Queues and Priority Queues

Chapter 13

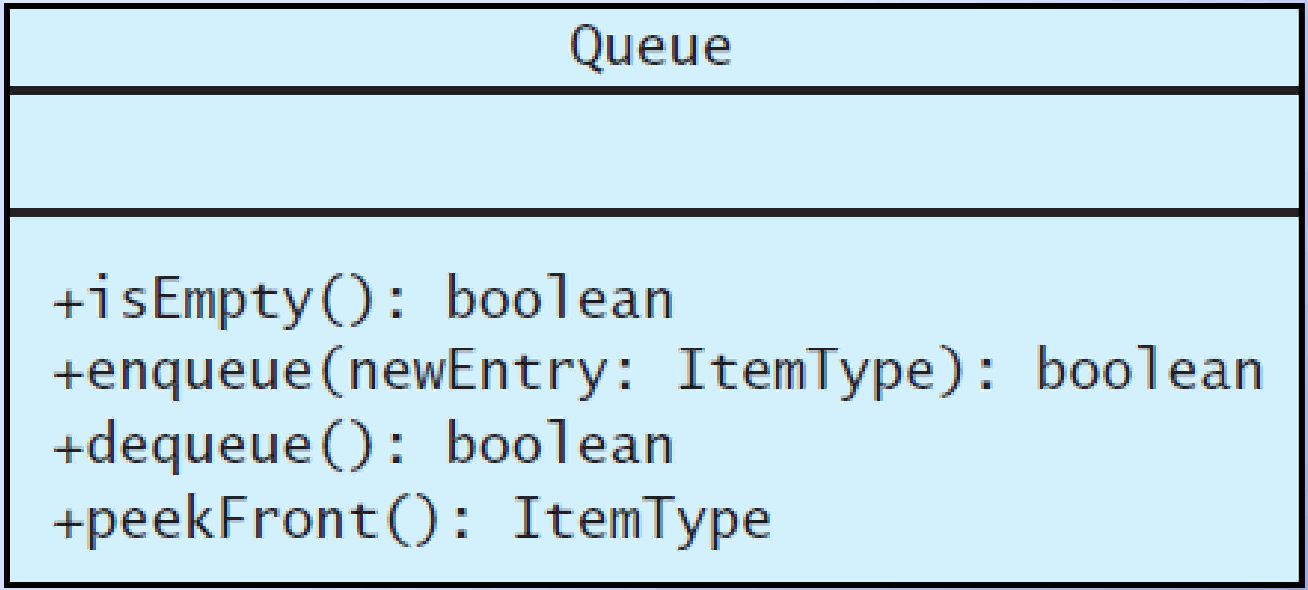
Content

- The ADT Queue
- Simple Applications of the ADT Queue
- The ADT Priority Queue
- The ADT Priority Queue
- Position-Oriented and Value-Oriented ADTs

The ADT Queue

- A queue is like a line of people
  - New items enter at the back (rear) of the queue
  - Items leave the queue from the front
- ADT Queue operations
  - Test whether a queue is empty.
  - Add new entry to back of queue.
  - Remove entry at front of queue
  - Get the entry added earliest to queue.

The ADT Queue



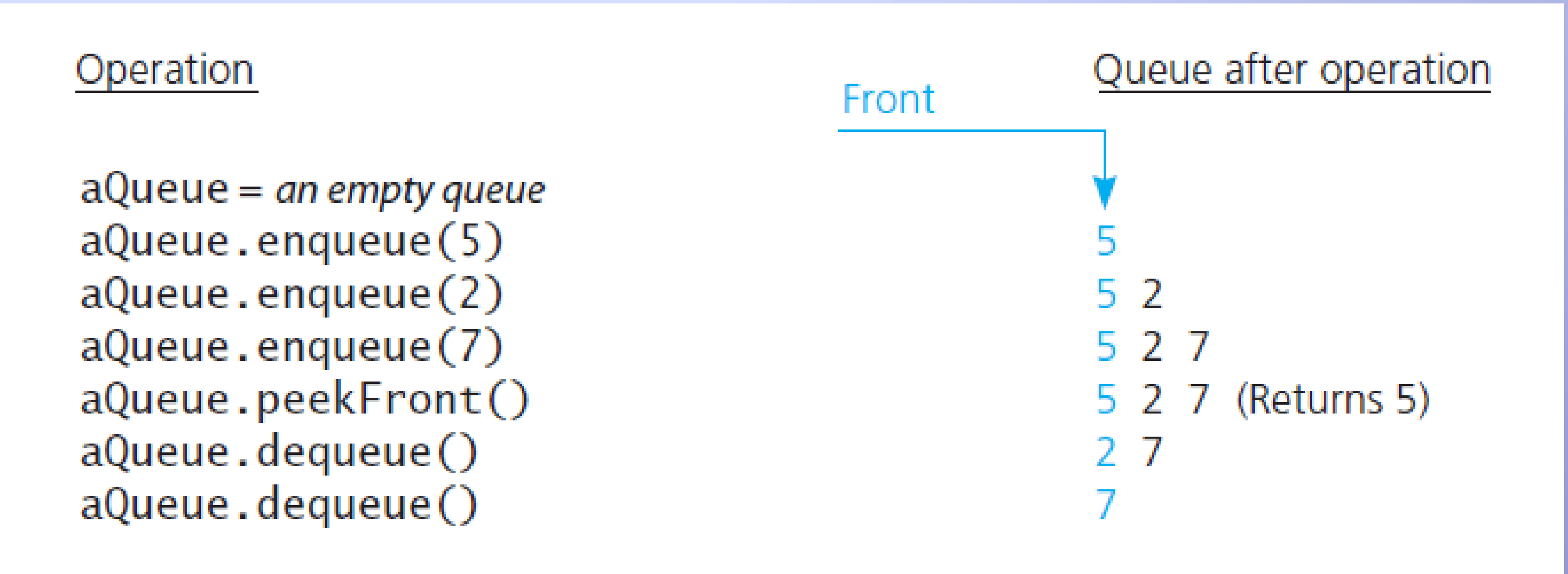
Queue

+isEmpty(): boolean
+enqueue(newEntry: ItemType): boolean
+dequeue(): boolean
+peekFront(): ItemType

FIGURE 13-1 UML diagram for the class **`Queue`**

The ADT Queue

- View interface for queues, Listing 13-1

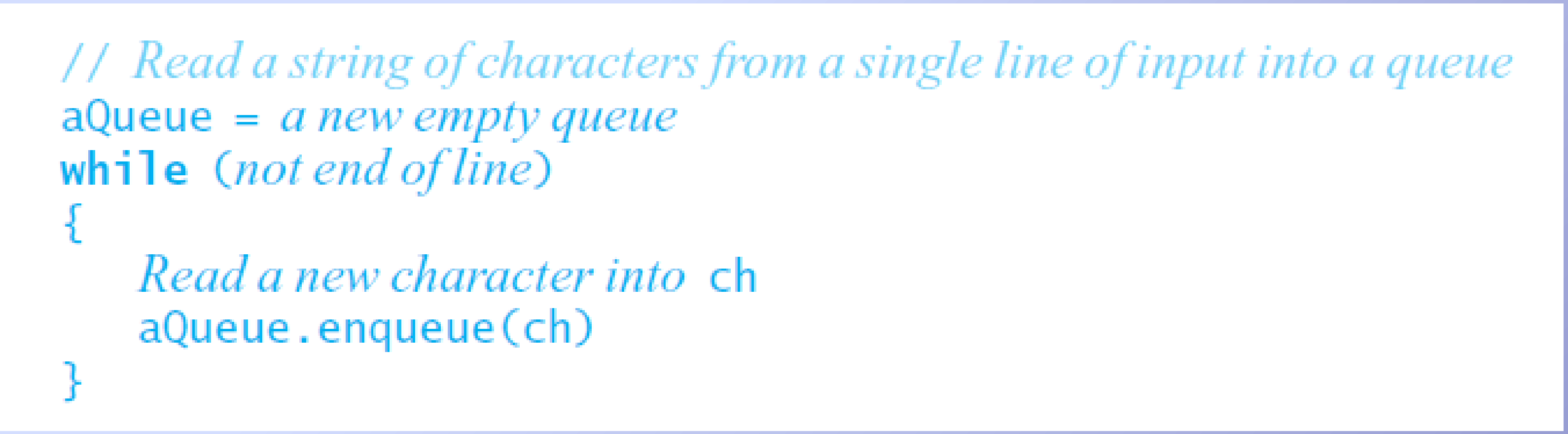


Operation	Queue after operation
aQueue = *an empty queue*	
aQueue.enqueue(5)	5
aQueue.enqueue(2)	5 2
aQueue.enqueue(7)	5 2 7
aQueue.peekFront()	5 2 7 (Returns 5)
aQueue.dequeue()	2 7
aQueue.dequeue()	7

FIGURE 13-2 Some queue operations

Simple Applications of the ADT Queue

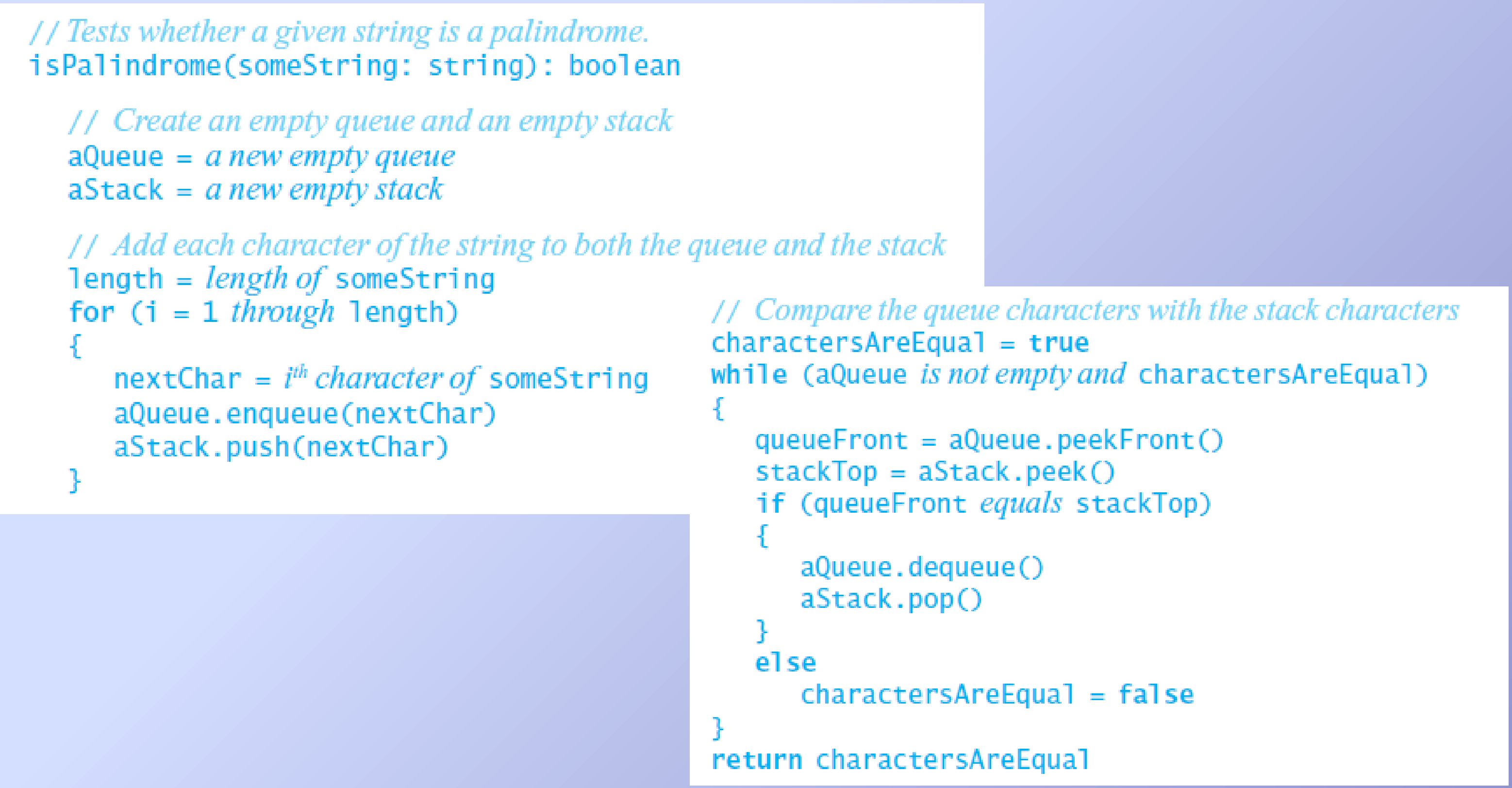
- Reading a string of characters



```
// 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)
}
```

Simple Applications of the ADT Queue

- Recognizing palindromes



```
// Tests whether a given string is a palindrome.
isPalindrome(someString: string): boolean

   // Create an empty queue and an empty stack
   aQueue = a new empty queue
   aStack = a new empty stack

   // Add each character of the string to both the queue and the stack
   length = length of someString
   for (i = 1 through length)
   {
      nextChar = ith character of someString
      aQueue.enqueue(nextChar)
      aStack.push(nextChar)
   }
```

```
// Compare the queue characters with the stack characters
charactersAreEqual = true
while (aQueue is not empty and charactersAreEqual)
{
   queueFront = aQueue.peekFront()
   stackTop = aStack.peek()
   if (queueFront equals stackTop)
   {
      aQueue.dequeue()
      aStack.pop()
   }
   else
      charactersAreEqual = false
}
return charactersAreEqual
```

Recognizing Palindromes

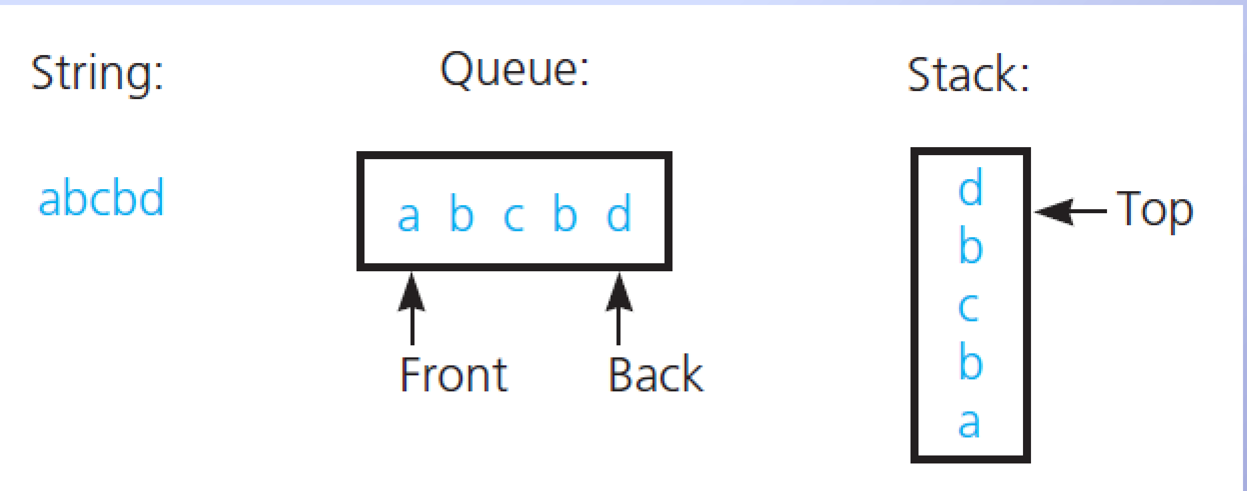
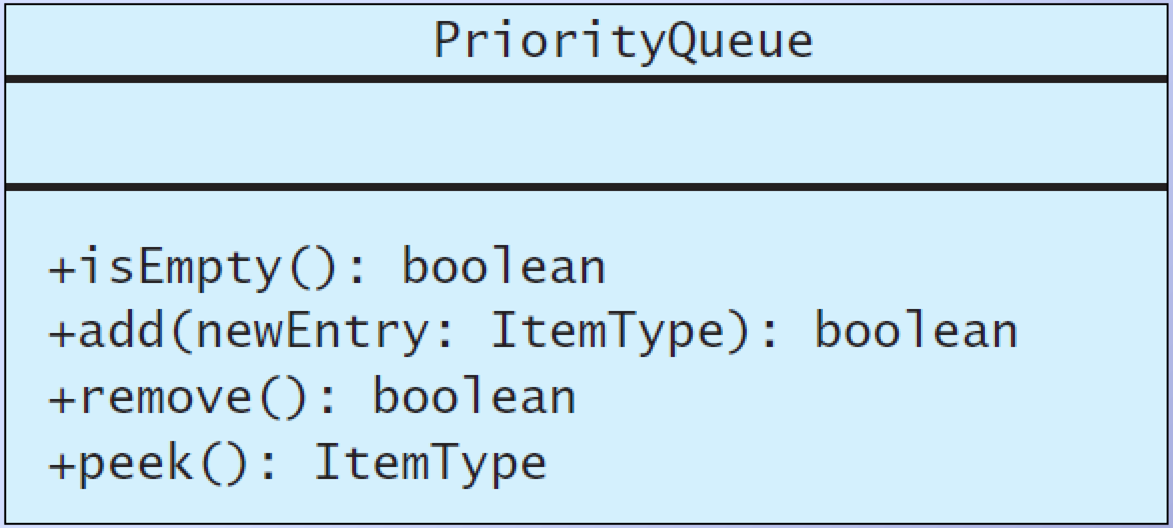


FIGURE 13-3 The results of inserting the characters a, b, c, b, d into both a queue and a stack

The ADT Priority Queue

- Example : triage in a hospital emergency room
- Operations
  - Test whether priority queue empty.
  - Add new entry to priority queue in sorted position based on priority value.
  - Remove from priority queue entry with highest priority
  - Get entry in priority queue with highest priority.

The ADT Priority Queue

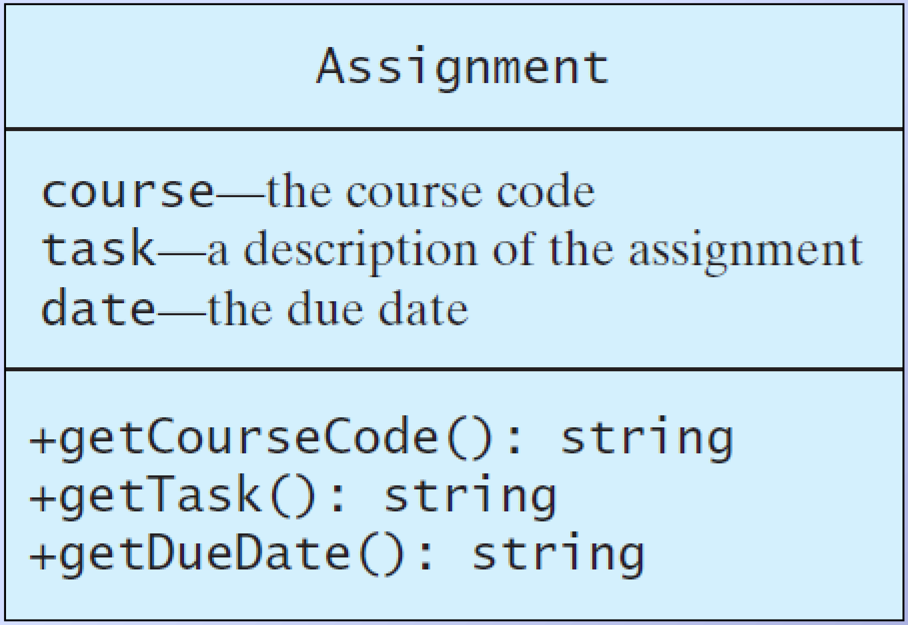


PriorityQueue

+isEmpty(): boolean
+add(newEntry: ItemType): boolean
+remove(): boolean
+peek(): ItemType

FIGURE 13-4 UML diagram for the class **PriorityQueue**

Tracking Your Assignments



```
Assignment
```

course—the course code
task—a description of the assignment
date—the due date

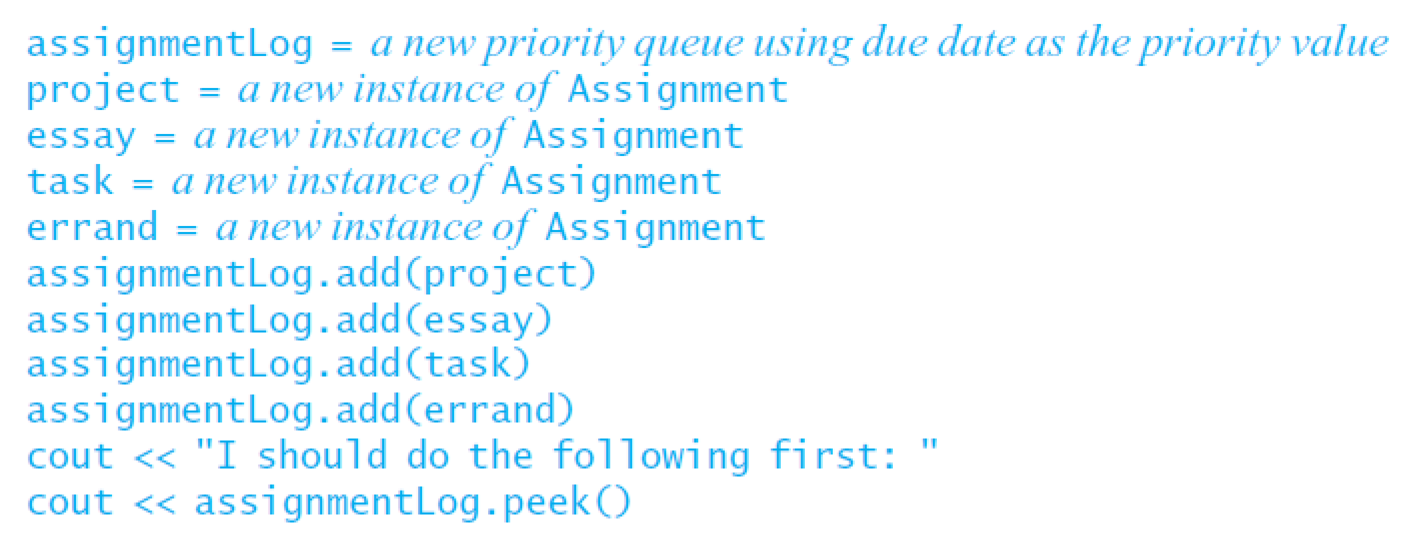
```
+getCourseCode(): string
+getTask(): string
+getDueDate(): string
```

FIGURE 13-5 UML diagram for the class Assignment

Data Structures and Problem Solving with C++: Walls and Mirrors, Carrano and Henry, © 2013

Tracking Your Assignments

- Pseudocode for tracking 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
task = a new instance of Assignment
errand = a new instance of Assignment
assignmentLog.add(project)
assignmentLog.add(essay)
assignmentLog.add(task)
assignmentLog.add(errand)
cout << "I should do the following first: "
cout << assignmentLog.peek()
```

Application: Simulation

- Simulation: technique for modeling behavior of natural and human-made systems.
- Problem to simulate: model bank queue wait times
  - Average time customer waits to begin service from current single teller
  - Decrease in customer wait time with each new teller added

Application: Simulation

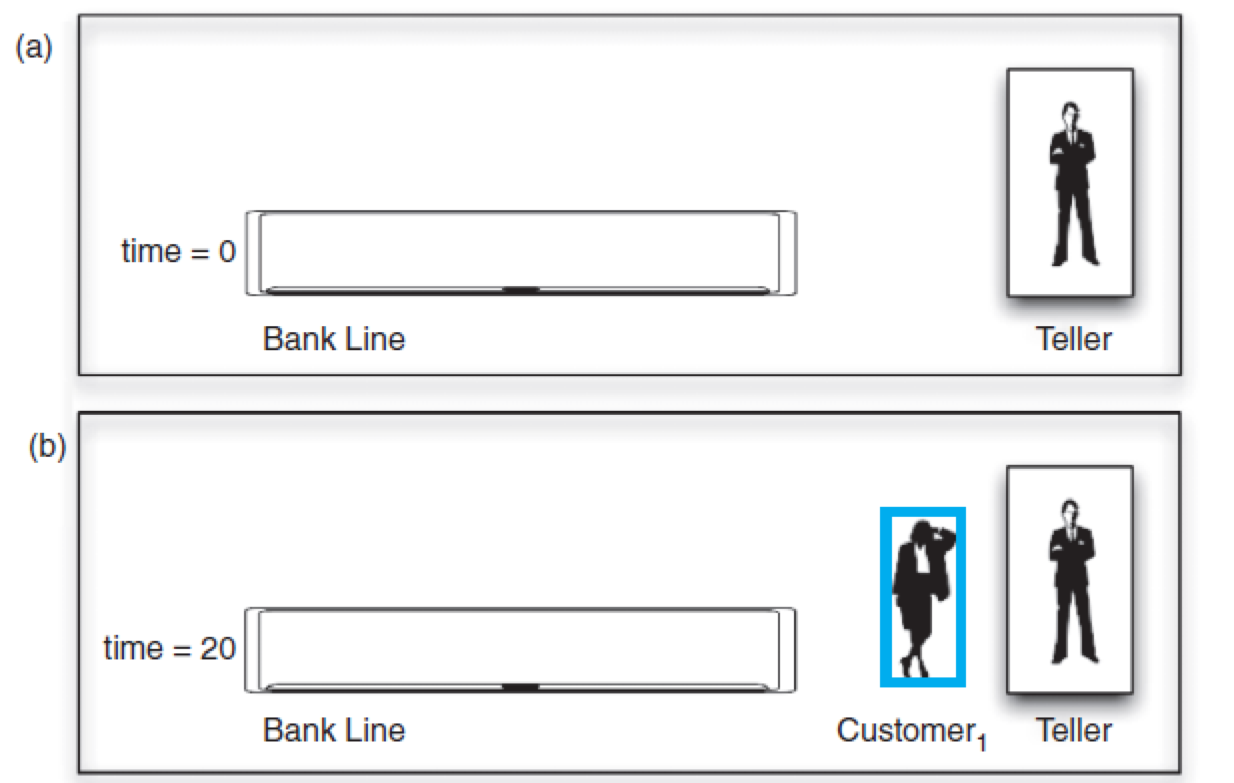


FIGURE 13-6 A bank line at time (a) 0; (b) 20;

Application: Simulation

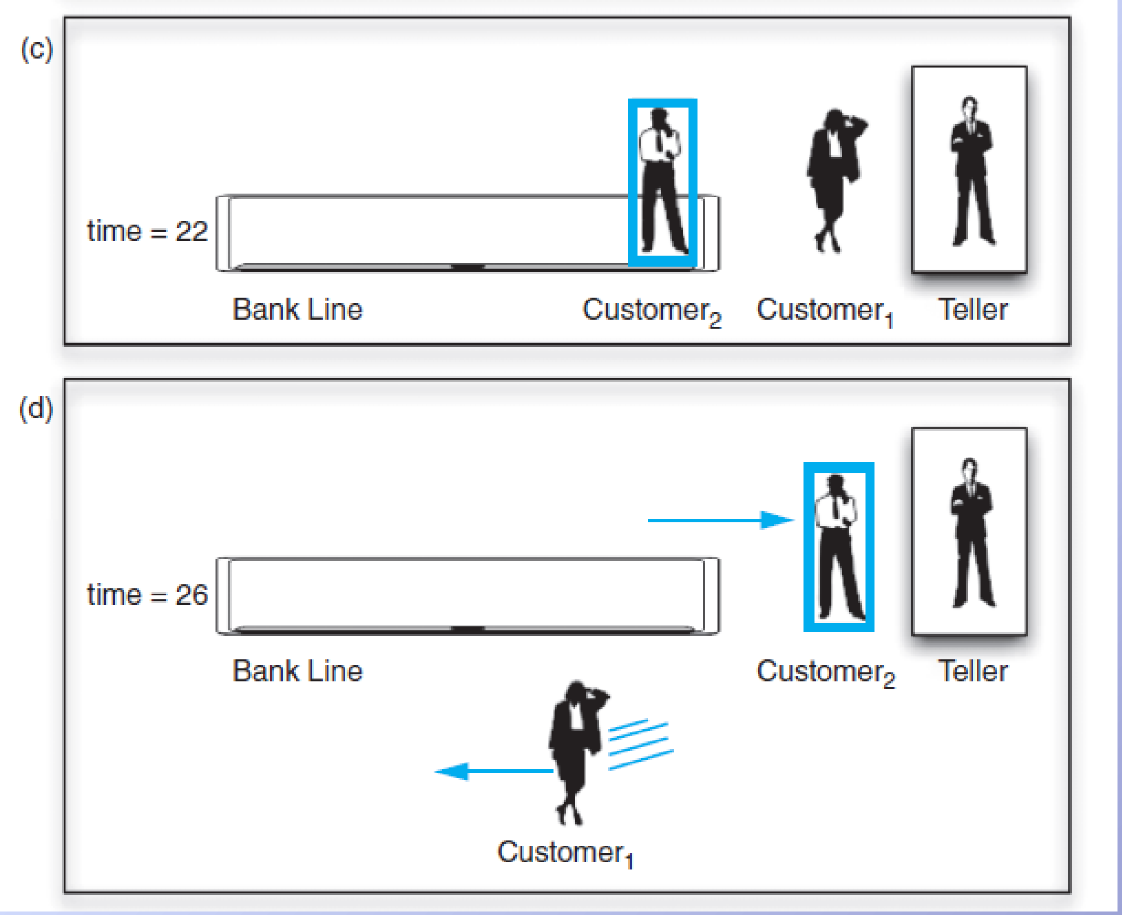
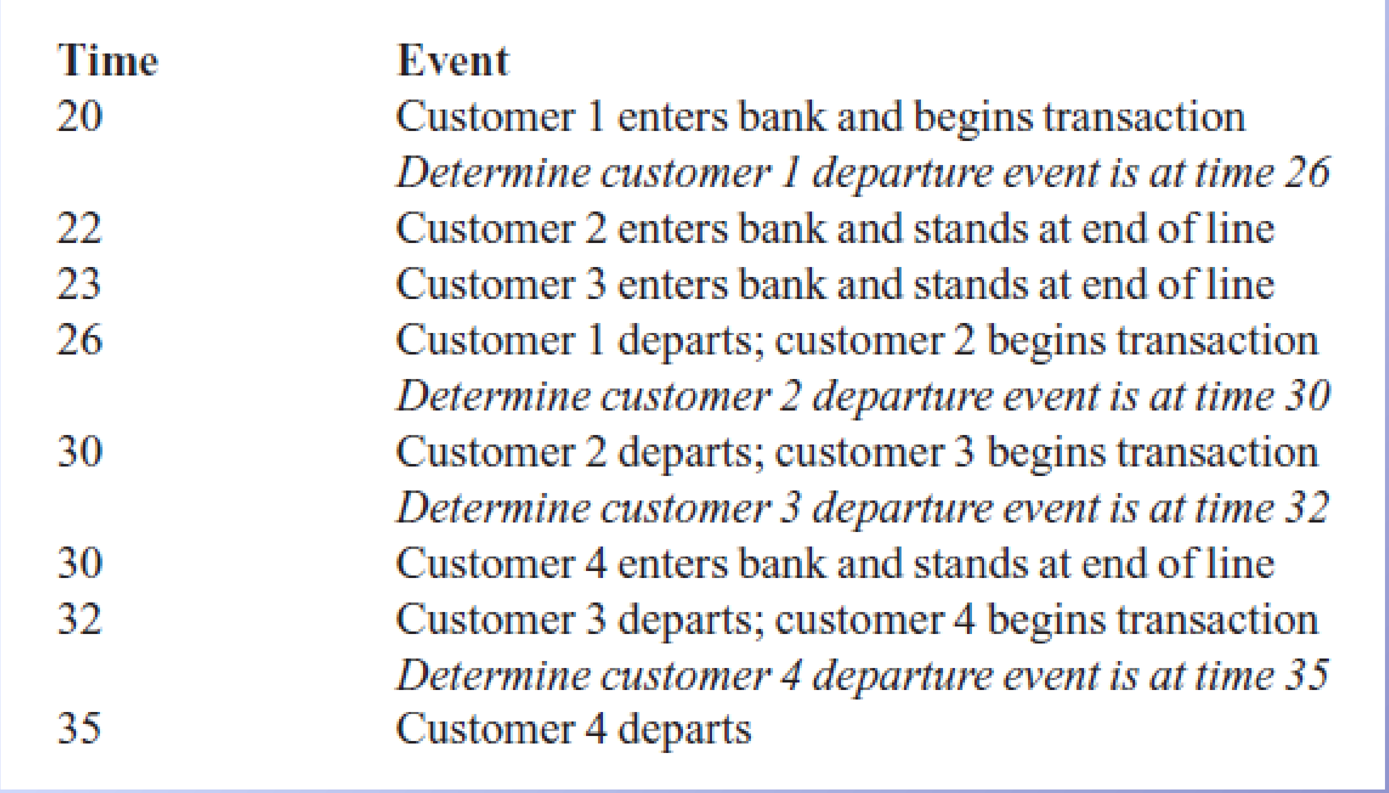


FIGURE 13-6 A bank line at time (c) 22; (d) 26

Application: Simulation

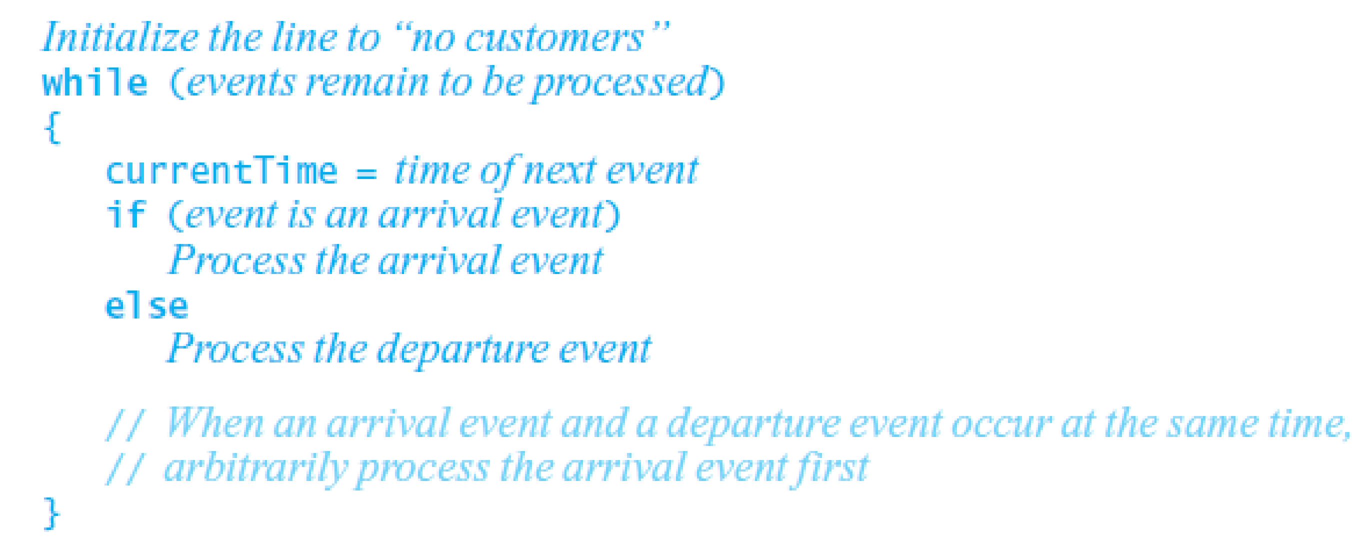
- Results of a simulation:



Time	Event
20	Customer 1 enters bank and begins transaction
	Determine customer 1 departure event is at time 26
22	Customer 2 enters bank and stands at end of line
23	Customer 3 enters bank and stands at end of line
26	Customer 1 departs; customer 2 begins transaction
	Determine customer 2 departure event is at time 30
30	Customer 2 departs; customer 3 begins transaction
	Determine customer 3 departure event is at time 32
30	Customer 4 enters bank and stands at end of line
32	Customer 3 departs; customer 4 begins transaction
	Determine customer 4 departure event is at time 35
35	Customer 4 departs

Application: Simulation

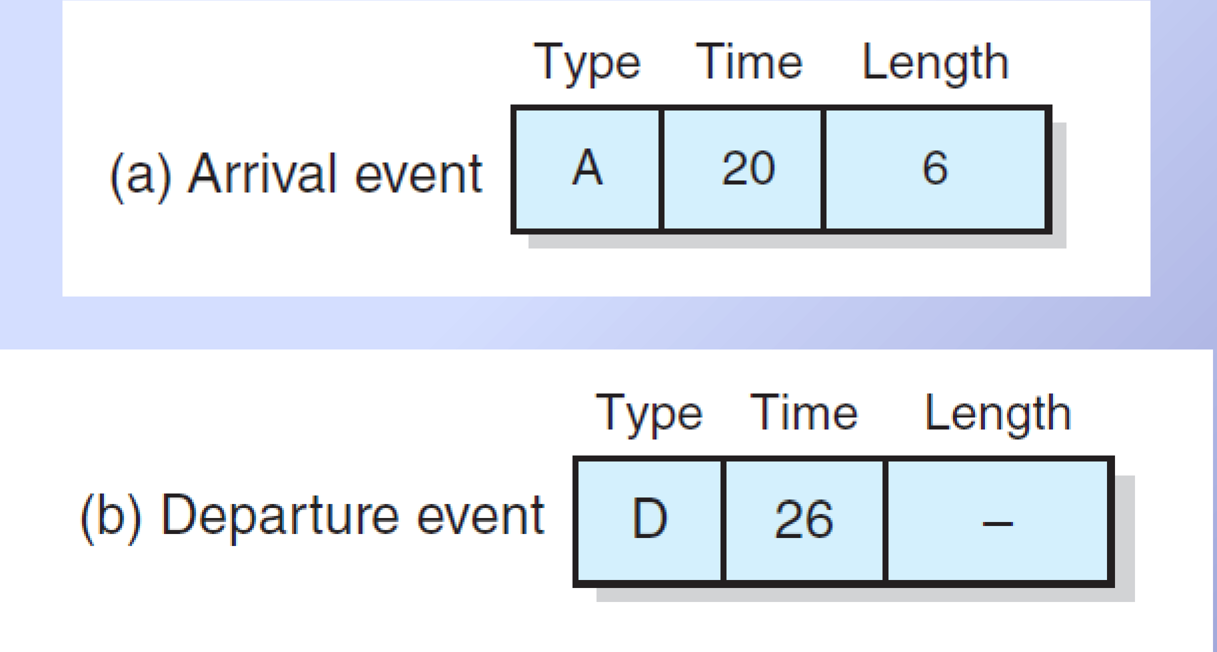
- An event-driven simulation considers only times of certain events
    - In this case, arrivals and departures
- Algorithm



```
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
}
```

Application: Simulation

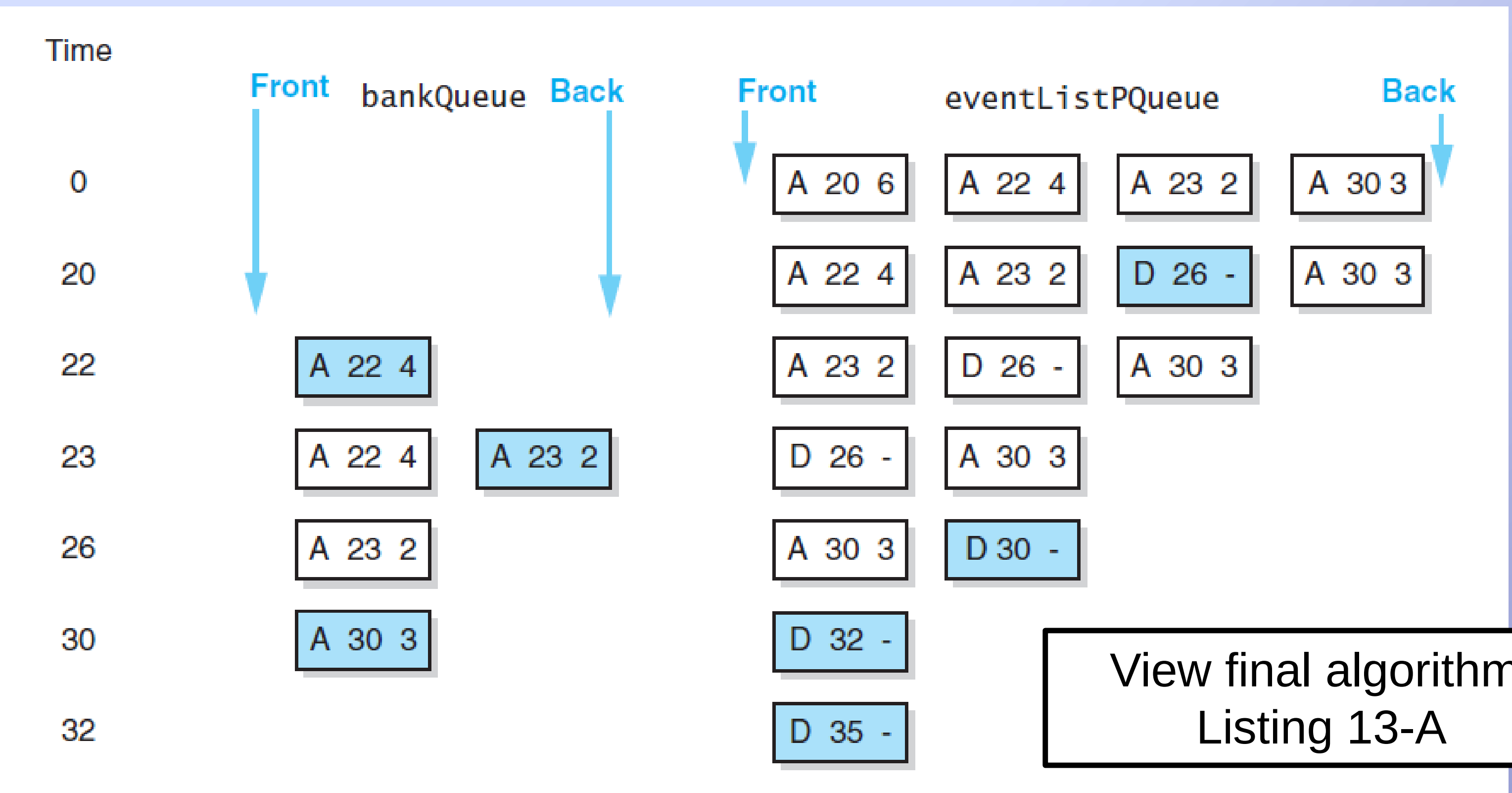


	Type	Time	Length
(a) Arrival event	A	20	6

	Type	Time	Length
(b) Departure event	D	26	–

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

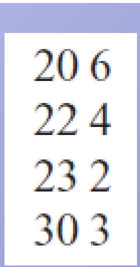
Application: Simulation



Time	Front bankQueue	Back	Front eventListPQueue			Back
0			A 20 6	A 22 4	A 23 2	A 30 3
20			A 22 4	A 23 2	D 26 -	A 30 3
22	A 22 4		A 23 2	D 26 -	A 30 3	
23	A 22 4	A 23 2	D 26 -	A 30 3		
26	A 23 2		A 30 3	D 30 -		
30	A 30 3		D 32 -			
32			D 35 -			

View final algorithm, Listing 13-A

FIGURE 13-8 A trace of the bank simulation algorithm for the data



20 6
22 4
23 2
30 3

Comparison of Stack and Queue Operations

- Note same task of **`isEmpty`** functions
- **`push`** and **`enqueue`** similar job
- **`pop`** and **`dequeue`** similar tasks
- Also **`peek`** and **`peekFront`**

- Differences are whether function manipulates front or back of ADT

ADT List Generalizes Stack and Queue

- List has **`getLength`**
- **`insert`** replicates **`push`** and **`enqueue`**
- **`remove`** replicates **`pop`** and **`dequeue`**
- **`getEntry`** replicates **`peek`** and **`peekFront`**

End

Chapter 13