CSCI 1300 Introduction to Computer Programming
Instructor: Knox
Assignment 5
Due Sunday, October 15, by 6:00 pm
(5% bonus 6pm Fri Oct 13th, 2% bonus 6pm Sat Oct 14th)

This assignment requires you to write two files: *main.cpp* and *Assignment5.cpp*. There are seven functions that you are required to write for this assignment. All these should be written in a file called *Assignment5.cpp*. If you write more functions than the seven that are required, these should be kept in *Assignment5.cpp* as well. Your main.cpp file should just implement your **`main()`** function. Linking the two files can be done with **`#include "Assignment5.cpp"`** in your main file.

Once you have your code running on your virtual machine (VM), you must zip the file containing your functions into a .zip file (both *main.cpp* and *Assignment5.cpp* in the same .zip file) and submit that file to the autograder COG. You **must also** submit your code (same zip file) to Moodle to get full credit for the assignment.

Submitting Your Code to the Autograder:
Before you submit your code to COG, make sure it runs on your computer. If it doesn't run on the VM, it won't run on COG. The computer science autograder, known as COG, can be found here: **https://web-cog-csci1300.cs.colorado.edu**

- Login to COG using your identikey and password.
- Select the CSCI1300 - Assignment 5 from the dropdown.
- Upload your .zip file and click Submit.

Your file within the .zip must be named *Assignment5.cpp and main.cpp* for the grading script to run. COG will run its tests and display the results in the window below the *Submit* button. If your code doesn't run correctly on COG, read the error messages carefully, correct the mistakes in your code, and upload a new file. You can modify your code and resubmit as many times as you need to, up until the assignment due date.

Submitting Your Code to Moodle:
You must also submit your code to Moodle to get full credit for the assignment, even if the computer science auto-grader gives you a perfect score.

Comments at the top of your source files should include your name, recitation TA, and the assignment and problem number. **Please also include comments in your code submission to describe what your code is doing. TAs will be checking that your code has comments.**

Upload one .zip file to Moodle containing two files: your own main.cpp *and* Assignment5.cpp *which was sent to COG.*

Assignment Details:

This problem is divided into seven parts. Your main program should make a call to these seven functions, passing the necessary parameters and getting the necessary return values. You **must** provide functions with the given names and parameters. The names of any other supporting functions are named at your discretion. The names of the supporting functions that you implement should be reflective of the function they perform.

You will not take any user inputs in the functions in your Assignment5.cpp *file. The problem statement is designed in a way that all the inputs to the functions are being passed from your main function. Secondly, you will* ***not*** *be printing any values in these functions. You need to evaluate the correctness of your implementation by writing code in your main program and validating the return values from these function. If you add debugging outputs, please remove that code before submitting. In this assignment you will make use of arrays as your data structure to store values and looping should be preferably done using* **for** *loops.*

Treat every function as an independent function, where each function can take a different array as an input. Each of these functions will return a value or modify an array parameter. Your main should be used to setup the parameters and print out the returned value(s) of the function.

Homework Problem:

There are seven parts to the problem. The following is the step by step description of each.

Part 1 – Sum all the values of the array

In this part, you will sum all the values in an array.

```
float sumArray(float array[], int size)
```

The **`sumArray`** function will take two arguments, a reference to an array and the size of the array. The function will return the sum of all the values in the array as a float.

Part 2 - Search a value in an array

In this part, you will search a target value in an array, and return the index of that target value.

```
int search(int array[],int size, int target)
```

The **`Search`** function takes in three arguments: an integer array, the size of that array and the target value we are searching for. Your function should return the index of that target value in the input array. If the target is not found, return -1.

Part 3 – Calculate sum of squared differences between two arrays

In this part, you will find the sum of squared differences between two arrays of the same size.

```
float calculateDifference(int a[], int b[], int size)
```

The **`calculateDifference`** function takes in three arguments: a reference to array **a**, reference to array **b**, and the common size of the arrays. The function will use these values to calculate the sum of squared differences between each value of the two arrays and return that value as a float.

For example, the squared differences of two numbers ***a*** and ***b*** is $(a-b)^2$.

Part 4 – Sort the array

In this part, you will arrange the items in your array in ascending order, with the lowest item being at the start of the array and the highest item at the end. You can use the *Bubble sort* algorithm or the *selection sort* algorithm to sort the array.

```
void sortArray(float unsortedArray[], int size)
```

The **`sortArray`** function will take two arguments, i.e. a reference to the array and the size of the array. The function does not return any value, but will modify the elements of the array given as a parameter.

Part 5 – Copy the array

In this part, you will copy the values from one array to another array.

```
void copyArray(float source[], int size, float dest[])
```

The **`copyArray`** function will take three arguments, a reference to the source array to copy the values from, the size of this array, and a destination array.

Part 6 - Convert and fill another array

In this part, you will fill in the values of array b, corresponding to each value in array a. Map the values as follows:

Rating	Text	Meaning
0	Not-read	Haven't read it
-5	Terrible	Hated it
-3	Disliked	Didn't like it
1	Average	Ok, neither hot nor cold about it
3	Good	Liked it!
5	Excellent	Really liked it!

For example, when **`a`** is **`[0, 3, -5]`**, **`b`** should be ["Not-read", "Good", "Terrible"] after the function is executed. Use "INVALID" if the value does not match any valid rating.

```
void convert(int rating[], string text[],int size)
```

Part 7 – Find the median score

In this part, you will find the median item in the array. **The median value of a list arranged in numerical order is the middle element of that list. However, you are not allowed to change the order of the elements or change the values in the array that you receive as a parameter.**

```
float findMedian(float array[],int size)
```

The **`findMedian`** function should take two arguments, i.e. a reference to the array and the size of the array. The function should return the median of numbers in the array.

Challenge Problem

For this challenge problem, you are going to use something called a two-dimensional array, which is an array of arrays. You will write a function that will multiply every value inside the two-dimensional array with an integer, and return the resulting two-dimensional array.

In a one-dimensional array, it works like this:

[2 4 6 8] × 3 = [6 12 18 24]

And two-dimensional arrays work exactly the same.

```
void simple2DMultiplication(int multiplier, int matrix[10][10])
```

The **`Simple2DMultiplication`** function will take two parameters, an integer and a two-dimensional array. The function should modify the values in the two-dimensional array. Every value in the modified array should be the corresponding original value in **`matrix`** multiplied by **`multiplier`**.