Experiment No.5
Implement Circular Queue ADT using array
Name: Swarup Satish Kakade
Roll No: 19
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:

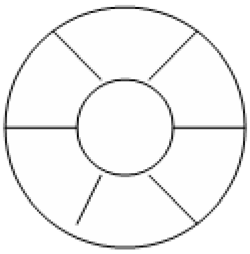
Circular queue is an data structure in which insertion and deletion occurs at an two ends rear and front respectively. Eliminating the disadvantage of linear queue that even though there is a vacant slots in array it throws full queue exception when rear reaches last element. Here in an circular queue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

Q. 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 puzzle and a frequently used programming challenge. It unfolds as follows: N individuals, often arranged in a circular formation, take turns counting around the circle. At every M-th count, the M-th person is removed from the circle, and this cycle continues until only a solitary person remains. The ultimate aim is to determine the position of this last remaining individual.
Efficiently tackling the Josephus Problem involves a clever strategy that employs a circular queue. The method can be summarized as follows:

1. **Initializing the Circular Queue:**
    - Create a circular queue with a capacity of N to represent the N individuals within the circular arrangement.
    - Populate this circular queue with values ranging from 1 to N, signifying the positions of each person in the circle.

2. **Commencing the Elimination Process:**
    - Maintain a count variable, initially set to 0, to keep track of the number of eliminated individuals.
    - While there are more than one person in the queue (i.e., the queue's size is greater than 1):
        - Dequeue (remove) the person situated at the front of the circular queue.
        - Increment the count by 1.
        - If the count matches M (indicating the person to be eliminated), perform the following actions:
            - Exclude this individual from the circle (i.e., do not reinsert them into the circular queue).
            - Reset the count to 0.
        - If the count does not equal M, enqueue the person back into the circular queue.
3. **Determining the Last Person Standing:**
    - Upon completing all eliminations, only one person will remain in the circular queue.
    - The position of this individual represents the solution to the Josephus Problem.

The crux of this approach lies in the circular nature of the queue, which efficiently models the process of encircling the group and eliminating individuals. The time complexity of this solution is O(N*M), with N representing the number of people and M indicating the counting interval.