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

Polish Notation

Algebraic Expressions:

An algebraic expression is a legal combination of operators and operands. Operand is the quantity on which a mathematical operation is performed. Operand may be a variable like x, y, z or a constant like 5, 4, 6 etc. Operator is a symbol which signifies a mathematical or logical operation between the operands. Examples of familiar operators include +, -, *, /, ^etc.

An algebraic expression can be represented using three different notations. They are infix, postfix and pre fix notations:

Infix: It is the form of an arithmetic expression in which we fix (place) the arithmetic operator in between the two operands.

Example: (A + B) * (C - D)

Prefix: It is the form of an arithmetic notation in which we fix (place) the arithmetic operator before(pre) its two operands. The prefix notation is called as polish notation (due to the polish mathematician Jan Lukasiewicz in the year 1920).

Example: * + A B – C D

Postfix: It is the form of an arithmetic expression in which we fix (place) the arithmetic operator after (post) its two operands. The postfix notation is called as *suffix notation* and is also referred to *reverse polish notation*.

Example: A B + C D - *

The three important features of postfix expression are:

1. The operands maintain the same order as in the equivalent infix expression.
2. The parentheses are not needed to designate the expression un- ambiguously.
3. While evaluating the postfix expression the priority of the operators is no longer relevant.

We consider five binary operations: +, -, *, / and $ or ↑ or ^ (exponentiation). For these binary operations, the following in the order of precedence (highest to lowest):

OPERATOR	PRECEDENCE	VALUE
Exponentiation ($ or ↑ or ^)	Highest	3
*, /, %	Next highest	2
+, -	Lowest	1

Converting expressions using Stack:

Conversion from infix to postfix:

Procedure to convert from infix expression to postfix expression is as follows:

1. Scan the infix expression from left to right.

2. a) If the scanned symbol is left parenthesis, push it onto the stack.

   b) If the scanned symbol is an operand, then place directly in the postfix expression(output).

   c) If the symbol scanned is a right parenthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.

   d) If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in the postfix expression, if and only if the precedence of the operator which is on the top of the stack is greater than (*or greater than or equal*) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack.

Example 1:

Convert ((A – (B + C)) * D) ↑ (E + F) infix expression to postfix form:

SYMBOL	POSTFIX STRING	STACK	REMARKS

(		(	
(		((	
A	A	((	
-	A	((-	
(	A	((- (	
B	A B	((- (	
+	A B	((- (+	
C	A B C	((- (+	
)	A B C +	((-	
)	A B C + -	(	
*	A B C + -	(*	
D	A B C + - D	(*	
)	A B C + - D *		
↑	A B C + - D *	↑	
(	A B C + - D *	↑ (	
E	A B C + - D * E	↑ (	
+	A B C + - D * E	↑ (+	
F	A B C + - D * E F	↑ (+	
)	A B C + - D * E F +	↑	
End of string	A B C + - D * E F + ↑	The input is now empty. Pop the output symbols from the stack until it is empty.	

Example 2:

Convert a + b * c + (d * e + f) * g the infix expression into postfix form.

SYMBOL	POSTFIX STRING	STACK	REMARKS
a	a		
+	a	+	
b	a b	+	
*	a b	+ *	
c	a b c	+ *	
+	a b c * +	+	
(	a b c * +	+ (	

d	a b c * + d	+ (	
*	a b c * + d	+ (*	
e	a b c * + d e	+ (*	
+	a b c * + d e *	+ (+	
f	a b c * + d e * f	+ (+	
)	a b c * + d e * f +	+	
*	a b c * + d e * f +	+ *	
g	a b c * + d e * f + g	+ *	
End of string	a b c * + d e * f + g * +	The input is now empty. Pop the output symbols from the stack until it is empty.	

Example 3:

Convert the following infix expression A + B * C – D / E * H into its equivalent postfix expression.

SYMBOL	POSTFIX STRING	STACK	REMARKS
A	A		
+	A	+	
B	A B	+	
*	A B	+ *	
C	A B C	+ *	
-	A B C * +	-	
D	A B C * + D	-	
/	A B C * + D	- /	
E	A B C * + D E	- /	
*	A B C * + D E /	- *	
H	A B C * + D E / H	- *	
End of string	A B C * + D E / H * -	The input is now empty. Pop the output symbols from the stack until it is empty.	

Example 4:

Convert the following infix expression A + (B * C – (D / E ↑ F) * G) * H into its equivalent postfix expression.

SYMBOL	POSTFIX STRING	STACK	REMARKS
A	A		
+	A	+	
(	A	+ (	
B	A B	+ (	
*	A B	+ (*	
C	A B C	+ (*	
-	A B C *	+ (-	
(	A B C *	+ (- (	
D	A B C * D	+ (- (	
/	A B C * D	+ (- (/	
E	A B C * D E	+ (- (/	
↑	A B C * D E	+ (- (/ ↑	
F	A B C * D E F	+ (- (/ ↑	
)	A B C * D E F □/	+ (-	
*	A B C * D E F □/	+ (- *	
G	A B C * D E F □/ G	+ (- *	
)	A B C * D E F □/ G * -	+	
*	A B C * D E F □/ G * -	+ *	
H	A B C * D E F □/ G * - H	+ *	
End of string	A B C * D E F □/ G * - H * +	The input is now empty. Pop the output symbols from the stack until it isempty.	