ICS 46 Lecture B Fall 2019 Project 2: Students on the Bus

Due **November 3, 2019, 11:59 PM**

Note that this is well later than the syllabus originally stated. Project 3 has been moved to have a checkpoint due November 11 and a due date of November 16.

Introduction

We are planning a bus trip. We have some number of people who are going to ride the bus. The number of people going is an even number. Half of them want to sit on an aisle seat and half want to sit in a window seat. These are the only two types of seats.

Everyone who wants a window seat has ranked the "aisle seat people" from most-preferred to least-preferred to sit next to, and the people who prefer an aisle seat have done similar for their window-preferring friends. Our goal is to have an *acceptable* seating arrangement, where an *unacceptable* seating arrangement is one where there is a person *A* assigned to an aisle seat and a person *W* assigned to a window seat, not sitting next to person A. However, person A would prefer to sit next to W instead of whoever A is paired with, and W would prefer to sit with A instead of whoever W is paired with.

Any seating arrangement with no unacceptable situations is acceptable.

We need only come up with pairings: who is sitting next to who. We do not need to decide which row or side of the bus.
This sounds like a tough problem, but fortunately, there is a good way to arrange the seating:

- Initially, mark all people as unassigned.
- Start a queue of everyone who wants a window seat.
- While that queue has non-zero people in it:
    - Consider the first person *W* in the queue.
    - Consider the first person *A* in *W*'s preference list who W has not yet interacted with during this program.
    - If A does not currently have a buddy to sit with, pair W and A (tentatively)
    - If A does have a seat buddy[1], but prefers W to whoever the current seat buddy is:
        - Pair A and W (tentatively)
        - Take whoever A was paired with and put him or her back in the queue
    - If A prefers the current seat buddy to W,
        - Put W back in the queue of unmatched window people.

Believe it or not, if this is done correctly, this will *always* lead to an acceptable seating arrangement! Don't believe me? Look in the Google test directory, write out one or both of the examples given, and walk through them by hand here. That's just anecdotal, of course.

[1] This means they are seated adjacent, or sharing a bench if it's a school bus.

Getting Started

Before you begin work on this project, there are a couple of chores you'll need to complete on your ICS 46 VM to get it set up to proceed.

Refreshing your ICS 46 VM environment

Even if you previously downloaded your ICS 46 VM, you will probably need to refresh its environment before proceeding with this project. Log into your VM and issue the command ics46 version to see what version of the ICS 46 environment you currently have stored on your VM. Note, in particular, the timestamp; if you see a version with a timestamp older than the one listed below, you'll want to refresh your environment by running the command ics46 refresh to download the latest one before you proceed with this project.

If you're unable to get outgoing network access to work on the ICS 46 VM — something that afflicts a handful of students each quarter — then the ics46 refresh command won't work, but an alternative approach is to download the latest environment from the link below, then to upload the file on to your ICS 46 VM using SCP. (See the Project #0 write-up for more details on using SCP.) Once the file is on your VM, you can run the command **ics46b refresh_local NAME_OF_ENVIRONMENT_FILE**, replacing **NAME_OF_ENVIRONMENT_FILE** with the name of the file you uploaded; note that you'd need to be in the same directory where the file is when you run the command.

The file is linked from the “public” ICS 46 page; click this link and enjoy the amazing web design skill that put it together: https://www.ics.uci.edu/~mikes/ics46/

Creating your project directory on your ICS 46 VM

A project template has been created specifically for this project, containing a similar structure to the basic template you saw in Project #0.

Decide on a name for your project directory, then issue the command **ics46b start YOUR_CHOSEN_PROJECT_NAME project1** to create your new project directory using the project1 template. (For example, if you wanted to call your project directory proj1, you would issue the command ics46 start proj1 project1 to create it.) Now you're ready to proceed!

Reviewing related material

I encourage you to read your textbook; in the second edition, section 5.2 deals with queues. Your book is good at getting to the point, so this should not be a long read. Furthermore, you should look at your notes from the associated lectures. Chapter 3 deals with various ways to implement these, including 3.2 which covers singly-linked lists.

Requirements

- Fill in LLQueue.hpp
    - Your implementation must be based on a LinkedList.
    - Your implementation must fit the interface given
    - Your implementation must be templated
    - When attempting to access the front of an empty queue, or to dequeue from an empty queue, you must throw a QueueEmptyException as appropriate.
    - Do not throw an exception upon addition of an element: there is no capacity (beyond what memory allows).
    - **Please note:** the template will not build successfully until you have at least stubbed code for the Queue functions.

    - You do not need to write a copy constructor or an assignment operator, but knowing how to do so is generally a good thing.

- Write function `std::map<int, int> assignBusSeats(std::istream & in)` in `proj2.cpp`
    - This function will read a problem statement from the given istream. The format is described on the next page.
    - You can treat istream just like `std::cin` -- there is some sample code given. Please let me know if it is insufficient, I can explain more.
    - The return value is a mapping, where map.at(i) tells me which numbered aisle person the *i*th window person is sharing a bench with.

- Your implementation does not have to be the most efficient thing ever, but it cannot be "too slow." In general, any test case that takes over a minute on the grader's computer may be deemed a wrong answer, even if it will later return a correct one.

Input Format

The input is formatted in the following fashion:

- The first line of input is half the number of people who are going for the bus trip.
    - For example, if we have 8 people going, then there are 4 window seats and 4 aisle seats, so our first line is 4.
- Call that value *n* for purposes of this format.

- The next *n* lines of input each contain a permutation of integers *1* through *n*, separated by a space
    - The *i*th line (start counting at 1) contains the preference list for window person #i.
    - If the line is "3 2 1" this means that this person prefers to sit with aisle person #3, has aisle person #2 as their second choice, and aisle person #1 as their last choice.
- The next *n* lines of input each contain a permutation of integers *1* through *n*, separated by a space, representing the *i*th aisle person's preference list.

- Every input we give you for this assignment will be properly formatted; you do not need to check that we have provided a permutation, or that a person has not been repeated in someone's preference list, or that we have omitted someone, or that someone is lying about their preferences.

You are **explicitly permitted** to use std::set, std::list, and std::vector if you so choose. You are pretty much required to use std::map. You are welcome to ask anything you want about these libraries, or to look up material about them online. Information about how to use an explicitly-permitted library may always be shared among classmates, but refrain from telling one another how you solved a problem in the assignment with them. For example, answering "how do I check if an element is in a std::set?" is great and encouraged, while answering "what did you use std::set for in your project?" is not.

A good reference for the STL container classes (such as those listed above, including std::map) is http://www.cplusplus.com/reference/map/map/ .

Deliverables

After using the gather script in your project directory to gather up your C++ source and header files into a single **project2.tar.gz** file (as you did in previous assignments), submit that file (and only that file) to Checkmate. Refer back to Project #0 if you need instructions on how to do that. This time, it should give you the correct file name, insert innocent-looking face emoji here.

You will submit your project via Checkmate. Keep in mind that that you're responsible for submitting the version of the project that you want graded. We won't regrade a project simply because you submitted the wrong version accidentally. (It's not a bad idea to look at the contents of your tarball before submitting it; see Project #0 for instructions on how to do that.)

Can I submit after the deadline?
Yes, it is possible, subject to the late work policy for this course, which is described in the section titled Late work in the course reference and syllabus.

Grading
Your grade for this project will be two-thirds correctness: I will run some number of test cases using Google test. Each is worth some number of points and is graded based on whether or not your code correctly determines if the puzzle has a solution, and if so, what it is. If it is determined that your program does not make an attempt to solve the problem at hand, you will not get these points, regardless of the result from testing. The tests will look a lot like the tests in your Google Test starting directory for this assignment; if you pass those, you're off to a good start, but it's not a guarantee.

The other one-third is style. ICS 46 isn't strictly about C++, but good programming style is important. Your important variable names should have meaningful names, your code should be appropriately commented, and so on.