COSC 2336
Lab Assignment I
DUE DATE: Thursday, 19 February 2015

Write a complete Java program which will input from a file a series of infix expression strings involving addition and multiplication on the set of single-digit integer operands and then evaluate and output the modulo 10 result of each expression. An example of the output appears below:

THE MODULO 10 VALUE OF 6+9*(5*(3+4)) IS 1

Assume that each input expression is a valid expression. The following expression strings should be used as test data input for your program (create additional expression strings if you desire to further exercise the algorithm):

```
5
(5)
3+4
3*5
5*(3+4)
3+4*5+6
2*(((4+2)))
6+9*(5*(3+4))
7*3+5*6
1*2*3*4*5*6*7
(((((5)))))
1+2+3+4+5+6
(3*6+4)*(4+5*7)
```

The following BNF notation describes the recursive structure of expressions involving addition and multiplication on the set of integers modulo 10:

FILE ::= { LINE } <eof>
LINE ::= EXPRESSION <eoln>
EXPRESSION ::= TERM { '**+**' TERM }
TERM ::= FACTOR { '*****' FACTOR }
FACTOR ::= **digit** | '**(**' EXPRESSION '**)**'

Your program will employ several Java methods. The basic logical design for each method is given on the following page.

Be sure to follow the techniques of good programming style and use extensive comments to provide for internal documentation of your source program. You will be required to submit *listings* of your source program file, your input data file, and your output file (or screenshot of the output). These listings should be individually stapled and with all paper-clipped together. Please submit these deliverables on or before the assignment due date.

```
declarations for static class variables:
      token (a character), expr (a String), k = 0 (an integer)
method main
{ local variable: exprValue, an integer. }
      Open input and output files
      while (not end-of-file)
         Output “THE MODULO 10 VALUE OF ”
         Input next line into expr
         Assign to token the kth character of expr
         Put token to output file
         Assign value of expression() to exprValue
         Output to file “ IS ” and exprValue followed by two newlines
         Initialize k back to zero
      end while
      Close input and output files
end method main

method getToken()
      Increment k by 1
      if ( k IS LESS THAN the length of expr )
            Assign to token the kth character of expr
            Put token to output file
      end if
end method getToken

method expression()
{ local variables:  termValue, an integer; exprValue, an integer. }
      Assign value of term() to exprValue
      while ( token IS EQUAL TO ‘+’ )
            getToken()
            Assign value of term() to termValue
            Assign MOD 10 sum of exprValue and termValue to exprValue
      end while
      return exprValue
end method expression

method factor()
{ local variable:  factorValue, an integer. }
      if ( token is a digit )
            Assign the value of the digit to factorValue
            getToken()
      else if ( token IS EQUAL TO ‘(’ )
                  getToken()
                  Assign value of expression() to factorValue
                  if ( token IS EQUAL TO ‘)’ )
                        getToken()
                  end if
            end if
      end if
      return factorValue
end method factor

method term()
{ local variables:  factorValue, an integer; termValue, an integer. }
      Assign value of factor() to termValue
      while ( token IS EQUAL TO ‘*’ )
            getToken()
            Assign value of factor() to factorValue
            Assign MOD 10 product of termValue and factorValue to termValue
      end while
      return termValue
end method term
```