CSE30: Lab #5 – Binary Search

Overview

This lab will explore ***Binary Search*** and put together various building blocks covered in previous lectures and labs to implement a binary search in an array. 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 the previous Labs can be reused for this lab.

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 (probably on Desktop/CSE30) called Lab_05. 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 1) Create – fileIO.cpp

In this part of the lab you will write a program that does the following operations:

- Reads words from a file (**words_in.txt**), with one word per line.
- Creates an array, using dynamic memory allocation, large enough to contain those words (one word per array element).
- Output the words (array elements) to another file (**words_out.txt**).

Before storing the words from a file in an array, you will need to first find out how many lines there are in the file: one way to do it is to check with a **counter** when you reach the end of the file, by using the function ***eof().*** eof() returns ***true (1)*** if the end of file is reached, otherwise a ***false (0)*** is returned. You can review the use of eof() at:

http://mathbits.com/MathBits/CompSci/Files/End.htm

Once you have found out how many lines there are in the file, you can create a **dynamically allocated array of strings**, which is as large as the number of words contained in the file (as per the counter you have implemented). You can review the syntax to declare arrays with dynamic memory allocation (***pointer / new***) in the lecture slides or at cplusplus.com.

Finally, you will write each word (array element) into the output file, one word per line.

Note:

- You do not need to declare an array with a pre-defined constant size nor ask the user to input the number of elements.
- Make sure that you create an input file with only one word per line.

(Exercise 2) Create – checkArray.cpp

In this part of the lab, you will create a function ***checkArraySort*** that checks if an array of strings (which was previously defined and initialized) is already sorted, and it is either increasing or decreasing. The function that you create is declared as follows:

- Input arguments:
    - A string array ***A*** **(remember that array is a pointer in this case, so use the * sign)**

- an integer variable ***array_size*** for the number of elements in array ***A*** (the counter value from previous exercise)

- Return in integer value:
  - -1 if the array is sorted in descending order
  - 0 if the array is not sorted
  - 1 if the array is sorted in ascending order

You can modify the main program that you developed in ***fileIO.cpp*** to call ***checkArraySort***, and to print out one of the following statements:

- If -1 "The array is sorted in descending order!"
- If 0 "The array is not sorted!"
- If 1 "The array is sorted in ascending order!"

You can use the same **words_in.txt** to test your program.

Note: In order to check the sorting of strings / words, you can use the comparison operators (<, <=, ==, >, >=) the same way you would compare numbers.

(Exercise 3) Create – searchArray1.cpp

In this part of the lab, you will create a **recursive function** that **searches** for a **key** in an array with a **binary search algorithm**. Revisit lecture# 4 for the concept of the binary search algorithm. All you need to do is to repeat splitting an array by half and compare the key to the value of the middle element.

In main program:

- Input an array from a file.
- Call function (***checkArraySort***) to check if the array is sorted. So far, you can use the code you created from the previous exercises.
- Exit program if the array is not sorted, or continue if it is. Therefore, you can use the code that you built so far (exercise 2), and continue to the next steps if the array is sorted.
- Prompt the user to input search key (***k***).
- Call function (***binarySearchR***) to search for the key recursively.
- Output your search result:
  - **"Found key *k* at index *i*!"** if the key was found.
  - **"The key *k* was not found in the array!"** if the key was not found.

Your ***binarySearchR*** function will return an integer value ***i*** as the index of the array ***A*** where the key is found (or -1 in key is not found), and it will take the following arguments:

- a string array ***A*** (again, it is a pointer)
- the number of elements in the array
- the key to search for (a string)

Before writing your ***binarySearchR*** function, think about the algorithm and write it in pseudocode using a piece of paper or a text editor. You need to turn in the pseudocode to receive full credit.

(Exercise 4) Create – searchArray2.cpp

In this part of the lab, you will create a **search function (*binarySearchL)*** that is similar to exercise 3 with the following change:

- You will implement the binary search algorithm using a **loop** instead of a recursive function.

This time all you need to do is to call a search function you create with the following arguments:

- a string array **A** (again, it is a pointer)
- the number of elements in the array
- the key to search for (a string)

Return:

- -1 if the key was not found
- index of the (first) element where you found the key.

Your program should behave the same way as searchArray1.cpp in the previous exercise; therefore, you may reuse the main function in searchArray1.cpp

Hint: You may use the same logic to find the beginning, end, and mid-point of an array you used for the recursive binary search. You will use a loop to compare the values of the array to the key until the calculated mid-point is the last element.

Before writing your ***binarySearchL*** function, think about the algorithm and write it in pseudocode using a piece of paper or a text editor. You need to turn in the pseudocode to receive full credit.

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 **fileIO.cpp**, **checkArray.cpp**, **searchArray1.cpp**, **searchArray2.cpp,** and a list of Collaborators.
- File attachments of the pseudocode you have created in this lab. If you write your pseudocode on papers, scan them as images and attached the images. If you write your pseudocode in a text editor, save them as text files and attach them with your submission.
- Documentation (in a text file) of code you used from the internet. You may want to cut-and-paste the original code as well.