## CS 4850 Fall 2015 Flex Bison Post Fix Assignment

Write a program using flex/lex and bison/yacc to parse infix expressions and output them as post fix expressions.

Write a C function that evaluates the post fix expression and prints the result to stdout.

You will submit at least 4 files.

1 lex input a grammar , not .c

2 bison input a grammar; not .c

3 with main and the post fix evaluation function(s) .c

Also a 4 makefile

Write a lex grammar that recognizes whole numbers, all the binary operators and parenthesis and end of line

+, -, /, \*, % , (, ), ‘\n’

Write a bison grammar that parses a line with an infix expression and outputs a string with it converted to a post fix expression.

Write a function that receives a string with a post fix expression, evaluates it and outputs the result or an error. Must follow precedence rules and use ( ) correctly.

Your program should do one line at a time until end of file.

Each line is an expression which converted and evaluated.

infix examples:

10 \* (2 + 3) + 4 \* 5

post fix:

10 2 3 + \* 4 5 \* +

Read a line one token at a time, convert to post fix, send string to evaluator and print the result.

Get next line.

If you use information from any other web site be sure to put it in your comments at the top of your programs along with your name. You must write your own code not copy any code from someone else.

If you copy stack code or a stack design from somewhere you must give the exact url or reference for it. This should be easy to write all by yourself. Simple array with a variable to keep track of the “top”.

The algorithm for evaluating any postfix expression is fairly straightforward:

* While there are input tokens left
  + Read the next token from input.
  + If the token is a value
    - Push it onto the stack.
  + Otherwise, the token is an operator.
  + It is known a priori that the operator takes n arguments.
  + If there are fewer than n values on the stack
    - (Error) The user has not input sufficient values in the expression.
  + Else, Pop the top n values from the stack.
  + Evaluate the operator, with the values as arguments.
  + Push the returned results, if any, back onto the stack.
* If there is only one value in the stack
  + That value is the result of the calculation.
* Otherwise, there are more values in the stack
  + (Error) The user input has too many values.