**NAME: ASHWINI SHIVAJI RATHOD**

**CLASS:SE**

**DIV:3**

**Experiment no 5:Circular Queue implementation in C**

**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’. In a normal Queue, we can insert elements until the queue becomes full. But once the queue becomes full, we can not insert the next element even if there is a space in front of the queue.

**Operations on Circular Queue:**

1)Front: Get the front item from the queue.

2)Rear: Get the last item from the queue.

3)enQueue(value) This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at the rear position.

4)Check whether the queue is full – [i.e., the rear end is just before the front end in a circular manner].

5)If it is full then the display Queue is full.

6)If the queue is not full then, insert an element at the end of the queue.

7)deQueue() This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from the front position.

8)Check whether the queue is Empty.

9)If it is empty then the display Queue is empty.

10)If the queue is not empty, then get the last element and remove it from the queue.

**Algorithm:**

**Circular Queue Operations**

**The circular queue work as follows:**

1)two pointers FRONT and REAR

2)FRONT track the first element of the queue

3)REAR track the last elements of the queue

4)initially, set value of FRONT and REAR to -1

**1. Enqueue Operation**

1)check if the queue is full

2)for the first element, set value of FRONT to 0

3)circularly increase the REAR index by 1 (i.e. if the rear reaches the end, next it would be at the start of the queue)

4)add the new element in the position pointed to by REAR

**2. Dequeue Operation**

1)check if the queue is empty

2)return the value pointed by FRONT

3)circularly increase the FRONT index by 1

4)for the last element, reset the values of FRONT and REAR to -1

**However, the check for full queue has a new additional case:**

Case 1: FRONT = 0 && REAR == SIZE - 1

Case 2: FRONT = REAR + 1

The second case happens when REAR starts from 0 due to circular increment and when its value is just 1 less than FRONT, the queue is full.

**Code:**

#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();

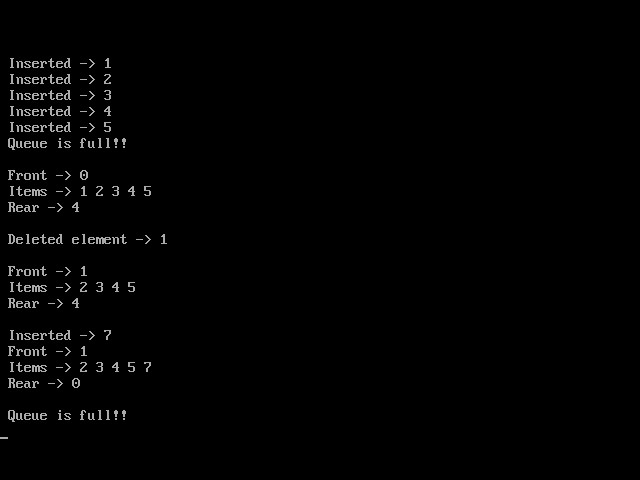
// Fails to enqueue because front == rear + 1

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