CSCI 2170 Spring 2006
OLA 7 (Due : beginning of class, Tuesday, April 25th, 2006. No late program is accepted.)

Implement the event-driven simulation of a bank. A queue of arrival events will represent the line of customers in the bank. Maintain the arrival events and departure events in an ADT event list, sorted by the time of the event. Use a pointer based implementation for the ADT event list.

The input is a text file of arrival and transaction times. Each line of the fie contains the arrival time and required transaction time for a customer. The arrival times are ordered by increasing time. Copy the data file into your directory with:

frank% cp ~cli/data/bank.dat .

Your program must compute the **average waiting time** and **maximum waiting time** of the customers, and compute the **maximum length of the waiting queue**. This requires your program to keep track of statistics including: the number of customers served, the cumulative waiting time, the maximum waiting time, the maximum length of the waiting queue, etc..

Your program should display a trace of the events executed, and display the final statistics at the end, for example:

Simulation begins:

Processing an arrival event at time : 1
Processing an arrival event at time : 4
Processing an departure event at time : 7
Processing an arrival event at time : 11
Processing an arrival event at time : 12
Processing an departure event at time : 13
…. ..

Final Statisitics:

Total number of people processed:	15
Average amount of time a customer spent waiting:	5.6 minutes
Maximum amount of time spent waiting	17 minutes
Total time during simulation:	210
Maximum length of the waiting line:	6

You are required to use pointer-based implementation of ADT queue, and the ADT EventList described in book (page 364-365).

Simulate()

```
Create an empty queue  bankQueue  to represent the bank line
Create an empty event list  eventList

Get the first arrival event from the input file
Place the arrival event in eventList

while (eventList is not empty)
{       newEvent = the first event in eventList

        if (newEvent is an arrival event)
                processArrival(newEvent, arrivalFile, eventList, bankQueue);
        else
                processDeparture(newEvent, eventList, bankQueue);
}
```

processArrival(arrivalEvent, arrivalFile, anEventList, bankQueue)

```
atFront = bankQueue.isEmpty();

bankQueue.enqueue(arrivalEvent)
delete arrival event from anEventList

if (atFront)
{
        insert into anEventList a departure event that corresponds to the new
        customer and has currentTime = currentTime + transaction length
}

if (not at end of input file)
{
        get the next arrival event from arrivalFile
        add the event to anEventList
}
```

ProcessDeparture(departureEvent, anEventList, bankQueue)

```
bankQueue.dequeue();
Delete departureEvent from anEventList;

if(!bankQueue.isEmpty())
{       insert into anEventList a departure event that corresponds to the customer
        now at the front of the line and has currentTime =
        currentTime+transaction length;
}
```