CSCI 1301 - Lab 23

Clément Aubert

April 10, 2018

Part I - First Array Maniplation

Write a program that

- declares an array `myArray` of `int` of size 5,
- intitializes `myArray` with the values 1, 2, 3, 4 and 5,
- displays the content of `myArray`, using `foreach`.

Now, let us write *incorrect* statements. Add the following statements one by one to your program, observe how C# react (that is, try to compile and execute after you added one, then remove it), and answer the following questions.

```
myArray = { 1, 2 ,3, 4, 5};
Console.WriteLine (myArray[5]);
myArray[5] = 12;
Console.WriteLine(myArray);
```

- One of this statement is not "incorrect" in the sense that it won't prevent your program from executing, but it is not doing what you could have expected: which one?
- Can you read and understand the errors messages you obtained for the others?

Part II - Manipulating Two Arrays at the Same Time

Write a program that

- declares two array of `int` of size `8`,
- intitializes the values of the first array with random numbers between 0 and 9,
- intitializes the values of second array with random numbers between 0 and 9,
- display the content of the two arrays, and, for each index, "W" if the value in the firt array is greater than the value of the second array, "T" if they are equal, and "L" if it is lesser.

An example of execution of this program would display:

```
0       8       L
5       3       W
```

```
3       3       T
1       2       L
3       1       W
9       0       W
9       0       W
1       5       L
```

Pushing Further (Optional)

For this lab, we will introduce only one notion, but a crucial one: the difference between value and reference types. This topic is introduced in your textbook ("Value Types vs. Reference Types", Chapter 7.17), and will be studied in CSCI 1302. You can have a look at https://docs.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/data-types/value-types-and-reference-types: this page is not so complex, a short (and excellent) read.

Let us motivate why this notion is so critical with an example:

```
int[] arrayA = { 1, 2, 3, 4, 5 };  // Let me declare a simple array of integer

// I'd like to make a copy of that array. Let me try the following:
int[] arrayCopyWrong = arrayA;

foreach (int i in arrayCopyWrong)
    Console.Write(i + " ");

Console.WriteLine();

// It seems to be working! Except that if we change a value in our copy:
arrayCopyWrong[0] = 6;

// It also changes the value in our original array!
foreach (int i in arrayA)
    Console.Write(i + " ");

Console.WriteLine();
```

What happened is that we copied the reference to the array, and not the array itself. We now have two ways of accessing our array, using `arrayA` or `arrayCopyWrong`, but still only one array.

To perform a copy of the array, we need to do something like the following:

```
int[] arrayB = { 1, 2, 3, 4, 5 };
int[] arrayCopyRight = new int[5];

// We copy each value, one by one:
```

```
for(int i = 0 ; i < arrayB.length; i++)
    arrayCopyRight[i] = arrayB[i];

// If we  change a value in our copy:
arrayCopyRight[0] = 6;

// It changes the value only in that copy:
foreach (int i in arrayB)
    Console.Write(i + " ");

Console.WriteLine();

foreach (int i in arrayCopyRight)
    Console.Write(i + " ");
```

`Array` is actually a class (cf. https://msdn.microsoft.com/en-us/library/system.array(v=vs.110).aspx), and as such provides several methods. For `x` an `int`, `array1` and `array2` two arrays containing the same type of values and of size at least `x`, you can copy the first `x` values of `array1` into `array2` using `Array.Copy(array1, array2, x);`. Try to use this method with the previous example.