DATA STRUCTURES & ALGORITHMS

COMP H3025

Lecture 5: Queues

Lecturer: Stephen Sheridan

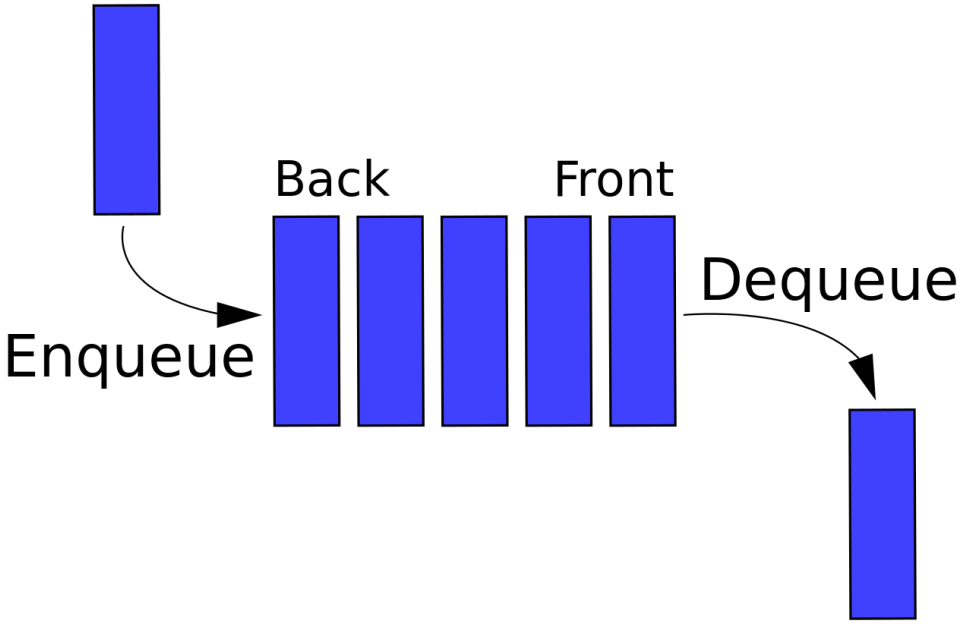
QUEUES

- A **Queue** is like a line of people. The first person to join the queue is the first person served and is therefore the first person to leave the queue.

  - Items enter at the back and leave at the front. This give the queue a **first in first out** order **(FIFO)**.
  - Operations on a queue occur only at its two ends.

- There are lots of examples of queue usage in computer science. For example, queues are used in operating systems (Event, I/O, CPU, Print), web servers, game engines and anytime a service can only deal with one item at a time.

QUEUES



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.

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

dequeueAll()
// remove all items from the front of a queue

peek() throws QueueException
// retrieves the front of a queue, that is,
// retrieves the item that was added earliest.
// Throws QueueException if the operation is not successful.
// The queue is unchanged
```

QUEUE OPERATIONS IN ACTION

Operation	Queue Status
queue.createQueue()	
queue.enqueue(5)	5
queue.enqueue(2)	5 2
queue.enqueue(7)	5 2 7
queueFront = queue.peek() queueFront = 5	5 2 7
queueFront = queue.dequeue() queueFront = 5	2 7
queueFront = queue.dequeue() queueFront = 2	7

EXAMPLE PROBLEM: RECOGNISING PALINDROMES

- A **palindrome** is a string of characters that read the same way from left to right and vice versa.
  - **NAVAN, RADAR, ABCBA**
- In our **Stack** lecture we learned that stacks reverse the order of items.
- A **Queue** can be used to preserve the order of items.
- Therefore, a **queue** and a **stack** can be used to detect palindromes.

PALINDROME SOLUTION

- Traverse the character string from left to right *inserting each character into a queue and a stack*.
  - The **first** character in the string is at the **front of the queue**
  - The **last** character in the string is at the **top of the stack**
- Therefore:
  - characters removed via a **dequeue** operation from the *queue* will occur in the order in which they appear in the string **(FIFO)**.
  - characters removed via a **pop** operation from the *stack* will occur in reverse order **(LIFO)**.

PALINDROME SOLUTION

- So we can easily compare the characters at the front of the queue and the characters at the top of the stack.

- If they are **equal**, we can delete them

- This process is repeated until the ADT's become empty

  ➤ **PALINDROME**

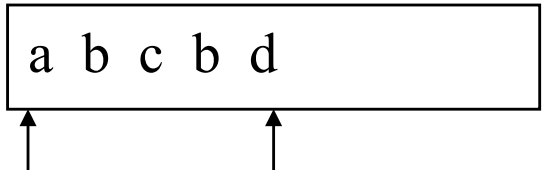
- If the two characters are not equal then

  ➤ **NOT PALINDROME**

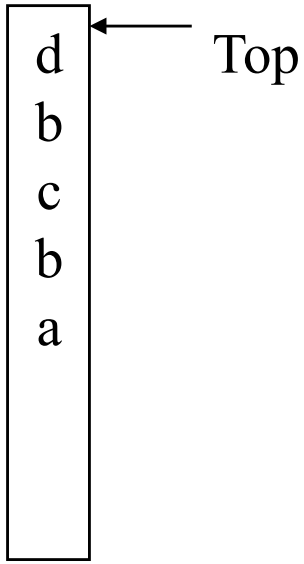
PALINDROME SOLUTION

String: **abcbd**

Queue:



Stack:



The results of inserting a string into both a queue and a stack

PALINDROME SOLUTION

```
isPal(str){ // determines whether str is a palindrome

//create an empty queue and an empty stack
queue.createQueue()
stack.createStack()

//insert each character of the string into both the stack and the queue
length = the length of str
for (k = 1 through length
{
    nextChar = kth character of str
    queue.enqueue(nextChar)
    stack.push(nextChar)
} //end for

//compare the queue characters with the stack characters
charactersAreEqual = true
while (queue is not empty and charactersAreEqual is true)
{
    queueFront = queue.dequeue()
    stackTop = stack.pop()
    if (queueFront not equal to stackTop)
    {
        charactersAreEqual = false
    } //end if
} //end while
return charactersAreEqual
}
```

IMPLEMENTATION OF ADT QUEUE

- Just like the *Stack*, the **Queue ADT** can be implemented in a number of ways

- For the purposes of this course we will focus on two implementations:

  - as an **array**

  - as a referenced based **linked list**

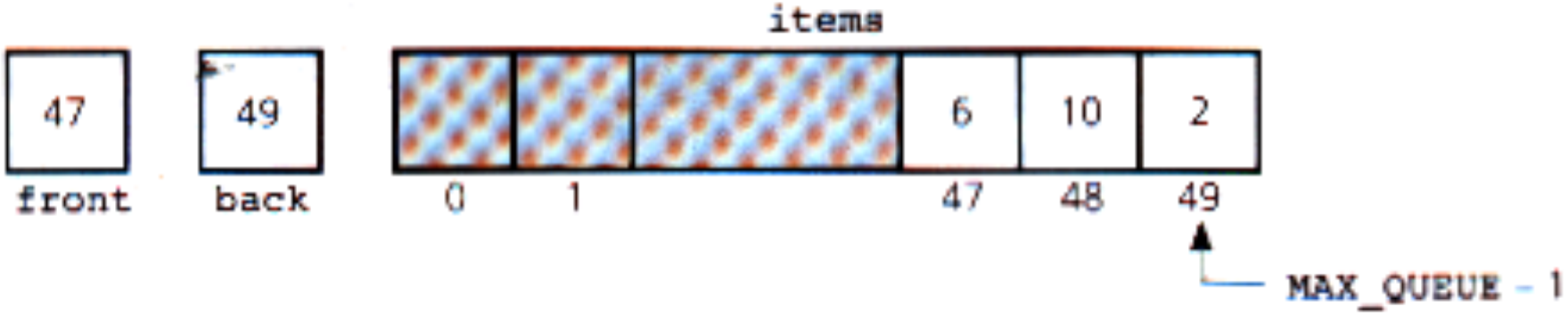
ARRAY-BASED IMPLEMENTATION

- An array can be used to implement a queue if a **fixed-size** does not present a problem.
  - Initially **front** is 0 and **back** is -1.
  - To *insert* a new item in the queue, you increment **back** and place the item in **items[back]**.
  - To *delete* an item, you simply increment **front.**
  - The queue is *empty* whenever **back** is less than **front**.
  - The queue is *full* when back equals MAX_QUEUE -1

ARRAY-BASED IMPLEMENTATION

- The problem with this strategy is *rightward drift*. This can occur after a sequence of *additions* and *removals*.

- The items in the queue will drift towards the end of the array and **back** could equal **MAX_QUEUE -1** even when the queue contains only a few items.



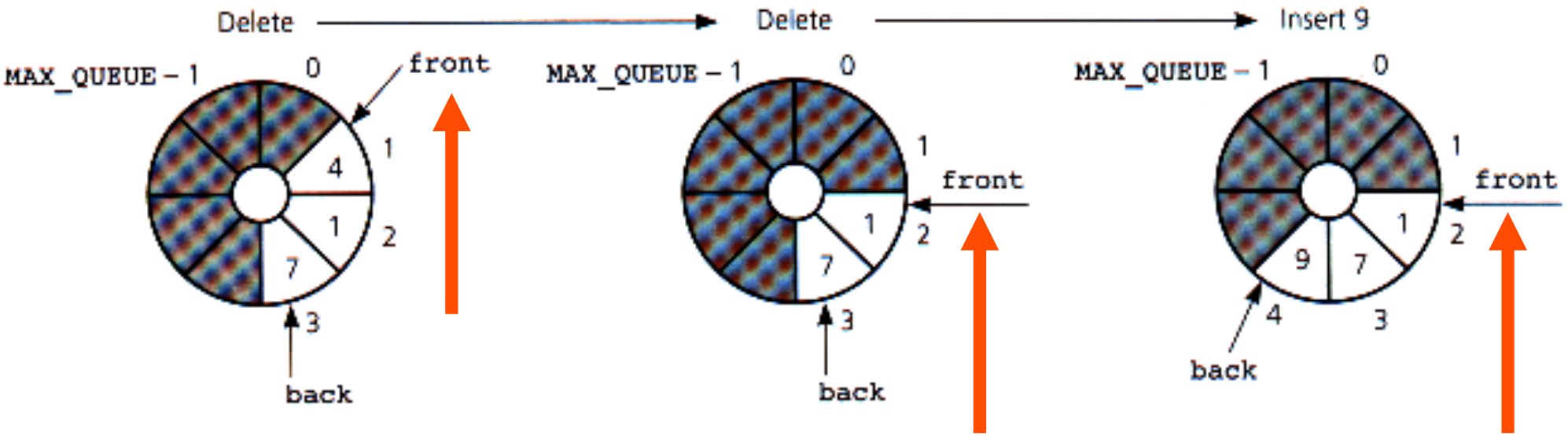
ARRAY-BASED IMPLEMENTATION

- A possible solution is to shift array elements to the left
  - either after each insertion **or**
  - whenever back equals MAX_QUEUE-1
- This will guarantee that the queue can always contain up to MAX_QUEUE elements.
- Shifting is not very satisfactory, as it would dominate the **performance cost** of the implementation.

CIRCULAR ARRAY-BASED IMPLEMENTATION

- A more elegant solution is possible if we view the array as a **circular array**.

- We can advance the queue indexes **front** (to delete an item) and **back** (to insert an item) by moving them **clockwise** around the array.



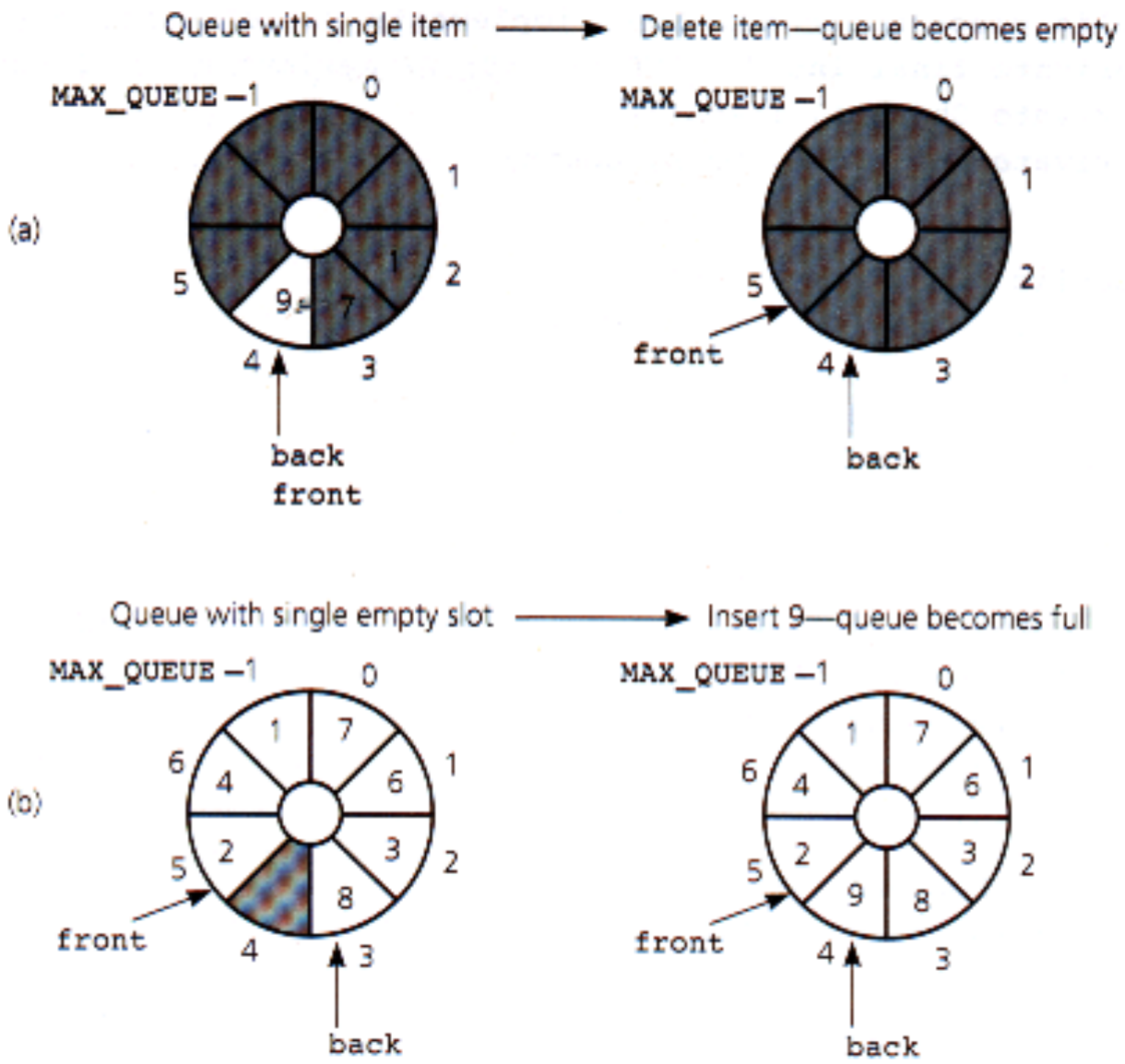
CIRCULAR ARRAY-BASED IMPLEMENTATION

- The only difficulty with this involves detecting the **queue-empty** and **queue-full** states.

- We could say that the queue is empty when **front** is one slot ahead of **back**. But this could also indicate a **full** queue.

- To distinguish between the two situations we need to *maintain* a **count** of the items in the queue.

CIRCULAR ARRAY-BASED IMPLEMENTATION

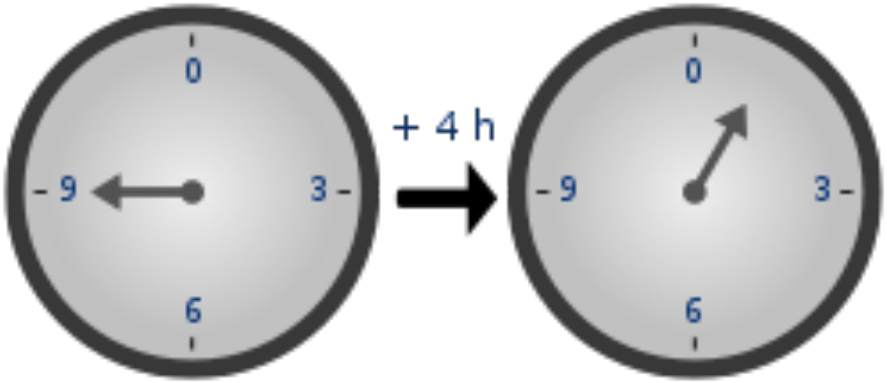


(a) ***front*** passes ***back*** when the queue becomes empty;
(b) ***back*** catches up to ***front*** when the queue becomes full

CIRCULAR ARRAY-BASED IMPLEMENTATION

- We can enforce the wraparound effect in a circular queue by using **modulo** arithmetic.

- In mathematics, *modular arithmetic* is a system of arithmetic for integers, where numbers "wrap around" upon reaching a certain value—the modulus.



$$9h + 4h \neq 13h$$

$$(9h + 4h) \% 12 = 1h$$

CIRCULAR ARRAY-BASED IMPLEMENTATION

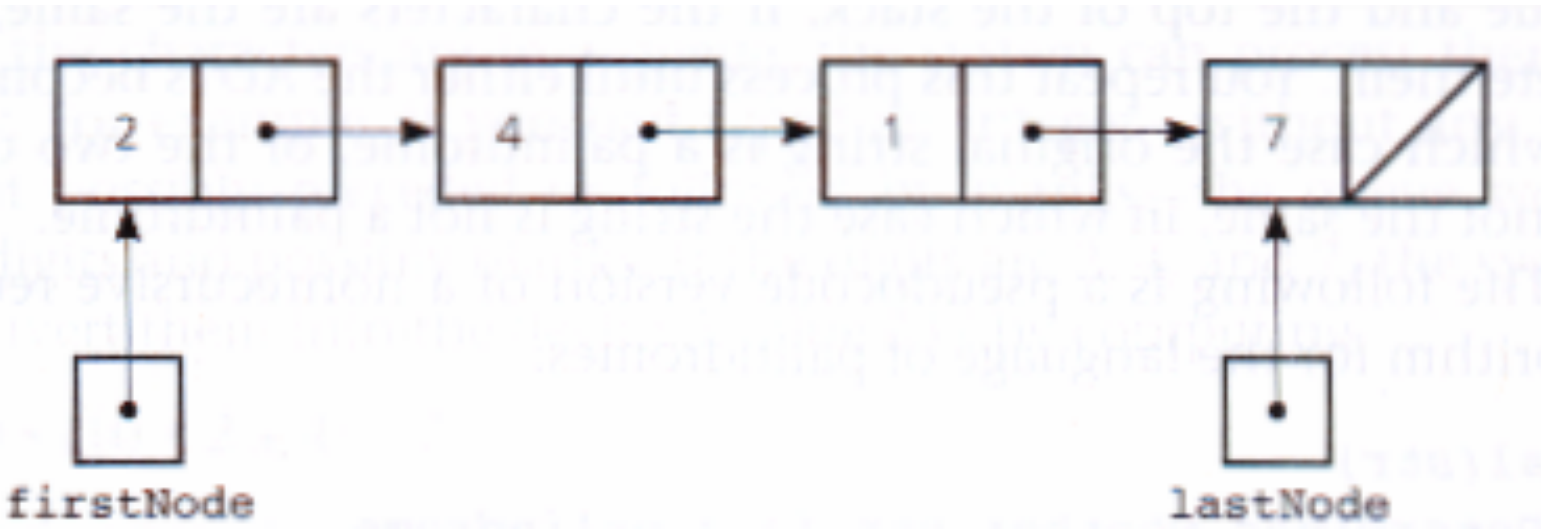
- Therefore, the code for queue *insertion* and *deletion* using modulo arithmetic might look as follows:

```
//Inserting into a queue
back = (back + 1) % MAX_QUEUE;
items[back] = newItem;
++count;

//Deleting from the front of a queue
front = (front + 1) % MAX_QUEUE;
--count;
```

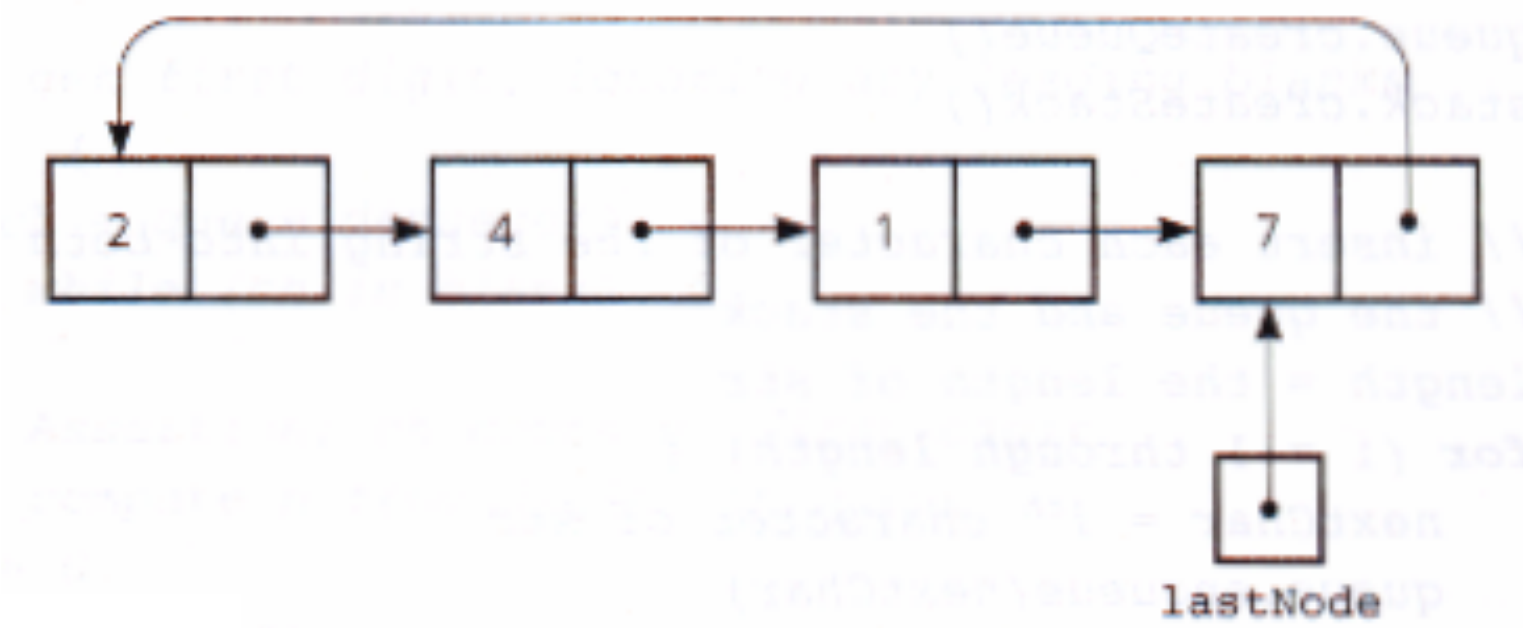
REFERENCE BASED IMPLEMENTATION

- A reference-based implementation could use a linear linked list with two external references, one to the **front** and one to the **back**.



REFERENCE BASED IMPLEMENTATION

- We can, however, get by with a single reference to the **back** if we use a *circular list*.

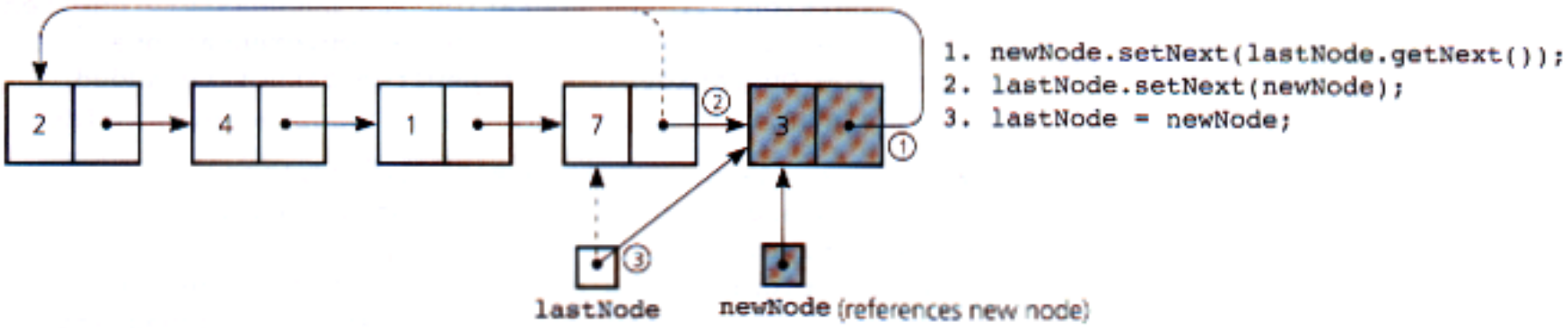


REFERENCE BASED IMPLEMENTATION

- When a circular linked list represents a queue…
  - the node at the **back** of the queue references the node at the **front**
  - **lastNode** references the item at the **back** of the queue
  - **lastNode.getNext()** references the item at the **front** of the queue
- This arrangement is useful as **insertion** at the **back** and **deletion** at the **front** are straightforward.

REFERENCE BASED IMPLEMENTATION

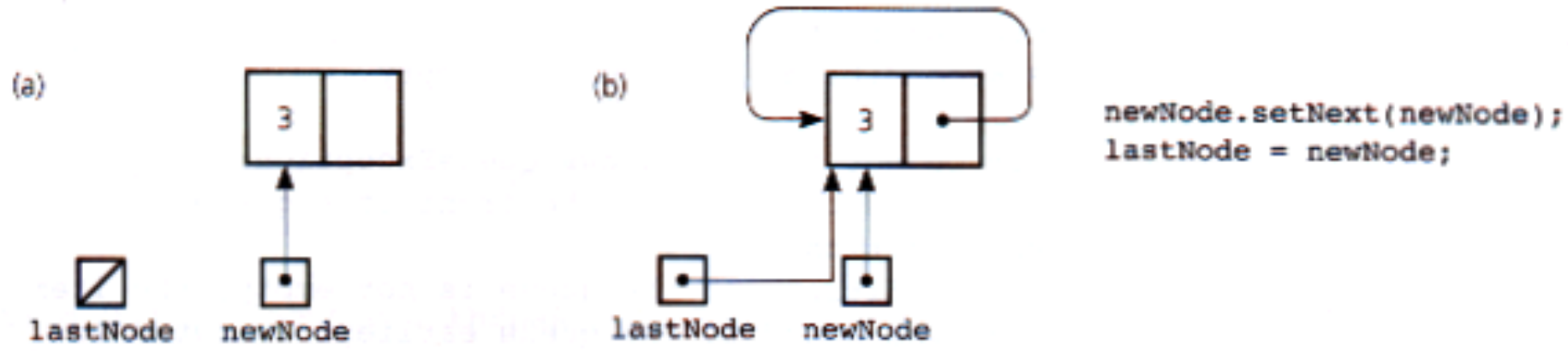
- Inserting a new node (newNode) at the back of the queue requires three reference changes:
  - set the **next** reference in newNode
  - set the **next** reference in the **back** node
  - set the external reference **lastNode**



Inserting an item into a nonempty queue

REFERENCE BASED IMPLEMENTATION

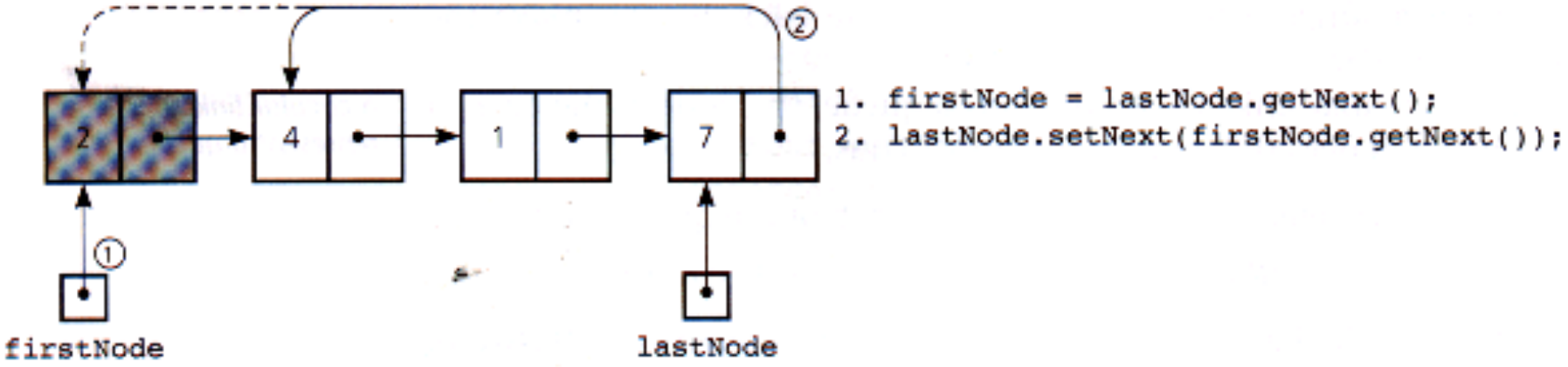
- The addition of an item into an empty queue is a special case:



Inserting an item into an empty queue: (a) before insertion; (b) after insertion

REFERENCE BASED IMPLEMENTATION

- Deletion from the **front** of the queue is more straightforward then insertion at the back.
- Notice that we need to change only one reference within the queue.
- Deletion from a queue with only one item is a special case.



Deleting an item from a queue of more than one item

TODO - WEEK 5

- Implement a Queue ADT with a circular array.
- Create a simple program to test the circular array based Queue ADT
- Implement a Queue ADT with a linear linked list using **firstNode** and **lastNode** references.
- Using the referenced based Queue and Stack verify that all strings in the palindromes.txt file (MOODLE) are palindromes.