# EXPERIMENT – 6

| | |
|---|---|
| **Student Name: Garv Khurana** | **UID: 21BCS6615** |
| **Branch: CSE-AIML** | **Section/Group: 21AML - 9 - "A"** |
| **Semester: 3rd** | |
| **Subject Name: Data structures** | **Subject Code: 21 CSH-241** |

## 1. Aim of experiment

**(a) Write a program for a circular queue.**

## 2. Algorithm

The circular queue work as follows:
- two pointers FRONT and REAR
- FRONT track the first element of the queue
- REAR tracks the last elements of the queue
- initially, set the value of FRONT and REAR to -1

**1. Enqueue Operation**
- check if the queue is full
- for the first element, set value of FRONT to 0
- 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)
- add the new element in the position pointed to by REAR

**2. Dequeue Operation**
- check if the queue is empty
- return the value pointed by FRONT
- circularly increase the FRONT index by 1
- 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

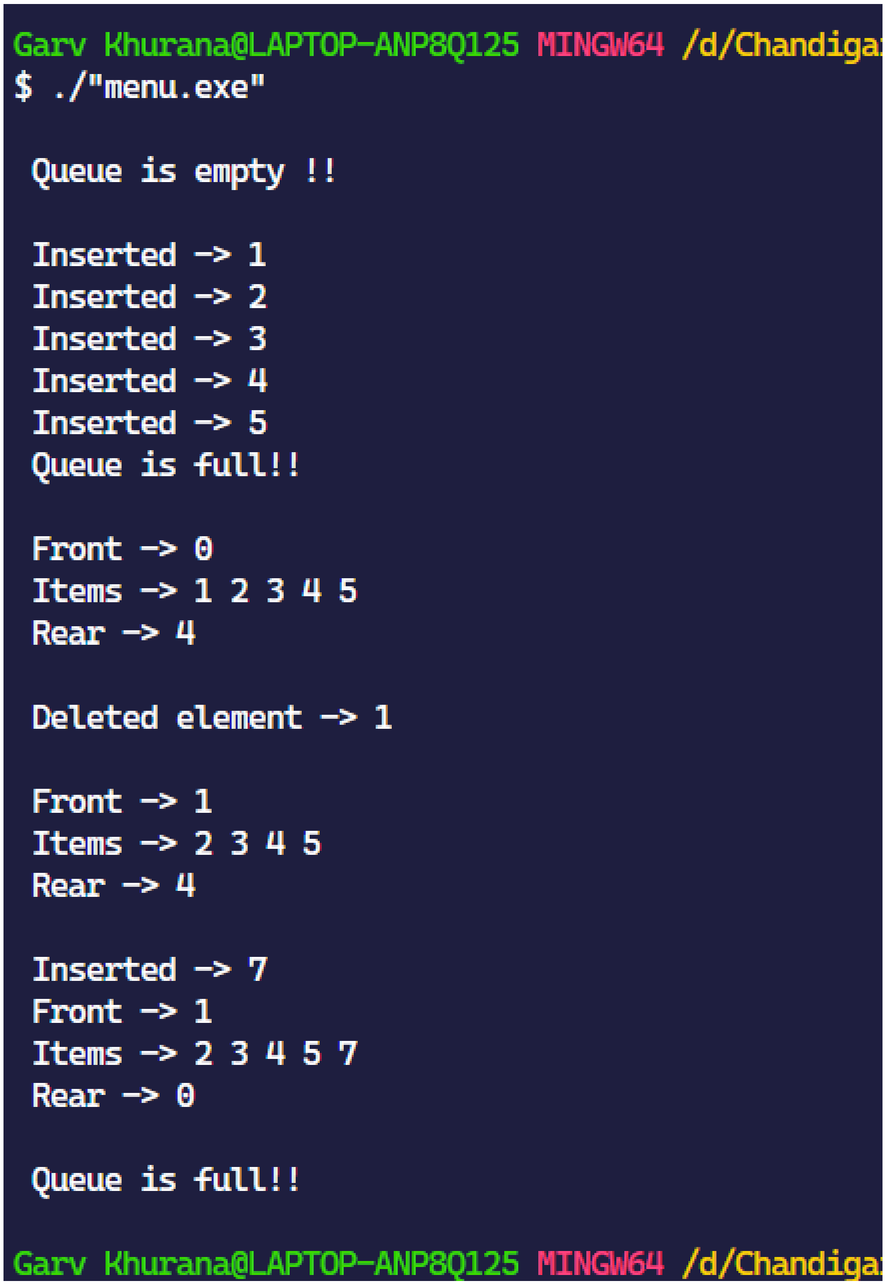
The second case happens when REAR starts from 0 due to circular increment and when its value is just 1 less than FRONT, the queue is full.

## Program 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



```
Garv Khurana@LAPTOP-ANP8Q125 MINGW64 /d/Chandiga
$ ./"menu.exe"

 Queue is empty !!

 Inserted -> 1
 Inserted -> 2
 Inserted -> 3
 Inserted -> 4
 Inserted -> 5
 Queue is full!!

 Front -> 0
 Items -> 1 2 3 4 5
 Rear -> 4

 Deleted element -> 1

 Front -> 1
 Items -> 2 3 4 5
 Rear -> 4

 Inserted -> 7
 Front -> 1
 Items -> 2 3 4 5 7
 Rear -> 0

 Queue is full!!

Garv Khurana@LAPTOP-ANP8Q125 MINGW64 /d/Chandiga
```

## Learning outcomes (What I have learned):

1. Concept of the circular queue.

2. Dequeue operation on circular queue.

3. Enqueue operation on circular queue.

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
|---|---|---|---|
| 1. | | | |
| 2. | | | |
| 3. | | | |
| | | | |