**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'.

**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

**Circular Queue implementation in C**

#include <stdio.h>

#define SIZE 5

int items[SIZE];

int front = -1, rear = -1;

// Check if the queue is full

int isFull() {

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

return 0;

}

// Check if the queue is empty

int isEmpty() {

if (front == -1) return 1;

return 0;

}

// Adding an element

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);

}

}

// Removing an 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;

}

// Q has only one element, so we reset the

// queue after dequeing it. ?

else {

front = (front + 1) % SIZE;

}

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

return (element);

}

}

// Display the queue

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() {

// Fails because front = -1

deQueue();

enQueue(1);

enQueue(2);

enQueue(3);

enQueue(4);

enQueue(5);

// Fails to enqueue because front == 0 && rear == SIZE - 1

enQueue(6);

display();

deQueue();

display();

enQueue(7);

display();

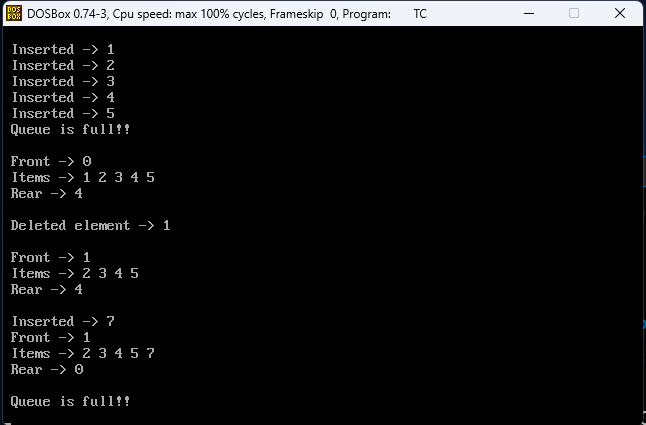
// Fails to enqueue because front == rear + 1

enQueue(8);

return 0;

}

**Output :**



**Conclusion:**

The circular queue is a linear data structure whose end is connected to the start and is used in the traffic system, memory management, and CPU scheduling.