Ex : 3

Title: Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression and Evaluatethe converted postfix expression.

Problem Description: The problem involves implementing a function that convert the Infix Expression to Postfix Expression anda function that evaluates the converted expression and prints the result. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.

Method: In this program, Infix expression is given as an input andthe function for converting infix to postfix expression. After obtaining the postfix expression for each alphanumeric operand's values are given at the runtime and display the resultant value.

Choose the best data structure for these operations.

Theory Reference: Module 2

Explanation

1. **Define the precedence of operators:**

   Define the precedence levels for operators like `+`, `-`, `*`, `/`, `%`, and `^`. Higher precedence operators should be placed above lower precedence ones.

2. **Convert the Infix to Postfix:**

   We use a stack to convert the infix expression (which may have parentheses) into a postfix expression.

3. **Evaluate the Postfix expression:**

   Use a stack to evaluate the postfix expression. Each operand is pushed onto the stack, and operators pop the required number of operands to perform the operation, pushing the result back onto the stack.

4. **Handle Parentheses:**

Parentheses should control the precedence of operators and should be handled properly during both conversion and evaluation.

5. **Support for Alphanumeric Operands:**

   Alphanumeric operands should be stored directly in the postfix expression. During evaluation, variables need to be given a value or assumed.

Algorithm

Step 1: Initialize:

- tos = -1, top = -1
- istack[] for operators, stack[] for operands.

Step 2: Convert Infix to Postfix:

- Push '(' onto istack and add ')' to the end of Q.
- For each character in the infix expression:
  - If operand (alphanumeric), add to postfix.
  - If (, push to istack.
  - If), pop and append to postfix until (is found, then pop (.
  - If operator:
    - While precedence of operator <= precedence of top of stack, pop and append to postfix.
    - Push operator to stack.
- After scanning all characters, pop remaining operators from istack and append to postfix.

Step 3: Evaluate Postfix:

- For each character in postfix:
  - If operand:
    - Prompt for value and push to stack.

    - If operator:
        - Pop top two values from stack, perform operation, push result back.
- Pop and display final result.

Step 4: Main Execution

- Prompt the user to **input an infix expression**.
- **Convert the infix expression to a postfix expression** using the convertip() function.
- Display the **converted postfix expression**.
- **Evaluate the postfix expression** using the evaluate() function and display the result.

Step 5: Stop