ECE 150 LAB 4: FUNCTIONS

DUE: SECTION 001: THURSDAY, OCTOBER 12th AT 10:00 PM
SECTIONS 002/003: FRIDAY, OCTOBER 13th AT 10:00 PM

This lab consists of 4 questions worth 4 marks.

If you are in Section 001 you submit your work to http://marmoset01.shoshin.uwaterloo.ca

If you are in either Section 002 or Section 003 you submit your work to http://marmoset03.shoshin.uwaterloo.ca

Non quia difficilia sunt non audemus, sed quia non audemus, difficilia sunt.

1 OBJECTIVES

This lab is intended to have students:

- Take code they have already written and encode it within functions
- Use different techniques for returning error and warning status from functions
- Use call-by-value and const for passing arguments to functions
- Use call-by-reference for returning values from functions

LEARNING OUTCOMES

After completing this assignment, students should be familiar with the following:

A. C functions

B. Passing by value

C. Returning values by reference

D. Returning errors and warnings

1. GCD and LCM as Functions [1 marks]

In Question 1 of Lab 3 you were required to write a program that accepted two integers and computed their Greatest Common Denominator (GCD) and Least Common Multiple (LCM). For this question, you need to create C functions to implement that capability. The signatures of your C functions should be:

```
int greatestCommonDenominator(const int a, const int b);
int leastCommonMultiple(const int a, const int b);
```

The `greatestCommonDenominator()` function accepts two `int` parameters and computes and returns their greatest common denominator provided that both parameters are positive integers. Since the result of a valid GCD computation is always a postive integer, if there are any errors, the function should return -1. It should not output any messages to either `cerr` or `cout`.

The `leastCommonMultiple()` function accepts two `int` parameters and computes and returns their least common multiple provided that both parameters are positive integers. As with Question 1 of Lab 3, we will not test the function with arguments that cause the value of LCM(a,b) to exceed `INT_MAX`. Since the result of a valid GCD computation is always a postive integer, if there are any errors, the function should return -1. It should not output any messages to either `cerr` or `cout`.

Bonus: implement the `leastCommonMultiple()` function such that it *can* handle the case where the LCM(a,b) $>$ `INT_MAX`. In such a case, the function should return -2.

Given that you are implementing functions, we will provide a template `main()` function in the file `GcdLcmFunc.cpp` that you may edit as you see fit to test your functions.

All code will be subject to style checking.

Submission: You should submit the result of this exercise to the project A4-GCD-LCM-Functions. Your submission file must be named **GcdLcmFunc.cpp**. Note that the filename is *case sensitive*.

2. Nth-Root Function [1 marks]

In Question 2 of Lab 3 you were required to write a program that accepted a number, S, a root, n, and a precision, p and computed $\sqrt[n]{S}$ with precision p. For this question, you need to create a C function to implement that capability. The signature of your C function should be:

```
float NthRoot(const float S, const int N, const float precision);
```

Given that there are cases where you cannot compute the N^{th} root (*e.g.,* $\sqrt{-1}$), we need to be able to show that there is an error. However, unlike Question 1, where a -1 could be used to indicate an error, there is no floating point number that we could return that might not be a valid N^{th} root (*e.g.,* $\sqrt[3]{-1} = -1$ so we cannot use -1 as an error code.). Fortunately, the IEEE 754 floating-point standard includes "NaN" or Not-a-Number. If there are any errors such that your function cannot compute the N^{th} root *to the desired precision*, your function should return a NaN (to do so, use `std::numeric_limits<float>::quiet_NaN()`). Your function should not output any messages to either `cerr` or `cout`.

Given that you are implementing functions, we will provide a template `main()` function in the file `NthRootFunc.cpp` that you may edit as you see fit to test your functions.

All code will be subject to style checking.

Submission: You should submit the result of this exercise to the project A4-NthRoot-Functions. Your submission file must be named **NthRootFunc.cpp**. Note that the filename is *case sensitive*.

3. Statistics Function [1 marks]

In Question 3 of Lab 3 you were required to write a program that accepted a sequence of floating-point numbers and then computed the minimum, average, maximum, population standard deviation, and sample standard deviation. For this question, you need to create C functions to implement that capability, together with the additional statistic: median. The signature of your main C function should be:

```
bool statistics(const float dataset[], const int size,
                float& minimum, float& average, float& maximum,
                float& popStdDev, float& smplStdDev, float& median);
```

The function should take a dataset array and its size, and compute the required statistics. It should return "`true`" if this was possible and done correctly. If there are any errors, for any reason, your function should return "`false`". It should not output any messages to either `cerr` or `cout`.

To facilitate the creation of `statistics()` you should create six separate C functions to compute each of the six statistics:

```
float minimum(const float dataset[], const int size);
float average(const float dataset[], const int size);
float maximum(const float dataset[], const int size);
float popStdDev(const float dataset[], const int size);
float smplStdDev(const float dataset[], const int size);
float median(const float dataset[], const int size);
```

Each of these functions should take a dataset array and its size, and compute the required statistic. It should return the value of the statistic if this was possible and done correctly. If there are any errors, for any reason, your function should return "`NaN`", per the previous question. It should not output any messages to either `cerr` or `cout`.

Note that IEEE 754 not only has the capacity to specify "`NaN`" but it can also encode `+infinity` and `-infinity` which may be useful for certain inputs to these functions.

You may compute the median using whatever algorithm you prefer, though a relatively straightforward technique is to sort the array using your favourite sorting algorithm (there should be sample sorting code on Learn for at least bubblesort of an `int` array, and likely more than just that) and then select the middle element. (If the array has an even number of elements, then the median is the average of the middle two elements.)

Given that you are implementing functions, we will provide a template `main()` function in the file `StatisticsFunc.cpp` that you may edit as you see fit to test your functions.

All code will be subject to style checking.

Submission: You should submit the result of this exercise to the project A4-Statistics-Functions. Your submission file must be named **StatisticsFunc.cpp**. Note that the filename is *case sensitive*.

4. Sliding-Window Statistics Functions [1 marks]

In Question 4 of Lab 3 you were required to write a program that accepted window size and a sequence of floating-point numbers and then computed the minimum, average, and maximum over a sliding window of data. For this question, you need to create C functions to implement that capability, together with the additional statistics over the window: sample standard deviation and median (we do not compute the population standard deviation since, by definition, the window is a sample, not the whole population). The signature of your main C function should be:

```
int SWStats(const float dataset[], const int size,
            const int currentSample, const int windowSize,
            float& minimum, float& average, float& maximum,
            float& smplStdDev, float& median);
```

The function should take a dataset array and its size, the index in the dataset of the current sample, and the window size. For example, if we are processing sample number 247 and the window size is 5, then the statistics should be computed over data sample numbers 243, 244, 245, 246, and 247. As with Lab 3 Question 4, if the window includes negative sample numbers, then we use the value of sample number 0 for those values. For example, when processing the first sample (sample number 0), and a window size of 3, then the other elements of the sliding window would be sample numbers -1 and -2. Since neither sample number -1 nor -2 exist, we use the value of sample number 0 instead for those values.

The `SWStats()` function should compute the required statistics for the current sample over the window. However, as with Lab 3, Question 4, it is possible that there are both errors and warnings. To resolve this, your `SWStats()` function should return "`0`" if the statistics were calculated with neither errors nor warnings. If there are any errors, for any reason, your function should return a negative integer; you should use a different negative integer for each different error type. If there are warnings, your function is expected to calculate the statistics, but return a positive integer; as with errors, you should use a different positive integer for each different warning type. Your function should not output any messages to either `cerr` or `cout`.

To facilitate the creation of `SWStats()` you should create five separate C functions to compute each of the five statistics:

```
int minimum(const float dataset[], const int size,
            const int currentSample, const int windowSize,
            float &minimum);
int average(const float dataset[], const int size,
            const int currentSample, const int windowSize,
            float &average);
int maximum(const float dataset[], const int size,
            const int currentSample, const int windowSize,
            float &maximum);
```

```
int smplStdDev(const float dataset[], const int size,
               const int currentSample, const int windowSize,
               float &smplStdDev);
int median(const float dataset[], const int size,
           const int currentSample, const int windowSize,
           float &median);
```

Each of these functions should take a dataset array and its size, the index in the dataset of the current sample, and the window size, and compute the required statistic. As with the `SWStats()` function, each of these functions should return `0` if the statistic could be computed without error or warning, a negative number for an error, and a positive number for a warning. It should not output any messages to either `cerr` or `cout`.

Note that IEEE 754 not only has the capacity to specify "`NaN`" but it can also encode `+infinity` and `-infinity` which may be useful for certain inputs to these functions.

You may find the code you created in the previous question useful for this question and you may call those functions if necessary.

Given that you are implementing functions, we will provide a template `main()` function in the file `SWStatsFunc.cpp` that you may edit as you see fit to test your functions.

All code will be subject to style checking.

Submission: You should submit the result of this exercise to the project A4-SWStats-Functions. Your submission file must be named **SWStatsFunc.cpp**. Note that the filename is *case sensitive*.