**Arrays**

**Declaration:**

* int[] myIntArray = new int[];
* int[] myIntArrat = {1, 2, 3, 4, 5};
* char[] myCharArray = “string”.ToCharArray();
* int[] numbers = Enumerable.Range(0, 1000000).ToArray();
* “1 56 20 3”.Split(‘ ‘);

**Output:**

* With for or foreach
* Console.WriteLine(string.Join(“, “, array));
* Console.WriteLine(string.Join(Environment.NewLine, array));

**Reversing:**

* int[] array = new int[] {1, 2, 3, 4, 5};

int length = array.Length;

int[] reversed = new int[length];

for (int index = 0; index < length; index++)

{

reversed[length-index-1] = array[index];

}

* for ( int i = 0; i < symbols.Lenght / 2; i++)

{

char store = symbols[i];

symbols[i] = symbols[symbols.Lenght – i – 1];

symbols[symbols.Lenght – i – 1] = store;

}

* using System.Linq

char[] symbols = “string”.Reverse().ToArray();

**Check Symmetric:**

Bool isSymmetric = true;

For ( int i = 0; i < array.Lenght / 2; i++)

{

If(array[i] != array(n – i – 1])

{

isSymmetric = false;

}

}

**Copying Arrays:**

* **Array.Copy(array, copiedArray, array.Lenght);**

**Lists**

**Declaration:**

using System.Collections.Generic;

List<int> intList = new List<int>();

**Method and Properties:**

* **list.Add();**
* **list.Remove();**
* **list.Insert();**
* **list.Count**
* **list.Capacity**
* **list.Clear()**
* **List.TrimExcess() – сетва капацитета на точния брой елементи, който има в момента**

**Binary search algorithm**

Обикновен начин:

Array.Sort(array);

int left = 0;

int right = N;

int result = -1;

while(left <= right)

{

int middle = (left + right) / 2;

if (array[middle] == X)

{

result = middle;

break;

}

else if (array[middle] > X)

{

right = middle - 1;

}

else

{

left = middle + 1;

}

}

if (result != -1)

{

Console.WriteLine(result);

}

else

{

Console.WriteLine("-1");

}

**С обновяване на отговора:**

int ans = arraySize;

int left = 0, right = arraySize - 1;

**while** (left <= right) {

int mid = (left + right) / 2;

**if** (array[mid] < x)

left = mid + 1;

**else**

right = mid - 1, ans = mid;

**С използване на една от границите:**

**Връща най-малкия индекс**

int left = 0;

int right = N - 1;

int result = 0;

while (left <= right)

{

int mid = (left + right) / 2;

if (array[mid] < X)

{

left = mid + 1;

}

else

{

right = mid - 1;

}

result = right + 1;

}

Console.WriteLine(result);

**Връща най-големия индекс**

int left = 0;

int right = N - 1;

int result = 0;

while (left <= right)

{

int mid = (left + right) / 2;

if (array[mid] > X)

{

right = mid - 1;

}

else

{

left = mid + 1;

}

result = left - 1;

}

Console.WriteLine(result);

**Multidimensional Arrays**

**Declaration:**

* int[ , ] array = { {1, 2}, {3, 4}};
* int[ , ] array = new int[3, 4];

for(int row = 0; row < array.GetLenght(0); row++)

for(int col = 0; col < array.GetLenght(1); col++)

array[row, col] = row + col;

* int[,] matrix =

{

{ 1, 2, 3, 4 }, // row 0 values

{ 5, 6, 7, 8 }, // row 1 values

};

**Printing:**

* for (int row = 0; row < matrix.GetLength(0); row++)

{

for (int col = 0; col < matrix.GetLength(1); col++)

{

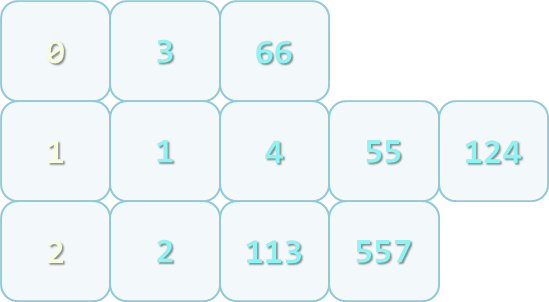
Console.Write("{0, 4}", matrix[row, col]);

}

Console.WriteLine();

}

**Jagget Arrays**

****

**Declaration:**

* int[][] jagged = new int[3][];

jagged[0] = new int[3];

jagged[1] = new int[2];

jagged[2] = new int[5];

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

for (int i = 0; i < n; i++)

{

jaggedArray[i] = new int[i];

}

**Example of Jagget Arrays:**

**(for result look the picture)**

int[] numbers = {0, 1, 4, 113, 55, 3, 1, 2, 66, 557, 124, 2};

int[] sizes = new int[3];

int[] offsets = new int[3];

foreach (var number in numbers)

{

int remainder = number % 3;

sizes[remainder]++;

}

int[][] numbersByRemainder = new int[3][]

{

new int[sizes[0]],

new int[sizes[1]],

new int[sizes[2]]

};

foreach (var number in numbers)

{

int remainder = number % 3;

int index = offsets[remainder];

numbersByRemainder[remainder][index] = number;

offsets[remainder]++;

}

**The Array Class**

**Methods of Array:**

* GetEnumerator() – returns IEnumerator for the array elements
* BinarySearch(…) – searches for a given element into a sorted array (uses binary search)
* IndexOf(…) – searches for a given element and returns the index of the first occurrence (if any)
* LastIndexOf(…) – searches for a given element and returns the last occurrence index
* Copy(src, dest, len) – copies array elements; has many overloads
* Reverse(…) – inverts the arrays elements upside down
* Clear(…) – assigns value 0 (null) for each elements
* CreateInstance(…) – creates an array
  + Accepts as parameters the number of dimensions, start index and number of elements
* Implements ICloneable, IList, ICollection and IEnumerable interfaces

Sorting in .NET is usually done with System.Array.Sort()

* Sort(Array) – sorts array elements
  + Elements should implement IComparable
* Sort(Array, IComparer) – sorts array elements by given external IComparer
* Sort(Array, Comparison<T>) – sorts array elements by given comparison operation
  + Can be used with lambda expression

# Advices for Working with Arrays

* When a given method returns an array, it should return an empty array (array with 0 elements) instead of null
* Arrays are passed by reference
  + To be sure that given method will not change the passed array, pass a copy of it
* Clone() returns shallow copy of the array
  + You should implement your own deep clone when working with **custom** reference types

**A method in C# can take variable number of parameters by specifying the params keyword:**

**static long CalcSum(params int[] elements)**

**{**

**long sum = 0;**

**foreach (int element in elements)**

**sum += element;**

**return sum;**

**}**

**static void Main()**

**{**

**Console.WriteLine(CalcSum(2, 5));**

**Console.WriteLine(CalcSum(4, 0, -2, 12));**

**Console.WriteLine(CalcSum());**

**}**

**C# supports optional parameters since v4.0 with default values assigned at their declaration:**

**static void PrintNumbers(int start = 0, int end = 100)**

**{**

**for (int i = start; i <= end; i++)**

**{**

**Console.Write("{0} ", i);**

**}**

**}**

**Method with reference**

**Static int ChangeIntByRefernece(ref int number)**

**{**

**number = 10;**

**}**

**Static void Main()**

**{**

**int a = 5;**

**ChangeIntByReference(ref a)**

**Console.WriteLine(a); // output 10**

**}**

**Strings and Text Processing**

**Strings are immutable (read only)**

**Manipulating Strings**

* **Comparing**
* **Case-insensitive**

**int result = string.Compare(str1, str2, true);**

* **Case-sensitive**

**string.Compare(str1, str2, false);**

* **Can use and == or Equals()**
* **Concatenating**
* **Using + or the += operators**
* **Any object can be appendet to a string**
* **Searching**
* **Case sensitive**
* **First occurrence**

**string.IndexOf(string str)**

* **First occurrence starting at given position**

**string.IndexOf(string str, int startIndex)**

* **And LastIndexOf**
* **Extracting Substring**
* **string.Substring(int startIndex, int lenght)**
* **string.Substring(int startIndex)**
* **Splittng**
* **string.Split(‘ ‘);**
* **string.Split(new[] {‘ ‘, ‘-‘,..}, StringSplitOptions.RemoveEmptyEntries);**
* **Vrushta masiv ot charove**
* **Replace**
* **string.Replace(string, string)**
* **result is new string(strings are immutable)**
* **Remove**
* **string.Remove( index, length)**
* **ToLower**
* **string.ToLower()**
* **ToUpper**
* **string.ToUpper()**
* **Trim (TrimStart/TrimEnd)**
* **stirng.Trim()**
* **string.Trim(chars)**
* **string.TrimStart() and string.TrimEnd()**
* **PadLeft**
* **PadRight**
* **ToCharArray()**

**How are Strings Constructed?**

* **String are immutable!**
* **Concat(), Replace(), Trim()**
* **Do not use + for strings in a loop!**

**StringBuilder**

* **StringBuilder keeps buffer memory, allocated in advance**
* **Using System.Text**
* **StringBuilder(int capacity)**
* **string.Capacity**
* **this[int index]**
* **string.Lengh**
* **StringBuilder.Append()**
* **StringBuilder.AppendLine()**
* **StringBuilder.Remove(int start index, int length)**
* **StringBuilder.Instert(int index, int str)**
* **StringBuilder.Replace(string oldStr, string newStr)**
* **StringBuilder.ToString**

**Formatting Strings**

* **ToString(formatString) (example ToString(“X”))**

**Some formatting string for numbers:**

* **D – number (for integer types)**
* **C – currency (according to current culture)**
* **E – number in exponential notation**
* **P – percentage**
* **X – hexadecimal number**
* **F – fixed point (for real numbers)**
* **String.Format(template, params…);**

**double d = 0.375;**

**s = String.Format("{0,10:F5}", d);**

**// s = " 0,37500"**

* **Formatting Dates**
* **d, dd – day (with/without leading zero)**
* **M, MM – month**
* **yy, yyyy – year (2 or 4 digits)**
* **h, HH, m, mm, s, ss – hour, minute, second**
* **Example**

**DateTime now = DateTime.Now;**

**Console.WriteLine ("Now is {0:d.MM.yyyy HH:mm:ss}", now);**

**// Now is 31.11.2009 11:30:32**

* **Cultures**
* **Printing the current culture:**

**Console.WriteLine(System.Threading.Thread.CurrentThread.CurrentCulture);**

* **Changing the current culture:**

**System.Threading.Thread.CurrentThread.CurrentCulture = new CultureInfo("en-CA");**

* **Culture-sesitive ToString():**

**CultureInfo culture = new CultureInfo("fr-CA");**

**string s = number.ToString("C", culture); // 42,00 $**

* **Parsing a real number using "." as separator:**

**string str = "3.14";**

**Thread.CurrentThread.CurrentCulture = CultureInfo.InvariantCulture;**

**float f = float.Parse(str); // f = 3.14**

* **Parsing a date in specific format:**

**string dateStr = "25.07.2011";**

**DateTime date = DateTime.ParseExact(dateStr, "dd.MM.yyyy", CultureInfo.InvariantCulture);**

**Collections**

**System.Collections.Generic**

* **List< >**
* **.RemoveAll(ламбда)**
* **.FindAll(ламбда)**
* **HashSet< > - запазва уникални стойности. Няма индексация. Вадят се с foreach.**
* **SortedSet - сортиран HashSet**
* **Dictionary< , > - работи като масив, но може да приема различни индексации**
* **.ContainsKey – проверява дали даден ключ се съдържа вече**
* **.Value – изкарва стойноста**
* **.Key – изкарва ключа**
* **SortedDictionary – сортира по ключ**
* **Stack< > - първи влязъл последен илиза**
* **.Push – слага в стека**
* **.Pop – вади от стека**
* **.Peek – проверява какво има отгоре(не го вади)**
* **Queue< > - първи влязъл първи излиза**
* **.Enqueue**
* **Dequeue**
* **.Peek**

**LINQ**

* **Where(ламбда) – връща всички елементи, които отговарят на условието**
* **FirstOrDefault() – връща първия намерен елемент, ако няма елементи връще default стойност на елементите**
* **First() – ако няма намерен елемент, задачата гърми**
* **Last()**
* **LastOrDefault(може и с ламбда да търсят)**
* **Select() – променя елемента според някакво услвие**
* **.Select(t => t.Lenght)**
* **.Select(t => t + 6)**
* **.Select(t => int.Parse(t))**
* **OrderBy()/ThenBy – подрежда/сортира**

**.Orderby(p => p.Lenght)**

**.ThenBy(p => p);**

* **OrderByDescending – подрежда в обратен ред**
* **Any() – проверява дали поне един от елементите отговоря на дадено условие**
* **All() – проверява дали всички елементи отговарят на условието – връща bool**
* **ToArray()/ToList()/AsEnumerable()**
* **Reverse()**
* **Average()**
* **Count()**
* **Max()**
* **Min()**
* **Sum()**

**Прилагаме първо това, което отнема по-малко ресурси.**

**Можем също:**

**foreach (var number in list.OrderByDescending(n => n.Length).Select(int.Parse))**

**{**

**Console.WriteLine(number \* 1000);**

**}**

**Exceptions**

**try**

**{**

**// Do some work that can raise an exception**

**}**

**catch – прихваща всички възможни грешки //**

**catch (Exception) – прихваща всички грешки от c#//**

**catch(SomeException) – прихваща различни грешки от C#//**

**catch(SomeException ex) – можем да я достъпваме като обект (име на обекта – ex)**

**{**

**// Handle the caught exception**

**}**

**finally**

**{**

**// This block will always execute**

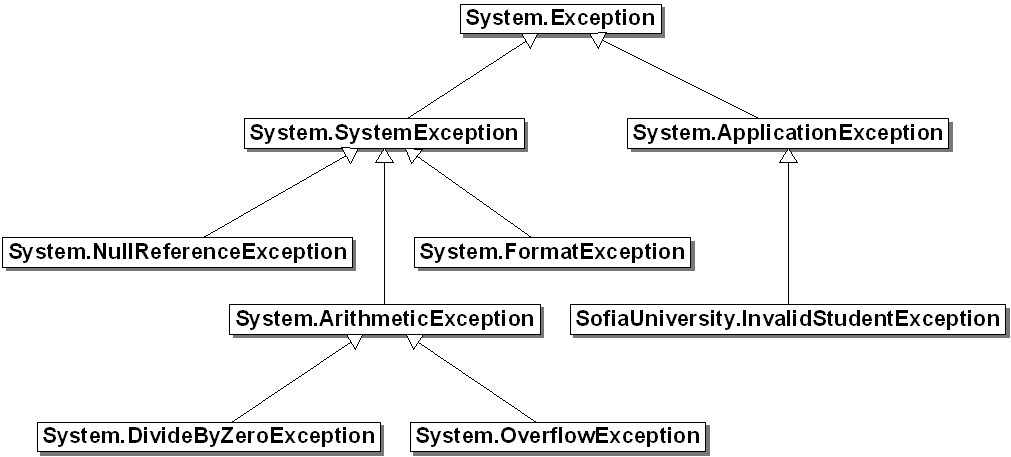
**}**

* **Exception Properties**
* **Message – text description of the exception**
* **StackTrace – the snapshot of the stack at the moment of exception throwing (main – method1 – method2 – read – parse)**
* **InnerException – exception that caused the current exception (if any). The StackTrace property is extremely useful when identifying the reason caused the exception**

**File names and line numbers are accessible only if the compilation was in Debug mode**

**When compiled in Release mode, the information in the property StackTrace is quite different**

* **Exceptions Hierarchy**

****

* **Types of Exceptions**
* **System.Exception**
* **System.ArgumentException**
* **System.NullReferenceException**
* **System.OutOfMemoryException**
* **System.StackOverflowException**
* **IndexOutOfRangeException**
* **FormatException**
* **ArighmeticException**
* **Throwing Exceptions**

**try**

**{**

**int.Parse(str);**

**}**

**catch (FormatException fe)**

**{**

**throw new ArgumentException("Invalid number", fe);**

**}**

* **Re-Throwing Exceptions**

**try**

**{**

**int.Parse(str);**

**}**

**catch (FormatException fe)**

**{**

**Console.WriteLine("Parse failed!");**

**throw fe; // Re-throw the caught exception**

**}**

**catch (FormatException)**

**{**

**throw; // Re-throws the last caught exception**

**}**

* **Choosing the Exception Type**
* **When an invalid parameter is passed to a method:**
* **ArgumentException, ArgumentNullException, ArgumentOutOfRangeException**
* **When requested operation is not supported**
* **NotSupportedException**
* **When a method is still not implemented**
* **NotImplementedException**
* **If no suitable standard exception class is available**
* **Create own exception class (inherit System.Exception)**

**Започваме да прихващаме първо по-конкретните грешки!**