**Theory**

**Circular Queue:**

A circular queue is a data structure that represents a collection of elements arranged in a circular fashion. Unlike a regular queue, where the front and rear are fixed, in a circular queue, when the rear reaches the end, it wraps around to the beginning of the array.

Operations in a Circular Queue:

1. Enqueue (Insertion): Adds an element to the rear of the queue.

2. Dequeue (Deletion): Removes an element from the front of the queue.

3. Front: Returns the front element of the queue.

4. Rear: Returns the rear element of the queue.

5. Overflow: Checks if the queue is full.

6. Underflow: Checks if the queue is empty.

**Recursion in Stacks:**

Recursion is a programming concept where a function calls itself directly or indirectly to solve a particular problem. In the context of stacks, recursion often involves using the call stack to manage function calls. A recursive function is a programming construct where a function calls itself during its execution, allowing for a concise and elegant solution to problems with repetitive structures. It typically consists of two main components: the base case and the recursive case.

The base case serves as the termination condition, preventing the function from calling itself indefinitely. In contrast, the recursive case involves the function calling itself with a reduced or simpler version of the original problem. As the recursive calls unfold, the call stack manages the sequence of function calls and returns, and the final result is obtained by resolving the calls in reverse order. This technique is demonstrated in the example of calculating the factorial of a number, where the function progressively breaks down the problem until it reaches the base case, resulting in a clear and modular solution. Understanding recursion is crucial for tackling problems with repetitive structures in an efficient and readable manner.

**Problem:** Write a program to enqueue, dequeue and display the elements in a circular queue.

**Algorithm:**

1. Initialize the circular queue with the required size (maxSize), front, and rear pointers.

- Set front = -1

- Set rear = -1

- Create an array of size maxSize to store elements.

2. Check if the circular queue is empty:

- If front is equal to -1 and rear is equal to -1, the queue is empty.

3. Check if the circular queue is full:

- If (rear + 1) % maxSize is equal to front, the queue is full.

4. Enqueue (Insertion) operation:

- Check if the queue is full.

- If full, display an overflow message.

- If not full, increment the rear pointer.

- Set rear = (rear + 1) % maxSize.

- Insert the element at the rear position in the array.

5. Dequeue (Deletion) operation:

- Check if the queue is empty.

- If empty, display an underflow message.

- If not empty, check if there is only one element in the queue.

- If true, set front and rear to -1.

- If false, increment the front pointer.

- Set front = (front + 1) % maxSize.

6. Peek (Front) operation:

- Check if the queue is empty.

- If empty, display an underflow message.

- If not empty, return the element at the front position.

7. Display the circular queue:

- Display all elements from front to rear in a circular manner.

8. Exit.