CSE 330 Algorithm "Infix to Postfix Conversion"
A Stack ADT Application

KV, Winter 2020

This algorithm reads an arithmetic expression in "infix notation" from left to right; while doing so, it writes the expression out in "postfix notation".

Infix: operand – *operato*r – operand (e.g., 5 + 3)
Postfix: operand – operand – *operator* (e.g., 5 3 +)

Infix expressions may contain parenthesis ('(' and ')'); these may be used to override the precedence order between two operators, or they may be used for readability. Postfix expressions do not ever have parenthesis. On order to facilitate and expedite the algorithm implementation, assume that the input expressions are always given so that each operand and operator, including '(' and ')', are surrounded by white space. Example:

(5+3)*7 will be input as: '(5 + 3) * 7 =' (the '=' symbol is to mark the end of the input).

The infix-to-postfix conversion algorithm:
The user's input is processed one operator or operand at a time, as the input is read from beginning to end. Let X be the next operator or operand, including parenthesis. Let STACK be the stack data structure that is central to the algorithm.

Start loop …

1. Read next X from the input. If X is "=" exit loop, go to 6.
2. If X is an operand, write it out as is. Repeat loop.
3. If X is equal to "(", push X onto STACK. Repeat loop .
4. If X is an operator do the following:
    a. If STACK is empty
        i. Push the operator into STACK; Repeat loop.
    b. Else
        i. Keep popping the next operator from the stack and write it out. Do so until one of three conditions holds:
            1. The top of STACK is "("
            2. The top of STACK is an operator of lower precedence than X
            3. The STACK is empty.
        ii. Push new operator X onto STACK. Repeat loop.
5. If X is equal to ")", keep popping operators from STACK and write them out until top of STACK is "(". Pop STACK on more time to remove "(". Repeat loop.
6. (The input is fully read.) Pop and write out the top of STACK until STACK is empty.

At the end of step 6., the infix has been completely read and its equivalent in postfix notation has been printed out.

A running example:

Infix: A – (B + C * D) / E =

Reading X	STACK	Postfix Output	Alg Step
A		A	1. + 2.
-	-		4.a.i.
(	-(	A	3.
B	-(	A B	2.
+	-(+	A B	4.b.i.+ii.
C	-(+	A B C	2.
*	-(+*	A B C	4.b.i.+ii.
D	-(+*	A B C D	2.
)	-(+	A B C D *	5.
	-(	A B C D * +	5. cont
	-	A B C D * +	5. cont
/	-/	A B C D * +	4.b.i.+ii.
E	-/	A B C D * + E	2.
=	-	A B C D * + E /	6.
		A B C D * + E / -	6.

Thus: infix A – (B + C * D) / E =

Printed out as postfix A B C D * + E / -