Experiment No.5
Implement Circular Queue ADT using array
Name:Priyanka Narendra Bhandari
Roll No:02
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:

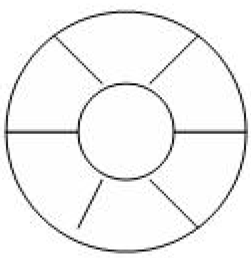
Circular queue is an data structure in which insertion and deletion occurs at an two ends rear and front respectively. Eliminating the disadvantage of linear queue that even though there is a vacant slots in array it throws full queue exception when rear reaches last element. Here in an circular queue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

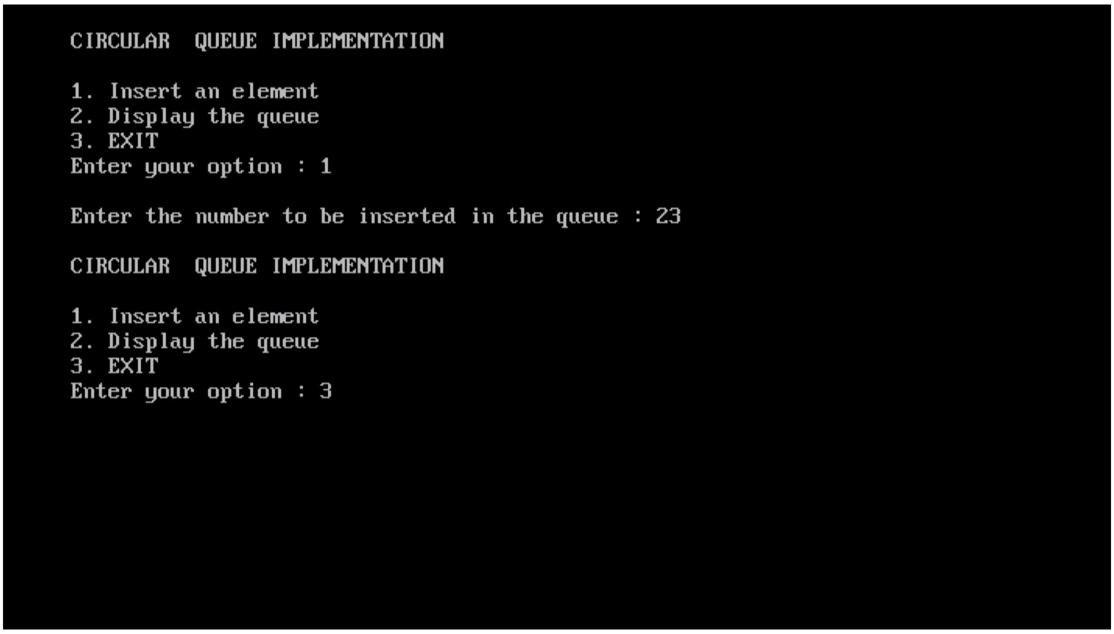
```
#include <stdio.h>

#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:



Conclusion:

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

- The Josephus Problem is a famous theoretical problem and can be resolved using a circular queue. The problem is stated as follows: N people (numbered 1, 2, 3, ..., N) are standing in a circle. They are required to count off in turn around the circle and eliminate every Mth person. The counting starts with person 1, and the process continues until only one person remains.

Here's how this problem can be resolved using a circular queue:

1. Initialize a circular queue with N elements, where each element represents a person numbered from 1 to N.

2. Start with person 1 as the current position in the queue.

3. Begin counting M persons in the queue, and when you reach the Mth person, remove them from the queue (i.e., eliminate them).

4. Continue counting and eliminating in a circular manner until only one person remains in the queue.

The operations used for resolving the Josephus Problem with a circular queue are:

- Insertion: In this case, insertion into the circular queue is similar to enqueuing the N people initially.

- Deletion (Elimination): After counting M persons, you remove the Mth person from the queue, which essentially means dequeuing them. This operation simulates the elimination of a person.

- Circular Movement: The key idea in this problem is that after reaching the end of the queue, you wrap around to the beginning, ensuring that the counting and elimination continue in a circular manner.

- Stopping Condition:The process continues until only one person remains in the queue. This stopping condition ensures that the problem is resolved.

The Josephus Problem is a classic example of how data structures like circular queues can be used to model and solve real-world problems in a structured and efficient way. By implementing the described operations within the circular queue, you can find the position of the last person who survives in the circle.