Assignment 08: Arrays: Processing Arrays of Values

COSC 1437: Programming Fundamentals II

Objectives

- Practice using arrays, see examples of declaring arrays statically
- See examples of passing arrays to functions and using them as parameters in functions
- Practice common operations over arrays, such as summing up values and counting occurrences of values found in an array of values.

Description

In this assignment you will perform some basic operations on arrays of values. We will continue to use functions in our assignments for testing, and for this assignment you have been given the function prototypes and declarations that declare functions that pass in arrays of values to be processed. Your task will be to implement the functions to perform the described operations on the arrays.

All three tasks in this assignment ask for very common array operations to be performed. In all of the tasks, you probably should use a `for` index controlled loop to iterate over the elements of the array to perform the operations. The operations you will perform will be to sum up all of the values in an array of values, find the largest value in an array of values, and count the number of occurrences of a particular value in an array of values.

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/assg08-tests.cpp`	Unit tests for the tasks that you need to successfully pass
`src/assg08-library.cpp`	File that contains the code implementations, all your work will be done here
`include/assg08-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 08' 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 an Array of Values

A common task that you will need to perform when working with arrays is to sum up all of the values of an array. For this task and the tasks in this assignment, you have been given the function prototype for the `sumValues()` function in the `assg08-library.hpp` header file, and a stub implementation in the `assg08-library.cpp` file, you only need to implement the function.

The function prototype for the `sumValues()` is typical for a function that takes an array as input and performs operations on the array. The function takes two parameters as input, first is the `numValues`, the number of values that are in the array that will be passed in to be summed up. E.g. this is the size of this list of values. In this task you are summing up an array of double values. Notice that the array is passed in as the second parameter to the `sumValues()` function. Notice the syntax of how you declare an array to be passed in as a parameter to a function. For this assignment, all of the arrays will be passed in as `const` reference parameters, since you are not supposed to modify the values in the array, simply perform calculations with the arrays.

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: Find maximum value in an array of values

Another common task one performs with arrays is to find the maximum (or minimum) value in the array of values. For task two you are given the function prototype and stub function for the `maximumValue()` function again. This function has a similar signature to task 1, but notice we pass in an array of integer `values[]` this time (instead of an array of `double` values).

This function should not use a special flag variable like `INT_MIN` to perform the work. Instead implement the algorithm as described here:

1. Create a local variable to keep track of the maximum value seen so far. Initialize this to the value at index 0 of the `values[]` array that is passed in to this function. This function assumes it is always called with an array of values of at least size 1 or bigger.
2. Use an index controlled `for` loop again to search indexes 1, 2 and so on up to the number of values in the array. Anytime you find a value that is larger than any you have previously found, you should remember it as the maximum value you have seen so far. If you implement this `for` loop correctly, when the size of the array passed in is just 1 item, the for loop will not run any iterations, and you should end up returning the value at index 0 of the passed in `values[]` array.
3. After you have searched for the maximum value, you should return the largest value that you see in the given input array of `values[]`.

When finished, save compile and rerun the tests. Once you are satisfied, make and push your commit of Task 2 to your GitHub classroom repository, and check that it compiles and passes tests in the autograder.

Task 3: Count occurrences of a value in an array

In task 3 you will perform another common task on an array, count up the duplicate occurrences of some value or record in the array. In this task we will practice using the `string` data type some more, which is available as a basic type in the `C++ language` (read chapter 7 on using the `string` type if you haven't done so already).

The signature and stub function for the `countOccurrencesOfName()` function have been given again to you for task 3. The signature for this third function is slightly different from the previous two, besides the fact that we are passing in an array of `string` values as input in the second parameter. This function has a third parameter called `name`. Don't confuse `names[]` the second parameter with `name` the third parameter. The `names[]` second parameter will be an array of `string` names to be searched. The third parameter called `name` is the name that will be searched and counted up in the array.

So to implement the third task perform the following steps:

1. Again create a local variable to keep track of the number of occurrences of the name that you see while counting them up when processing the `names[]` array. This count should be initialized to 0, and it is possible to be given an empty array of `names[]` for this function in which case a result of 0 is expected to be returned.
2. Again you are required to use an index controlled `for` loop that iterates over all indexes from 0 to the number of names in the input array. You should test each value in the `names[]` array to see if it contains the `name` being searched for. As a big hint, you can use a simple boolean expression to test if an array value is equal to the `name` being searched for, for example:

```
if (names[index] == name)
{
  // then the name at the current index is an occurrence of the name we are counting up
}
```

3. Once you have iterated over the array and counted all occurrences, you need to return the count of the occurrences/duplicates of the indicated name as the result from this function.

When finished, save, compile and rerun the tests. Once you are satisfied, make and push your commit of Task 3 to your GitHub classroom repository, and check that it compiles and passes tests in the autograder.

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