Funky Solver Modules

This is an exercise with funky solver modules (more commonly known as finite state machines - fsm). You are given descriptions of fsm's along with some input strings to feed into them. You then trace the route followed through the fsm for each string. Submit your solution under the file name *FunkySolverModule* with the appropriate language extension.

Input

There may be multiple fsm's to construct and test. Each case consists of the description of an fsm followed by input strings for testing. The last case is followed by a line containing 0 0 (two zeroes).

An fsm is described by a set of states and the transitions between the states. On the first line are two positive integers indicating the number of transitions and the number of test strings respectively. The transitions are described on the following lines, one per line, as the name of the first state, an input character, an output string, and the name of the next state. These fields are delimited only by commas (any spaces are part of the field).

State names are strings of up to 10 alphabetic characters and (embedded) spaces. Input characters can be any printable ASCII character, including the space and comma. The starting state is named *Start*. An accepting final state, if one exists, is named *Accept* which can be appended with an optional digit to disambiguate multiple accepting states. Whether or not explicitely included, a *Reject* state is assumed to exist and is the destination of any unspecified transition. An output string may be from 0 to 20 printable characters, including the newline represented as *n*. If the string contains a comma, the entire string should be quoted with double quotes. Any double quotes or literal back slashes in the output string should be doubled (with the first one serving as an escape to disambiguate the string).

Each of the test strings is then presented, one per line. Trailing blanks are possible if blanks are allowed as input characters of the fsm. A test string will not be longer than 100 characters. Any invalid characters should be ignored.

Output

For each new fsm, print its number formatted as shown in the example. For each input string, print its number followed by the string, formatted as in the example. (Input string numbering restarts at 1 with a new fsm.) On the next line print the resulting output string. This could continue onto additional lines if newline characters are found in the output string. If an Accept state is defined, use a third line to indicate if exectution ended at that state (Accept) or not (Fail).

Sample Input

```
7 3
Start,+,positive,Even
Start,-,negative,Even
Even,0,,Even
Even,1, flip,Odd
Odd,0,,Odd
Odd,1, flip,Even
Even,?," ""Good,\nat last!""",Final
+1010?
-000111?
+abce
0 0
```

Sample Output

```
fsm 1.
1. +1010?
positive flip flip "Good,
at last!"
Accept
2. -000111
negative flip flip flip
Fail
3. +abcd
positive
Fail
```