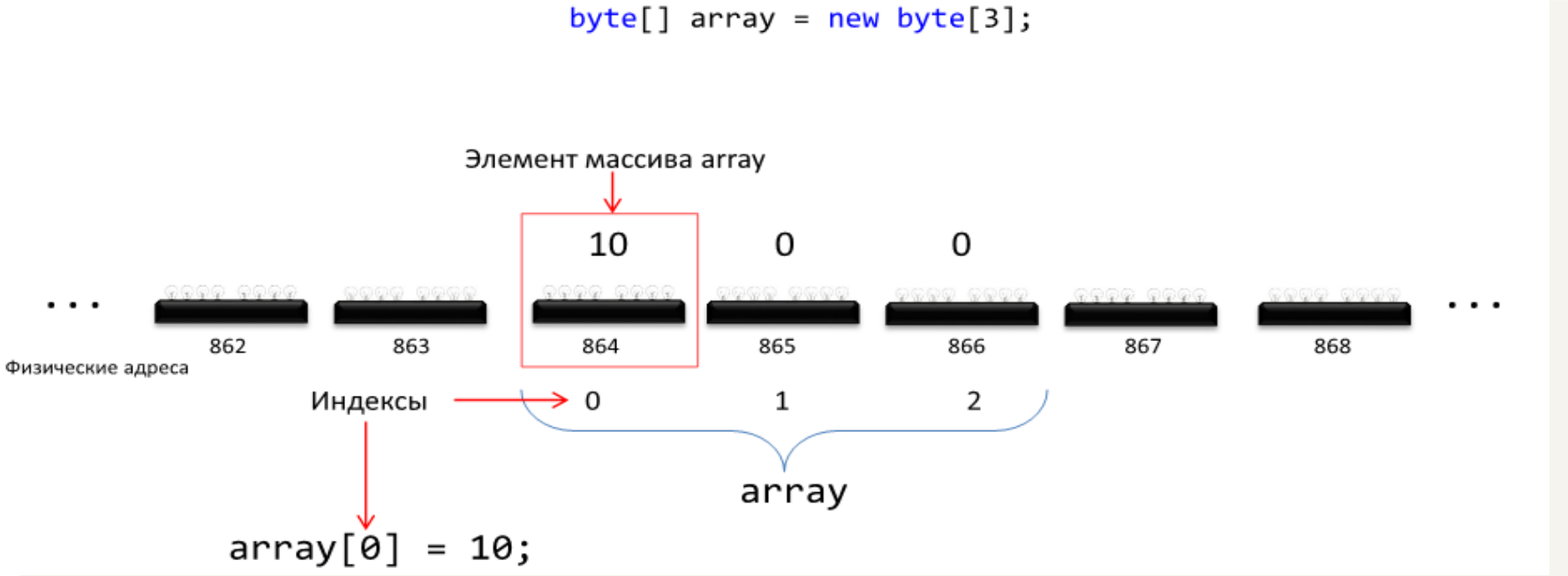
Массивы и строки

Массив представляет собой совокупность переменных одного типа, расположенных в памяти непосредственно друг за другом, с общим для обращения к ним именем.



Особенности массивов в C#

- Все массивы в C# унаследованы от класса System.Array.

Следовательно:

- наличие методов различного назначения
- освобождение памяти сборщиком мусора

- Изменен синтаксис объявления массивов
- После выделения памяти инициализация элементов происходит следующим образом: значения всех простых типов устанавливаются в «0», значения логического типа – в false, ссылки – в null.

Массивы

Одномерные массивы.

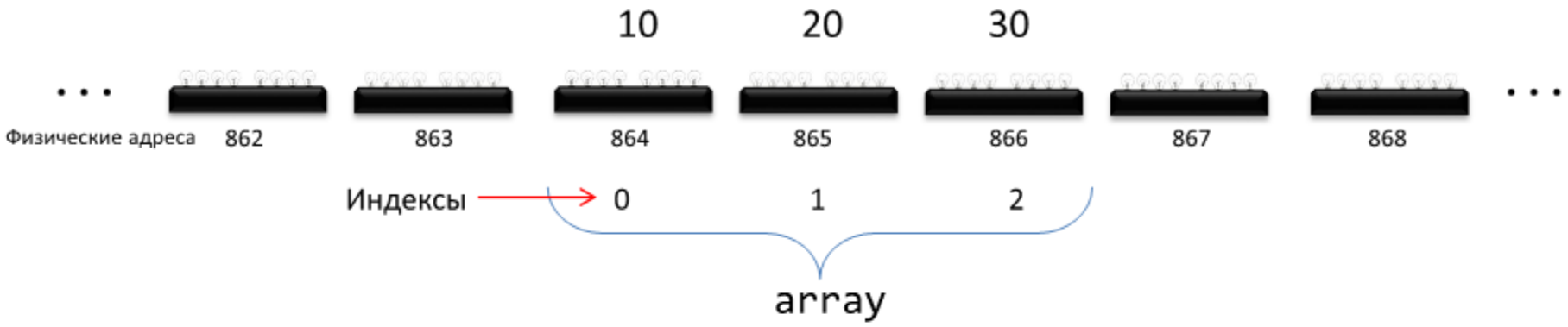
Многомерные массивы.

Рваные массивы.

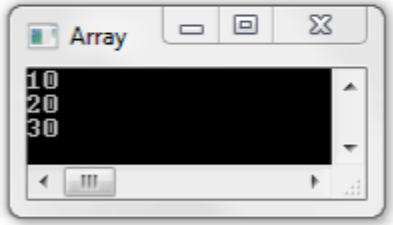
Использование цикла foreach.


byte - тип элементов массива
имя массива
[3] - количество элементов массива
byte[] array = new byte[3];
квадратные скобки указывают на то,
что переменная array типа byte - массив
выражение создания массива

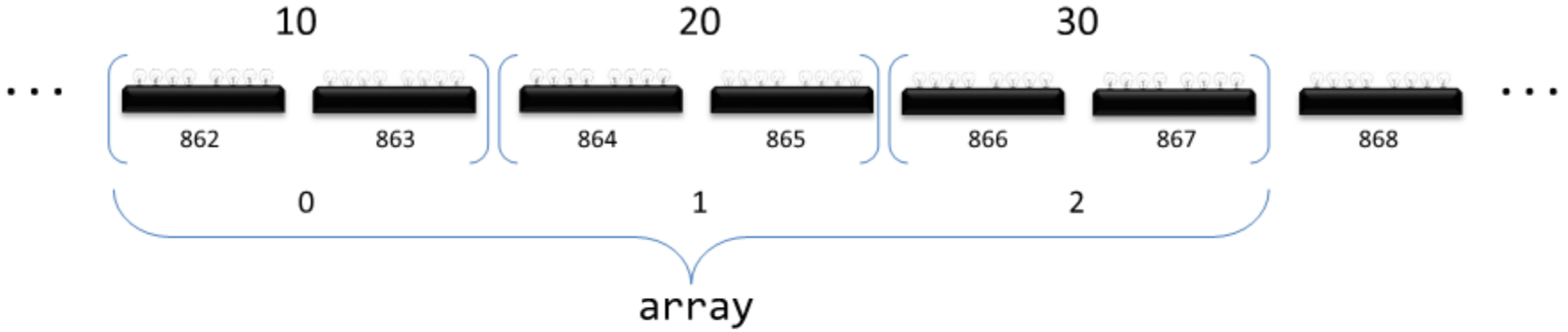
Объявление и инициализация массивов



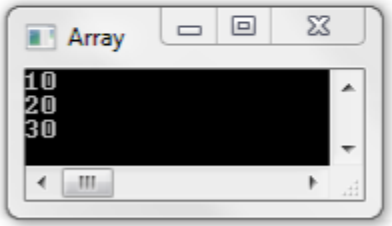

```
byte[] array = new byte[3];

array[0] = 10;
array[1] = 20;
array[2] = 30;

Console.WriteLine(array[0]);
Console.WriteLine(array[1]);
Console.WriteLine(array[2]);
```



Объявление и инициализация массивов




```
short[] array = new short[3];

array[0] = 10;
array[1] = 20;
array[2] = 30;

Console.WriteLine(array[0]);
Console.WriteLine(array[1]);
Console.WriteLine(array[2]);
```



Объявление и инициализация массивов

```
// объявление массива (целых чисел)
   int[] mas1;  mas1=new int[5];
   int[] mas2 = new int[5];

// объявление массива + инициализация
   int[] mas3 = new int[5]{1,2,3,4,5};
   int[] mas4 = new int[] { 1, 2, 3, 4, 5 };
   int[] mas5 =  { 1, 2, 3, 4, 5 };

// объявление массива (строки)
   string[] info = { "Фамилия", "Имя", "Отчество" };

// объявление массива (символы)
   char[] symbol = new char[4] { 'X', 'Y', 'Z', 'M' };
```

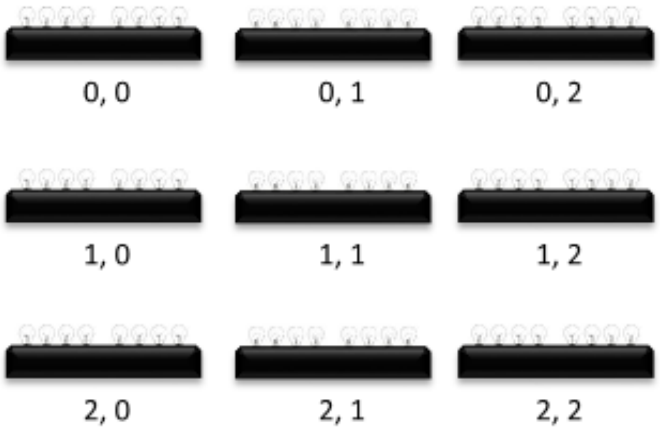
Многомерные

Многомерные массивы – массивы имеющие более одного индекса

Многомерные массивы

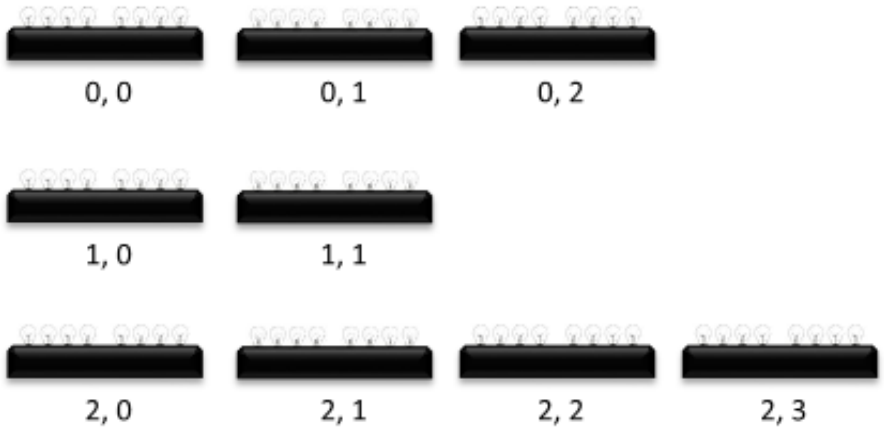
Прямоугольные

Массивы, которые содержат несколько измерений, где все строки имеют одинаковую длину.



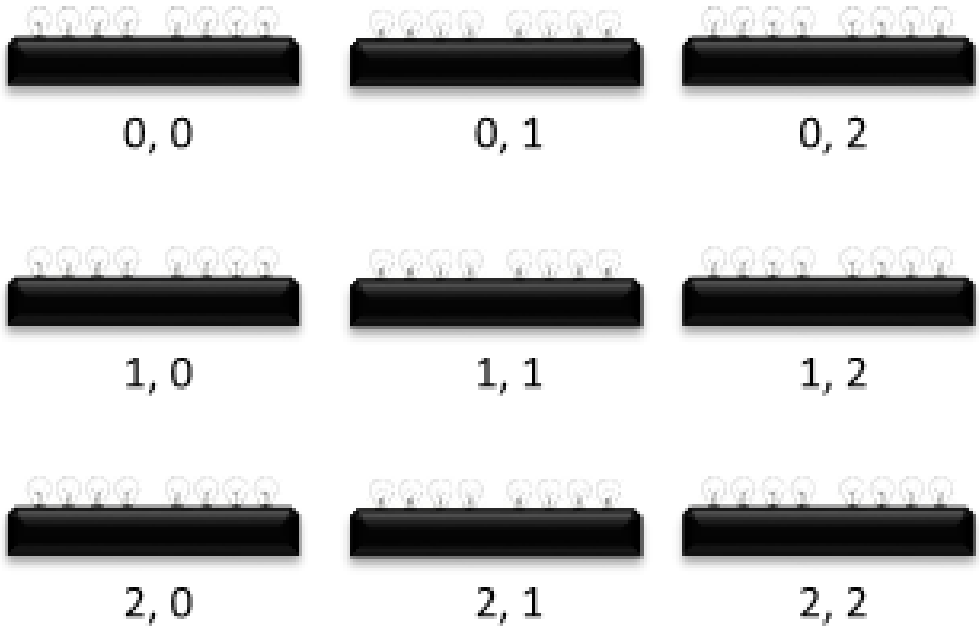
Зубчатые

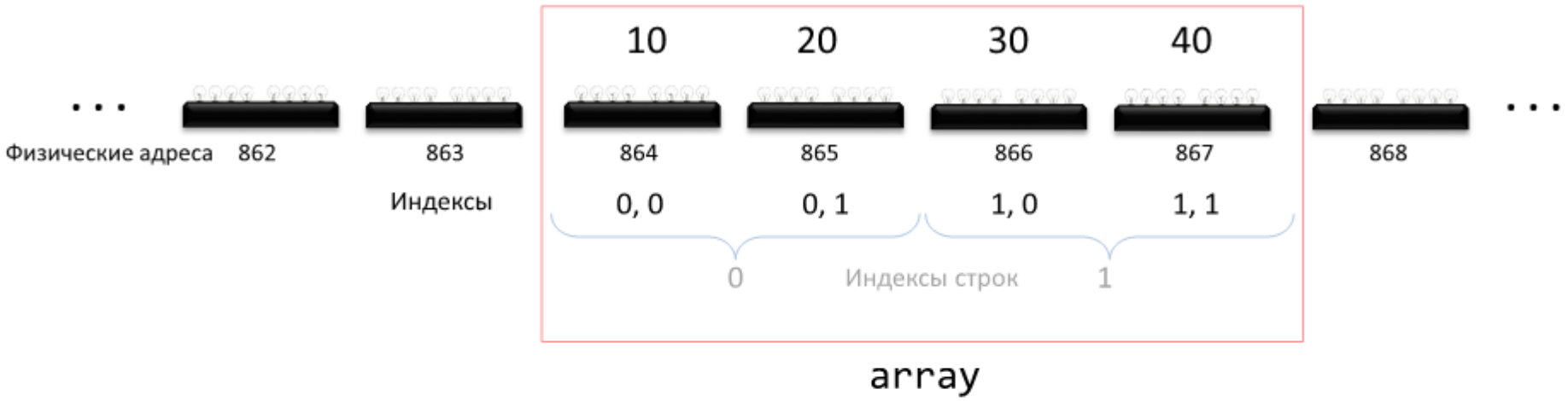
Массивы, которые содержат некоторое количество внутренних массивов, каждый из которых может иметь собственный уникальный верхний предел.



Двумерный массив — прямоугольный массив содержащий два индекса.

```
int[,] array = new int[3,3];
```





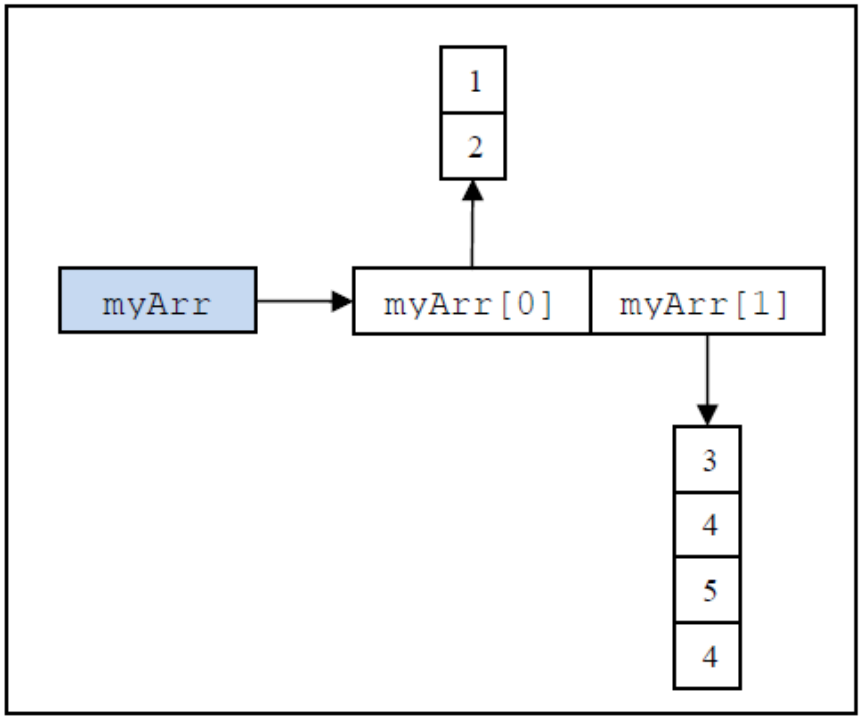

```
byte[,] array = new byte[2,2];

array[0,0] = 10;
array[0,1] = 20;
array[1,0] = 30;
array[1,1] = 40;
```

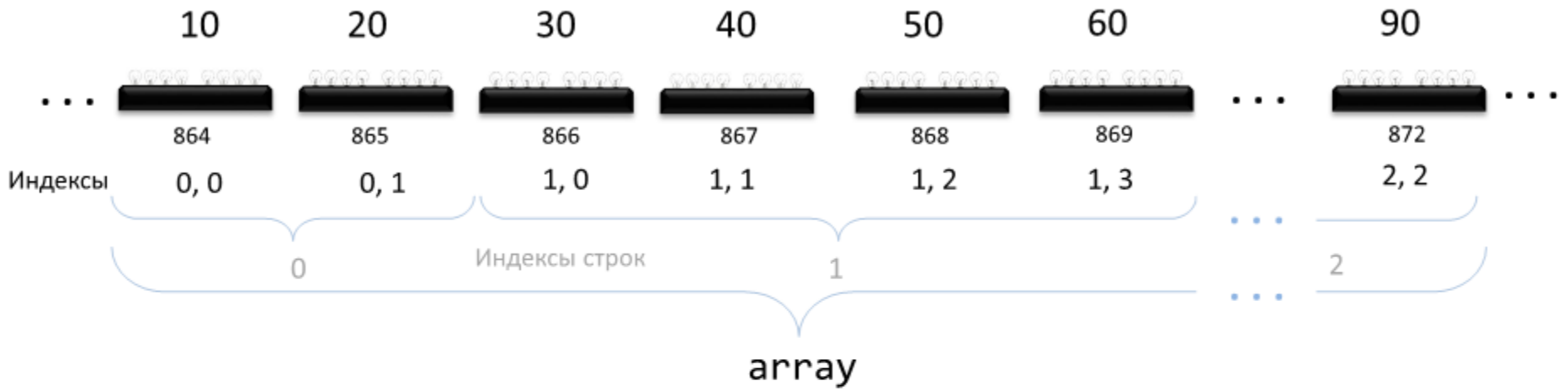
```
// объявление двумерного массива
// + инициализация
    int[,] mas2D = new int [3,2]{ {1,2},
                                  {3,4},
                                  {5,6} };
```

Рваные массивы

```
тип[][] имя_массива = new тип [размер][];
```

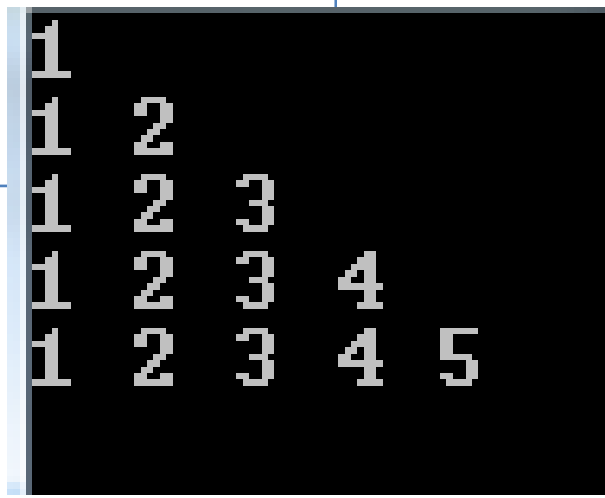
```
// Создание  внешнего массива  на  две  ячейки
int[][] myArr = new int[2][];
// Создание внутреннего массива в  первой  ячейке
myArr[0] = new int[] { 1, 2 };
// Создание внутреннего массива во второй ячейке
myArr[1] = new int[] { 3, 4, 5, 6 };
```






```
byte[][] array = new byte[3][];

array[0] = new byte[] {10,20};
array[1] = new byte[] {30,40,50,60};
array[2] = new byte[] {70,80,90};
```

```
int size = 5;
// Объявление массива
int[][] myArr = new int[size][];
for (int i = 0; i < size; i++)
{
    // Создание внутренних массивов
    myArr[i] = new int[i + 1];
    for (int j = 0; j < i + 1; j++)
    {
        // Заполнение внутренних массивов
        myArr[i][j] = j + 1;
        // Вывод на экран элементов
        Console.Write(myArr[i][j] + " ");
    }
    Console.WriteLine();
}
```



Класс Array.

```
int[] myArr = new int[10] {1,2,5,9,4,9,3,10,1,4};

“Размер массива: "+ myArr.Length
“Максимальный элемент: " + myArr.Max()
“Минимальный элемент : " + myArr.Min()
“Среднее арифметическое элементов: " + myArr.Average()
“Сумма элементов" + myArr.Sum()

Array.Sort(myArr);       // Сортировка всего массива
Array.Reverse(myArr);    // Реверсирование всего массива


"Индекс элемента 9 в массиве - "+ Array.BinarySearch(myArr, 9)
"Индекс 9 в массиве (первый) - " + Array.IndexOf(myArr, 9)
"Индекс 9 в массиве (последний)- " + Array.LastIndexOf(myArr, 9)


 Array.Clear(myArr1, 0, 10); //очистка массива (0-индекс, 10-размер)
```

Класс Array.

```
int[] myArr = new int[10] { 1, 2, 5, 9, 4, 9, 3, 10, 1, 4 };

// значимые типы

int[] myArrValues = new int[10];
myArr.CopyTo(myArrValues, 0);       // Копирование значений массива (0-индекс)

int[] myArrClone = (int[])myArr.Clone();        // Клонирование
Console.WriteLine(myArrClone.Equals(myArr));      // false


// ссылочные типы

string[] myArrStr = { "C#", "WPF","ASP.NET" };

string[] myArrStrValues = new string[3];
myArrStr.CopyTo(myArrStrValues, 0);          // Копирование значений массива (0-индекс)

string[] myArrStrClone = (string[])myArrStr.Clone();    // Клонирование
Console.WriteLine(myArrStrClone.Equals(myArrStr));          // false
Console.WriteLine(myArrStrClone[0].Equals(myArrStr[0]));    // true

Car[] myArrCar = { new Car(), new Car()};
Car[] myArrCarClone = (Car[])myArrCar.Clone();  // Клонирование
Console.WriteLine(myArrCarClone.Equals(myArrCar));          // false
Console.WriteLine(myArrCarClone[0].Equals(myArrCar[0]));    // true
```

• Метод **GetLength** возвращает количество элементов массива по заданному измерению;

```
int[, ,] a = new int[10, 11, 12];
Console.WriteLine(a.Length); // 1320
Console.WriteLine(a.GetLength(0)); // 10
Console.WriteLine(a.GetLength(1)); // 11
Console.WriteLine(a.GetLength(2)); // 12
```

• Методы **GetLowerBound** и **GetUpperBound** возвращают соответственно нижнюю и верхнюю границы массива по заданному измерению (например, если есть одномерный массив на 5 элементов, то нижняя граница будет «0», верхняя – «4»)

• Статический метод **Clear** очищает массив (диапазон массива). При этом ссылочные элементы устанавливаются в **null**, логические – в **false**, типы значений – в «0»

• Статический метод **Resize** изменяет размер массива.

Строки.

Создание строки.

Операции со строками.

Особенности использования строк.

`String` представляет собой неизменяемый упорядоченный набор символов. Он является ссылочным типом и по этой причине строки всегда размещаются в куче и никогда - в стеке.

String считается элементарным типом (компилятор разрешает вставлять литеральные строки непосредственно в исходный код). Компилятор помещает эти литеральные строки в метаданные при компиляции, в процессе выполнения строки загружаются и на них ссылаются переменные типа String.

```
String str1 = new String("Работа со строками");  // Ошибка
String str2 = "Работа со строками";
```

```
void Main()
{
  String str="C# 6.0";
  Point p=new Point();
}
```

```
IL_0000: nop
IL_0001: ldstr "C# 6.0"
IL_0006: stloc.0 // str
IL_0007: newobj UserQuery+Point..ctor
IL_000C: stloc.1 // p
IL_000D: ret
```

Класс String. Особенности применения

```
String str1 = "Работа со строками" + Environment.NewLine + "в C#";
```

Задавать в коде последовательность символов конца строки и перевода каретки (“\n”) напрямую не рекомендуется. Лучше использовать **Environment.NewLine,** который зависит от платформы и возвращает ту строку, которая обеспечивает создание разрыва строк на конкретной платформе

```
String file = "C:\\ Windows\\System32\\Notepad.exe";
String file = @"C:\ Windows\System32\Notepad.exe";
```

```
String strUserName = "Ivan" + " " + "C#";
```

Компилятор выполняет конкатенацию литеральных строк на этапе компиляции, в результате в метаданных модуля оказывается одна строка “Ivan C#". Конкатенация нелитеральных строк с помощью оператора «+» происходит на этапе выполнения. Конкатенацию строк оператором «+» лучше не выполнять т.к. создается много строковых объектов.

Класс String. Особенности применения

```
strUserName.Substring(0,4).ToUpper().EndsWith(" .NET");
```

объект String – неизменяем, созданную однажды строку нельзя сделать длиннее или короче, в ней нельзя изменить ни одного символа. Если выполняется много операций со строками, в куче создается много объектов String

Сравнение строк

```
Boolean  Equals(String  value,  StringComparison  comparisonType) {}
static  Boolean  Equals(String  a,  String  b,
                        StringComparison  comparisonType) {}

static  Int32  Compare (String  strA,  String  strB,
                       StringComparison  comparisonType ) {}
static Int32 Compare(string strA, string strB,
                     Boolean ignoreCase, CultureInfo culture)  {}
```

```
public enum StringComparison
{
    CurrentCulture = 0,
    CurrentCultureIgnoreCase = 1,
    InvariantCulture = 2, //не зависящий от языка и региональных параметров
    InvariantCultureIgnoreCase = 3,
    Ordinal = 4,
    OrdinalIgnoreCase = 5,
 }
```

```
Ordinal  comparison :  'Strasse '  ! =  'StraBe '
Cultural  comparison:  'Strasse '  ==  'StraBe '
```

Интернирование строк (string interning)

Используется для повышения производительности, если в приложении ожидается появление множества одинаковых строковых объектов.

При инициализации CLR создает внутреннюю хэш-таблицу, в которой ключами являются строки, а значениями - ссылки на строковые объекты в управляемой куче.

Регулируется настройками сборки (атрибутом).

```
// отключено интернирование строк
String s1  =  "Ivan";
String s2  =  "Ivan";
Console.WriteLine(Object.ReferenceEquals (s1 ,  s2 )); //   'False'

s1  = String.Intern(s1);
s2  = String.Intern(s2); // Находит в хэш-таблице и возвращает ссылку
Console.WriteLine(Object.ReferenceEquals (s1 ,  s2 ));  //   'True'
```

Класс String. Основные методы

```
//Реверсирование строки
char[] chrArrFull = str.ToCharArray();
Array.Reverse(chrArrFull);
Console.Write(chrArrFull);

//Копируем шесть символов начиная с восьмой позиции
// и помещаем в массив символов (0-индекс в массиве))
str.CopyTo(8, chrArr, 0, 6);

// Вывод подстроки с 11 символа по 23-й (12- размер)
str1.Substring(11, 12)

// Замена подстроки
str4.Replace("учу", "изучаю")

// вставка строки (2- индекс)+ приведение к верхнему регистру
str4.Insert(2, "настройчиво ").ToUpper();

// проверка наличия строки
str4.Contains("настройчиво")
```

Класс String. Основные методы

Член	Тип метода	Описание
Clone	Экземплярный	Возвращает ссылку на тот же самый объект (this). Это правильно, так как объекты String неизменяемы. Этот метод реализует интерфейс Icloneable класса StringCopy
Copy	Статический	Возвращает новую строку — дубликат заданной строки. Используется редко и нужен только для приложений, рассматривающих строки как лексемы. Обычно строки с одинаковым набором символов интернируются в одну строку. Этот метод, напротив, создает новый строковый объект и возвращает иной указатель (ссылку), хотя в строках содержатся одинаковые символы
CopyTo	Экземплярный	Копирует группу символов строки в массив символов
Substring	Экземплярный	Возвращает новую строку, представляющую часть исходной строки
ToString	Экземплярный	Возвращает ссылку на тот же объект (this)

Класс String. Основные методы

```
// добавление '*' слева от строки (всего 30 символов в строке)
str4.PadLeft(30, '*');
// добавление '$' справа от строки (всего 35 символов в строке)
str4.PadRight(35, '$');

// удаление ‘пробелов' в начале  и конце строки
str4.Trim();
// удаление '*' в начале строки
str4.TrimStart(new char[] { '*' });
// удаление '$' с конца строки
str4.TrimEnd("$".ToCharArray());

// формирование массива строк из строки (разделитель "пробел")
string[] strArr = str4.Split(new char[]{' '});
```

Класс String. Основная проблема

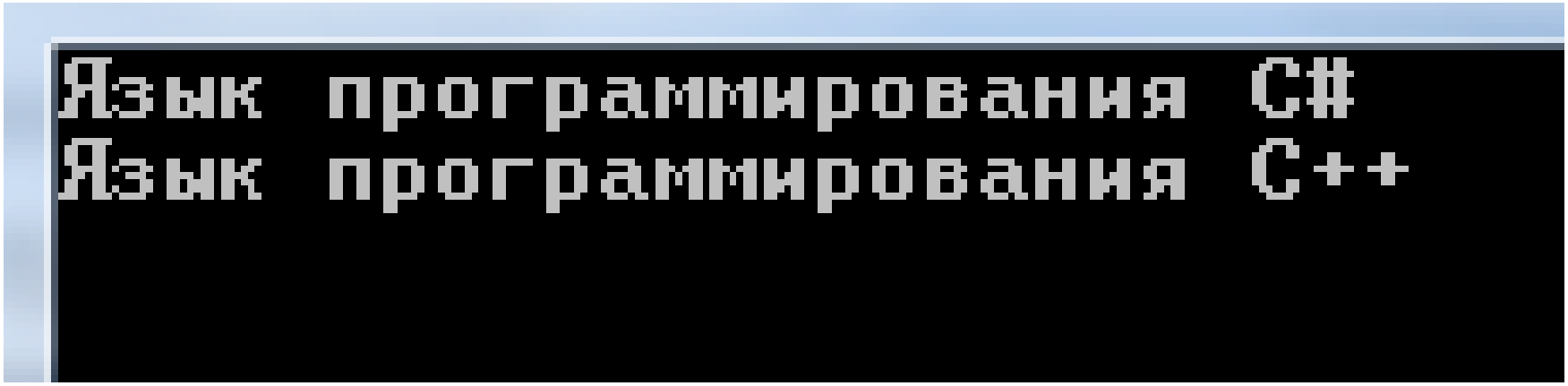
```
static void addNewString()
{
    string s = "This is my stroke";
    s = "This is new stroke";
}
```

```
.method private hidebysig static void  addNewString() cil managed
{
  // Размер кода:       14 (0xe)
  .maxstack  1
  .locals init (string V_0)
  IL_0000:  nop
  IL_0001:  ldstr      "This is my stroke"
  IL_0006:  stloc.0
  IL_0007:  ldstr      "This is new stroke"
  IL_000c:  stloc.0
  IL_000d:  ret
} // end of method Program::addNewString
```

Класс StringBuilder.

Предоставляет изменяемую строку символов.

```
String str = "Язык программирования C#";
str.Replace("#", "++");  // здесь создается еще одна строка
Console.WriteLine(str);

StringBuilder strB = new StringBuilder();
strB.Append("Язык программирования C#");
strB.Replace("#", "++");
Console.WriteLine(strB);
```



Класс StringBuilder. Основные методы.

```
  StringBuilder strB = new StringBuilder("Язык C#. ");
  Console.WriteLine(strB);
// Добавление строки к существующей
  strB.Append("Платформа .NET ");
  Console.WriteLine(strB);
// Добавление строки к существующей + форматирование
  strB.AppendFormat("{0} {1,3:N1}", "версия",4.5);
  Console.WriteLine(strB);
// Вставка строки (6- индекс, место вставки)
  strB.Insert(5,"программирования ");
  Console.WriteLine(strB);
// Замена подстроки
  strB.Replace("4.5", "5.0");
  Console.WriteLine(strB);
// Удаление подстроки
  strB.Remove(5, 17);
  Console.WriteLine(strB);
// Очистка
  strB.Clear();
  Console.WriteLine(strB);
```

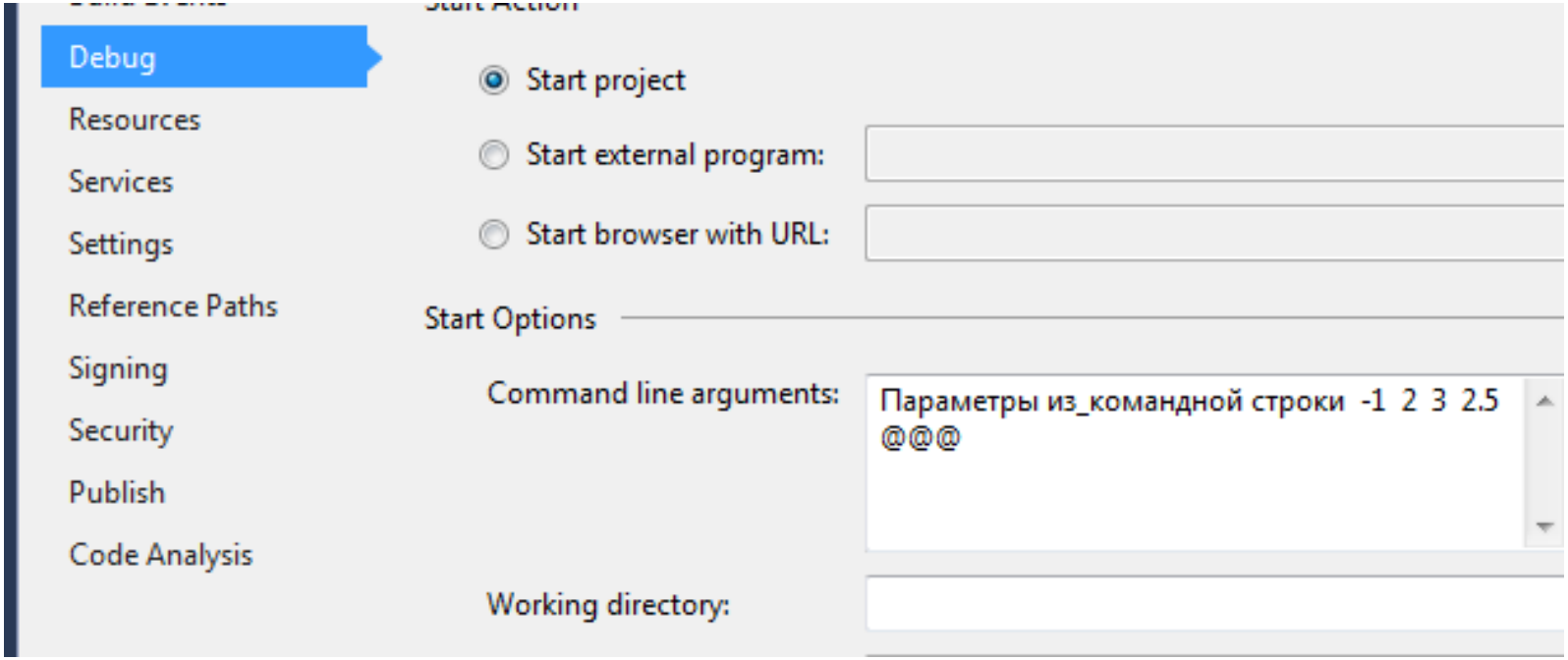
```
Язык C#.
Язык C#. Платформа .NET
Язык C#. Платформа .NET версия 4,5
Язык программирования C#. Платформа .NET версия 4,5
Язык программирования C#. Платформа .NET версия 4,5
Язык C#. Платформа .NET версия 4,5
```

Класс StringBuilder. Основные методы.

```
StringBuilder strS = new StringBuilder(10);
  strS.Append("строка!!!!");
Console.WriteLine("Length: {0}, Capacity: {1}, MaxCapacity: {2:N2}",
strS.Length, strS.Capacity, strS.MaxCapacity);
  strS.Append("строка!!!!");
Console.WriteLine("Length: {0}, Capacity: {1}, MaxCapacity: {2:N2}",
strS.Length, strS.Capacity, strS.MaxCapacity);
  strS.Append("строка!!!!");
Console.WriteLine("Length: {0}, Capacity: {1}, MaxCapacity: {2:N2}",
strS.Length, strS.Capacity, strS.MaxCapacity);
  strS.Append("строка!!!!");
Console.WriteLine("Length: {0}, Capacity: {1}, MaxCapacity: {2:N2}",
strS.Length, strS.Capacity, strS.MaxCapacity);
  strS.Append("строка!!!!");
Console.WriteLine("Length: {0}, Capacity: {1}, MaxCapacity: {2:N2}",
strS.Length, strS.Capacity, strS.MaxCapacity);
```

```
Length: 10, Capacity: 10, MaxCapacity: 2 147 483 647,00
Length: 20, Capacity: 20, MaxCapacity: 2 147 483 647,00
Length: 30, Capacity: 40, MaxCapacity: 2 147 483 647,00
Length: 40, Capacity: 40, MaxCapacity: 2 147 483 647,00
Length: 50, Capacity: 80, MaxCapacity: 2 147 483 647,00
```

Использование аргументов командной строки.




```
static void Main(string[] args)
{
 foreach (string str in args)
 {
 Console.WriteLine("Параметр {0}", str);
 }
 Console.ReadKey();
}
```

```
Параметр Параметры
Параметр из_командной
Параметр строки
Параметр -1
Параметр 2
Параметр 3
Параметр 2.5
Параметр @@@
```

Класс Random. Генерация случайных чисел.

Генерация псевдослучайной последовательности чисел.

```
Random rnd = new Random();
rnd.Next();                     // случайное число
rnd.Next(50);                   // случайное число от 0 до 50
rnd.Next(10, 100);              // случайное число от 10 до 100
rnd.NextDouble();               // случайное число от 0.0 до 1.0
byte[] t = new byte[10];
rnd.NextBytes(t);               // заполняет массив случайными числами
```

```
// заполнение массива случайными числами от -10 до 10
int[] arrNums = new int[5];
for (int i = 0; i < arrNums.Length; i++)
{
  arrNums[i] = rnd.Next(-10,10);
}
```

Исходный код:
http://referencesource.microsoft.com/#mscorlib/system/random.cs,bb77e610694e64ca