Assignment 09: Arrays: Passing Arrays to Functions and more Array Processing

COSC 1437: Programming Fundamentals II

Objectives

- Review and practice writing user defined functions
- Learn about code reuse through reusing user defined functions.
- Declare and process arrays in C/C++
- Declare arrays as parameters to functions and pass arrays into functions for processing.
- Practice using `<cmath>` predefined library functions

Description

In this programming assignment you will write a program that will take an array of integers, and calculate the mean (average) and standard deviation of the numbers. If the numbers in the array are

```
int n = 5;
int x[] = {5, 3, 8, 2, 1};
```

Then to calculate the mean, we use the formula:

$$\bar{x} = \frac{1}{n} \sum_{i=0}^{n-1} x_i$$

where $\bar{x}$ represents the calculated mean, and n is the total number of values in the array x. In other words, the mean is simply the sum of the values divided by the total number of values (n), or

$$\bar{x} = \frac{x_0 + x_1 + x_2 + x_3 + x_4}{n} = \frac{5 + 3 + 8 + 2 + 1}{5} = 3.8$$

Likewise, the formula for calculating the standard deviation of a set of values is given by:

$$s = \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} (x_i - \bar{x})^2}$$

Here you should notice that $\bar{x}$ is the calculated mean of the values in the x array we just showed previously. In English, the standard deviation is the sum of the square of the differences of each value from the mean. This sum of the squared differences is again divided by n the total number of values, then we take the square root of this whole calculation to get the final standard deviation.

So for our example x array, the standard deviation would be calculated as

$$\begin{aligned} s &= \sqrt{\frac{1}{n}\left[(x_0 - \bar{x})^2 + (x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + (x_3 - \bar{x})^2 + (x_4 - \bar{x})^2\right]} \\ &= \sqrt{\frac{1}{5}\left[(5 - 3.8)^2 + (3 - 3.8)^2 + (8 - 3.8)^2 + (2 - 3.8)^2 + (1 - 3.8)^2\right]} \end{aligned}$$

$$\approx 2.4819$$

We will be emphasizing the writing and reuse of functions in this assignment. You will build several smaller functions that will be reused to do most of the work of the functions to calculate the mean and standard deviation.

Overview and Setup

For all assignments, it will be assumed that you have a working VSCode IDE with remote Dev Containers extension installed, and that you have accepted and cloned the assignment to a Dev Container in your VSCode IDE. We will start each assignment with a description of the general setup of the assignment, and an overview of the assignment tasks.

For this assignment you have been given the following files (among some others):

File Name	Description
`src/assg09-tests.cpp`	Unit tests for the tasks that you need to successfully pass
`src/assg09-library.cpp`	File that contains the code implementations, all your work will be done here
`include/assg09-library.hpp`	Header include file of function prototypes

As usual, before starting on the assignment tasks proper, you should make sure you have completed the following setup steps.

1. Copy the assignment repository on GitHub by accepting the assignment using the provided assignment invitation link for 'Assignment 09' for our current class semester and section.
2. Clone the repository using the SSH url to your local class DevBox development environment. Make sure that you open the cloned folder and restart inside of the correct Dev Container.
3. Confirm that the project builds and runs, though no tests may be defined or run initially for some assignments. If the project does not build on the first checkout, please inform the instructor.
4. You should create the issue for Task 1 and/or for all tasks for the assignment now before beginning the first task. On your GitHub account, go to issues, and create it/them from the issue templates for the assignment.

Assignment Tasks

Lab Task 1 / In-Lab Work : Summing Values in an Array

This is a repeat of a task in the previous assignment, though you have not been given the function prototype and a stub function this time for this or any function for this assignment.

For the first task, you will implement a function named `sumOfValues()`. This function will take an array of double values, sum up all of the values in the array, and return the sum as a double result. Notice by looking at the tests that this function, like all of the functions you will write, actually first takes an integer parameter, which is the number of values in the array that will be passed in as the second parameter.

NOTE: Actually the array of `double` values should be declared to be a `const` parameter. Do you know why? We discussed passing in arrays as `const` parameters vs. as non constant in this units lecture videos.

The `sumOfValues()` function should be relatively simple to implement hopefully. There are several tests that test summing up arrays of various size in the given unit tests. Your implementation needs to sum up all of the values and return the resulting sum as a `double` data type. The unit tests do test some edge cases, like for example what if you are given an array of size 0 (an empty array), do you return a default sum of 0.0 in that case?

To complete the work for this task, perform the following steps:

1. Declare a local variable that will keep track of the running sum of the values in the array. This sum will be returned as the `double` result of this function. You should initialize the calculated sum to `0.0` and if the `numValues` of the array is 0, then the sum returned is expected to be `0.0`.
2. You are required to use an indexed controlled `for` loop to process and sum up the values in the `values[]` array that is passed in as the second parameter. The loop needs to access and add each value in the `values[]` array to the running sum.

3. After all values are processed, the sum of the values should be returned as the result of calling this function.

If you successfully complete the function, and save and compile and rerun the tests, you should find that all of the unit tests for Task 1 pass successfully. When you are satisfied your code is correct, create a commit and push your commit to your GitHub classroom repository. It is always a good idea to check the autograder after pushing a commit to ensure it is also compiling and passing the tests for the task you just completed on GitHub.

Task 2: Implement `calculateMean()` Function

As usual, create the function prototype and a stub, and uncomment the tests for task 2, to make sure your project still compiles and can now run the unit tests that you need to pass for this task 2.

`calculateMean()` will have the same signature as your first function. The function should take a `const` array of `double`, and calculate the mean of these values, returning their mean as a `double` result.

You are required to reuse the `sumOfValues()` function in order to implement your task 2 `calculateMean()` function. As discussed in the assignment description above, the mean is simply the sum of the values divided by the total number of values being averaged. So you can reuse the first function to determine the sum of the values that you are calculating the average of.

NOTE: be careful about integer division. Since the number of values in the array is given to you as an integer, you have to divide by this to get your average. Sometimes, when you divide by an integer in C/C++ you end up with an integer result. You should google integer division in C/C++ if you do not know what the potential issue is here with dividing by an integer data type.

When you are satisfied your function works (make sure your project compiles and runs still), you should commit your changes and push them to the `Feedback` pull request in your classroom repository.

Task 3: Implement `differenceOfValues()` Function

Make sure you do your prerequisite steps first, such as creating the issue for this task. Then uncomment the tests for this function, and get your project back to a compilable and runnable state before proceeding.

If you look closely at the definition of the standard deviation, we can break it up into several steps. First of all, the standard deviation is calculated by taking the difference of each value with the mean. Then all of the differences are squared. The standard deviation is actually the mean of these squared differences, e.g. if we calculate the square of the difference of each value with the mean, then take the average of these squared differences, then we have the standard deviation (after we take a final square root of this calculation).

So to calculate the standard deviation, we want to reuse the `calculateMean()` function, and we want to create some other smaller functions we can use in that calculation and potentially reuse in other ways.

The `differenceOfValues()` function works a bit differently from the previous two. We need to pass in an array of `double` values (and the array size) as before. But you will pass in a third parameter of type `double`. This value is the value you are to subtract from each of the values in the array passed as the second parameter.

NOTE: This time the array of values cannot be a `const` parameter. Do you understand why not?

This function is a `void` function this time, which is different as well. You will be calculating the difference of each value, and storing the result back in the array passed in as a parameters. As you should have learned/reviewed from this units materials, arrays are passed in by reference implicitly. So if you modify the values of the array to contain the calculated differences, the results of this calculation will be returned back to the caller of your function.

When you are satisfied your function is working, the project still compiles, and you can run and pass the tests, perform the usual to create and push a commit to the `Feedback` pull request.

Task 4: Implement `squareOfValues()` Function

Perform the usual prerequisite steps before starting task 4.

Once you have your program running the tests again, implement the `squareOfValues()` function. This function will work in a similar manner to the previous function. It is a void function because the work it does is done on the values of the array passed in as a parameter.

This function simply squares all of the values in the array, so there will be no third parameter for this function, simply the array of doubles (and the array size) passed in as the 2 input parameters. But like before this is a void function because the results will be stored back in the array to be returned to the caller by reference.

NOTE: you are required to use the `pow()` function from the `cmath` C/C++ library to implement the squaring calculation of each element. You could calculate the square by multiplying each value by itself. But this is not as clear as explicitly using a `pow()` function to square each value. Also you are practicing code reuse here still, and you should always reuse function from the standard libraries in the language you are programming with where available and where these make your code clearer and more readable.

When satisfied with your implementation, commit and push your work to the `Feedback` pull request of your classroom repository.

Task 5: Implement `calculateStandardDeviation()` Function

Perform the usual prerequisite steps before starting task 5.

The final task will be to put together and reuse all of the previous functions to implement the standard deviation calculation. You will be using all 4 of the previous functions, as well as the `cmath` `sqrt()` function, either directly or indirectly here.

This function has almost the same signature as the `calculateMean()` function. It takes an array of `double` values and its size as input parameters. And it returns a `double` result, which should be the final calculated standard deviation of the original array of values.

NOTE: in theory the array of values should be `const` for this function since it doesn't make sense that calling this function to calculate the standard deviation would have as a side effect the property of altering the values. Why can't the parameter be a `const` like in `calculateMean()`? We will later discuss how to fix this little issue when we talk about dynamic memory management.

If all of your previous functions are working, you can implement the calculation of the standard deviation in the following manner.

1. Calculate the mean of the initial values in the array by reusing your `calculateMean()` function.
2. Calculate the difference of each value in the array of values from the mean value by calling your `differenceOfValues()` function.
3. Calculate the square of each of the differences with your `squareOfValues()` function.
4. Calculate the mean of the squared differences using the `calculateMean()` again. Notice you are (re)using this function 2 times in the implementation of your standard deviation calculation.
5. The final result then is the square root of the mean of these squared differences. So you are require to use the `sqrt()` function from `cmath` to get the square root of this result. This should be the final standard deviation that your function will return.

If you have gotten all of the tasks completed to this point, you should be able to run and pass all of the original tests once your `calculateStandardDeviation()` is working correctly. When satisfied you should commit and push your changes. Check your final GitHub actions at this point. If your work is committed correctly, you should see you now get a green check mark and your most recent action passes all actions on the GitHub classroom repository now.

Assignment Submission

For this class, the submission process is to correctly create pull request(s) with changes committed and pushed to your copied repository for grading and evaluation. For the assignments, you may not be able to complete all tasks and have all of the tests successfully finishing. This is OK. However, you should endeavor to have as many of the tasks completed before the deadline for the assignment as possible. Also, try and make sure that you only push commits that are building and able to run the tests. You may loose points for pushing a broken build, especially if the last build you submit is not properly compiling and running the tests.

In this assignment, up to 50 points will be given for having completed at least the initial Task 1 / In Lab task. Thereafter 25 additional points are given by the autograder for completing tasks 2 and 3 successfully.

Program Style

At some point you will be required to follow class style and formatting guidelines. The VSCode environment has been set up to try and format your code for some of these guidelines automatically to conform to class style requirements. But not all style issues can be enforced by the IDE/Editor. The instructor may give you feedback in your pull request comments and/or create issues for you for the assignment that you need to address and fix. You should address those if asked, and push a new commit that fixes the issue (or ask for clarification if you don't understand the request). In general the following style/formatting issues will be required for programs for this class:

1. All programs must be properly indented. All indentation must be consistent and lined up correctly. Class style requires 2 spaces with no embedded tabs for all code indentation levels. The editor style checker should properly indent your code when you save it, but if not you may need to check or correct this if code is misaligned or not properly indented.
2. Variable and function names must use `camelCaseNameingNotation`. All variable and function names must begin with a lowercase letter. Do not use underscores between words in the variable or function name. Often function names will be given to you, but you will need to create variables, and maybe some functions, that conform to the naming conventions.
   - Global constants should be used instead of magic numbers. Global constants are identified using `ALL_CAPS_UNDERLINE_NAMING`.
   - User defined types, such as classes, structures and enumerated types should use camel case notation, but should begin with an initial upper case letter, thus `MyUserDefinedClass`.
3. You are required to use meaningful variable and function names. Choosing good names for code items is an important skill. The code examples and starting code tries to give examples of good and meaningful names. In general, do not use abbreviations. Single variable names should be avoided, except maybe for generic loop index variables `i`, `j`, etc. Make your code readable, think of it as writing a document to communicate with other developers (and with your instructor who will be evaluating your code).
4. There are certain white space requirements. In general there should usually never be more than 1 blank line in a row in your code. Likewise there should usually not be more than 1 blank space on a line. There should be 1 blank space before and after all binary operators like `+`, `*`, `=`, `or`.
5. Function documentation is required for all regular functions and all class member functions. You need to follow the correctly formatted Doxygen function documentation format. We will use function documentation generation, and you should be sure your documentation can be built without emitting warnings or errors. Likewise all files should have a file header documentation at the top. You should edit the file header of files where you add in new code (not simply uncommenting existing code). Make sure the information has your correct name, dates, and other information.
6. Practice using proper Git commit messages. You should refer to issues and tasks correctly in commit messages.

Additional Information

The following are suggested online materials you may use to help you understand the tools and topics we have introduced in this assignment.

- Git Tutorials
- Git User Manual
- Git Commit Messages Guidelines
- Test-driven Development and Unit Testing Concepts
- Catch2 Unit Test Tutorial
- Getting Started with Visual Studio Code
- Visual Studio Code Documentation and User Guide
- Make Build System Tutorial
- Markdown Basic Syntax