DATA STRUCTURES
USING "C"

(Polish Notation)

For

BCA Part-II (Session 2018-21) Students

BY

ANANT KUMAR
MCA, M. Phil, M. Tech.
Faculty Member
Department of Computer Science
J. D. Women's College, Patna

Infix, Prefix and Postfix Expressions

Infix Expression	Prefix Expression	Postfix Expression
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +

A + B * C would be written as + A * B C in prefix. The multiplication operator comes immediately before the operands B and C, denoting that * has precedence over +. The addition operator then appears before the A and the result of the multiplication.

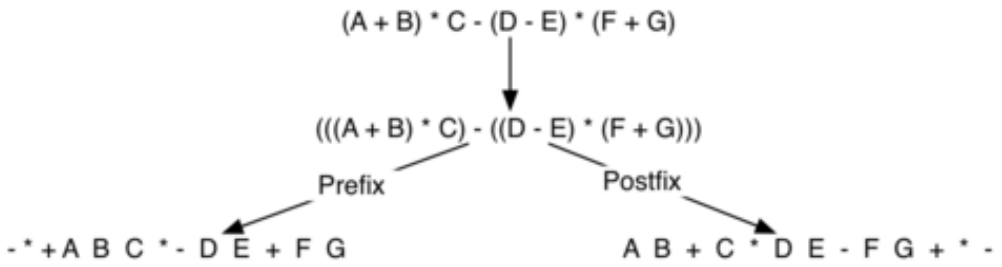
In postfix, the expression would be A B C * +. Again, the order of operations is preserved since the * appears immediately after the B and the C, denoting that * has precedence, with + coming after. Although the operators moved and now appear either before or after their respective operands, the order of the operands stayed exactly the same relative to one another.

(A + B) * C	* + A B C	A B + C *

The infix expression (A + B) * C. Recall that in this case, infix requires the parentheses to force the performance of the addition before the multiplication. However, when A + B was written in prefix, the addition operator was simply moved before the operands, + A B. The result of this operation becomes the first operand for the multiplication. The multiplication operator is moved in front of the entire expression, giving us * + A B C. Likewise, in postfix A B + forces the addition to happen first. The multiplication can be done to that result and the remaining operand C. The proper postfix expression is then A B + C *.

A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +
A + B + C + D	+ + + A B C D	A B + C + D +

Expression: (A + B) * C - (D - E) * (F + G) convert it into postfix and prefix notations.



Infix to postfix conversion

Scan through an expression, getting one token at a time.

1. Fix a priority level for each operator. For example, from high to low:

3. - (unary negation)
2. * /
1. + - (subtraction)

Thus, high priority corresponds to high number in the table.

2. If the token is an operand, do not stack it. Pass it to the output.

3. If token is an operator or parenthesis, do the following:

- Pop the stack until you find a symbol of lower priority number than the current one. An incoming left parenthesis will be considered to have higher priority than any other symbol. A left parenthesis on the stack will not be removed unless an incoming right parenthesis is found. The popped stack elements will be written to output.
- Stack the current symbol.
- If a right parenthesis is the current symbol, pop the stack down to (and including) the first left parenthesis. Write all the symbols except the left parenthesis to the output (i.e. write the operators to the output).
- After the last token is read, pop the remainder of the stack and write any symbol (except left parenthesis) to output.

Example:

Convert A * (B + C) * D to postfix notation.

Move	Curren Ttoken	Stack	Output
1	A	empty	A
2	*	*	A
3	(	(*	A
4	B	(*	A B
5	+	+(*	A B
6	C	+(*	A B C
7	)	*	A B C +
8	*	*	A B C + *
9	D	*	A B C + * D
10		empty	

Evaluating Postfix Expressions

Once an expression has been converted to postfix notation it is evaluated using a stack to store the operands.

- Step through the expression from left to right, getting one token at a time.
- Whenever the token is an operand, stack the operand in the order encountered.
- When an operator is encountered:
- If the operator is binary, then pop the stack twice
- If the operator is unary (e.g. unary minus), pop once
- Perform the indicated operation on the operator(s)
- Push the result back on the stack.
- At the end of the expression, the top of the stack will have the correct value for the expression.

Example:

Evaluate the expression 2 3 4 + * 5 * which was created by the previous algorithm for infix to postfix.

Move	Current Token	Stack (grows toward left)
1	2	2
2	3	3 2
3	4	4 3 2
4	+	7 2
5	*	14
6	5	5 14
7	*	70