EECS 268: Spring 2016

Laboratory 5

Due: This lab is due before your next lab begins.

Background

Lydia's Luscious Land of Lattes is one of the most popular spots in town. It is actually run out of a main floor area of the ACME corporation, but people come from all over to enjoy Lydia's Lattes. Lydia likes to keep up with her customers and their lives, and she is quite good at remembering faces and stories. She also is politically savvy. She realizes that ACME is being very nice by letting her run her shop in their building without charging her any rent. As a result, she tries to give special service whenever one of the ACME executives stops in. She calls them her "VIP customers".

Lydia prides herself on fast fair service and good conversation. As customers arrive, she wants to make sure they are served precisely in the order they walk in the door. But whenever one of the executive VIPs arrives, she stops talking with whoever is currently being served and attends to the VIP. Moreover, there is a "pecking order" among the VIPs. In fact, Lydia refers to them (although not to their faces!) as VIP0, VIP1, VIP2, ..., VIP9. (There are only 10 VIPs at ACME.) VIP_i is a more important VIP than VIP_{i-1}.

The Assignment

Here's how the whole thing works. As customers arrive, they are added to Lydia's queue. The person at the head of the queue is the one Lydia is currently serving. If a VIP comes along while another customer is being served, Lydia stops serving that person and switches to the VIP. If a more important VIP comes along, Lydia suspends working on the current VIP and starts on the more important one. When Lydia finishes with a customer, the VIP most recently suspended, if any, resumes being served. If no lesser VIPs are waiting, then the person at the head of the queue resumes being served. If a VIP is being served, and another (but *lesser*) VIP arrives, that person simply walks away so as to not upset anything going on. (That very considerate behavior will also make *your* task of simulating this whole scenario much simpler.)

There is no pecking order among customers; e.g., Chris is no more or less important than is Terry. An example of how this all works is illustrated below. VIPs are named VIP0, VIP1, etc. Assume the day is just starting, so Lydia's queue is empty.

Event	Resulting Action
John arrives	added to queue; immediately starts being served, and Lydia starts talking to him
Mary arrives	added to queue
VIP5 arrives	immediately starts getting served (John simply remains at the head of the queue). Lydia talks to the VIP.
Jessie arrives	added to queue
VIP8 arrives	VIP5 is suspended; VIP8 immediately starts getting served, and Lydia starts talking to him/her.
VIP4 arrives	Sees VIP8 being served; immediately leaves

VIP9 arrives	VIP8 is suspended; VIP9 immediately starts getting served, and Lydia starts talking to him/her.
Person being served (VIP9) finishes.	VIP8 resumes being served, Lydia starts talking.
Sally arrives	added to queue
Person being served (VIP8) finishes.	VIP5 resumes being served, Lydia starts talking.
Dana arrives	added to queue
Person being served (VIP5) finishes	Since no VIPs are waiting (remember VIP4 just walked away), person at head of queue (John) resumes being served. Lydia resumes talking to him.
Person being served (John) finishes	John leaves the queue. There are still no VIPs waiting, so person now at the head of the queue (Mary) starts being served. Lydia starts talking.
and so goes Lydia's day....	

Your job is to write a program to simulate this behavior. You will read a series of events like those in the first column of the table above. Manage a stack and a queue so that you can tell who is being served and who finishes as the day goes on. The file you read will be a series of one-word strings. The possibilities for the strings are as follows:

`done`	the person currently being served finishes. Print a message something like: `.. John is done; Mary is starting ..` or `.. VIP8 is done; Laurie is resuming ..` or `.. Laurie is done; no one is waiting, Lydia rests ..` (whichever is appropriate)
`show`	Print a message reporting the state of the simulation. Be as complete as the `Stack` and `Queue` APIs allow you to be. For example, something like the following: `*** VIP5 is currently being served; John is waiting at the front of the queue ***`
anything else	the name of a person arriving for service (e.g., John, Jackie, VIP6, etc.)

Here is a sample file that mimics the initial sequence of events in the first table above with some interspersed "`show`" requests:

```
John Mary VIP5
Jesse
```

```
VIP8
show VIP4 VIP9 show
done Sally show done show Dana done show done
```

When you encounter end of file on the transactions, the stack and/or queue may not be empty. If this is the case, simulate the processing of additional "`done`" requests (including printing the appropriate messages) until both the stack and the queue are empty.

When processing a "`done`" request, simply pop the stack or dequeue the queue, whichever is appropriate. Do *not* first test for it being empty. Your stack and queue implementations will throw exceptions when appropriate, and you must arrange to catch those exceptions at the appropriate place. When you catch an exception, report it as accurately as you can, and then resume processing the transaction file.

You may use the file above and/or one of you own design while testing. Your GTA will create and use an additional test file when grading. This test file will not be available until all students have submitted their projects. Make sure your code is very robust. You will lose significant points if your program aborts and/or produces incorrect output on data files used to test your code.

Implementation

Start from these implementation files only:

- PrecondViolatedExcep.h
- PrecondViolatedExcep.cpp
- StackInterface.h
- QueueInterface.h

You need to decide upon a stack and a queue implementation. The scenario outlined above should make it clear which type of implementation is best for each. Notice, for example, that nothing was said about how many non-VIP customers there might be. On any given day, there may be ten or hundreds. Pick an ideal implementation for each, and justify your two choices in your internal documentation, and then develop implementations. Implementations are largely laid out in the book, and you can certainly refer to them, but you must do your own implementations to be sure you understand how they work. You will be responsible for knowing how the implementations work on exams.

You should always have the VIP being served (if any) on the top of the stack. Similarly, the non-VIP being served should always be at the front of the queue. These conventions will make it a little easier by avoiding some special case tests.

Grading Criteria

Grades will be assigned according to the following criteria:

- Stack implementation (appropriate choice, including your explanation of why you made the choice you did; correctness of the implementation): 22%
- Queue implementation (appropriate choice, including your explanation of why you made the choice you did; correctness of the implementation): 22%
- Overall simulation (includes reading & correctly processing the directives; proper handling of any exceptions that are thrown): 30%
- Program structure (internal documentation; modularity; consistent paragraphing; all instance variables in all classes in `private`; proper packaging and make files): 16%

- Well-formatted output: 10%

Submission

Once you have created the tarball with your submission files, email it to your TA. The email subject line must look like "[EECS 268] SubmissionName" as follows:

```
[EECS 268] Lab 05
```

Note that the subject should be exactly like the line above. Do not leave out any of the spaces, or the bracket characters ("[" and "]"). In the body of your email, include your name and student ID.