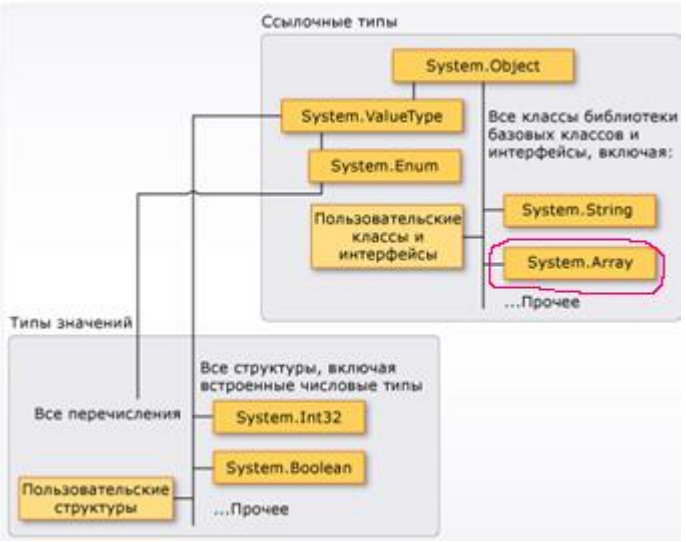
Массивы

Массивом называют упорядоченную последовательность элементов одного типа. Каждый элемент массива имеет индексы, определяющие порядок элементов.

Массив – множество однотипных элементов. Любой массив является производным от класса System.Array



Массивы бывают:

- Статические - если все границы заданы константными выражениями
- Динамические - если выражения, задающие границы, зависят от переменных

Доступ к элементам массива реализуется в соответствии с правилом индексации – по каждому измерению индексация осуществляется с НУЛЯ до n–1, где n – количество элементов размерности.

Синтаксис объявления и инициализации массива позволяет определять две различных категории массивов:

- ***простые*** (прямоугольные) массивы,
- ***jagged*** (ступенчатые, зубчатые, неровные) массивы

Одной из характеристик массива является размерность массива.

Массив размерности N – это массив массивов ранга N–1. Составляющие массива – это массивы меньшей размерности, являющиеся элементами данного массива.

- Одномерные массивы
- Многомерные массивы
- Ступенчатые массивы

Одномерный массив

Пример объявления трёх массивов с отложенной инициализацией

```
int[ ] a, b, c;
```

При объявлении с отложенной инициализацией сам массив не формируется, а создаётся только ссылка на массив, имеющая неопределённое значение Null

Поэтому пока массив не будет реально создан и его элементы инициализированы, использовать его в вычислениях нельзя

Чаще всего при объявлении массива используется имя с инициализацией. Может быть два варианта инициализации. В первом случае инициализация является явной и задаётся константным массивом. Вот пример:

```
double[ ] x = {5.5, 6.6, 7.7};
```

Если массив инициализируется константным массивом, то в динамической памяти создаётся константный массив с заданными значениями, с которым и связывается ссылка.

Во втором случае создание массива выполняется с помощью операции new.

```
int[ ] d = new int[5];
```

Здесь объявлен динамический целочисленный массив, в котором будут храниться 5 целых чисел. Массив создаётся в динамической памяти, его элементы получают начальные нулевые значения, и ссылка связывается с этим массивом.

Как обычно задаются элементы массива, если они не заданы при инициализации? Они либо вычисляются, либо вводятся пользователем.

```
int[ ] A = new int[5], B= new int[5], C= new int[5];

for(int i = 0; i < 5; i++) A[i] = int.Parse(Console.ReadLine());

for(int i = 0; i < 5; i++) C[i] = A[i] + B[i];

int[ ] u, v;
u = new int[3];
for(int i = 0; i < 3; i++) u[i] = i + 1;

// v = {1,2,3};     // присваивание константного массива недопустимо!

v = new int[4];
```

```
v = u;                // допустимое присваивание – массивы одного класса
```

Это корректное ссылочное присваивание, хотя u и v имеют разное число элементов, но они являются объектами одного класса – оба массива целочисленные.

В результате присваивания память, отведённая массиву v, освободится, ею займется теперь сборщик мусора. Обе ссылки u и v будут теперь указывать на один и тот же массив, так что изменение элемента одного массива немедленно отразится на другом массиве. Имена u и v становятся синонимами (или псевдонимами друг друга…).

В примерах объявлялись статические массивы, поскольку нижняя граница равна нулю по определению, а верхняя всегда задавалась в этих примерах константой.

В **C#** все массивы, независимо от того, каким выражением описывается граница, рассматриваются как динамические, и память для них распределяется в "куче". Чисто синтаксически нет существенной разницы в объявлении статических и динамических массивов. Выражение, задающее границу изменения индексов, в динамическом случае содержит переменные.

Единственное требование – значения переменных должны быть определены в момент объявления. Это ограничение в **C#** выполняется автоматически, поскольку хорошо известно, насколько требовательно **C#** контролирует инициализацию переменных.

Объявление динамического массива :

```
Console.WriteLine("Введите число элементов массива A1");
int size = int.Parse(Console.ReadLine());
int[ ] A1 = new int[size]
```

Рассмотрим особый вид оператора цикла – оператор foreach. Его синтаксис:

foreach(тип идентификатор in массив) оператор

Цикл работает в полном соответствии со своим названием – тело цикла выполняется для каждого элемента в контейнере. Тип идентификатора должен быть согласован с типом элементов, хранящихся в массиве данных. Предполагается также, что элементы массива упорядочены. На каждом шаге цикла идентификатор, задающий текущий элемент массива, получает значение очередного элемента в соответствии с порядком, установленным на элементах массива. С этим текущим элементом и выполняется тело цикла - выполняется столько раз, сколько элементов находится в массиве. Цикл заканчивается, когда полностью перебраны все элементы массива.

Серьёзным недостатком циклов **foreach** в языке **C#** является то, что цикл работает **только на чтение**, но не на запись элементов.

Так что наполнять массив элементами приходится с помощью других операторов цикла.

В приведённом ниже примере показана работа с трёхмерным массивом. Массив создаётся с использованием циклов типа for, а при нахождении суммы его элементов, минимального и максимального значения используется цикл foreach

```
int [,,] arr3d = new int[10, 10, 10];
for(int i = 0; i < 10; i++)
   for(int j = 0; j < 10; j++)
      for(int k = 0; k < 10; k++)
         arr3d[i, j, k] = int.Parse(Console.ReadLine());
long sum = 0;
int min = arr3d[0, 0, 0], max = arr3d[0, 0, 0];

foreach(int item in arr3d)
{
   sum += item;
   if(item > max) max = item;
     else     if (item < min) min = item;
}

Console.WriteLine("sum = {0}, min = {1}, max = {2}",  sum, min, max)
```

Многомерный массив

Спецификатор размерности N, состоящий из N неявных спецификаторов [„„ ... ,], специфицирует составляющую массива размерности N.

Ссылка на ОДНОМЕРНЫЙ массив ОДНОМЕРНЫХ элементов массива, каждый из которых является ОДНОМЕРНЫМ массивом элементов типа int

```
// Ссылка на трехмерный массив типа int
// Размеры всех составляющих массивов одного уровня равны между собой ("прямоугольные" массивы).
int[ , , ] arr0;


int[,] k = new int [2,3];                  //  двумерный массив
int[,] k = {{2,-2},{3,-22},{0,4}};         //  инициализация

int[,,] k1 = new int [10,10,10];           // трехмерный массив
```

```
// пример перемножения матриц

public void MultMatr(int[,]A,  int[,]B,  int[,]C)
{
   if (A.GetLength(1) != B.GetLength(0))
                Console.WriteLine("MultMatr: ошибка размерности!");
   else
      for(int i = 0; i < A.GetLength(0); i++)
         for(int j = 0; j < B.GetLength(1); j++)
            {
               int s=0;
               for(int k = 0; k < A.GetLength(1); k++)  s+= A[i,k]*B[k,j];
               C[i,j] = s;
            }
}
```

Ступенчатые (jagged) массивы

```
int[ ][ ] k = new int [2][ ];   //Объявление двумерного ступенчатого массива
k[0]=new int[3];                // 0-й элемент ступенчатого массива
k[1]=new int[4];                // 1-й элемент ступенчатого массива
```

```
int[ ][ ] jagger = new int[3][ ]    // инициализация значениями
{
        new int[ ] {5,7,9,11},
        new int[ ] {2,8},
        new int[ ] {6,12,4}
};
```

```
int[ ][ ] jagger1 = new int[3][ ]    // инициализация нулевыми значениями
{
    new int[4],
    new int[2],
    new int[3]
};

int[ ][ ]  jagger2 =
{
    new int[4],
    new int[2],
    new int[3]
};
```

Ниже представлены способы объявления ссылок на массивы различной размерности и конфигурации:

```
int[ ][ ][ ] arr3;

// Ссылка на ОДНОМЕРНЫЙ массив составляющих, каждая из которых является
// ДВУМЕРНЫМ массивом массивов элементов типа int.
// При этом никаких ограничений на размеры "прямоугольных" составляющих
// данное объявление не содержит. У всех составляющих могут быть разные
// размеры.

 int[ ][ , ] arr4;

// Ссылка на ДВУМЕРНЫЙ массив составляющих, каждая из которых является
// ОДНОМЕРНЫМ массивом элементов типа int.
// При этом никаких ограничений на размеры одномерных составляющих
// данное объявление не содержит. У всех составляющих могут быть разные
// размеры.

int[ , ][ ] arr5;
```

Ниже показано создание одномерных, многомерных массивов и массивов массивов.

```
class TestArraysClass
{
    static void Main()
    {
        int[ ] array1 = new int[5];
        int[ ] array2 = new int[] { 1, 3, 5, 7, 9 };
        int[ ] array3 = { 1, 2, 3, 4, 5, 6 };
        int[ , ] multiDimensionalArray1 = new int[2, 3];
        int[ , ] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };
        int[ ][ ] jaggedArray = new int[6][ ];
        jaggedArray[0] = new int[4] { 1, 2, 3, 4 };
    }
}
```

Инициализация массивов

Массив нужно обязательно инициализировать. Только тогда он будет размещен в памяти.

В управляемой памяти нет ничего, что бы создавалось без конструктора, – даже если его присутствие не обозначено явным образом:

```
string s = "qwerty";
int[ ] intArray = {1,2,3,4,5};
```

Эти операторы являются всего лишь сокращенной формой следующих операторов определения:

```
string s = new string( new char[ ]{'q','w','e','r','t','y'}) ;
int[ ] intArray = new int[5]{1,2,3,4,5};
```

К моменту создания массива должна быть определена его основная характеристика – количество составляющих данный массив элементов.

```
int x;
int[  ] arr0 = new int[x];      // К моменту определения массива значение
                                // x должно быть определено!

int x = 10;
int[ ][ ] arr1 = new int[x][125];
```

При определении такой конструкции, как массив массивов, инициализирующая запись должна содержать **полную** информацию о характеристиках **первой** составляющей массива, то есть об общем количестве составляющих данный массив элементов массивов. Все остальные описатели могут оставаться пустыми.

```
int val1 = 100;
int [  ][  ]        x0 = new int[15][ ];
int [  ][  ][  ]    x1 = new int[val1][ ][  ];
int [ , ][  ]       x2 = new int[val1,7][ ];
int [ , ,][ , ][ ]  x3 = new int[2,val1,7][ , ][  ];
```

Инициализация многомерных массивов – требует особого внимания.

В силу того, что массив является объектом ***ссылочного типа***, составляющие одного массива могут быть использованы для инициализации другого массива. Ответственность за возможные переплетения ссылок возлагается на программиста:

```
int [ ] q = {0,1,2,3,4,5,6,7,8,9};

int [ ][ ] d1 = {
                  new int[3],
                  new int[5]
                 };

 int [ ][ ][ ] d2 = new int[2][ ][ ];

 d2[0] = new int[50][ ];
 d2[0][0] = d1[0];              // d2[0][0] ссылается на составляющую массива d1.
 d2[0][1] = new int[125];
 d2[1] = new int[50][ ];
 d2[1][1] = new int[10]{1,2,3,4,5,6,7,8,9,10};
 d2[1][0] = q;                  // d2[1][0] ссылается на ранее объявленный и определенный массив q.
```

В следующем примере одномерный массив myArray строится из элементов типа int.

```
int[ ] myArray = new int [5];
```

Операция new используется для создания массива и инициализации его элементов предопределенными значениями. В результате выполнения этого оператора все элементы массива ***будут установлены в ноль***.

Простой строковый массив можно объявить и проинициализировать аналогичным образом:

```
string[ ] myStringArray = new string[6];
```

При совмещения явной инициализации элементов массива с его объявлением спецификатор размерности не требуется, поскольку соответствующая информация может быть получена непосредственно из инициализирующего списка. Например:

```
int[ ] myArray = new int[ ] {1, 3, 5, 7, 9};
```

Строковый массив может быть проинициализирован аналогичным образом:

```
string[ ] weekDays = new string[ ]{"Sun","Sat","Mon","Tue","Wed","Thu","Fri"};
```

Это не единственный способ объявления и инициализации. Если объявление совмещается с инициализацией, операция new может быть опущена:

```
string[ ] weekDays = {"Sun","Sat","Mon","Tue","Wed","Thu","Fri"};
```

Объявление и инициализация вообще могут быть размещены в разных операторах.

Но в этом случае без операции new нельзя:

```
int[ ] myArray;
myArray = new int[ ] {1, 3, 5, 7, 9}; // Так можно.

myArray = {1, 3, 5, 7, 9};            // Так нельзя.
```

Объявление одномерного массива, состоящего из трех элементов, каждый из которых является одномерным массивом целых:

```
int[ ][ ] myJaggedArray = new int[3][ ];
```

Дальнейшее использование этого массива требует инициализации его элементов.

```
myJaggedArray[0] = new int[5];
myJaggedArray[1] = new int[4];
myJaggedArray[2] = new int[2];
```

Каждый из элементов является одномерным массивом целых. Количество элементов каждого массива задается соответствующим оператором определения.

Ниже показан пример использования заполняющей инициализации, при которой одновременно с определением (созданием) массивов производится присвоение элементам новорожденных массивов конкретных значений:

```
myJaggedArray[0] = new int[ ] {1,3,5,7,9};
myJaggedArray[1] = new int[ ] {0,2,4,6};
myJaggedArray[2] = new int[ ] {11,22};
```

Этот же массив может быть объявлен и проинициализирован и таким образом:

```
int[ ][ ] myJaggedArray = new int [ ][ ] {     new int[ ] {1,3,5,7,9},
                                                new int[ ] {0,2,4,6},
                                                new int[ ] {11,22}
                                            };
```

Ещё один эквивалентный способ инициализации массива.

```
int[ ][ ] myJaggedArray =  {      new int[ ] {1,3,5,7,9},
                                  new int[ ] {0,2,4,6},
                                  new int[ ] {11,22}
                           };
```

Важно, что при определении составляющих этого массива операция new ***опущена быть не может***.

Доступ к элементам ступенчатого массива обеспечивается посредством выражений индексации:

```
myJaggedArray[0][1] = 33; // Assign 33 to the second element of the first array
myJaggedArray[2][1] = 44; // Assign 44 to the second element of the third array
```

C# позволяет собирать разнообразные конструкции на основе jagged многомерных массивов.

Ниже приводится пример объявления и инициализации одномерного jagged - массива, содержащего в качестве элементов двумерные массивы различных размеров:

```
int[ ][ , ] myJaggedArray = new int [3][ , ] { new int[,] { {1,3}, {5,7} },
                                               new int[,] { {0,2}, {4,6}, {8,10} },
                                               new int[,] { {11,22}, {99,88}, {0,9} }
                                             };
```

```
int[ ][ , ] myJaggedArray = new int [3][ , ] { new int[,] { {1,3}, {5,7} },
                                               new int[,] { {0,2}, {4,6}, {8,10} },
                                               new int[,] { {11,22}, {99,88}, {0,9} }
                                             };
```

Доступ к отдельным элементам jagged - массива обеспечивается различными комбинациями выражений индексации. В приводимом ниже примере выводится значение элемента массива [1,0], расположенного по нулевому индексу myJaggedArray (это 5):

```
Console.Write("{0}", myJaggedArray[0][1,0]);
```

И еще один пример, где строится массив myArray, элементами которого являются массивы. Каждый из составляющих имеет собственные размеры.

```
using System;
public class ArrayTest
{   public static void Main( )
    { int[ ][ ] myArray = new int[2][ ];
      myArray[0] = new int[5] {1,3,5,7,9};
      myArray[1] = new int[4] {2,4,6,8};

      for (int i=0; i < myArray.Length; i++)
            { Console.Write("Element({0}):", i);
              for (int j = 0 ; j < myArray[i].Length ; j++)
                        Console.Write("{0}{1}",myArray[i][j], j == (myArray[i].Length–1) ? "" : " ");
              Console.WriteLine();
             }
    }
}
```

- Результат:

```
Element(0): 1 3 5 7 9
Element(1): 2 4 6 8
```

Рассмотрим следующее объявление:

```
MyType[ ] myArray = new MyType[10];
```

Результат выполнения этого оператора зависит от того, что собой представляет тип MyType.

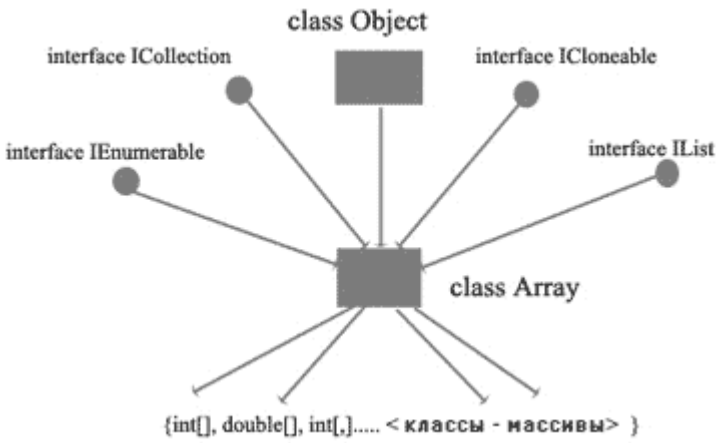
Возможны всего два варианта: MyType может быть ***типом-значением*** (**Value Type**) или ***типом-ссылкой*** **(Reference Type)**

- Если это тип-значение, результатом выполнения оператора будет массив, содержащий 10 объектов MyType с предопределенными значениями.
- Если MyType является ссылочным типом, то в результате выполнения данного оператора будет создан массив из 10 элементов типа "ссылка", каждый из которых будет проинициализирован пустой ссылкой – значением null.

Встроенный сервис по обслуживанию простых массивов

В .NET все массивы происходят от одного общего (базового) класса System.Array. Это означает, что массивы обеспечиваются специальным набором методов для создания, управления, поиска и сортировки элементов массива.

Помимо наследования свойств и методов класса Object и ряда интерфейсов: ICloneable, IList, ICollection, IEnumerable, класс Array имеет довольно большое число собственных методов и свойств. Взгляните, как выглядит отношение наследования на семействе классов, определяющих массивы



Статические методы класса Array позволяют решать самые разнообразные задачи:

Copy – позволяет копировать весь массив или его часть в другой массив.

IndexOf, LastIndexOf – определяют индексы первого и последнего вхождения образца в массив, возвращая значение -1, если такового вхождения не обнаружено.

Reverse – выполняет обращение массива, переставляя элементы в обратном порядке.

Sort – осуществляет сортировку массива.

BinarySearch – определяет индекс первого вхождения образца в отсортированный массив, используя алгоритм двоичного поиска.

К числу таких методов и свойств, в частности, относятся следующие свойства:

```
public int Length { get; }        //  Возвращает целое число, представляющее общее количество
                                  //  элементов во всех измерениях массива.
public int Rank { get ;}          // Возвращает целое число, представляющее количество
                                  // измерений массива.
```

И методы:

```
public static Array CreateInstance(Type, int, int);
                                          // Статический метод (один из вариантов),
                                          //  создает массив элементов заданного типа и
                                          //  определенной размерности:
public int GetLength (int dimension); //  Возвращает количество элементов заданной
                                         //  параметром размерности
public void SetValue(object, int, int);  //  Присваивает элементу массива значение,
                                          //  представленное первым параметром (один из
                                          //   вариантов)
public object GetValue(int, int);         // Извлекает значение из двумерного массива по индексам
                                          // (один из вариантов):
```

```
using System;
public class DemoArray
{
   public static void Main()
      { // Создание и инициализация двумерного массива строк
       Array myArray=Array.CreateInstance( typeof(String), 2, 4 );
       myArray.SetValue( "Вышел ", 0, 0 );
       myArray.SetValue( "Месяц, ", 0, 1 );
       myArray.SetValue( "из", 0, 2 );
       myArray.SetValue( "тумана,", 0, 3 );
       myArray.SetValue( "вынул", 1, 0 );
       myArray.SetValue( "ножик", 1, 1 );
       myArray.SetValue( "из", 1, 2 );
       myArray.SetValue( "кармана.", 1, 3 );
       // Показали содержимое массива.
       Console.WriteLine( "The Array contains the following values:" );
       for ( int i = myArray.GetLowerBound(0); i <= myArray.GetUpperBound(0); i++ )
       for ( int j = myArray.GetLowerBound(1); j <= myArray.GetUpperBound(1); j++ )
       Console.WriteLine( "\t[{0},{1}]:\t{2}", i, j, myArray.GetValue( i, j ) );
      }
}
```

Использование массивов в качестве параметров

Полностью построенный массив можно передать в качестве входного параметра. В качестве параметра методу можно передать **ссылку на массив**

Тип и количество составляющих данный массив компонентов для передачи параметров значения не имеют. Важно, что в стеке будет выделено определенное (***соответствующее значению первой размерности***) количество проинициализированных ссылок на составляющие данный одномерный массив элементы

Пустая ссылка на массив может быть передана методу в качестве ***выходного*** параметра. Например:

```
int[ ] myArray;                    // myArray == null
PrintArray(myArray);
```

Передаваемый в качестве параметра массив может быть предварительно проинициализирован:

```
// cs_sd_arrays.cs

using System;
 public class ArrayClass
{
  static void PrintArray(string[ ] w)
    {  for (int i = 0 ; i < w.Length ; i++)
                                  { Console.Write(w[i] + "{0}", i < w.Length – 1 ? " " : ""); }
        Console.WriteLine();
     }

   public static void Main( )
   {   // Declare and initialize an array:
        string[ ] WeekDays = new string [ ] {"Sun","Sat","Mon","Tue","Wed","Thu","Fri"};
        PrintArray(WeekDays);   // Pass the array as a parameter:
   }
}
```

Пример совмещения инициализации массива с его передачей как параметра методу:

```
using System;
public class ArrayClass
{
    static void PrintArray(int[,] w)
      { // Display the array elements:
        for (int i=0; i < 4; i++)
           for (int j=0; j < 2; j++)
                Console.WriteLine("Element({0},{1})={2}", i, j, w[i,j]);
      }

   public static void Main( )
      { // Pass the array as a parameter:
        PrintArray(new int[,] {{1,2}, {3,4}, {5,6}, {7,8}});
       }
}
```

Спецификатор params

- Неопределенное (переменное) количество однотипных параметров или список параметров переменной длины передается в функцию в виде ссылки на одномерный массив. Эта ссылка в списке параметров функции должна быть последним элементом списка параметров. Ссылке должен предшествовать спецификатор params.

- В выражении вызова метода со списком параметров, члены списка могут присутствовать либо в виде списка однотипных значений (этот список преобразуется в массив значений), либо в виде ссылки на ***одномерный массив*** значений определенного типа:

```
using System;

 class Class1
{    // Методы, принимающие списки параметров переменной длины.
     // Их объявление. Единственный параметр способен принимать
     // массивы значений переменной длины.

     static void f1(params int[ ] m)
       { Console.WriteLine(m.Length.ToString()); }

      static void f2(params int[ ][ , ] p)
        { Console.WriteLine(p.Length.ToString()); }

      static void Main(string[ ] args)
          { // Выражения вызова. На основе списков однотипных
            // значений формируются массивы.
            f1(0,1,2,3,4,5);
            f1(0,1,2,3,4,5,6,7,8,9,10);
            f2(new int[ , ]{{0,1},{2,3}}, new int[ , ]{{0,1,2,3,4,5},{6,7,8,9,10,11}});
            f2(new int [ , ]{{0,1},{2,3}});
          }
}
```