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:

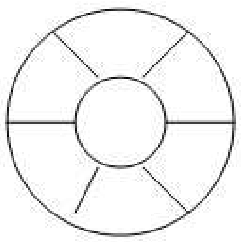
Circular queue is an data structure in which insertion and deletion occurs at an two ends rear and front respectively. Eliminating the disadvantage of linear queue that even though there is a vacant slots in array it throws full queue exception when rear reaches last element. Here in an circular queue 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>

# define max 6

int queue[max]; // array declaration

int front=-1;

int rear=-1;

void enqueue(int element) {

if(front==-1 && rear==-1) {

front=0;
```

```
rear=0;
queue[rear]=element;
} else if((rear+1)%max==front) {
printf("Queue is overflow..");
} else {
rear=(rear+1)%max; // rear is incremented
queue[rear]=element; // assigning a value to the queue at the rear position.
} } int dequeue() {
if((front==-1) && (rear==-1))  {
printf("\nQueue is underflow..");
} else if(front==rear) {
printf("\nThe dequeued element is %d", queue[front]);
front=-1;
rear=-1;
} else {
printf("\nThe dequeued element is %d", queue[front]);
front=(front+1)%max;
} } void display() {
int i=front;
if(front==-1 && rear==-1) {
printf("\n Queue is empty..");
} else {
printf("\nElements in a Queue are :");
while(i<=rear) {
printf("%d,", queue[i]);
i=(i+1)%max;
} } }
int main() {
int choice=1,x;
while(choice<4 && choice!=0) {
printf("\n Press 1: Insert an element");
printf("\nPress 2: Delete an element");
printf("\nPress 3: Display the element");
printf("\nEnter your choice");
scanf("%d", &choice);
switch(choice) {
case 1:
printf("Enter the element which is to be inserted");
scanf("%d", &x);
enqueue(x);
break;
case 2:
dequeue();
break;
case 3:
display();
}}
return 0;
}
```

Output:

```
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element

Enter your choice 1

1

Enter the element which is to be inserted 10

10

Enter your choice 1

1

Enter the element which is to be inserted 20

Enter your choice 1

1

Enter the element which is to be inserted 0

Enter your choice 2

The dequeued element is 10

Elements in a Queue are :20,0,
```

Conclusion:

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

Ans. The Josephus problem is a mathematical puzzle that involves a group of people standing in a circle, waiting to be executed. The executioner starts from a certain point and skips a fixed number of people before killing the next one. This process continues until only one person is left alive, who is then spared.

To solve the Josephus problem using a circular queue, we can follow these steps:

- ➢ Create a circular queue of size N, where N is the number of people in the circle. Each element in the queue represents a person's position in the circle, starting from 1 to N.
- ➢ Initialize a variable M, where M is the number of people to be skipped before each execution.
- ➢ Initialize a pointer P, which points to the front of the queue.
- ➢ Repeat until the queue has only one element left:
  - o Move the pointer P forward by M positions, wrapping around the queue if necessary.
  - o Delete the element at P from the queue, which represents the person to be executed.
  - o Move the pointer P forward by one position, which represents the next starting point for counting.
- ➢ Return the remaining element in the queue, which represents the position of the survivor.