# EECS 2500 – Fall 2015

## Programming Project #1: Due Monday, 10/12/2015 @ 23:59:59

For this assignment, you will implement a program to convert infix notation to postfix notation, and then evaluate the result.

The following algorithm can be used to convert infix to postfix:

```
convertExpression
while (there are more tokens in the infix expression)
      get the next token
      if the token is an operator
            call processOperator(token)
      else if the token is a left parenthesis
            push token onto stack
      else if the token is a right parenthesis
            call processRightParen()
      else  //token is an operand
            append token to postfix expression

//now infix expression is empty
pop whatever is left on the stack and append to the postfix expression


processRightParen()
get the top token on the stack
while (top token is not a left parenthesis)
      append top token to the postfix expression
      pop the stack
      get the next top token on the stack
pop once more to get rid of the left parenthesis


processOperator(String token)
get the top token on the stack
while (the stack is not empty AND
       the top token is not a left parenthesis AND
        the precedence of the token <= precedence of the top token)
           pop the stack
           append the top token to the postfix expression
           get the next top token on the stack
push the token onto the stack
```

Your program should prompt the user for an infix notation expression, print the postfix representation of that expression, and then evaluate the postfix version (the postfix expression evaluator can be found in Case Study 3.8 from the text (pp. 214 – 226). You may use the postfix evaluator code from the book – just make sure you understand how it works, and don't just copy it blindly. YOU WILL HAVE TO ADD TO THIS CODE; IT DOES NOT SUPPORT ALL OF THE OPERATORS YOU NEED, SO IT'S ONLY A STARTING POINT (but it's a *good* head start)

Your program should support the following operations (note that some are unary; some are binary) on integer values: +, -, ×, /, ^, Q, C, <, >, and %, where:

+, -, ×, / are conventional addition, subtraction, multiplication, and division
^ is exponentiation (3 ^ 4 is $3^4 = 81$)
Q is square root (return $\lfloor\sqrt{x}\rfloor$, so Q(11) returns 3, Q(8) returns 2, Q(9) returns 3)
C is the cube root (again, discard fractions) – recall that the cube root of x is $10^{(\log x / 3)}$ or $2^{(\lg x / 3)}$
< is shift left (3 < 2) shifts the bits of "3" (0011) to the left two places, resulting in 1100 – 12
> is shift right (15 > 1) shifts the bits of "15" (1111) to the right one place, resulting in 0111 – 7
% is the modulus operator (7 % 2 is 1; 15 % 5 is 0; 17 % 5 is 2)

For this project, you may ignore operator precedence, and assume that all operators have the same level of precedence. Your code should be able to handle (2 + 3 + 4) or (2 * 3 * 4). If you are given (3 + 4 * 5), just process them left-to-right (add 3 and 4, and then multiply the result by 5).

You should test your program on a variety of expressions, and make absolutely sure that it handles them all. Your input expressions may or may not have spaces delimiting the parentheses, operators, and/or operands. Your program should support all of the following:

```
( 2 + 3 ) * ( 4 % 2 )
(2+3)*(4%2)
Q(2+(4*2))
C(27)
```

The individual operands will all be positive integers, but may be anywhere from 1 through 2 billion. Your expression evaluator should work with a long integer, and indicate overflow if the result will be greater than 2 billion (2,147,483,647, to be exact). Note, however, that you may still wind up with a negative result:

(0 – 1) * (2 < 10)

Would produce output of -1024

Implement your stack(s) with a linked list(s) of nodes. Think about what kind of information each node needs to contain.

Submit a ZIP/7-zip file of your ENTIRE Java Workspace directory (and its subfolders) to Blackboard. I should be able to unzip your workspace, point Eclipse to it, and run your code without modification. You should be using the 32-bit versions of Eclipse Standard (Mars), with the (also 32-bit) JDK 8u60.

If you need them, they are available at:

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/mars/R/eclipse-java-mars-R-win32.zip
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

MAKE SURE YOU GET THE 32-BIT VERSIONS, EVEN IF YOU'RE RUNNING A 64-BIT O/S!!!!!

It should go without saying, but you are not allowed to use any Java built-in collection classes like stacks! Implement your own!!

If you have questions, please let me know ASAP.