I - Infix to Prefix

Jaap wrote a solution to a lab assignment. The task was quite simple: convert an arithmetic expression in infix notation to an expression in Polish (prefix) notation. In infix notation operators are written between the operands (e.g. 12 + 5) while prefix notation places operators to the left of their operands (e.g. + 12 5).

This is the syntax of the expression Jaap had to convert:

Expression ::= *Number*
| '(' *Expression Op Expression* ')'
| '(' '−' *Expression* ')'
Op ::= '+' | '−'
Number ::= *Digit* | *Number Digit*
Digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

If a *Number* has more than one digit, it will not start with a '0'.
A *Number* has no more than 9 digits.

At this point we have to admit that the assignment was not very well specified. More specific, the syntax of the resulting expression was not given. So Jaap had to make some decisions himself — and he made the wrong decisions.

This was his first mistake: he believed that in prefix notation spaces are superfluous. This is true in infix notation, as there will always be an operator between two numbers. In prefix notation, however, numbers must be separated from one another. Omitting spaces in prefix notation, as Jaap did, gives rise to expressions like `+1234`, which has three different interpretations. (Exercise: draw the three different syntax trees.)

This was Jaap's second mistake: he believed that in prefix notation parentheses are superfluous. Prefix notation without parentheses is unambiguous only if the arity of the operators is fixed. Ambiguity occurs, for example, in the presence of both a unary and a binary minus. The expression: `--34`, can be read as `(- (- 3 4))`, evaluating to 1, or as `(- (- 3) 4)`, evaluating to −7, and even as `(- (- 34))`, evaluating to 34.

We do not ask you to reconstruct Jaap's program. We ask you to find out how ambiguous his output is.

Input

Each test case consists of a single line with a nonempty string of length ≤ 1000. The string contains only the characters '+' and '−' and digits '0' through '9'. This string is the output of Jaap's program.

Output

For each test case, print one line containing two numbers: the smallest and largest value that can be obtained by different interpretations of the input expression.

Example

input	output

`--34`
`+1234`
`--09`
`+111111111111`
`--0071`
	`-7 34`
	`46 235`
	`-9 9`
	`222222 111111222`
	`-71 -71`