Experiment No.5
Implement Circular Queue ADT using array
Name:Nitish Jha
Roll No: 18
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:

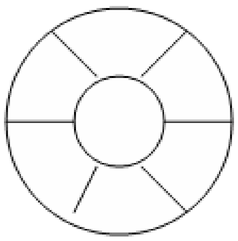
Circular queue is an data structure in which insertion and deletion occurs at an two ends rear and front respectively. Eliminating the disadvantage of linear queue that even though there is a vacant slots in array it throws full queue exception when rear reaches last element. Here in an circular queue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

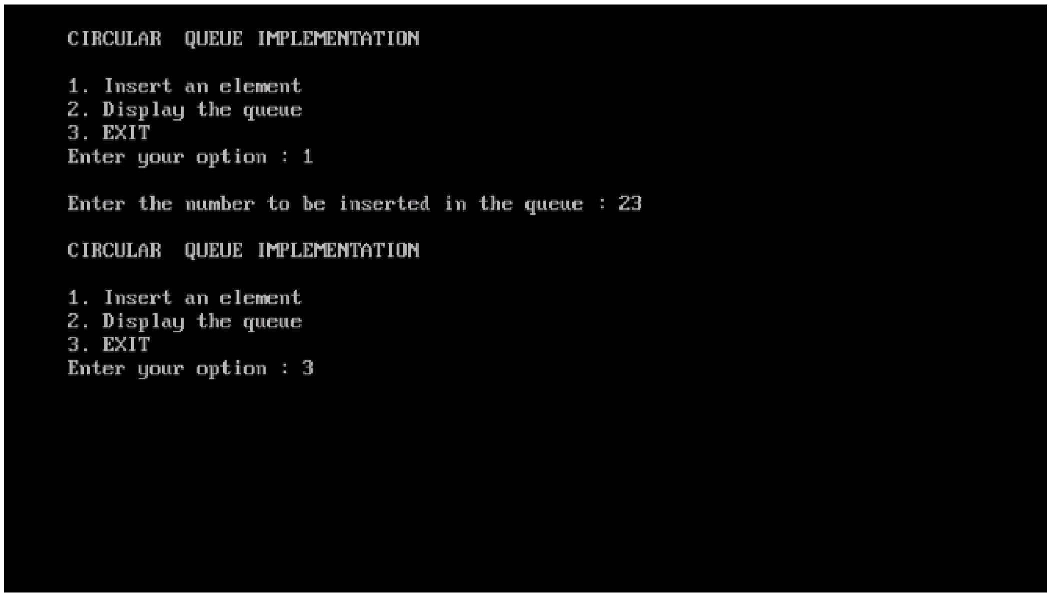
```
    1.  If front = 0
            Print “Queue is empty”
            Exit
    2.  else
            item = Q[front]
            if front = rear then
                rear = 0
                front=0
            else
                front = front+1
            end if
        end if
    3.  stop
```

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, a renowned theoretical challenge, discovers its solution through the clever implementation of a circular queue. This puzzle presents a scenario where N individuals, numbered from 1 to N, form a circle. They are tasked with counting off in a circle, with every Mth person being eliminated. The counting commences with person 1 and persists until only one individual remains.

The resolution of this enigmatic problem through a circular queue unfolds as follows:

1. Initialization: A circular queue with N elements is established, each element symbolizing a person with a number from 1 to N.

2. Kickoff: Person 1 takes the lead in the queue.

3. Count and Eliminate: The process involves counting M persons in the queue, and when the Mth individual is reached, they are removed (i.e., eliminated).

4. Circular Flow: A pivotal concept here is the cyclic nature of movement. After reaching the end of the queue, the counting and elimination wrap around to the queue's beginning.

5. Termination Criterion: This process repeats until only one person remains in the queue. This criterion marks the completion of the problem's solution.

The methods employed in addressing the Josephus Problem with a circular queue encompass:

- Insertion: It mirrors the initial enqueuing of the N individuals into the circular queue.
- Deletion (Elimination): After counting off M persons, the removal of the Mth person mirrors dequeuing, symbolizing their elimination.
- Circular Progression: The intrinsic idea is to ensure the counting and elimination proceed in a cyclic fashion, regardless of reaching the queue's end.
- Termination Condition: The process persists until only a single individual remains in the queue, signifying the problem's resolution.

The Josephus Problem brilliantly exemplifies how data structures like circular queues can be harnessed to model and adeptly solve real-world challenges in an organized and efficient manner. By executing the specified operations within the circular queue, the position of the last survivor in the circular arrangement can be ascertained.