Experiment No.5
Implement Circular Queue ADT using array
Name: Gautam D. Chaudhari
Roll No: 04
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:

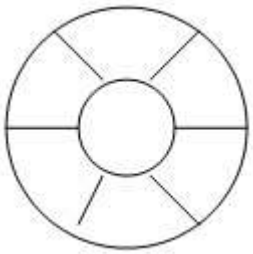
Circular queue is an data structure in which insertion and deletion occurs at an two ends rear and front respectively. Eliminating the disadvantage of linear queue that even though there is a vacant slots in array it throws full queue exception when rear reaches last element. Here in an circular queue 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 6

int queue[max]; // Array declaration
int front = -1;
int rear = -1;
void enqueue(int element) {
    if (front == -1 && rear == -1) {
            front = 0;
            rear = 0;
            queue[rear] = element;
    } else if ((rear + 1) % max == front) {
            printf("Queue is overflow.\n");
    } else {
            rear = (rear + 1) % max;
            queue[rear] = element;
    }
}
int dequeue() {
    if (front == -1 && rear == -1) {
            printf("Queue is underflow.\n");
    } else if (front == rear) {
            printf("The dequeued element is %d\n", queue[front]);
```

```
        front = -1;
        rear = -1;
    } else {
        printf("The dequeued element is %d\n", queue[front]);
        front = (front + 1) % max;
    }
}
void display() {
    int i = front;
    if (front == -1 && rear == -1) {
        printf("Queue is empty.\n");
    } else {
        printf("Elements in the queue are: ");
        while (i <= rear) {
            printf("%d, ", queue[i]);
            i = (i + 1) % max;
        }
        printf("\n");
    }
}

int main() {
    int choice = 1, x;
    clrscr();
    while (choice < 4 && choice != 0) {
        printf("Press 1: Insert an element\n");
```

```
        printf("Press 2: Delete an element\n");
        printf("Press 3: Display the elements\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to be inserted: ");
                scanf("%d", &x);
                enqueue(x);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
        }
    }

    return 0;
}
```

Output:

```
Enter the number to be inserted in the queue : 30

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 : 2

CIRCULAR  QUEUE IMPLEMENTATION

1. Insert an element
2. Display the queue
3. EXIT
Enter your option : 2

        25      30      2
CIRCULAR  QUEUE IMPLEMENTATION

1. Insert an element
2. Display the queue
3. EXIT
Enter your option :
```

Conclusion:

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

- Initialize a circular queue with "n" elements representing people.
- Enqueue all "n" people, assigning unique identifiers.
- Begin elimination:
  Dequeue and count "k-1" people.
  Dequeue the "k-th" person.
  Continue from the next person.
- Repeat until only one person is left in the queue.
- The last person is the solution to the problem.

The circular nature of the queue ensures that we move in a circle as we eliminate people, matching the problem's circular arrangement. The last person remaining in the queue is the solution to the Josephus Problem, representing the person who avoids elimination.