Problem L – Latam++

The world does not have enough programming languages yet. To help with that, the Internal Committee for the Perfection of C (ICPC) is planning to build a brand new programming language: `Latam++`.

In `Latam++`, a variable name consists exclusively of one or more lowercase letters of the English alphabet. A valid expression is a "well-formed" string, expressing how to combine variables by using the four arithmetic binary operators "`+`", "`-`", "`*`" and "`/`", possibly with parentheses.

Formally, valid expressions are exactly those strings that can be produced by the following rules.

- A variable name is a valid expression.
- Surrounding any valid expression in parentheses produces another valid expression.
- If A and B are valid expressions, then the concatenation AcB is a valid expression, where c is any of the four arithmetic binary operators "`+`", "`-`", "`*`" and "`/`".

Thus, the following are all valid expressions:

- `a+b`
- `a+b*(c+b)`
- `atoms+boots*(charly+bob)`
- `(((a)))*(bbasdsaqwe/a/a/a)`

On the contrary, the following are not valid expressions:

- `a+`
- `a+b(c+b)`
- `atoms+boots*((charly+bob)`
- `((()))*(bbasdsaqwe/a/a/a)`

The language is far from complete, and it will likely take ICPC decades of debates until the first version of `Latam++` is released. Meanwhile, we will focus only on a specific and very special feature of its compiler, called Automatic Valid Substring Expression Counting (AVSEC).

AVSEC is an extremely useful feature, where the compiler reports the total number of substrings of a given string that are valid expressions. Your task is to implement AVSEC.

For counting purposes, two substrings are considered different if they start or end at different indexes, even if the corresponding strings are identical (that is, they are the same sequence of characters).

Input

The input consists of a single line that contains a string S ($1 \leq |S| \leq 2 \times 10^5$), which is made up of lowercase letters, opening or closing parenthesis, and the four characters "`+`", "`-`", "`*`" and "`/`".

Output

Output a single line with an integer indicating the number of substrings of S that are valid expressions.

Sample input 1	**Sample output 1**
`a+b(c+b)`	`7`

Explanation of sample 1:
The seven substrings are "`a`", "`a+b`", "`b`" (third character), "`(c+b)`", "`c`", "`c+b`", and "`b`" (seventh character).

Sample input 2	**Sample output 2**
`aa`	`3`

Sample input 3	**Sample output 3**
`a-a`	`3`