Homework 3

COP 3223C Introduction to Programming with C, Section 0V06

Spring 2021

1 Description

For this assignment you will be working with numeric types, functions, and arrays. You will write three functions named `CountWords`, and `fix_sorted_array`. Both functions should be in a single C source file named **homework3.c**. The descriptions of each function are listed below in the Function Requirements Section. Feel free to implement additional "helper" functions as you see fit.

2 Deliverables

A single C source file named homework3.c must be submitted to Webcourses by the assignment deadline (posted in Webcourses).

3 Grading Rubric

Uses correct filename (See Deliverables)	5 pts
Includes appropriate header comment (See Style Requirements)	5 pts
Uses tasteful comments (See Style Requirements)	10 pts
Follows all style requirements (See Style Requirements)	20 pts
Required functions work correctly and produce correct output (See Function Requirements)	60 pts
Total	**100 pts**

4 SUPER IMPORTANT INFORMATION

- **Sharing code or posting assignment details in public places will be considered academic dishonesty and will result in an automatic 0 grade for this assignment.** Feel free to have *high level* discussions about the assignment and your solution with your classmates. In general, discussions about the assignment are encouraged if they are only with students actively enrolled in this course and do not include any sharing of code.
- **Copying source code from the internet or other sources other than your own brain will be considered academic dishonesty and will result in an automatic 0 grade for this assignment.**
- Your source file must be named correctly to receive full credit. See Deliverables for the required filename. If you submit your file multiple times to Webcourses an additional "-n" will be added to the end of the filename (example: homework3.c, homework3-1.c, homework3-2.c, etc.). Files with this suffix will also be accepted.
- Your source file must contain the exact method signatures listed in Function Requirements to receive full credit.
- Your source file must compile and run to receive credit. **Submissions that do not compile will receive an automatic 0 grade.**
- **Submissions that print extra information to the console will not receive full credit.** Your submission should only produce console output if specifically requested in Function Requirements.

5 Function Requirements

The source file you submit should contain the following functions. Please note that these functions are useful tools, which are not a part of the C Standard Library. Being developed well, you can use them in real-world projects.

`unsigned long long CountWords(char text[]);`

Description: The function takes a text (character array) and returns the number of words in it. We call a "word" a part of the text that contains at least one symbol and separated from other words by one or more of the following symbols: space (' '), comma (','), dot ('.'), colon (':'), semi-colon (';'), question sign ('?'), or exclamation sign ('!').

Side note: In a real-world project, the set of symbols separating words could be different. For example, dash ('-') and plus ('+') could be included. For this assignment, please use the list of symbols above. And remember that following project requirements is very important. If you disagree with the requirements, you shall raise this question in your team and decide if you need to change project requirements or not. But you cannot decide this on your own without letting know your team about this.

Please note that the variable `text` could be an empty string, or a string, which does not contain anything but spaces and dots. In that case, the function should return zero (0).

Examples:

Text	**Return Value**
"H ello, World! "	3
" - . "	1
"This is a test text!!!"	5
": ; ??? "	0
"Well... ??? !!!!!!!"	1

`long long int fix_sorted_array(double arr[], size_t arr_size);`

Description: You have an array of elements of type double. It must be sorted, but recently, because of a system glitch, the sorting-in-place mechanism might allow **ONE** outlier, i.e., your (supposed to be) sorted array might contain one element in a wrong place. The elements in the array are not necessarily distinct.

Write a function that takes the array and the array's size (number of elements in the array) as arguments, finds the place the index of the outlier, fixes the array in place (that is puts the outlier to a place it is supposed to be), and returns the old index where the outlier was found. In case there was no outlier, the function should return -1. In any case, after running the function the array corresponding to the argument `arr` in the calling routine must be sorted.

Examples:

The Array	**Return Value**	**Array after running the function**
{-17,-5,-5,-2,1,17,289,17,17,395}	6 (index of the outlier)	{-17,-5,-5,-2,1,17,17,17, 289,395}
{289,17,17,17,100,250,300,1000,5000}	0 (index of the outlier)	{17,17,17,100,250,289,300,1000,5000}
{1,2,3,25,250,357,1000}	-1 (no outliers)	{1,2,3,25,250,357,1000}
{}	-1 (no outliers: empty array)	{}
{-1,23,70.39,500,1000,7000,23520,10}	7 (index of the outlier)	{-1,10,23,70.39,500,1000,7000,23520}

6 Style Requirements

- **Header Comments:** Submissions must include a header comment of the following form on the very first line of the file:

```
// <Your Name>
// NID: <Your NID>
// <Assignment Name> "<Filename>"
// <Course ID> <Section #>, < Current Semester>
```

```
// John Smith
// NID: jo123456
// Homework 3 "homework3.c"
// COP 3223C Section 0V06, Spring 2021
```

- **Function Comments:** All required functions must have a comment directly above them that describes what they do. Function comments should include a description of a function's expected inputs and expected outputs:

```
// Takes a non-negative integer n as input and returns n! where
// n! is a positive integer and  n! = 1 * 2 * 3 * 4 * … * n
int factorial(int n)
{
        …
}
```

- **Tasteful Comments:** You should add comments throughout your code that make the code easier to read. Be careful when adding tasteful comments to your code; only lines that *need* tasteful comments should have them. Tasteful comments should be placed directly above the line or block of code they describe, or on the same line as the line they describe:

```
void foo(int n)
{
        int i, a = 0, b = 1;

        // You can put comments like this :)
        for (i = 0; i < 10; i++)
        {
                a++; // You can put comments like this :)
                b--;

// You can NOT put comments like this (bad indentation)
                a = a * b;
        }
}
```

Please also **avoid very long comments**. Comments must usually fit one visible line in the editor.

- **Whitespace:** Variables, values, and operators should be separated by a single space in your code:

```
void foo(int n)
{
        int a = 0, b = 1;

        a = a + 1; // This is allowed
        b=b+1; // This is NOT a good style because there are no spaces between b, =, +, and 1
}
```

Whitespace: Your submission should include blank lines as needed to make your code easier to read. In general, you should include blank lines to separate statements that preform different tasks:

```
void foo(int n)
{
        int i, a = 0, b = 1;                    // Initialize some variables
                                                // Blank line for readability
        for (i = 0; i < 10; i++)                // For loop performing a new task
        {
                a++;
                b--;
        }
}
```

- **Indentation:** Submissions must have consistent indentation. This means that any two lines of code inside the same code block (functions, if statements, loops, etc.) must have the exact same indentation:

```
0 |void foo(int n)
1 |{
2 |        int i, a = 0, b = 1; // Lines 2 and 4 must have the same indentation
3 |
4 |        for (i = 0; i < 10; i++)
5 |        {
6 |                a++; // Both of these lines are in the same for loop,
7 |                b--;  // so they have the same indentation.
8 |        }
9 |}
```

- **Indentation (Contd.):** When a nested code block is created (like a loop inside a function), all lines of code inside the nested code block should be indented one more time than the nested block's container (I know that sounds confusing, so take a look at the example below):

```
0 |void foo(int n) // This line has an indentation of 0 (no indentation)
1 |{
2 |        int i, a = 0, b = 1; // This line has an indentation of 1 because it is inside of foo
3 |
4 |        for (i = 0; i < 10; i++) // This line has an indentation of 1 because it is inside of foo
5 |        {
6 |                a++; // Both of these lines have an indentation of 2
7 |                b--;  // because they are inside a for loop inside of foo.
8 |        }
9 |}
```