# Assignment 10: Arrays: Searching and Sorting Arrays

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++
- More practice declaring arrays as parameters to functions and pass arrays into functions for processing.
- Implement a sort to sort the values of an array
- Implement a binary search to search for the values in a sorted array

## Description

In this assignment we will be continuing to practice using arrays, and passing arrays into functions to process the values of an array. And we are going to implement a function to sort an array of values in this assignment, and another function to implement a binary search to search an array of values. There are 3 different sorting algorithms / functions given in our Malik c++ textbook: bubble sort, insertion sort and selection sort. In task 3 you can choose any of these three. You are given the code in the text of these algorithms, but you will have to modify the textbook code to work to sort an array of doubles. Likewise in task 4 you are to implement the binary search that is given in our textbook, with a few modifications, for this assignment.

## 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/assg10-tests.cpp` | Unit tests for the tasks that you need to successfully pass |
| `src/assg10-library.cpp` | File that contains the code implementations, all your work will be done here |
| `include/assg10-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 : Determine if an array is sorted

This in-lab task and the next will actually be implementations of functions that we will be reusing in the tests for the assignment. In this first task, we need to write a function that will take an array of `double` values, and determine if the array is already sorted in ascending order or not. The function should be named `isSorted()`, and it should return a `bool` boolean result of `true` if the array is sorted, and `false` if the array is not sorted. This (and all functions in this assignment) should work for arrays with only a single value, and empty arrays with no values. This function takes two parameters as input, the first called `numValues` is the number of values in the array to be tested, and the second called `values` is an array of double values that will be tested.

The algorithm to test if the array of `values` is sorted is relatively simple, though be careful that your code works correctly when `numValues` is 0 or 1. Basically you want to start by testing if the value at index 0 is greater than the value at index 1. If it is you know the array is not sorted, and should therefore return `false`. But if index 0 and 1 are in order, then you need to test index 1 and index 2. So you need a loop that starts at index 0 and tests the index above it. Be careful that your loop correctly terminates. For example, if `numValues` is 10, that means the array of `values` has 10 values in it in indexes 0 to 9. So your loop needs to start at index 0 to test 0 and 1, but it needs to stop at index 8 when you test indexes 8 and 9. It is an error if your loop goes up to index 9 and tests indexes 9 and 10 (e.g. this is an array bounds error, an array of size 10 only has valid indexes from 0 up to 9).

So the suggested algorithm for the `isSorted()` function is as follows:

1. Perform a `for` loop that will iterate from index 0 up to the 1 less than the last valid index. For example if numValues i 10, then the for loop should iterate from 0 up to 8. The loop should not be entered when `numValues` is either 0 or 1 if you write it correctly.
   - Inside of the loop, test the current `index` and `index + 1` locations. If the value at `index` is larger, then it is out of order, so you should immediately return `false` if you detect this.
2. But if the loop completes and never returns `false` then you know all values successfully tested as in the correct order, so return `true` at the end of your function after the loop completes.

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 `areEqual()` 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.

The second task function will also be (re)used when testing your search and sort implementations later on. This function, named `areEqual()` takes two `double` arrays of values as input and its purpose is to test that all of the values in both arrays are the same or not. This function should return a `bool` result again of `true` if the two arrays that are passed in are completely equal (all values at all indexes are the same), or `false` if any values are not the same. This function will assume that the arrays are both of the same size, so the first parameter will again be `numValues` an `int`. But then two arrays of `double` values should be passed in, you can call them `values1` and `values2` respectively for this task.

This function is probably simpler than the previous one, though you have to pass in and compare two arrays for the first time in an assignment. But in this case, you should need only a simple loop that iterates over all of the `numValues` indexes in the passed in arrays, and returns `false` immediately if any value at any index is not equal. If no values are detected as not being the same, then this function should return `true` after testing all indexes of the two arrays.

So the suggested algorithm for `areEqual()` is as follows:

1. Perform a `for` loop over all of the indexes 0 to `numValues` of the two arrays. If the passed in arrays are empty, then `numValues` will be 0, and your loop should not execute any iterations in that case.
   - In the loop, test each value at each index to see if it is equal in the two arrays passed into this function. If you find any two values that are not equal, just immediately return a result of `false`.

2. Then after the loop finishes, if no values were detected that were not equal, you should return `true` because all of the values tested as being equal.

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 a `sort()` Function

Our textbook has 3 different implementations of sorting algorithms. The selection sort is discussed in chapter 8 when searching and sorting is first introduced in our text. And later in chapter 16, bubble sort and insertion sort are also both presented. In all of these cases the function sorts an array of integers in the example, and the array is passed in as the first parameter, and its length as the second.

For task 3, create a function named simply `sort()`. You can choose any of the 3 sorting algorithms that you like that are presented in our text (but it should be one of the 3). You need to modify the algorithm to sort an array of `double` values, and by convention we pass in the number of values to be sorted as the first parameter, and the array as the second parameter.

But otherwise you should find that, with these small changes, you can mostly use the given function(s) from our text to implement your task 3 `sort()`. If you like, you could copy the `swap()` function you implemented previously or use the swap function from the c++ standard library, instead of swapping the values using 3 statements as shown in our text for all of the sorts.

Also as a note, if you look at the tests of your sort in task 3, we are reusing your `isSorted()` and `areEqual()` functions when performing our tests. So if these functions are not working correctly, you will have problems passing the tests of your `sort()` in task 3.

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 `search()` Function

Perform the usual prerequisite steps before starting task 4.

Also in chapter 8 and 16 of our textbook, searching an array for a value is discussed. In chapter 16, an implementation of a binary search is presented to search for a value in an array of integers.

In this task we are going to be modifying the binary search a bit. You need to search an array of `double` values. We pass in the number of values as the first parameter, and the array of doubles as the second parameter. The third parameter should be a `double` which is the value to be searched for in the array.

But also, we will not assume that the array of `double` values to be searched is already sorted before performing the binary search. But you already have all of the functions you need to check if the array is unsorted (task 1), and to sort if if needed (task 3). So before performing the actual search, first check if the array is unsorted, reusing your `isSorted()` function. And if it is not sorted, reuse your `sort()` function to sort the values in preparation for a binary search. Note that since the array might be sorted now by your function, you cannot pass the array in as a `const` parameter. So you should not use the `const` keyword for the array of values parameter in this function.

After testing and sorting if necessary, implement the binary search as shown in our text. Basically the search works by checking if the value at the middle of the unsearched part of the array is the one being looked for. If it is we return the index where we found the value in the array. If we don't find it in the middle index, we know that if the value is smaller than the one at the middle, we need to look at indexes before the mid point, since the array should be sorted at this point. Likewise if the value we are looking for is larger than the one at the middle index, we need to look at the larger indexes in the array. This is what the manipulations of `first` and `last` are doing in the loop when we haven't found the value yet.

Also note that a global constant named `NOT_FOUND` has been declared in the header file for this assignment. You should use `NOT_FOUND` instead of `-1` to indicate that the search has failed to find the value in the array.

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

## 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