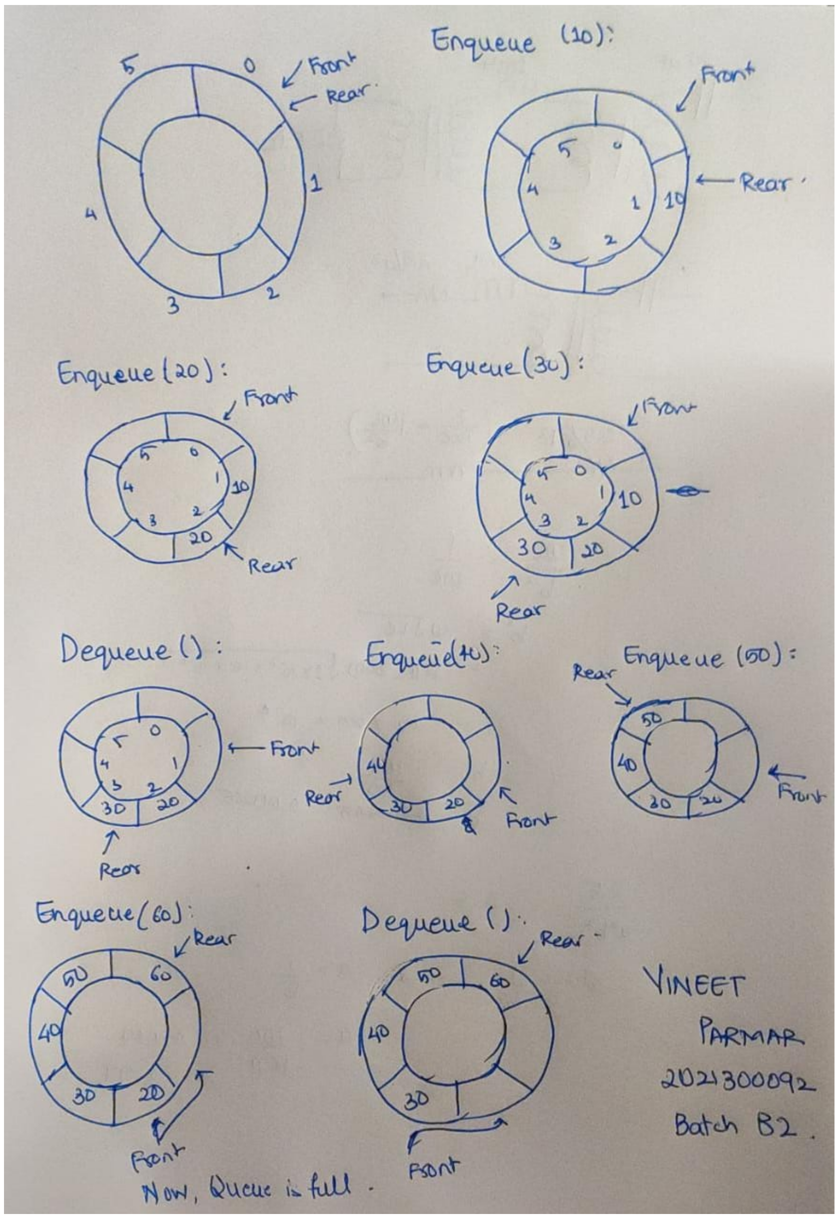
Name	**Vineet Parmar**
UID no.	**2021300092**
Experiment No.	2

AIM:	To implement a menu-driven program based around Circular Queue.
THEORY:	• Queue: A queue is defined as a linear data structure that is open at both ends and the operations are performed in First-In-First-Out (FIFO) order. We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the operation is first performed on that. • Characteristics of Queue: Queue can handle multiple data. We can access both ends. They are fast and flexible. • Queue Representation: Like stacks, Queues can also be represented in an array: In this representation, the Queue is implemented using the array. Variables used in this case are - Queue: the name of the array storing queue elements. - Front: the index where the first element is stored in the array representing the queue. - Rear: the index where the last element is stored in an array representing the queue. • Circular Queue: Circular Queue is an alternate that was designed to overcome the non-reusability of a linear queue. It is a special version of Queue where the last memory location of the queue is connected to the first memory location of the queue forming a circle. But there is an issue, in order for all conditions to work properly, one memory location has to be sacrificed. • Queue Operations: Enqueue: inserts an element at the rear of the queue. Dequeue: removes an element from the rear of the queue. Queue Front: examines the element at the front of the queue. Queue Rear: examines the element at the rear of the queue.

	• Circular Queue Initialization Conditions: Front , rear = 0 Queue Empty : front = rear Queue Full : (rear + 1) % size = front
ALGORITHM:	Class Circular Queue: Data Members: Integer array Queue[] Integer variables front, rear, size Void Enqueue(int el) 1.Check if queue is full or not 2.If no, then rear becomes (rear + 1) % size 3.Queue[rear] = el 4.If yes, then print queue full Void Dequeue() 1. Check if queue is empty or not 2. If no then front becomes (front + 1) % size 3. If yes, then print queue empty Void PrintQueue() 1.Check if queue is empty or not 2.If no, then check if rear < front 3.If yes then print elements of queue in loop from front + 1 to size – 1 and then from 0 to rear 4.If no, then print elements of queue in loop from front + 1 to rear. 5.If queue is empty, print queue empty. Int QueueFront() 1. If queue is empty, print queue empty else return queue[front + 1] Int QueueRear() 1. If queue is empty, print queue empty else return queue[rear] Boolean IsEmpty() 1. Return true if front equals rear, else return false Boolean isFull() 1. Return true if (rear + 1) % size equals front, else return false. Class Main Main Method 1. Input size of queue

	2. Display Options 3. Enter choice 4. If choice is 1, input element call enqueue() and printqueue() 5. If choice is 2, call dequeue() and printqueue() 6. If choice is 3, call Queuefront() 7. If choice is 4, call Queuerear() 8. If choice is 5, exit 9. Repeat steps 3 to 8 until choice is not equal to 5 10. STOP

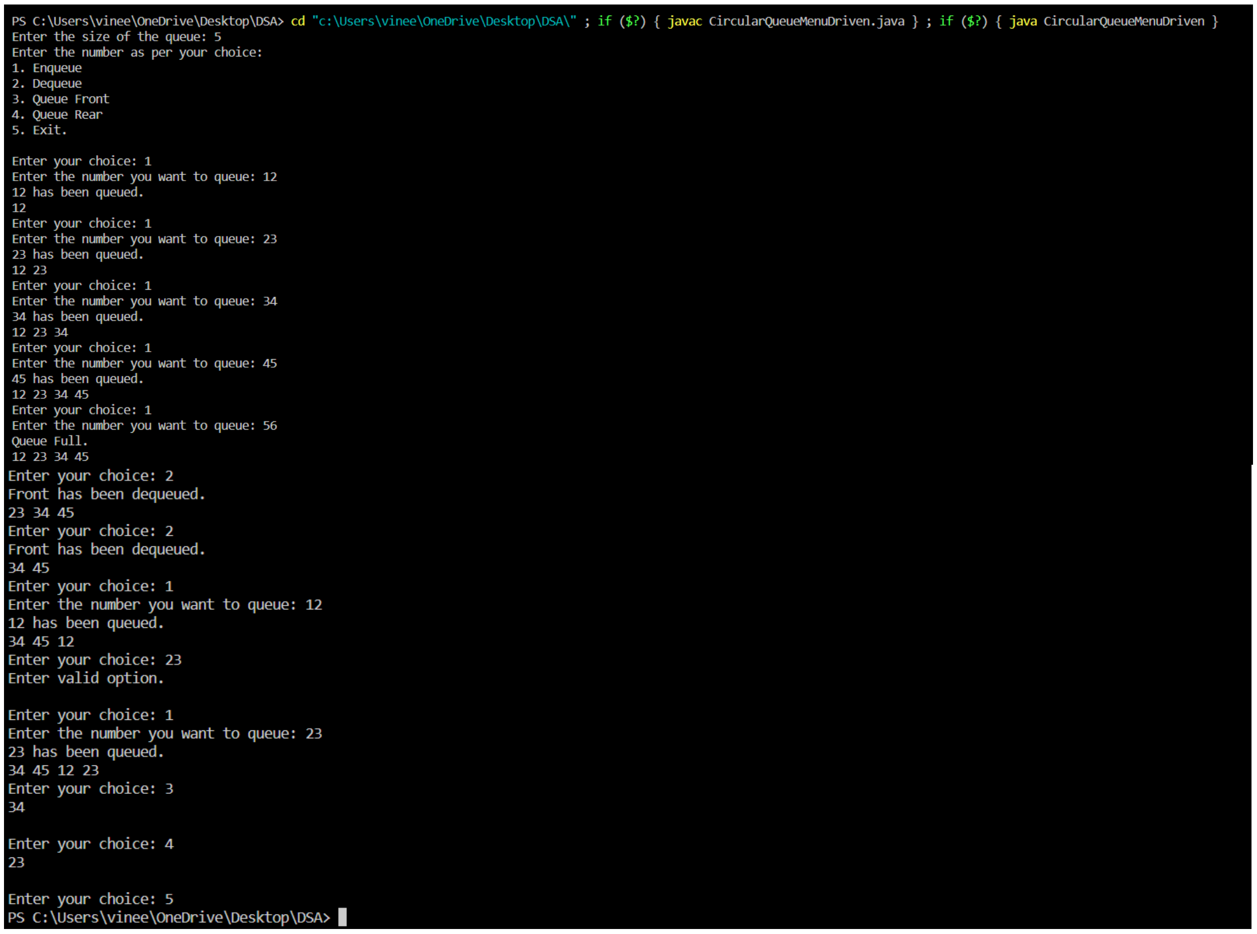
PROBLEM SOLVING:	 Front
	Rear
	Enqueue (10):
	Enqueue (20):
	Enqueue (30):
	Dequeue ():
	Enqueue(40):
	Enqueue (50):
	Enqueue(60):
	Dequeue ():
	Now, Queue is full.
	VINEET PARMAR
	2021300092
	Batch B2.

PROGRAM:

```
package dsa;
public class intCircularQueue
{
    int front, rear, size;
    int[] queue;
    public intCircularQueue(int n)
    {
        front = 0;
        rear = 0;
        size = n;
        queue = new int[n];
    }
    public void Enqueue(int c)
    {
        if(!isFull())
        {
            rear = (rear + 1) % size;
            queue[rear] = c;
            System.out.println(c + " has been queued.");
        }
        else
        {
            System.out.println("Queue Full.");
        }
    }
    public void Dequeue()
    {
        if(!isEmpty())
        {
            front = (front + 1) % size;
            System.out.println("Front has been dequeued.");
        }
        else
        {
            System.out.println("Queue Empty.");
        }
    }
    public int QueueFront()
    {
        return queue[(front + 1) % size];
    }
    public int QueueRear()
    {
        return queue[rear];
    }
    public void PrintQueue()
    {
        if(!isEmpty())
        {
            for(int i = front + 1; i <= (rear < front ? size - 1 : rear); i++)
                System.out.print(queue[i] + " ");
            if(rear < front)
                for(int i = 0; i <= rear; i++)
                    System.out.print(queue[i] + " ");
        }
    }
    public boolean isEmpty()
    {
        return front == rear;
    }
    public boolean isFull()
    {
        return ((rear + 1) % size == front);
    }
    public int size()
    {
        return (rear - front + size) % size;
    }
}

import dsa.*;
import java.util.*;
class CircularQueueMenuDriven
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the queue: ");
        int size = sc.nextInt();
        intCircularQueue q = new intCircularQueue(size);
        System.out.println("Enter the number as per your choice: ");
        System.out.println("1. Enqueue\n2. Dequeue\n3. Queue Front\n4.
Queue Rear\n5. Exit.");
        int ch = 0;
        do{
            System.out.print("\nEnter your choice: ");
            ch = sc.nextInt();
            switch(ch)
            {
                case 1:
                    System.out.print("Enter the number you want to queue: ");
                    int el = sc.nextInt();
                    q.Enqueue(el);
                    q.PrintQueue();
                    break;
                case 2:
                    q.Dequeue();
                    q.PrintQueue();
                    break;
                case 3:
                    System.out.println(q.isEmpty() ? "Queue Empty." :
q.QueueFront());
                    break;
                case 4:
                    System.out.println(q.isEmpty() ? "Queue Empty." :
q.QueueRear());
                    break;
                case 5:
                    System.exit(0);
                    break;
                default:
                    System.out.println("Enter valid option.");
                    break;
            }
        }while(ch != 5);
    }
}
```

OUTPUT:



```
PS C:\Users\vinee\OneDrive\Desktop\DSA> cd "c:\Users\vinee\OneDrive\Desktop\DSA\" ; if ($?) { javac CircularQueueMenuDriven.java } ; if ($?) { java CircularQueueMenuDriven }
Enter the size of the queue: 5
Enter the number as per your choice:
1. Enqueue
2. Dequeue
3. Queue Front
4. Queue Rear
5. Exit.

Enter your choice: 1
Enter the number you want to queue: 12
12 has been queued.
12
Enter your choice: 1
Enter the number you want to queue: 23
23 has been queued.
12 23
Enter your choice: 1
Enter the number you want to queue: 34
34 has been queued.
12 23 34
Enter your choice: 1
Enter the number you want to queue: 45
45 has been queued.
12 23 34 45
Enter your choice: 1
Enter the number you want to queue: 56
Queue Full.
12 23 34 45
Enter your choice: 2
Front has been dequeued.
23 34 45
Enter your choice: 2
Front has been dequeued.
34 45
Enter your choice: 1
Enter the number you want to queue: 12
12 has been queued.
34 45 12
Enter your choice: 23
Enter valid option.

Enter your choice: 1
Enter the number you want to queue: 23
23 has been queued.
34 45 12 23
Enter your choice: 3
34

Enter your choice: 4
23

Enter your choice: 5
PS C:\Users\vinee\OneDrive\Desktop\DSA>
```

CONCLUSION:

In this experiment, we learned about the basic concepts of queue. We learned about the data members front, rear, queue[] and size and the operations enqueue, dequeue, queue front, queue rear, etc along with initialisation, full and empty conditions. After that, we moved on to Circular Queue to overcome the reusability issue of a linear queue. Herein also, we learnt about the operations enqueue, dequeue, etc and the initialisation, full and empty conditions as in how they are different from the latter. We learnt that in order for all conditions to exist properly, a memory slot has to be sacrificed. Lastly, with all this knowledge, we made a menu driven program of a circular queue covering all the operations such as enqueue, dequeue, print queue, etc.