```
#include <iostream>
using namespace std;
int main()
{int x; cin >>x;
switch (x)
{ case 1: cout <<"понедельник"; break;
  case 2: cout <<"вторник"; break;
  case 3: cout <<"среда"; break;
  case 4: cout <<"четверг"; break;
  case 5: cout <<"пятница"; break;
  case 6: cout <<"суббота"; break;
  case7: cout <<"воскресенье";break;
  default: cout <<"вы ошиблись";}
return 0;}
```

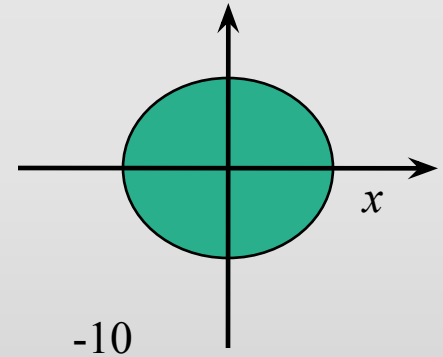
# Использование операторов

## ВЕТВЛЕНИЯ:

- Дана точка на плоскости с координатами  $(x, y)$ .
- Составим программу, которая выдает одно из сообщений «Да», «Нет», «На границе» (в зависимости от того, лежит ли точка внутри заштрихованной области, вне заштрихованной области или на ее границе)

□ *Заданная область разбивает всю плоскость на 3 непересекающихся множества точек.*

- $I_1$  - множество точек, лежащих внутри области;
- $I_2$  - множество точек, лежащих вне области;
- $I_3$  - множество точек, образующих границу области.



- Точка с координатами  $(x, y)$  может принадлежать только одному из них;
- Множества  $I_1, I_2, I_3$  значительно труднее описать математически, чем интервалы в примере 2, поэтому для проверки выбираются те два множества, которые наиболее просто описать математически. (труднее всего описать точки границы области).

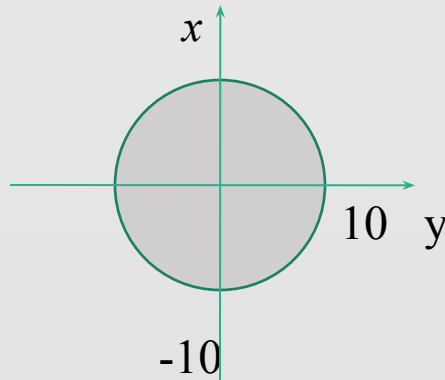
- Для рис. 1 множества задаются следующим образом;  $I_1: x^2 + y^2 < 10^2$ ;  $I_2: x^2 + y^2 > 10^2$ ;  
 $I_3: x^2 + y^2 = 10^2$ .

□ Множества

□  $I_1: x^2 + y^2 < 10^2;$      $I_2: x^2 + y^2 > 10^2;$

□  $I_3: x^2 + y^2 = 10^2.$

рис.1



```
#include < iosfream>
#include <cmath>
using namespace std;
int main()
{ float x,y;
  cout << "x="; cin >>x;
  cout << "y"; cin >>y;
  if (x*x+y*y< 100) //точки внутри области ?
  cout <<"Да";
  else if (x*x+y*y>100) //точки вне области?
    cout<<"НЕТ";
    else cout << "на границе";
return 0;}
```

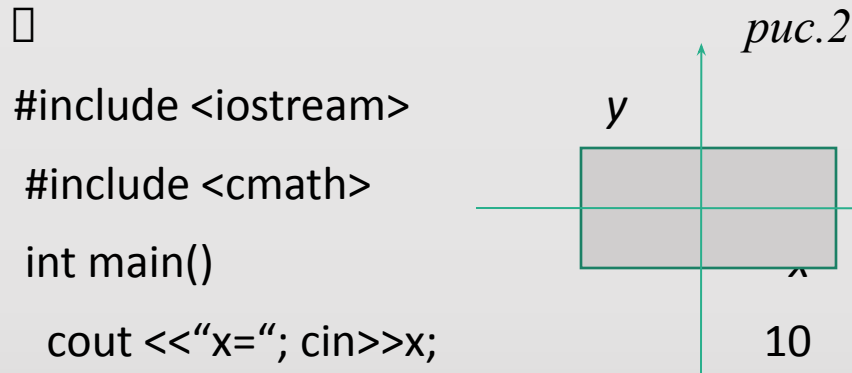
Результат программы:

<u>Координаты точек</u>	<u>ответ</u>
0	да
10	на границе
-12	нет

□ Множества задаются (для рисунка 2) :

$I_1: |x| < 10$  и  $|y| < 5$ ;  $I_2: |x| > 10$  или  $|y| > 5$ ;

$I_3: (|x| \leq 10$  и  $y = 5)$  или  $(|x| \leq 10$  и  $y = -5)$  или  $(|y| < 5$  и  $x = 10)$  или  $(|y| < 5$  и  $x = -10)$ .



```
#include <iostream>
```

```
#include <cmath>
```

```
int main()
```

```
    cout <<"x="; cin>>x;
```

```
    cout <<"y="; cin>>y;
```

```
    if (fabs(x)<10 && fabs(y)<5)    //точки внутри области?
```

```
        cout <<"Да";
```

```
    else if (fabs(x)>10 || fabs(y)>5)    //точки вне области?
```

```
        cout <<"Нет";
```

```
        else cout <<"на границе";
```

```
return 0;}
```

**Результат:**

координаты точек

ответ

0 0

да

10 5

на границе

-12 13

нет

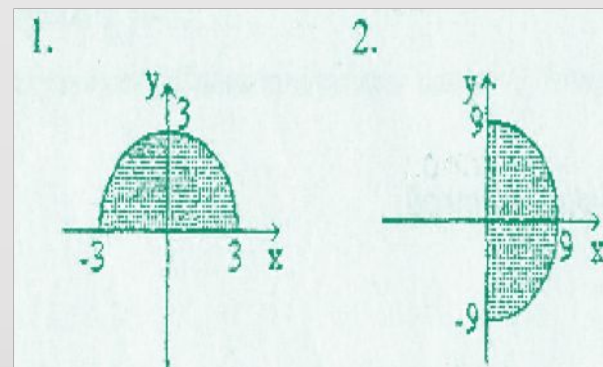
- Дан номер фигуры (1- квадрат, 2 - треугольник);
- по номеру фигуры запросить необходимые данные для вычисления площади;
- произвести вычисление площади фигуры и вывести получ-ые данные на экран.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{ int x;
  cout << "Программа подсчитывает площадь:\n1. квадрата;\n2. треугольника.\n3. выход из программы";
  cout << "Укажите номер фигуры или завершите работу с программой.\n";
  cin >> x;
  switch (x)
  {case 1 :{cout << "введите длину стороны квадрата\n";
    float a; cin >>a;
    if (a>0) cout<< "Площадь квадрата со стороной" <<a <<"равна\t <<a*a;
    else cout << "Квадрат не существует\n";
    break;}
  case 2: {cout<< "введите длины сторон треугольника\n";
    float a,b,c,p, s; cin >>a >>b >>c;
    if (a+b>c && a+c>b && b+c>a)
    {p=(a+b+c)/2; s= sqrt(p*(p-a)*(p-b)*(p-c));
    cout << "Площадь треугольника со сторонами" <<a <<b <<c << "равная\t" <<s;}
    else cout<< "Треугольник не существует\n";
    break;}
  case 3:break;
  default: cout << "Номер фигуры указан не верно\n";}
  return 0;}
```



# задание

- 1. Дана точка на плоскости с координатами  $(x, y)$ .
- Составить программы, которые выдают одно из сообщений:
- «Да», «Нет», «На границе» ,
- в зависимости от того, лежит ли точка:
  - внутри заштрихованной области,
  - вне заштрихованной области
  - или на ее границе.



Области задаются графически следующим образом:

- 2. Дан порядковый номер месяца, вывести на экран его название.
- 3. Дан порядковый номер дня недели, вывести на экран количество дней оставшихся до конца недели.

# Операторы цикла

□ *Операторы цикла используются для организации многократно повторяющихся вычислений.*

- цикл с предусловием `while`,
- цикл с постусловием `do while`
- цикл с параметром `for`.

## **Цикл с предусловием `while`:**

□ Оператор цикла `while` организует выполнение одного оператора (простого или составного) неизвестное заранее число раз.

□ **Формат цикла `while`:** `while (B) S;`

□ **`B` - выражение, истинность которого проверяется (условие завершения цикла);**

□ **`S` - тело цикла: один оператор (простой или составной).**

□ Перед каждым выполнением тела цикла анализируется значение выражения `B`:

- если оно истинно, то выполняется тело цикла, и управление передается на повторную проверку условия `B`;
- если значение `B` ложно - цикл завершается и управление передается на оператор, следующий за оператором `S`.
- если результат выражения `B` окажется ложным при первой проверке, то тело цикла не выполнится ни разу

- *если условие  $B$  во время работы цикла не будет изменяться, то возможна ситуация заикливания, то есть невозможность выхода из цикла.*

Внутри тела должны находиться операторы, приводящие к изменению значения выражения  $B$  так, чтобы цикл мог завершиться.

*Рассмотрим программу вывода на экран целых чисел из интервала от 1 до  $n$ .*

- `#include <iostream>`
- `using namespace std;`
- `int main()`
- `{ int n, i=1;`
- `cout <<"n="; cin >>n;`
- **`while (i<=n)`***//пока  $i$  меньше или равно  $n$  Результаты работы программы:*
- `{ cout<<i<<"\t";` *//выводим на экран значение  $i$*   $n$       ответ
- `++i;} //увеличиваем  $i$  на единицу` 10      12345678910
- `return 0;}`

- **Замечание:** используя операцию постфиксного инкремента, тело цикла можно заменить одной командой `cout <<'++ <<"\t"`.

# Цикл с постусловием **do while**

□ В отличие от цикла *while* условие завершения цикла проверяется после выполнения тела цикла.

□ **Формат цикла *do while*:**            **do S while (B);**

*B* - выражение, истинность которого проверяется (условие завершения цикла);

*S* - тело цикла: один оператор (простой или блок).

□ Сначала выполняется оператор *S*, а затем анализируется значение выражения *B*:

- если оно истинно, то управление передается оператору *S*,

- если ложно - цикл заверш. и управление передается на оператор, следующий за условием *B*.

**Пример(*do while*): программа вывода на экран целых чисел из интервала от 1 до *n*.**

□ `#include <iostream>`

□ `using namespace std;`

□ `int main()`

□ `{int n, i=1;`

□ `cout <<"n="; cin >>n;`

□ `do//выводим на экран i, а затем увеличиваем`

*Результаты работы программы:*

□ `cout<<i++<<"\t"; //ее значения на единицу`

*n*

ответ

□ `while (i<=n); //до тех пор пока i меньше или равна n`

10

1 2 3 4 5 6 7 8 9 10

□ `return 0;}`

# Цикл с параметром **for**

- Цикл с параметром имеет следующую структуру:
- **for (<инициализация>; <выражение>; <модификации>) <оператор>;**
- **Инициализация** используется для объявления и присвоения начальных значений величинам, используемым в цикле.
- В этой части можно записать несколько операторов, разделенных запятой. Областью действия переменных, объявленных в части инициализации цикла, является цикл и вложенные блоки.
- **Выражение** определяет условие выполнения цикла:
  - *если его результат истинен, цикл выполняется.*
- **Истинность** выражения проверяется перед каждым выполнением тела цикла, таким образом, цикл с параметром реализован как цикл с предусловием.
- **Модификации** выполняются после каждой итерации цикла и служат обычно для изменения параметров цикла.
- В части модификаций можно записать несколько операторов через запятую.
- **Оператор** (простой или составной) представляет собой тело цикла.

- Любая из частей оператора *for* (инициализация, выражение, модификация, оператор) может отсутствовать, но точку с запятой, определяющую позицию пропускаемой части, надо оставить.

- `#include <iostream>`
- `using namespace std;`
- `int main()`

- `{ int n; cout <<"n="; cin >>n;`

### ***Результаты работы программы:***

- `for (int i=1; j<=n; i++)` //для *i* от 1 до *n* с шагом 1     <sup>n</sup> ответ
- `cout<<i<<"\t";`     //выводить на экран значение *i*     10 123456789 10
- `return 0;}`

- **Замечание.** Используя операцию постфиксного инкремента при выводе данных на экран, цикл *for* можно преобразовать следующим образом:

- `for (int i=1 ;j<=n;) cout<<i+ + <<"\t";`

- В этом случае в заголовке цикла *for* отсутствует блок модификации.

# Вложенные циклы

- Циклы могут быть простые или вложенные (кратные, циклы в цикле).
- Вложенными могут быть циклы любых типов: *while*, *do while*, *for*.
- Структура вложенных циклов на примере *типа for* приведена ниже:

```
for(i=1;i<k;i++)
```

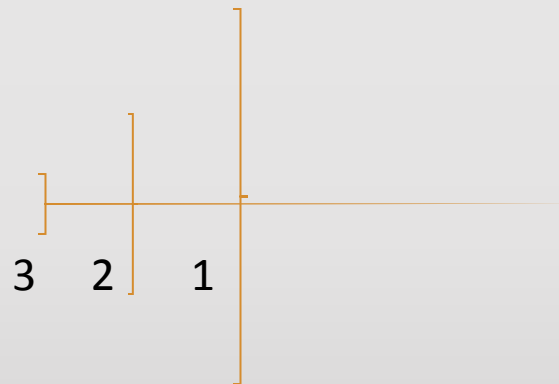
```
{...
```

```
  for (j=10; j>k;j- -)
```

```
    {...for(k=1;k<k;k;j+=2){...}
```

```
    ...}
```

```
  ...}
```



Каждый внутренний цикл должен быть полностью вложен во все внешние циклы.

--- «Пересечения» циклов не допускается.

□ Рассмотрим пример использования вложенных циклов, который позволит вывести на экран следующую таблицу:

```
□ 2  2  2  2  2      #include <iostream>
□ 2  2  2  2  2      using namespace std;
□ 2  2  2  2  2      int main()
□ 2  2  2  2  2      { for (int i=1; i<=4;++i,cout<<endl) //внешний цикл
                        for (int j= 1; j<=5; ++j) //внутренний цикл
                        cout<<"2\t"; //тело внутреннего цикла
                        return 0;}
```

□ Внешний цикл определяет количество строк, выводимых на экран. В блоке модификации данного цикла стоят два оператора.

□ Первый ++/ будет увеличивать значение *i* на единицу после каждого выполнения внутреннего цикла, а второй *-cout <<endl* будет переводить выходной поток на новую строку.

□ Внутренний цикл является телом внешнего цикла.

□ Внутренний цикл определяет, сколько чисел нужно вывести в каждой строке, а в теле внутреннего цикла выводится нужное число.



- Рассмотрим еще один пример использования вложенных циклов, который позволит вывести на экран следующую таблицу:

```
                                #include <iostream>
1   3                               using namespace std;
1   3   5                           int main()
1   3   5   7   { for (int i=1; i<=5; ++i, cout<<endl) //внешний цикл
1   3   5   7   9   for(int j=1;j<=2*i-1;j+=2) //внутренний цикл
                                cout<<j<<"\t"; //тело внутреннего цикла
                                return 0;}
```

- В данном случае таблица состоит из пяти строчек, в каждой из которых печатаются только нечетные числа.
- Последнее нечетное число в строчке зависит от ее номера.
- Эта зависимость выражается через формулу  $k = 2i - 1$  (зависимость проверить самостоятельно), где  $k$  - последнее число в строке,  $i$  - номер текущей строки. Внешний цикл следит за номером текущей строки  $i$  а внутренний цикл будет печатать нечетные числа из диапазона от 1 до  $2i - 1$ .

# задание

- Вывести на экран числа в виде следующих таблиц:

• 1.)

```
5 5 5 5 5 5
5 5 5 5 5 5
5 5 5 5 5 5
5 5 5 5 5 5
```

2.)

```
5
5 5
5 5 5
5 5 5 5
5 5 5 5 5
```

# Использование операторов цикла

Программу, которая выводит на экран квадраты всех целых чисел от А до В (А и В целые числа, при этом  $A < B$ ).

- Необходимо перебрать все целые числа из интервала от А до В.
- Эти числа представляют собой упорядоченную последовательность, в которой каждое число отличается от предыдущего на 1.

```
#include <iostream>
using namespace std;
int main()
{ int a, b;
  cout <<"a="; cin >>a;
  cout <<"b="; cin >>b;
  int i=a;
  while (i<=b)
    cout<<i*i++<<"\t";
return 0;}

#include <iostream>
using namespace std;
int main()
{ int a, b;
  cout <<"a="; cin >>a;
  cout <<"b="; cin >>b;
  int i=a;
  do cout<<i*i++<<"\t";
  while (i<=b);
return 0;}
```

- Выражение  $i*i++$  (значение которого выводится на экран в теле каждого из циклов)
- С учетом приоритета в начале выполнится операц.умножение, результат которой будет помещен в выходной поток, а затем постфиксный инкремент увеличит значение  $i$  на 1

# задание

- Написать программу, которая выводит на экран квадраты всех четных чисел из диапазона от  $A$  до  $B$  ( $A$  и  $B$  целые числа, при этом  $A < B$ ).
- (решить можно с помощью любого оператора цикла);

## Операторы безусловного перехода

- В C++ есть четыре оператора, изменяющие естественный порядок выполнения операторов:
- оператор безусловного перехода *goto*,
- оператор выхода *break*,
- оператор перехода к следующей итерации цикла *continue*,
- оператор возврата из функции *return*.

# Оператор безусловного перехода *goto*

- Оператор безусловного перехода *goto* имеет формат: `goto <метка>;`
- В теле той же функции должна присутствовать ровно одна конструкция вида:

- `<метка>: <оператор>;`

- Оператор *goto* передает управление на помеченный меткой оператор

- пример использования оператора *goto*:

- ```
#include <iostream>
using namespace std;
int main()
{float x;
metka: cout <<"x=";      //оператор, помеченный меткой
```

- ```
    cin>>x;
```

- ```
    if (x) cout<<"y="<<1/x<<endl;
```

- ```
    else { cout<<"функция не определена\n";
```

- ```
        goto metka;}      // передача управление метке
```

- ```
    return 0;}
```

- - при попытке ввести 0 на экран будет выведено сообщение «функция не определена», после чего управление будет передано оператору, помеченному меткой, и программа повторно попросит ввести значение *x*.

- *использование оператора *goto* затрудняет чтение больших по объему программ, поэтому использовать метки нужно только в крайних случаях.*

## Оператор выхода *break*

- Оператор *break* используется внутри операторов ветвления и цикла для обеспечения перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится *break*.
- Оператор *break* применяется также для выхода из оператора *switch*, аналогичным образом он может применяться для выхода из других операторов.

# Оператор перехода к следующей итерации **цикла** **continue**

- Оператор перехода к следующей итерации цикла **continue** пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации (повторение тела цикла).
- Рассмотрим оператор **continue** на примере:
- `#include <iostream>`
- `using namespace std;`
- `int main()`
- `{ for (int i=1; i<100; ++i) //перебираем все числа от 1 до 99`  
`{if (i % 2) continue; //если число нечетное, то переходим к следующей`  
`итерации`  
`cout<<i<<"\t";} //выводим число на экран`
- `return 0;}`
- В результате данной программы на экран будут выведены только четные числа из интервала от 1 до 100, т.к. для нечётных чисел текущая итерация цикла прерывалась и команда `cout<<i<<"\t"` не выполнялась.
- **Оператор возврата из функции **return**:**
- Оператор возврата из функции **return** завершает выполнение функции и передает управление в точку ее вызова.



# Массивы. Указатели.

- Когда компилятор обрабатывает оператор определения переменной,
- например, `int a =50;` , то он выделяет память в соответствии с типом `int` и записывает в нее значение 50)
- Все обращения в программе к переменной по ее имени заменяются компилятором на адрес области памяти, в которой хранится значение переменной., **такие переменные называются указателями.**
- **В с++ различают три вида указателей:**
  - **- указатели на объект,**
  - **- на функцию и на void;**
- **Указатель на объект** содержит адрес области памяти, в которой хранятся
- данные определенного типа (простого или составного).
- *Объявление указателя на объект имеет следующий вид:*  
**<базовый тип> [<модификатор>] \* <имя указателям**
- **базовый тип** — имя типа переменной, адрес которой будет содержать переменная указатель;
- **модификатор** необязателен., может иметь значение: `near`, `far` или `huge`

□ Указатель может быть переменной или константой, указывать на переменную или константу, а также быть указателем на указатель.

### □ **Например:**

□ `int i;` //целочисленная переменная

□ `const int j=10;` //целочисленная константа

□ `int *a;` //указатель на целочисленное значение

□ `int **x;` //указатель на указатель на целочисленное значение

□ `const int *b;` //указатель на целочисленную константу

□ `int * const c=&i;` //указатель-константа на целочисленную переменную

□ `const int 'const d=&j;` //указатель константа на целую переменную

□ **Указатель типа `void`** применяется в тех случаях, когда конкретный тип объекта,

□ адрес которого нужно хранить, не определен.

□ Указателю на `void` можно присвоить значение указателя любого типа, а также сравнить его с любым указателем, но перед выполнением каких-либо действий с областью памяти, на которую он ссылается, требуется преобразовать его к конкретному типу явным образом.

- **Перед использованием указателя надо выполнить его *инициализацию*, т.е. присвоение нач. значения.**
- **Существуют следующие способы инициализации указателя:**
- **1) присваивание указателю адреса существующего объекта:**
  - с помощью операции получения адреса:
    - `int a=50; //целая переменная`
    - `int *x=&a; //указателю присваивается адрес целой переменной a`
    - `int *y (&a); // указателю присваивается адрес целой переменной a`
      - с помощью значения другого инициализированного указателя
    - `int *z=x; //указателю присваивается адрес, хранящийся в x:`
      - с помощью имени массива или функции (рассмотрим позже).
- **2) присваивание указателю адреса области памяти в явном виде:**
  - `int *p=(int *) 0xB8000000;`
  - где `0xB8000000` - шестнадцатеричная константа, `(int *)` - операция явного приведения типа к типу указатель на целочисленное значение.
- **3) присваивание пустого значения:**
  - `int *x=NULL; int *y=0;`
  - где `NULL` стандартная константа, определенная как указатель равный 0
- **4) выделение участка динамической памяти и присваивание ее адреса указателю:**
  - `int *a = new int; //1`
  - `int *b = new int (50); //2`

- **4)** выделение участка динамической памяти и присваивание ее адреса указателю:
- `int *a = new int; //1`
- `int *b = new int (50); //2`
- `// 1` операция *new* выполняет выделение достаточного для размещения величины типа *int* участка динамической памяти и записывает адрес начала этого участка в переменную *a*.
- Память под переменную *a* выделяется на этапе компиляции.
- `//2`, кроме действий описанных выше, производится инициализация выделенной динамической памяти значением 50.
- Освобождение памяти, выделенной с помощью операции *new*, должно выполняться с помощью операции *delete*.
- При этом переменная-указатель сохраняется и может инициализироваться повторно.
- пример использования операции *delete*:
- `delete a; delete []b;`

# ССЫЛКИ

- Ссылка представляет собой синоним имени, указанного при инициализации ссылки.
- Ссылку можно рассматривать как указатель, который разыменовывается неявным образом.
- Формат объявления ссылки: **<базовый тип> & <имя ссылки>**

Например:

```
int a; //целочисленная переменная
```

- `int &b=a; //ссылка на целочисленную переменную a`

Следующий пример:

- `#include <iostream>`
- `using namespace std;`
- `int main()`
- `{int a=50; //целочисленная переменная a`
- `int &b=a; //ссылка b - альтернативное имя для переменной a`
- `cout <<"\t b\n";`
- `cout «a <<"\t" «b«endl;`
- `a++; //1`
- **`cout «a <<"\t" «=b«endl;`**
- `b++; //2`
- **`cout «a <<"\t" «b«endl;`**
- **`return 0;}`**

# Одномерный массив

- **Одномерный массив** - это фиксированное количество элементов одного и того же типа, объединенных общим именем, где каждый элемент имеет свой номер.
- ***Нумерация элементов массива в C++ начинается с нулевого элемента, то есть, если массив состоит из 10 элементов, то его элементы будут иметь следующие номера: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.***
- Элементы массива могут быть любого типа, в том числе и структурированного (кроме файлового).
- Между указателями и массивами существует взаимосвязь: любое действие над элементами массивов, которое достигается индексированием, может быть выполнено и с помощью указателей.
- Вариант программы с указателями будет работать быстрее.

□ Дан массив из 10 целых чисел. Написать программу, которая заменяет в данном массиве все отрицательные элементы нулями.

```
#include <iostream>
using namespace std;
int main()
{ int n; cout<<"n="; cin>>n;    //ввели количество элементов массива
  int a[n];    // объявляем статический массив размерностью n
  for (int i=0;i<n; ++i)    //ввод и обработка данных
  {cout<<"a["<<i<<"]="; cin>>a[i];    //ввод очередного элемента
    if (a[i]<0) a[i]=0;} //если i-тый элемент массива отрицат., то заменяем его на
0
  for (int i=0;i<10;++i) cout<<a[i]<<"M"; //вывод массива на экран
return 0;}
```

□ Результат работы программы: Исходные данные                      Ответ

2 -4 1 2 -2 0 23 -12 1 -1    2 0 1 2 0 0 23 0 1 0

- Дан массив из  $n$  действительных чисел ( $n < 100$ ).
- Написать программу для подсчета суммы этих чисел.

```
□ #include <iostream>
□ using namespace std;
□ int main()
□ { int n; cout<<"n="; cin>>n;
□ float a[n];
□ float s=0;
□ for (int i=0;i<n; ++i)
□ {cout<<"a["<<i<<"]="; cin>>a[i]; //ввод очередного элемента в массив
□ s+=a[i];} //добавление значения элемента массива к сумме
□ cout <<"s="<<s<<endl;
□ return 0;}
```

□ *Результат работы программы:*      $n$      Исходные данные     Ответ

□                                    5     23 0 2.5 1.7 -1.5                     $S = 5$

□ **при подсчете суммы используется прием накопления суммы  $s+=a[i]$ .**



□ Дан массив из  $n$  целых чисел ( $n < 100$ ). Написать программу для подсчета среднего арифметического четных значений данного массива.

```
□ #include <iosfream>
□ using namespace std;
□ int main()
□ { int n; cout<<"n="; cin>>n;
□ int a[n], k=0;
□ float s=0;
□ for (int i=0;i<n; ++i)
□ { cout<<"a["<<i<<"]="; cin>>a[i]; //ввод очередного элемента в массив
□ if (!(a[i]%2)) //если остаток при делении элемента на 2 равен 0
□ {s+=a[i];k++;} //то элемент четный - добавить его к сумме и увеличить
□ } // количество четных элементов на 1
□ if (k) //если k не нулевое, то четные числа в последовательности есть
□ cout <<"sr="<< s/k<<endl; //и можно вычислить их среднее арифметическое значение else cout<<"
□ четных чисел в последовательности нет "<<endl;
□ return 0;}
```

□ Результат работы	n	Исходные данные	Ответ
□ программы:	5	1 3 7-41 9	четных чисел в послед-сти нет
□	4	2 4 6 4	sr = 4.00

**Выражение  $a[i] \% 2$  будет давать 0, если  $a[i]$  четное число. В C++ 0 трактуется как ложь, поэтому**

**операторе *if* ставим операцию логического отрицания (!) перед этим выражением.**

□ Дан массив из  $n$  целых чисел ( $n < 100$ ). Написать программу, которая определяет наименьшее элемент в массиве и его порядковый номер.

```
□ #include <iostream>
```

```
□ using namespace std;
```

```
□ int main()
```

```
□ { int n; cout<<"n="; cin>>n;  
  int a[n];
```

```
□ for (int i=0;i<n; ++i) { cout<<"a["<<i<<"]="; cin>>a[i];}
```

```
□ int min=a[0]; //в качестве наименьш.значения полагаем нулевой элемент массива
```

```
□ int nmin=0; //соответственно его порядковый номер равен 0
```

```
□ for (int i=1;i<n; ++i) //перебираем все элементы массива с первого по последний
```

```
□ if (a[i]<min) //если очередной элемент окажется меньше значения min то в  
  качестве
```

```
□ {min=a[i]; //нового наименьш.значения запоминаем значение текущего  
  элемента
```

```
□ nmin=i;} //массива и, соответственно, запоминаем его номер
```

```
□ cout <<"min="<< min<<"\t nmin=" << nmin<<endl;
```

```
return 0;}
```

□ *Результат работы*      $n$      Исходные данные     Наименьшее значение     Его номер

□ программы:     5     13 7 41 0     41     4

□ Дан массив из  $n$  действительных чисел ( $n < 100$ ). Написать программу, которая меняет местами в этом массиве наибольший и наименьший элемент местами (считается, что в послед-ти только один наибольший и один наименьший элементы).

```
□ #include <iostream>
```

```
□ using namespace std;
```

```
□ int main()
```

```
□ { int n; cout<<"n="; cin>>n; ,
```

```
□ float a[n];
```

```
□ for (int i=0;i<n; ++i) {cout<<"a["<<i<<"]=", cin>>a[i];}
```

```
□ //первоначально полагаем элемент с номером 0 минимальным и максимальным
```

```
□ float min=a[0], max=a[0];
```

```
□ int nmin=0, nmax=0;
```

```
□ for (int i=1 ;i<n; ++i) //поиск наибольшего и наименьшего значения в массиве и их номеров  
□ { if (a[i]<min){min=a[i];nmin=i;}
```

```
□ if(a[i]>max){max=a[i];nmax=i;}}
```

```
□ a[nmax]=min; //в позицию наименьшего элемента записываем значение наибольшего
```

```
□ a[nmin]=max; //в позицию наибольшего элемента записываем значение наименьшего
```

```
□ for (int i=0;i<n; ++i) cout<<a[i]<<"\t"; //выводим измененный массив на экран
```

```
□ return 0;}
```

□ *Результат работы программы:*      $n$      Исходные данные     Измененные данные

# задание

- Дана последовательность целых чисел.
- 1. заменить все положительные элементы противоположными числами;
- 2. заменить все отрицательные элементы, не кратные 3, противоположными им числами.
- 3. подсчитать среднее арифметическое нечётных элементов массива ( для двумерного массива)
- 4. подсчитать сумму элементов кратных 9.
- 5. заменить все минимальные элементы на противоположные.

# Двумерные массивы

- Двумерные массивы (матрицы, таблицы) - представляют собой фиксированное количество элементов одного и того же типа, объединенных общим именем, где каждый элемент определяется номером строки и номером столбца, на пересечении которых он находится.
- Двумерные массивы находят свое применение тогда, когда исходные данные представлены в виде таблицы, или когда для хранения данных удобно использовать табличное представление.
- Нумерация строк и столбцов начинается с нулевого номера.
- Поэтому если массив содержит три строки и четыре столбца, то строки нумеруются: 0, 1, 2; а столбцы: 0, 1, 2, 3.
- В C++ двумерный массив реализуется как одномерный, каждый элемент которого также массив.

- В двумерном массиве, элементами которого являются целые числа, подсчитать среднее арифметическое четных элементов массива.

- #include <iostream>

- using namespace std;

- //Функция создает и заполняем двумерный массив

- int \*\* creat(int &n, int &m)

- {cout <<"n="; cin >>n; cout <<"m="; cin >>m;

- int \*\*mas=new int \*[n];

- for (int i=0; i<n; ++i) mas[i]=new int [m];

- for (int i=0; i<n; ++i)

- for (int j=0; j<m; ++j) {cout<<"mas["<<i<<"]["<<j<<"]="; cin.>>mas[i][j];}

- return mas;}

- int main()
- { int n,m, k=0;
- int \*\*a=creat(n,m);
- for (int i=0;i<n; ++i) *//обработка элементов массива*
- for (int j=0;j<rn; ++j)
- {if (!(a[i][j]%2))*//если элемент массива четный, то добавляем его к сумме и*
- {s+=a[i][fj; k++;}} *//увеличиваем количество четных элементов на 1*
- if (k) cout <<s/k;
- else cout<<" Четных элементов в массиве нет";
- for (int i=0;i<n; i++) delete [] a[ij; *//освобождаем память, выделенную под массив delete*
- [] a;
- return 0;}

- *Результат работы программы:*

<i>n</i>	<i>m</i>	Массив Anxm	Ответ
2	3	213	4.00
		136	
3	2	3	чётных элементов в массиве нет

□ Дан двумерный массив, элементами которого являются целые числа.

□ Найти значение максимального элемента массива.

```
□ #include <iostream>
```

```
□ using namespace std;
```

```
□ int ** creat(int &n, int &m)
```

```
□ {cout <<"n=";"cin >>n; cout <<"m="; cin >>m;
```

```
□ int **mas=new int *[n];
```

```
□ for (int i=0; i<n; ++i) mas[i]=new int [m];
```

```
□ for (int i=0; i<n; ++i)
```

```
□   for (int j=0; j<m; ++j) {cout<<"mas["<<i<<"]["<<j<<"]="; cin>>mas[i][j]}
```

```
□ return mas;}
```

□ P.s



- `int main()`
- `{int n,m;`
- `cout <<"n="; cin >>n; cout <<"m="; cin >>m; //ввели размерность массива`
- `int **a=creat(n,m);`
- `int max=a[0][0]; //первоначально качестве максимального элемента полагаем a[0][0]`
- `for (int i=0;i<n; ++i) // просматриваем все элементы массива`
- `for(intj=0;j<m; ++j)`
- `if (a[i][j]>max) //если очередной элемент больше значения максимального,`
- `max=a[i][j]; //то в качестве максимального запоминаем этот элемент`
- `cout<<"max="<<max;`
- `for (int i=0,i<n; i++) delete [] a[i] //освобождаем память, выделенную под массив`
- `delete [] a;`
- `return 0;}`
- *Результат работы программы:*     `n m` Массив  $A_{n \times m}^*$      Ответ

□ Дана квадратная матрица, элементами которой являются вещественные числа. Подсчитать сумму элементов главной диагонали.

□ Для элементов, стоящих на главной диагонали характерно то, что номер строки совпадает с номером столбца. Этот факт будем учитывать при решении задачи.

```
□ #include <iostream>
```

```
□ using namespace std;
```

```
□ float** creat(int &n)
```

```
□ {cout <<"n="; cin >>n;
```

```
□ float **mas=new int *[n];
```

```
□ for (int i=0; i<n; ++i) mas[i]=new int [n];
```

```
□ for (int i=0; i<n; ++i)
```

```
□ for (int j=0; j<n; ++j) {cout<<"mas["<<i<<"]["<<j<<"]; cin>>mas[i][j];}
```

```
□ return mas;}
```

□ P.s

- int main()
- { int n;
- float \*\*a=creat(n);
- float s=0;
- for (int i=0;i<n; i++) *//просматриваем все строки массива*
- s+=a[i][i]; *//добавляем к сумме значение элемента стоящего на главной диагонали*
- cout<<" Сумма элементов главной диагонали ="<<s;
- for (int i=0;i<n; i++) delete [] a[i]; *//освобождаем память, выделенную под массив*
- delete [] a;
- return 0;}

• *Результат работы* n Массив  $A_{n \times n}$  Ответ

- *программы:*            3        2.4 -1.9   3.1        Сумма элементов главной диагонали
- 1.1   3.6   -1.2                            =4.300
- -2.1   4.5   -1.7