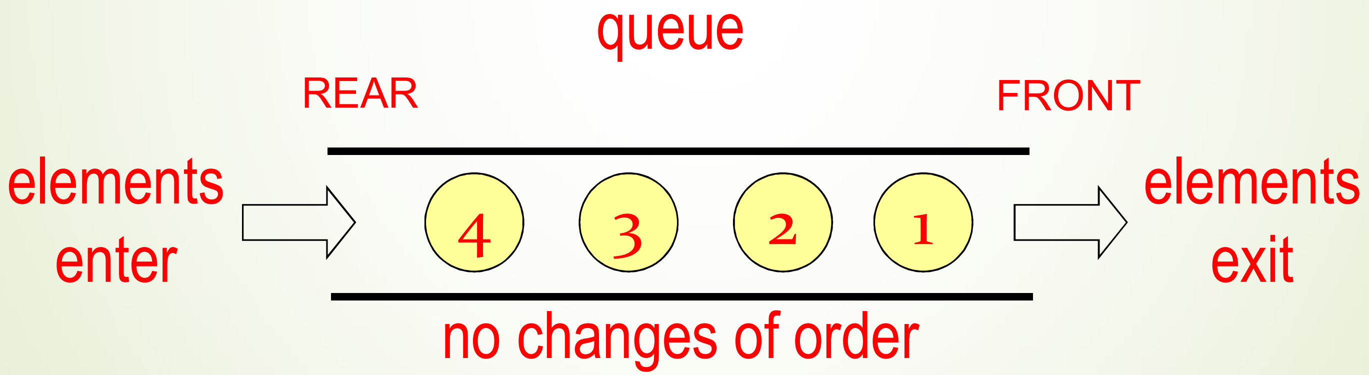
Data Structures

QUEUE

- It is a non-primitive, linear data structure in which insertion (i.e. ENQUEUE) takes place from one end called **REAR** and deletion (i.e. DEQUEUE) of elements takes place from other end called **FRONT**.



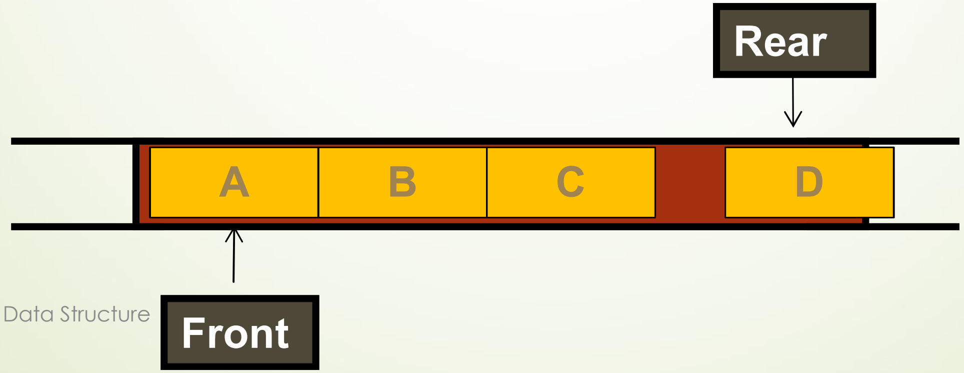
QUEUE

- First element inserted in list, will be the first to be removed.
- Thus, Queue is known as FIFO (**First In-First Out) or FCFS (First Come First Serve).**
- **Examples of Queue**
  - People waiting in Queue to purchase tickets at railway station or cinema hall,

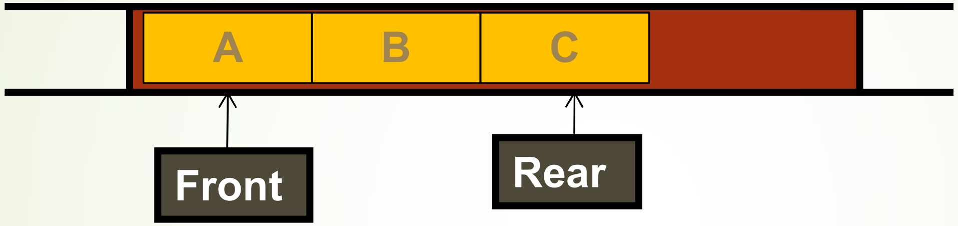
    First person in the queue will be served first.

Representation of Queue

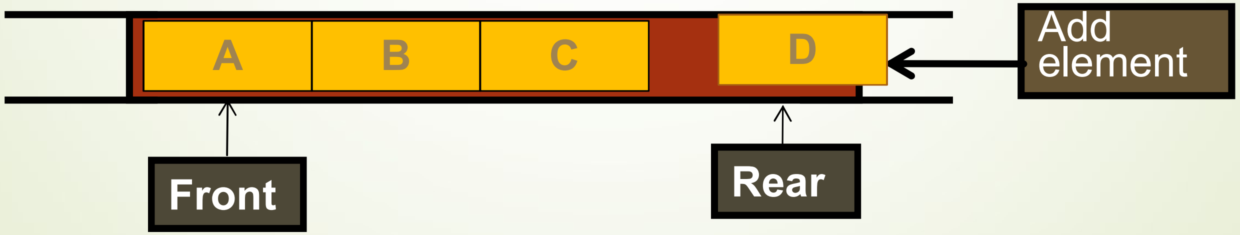
- It has two pointer variables
  - FRONT : Points to be element to be deleted
  - REAR : Points to newly inserted element
- When queue is empty
  - FRONT = -1 and REAR = -1
- When queue is not empty



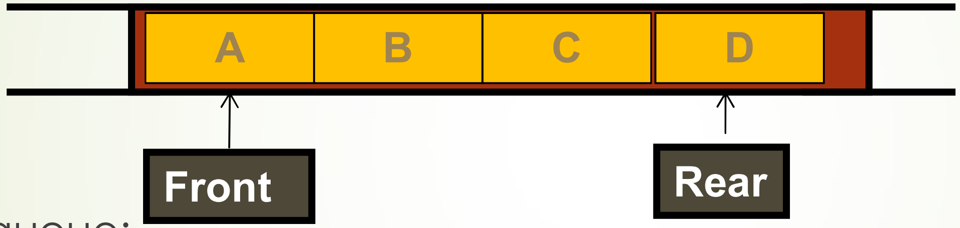
Representation of Queue



- Enqueue:



Representation of Queue

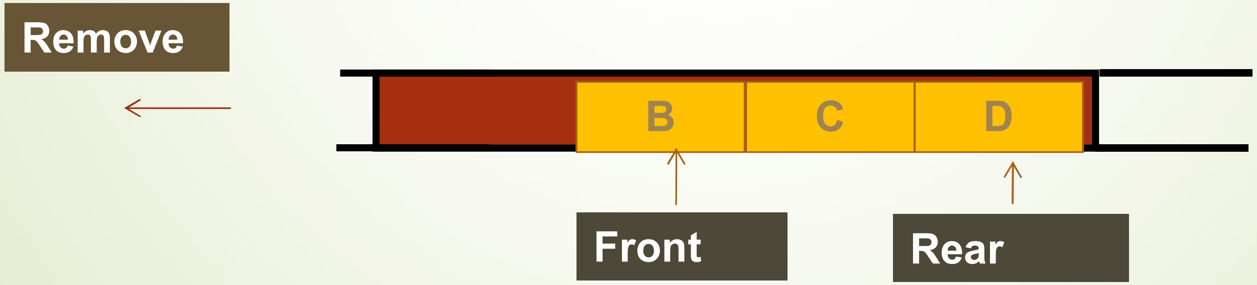


A B C D

Front

Rear

- Dequeue:



Remove

B C D

Front

Rear

Representation of Queue

Implementation Data Structure

- Array
- Stack
- Linked List

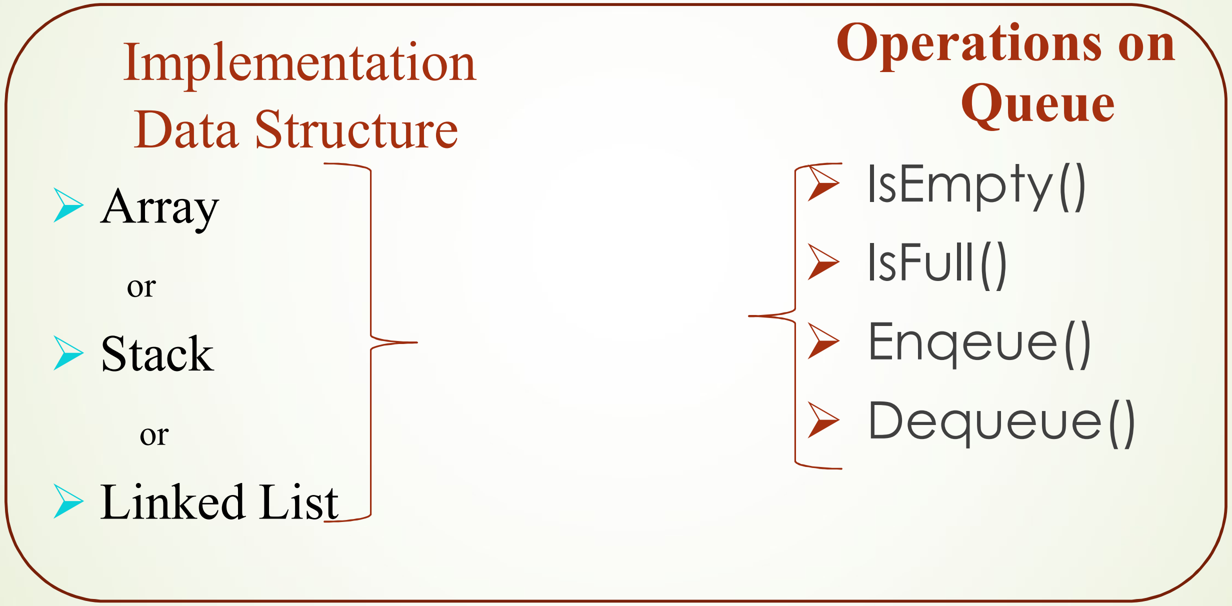
Types of Queue

- Linear Queue
- Circular Queue
- Deque
- Priority Queue

Operations on Queue

- IsEmpty()
- IsFull()
- Enqeue()
- Dequeue()

ADT of Queue



Implementation Data Structure

- Array

or

- Stack

or

- Linked List

Operations on Queue

- IsEmpty()
- IsFull()
- Enqeue()
- Dequeue()

Linear Queue

- It is like a **straight line** in which all elements stand one behind the other.
- It has a definite beginning and a definite end.

Status of Linear Queue after some operations (Example)

A Queue of N=5 elements:

- Empty Queue => FRONT = -1 ,REAR = -1

=> FRONT = 0 ,REAR =0

- Enqueue(10)

=> FRONT = 0 ,REAR =1

- Enqueue(20)

=> FRONT = 0 ,REAR =2

- Enqueue(30)

=> FRONT = 0 ,REAR =3

- Enqueue(40) => FRONT = 0 ,REAR =4
- Enqueue(50) => FRONT = 1 ,REAR =4
- Dequeue()

0	1	2	3	4
10				
10	20			
10	20	30		
10	20	30	40	
10	20	30	40	50
	20	30	40	50

Enqueue(60) ➔ Since REAR = 4 (i.e. N-1)➔ Overflow

Operations in Linear Queue

Algorithm IsEmpty()

```
step1. Start
step2. if  (F=-1 and R = -1) return True  ➔ Queue is Empty
step3. else return False
Step4. End
```

Operations in Linear Queue

Algorithm IsFull()

```
step1. Start
step2. if   R = N-1 return True ➔ Queue is Full
step3. else return False
Step4. End
```

Operations in Linear Queue

Algorithm EnQueue(value)

```
Step 1: start
Step2: [check for queue is over flow or not] ➔ operation Validation
        If (R=N-1)
                Print "queue is overflow"
                    go to step 5
      else      go to step 3
Step 3: [check condition]            ➔ If it is  First Element of Queue
          If(R=-1)    R:=F:=0
          else R:=R+1
Step 4:[insert  item into Queue] ➔ insert Element
          QUEUE[R]:=value
Step 5:end
```

Operations in Linear Queue

Algorithm DeQueue()

```
Step 1: start
Step2: [check for queue is under flow or not] → operation Validation
        If (F = -1)
                Print "queue is underflow"
                    go to step 5
        else        go to step 3
Step 3: [Copy item From Queue]
            X:= QUEUE[F]
Step 4:[check condition]  → It is only Element of Queue or not
            If(F=R)     F :=  R := -1 → Empty Queue
            Else F:=F+1
Step 5:end
```

Possible Values for Front F and Rear R in Linear Queue

- Assume Queue of Size of N Elements (i.e. Q[0…. N-1]).

1. F = -1 and R= -1 ➔ Empty Q
2. F= 0 and R =0 ➔ only one element present in the Q
3. F = R != -1 ➔ only one element present in the Q
4. F=0 and R=N-1 ➔ Q is Full
5. R= N-1 ➔ Q is Full
6. R= N-1 ➔ F € (0 to N-1) ➔ Q is Full

No further Enqueue Operation till R= N-1

Note1: R= N-1 ➔ F € (1 to N-1) ➔ Q is full even it has space to store more elements ➔ i.e. **Disadvantage of Linear Queue**

Note2: [F= -1 and R € (0 to N-1)]OR [F € (0 to N-1) and R=N-1] ➔ impossible Case.

That means Both pointers F and R Either will be inside or Out side of Queue

Drawback of Linear Queue

- Once the queue is full, even though few elements from the front are deleted and some occupied space is relieved, it is not possible to add anymore new elements, as the rear has already reached the Queue's rear most position.

To Overcome From the Drawback of Linear, Move to the Circular Queue