



Chapter 8

Queues

CS102 Sections 51 and 52

Marc Smith and Jim Ten Eyck

Spring 2008

The Abstract Data Type Queue

- A queue
 - New items enter at the back, or rear, of the queue
 - Items leave from the front of the queue
 - First-in, first-out (FIFO) property
 - The first item inserted into a queue is the first item to leave

The Abstract Data Type Queue

- ADT queue operations
 - Create an empty queue
 - Determine whether a queue is empty
 - Add a new item to the queue
 - Remove from the queue the item that was added earliest
 - Remove all the items from the queue
 - Retrieve from the queue the item that was added earliest

The Abstract Data Type Queue

- Queues
 - Are appropriate for many real-world situations
 - Example: A line to buy a movie ticket
 - Have applications in computer science
 - Example: A request to print a document
- A simulation
 - A study to see how to reduce the wait involved in an application

The Abstract Data Type Queue

- Pseudocode for the ADT queue operations


```
createQueue()
// Creates an empty queue.

isEmpty()
// Determines whether a queue is empty

enqueue(newItem) throws QueueException
// Adds newItem at the back of a queue. Throws
// QueueException if the operation is not
// successful
```

The Abstract Data Type Queue

- Pseudocode for the ADT queue operations (Continued)


```
dequeue() throws QueueException
// Retrieves and removes the front of a queue.
// Throws QueueException if the operation is
// not successful.

dequeueAll()
// Removes all items from a queue

peek() throws QueueException
// Retrieves the front of a queue. Throws
// QueueException if the retrieval is not
// successful
```

The Abstract Data Type Queue

Operation

```

queue.createQueue()
queue.enqueue(5)
queue.enqueue(2)
queue.enqueue(7)
queueFront = queue.peek()
queueFront = queue.dequeue()
queueFront = queue.dequeue()
    
```

Queue after operation

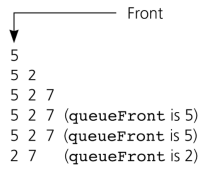


Figure 8-2
Some queue operations

Simple Applications of the ADT Queue: Reading a String of Characters

- A queue can retain characters in the order in which they are typed


```

queue.createQueue()
while (not end of line) {
    Read a new character ch
    queue.enqueue(ch)
}
            
```
- Once the characters are in a queue, the system can process them as necessary

Recognizing Palindromes

- A palindrome
 - A string of characters that reads the same from left to right as its does from right to left
- To recognize a palindrome, a queue can be used in conjunction with a stack
 - A stack can be used to reverse the order of occurrences
 - A queue can be used to preserve the order of occurrences

Recognizing Palindromes

- A nonrecursive recognition algorithm for palindromes
 - As you traverse the character string from left to right, insert each character into both a queue and a stack
 - Compare the characters at the front of the queue and the top of the stack

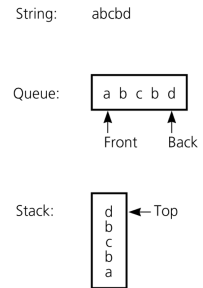


Figure 8-3
The results of inserting a string into both a queue and a stack

Implementations of the ADT Queue

- A queue can have either
 - An array-based implementation
 - A reference-based implementation

A Reference-Based Implementation

- Possible implementations of a queue
 - A linear linked list with two external references
 - A reference to the front
 - A reference to the back

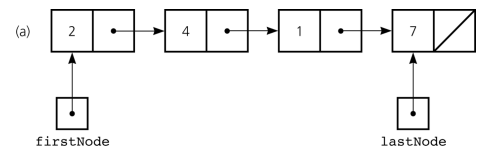


Figure 8-4a
A reference-based implementation of a queue: a) a linear linked list with two external references

A Reference-Based Implementation

- Possible implementations of a queue (Continued)
 - A circular linked list with one external reference
 - A reference to the back

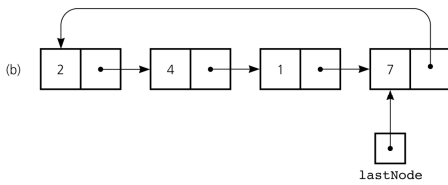


Figure 8-4b
A reference-based implementation of a queue: b) a circular linear linked list with one external reference

A Reference-Based Implementation

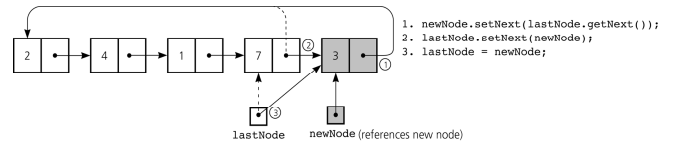


Figure 8-5
Inserting an item into a nonempty queue

A Reference-Based Implementation

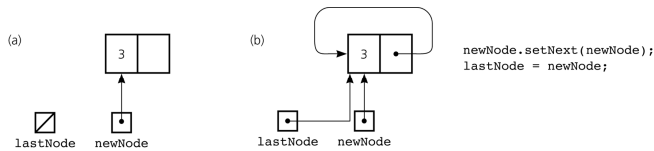


Figure 8-6
Inserting an item into an empty queue: a) before insertion; b) after insertion

A Reference-Based Implementation

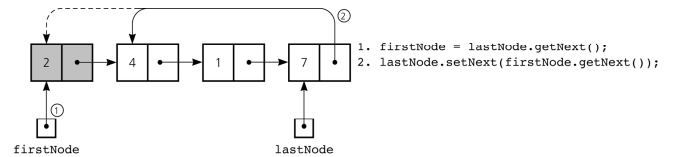


Figure 8-7
Deleting an item from a queue of more than one item

An Array-Based Implementation

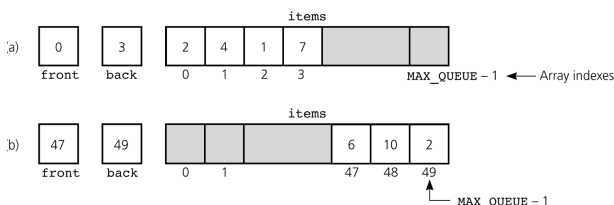


Figure 8-8
a) A naive array-based implementation of a queue; b) rightward drift can cause the queue to appear full

An Array-Based Implementation

- A circular array eliminates the problem of rightward drift

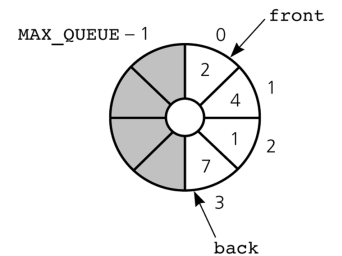


Figure 8-9
A circular implementation of a queue

An Array-Based Implementation

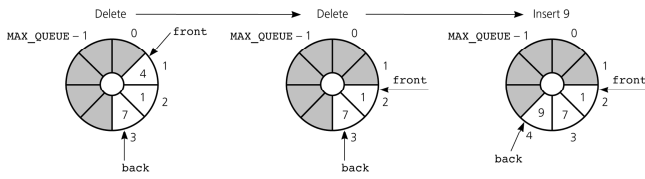


Figure 8-10

The effect of some operations of the queue in Figure 8-8

An Array-Based Implementation

- A problem with the circular array implementation
 - front and back cannot be used to distinguish between queue-full and queue-empty conditions

An Array-Based Implementation

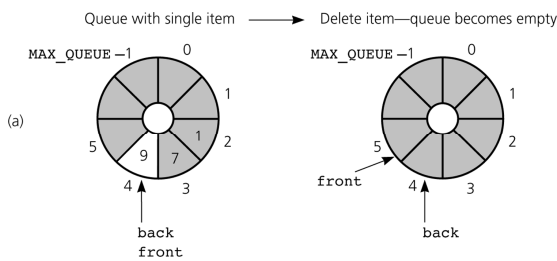


Figure 8-11a

a) **front** passes **back** when the queue becomes empty

An Array-Based Implementation

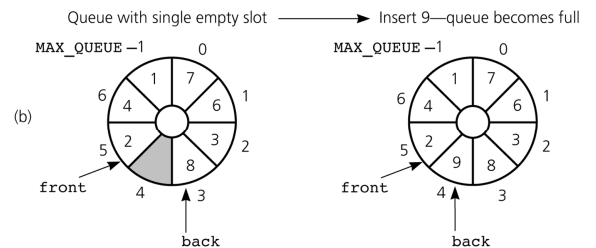


Figure 8-11b

b) **back** catches up to **front** when the queue becomes full

An Array-Based Implementation

- To detect queue-full and queue-empty conditions
 - Keep a count of the queue items
- To initialize the queue, set
 - front to 0
 - back to MAX_QUEUE - 1
 - count to 0

An Array-Based Implementation

- Inserting into a queue


```
back = (back+1) % MAX_QUEUE;
items[back] = newItem;
++count;
```
- Deleting from a queue


```
front = (front+1) % MAX_QUEUE;
--count;
```

An Array-Based Implementation

- Variations of the array-based implementation
 - Use a flag `full` to distinguish between the full and empty conditions
 - Declare `MAX_QUEUE + 1` locations for the array items, but use only `MAX_QUEUE` of them for queue items

An Array-Based Implementation

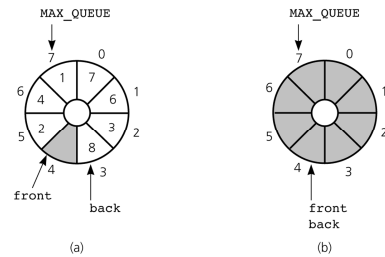


Figure 8-12
A more efficient circular implementation: a) a full queue; b) an empty queue

An Implementation That Uses the ADT List

- If the item in position 1 of a list `list` represents the front of the queue, the following implementations can be used
 - `dequeue()`
`list.remove(1)`
 - `peek()`
`list.get(1)`

An Implementation That Uses the ADT List

- If the item at the end of the list represents the back of the queue, the following implementations can be used
 - `enqueue(newItem)`
`list.add(list.size()+1, newItem)`

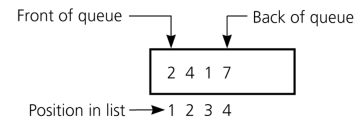


Figure 8-13
An implementation that uses the ADT list

The Java Collections Framework Interface `Queue`

- JCF has a queue interface called `Queue`
- Derived from interface `Collection`
- Adds methods:
 - `element`: retrieves, but does not remove head
 - `offer`: inserts element into queue
 - `peek`: retrieves, but does not remove head
 - `poll`: retrieves and removes head
 - `remove`: retrieves and removes head

Comparing Implementations

- All of the implementations of the ADT queue mentioned are ultimately either
 - Array based
 - Reference based
- Fixed size versus dynamic size
 - A statically allocated array
 - Prevents the `enqueue` operation from adding an item to the queue if the array is full
 - A resizable array or a reference-based implementation
 - Does not impose this restriction on the `enqueue` operation

Comparing Implementations

- Reference-based implementations
 - A linked list implementation
 - More efficient
 - The ADT list implementation
 - Simpler to write

© 2006 Pearson Addison-Wesley. All rights reserved

8-31

A Summary of Position-Oriented ADTs

- Position-oriented ADTs
 - List
 - Stack
 - Queue
- Stacks and queues
 - Only the end positions can be accessed
- Lists
 - All positions can be accessed

© 2006 Pearson Addison-Wesley. All rights reserved

8-32

A Summary of Position-Oriented ADTs

- Stacks and queues are very similar
 - Operations of stacks and queues can be paired off as
 - `createStack` and `createQueue`
 - `Stack isEmpty` and `queue isEmpty`
 - `push` and `enqueue`
 - `pop` and `dequeue`
 - `Stack peek` and `queue peek`

© 2006 Pearson Addison-Wesley. All rights reserved

8-33

A Summary of Position-Oriented ADTs

- ADT list operations generalize stack and queue operations
 - `length`
 - `add`
 - `remove`
 - `get`

© 2006 Pearson Addison-Wesley. All rights reserved

8-34

Application: Simulation

- Simulation
 - A technique for modeling the behavior of both natural and human-made systems
 - Goal
 - Generate statistics that summarize the performance of an existing system
 - Predict the performance of a proposed system
 - Example
 - A simulation of the behavior of a bank

© 2006 Pearson Addison-Wesley. All rights reserved

8-35

Application: Simulation



Figure 8-14a and 8-14b
A blank line at at time a) 0; b) 12

© 2006 Pearson Addison-Wesley. All rights reserved

8-36

Application: Simulation

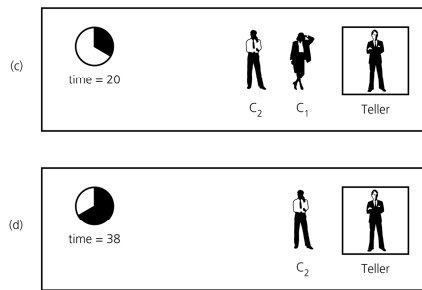


Figure 8-14c and 8-14d
A blank line at at time c) 20; d) 38

© 2006 Pearson Addison-Wesley. All rights reserved

8-37

Application: Simulation

- An event-driven simulation
 - Simulated time is advanced to the time of the next event
 - Events are generated by a mathematical model that is based on statistics and probability
- A time-driven simulation
 - Simulated time is advanced by a single time unit
 - The time of an event, such as an arrival or departure, is determined randomly and compared with a simulated clock

© 2006 Pearson Addison-Wesley. All rights reserved

8-38

Application: Simulation

- The bank simulation is concerned with
 - Arrival events
 - Indicate the arrival at the bank of a new customer
 - External events: the input file specifies the times at which the arrival events occur
 - Departure events
 - Indicate the departure from the bank of a customer who has completed a transaction
 - Internal events: the simulation determines the times at which the departure events occur

© 2006 Pearson Addison-Wesley. All rights reserved

8-39

Application: Simulation

- An event list is needed to implement an event-driven simulation
 - An event list
 - Keeps track of arrival and departure events that will occur but have not occurred yet
 - Contains at most one arrival event and one departure event

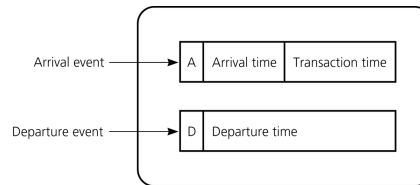


Figure 8-15
A typical instance of the event list

© 2006 Pearson Addison-Wesley. All rights reserved

8-40

Summary

- The definition of the queue operations gives the ADT queue first-in, first-out (FIFO) behavior
- A reference-based implementation of a queue uses either
 - A circular linked list
 - A linear linked list with a head reference and a tail reference
- An array-based implementation of a queue is prone to rightward drift
 - A circular array eliminates the problem of rightward drift

© 2006 Pearson Addison-Wesley. All rights reserved

8-41

Summary

- To distinguish between the queue-full and queue-empty conditions in a queue implementation that uses a circular array, you can
 - Count the number of items in the queue
 - Use a full flag
 - Leave one array location empty
- Models of real-world systems often use queues
 - The event-driven simulation in this chapter uses a queue to model a line of customers in a bank

© 2006 Pearson Addison-Wesley. All rights reserved

8-42

Summary

- Simulations
 - Central to a simulation is the notion of simulated time
 - In a time-driven simulation
 - Simulated time is advanced by a single time unit
 - In an event-driven simulation
 - Simulated time is advanced to the time of the next event
 - To implement an event-driven simulation, you maintain an event list that contains events that have not yet occurred