EECS402, Fall 2018, Project 5

Overview:

For this project, you will start by updating some of the data structures you implemented in project 4. Once the data structures we need are ready, you will implement a small event-driven simulation of a fast food restaurant. The simulation will allow us to play "what if" games to see how a change we might make could affect overall statistics for our restaurant business.

The specifications for the simulation are very vague, and that is done on purpose. The idea for this project is for you to come up with a solution entirely your own, including the planning, design, implementation, and testing. Despite the vagueness, it is critical that you pay close attention to the specific things that are required. We will be enforcing those requirements, and if your implementation is not designed the way the requirements call for, your program will be considered to be incorrect.

Due Date and Submitting:

This project is due on **Friday, December 7 at 4:00pm**. Early submissions are allowed, with corresponding bonus points, according to the policy described in detail in the course syllabus

For this project, you must submit several files. You must submit each header file (with extension .h), each source file (with extension .cpp), and each template implementation file (with extension .inl) that you create in implementing the project. In addition, you must submit a valid UNIX Makefile, that will allow your project to be built using the command "make" resulting in an executable file named "proj5.exe". Also, your Makefile must have a target named "clean" that removes all of your .o files and your executable (but not your source code!).

When submitting your project, be sure that **every** source file (.h, .cpp, and .inl files!!!) and your valid Makefile are attached to the submission email. The submission system will respond with the number of files accepted as part of your submission, and a list of all the files accepted – it is ***your responsibility*** to ensure all source files were attached to the email and were accepted by the system. If you forget to submit a file on accident, we will not allow you to add the file after the deadline, so please take the time to carefully check the submission response email to be completely sure every single file you intended to submit was accepted by the system.

Detailed Description:

This project will be split up into two phases. It is very strongly recommended that you perform phase 1 fully, including exhaustive testing, before moving on to work on phase 2.

Phase 1:

In phase 1, you will update some of the data structures you developed in project 4 to be "templated" – that it, to be associated with a generic data type using C++ templates such that the data structures are able to hold more data types than just integers.

The SortedListClass and the FIFOQueueClass must be modified to be templated classes. Most likely, you will not be using the LIFOStackClass in this project, so it is not required that you make it a templated class or include it in your submission. If you find a need for a LIFOStackClass in your implementation and want to make use of it, you may do so.

The design, names and functional interface to your templated data structure classes must match those developed in project 4 exactly. In other words, you may not add new member functions, add new member variables, change the number or meaning of function parameters, etc. The only difference between the data structures developed in project 4 and the modified versions developed in project 5 will be that the project 4 versions are specifically tied to integers, while the project 5 versions are templated.

Phase 1 of this project should NOT result in writing much "new code", except for developing test cases to make sure your updated data structures work with different data types.

Phase 2:

Once your data structures are implemented and tested, you will develop an event driven simulation. Create an event class that will be inserted into a SortedListClass in a sorted way based on the time that the event is scheduled to occur. Then, handle one event at a time, as discussed in lecture. As you handle certain events, new events will be generated to occur at a future time (randomly drawn from a specified distribution), and the simulation will advance much like the airport example demonstrated in class. Note: you must use the data structures you developed in project 4 and "templatized" in this project – do not use any STL containers when implementing this project.

The simulation to implement will be a server simulation at a fast food restaurant. Some important requirements or additional information about the simulation are:

- Customers come into the restaurant on a pseudo-random basis, where each customer enters the restaurant some amount of time after the previous customer. The amount of time between customers should be drawn from a *uniform distribution*, which has specified min and max values.
- Your restaurant will have only a single server, in order to make things much simpler. If the server is not currently waiting on a customer, a newly arriving customer can immediately start being served. Otherwise, the customer will have to wait in a first-in-first-out queue and wait their turn.
- The amount of time it takes for the server to wait on a customer will be drawn from a *normal distribution* with a specified mean and standard deviation.
- Some customers don't like to wait in line. If this kind of customer arrives at the restaurant and sees a long line, they leave the restaurant and go to one of our competitors instead (that is, they do not get in line, and they do not get served, etc.). This will be described at run time using two values – the percentage of customers that don't like excessively long lines, and the length of the line to be considered excessively long. For example, we might say "40% of our customers won't wait in excessively long lines, and they consider a line of more than 10 people as excessively long". When a customer arrives, if they are a customer that doesn't like long lines (40% chance in this example) and the line is seen to be excessively long (greater than 10 customers long in this example), the new customer will leave.
- The restaurant will close at some specified time. We are nice restaurant owners, though, and will continue to serve anyone who is waiting in line after closing time. That is, anyone who is already in line at closing time will get served, but no additional customers will be allowed to get in line after the closing time.

This simulation ***MUST*** be implemented as an event-driven simulation as described in lecture. Note that the word "must" is all caps, bolded, underlined, and italicized! If your simulation is implemented as a time-driven simulation (or any variant on a time-driven simulation) or you otherwise generate and handle events in a way that is not the way described in lecture for an event-driven simulation, you will not receive credit for your simulation.

Following are some important requirements about how the simulation itself must work:

- The SortedListClass will contain events and act as a priority queue that contains your "event list" as described in lecture
- When the server is busy, you must utilize a FIFOQueueClass to allow a line to form
- You must generate new objects only when you need them. That is, do NOT generate a full list of customers and their arrival times at the beginning of the simulation. Instead, generate a single customer to arrive at a determined time to start off the simulation. Then, when handling that arrival event, determine when the next customer will arrive and generate a new arrival event. That is (please note!):
  - There must only be at most one customer arrival event in the event list at any given moment
  - At the time you handle a customer's arrival event, determine how far in the future the next arrival event occurs using BOTH a minimum and a maximum (do NOT assume a minimum of 0).

    - For example, if a customer's arrival time is 100, then, when you are handling that event, you will generate the next arrival event. If the uniform distribution has a min of 10 and a max of 20, then the next arrival event must be within the range 110 to 120 (as opposed to 100 to 120).
- Similarly, do not compute a customer's service time until that customer begins being served (in other words, do not compute all the timing for a customer right when the customer is first created – instead, compute the duration of the customer's service time only when the customer reaches the front of the line and you're able to compute when the time the service will end).
- Your simulation must provide enough output to allow a user to easily follow and understand what is happening at the restaurant. For example, you should print a description when a customer enters the restaurant, whether the customer is served or has to wait in line or leaves due to excessive line length, how many people are in front of the customer in the line, when the server finished serving a customer, etc., etc., etc. I should be able to look at your simulation's output and "see" what is happening in the restaurant and follow a customer through the process. The simulation should run from time = 0 until all events are handled, after which the simulation ends and maintained statistics are output.
- You must maintain data to output some important statistics upon completion of the simulation. The following statistics are required to be output:
    - Total number of customers that showed up
    - Percentage of time the server was busy helping customers
    - Percentage of customers had to wait in line
    - The number of customers that arrived, but left due to excessive line length
    - The longest the line was throughout the simulation
    - Come up with at least 2 more statistics you think might be interesting and include them in your output. Your statistics must be different than the above required statistics.

"Specific Specifications"

These "specific specifications" are meant to state whether or not something is allowed. A "no" means you definitely may NOT use that item. We have not necessarily covered all the topics listed, so if you don't know what each of these is, it's not likely you would "accidentally" use them in your solution. Those types of restrictions are put in place mainly for students who know some of the more advanced topics and might try to use them when they're not expected or allowed. In general, you can assume that you should not be using anything that has not yet been covered in lecture (as of the first posting of the project).

- Use of Goto: No
- Global Variables / Objects: No
- Global Functions: As needed
- Use of Friend Functions / Classes: No
- Use of Structs: No
- Use of Classes: Yes – required!
- Public Data In Classes: No (all data members must be private)
- Use of Inheritance / Polymorphism: No
- Use of Arrays: If needed
- Use of C++ "string" Type: Yes
- Use of C-Strings: No
- Use of Pointers: Yes – required!
- Use of STL Containers: **No** (No!!!)
- Use of Makefile / User-Defined Header Files / Multiple Source Code Files: Yes – required!
- Use of exit(): No
- Use of overloaded operators: Yes – required!
- Use of float type: No