**Programming Problems**

1. Using the class `queue` in the Standard Template Library, define and test the class `OurQueue` that is derived from `QueueInterface`, as given in Listing 13-1. The class `queue` has the following methods that you can use to define the methods for `OurQueue`:

   ```cpp
   queue<ItemType>();               // Default constructor
   bool empty() const;              // Tests whether the queue is empty
   void push(const ItemType& newEntry); // Adds newEntry to the back of the queue
   void pop();
   ItemType& front();               // Returns a reference to the front of the queue
   ```

   To access queue, use the following `include` statement:

   ```cpp
   #include <queue>
   ```

2. Using the class `priority_queue` in the Standard Template Library, define and test the class `OurPriorityQueue` that is derived from `PriorityQueueInterface`, as developed in Exercise 7. The class `priority_queue` has the following methods that you can use to define the methods for `OurPriorityQueue`:

   ```cpp
   priority_queue<ItemType>();      // Default constructor
   bool empty() const;              // Tests whether the priority queue is empty
   void push(const ItemType& newEntry); // Adds newEntry to the priority queue
   void pop();                      // Removes the entry having the highest priority
   ItemType& top();                 // Returns a reference to the entry having the highest priority
   ```

   To access priority queue, use the following `include` statement:

   ```cpp
   #include <queue>
   ```

   *Whenever you need a queue or a priority queue for any of the following problems, use the classes `OurQueue` and `OurPriorityQueue` that Programming Problems 1 and 2 ask you to write.*

3. Implement the palindrome-recognition algorithm described in Section 13.2.2.

4. Implement the recognition algorithm that you wrote to solve Exercise 2 using the classes `OurQueue`, as described in Programming Problem 1, and `OurStack`, as described in Programming Problem 1 of Chapter 6.

5. Implement the radix sort of an array by using a queue for each group. The radix sort is discussed in Section 11.2.3 of Chapter 11.

6. Implement the event-driven simulation of a bank that this chapter described. A queue of arrival events will represent the line of customers in the bank. Maintain the arrival events and departure events in a priority queue, sorted by the time of the event. Use a link-based implementation for the priority queue.

   The input is a text file of arrival and transaction times. Each line of the file contains the arrival time and required transaction time for a customer. The arrival times are ordered by increasing time.

   Your program must count customers and keep track of their cumulative waiting time. These statistics are sufficient to compute the average waiting time after the last event has been processed. Display a trace of the events executed and a summary of the computed statistics (the total number of arrivals and average time spent waiting in line). For example, the input file shown in the left columns of the following table should produce the output shown in the right column.
Modify and expand the event-driven simulation program that you wrote in Programming Problem 6.

a. Add an operation that displays the event queue and use it to check your hand trace in Exercise 10.
b. Add some statistics to the simulation. For example, compute the maximum wait time.

8. The people who run the Motor Vehicle Department (MVD) have a problem. They are concerned that the average time spent waiting in lines to appreciate the privilege of owning and driving an automobile. The current arrangement is as follows:

- When people walk in the door, they must wait in a line to sign in.
- Once they have signed in, they are told either to stand in line for registration renewal or to wait for their turn in the service line.
- When the service person is ready for them, he or she asks for the license renewal or drivers’ license application.
- The service person then tells the person at the front of the line to go to an available teller. The teller follows the following guidelines:

  - There is a single line for each teller.
  - The teller who is available (not currently at work) is the next one in line.
  - A rule that chooses a line when processing an arrival event (for example, enter the shortest line)
  - Three distinct departure events (one for each line)
  - Rules for breaking ties in the event queue

People do not spend enough time waiting for the privilege of owning and driving an automobile. What can you conclude about having a single line as opposed to having distinct lines?

- What is the effect of having three distinct lines? Would you expect each to have the same number of people?
- What is the effect of having three distinct lines? Would you expect each to have the same number of people?