

Introduction to C# - Lesson 2 exercises

Note: for all these exercises, ignore input validation unless otherwise directed. Assume the user enters a value in the format the program expects. For example, if the program expects the user to enter a number, don't worry about validating whether the input is a number or not. When testing your program, simply enter a number.

Task 2.1 – Word counter

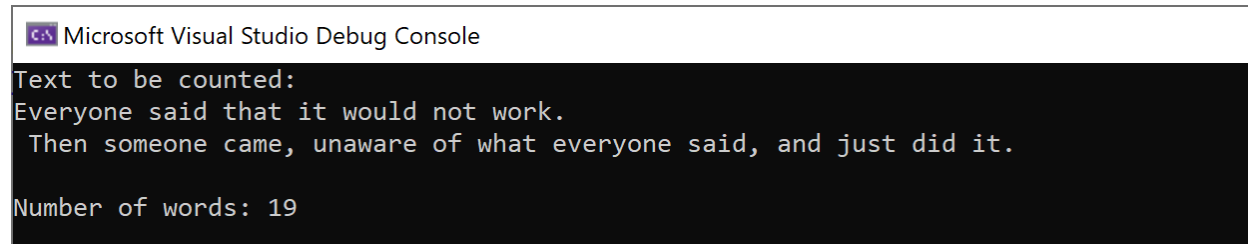
Count the words in a text and print out the text as well as the number of words.

Example string:

"Everyone said that it would not work.\n Then someone came, unaware of what everyone said, and just did it."

TIP: Look into the `Split()` method for strings for this task:

Example of output:

A screenshot of the Microsoft Visual Studio Debug Console. The title bar at the top says "Microsoft Visual Studio Debug Console". The console output shows the text "Text to be counted:" followed by two lines of text: "Everyone said that it would not work." and "Then someone came, unaware of what everyone said, and just did it." Below this, it shows "Number of words: 19".

```
Text to be counted:
Everyone said that it would not work.
Then someone came, unaware of what everyone said, and just did it.

Number of words: 19
```

Extra: In today's lecture we learned about files and reading from files. If you want to, see if you manage to count all words from a text file as well.

Task 2.2 – Creating methods (functions)

Create a method, `ExtractEvenNumbersFromArray`, that takes an array of integer numbers. The method should return a list that only contains the even numbers from the array. To do this, you will need some way of checking if a number is odd or even, for example through the use of the modulus ('%') operator.

Test the method. For example, try the following:

```
int[] numberArray = {9, 2, 2, 0, 9, 3, 9, 4};
Console.WriteLine("Array contains:");
foreach (int element in numberArray) {
    Console.Write(element + ", ");
}
```

```

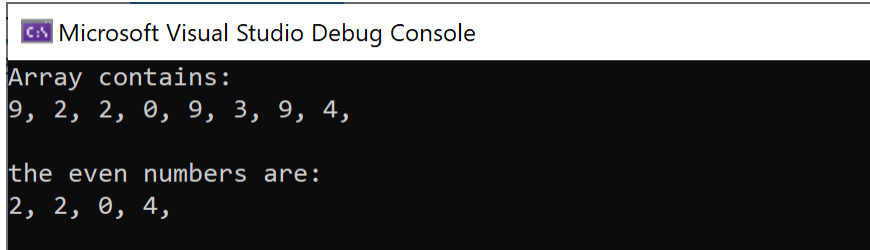
Console.WriteLine("\n");

List<int> evenNumberList = ExtractEvenNumbersFromArray(numberArray);

Console.WriteLine("the even numbers are:");
foreach (int element in evenNumberList) {
    Console.Write(element + ", ");
}
Console.WriteLine();

```

Here is the result of the example Main, above:



```

Microsoft Visual Studio Debug Console
Array contains:
9, 2, 2, 0, 9, 3, 9, 4,
the even numbers are:
2, 2, 0, 4,

```

Extra: When writing all contents of an array or a list to the console or to a string, there is a more elegant way than with a foreach loop: `string.Join()`. If you want to, improve on the example code above by using `Join`!

Task 2.3 – Calculator

Implement four "calculator" methods for the basic math operations: addition, subtraction, multiplication, and division. The methods should each take 2 integers as parameters and give the result of the mathematic operation as their return value.

(Note: When dividing, C# will automatically drop decimals. This is OK.)

Test the calculator methods in Main. For example, try the following:

```

int result = 0;

result = Add(result, 8);
Console.WriteLine(result);

result = Multiply(result, 2);
Console.WriteLine(result);

result = Subtract(result, 4);
Console.WriteLine(result);

result = Divide(result, 3);
Console.WriteLine(result);

```

Here is the result of the example Main, above:

```
Microsoft Visual Studio Debug Console
8
16
12
4
```

Extra: Instead of summarizing fixed values as in the example above, let the user enter valueA and valueB. Also, feel free to add more advanced math operations as extra methods. :-)

Task 2.4 – Matching array elements

Create the following method:

```
static int SearchArray(string searchWord, string[] arrayToSearch)
```

The method should receive a search word and an array. For example, an array of fruits:

```
string[] fruits = {"Apple", "Banana", "Grape", "Orange", "Pear", "Watermelon"};
```

The method should check if the array includes the search word. If so, return the index (position) of the fruit in the array. If the search word is not in the array, the method should return -1.

Task 2.5 – Replacing words

Write a C# program that formats an imported string according to the following rules.

- If a "_" is prefixed and appended to a word, it will be displayed completely in capital letters (for example "_CSharp_" would become "CSHARP").
- If a word is enclosed in two "#" it will be written in lower case (for example "#CSharp#" would become "csharp").
- A word without formatting marks remains as it was typed.
- The formatting character is neglected during output.

For example, this string:

```
"Everyone _said_ that it would not work.\n Then someone came, #UNAWARE# of  
what everyone said, and just did it."
```

becomes this:

```
"Everyone SAID that it would not work.\n Then someone came, unaware of what  
everyone said, and just did it."
```

TIP: You should look into the following string methods for this task:

- Substring
- IndexOf & LastIndexOf
- Replace
- ToLower & ToUpper

Example of output:

```
Microsoft Visual Studio Debug Console
Everyone SAID that it would not work.
Then someone came, unaware of what everyone said, and just did it.
```

Task 2.6 – Methods in the string class

Create a string variable and assign it the sample text:

"Everyone said that it would not work. Then someone came, unaware of what everyone said, and just did it."

Perform the following text edits using the methods in the String class and output the results on the console:

- Get the two capital letters out of the text (by using the [] operator).
- Compare the text with your name and interpret the result (CompareTo).
- Add your first name to the text (+).
- See if the words "did" and "CSharp" occur (Contains).
- Compare two strings for equality (Equals or ==).
- Output the index of the first "d" in the text (IndexOf).
- Check for the empty string "" (IsEmpty).
- Check if a string is null (IsNullOrEmpty).
- Output the length of the text (Length).
- Read the words "said" and "did" (Substring).

Task 2.7 – Shape classes

We did a class example with Square in today's lecture.

- a) Make similar classes for: (alone or in a small group)
 - Rectangle
 - Circle
 - Triangle
 - *Any other polygons you want to implement. :-)*
- b) Test your classes.

Task 2.8 – TimePeriod class

- a) Create a TimePeriod class with functionality much like what we did in the lecture.
- b) Add properties for Minute and Second.

- c) Test your added functionality. Can you set x number of hours, and then retrieve it again as hours, but also retrieve it as the correct number of minutes or seconds if you prefer that format?

Task 2.10 – The shape classes revisited: properties

- a) Make C# properties in one or more of your shape classes, to replace the getters and setters.
- b) Test your changes.

Task 2.11 – Storing data to file

- a) For the Rectangle class we created in an earlier task, implement support for saving the values to a text file, as well as reading them from a text file. (You can use a very simple format, for example just two lines with one number on each of them).
- b) Test your new functionality.
- c) Try to make the formatting in the file fancier. For a Rectangle class, load data regardless of order, as long as it comes in this format: (numerical values can of course vary)
"shortSide": 3
"longside": 5
- d) Add code that saves values to a file in the same format.

Task 2.12 – Online coding challenges

As for last week, we use edabit.com/challenges/csharp to get customized challenges. Pick challenges yourself, fitting to your skill-level and the tasks you have worked on this week.