Stack Machine

Write a program that translates algebraic expressions into code for a stack machine.
An **expression** consists of integer constants, variable names, binary operators, and parentheses.

- A **constant** is a string of digits representing a positive integer.
- A **variable** is a string of lower case letters.
- The **operator symbols** are +, -, *, and /. There is no unary minus.
- Parentheses, (and), are used for grouping.
- There are no blanks in an expression.
- The expression is terminated by a dollar character, $. Following the usual algebraic convention, * and / have higher precedence than + and -.

In the output code, operands come before the corresponding operators.
Each operand is pushed onto the stack. An operator removes two operands from the stack, applies the operation, and pushes the result onto the stack.
The following table shows the code for each kind of expression:

Expression	Value	Output Code
constant	n	ldi n;
variable	s	ldv s;
operator	E1 + E2	E2 code for E1; code for E2; add;
operator	E1 – E2	code for E1; code for E2; sub;
operator	E1 * E2	code for E1; code for E2; mul;
operator	E1 / E2	code for E1; code for E2; div;

Input Format

The input file will contain a series of strings (each terminated with a $ character)
Each string containing digits, letters, '+', '-', '*', '/', '(', ')', and '$'.
If the input has incorrect syntax, the program should just write "Fail" and stop.

Output Format

The generated code, with each instruction followed by a semicolon.

Test Data

The following table gives examples of input strings and the expected output.
Note the effect of the parentheses in the third example.

Input	Output
ab/cd$	ldv "ab"; ldv "cd"; div;
3*x-y$	ldi 3; ldv "x"; mul; ldv "y"; sub;
3*(x-y)$	ldi 3; ldv "x"; ldv "y"; sub; mul;
(a-b)/(a+b)$	ldv "a"; ldv "b"; sub; ldv "a"; ldv "b"; add; div;