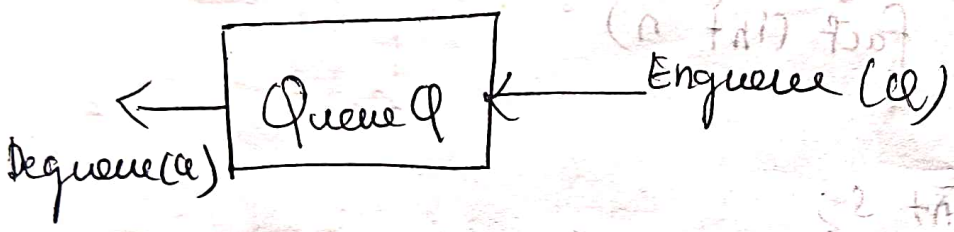
The Queue ADT

Queue is a linear data structure which follows First In First out Principle in which insertion is performed at rear end and deletion is performed at front end.

Model of a queue



operations on Queue

Enqueue :-

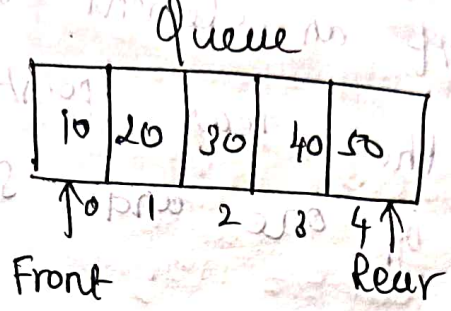
which inserts an element at the end of the list (called the rear).

Dequeue :-

which deletes the element at the start of the list (known as front)

* overflow :-

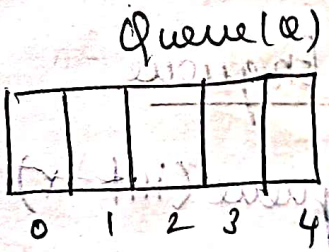
Attempt to insert an element when the queue is full it said to be overflow



Enqueue (60) - it causes overflow

* Underflow

Attempt to delete an element when the queue is empty it said to be under flow.



Front = -1
Rear = -1

Dequeue(Q) causes underflow

Queue Implementation

There are two ways for implementing the queue

* Array Implementation
* Linked list implementation

Array Implementation

There are two pointers used for implementing. one is Rear and other one is Front pointer

* To insert an element 'x' on to the queue(Q). the rear pointer is incremented by one and said

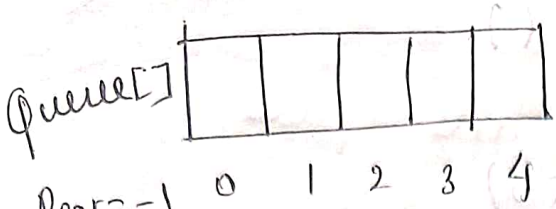
```
Queue[rear] = x;
```

* To delete an element Q[front] is returned and the front pointer is incremented

Routine to Enqueue

```
Void enqueue (int x)
{
  if (rear > max_Arraysize)
  printf ("Queue overflow");
  else
  {
    rear = rear+1;
    Queue[rear] = x;
  }
}
```

Example :-



Rear = -1

Front = 0

Enqueue (10)

if (rear > maxArraySize)

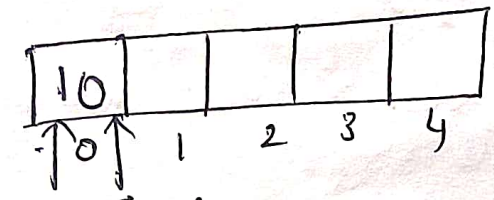
-1 > 5 ✗

else

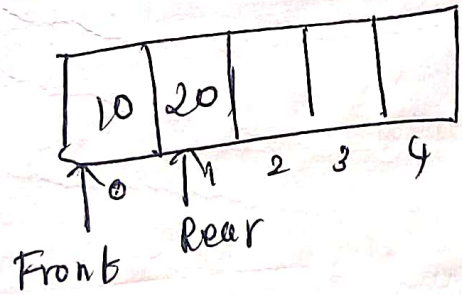
rear = rear + 1

rear = -1 + 1 = 0

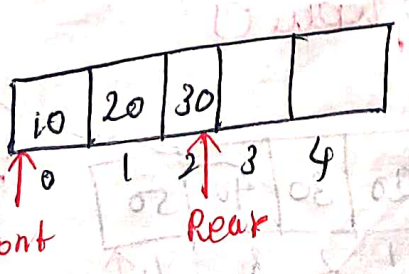
Queue [0] = 10



Enqueue (20)



Enqueue (30)

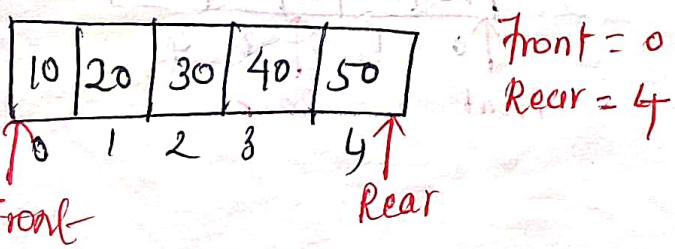


Dequeue (Q)

```
Void delete ()
{
 if (front < 0)
 printf ("underflow");
 else
 {
  x = queue [front];
  if (front == rear)
  {
    front = 0;
    rear = -1;
  }
 else
 front = front + 1;
 }
return x;
}
```

Example

① Dequeue ()



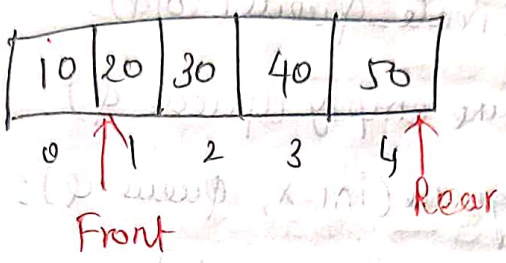
x = Queue[Front]

x = Queue[0]

x = 10

front = front + 1

front = 0 + 1 = 1



② Dequeue()

Front = 1

Rear = 4

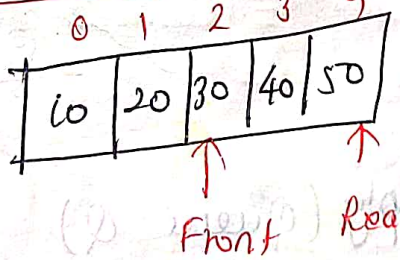
x = Queue[1]

x = 20

Front = Front + 1

= 1 + 1

Front = 2



Front = 2

Rear = 4

Linked List Implementation of Queue

Enqueue operation is performed at end of the list. Dequeue operation is performed at Front of the list.

Routine to Enqueue an Element

```
Void Enqueue(int x)
{
 struct node *newnode;
 newnode = malloc (sizeof(struct node));
 if (rear == NULL)
 {
  newnode -> data = x;
  newnode -> next = NULL;
  front = newnode;
  rear = newnode;
 }
 else
 {
  newnode -> data = x;
  newnode -> next = NULL;
  rear -> next = newnode;
  rear = newnode;
 }
}
```

Example

Enqueue x = 10

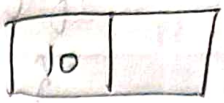
newnode = 

rear = NULL

front = NULL

if (rear == NULL) (True)

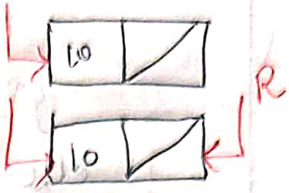
newnode → data = 10



newnode → next = NULL

front = newnode

rear = newnode



Enqueue X = 20

newnode =

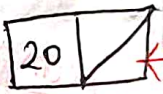
- if (rear == NULL) (False)

else

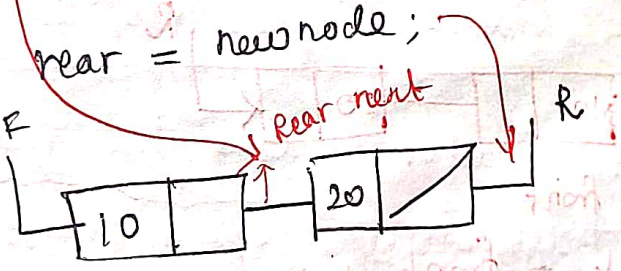
newnode → data = 20 ⇒

newnode → next = NULL ⇒

rear → next = newnode;



rear = newnode;

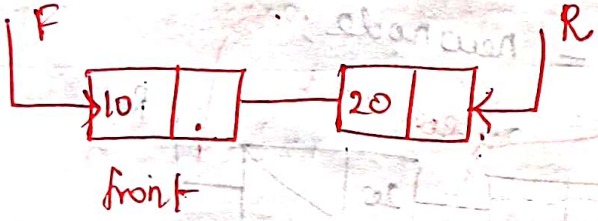


Routine to Dequeue

```
Void dequeue (Queue Q)
{
    Struct node *temp;
    if (front == NULL)
    Error ("Underflow");
    else
    { temp = front;
```

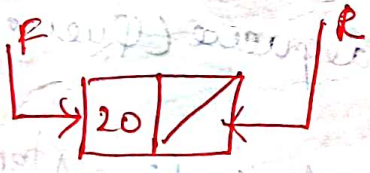
```
if (front == rear)
 {
  front = NULL;
  rear = NULL;
 }
else
{
front = front -> next;
printf ("temp -> data");
free (temp);
}
}
}
```

Example



front = front -> next;

front = 20



Displays the deleted data & deallocate the memory allocated for the variable temp