A Context Free Grammar (CFG) consists of the followings: a set of *nonterminal* symbols V; a set of *terminal* symbols T; a special *nonterminal* symbol called the *root* and a set of *production rules*. If all the production rules are either of the form A -¿ BC, or A -¿ a, where A, B, C is a member of set V and a is a member of set **T** then we say that the grammar is in Chomsky Normal Form (CNF).

If we repeatedly apply the production rules over the *root* symbol we will finally end up with a string of *terminals*. Alternatively, we can start with a string of *terminals* and reduce it using given production rules.

For example the string '`ab`' can be obtained by the first CFG presented in the sample input in the following way:

```
S -> AB
AB -> aB ; because A -> a
aB -> ab ; because B -> b
```

But, we cannot obtain '`a`' from S by applying the production rules. The set of strings of *terminals* derivable from the *root* symbol of a CFG is called the Language of the CFG. In this problem you are required to determine whether a given string of *terminals* is in the Language of a CFG or not.

Input

There will be several test cases in the input. Each test case describes a CFG in Chomsky Normal Form and will adhere to the following description. In the first line there will be the *root* symbol. It will always be an uppercase English letter. In line 2 the set V will be presented as a string of uppercase letters. Each character of the string will be identified as a member of V. The set T will be given as a string of printable characters (except '`#`' or any whitespace characters) in line 3. Each character of the string will be identified as a member of T. Then there will be several lines for each production rule. A production rule will be of the form '`A -> BC`' or of the form ' A -¿ a'. Here `A, B, C` are from set V and `a` is from set T. A production rule of the form '`# -> #`' indicates the end of production rules. After that there will be several lines each containing a candidate string of printable characters. This string will not contain any character from **V** and there will be no more than 50 characters in it. The list of candidate strings will be terminated with a line containing a '`#`' in the first column.

Output

For each candidate string α print 'α `is in L(G)`' if it can derived from the given grammar otherwise print '*alpha* `is not in L(G)`'. Output a blank line after each test case.

Sample Input

```
S
SABC
ab
S -> AB
S -> BC
A -> BA
A -> a
B -> CC
B -> b
C -> AB
C -> a
# -> #
baaba
ab
abaa
a
aaaaa
bbbbb
#
S
SAB
ab
S -> AB
A -> AA
A -> a
B -> b
# -> #
ab
aaab
aba
baaaaaaaa
abbbbbb
aaaaaba
baaaaaaaab
aaaa
a
ab
#
```

Sample Output

```
baaba is in L(G)
ab is in L(G)
abaa is in L(G)
a is not in L(G)
aaaaa is in L(G)
bbbbb is not in L(G)

ab is in L(G)
aaab is in L(G)
aba is not in L(G)
baaaaaaaa is not in L(G)
abbbbbb is not in L(G)
aaaaaba is not in L(G)
baaaaaaaab is not in L(G)
aaaa is not in L(G)
a is not in L(G)
ab is in L(G)
```