University of Kelaniya
Sri Lanka

COSC 21063

Queue

Lecturer : Ms.D.M.L.Maheshika Dissanayake

Structure of Lesson

- Queue
  - Definition
  - Operations
  - Implementation
    - Contiguous implementation
    - Linked Implementation

Learning Outcome:

By the end of this lesson you should be able to:

- Understand the concept of queue
- Create the data structure queue and define the necessary operations
- Implement the queue using the contiguous and linked representation
- Solve simple problems using the queue concept

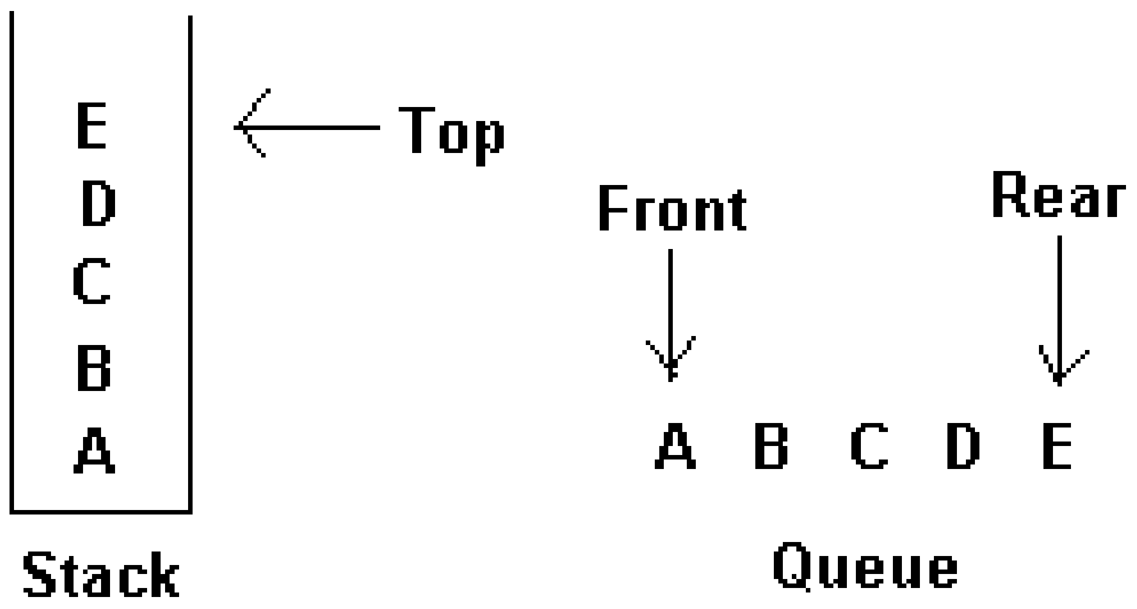
Queues

- Based on the everyday notion of a queue, such as a line of people waiting to buy a ticket or waiting to be served in a bank.
- Definition
  - A queue is a chronologically ordered list; values are added at one end (the rear) and removed from the other end (the front).
- The active part of a queue is the front and rear.

Queues Contd…

- The only difference between a stack and a queue is that,
    - in a stack, elements are added and removed at the same end (the top),
    - whereas in a queue, values are added at one end (the rear) and removed from the other end (the front).

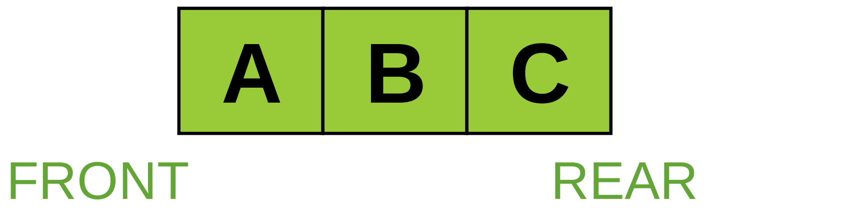
Queues Contd…



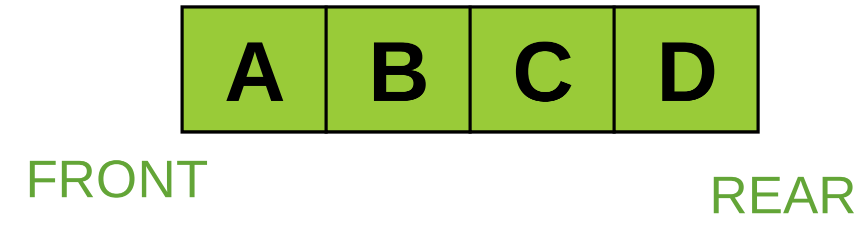
Queues Contd…

- When we add an item to a queue, we say we **append or insert** it onto the queue
- when we remove an item, we say that we **serve or remove** it from the queue.
- So the first item pushed onto the queue is always the first that will be removed from the queue. This property is called **first in**, **first out** (FIFO).

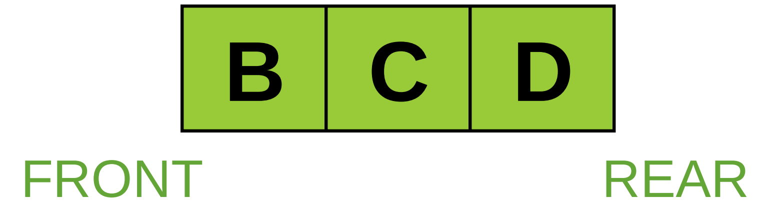
Queue Functions



INSERT (D)



REMOVE ( )



Exercise: add elements A B C D E into a queue.

Queue Specification

- Queue Specification
  - without any mention of how queues will be implemented.
- A queue is either empty or it consists of:
  - a front element
  - a rear element
  - other elements
- The elements in a queue may be of any type, but all the elements in a given queue must be the same type.

Operations on the Queue

CreateQueue

- Inputs: none
- Outputs: Q (a queue)
- Preconditions: none
- Postconditions: Q is defined (i.e. created) and is empty (i.e. initialised to be empty)

Operations on the Queue Contd..

IsQueueEmpty

- Inputs: Q (a queue)
- Outputs: IsQueueEmpty true or false
- Preconditions: Q has been created
- Postconditions: IsQueueEmpty is true iff Q is empty.

Operations on the Queue Contd…

Serve/Remove

- Inputs: Q (a Queue)
- Outputs: Q' (i.e. Q is changed) and X the element removed i.e. this is the first entry in Q
- Preconditions: Q has been created and is not empty
- Postconditions: The first entry of the Q has been removed and returned as the value of X

- Note Q’ is the updated Q.

Operations on the Queue Contd…

Write the specification for the following operations

- DestroyQueue
- IsQueueFull
- Append/Insert

Operations on the Queue Contd…

DestroyQueue

- Inputs: Q (a queue)
- Outputs: Q' (i.e. Q changed)
- Preconditions: Q has been created
- Postconditions: Q' is undefined. All resources (e.g. memory) allocated to Q have been released. No queue operation can be performed on Q'.

- Note Q’ is the updated Q.

Operations on the Queue Contd…

IsQueueFull

- Inputs: Q (a queue)
- Outputs: IsQueueFull true or false
- Preconditions: Q has been created
- Postconditions: IsQueueFull is true iff Q is full.

Operations on the Queue Contd…

Append

- Inputs: Q (a queue) and X (a value)
- Outputs: Q' (i.e. Q changed)
- Preconditions: Q has been created and not full and X is of appropriate type for an element of Q
- Postconditions: Q' has X as its last entry.

▪ Note Q’ is the updated Q

Operations on the Queue Contd…

- The operations specified above are *core* operations - any other operation on queues can be defined in terms of these ones

Implementation of queues

- Contiguous implementation
  - Linear Queue and Circular Queue
- Linked implementation

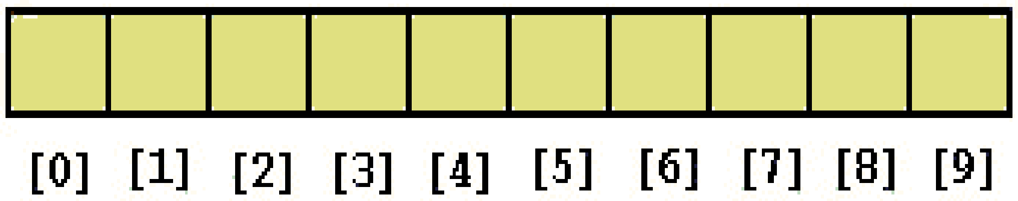
Contiguous Implementation

- Define a queue in java.
- Identify the information needed.
  - An array to hold the elements of the queue.
  - and two integer variables front and rear, the pointers of the first element and the last element in the array.
  - A variable count to indicate the number of elements in the queue (This is optional)

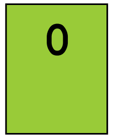
```
public class Queue {
  int front;
  int rear;
  int maxSize;
  int[] queue;
  int count;

 Queue(int size) {
       maxSize = size;
       queue = new int[maxSize];
       front = 0;
       rear = -1;
       count = 0;
   } }
```

Initial Picture – Empty Queue

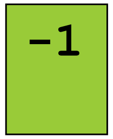
Name: queueArray
Type: Array



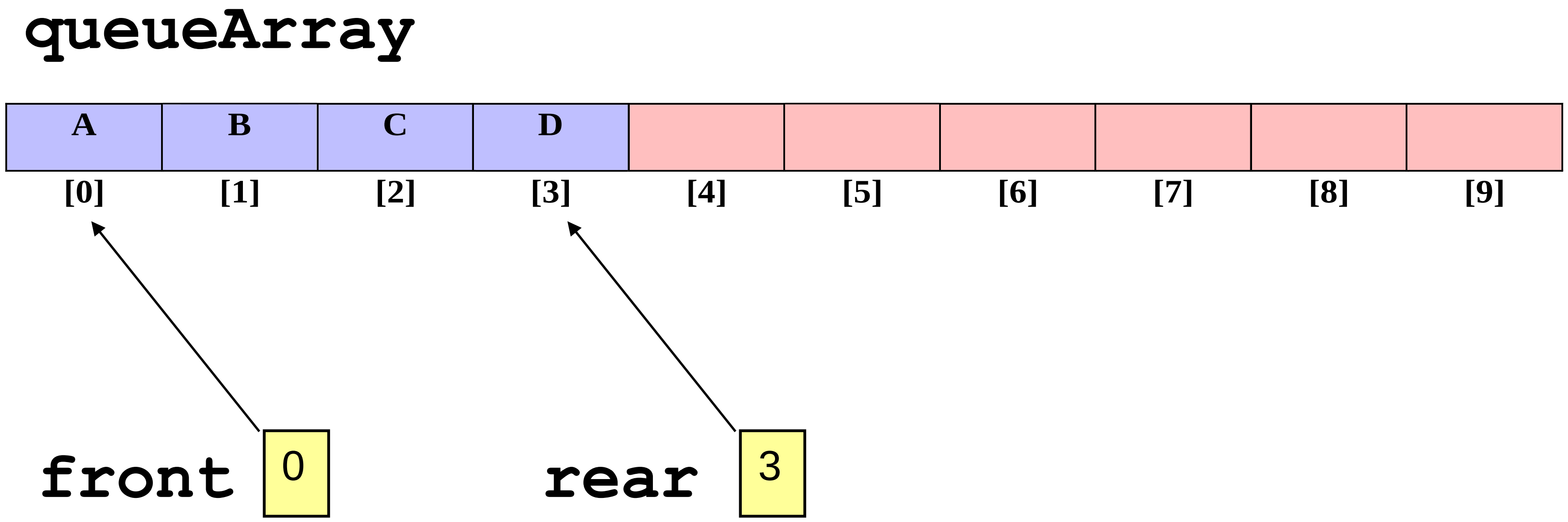
Name: front
Type: Integer



Name: rear
Type: Integer



An Array-Based Queue



Implement the operation : IsQueueEmpty

```
boolean IsQueueEmpty() {
   if (rear<front )
     return true;
   else
     return false;
 }
```

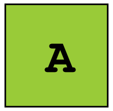
Implement the operation : IsQueueFull

```
boolean IsQueueFull() {
  if (rear == maxSize - 1) {
    return true;
  }
    return false;
 }
```

Adding Elements – INSERT

Name: x

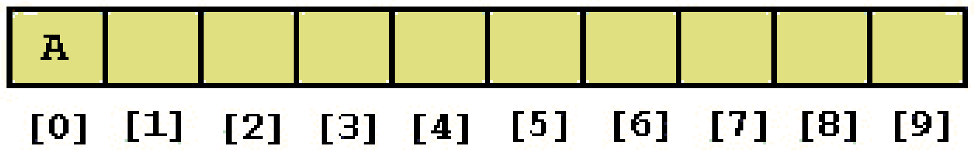
Type: Character



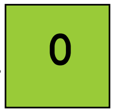
Name: insert(x)

Type: Function

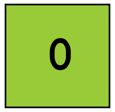
Step 1: queueArray[0] = x



Step 2: front

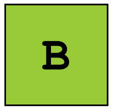


rear



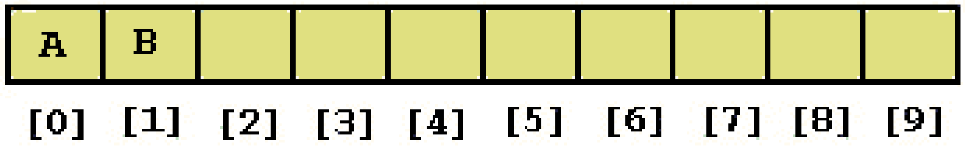
Adding Elements – INSERT

```
Name:  x
Type:  Character
```

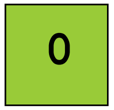


```
Name:  insert(x)
Type:  Function
```

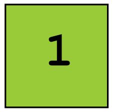
Step 1: queueArray[1] = x



Step 2: front



rear

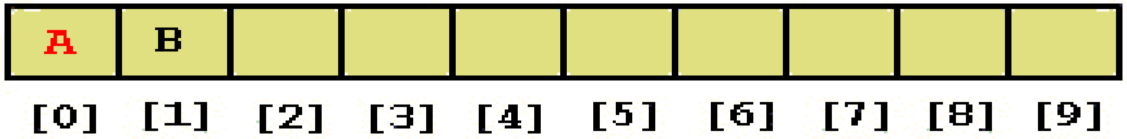


Append

```
void Append(int item)
{
     if (IsQueueFull())
          {
             System.out.printf("\nQueue is full\n");
          }

     else {
              queue[++rear] = item;
              count++;
          }
 }
```

Removing Elements – REMOVE

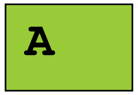
x = remove()



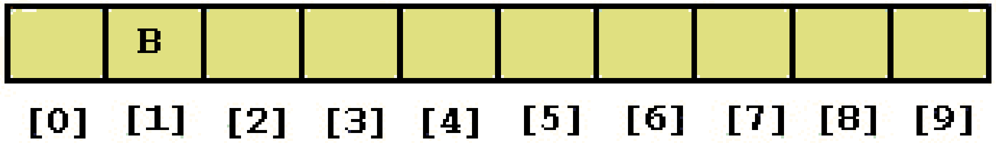
Name: remove()

Type: Function

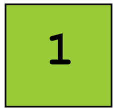
Step 1: x



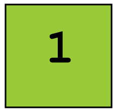
Step 2: stackArray[0] = NULL



Step 3: front



rear

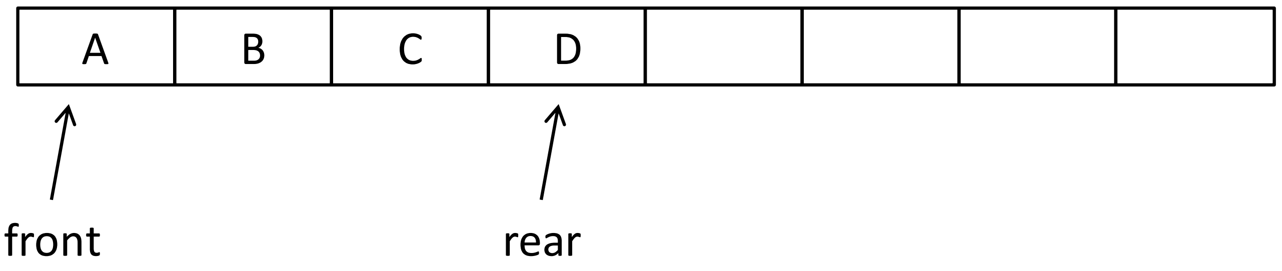


Serve

```
int  Serve()  {
      if (IsQueueEmpty()) {
          System.out.printf("\nQueue is empty\n");
          return 0;
      }

      else {
          int x = queue[front++];
          count--;
          return x;
      }
   }
```

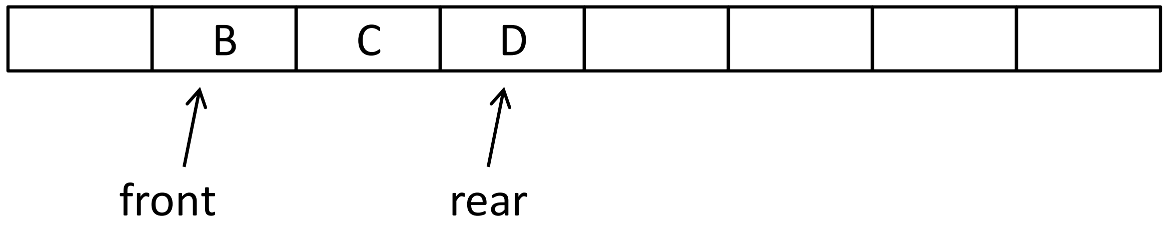
Circular Queue

- The need of circular queue
- **Consider the Linear Implementation**
    - Uses an ordinary array to hold the entries.
    - Need two indices, to keep track of both the front and the rear of the queue
    - Entries are not moved

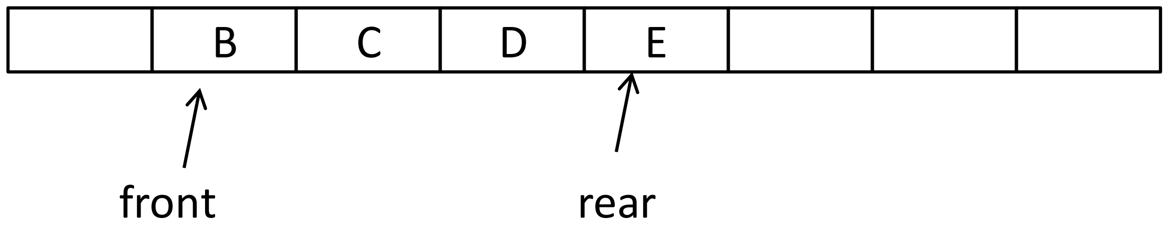


Circular Queue Contd…

- Delete an entry (i.e. operation Serve)



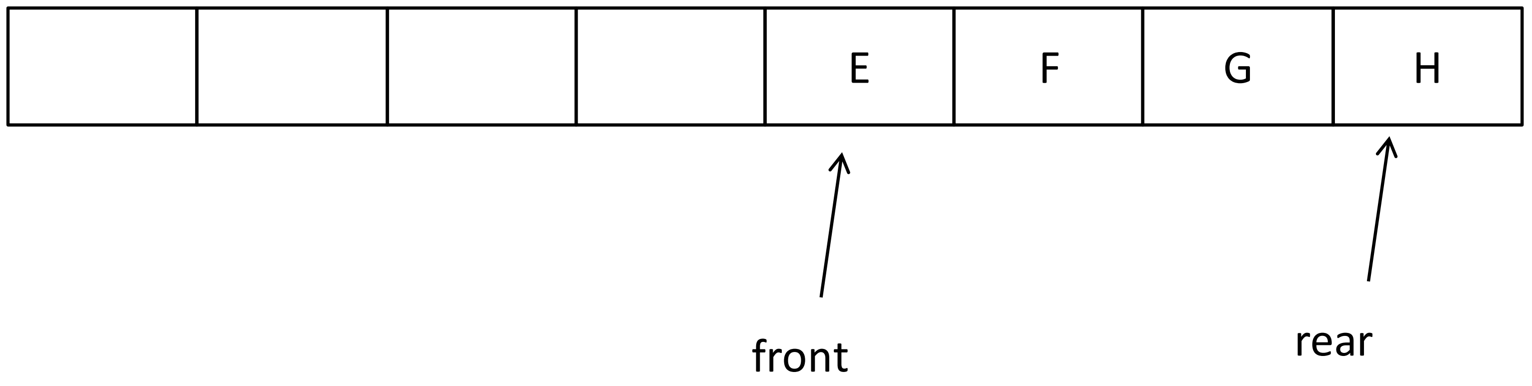
- Append an entry



Circular Queue Contd…

- Both the front and rear indices are increased but never decreased.
- Even if there are never more than two entries an unbounded amount of storage is needed.
- As the queue moves down the array, the storage space at the beginning of the array is discarded and never used again. (The queue may be full (out of space) but the beginning of the array may be unused.)

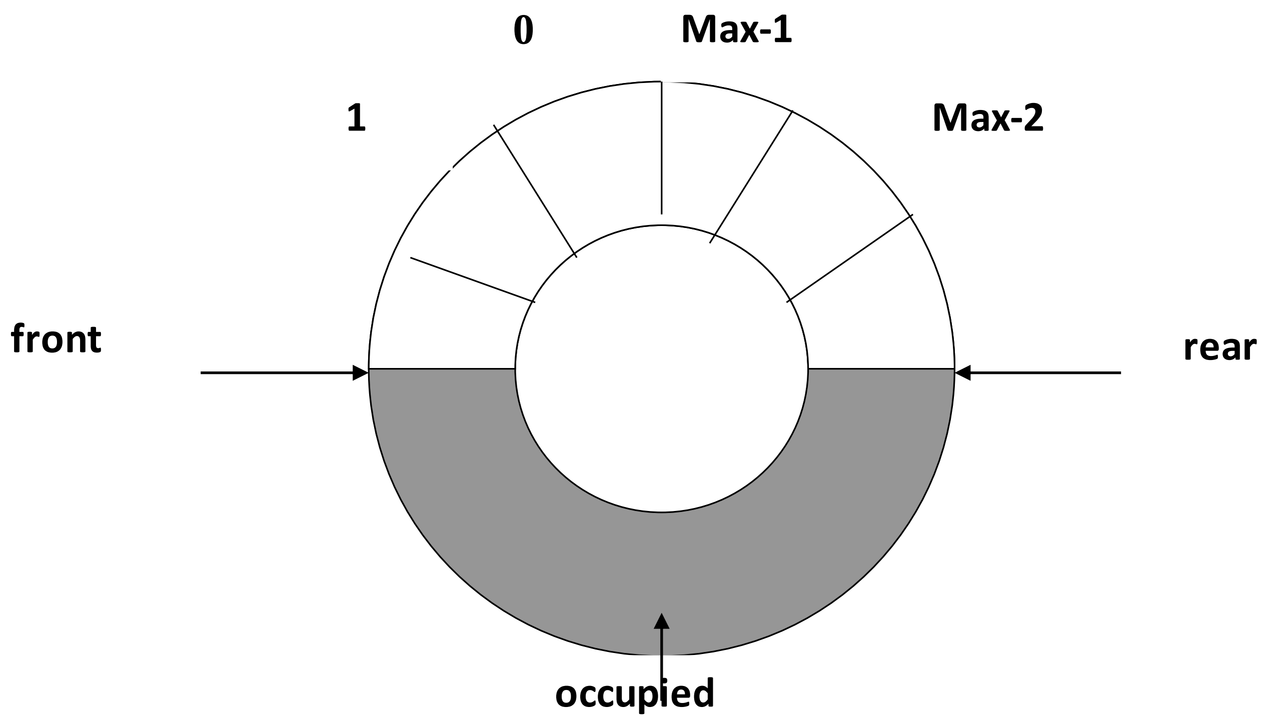
Circular Queue Contd…



Circular Queue Contd…

- Think the array as a circle rather than a straight line.
- Need two indices, to keep track of both the front and the rear of the queue

Circular Queue Contd…



Circular Queue Contd…

- When entries are added and removed from the queue, the head will continually chase the tail around the array.
- Need not worry about out of space unless the array is fully occupied.

Implementation of circular arrays (Contiguous implementation)

- Uses an ordinary linear array
- How to increase the index?

Circular Arrays - Contiguous implementation

- When we increase an index past Max –1 , start over again at 0.

  i.e.

```
 if ( i >= Max –1)
        i = 0;
 else
        i ++;
```

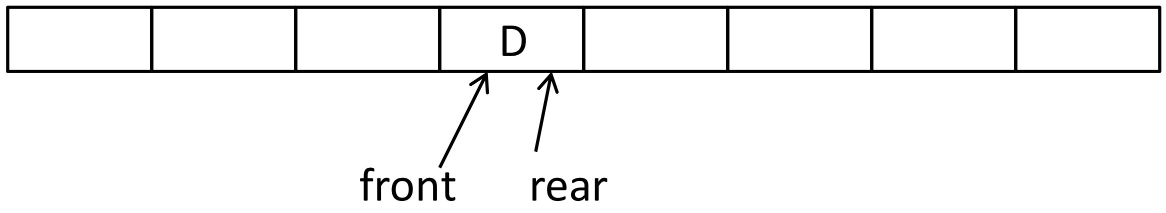
  This could be written as

```
    i = (i + 1 ) % Max;
```

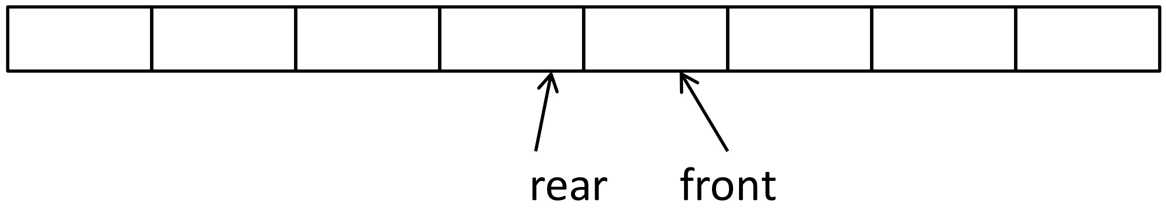
Circular Arrays - Contiguous implementation

Boundary conditions

- Queue containing one element



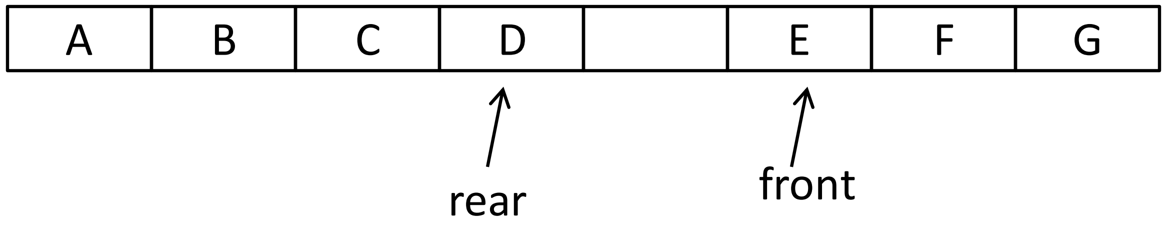
- Remove the item



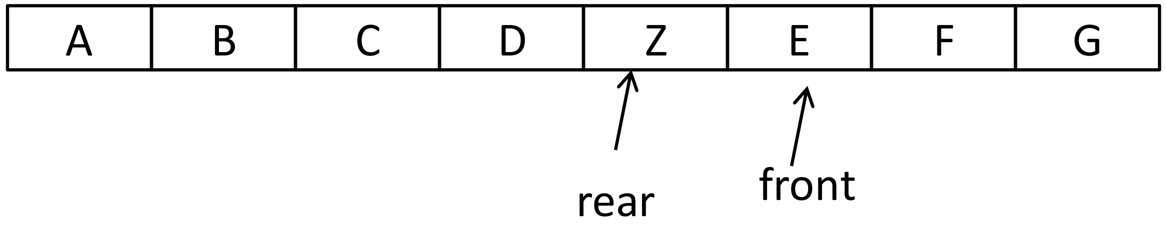
- Empty queue

Circular Arrays - Contiguous implementation

- Queue with one empty position



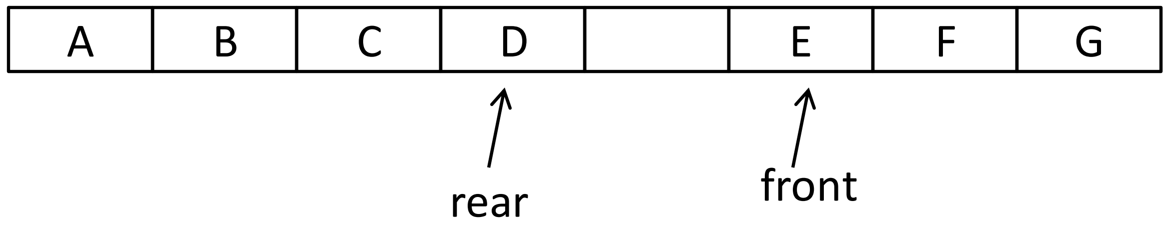
- Insert an item



- Note: the front and rear indices are in exactly the same relative positions for an empty queue and for a full queue.
- Cannot tell whether a queue is empty or full by looking at the indices alone.

Circular Arrays - Contiguous implementation

- Leave one empty position in the array, so that the queue is considered full when the rear index has moved within two positions of the front.
- Queue full



Circular Arrays - Contiguous implementation

- Have a Boolean variable that will be used when the rear comes just before the front to indicate whether the queue is full or not.
- Have an integer variable to count the number of entries in the queue.
- Front and rear indices taking special values to indicate emptiness.

```
class Queue
{
   private int[] queue;
   private int front;
   private int rear;
   private int maxSize;
   private int count;

   Queue(int size){
        queue = new int[size];
        maxSize = size;
        front = 0;
        rear = -1;
        count = 0;
       }
}
```

IsQueueEmpty

```
public boolean isQueueEmpty()
   {
       return (count == 0);
   }
```

IsQueueFull

```
public boolean isQueueFull()
   {
       return (count == maxSize);
   }
```

Append

```
public void append(int item)
   {
     if (isQueueFull()) {
          System.out.println(" Queue Full\n ");
     }
    else{
         System.out.println("Inserting " + item);
         rear = (rear + 1) % maxSize;
         queue[rear] = item;
         count++;}
 }
```

Serve

```
public int serve()
  {
     if (isQueueEmpty()){
         System.out.println(" Queue is Empty");
         return 0;}
    else {
              int x = queue [front];
              front = (front + 1) % maxSize;
              count--;
              return x;
         }
     }
```

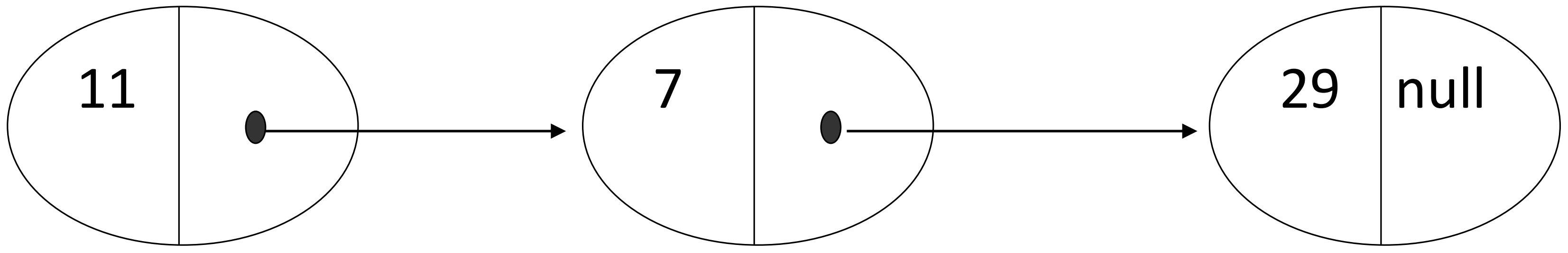
QueueSize

```
public int QueueSize()
  {
       return count;
   }
```

Linked Implementation

- Consider the queue of integers Q = 29,7,11 (29 is the rear, 11 the front).
- One memory cell or node for each element in the queue;
  - stores *two* things: the value of the element, which is an integer in this example but could be anything,
  - and a pointer to the memory cell of the next element.

Linked Implementation



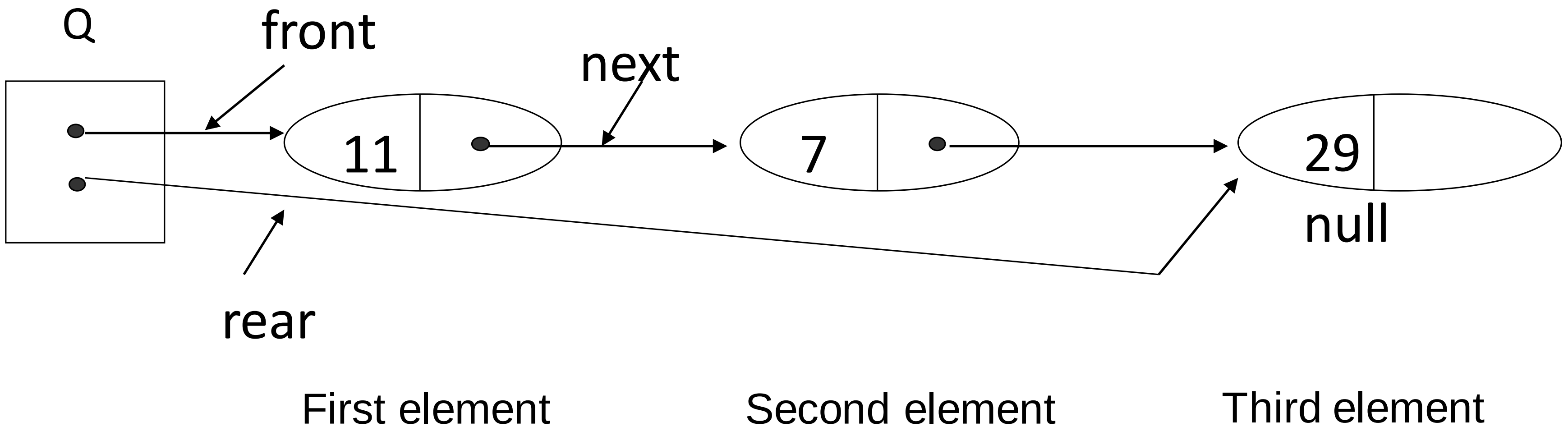
First element
(front)

Second element

Third element
(rear)

Linked Implementation

Where is the queue?



Linked Implementation

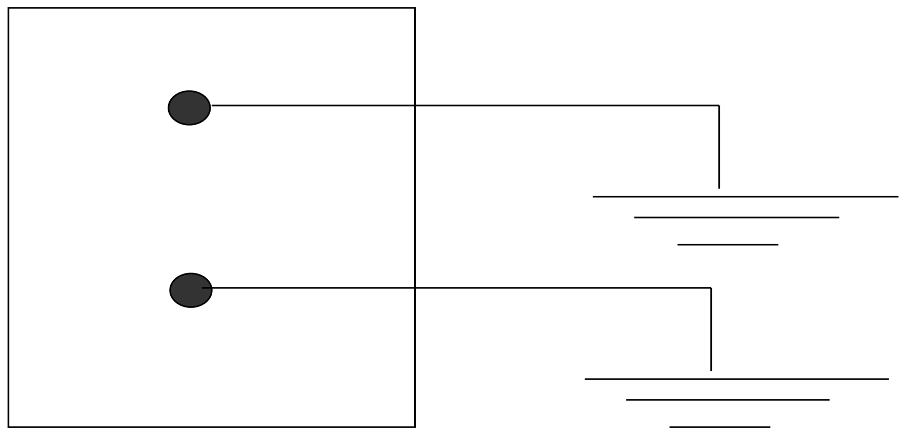
How will we represent the Empty queue?

Linked Implementation

- In Q we could store the *number* of elements that it contains, in which case the empty queue would be represented by setting this value to zero.
- Or we could have a boolean (logical) variable telling us whether or not Q is empty.
- The simplest solution is to set Q's pointers (i.e. front and rear) to NULL when Q is empty

Empty Queue

Q



Linked Implementation

- Establishing the structure of a cell or node.
- Identify the information needed for the node

  - The value of the element (in this example an integer, but could be anything)

  - A pointer to the next node

Linked Implementation

```
public class Node {
    int data;
    Node next;


}
```

Linked Implementation

The information for the queue is:

- A pointer (front) to point to the beginning of the queue
- A pointer (rear) to point to the end of the queue
- The number of elements of the queue.
- A Boolean variable telling whether the queue is full or not.

```
public class Queue {
    private Node front;
    private Node rear;
    private int queueSize;
    public Queue()
    {
        front = null;
        rear = null;
        queueSize = 0;
    }
}
```

IsQueueEmpty

```
public boolean isQueueEmpty()
{
    return (queueSize == 0);
}
```

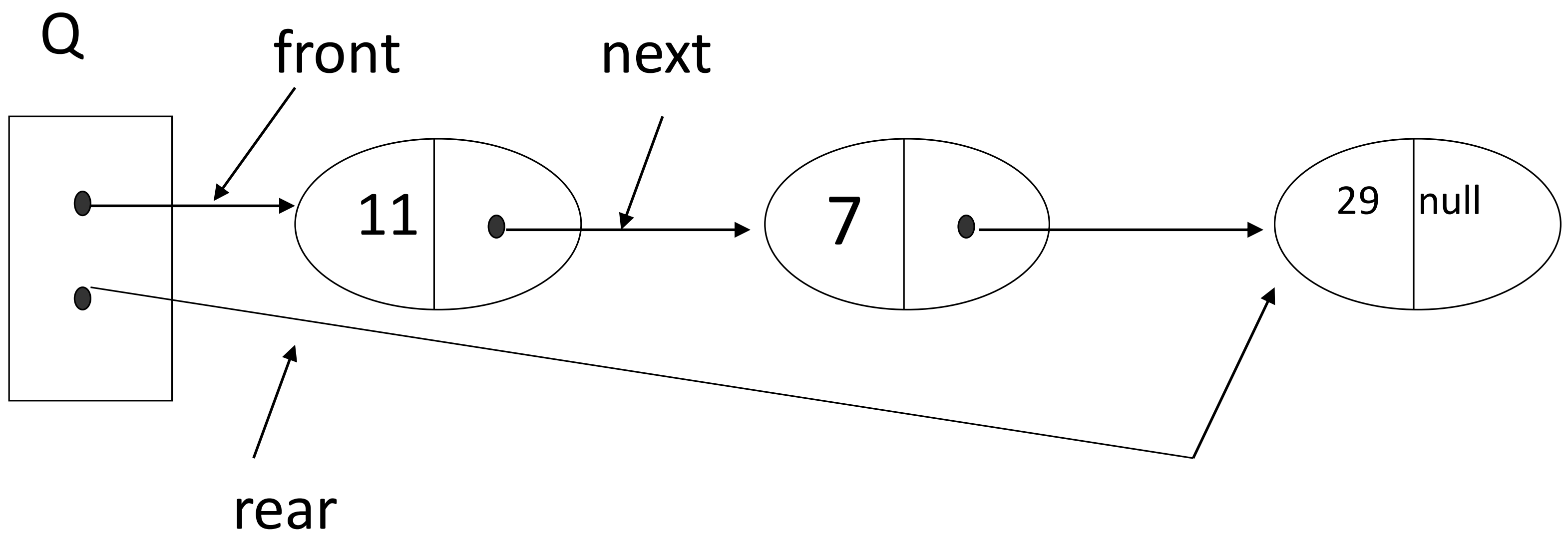
Draw the node-and-arc diagram to implement the Serve operations in java

Node-and-arc diagrams

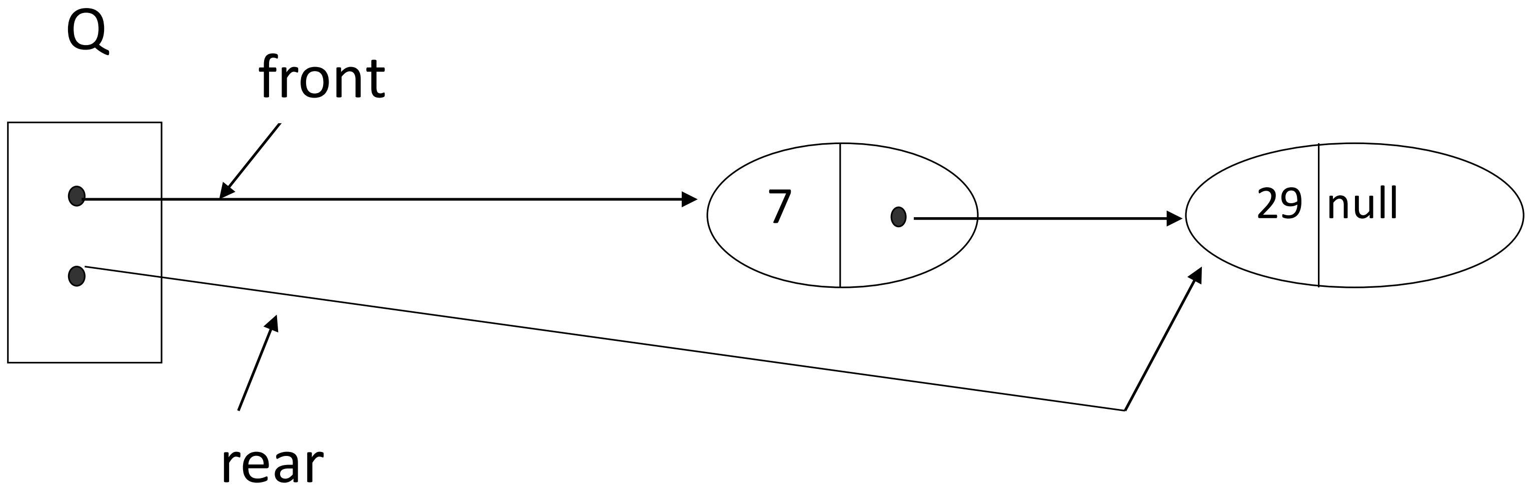
draw a *before* and *after* picture - two node-and-arc diagrams

- one showing the values passed to the operation *before* it is executed,
- the other showing the values created/changed by the operation *after* it has finished.

Example Serve operation Before diagram

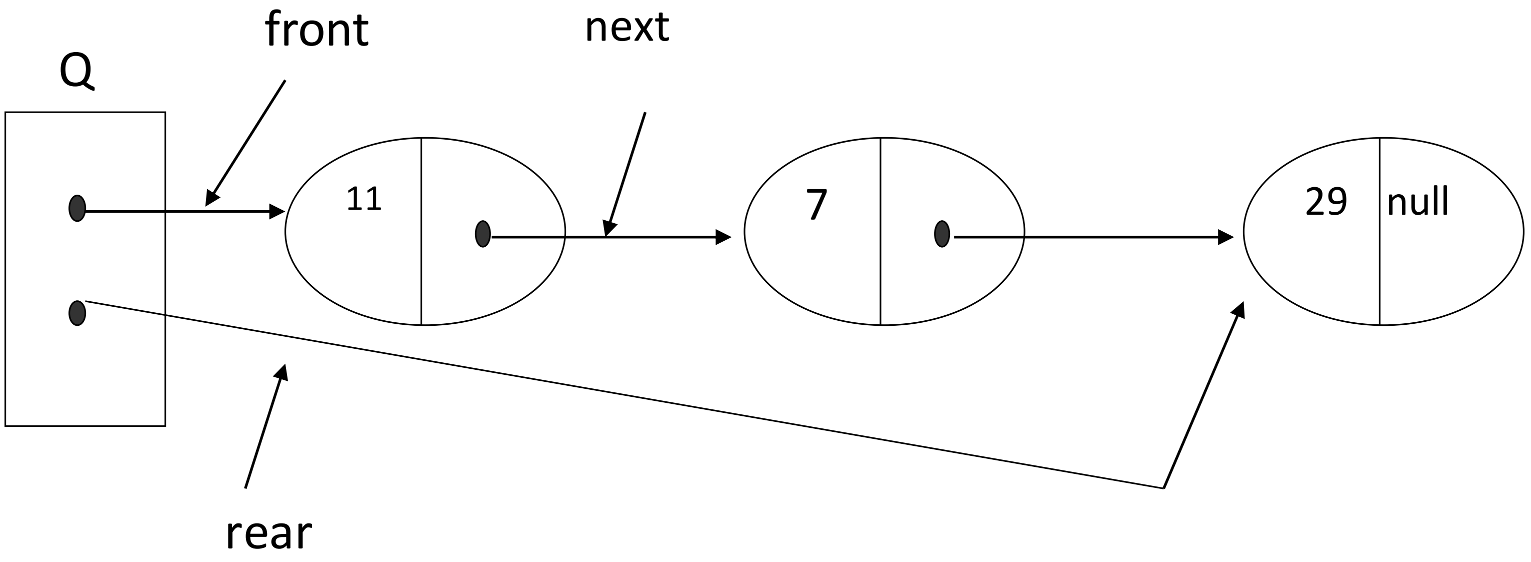


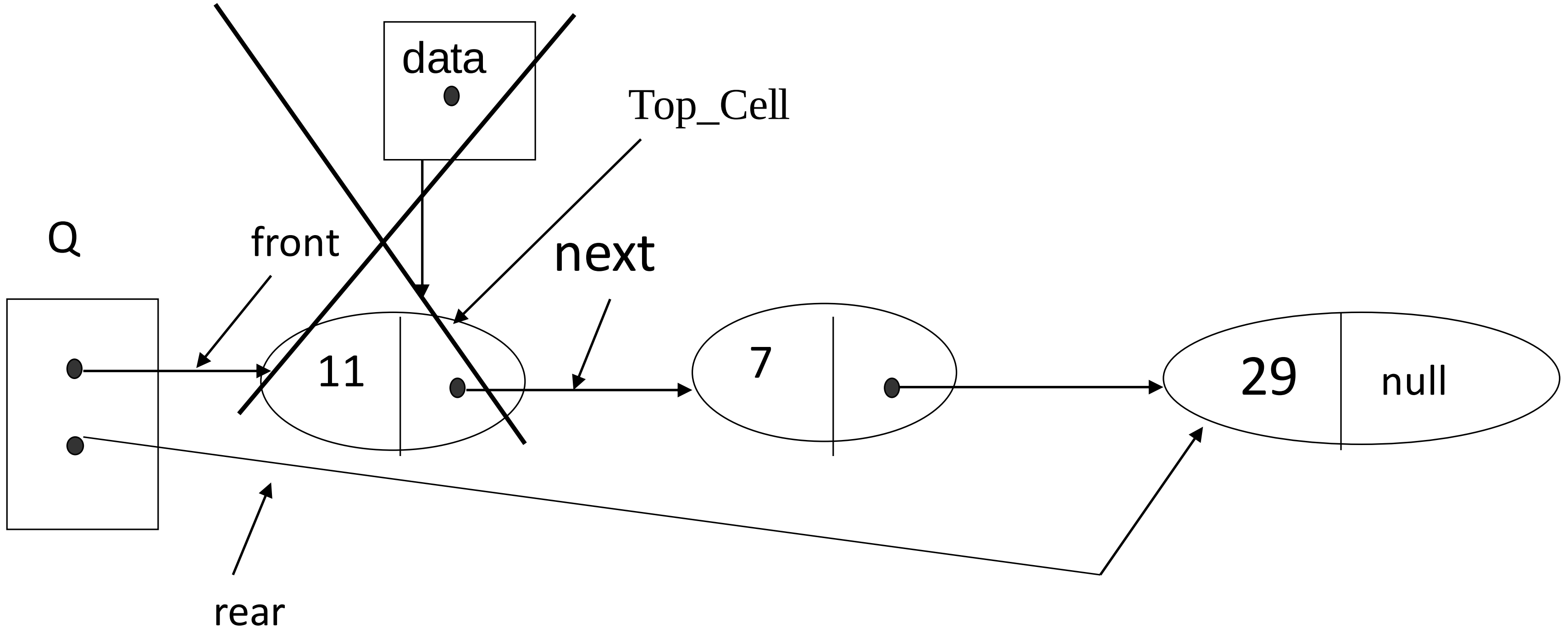
Example Serve operation After Diagram

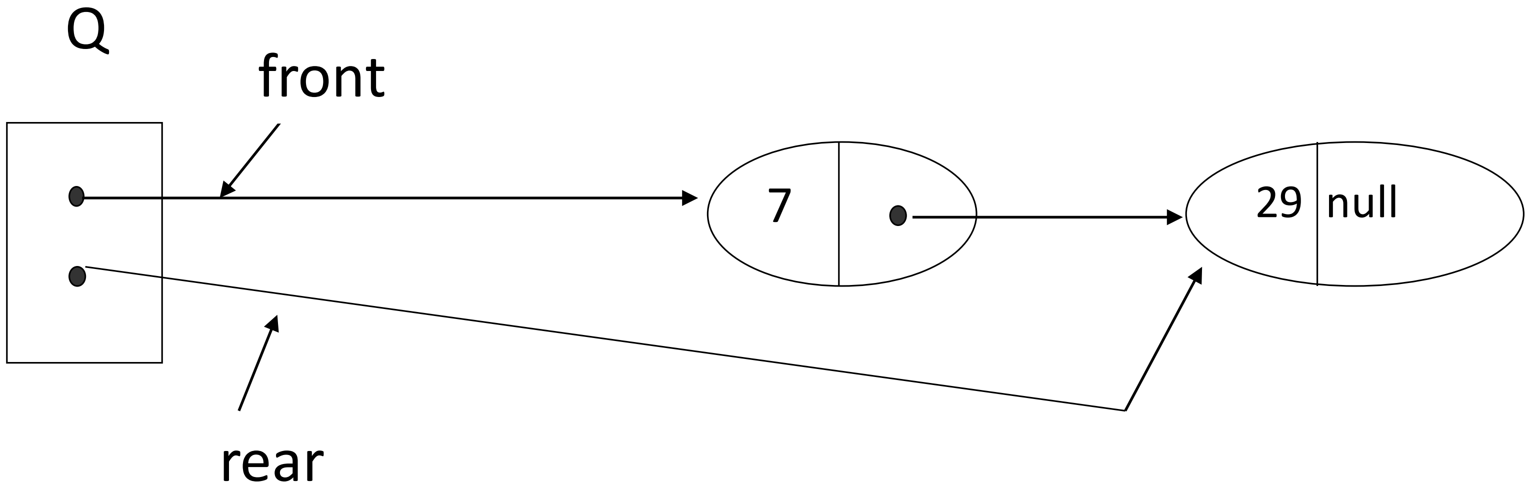


What should you do?

1. Assign the value of the first node to a variable (This is the value that we need)
2. Assign the address of the first node to a variable.
3. Assign the address of the second node to the front of the queue.
4. Update any queue values as necessary (example Decrease the number of elements in the queue by one.)
5. Check the queue is empty, if so set rear to NULL
6. Free the memory of the first node.


Q
front
next
11
7
29
null
rear


data
Top_Cell
Q
front
next
11
7
29
null
rear


Q
front
7
29
null
rear

Serve

```
public int Serve()
 {
     if (isQueueEmpty()) {
          System.out.printf("\nQueue is empty\n");
          return 0;}
      else {
          int data = front.data;
          front = front.next;
          queueSize--;
          return data;}
}
```

Append

```
public void Append(int data)
 {
     Node oldRear = rear;
     rear = new Node();
     rear.data = data;
     rear.next = null;
     if (isQueueEmpty()) {
         front = rear;
      }
     else  {
        oldRear.next = rear;
      }
      queueSize++;
}
```

Summary of main points

- Definition of a Queue
- Implementation of the Queue operation