Final Due Date: Wednesday, 9/23/09
Milestone Due Date: Wednesday, 9/16/09

CSCI 36200 Program 3

FedEx has hired you to do a simulation of activities at one of their airport loading docks so that they can decide whether to hire more dock workers, use fewer planes, etc. Planes carry packages from the airport to either Chicago or Memphis for redistribution.

FedEx knows a lot of statistics about how the loading dock currently works. In particular:

A. A package arrives every 10 units of time.
B. A package has a 50/50 chance of being destined to Chicago or to Memphis
C. Processing a package takes 8 units of time.
D. Maintaining the forklift takes 6 units of time
E. A plane for Chicago arrives every 150 units of time.
F. A plane for Memphis arrives every 180 units of time.
G. Each plane can load a maximum of 25 packages.
H. Loading a plane takes 25 units of time regardless of the size of the stack.
(These could be constants in your program so they could easily be changed to model different conditions).

Here's another constant, although it doesn't come from a FedEx statistic: The simulation will run until 2000 units of time have passed.

The loading dock operates as follows:

1. An arriving package goes at the back of a queue for packages, Q1.
2. A package is processed by removing it from the queue, checking its destination, and adding it to the stack on the dock for the appropriate destination. The server is busy while processing a package.
3. An arriving plane goes at the back of a queue for planes, Q2.
4. A plane is processed by removing it from the queue and loading packages from the stack for its destination. The server is busy while the plane is being loaded.
5. Stacks and queues are unbounded in size.
7. When a service event occurs, the server services Q2 (process a plane) if Q2 is not empty, otherwise the server services Q1 (process a package) if Q1 is not empty, otherwise the server maintains his or her forklift, during which the server is busy.

Note that the server services Q1 and Q2, not the event list. The possible events in the event list are:

- package arrives
- plane arrives
- end of service event

For your own clarification, fill in the following table to be sure you understand the corresponding response to each of these events (this can be a pseudocode description of what is to be done).

Event	Response

package arrives	if(Q2 is not empty)
	    service Q2
	else if(Q1 is not empty) //it won't be, a package arrived
	    service Q1

plane arrives	if(Q2 is not empty) // it won't be, a plane arrived
	    service Q2

end of service event	if(Q2 is not empty)
	    service Q2
	else if(Q1 is not empty)
	    service Q1
	else // this portion is executed
	     maintain forklift

What should your program output?

"Real" output at the end of the simulation: The number of packages processed, the minimum time, maximum time, and average time a package is in the system (from arrival to when its plane departs).

"Debug" output: A snapshot of the system, i.e., the contents of the queues and stacks and event list at any moment in time the user requests.

The event list initially contains the following data:

Event List

Time	Event Code	Attribute 1
5	Package arrives	Memphis
8	End of Service	
80	Plane arrives	Chicago
100	Plane arrives	Memphis

Preliminary work:

1. It's often suggested that in object-oriented programming, the nouns in the problem statement represent objects, which in turn will suggest the needed classes. Reading the problem statement for Program 3, one would think that packages and planes would be separate objects from separate classes since they are two nouns in the system. However, packages and planes have identical properties of timestamp (arrival time, an integer) and destination (a string), and the same member functions (constructors and accessors). Therefore we can use a single class for both packages and planes. Create a class called Item. Include an overloaded << operator to make it easy to write out the Items in stacks and queues for a project snapshot. Create a driver program to test this class (a driver program is simple, throw-away code – create an item, set its data values, use accessor functions to write out the data values, and test the overloaded << operator).

2. This program will require stacks of Items (for packages) and queues of Items (for both packages and planes). Use the Queue class and the Stack class from the Standard Template Library. Write a driver program that creates a queue of Items, enqueues new Items at the back, dequeues them from the front, and also creates a stack of Items, pushes new Items on the stack, pops Items off the stack. In addition, create functions that write out the contents of a queue from front to back and the contents of a stack from top to bottom. These are not normal operations for queues or stacks, but are there for debugging (snapshot) purposes.

3. The simulation runs by processing events. There are three kinds of events: package arrives, plane arrives, end-of-service event. The important properties of an event are its timestamp (when the event occurs, an integer), a key to the kind of event it is (a string), and a destination code (a string, meaningful for planes and packages but not for end-of-service events). Create a class called Event. Include an overloaded << operator to make it easy to write out the Events in the event list for a project snapshot. Also include an overloaded < operator to compare Event objects by their timestamp. Create a driver program to test this class – create an item, set its data values, use accessor functions to write out the data values, and test the overloaded << operator and the < operator.

4. This program will need to maintain a list of Events. Use the List class from the

Standard Template Library. Write a driver program that creates a list of Events. Items should be inserted into the list so that the list is sorted in ascending order based on the timestamp of an event. This means that Events with the lower timestamps are closer to the front of the list than those with higher timestamps. (This is essentially a priority queue, where the timestamp is the priority and lower priorities beat higher ones.) Because you defined a < operator for Events, you can take the easy way out and use the list.sort() function rather than write a special insert function,. Also test that you can retrieve and remove Events from the front of the list, and create a function to write out the contents of the list in order of increasing timestamp.

This is kind of a big project, so I strongly suggest you do the above 4 preliminary steps by Wednesday 9/14/09 (milestone due date). I won't collect anything, but this milestone will help ensure that you can meet the actual due date for the complete program on 9/23/09.

One final comment:

There is a very simple distribution of package arrivals in this program. Packages have a 50/50 change of going to either Chicago or Memphis. When you generate a new package, create a random number between 0 and 9. If it's less than 5 it's a package to Chicago, otherwise it's a package to Memphis. We've split the range in half, so there is a 50/50 chance of getting either destination. Ordinarily you would use truly random numbers here, but use only pseudorandom numbers for repeatability.