Experiment – 2.4

Student Name: Rohit Panghal **UID: 21BCS9294**
Branch: CSE **Section/Group: 902A**
Semester: 3rd **Date:**
Subject Name: Data Structures **Subject Code: 21CSH-211**

Aim of the practical:

Write a program to demonstrate the implementation of various operations on a linear queueand circular represented using a linear array.

Algorithm:

- **Working of Queue**

  - Queue operations work as follows:
  - two pointers FRONT and REAR
  - FRONT track the first element of the queue
  - REAR track the last element of the queue
  - initially, set value of FRONT and REAR to -1

- **EnQueue Operation**

  - check if the queue is full
  - for the first element, set the value of FRONT to 0
  - increase the REAR index by 1
  - add the new element in the position pointed to by REAR

- **DeQueue Operation**

  - check if the queue is empty
  - return the value pointed by FRONT
  - increase the FRONT index by 1
  - for the last element, reset the values of FRONT and REAR to -1

Program Code:

Linear Queue :

```
#include <iostream>
using namespace std;

struct Queue
{

   int front, rear, capacity;
   int* queue;

   Queue(int c)
   {
```

```
    front = rear = 0;
    capacity = c;
    queue = new int;
}

~Queue()
{
    delete[] queue;
}
void Enqueue(int data)
{
    if (capacity == rear)
    {
        printf("\nQueue is full\n");
        return;
    }
    else
    {
        queue[rear] = data;
        rear++;

    }
    return;
```

```
}

void Dequeue()
{
    if (front == rear)
    {
        printf("\nQueue is empty\n");
        return;
    }
    else
    {
        for (int i = 0; i < rear - 1; i++)
        {
            queue[i] = queue[i + 1];
        }
        rear--;
    }
    return;
}

void Display()
{
```

```
    int i;
    if (front == rear)
    {

        printf("\nQueue is Empty\n");
        return;
    }
    for (i = front; i < rear; i++)
    {
        printf(" %d <-- ", queue[i]);
    }
    return;
}

void Front()
{
    if (front == rear)
    {
        printf("\nQueue is Empty\n");
        return;
    }
    printf("\nFront Element is: %d", queue[front]);
```

```
        return;
    }
};

int main(void)
{
    Queue q(4);
    q.Display();
    q.Enqueue(10);
    q.Enqueue(30);
    q.Enqueue(50);
    q.Enqueue(70);
    q.Display();
    q.Enqueue(60);
    q.Display();
    q.Dequeue();
    q.Dequeue();
    printf("\n\nafter two node deletion\n\n");
    q.Display();
    q.Front();
    return 0;
}
```

Circular Queue :

```
#include<bits/stdc++.h>
using namespace std;

class Queue
{
   int rear, front,size,*arr; public:
   Queue(int s)
   {

      front = rear = -1;
      size = s;
      arr = new int[s];
   }
   void enQueue(int value);
   int deQueue();
   void displayQueue();
};

void Queue::enQueue(int value)
```

```
{
   if ((front == 0 && rear == size-1) || (rear == (front-1)%(size-1)))
   {
      printf("\nQueue is Full");
   return;
   }
   else if (front == -1)
   {
      front = rear = 0;
      arr[rear] = value;
   }
   else if (rear == size-1 && front != 0)
   {
      rear = 0;
      arr[rear] = value;
   }
   else
   {
      rear++;
      arr[rear] = value;
   }
```

```
}

int Queue::deQueue()
{
    if (front == -1)
    {
        printf("\nQueue is Empty");
        return INT_MIN;
    }
    int data = arr[front];
    arr[front] = -1;
    if (front == rear)
    {
        front = -1;
        rear = -1;
    }
    else if (front == size-1)
        front = 0;
    else
        front++;
```

```
    return data;
}

void Queue::displayQueue()
{
    if (front == -1)
    {
        printf("\nQueue is Empty");
        return;
    }
    printf("\nElements in Circular Queue are: ");
    if (rear >= front)
    {
        for (int i = front; i <= rear; i++)
            printf("%d ",arr[i]);
    }
    else
    {
        for (int i = front; i < size; i++)
            printf("%d ", arr[i]);
        for (int i = 0; i <= rear; i++) printf("%d ", arr[i]);
```

```
    }
}

int main()
{
    Queue q(5);
    q.enQueue(13);
    q.enQueue(24);
    q.enQueue(17);
    q.enQueue(-4);
    q.displayQueue();
    printf("\nDeleted value = %d", q.deQueue());
    printf("\nDeleted value = %d", q.deQueue());
    q.displayQueue();
    q.enQueue(71);
    q.enQueue(81);
    q.enQueue(1);
    q.displayQueue();
    q.enQueue(20);
    return 0;
}
```

Output:

Linear Queue :

```
Queue is Empty
 10 <--  30 <--  50 <--  70 <--
Queue is full
 10 <--  30 <--  50 <--  70 <--

after two node deletion

 50 <--  70 <--
Front Element is: 50
```

Circular Queue :

```
Elements in Circular Queue are: 13 24 17 -4
Deleted value = 13
Deleted value = 24
Elements in Circular Queue are: 17 -4
Elements in Circular Queue are: 17 -4 71 81 1
Queue is Full
```

Learning outcomes (What I have learnt):

- Learnt the concept of queue.
- How to perform various operations on queue.
- Learnt about linear and circular queue.