CPSC 350 – Data Structures
Fall 2019
Assignment 4 - Registrar's Office Simulation
Due: Nov. 6, 2019 11:59pm

Overview

Now that we have covered queues and linked lists, we will apply these data structures to a simulation problem that will make extensive use of them.

Imagine that the registrar's office has approached you with the following issues. At certain times during the day students arrive in large numbers. If there are not enough windows open, 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.

The Specifics

You will model the above problem to meet the registrar's requirements. As students arrive in the registrar's office, they will 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. Your first task will be to build a generic doubly-linked list data structure, from scratch, supporting the usual operations. 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 STL libraries. You will then use your queue implementation to finish the problem. (Note that in the spirit of reusability, you should have 3 classes…one for the list implementation, one for the list interface (pure virtual base class) and one for the queue. Of course, you'll need additional classes to model the rest of the assignment.)

Input will be in the form of a command line argument (./myprog test.in) that specifies the location of a text file. The text file will define at what times students arrive and will have the following format. The first line will be the number of windows open. 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 a window. This is followed by the next clock tick, number of students, etc. For example, and input of:

```
5
1
2
5
10
3
1
4
```

Means that 5 windows will be open. At time 1, 2 students arrive. One will need 5 minutes at a window, and the other 10. Then, at time 3, 1 student arrives and needs 4 minutes at a window, etc, etc.

When students arrive at the same time, assume that the student listed first in the text file is also the first to get into line. In the above, at time 1 there will be 2 students that arrive. The student requiring 5 minutes will get into line before the student requiring 10 minutes.

The simulation will start at time 0, and run until all student requests have been addressed, meaning the queue is empty and no more students are going to arrive. (This should tell you the main body of the program is going to be a huge loop.)
At the end of the simulation, your program will display (on standard out) the following (labeled) metrics:

1. The mean student wait time.
2. The median student wait time.
3. The longest student wait time.
4. The number of students waiting over 10 minutes
5. The mean window idle time
6. The longest window idle time
7. Number of windows idle for over 5 minutes.

The Rules

-You may work in pairs on this assignment.
-All code submitted must be your own.
-Your queue and doubly-linked list implementation must be coded from scratch. Feel free to use whatever books you like for reference, but please cite them in a README.
-You may develop on any platform you wish, but make sure your program compiles and runs the way you want it to using g++ within your Docker container.

Deliverables

Submit ONLY your commented source code , Makefile, README, etc to you github repository and provide the link in Blackboard. The README should include any special instructions for running.

Due Date:

This assignment is due at 11:59pm on 11-06-2019.

Grading:

Grades will be based primarily on correctness, adherence to the above guidelines, and elegance of implementation (an OO, modular design as opposed to procedural spaghetti code). I will also take into consideration comments and coding style.

Acknowledgement:

This program is based on a similar assignment by Dr. Drew Moshier.