COSC 5390
Lab Assignment III
DUE DATE: Thursday, 12 November 2015

Write a program to implement a *recursive-descent* infix-to-postfix translator to accommodate statements with the standard arithmetic operations along with the assignment operator (**=**) and operations for **mod**, **div**, exponentiation (**^**), prefix unary plus (**#**), and prefix unary minus (**~**). You may assume the following regarding operator precedence from highest to lowest:

(1) Unary operators **#** and **~**
(2) Exponentiation operator (w/right associativity)
(3) Multiplicative operators ***, /, div, mod**
(4) Additive operators **+** and **-**

Your program should function as an infix-to-postfix translator for a language consisting of sequences of statements terminated by semicolons. The expressions of statements consist of numbers, identifiers, and the operators **#**, **~**, **^**, **+**, **-**, *****, **/**, **div**, and **mod**, as well as allow for parenthesized expressions. The output of the translator is a postfix representation for each statement. From the following context-free grammar describing statements written in this language (with left recursion removed), you will develop a syntax-directed translation scheme (note the output actions embedded in production rules) which will serve as a model for your translator:

```
Statement → id = {print(id.lexeme)} Expression {print('=')} ;
Expression → Term Moreterms
Moreterms → + Term {print('+')} Moreterms
          | - Term {print('-')} Moreterms
          | ∈
Term → Factor Morefactors
Morefactors → * Factor {print('*')} Morefactors
            | / Factor {print('/')} Morefactors
            | div Factor {print('div')} Morefactors
            | mod Factor {print('mod')} Morefactors
            | ∈
Factor → Base ^ Factor {print('^')}
       | Base
Base → # Value {print('#')}
     | ~ Value {print('~')}
     | Value
Value → ( Expression )
      | id {print(id.lexeme)}
      | num {print(num.value)}
```

Obviously, your program should incorporate some rudimentary lexical analyzer (perhaps some modified version of your second lab assignment) which can identify and return "the next input symbol" (token). Input to your program should consist of a data file containing the statements found in the file "DATA FOR LAB ASSIGNMENT III" (feel free to include additional statements at the end of this file for further exercising of your translator).

Be sure to follow the techniques of good programming style and use extensive comments to provide for internal documentation of your source program(s). The output of your program should be a file of the postfix equivalents of each program statement. You will be required to submit *listings* of your source program file(s), your input data file, and your output file (listings should be individually stapled and clipped together). Graduate students are required to include additional statements at the end of this file for further exercising the translator.