4941 Module Madness

While going through the Swamp County College archives some strange drawings were discovered. They appear to specify some sort of computing device and seem to be written in a predecessor of today's relay logic ladder diagrams which are used for Programmable Logic Controllers. Luckily it isn't necessary to understand relay logic to understand them since they can easily be translated to boolean equations. Your job is to read in the diagram of a module and a list of input states, simulate the module, and produce the output each time the input changes.

See Figure 1 for a simple diagram and the equivalent boolean equation.

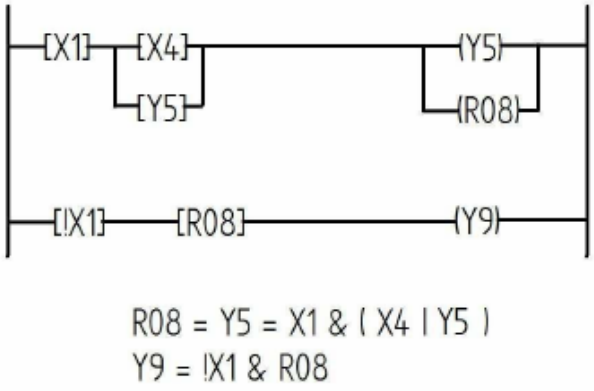

Figure 1. Sample ladder rungs and corresponding boolean equations.

The components of a diagram are gates (relay contacts) and gate controllers (relay coils). The diagrams are called ladder logic because you can picture the ladder rails as the power buses at each side of the diagram and the rungs as connected lines with one and only one connection to each bus on the left and right. The controllers can have no other components (gates or controllers) between them and the right hand bus, only wires. All gates are to the left of the controllers. Gates in series are an AND circuit and gates in parallel are an OR circuit. A controller is set to the results of the equations determined by the gates to its left. In turn, its associated gates are set by the state of the controller.

Input gates are set either by the outputs of another module or some input device. Output controllers can set the inputs of another module, control an output device, or control their associated gates in the diagram. Relay controllers may only control their associated gates in the diagram.

The rules for the character representation of the relay ladder logic diagrams follow:

- The left and right rails are not drawn. Wires are drawn with '`-`' characters. There will always be at least one '`-`' between components.
- Gates (which include input, relay, and output gates) are drawn enclosed in square brackets. There are up to ten possible input gates `X0`..`X9`, up to ten possible output gates `Y0`..`Y9`, and up to 100 possible relay gates `R00`..`R99`. Each of these may appear multiple times in the diagram and in any rung. They take either the boolean value of the associated input or controller, for example, `[R18]` or its logical negation `[!R18]`. The numbering is arbitrary — for example, a given diagram may only use `X2`, `X5`, and `X8` as inputs.
- Gates can be connected in series, parallel, or a combination.
- Controllers appear in the diagram enclosed in parentheses. There are up to ten possible output controllers `Y0`..`Y9` and up to 100 possible relay controllers `R00`..`R99`. A given controller may appear only once in a diagram. A controller will appear if its corresponding gate appears. However, a controller may appear without any corresponding gate. Controller numbering is arbitrary.
- Only controllers can connect to the right rail. Every rung has at least one controller connecting to the right rail. Controllers may not appear in series, only in parallel on a rung. If controllers

appear in parallel they each take the value determined by the boolean equation of all the gates to the left of their common junction on the rung.

- Connections can only be made to adjacent lines within a rung. A connection to a lower line is indicated by '`U`' and to an upper line by '`L`'. These indicators must line up properly on the two lines. There will be at least one '`-`' between a connector and a component.

The diagram is drawn so logic always flows to the right, assuming the left rail is the source, or up or down one line at one of the connector pairs. The text representation of the diagram in Figure 1 is:

```
----[X1]---U----[X4]----U-------U----(Y5)----U------>
           L----[Y5]----L L---(R08)----L
----[!X1]-----------------[R08]------(Y9)----------->
```

These diagrams do not have any race or other anomalous conditions, such as "`---[!R0]---(R0)---`", so the controllers are guaranteed to settle into a stable state in a short time but it is very likely several iterations will be needed before they do so.

On initial power up, all inputs and controllers are off, which is logical 0.

Input

The input diagram is a series of lines of at most 72 characters, terminated by an empty line. To save typing, a line ending in '`->`' is considered to extend to the right rail. There are at most 100 lines in the diagram.

The test inputs follow, one case per line, terminated by end of file. Each test case consists of exactly ten binary digits ('`0`' for off and '`1`' for on) with the left digit being `X0` and the rightmost digit being `X9`. Any input not appearing in the diagram is ignored. There are at most 100 test inputs.

Output

Your program is to first print the outputs after the power on transients have settled, then apply each input in turn and print the outputs when they have settled. Note that this means applying each input to the state resulting from the previous test case, not power cycling the module between cases. Also note that the execution order is not specified, since relays run on their own time.

Print the outputs for each test case as exactly ten characters on one line with the left character being '`Y0`' and the rightmost being '`Y9`'. Print '`0`' for off, '`1`' for on, and '`X`' if the output does not appear in the diagram.

Sample Input

```
---U---[!X3]---[X1]--U-----(Y2)-->
   L---[X3]----[!X1]-L
-----[X3]---[X1]-------U--(R00)-U-->
                         L--(Y9)--L
          U---[!X0]--[!X2]---[R00]--U
----U-----L----[X0]---[X2]---[R00]--L------U----------(Y1)-->
    L-U--------[X0]--[!X2]--[!R00]-----U---L
      L-------[!X0]---[X2]--[!R00]-----L
--U----[X0]----[X2]------------------U-------------(Y0)-->
  L-----[R00]-------U-----[X0]----U--L
                      L-----[X2]----L
---[X9]------U----[Y0]----U---(Y8)--->
```

```
             L----[Y8]----L

0000000000
0001000000
0010000000
0011000000
0100000000
0101000000
0110000000
0111000000
1000000000
1001000001
1010000001
1011000001
1100000001
1101000001
1110000001
1111000001
0000000001
0000000000
```

Sample Output

```
000XXXXX00
000XXXXX00
001XXXXX00
010XXXXX00
011XXXXX00
001XXXXX00
010XXXXX01
011XXXXX00
100XXXXX01
010XXXXX00
011XXXXX00
100XXXXX10
101XXXXX10
011XXXXX10
100XXXXX11
101XXXXX10
110XXXXX11
000XXXXX10
000XXXXX00
```