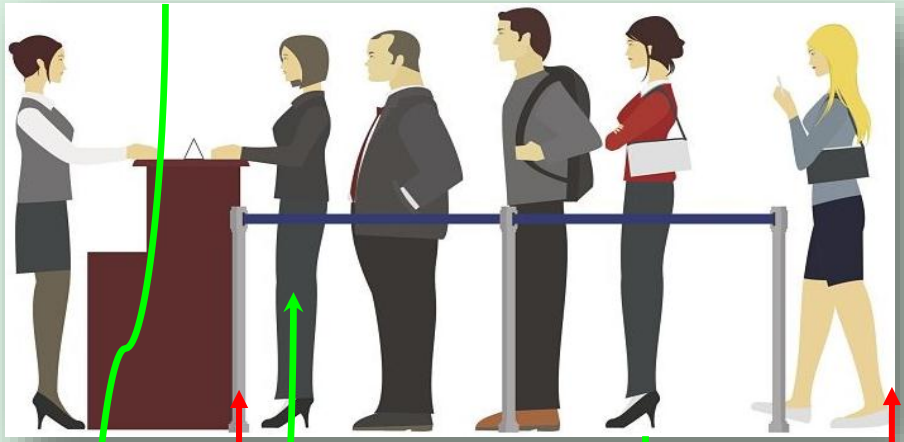
QUEUE

Rajiv.S.Bal
M.Tech(CSE),Ph.D(Conti.)
Lecturer

shopping mall Or Bank

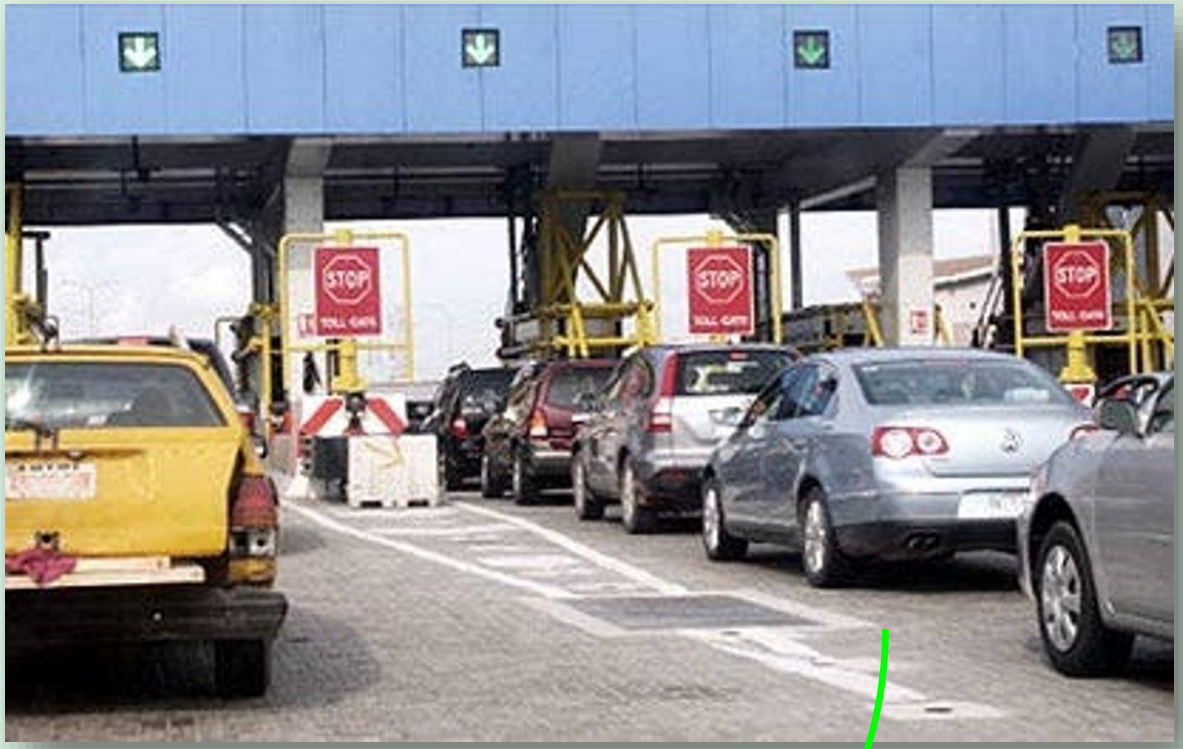


Counter

QUEUE

First Entry (in) and First Exit (out)

QUEUE



QUEUE

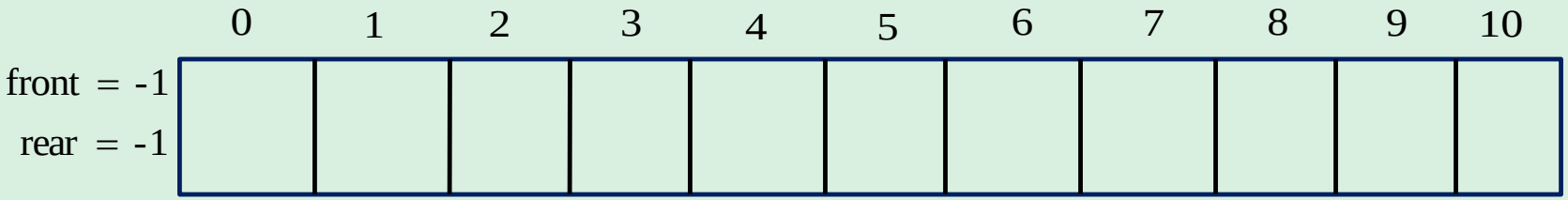
What is Queue ?

- Queue is a linear data structure where the first element is inserted from one end called REAR and deleted from the other end called as FRONT.
- Front points to the beginning of the queue and Rear points to the end of the queue.
- Queue follows the FIFO (First - In - First Out) structure.
- According to its FIFO structure, element inserted first will also be removed first.
- In a queue, one end is always used to insert data (enqueue) and the other is used to delete data (dequeue), because queue is open at both its ends.
- The enqueue() and dequeue() are two important functions used in a queue.

Example of Queue ?

A Queue is a linear list of elements in which insertions can take place at one end called rear and deletions can take place only at other end called front.
Thus, queues are called FIFO (First - In - First –Out).

If front = -1 and rear = -1 is called empty queue.



The queue abstract data type (ADT) defines a collection that keeps objects in a sequence, where:

- *element access and deletion are restricted to the first element in the sequence, which is called the front of the queue, and*
- *element insertion is restricted to the end of the sequence, which is called the rear of the queue.*

In a linear queue, the **traversal through the queue is possible only once,i.e.,once an element is deleted, we cannot insert another element in its position. This disadvantage of a linear queue is overcome by a circular queue, thus saving memory.**

This restriction enforces the rule that items are inserted and deleted in a queue according to the first-in first-out (FIFO) principle. The queue supports the following two fundamental methods or operations :

- ***Enqueue(e): Insert element e at the rear of the queue (end).***
- ***Dequeue(): Remove and return from the queue the object at the front; an error occurs if the queue is empty.***

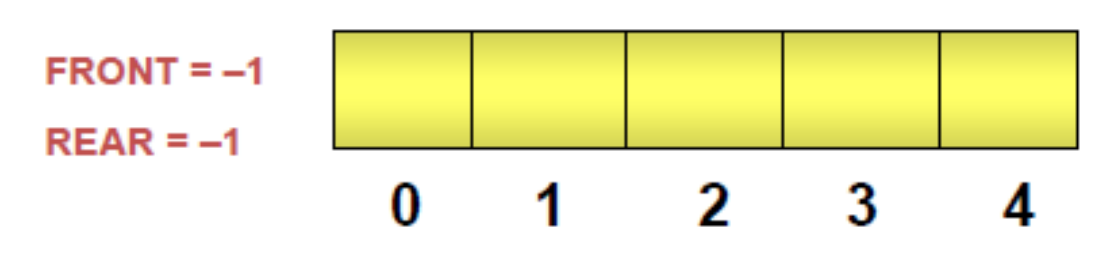
Linear Queue Implementation

A Queue may be represented or implemented in memory in two ways :

1. Contiguous linear queue: the queue is implemented as an array (static memory allocation).
2. Linked linear queue: pointers and dynamic memory allocation is used to implement the queue.

Implementing a Queue Using an Array

Now , we implement a queue using an array that stores these request numbers in the order of their arrival.



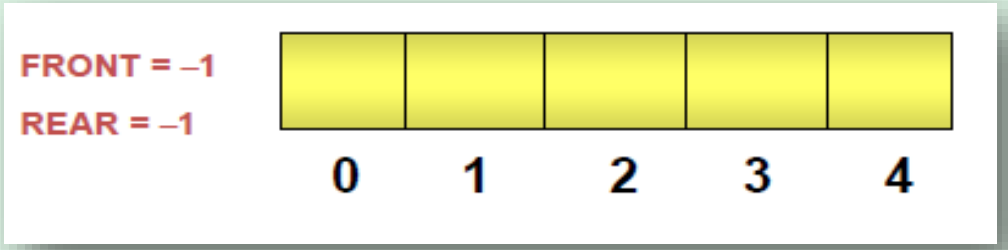
To insert a request number, we need to perform the following steps:

- Increment the value of REAR by 1.
- Insert the element at index position REAR in the array.

Let us now insert request numbers in the following queue.

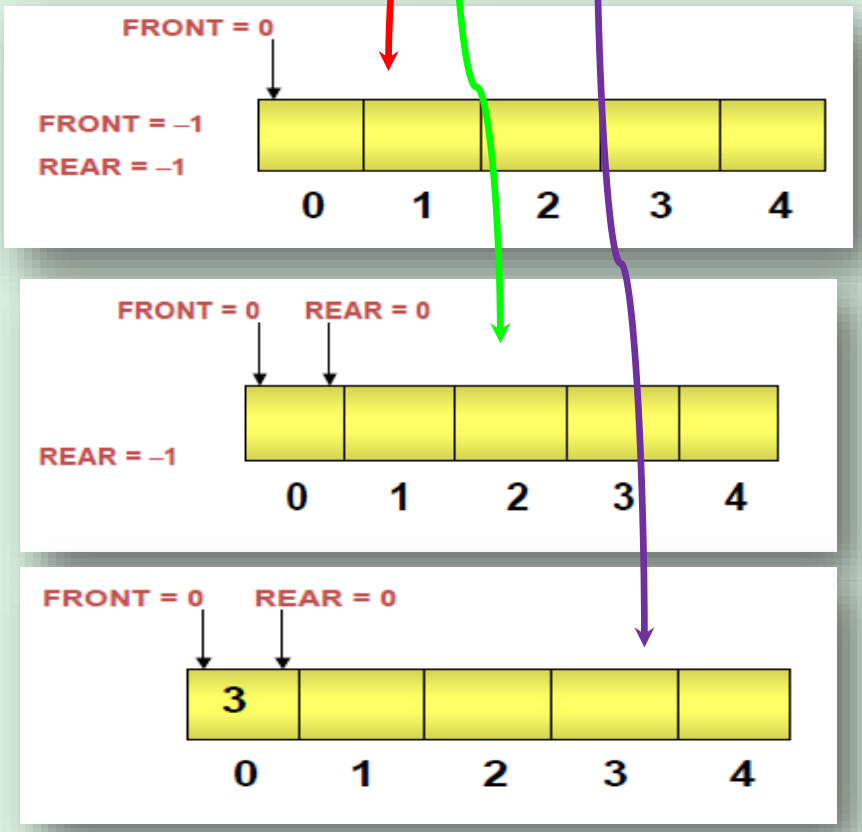
Algorithm:

1. IF(FRONT==-1)
1.1 FRONT = 0
2. REAR = REAR+1
3. QUEUE[REAR] = ITEM



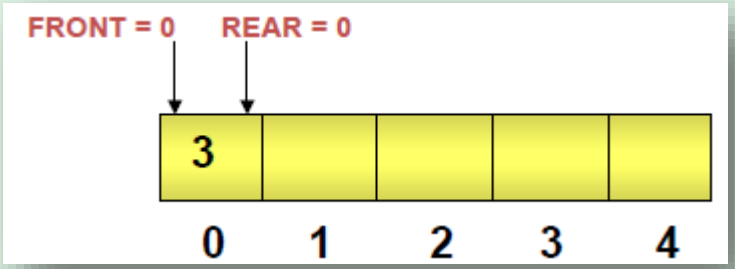
Empty Queue

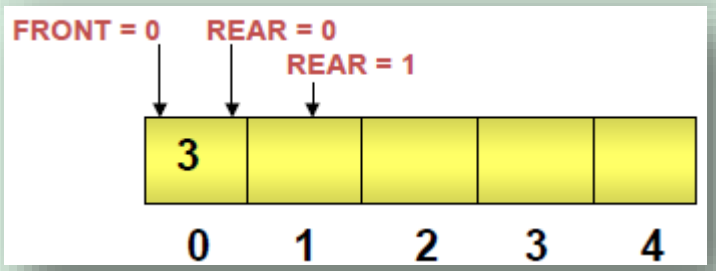
Now, we insert an ITEM = **3**

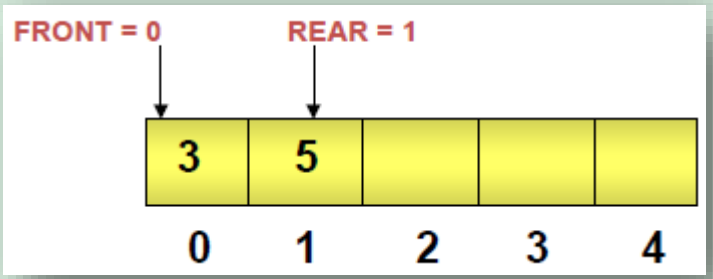


Insertion complete

Next, we insert an ITEM = **5**

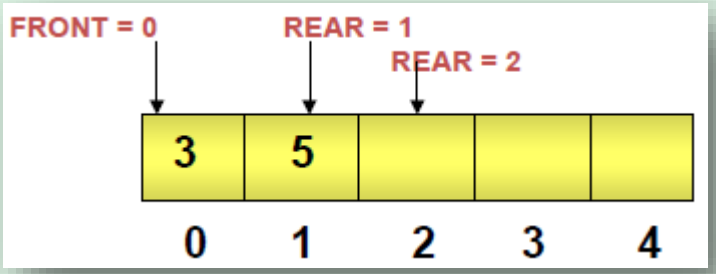


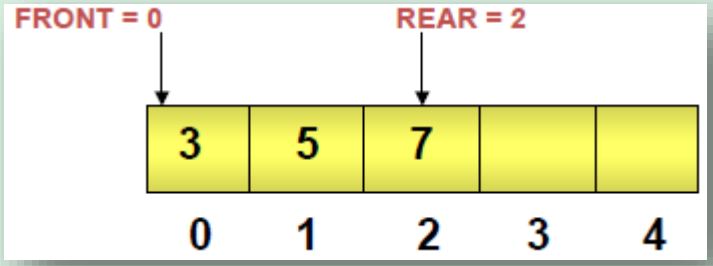




Insertion complete

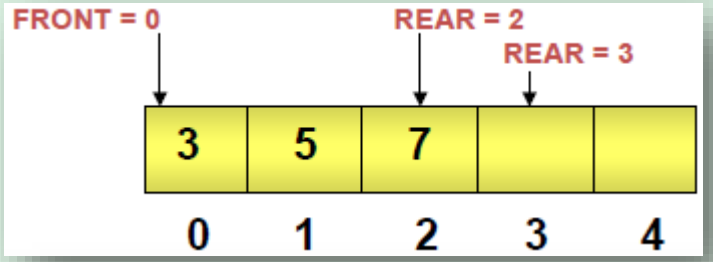
Next, we insert an ITEM = **7**

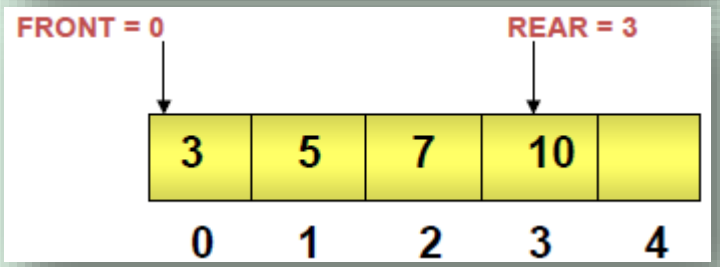




Insertion complete

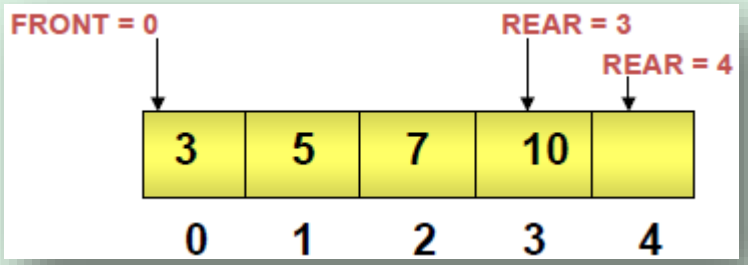
Next, we insert an ITEM = **10**

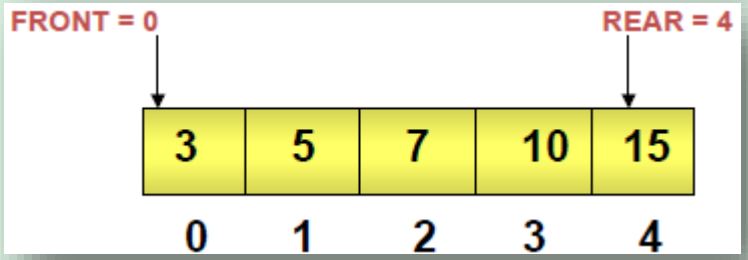




Insertion complete

Next, we insert an ITEM = **15**



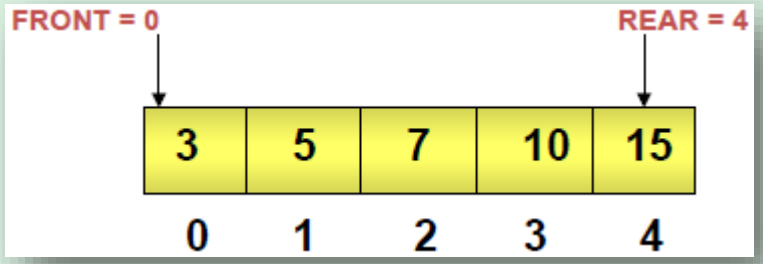


Insertion complete

Algorithm to insert data in a Queue

FRONT = -1, REAR = -1, TO REPRESENT QUEUE IS EMPTY
Algorithm INSERT(QUEUE[N],FRONT,REAR,ITEM)
{
/*QUEUE is an array of size N ,ITEM is element to be inserted.*/
1. if (REAR == N-1)
1.1 Print “OVERFLOW”
else
1.1 if (FRONT == -1)
1.1.1 FRONT = 0
1.2 REAR = REAR+1
1.3 QUEUE[REAR] = ITEM
}

To deleted an ITEM from the queue once they get processed.



1.If(FRONT == – 1)

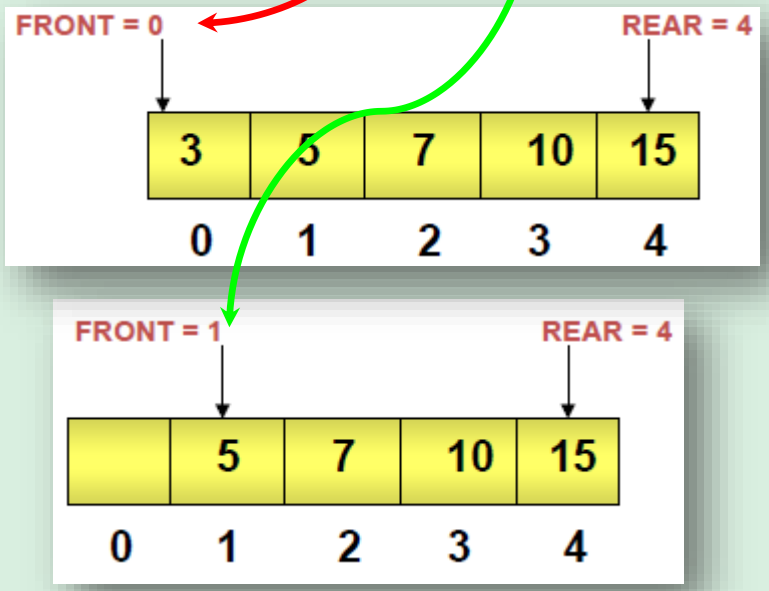
1.1 print “QUEUE EMPTY”

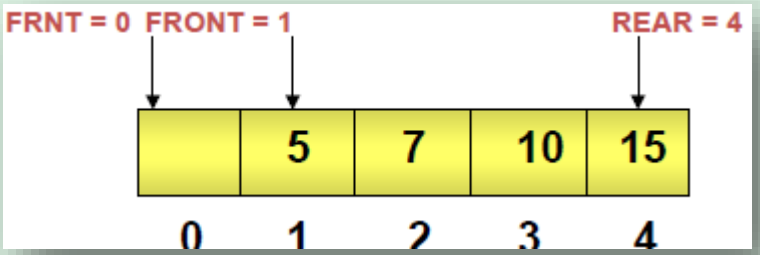
2. else

2.1 ITEM = QUEUE[FRONT]

2.2 FRONT = FRONT +1

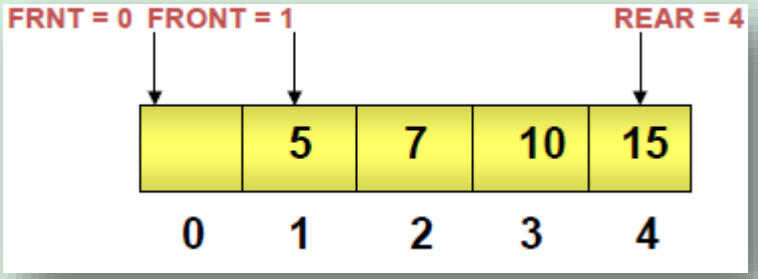
Now, we deleted an ITEM = 3 from the queue.

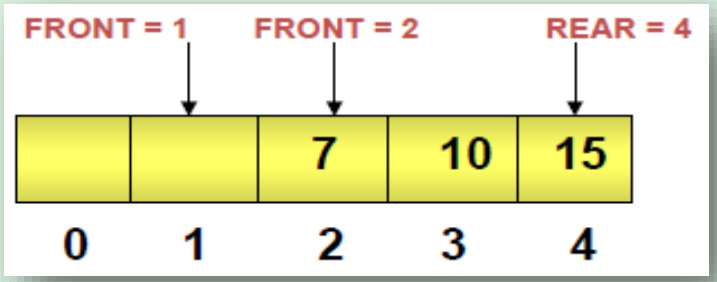




Delete operation complete

Next, we deleted an ITEM = 5 from the queue.





Delete operation complete

Next, we deleted an ITEM = 7 from the queue and so on.

```
Algorithm DELETE(QUEUE[N],ITEM,FRONT,REAR)
{
1. if (( FRONT == – 1 ) || (FRONT == REAR+1))
       1.1 Print “QUEUE EMPTY”
2. else
       2.1 ITEM = QUEUE[FRONT]
       2.2 FRONT = FRONT +1
}
```

Applications of Queues

Queues offer a lot of practical applications, such as:

- Printer Spooling
- CPU Scheduling
- Mail Service
- Keyboard Buffering
- Elevator

Example: Program to implement a queue using Array

```
#include <stdio.h>
#define MAX 50
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
    int choice;
    while (1)
    {
        printf("1.Insert \n");
        printf("2.Delete\n");
        printf("3.Display \n");
        printf("4.Exit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
```

```
 switch (choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(1);
            default:
            printf("Inavlid choice \n");
        } /*End of switch*/
    } /*End of while*/
} /*End of main()*/
```

```
insert()
{
    int add_item;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
} /*End of insert()*/
```


```
delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Deleted Element is : %d\n", queue_array[front]);
        front = front + 1;
    }
} /*End of delete() */
display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
} /*End of display() */
```

What is Circular Queue ?

- In a circular queue, all nodes are treated as circular. Last node is connected back to the first node.
- Circular queue is also called as Ring Buffer.
- It is an abstract data type.
- Circular queue contains a collection of data which allows insertion of data at the end of the queue and deletion of data at the beginning of the queue.

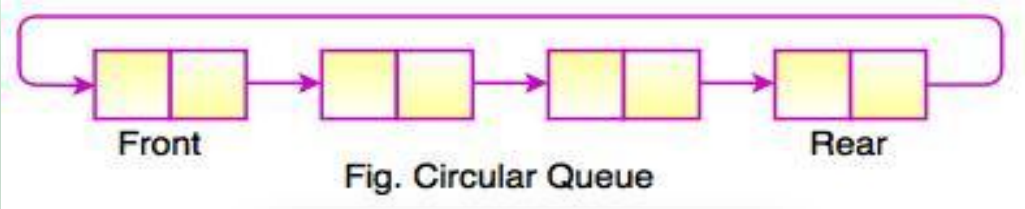


Fig. Circular Queue

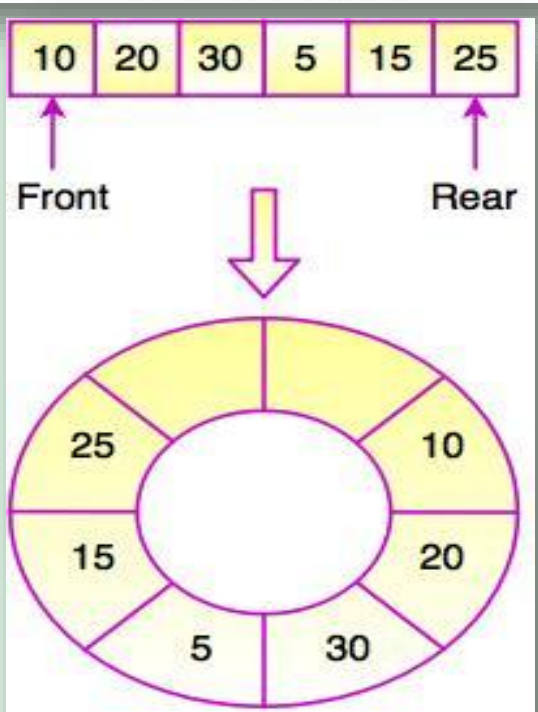


Fig. Circular Queue

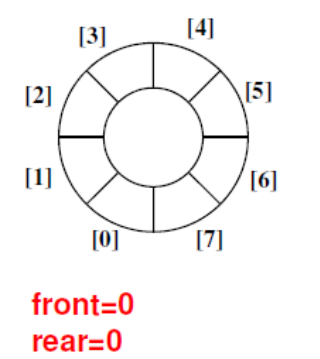
The above figure shows the structure of circular queue. It stores an element in a circular way and performs the operations according to its FIFO structure.

Operations in Circular Queue :

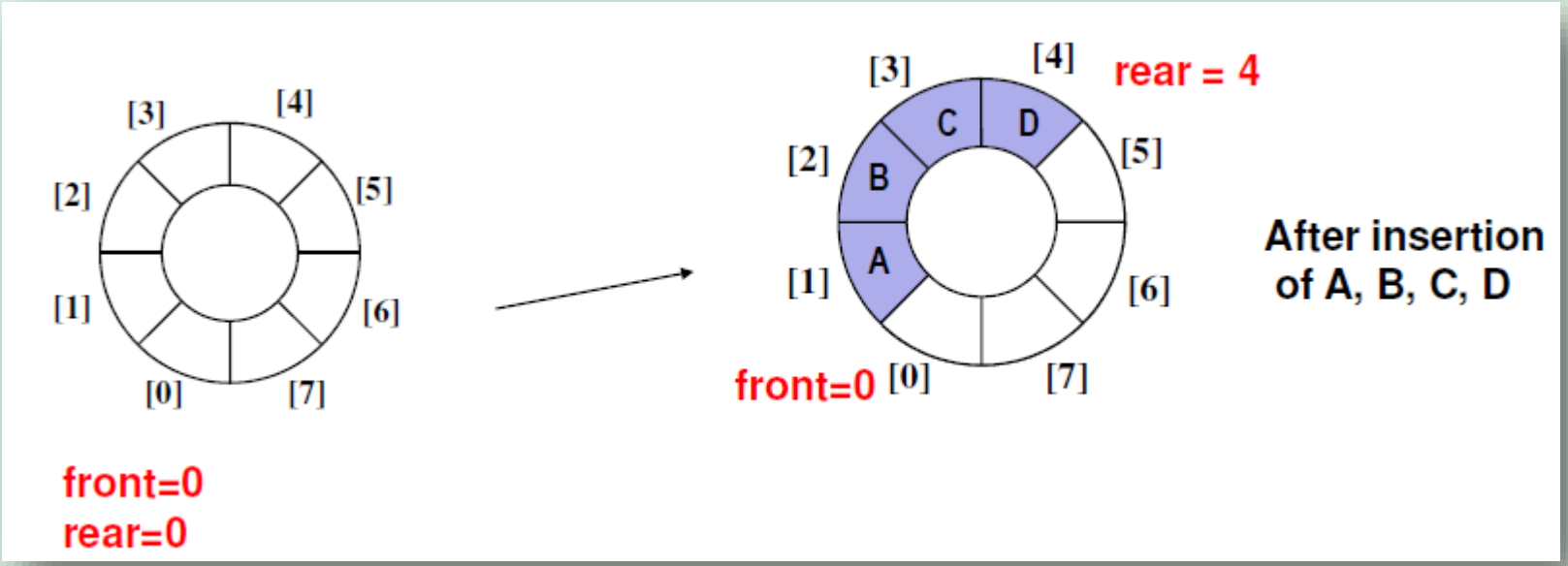
There are two main operations that can be performed on Circular Queue.

- Insert Operation
- Delete Operation

Insertion Operation in Circular Queue Example :



Queue Empty



Insert operation is used to insert an element into Circular Queue. In order to insert an element into Circular Queue first we have to check weather space is available in the Circular Queue or not. If Circular Queue is full then we can not insert an element into Circular Queue. If value of REAR variable is greater then or equal to SIZE – 1 and value of FRONT variable is equal to 0 then we can not insert an element into Circular Queue. This condition is known as **"Overflow"**.

If Queue is not overflow then we have to set the value of REAR variable and then we can insert an element into Circular Queue.

Algorithm of Insertion Operation :

For Insert Operation

Insert-Circular-Q(CQueue, Rear, Front, N, Item)

Here, CQueue is a circular queue where to store data. Rear represents the location in which the data element is to be inserted and Front represents the location from which the data element is to be removed. Here N is the maximum size of CQueue and finally, Item is the new item to be added. Initailly Rear = 0 and Front = 0.

1. If Front = 0 and Rear = 0 then Set Front := 1 and go to step 4.
2. If Front =1 and Rear = N or Front = Rear + 1

   then Print: “Circular Queue Overflow” and Return.
3. If Rear = N then Set Rear := 1 and go to step 5.
4. Set Rear := Rear + 1
5. Set CQueue [Rear] := Item.
6. Return

Algorithm of Deletion Operation :

Delete-Circular-Q(CQueue, Front, Rear, Item)

Here, CQueue is the place where data are stored. Rear represents the location in which the data element is to be inserted and Front represents the location from which the data element is to be removed. Front element is assigned to Item. Initially, Front = 1.

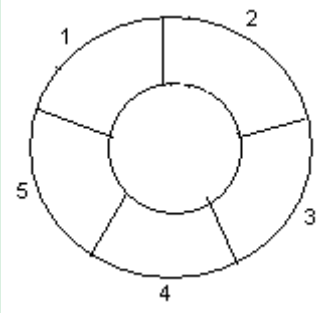
1. If Front = 0 then

   Print: “Circular Queue Underflow” and Return. /*..Delete without Insertion*/

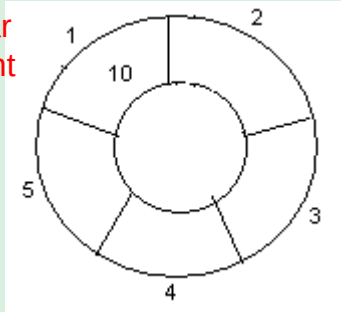
2. Set Item := CQueue [Front]
3. If Front = N then Set Front = 1 and Return.
4. If Front = Rear then Set Front = 0 and Rear = 0 and Return.
5. Set Front := Front + 1
6. Return.

Example: Consider the following circular queue with N = 5.

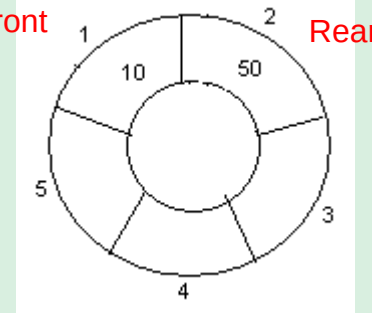
1. Initially, Rear = 0, Front = 0.



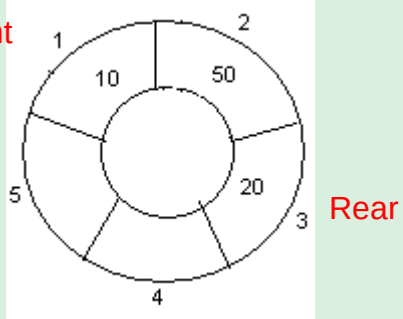
2. Insert 10, Rear = 1, Front = 1.



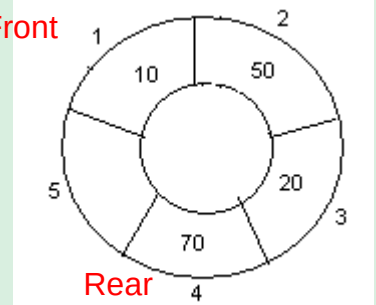
3. Insert 50, Rear = 2, Front = 1.



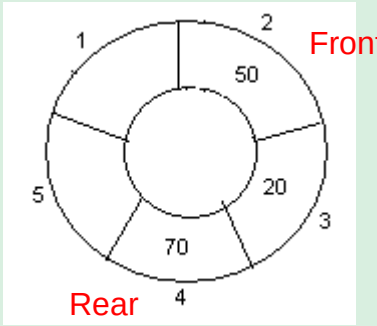
4. Insert 20, Rear = 3, Front = 0.



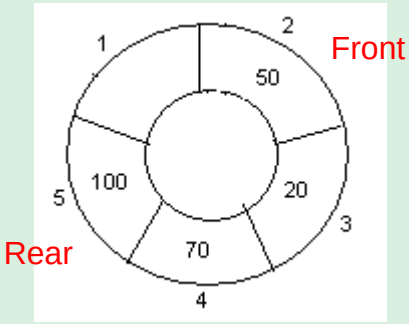
5. Insert 70, Rear = 4, Front = 1.



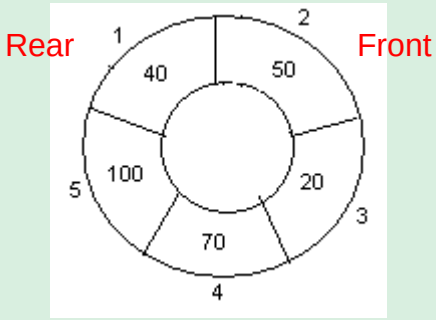
6. Delete front, Rear = 4, Front = 2.



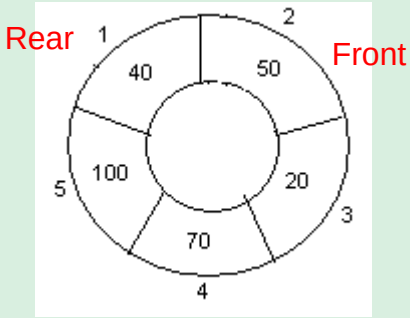
7. Insert 100, Rear = 5, Front = 2.



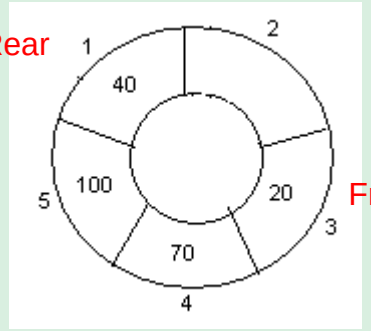
8. Insert 40, Rear = 1, Front = 2.



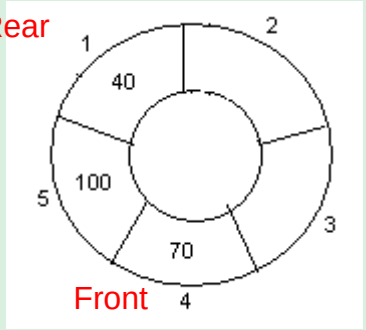
9. Insert 140, Rear = 1, Front = 2.
   As Front = Rear + 1, so Queue overflow.



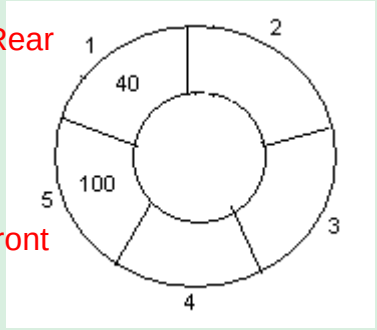
10. Delete front, Rear = 1, Front = 3.



11. Delete front, Rear = 1, Front = 4.



12. Delete front, Rear = 1, Front = 5.



Delete operation is used to delete an element from Circular Queue. In order to delete an element from Circular Queue first we have to check weather Circular Queue is empty or not. If Circular Queue is empty then we can not delete an element from Circular Queue. This condition is known as **"Underflow"**.

If Circular Queue is not underflow then we can delete an element from Circular Queue.

After deleting an element from Circular Queue we have to set the value of FRONT and REAR variables according to the elements in the Circular Queue.

Algorithm of Deletion Operation :

Step 1:

If FRONT = -1 then

Write ("Circular Queue Underflow")

Step 2:

Return (CQ [FRONT])

Step 3:

If FRONT = REAR then

FRONT=REAR=-1

Step 4:

If FRONT = SIZE-1 then

FRONT=0

Else

FRONT=FRONT+1

Conditions in Circular Queue :

There are two main conditions that can be performed on Circular Queue.

- **Queue Empty Condition:** ***front == rear***
- **Queue Full Condition:** ***front == (rear + 1) % MAX_Q_SIZE***

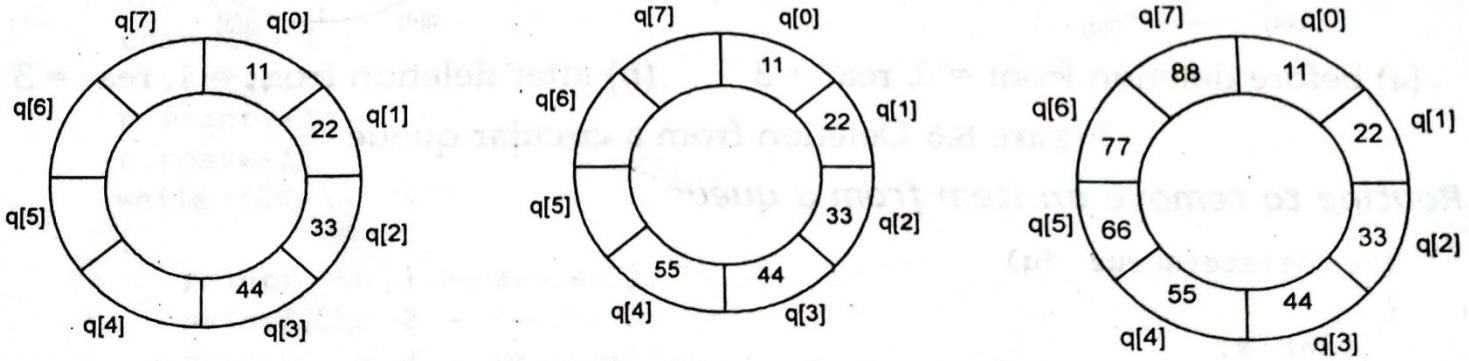
- Insert Operation

Consider the circular queue shown in Figure , the insertion in this queue will be same as with linear queue, we only have to keep track of *front* and *rear* with some extra logic. A circular queue is shown in the Figure . When an *item* is inserted into the queue we have to write:

```
rear = (rear + 1) % MAXQ;
q[rear] = item;
```

Figure (b) shows insertion of an element into the queue.

Figure (c) shows queue is full, so no more insertion is possible. This can be implemented by the condition *front* == (*rear* + 1) % *MAXQ;*

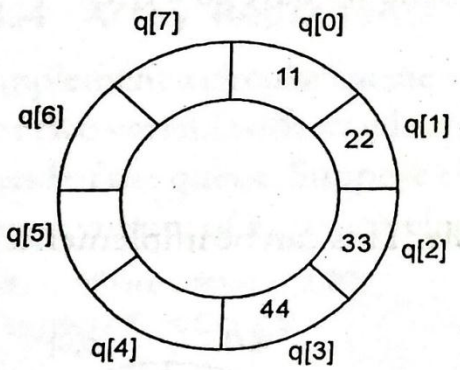


(a) front = 0, rear = 3 **(b)** item = 55 is to be inserted rear = rear +1 and q[rear] = item **(c)** queue full

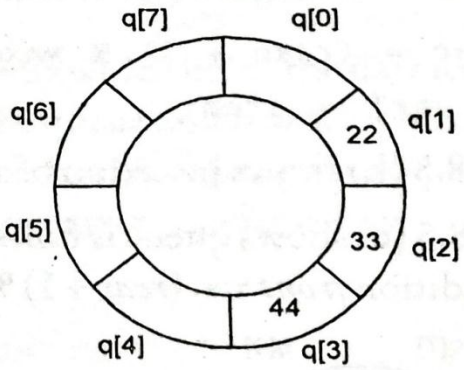
Figure : Insertion into a circular queue

- Delete Operation

Whenever an element is deleted from the queue, the value of *front* is increased by 1 i.e., `front = (front + 1) % MAXQ;` If *front == rear*, then queue is empty. Figure shows the deletion of elements from a circular queue.



(a) before deletion front = 0, rear = 3



(b) after deletion front = 1, rear = 3

Example: Program for Circular Queue

```
#include<stdio.h>
#include<cstdlib>
#define max 6
int q[10],front=0,rear=-1;
int main()  {
    int ch;
    void insert();
    void delet();
    void display();
    printf("\nCircular Queue Operations\n");
    printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
```

```
    while(1)
    {
        printf("Enter Your Choice:");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1: insert();
            break;
        case 2: delet();
            break;
        case 3:display();
            break;
        case 4: exit(0);
        default:printf("Invalid option\n");
        }
    }
}
```

```
void insert()
{
    int x;
    if((front==0&&rear==max-1)||(front>0&&rear==front-1))
        printf("Queue is Overflow\n");
    else
    {
        printf("Insert Element :");
        scanf("%d",&x);
        if(rear==max-1&&front>0)
        {
            rear=0;
            q[rear]=x;
        }
        else
        {
            if((front==0&&rear==-1)||(rear!=front-1))
             q[++rear]=x;
        }
    }
}
```

```
void  delet()
{
    int a;
    if((front==0)&&(rear==-1))
    {
    printf("Queue is Underflow\n");
    exit(0);
    }
    if(front==rear)
    {
        a=q[front];
        rear=-1;
        front=0;
    }
    else
        if(front==max-1)
        {
            a=q[front];
            front=0;
        }
        else a=q[front++];
        printf("Deleted Element is : %d\n",a);
}
```

```
void display()  {
    int i,j;
    if(front==0&&rear==-1)
    {
        printf("Queue is Underflow\n");
        exit(0);
    }
    if(front>rear)    {
        for(i=0;i<=rear;i++)
            printf("\t%d",q[i]);
        for(j=front;j<=max-1;j++)
            printf("\n \t%d",q[j]);
        printf("\nRear is at %d\n",q[rear]);
        printf("\nFront is at %d\n",q[front]);
    }
    else     {
        for(i=front;i<=rear;i++)
        {
            printf("\t%d",q[i]);
        }
        printf("\nRear is at %d\n",q[rear]);
        printf("\nFront is at %d\n",q[front]);
    }
    printf("\n");
}
```

Deque (Double – Ended - Queue) :

A deque is a linear in which can added or removed or deleted at either end but not in the middle. The figure shows Deque.

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

i) Input-restricted deque and

ii) Output restricted deque

Input restricted and output restricted deques are intermediate between a deque and a queue.

Specifically, an input restricted deque is a deque which allows insertions at only one end of the list but allows deletions at both ends of the list; and an output-restricted deque is a deque which allows deletion at only one end of the list but allows insertions at both ends of the list.

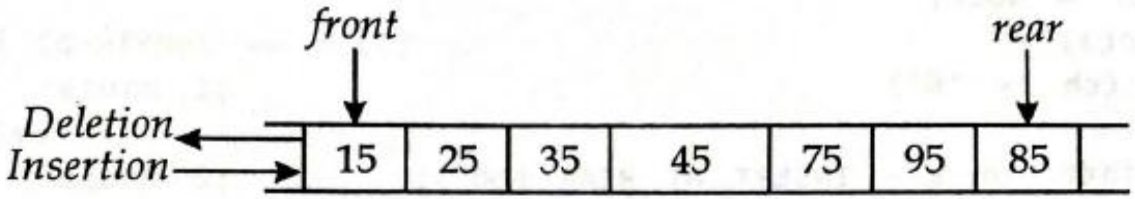


Figure: Double ended queue

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

a) Insertion of an element at the rear end of the queue.
b) Deletion of an element from the front end of the queue.
c) Insertion of an element at the front end of the queue.
d) Deletion of an element from the rear end of the queue.

Priority Queues

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:

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

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

An example of priority queue in computer science occurs in timesharing system in which the processes of higher priority is executed before any process of lower priority.

There are two types of priority queues:

i) ascending priority queue

ii) descending priority queue

An ascending priority queue is a collection of items in to which items can be inserted arbitrarily and from which only the smallest item can be removed. A descending priority queue is similar but allows deletion of only the largest item. The development of algorithm and program is left to the reader.

Application of Queue

As Queue is a FIFO structure, it is an appropriate data structure for used in numerous applications of computer science. One important application of queue is in simulation. Queues are mainly used to implement various aspects of operating systems. It is used to implement different CPU scheduling algorithms. Multiprogramming environment uses several queues to control various programs. Some of the applications of queue are as follows:

- Simulation of traffic control system
- CPU scheduling in multiprogramming and time sharing environment
- Multilevel queue scheduling
- Multilevel feedback queue scheduling
- Round Robin scheduling