Queue

A queue is a non-primitive linear data structure. It is a homogeneous collection of elements in which new elements are added at one end called the Rear end, and the existing elements are deleted from the other end called the Front end.

A queue is logically a First in First Out (FIFO) type of list.

Eg New customers got into the queue from the rear end, whereas the customers are serviced in the order in which they arrive at the service center. (ie First come First Serve (FCFS) type of Service)

* Suppose that at a service centre, t1 units of time is needed to provide a service to a single customer and on average a new customer arrives at service center in t2 units of time.

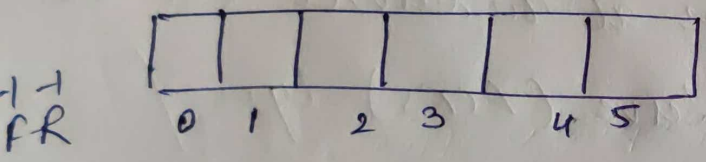
(1) If t1 < t2 then the service counter will be free for some time before a new customer arrives at the service counter. Hence there will be no customer standing at a particular time in the queue or the queue will be empty.

(2) If $t1 > t2$ then the service counter will be busy for some time even after the arrival of a new customer. Therefore, the new customers have to wait for some time. This situation gives rise to a queue.

(3) If $t1 = t2$ then as one customer leaves the counter other will arrive at the same instant, hence queue will be single element long.

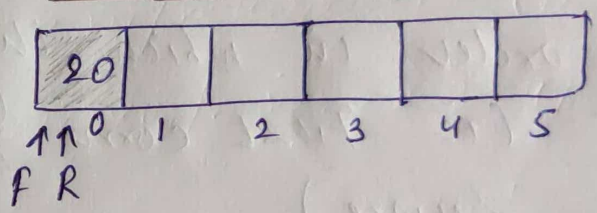
The following figures show queue graphically during insertion operation:

R=-1 and F=-1



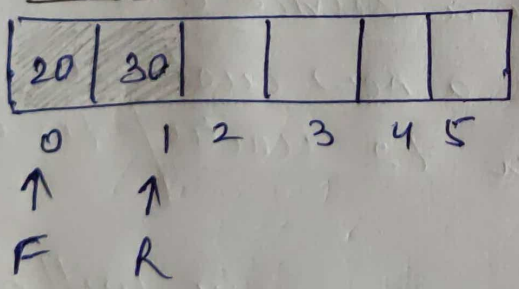
(a) Empty Queue

R=0 and F=0



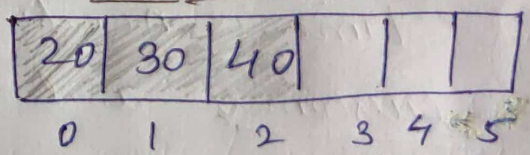
(b) Queue with one Element

R=1 and F=0



(c) Queue with two elements

R=2 and F=0



(d) Queue with three elements

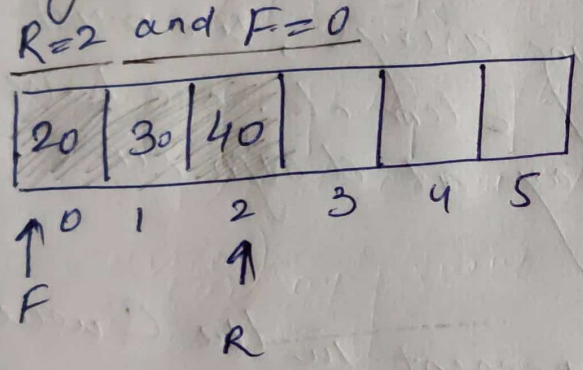
Whenever we insert an element in the queue, the value of Rear is incremented by one ie

$$Rear = Rear + 1$$

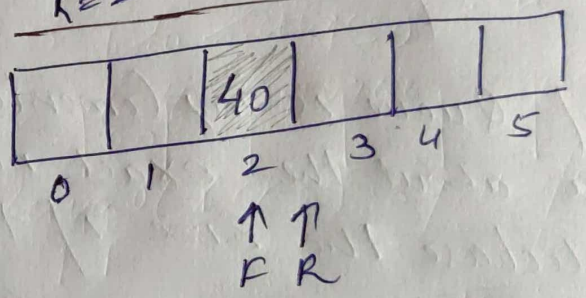
Note during the insertion of the first element in the queue we always increment the front by one ie

$$Front = Front + 1$$

The following figure shows queue graphically during deletion operation.

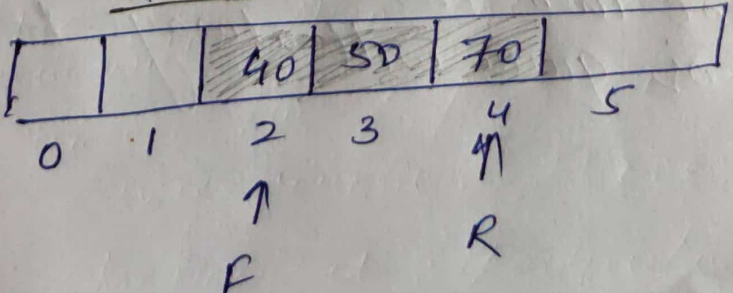


(a) Queue with three elements



(b) Two elements deleted from front (20 & 30)

* If we insert two elements in the queue, the queue will look like:



* Whenever an element is removed or deleted from the queue the value of front is incremented by one ie.

$$front = front + 1$$

Implementation of Queue

Queues can also be implemented in two ways:
(a) Static implementation (using arrays)
(b) Dynamic implementation (using pointers)

If Queue is implemented using arrays, we must be sure about the exact number of elements. we have to declare the size of the array at design time or before the processing starts.

Total number of elements present in the Queue, when implemented using arrays:

$$front - rear + 1$$

If rear < front Then there will be no elements in the queue or queue will always be empty.

operations on a Queue

The basic operations that can be performed on queue are:

(a) To insert an element in a queue.
(b) To delete an element from the queue.

Algorithms for Insertion and Deletion in Queue (Using arrays)

Let Queue be the array of some specified size say MAXSIZE.

Algorithm for Insertion in a Queue

(A) QINSERT(QUEUE[MAXSIZE], ITEM)

```
Step1 : Initialization
        Set Front = -1
        set Rear = -1
Step2 : Repeat step 3 to until Rear < MAXSIZE-1
Step3 : Read item
Step4 : If front == -1 then
            front = 0
            Rear = 0
        else
            Rear = Rear + 1
        EndIf
Step5 : Set Queue[Rear] = item
Step6 : Print, Queue Overflow
```

(B) Algorithm for Deletion From a Queue

QDELETE (QUEUE[MAXSIZE], ITEM)

Step 1 : Repeat step 2 to 4 until front >=0

Step 2 : set item = queue[front]

Step 3 : If front == rear
Set front = -1
Set Rear = -1

Else
front = front + 1

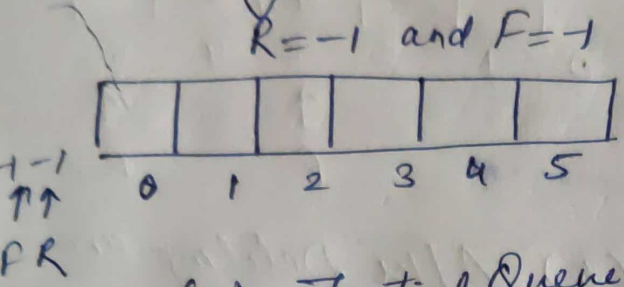
EndIf

Step 4 : Print, "No Deleted is ", Item

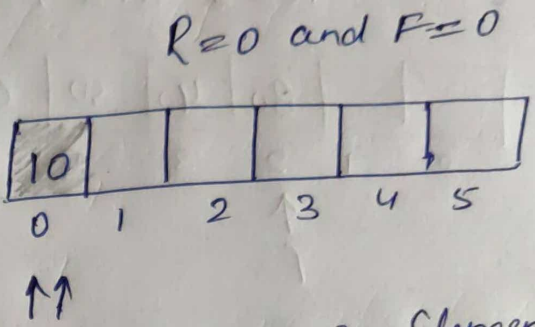
Step 5 : Print, "queue is Empty"

Limitations of Simple Queue

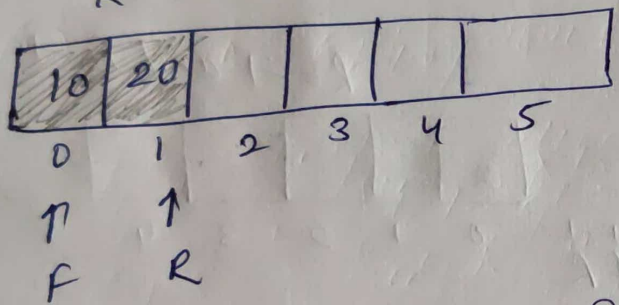
There are certain problems associated with a simple queue when queue is implemented using arrays. Consider an example of a simple queue Q[6], which is initially empty.



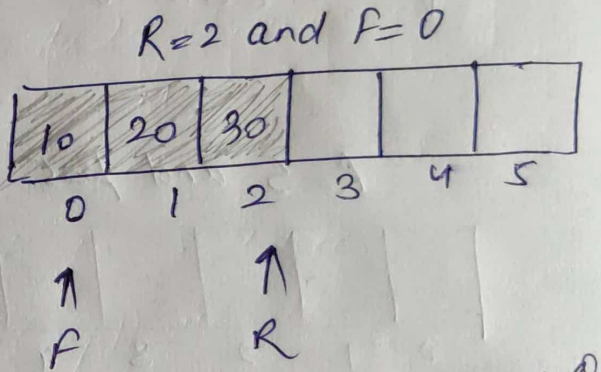
(a) Initial Queue



(b) One Element Queue

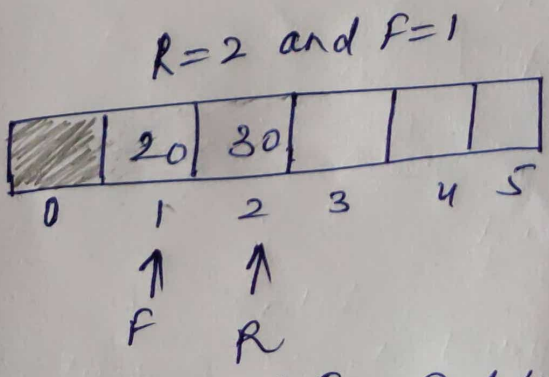


(c) Two Element Queue

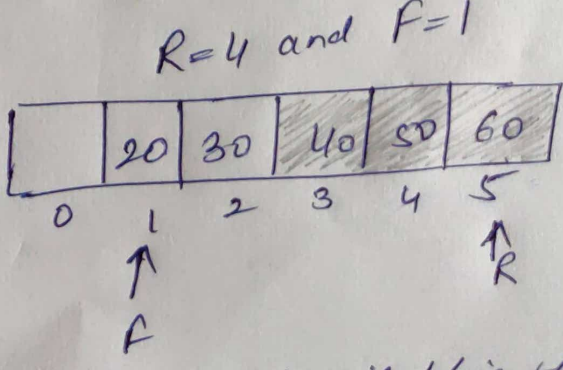


(d) Three Element Queue

Upto now there is no problem, the problem arises when we delete an element from the queue.



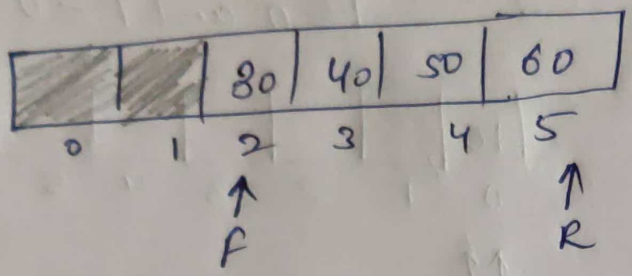
(e) After Deleting an Element



(f) After Adding three elements

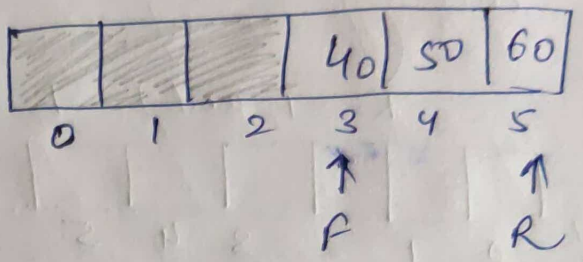
If we further delete items to the queue, the queue looks like as follows:

R=5 and F=2



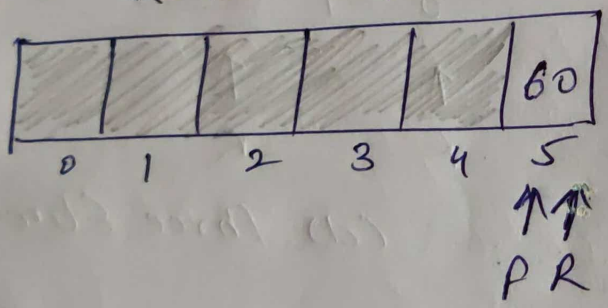
(g) After Deleting an Element

R=5 and F=3



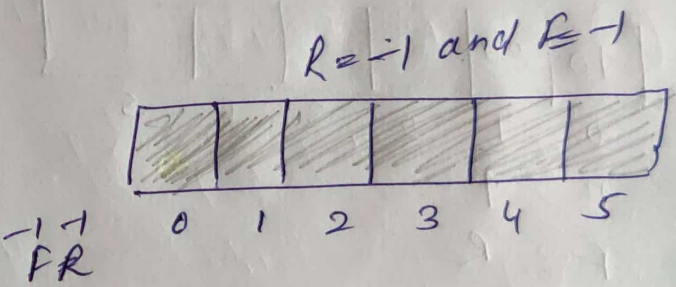
(h) After Deleting an Element.

R=5 and F=5



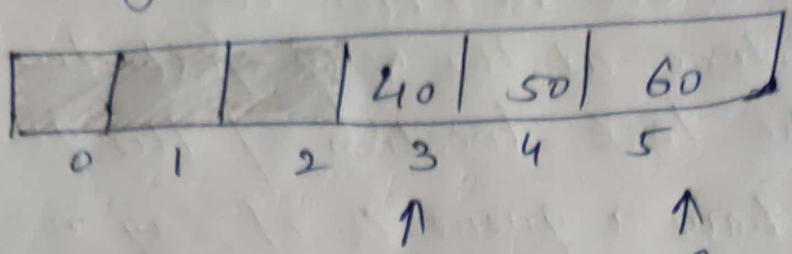
(i) After Deleting two Elements.

R=-1 and F=-1



(j) After Deleting an Element

In Linear Queue implemented using arrays if the below figure situation occurs. If we attempts to



add more elements, the elements can't be inserted. because in a queue the new elements are always added from the rear end. and rear here indicates to last locations of the array. Hence though the space is there in the queue we are not able to insert the elements. The it shows "Queue is Full" though it is empty.

* To remove this problem, the first and obv solution which comes in our mind is whenever an element is deleted, shift all afterward elements to the left by one position.

But if the queue is too large say 5000 elements it will be a difficult job to do and time consuming too.

* To remove this problem we use circular queue

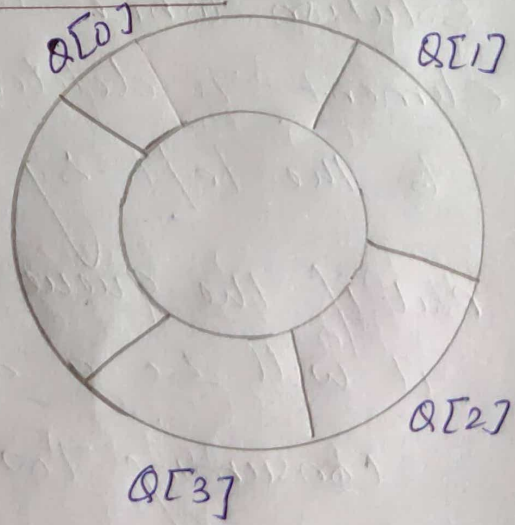
Circular Queue

A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location of the queue is full.

* Suppose if we have a queue Q of say n elements, then after inserting an element last (i.e. $n-1^{th}$) location of the array the next element will be inserted at the very first location of the array. (i.e. location with subscript 0) of the array.

It is possible to insert new elements, if and only if those locations (slots) are empty

* A circular queue is one in which the first element comes just after the last element.



* A circular queue overcomes the problem of untilized space in linear queue implemented as queue.
* A circular queue also has a front and Rear to keep track of the elements to be deleted and inserted and therefore to maintain the unique characteristic of the

Circular Queue

queue.

The below assumptions are made:

1) Front will always be pointing to the first element.

2) If Front = Rear the queue will be empty.

3) Each time a new element is inserted into the queue the Rear is incremented by one.

$$Rear = (Rear + 1) \% MAXSIZE$$

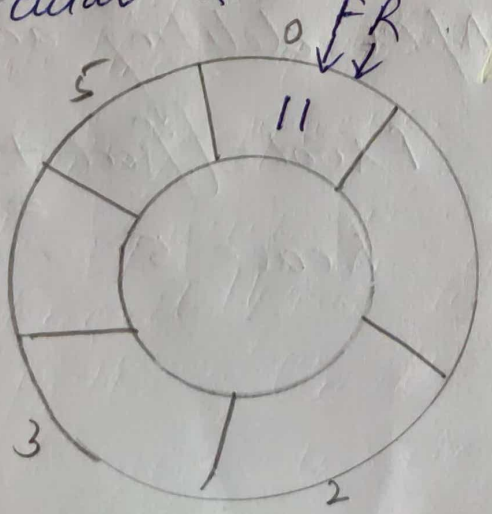
4) Each time an element is deleted from the queue the value of front is incremented by one.

$$front = (front + 1) \% Maxsize$$

Representation of Circular Queue

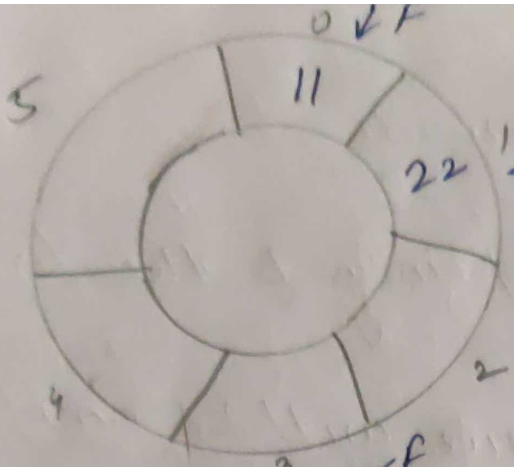
A Circular Queue is representation is shown:

(a)



MAX = 6
FRONT = REAR = 0

(b)



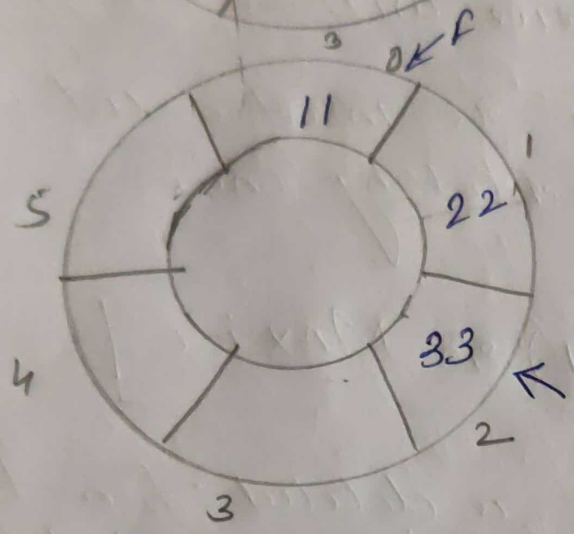
FRONT = 0

Real = (Real + 1) % 6

= (0 + 1) % 6

= 1

(c)



front = 0

Next location :

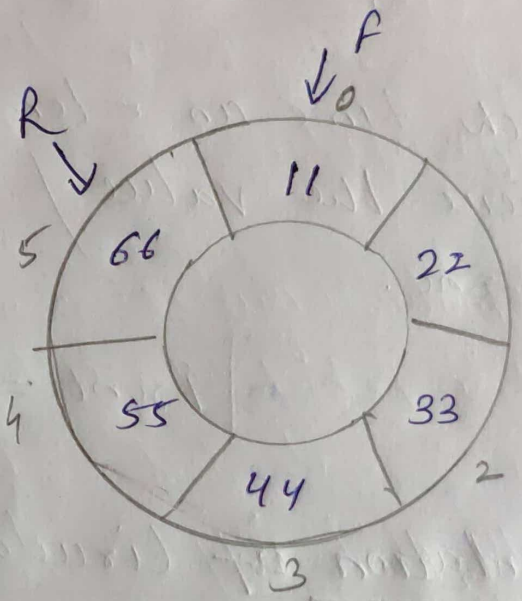
Real = (Real + 1) % 6

= (1 + 1) % 6 = 2

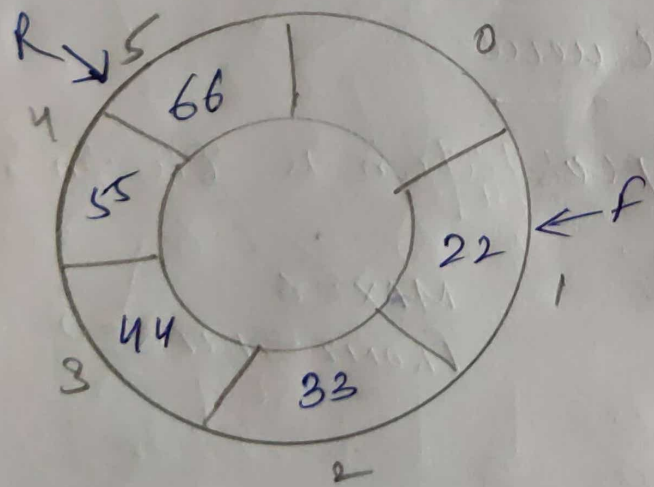
(d) After insertion of three more element the status of circular Queue.

front = 0

Real = 5



(e)



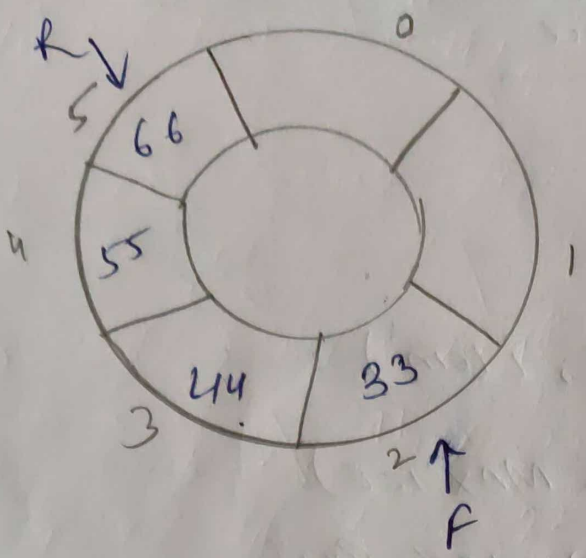
After deleting an element

front = (front + 1) % 6

= 1

Real = 5

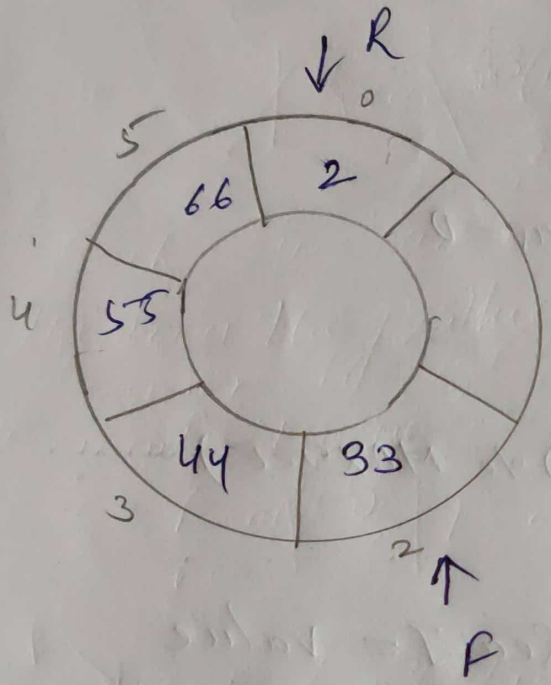
(f)



After deleting another element

front = (front + 1) % 6
= (1 + 1) % 6
= 2

Rear = 5



To insert new elements into the circular Queue

front = 2

Rear = (Rear + 1) % 6
= (5 + 1) % 6
= 0

Algorithm to Insert and Delete an Element from Circular Queue

Insertion

QINSERT(Queue[MAXSIZE], item)

1. If (front == (rear+) % MAXSIZE)
   write Queue Overflow and Exit

Else

If (front == -1)

Set front = rear = 0

else

rear = ((rear+1) % MAXSIZE)

[Assign value] Queue[rear] = value

[End if]

Deletion

QDELETE (Queue[MAXSIZE], item)

1. If (front = -1)
   Write Queue Underflow and Exit.

   else : Item = Queue[front]

   if (front == Rear) # After deleting single element both pointer reset to -1

   Set front = -1

   Set Rear = -1

   Else: front = (front + 1) % MAXSIZE

   [End if Structure]

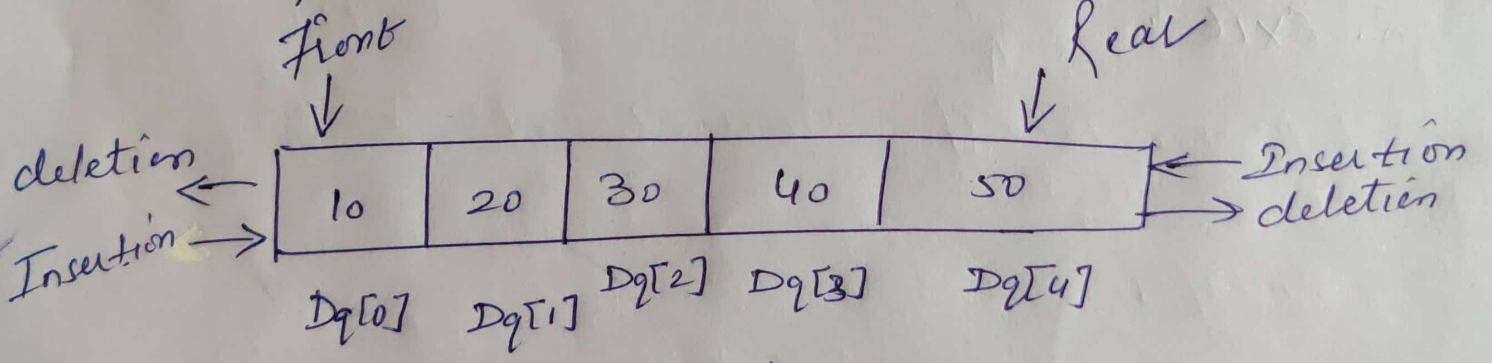
2. Exit

Double Ended Queues (DE QUE)

It is also a homogeneous list of elements in which insertion and deletion operations are performed from both the ends ie we can insert elements from the rear end or from the front ends. Hence it is called double ended queue. It is commonly referred to as deque.

There are two types of deques. These two types are due to the restrictions put to perform either the insertions or deletions only at one end. They are:

(1) Input restricted deque.
(2) Output restricted deque.



A deque

Since both insertion and deletion are performed from either end, it is necessary to design an algorithm to perform the following four operations.

1) Insertion of an element at the rear end of the queue.
2) Deletion of an element from the front end of the queue.
3) Insertion of an element at the front end of the queue.
4) Deletion of an element from the rear end of the queue.

Functions to carry out these four operations given below :

(1) Insertion of an element at the rear end of the queue.

```
void dqinsert_rear( int q[10], int front, int rear,
                    int item, int maxsize)
{
    if (rear == (MAXSIZE-1))
        printf("Queue is full");
    else
        rear = rear + 1;
    q[rear] = item;
}
```

1) Insertion of an element at the rear end of the queue.
2) Deletion of an element from the front end of the queue.
3) Insertion of an element at the front end of the queue.
4) Deletion of an element from the rear end of the queue.

Functions to carry out these four operations given below:

(1) Insertion of an element at the rear end of the queue.

```
void dqinsert_rear( int q[10], int front, int rear,
                    int item, int maxsize)
{
    if (rear == (MAXSIZE-1))
        printf("Queue is full");
    else
        rear = rear + 1;
    q[rear] = item;
}
```

(2) Deletion of an element from the front end of the queue.

```
void dqdelete_front(int q[10], int front, int rear,
                    int item)
{
    if(front == rear)
        printf("Queue is Empty");
    else
        front = front+1;
    item = q[front];
    printf("Element deleted is %d", item);
```

(3) Inseption of an element at the front end of the queue

```
void dqinsert_front(int q[10], int front, int rear, int
                    item)
{
    if (front == 0)
        printf("Queue is full");
    else
        front = front-1;
    q[front] = item;
}
```

4) Deletion of an element from the Rear end of the queue

```
void dqdelete_rear (int q[10], int front, int rear,
                    int item)
{
    if (front == rear)
        printf("Queue is empty");
    else
        rear = rear - 1;
    item = q[item];
    printf("Element deleted is = %d", item);
}
```

5) function to display the contents of a queue.

```
void dq_display (int q[10], int front, int rear)
{   if (front <= rear)
    {   printf("Status of the queue");
        for (i = front; i <= rear; i++)
            printf("%d", dq[i]);
    }
    else printf("Queue is empty");
}
```

Priority Queue

A priority queue is a collection of elements such that each element has been assigned a priority and the order in which elements are deleted and processed comes from the following rules:

(a) An element of higher priority is processed before any element of lower priority.

(b) Two elements with the same priority are processed according to the order in which they were added to the queue.

There can be two types of implementations of priority queue:

(a) Ascending Priority Queue (b) Descending Priority Queue

Ascending Priority Queue: A collection of items into which items can be inserted arbitrarily and from which only the smallest item can be removed is called Ascending Priority Queue.

Descending Priority Queue: allows only the largest item to be removed.

* The priority queue is a data structure in which intrinsic ordering of the elements determine the result of its basic operation.

Insertion: The insertion in priority queues is the same as in non-priority queues.

Deletion: Deletion requires a search for the element of higher priority and deletes the element with highest priority.

Applications of Queues

(1) Round Robin technique for processor scheduling is implemented using queues.

(2) All types of customer services (like railway ticket reservation) center softwares are designed using queues to store customers information.

(3) Printer Server routines are designed using queues. A number of users share a printer using print Server, the printer server then spools all the jobs from all the users, to the server's hard disk in a queue. From here jobs are printed one by one according to their number in the queue.

PROBLEM 1

Consider the following queue of characters, where queue Q is a circular array which is allocated 5 memory cells :

Front = 2, Rear = 4, Q : _, P, Q, _, _

Describe the following operations on queue.

(a) R is added to the queue

SOLUTION

Front = 4, Rear = 5, Q : _, P, Q, R, _

Rear is increased by 1.

(b) Two letters are deleted.

SOLUTION

The two letters, P and Q, are deleted.

Front = 4, Rear = 5, Q : _, _, _, R, _

(c) S, T and U are added to queue.

SOLUTION

S, T and U are added to the rear of the queue. Since S is placed in the last memory cell of queue Q, T and U are placed in the first two memory cells. This gives

Front = 4, Rear = 2, Q : T, U, _, R, S

(d) Three letters are deleted

SOLUTION

Two front letters R, S, and T are deleted, leaving

Front = 2, Rear = 3, Q : _,U, _, _, _

(e) V is added to the queue.

SOLUTION

Front = 2, Rear = 3, Q : _,U, V, _, _

PROBLEM 4

Consider the following circular queue capable of accommodating maximum six elements.

Front = 2, Rear = 4

Queue : _, L, M, N, _, _,

Describe the queue as the following operations take place

(a) Add O, (b) Add P (c) Delete two letters

(d) Add Q, R, S (e) Delete one letter.

SOLUTION

(a) front = 2, Rear = 5
Queue : _, L, M, N, O, -

(b) front = 2, Rear = 6
Queue : _, L, M, N, O, P

(c) front = 4, Rear = 3
Queue : _, _, _, N, O, P

(d) front = 5, Rear = 3
Queue : Q, R, S, N, O, P

(e) front = 5, Rear = 3
Queue : Q, R, S, _, O, P

PROBLEM 5

Consider the following queue of characters, where QUE is a circular array which is allocated six memory cells :

FRONT = 2, REAR = 4 *QUEUE : _ A, C, D, E, _*

Describe the queue as the following operations take place.

(a) *F is added to the queue.*
(b) *two letters are deleted.*
(c) *K, L and M are added to the queue.*
(d) *two letters are deleted.*
(e) *R is added to the queue.*
(f) *two letters are deleted.*
(g) *S is added to the queue.*
(h) *two letters are deleted.*
(i) *one letter is deleted.*
(j) *one letter is deleted.*

SOLUTION

(a) F is added to the rear of the queue, yielding

FRONT = 2, REAR = 5 QUEUE : _ A, C, D, F, _

Note that REAR is increased by 1.

CHAPTER 6

(*b*) The two letters, A and C, are deleted, leaving

FRONT = 4, REAR = 5 QUEUE : _, _, _, D, F, _

Note that FRONT is increased by 2.

(*c*) K, L and M are added to the rear of the queue. Since K is placed in the last memory cell of QUEUE, L and M are placed in the first two memory cells. This yields

FRONT = 4, REAR = 2 QUEUE : L, M, _ , D, F, K

Note that REAR is increased by 3 but the arithmetic is modulo 6.

REAR = 5 + 3 = 8 = 2 (mod 6)

(*d*) The two front letters, D and F are deleted, leaving

FRONT = 6, REAR = 2 QUEUE : L, M, _, _, _, K

(*e*) R is added to the rear of the queue, yielding

FRONT = 6, REAR = 3 QUEUE : L, M, R, _, _, K

(*f*) The two front letters, K and L, are deleted, leaving

FRONT = 2, REAR = 3 QUEUE : _, M, R, _, _, _

Note that FRONT is increased by 2 but the arithmetic is modulo 6:

FRONT = 6 + 2 = 8 = 2 (mod 6)

(*g*) S is added to the rear of the queue, yielding

FRONT = 2, REAR = 4 QUEUE : _, M, R, S, _, _

(*h*) The two front letters, M and R, are deleted, leaving

FRONT = 4, REAR = 4 QUEUE : _, _, _, S, _, _

(*i*) The front letter S is deleted. Since FRONT = REAR, this means that the queue is empty, hence we assign NULL to FRONT and REAR. Thus

FRONT = 0, REAR = 0 QUEUE : _, _, _, _, _, _

(*j*) Since FRONT = NULL, no deletion can take place. That is, underflow has occurred.

PROBLEM 7

Consider the following deque of characters where DEQUE is a circular array which is allocated six memory cells :

LEFT = 2, RIGHT = 4, DEQUE : , A, C, D, _

Describe the deque while the following operations take place.

(a) *F is added to the right of the deque.*
(b) *Two letters on the right are deleted.*
(c) *K, L and M are added to the left of the deque.*
(d) *One letter on the left is deleted.*
(e) *R is added to the left of the deque.*
(f) *S is added to the right of the deque.*
(g) *T is added to the right of the deque.*

SOLUTION

(a) F is added on the right, yielding

LEFT = 2 RIGHT = 5 DEQUE : , A, C, D, F

Note that RIGHT is increased by 1.

(b) The two right letters, F and D, are deleted, yielding

LEFT = 2, RIGHT = 3 DEQUE : , A, C, _, _

Note that RIGHT is decreased by 2.

(c) K, L and M are added on the left. Since K is placed in the first memory cell, L is placed in the last memory cell and M is placed in the next – to – last memory cell. This yields

LEFT = 5, RIGHT = 3 DEQUE : K, A, C, _ , M, L

Note that LEFT is decreased by 3 but the arithmetic is modulo 6:

LEFT = 2*3 - 1 = 5 (mod 6)

(*d*) The left letter, M, is deleted, leaving

LEFT = 6, RIGHT = 3 DEQUE : K, A, C, _, _ , L

(*e*) R is added on the left, yielding

LEFT = 5, RIGHT = 3 DEQUE : K, A, C, _ , R, L

(*f*) S is added on the right, yielding

LEFT = 5, RIGHT = 4 DEQUE : K, A, C, S, R, L

(*g*) Since LEFT = RIGHT + 1, the array is full, and hence T cannot be added to the deque. That is, overflow has occurred.