Перечисления и массивы

Перечислимый тип данных

- *Перечисление* — отдельный тип-значение, содержащий совокупность именованных констант.
- Пример:

```
enum Color : long
{
    Red,
    Green,
    Blue
}
```

*Базовый класс - **System.Enum.***

*Перечисление может иметь **модификатор** (new, public, protected, internal, private). Он имеет такое же значение, как и при объявлении классов.*

- Каждый элемент перечисления имеет связанное с ним константное значение, тип которого определяется базовым типом перечисления.
- Базовые типы: byte, sbyte, short, ushort, int, uint, long и ulong. По умолчанию – int.

Значения элементов перечисления

- Значение элемента задается либо явно, либо неявно, а именно:
  - Первый элемент автоматически принимает значение 0.
  - Последующие элементы принимают значение предыдущего + 1.

```
enum Button {Start, Stop, Play, Next, Prev };     // неявно
enum Color
{   Red,                  // 0  неявно
    Green = 10,           // 10 явно
    Blue                  // 11 неявно
}
enum Nums { two = 2, three, ten = 10, eleven, fifty = ten + 40 };
```

- Несколько элементов перечисления **могут** иметь одно и то же значение.
- Элементы одного перечисления **не могут** иметь одинаковые имена.

Действия с элементами перечислений

- арифметические операции (+, –, ++, —)
- логические поразрядные операции (^, &, |, ~)
- сравнение с помощью операций отношения (<, <=, >, >=, ==, !=)
- получение размера в байтах (sizeof)
- вывод на консоль

```
enum Menu { Read, Write, Edit, Quit };
Menu m, n;
…
m = Menu.Read; n = m; n++;
if (n > m ) Console.WriteLine(n);
```

Каждое перечисление определяет отдельный тип; для преобразования между перечислением и целым типом или между двумя перечислениями требуется явное приведение типа.

Пример

```
enum Color
{  Red = 0x000000FF, Green = 0x0000FF00, Blue = 0x00FF0000 }
class Test
 {  static void Main()
        {  Console.WriteLine(StringFromColor(Color.Green)); }

    static string StringFromColor(Color c)
        {  switch (c)
           {  case Color.Red:
                  return String.Format("Red = {0:X}", (int)c);
              case Color.Green:
                  return String.Format("Green = {0:X}", (int)c);
              case Color.Blue:
                  return String.Format("Blue = {0:X}", (int)c);
              default:
                  return "Invalid color";
           } } }
```

Массивы

- *Массив* — ограниченная совокупность однотипных величин
- Элементы массива имеют одно и то же имя, а различаются по порядковому номеру (*индексу*)
- **Виды** массивов в C#:
  - одномерные
  - многомерные (например, двумерные, или прямоугольные)
  - массивы массивов (др. термины: невыровненные, ступенчатые).

Создание массива

- **Массив относится к ссылочным типам данных** (располагается в хипе), поэтому *создание массива* начинается с выделения памяти под его элементы.
- *Элементами массива* могут быть величины как значимых, так и ссылочных типов (в том числе массивы), например:

```
int[]    w = new int[10];        // массив из 10 целых чисел
string[] z = new string[100];    // массив из 100 строк
```

- Массив значимых типов хранит значения, массив ссылочных типов — ссылки на элементы.
- Всем элементам при создании массива присваиваются *значения по умолчанию*: нули для значимых типов и null для ссылочных.

Размещение массивов в памяти

Размерность массива

- Количество элементов в массиве (*размерность*) задается при выделении памяти и **не может** быть изменена впоследствии. Она может задаваться выражением:

```
short n = …;
string[] z = new string[2*n + 1];
```

- Размерность не является частью типа массива.
- Элементы массива нумеруются *с нуля*.

Для *обращения к элементу массива* после имени массива указывается номер элемента в квадратных скобках, например:

```
w[4]      z[i]
```

- С элементом массива можно делать все, что допустимо для переменных того же типа.
- При работе с массивом автоматически выполняется *контроль выхода за его границы*: если значение индекса выходит за границы массива, генерируется исключение IndexOutOfRangeException.

Действия с массивами

- Массивы одного типа можно *присваивать* друг другу. При этом происходит присваивание ссылок, а не элементов:

```
int[] a = new int[10];
int[] b = a;        // b и a указывают на один и тот же массив
```

- Все массивы в C# имеют общий базовый класс Array, определенный в пространстве имен System. Некоторые элементы класса Array:
  - **Length** (Свойство) - Количество элементов массива (по всем размерностям)
  - **BinarySearch** (Статический метод) - Двоичный поиск в отсортированном массиве
  - **IndexOf** – (Статический метод) - Поиск первого вхождения элемента в одномерный массив
  - **Sort** (Статический метод) - Упорядочивание элементов одномерного массива

Одномерные массивы

- Варианты описания массива:

тип[] имя;

тип[] имя = new тип [размерность];

тип[] имя = { список_инициализаторов };

тип[] имя = new тип [] { список_инициализаторов };

тип[] имя = new тип [размерность] { список_инициализаторов };

- Примеры описаний (один пример на каждый вариант описания, соответственно):

```
int[] a;                                    //  элементов нет
int[] b = new int[4];                       //  элементы равны 0
int[] c = { 61, 2, 5, -9 };                 //  new подразумевается
int[] d = new int[] { 61, 2, 5, -9 };       // размерность вычисляется
int[] e = new int[4] { 61, 2, 5, -9 };      // избыточное описание
```

Пример (не лучший способ)

Для массива, состоящего из 6 целочисленных элементов, программа определяет:

- сумму и количество отрицательных элементов;
- максимальный элемент.

Программа

```
const int n = 6;
int[] a = new int[n] { 3, 12, 5, -9, 8, -4 };

Console.WriteLine( "Исходный массив:" );
for ( int i = 0; i < n; ++i ) Console.Write( "\t" + a[i] );
Console.WriteLine();

long sum_otr = 0;              // сумма отрицательных элементов
int  num_otr = 0;              // количество отрицательных элементов
for ( int i = 0; i < n; ++i )
      if ( a[i] < 0 ) {
         sum_otr += a[i];  ++num_otr;
      }
 Console.WriteLine( "Сумма  отрицательных = " + sum_otr );
 Console.WriteLine( "Кол-во отрицательных = " + num_otr );

 int max = a[0];               // максимальный элемент
 for ( int i = 0; i < n; ++i )
         if ( a[i] > max ) max = a[i];
 Console.WriteLine( "Максимальный элемент = " + max );
```

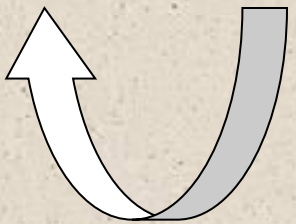
Оператор foreach (упрощенно)

- Применяется для перебора элементов массива. Синтаксис:

foreach (тип *имя* **in** имя_массива **) тело_цикла**

- *имя* задает локальную по отношению к циклу переменную, которая будет по очереди принимать все значения из массива, например:

```
int[] massiv =  { 24, 50, 18, 3, 16, -7, 9, -1 };
foreach ( int x in massiv ) Console.WriteLine( x );
```



Программа с использованием foreach

```
int[] a = { 3, 12, 5, -9, 8, -4 };
Console.WriteLine( "Исходный массив:" );
foreach ( int elem in a )
      Console.Write( "\t" + elem );
Console.WriteLine();

long sum_otr = 0;       // сумма отрицательных элементов
int  num_otr = 0;        // количество отрицательных элементов
foreach ( int elem in a )
       if ( elem < 0 ) {
            sum_otr += elem;  ++num_otr;
       }
Console.WriteLine( "sum = " + sum_otr );
Console.WriteLine( "num = " + num_otr );

int max = a[0];           // максимальный элемент
foreach ( int elem in a )
      if ( elem > max ) max = elem;
Console.WriteLine( "max = " + max );
```

```
for ( int i = 0; i < n; ++i )
    if ( a[i] < 0 ) {
        sum_otr += a[i];  ++num_otr;
    }
```

Программа в true style ☺

```
class Mas_1                    // класс для работы с 1-мерным массивом
{
    int[] a = { 3, 12, 5, -9, 8, -4 };          // для простоты слайда

    public void PrintMas()                              // вывод массива
    {  Console.Write("Массив: ");
       foreach (int elem in a) Console.Write("  " + elem);
       Console.WriteLine();
    }

    public long SumOtr()        // сумма отрицательных элементов
    {
       long sum_otr = 0;
       foreach (int elem in a)
           if (elem < 0) sum_otr += elem;
       return sum_otr;
    }
```

```
public int NumOtr()                    // кол-во отрицательных элементов
    {
        int num_otr = 0;
        foreach (int elem in a)
            if (elem < 0) ++num_otr;
        return num_otr;
    }

public int MaxElem()                   // максимальный элемент
    {
        int max = a[0];
        foreach (int elem in a) if (elem > max) max = elem;
        return max;
    }
}
```

```
class Program                                   // класс-клиент
{   static void Main(string[] args)
    {
      Mas_1 mas = new Mas_1();
      mas.PrintMas();

      long sum_otr = mas.SumOtr();
      if (sum_otr != 0) Console.WriteLine("Сумма отриц. = " + sum_otr);
      else Console.WriteLine("Отриц-х эл-тов нет");

     int num_otr = mas.NumOtr();
     if (num_otr != 0) Console.WriteLine("Кол-во отриц. = " + num_otr);
     else Console.WriteLine("Отриц-х эл-тов нет");

     Console.WriteLine("Макс. элемент = " + mas.MaxElem());
   }
}
```

Пример анализа задания

Найти среднее арифметическое элементов, расположенных между минимумом и максимумом

- Варианты результата:
  - выводится среднее арифметическое
  - выводится сообщение «таких элементов нет» (мин. и макс. рядом или все элементы массива одинаковы)
- Вопрос: если макс. или мин. эл-тов несколько?
- Варианты тестовых данных:
  - минимум левее максимума
  - наоборот
  - рядом
  - более одного мин/макс
  - все элементы массива равны
  - все элементы отрицательные

Еще один пример анализа задания

Найти сумму элементов, расположенных между первым и последним элементами, равными нулю

- Варианты результата:
  - выводится сумма
  - выводится сообщение «таких элементов нет» (нулевые эл-ты рядом или их меньше двух)
- Варианты тестовых данных:
  - два эл-та, равных нулю, не рядом
  - два эл-та, равных нулю, рядом
  - один эл-т, равный нулю
  - ни одного
  - более двух
  - …

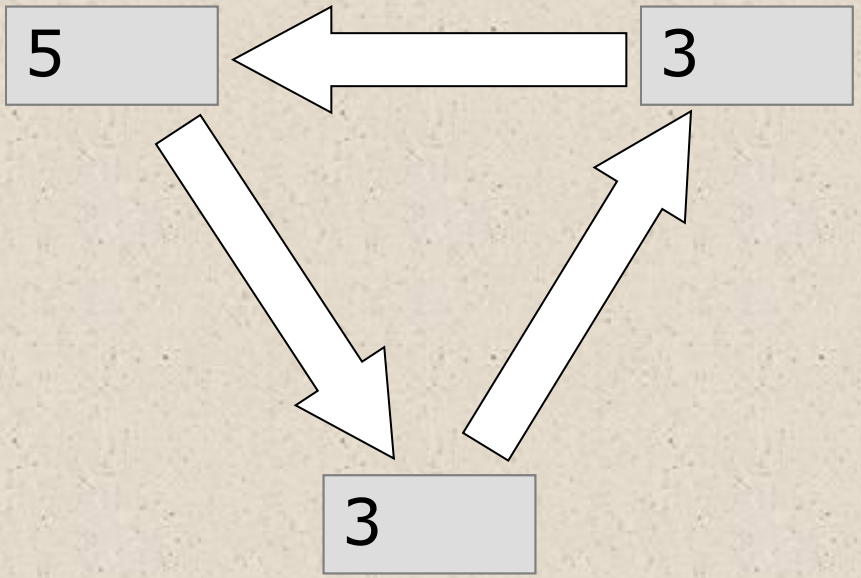
Сортировка выбором

Алгоритм сортировки

Повторить (n-1) раз (i := 1 to n-1):

- Среди элементов, начиная с i-го, найти, где расположен минимальный элемент массива
- Поменять его местами с i-м элементом. i-й элемент теперь на нужном месте.

Обмен значений двух переменных



Использование методов класса Array

```
static void Main()
        {
            int[] a =  { 24, 50, 18, 3, 16, -7, 9, -1 };
            PrintArray( "Исходный массив:", a );
            Console.WriteLine( Array.IndexOf( a, 18 ) );
            Array.Sort(a);     // Array.Sort(a, 1, 5);
            PrintArray( "Упорядоченный массив:", a );
            Console.WriteLine( Array.BinarySearch( a, 18) );
            Array.Reverse(a);     // Array.Reverse(a, 2, 4);
        }
public static void PrintArray( string header, int[] a ) {
            Console.WriteLine( header );
            for ( int i = 0; i < a.Length; ++i )
                Console.Write( "\t" + a[i] );
            Console.WriteLine();
        }
```

Что вы должны уметь найти в массиве:

- минимум/максимум [по модулю]
- номер минимума/максимума [по модулю]
- номер первого/второго/последнего положительного/отрицательного/нулевого эл-та
- сумма/произведение/количество/сред. арифм-е положительных/отрицательных/нулевых эл-тов

- упорядочить массив НЕ методом пузырька.

- анализировать все возможные варианты расположения исходных данных

Прямоугольные массивы

- *Прямоугольный массив* имеет более одного измерения. Чаще всего в программах используются двумерные массивы. Варианты описания двумерного массива:

тип[,] имя;

тип[,] имя = new тип [разм_1, разм_2];

тип[,] имя = { список_инициализаторов };

тип[,] имя = new тип [,] { список_инициализаторов };

тип[,] имя = new тип [разм_1, разм_2] { список_инициализаторов };

- Примеры описаний (один пример на каждый вариант описания):

```
int[,] a;                                       // элементов нет
int[,] b = new int[2, 3];                       // элементы равны 0
int[,] c = {{1, 2, 3}, {4, 5, 6}};              // new подразумевается
int[,] c = new int[,] {{1, 2, 3}, {4, 5, 6}};   // разм-сть вычисляется
int[,] d = new int[2,3] {{1, 2, 3}, {4, 5, 6}}; // избыточное описание
```

- К элементу двумерного массива обращаются, указывая номера строки и столбца, на пересечении которых он расположен:

a[1, 4] b[i, j] b[j, i]

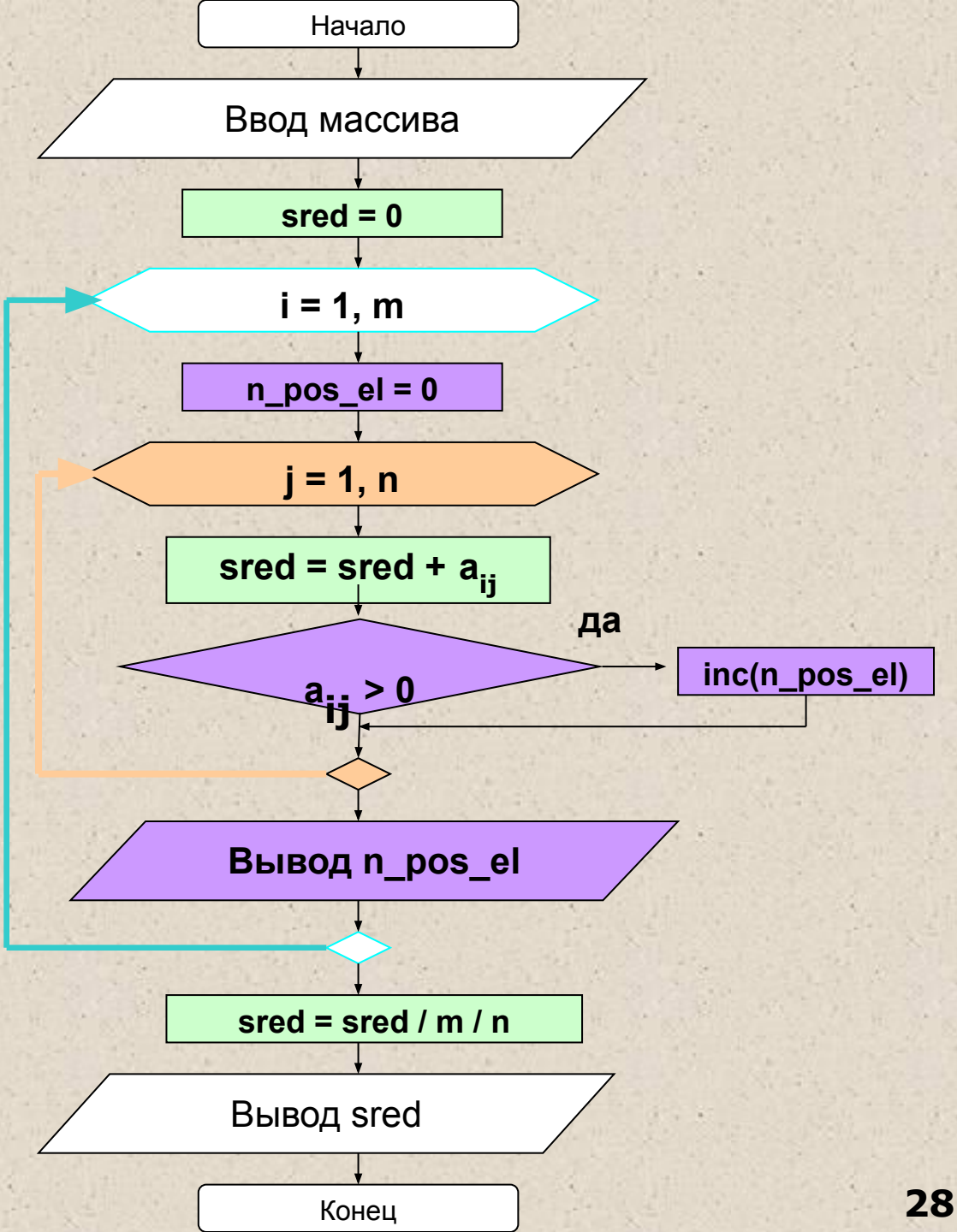
- Компилятор воспринимает как **номер строки первый индекс**, как бы он ни был обозначен в программе.

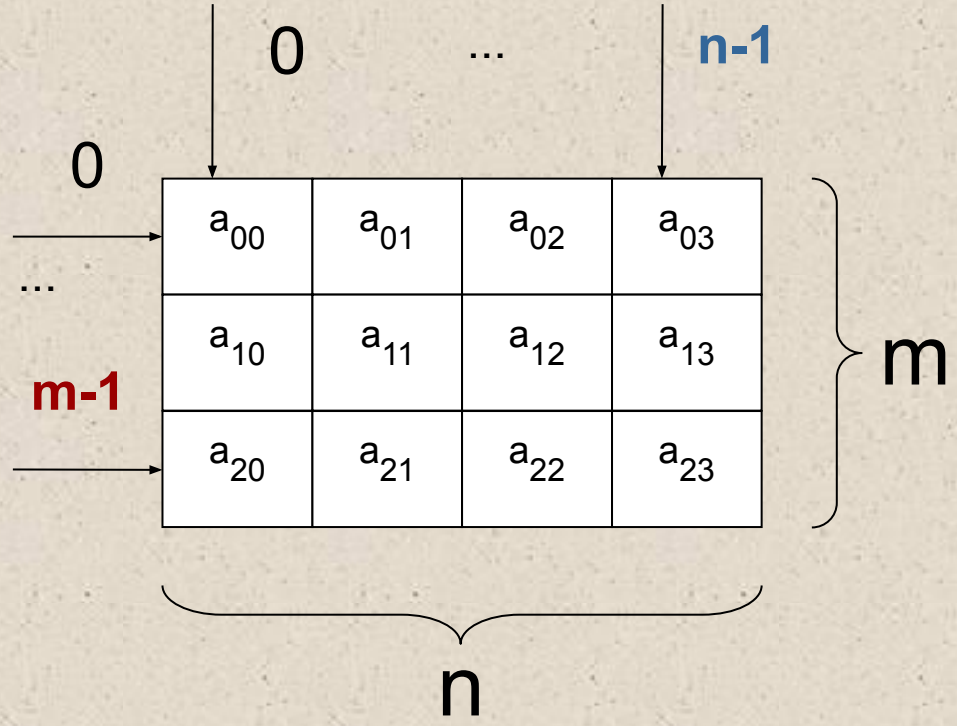
Пример

Программа определяет:

- среднее арифметическое всех элементов;
- количество положительных элементов в каждой строке

для целочисленной матрицы размером 3 x 4





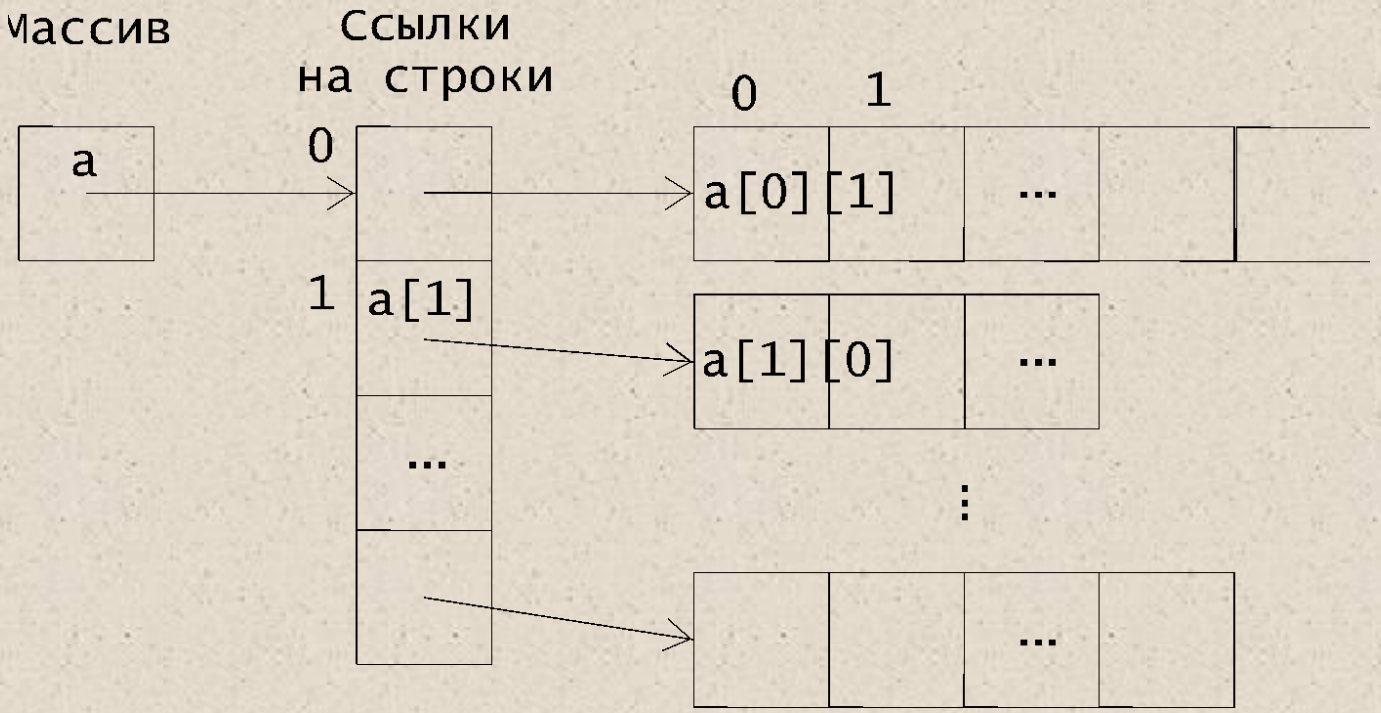

```
const int m = 3, n = 4;
int[,] a = new int[m, n] {
              { 2,-2, 8, 9 },
              {-4,-5, 6,-2 },
              { 7, 0, 1, 1 }
          };
Console.WriteLine( "Исходный массив:" );
for ( int i = 0; i < m; ++i )
{   for ( int j = 0; j < n; ++j )
        Console.Write( "\t" + a[i, j] );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

```
int nPosEl;
for ( int i = 0; i < m; ++i )
{
    nPosEl = 0;
    for ( int j = 0; j < n; ++j )
        if ( a[i, j] > 0 ) ++nPosEl;
    Console.WriteLine( "В строке {0} {1} положит-х эл-в", i, nPosEl);
}

double sum = 0;
foreach ( int x in a ) sum += x;  // все элементы двумерного массива!
Console.WriteLine( "Среднее арифметическое всех элементов: "
                        + sum / m / n );
```

Ступенчатые массивы

В *ступенчатых массивах* количество элементов в разных строках может различаться. В памяти ступенчатый массив хранится иначе, чем прямоугольный: в виде нескольких внутренних массивов, каждый из которых имеет свой размер. Кроме того, выделяется отдельная область памяти для хранения ссылок на каждый из внутренних массивов.



Описание ступенчатого массива

тип[][] имя;

Под каждый из массивов, составляющих ступенчатый массив, память требуется выделять явным образом:

```
int[][] a = new int[3][];   // память под ссылки на 3 строки
    a[0] = new int[5];     // память под 0-ю строку (5 эл-в)
    a[1] = new int[3];     // память под 1-ю строку (3 эл-та)
    a[2] = new int[4];     // память под 2-ю строку (4 эл-та)
```

Или:

```
int[][] a = { new int[5], new int[3], new int[4] };
```

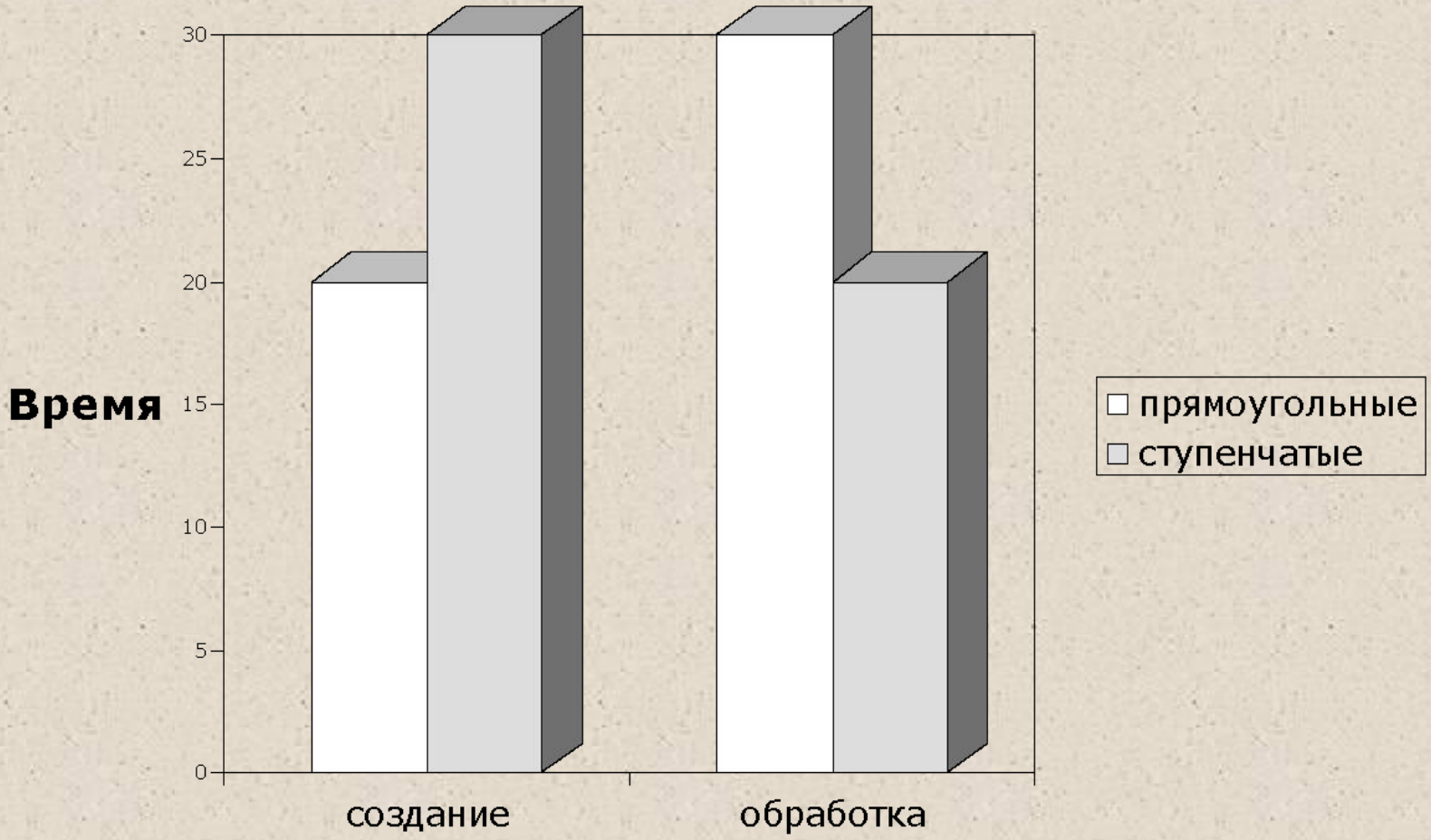
Обращение к элементу ступенчатого массива:

a[1][2] a[i][j] a[j][i]

Пример

```
int[][] a = new int[3][];
        a[0] = new int [5] { 24, 50, 18, 3, 16 };
        a[1] = new int [3] { 7, 9, -1 };
        a[2] = new int [4] { 6, 15, 3, 1 };
        Console.WriteLine( "Исходный массив:" );
        foreach ( int [] mas1 in a )
           {
              foreach ( int x in mas1 )
                    Console.Write( "\t" + x );
              Console.WriteLine();
           }
        // поиск числа 18 в нулевой строке
        Console.WriteLine( Array.IndexOf( a[0], 18 ) );
```

Эффективность работы с двумерными массивами



Передача массивов как параметров метода

```
class Program {
    static void Main(string[] args)
    {   const int n = 3, m = 4;
        double[,] a = new double[n, m] {{2,3,4,7}, {4,3,2,0}, {2,0,1,8}};
        Console.WriteLine("Сумма элементов: " + Sum(a));
        bool[] nums = RowsWithNulls(a, n, m);
        Console.Write("Номера строк, содержащих нули: ");
        for (int i = 0; i < n; ++i)
            if (nums[i]) Console.Write("   " + i);
    }
    static double Sum(double[,] x)
    static bool[] RowsWithNulls(double[,] x, int n, int m)
    {   bool[] nums = new bool[n];
        for( int i = 0; i < n; ++i)
            for( int j = 0; j < m; ++j)
                if(Math.Abs(x[i,j]) < 1e-9) nums[i] = true;
        return nums;
    }
}
```