CPSC 350: Data Structures and Algorithms

Spring 2023

Programming Assignment 3: Do You See What I See?

Due: See Canvas for the due date

Please fill out this form if you have any inquiries on your PA grades. This request will go to our grader. If you have a personal question, please message your instructor first, and they'll let you know how to proceed.

Clarification

If this is your input:

```
BEGIN
81.4 121.4 50.2 75.3 75.1
90.0 90.5 80.2 60.3 50.2
80.8 80.3 90.7 89.9 94.7
82.2 72.1 92.6 54.4 95.1
END
```

This would be the Output:
In column 0 there are 2 that can see. Their heights are: 81.4, 90 inches.
In column 1 there are 1 that can see. Their heights are: 121.4 inches.
In column 2 there are 4 that can see. Their heights are: 50.2, 80.2, 90.7, 92.6 inches.
In column 3 there are 2 that can see. Their heights are: 75.3, 89.9 inches.
In column 4 there are 3 that can see. Their heights are: 75.1, 94.7, 95.1 inches.

So, each row in the input is a row in the audience and the stage would be where BEGIN is. Hope this helps/ makes sense 😊

The Assignment

As part of a big celebration, a company has paid an exorbitant amount of money to fly in a celebrity speaker to give a motivational speech to their employees. The event will take place in a conference room with a podium at the front of the room and a fixed number of chairs arranged in an N x P rectangle, with row 0 being the closest to the speaker.

To avoid commotion, the event planners have decided that all seating will be assigned. However, there is a fear that shorter audience members may not be able to see the celebrity speaker if people who are sitting in front of them (in their column of seats) are taller than them. And so they come to you for help with the problem. They would like to provide you with a text file that consists of candidate seat assignments in the form of N lines, each with P entries, where each entry is the height of the person assigned to the seat at that location, represented

as a floating point number. For each column of seating, they would like to determine how many people have an unobstructed view of the speaker. It's up to you to determine this as efficiently as possible.

Solving the Problem with Stacks

A monotonic stack is a stack whose elements appear in either monotonically increasing or decreasing order from the bottom of the stack to the top of the stack. Duplicate values are not allowed in the stack. As with a traditional stack, a monotonic stack is a LIFO data structure. The only difference is the push function, which must ensure that the values remain in monotonically increasing or decreasing order. This means that for a monotonically increasing stack, to push a value of X on the stack, we must first pop and discard all values in the stack that are larger than X. Similarly, for a monotonically decreasing stack, to push a value of X on the stack, we must first pop and discard all values in the stack that are smaller than X. All other operations stay the same.

Your first job is to build a templated implementation of a monotonic stack. This class will be called *MonoStack*. You may use the array-based stack code from the class as a starting point. Your constructor for the stack should take the initial size, as well as a character, *o*, as parameters. If o=='i', the stack should be monotonically increasing. If o=='d', the stack should be monotonically decreasing. Your implementation should handle any error conditions appropriately for the various methods, which should consist of all the methods we implemented in class for our traditional stack.

💭Monotonic means "One Way Only." The stack must always stay in decreasing or increasing order.

The Program

Your goal is to build a class, *SpeakerView*, that can take a plain text file that consists of N+2 lines. The first line of the file will be the string "BEGIN." The last line of the file will be the string "END." Each of the N lines between the first and last will consist of P doubles (representing heights) separated by spaces. Your program will then use a **monotonically decreasing stack** to determine the number of people in each column of seats that are able to see the speaker clearly, as well as the heights of those people. Your output can be displayed to the terminal but should be descriptive so it is easy to understand. Your *SpeakerView* class should be as OO as possible.

You will also have a main.cpp file that consists only of a main method which takes the input file name as a command line argument and then uses the *SpeakerView* class to compute the output.

Rules of Engagement

- You may **NOT** use any non-primitive data structures (eg. vectors, lists, etc) other than your own implementation of a monotonic stack to solve the problem. Of course, to do the file processing you may use any of the standard C++ IO classes.
- For this assignment, you must work individually.
- Develop using VSCode and make sure your code runs correctly with g++ using the course docker container.
- Feel free to use whatever textbooks or Internet sites you want to refresh your memory with C++ IO operations, just cite them in a README file turned in with your code. All code you write, of course, must be your own. In your README, please be sure to include the g++ command for compiling your code.

Due Date

This assignment is due on the day and time specified on Canvas. Submit all your commented code as a zip file to Canvas. The name of the zip file should be LastName_FirstInitial_A3.zip

Grading

Grades will be based on correctness, adherence to the guidelines, and code quality (including the presence of meaningful comments). An elegant OO solution will receive much more credit than procedural spaghetti code. I assume you are familiar with the standard style guide for C++, which you should follow. (See the course page on Canvas for a C++ style guide and Coding Documentation Requirements.)

Code that does not follow the specification EXACTLY will receive an automatic 25% deduction. Code that does not compile will receive an automatic 50% deduction.

Readme & References

More info on Canvas!

- **README file:** All source code will be accompanied with a plain text README file. We encourage students to take advantage of markdown technology if they are inclined to do so. This file will contain:
    - The following identifying information:
        - Full name
        - Student ID
        - Chapman email
        - Course number and section
        - Assignment or exercise number
    - A list of all source files submitted for the assignment
    - A description of any known compile or runtime errors, code limitations, or deviations from the assignment specification (if applicable)
    - A list of all references used to complete the assignment, including peers (if applicable)

    - Instructions for running the assignment. (Typically applicable in advanced courses using build systems or third party libraries) for us it would be something like:
        - To compile: g++ *.cpp -o A1.exe
        - To run: ./A1.exe input.txt output.html
- **In-code citations:** You are allowed to use small, isolated lines of code from external sources in your programming assignments as long as they are appropriately cited. Any time you are including code that you did not write yourself, it must be cited. This includes sources from StackOverflow and similar forums, other current or previous students, tutors, books, tutorials, etc. You should wrap the copied code in a comment denoting the start and end of said code, like this:

```
# code that is not copied (you wrote this)
someCode = 0;
print("hi I wrote this!")

"""BEGIN CODE FROM SOURCE: link/name of source"""
print("I did not write this");
"""END OF CODE FROM SOURCE: link/name of source"""

"""BEGIN CODE FROM CHAT GPT, PROMPT ASKED: how do you ...?
"""
print ("I did not write this");
"""END OF CODE FROM CHAT GPT"""

# back to code that you personally wrote
codeIWrote = 10;
```

- **Info in code files:** Additionally, all source files will start with a header comment containing the following items (one per line):
    - Full name
    - Student ID
    - Chapman email
    - Course number and section
    - Assignment or exercise number