Experiment No.5
Implement Circular Queue ADT using array
Name: Jaffari Mohammed Ali Sayyed Naqi Ali
Roll No: 16
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:

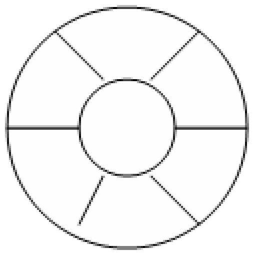
Circular queue is an data structure in which insertion and deletion occurs at an two ends rear and front respectively. Eliminating the disadvantage of linear queue that even though there is a vacant slots in array it throws full queue exception when rear reaches last element. Here in an circular queue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

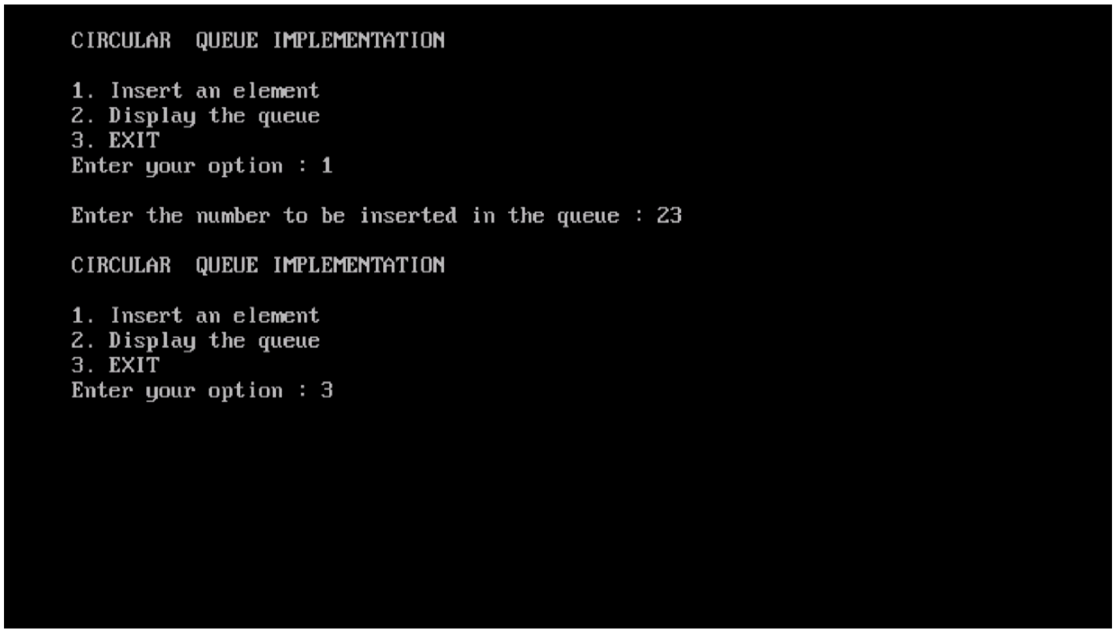
Code:

```
#include <stdio.h>

#include <conio.h>

#define MAX 10
```

```
int queue[MAX];

int front=-1, rear=-1;

void insert(void);


void display(void);

int main()

{

int option;

clrscr();

do

{

 printf("\n CIRCULAR  QUEUE IMPLEMENTATION ");

 printf("\n");

 printf("\n 1. Insert an element");

 printf("\n 2. Display the queue");

 printf("\n 3. EXIT");

 printf("\n Enter your option : ");

 scanf("%d", &option);

switch(option)

 {

 case 1:

 insert();

break;

case 2:
 display();

break;

 }

}while(option!=3);

getch();

return 0;

}

void insert()

{

int num;

printf("\n Enter the number to be inserted in the queue : ");

scanf("%d", &num);

if(front==0 && rear==MAX-1)

 printf("\n OVERFLOW");

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

{

front=rear=0;

 queue[rear]=num;

}

else if(rear==MAX-1 && front!=0)

{

rear=0;

 queue[rear]=num;

}
```

```
else
{
 rear++;
 queue[rear]=num;
}
}
void display()
{
int i;
printf("\n");
if (front ==-1 && rear==-1)
 printf ("\n QUEUE IS EMPTY");
else
{
 if(front<rear)
 {
 for(i=front;i<=rear;i++)
 printf("\t %d", queue[i]);
 }
 else
 {
 for(i=front;i<MAX;i++)
 printf("\t %d", queue[i]);
for(i=0;i<=rear;i++)
printf("\t %d", queue[i]);
}
}
}
```

Output:



```
CIRCULAR  QUEUE IMPLEMENTATION

1. Insert an element
2. Display the queue
3. EXIT
Enter your option : 1

Enter the number to be inserted in the queue : 23

CIRCULAR  QUEUE IMPLEMENTATION

1. Insert an element
2. Display the queue
3. EXIT
Enter your option : 3
```

Conclusion:

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

- The Josephus Problem is a well-known theoretical problem and counting-out game. It goes like this: N people (usually represented by numbers from 1 to N) stand in a circle, and they are counted off in turn. Every Mth person is eliminated from the circle, and the process continues until only one person remains. The problem is to find the position of the last person standing.

A circular queue can be used to solve the Josephus Problem efficiently. Here's how it works:

1. Setup: Create a circular queue and populate it with the initial list of N people, usually represented by numbers from 1 to N. Set the elimination count to M, and initialize a counter to keep track of how many people have been eliminated.

2. Elimination Process: Start with the first person in the queue. Dequeue the first M-1 people and enqueue them back into the queue in the same order. The Mth person is eliminated from the circle and removed from the queue. Increment the elimination counter.

3. Repeat:Continue this process, dequeuing M-1 people, eliminating the Mth, and incrementing the counter, until only one person remains in the queue.

4. Result: The last person remaining in the queue is the solution to the Josephus Problem.

Here's a step-by-step elaboration of the operation:

1. Create a circular queue with N elements, representing the N people in the circle.

2. Initialize variables for the elimination count (M) and a counter for eliminated people (initially set to 0).

3. Start with the first person in the queue, typically the one at the front.

4. Dequeue (remove) the first M-1 people from the queue and immediately enqueue (reinsert) them back into the queue in the same order.

5. The Mth person in the queue is the one to be eliminated. Remove them from the queue.

6. Increment the counter to keep track of the number of people eliminated.

7. Continue the process by returning to step 4 until there is only one person left in the queue.

8. The last person remaining in the queue is the solution to the Josephus Problem, representing the position of the survivor.

Using a circular queue for the Josephus Problem is efficient because it simulates the process of eliminating people in a circle without the need for shifting the entire circle after each elimination. The circular queue operation keeps the problem-solving process orderly and avoids the need for frequent data movement.