Queues and Priority Queues

Chapter 13

The ADT Queue

- Like a line of people
  - First person in line is first person served
  - New elements of queue enter at its back
  - Items leave the queue from its front
- Called FIFO behavior
  - **F**irst **I**n **F**irst **O**ut

The ADT Queue

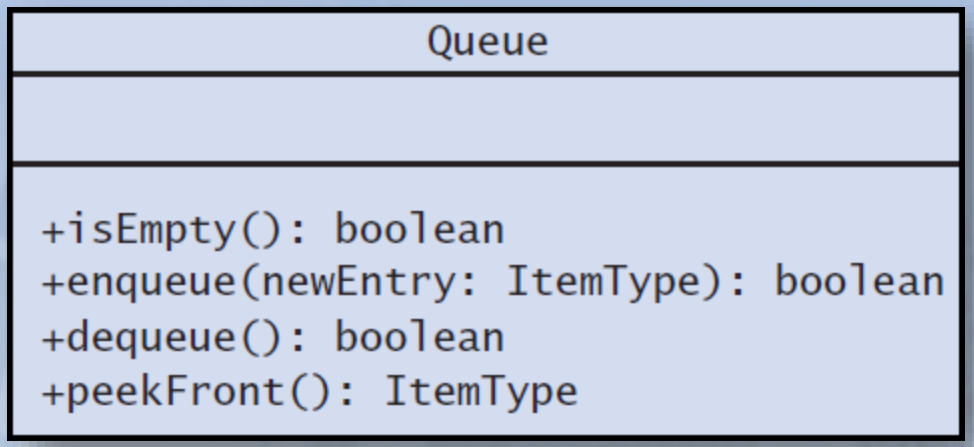

FIGURE 13-1 UML diagram for the class Queue

The ADT Queue

Operation	Queue after operation
aQueue = *an empty queue*	
aQueue.enqueue(5)	5
aQueue.enqueue(2)	5 2
aQueue.enqueue(7)	5 2 7
aQueue.peekFront()	5 2 7 (Returns 5)
aQueue.dequeue()	2 7
aQueue.dequeue()	7

Front

FIGURE 13-2 Some queue operations

The ADT Queue

```
1  /** @file QueueInterface.h */
2  #ifndef QUEUE_INTERFACE_
3  #define QUEUE_INTERFACE_
4
5  template<class ItemType>
6  class QueueInterface
7  {
8  public:
9     /** Sees whether this queue is empty.
10     @return  True if the queue is empty, or false if not. */
11    virtual bool isEmpty() const = 0;
12
13    /** Adds a new entry to the back of this queue.
14     @post  If the operation was successful, newEntry is at the
15        back of the queue.
16     @param newEntry  The object to be added as a new entry.
17     @return  True if the addition is successful or false if not. */
18     virtual bool enqueue(const ItemType& newEntry) = 0;
```

LISTING 13-1 A C++ interface for queues

The ADT Queue

```
18    virtual bool enqueue(const ItemType& newEntry) = 0;
19
20    /** Removes the front of this queue.
21     @post  If the operation was successful, the front of the queue
22        has been removed.
23     @return  True if the removal is successful or false if not. */
24    virtual bool dequeue() = 0;
25
26    /** Returns the front of this queue.
27     @pre  The queue is not empty.
28     @post  The front of the queue has been returned, and the
29        queue is unchanged.
30     @return  The front of the queue. */
31    virtual ItemType peekFront() const = 0;
32
33    /** Destroys this queue and frees its memory. */
34    virtual ~QueueInterface() { }
35 }; // end QueueInterface
36 #endif
```

LISTING 13-1 A C++ interface for queues

Applications
Reading a String of Characters

```
// Read a string of characters from a single line of input into a queue
aQueue = a new empty queue
while (not end of line)
{
    Read a new character into ch
    aQueue.enqueue(ch)
}
```

Pseudocode to read a string of characters into a queue.

Applications
Recognizing a Palindrome

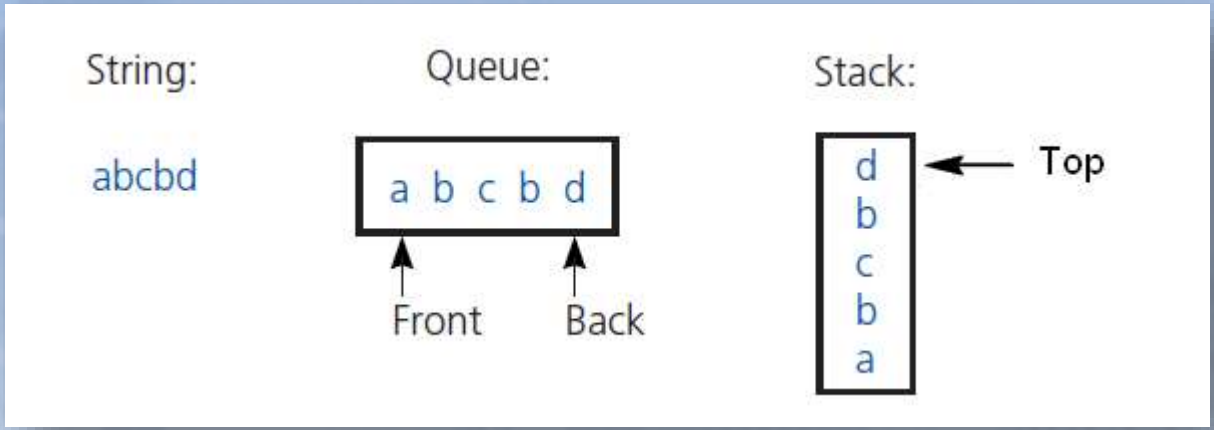

FIGURE 13-3 The results of inserting the characters a, b, c, b, d into both a queue and a stack

- Remove characters from front of queue, top of stack
- Compare each pair removed
- If all pairs match, string is a palindrome

The ADT Priority Queue

- Organize data by priorities
  - Example: weekly “to do” list
- Priority value
  - We will say high value ⇨ high priority
- Operations
  - Test for empty
  - Add to queue in sorted position
  - Remove/get entry with highest priority

The ADT Priority Queue

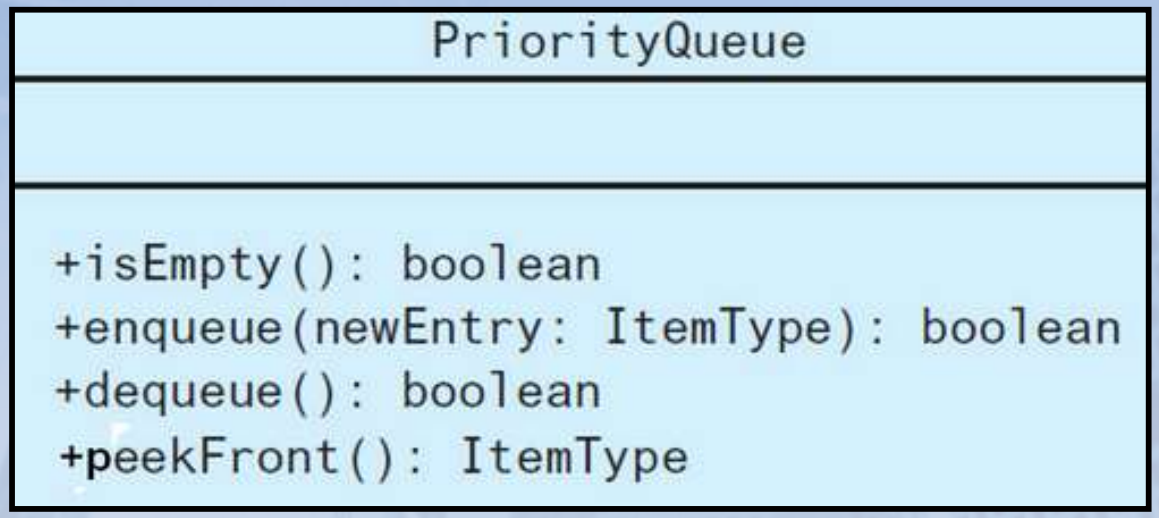

FIGURE 13-4 UML diagram for the class PriorityQueue

Tracking Your Assignments

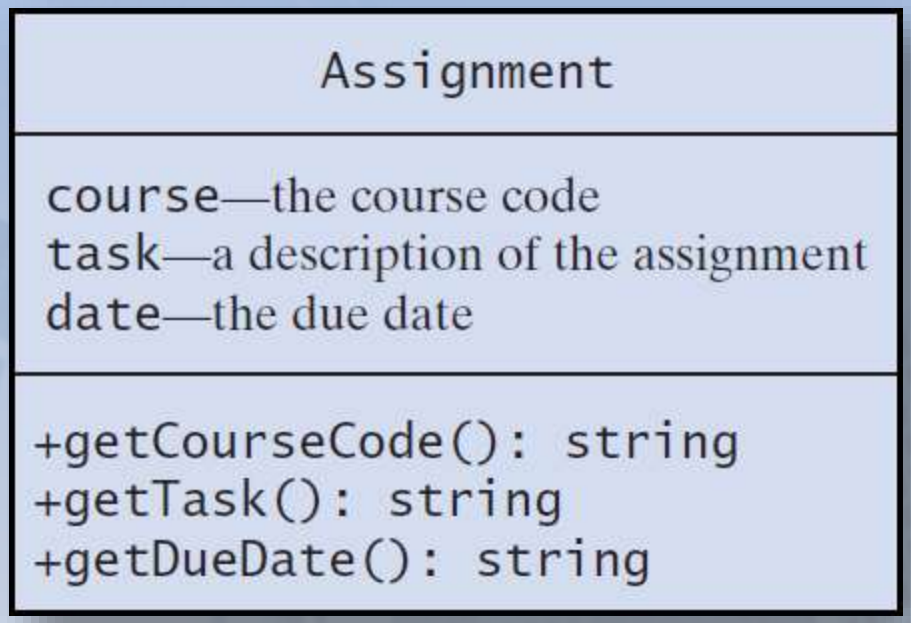

FIGURE 13-5 UML diagram for the class Assignment

Tracking Your Assignments

```
assignmentLog = a new priority queue using due date as the priority value
project = a new instance of Assignment
essay = a new instance of Assignment
quiz = a new instance of Assignment
errand = a new instance of Assignment
assignmentLog.enqueue(project)
assignmentLog.enqueue(essay)
assignmentLog.enqueue(quiz)
assignmentLog.enqueue(errand)
cout << "I should do the following first: "
cout << assignmentLog.peekFront()
```

Pseudocode to organize assignments, responsibilities

Application: Simulation

- Simulation models behavior of systems
- Problem to solve
  - Approximate average time bank customer must wait for service from a teller
  - Decrease in customer wait time with each new teller added

Application: Simulation

Arrival time	Transaction length
20	6
22	4
23	2
30	3

Sample arrival and transaction times

Application: Simulation

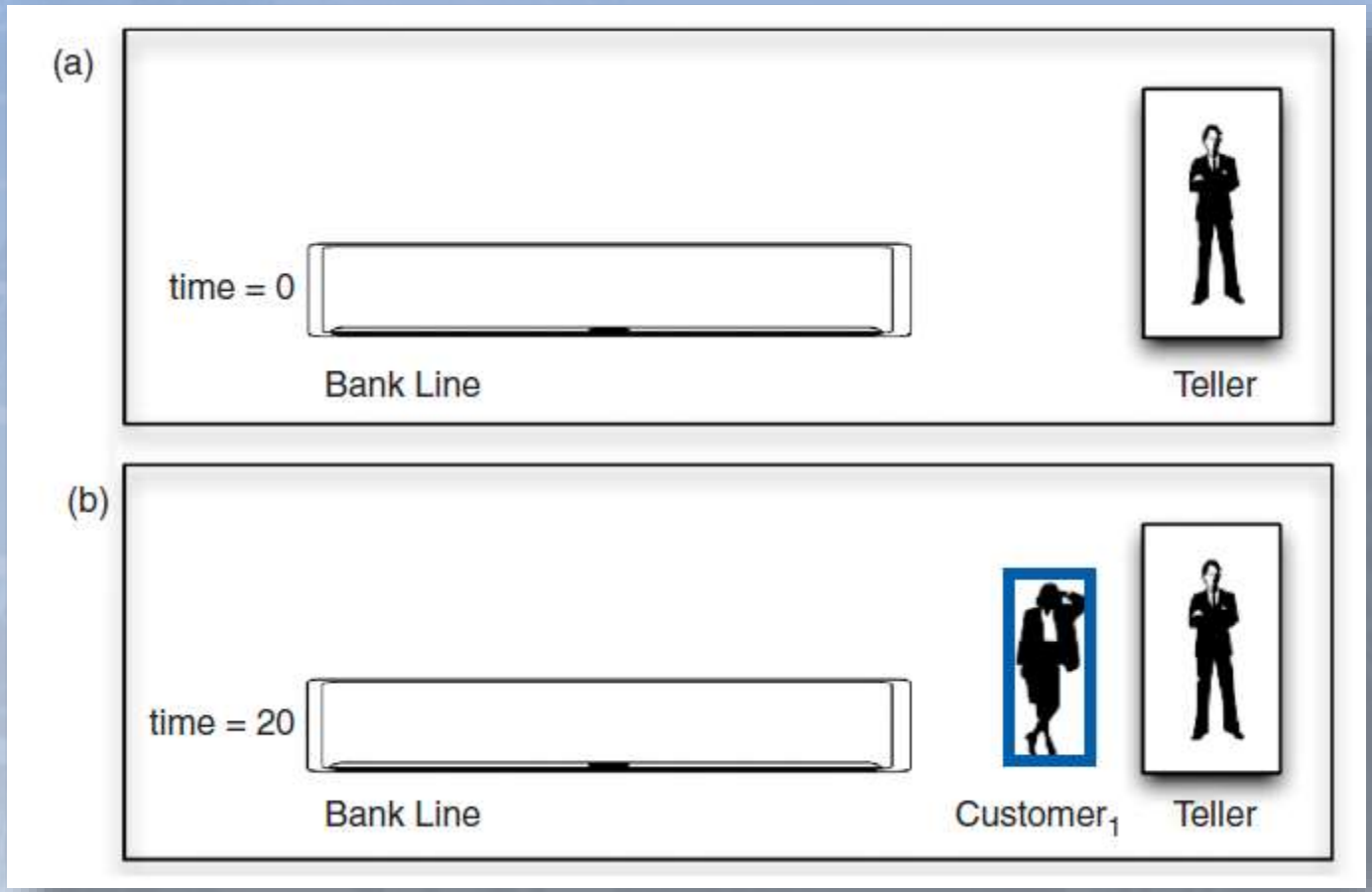

FIGURE 13-6 A bank line at time (a) 0; (b) 20; (c) 22; (d) 26

Application: Simulation

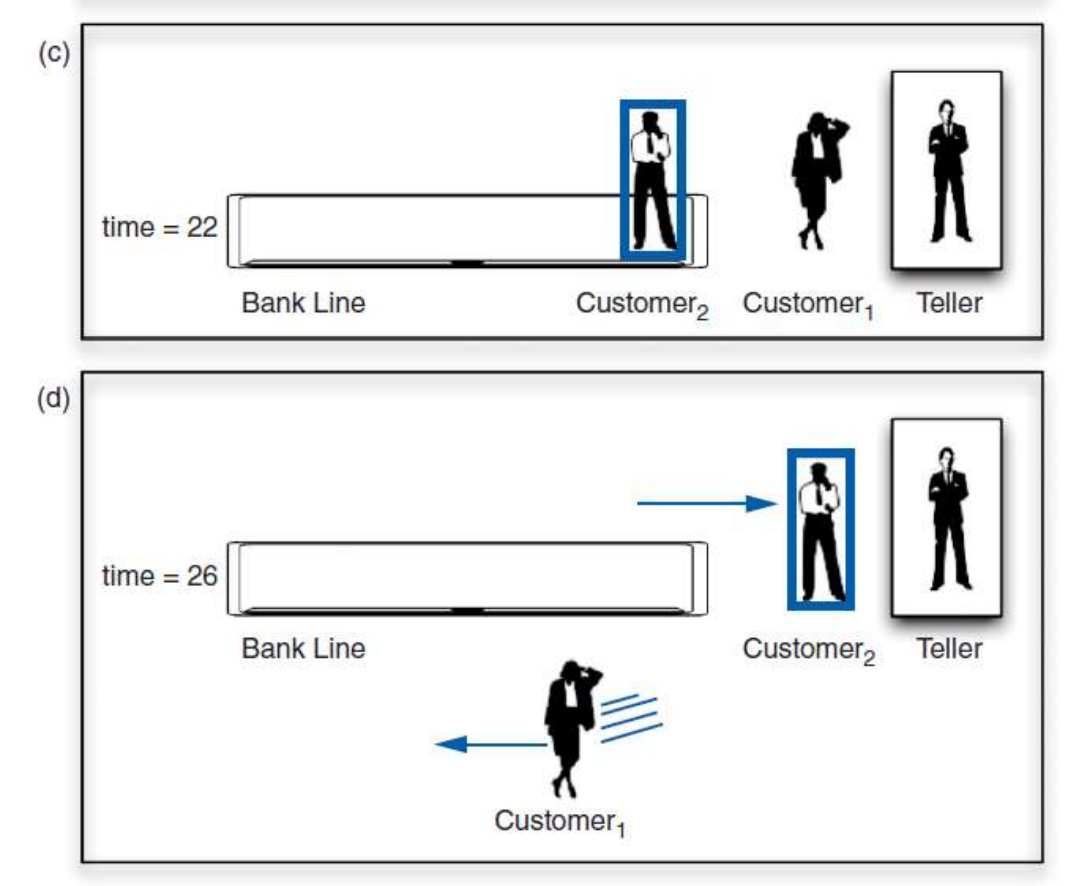

FIGURE 13-6 A bank line at time (a) 0; (b) 20; (c) 22; (d) 26

Application: Simulation

```
Initialize the line to "no customers"
while (events remain to be processed)
{
    currentTime = time of next event
    if (event is an arrival event)
        Process the arrival event
    else
        Process the departure event

    // When an arrival event and a departure event occur at the same time,
    // arbitrarily process the arrival event first
}
```

Pseudocode for an event loop

Application: Simulation

- Time-driven simulation
  - Simulates the ticking of a clock
- Event-driven simulation considers
  - Only the times of certain events,
  - In this case, arrival-s and departures
- Event list contains
  - All future arrival and departure events

Application: Simulation

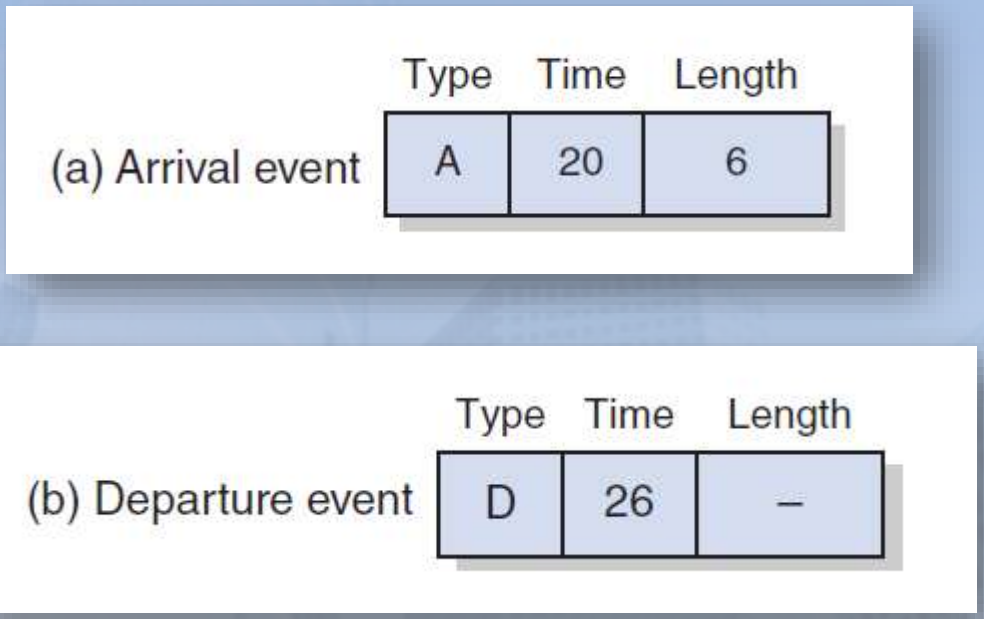

FIGURE 13-7 A typical instance of (a) an arrival event; (b) a departure event

Application: Simulation

- Two tasks required to process each event
  - Update the bank line: Add or remove customers.
  - Update the event queue: Add or remove events.
- New customer
  - Always enters bank line
  - Served while at the front of the line

Application: Simulation

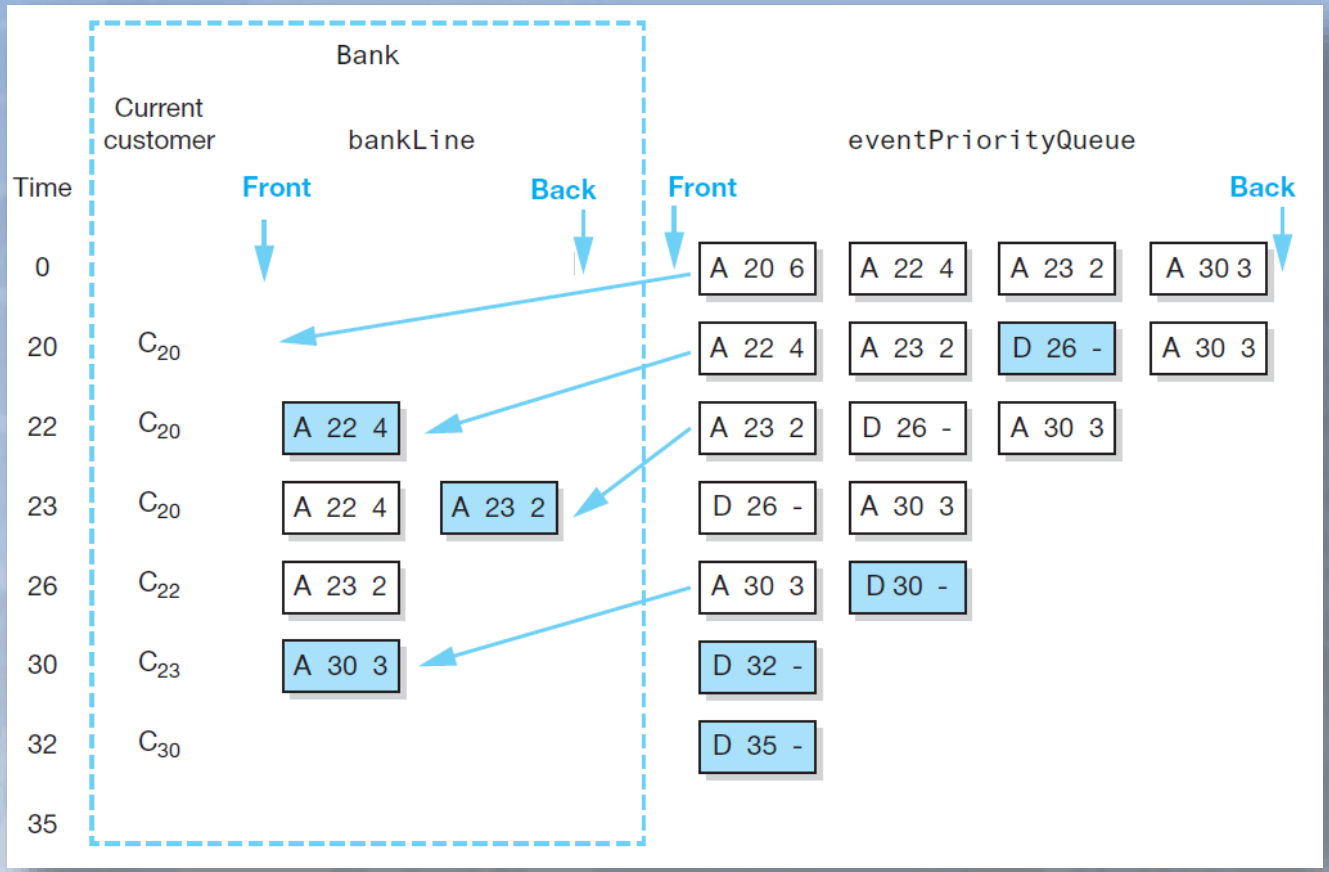

FIGURE 13-8 A trace of the bank simulation algorithm for the data (20, 6), (22, 4), (23, 2), (30, 3)

Position-Oriented and Value-Oriented ADTs

- Position-oriented ATDs
  - Stack, list, queue
- Value-oriented ADTs
  - Sorted list

Position-Oriented and Value-Oriented ADTs

- Comparison of stack and queue operations
  - isEmpty for both
  - pop and dequeue
  - peek and peekFront

Position-Oriented and Value-Oriented ADTs

- ADT list operations generalize stack and queue operations
  - getLength
  - insert
  - remove
  - getEntry

End

Chapter 13