Practical No 6

Aim: Write A Program To Implement Linear Queue and Circular Queue

```
class LinearQueue:
    def __init__(self, capacity):
        self.capacity = capacity
        self.queue = [None] * capacity
        self.front = 0
        self.rear = 0
    def enqueue(self, task):
        if self.rear == self.capacity:  # No space left, even if front moved
            print("Linear Queue Overflow! Cannot enqueue:", task)
            return
        self.queue[self.rear] = task
        self.rear += 1
        print(f"Enqueued (Linear): {task}")
    def dequeue(self):
        if self.front == self.rear:
            print("Linear Queue Underflow! Cannot dequeue.")
            return
        task = self.queue[self.front]
        self.front += 1
        print(f"Dequeued (Linear): {task}")
        return task
    def display(self):
        if self.front == self.rear:
            print("Linear Queue is empty.")
        else:
            print("Linear Queue:", self.queue[self.front:self.rear])

class CircularQueue:
    def __init__(self, capacity):
        self.capacity = capacity
        self.queue = [None] * capacity
        self.front = -1
        self.rear = -1
    def enqueue(self, task):
        # Queue is full when next position of rear == front
        if (self.rear + 1) % self.capacity == self.front:
            print("Circular Queue is Full! Cannot enqueue:", task)
            return
        if self.front == -1:  # First element
            self.front = 0
        self.rear = (self.rear + 1) % self.capacity
        self.queue[self.rear] = task
        print(f"Enqueued (Circular): {task}")
    def dequeue(self):
        if self.front == -1:
            print("Circular Queue is Empty! Cannot dequeue.")
            return
        task = self.queue[self.front]
```

```
        if self.front == self.rear:  # Queue has only one element
            self.front = -1
            self.rear = -1
        else:
            self.front = (self.front + 1) % self.capacity
        print(f"Dequeued (Circular): {task}")
        return task
    def display(self):
        if self.front == -1:
            print("Circular Queue is empty.")
            return
        print("Circular Queue:", end=" ")
        i = self.front
        while True:
            print(self.queue[i], end=" ")
            if i == self.rear:
                break
            i = (i + 1) % self.capacity
        print()

# Example Simulation of Task Scheduling
if __name__ == "__main__":
    tasks = ["Task1", "Task2", "Task3", "Task4", "Task5"]
    print("=== Linear Queue Simulation ===")
    lq = LinearQueue(3)
    for task in tasks:
        lq.enqueue(task)
    lq.display()
    lq.dequeue()
    lq.dequeue()
    lq.enqueue("Task6")
    lq.display()
    print("\n=== Circular Queue Simulation ===")
    cq = CircularQueue(3)
    for task in tasks:
        cq.enqueue(task)
    cq.display()
    cq.dequeue()
    cq.enqueue("Task6")
    cq.display()
```

```
=== Linear Queue Simulation ===
Enqueued (Linear): Task1
Enqueued (Linear): Task2
Enqueued (Linear): Task3
Linear Queue Overflow! Cannot enqueue: Task4
Linear Queue Overflow! Cannot enqueue: Task5
Linear Queue: ['Task1', 'Task2', 'Task3']
Dequeued (Linear): Task1
Dequeued (Linear): Task2
Linear Queue Overflow! Cannot enqueue: Task6
Linear Queue: ['Task3']
=== Circular Queue Simulation ===
Enqueued (Circular): Task1
Enqueued (Circular): Task2
Enqueued (Circular): Task3
Circular Queue is Full! Cannot enqueue: Task4
Circular Queue is Full! Cannot enqueue: Task5
Circular Queue: Task1 Task2 Task3
Dequeued (Circular): Task1
Enqueued (Circular): Task6
Circular Queue: Task2 Task3 Task6
```