Aim: Queue Implementation.

Objectives: The main aim is to understand and implement the stack data structure and its operations such as push and pop using arrays. Additionally, apply the stack for practical problems like infix-to-postfix conversion, parenthesis balancing, and postfix expression evaluation.

Tools Used: Visual Studio Code.

Concept:

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. This means that the last element added to the stack is the first one to be removed.

Operations of Stack**:**

1) **Push (Insertion)**
   Push operation adds an element to the top of the stack.
   Example:
   Initial Stack: [10, 20, 30]
   Push 40:
   Resulting Stack: [10, 20, 30, 40]

Algorithm:

- Check if the stack is full. If full, return "Stack Overflow".
- Increment the top pointer.
- Insert the new element at the position pointed to by the top.

Steps:

1.) if top == MAX - 1: print("Stack Overflow")
2.) else: stack[++top] = element

2) **Pop (Deletion)**
   Pop operation removes and returns the topmost element of the stack.
   Example:
   Initial Stack: [10, 20, 30, 40]
   Pop:
   Resulting Stack: [10, 20, 30]

Algorithm:

- Check if the stack is empty. If empty, return "Stack Underflow".
- Access the topmost element.
- Decrement the top pointer.

Steps:

1.) if top == -1: print("Stack Underflow")
2.) else: element = stack[top--]

Problem Statement:

1. Implement Queue Using array.
2. Circular Queue Using array

Solution:

1.) Implement Queue Using array.

Observation: A stack's LIFO (Last In, First Out) nature makes it useful for many tasks. Using arrays for stack implementation is simple but limited by size. Converting infix to postfix uses the stack to handle operator precedence. Balancing parentheses is easily managed by pushing and popping brackets. For postfix evaluation, the stack stores numbers and calculates results step by step.