Lab 4: Arrays

Assigned: *Tuesday, October 27th, 2020*
Due: *Monday, November 2nd, 2020*
Points: *100*

- There is NO MAKEUP for missed assignments.
- We are strict about enforcing the LATE POLICY for all assignments (see syllabus).

Introduction

The assignment for this week will utilize concepts of arrays.

We will be looking for (and grading) programming stylizations, such as proper use of comments, tab indentation, good variable names, and overall block and function designs. We will also be grading for meeting requirements, using "class legal" code, and plagiarism. So, it is not enough for your lab to just pass the Gradescope autograder! Please read the instructions herein carefully.

It is HIGHLY RECOMMENDED that you develop the algorithms for each program FIRST and THEN develop the C++ code for it.

Step 1: Getting Ready

First, open a terminal window and log into the correct machine. Change into your CS 16 directory, create a **lab04** directory and change into it. Remember that at any time, you can check what directory you are currently in with the command **pwd**.

Step 2: Create and Edit Your C++ Files

This week, you will need to create **one (1) C++ program called arrays.cpp**. It is worth 100 points and submitted for full assignment credit.

IMPORTANT NOTE: As usual, this lab will be also graded for use of comments and style. See below for details. In addition, we will take *major* points OFF if you use C++ instructions/libraries/code that either (a) was not covered in class, or (b) was found to be copied from outside sources (or each other) without proper citation.

arrays.cpp

A. The Setup

For this project, you will start with **FOUR (4)** files: **ArrayFile.txt**, **constants.h**, **headers.h**, and **skeleton_arrays.cpp**. These are provided for you in 2 places for your convenience: on our class website and on CSIL. You can copy all these files, once logged into CSIL, like this:

```
$ cp /cs/faculty/zmatni/public_html/cs16f20/lab4/* .
```

Please note that there is a dot (**.**) at the end of that line. This assumes you are in your **lab4** directory.

It is a very good idea to take a look at those files before you continue with anything, starting with **skeleton_array.cpp**. You can view them using your usual text editor. You will also want to make a copy of the skeleton file (**skeleton_arrays.cpp**) and call it **arrays.cpp**, edit *that* file, and eventually submit it (your submission must be called **arrays.cpp**):

```
$ cp skeleton_arrays.cpp arrays.cpp
```

You will **ONLY** be submitting **arrays.cpp** into Gradescope, but you will need the other 3 files to be able to run your program on CSIL. The autograder will have those same 3 files (i.e. **constants.h**, **headers.h**, and **ArrrayFile.txt**).

The first thing you should do is open and read **skeleton_arrays.cpp**. Upon closer examination, you notice that there are two "new" types of instructions there:

```
#include "headers.h"
#include "constants.h"
```

These lines are there to tell the compiler to go get the files "**headers.h**" and "**constants.h**" and place them in the program there. This is one way to make your programs be a little more compact. This is what each contains:

a. "**headers.h**" contains all the function declarations that you will need for "**array.cpp**". There are 8 function declarations in there. You will need to create the function definitions from these:

```
void print_array (int arr[], int asize);
int maxArray (int arr[], int asize);
int minArray (int arr[], int asize);
int sumArray (int arr[], int asize);
void evensArray (int arr[], int asize);
void primesArray (int arr[], int asize);
int SeqSearch (int arr[], int array_size, int target);
void AllSearches (int arr[], int array_size);
```

The file has pre- and post-condition comments with these declarations. **Make sure you READ THEM**.

In addition, this file also contains a definition for a function called **getArray()**. It involves getting data from an external file and you are **not** responsible for doing anything with it (we haven't learned how to do file I/O yet). **DO NOT CHANGE THE getArray() DEFINITION AT ALL!!**

b. The file "**constants.h**" contains declarations for **constant global variables** that you will need for "**array.cpp**".

Your program will be reading in a text file called "**ArrayFile.txt**", which contains a `MAXSIZE` number of integers. The program assigns those integers as elements of an array called **array[ ]** via the function called **getArray()**. Again, this is a function that you do NOT have to define yourself – it is written for you in **headers.h**. More on this in a bit. The other 2 things in the **constants.h** file are:

1. a constant int called `NSEARCHES`, which is set to **10**.
2. a constant int array called `SEARCHES[ ]` that contains `NSEARCHES` elements. `SEARCHES[ ]` is initialized to certain int values. Again, more on this in a bit.

B. Designing Your Program

Your program **array.cpp** will do **EIGHT (8)** tasks (they're not big tasks, individually). Each one of these 8 tasks has to do with one of the 8 function declarations mentioned earlier. I've placed a hint in the skeleton file for you about how to *call* these functions. Once your program reads the array data (via **getArray()**, which you will not modify), it should do the following things (these should all be done with 8 functions that you will have to define in your submitted file):

1. Print the array that it just read in. It does this using the function **printArray()**, which is declared in the **headers.h** file. You are required to do this using this function only. The function call should be all it takes to execute this task.

2. Find the *largest* integer in this array and return it. It does this using the function **maxArray()**, which is declared in the **headers.h** file. You are required to do this using this function only. Once this function is called, you should print out the result (see the example run below).

3. Find the *smallest* integer in this array and return it. It does this using the function **minArray()**, which is declared in the **headers.h** file. You are required to do this using this function only. Once this function is called, you should print out the result (see the example run below).

4. Will find the *sum of all the* integers in this array and return it. It does this using the function **sumArray()**, which is declared in the **headers.h** file. You are required to do this using this function only. Once this function is called, you should print out the result (see the example run below).

5. Find *all the even number integers* in this array and print them. It does this using the function **evensArray()**, which is declared in the **headers.h** file. You are required to do this using this function only. Once this function is called, you should print out the result (see the example run below).

6. Find *all the prime number integers* in this array and print them. It does this using the function **primesArray()**, which is declared in the **headers.h** file. You are required to do this using this function only. A prime number is a number that is only divisible by itself and 1. 1 is, itself, not considered a prime number (2 is, however). Once this function is called, you should print out the result (see the example run below).

7. Will search the array for *every* integer number in the global array `SEARCHES[]`. It will use **both** the function **SeqSearch()** and the function **AllSearches()** to do this, both of which are declared in the **headers.h** file. You are required to do this using these functions only. Here's what you have to pay mind to: **SeqSearch()** only finds ONE target in an array (the first incidence of the target). **AllSearches()** runs *all* the targets from the global array `SEARCHES[]`.

   Remember that the program must print out a mini-report about what it found and did not find (see the example run below).

   Here is an example run using the **ArrayFile.txt** data:

```
$ ./arrays
23, 22, -5, -21, 21, 0, -12, -55, 9111, 1233, 1, 3, 5, 6, 7, 8, 9, 12, -11, 17
Max = 9111
Min = -55
Sum = 10374
Evens: 22, 0, -12, 6, 8, 12, end
Primes: 23, 3, 5, 7, 17, end
Searches:
Looking for -5. Found at index: 2
Looking for -4. Not Found!
Looking for -3. Not Found!
Looking for -2. Not Found!
Looking for -1. Not Found!
Looking for 0. Found at index: 5
Looking for 1. Found at index: 10
Looking for 2. Not Found!
Looking for 3. Found at index: 11
Looking for 4. Not Found!
```

Note that the first line is a print out of the array, then it shows the maximum, the minimum, the sum, the evens, the primes, and the searches "mini-report". The integer array is the one that is read in at the start of the program (by the **getArray()** function) and the array of searched numbers is the one defined in the `SEARCHES[]` declaration.

Please make sure your output matches the above. For example, notice how the printout of the array has commas and a space between each element, except for at the last one. Also notice that the evens and primes lists also has commas and a space between each element and they end with the word "end".

REQUIREMENT: Do not create/define any functions other than the ones that are declared in the **headers.h** file. Do not change the function declarations or modify **headers.h**.

When you test your program, it will use the **ArrayFile.txt** data as your source. When you want to test your program *further*, you can create another data file (give it another name, like **MyFile1.txt** or something) with just integers in it (separated by whitespaces). You will have to modify the `MAXSIZE` and `FILENAME` global variables accordingly in the **constants.h** file (only modify these).

One more note: For this lab, other than following styling guidelines, I would like you to write comments in your code, as usual, BUT you don't have to do the Pre/Post-Condition commenting as it is done for you (in the **headers.h** file).

Step 3: Create a makefile and Compile the Codes with the make Command

In order to learn another way to manage our source codes and their compilations, we will first create a **makefile** and put in the usual g++ commands in it. Afterwards, whenever we want to compile our programs, the Linux command is a lot shorter – so this is a convenience.

Using your text editor, create a new file called **makefile** and enter the following into it:

```
arrays: arrays.cpp headers.h constants.h
        g++ -o arrays arrays.cpp -Wall -std=c++11
```

Then from the Linux prompt, you can issue one command that will compile this code, like so:

```
$ make
```

If the compilation is successful, you will not see any output from the compiler. You can then run your program like this:

```
$ ./arrays
```

If you encounter an error, use the compiler hints and examine the line in question. If the compiler message is not sufficient to identify the error, you can search online to see when the error occurs in general.

Remember to re-compile the relevant files after you make any changes to the C++ code.

Step 4: Submit using GRADESCOPE

BEFORE YOU SUBMIT YOUR FINAL VERSION FILES: Make sure that you:

a) **STYLIZE** your program appropriately to additionally make it easier on others reading your code. Apply ALL the style pointers I have posted on Piazza. This includes the use of comments.

This will be graded for an *extra 20 points* beyond the automatic grade you get from Gradescope.

b) **MEET THE REQUIREMENTS OF THIS LAB**. This means, if I said certain features or program aspects were *required* to be in your final code, then they must be there.

If these requirements are not met, you will lose MAJOR points on the exercise. This simulates a real-world situation (e.g. your customer wants certain features, so you must provide it – while I am not your customer, per se, I *am* giving you points for your efforts!)

Log into your account on **https://www.gradescope.com/** and navigate to our course site. Select this assignment. Then click on the "Submit" button on the bottom right corner to make a submission. You will be given the option of uploading files from your local machine or submitting the code that is in a github repo. Choose the first option and follow the steps to **upload arrays.cpp** to Gradescope.

You may submit this lab multiple times. You should submit only after you test it on your computer (or CSIL) and are satisfied that the programs run correctly. The score of the last submission uploaded before the deadline will be used as your assignment grade.

Step 5: Done!

Once your submission receives a score of 100/100 on the autograder, you are done with this assignment. REMEMBER that there are 20 additional points that will be scored according to your proper use of comments and styling. The total will then be normalized to 100 points again (i.e. 120 score will be 100%) to grade this lab.

WE WILL BE CHECKING FOR PLAGIARISM – DO NOT COPY FROM OTHER STUDENTS OR FROM SOURCES ONLINE! USE PROPER CITATION OF CODE THAT YOU DID NOT WRITE! THE CONSEQUENCES WILL - AT MINIMUM - BE A ZERO ON THIS LAB AND POSSIBLY A ZERO IN THIS COURSE AND YOU WILL BE REPORTED TO THE UNIVERSITY.