Queue

A Queue is an organisation which restricts the entry and exit from to opposite points. such as organisation is required for the uniform distribution of a limited resource that has to be shared by multiple users. When we store a list of data items in the memory. We can make a constraints to the array or lisked list that the new items can be inserted from a certain point called as the rear of Queue. while an item can be deleted from an another point called as front of the Queue.

first when we insert an a item in a queue both the rear and fronts are set a similar points. As soon as a new items are inserted the rear bron up and the front remains same. Since the item is deleted from the front position it would be the item inserted first. For this reason the Queue also called FIFO list (First in first out).

In our daily life there are so many ~~user~~ uses of Queue. A Queue is organised in front of the ticket windows, at the the bus stop, in the doctor Cabin and so on. In the computer's memory we may get so many queue organised to share the resources. The OS itself organises multiple queue in its buffer and in the memory to perform multi-tasking in a smooth way.

Some examples of such Queue are job Queue, ready Queue, wait Queue, I/O Queue, etc.

Operation on the Queue

The Queue can be sorted both as an array and as a linked list. In any case four operations are performed on a Queue :-

[illegible]

(I) Creation
(II) EN Queue
(III) DE Queue
(IV) Traversal

Array based Queue creation :-

To store a Queue as an array at first we need an array of some primitive datatype and two integer variable to to indicate the rear and front of the the Queue. We can take them individually or we can tie up them in a structure. The algorithm is given below :-

```
Step-1 :- Start Creation
Step-2 :- Define structure with member
          val[], R and F
Step-3 :- Declare variable Q of
          Queue type.
Step-4 :- let R of Q = -1
Step-5 :- let F of Q = -1.
Step-6 :- end.
```

EN Queue

The insert operation to a Queue is called as EN Queue. In the array based Queue we can inserted a maximum of items as the size of array. ~~There~~ Therefore ~~are~~ to perform the NQ operation at first we need to check either the Queue is full or not.

The another condition is that after frequent insert and delete the rear may arrive at its maximum while the front is present after zero that means the vacant element are present behind the front. In such situation we need to pull down the data item toward zero element to make the vacancy at rear side. After that we adjust the rear and assign the value to that element. The algorithm is given below:-

```
Step 1:- Start enqueue (v)
Step 2:- If f of Q = 0 And R of Q = n-1
         then
    2.A:- Printf "Queue is full"
    2.B:- Return
Step 3:- If R of Q = -1 then
    3.A:- let R of Q = 0
    3.B:- let F of Q = 0
Step 4:- Else if Q.f > 0 and Q.R = n-1 Then
    4.A:- For i = Q.f To Q.R
    4.B:- let Q.val [i - Q.f] = Q.val [1]
    4.C:- End for
    4.D:- let Q.R = Q.R - Q.f
    4.E:- let Q.f = 0
Step 5:- else
    5.A:- let Q.R = Q.R + 1
Step 6:- end of if
Step 7:- let Q.val [Q.R] = V
Step 8:- Return.
```

DE Queue

The delete operation from a Queue is called as de Queue. A value can be deleted only from the front position of the Queue. To perform the DQ operation 1st we need to check that either there is any value in the Queue or not. The next condition we need to check that if there is a single value in the Queue then it would be empty after deleting that value. Then after we incr-rease the front of Queue to assume the deletion of value. The algorithm is given below:-

```
Step 1:- Start dequeue
Step 2:- Declare variable v.
Step 3:- If Q.F = -1 Then
        3.A:- Print Queue is empty.
        3.B:- return 0
Step 4:- end of If
Step 5:- let v= Q.val [Q.F].
Step 6:- If Q.f = Q.R then
        6.A:- let Q.F =-1
        6.B:- let Q.R = -1
Step 7:- else
      7.A:- let Q.F = Q.F + 1
Step 8:- end of If
Step 9:- Return v.
```

Traversal of Queue

Before traversing the Queue

we need to check that it should not be empty then after we start a loop initializing the counter with front and going upto the rear of ~~ele~~ Queue. The algorithm is given below :-

```
Step-1 :-   Start display
Step-2 :-   If Q.F = -1 then
    2.A :-  Printf "Queue is empty"
    2.B :-  Return with 0
Step 3 :-   end of If
Step 4 :-   for i = Q.F to Q.R
Step 5 :-   Printf Q.Val [i]
Step 6 :-   Return
```

Circular Queue

Since a value can be inserted only from the rear point, if there are frequent insert and delete from a linear queue, it is suffered with a prob-lem that the vacant element reside behind the front and to set the rear at appropri-ate position we need to pull down the value, which may take a large processing time. To eliminate this problem a new kind of Queue is designed ~~it is~~ called as circular queue. In the circular design it is assumed that the zero also comes after n-1. So that if the front is greater than zero and rear is at n-1 the next rear could be zero. This assemption reduces the processing time taken in Queue Push down operation. Creation is as like the linear ~~operation~~ Queue.

En Queue in Circular Queue

As like the linear Queue we need to check three condition for full Queue, for empty Queue and for the vacant element behind front. But this time if the third condition is true we set the rear to the zero index. There is a shortcut way also that if the

Queue is not full we assign the rear with remainder of max. by rear + 1 which set it to appropriate position.

Algorithm of En Queue

```
Step 1 :- Start enqueue (v)
Step 2 :- If Q.F = 0 and Q.R = n-1 then
        2.A :- Print Queue is full
        2.B :- Return with 0
Step 3 :- end of If
Step 4 :- If Q.R = -1 then
        4.A :- let Q.R = 0
        4.B :- let Q.F = 0
Step 5 :- else if Q.F > 0 and Q.R = n-1
          then
        5.A :- let Q.R = 0
Step 6 :- else
        6.A :- let Q.R = Q.R + 1
Step 7 :- end of If
Step 8 :- let Q.val [Q.R] = v
Step 9 :- Return
```

DEQueue from Circular Queue

The dequeue operation from the circular Queue is started by checking the Queue that either it has value or not. Then after it is checked that either there is a single value ~~it~~ if both the front and rear are assigned with -1. Otherwise according to the position of front either it is ~~a~~ increased by one or assigned with zero. If it is present at the n-1 position. The algorithm is given Below:-

```
Step 1 :- Start dequeue()
Step 2 :- Declare variable V
Step 3 :- If Q.F = -1 then
    3.A :- Print Queue is empty
    3.B :- Return with 0
Step 4 :- let V = Q.Val[Q.F]
Step 5 :- If Q.F = Q.R then
    5.A :- let Q.F = -1
    5.B :- Q.R = -1
Step 6 :- If Q.F = n-1 then
    6.A :- let Q.F = 0
Step 7 :- else
~~Step 8 :~~ 7.A :- let Q.F = Q.F+1
Step 8 :- end of if
Step 9 :- end of if
Step 10 :- Return V.
```

Linked based Queue

As we know an array is a static structure which limits the no of items to be so stored. As well as it may cause misuse of memory. To ~~op~~ overcome ~~there~~ these drawbacks of array, we prefer to store Queue as linked list.

A linked Queue is a collection of nodes with two pointers named rear and front. The rear tracks the address of the node to which the new node would be attached while the front tracks the address of the node which would be deleted.

Creation of linked Queue

To create a linked list based Queue at first we need to define the structure of node. After then an another structure is defined with two pointer members of node type to represent the rear and front of the Queue. The algorithm is given below :-

Step1:- Start create
Step2:- Define a structure node with the members val and pointer link of node type.
Step3:- Define an another structure with two pointer of node type rear and front named Queue.
Step4:- Declare a variable Q of structure type.
Step5:- let Q.R = Null then
Step6:- let Q.F = Null
Step7:- end.

En Queue to Linked Queue

To enqueue a value to a linked queue, we allocate memory to a node type pointer. Assign its value part with the given value and the linked part with null. After that we check that if the Queue is empty then both the rear and front are assign with the address of

new node. Otherwise the linked part of rear node and the rear pointer are assign with the new node.
The algorithm is given below:-

```
Step 1:- Start enqueue
Step 2:- Declare pointer ptr of node
         type.
Step 3:- Allocate memory to ptr.
Step 4:- let val [Ptr] = v
Step 5:- let link of ptr = Null.
Step 6:- If rear of Q = Null then
      6.A:- let rear of Q = Ptr
      6.B:- Let front of Q = ptr.
Step 7:- else
~~8~~ 7.A :- let Link of rear = ptr
      7.B :- let
Step 8:- End of if
Step 9:- Return
```