Exercise 4 Stack Applications

4. a) Write a C++ Program for Evaluation of postfix expression.
b) Write a C++ Program for conversion of Infix to postfix notation.

Objective: The objective of this exercise is to enable you to perform evaluation of postfix expression and converting infix notation into postfix notation.

Procedure and description:
Generally in all arithmetic expression the operators are placed in between the operands, this is called infix notation.
A+B and (X+Y)* Z

In some type of notation the operator is placed before its two operands, this is called prefix notation or polish notation.
+AB and *+XYZ

In another type of notation the operator is placed after its two operands, this is called postfix notation or reverse polish notation.
AB+ and XY+Z*

a) Evaluation of a postfix expression
Suppose we have an arithmetic expression written in postfix notation. By using STACK we are going to evaluate the expression.

Algorithm: Evaluation of Postfix Expression.

Step 1: Add a right parenthesis “)” at the end of X. [This acts as a sentinel]
// X: Arithmetic expression written in postfix notation

Step 2: Scan X from left to right and repeat Steps 3 and 4 for each element of X until the sentinel “)” is encountered.

Step 3: [Start of If structure]
If an operand is encountered put it on STACK.

Step 4: if an operator is encountered, then:
a) Remove the two top element of STACK,
b) Evaluate the two top elements of STACK.
c) After evaluation place result value in STACK.
[End of If structure]
[End of Step 2 loop.]

Step 5: Set VALUE equal to the top element on STACK.

Step 6: Exit.

Expected Output:

After execution of program enter postfix expression (“12*3+ ").it executes algorithm steps and then calculates the result value.

INPUT: Postfix Expression: 12*3+ //infix :(1*2) +3

Result: 5

b) Conversion of Infix to postfix notation

Generally the expression we use for algebraic notations are in infix form, for conversion of this infix notation to postfix notation,

In this method, we read each character one by one from the infix expression and follow a certain set of steps. The Stack is used to hold only the operators of the expression. This is required to ensure that the priority of operators is taken into consideration during conversion

The following algorithm helps us to proceed with. Infix is the input string and the output what we get after the process is the postfix string.

Algorithm:

Step 1: Initialize a STACK as empty in the beginning.

Step 2: Inspect the infix string from left to right.

While reading character from a string we may encounter

Operand - add with the postfix string.

Operator – if the stack is empty push the operator into stack, if any operator available with the stack compare the current operator precedence with the top Stack if it has higher precedence over the current one pop the stack and add with the post string else push the current operator to the stack. Repeat this process until the stack is empty.

Left Parenthesis: Push in to the STACK.

Right Parenthesis: Pop everything until you get the left parenthesis or end of STACK.

Step 3: Repeat Process with the infix string until all the
Characters are read.

Step 4: Check for stack status if it is not empty add top Stack to
Postfix string and repeat this process until the
STACK is empty

Step 5: return the postfix string.

Step 6: Exit.

Expected Output:

After execution of program. Enter input infix expression (a + b * c - d).it performs algorithm steps and then converts the infix expression into resultant postfix output expression (abc*+d-).