Circular Queue

एक मानक कतार डेटा संरचना फिर बफरिंग समस्या में प्रत्येक विपंक्ति आपरेशन के लिए होता है. एक परिपत्र पंक्ति के रूप में पंक्ति बनाने के लिए एक कतार के सामने और पीछे के सिरों में शामिल होने से इस समस्या को हल करने के लिए
परिपत्र कतार एक रेखीय डेटा संरचना है. यह फीफो सिद्धांत को मानता है.

• परिपत्र कतार में पिछले नोड एक चक्र बनाने के लिए पहला नोड को वापस जुड़ा हुआ है.

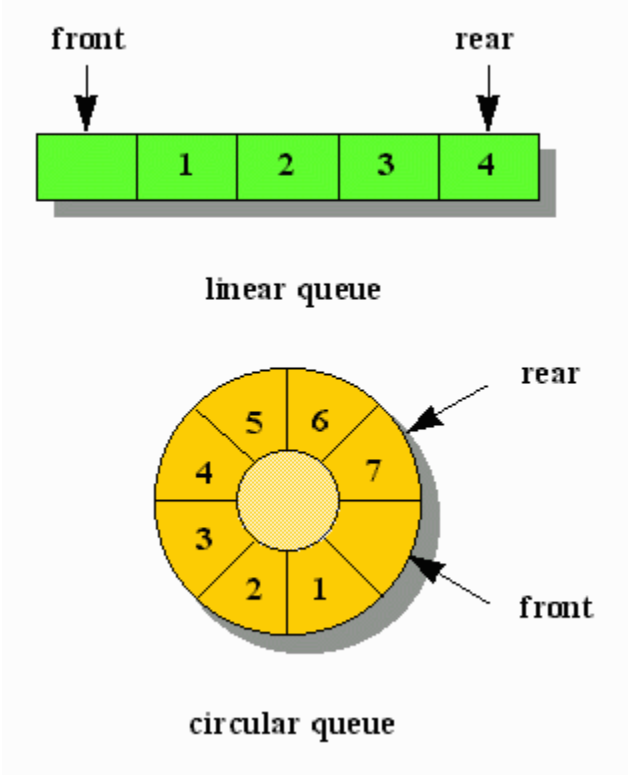
सबसे पहले आउट सिद्धांत रूप में परती पहले • परिपत्र लिंक की गई सूची

• तत्वों पीछे के अंत में जोड़ा जाता है और तत्वों लाइन के सामने अंत में नष्ट हो जाती हैं

• सामने और सरणी की शुरुआत करने के पीछे संकेत अंक दोनों.

• यह भी "रिंग बफर" के रूप में कहा जाता है.

• आइटम हे (1) समय में एक कतार से डाला जाता है और नष्ट कर सकते हैं.



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]);
  }
 }
}
```