**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 8. The class `priority_queue` has the following methods that you can use to define the methods for `OurPriorityQueue`:

```
priority_queue();                       // 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:

```
#include <priority_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 event list.

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.

**Input file**

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

**Output from processing file on left**

```
Simulation Begins
Processing an arrival event at time:    1
Processing an arrival event at time:    2
Processing an arrival event at time:    4
Processing a departure event at time:   6
Processing a departure event at time:  11
Processing a departure event at time:  16
Processing an arrival event at time:   20
Processing an arrival event at time:   22
Processing an arrival event at time:   24
Processing a departure event at time:  25
Processing an arrival event at time:   26
Processing an arrival event at time:   28
Processing an arrival event at time:   30
```

```
Processing a departure event at time:  30
Processing a departure event at time:  35
Processing a departure event at time:  40
Processing a departure event at time:  45
Processing a departure event at time:  50
Processing an arrival event at time:   88
Processing a departure event at time:  91
Simulation Ends

Final Statistics:

   Total number of people processed: 10
   Average amount of time spent waiting: 5.6
```

**7.** Modify and expand the event-driven simulation program that you wrote in Programming Problem 6.

**a.** Add an operation that displays the event list, and use it to check your hand trace in Exercise 11.

**b.** Add some statistics to the simulation. For example, compute the maximum wait in line, the average length of the line, and the maximum length of the line.

**c.** Modify the simulation so that it accounts for three tellers, each with a distinct line. You should keep in mind that there should be

- Three queues, one for each teller
- 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 list

Run both this simulation and the original simulation on several sets of input data. How do the statistics compare?

**d.** The bank is considering the following change: Instead of having three distinct lines (one for each teller), there will be a single line for the three tellers. The person at the front of the line will go to the first available teller. Modify the simulation of part *c* to account for this variation. Run both simulations on several sets of input data. How do the various statistics compare (averages and maximums)? What can you conclude about having a single line as opposed to having distinct lines?

**8.** The people who run the Motor Vehicle Department (MVD) have a problem. They are concerned that people do not spend enough time 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 until they are called for license renewal.
- Once they have completed their desired transaction, they must go and wait in line for the cashier.
- When they finally get to the front of the cashier's line, if they expect to pay by check, they are told that all checks must get approved. To do this, it is necessary to go to the check-approver's table and then reenter the cashier's line at the end.