Given these function declarations, suppose the program contains the following variable declarations and function call:

```c
float a = 1.0, b = 2.0;
int c = 3;
float d = g(c, f(a, b));
```

The first two declarations of $g$ would not work here, but the third one does: $f(<\text{float}>, <\text{float}>)$ returns $<\text{float}>$, which reduces the function call to $g(<\text{int}>, <\text{float}>)$, and the second declaration of $g$ will return the $<\text{float}>$ that can be assigned to variable $d$. Since we used the 3rd version of $f$ and the 2nd version of $g$, we say that the given function call is resolved by $d = g(c, f(a, b))$.

Using the same declarations, the function call $e = g(a, f(a, c))$ cannot be resolved. As a final example, consider the function declarations:

```c
float f(float v)
int g(int v)
```

and the variable declaration and function call:

```c
float a = 1.0;
char c = y(x(a));
```

In this case, we see that the resolution of the given function call is ambiguous.

### Input

The input will consist of a list of function declarations (one per line). Each function declaration in the input will have the form:

```c
```

where `name` is the function name, `param[i]` is the data type of the i-th parameter, and `rettype` is the data type of the return value (this problem does not deal with ‘void’ functions). `num_params` is at least 1 and at most 9. Note that the parameters do not have names, it is only their data types that are specified. Function names are single lower case letters, while data types are single upper case letters.

The list of function declarations in the input will be concluded by a line containing only a pound sign. Thereafter will come a list of function calls (one per line).

The list of function declarations in the input will be concluded by a line containing only a pound sign. Thereafter will come a list of function calls (one per line).

### Output

For each function call in the input, there will be one line of output. If the function call can be resolved uniquely, the output will be the same as the input function call, but each function name will have a serial number appended to it, to indicate which version of the function was used there. Otherwise, the output will be either ‘impossible’ or ‘ambiguous’, as explained above. If it is ambiguous, also output the number of ways the function call can be resolved, or print ‘>1000’ if there are more than 1000 ways.