773 The JustaPox Language

A new programming language is being designed. One of its new features is related to the specification of arrays. The language deals only with arrays of integers, and there are two ways of specifying new arrays:

1. by explicit declaration of its size. For example:

   $$A \text{ is } [1]$$
   $$B \text{ is } [3]$$

   These sentences specify A and B as arrays of 1 and 3 elements, respectively. This implies that any sequence with the specified size is a possible value for the array.

2. by inclusion of previously defined arrays. For example:

   $$C \text{ is } A\ B$$
   $$D \text{ is } A\ A$$
   $$SEQ \text{ is } [1]\ A\ [4]\ C\ B\ A$$

   The size for arrays specified in this manner is determined by the size of its sub-arrays. Besides, if a specification includes the same array more than once, the subsequences in the array corresponding to the repeated parts have to hold the same value.

   For the aforementioned specification of A and B, we have C with size 4, D with size 2, and SEQ with size 14.

Assignments are expressed in the language by the following statement:

$$< array > \texttt{=} < sequence\ of\ integers >$$

For example,

$$B = 3\ 4\ 15$$

is a valid assignment for B. Notice that the assignment $B = 3\ 4$ would be inconsistent with B's specification (which stated that B would hold a sequence of integers with size 3). The assignment $D = 1\ 2$ would also be inconsistent, since D's specification implies that D is formed by the repetition of a size 1 sequence of integers. $D = 1\ 1$ is an example of a consistent assignment for D.

In other words, for the specification

$$S \text{ is } S_1\ S_2\ \ldots\ S_m,$$

the assignment

$$S = i_1\ i_2\ \ldots\ i_n$$

is said to be **consistent** if and only if S is a sequence

$$i_{11} i_{12} \ldots i_{1k_1} i_{21} \ldots i_{2k_2} \ldots i_{m1} i_{m2} \ldots i_{mk_m},$$

where

- $k_i = |S_i|$ $(1 \leq i \leq m)$

and

- for every previously defined subsequences S_a, S_b $(1 \leq a, b \leq m)$ such that $S_a = S_b$ we have $(i_{a1}, i_{a2}, \ldots, i_{ak_a}) = (i_{b1}, i_{b2}, \ldots, i_{bk_b})$.

The language designers want you to cooperate with the compiler construction effort: you have to write a program that receives as input a collection of array specifications and a sequence of assignments, and produces as output the list of inconsistent assignments.

Input

The input file is composed by a list of array specifications followed by a list of assignments.

An array specification has the following format:

$< id >$ is $d_1\ d_2\ \ldots d_m$.

$< id >$ is a sequence of characters representing the array name. You can assume that (1) the length of this sequence is between 1 and 32 and (2) the sequence follows the rules for identifier names in the programming language that you are using.

d_i is

either $[k]$ where k is a strictly positive integer (there may be spaces between k and its surrounding brackets)

or $< id >$, where $< id >$ is the name of a previously specified array.

The sequence of d_i's is terminated by a period.

An assignment sepcification has the following format:
$< id > = i_1\ i_2\ \ldots\ i_n$.

$< id >$ is the name of a previously specified array and i_j are positive integers. The list of i_j's is terminated by a period.

In both types of specifications, an arbritrary number of spaces or empty lines can appear between the tokens.

Output

The output file must contain the list of assignment specifications from the input file that are inconsistent. The output for each of the inconsistent assignments should be its specification as it appeared in the input file.

Sample Input

```
A is [1].
B is [3].
C is A B.
SEQ is [1 ] A [4] B A.
A = 10.
B = 1 2 3.
C = 1 1 2 3 4.
SEQ = 1 10 1 1 1 1 2 2 2 10.
SEQ = 1 10 1 1 1 1 1 2 3 9.
```

Sample Output

```
C = 1 1 2 3 4.
SEQ = 1 10 1 1 1 1 1 2 3 9.
```