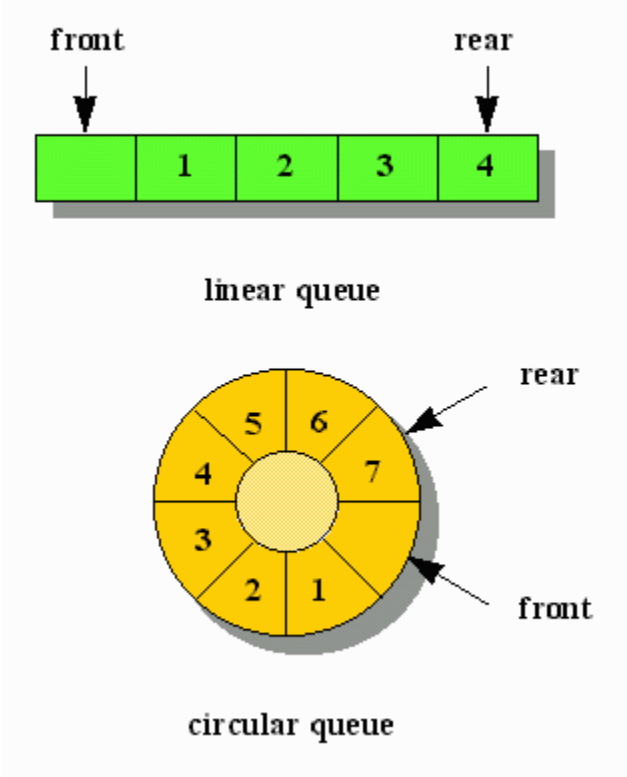
Circular Queue

In a standard queue data structure re-buffering problem occurs for each dequeue operation. To solve this problem by joining the front and rear ends of a queue to make the queue as a circular queue
Circular queue is a linear data structure. It follows FIFO principle.

- In circular queue the last node is connected back to the first node to make a circle.
- Circular linked list fallow the First In First Out principle
- Elements are added at the rear end and the elements are deleted at front end of the queue
- Both the front and the rear pointers points to the beginning of the array.
- It is also called as “Ring buffer”.
- Items can inserted and deleted from a queue in O(1) time.



CODES (C)

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
typedef struct queue
{
	int *a  ;
	int ms,r,f;
}Q;
void init(Q *q,int x)
{
	q->ms=x;
	q->a=(int *)malloc(q->ms*sizeof(int));
	q->f=0;
	q->r=-1;
}
int isfull(Q *q)
{
	if((q->r==q->ms-1 && q->f==0)||(q->f==q->r+1 && q->r!=-1))
		return 1;
	else
		return 0;
}
int isempty(Q *q)
{
	if(q->r==-1&&q->f==0)
		return 1;
	else
		return 0;
}
void enqueue(Q *q,int z)
{
	if(isfull(q))
		printf("queue full ");
	else
	{
		q->r=(q->r+1)%q->ms;
		q->a[q->r]=z;
	}
}
```

```
int dequeue(Q *q)
{
	int j=0;
	if(isempty(q)==1)
		printf("queue empty") ;
	else
	{

		if(q->r=q->f)
		{
			q->r=-1;
			q->f=0;
		}
		else
		{
			q->f=(q->f+1)%q->ms;
			j=q->a[q->f];
		}

	}
	return j;

}
```

CODES (JAVA)

```
import java.io.*;
 import java.lang.*;
 class clrqueue
 {
  DataInputStream get=new DataInputStream(System.in);
  int a[];
  int i,front=0,rear=0,n,item,count=0;
  void getdata()
  {
  try
   {
   System.out.println("Enter the limit");
   n=Integer.parseInt(get.readLine());
   a=new int[n];
   }
  catch(Exception e)
   {
   System.out.println(e.getMessage());
   }
  }
  void enqueue()
  {
   try
   {
   if(count<n)
    {
    System.out.println("Enter the element to be added:");
    item=Integer.parseInt(get.readLine());
    a[rear]=item;
     rear++;
    count++;
    }
   else
    System.out.println("QUEUE IS FULL");
   }
  catch(Exception e)
   {
   System.out.println(e.getMessage());
   }
  }
```

```
 void dequeue()
 {
  if(count!=0)
   {
   System.out.println("The item deleted is:"+a[front]);
   front++;
   count--;
   }
  else
   System.out.println("QUEUE IS EMPTY");
 if(rear==n)
  rear=0;
 }
 void display()
 {
  int m=0;
  if(count==0)
  System.out.println("QUEUE IS EMPTY");
  else
  {
  for(i=front;m<count;i++,m++)
  System.out.println(" "+a[i%n]);
  }
 }
}
```