Khushi panjwani_D10B42 **lab-04**

[1]AIM- to understand the concept of circular queue data structure in array

Theory-
A Circular Queue is a data structure that overcomes the limitations of a linear queue. In a linear queue, once the queue is full, no more elements can be added, even if there is empty space created by dequeuing elements. A circular queue, on the other hand, treats the array as circular, allowing efficient utilization of space by wrapping around the indices when the end of the array is reached.

Here's a simple introduction to a circular queue using an array in C:

Key Concepts:

1. Front and Rear: These are two pointers that keep track of the beginning and the end of the queue.

2. Circular Nature: When rear reaches the end of the array, it wraps around to the beginning (index 0), as long as there is free space.

3. Queue Full Condition: The queue is full when the next position of rear would be front.

4. Queue Empty Condition: The queue is empty when front == -1.

Code-

```
#include <stdio.h>
#define SIZE 5 // Maximum size of the circular queue

int queue[SIZE];
int front = -1, rear = -1;

// Function to add an element to the circular queue
void enqueue(int value) {
    if ((front == 0 && rear == SIZE - 1) || (rear == (front - 1) % (SIZE - 1))) {
        printf("Queue is Full\n");
        return;
    }
    else if (front == -1) { // First element being inserted
        front = rear = 0;
        queue[rear] = value;
    }
```

[1]

```
    else if (rear == SIZE - 1 && front != 0) {
        rear = 0;
        queue[rear] = value;
    }
    else {
        rear++;
        queue[rear] = value;
    }
}

// Function to remove an element from the circular queue
int dequeue() {
    if (front == -1) {
        printf("Queue is Empty\n");
        return -1;
    }

    int data = queue[front];
    queue[front] = -1; // Reset the dequeued position

    if (front == rear) { // Queue becomes empty
        front = rear = -1;
    }
    else if (front == SIZE - 1) {
        front = 0;
    }
    else {
        front++;
    }

    return data;
}

// Function to display the queue
void displayQueue() {
    if (front == -1) {
        printf("Queue is Empty\n");
        return;
    }

    printf("Queue elements: ");
    if (rear >= front) {
        for (int i = front; i <= rear; i++)
            printf("%d ", queue[i]);
```

```
    }
    else {
        for (int i = front; i < SIZE; i++)
            printf("%d ", queue[i]);
        for (int i = 0; i <= rear; i++)
            printf("%d ", queue[i]);
    }
    printf("\n");
}

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    enqueue(40);
    enqueue(50);

    displayQueue();

    dequeue();
    displayQueue();

    enqueue(60);
    displayQueue();

    return 0;
}
```

Output-

```
/tmp/uMzPbUshY1.o
Queue elements: 10 20 30 40 50
Queue elements: 20 30 40 50
Queue elements: 20 30 40 50 60
```

Conclusion-

A circular queue efficiently utilizes array space by wrapping around when the end is reached, preventing overflow until all slots are filled. This implementation in C allows dynamic insertion and deletion while maintaining queue order.