Experiment No.5
Implement Circular Queue ADT using array
Name: Shravani Sandeep Raut
Roll No: 48
Date of Performance:
Date of Submission:

Experiment No. 5: Circular Queue

Aim: To Implement Circular Queue ADT using array

Objective:

Circular Queues offer a quick and clean way to store FIFO data with a maximum size

Theory:

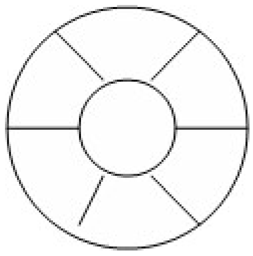
Circular queue is an data structure in which insertion and deletion occurs at an two ends rear and front respectively. Eliminating the disadvantage of linear queue that even though there is a vacant slots in array it throws full queue exception when rear reaches last element. Here in an circular queue if the array has space it never throws an full queue exception. This feature needs an extra variable count to keep track of the number of insertion and deletion in the queue to check whether the queue is full or not.Hence circular queue has better space utilization as compared to linear queue. Figure below shows the representation of linear and circular queue.

Linear queue

Front rear

0	…	…		n

Circular Queue



Algorithm

Algorithm : ENQUEUE(Item)

Input : An item is an element to be inserted in a circular queue.

Output : Circular queue with an item inserted in it if the queue has an empty slot.

Data Structure : Q be an array representation of a circular queue with front and rear pointing to the first and last element respectively.

```
1. If front = 0
```

```
        front = 1
        rear =1 Q[front]
        = item
  2. else
        next=(rear mod length)
        if next!=front then
            rear = next
            Q[rear] = item
        Else
            Print "Queue is full"
        End if
     End if
  3. stop
```

Algorithm : DEQUEUE()

Input : A circular queue with elements.

Output :Deleted element saved in Item.

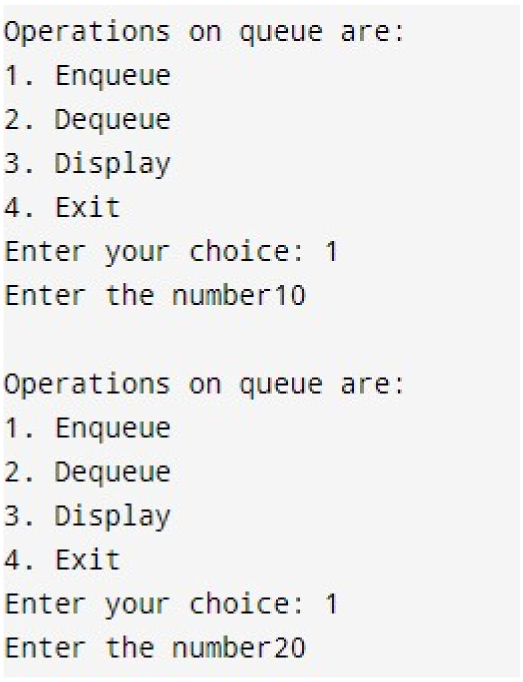
Data Structure : Q be an array representation of a circular queue with front and rear pointing to the first and last element respectively.

```
  1. If front = 0
        Print "Queue is empty"
        Exit
  2. else
        item = Q[front]
        if front = rear then
            rear = 0
            front=0
        else
            front = front+1
        end if
     end if
  3. stop
```

Code:

```
#include <stdio.h>
#include <conio.h>
#define MAX 4
int q[MAX], front =-1, rear =-1, num, n;

void enqueue()
{
        if(front==(rear +1)%MAX)
        {
                printf("Queue is full");
        }
        else
        {
                printf("Enter the number");
                scanf("%d",&n);
                rear = (rear+1) %MAX;
                q[rear] = num;
                if(front == -1)
                {
                     front ++;
                }
         }
}

void dequeue()
{
   int num;
        if(front == -1 && rear == -1)
        {
                printf("Queue is empty");
        }
        else
        {
                num = q[front];
                printf("%d is deleted",num);
                if(front == rear)
                {
                        front = -1;
                        rear = -1;
                }
                else
                {
                        front = (front +1)% MAX;
                }
        }
}

void display()
{
        int i;
        if(front == -1 && rear == -1)
        {
                printf("Queue is empty");
        }
        else
        {
                for(i=front ; i != rear; i =(i+1)% MAX);
                {
                        printf("%d\t",q[i]);
                }
                printf("%d", q[i]);
        }
}

void main()
{
        int m;
        clrscr();
        do{
        printf("\nOperations on queue are:");
        printf("\n1. Enqueue");
        printf("\n2. Dequeue");
        printf("\n3. Display");
        printf("\n4. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &m);
        switch(m)
        {
                case 1: enqueue();
                break;
                case 2: dequeue();
                break;
                case 3: display();
                break;
        }
        }while(m!=4);
}
```

Output:



```
Operations on queue are:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the number10

Operations on queue are:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the number20
```

Conclusion:

Explain how Josephus Problem is resolved using circular queue and elaborate on operation used for the same.

- The Josephus Problem can be solved using a circular queue by initializing the queue with n people.
- Starting from an index, you move k−1 steps, eliminate the person at that position, and adjust the index.
- This process continues until one person remains, efficiently simulating the elimination process with O(n) time complexity.