1 Arrays

1.1 Motivation

Arrays are collection, or grouping, of values held in a single place. They can store multiple values of the same datatype, and are useful, for instance,

- When we want to store a collection of related values,
- When we don’t know in advance how many variables we need.

1.2 Declaration and Initialization of Arrays

Declaration and assignment

```java
int[] myArray;
myArray = new int[3]; // 3 is the size declarator
// We can now store 3 ints in this array,
// at index 0, 1 and 2
myArray[0] = 10; // 0 is the subscript, or index
myArray[1] = 20;
myArray[2] = 30;

// the following would give an error:
//myArray[3] = 40;
// Unhandled Exception: System.IndexOutOfRangeException: Index was outside the bounds of
// the array at Program.Main()
// "Array bound checking": happen at runtime.
```

As usual, we can combine declaration and assignment on one line:

```java
int[] myArray = new int[3];
```

We can even initialize and give values on one line:

```java
int[] myArray = new int[3] { 10, 20, 30 };
```

And that statement can be rewritten as any of the following:

```java
int[] myArray = new int[] { 10, 20, 30 };
t[] myArray = new[] { 10, 20, 30 };
t[] myArray = { 10, 20, 30 };
```

But, we should be careful, the following would cause an error:
int[] myArray = new int[5];
myArray = {1, 2, 3, 4, 5}; // ERROR

If we use the shorter notation, we have to give the values at initialization, we cannot re-use this notation once the array was created.

Other datatype, and even objects, can be stored in arrays:

string[] myArray = {"Bob", "Mom", "Train", "Console"};
Rectangle[] arrayOfRectangle = new Rectangle[5];

1.3 Custom Size and Values

Console.WriteLine("What is size of the array that you want?");
int size = int.Parse(Console.ReadLine());
int[] customArray = new int[size];

How can we fill it with values, since we do not know its size? Using iteration!

int counter = 0;
while (counter < size)
{
    Console.WriteLine($"Enter the {counter + 1}th value");
    customArray[counter] = int.Parse(Console.ReadLine());
    counter++;
}

We can use length, a property of our array. That is, the integer value myArray.Length is the length (= size) of the array, we can access it directly.

To display an array, we need to iterate as well (this time using the Length property):

int counter2 = 0;
while (counter2 < customArray.Length)
{
    Console.WriteLine($"{counter2}: {customArray[counter2]}");
    counter2++;
}

1.4 Changing the Size

Array is actually a class, and it comes with methods!

Array.Resize(ref myArray, 4);
myArray[3] = 40;
Array.Resize(ref myArray, 2);

Resize shrinks (and content is lost) and extends (and store the default value, i.e., 0 for int, etc.)!
2 For Loops

2.1 for Loops

```csharp
1 int i = 0;
2 while (i <= 5)
3 {
4     Console.WriteLine(i + " ");
5     i++;
6 }

1 int j = 0;
2 do
3 {
4     Console.WriteLine(j + " ");
5     j++;
6 } while (j <= 5);

1 int k = 0;
2 for (k = 0; k <= 5; k++)
3 {
4     Console.WriteLine(k + "");
5 }

1 for (int l = 0; l <= 5; l++)
2 {
3     Console.WriteLine(l + "");
4 }
```

Structure: initialization / condition / update

2.2 Ways Things Can Go Wrong

Don’t:

- Increment the counter in the body of the for loop!
- Assume that a variable declared in the header of a for loop will be accessible in the rest of the code. / Use `for` if you want to use the counter for anything else.
- Declare the variable twice.

2.3 For loops With Arrays

`for` loops actually go very well with arrays:

```csharp
1 for (int i = 0; i < size; i++)
2 {
3     Console.WriteLine($"Enter the \{(i + 1)\}th value");
4     customArray[i] = int.Parse(Console.ReadLine());
5 }
```
Remember that we can use the Length property of our array. The previous code could become (only the first line changed):

```csharp
for (int i = 0; i < customArray.Length; i++)
{
    Console.WriteLine($"Enter the {i + 1}th value");
    customArray[i] = int.Parse(Console.ReadLine());
}
```

### 2.4 Nested Loops

Of course, exactly as we could nest if statements, we can nest looping structures!

```csharp
for (int o = 0; o < 11; o++)
{
    for (int p = 0; p < 11; p++)
    {
        Console.WriteLine($"{o} \times {p} = {o * p} \t ");
        Console.Write();
    }
}
```

### 2.5 Mixing Control Flows

And we can use if statements in the body of for loops:

```csharp
for (int m = 0; m < 10; m++)
{
    if (m % 2 == 0) Console.WriteLine("This is my turn.");
    else Console.WriteLine("This is your turn.");
}
```

### 2.6 Iterations

There is another, close, structure that allows to iterate over the elements of an array, but can only access them, not change their values (they are “read only”).

```csharp
for (int i = 0; i < myArray.Length; i++)
    Console.Write(myArray[i] + " ");
foreach (int i in myArray) // "Read only"
    Console.Write(i + " ");
```

Difference is w.r.t. to modifying the array “read Vs write”. Having i = 2 in the foreach would cause an error!

That last structure is given for the sake of completeness, but it’s ok if you’d rather not use it.