Experiment No.5
Implement Circular Queue ADT using array
Name: Sharvari Anand Bhondekar
Roll No: 06
Date of Performance:
Date of Submission:
Marks:
Sign:

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:

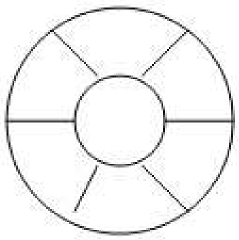
Circular queue is an data structure in which insertion and deletion occurs at an two ends rear and front respectively. Eliminating the disadvantage of linear queue that even though there is a vacant slots in array it throws full queue exception when rear reaches last element. Here in an circular queue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

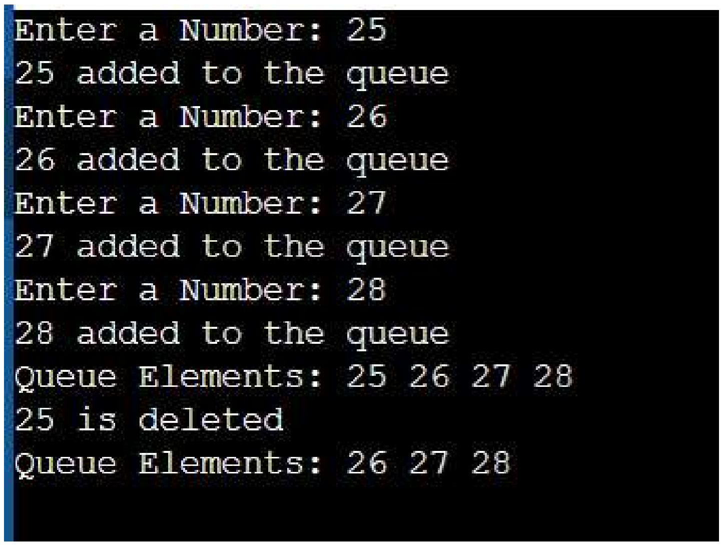
```
    1.  If front = 0
          Print "Queue is empty"
          Exit
    2.  else
          item = Q[front]
          if front = rear then
              rear = 0
              front=0
          else
              front = front+1
          end if
        end if
    3.  stop
```

Code:

```
#include <stdio.h>

#include <conio.h>

#define MAX 100


int front = -1, rear = -1, q[MAX], n;


void enqueue() {

printf("Enter a Number: ");

scanf("%d", &n);

if ((rear + 1) % MAX == front) {

printf("\nQueue Is Full\n");

} else {

if (front == -1) {

front = 0;

}

rear = (rear + 1) % MAX;

q[rear] = n;

printf("%d added to the queue\n", n);

}

}


void dequeue() {
```

```
if (front == -1) {
printf("\nQueue is Empty\n");
} else {
n = q[front];
printf("%d is deleted\n", n);
if (front == rear) {
front = -1;
rear = -1;
} else {
front = (front + 1) % MAX;
}
}
}
int isFull() {
return ((rear + 1) % MAX == front);
}
int isEmpty() {
return (front == -1);
}


void display()
{ int i;
if (front == -1) {
printf("Queue is Empty\n");
} else {
printf("Queue Elements: ");
i = front;
while (i != rear) {
printf("%d ", q[i]);
i = (i + 1) % MAX;
}
printf("%d\n", q[rear]);
}
}

int main() {
enqueue();
enqueue();
enqueue();
enqueue();
display();
dequeue();
display();
return 0;
}
```

Output:



```
Enter a Number: 25
25 added to the queue
Enter a Number: 26
26 added to the queue
Enter a Number: 27
27 added to the queue
Enter a Number: 28
28 added to the queue
Queue Elements: 25 26 27 28
25 is deleted
Queue Elements: 26 27 28
```

Conclusion:

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

➔ The Josephus Problem can be solved using a circular queue by simulating the elimination in a circle.

● People are enqueued into the circular queue.

● The modulo operator is used to maintain the circular nature of the queue (wrapping around).

● To eliminate the k-th person:

○ The queue dequeues and enqueues the first (k-1) people, simulating counting.

○ On the k-th operation, the person is dequeued and eliminated.

● This process is repeated until one person remains, representing the solution.