QUEUES:

The Queue can be formally defined as ordered collection of elements that two ends named as front and rear. From the front end. One can delete the elements and from the rear end one can insert the elements.

94	10	20	11	55	72	61

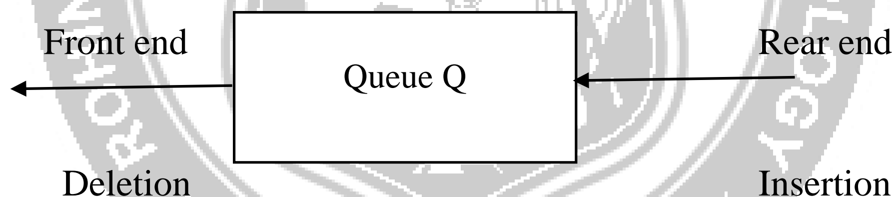
Front Rear

QUEUE ADT

Queue is a collection of elements in which the element can be inserted by one end called rear and elements can be deleted by other end called front. Queue is a Linear Data Structure that follows First in First out (FIFO) principle.

Before the insertion of the element in the qu

- Insertion of element is done at one end of the Queue called "**Rear** "end of the Queue.
- Deletion of element is done at other end of the Queue called "**Front** "end of the Queue.
- Example: - Waiting line in the ticket counter.



Queue Model

Operations on Queue

Fundamental operations performed on the queue are

1. EnQueue
2. DeQueue

1. EnQueue operation:-

- It is the process of inserting a new element at the rear end of the Queue.
- For every EnQueue operation
  - Check for Full Queue
  - If the Queue is full, Insertion is not possible.
  - Otherwise, increment the rear end by 1 and then insert the element in the rear end of the Queue.

2. DeQueue Operation:-

- It is the process of deleting the element from the front end of the queue.
- For every DeQueue operation
  - Check for Empty queue
  - If the Queue is Empty, Deletion is not possible.
  - Otherwise, delete the first element inserted into the queue and then increment thefront by 1.

Exceptional Conditions of Queue

- Queue Overflow
- Queue Underflow

(i) Queue Overflow:

- An Attempt to insert an element X at the Rear end of the Queue when theQueue is full is said to be Queue overflow.
- For every Enqueue operation, we need to check this condition.

(ii) Queue Underflow:

- An Attempt to delete an element from the Front end of the Queue when the Queue is empty is said to be Queue underflow.
- For every DeQueue operation, we need to check this condition.

<u>Implementation of Queue</u>

Queue can be implemented in two ways.

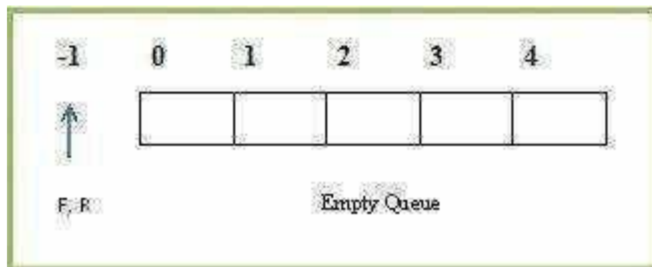
1. Implementation using Array (**Static Queue**)
2. Implementation using Linked List (**Dynamic Queue**)

1. Array Implementation of Queue

Array Declaration of Queue:

#define ArraySize 5int Q [ArraySize];
or
int Q [5];

Initial Configuration of Queue:



(i) Queue Empty Operation:

Initially Queue is Empty.

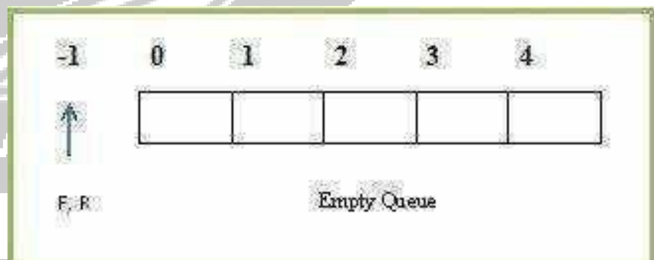
- With Empty Queue, Front (F) and Rear (R) points to – 1.

It is necessary to check for Empty Queue before deleting (DeQueue) an element from theQueue (Q).

Routine to check for Empty Queue

```
int IsEmpty ( Queue Q )
{
if( ( Front = = - 1) && ( Rear = = - 1 ) )return ( 1 );
}
```

```
int IsEmpty ( Queue Q )

{

if( ( Front = = - 1) && ( Rear = = - 1 ) )

        return ( 1 );

}
```



(ii) Queue Full Operation

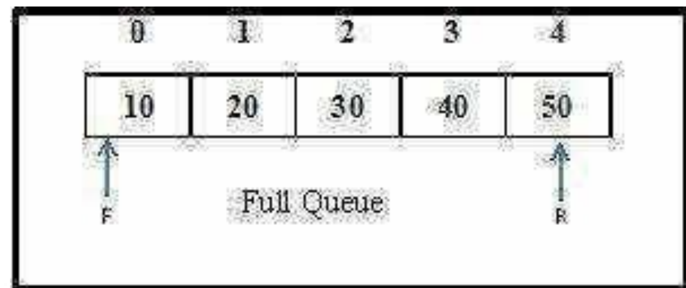
As we keep inserting the new elements at the Rear end of the Queue, the Queue becomesfull.

When the Queue is Full, Rear reaches its maximum Arraysize.

- For every Enqueue Operation, we need to check for full Queue condition.

Routine to check for Full Queue

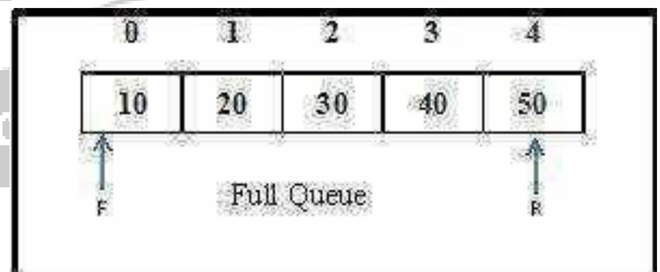
```
int IsFull( Queue Q )

{

if ( Rear = = ArraySize -

    1 )return ( 1 );
```



(i) Enqueue Operation

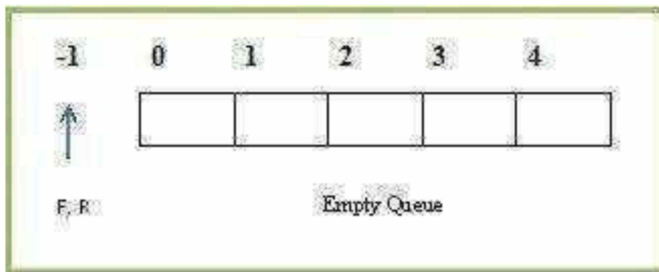
- It is the process of inserting a new element at the Rear end of the Queue.
- It takes two parameters, Enqueue(X, Q). The elements X to be inserted at the Rear end ofthe Queue Q.
- Before inserting a new Element into the Queue, check for Full Queue.If the Queue is already Full, Insertion is not possible.
- Otherwise, Increment the Rear pointer by 1 and then insert the element X at the Rear end of the Queue.
- If the Queue is Empty, Increment both Front and Rear pointer by 1 and then insert the element X at the Rear end of the Queue.

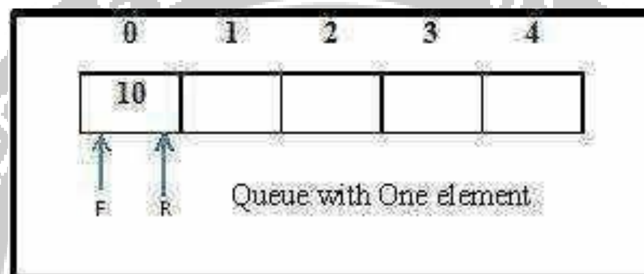
Routine to Insert an Element in a Queue




```
void EnQueue (int X , Queue Q)
{
if ( Rear = = Arraysize - 1)
print (" Full Queue !!!!. Insertion notpossible");
else if (Rear = = - 1)
{
Front = Front + 1;
Rear = Rear + 1;
Q [Rear] = X;
}
else
{
```

```
Rear = Rear + 1;
Q [Rear] = X;
}
}
```

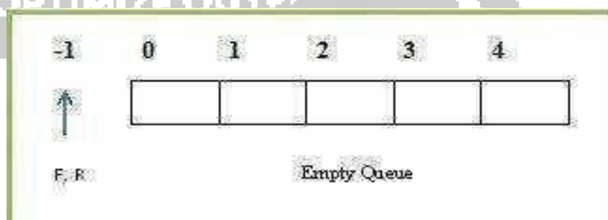


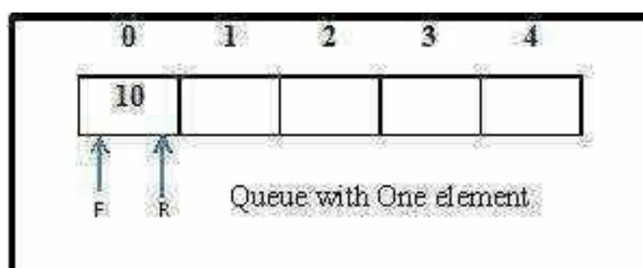


(iv)DeQueue Operation

- It is the process of deleting a element from the Front end of the Queue.
- It takes one parameter, DeQueue (Q). Always front element in the Queue will be deleted.Before deleting an Element from the Queue, check for Empty Queue.
- If the Queue is empty, deletion is not possible.
- If the Queue has only one element, then delete the element and represent the empty queueby updating Front = - 1 and Rear = - 1.
- If the Queue has many Elements, then delete the element in the Front and move the Frontpointer to next element in the queue by incrementing Front pointer by 1.

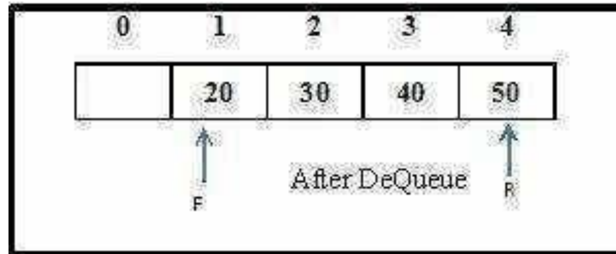
ROUTINE FOR DEQUEUE






```
void DeQueue ( Queue Q )
{
if ( Front = = - 1)
print (" Empty Queue !. Deletion not possible " );
else
if( Front = = Rear )
{
X = Q [ Front ];
Front = - 1;
Rear = - 1;
```

```
}
else
{
X = Q [ Front ];
 Front = Front + 1 ;
}}
```

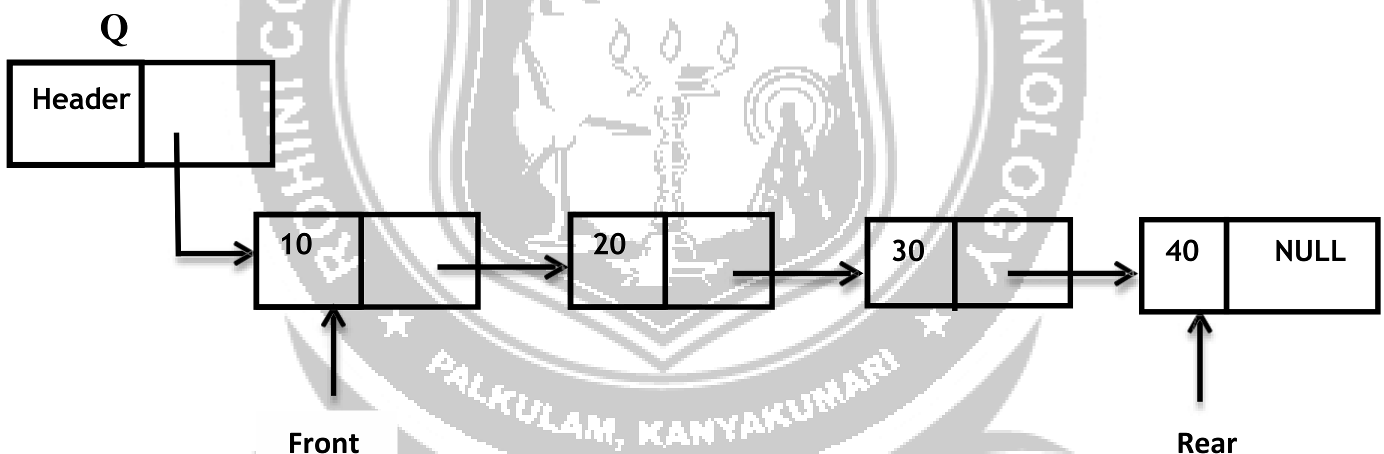


Linked List Implementation of Queue

- Queue is implemented using SLL (Singly Linked List) node.
- Enqueue operation is performed at the end of the Linked list and DeQueue operation is performed at the front of the Linked list.
- With Linked List implementation, for Empty queue

Front = NULL & Rear = NULL

Linked List representation of Queue with 4 elements



Declaration for Linked List Implementation of Queue ADT

```
struct node;
typedef struct node * Queue;
typedef struct node * position;
int IsEmpty (Queue Q);
Queue CreateQueue (void);
void MakeEmpty (Queue Q);
void Enqueue (int X, Queue Q);
void Dequeue (Queue  Q);
struct node
{
```

```
int data ;
position next;
}* Front = NULL, *Rear = NULL;
```

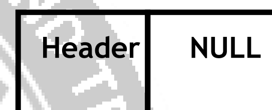
(i) Queue Empty Operation:

- Initially Queue is Empty.
- With Linked List implementation, Empty Queue is represented as S -> next = NULL.
- It is necessary to check for Empty Queue before deleting the front element in the Queue

ROUTINE TO CHECK WHETHER THE QUEUE IS EMPTY

```
int IsEmpty (Queue Q)
{
if(Q→ next==NULL)
return(1);

}
```

Q



Empty Queue

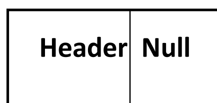
(i) EnQueue Operation

- It is the process of inserting a new element at the Rear end of the Queue.
- It takes two parameters, EnQueue (int X , Queue Q). The elements X to be inserted intothe Queue Q.
- Using malloc ( ) function allocate memory for the newnode to be inserted into the Queue.
- If the Queue is Empty, the newnode to be inserted will become first and last node in thelist. Hence Front and Rear points to the newnode.
- Otherwise insert the newnode in the Rear -> next and update the Rear pointer.

Routine to EnQueue an Element in Queue

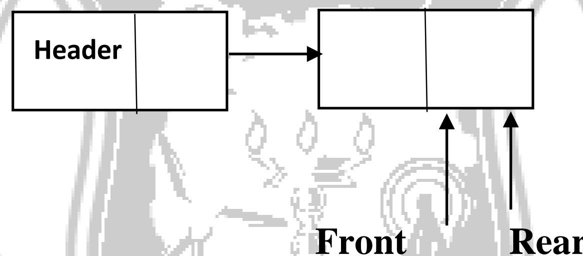
```
void EnQueue ( int X, Queue Q )
{
struct node *newnode;
newnode = malloc (sizeof (struct  node));
if (Rear = = NULL)
{
newnode→ data = X;
newnode→ next=NULL;
Q -> next = newnode;
Front = newnode;
```

```
Rear = newnode;
}
else
{
newnode→    data = X;
newnode→    next=NULL;
Rear -> next = newnode;
Rear = newnode;
}
}
```



Empty Queue

Before Insertion

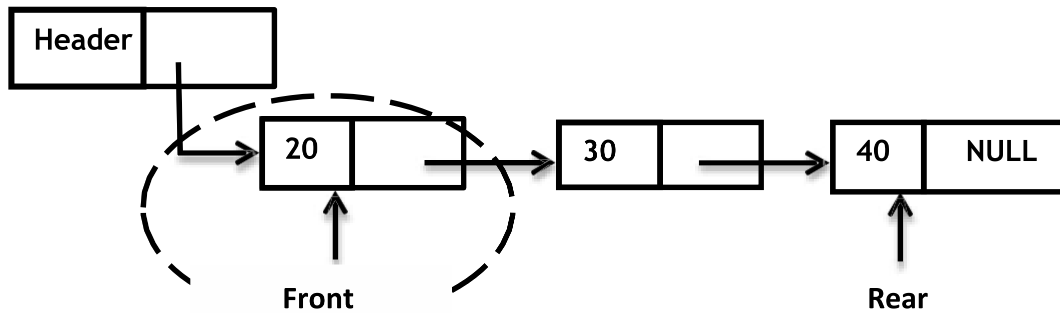


After Insertion

DeQueue Operation

- It is the process of deleting the front element from the Queue.
- It takes one parameter, Dequeue (Queue Q). Always element in the front (i.e) element pointedby Q -> next is deleted always.
- Element to be deleted is made "temp".
- If the Queue is Empty, then deletion is not possible.
- If the Queue has only one element, then the element is deleted and Front and Rear pointer ismade NULL to represent Empty Queue.
- Otherwise, Front element is deleted and the Front pointer is made to point to next node in the list. The free ( ) function informs the compiler that the address that temp is pointing to, is unchanged but the data present in that address is now

undefined



Routine to DeQueue an Element from the Queue

```
void DeQueue ( Queue Q )
{
struct node *temp;
if ( Front = = NULL )
error ("Empty Queue!!! Deletion not possible." );
else
if (Front = = Rear)
{
temp = Front;
Q -> next = NULL;
Front = NULL;
Rear = NULL;
free ( temp );
}
else
{
temp = Front;
Q -> next = temp -> next;
Front = Front  ->Next; free (temp);
}
}
```

Applications of Queue

1. Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
2. In real life, Call Center phone systems will use Queues, to hold people calling them in anorder, until a service representative is free.
3. Handling of interrupts in real-time systems. The interrupts are handled in the same orderas they arrive, First come first served.
4. Batch processing in operating system.

5. Job scheduling Algorithms like Round Robin Algorithm uses Queue.

Drawbacks of Queue (Linear Queue)

- With the array implementation of Queue, the element can be deleted logically only bymoving Front = Front + 1.
- Here the Queue space is not utilized fully.