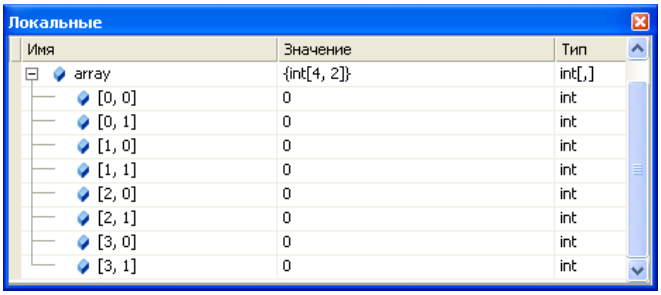
Работа с матрицами

Матрица – это двухмерный массив, который можно представить себе как совокупность строк (или совокупность столбцов). Положение элемента в массиве определяется двумя индексами: номером строки и номером столбца. Нумерация, как и для одномерных массивов, начинается с нуля. Объявление двухмерного массива выполняется аналогично объявлению одномерных массивов. Так, следующее объявление создает двухмерный массив (матрицу) целых чисел из четырех строк и двух столбцов. Элементам массива при этом автоматически присваивается значение ноль:

```
int[,] array = new int[4, 2];
```

Отображение значений и типа элементов массива в окне «Локальные» после выполнения кода:

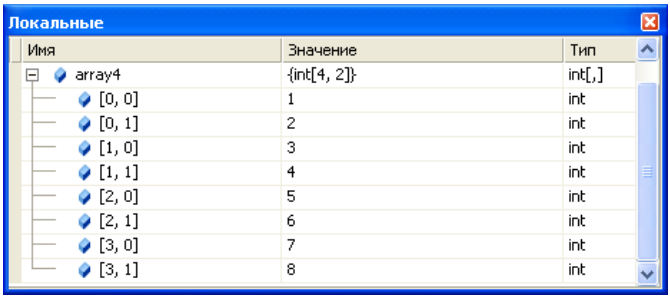


Для массива из четырех строк (как в данном примере) строки нумеруются от 0 до 3. Аналогично для столбцов.

Массив можно инициализировать при объявлении, например,

```
int[,] array4 = { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

Отображение значений и типа элементов массива в окне «Локальные» после выполнения кода:



Доступ к элементу массива осуществляется указанием двух индексов. Например,

```
array4[2, 1] = 25;
```

В результате в строку с индексом 2 и в столбец с индексом 1 будет присвоено значение 25.

Каждая строка (и каждый столбец) матрицы представляет собой одномерный массив. Поэтому при обработке матриц в основном используются типовые алгоритмы обработки одномерных массивов.

При работе с матрицами, как правило, используются вложенные циклы, например в цикле по строкам необходимо получить доступ к каждому элементу строки, т.е. организовать цикл по столбцам.

Ввод и вывод матриц

Ввод матриц можно осуществлять поэлементно с использованием вложенных циклов. Элементы матрицы вводятся, как правило, по строкам. После ввода каждого элемента необходимо нажать клавишу [Enter]. *Вывод матриц* должен осуществляться в наглядной форме, т.е. каждая строка матрицы должна выводиться в новую строку экрана с использованием подходящего формата:

```
   // 3 строки и 5 столбцов
            int[,] matr = new int[3, 5];
            int rows = matr.GetLength(0);
            int cols = matr.GetLength(1);

            Console.WriteLine(rows + "x" + cols);
            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < cols; j++)
                {
                    matr[i, j] =
int.Parse(Console.ReadLine());
                }
            }

            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < cols; j++)
                {
                    Console.Write(matr[i, j]+" ");
                }
                Console.WriteLine();

                                    }
```

Найти минимум в матрице, вывести его, номер строки и номер столбца

```
int iMin = 0, jMin = 0;


            for (int i = 0; i < rows; i++)
            {
                for (int j = 0; j < cols; j++)
                {
                    if (matr[i, j] < matr[iMin, jMin])
                    {
                        iMin = i;
                        jMin = j;
                    }
                }

            }
            Console.WriteLine(matr[iMin,  jMin]+"  "  +
iMin + " " + jMin);
```

Вывести на экран строку, сумма элементов которой максимальна.

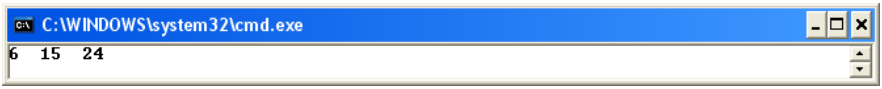
```
int maxSum=0, maxRow = -1, sum;
for (int i = 0; i < rows; i++)
{
    sum = 0;
    for (int j = 0; j < cols; j++)
    {
        sum += matr[i, j];
    }

    if (maxRow == -1 || sum > maxSum)
    {
        maxSum = sum;
        maxRow = i;
    }
}

for (int j = 0; j < cols; j++)
{
    Console.Write(matr[maxRow, j] + " ");
}
```

Просуммировать элементы строк матрицы a. Результат получить в виде вектора (одномерного массива) *b*:

```
int[,] a = new int[3, 3] { { 1, 2, 3 }, { 4, 5, 6 }, {
7, 8, 9 } };
int[] b = new int[3];
for (int i = 0; i < 3; i++)
{
    int s = 0;
    for (int j = 0; j < 3; j++) {
        s += a[i, j];
    }
    b[i] = s;
}
for (int i = 0; i < 3; i++) {
    Console.Write(b[i]+" ");
}
Console.WriteLine();
Console.ReadKey();
```



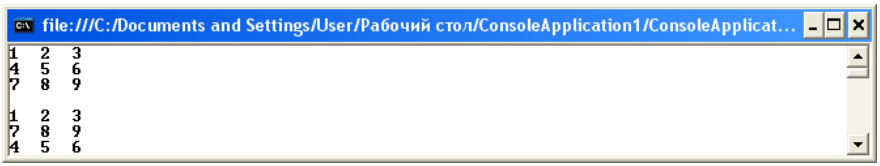
Аналогично осуществляется формирование одномерных массивов из значений максимальных (минимальных) элементов строк, индексов максимальных (минимальных) элементов строк и т.п. (см. соответствующие алгоритмы для одномерных массивов).

Если необходимо осуществить обработку столбцов матрицы, то внешний цикл необходимо организовать по номеру столбца, а внутренний – по номеру строки. В остальном алгоритмы аналогичны.

Поменять местами ii-ю и jj-ю строки матрицы.

Поменять местами строки означает поменять местами каждую пару соответствующих элементов этих строк, т.е. необходим цикл по столбцам:

```
int[,] a = new int[3, 3] { { 1, 2, 3 }, { 4, 5, 6 }, {
7, 8, 9 } };
int p = 0;
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
        Console.Write("{0:d}  ", a[i,j]);
    Console.WriteLine();
}
int ii = 1, jj = 2;
for (int k = 0; k < 3; k++)
{
    p = a[ii, k]; a[ii, k] = a[jj, k]; a[jj, k] = p;
}
Console.WriteLine();
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
        Console.Write("{0:d}  ", a[i, j]);
    Console.WriteLine();
}
Console.ReadKey();
```

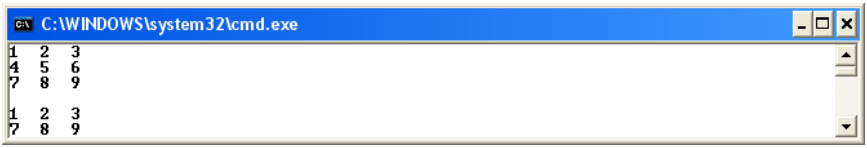


Столбцы меняются местами аналогично (цикл организуется по строкам).

Удалить k-ю строку матрицы.

Чтобы удалить строку, необходимо переместить все строки, расположенные после k-й, на одну позицию вверх. Для перемещения строки нужно организовать цикл по элементам строки, т.е. по столбцам:

```
int[,] a = new int[3, 3] { { 1, 2, 3 }, { 4, 5, 6 }, {
7, 8, 9 } };
int n = 3, m = 3;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
        Console.Write("{0:d}  ", a[i, j]);
    Console.WriteLine();
}
n = n - 1;
int k = 1;
for (int i = k; k < n; k++)
    for (int j = 0; j < m; j++)
        a[i, j] = a[i + 1, j];
Console.WriteLine();
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
        Console.Write("{0:d}  ", a[i, j]);
    Console.WriteLine();
}
Console.ReadKey();
```



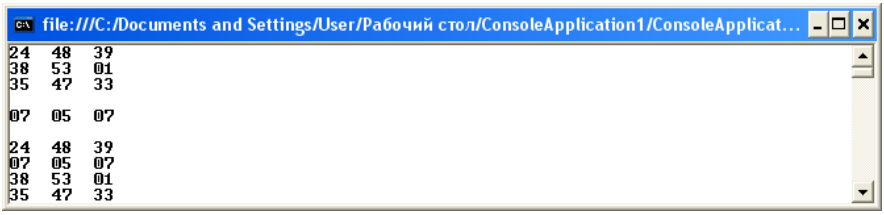
Удаление столбца осуществляется аналогично (внутренний цикл по элементам столбца, т.е. по строкам).

Вставить новую строку, заданную вектором b[m], после k-й строки матрицы.

Вначале нужно освободить место для новой строки, переместив строки, расположенные после *k*-й, на одну позицию вниз. (При объявлении матрицы предусмотреть необходимость соответствующего увеличения ее размера.)

Перемещение строк нужно начинать с последней строки (аналогичный алгоритм для одномерных массивов в см. п. 3.1):

```
int[,] a = new int[4, 3];
int[] b = new int[3] { 7, 5, 7 };
int n = 3, m = 3;
Random r = new Random();
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        a[i, j] = r.Next(55);
        Console.Write("{0:d2}  ", a[i, j]);
    }
    Console.WriteLine();
}
Console.WriteLine();
for (int i = 0; i < m; i++)
{
    Console.Write("{0:d2}  ", b[i]);
}
Console.WriteLine();
Console.WriteLine();
int k = 1;
for (int i = n - 1; i >= k; i--)
    for (int j = 0; j < m; j++)
        a[i + 1, j] = a[i, j];
for (int j = 0; j < m; j++)
    a[k, j] = b[j];
for (int i = 0; i < n + 1; i++)
{
    for (int j = 0; j < m; j++)
        Console.Write("{0:d2}  ", a[i, j]);
    Console.WriteLine();
}
Console.ReadKey();
```



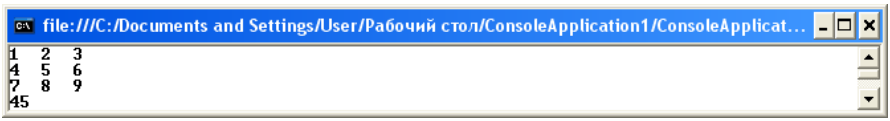
З а м е ч а н и е . Здесь переменная `r` – экземпляр класса `Random`, который представляет генератор псевдослучайных чисел. Метод `Next(maxValue)` создает случайное число в диапазоне значений от нуля до числа `maxValue`, указанного в качестве аргумента метода. `r.Next(55)` – метод `Next`, определенный в классе `Random` и примененный к экземпляру класса `r` – генерирует следующее псевдослучайное число.

Вставка столбца осуществляется аналогично.

Найти сумму элементов матрицы.

Здесь нужно обратиться к каждому элементу матрицы и добавить его к сумме:

```
int[,] a = new int[3, 3] { { 1, 2, 3 }, { 4, 5, 6 }, {
7, 8, 9 } };
int n = 3, m = 3;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
        Console.Write("{0:d}  ", a[i, j]);
    Console.WriteLine();
}
int s = 0;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
        s += a[i, j];
}
Console.WriteLine(s);
```



Аналогично осуществляется поиск максимального элемента во всей матрице.

Далее будут рассмотрены типовые алгоритмы для квадратной матрицы *a* размера *n*×*n*.

Найти сумму элементов, расположенных на главной диагонали (след матрицы).

Элементы, расположенные на главной диагонали, имеют одинаковые индексы (номера строк и столбцов совпадают) и представляют, таким образом, одномерный массив:

```
int s = 0;
for (int i = 0; i < n; i++)
{
 s += a[i, i];
}
```

Аналогично можно организовывать и другие алгоритмы для работы с диагональными элементами (нахождение максимального элемента и т.п.).

7. *Найти сумму элементов, расположенных ниже главной диагонали (включая диагональ), т.е. просуммировать элементы нижнего треугольника матрицы.*

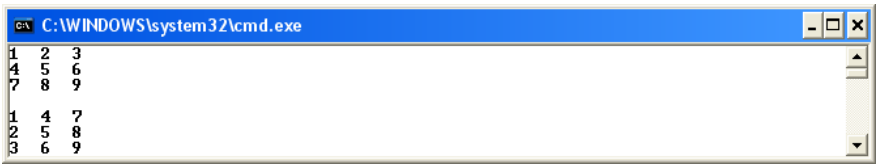
Здесь во внешнем цикле (по строкам) номера строк изменяются от 0 до n –1, но в каждой i-й строке суммируются только элементы, расположенные до диагонального элемента этой строки, т.е. до i-го:

```
int s = 0;
for (int i = 0; i < n; i++)
{
 for (int j = 0; j <= i; j++)
 s += a[i, j];
}
```

Транспонирование матрицы с получением результата в том же массиве.

Для квадратной матрицы размера $n \times n$ требуется переставлять элементы, расположенные симметрично относительно главной диагонали:

```
int p = 0;
for (int i = 0; i < n - 1; i++)
{
    for (int j = i + 1; j < n; j++)
   {
      p = a[i, j]; a[i, j] = a[j, i]; a[j, i] = p;
   }
}
```



Для прямоугольной матрицы размера $n \times m$ транспонированная матрица может быть получена на месте исходной, если последняя размещена в массиве размера не менее чем $n \times n$ (предполагается, что $n > m$). При этом можно использовать приведенный выше алгоритм. Фиктивные столбцы, дополняющие исходную матрицу до квадратной, помещаются в этом случае в фиктивные строки транспонированной матрицы.

Является ли матрица симметричной

Самостоятельно