Module 5 Lab – Stacks and Queues

Government Queues

For this lab, you are to simulate what we shall call a corrupt office. Since the government has recently enforced a cashless method of toll payments for our expressways, obtaining an RFID is a must. To avail of this service, the client has to line up and wait for his/her turn. Once it is the client's turn, he/she gets served and leaves the line, with the RFID system installed in his/her vehicle. Now, this office serves two types of clients: regular clients and VIP clients.

As this is a corrupt office, if the office supervisor is present, then the clients line up accordingly. The client comes in and queues up at the end of the line and waits for his/her turn. The client at the front of the line is served first.

If the supervisor is not present, then two lines are made. The first line acts like a queue and is for regular clients while the second line acts like a stack and is for VIP clients. If a regular client comes in, then he/she will have to go to the queue and wait for his/her turn to be served. When a VIP client comes in, then he/she will have to go to the stack and wait for his/her turn (on a last-in first-out basis—late VIPs take precedence over earlier VIPs!). When a signal indicates it is time to serve a client, it prioritizes the stack. If there are VIP clients, then they are served first over the regular clients. It is only when the stack is empty will the regular clients be then served.

Problem 1. Define a CorruptQueue class that contains attributes RegularQueue and VIPStack, which is a representation of a queue and a stack, respectively. The constructor reads its data from a text file called `officeinput.txt`. This text file contains information on what CorruptQueue should do a line at a time. As CorruptQueue reads a line, it should process the information and printout a corresponding output on the command prompt.

The text file contains a variation of these lines:

- `lineup,<name>,<VIP/regular>`
- `serve`
- `arrive,supervisor`
- `leave,supervisor`

`arrive` and `leave` indicate when the supervisor is in the office or not. CorruptQueue prints out "`Supervisor present`" or "`Supervisor not here`" if it reads the corresponding command. When a supervisor arrives and there the VIP stack has contents, pop them all off and enqueue them onto the regular queue, in the popped order. Once VIP client is in the regular queue, the client never transfers to the VIP stack, even when the supervisor leaves. Also, assume that at the beginning of the simulation, the supervisor is not in the office.

`lineup` means that a client has entered the office and is waiting to be served. If the supervisor is present, all clients, whether regular or VIP, line up at the queue. If the supervisor is not in the office, then it determines where a client lines up. If the client is a regular client, then this client lines up at the RegularQueue. If the client is a VIP client, then the client is pushed into the VIPStack. CorruptQueue prints out where this client lines up. Examples would be "`Regular client Juan dela Cruz lines up at RegularQueue`" or "`VIP client John Smith lines up at VIPStack`".

`serve` signals that the counter is free and can accommodate one client. If the VIPStack contains clients, these clients are prioritized over the clients in the RegularQueue. CorruptQueue prints out the name of the client being served and which data structure the client came from. Examples are "`Now serving Juan dela Cruz from RegularQueue`" or "`Now serving John Smith from VIPStack`".

Problem 2. Create a simulation that asks for a text file input from a user (which will be the file used in the constructor in the CorruptQueue class. There will just be one action per line, and you can assume that each line is always correct. The simulation will then output all the output lines in a separate text file (also to be inputted by the user). Sample input and output files will be provided for reference.

Single Server Queues

Being a government office with a (surprisingly) single queue as long as a supervisor present, we will add new parameters to increase the complexity of the otherwise simplified simulation. We assume that the arrivals have a Poisson distribution with a parameter $\boldsymbol{\lambda}$, and the service times have a normal distribution with mean $\boldsymbol{\mu}$ and standard distribution $\boldsymbol{\sigma}$, Further, the unit of measure for the simulation is in minutes.

Problem 3. Add the following attributes to the CorruptQueue:

- lambda – a numeric value that represents the distribution of people lining up the queue, or the average number of people lining up per minute
- mu – a numeric value that represents the mean or average service time
- sigma – a numeric value that represents the standard deviation of the service time

Provide get() and set() methods as well for these new attributes.

Instead of reading input from a text file in the previous problem, create a class called CQSimulation that first asks for a positive integer indicating the number of times the simulation will occur. Then, for each iteration of the simulation, values for the three new attributes will be entered, and the results temporarily stored. Using the results of the calculations of the simulation, display the following:

- average wait time
- average time a customer is in the system

In this simulation, the assumption is that the supervisor is ALWAYS present, meaning all customers, VIP or regular, fall in line in a single queue.