**1. isOperator Function**

* Checks if a character is one of the supported operators (**+**, **-**, **\***, **/**).

**2. precedence Function**

* Assigns precedence values to operators (**+** and **-** have lower precedence than **\*** and **/**).

**3. infixToPostfix Function**

* Converts an infix expression to postfix.
* Handles operands, operators, and parentheses.
* Uses a stack to manage the conversion process.

**4. evaluatePostfix Function**

* Evaluates a postfix expression.
* Uses a stack to process operands and operators.
* Applies operators to operands to obtain the result.

**5. main Function**

* Takes an infix expression as input from the user.
* Calls **infixToPostfix** to convert the infix expression to postfix.
* Prints the resulting postfix expression.
* Calls **evaluatePostfix** to compute the final result.
* Displays the final result.

**Note:**

* The program assumes that the input expressions are valid and do not contain syntax errors.
* It performs basic arithmetic operations on integers.
* The conversion and evaluation functions use stacks to handle expressions and results.
* The user interacts with the program by providing infix expressions.

Algorithm:

1. \*Function Definitions:\*

1. isOperator(char c): Returns true if the character is one of the operators (+, -, \*, /).

2. precedence(char op): Returns the precedence of the operator.

3. infixToPostfix(string expression): Converts an infix expression to a postfix expression.

4. evaluatePostfix(string expression): Evaluates a postfix expression.

2. \*Infix to Postfix Conversion:\*

1. Initialize an empty stack of characters (stack) and an empty string (postfix).

2. Iterate through each character (c) in the infix expression.

3. If c is a space, skip to the next iteration.

4. If c is a digit or alphabet, append it to the postfix expression.

5. If c is '(', push it onto the stack.

6. If c is ')', pop and append operators from the stack to the postfix expression until '(' is encountered, then pop '('.

7. If c is an operator:

- Pop and append operators from the stack to the postfix expression while the stack is not empty and the precedence of the operator on the stack is greater than or equal to the precedence of c.

- Push c onto the stack.

8. After processing all characters, pop and append any remaining operators from the stack to the postfix expression.

9. Return the postfix expression.

3. \*Postfix Evaluation:\*

1. Initialize an empty stack of integers (stack).

2. Iterate through each character (c) in the postfix expression.

3. If c is a digit, push its integer value onto the stack.

4. If c is an operator:

- Pop the top two elements (operand2 and operand1) from the stack.

- Perform the operation based on the operator and push the result back onto the stack.

5. After processing all characters, the final result is on the top of the stack.

6. Return the result.

4. \*Main Function:\*

1. Read the infix expression from the user.

2. Call infixToPostfix to convert the infix expression to postfix.

3. Print the postfix expression.

4. Call evaluatePostfix to evaluate the postfix expression.

5. Print the result.

Top of Form