Yash Sarang.
Roll No: 47, Class : D6AD.
Data Structures. Experiment-06.

Aim : Implement Circular Queue ADT using array.

Theory: A circular queue is similar to a linear queue as it is also based on the FIFO (First In First Out) principle except that the last position is connected to the first position in a circular queue that forms a circle. It is also known as a Ring Buffer.

Operations on Circular Queue

The following are the operations that can be performed on a circular Queue:

- Front: It is used to get the front element from the Queue.
- Rear: It is used to get the rear element from the Queue.
- enQueue(value): This function is used to insert the new value in the Queue. The new element is always inserted from the rear end.
- deQueue(): This function deletes an element from the Queue. The deletion in a Queue always takes place from the front end.

A.Enqueue operation

The steps of enqueue operation are given below:

1.First, we will check whether the Queue is full or not.

2.Initially the front and rear are set to -1. When we insert the first element in a Queue, front and rear both are set to 0.

3.When we insert a new element, the rear gets incremented, i.e., rear=rear+1. Scenarios for inserting an element

There are two scenarios in which queue is not full:

- If rear != max - 1, then rear will be incremented to mod(maxsize) and the new value will be inserted at the rear end of the queue.

• If front != 0 and rear = max - 1, it means that queue is not full, then set the value of rear to 0 and insert the new element there.

• There are two cases in which the element cannot be inserted:
When front ==0 && rear = max-1, which means that front is at the first position of the Queue and rear is at the last position of the Queue. front== rear + 1;

Algorithm:
Algorithm to insert an element in a circular queue

```
Step 1:     IF (REAR+1)%MAX = FRONT
                Write " OVERFLOW "
                Goto step 4
            [ End OF IF]

Step 2:     IF FRONT = -1 and REAR = -1
                SET FRONT = REAR = 0
            ELSE IF
                REAR = MAX - 1 and FRONT ! = 0
                SET REAR = 0
            ELSE
                SET REAR = (REAR + 1) % MAX
            [END OF IF]

Step 3: SET QUEUE[REAR] = VAL

Step 4: EXIT
```

B.Dequeue Operation

The steps of dequeue operation are given below:

1.First, we check whether the Queue is empty or not. If the queue is empty, we cannot perform the dequeue operation.

2.When the element is deleted, the value of front gets decremented by 1.

3.If there is only one element left which is to be deleted, then the front and rear are reset to -1.

Algorithm to delete an element from the circular queue

Step 1: IF FRONT = -1
Write " UNDERFLOW "
Goto Step 4
[END of IF]

Step 2: SET VAL = QUEUE[FRONT]

Step 3: IF FRONT = REAR
SET FRONT = REAR = -1
ELSE IF FRONT = MAX -1
SET FRONT = 0
ELSE
SET FRONT = FRONT + 1
[END of IF]

Step 4: EXIT

C Program:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 4

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

void enquene(int data)
{
	if (front == -1 && rear == -1)
	{
		front = 0;
		rear = 0;
		cQuene[rear] = data;
	}
	else if ((rear + 1) % SIZE == front)
	{
		printf("Quene is overflow");
	}
	else
	{
```

```
                rear = (rear + 1) % SIZE;
                cQuene[rear] = data;
        }
}
int dequeue()
{
        if ((front == -1) && (rear == -1)) //
        {
                printf("\nQueue is underflow..");
        }
        else if (front == rear)
        {
                printf("\nThe dequeued element is %d", cQuene[front]);
                front = -1;
                rear = -1;
        }
        else
        {
                printf("\nThe dequeued element is %d", cQuene[front]);
                front = (front + 1) % SIZE;
        }
}
void Display()
{
        int i = front;
        if (front == -1 && rear == -1)
        {
                printf("\n Quene is Empty");
        }
        else
        {
                printf("\n Elements in the Quene Are: ");
                while (i != rear)
                {
                        printf("%d ", cQuene[i]);
                        i = (i + 1) % SIZE;
                }
        }
}
int main()
{
        int choice, a;
        do
        {
                printf("\n ***** Circular Quene ****");
                printf("\n 1. Insert an Element");
                printf("\n 2. Delete an Element");
```

```
            printf("\n 3. Dispaly The Quene");
            printf("\n 4. Exit ");
            printf("\n Enter a choice");
            scanf("%d", &choice);
            switch (choice)
            {
                case 1:
                    printf("\n Enter the element to be inserted : ");
                    scanf("%d", &a);
                    enquene(a);
                    break;
                case 2:
                    dequeue();
                    break;
                case 3:
                    Display();
                    break;
                case 4:
                    exit(0);
                    break;
                default:
                    printf("Invalid Input");
                    break;
            }
        } while (choice < 4);
        return 0;
}
```

Output:

```
***** Circular Quene ****
1. Insert an Element
2. Delete an Element
3. Dispaly The Quene
4. Exit
Enter a choice1

Enter the element to be inserted : 3

***** Circular Quene ****
1. Insert an Element
2. Delete an Element
3. Dispaly The Quene
4. Exit
Enter a choice1

Enter the element to be inserted : 32

***** Circular Quene ****
1. Insert an Element
2. Delete an Element
3. Dispaly The Quene
4. Exit
Enter a choice3

Elements in the Quene Are: 3
***** Circular Quene ****
1. Insert an Element
2. Delete an Element
3. Dispaly The Quene
4. Exit
Enter a choice2

The dequeued element is 3
***** Circular Quene ****
1. Insert an Element
2. Delete an Element
3. Dispaly The Quene
4. Exit
Enter a choice4


...Program finished with exit code 0
Press ENTER to exit console.
```