CSC 2431 Assignment 4
Due: Wed. 5/18/16 by 11:59pm

Part 1 – Textbook Exercises

Just one! Complete Chapter 17, Exercise #18 (**not** programming exercise #18). This is meant to reinforce/review the use of the STL queue.

Part 2 – Programming Assignment

It's blockbuster movie season! Your local theater has contracted you to design a simulator for the lines of people waiting to see the various movies. The theater is trying out the following new policy:

1. Patrons may begin lining up 2 hours before the movie starts,
2. Once a patron is in line, there is no way out except through the front of the line (i.e., into the theater). Patrons are advised to use the restroom before getting in line. ☺
3. The doors to the theater are opened 1 hour before the movie starts, and patrons are let out of the line one at a time at a rate of *one per minute* to give each time to find a seat without others rushing in behind them.
4. Patrons may enter the line up until the start of the movie – e.g., if the movie starts at 17:30, patrons can get in line at 17:29 but the line closes as soon as the clock reads 17:30.
5. If the line fills up, or the doors close with patrons left in the line, those patrons will complain to management, leave, and write a bad review on Yelp. This is where you come in.

Your goal is to create a class to model this new policy for the theater. You will be given sample data representing the typical flow of patrons into the line and help the theater managers decide if their new policy makes sense, or if the policy will cause lines to "overflow."

The theaterLine Class

You will create a class with the following properties:

1) A private member: an STL `queue` to model a single line of patrons waiting to enter a theater. The queue will hold the names of the people in line (a `string`).
2) A private member: the maximum number of people allowed in the `queue` (an `int`).
    a. Add an accessor for this member, but *not* a mutator.
3) A default constructor and parameterized constructor to initialize the `queue` appropriately.
4) A method to return the number of patrons currently in the `queue`.
5) A method to add a single patron to the end of the `queue` *if there is space available.*
6) A method that both removes a single patron from the front of the `queue` and returns this person's name (a `string`).

Testing

You will be given test files representing the flow of patrons into the line for a single movie showing. The first line of the file will list the start hour and minute, followed by size of the theater line, and the title of the movie. Each subsequent line in the file has a time (hour, followed by minute) and a list of patrons (represented by single letters for simplicity). All times are in military (24-hour) time.

Here is an example for a showing of *The Empire Strikes Back* which starts at 17:30, with a line size of 5.

```
17 30 5 The Empire Strikes Back
15 31 A X Y Z
15 32 B
15 33 C
15 34 D Q M P
15 35 E
15 36 F
15 37 G
15 38 H
15 39 I
16 40 J
16 50 K
17 29 R S T U V
```

Your Client

Most of the code you write will be in your client. Your client should include a function to simulate the behavior of the line. It will take the start hour/min, line size, movie title, and an input file stream. In other words, your main() should open the file, read the first line, and pass all this information to the client function. Here is the header for my client function:

```
void simulateMovie(int startHour, int startMin, int lineSize, string title, ifstream &fs)
```

The client should have a loop that simulates *every minute* from the opening of the line to the closing of the line. Each "tick" of the clock in your loop should call appropriate functions on a `theaterLine` object to add and remove patrons from the line as needed. If the theater line fills up print a message to the screen to report this. When the movie begins (e.g., 17:30 in the example above), end the simulation and report how many people are still left waiting to get into the theater. Here is an example run with my program for the *The Empire Strikes Back* showing above.

```
Starting simulation for The Empire Strikes Back which starts at 17:30.
The line can hold a maximum of 5 patrons.
The time is currently 15:30.
15:31   A is entering the line
        X is entering the line
        Y is entering the line
        Z is entering the line
15:32   B is entering the line
15:33   Line has overflowed!  Patron C was turned away!
15:34   Line has overflowed!  Patron D was turned away!
        Line has overflowed!  Patron Q was turned away!
        Line has overflowed!  Patron M was turned away!
        Line has overflowed!  Patron P was turned away!
15:35   Line has overflowed!  Patron E was turned away!
15:36   Line has overflowed!  Patron F was turned away!
15:37   Line has overflowed!  Patron G was turned away!
15:38   Line has overflowed!  Patron H was turned away!
15:39   Line has overflowed!  Patron I was turned away!
```

```
16:30 A has entered the theater!
16:31 X has entered the theater!
16:32 Y has entered the theater!
16:33 Z has entered the theater!
16:34 B has entered the theater!
16:40   J is entering the line
16:40 J has entered the theater!
16:50   K is entering the line
16:50 K has entered the theater!
17:29   R is entering the line
        S is entering the line
        T is entering the line
        U is entering the line
        V is entering the line
17:29 R has entered the theater!
Doors closing, ending simulation!
There are 4 patrons left in line!
Press any key to continue . . .
```

What to turn in:

1. Your header (.h) for the class, and source for your client (.cpp).
2. Your output after running the **three** test files posted on Blackboard in your client. Sample output will be provided for each.
3. All documents should be uploaded to Blackboard following the assignment submission instructions (linked on Blackboard).

Extra Credit

Worth 5 extra points on the assignment (which is out of 50 points).

The output generated above is pretty boring, but gets the job done. Better would be a "live" representation of the queue as patrons enter and leave the line. For example, on every iteration of your simulation loop (i.e., after the "clock" advances one minute) you could print the current contents of the queue, what has changed, and a list of keyboard controls to interact with the simulation. Your improved output doesn't need to look exactly like this, but should show the line in some way for each minute.

Example:

```
13:30 A enters the line

------------------
A
------------------

<hit s to step to next time, c to run sim to completion, x to quit>


13:31 B, C, D enter the line

------------------
A   B   C   D
------------------

<hit s to step to next time, c to run sim to completion, x to quit>
```

```
13:45 E enters the line

--------------------
A   B   C   D  E
--------------------

<hit s to step to next time, c to run sim to completion, x to quit>


14:00 A leaves the line

--------------------
B   C   D   E
--------------------

<hit s to step to next time, c to run sim to completion, x to quit>
```

And so on for each time step (i.e., each one where something happens).