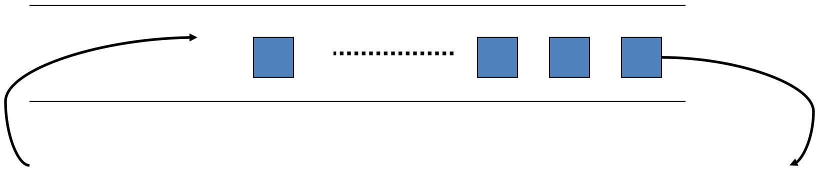
Queues

The class notes are a compilation and edition from many sources. The instructor does not claim intellectual property or ownership of the lecture notes.

Queue: Introduction

- Another subclass of lists that permits deletions to be performed at one end of the list and insertions at the other.
- Information is processed in the same order as it was received
  - i.e. First In First Out (FIFO) or First Come First Serve(FCFS)
- Examples: a checkout line at supermarket cash register, timesharing computer system, line of cars waiting to proceed in some fixed direction at an intersection of streets….

A Graphical Model of a Queue

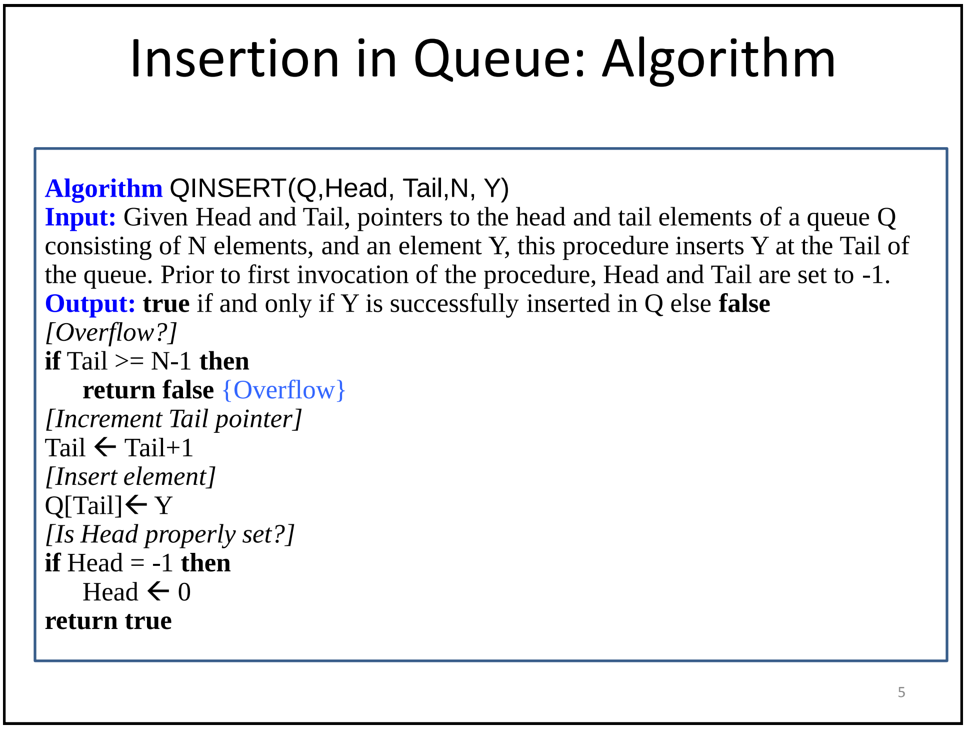


Rear/Tail:
All new items
are added on
this end

Front/Head:
All items are
deleted from
this end

Operations on Queues

- **Insert(**item**)**: (also called enqueue)
  - It adds a new item to the tail of the queue
- **Delete( )**: (also called dequeue)
  - It deletes the head item of the queue, and returns to the caller. If the queue is already empty, this operation returns NULL
- **getHead( ):**
  - Returns the value in the head element of the queue
- **getTail( ):**
  - Returns the value in the tail element of the queue
- **isEmpty( )**
  - Returns **true** if the queue has no items
- **size**( )
  - Returns the number of items in the queue



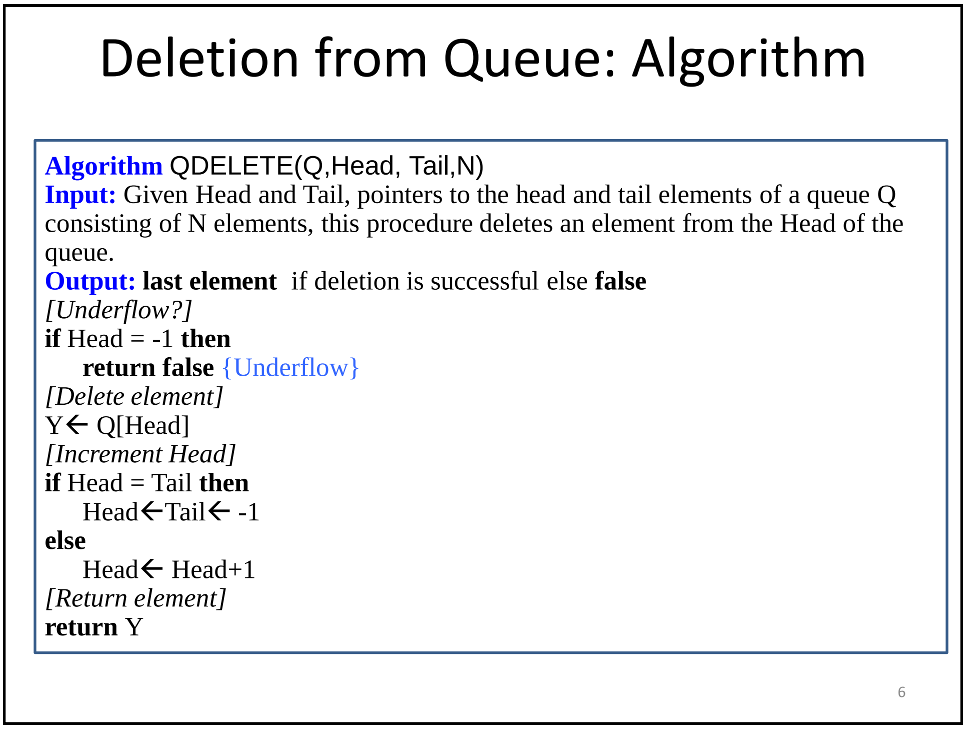
Insertion in Queue: Algorithm

Algorithm QINSERT(Q,Head, Tail,N, Y)

Input: Given Head and Tail, pointers to the head and tail elements of a queue Q consisting of N elements, and an element Y, this procedure inserts Y at the Tail of the queue. Prior to first invocation of the procedure, Head and Tail are set to -1.

Output: **true** if and only if Y is successfully inserted in Q else **false**

```
[Overflow?]
if Tail >= N-1 then
    return false {Overflow}
[Increment Tail pointer]
Tail ← Tail+1
[Insert element]
Q[Tail]← Y
[Is Head properly set?]
if Head = -1 then
    Head ← 0
return true
```



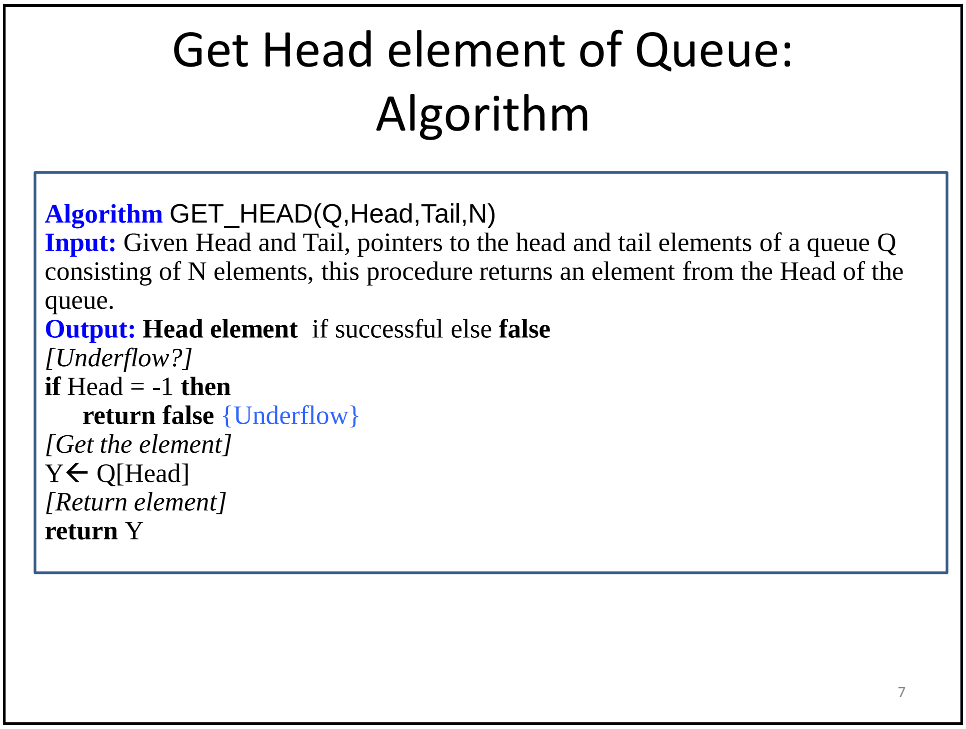
Deletion from Queue: Algorithm

Algorithm QDELETE(Q,Head, Tail,N)

Input: Given Head and Tail, pointers to the head and tail elements of a queue Q consisting of N elements, this procedure deletes an element from the Head of the queue.

Output: **last element** if deletion is successful else **false**

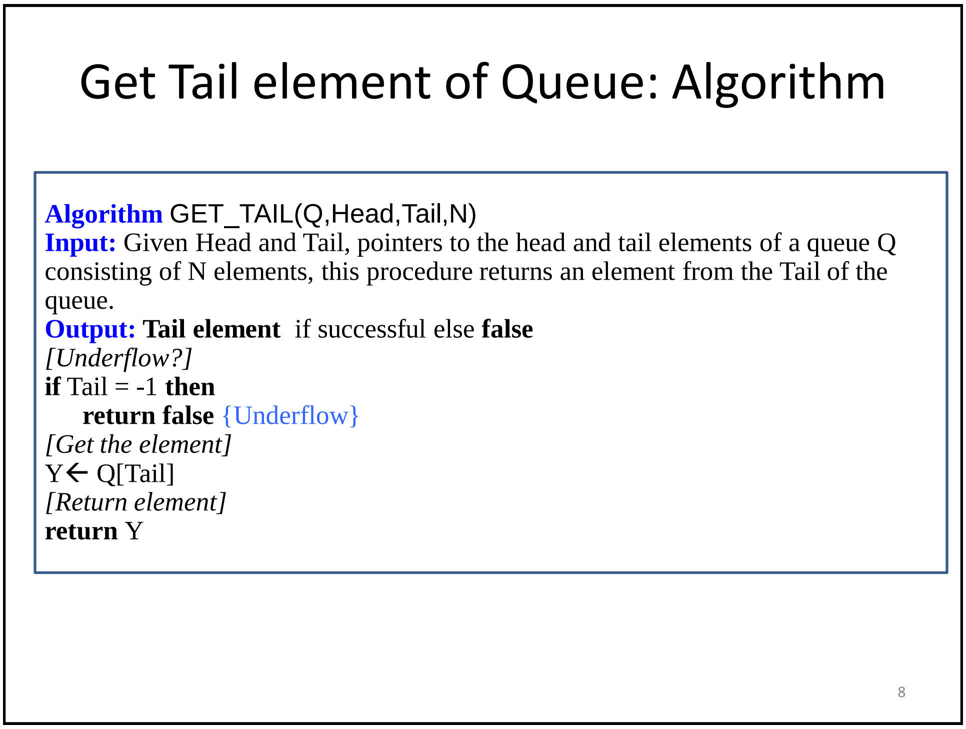
```
[Underflow?]
if Head = -1 then
    return false {Underflow}
[Delete element]
Y← Q[Head]
[Increment Head]
if Head = Tail then
    Head←Tail← -1
else
    Head← Head+1
[Return element]
return Y
```



Get Head element of Queue: Algorithm

Algorithm GET_HEAD(Q,Head,Tail,N)
Input: Given Head and Tail, pointers to the head and tail elements of a queue Q consisting of N elements, this procedure returns an element from the Head of the queue.
Output: **Head element** if successful else **false**

```
[Underflow?]
if Head = -1 then
    return false {Underflow}
[Get the element]
Y← Q[Head]
[Return element]
return Y
```



Get Tail element of Queue: Algorithm

Algorithm GET_TAIL(Q,Head,Tail,N)
Input: Given Head and Tail, pointers to the head and tail elements of a queue Q consisting of N elements, this procedure returns an element from the Tail of the queue.
Output: **Tail element** if successful else **false**

```
[Underflow?]
if Tail = -1 then
    return false {Underflow}
[Get the element]
Y← Q[Tail]
[Return element]
return Y
```

IsEmpty(): Algorithm

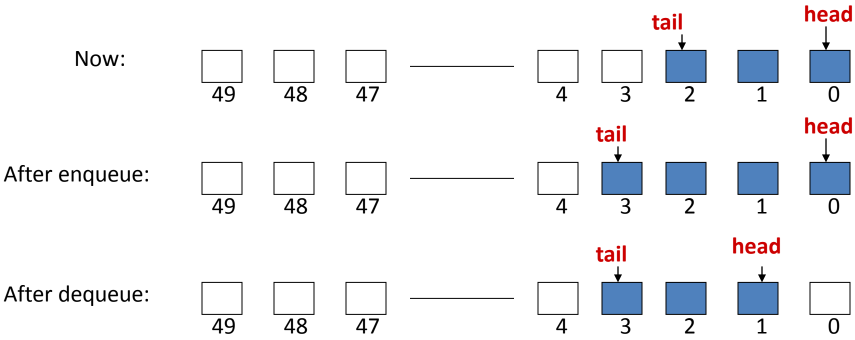
```
Algorithm ISEMPTY(Q,Head,Tail,N)
Input: Head and Tail, pointers to the head and tail elements of a queue Q
consisting of N elements.
Output: true  if Q is empty else false
[check if Q is empty or not]
if Head = Tail =  -1 then
    return true
else
    return false
```

Size(): Algorithm

```
Algorithm SIZE(Q,Head,Tail,N)
Input: Head and Tail, pointers to the head and tail elements of a queue Q
consisting of N elements.
Output: number of elements  in Q
[check if Q is empty or not]
if Head = Tail =  -1 then
    return 0
else
    return (Tail – Head +1)
```

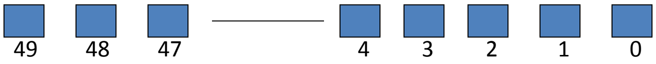
How **head** and **tail** Change

- **Tail** increases by 1 after each Insert( )
- **Head** increases by 1 after each Delete( )

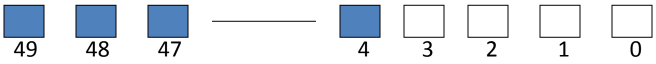


False-Overflow

- Suppose 50 calls to Insert() have been made, so now the queue array is full



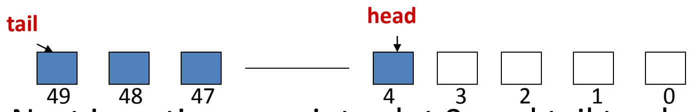
- Assume 4 calls to Delete( ) are made



- Assume a call to Insert( ) is made now. The tail part seems to have no space, but the front has 4 unused spaces; if never used, they are wasted.

Solution: A Circular Queue

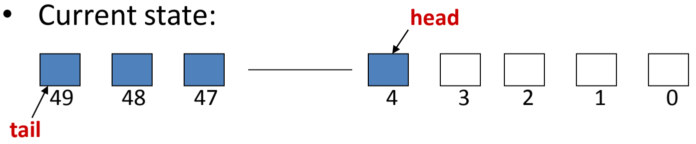
- Allow the head (and the tail) to be moving targets
- When the tail end fills up and front part of the array has empty slots, new insertions should go into the front end



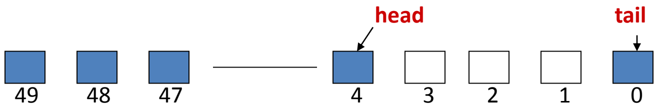
- Next insertion goes into slot 0, and tail tracks it. The insertion after that goes into a lot 1, etc.

Illustration of Circular Queue

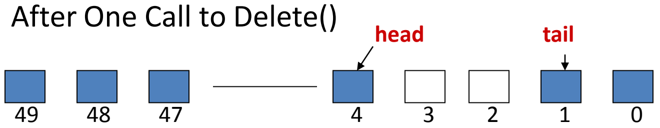
- Current state:



- After One Call to Insert()



- After One Call to Delete()



Numerics for Circular Queues

- **head** increases by (1 modulo capacity) after each Delete( ):

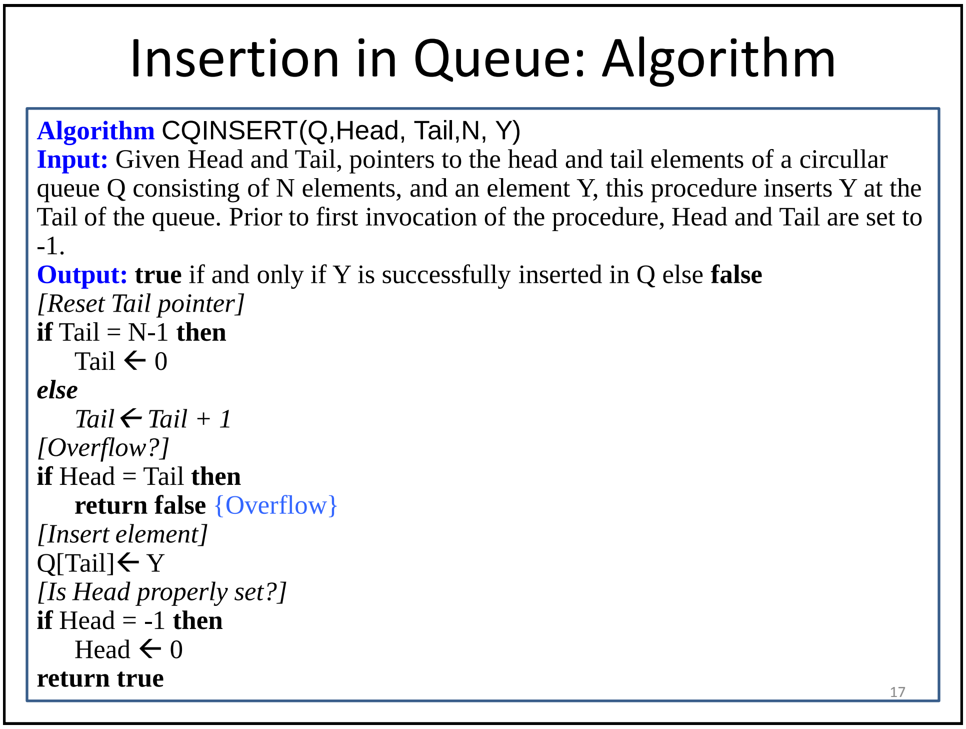
  head = (**head** +1) % capacity;

- **tail** increases by (1 modulo capacity) after each Insert( ):

  tail = (**tail** +1) % capacity;

Operations on Circular Queues

- **Insert(**item**)**:
  - It adds a new item to the tail of the circular queue
- **Delete( )**:)
  - It deletes the head item of the circular queue, and returns to the caller. If the queue is already empty, this operation returns false.
- **isEmpty( )**
  - Returns **true** if the queue has no items

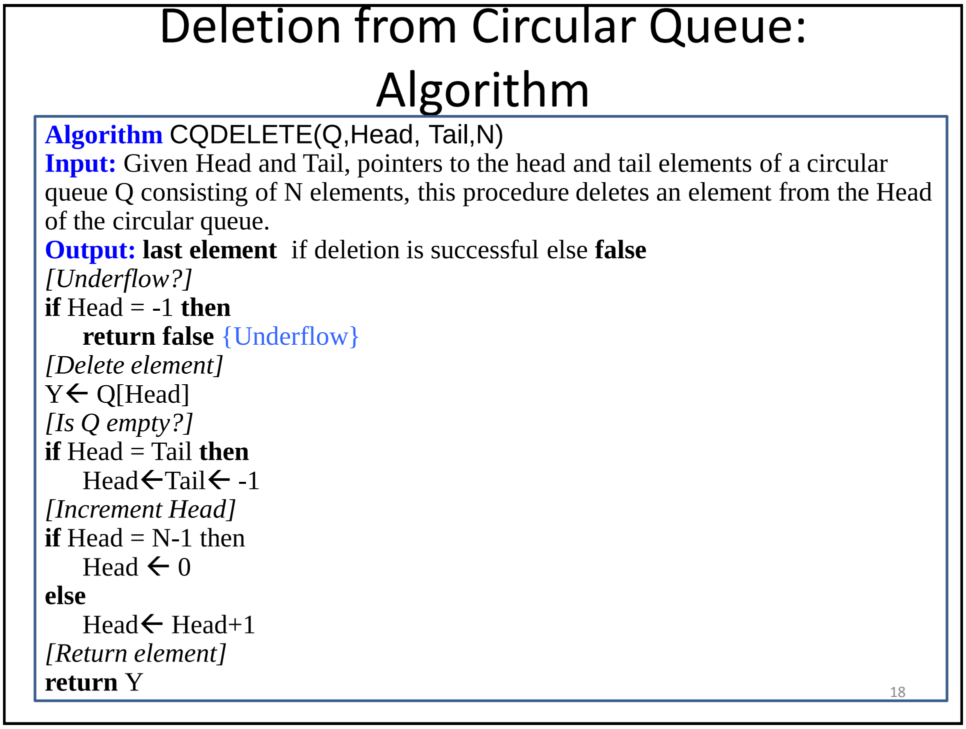


Insertion in Queue: Algorithm

Algorithm CQINSERT(Q,Head, Tail,N, Y)
Input: Given Head and Tail, pointers to the head and tail elements of a circullar queue Q consisting of N elements, and an element Y, this procedure inserts Y at the Tail of the queue. Prior to first invocation of the procedure, Head and Tail are set to -1.
Output: **true** if and only if Y is successfully inserted in Q else **false**

```
[Reset Tail pointer]
if Tail = N-1 then
    Tail ← 0
else
    Tail ← Tail + 1
[Overflow?]
if Head = Tail then
    return false {Overflow}
[Insert element]
Q[Tail] ← Y
[Is Head properly set?]
if Head = -1 then
    Head ← 0
return true
```

17

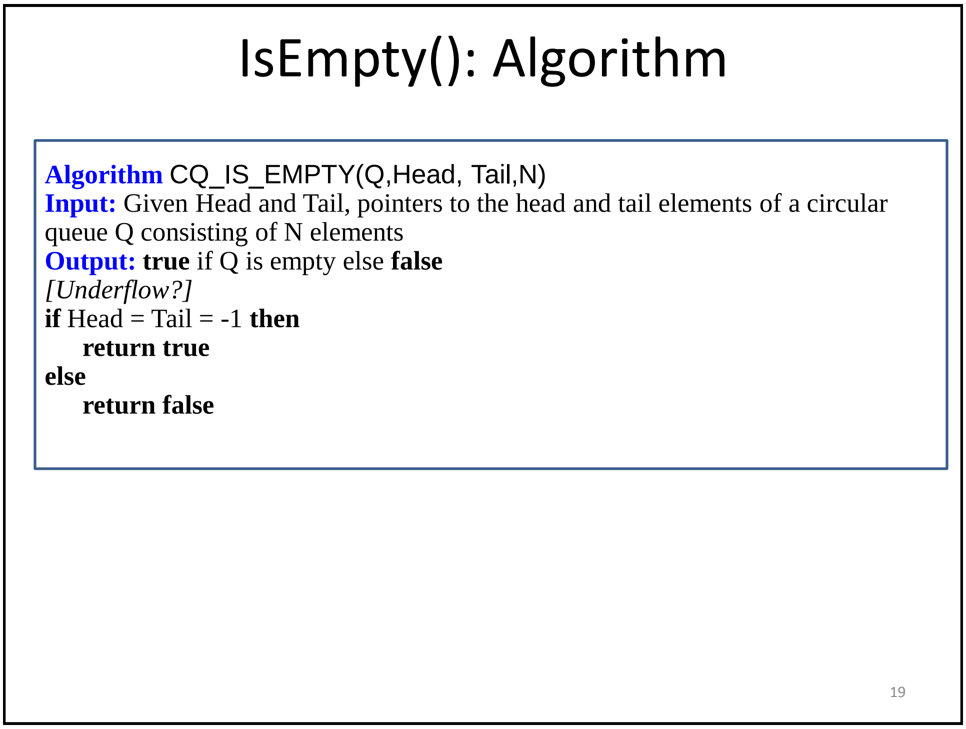


Deletion from Circular Queue: Algorithm

Algorithm CQDELETE(Q,Head, Tail,N)
Input: Given Head and Tail, pointers to the head and tail elements of a circular queue Q consisting of N elements, this procedure deletes an element from the Head of the circular queue.
Output: **last element** if deletion is successful else **false**

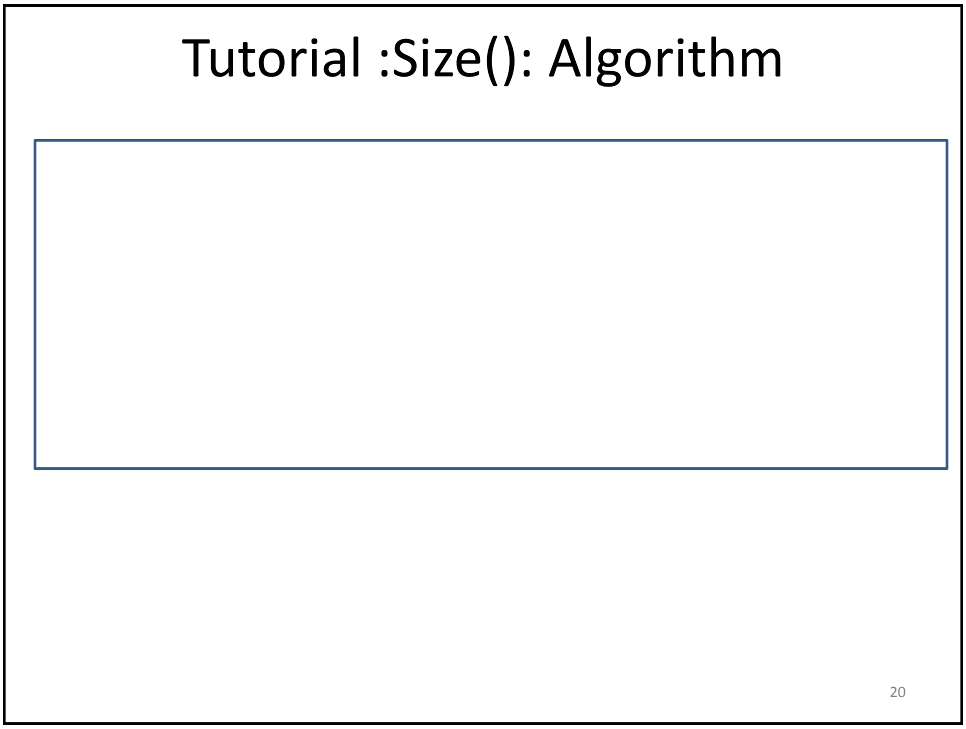
```
[Underflow?]
if Head = -1 then
    return false {Underflow}
[Delete element]
Y ← Q[Head]
[Is Q empty?]
if Head = Tail then
    Head ← Tail ← -1
[Increment Head]
if Head = N-1 then
    Head ← 0
else
    Head ← Head+1
[Return element]
return Y
```

18



IsEmpty(): Algorithm

```
Algorithm CQ_IS_EMPTY(Q,Head, Tail,N)
Input: Given Head and Tail, pointers to the head and tail elements of a circular
queue Q consisting of N elements
Output: true if Q is empty else false
[Underflow?]
if Head = Tail = -1 then
    return true
else
    return false
```



Tutorial :Size(): Algorithm

Double ended queue (Deque)

pronounced deck

- More general than a stack and a queue
- A linear list in which insertions and deletions are made to or from either end of the structure
- Two variations
  - Input-restricted deque
    - allows insertions at only one end
  - Output-restricted deque
    - allows deletions from only one end

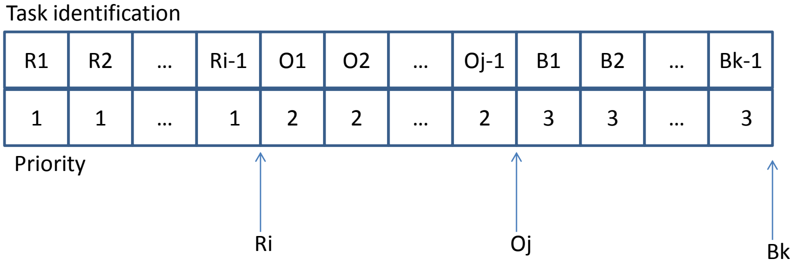
Double ended queue (Deque)

pronounced deck

- Tutorial :
  1. Formulate an algorithm for performing an insertion into an input-restricted deque
  2. Formulate an algorithm for performing a deletion from an input-restricted deque
  3. Repeat 1 & 2 for output-restricted deque

Priority Queue

- A queue in which we are able to insert or remove items from any position based on some priority



Task identification

R1	R2	…	Ri-1	O1	O2	…	Oj-1	B1	B2	…	Bk-1
1	1	…	1	2	2	…	2	3	3	…	3

Priority

Ri

Oj

Bk

Applications of Queue

Applications

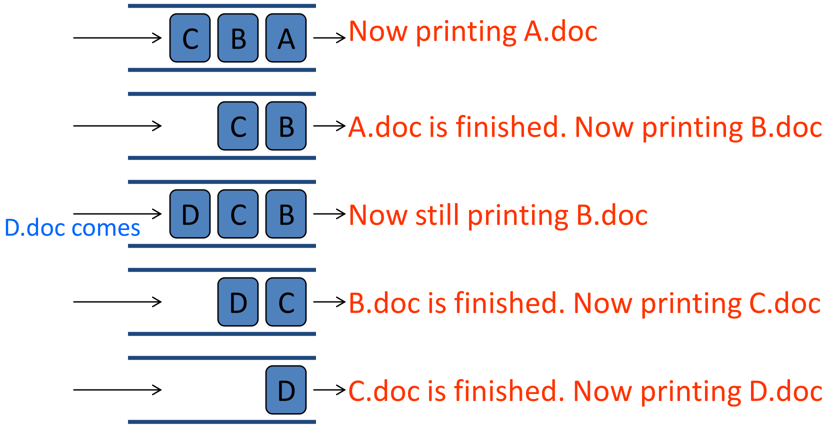
- Shared resources management (system programming):
  - Access to the processor;
  - Access to the peripherals such as disks and printers.
- Application programs:
  - Simulations

Printing Job Management

- Many users send their printing jobs to a public printer
- Printer will put them into a queue according to the arrival time and print the jobs one by one
- These printing documents are A.doc, B.doc, C.doc and D.doc

Printing Queue

- A.doc B.doc C.doc arrive to printer.



Now printing A.doc

A.doc is finished. Now printing B.doc

D.doc comes

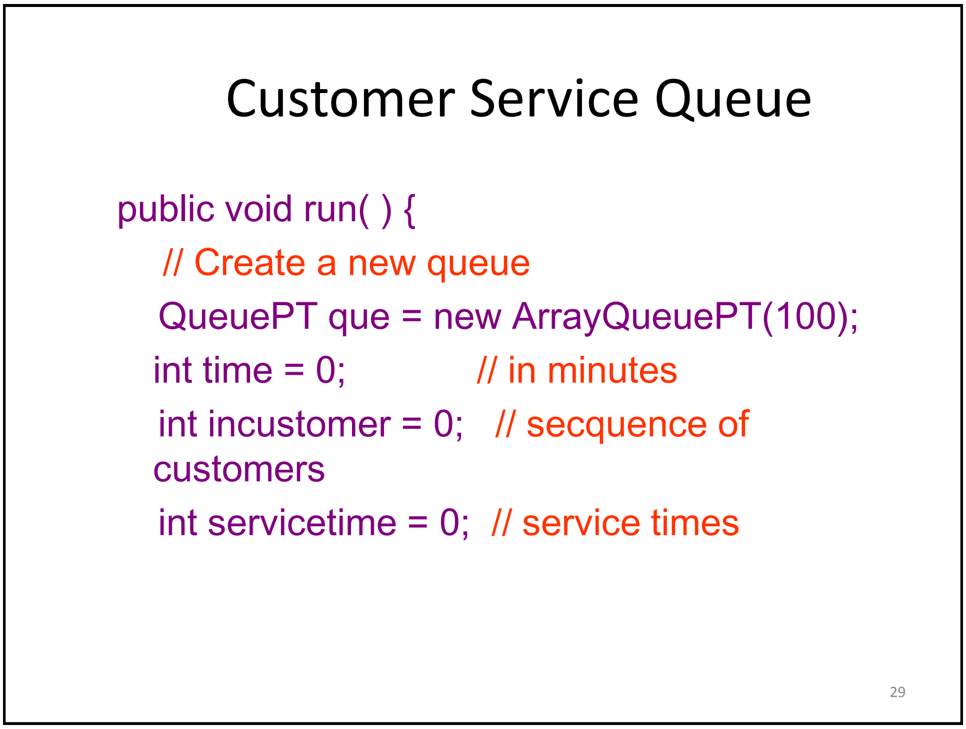
Now still printing B.doc

B.doc is finished. Now printing C.doc

C.doc is finished. Now printing D.doc

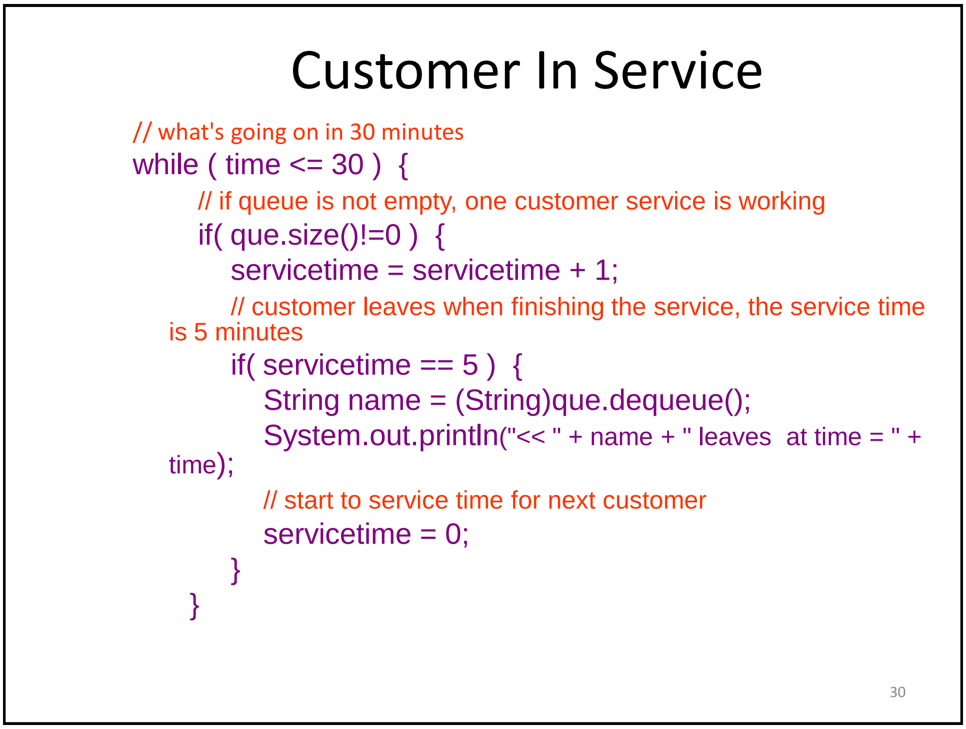
Customer Service In Royal Bank

- Suppose there is only one customer service available in Royal Bank on Saturday morning
- In every 3 minutes, a new customer arrives at the end of waiting line
- Each customer will need 5 minutes for the service
- Print out the information after the first 30 minutes
  - The time of arriving and leaving for each customer
  - How many customers are in the line?
  - Who is the current serving customer?



Customer Service Queue

```
public void run( ) {
    // Create a new queue
    QueuePT que = new ArrayQueuePT(100);
   int time = 0;             // in minutes
    int incustomer = 0;   // secquence of
   customers
    int servicetime = 0;  // service times
```



Customer In Service

```
// what's going on in 30 minutes
while ( time <= 30 )  {
        // if queue is not empty, one customer service is working
        if( que.size()!=0 )  {
            servicetime = servicetime + 1;
            // customer leaves when finishing the service, the service time
    is 5 minutes
            if( servicetime == 5 )  {
                String name = (String)que.dequeue();
                System.out.println("<< " + name + " leaves  at time = " +
    time);
                // start to service time for next customer
                servicetime = 0;
            }
       }
```

New Customer Comes

```
// every 3 minutes, there is a new customer coming.
if( time%3==0 )
{
    incustomer = incustomer + 1;
    String name = "CUSTOMER " + incustomer;
    que.enqueue( name );
    System.out.println(">> " + name + " arrives at time
  = " + time);
  }
time = time + 1;
}
```

Print Status After 30 Minutes

```
// print the status after 30 minutes
System.out.println("\n=======================" );
if( que.size()!=0 )
{
    System.out.println("There are " + que.size() + " customers in
 the line" );
    System.out.println("The current serving customer is " +
 que.peek() );
}
else
{
    System.out.println("There are no customers in the line" );
}
}
```

Priority Queue --- Air Travel

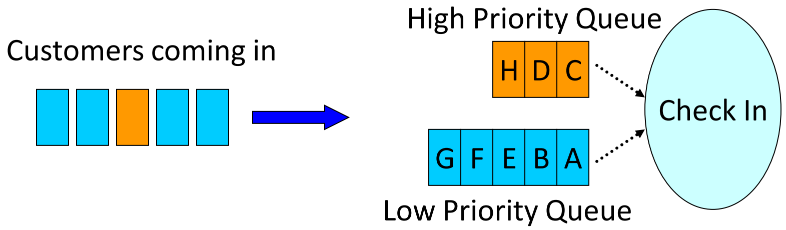
- Only one check-in service in Air Canada at airport
- Two waiting lines for passengers
  - one is First class service
  - the other is Economy class service
- Passengers in the first-class waiting line have higher priority to check in than those in the economy-class waiting line.

Priority Queue

- Two queues
  - one is high priority queue
  - the other is low priority queue
- Service rules:
  - First serve the people in high priority queue
  - If no passengers are in high priority queue, serve the passengers in low priority queue

Two Queues

- High Priority Queue,🟧 will come in *hpQue*
- Low Priority Queue,🟦 will come in *lpQue*



Pseudocode For Arrival

Passengers Arrival:

```
if( new Passenger comes )  {
       if( is First Class)
           hpQue.enqueue( new Passenger );
       else
           lpQue.enqueue( new Passenger );
  }
```

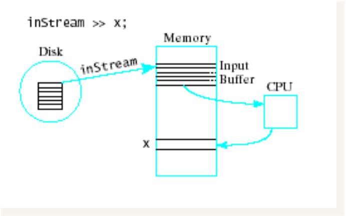
Pseudocode For Service

Check-In Service:

```
if( hpQue is not empty ) {
        serve the passenger from high priority queue,
        hpQue.dequeue();
 }
 else {
        serve the passenger from low priority queue,
        lpQue.dequeue();
  }
```

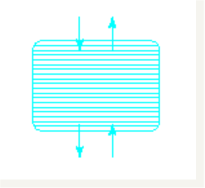
Application of Queues: Buffers and Scheduling

- Important use of queues is I/O scheduling
  - Use buffers in memory to improve program execution
  - Buffer arranged in FIFO structure



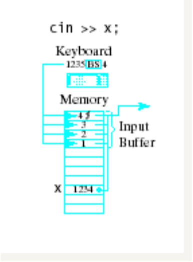
Application of Queues: Buffers and Scheduling

- Also times when insertions, deletions must be made from both ends
  - Consider a scrolling window on the screen
- This requires a double ended queue



Application of Queues: Buffers and Scheduling

- Consider a keyboard buffer
  - Acts as a queue
  - But elements may be removed from the back of the queue with backspace key



```
cin >> x;
```