Problem E. lispp3

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

small 11 points *2 minute timeout*	The contest is finished.
large 22 points *10 minute timeout*	The contest is finished.

Problem

Lisp+++

After a year of parenthetical bliss using Lisp++, Alyssa thought it was finally time to go beyond the parentheses. She decided to create an extension of the language that adds the + operator to simplify the semantics. She fittingly named the new language Lisp+++.

A valid Lisp+++ program consists of one of the following. (In this specification, *P* stands for some valid program -- not necessarily the same program each time.)

- `()` Literally, just an opening parenthesis and a closing parenthesis.
- *PP* Two programs (not necessarily the same), back to back.
- (*P*+*P*) A pair of parentheses enclosing two programs (not necessarily the same), separated by a + character.

Alyssa's new Lisp+++ requires a compiler that must be able to evaluate a string consisting of `(`, `)`, and/or + characters and determine whether it is a valid Lisp+++ program, and provide the user with some helpful information if it is not. If the program is valid, the compiler should print -1. Otherwise, it should print the length of the longest prefix that could be extended into a valid program by adding one or more additional characters to the end. If that prefix is the empty prefix, the compiler should print 0. In particular, if the input string is not a valid program, but can be extended to a valid program, the compiler should print the length of the input string.

For example:

- `(()+())()` is a valid program, so the compiler should print -1.
- `(()()+)` is not a valid program. The prefix `(()()+` is the longest prefix that is or could be extended into a valid program; in this case, one possible valid extension is `(()()+())`. So, the compiler should print 6. The only longer prefix is `(()()+)` (i.e., the entire string), but there is no way to add (any number of) characters to the end of that string to make it into a valid program.
- `)` is not a valid program. The prefix `)` cannot be extended into a valid program. The empty prefix is not a valid program, but it can easily be extended into one (by adding `()`, for example). So the compiler should print 0.

Given a string, what should Alyssa's Lisp+++ compiler print?

Input

The input library is called "lispp3"; see the sample inputs below for examples in your language. It defines two methods:

- **GetLength()**:
  - Takes no argument.
  - Returns a 32-bit integer: the length of the string.
  - Expect each call to take 0.08 microseconds.
- **GetCharacter(i)**:
  - Takes a 64-bit integer in the range $0 \le i <$ GetLength().
  - Returns a character `(`, `)`, or +: the character at position i, counting from left to right.
  - Expect each call to take 0.08 microseconds.

Output

Output a single line with a single integer: what the compiler should print, as described above.

Limits

Number of nodes: 100 **(for both the Small and Large datasets)**.
Time limit: 5 seconds.
Memory limit per node: 128 MB.

Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 128 KB. **(Notice this is less than usual.)**
$1 \le$ GetLength() $\le 10^9$. GetCharacter(i) is always (,), or +.

Small input

number of i such that GetCharacter(i) = + ≤ 100. (There are at most 100 +s in the input.)

Large input

No additional limits. (There may be more than 100 +s in the input.)

Sample

Input	Output
See input files below.	For sample input 1: -1 For sample input 2: 6 For sample input 3: 0

The sample input files are the same examples from the problem statement, in the same order.

Sample input libraries:
Sample input for test 1: lispp3.h [CPP] lispp3.java [Java]
Sample input for test 2: lispp3.h [CPP] lispp3.java [Java]
Sample input for test 3: lispp3.h [CPP] lispp3.java [Java]