Programming Assignment 1

CS 311 Data Structures
Spring Quarter, 2006

Due: Thursday 04/13/06, 11:59pm

Goals of this assignment

- Get re-acquainted with C++ and OOP.
- Use the queue data type in an event-driven simulation
- Determine and, if necessary, implement the best data structure to store certain events.

Introduction

In an effort to improve customer service, UFIE Tech Support wishes to find out how much time its callers spend on hold. The plan is to use existing call data spanning the period of a day and find out how many customers were helped, the average waiting time and the maximum waiting time per customer.

Furthermore, the company would like to add a feature to its call system that informs each customer approximately how long they should expect to wait before they can talk to a tech support specialist.

Event-Driven Simulation

You will use an event-driven simulation to collect the waiting time statistics, given information about call times and the durations of calls.

In an event-driven simulation time advances based on when certain events occur. In this particular application there are two types of events:

- **Call Events** occur when a customer calls the tech support line. If, at that time, a representative is available, and the customer is first in the queue, then the customer is connected to that representative. Otherwise, the customer remains on hold.
- **HangUp Events** occur when a customer's call is complete (and hopefully the problem resolved).

Input

A file whose name is passed as a command-line argument contains the call time information. The first line of the file contains the number of tech support representatives that are on call during the day. This never changes during the simulation.

Each remaining line contains a pair of integers: the first is the time a customer called, measured in minutes after the start of the day, and the second is the time span from the moment the customer was connected to a representative to the moment the call ended.

The pairs are sorted by call time.

You may assume that the call times are distinct (no two customers call at the exact same minute) and are whole numbers. The time span may not be 0.

A call time and/or time span of 0 signifies end of input.

Simulation

During the simulation we maintain a collection of events that could be Call or HangUp Events. At each step, we need to retrieve the event with the smallest occurrence time (i.e. the next event that should happen in time).

At each step, an event is extracted from the collection and examined. If it is a Call, then the customer is placed in line. If it is a HangUp, then a representative becomes available.

When a representative becomes available to take a call, the next person in line is connected. At this time, since we know how long the call will take, we can create the corresponding HangUp event and add it to the collection of events.

During the run of the simulation, you will need to keep track of the total waiting time for the customers (the time spent talking to a representative should not be included in that), the maximum waiting time and the total number of customers who called. You may assume that all calls will be answered.

Output

During the simulation, print out debugging information in a debug file whose name is the same as the data file's, with extension .dbg

For example, the input file `data.1`:

```
3
1 10
2 9
3 10
4 5
5 4
11 20
0 0
```

should result in the debug file `data.1.dbg`:

```
Time:
-----
    1:  A customer with an estimated 10-minute call has been placed on hold.
    1:  The customer who called at time 1 is connected to a representative,
        after waiting for 0 minutes.

    2:  A customer with an estimated 9-minute call has been placed on hold.
    2:  The customer who called at time 2 is connected to a representative,
        after waiting for 0 minutes.

    3:  A customer with an estimated 10-minute call has been placed on hold.
    3:  The customer who called at time 3 is connected to a representative,
        after waiting for 0 minutes.

    4:  A customer with an estimated 5-minute call has been placed on hold.

    5:  A customer with an estimated 4-minute call has been placed on hold.

   11:  A customer with an estimated 20-minute call has been placed on hold.

   11:  A customer has hung up.
   11:  The customer who called at time 4 is connected to a representative,
        after waiting for 7 minutes.

   11:  A customer has hung up.
   11:  The customer who called at time 5 is connected to a representative,
        after waiting for 6 minutes.

   13:  A customer has hung up.
   13:  The customer who called at time 11 is connected to a representative,
        after waiting for 2 minutes.

   15:  A customer has hung up.

   16:  A customer has hung up.

   31:  A customer has hung up.

Number of customers: 6
Average waiting time: 2.50 minutes.
Maximum waiting time: 7 minutes.
```

Note that when calls and hangups occur at the same time, calls are processed first.

Implementation issues

Classes

For this program you will need to define your own classes. Some of the objects involved are:

Events An Event always has a start or occurrence time. Arrival Events also have a transaction time. Use inheritance for an accurate representation of this model.

Customers Each customer has an assigned arrival time and transaction time.

When you create your classes, make certain to keep definitions and implementations separate.

Data structures

You will need to maintain a queue of customers and a structure to hold the events. You may use STL containers or write your own class templates.

Try to use the most suitable and efficient structure to hold the events. Data structures you may want to consider include the sorted list, the skip list, the queue, the priority queue. Not all of these are suitable. Include comments in the header section of the file that document your reasons for choosing a particular data structure.

For reference, insert and delete times in the STL priority queue are $O(\lg n)$ each.

Your code will be graded on correctness, class design, data structure selection and maintainability.

Estimating waiting time

Recall that the company would like to add a feature to its call system that informs each customer approximately how long they should expect to stay on hold before they can talk to a tech support representative. Given any real time information that can be collected, as well as any simulation-based information that could be useful, how would you estimate the waiting time for a customer who has just called? What information would you collect, how would you store it, how would you retrieve it? If an operation such as retrieval needs to be fast, then explain why and select your structure accordingly.

Type your answer in a **`text`** file called `part2.txt`. We are not looking for an essay. 2-3 paragraphs should be sufficient.

What to submit

When you finish your program, remove any executables and `core*` files. Submit your work (.h, .cpp and .txt) as a compressed archive (tarball). To do that, `cd` into your PA1 directory and then type:

```
tar  czf  your_name.tgz  *.h *.cpp part2.txt
```

where `your_name` is your name. Email the file to `cs311@cs.northwestern.edu` AND to yourself. When you receive your copy, verify that you sent the right files. We will send a confirmation that the file was received, but we will not be checking whether the tarball was attached correctly or contains the correct files.

Getting help

If you have any questions, post to the newsgroup. Never use the `cs311` email for time-critical questions, as it is only checked after assignment deadlines.