



Experiment No.5
Implement Circular Queue ADT using array
Name:Umang Borse
Roll No:03
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:

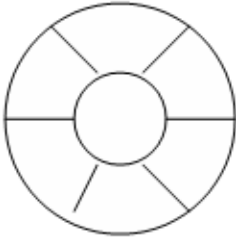
Circular queue is an data structure in which insertion and deletion occurs at an two ends rear and front respectively. Eliminating the disadvantage of linear queue that even though there is a vacant slots in array it throws full queue exception when rear reaches last element. Here in an circular queue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

If front = 0

front = 1

rear = 1

Q[front] = item

else

next = (rear mod length)

if next != front then

rear = next

Q[rear] = item

Else

Print "Queue is full"

End if

End if

stop

Algorithm : DEQUEUE()

Input : A circular queue with elements.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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.

```
If front = 0
    Print "Queue is empty"
    Exit
else
    item = Q[front]
    if front = rear then
        rear = 0
        front=0
    else
        front = front+1
    end if
end if
stop
```

Code:

```
#include <stdio.h>

#include <conio.h>

#define MAX 10

int queue[MAX];

int front=-1, rear=-1;
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
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;
```

```
}
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
}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;
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
}  
  
}  
  
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]);  
        }  
    }  
}
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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:

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

The Josephus Problem is a theoretical scenario that involves a group of people standing in a circle waiting to be executed. The problem revolves around determining the last remaining person in the circle after every (k) th person is executed. Circular queues can be used to simulate this problem.

Here's how the Josephus Problem can be resolved using a circular queue:

1. Setup: Place all the people in a circular queue, simulating the circle in which they stand.
2. Operation
 - Start with the first person in the queue.
 - Remove the first $(k-1)$ people from the queue and add them back to the end of the queue, simulating the process of counting and executing every (k) th person.



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

- The next person in the queue is the (k) th person from the initially removed set. Remove this person from the queue.

- Continue this process until only one person remains in the queue.

This operation is typically implemented using the following steps:

- Enqueue: Add a person to the circular queue.

- Dequeue: Remove and return the (k) th person from the queue, simulating their execution.

- Rotate: Rotate the queue to the left by $(k-1)$ positions, simulating the counting process.

- Peek: View the current person at the front of the queue without removing them.

- IsEmpty: Check if the queue is empty.

- Size: Return the number of people remaining in the queue.

By repeatedly rotating the circular queue and dequeuing the (k) th person, the Josephus Problem can be simulated and resolved. The last person remaining in the queue is the survivor of the execution process. Circular queues are instrumental in providing an efficient way to model and solve this problem.