CSCE 434 Programming Assignment #1
Due Date: 11:59 p.m., 7 February, 2021 on Gradescope

Problem Description:

Write a recursive descent compiler for the following grammar:

<program> → **begin** <stmt_list> **end**
<stmt> → <id> **:=** <expr> | ε
<stmt_list> → <stmt_list> **;** <stmt> | <stmt>
<expr> → <expr> **+** <term> | <expr> **-** <term> | <term>
<term> → <term> ***** <factor> | <term> **div** <factor> | <term> **mod** <factor> | <factor>
<factor> → <primary> **^** <factor> | <primary>
<primary> → <id> | <num> | (<expr>)

Here, <id> represents any valid sequence of characters and digits starting with a character; and <num> represents any valid integer literal. "^" is the exponentiation operator.

Since <program> is the start symbol, you may assume that a valid program consists of the keyword **begin** at the beginning, the keyword **end** at the end, and zero or more assignment statements separated by semicolons.

Your lexical analyzer should be hand-coded for this assignment; that is, you will not use a lexical analyzer generator.

The statements will be encoded for a hypothetical stack machine with the following instructions:

PUSH	*v*	-- push *v* (an integer constant) on the stack
RVALUE	*l*	-- push the contents of variable *l*
LVALUE	*l*	-- push the address of the variable *l*
POP		-- throw away the top value on the stack
STO		-- the rvalue on top of the stack is place in the lvalue below it and both are popped
COPY		-- push a copy of the top value on the stack
ADD		-- pop the top two values off the stack, add them, and push the result
SUB		-- pop the top two values off the stack, subtract them, and push the result
MPY		-- pop the top two values off the stack, multiply them, and push the result
DIV		-- pop the top two values off the stack, divide them, and push the result
MOD		-- pop the top two values off the stack, compute the modulus, and push the result
POW		-- pop the top two values off the stack, compute the exponentiation operation, and push the result
HALT		-- stop execution

You may use C, Java, or Python to write your program. You will, of course, find it necessary to remove left recursion in certain productions. Your compiler will prompt for the name of the input program file, and send the output to standard output.

No symbol table is necessary for this exercise. If you construct one, management would be suitably impressed.

On errors, a "meaningful" error message should be printed; i.e., "DO expected in line xx", "expression expected", *etc*. Management would be impressed if you attempted to recover from an error rather than simply halting compilation.