CSE30: Lab #3 – Searching and Sorting Arrays

Overview

This lab will explore ***searching and sorting arrays*** in the C++ programming language. The goal of this lab is to create searching and sorting programs running on an array of integers. The knowledge required to complete this lab will be everything learned so far, either in lecture or in the lab. Specifically, significant portions of code written for Lab 2 can be re-used for this lab.

When you use any code obtained from the Internet, you need to document the source.

Note: You need to have a separate program for each part of this lab. When you submit your assignment through CatCourses, make sure that ALL PARTS are included.

Getting started

Create a new directory in your main development directory (CSE030) called Lab_03. Try to use the terminal on your own, without getting help from the TA to setup the new directories (try to learn/remember the terminal commands).

Reminder: Once you have created a file and understood its content, you want to compile the source file (ex. main.cpp) into an executable so that you can run it on your computer. Note: EVERYTIME you modify your source file, you MUST re-compile your source file for the changes to take effect. You will compile from the command line by running:

```
g++ <source> -o <executable>
```

where g++ is a program (already installed on your Linux system) that compiles C++ source files, **source** is the source file for your program (main.cpp in this example), **-o** tells the compiler that you want to give the executable its own name, and **executable** is the name you want to give your program. In order to compile your first program, you would have to run:

```
g++ main.cpp -o main
```

Assuming that your program compiled successfully (i.e. no errors were found), you can run your program by typing **"./main"** in the terminal/console.

Good coding practices (worth 2 points!)

Writing code that is understandable by humans is as important as being correct for compilers. Writing good code will help you complete the code, debug it and … get good grades. It is very important to learn as soon as possible, because bad habits are hard to get rid of and good habits become effortless. Someone (guess who) reads your code will be in a better mood if it is easy to understand … leading to better grades! This lab will include 2 points (10% for code quality):

- Explanations with comments
- Meaningful names
- Indenting of blocks `{ }` and nesting …

- Proper use of spaces, parentheses, etc. to
- Visible, clear logic
- One / simple statements per line
- Anything that keeps your style consistent

(Exercise) Create – searchArray.cpp

As you will solve more complex problems, you will find that searching for values in arrays becomes a crucial operation. In this part of the lab, you will input an array from the user, along with a value to search for. Your job is to write a C++ program that will look for the value, using the linear (sequential) search algorithm. In addition, you should show how many steps it took to find (or not find) the value, and indicate whether this was a best or worst case scenario (… remember the lecture). You may refer to the pseudo-code in the lecture, but remember that the details of the output and error conditions are not specified there. Also, note that in pseudo-code arrays are sometimes defined with indices 1 to N, whereas in C++ the indexes span from 0 to N-1.

The program will work as follows. First, you should ask the user to enter the size of the array, by outputting:

"Enter the size of the array: "

to the console. If the user enters an incorrect size, you should output the error message:

"ERROR: you entered an incorrect value for the array size!"

and **exit the program**. If the input is a valid size for the array, ask the user to enter the data, by outputting

"Enter the numbers in the array, separated by a space, and press enter: "

Let the user enter the numbers of the array and store them into a C++ array. Up to now, this should be exactly the same code as Lab#2. Next, ask the user a value ***v*** to search for in the array (called the "key"), by outputting:

"Enter a number to search for in the array: "

to the screen, and read the key ***v***.

Search for ***v*** by using the linear search algorithm. If your program finds the key, you should output:

"Found value v at index i, which took x checks."

where ***i*** is the index of the array where ***v*** is found, and ***x*** is the number of search operations taken place. Hint: ***i*** and ***x*** are not the same, think about the starting value of ***i*** and starting value of ***x***.

If you doesn't find the key, then output:

"The value v was not found in the array!"

Then indicate whether you happened to run into a best or worst case scenario. In case you did, output:

"We ran into the best case scenario!"

or

"We ran into the worst case scenario!"

otherwise, don't output anything.

Example runs (input is in italic and bold):
Enter the size of the array: ***5***
Enter the numbers in the array, separated by a space, and press enter: ***1 5 9 7 3***
Enter a number to search for in the array: ***9***
Found value 9 at index 2 which took 3 checks.

Enter the size of the array: ***-5***
ERROR: you entered an incorrect value for the array size!

Enter the size of the array: ***5***
Enter the numbers in the array, separated by a space, and press enter: ***9 5 1 7 3***
Enter a number to search for in the array: ***9***
Found value 9 at index 0 which took 1 checks.
We ran into the best case scenario!

Enter the size of the array: ***5***
Enter the numbers in the array, separated by a space, and press enter: ***4 5 6 8 2***
Enter a number to search for in the array: ***2***
Found value 2 at index 4 which took 5 checks.
We ran into the worst case scenario!

(Exercise) Create – sortArray1.cpp

As you have seen in lecture, it is often necessary to sort an array in order to make searching faster. In this part of the lab, you will implement ***selection*** sort. The selection sort algorithm consists of traversing the collection of unsorted elements to find the maximum (or minimum) value so far, and to swap it with the last one that is unsorted, and then to repeat the process. There are multiple ways to implement the selection sort algorithm, in addition to deciding whether to sort the array in an ascending versus descending order. More specifically, you may choose to:

- Sort the array in an ascending versus descending order,
- Traverse the array from beginning to end, or vice versa, and
- Search for the minimum versus the maximum for the traverse.

The first part of this program, which reads an array from the user, is exactly the same as what you have done in the previous parts of this lab and lab#2. Specifically, you should ask the user to enter the size of the array, by outputting:

"Enter the size of the array: "

to the console. If the user enters an incorrect size, you should output the error message

"ERROR: you entered an incorrect value for the array size!"

and **exit the program**. If the input is a valid size for the array, ask the user to enter the data, by outputting

"Enter the numbers in the array, separated by a space, and press enter: "

Let the user enter the numbers of the array and store them into a C++ array. Up to now, this should be exactly the same code as Lab#2.

Now, sort the array using a selection sort algorithm of your choice. Once the array is sorted, output first to the console if the output is sorted in ascending or descending order (depending on what you decided to use):

"This is the sorted array in <ascending or descending> order:"

and then output the array in a newline. Also, write if you chose the max or min for the traverse:

"The algorithm selected the <minimum or maximum> for the traverse of the array."

Example runs (input is in italic and bold):
Enter the size of the array: ***5***
Enter the numbers in the array, separated by a space, and press enter: ***4 6 8 2 5***
This is the sorted array in descending order: 8 6 5 4 2
The algorithm selected the maximum for the traverse of the array.

(Exercise) Create – sortArray2.cpp

In this part of the lab, you will write a program that uses a **different implementation** of selection sort from the one you chose before:

- if you had chosen to search for the minimum for the traverse of the array, now choose the maximum, or
- if you had chosen to search for the maximum for the traverse of the array, now choose the minimum.

You may choose whether to traverse the array forward or backward.

Also, this time you are requested to calculate the **number of swaps** used to complete sorting the array.

You can write a program starting from the previous one, and modify it to sort the array using another selection sort implementation. Once the array is sorted, output first to the console if the output is sorted in ascending or descending order (depending on what you decided to use):

"This is the sorted array in an <ascending or descending> order:"

and then output the array in a newline. Also, write if you chose the max or min for the traverse (which should be the opposite of sortArray1.cpp):

"The algorithm selected the <minimum or maximum> for the traverse of the array."

Finally, write how many swaps were used to sort this array. In the next line, output:

"It took x swaps to sort the array..."

where ***x*** is the number of swaps calculated by the program. Hint: declare a counter that increments whenever a swap takes place.

Example runs (input is in italic and bold):
Enter the size of the array: ***5***
Enter the numbers in the array, separated by a space, and press enter: ***4 6 8 2 5***
This is the sorted array in an ascending order: 2 4 5 6 8
The algorithm selected the minimum for the traverse of the array.
It took 3 swaps to sort the array...

What to hand in

When you are done with this lab assignment, you are ready to submit your work. Make sure you have included the following ***before*** you press Submit:

- Your **searchArray.cpp**, **sortArray1.cpp**, **sortArray2.cpp**.
- Documentation (in a text file) of code you used from the internet. You may want to cut-and-paste the original code as well.
- A list of collaborators.