CPSC 350: Data Structures and Algorithms

Spring 2024

Programming Assignment 4: The Waiting Game

Due: April 19th, 2024 at 11:59 pm

The Assignment

Imagine that the student services center (SSC), which houses the registrar, cashier, and financial aid office, has approached you with the following issues. At certain times during the day, students arrive in large numbers, each needing to visit one or more offices. If there are not enough windows open at each, the average student wait times are too long, and they receive many complaints. On the other hand, if they open too many windows, their staff will be idle for most of the time, and they will be wasting money. They want you to program a simulation that will allow them to calculate metrics on student wait times and window idle times given a specific traffic flow of students through the SSC.

The Simulation

You will model the above problem to meet the SSC's requirements. As students arrive in the SSC, they will go to one of the offices they need to visit and wait in a queue. When a window becomes available, they will exit the queue and remain at the service window for however long they need. They will then move on to the next office they need to visit and repeat the process until they have completed all their tasks. Your first job will be to build a generic doubly-linked list data structure, from scratch, supporting the usual operations, which we have largely completed in class. You will then use this linked list to build a generic queue data structure, supporting the usual operations. You may not use a queue or linked list from the C++ STL libraries. You will then use your queue implementation to implement the simulation.

Input will be in the form of a command line argument that specifies the location of a text file. The text file will define at what times students arrive, and what offices they will visit and in what order. It will have the following format. The first line will be the number of windows open at the registrar. The second line will be the number of windows open at the cashier. The third line will be the number of windows open at the financial aid office. The next line will be the time (or clock tick) at which the next students arrive. The next line will be the number of students that arrive at that time. The following lines will be the amount of time each student needs at the registrar, cashier, and financial aid office, respectively, as well as the order they visit them. This is followed by the next clock tick, number of students, etc. For example, and input of:

```
2
3
1
1
2
5 1 2 R C F
```

10 5 1 F R C
3
1
4 2 6 R C F

Means that 2 windows will be open at the registrar, 3 at the cashier, and 1 at the financial aid office. At time 1, 2 students arrive. The first will need 5 minutes at the registrar, 1 minute at the cashier, and 2 minutes at financial aid, in that order. The second student will need 10 minutes at financial aid, 5 minutes at the registrar, and 1 minute at the cashier, in that order. Then, at time 3, 1 student arrives and needs 4 minutes at the registrar, 2 minutes at the cashier, and 6 minutes at financial aid.

When students arrive at the same time and visit the same office first, assume that the student listed first in the text file is also the first to get into line. If 2 students finish at different offices at the same time and head to the same office next, assume that they enter the line based on alphabetical order of the office they are coming from (eg. cashier always beats financial aid which always beats registrar).

The simulation will start at time 1, and run until all student requests have been addressed, meaning the queue at each office is empty and no more students are going to arrive.

At the end of the simulation, your program will display (on standard out) the following (labeled) metrics:

1. The mean student wait time for each office.
2. The longest student wait time for each office.
3. The number of students waiting over 10 minutes total across all offices.
4. The mean window idle time for each office
5. The longest window idle time for each office
6. Number of windows idle for over 5 minutes across all offices.

The Design

Your simulation will consist of the following classes (bolded classes are required):

- **DblList** - the doubly linked list template class
- **ListQueue** - the list-based queue template class
- **ServiceCenter** - models the service center
- **Office** - models an individual office in the service center
- **Window** - models a single window in one of the offices
- **Customer** - models a student coming into the service center
- Any other classes you want to create for the simulation

You should also provide a main.cpp to run the program, which takes the input specification file as a command line argument.

Rules of Engagement

- You may **NOT** use any non-primitive data structures other than what we have implemented in class. You may not use any data structures from the C++ STL. Of course, to do the file processing you may use any of the standard C++ IO classes.
- **For this assignment, you may work in groups of AT MOST 2 students.**
- 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 at 11:59 pm on 4-19-2024. Submit all your commented code as a zip file to canvas. The name of the zip file should be LastName_FirstInitial_A4.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.