To devise an algorithm to perform this conversion of infix to postfix expression a stack data structure is created and two precedence functions are defined:

P_TOS: Precedence value of the symbol of the top of the stack

P_INP: Precedence value of the symbol in the input infix expression

The given input expression which consists of operators and operands are scanned from left to right. Symbols in the input expression are pushed and popped from the stack. For each symbol in the input expression the precedence values are assigned as per the below table:

Symbols	Input precedence function P_INP	Stack precedence function - P_TOS
+ , -	1	2
* , /	3	4
$ or ^	6	5
operands	7	8
(	9	0
)	0	-
#	-	-1

If an operator is left associative, the input precedence is less than the stack precedence and if an operator is right associative, the input precedence is higher than the stack precedence. Symbol $ or ^ which corresponds to exponentiation and is right associative. Hence, the input precedence value of $ or ^ is higher than that of the stack precedence.

The initial configuration of the stack, input and output strings to convert an infix expression to postfix expression is:

Stack	Input	Output
#	Infix expression (A+(B-C)*D)	-

Final configuration after the conversion of infix to postfix expression:

Stack	Input	Output
#	-	Postfix expression A B C – D * +

In the final configuration, the stack contains the symbol '#' and input is empty (which means that the entire input string is processed).

Note that before and after the conversion, top of the stack contains '#'.

3

‹ Conversion of an Infix to Postfix Expression Demo (/?q=MULNCourseBook/listDetailedCBLearningContents/178099/cidb/full/view/153700/431513/431502/39729)

Data Structures and Algorithms › (/?q=MULNCourseBook/listDetailedCBLearningContents/178099/cidb/full/view/153785/431527/431502/39729)