EXPERIMENT 5

Aim:

1. Implement the following functionalities of the Circular queue using Arrays:
    a. isFull – to check if the queue is full or not.
    b. isEmpty – to check if the queue is empty or not.
    c. enqueue – to insert the element in the queue.
    d. dequeue – to delete the element from the queue.
    e. front and rear – to print the front and rear element of the queue.

Theory (Circular Queue using Array):

- **Queue Representation**: The queue is implemented as an array with fixed size, and the elements are added and removed in a circular manner.
- **Front and Rear Pointers**:
    - front: Points to the element at the front of the queue.
    - rear: Points to the most recently added element.
    - Both are initialized to -1, indicating the queue is empty.
- **Circular Nature**:
    - When rear reaches the end of the array, it wraps around to the beginning $(rear = (rear + 1) \% SIZE)$.
    - Similarly, front increments circularly after each removal $(front = (front + 1) \% SIZE)$.
- **isEmpty**: The queue is empty if $front == -1$.
- **isFull**: The queue is full if $((rear + 1) \% SIZE == front)$.
- **enqueue Operation**:
    - If the queue is not full, the element is inserted at rear, and rear is incremented circularly.
    - If the queue was empty, both front and rear are set to 0 to handle the first insertion.
- **dequeue Operation**:
    - Removes the element at front and circularly increments front.
    - If front equals rear after removal, both pointers are reset to -1 to mark the queue as empty.

Program:

```
#include <stdio.h>
#include <stdbool.h>

#define SIZE 10

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

bool isEmpty();
```

```
bool isFull();
void enqueue(int ele);
void dequeue();
void printQueue();

int main(void)
{
   printQueue();
   enqueue(18);
   enqueue(27);
   enqueue(36);
   enqueue(45);
   printQueue();
   enqueue(54);
   enqueue(63);
   enqueue(72);
   enqueue(81);
   enqueue(90);
   enqueue(99);
   enqueue(99);
   printQueue();
   dequeue();
   dequeue();
   dequeue();
   printQueue();
   enqueue(999);
   enqueue(153);
   // Roll no. above
   printQueue();

   return 0;
}

bool isEmpty()
{
   return (front == -1);
}

bool isFull()
{
   return ((rear + 1) % SIZE == front);
}

void enqueue(int ele)
{
   if (isFull())
   {
      printf("Queue is FULL!\n");
      return;
   }
   if (isEmpty())
```

```
    {
        front = rear = 0;
    }
    else
    {
        rear = (rear + 1) % SIZE;
    }
    queue[rear] = ele;
    printf("Inserted: %d\n", ele);
}

void dequeue()
{
    if (isEmpty())
    {
        printf("Queue is EMPTY!\n");
        return;
    }

    printf("Deleted: %d\n", queue[front]);
    if (front == rear)
    {
        front = rear = -1;
    }
    else
    {
        front = (front + 1) % SIZE;
    }
}

void printQueue()
{
    if (isEmpty())
    {
        printf("Queue is EMPTY!\n");
        return;
    }
    printf("Queue: ");
    int i = front;
    while (i != rear)
    {
        printf("%d ", queue[i]);
        i = (i + 1) % SIZE;
    }
    printf("%d\n", queue[rear]);
}
```

Output:

```
PS B:\sem3\ds\23bcp153_dsa\lab5> gcc qarr.c -o qarr
PS B:\sem3\ds\23bcp153_dsa\lab5> ./qarr
Queue is EMPTY!
Inserted: 18
Inserted: 27
Inserted: 36
Inserted: 45
Queue: 18 27 36 45
Inserted: 54
Inserted: 63
Inserted: 72
Inserted: 81
Inserted: 90
Inserted: 99
Queue is FULL!
Queue: 18 27 36 45 54 63 72 81 90 99
Deleted: 18
Deleted: 27
Deleted: 36
Queue: 45 54 63 72 81 90 99
Inserted: 999
Inserted: 153
Queue: 45 54 63 72 81 90 99 999 153
```

Time Complexity:

The time complexity for **enqueue** and **dequeue** operations in a circular queue is $\boldsymbol{O(1)}$, as both involve constant-time operations to add or remove an element, regardless of the queue's size.