Динамические массивы

Когда необходимы динамические массивы

- Если до начала работы программы неизвестно, сколько в массиве элементов, или необходимо осовбодить память, удалив массив, в программе следует использовать ***динамические массивы.***
- Память под них выделяется с помощью операции ***new*** или функции ***malloc*** в динамической области во время выполнения программы.
- *Адрес начала массива хранится в переменной, называемой указателем*.

Способы описания *одномерных* динамических массивов

int n = 10;

int *a = new int[n];

Во второй строке описан указатель на целую величину, которому присваивается адрес начала непрерывной области динамической памяти, выделенной с помощью функции ***new***.

Выделяется столько памяти, сколько необходимо для хранения ***n*** величин типа ***int***.

Величина ***n*** может быть ***переменной.***

Обнуление памяти при её выделении не происходит.

Инициализировать динамический массив с помощью списка констант нельзя.

Еще один способ описания динамического массива:

double *b = (double *)malloc(n* sizeof (double));

Способы обращения к элементам динамического массива

Обращение к элементу динамического массива осуществляется так же, как и к элементу обычного: **a[3]**.

Но можно и так: ***(a+3).**

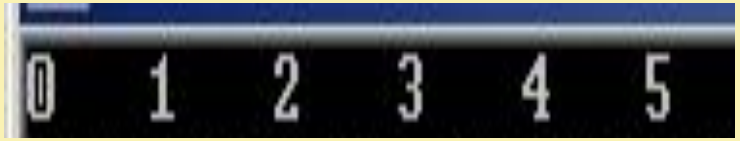
В переменной-указателе хранится адрес начла массива. Для получения адреса *третьего* элемента, к этому адресу добавляется 3.

Если динамический массив в какой-то момент работы программы становится не нужен, и мы собираемся в дальнейшем эту память использовать повторно, необходимо освободить её сл. образом:

delete [ ] a;

1. Инициализация одномерного массива

```
#include <iostream>
using namespace std;

// функция инициализации массива индексами
void f(int  *tf,  int nf)
{
    for(int   i = 0  ;   i < nf;   i ++)  // цикл по индексу эл-та массива
      tf [ i ] = i;          // эл-ту массива присваивается его индекс
    return;
}

// Главная функция
int main( )
{
    int n = 6;    // размер массива
    int *t = new int [n]; // присваивание указателю адреса массива и
                // выделение памяти под эл-ты массива
    f(t, n);  // обращение к функции инициализации массива
    for(int i=0; i<n; i++)     // цикл по индексу эл-та массива
      cout << t[i] << "   " ;   // вывод:  значения эл-та массива и двух пробело
    cout << endl;        // переход на другую строку
    return 0;}
```



Способы описания двумерных массивов

```
int n;
const int m = 5;
cin >> n;
int (*a)[m] = new int [n][m];      //1
int **b = (int **) new int [n][m]; //2
```

В операторе 1: адрес начала выделенного блока с помощью ***new*** участка памяти присваивается переменной ***a*** , определенной как указатель на массив типа ***int***. Этот тип возвращается операцией ***new.***

Скобки необходимы, т.к. без них конструкция интерпретировалась бы как массив указателей. Всего выделяется ***n*** элементов.

В операторе 2: адрес начала выделенной памяти присваивается переменной ***b***, которая описана как «указатель на указатель на ***int***», поэтому перед присваиванием требуется выполнить преобразование типа.

Интерпретация представленных операторов

Двумерный массив в C++ всегда представляется как массив, состоящий из массивов.

При выделении памяти сразу под весь массив количество строк (левую размерность) можно задать с помощью *переменной* или *выражения.*

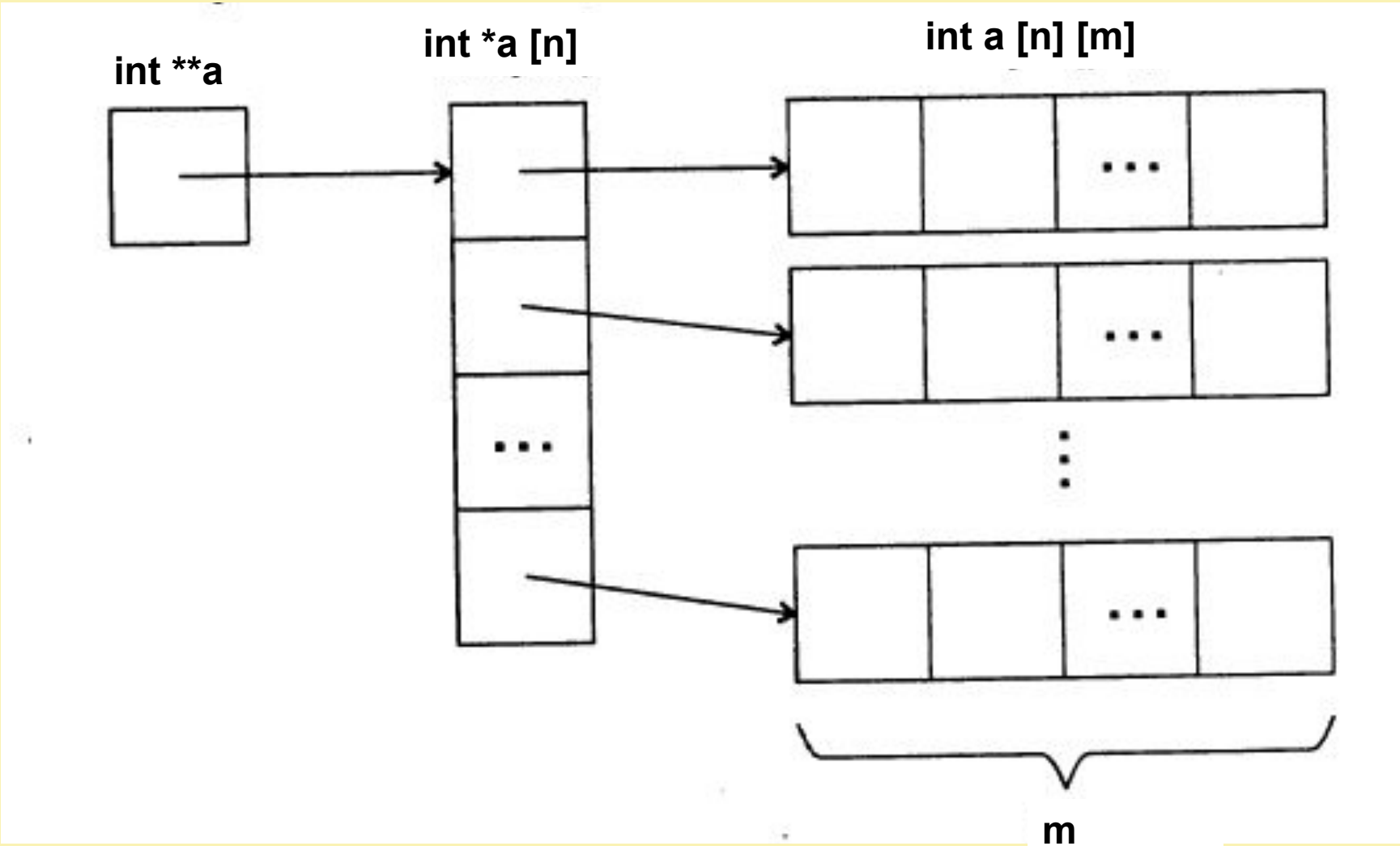
НО: количество столбцов должно быть *константным выражением,* т.е. явно определено ДО выполнения программы.

Универсальное и безопасное выделение памяти под двумерный массив

```
int n, m;
cout << “Input number of rows and columns: “;
cin>> n>>m;
int **a=new int *[n]; // выделяется память под массив указателей на строки массива (n).
for (int i = 0;  I < n;  i++) //цикл для выделения памяти под каждую строку массива.
    a[ i ] = new int [m]; //каждому элементу массива указателей на строки присваивается адрес начала участка памяти, выделенного под строку двумерного массива.
```

Каждая строка состоит из *m* элементов типа *int.*

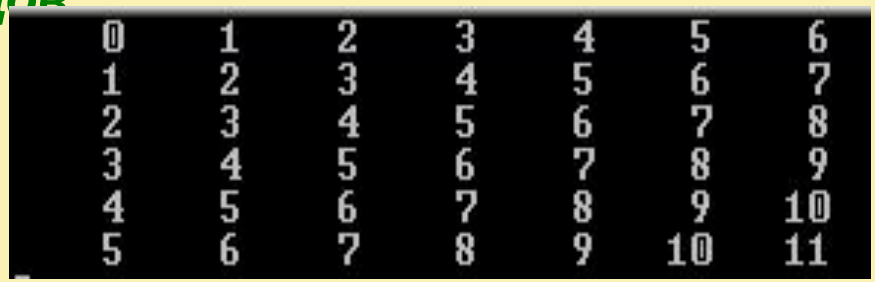
Двумерный массив – это «массив строк, каждая из которых - тоже массив»



2. Инициализация двумерного массива

```
#include <iostream>
#include <iomanip>
using namespace std;

// функция инициализации массива суммой
    индексов
void f(int  **tf,  int nf,  int mf)
{
    for(int i=0; i<nf; i++)      // цикл по строкам
     for(int j=0; j<mf; j++)       // цикл по столбцам
     tf[i][j]=i+j;  // эл-ту массива присваивается сумма
    индексов
    return; // возврат обратно в место вызова
}
```

```
int main( )        // главная функция
{
    int n = 6;     // кол-во строк
    int m = 7;     // кол-во столбцов

    int **t = new int *[n];

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

    f(t, n, m);  // обращение к функции инициализации массива

    for(int i=0; i<n; i++) // цикл по строкам
    {
        for(int j=0; j<m; j++)  // цикл по столбцам
     cout << setw(5) << t[i][j];    // вывод значения эл-та массива
         cout << endl; // переход на новую строку
    }
    return 0;  // выход из функции
}
```



3. Задача (двумерный массив)

Дан **двумерный** массив, содержащий 3 строки и 4 столбца.

Элементами массива являются **целые** числа.

Элементы каждого из **столбцов** увеличить на найденное
максимальное значение этого **столбца**.

Результат получить **в другом массиве**.

```
#include <iostream>
#include <iomanip>
#include <ctime>
using namespace std;

// инициализация массива случайными числами
void f_rand(int  **tf, int nf,  int mf, int r_min,  int  r_max)
{
    srand(  (unsigned  int)  time(NULL)); // рандомизация генератора

    for(int i=0; i<nf; i++)  // цикл по строкам
     for(int j=0; j<mf; j++)    // цикл по столбцам
     tf[i][j]=rand( ) % (r_max - r_min)  + r_min;  // инициализация эл-та массива
    return; // возврат в место вызова функции
}
```

```
// вывод массива на экран

void f_print(int  **tf, int nf,  int mf)
{
    cout << endl;     // переход на новую строку
    for(int i=0; i<nf; i++)  // цикл по строкам
    {
       for(int j=0; j<mf; j++)  // цикл по столбцам
           cout << setw(6) << tf[i][j] ; // вывод эл-та массива

       cout << endl; //  переход на новую строку
    }
return;
}
```

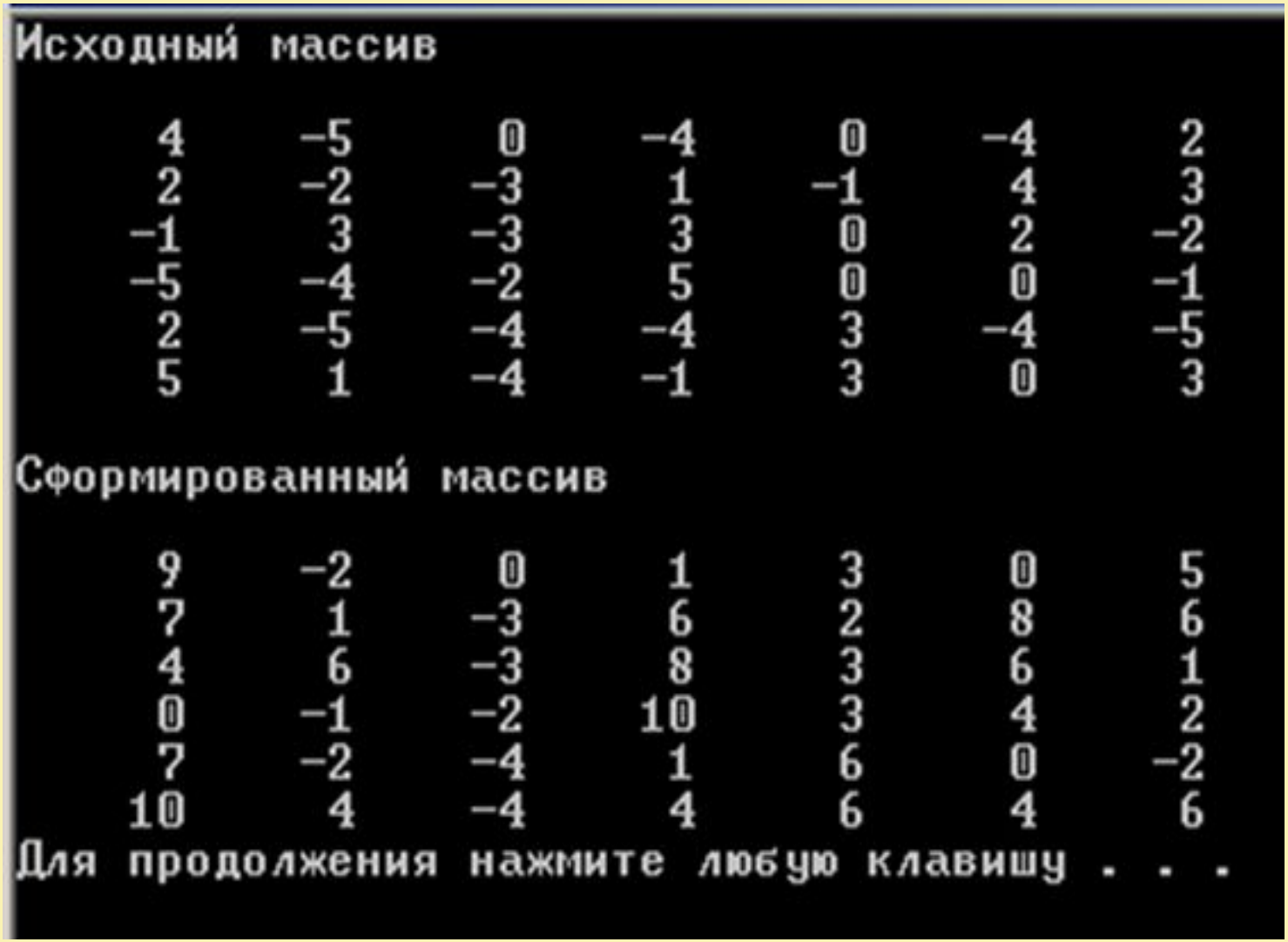
Элементы каждого из столбцов увеличить на найденное максимальное значение этого столбца.

Результат получить в другом массиве.

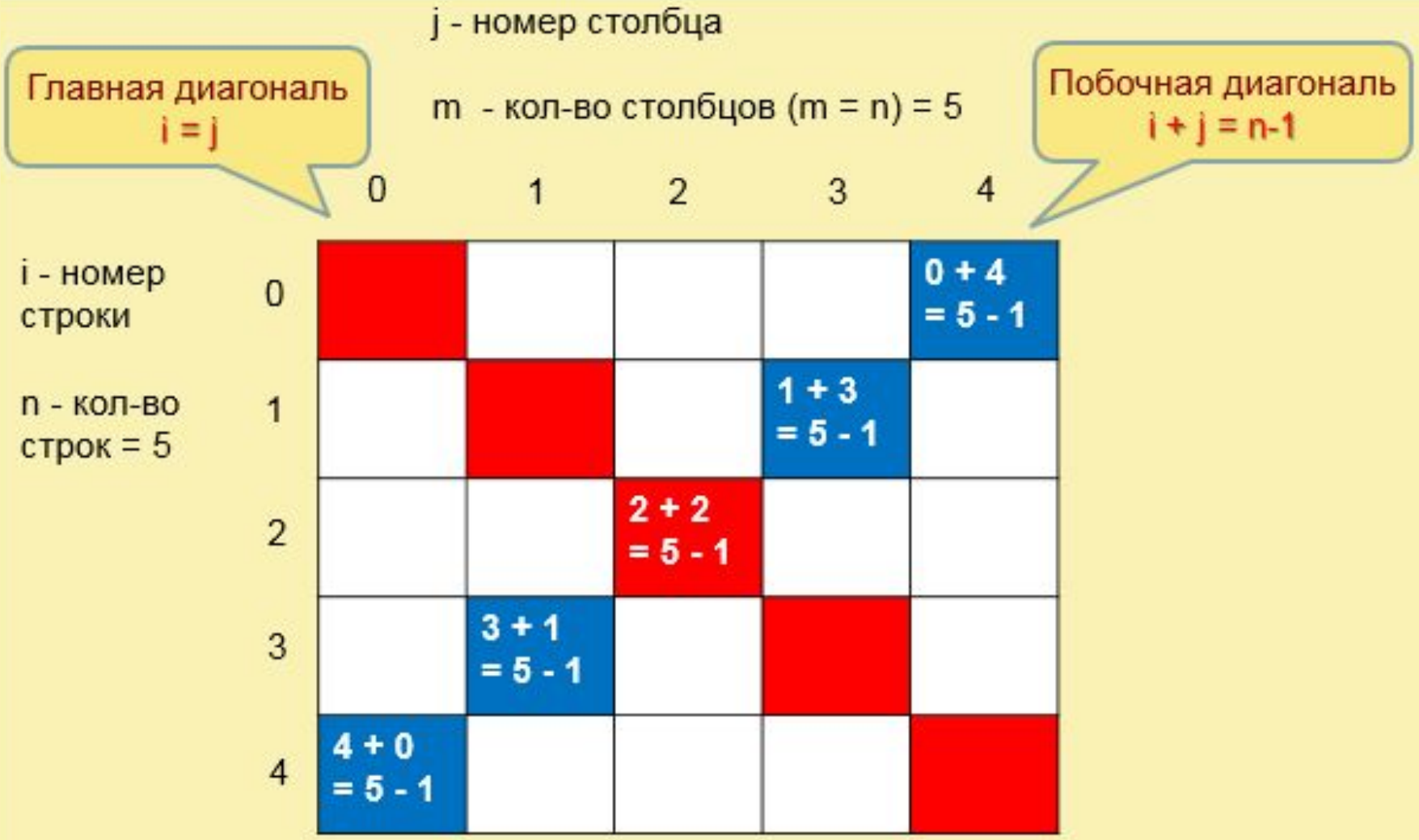
```
// функция решения задачи
void f_task(int  **t1f, int  **t2f, int nf,  int mf)
{
    for(int j=0; j<mf; j++)  // цикл по СТОЛБЦАМ
    {
     int im = 0;   // номер максимального эл-та в столбце
     for(int i=1; i<nf; i++)     // цикл по строке
      {
           if(t1f[i][j] > t1f[im][j]) // если найден больший эл-т
          im = i; // запоминаем этот номер
             }
     for(int i=0; i<nf; i++)  // цикл по строке
           t2f[i][j] = t1f[i][j] + t1f[im][j];  // задание значений другого
    массива
     }
    return;  // возврат обратно в место вызова
}
```

```
int main( )  // главная функция
{
    setlocale(0,"");   // поддержка кириллицы
    int n = 6; // кол-во строк в массиве
    int m = 7;  // кол-во столбцов в массиве
    int **t1 = new int *[n];  // указатель на массив указателей на int
    for(int i = 0; i < n; i++)    // цикл по строкам
      t1[i] = new int [m]; // каждому эл-ту массива указателей на строки
// присваивается адрес начала участка памяти,
// выделенного под строку двумерного массива
//и выделяется память под каждую строку массива
    int **t2 = new int *[n];  // указатель на массив указателей на int
    for(int i = 0; i < n; i++)    // цикл по строкам
      t2[i] = new int [m];  // присваивание адреса и выделение памяти
```

```
    // обращение к функции инициализации массива
    f_rand(t1, n, m, -5, 6);


    cout << "Исходный массив"<< endl;
    // обращение к функции вывода массива
    f_print(t1, n, m);


    // обращение к функции решения поставленной задачи
    f_task(t1, t2, n, m);


    cout << endl << "Сформированный массив"<< endl;
    // обращение к функции вывода массива
    f_print(t2, n, m);


    return 0;  // выход из главной функции
}
```


Исходный массив
4 -5 0 -4 0 -4 2
2 -2 -3 1 -1 4 3
-1 3 -3 3 0 2 -2
-5 -4 -2 5 0 0 -1
2 -5 -4 -4 3 -4 -5
5 1 -4 -1 3 0 3
Сформированный массив
9 -2 0 1 3 0 5
7 1 -3 6 2 8 6
4 6 -3 8 3 6 1
0 -1 -2 10 3 4 2
7 -2 -4 1 6 0 -2
10 4 -4 4 6 4 6
Для продолжения нажмите любую клавишу . . .

Примеры обработки двумерных матриц



```
// функция инициализации массива случайными числами
void initR(int  **tf,  int nf,  int mf, int  r_min,  int  r_max)
{
    srand((unsigned int) time(NULL));

    for(int i=0; i<nf; i++)// цикл по строкам
     for(int j=0; j<mf; j++)        // цикл по столбцам
     tf[i][j] = rand() % (r_max - r_min) + r_min;  // эл-ту массива присваивается
                                    //                случайное число
    return;   // возврат обратно в место вызова
}
```

// функция вывода массива

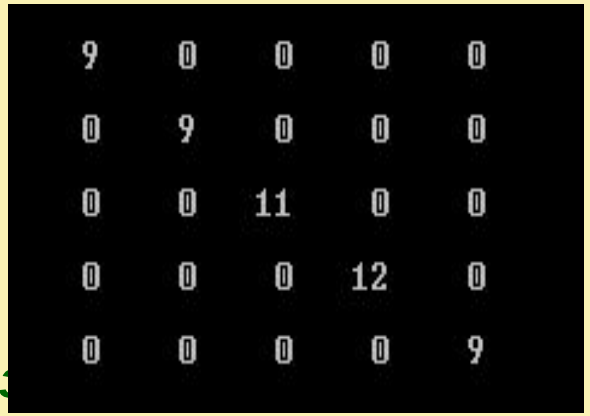
```
void print(int  **tf,   int  nf,   int  mf)
{
    cout << endl;
    for(int i=0; i<nf; i++)
    {
       for(int j=0; j<mf; j++)
     cout << setw(5) << tf[i][j];
      cout << endl << endl;
    }
    return;   // возврат обратно в место вызова
}
```

```
    9    6    8    7   14
    7    9   15   12    8
    9   13   11    6   14
    9   14   11   12    8
   15   13   12   15    9
```

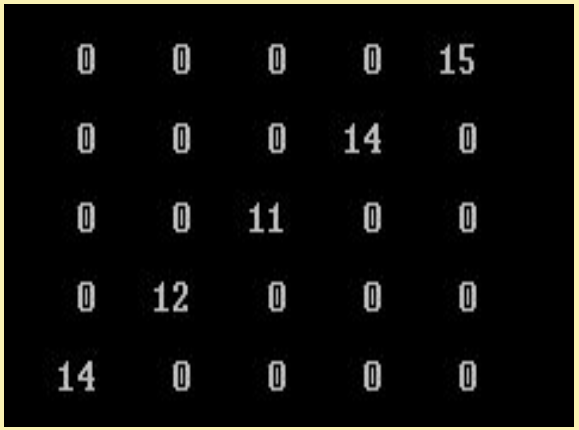
// функция инициализации массива нулями

```
void init0(int  **tf,  int nf,  int mf)
{
   for(int i=0; i<nf; i++)     // цикл по строкам
       for(int j=0; j<mf; j++)     // цикл по столбцам
    tf[i][j] = 0;    // эл-ту массива присваивается нуль
   return;  // возврат обратно в место вызова
}
```

```
// функция главная диагональ
void glavn(int  **tf1, int  **tf2,  int nf,  int mf)
{
    for(int i=0; i<nf; i++)
      tf2[i][i] = tf1[i][i];
    return;   // возврат обратно в место вы
}
```




```
// функция побочная диагональ
void poboch(int  **tf1, int  **tf2,  int nf,  int mf)
{
    init0(tf2, nf, mf);
    for(int i=0; i<nf; i++)
       tf2[i][nf-i-1] = tf1[i][nf-i-1];
    return;   // возврат обратно в место вызова
}
```



```
// функция над побочной диагональю
void nadpoboch(int  **tf1, int  **tf2,  int nf,  int mf)
{
    init0(tf2, nf, mf);  // обращение к функции инициализации
    массива нулями
    for(int i=0; i<nf; i++)
      for(int j=0; j<mf-i-1; j++)
      tf2[i][j] = tf1[i][j];
    return;   // возврат обратно в место вызова
}
```

```
9    7    9    9    0
6    9   13    0    0
8   15    0    0    0
7    0    0    0    0
0    0    0    0    0
```

```
// функция сумма эл-тов главной и произв. побочной
void SumProizv(int  **tf1,  int nf,  int mf, int &sf, int &pf)
{
     sf = 0; pf = 1;
     for(int i=0; i<nf; i++)
       for(int j=0; j<mf; j++)
       {
          if(i == j) sf += tf1[i][j];
          if(i + j == nf - 1) pf *= tf1[i][j];
       }
     return; // возврат обратно в место вызова
}
```

```
 9   6   8   7  14
 7   9  15  12   8
 9  13  11   6  14
 9  14  11  12   8
15  13  12  15   9
```

```
s = 50   p = 388080
```

```
// функция транспонир
void transp(int  **tf1,  int nf,  int mf)
{
    int c;
    for(int i=0; i<nf; i++)
      for(int j=i+1; j<mf; j++)
      {
        c = tf1[i][j];
        tf1[i][j] = tf1[j][i];
        tf1[j][i] = c;
      }
    return;   // возврат обратно в место вызова
}
```

9	6	8	7	14
7	9	15	12	8
9	13	11	6	14
9	14	11	12	8
15	13	12	15	9

9	7	9	9	15
6	9	13	14	13
8	15	11	11	12
7	12	6	12	15
14	8	14	8	9

```
int main()
{
    int n = 5;
    int m = 5;

    int **t1 = new int *[n];  // указатель на массив указателей на int

    for(int i = 0; i < n; i++)
      t1[i] = new int [m];

    int **t2 = new int *[n];  // указатель на массив указателей на int

    for(int i = 0; i < n; i++)
      t2[i] = new int [m];
```

```
initR(t1, n, m, 5, 16);  // инициализация массива t1 случайными числами
print(t1, n, m);     //  вывод массива

 init0(t2,n, m);  // инициализация массива t2 нулями
 glavn(t1, t2, n, m); // выборка эл-тов главн. диагонали t1
print(t2, n, m); //  вывод массива t2

init0(t2, nf, mf);       // инициализация массива t2 нулями
poboch(t1, t2, n, m);      // выборка эл-тов побочной диагонали t1
print(t2, n, m);  //  вывод массива t2

init0(t2, nf, mf);       // инициализация массива t2 нулями
nadpoboch(t1, t2, n, m); // выборка эл-тов над побочной диагональю t1
print(t2, n, m);  //  вывод массива t2

int s; // описание переменной – сумма эл-тов главной
int p; // описание переменной – произведение эл-тов побочной
SumProizv(t1, n, m, s, p); // определение суммы и произведения
cout << endl << "    s = " << s << "   p = " << p  << endl;  // вывод суммы и
произв.

print(t1, n, m);  //  вывод массива до транспонирования
transp(t1, n, m);       // транспонирование матрицы
print(t1, n, m);  //  вывод массива после транспонирования
cout << endl;  // перевод строки
return 0;}
```