CMPT 225 Assignment 3: Banking Simulation

Please, read the entire assignment first before starting it!

Suggestion: let's start our assignment as early as possible!

This assignment can be done in pairs!

Objectives

In Assignment 3, our objectives are:

- To learn a new data collection: the priority queue.
- To practise writing complete data collection ADT classes based on given public interfaces.
- To practise implementing and using template data collection ADT classes.
- To practise using C++ exception handling mechanism.
- To implement a solution to a problem by following a given algorithm.
- To solve a problem going through the steps of the software development process.

Simulation

Problem Statement

In this Assignment 3, we are to design and implement an event-driven simulation application. The problem statement we are solving in this assignment is described in Programming Problem 6 of Chapter 13 of our textbook (pages 413 and 414) and in its Section 13.4. So, before we dive into this assignment, let's first read Section 13.4: Application Simulation of Chapter 13 and Programming Problem 6 from our textbook.

Classes

In order to solve this problem, we will need the following classes. We may wish to implement each of these classes incrementally, starting with the first class (described below) then moving on to the second class only once the first class has been fully tested (using a test driver). Here are the classes:

1. Event class
    - The Event class that models the events described in Section 13.4.
    - It must be designed and implemented as an ADT and have getters and setters for the data members "type", "time" and "length". Note that we cannot use C++ "struct".
    - The number and kinds of constructors is up to us as well as whether the class should have a destructor.
    - We may find it useful to overload an operator such as <. We may also find it useful to overload the << operator (cout) (see C++ Interlude 6 in our textbook). This would allow us to print the content of an object of this class in the client code (such as in our application "class" or a test driver) like this:

    ```
    cout << anEvent;
    ```

    - **About IO - Testing output statements**

      Once we have tested our class, let's comment out/remove any testing output statements from the methods of our class before submitting our Assignment 3.
    - Hint: It is not the responsibility of the Event class to print "Processing an arrival event at time:" on the computer monitor screen.
2. Queue class
    - A template Queue data collection ADT class.
    - In order to construct the Queue class as a template, follow the instructions given in Lab 4. Since we can only submit one file for our Queue class, lets make sure we produce only one file, i.e., Queue.h.
    - Feel free to use the Queue we implemented in Question 4 of our Assignment 2 and modify it so it satisfies the following public interface:

    ```
    /******* Queue Public Interface   START   *******/
    ```

```
/******* Queue Public Interface - START - *******/
   // Class Invariant:  FIFO or LILO order

   // Description: Returns "true" is this Queue is empty, otherwise "false".
   // Time Efficiency: O(1)
   bool isEmpty() const;

   // Description: Inserts newElement at the "back" of this Queue
   //              (not necessarily the "back" of its data structure) and
   //              returns "true" if successful, otherwise "false".
   // Time Efficiency: O(1)
   bool enqueue(const ElementType& newElement);

   // Description: Removes the element at the "front" of this Queue
   //              (not necessarily the "front" of its data structure) and
   //              returns "true" if successful, otherwise "false".
   // Precondition: This Queue is not empty.
   // Time Efficiency: O(1)
   bool dequeue();

   // Description: Returns the element located at the "front" of this Queue.
   // Precondition: This Queue is not empty.
   // Postcondition: This Queue is unchanged.
   // Exceptions: Throws EmptyDataCollectionException if this Queue is empty.
   // Time Efficiency: O(1)
   ElementType& peek() const throw(EmptyDataCollectionException);

/******* Queue Public Interface - END - *******/
```

- The number and kinds of constructors is up to us as well as whether it should have a destructor.
- We may also find it useful to overload the << operator (cout) for testing purposes. This would allow us to print the content of our Queue from the client code (such as in our application "class" or a test driver) like this:

```
cout << aQueue;
```

- **About IO - Testing output statements**

  Once we have tested our class, let's comment out/remove any testing output statements from the methods of our class before submitting our Assignment 3.

3. PriorityQueue class

   - A template PriorityQueue data collection ADT class. See Section 13.3 and 14.2 in our textbook for more information.
   - Here is the header file PriorityQueue.h and the implementation file PriorityQueue.cpp to use and complete for this assignment.
   - Make sure completely transform it into a template, in the same way we transformed the Queue class into a template.
   - We will also need a template Node class.
   - We may also find it useful to overload the << operator (cout) for testing purposes. This would allow us to print the content of our PriorityQueue from the client code (such as in our application "class" or a test driver) like this:

     ```
     cout << aPQueue;
     ```

   - **About IO - Testing output statements**

     Once we have tested our class, let's comment out/remove any testing output statements from the methods of our class before submitting our Assignment 3.

   - We need to decide what is the "priority value" in this problem. We also need to keep in mind that *"highest" priority* does not always mean largest value. The following will help us figure out what to use as "priority value" and in which sort order to keep the elements of our Priority Queue:

     - Figure 13-8 of our textbook shows that an Event object of type "A" and time "20" has a "higher" priority than an Event object of type "A" and time "22" and therefore would be the next to be dequeued from the Priority Queue *eventPQueue* even though "20" < "22".
     - Figure 13-8 also shows that an Event object of type "A" and time "30" has a "higher" priority than an Event object of type "D" and time "30" and therefore would be the next to be dequeued from the Priority Queue even though both

Event objects have the same time.

4. EmptyDataCollectionException class

    - An exception class called EmptyDataCollectionException. This class has been provided for us to use: EmptyDataCollectionException.h and EmptyDataCollectionException.cpp. It is ready "to go", i.e., it does not require any modifications. Please, refer to C++ Interlude 3 of our textbook to learn how to use exceptions. Feel free to refer to other resources found on the Internet regarding exceptions such as this web site.

    - Hint: When calling methods (such as peek()) that throw exceptions, we must do so within try/catch blocks.

5. Simulation application "class"

    - We must create a simulation application "class" named "SimApp.cpp" which is to contain the main() function and perhaps other functions. We must use the algorithm outlined in Section 13.4 of our textbook to guide the implementation of our simulation application.

    - Contrary to what is stated in Programming Problem 6 of our textbook, our simulation application is not to open and read input files, but is to read input (file) from the command line. More on this subject in the "Input File" section below.

    - If we are having difficulty understanding how the simulation works, stepping" through Figure 13-8 in our textbook (see Section 13.4) will be helpful.

Documentation

- Let's make sure all our classes have the following documentation:

    - Header comment block containing: filename, class description, class invariant (if any), author, date of creation/last modification.

    - A description, a precondition (if any) and a postcondition (if any) for each of the class method (public and private).

    - All data members and methods of a class must be descriptively named. If needed, comments must be added to the data members and/or methods.

    - Let's make sure our code satisfies the "good programming style" described on the GPS web page of our course web site.

Testing

Makefile

We can use this makefile, but we cannot modify it.

Input File

Here are three input files, and their corresponding "expected results" file, to be used to test our simulation: simulationShuffled1.in and simulation1.er, simulationShuffled2.in and simulation2.er, simulationShuffled3.in and simulation3.er.

Contrary to what is stated in Programming Problem 6 of our textbook, our simulation application is not to open and read input files, but is to read input (file) from the command line as follows:

```
uname@hostname: ~$ ./sim < simulationShuffled3.in
```

where *sim* is the executable our makefile created.

Output Format

As indicated in Programming Problem 6, our simulation application must produce a very specifically formatted result. Let's make sure our simulation application prints on the computer monitor screen its result in exactly the same format as the results displayed in Programming Problem 6, i.e., same layout, same words with same lower/upper cases, same number of whitespaces, empty lines, etc...). The above three "expected results" (*.er) files show the expected format of the results our simulation application should produce given the corresponding input file.

This is why our classes should not output any statements (such as testing statements) to the computer monitor screen aside from the ones expected. Superfluous output statements will not allow us to create results which will exactly match the expected results.

Marking Scheme

When marking Assignment 3, we shall consider some or all of the following criteria:

- Solving the simulation problem: Does our solution solve the problem stated in this assignment?
- Correctness of the submitted classes: do they abide to the above requirements and do Queue and PriorityQueue classes work with test driver programs written based on their public interfaces?
- Requirements: Does our solution satisfy the requirements described in this assignment?

- Coding style: Has *Good Programming Style*, described on the Good Programming Style web page of our course web site, been used?
- Documentation: Does our solution satisfy the documentation requirements described in this assignment?

Remember ...

- We can use any C++ IDE we wish to do our assignments in this course as long as the code we submit compiles and executes on the CSIL computers. Why? Because our assignments will be marked using this platform (Linux and C++ - version running on CSIL computers). Suggestion: Compile and execute our code using the makefile at the Linux command line in CSIL before we submit your assignment to CourSys.
- We cannot make use of the STL library.

Submission

- Assignment 3 is due Wednesday, February 27 at 15:00.
- In the interest of fairness to our classmates, and expedient marking, late assignments will not be marked.
- Since this assignment can be done in pairs, we are required to form groups of 2 on CourSys in order to submit our work - even if we have done this assignment on our own. In this case, we must form a group of 1 on CourSys.
    - Let's form groups of 2 on CourSys when submitting our work and name our groups as follows: **userIDofStudent1-userIDofStudent2** (if we are working in pairs) or **userIDofStudent** (if we are working on our own). In other words: concatenate the userID (not the student number) of the first student with the userID of the second student and separate them with an hyphen "-".
- Submit the following files on CourSys: Event.h, Event.cpp, Queue.h, PriorityQueue.h, Node.h, SimApp.cpp. Since we are not to modify the makefile and the Exception class files, there is no need to submit them. Only one group member is required to do the submission for the group.
