Homework 2

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 `IsDecimalInteger`, `GetSortingOrder`, and `RemoveExtraSpaces`. All three functions should be in a single C source file named **homework2.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 homework1.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: homework1.c, homework1-1.c, homework1-2.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.

`_Bool IsDecimalInteger(char str[]);`

Description: The function takes a string (character array) and checks if it is a valid decimal representation of an integer number. Please note that using functions like `atoi(…)` from the Standard Library is not really helpful here since the actual (mathematical) integer number might be larger than the maximum possible value of the type `unsigned long long int`. Your function must not have any restriction on the integer values (the length of the string representation). If `str` represents an integer, the function must return 1. Otherwise – return 0.

Examples:

str	Return Value
"23510"	1
" -2319"	0 (extra spaces in front)
"-2319"	1
"25g17"	0 (non-digit inside)
200-50	0 (the function does not compute expressions; it simply checks if `str` represents an integer number or not)
""	0 (empty string)
"999999999999999999999999999999999999999999999999"	1
"570.11"	0 (this is a decimal representation of a real number, but not an integer number)

`char GetSortingOrder(double arr[], size_t arr_size);`

Description: The function checks if the array `arr` (whose elements are of type `double`) is **sorted** (elements come in non-decreasing order: if $i < j$, then $arr[i] \leq arr[j]$), **reverse sorted** (elements come in non-increasing order), or **unsorted**.

Please note that, according to this definition, an array whose elements are all equal is both – sorted and reverse sorted. Let us call such an array a **constant** array.

Return Value: The function must return the following values:

- If the array is sorted, then return 1;
- If the array is reverse sorted, then return 2;
- If the array is constant, then return 3;
- If the array is unsorted, then return 0;

- If the array's size (`arr_size`) is zero (empty array), then return -1 (error codes are often negative – this is not required, but rather is a usual convention in the world of programming).

10 Bonus Points if your function will go through the array **only once**. It is important to reduce the number of operations in your code to make program run faster. For an array of 10 elements this would not be even noticed, but for an array of 2,000,000,000 elements this is very essential.

Examples:

The Array	Return Value
{-17,-5,-5,-2,1,17,17,17,289}	1 (sorted)
{289,17,17,17,1,-2,-5,-5,-17}	2 (reverse sorted)
{25,25,25,25,25,25,25}	3 (constant)
{-17,**-5,10,2**,-19,23}	0 (unsorted)
{}	-1 (empty array)

`void RemoveExtraSpaces(char txt[]);`

Description: The function takes a text (character array) and removes the extra spaces:

Any spaces at the beginning of the text;

Any spaces at the end of the text;

If there are more than one consecutive space character (' ') inside the text, one space remains, and the other ones are removed.

There is no output value. The function has to modify the array passed to it as the parameter txt. As you remember, modifications in an array passed to a function are reflected in the calling function.

Examples:

Input text	Output text
" Hello, Students ! "	"Hello, Students !"
""	"" (empty string)
" "	"" (empty string)

Please note that the string (character array) ends at the first End-Of-String symbol ('\0'). Thus, the length of the string can be determined easily.

Finally, having such a function in your library is a good thing. It can correct a text your program is obtaining from user's input or from files. You can further extend its functionality by removing spaces before punctuation signs and adding them after those. For example, "Thus , if we think carefully…" is replaced by "Thus, if we think carefully…".

This is one of those exercises, which can go a long way. In any case, working with strings efficiently is a desired quality for the real world of Engineering.

6 STYLE REQUIREMENTS

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

```
0 |// <Your Name>
1 |// NID: <Your NID>
2 |// <Assignment Name> "<Filename>"
3 |// <Course ID> <Section #>, < Current Semester>
```

```
0 |// John Smith
1 |// NID: jo123456
2 |// Homework 1 "homework2.c"
3 |// 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:

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

- **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:

```
0  |void foo(int n)
1  |{
2  |       int i, a = 0, b = 1;
3  |
4  |       // You can put comments like this :)
4  |       for (i = 0; i < 10; i++)
5  |       {
6  |                 a++; // You can put comments like this :)
7  |                 b--;
8  |
9  |// You can NOT put comments like this (bad indentation)
10 |                 a = a * b;
11 |       }
12 |}
```

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:

```
0 |void foo(int n)
1 |{
2 |        int a = 0, b = 1;
3 |
4 |        a = a + 1; // This is allowed
5 |        b=b+1; // This is NOT a good style because there are no spaces between b, =, +, and 1
6 |}
```

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:

```
0 |void foo(int n)
1 |{
2 |       int i, a = 0, b = 1;                    // Initialize some variables
3 |                                               // Blank line for readability
4 |       for (i = 0; i < 10; i++)                // For loop performing a new task
5 |       {
6 |               a++;
7 |               b--;
8 |       }
9 |}
```

- **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 |}
```