CSCI 1300 Introduction to Computer Programming
Instructor: Knox
Assignment 5
Due Friday, March 3, by 12:30 pm

This assignment requires you to write two files: *main.cpp* and *Assignment5.cpp*. There are eight 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 eight 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 and submit that file to the autograder COG. You **must also** submit your code (both main.cpp and Assignment5.cpp in the 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 # from the dropdown.
- Upload your .zip file and click Submit.

Your file within the .zip must be named Assignment5*.cpp* for the grading script to run. *Your main.cpp file is not needed in your submission to COG.* 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 <u>two files</u>: your own <u>main.cpp</u> *and* <u>Assignment5.cpp</u> *which was sent to COG.*

Assignment details:

This problem is divided into eight parts. Your main program should make a call to these eight 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. 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.*

NOTE: Treat every function as an independent function, wherein each function can take a different array as an input.

Homework Problem:

There are eight parts to the problem, following is the step by step description of the same.

Part 1 – Function call to initialize a given array with the given value.

void *fillArray*(int data[],int size, int value)
The ***fillArray****()* function will take in three arguments, an array of integers, the size of the array and an integer value with which the array has to be initialized. You should pass the reference of your integer array that you declare in your main function to this function. The function should fill the entire array with the given value. The function does not return any value.

Tip - To test the validity of the function you can print the values of the array in your main function after the function call, to check if the entire array has been initialized correctly. You can follow similar validation steps for all the other functions.

Part 2 – Calculate grades of students

In this part, you will calculate the grades of students based on their corresponding scores. Your main program should declare a character array which will hold the grades of students. In the main itself you should create and fill an array of scores which is of float type. While making the function call, you will pass three arguments to this function, i.e. the reference of your scores array, the reference of your grades array and the size of scores array (*size of grades and scores array would be the same*) to the calculateGrades() function.

The function does not return any value.

void calculateGrades(float scores[],char grades[],int size)

This function will populate your grades array based on the scores stored in your scores array. The grades should be calculated based on the following rules and stored into your grades array. The array index for the grades in the grades array would be the same as the array index of the corresponding score in the scores array.

For scores greater than and equal to 90
Grade is A
For scores greater than and equal to 80 and lesser than 90
Grades is B
For scores greater than and equal to 70 and lesser than 80
Grades is C
For scores greater than and equal to 60 and lesser than 70
Grades is D
For scores less than 60
Grades is F

Note: Please make sure you test the validity of every function by creating different test arrays in your main program and passing it to these functions. This holds for all the other functions as well.

Tip: You can print out the values of your grades array in your main program to check its correctness.

Part 3 – Calculate the average score of class

In this part, you will calculate the average score of the students. The average score is calculated by taking the sum of all the scores of the students and dividing it by the number of students.

float getAverageScore (float scores[], int size)
The **getAverageScore***()* function will take two arguments, i.e. the reference of your scores array and the size of the array. The function should return the average score of the students which should be of float type.

Tip - You can print out the returned value of the average score in your main program to check its correctness.

Part 4 – Find the minimum score

In this part, you will find the minimum score from your list of scores and return the minimum score to your main function.

float getMinScore(float scores[],int size)
The **getMinScore***()* function should take in two arguments, i.e. the reference of your scores array and the size of the array. The function should return the minimum score value which should be of float type.

Tip - You can print out the returned value of the minimum score in your main program to check its correctness.

Part 5 – Find the maximum score

In this part, you will find the maximum score from your list of scores and return the maximum score to your main function.

float getMaxScore(float scores[],int size)
The **getMaxScore***()* function should take two arguments, i.e. the reference of your scores array and the size of the array. The function should return the maximum score value which should be of float type.

Tip - You can print out the returned value of the maximum score in your main program to check its correctness.

Part 6 – Sort the Scores Array

In this part, you will rearrange the scores in your array in ascending order, with the lowest score being at the start of the array and the highest score at the end. You can should use *Bubble sort* algorithm to sort the array.

void sortScores(float scores[],int size)
The **sortScores***()* function will take two arguments, i.e. the reference of your scores array and the size of the array. The function does not return any value.

Tip: You can print out the values of your array in your main program to check if it's sorted correctly.

Part 7 – Find the median score.

In this part, you will find the median score from your scores array . You are not allowed to change the order of the elements in the array that you are passing. The median of a sorted list is the middle element of the list.

float getMedian(float scores[],int size)
The **getMedian** function should take two arguments, i.e. the reference of your scores array(which is not sorted) and the size of the array. The function should return the median score of the students which should be of float type.

Tip - You can print out the returned value of the median score in your main program to check its correctness.

Part 8 – Count the number of students with a particular grade

In this part, you will count the number of students who have secured a particular grade.

int countGrade(char grades[],char grade, int size)
This function takes in three arguments, i.e. the reference of the grades array, the grade character whose occurrence you want to count and the size of the array. The function should return the count of students who have secured this grade.