CSc 227 — Program Design and Development
(McCann)

Infix → Postfix Conversion Algorithms

1. **Manual Algorithm:**

   (a) Fully parenthesize the the infix expression (one set of parentheses per operator)
   (b) Replace the right parentheses with their corresponding operators
   (c) Remove the left parentheses

   Example: `A / (B + C) - D`

   ```
   (a)  ( ( A / ( B + C ) ) - D )
   (b)  ( ( A   ( B   C + /   D -
   (c)      A     B   C + /   D -
   ```

   The infix expression `A / (B + C) - D` is the same as the postfix expression `A B C + / D -`

2. **Stack-based Pseudocode Algorithm:**

```
while there are more symbols to be read
    read the next symbol
    case:
        operand   -->  output it.
          '('     -->  push it on the stack.
          ')'     -->  pop operators from the stack to the output
                       until a '(' is popped; do not output either
                       of the parentheses.
        operator  -->  pop higher- or equal-precedence operators
                       from the stack to the output; stop before
                       popping a lower-precedence operator or
                       a '('.  Push the operator on the stack.
    end case
end while
pop the remaining operators from the stack to the output
```

Example: `A / (B + C) - D`

Input Symbol	Stack Content	Output
A	*nil*	A
/	/	A
(	/(	A
B	/(	A B
+	/(+	A B
C	/(+	A B C
)	/	A B C +
-	-	A B C + /
D	-	A B C + / D
<*eof*>	*nil*	A B C + / D -

The infix expression `A / (B + C) - D` is the same as the postfix expression `A B C + / D -`