Stack

Definition:

A stack is a 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.

Example:

stack of plates where you add a plate to the top and remove it from the top.

Characteristics of Stacks:

1. **LIFO Principle:** The Last In, First Out principle is a fundamental characteristic of stacks. The last element added is the first one to be removed.
2. **Operations:**
    - **Push:** Adds an element to the top of the stack.
    - **Pop:** Removes the element from the top of the stack.
    - **Peek (or Top):** Retrieves the element at the top of the stack without removing it.
3. **Implementation:**

- Stacks can be implemented using arrays or linked lists. Arrays are a simple and efficient choice when the size of the stack is fixed, while linked lists are more flexible in handling dynamic sizes.

Applications of Stacks:

1. **Expression Evaluation:**
    - Stacks are employed to evaluate arithmetic expressions, especially in languages that use postfix (Reverse Polish Notation) or infix-to-postfix conversion.
2. **Undo Mechanisms:**
    - Stacks are utilized in implementing undo functionalities in applications. Each user action that can be undone is stored on a stack, and undoing involves popping actions off the stack.
3. **Browser History:**
    - Browser navigation history can be implemented using a stack. Each visited page is pushed onto the stack, and navigating back involves popping from the stack.

Time Complexity:

1. **Push, Pop, and Peek:**
    - In most cases, the time complexity for these operations is O(1), meaning they have constant time complexity.
2. **Space Complexity:**
    - The space complexity of a stack is O(n), where n is the number of elements in the stack.

Infix To Postfix:

1. **Initialize an empty stack.**
2. **Initialize an empty output list for the postfix expression.**
3. **Scan the infix expression from left to right.**
    - If the current token is an operand, add it to the output list.
    - If the current token is an operator, pop operators from the stack and add them to the output list until the stack is empty or the top of the stack has lower precedence than the current operator. Then push the current operator onto the stack.
    - If the current token is an open parenthesis '(', push it onto the stack.
    - If the current token is a close parenthesis ')', pop operators from the stack and add them to the output list until an open parenthesis is encountered. Pop and discard the open parenthesis from the stack.
4. **Pop any remaining operators from the stack and add them to the output list.**
5. **The output list is the postfix expression.**

Postfix Evaluation:

1. **Initialize an empty stack to store operands.**
2. **Iterate through each character in the postfix expression.**
    - For each character:
        - If the character is an operand (numeric digit):
            - Push it onto the stack.
        - If the character is an operator (+, -, *, /, ^):

        - Pop the required number of operands from the stack (usually two for binary operators).
        - Perform the operation using the popped operands and the operator.
        - Push the result back onto the stack.

3. **After processing all characters in the postfix expression:**
    - The result should be the only element left on the stack.
    - Pop the result from the stack.