Boss Attack 03

A. Matrix Calculator (Problem ID : 107BA3_A)

Problem Description

Be similar to a simple calculator. In the problem, you're asked to write a syntax and semantic parser for a matrix expression. The objective of this problem contents two parts. The first part is to build a AST for the input expression. The second part is to check if the dimension on both sides of each operator is valid. For example, of multiplication: $A_{1*2}B_{2*1}$ is valid which will generate a matrix with 1 x 1 dimension. But $A_{2*3}B_{2*4}$ is invalid. But if we take transpose of A in expression: $A^T_{2*3}B_{2*4}$ which equals to $A_{3*2}B_{2*4}$ the whole expression can be valid.

The following are supported operators in this compiler:

Operator	Symbol
addition	+
subtraction	-
multiplication	*
transpose	^T
parenthesis	()

All matrix are 2-dimensional matrices which are represented as [row number, column number] e.g. [2, 3] is a 2x3 matrix and [5,1] is a 5x1 matrix. Your output should show "Syntax Error" if the input expression does not follow the grammar. If it follows the grammar, then apply sematic check to see if the dimension on both side of each operator is correct.

Print "Semantic error on col" followed by the location of the first operator which makes the sematic error occur in post order of AST. If there is not any syntax errors or semantic errors, print the dimension of the matrix.

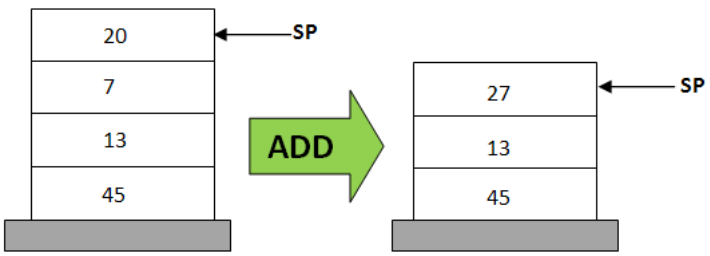
Sample Input and Output

Sample Input	Sample Output
[2,1]^T*[2,1]	[1*1]
([2,3]*[2, 3]^T)^T+[4,1]	Semantic error on col 19
([1,2]+[2,1]^T)*[1,3]*[1,3]^T*[3,3]	Semantic error on col 16

B. Stack-based machine (Problem ID : 107BA3_B)

Problem Description

Stack-based machine is a kind of CPU design architecture which sometimes is compared with register-based machine. Unlike register-based machine which used 3 address code taking 1 opcode and 2 corresponding operands e.g. add %eax, 3, Stack-based machine was designed with opcode with one or zero operand and one long stack which is served as registers to take temporary value in calculation.
It works as follow:



Like the figure above, we may first execute 4 instructions: load 45, load 13, load 7, load 20, then the stack will layout like the stack on left side (note that the "load" instruction is the only operator that takes 1 operand). At the moment, if we continue to execute instruction 'add'

which have 0 operand, it will pop two numbers (20, 7) from the top of stack, then execute 'add' with those numbers. Finally push the result (27) back to the stack which will have final layout like the stack on the right side.

In this problem, you will be given a program for stack-based machine. You need to write a yacc program to check if the format of the program and get its result. The instruction set of our machine are: `add`, `sub`, `mul`, `mod` which will pop 2 operands from stack, perform addition, subtraction, multiplication and modulation and push back the result, `inc`, `dec` which will pop 1 operand from stack increase/decrease operand by 1, then push back the result, "`load <number>`" which will push a constant `<number>` on the top of the stack, `copy` which will copy a constant `<number>` from stack top, then push it back, `delete` which will delete 1 operand from stack, `switch` which will pop 2 operands from stack, switch the order of them, and then push them back to the stack.

Sample Input and Output

Sample Input	Sample Output	Explanation
load 1 load 2 sub load 5 mod	0	5 % (2 - 1) = 0
load 3 load 4 inc inc	Invalid format	If after finish the program, there are more than 1 numbers in the stack it will be consider invalid.
load 1 load 1 add sub	Invalid format	No enough operands in stack to perform sub.
load 1 copy sub	0	1 - 1 = 0
load 1 delete load 4 inc	5	4 + 1 = 5, delete number 1
load 1 load 2 switch sub	-1	switch number 1 and 2, 1 - 2 = -1