Arrays Manipulation

https://csci-1301.github.io/about#authors

June 21, 2021 (04:15:30 PM)

Contents

1 Array Manipulation Practice (Advanced)

1.1 Set-Up

For this exercise:

- Download and extract the ArrayManiplutaion project[1]. It contains two .cs files, `ArrayLib.cs` and `Program.cs`.
- Compile and execute it.
- Observe `Program.cs`: this is a *test program* that you **should not modify**. It will be useful to test the methods that you will be writing in the `ArrayLib.cs` class file. For each method, this program displays the expected value, and what is actually returned. As you can see, only the `Display` method seems to be always correct.
- Now, read `ArrayLib.cs`. Every method used by `Program.cs` has a header, but all the bodies are returning "default" values or do nothing, with the exception of `Display`. This last method was written for you.

1.2 Your goal

Your goal is to write the body of the methods in the `ArrayLib` class. You should not change their headers. Modify only their bodies, so that they return the "right" values, according to their description (in comments after their headers) and the test given in `Program.cs`. You can change their order within `ArrayLib`, and you can write them in any order. Some of them are actually easier to write, and they are not the first ones: can you find a method that seems easy enough to start your project?

If you have the time and interest, have a look at the challenges offered at the end of the `ArrayLib.cs` file. You can find a possible solution in this archive[2].

[1] ArrayManipulation.zip
[2] ArrayManipulationSolution.zip

2 Pushing Further (Optional)

Here, we will explore the difference between value and reference types. We have mentioned this concept in class several times, but we have not used it in the programs we have written so far. Since arrays are reference types, however, it is now more important for you to understand how reference types work.

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

```
int[] arrayA = { 1, 2, 3, 4, 5 };  // Declare a simple array of integers

// 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();
```

Try running this program yourself to see what happens. The problem is that when we wrote the assignment statement `int[] arrayCopyWrong = arrayA`, we copied the *reference* to the array, but not the array itself. We now have two ways of accessing our array, using `arrayA` or `arrayCopyWrong`, but still only one array.

To correctly copy the array, we need to do something like the following:

```
int[] arrayB = { 1, 2, 3, 4, 5 };
// Create a new array object and assign it to a new reference variable
int[] arrayCopyRight = new int[arrayB.Length];

// Copy each value in the array, 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 + " ");

Console.WriteLine();
```

Try running this program. Can you see the difference?

`Array` is actually a class (documented at https://msdn.microsoft.com/en-us/library/system.array(v=vs.110).aspx), and as such provides several methods. If you have two arrays, `array1` and `array2` 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 using this method with the previous example to create a copy of `arrayB`.