Aim: Evaluation of infix and prefix expression

Theory:

Infix, Postfix and Prefix notations are three different but equivalent notations of writing algebraic expressions.

1) INFIX expression:

=> Operator is between two operand

"operand 1" OPERATOR "operand 2"

eg: A+B, A*(B+C)

1) POSTFIX expression:

=> Operator is after bot operand

"operand 1" "operand 2" OPERATOR

eg: AB+, ABC+*

1) PREFIX expression:

=> Operator is before both operand

OPERATOR "operand 1"   "operand 2"

eg: A+B, *A+BC

Benefits of using prefix and postfix expressions:

- Although it is easy to write expressions using infix notation, computers find it difficult to evaluate as they need a lot of information to evaluate the expression.
- Information is needed about operator precedence, associativity rules, and brackets which overrides these rules.
- A postfix or Prefix expression does not even follow the rules of operator precedence and associativity.
- So, computers work more efficiently with expressions written using prefix and postfix notations.

****

Algorithms:

Algorithm to evaluate a prefix expression

Step 1: Accept the Prefix expression

Step 2: Repeat until all characters in the prefix expression are scanned

a. Scan the prefix expression from right,one charcter at time

b. If an operand is encountered, push it on the stack

c. If an operator X is encountered, then

a. pop the top two elements from the stack as A and B

b. Evaluate A X B, where A was the topmost element and B was the element below A.

c. Push the result of evaluation on the stack

[END OF IF]

Step 4: SET RESULT equal to the topmost element of the stack

Step 5: END

Algorithm to evaluate infix expression:

There are two steps :-

Step 1=> Convert infix expression to its equivalent postfix expression

Algorithm to convert an Infix notation into postfix notation

Step 1: Add ‘)” to the end of the infix expression

Step 2: Push “(“ on to the stack

Step 3: Repeat until each character in the infix notation is scanned

>IF a “(“ is encountered, push it on the stack

>IF an operand (whether a digit or an alphabet) is encountered, add it to the postfix expression.

>IF a “)” is encountered, then;

a. Repeatedly pop from stack and add it to the postfix expression until a “(” is encountered.

b. Discard the “(“. That is, remove the “(“ from stack and do not add it to the postfix expression

>IF an operator X is encountered, then;

Repeatedly pop from stack and add each operator (popped from the stack which has the same precedence or a higher precedence than X) to the postfix expression.

If precedence of popped operator is less than that of x, push popped operator back to stack.

b. Push the operator X to the stack.

Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty

Step 5: END

Step 1=> Evaluate postfix expression

Algorithm to evaluate a postfix expression

Step 1: Add a ")" at the end of the postfix expression

Step 2: Scan every character of the postfix expression and repeat steps 3 and 4 until ")"is encountered

Step 3: IF an operand is encountered, push it on the stack

IF an operator X is encountered, then

a. pop the top two elements from the stack as A and B

b. Evaluate B X A, where A was the topmost element and B was the element below A.

c. Push the result of evaluation on the stack

[END OF IF]

Step 4: SET RESULT equal to the topmost element of the stack

Step 5: END

Thank you
