## Introduction

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | problem **0** | | **Programming with a Lithp** | | y points | |  |
|  |  |

LISP is the second-oldest high-level programming language that is still in use (the oldest is Fortran). While not as popular as Java or C++, LISP holds a tenacious niche in such diverse applications as databases, graphics, and artificial intelligence. The language’s name is a contraction of LISt Processing, which reflects its syntax. In LISP both code and data are written as lists contained in parentheses. So by contrast, a function call to compute cosine in Java is written cos(x) but in LISP the same call is written (cos x). There are no comma delimiters for list elements, so a function call to subtract seven minus two might be written like this: (sub 7 2). List expressions may of course be embedded, so the following two expressions would be equivalent:

|  |  |
| --- | --- |
| LISP expression | algebraic expression |
| (sub n (mul 2 (cos (add (mul 3 x) 1)))) | *n – 2 cos( 3x + 1 )* |

As you can see, even a short program can contain a lot of nested parentheses. LISP programmers face a real challenge trying to keep track of parentheses, especially as they change nested code by adding or removing expressions. That’s where you come in.

Write a program that can read a LISP expression and determine if all the function calls have the correct number of arguments.

# Sample Input

The first input section lists the minimum and maximum number of arguments for each function. An asterisk indicates no maximum. The second section consists of a LISP expression. You may assume there are no string constants, that all numbers will be non-negative integers, that the input will have correct pairing of parentheses, and that all tokens (function names, numbers, and variables) will be separated by white space (one or more spaces or line feeds).

3 SET 2 2 ADD 2 \* MUL 2 \*  
( SET Y ( ADD ( MUL A X X ) ( MUL B X ) C ) )

# Sample Output

If the LISP expression is valid the program should print “VALID EXPRESSION”. If a function call has too many arguments the program should print “FUNCTION foo: TOO MANY ARGUMENTS”. If a function call does not have enough arguments it should print “FUNCTION foo: NOT ENOUGH ARGUMENTS”. There will be a maximum of one error in the input.

VALID EXPRESSION

# More sample input/output…

6 SET 2 2 SUB 2 2 MUL 2 \* COS 1 1 ADD 2 \* DIV 2 2   
( SET T0 ( SUB X1 ( MUL 2 ( COS ( ADD ( MUL 3 THETA ) 1 ) ) ) ( DIV X1 3 ) ) )  
  
FUNCTION SUB: TOO MANY ARGUMENTS