Overload

Computer Science Society
Programming Contest
Winter 2009

In a purely object-oriented programming language with single inheritance, all types are defined as object classes and appear as nodes in a class hierarchy—a tree whose root class is named `Object` and whose other classes have exactly one parent class from which they inherit stuff. A class is an *ancestor* of another class if is identical to the second class or is an ancestor of the parent of the second class. In this problem, we're given signatures of method declarations and invocations, and determine what method declarations (if any) match each method invocation.

Input Format

The input consists of three sections of nonempty lines, separated from each other by single empty lines. The first section describes the *class hierarchy*; each line contains the name of a child class and its parent class with the string `extends` in between, separated by one or more blanks. The second section describes the signatures of *method declarations*; each line contains a formal result type, a method name, and zero or more formal argument types, separated by one or more blanks, commas and/or parentheses. The third section describes the signatures of *method invocations*; each line contains an actual context type, a method name, and zero or more actual argument types, separated by one or more blanks, commas and/or parentheses.

Output Format

A method declaration *matches* a method invocation if the method names are identical, the number of formal and actual arguments is equal, the actual context type is an ancestor of the formal result type (or both are `void`), and each formal argument type is an ancestor of the corresponding actual argument type. For each method invocation signature, output the method invocation signature, followed by matching method declaration signatures, followed by an empty line. Output the method invocation and matching method declaration signatures as shown in the output sample.

Input Sample

```
Number extends Object
Byte extends Number
Integer extends Byte
Widget extends Object
Shape extends Object
Rectangle extends Shape
Square extends Rectangle

void mash(Number, Shape)
Number area(Shape)
Integer area(Shape)
void area(Square)
Byte area(Rectangle)
Number side(Square)
void doIt()

Integer doIt()
Number area(Rectangle)
void mash()
void mash(Rectangle, Square)
void mash(Byte, Square)
```

Output Sample

```
actual: Integer doIt()

actual: Number area(Rectangle)
formal: Number area(Shape)
formal: Integer area(Shape)
formal: Byte area(Rectangle)

actual: void mash()

actual: void mash(Rectangle, Square)

actual: void mash(Byte, Square)
formal: void mash(Number, Shape)
```