**QURESHI FAISAL**

**SE-03**

**13**

**Experiment no 5**

**Aim:** To implement Circular Queue ADT using array

**Objective:**

Circular Queue offer a quick and clean way to store FIFIO data with maximum size

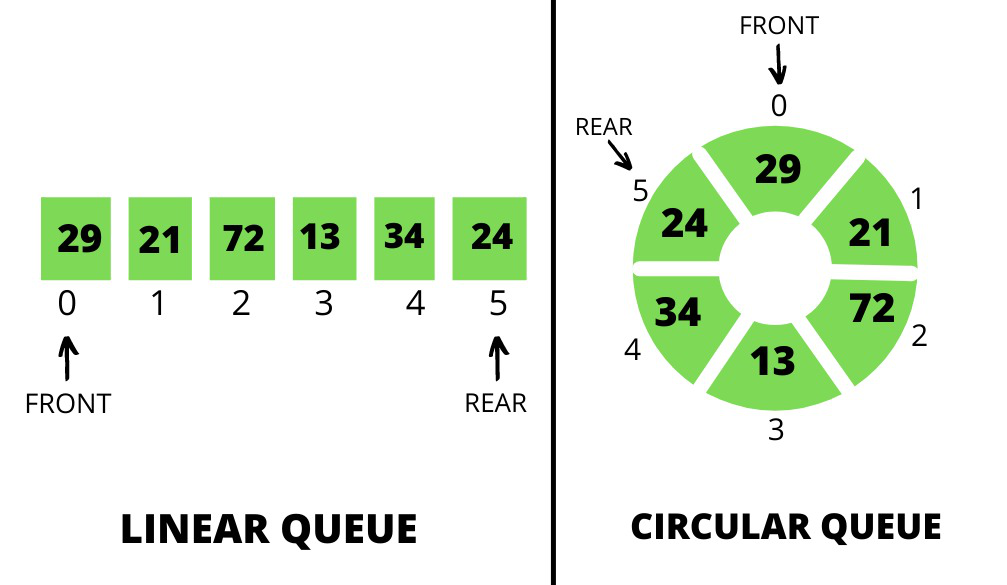
**Theory:**

A Circular Queue is an extended version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle. The operations are performed based on FIFO (First In First Out) principle. It is also called 'Ring Buffer'.

Solutions

• Solution 1: Shift the elements to the left.

• Solution 2: Use Circular Queue, where the first index comes right after last index.



**Algorithm :**

Algorithm to insert an element in a circular queue

Step 1: IF (REAR+1)%MAX = FRONT  
Write " OVERFLOW "  
Goto step 4  
[End OF IF]

Step 2: IF FRONT = -1 and REAR = -1  
SET FRONT = REAR = 0  
ELSE IF REAR = MAX - 1 and FRONT ! = 0  
SET REAR = 0  
ELSE  
SET REAR = (REAR + 1) % MAX  
[END OF IF]

Step 3: SET QUEUE[REAR] = VAL

Step 4: EXIT

Algorithm to delete an element from the circular queue

Step 1: IF FRONT = -1  
Write " UNDERFLOW "  
Goto Step 4  
[END of IF]

Step 2: SET VAL = QUEUE[FRONT]

Step 3: IF FRONT = REAR  
SET FRONT = REAR = -1  
ELSE  
IF FRONT = MAX -1  
SET FRONT = 0  
ELSE  
SET FRONT = FRONT + 1  
[END of IF]  
[END OF IF]

Step 4: EXIT

**Code:**

#include <stdio.h>

#include<conio.h>

#define SIZE 5

int items[SIZE];

int front = -1, rear = -1;

int isFull() {

if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;

return 0;

}

int isEmpty() {

if (front == -1) return 1;

return 0;

}

void enQueue(int element) {

if (isFull())

printf("\n Queue is full!! \n");

else {

if (front == -1) front = 0;

rear = (rear + 1) % SIZE;

items[rear] = element;

printf("\n Inserted -> %d", element);

}

}

int deQueue() {

int element;

if (isEmpty()) {

printf("\n Queue is empty !! \n");

return (-1);

} else {

element = items[front];

if (front == rear) {

front = -1;

rear = -1;

}

else {

front = (front + 1) % SIZE;

}

printf("\n Deleted element -> %d \n", element);

return (element);

}

}

void display() {

int i;

if (isEmpty())

printf(" \n Empty Queue\n");

else {

printf("\n Front -> %d ", front);

printf("\n Items -> ");

for (i = front; i != rear; i = (i + 1) % SIZE) {

printf("%d ", items[i]);

}

printf("%d ", items[i]);

printf("\n Rear -> %d \n", rear);

}

}

int main() {

deQueue();

enQueue(1);

enQueue(2);

enQueue(3);

enQueue(4);

enQueue(5);

enQueue(6);

display();

deQueue();

display();

enQueue(7);

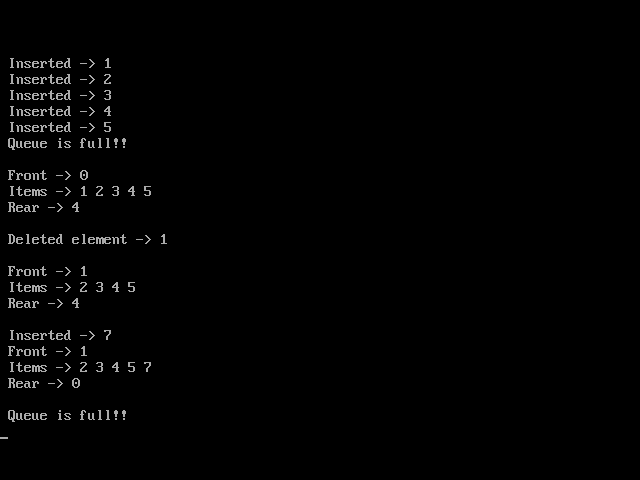
display();

enQueue(8);

getch();

return 0;

}

**Output:**

**Conclusion:**

we have successfully implemented the concept for circular queue.