10904 Structural Equivalence

In programming language design circles, there has been much debate about the merits of "structural equivalence" vs. "name equivalence" for type matching. Pascal purports to have "name equivalence", but it doesn't; C purports to have structural equivalence, but it doesn't. Algol 68, the *Latin* of programming languages, has pure structural equivalence.

A simplified syntax for an Algol 68 type definition is as follows:

type_def `->` `type` *T* `=` *type_expr*
type_expr `->` *T* `|` `int` `|` `real` `|`
`char` `|` `struct` `(` *field_defs* `)`
field_defs `->` `T` `|` *field_defs* `T`

In this syntax, *T* is a programmer-defined type name (in this problem, for simplicity, a single upper case letter). Plain text symbols appear literally in the input, and zero or more spaces may appear where there are spaces in the syntax.

Algol 68 type equivalence say that two types are equivalent if they are the same primitive type or they are both structures containing equivalent types in the same order.

Input

Input consists of several test cases. Each test case is a sequence of Algol 68 definitions, as described above, one per line. A line containing '`-`' separates test cases. A line containing '`--`' follows the last test case.

Output

The output for each case will consist of several lines; each line should contain a list of type names, all of which represent equivalent types. Each type name should appear on exactly one line of output, and the number of output lines should be minimized. The names in each list should be in alphabetical order; the lines of output should also be in alphabetical order. Output an empty line between test cases.

Sample Input

```
type A = int
type B = A
type C = int
type X = struct(A B)
type Y = struct(B A)
type Z = struct(A Z)
type S = struct(A S)
type W = struct(B R)
type R = struct(C W)
--
```

Sample Output

```
A B C
R S W Z
X Y
```