Experiment No.5
Implement Circular Queue ADT using array
Name:Yash Patil
Roll No:45
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:

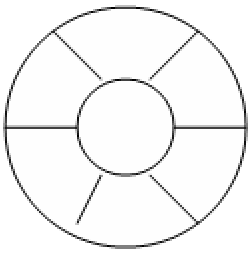
Circular queue is an data structure in which insertion and deletion occurs at an two ends rear and front respectively. Eliminating the disadvantage of linear queue that even though there is a vacant slots in array it throws full queue exception when rear reaches last element. Here in an circular queue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

```
void display(void);

int main() {

int option, val;

clrscr();

do {

printf("\n ***** MAIN MENU *****");

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

printf("\n 2. Delete an element");

printf("\n 3. Peek");

printf("\n 4. Display the queue");

printf("\n 5. EXIT");

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

scanf("%d", &option);

switch(option) {

case 1:

insert();

break;

case 2:

val = delete_element();

if(val!=–1)

printf("\n The number deleted is : %d", val);

break;

case 3:

val = peek();

if(val!=–1)
```

```
printf("\n The first value in queue is : %d", val);
 break;
case 4:
 display();
 break;
 }
 }
while(option!=5);
 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;
```

```
}
} int delete_element() {
int val;
if(front==–1 && rear==– 1)
{ printf("\n UNDERFLOW");
return –1;
}
val = queue[front];
if(front==rear)
front=rear=–1;
else {
if(front==MAX–1)
front=0;
else
front++;
} return val;
} int peek() {
if(front==–1 && rear==– 1) {
printf("\n QUEUE IS EMPTY");
return –1; }
else
{ return queue[front];
}
} void display() {
int i;
```

```
 printf("\n");

 if (front ==–1 && rear= =–1)

 printf ("\n QUEUE IS EMPTY");

else {

if(front<=rear;i++)

 printf("\t %d", queue[i]);

} else {

for(i=front;i<=rear;i++)

 printf("\t %d", queue[i]);

}

 }

 }
```

Output:

```
***** MAIN MENU *****
1. Insert an element
2. Delete an element
3. Peek
4. Display the queue
5. EXIT
Enter your option : 1
Enter the number to be inserted in the queue : 25
Enter your option : 2
The number deleted is : 25
Enter your option : 3
QUEUE IS EMPTY
Enter your option : 5
```

Conclusion:

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

1. Initialize a circular queue with the same number of elements as there are people in the circle.

2. Enqueue all people (or items) into the circular queue, assigning a position number to each.
3. Begin eliminating people by dequeuing every 'k-th' person from the queue.
4. Enqueue the eliminated person's position number back into the queue.
5. Repeat steps 3 and 4 until only one person (or item) remains in the queue.

The key operation used for resolving the Josephus Problem is dequeuing every 'k-th' person, effectively simulating the elimination process while maintaining the circular nature of the queue. Enqueuing the eliminated person's position number back into the queue ensures that the circle is maintained, and the process continues until the final person (or item) is left.

In summary, using a circular queue allows for an efficient and straightforward implementation of the Josephus Problem, ensuring that the 'k-th' person is eliminated in a circular fashion until only one remains.