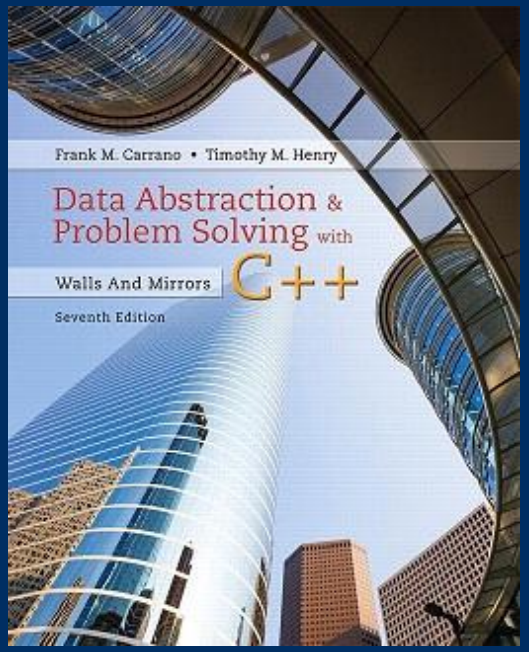
Chapter 13

Queues and Priority Queues

CS 302 - Data Structures

M. Abdullah Canbaz



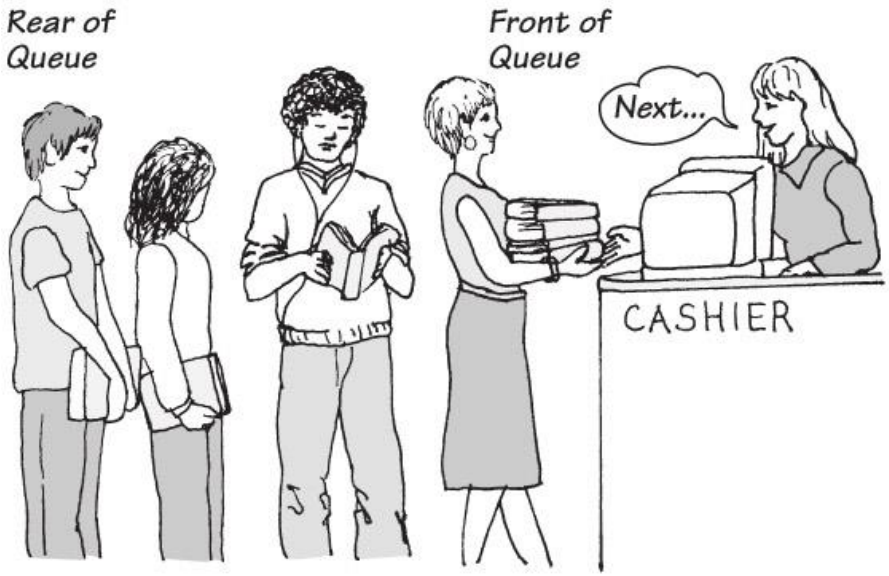
Modified from authors' slides

Reminders

- Assignment 4 is available
  - Due April 2nd at 2pm
- TA
  - Shehryar Khattak,
    Email: *shehryar [at] nevada {dot} unr {dot} edu*,
    Office Hours: Friday, 11:00 am - 1:00 pm at ARF 116
- Quiz 7 is available
  - Today between 4pm to 11:59pm

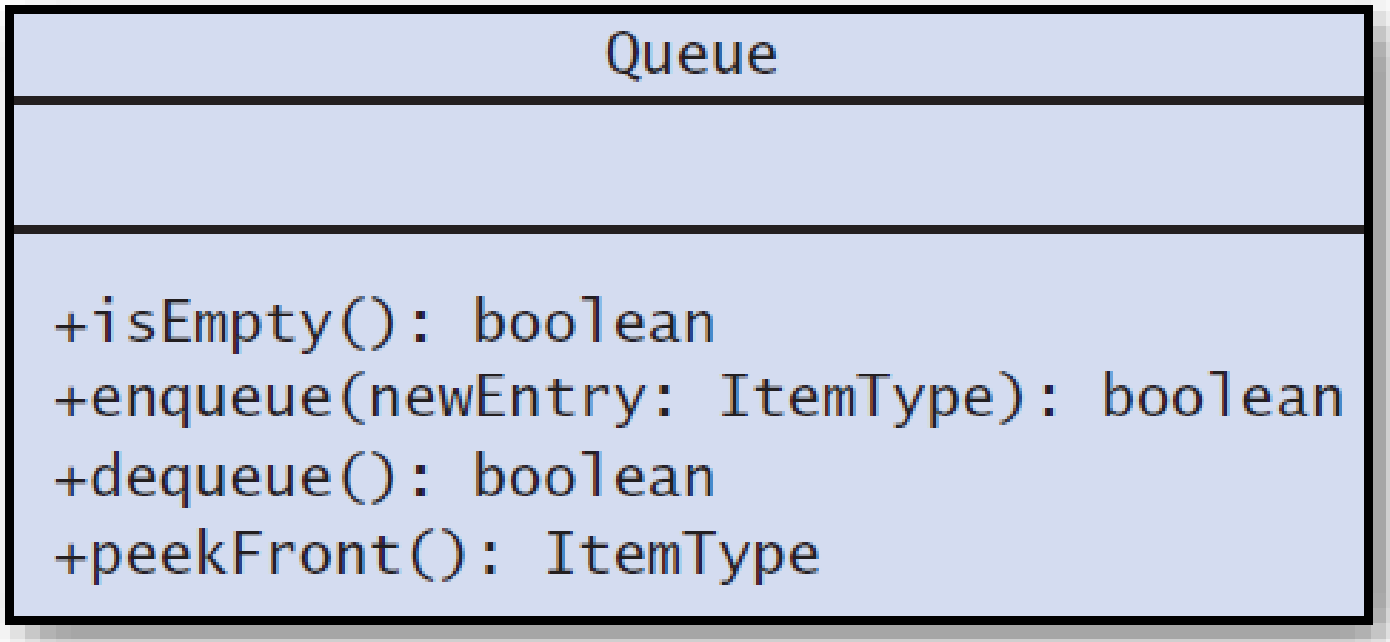
The ADT Queue

- 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

- UML diagram for the class Queue



The ADT Queue

- Some queue operations

Operation	Queue after operation
aQueue = *an empty queue*	Front →
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

The ADT Queue

- A C++ interface for queues

```
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;
```

The ADT Queue

- A C++ interface for queues

```
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
```

Applications Reading a String of Characters

- Pseudocode to read a string of characters into a queue.

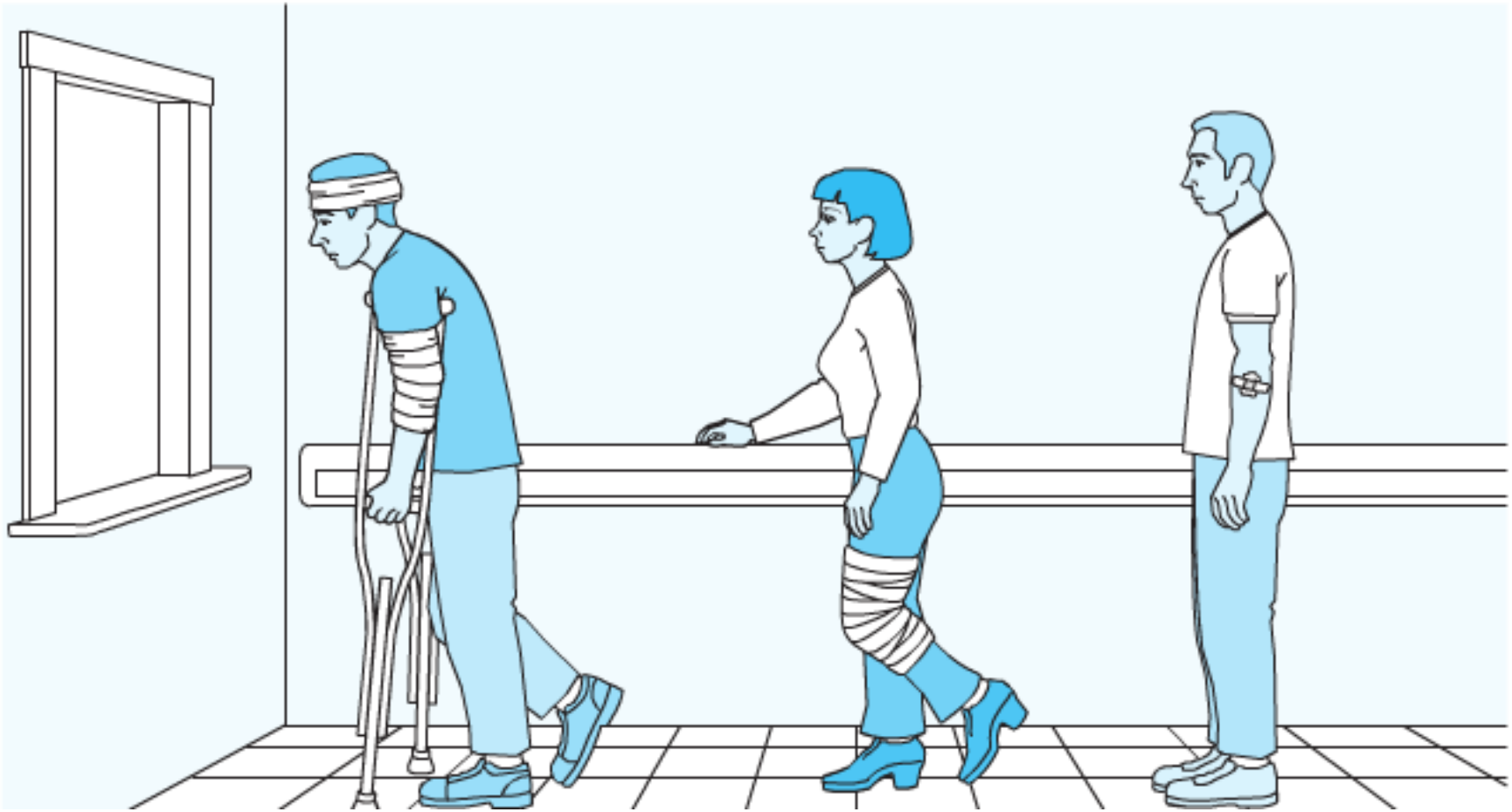
```
// Read a string of characters from a single line of input into a queue
aQueue = a new empty queue
while (not end of line)
{
    Read a new character into ch
    aQueue.enqueue(ch)
}
```

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

Real-life Priority Queue

- Example : triage in a hospital emergency room



Priority Queue

A priority queue is an ADT with the property that only the highest-priority element can be accessed at any time.

Queue

Enque an item

Item returned has been in the queue the longest amount of time.

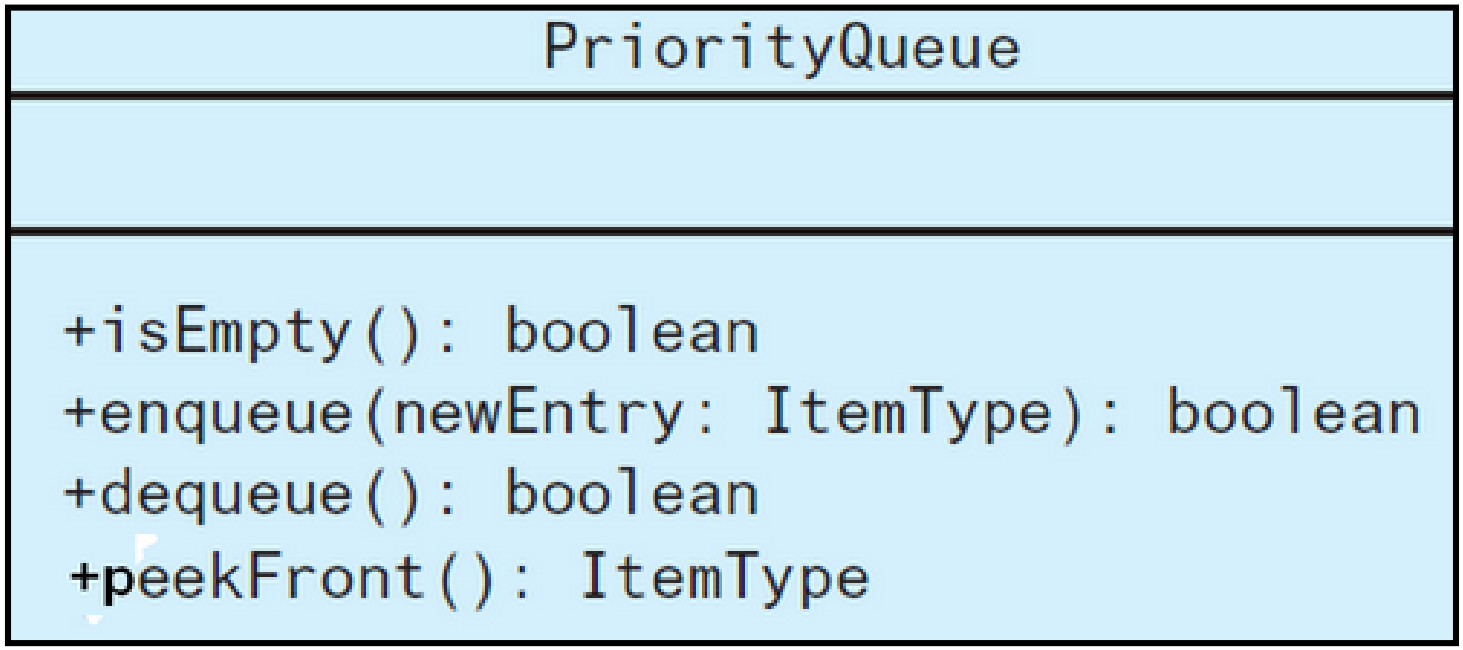
Priority Queue

Enque a pair <item, priority>

Item returned has the highest priority.

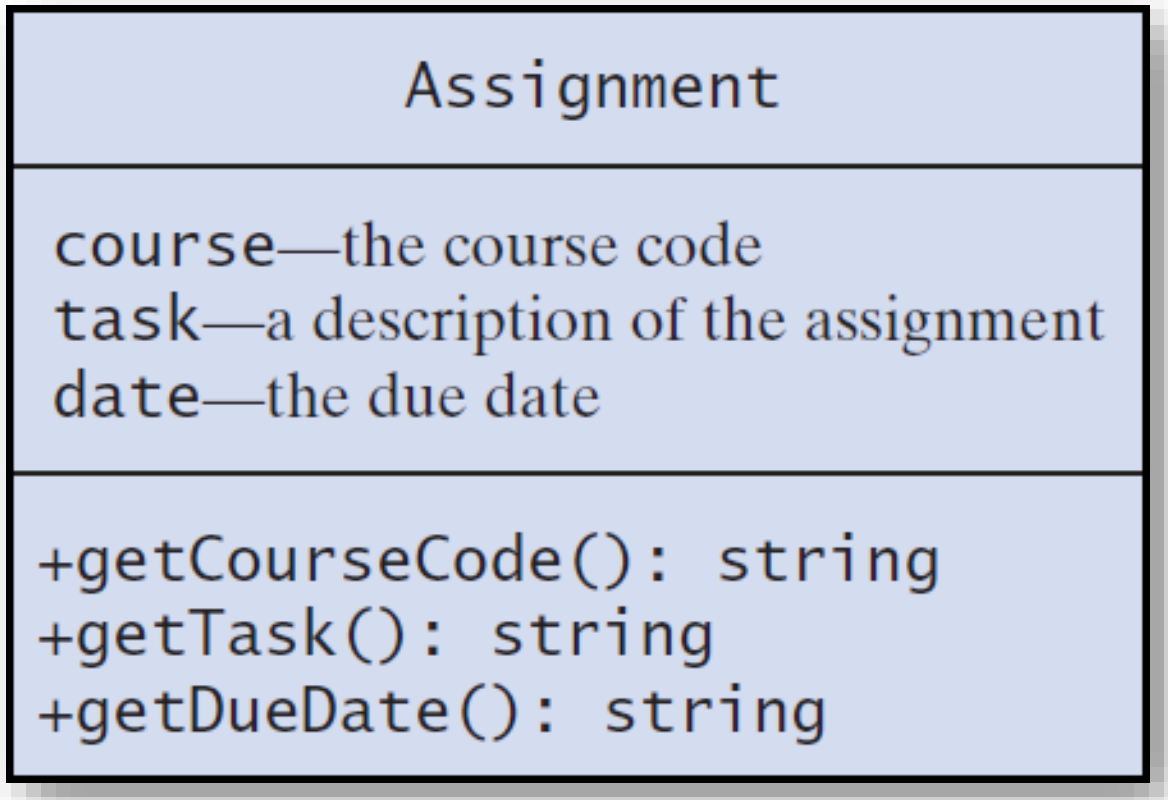
The ADT Priority Queue

- UML diagram for the class PriorityQueue



Tracking Your Assignments

- UML diagram for the class Assignment



Tracking Your Assignments

- Pseudocode to organize assignments, responsibilities

```
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()
```

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

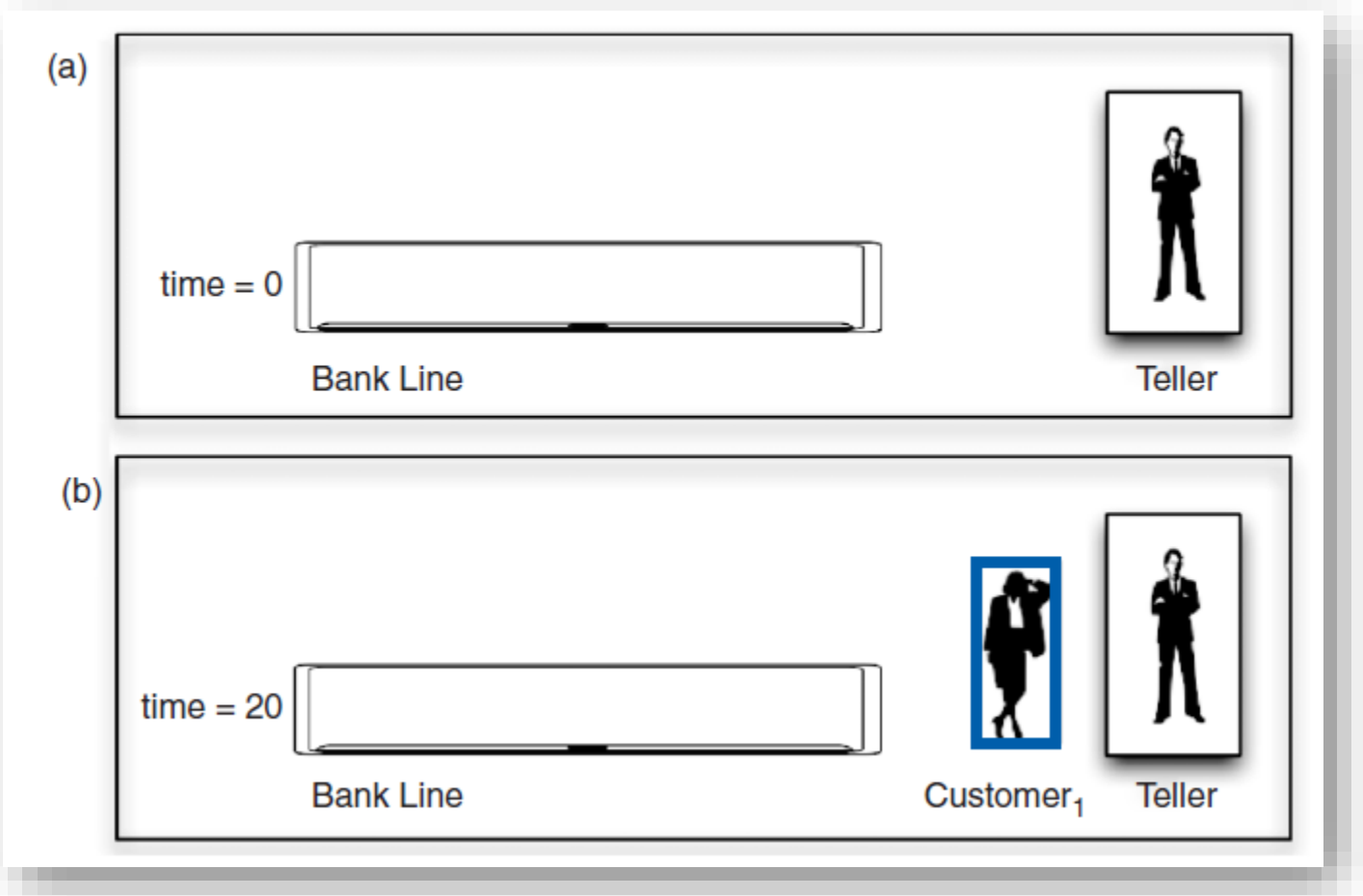
Application: Simulation

- Sample arrival and transaction times

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

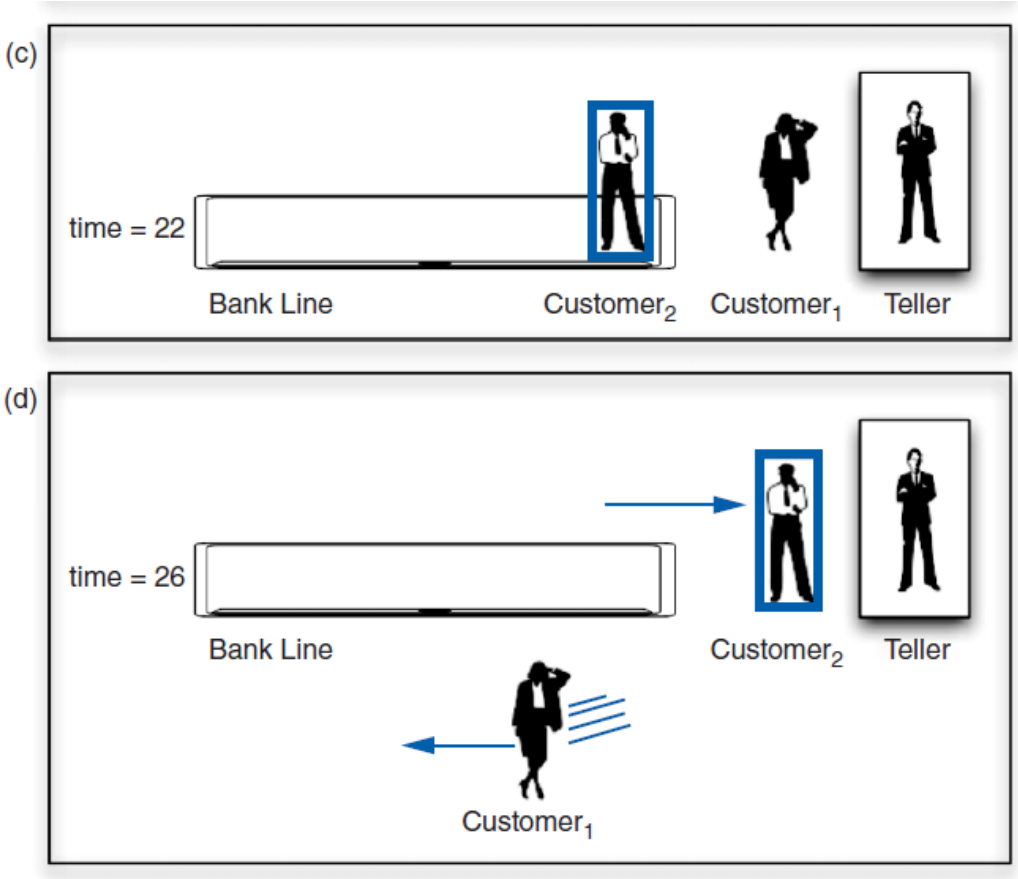
Application: Simulation

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



Application: Simulation

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



Application: Simulation

- Pseudocode for an event loop

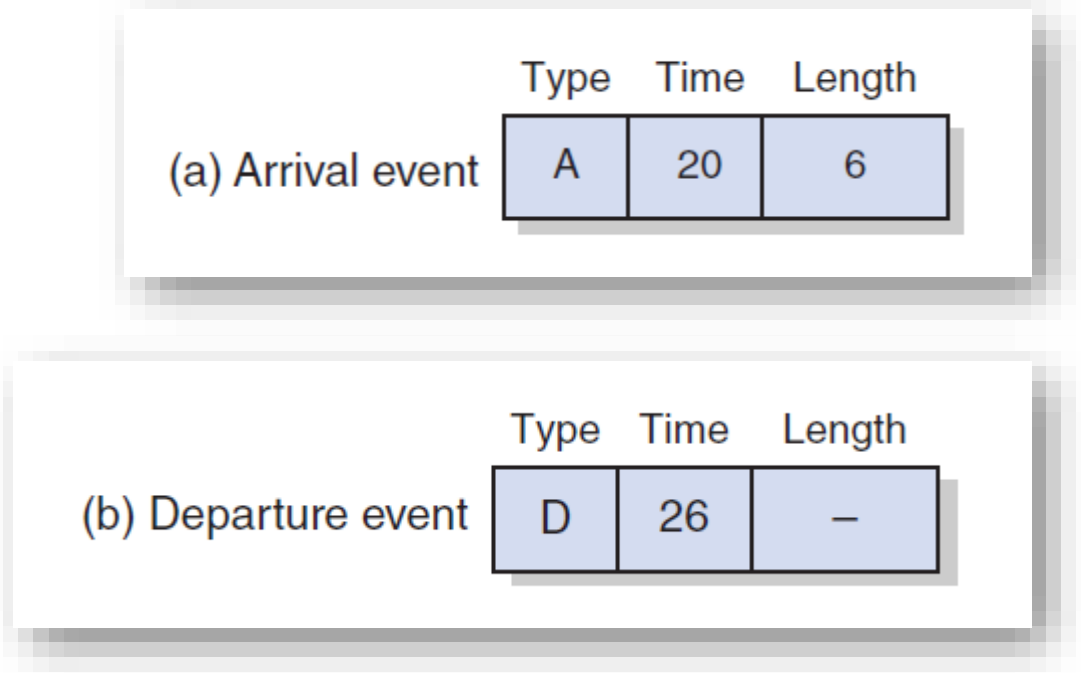
```
Initialize the line to "no customers"
while (events remain to be processed)
{
    currentTime = time of next event
    if (event is an arrival event)
        Process the arrival event
    else
        Process the departure event

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

Application: Simulation

- 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



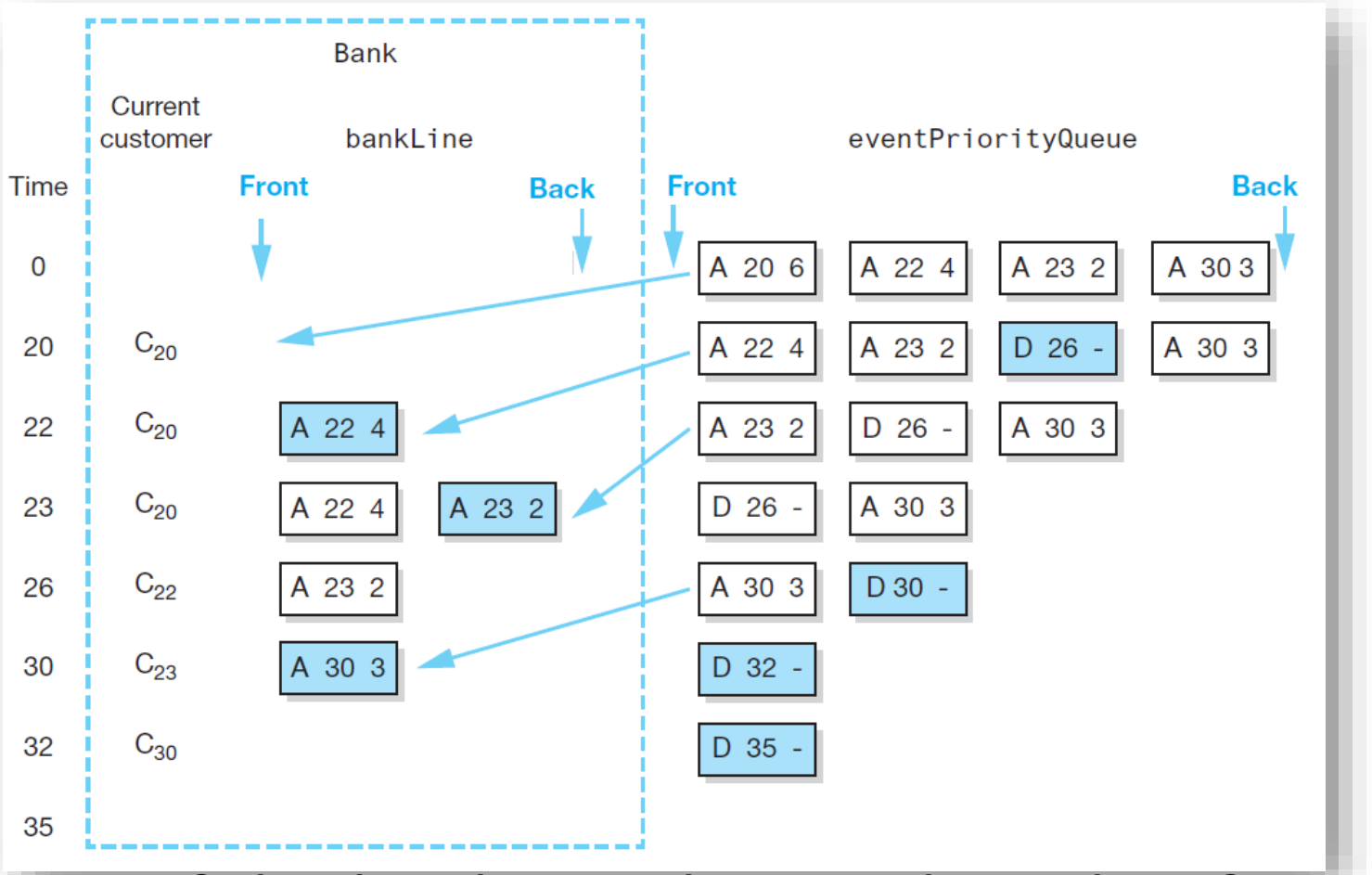
- A typical instance of (a) an arrival event; (b) a departure event

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



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

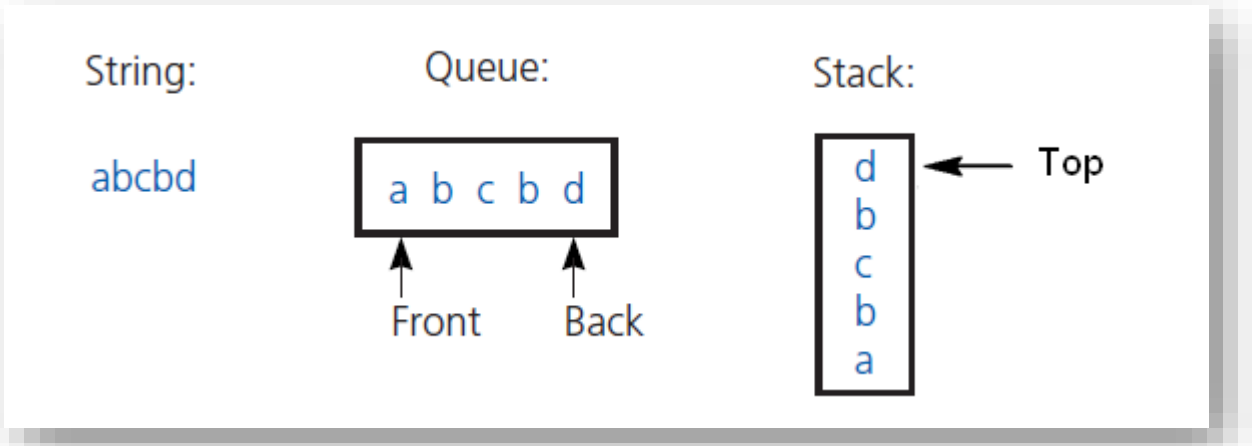
Position-Oriented and Value-Oriented ADTs

- 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

Applications Recognizing a Palindrome



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

Simple Applications of the ADT Queue

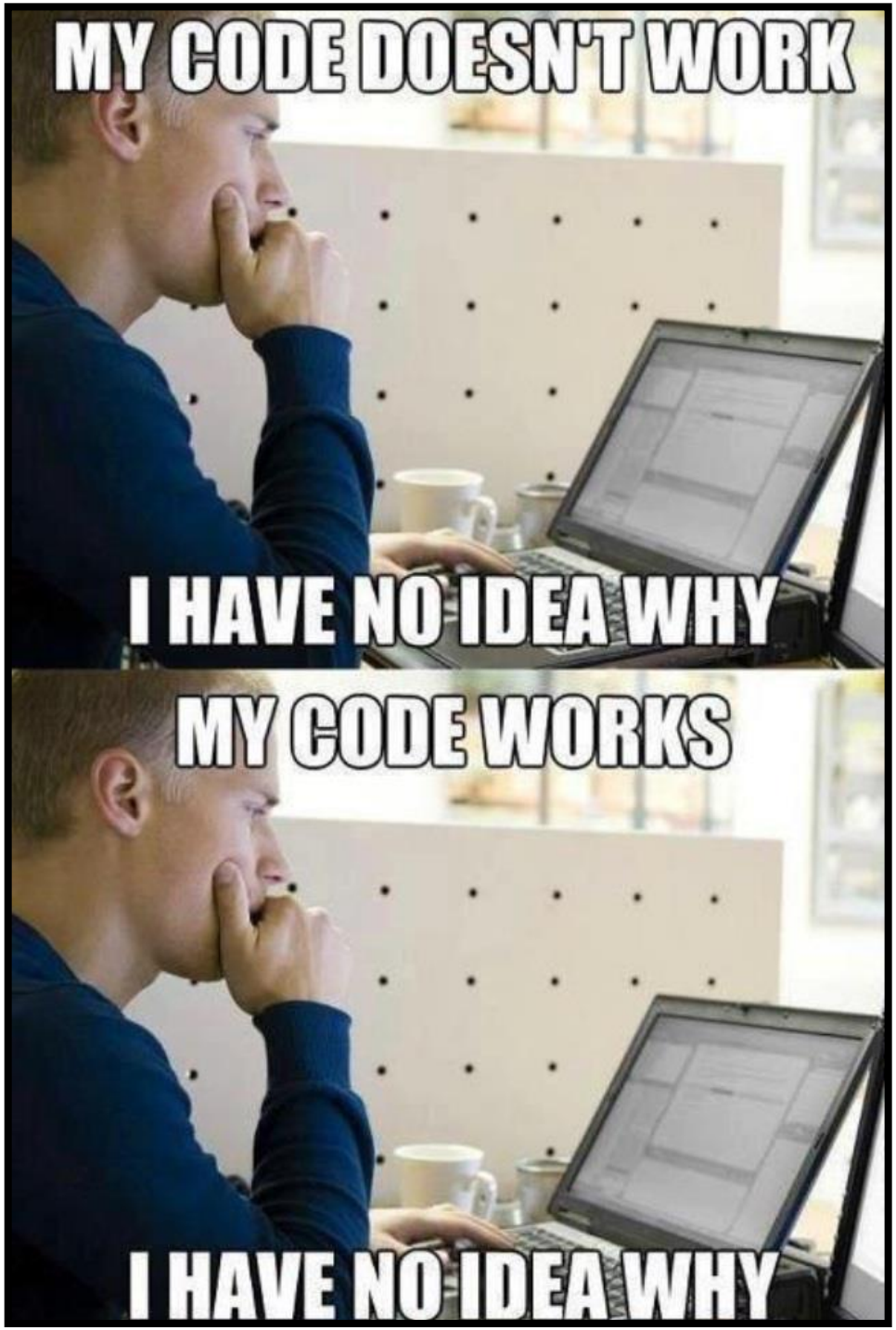
- Recognizing palindromes

```
// 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
```


MY CODE DOESN'T WORK
I HAVE NO IDEA WHY
MY CODE WORKS
I HAVE NO IDEA WHY