Data Structures and Algorithms using C
Unit: 2
Topic: Infix to Postfix and Infix to Prefix Conversion

Infix to Postfix Conversion:

Infix expression: The expression of the form a op b i.e. when an operator (op) is in-between every pair of operands.

Postfix expression: The expression of the form a b op i.e. when an operator is followed for every pair of operands.

Why postfix representation of the expression?
The compiler scans the expression either from left to right or from right to left.

Consider the below expression: a op1 b op2 c op3 d
If op1 = +, op2 = *, op3 = +

The compiler first scans the expression to evaluate the expression b * c, then again scan the expression to add a to it. The result is then added to d after another scan.

Now, Infix expressions are readable and solvable by humans. We can easily distinguish the order of operators, and also can use the parenthesis to solve that part first during solving mathematical expressions. The computer cannot differentiate the operators and parenthesis easily, that's why postfix conversion is needed.

Converting Infix to postfix form:

Expression: (A+B) / (C-D)

Solution:

X = (AB+)

Y = (CD-)

i.e. X/Y -> XY/

Replacing the values: AB+CD-/

To convert infix expression to postfix expression, we will use the stack data structure. By scanning the infix expression from left to right, when we will get any operand, simply add them to the postfix form, and for the operator and parenthesis, add them in the stack maintaining the precedence of them.

The repeated scanning makes it very in-efficient. It is better to convert the expression to postfix (or prefix) form before evaluation. Further, the postfix expressions can be evaluated easily using a stack.

Algorithm:

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, output it.
3. Else,

3.1 If the precedence of the scanned operator is greater than the precedence of the operator in the stack (or the stack is empty or the stack contains a '('), push it.
3**.2** Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

4. If the scanned character is an '(', push it to the stack.
5. If the scanned character is an ')', pop the stack and and output it until a '(' is encountered, and discard both the parenthesis.
6. Repeat steps 2-6 until infix expression is scanned.
7. Print the output
8. Pop and output from the stack until it is not empty.

Solved Question:

Consider the following arithmetic infix expression Q:

Q: A + (B * C - (D / E↑F) * G) * H

Transform Q into its equivalent postfix expression P.

Step 1: A + (B * C - (D / E↑F) * G) * H)

Step 2: The corresponding STACK form -

Symbol Scanned	STACK	Expression P
(1) A	(	A
(2) +	(+	A
(3) (	(+ (	A
(4) B	(+ (	AB
(5) *	(+ (*	AB
(6) C	(+ (*	ABC
(7) –	(+ (-	ABC*
(8) (	(+ (- (	ABC*
(9) D	(+ (- (	ABC*D
(10) /	(+ (- (/	ABC*D
(11) E	(+ (- (/	ABC*DE
(12) ↑	(+ (- (/ ↑	ABC*DE
(13) F	(+ (- (/ ↑	ABC*DEF
(14))	(+ (-	ABC*DEF↑/
(15) *	(+ (- *	ABC*DEF↑/
(16) G	(+ (- *	ABC*DEF↑/G
(17))	(+	ABC*DEF↑/G*-
(18) *	(+ *	ABC*DEF↑/G*-
(19) H	(+ *	ABC*DEF↑/G*-H
(20))		ABC*DEF↑/G*-H*+

Infix to Prefix conversion:

Given two operands a and b and an operator op, the infix notation implies that op will be placed in between a and b i.e a op b and when the operator is placed before the operands i.e op a b, the expression in prefix notation.

Algorithm:

1. Reverse the expression
2. Read expression from Left-to-Right and
    - If an operand is read, copy it to the output **(left-to-right)**,
    - If a right/closing parenthesis is read, push it into the stack,
    - If a left parenthesis is encountered, the operator at the top of the stack is popped off the stack and copied to the output.
    - As the symbol at the top of the stack is a right parenthesis both parentheses are discarded,
3. If an operator scanned:
    - If it has a higher or equal precedence than the operator at the top of the stack, the operator is pushed onto the stack,

- If the precedence of the operator is lower than the precedence of the operator at the top of the stack, the operator at the top of the stack is popped and copied to the output,
- When the end of the expression is reached on the input scan, remaining operators in the stack are popped and copied to the output.

4. Reverse the output.

__Infix to Prefix form__

Expression: (A / (B^C)) -D

Solution:

X = (B^C) => (^BC)

Y = (A/(^BC)) => (/A^BC)

Z = (/A^BC) – D => –(/A^BC)D

i.e. –/A^BCD

__Solved Question:__

Consider the following arithmetic infix expression Q:

2*3/(2-1)+5*(4-1)

Transform Q into its equivalent prefix expression P.

Solution:

Step 1: Read expression from R to L or Reverse the string and read it from left to right.

Step 2: Corresponding stack representation

Symbol Scanned	Stack	Prefix Expression
1.)	)	
2. 1	)	1
3. -	) -	1
4. 4	) -	1 4
5. (		1 4 -
6. *	*	1 4 -
7. 5	*	1 4 - 5
8. +	+	1 4 - 5 *
9.)	+)	1 4 – 5 *
10. 1	+)	1 4 – 5 * 1
11. -	+) -	1 4 – 5 * 1
12. 2	+) -	1 4 – 5 * 1 2
13. (	+	1 4 – 5 * 1 2 -
14. /	+ /	1 4 – 5 * 1 2 -
15. 3	+ /	1 4 – 5 * 1 2 - 3
16. *	+ / *	1 4 – 5 * 1 2 – 3
17. 2	+ / *	1 4 – 5 * 1 2 – 3 2
		1 4 – 5 * 1 2 – 3 2 * / +

*** End***