C# programming language
One-dimensional arrays

FIIT, Semester 2

Programming Languages

Mikhalkovich S.S.
Abramyan A.V.

One-dimensional arrays

```
// How to allocate memory to an array
int[] a = new int[5];


// How to fill an array
int[] b = { 1, 2, 3, 4, 5 };


// ArrGen functions are absent!
for (var i = 0; i < a.Length; i++)
    a[i] = i * i;

// Output of an array
// The .Print extension method is absent
foreach (var x in a)
    Console.Write($"{x} ");


// Return an allocated memory
a = null;
```

How to fill an array

```
// 1 3 5 7 9 11 13 15 17 19 - arithmetic
progression
int[] a = new int[10];
a[0] = 1;
for (var i = 1; i < a.Length; i++)
    a[i] = a[i-1] + 2;
```

```
// 1 2 4 8 16 32 64 128 256 512 - geometric
progression
int[] a = new int[10];
a[0] = 1;
for (var i = 1; i < a.Length; i++)
    a[i] = a[i-1] * 2;
```

```
// 1 1 2 3 5 8 13 21 34 55 – Fibonacci sequence
int[] a = new int[10];
a[0] = 1;
a[1] = 1;
for (var i = 2; i < a.Length; i++)
    a[i] = a[i-1] + a[i-2];
```

Some algorithms

```
int[] a = new int[10] {1,3,5,7,9,11,13,15,17,19};


// sum of array elements
var sum = 0;
for (var i = 0; i < a.Length; i++)
    sum += a[i];


// Another algorithm
var sum = 0;
foreach (var x in a)
    sum += x;


// Transformation
for (var i = 0; i < a.Length; i++)
    a[i] = a[i] * a[i];
```

Some algorithms

```
// Min/Max
var min = a[0];
for (int i = 1; i < a.Length; i++)
    if (a[i] < min)
        min = a[i];
```

```
// MinIndex/MaxIndex
var min = a[0];
var ind = 0;
for (int i = 1; i < a.Length; i++)
    if (a[i] < min)
    {
        min = a[i];
        ind = i;
    }
```

```
// MinIndex/MaxIndex - 2
var min = int.MaxValue;
var ind = -1;
for (int i = 0; i < a.Length; i++)
    if (a[i] < min)
    {
        min = a[i];
        ind = i;
    }
```

Some algorithms

```
// MinIndex/MaxIndex
var min = a[0];
var ind = 0;
for (int i = 1; i < a.Length; i++)
    if (a[i] < min)
    {
        min = a[i];
        ind = i;
    }
```

```
// MinIndex/MaxIndex - 2
var min = int.MaxValue;
var ind = -1;
for (int i = 0; i < a.Length; i++)
    if (a[i] < min)
    {
        min = a[i];
        ind = i;
    }
```

```
// Conditional Min/Max
var min = int.MaxValue;
var ind = -1;
for (int i = 0; i < a.Length; i++)
    if (a[i] < min && a[i] > 100)
    {
        min = a[i];
        ind = i;
    }
if (min == int.MaxValue)
    Console.WriteLine("no elements");
else Console.WriteLine(min);
```

Some algorithms

```
// Shift left

for (int i = 1; i < a.Length; i++)
    a[i - 1] = a[i];
a[a.Length - 1] = 0;
```

```
// Circular Shift left
var x = a[0];
for (int i = 1; i < a.Length; i++)
    a[i - 1] = a[i];
a[a.Length - 1] = x;
```

```
// Shift right

for (int i = a.Length - 1; i > 0; i--)
    a[i] = a[i-1];
a[0] = 0;
```

```
// Circular Shift right
var x = a[a.Length - 1];
for (int i = a.Length - 1; i > 0; i--)
    a[i] = a[i-1];
a[0] = x;
```

Sorting

```
// Bubble sort – Θ(n^2)

for (var i = 0; i < a.Length - 2; i++)
    for (var j = a.Length - 1; j >= i + 1; j--)
        if (a[j] < a[j - 1])
                Swap(ref a[j], ref a[j - 1]);
```

In C# the Swap method is absent (!)

```
static void Swap<T>(ref T a, ref T b)
    {
        var x = a;
        a = b;
        b = x;
    }
```

```
// Standard sort – Θ(n · log n)

System.Array.Sort(a);

a = a.OrderBy(x => x).ToArray();

a = a.OrderByDescending(x => x).ToArray();
```

Extension method Print

```
// How to write your own .Print method

public static class MyFuncs
{
    public static void Print<T>(this T[] a)
    {
        foreach (var x in a)
            Console.Write($"{x} ");
    }
}

class Program
{
    static void Main(string[] args)
    {
        int[] b = { 1, 2, 3, 4, 5 };
        b.Print();
    }
}
```

In static class

The "this" keyword means that we extend this type by this method

The generation of one-dimensional arrays

```
enum Color { Red, Green, Yellow, Blue, Black }

public static class MyFuncs
{
    public static T[] ArrGen<T>(int n, Func<int,T> fun)
    {
        var a = new T[n];
        for (var i = 0; i < a.Length; i++)
            a[i] = fun(i);
        return a;
    }
}

class Program
{
    static void Main(string[] args)
    {
       int[] a = MyFuncs.ArrGen(10, i => i * i);
       bool[] b = MyFuncs.ArrGen(10, i => i % 2 == 0);
       double[] c = MyFuncs.ArrGen(10, i => Math.Sqrt(i));
       char[] d = MyFuncs.ArrGen(10, i => (char)(i + 48));
       string[] e = MyFuncs.ArrGen(10, i => "!" + i + "!");
       int numberOfElements = Enum.GetNames(typeof(Color)).Length;
       Color[] f = MyFuncs.ArrGen(10, i => (Color)(i % numberOfElements));
     }
}
```

Type of function
int -> T

Lambda-expression

The End