CSCI 310 – 02 (Fall 2019)

Programming Foundations

Programming Assignment 5

DUE: Sun, Oct 27, 11:59 PM (`turnin` time)

First City Bank of Springfield: An Event–Driven Simulation

Programming Problem 13–6, pp. 393–394, Carrano and Henry 7/e.

For additional background information on this problem, be sure to read section 13.4 (*Queues and Priority Queues Application: Simulation*), pp.381–388 of our textbook.

Specifications

You are to implement the event–driven simulation of a bank that is described in Chapter 13 of our textbook.

A FIFO queue of arrival events will represent the line of customers in the bank waiting for service from a teller. Use the C++ `queue` container class for your FIFO queue. Maintain arrival events and departure events in a priority queue that keeps these events sorted by the time of the event. Use the C++ `priority_queue` container class for your priority queue. Note that, by default, the C++ `priority_queue` is a ***max****–queue* so you will have to figure out how to use this container class as a ***min****–queue* to correctly order and process the events (covered in lab).

Your program must count the total number of transactions and keep track of their cumulative transaction times and waiting times. These statistics are sufficient to compute the average transaction times and the average waiting time after the last event has been processed.

Input

Your program will use an input text file of arrival times and transaction times. The filename of this input text file will be provided by the user. Each line of the file contains the arrival time and required transaction time for a customer. You may assume that the input data from the provided filename is correct.

The arrival times can be ordered by increasing time, but they do not have to be ordered as such (can you determine why ordering is not crucial?). For additional information on C++ File I/O, see **Appendix G** on pp. 795–804 of our textbook.

Sample Input

The following sample input is from Programming Problem 13–6, pp. 393–394, Carrano and Henry 6/e.

```
1 5
2 5
4 5
20 5
22 5
24 5
26 5
28 5
30 5
88 3
```

Output

Your program will display a trace of the events executed and a summary of the computed statistics (the total number of transactions processed, the average transaction time, and the average wait time (the average time spent waiting in line).

All output should be sent to standard output and must exactly follow the format in the sample below.

Sample Output

For the sample input given above, the program should produce the following output (modified slightly from Programming Problem 13–6, pp. 393–394, Carrano and Henry 6/e):

```
Simulation Begins
Processing   arrival event at time:    1    5
Processing   arrival event at time:    2    5
Processing   arrival event at time:    4    5
Processing departure event at time:    6
Processing departure event at time:   11
Processing departure event at time:   16
Processing   arrival event at time:   20    5
Processing   arrival event at time:   22    5
Processing   arrival event at time:   24    5
Processing departure event at time:   25
Processing   arrival event at time:   26    5
Processing   arrival event at time:   28    5
Processing   arrival event at time:   30    5
Processing departure event at time:   30
Processing departure event at time:   35
Processing departure event at time:   40
Processing departure event at time:   45
Processing departure event at time:   50
Processing   arrival event at time:   88    3
Processing departure event at time:   91
Simulation Ends

Simulation Statistics
  Number of transactions processed:   10
           Average transaction time:    4.80
                  Average wait time:    5.60
```

Additional Requirements

Complete the implementation of an `Event` class given the following header file:

```
// Event.h - an event class for event-driven simulation

#ifndef EVENT_H_
#define EVENT_H_

class Event
{

    public:

        enum eventType { undefined , arrival , departure };

        // Creates an undefined event
        Event( );
        // Creates an event of a given type, occurrence time, and duration
        Event( eventType type , unsigned time , unsigned duration=0 );
        // Copy constructor
        Event( const Event& otherEvent );

        // Determines the type of the event
        eventType getType() const;

        // Determines the time of the event
        unsigned getTime() const;

        // Determines the duration of the (arrival) event
        unsigned getDuration() const;

        // Events are ordered by their event time (occurance)
        friend bool operator<( const Event& lhs , const Event& rhs );
        friend bool operator>( const Event& lhs , const Event& rhs );

    private:

        eventType eType;      // type of event
        unsigned  eTime;      // time of event occurance
        unsigned  eDuration;  // duration, if event is an arrival event

};

#endif
```

Our text provides the following pseudocode for an event–driven simulation:

```
// Performs the simulation.

Create an empty queue bankQueue to represent the bank line
Create an empty priority queue eventListPQueue for the event list

tellerAvailable = true

// Create and add arrival events to event list
while( data file is not empty )
{
     Get next arrival time a and transaction time t from file
     newArrivalEvent = a new arrival event containing a and t
     eventListPQueue.add( newArrivalEvent )
}
// Event loop
while( eventListPQueue is not empty )
{
     newEvent = eventListPQueue.peek()
     // Get current time
     currentTime = time of newEvent
     if( newEvent is an arrival event )
          processArrival( newEvent , eventListPQueue , bankQueue )
     else
          processDeparture( newEvent , eventListPQueue , bankQueue )
}
```

Note that since the above code merely provides the algorithm, you may need to adjust the parameters to the process functions to suit your implementation.

Here is the pseudocode for processing arrival events:

```
// Processes an arrival event.
processArrival( arrivalEvent: Event,
                eventListPQueue: PriorityQueue,
                bankQueue: Queue )
{
     // Remove this event from the event list
     eventListPQueue.remove()
     customer = customer referenced in arrivalEvent
     if( bankQueue.isEmpty() && tellerAvailable )
     {
          departureTime = currentTime + transaction time in arrivalEvent
          newDepartureEvent = a new departure event with departureTime
          eventListPQueue.add( newDepartureEvent )
          tellerAvailable = false
     }
     else
          bankQueue.enqueue( customer )
}
```

And here is the pseudocode for processing departure events:

```
// Processes a departure event.
processDeparture( departureEvent: Event,
                  eventListPQueue: PriorityQueue,
                  bankQueue: Queue )
{
    // Remove this event from the event list
    eventListPQueue.remove()
    if( !bankQueue.isEmpty() )
    {
        // Customer at front of line begins transaction
        customer = bankQueue.peek()
        bankQueue.dequeue()
        departureTime = currentTime + transaction time in customer
        newDepartureEvent = a new departure event with departureTime
        eventListPQueue.add( newDepartureEvent )
    }
    else
        tellerAvailable = true
}
```

This is a fairly involved project, so plan on starting on it early. The most important part of successfully solving this problem is to make sure you understand all the various components and how they relate to each other.

Deliverables

Your submission will consist of the following files, submitted using the Department of Computer Science's `turnin` facility:

- `Event.cpp` – implementation file for the required `Event` class
- `bankSim.cpp` – driver code containing `main()` function

Note that the following code will be available on `turnin`:

- `Event.h` specification file for the required `Event` class

We want you to develop good code documentation habits. Source code solutions submitted without any meaningful documentation will receive a total score of zero (0). You may refer to the *Google C++ Style Guide* section on source code comments as a guide.

Be sure to also review and adhere to the **Coding Standards** for this course.