Experiment No.5
Aim: To Implement Circular Queue ADT using array
Name: Nikhil Kamalaji Shingade
Roll no: 57
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

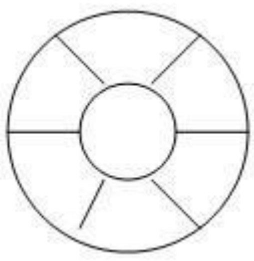
Theory:

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:

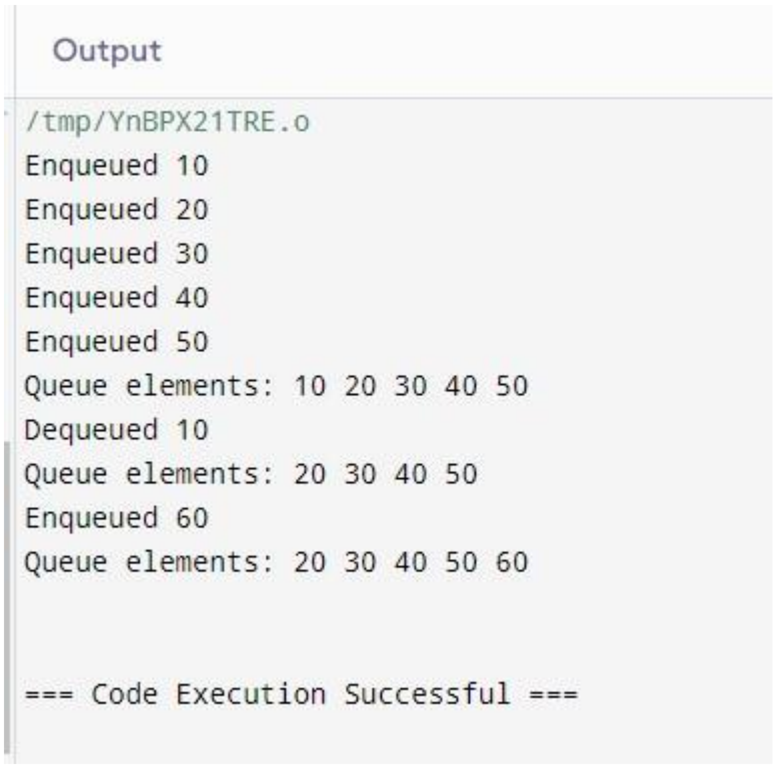
```
#include <stdio.h>
#include <stdlib.h>

#define MAX 5
```

```
int circularQueue[MAX];
int front = -1, rear = -1; int
isFull()
{
   return (front == 0 && rear == MAX - 1) || (front == rear + 1);
}
int isEmpty() {
   return front == -1;
}
void enqueue(int element)
{    if
(isFull())
   {
      printf("Queue is full. Cannot enqueue %d\n", element);
   } else {        if (front == -1) front =
0;         rear = (rear + 1) % MAX;
circularQueue[rear] = element;
      printf("Enqueued %d\n", element);
   }
}
int dequeue()
{
   int element;
   if (isEmpty())
   {
      printf("Queue is empty. Cannot dequeue.\n");
return -1;
   } else {
      element = circularQueue[front];
      if (front == rear)
      {
front = -1;
rear = -1;
      } else {
         front = (front + 1) % MAX;
      }
      printf("Dequeued %d\n", element);
      return element;
   }
}
void display()
{
   int i;    if
(isEmpty())
   {
        printf("Queue is empty.\n");
    } else {        printf("Queue
elements: ");
        for (i = front; i != rear; i = (i + 1) % MAX)
        {
            printf("%d ", circularQueue[i]);
        }
        printf("%d\n", circularQueue[i]);
    }
}

int main()
{
    enqueue(10);
enqueue(20);
enqueue(30);
enqueue(40);
enqueue(50);
display();    dequeue();
    display();
enqueue(60);
display();    return
0;
}
```

Output:



```
Output
/tmp/YnBPX21TRE.o
Enqueued 10
Enqueued 20
Enqueued 30
Enqueued 40
Enqueued 50
Queue elements: 10 20 30 40 50
Dequeued 10
Queue elements: 20 30 40 50
Enqueued 60
Queue elements: 20 30 40 50 60


=== Code Execution Successful ===
```

Conclusion:

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

ANS:

The Josephus Problem is a famous theoretical problem in mathematics and computer science. It inv olves people standing in a circle and eliminating every k-th person until only one person remains. Using a circular queue can provide an efficient solution. **Steps to Solve the Josephus Problem Using a Circular Queue**

1. **Initialization**:
    - Enqueue all people into the circular queue.
    - Let's assume you have n people labeled from 1 to n.
2. **Elimination**:
    - Dequeue the first k-1 people and enqueue them back to the end of the queue.
    - Dequeue the k-th person and remove them from the queue.
    - Repeat the process until only one person remains.

Operations Used

- **Enqueue**: Adds an element to the end of the circular queue.
- **Dequeue**: Removes an element from the front of the circular queue.
- **Modulo Operation**: Helps to wrap around the end of the queue to simulate the circle.