Iceman was in a room in Iceland. He wants to get out of it, but it's not as easy as it seems. To help him, you need to know something about Iceland and Iceman.

The room could be described by an $n \times m$ grid, so there are n rows numbered 1 to n from top to bottom, each with m squares numbered 1 to m from left to right. Each square may be empty, icy or rocky. An empty square is denoted by '`.`', while a rocky square is denoted by '`X`'. Ice squares are a bit complex, so we talk about it later. The first and last rows and columns are all rocky. The iceman's initial location is always an empty square at the beginning, denoted by '`@`'. His destination is also empty at the beginning, denoted by '`#`'. What's more, the destination is always directly above a rocky square. Though the iceman looks bigger than a square, he always occupies exactly one single empty square.

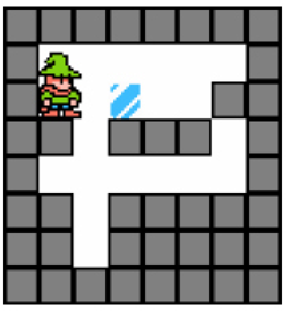
Fig 1. The iceman in the room

The iceman has four kinds of moves: go left (`L`), go right (`R`), magic wind left (`<`) and magic wind right (`>`). Suppose the iceman is at (r, c), then '`<`' move operates on the iceman's bottom-left square $(r+1, c-1)$. If the square is rocky, the move does nothing; if the square is empty, it becomes icy; if the square is icy, it becomes empty. There are four kinds of icy squares: ice with two free-ends (`O`), ice with left free-end (`[`), ice with right free-end (`]`), ice with no free-end (`=`). Here 'free' means 'not connected'. If an icy square is created by a magic-wind move, it connects to its left/right neighbor, provided the corresponding neighboring square is not empty (rocky or icy). The connections between neighboring squares are symmetric, so squares that are connected to each other behave as a whole. Connections could be created only by magic moves, and there are no vertical connections. We call the whole an ice bar. If an icy bar (no matter how many squares does it contain) has two free ends, we call it a free ice bar. Free ice bars immediate drop down vertically when all its supporting squares (squares directly below it) are empty. When an icy square is cleared, all connections of it (if any) are destroyed. Because rocky squares are fixed, if an ice bar connects to one or two rocky squares, it never drops down until its connections to rocky squares are all destroyed. The '`>`' move is symmetrical.

The '`L`' move is a little bit complex, compared to what you might expect. Again, suppose the iceman is located at (r, c). If $(r, c-1)$ is an empty square, the iceman go there. Now he stands on the square $(r+1, c-1)$, which might be empty. If this is true, he falls down until the square under him is not empty. The iceman can launch a move only when he's standing on a rocky or icy square, so when falling down, he cannot do anything. Now consider the second case, i.e. $(r, c-1)$ is rocky. Obviously the iceman cannot move to that square, so he checks the square above it and the square above himself (i.e the squares $(r-1, c-1)$ and $(r-1, c)$). If both are empty, he climbs to $(r-1, c-1)$, otherwise he remains at (r, c). The third and last case holds when $(r, c-1)$ is icy. In this case, the iceman tries to push it. The Iceman is not so powerful, so he can only push single (1×1) icy squares. That is, if only if $(r, c-2)$ is empty, the ice at $(r, c-1)$ is pushed left. It continues to move left until it is blocked by a rocky or icy square, or the square directly below is empty. In the latter case, the ice drops down, as stated before. Note that when stopped dropping, the ice does not move left again. Don't forget that when the ice is pushed away successfully, some free ice bars may drop down. The iceman does not move until everything stopped moving or dropping. If the ice at $(r, c-1)$ cannot be pushed, it is treated as a rocky square, so the iceman may climb on it.

Write a program to move the iceman to the destination with minimum number of moves.

Input

The input consists of several test cases. The first line of each case contains two integers, n and m $(1 \leq n, m \leq 10)$. This is followed by n lines, each containing m characters. Each character is one of '`.`', '`X`', '`@`', '`#`', '`O`', '`[`', '`]`', '`=`'. The last test case is followed by a single zero, which should not be processed.

Output

For each test case, print the case number and the move sequence. There is always a solution of at most 15 moves. It is guaranteed that the optimal solution is unique.

Sample Input

```
55
XXXXX
X@.#X
XX.XX
X...X
XXXXX
77
XXXXXXX
X.....X
X@[=].X
XXX.XXX
XXX.XXX
XXX#XXX
XXXXXXX
66
XXXXXX
X@...X
XXXX=X
X..O.X
X.#O.X
XXXXXX
0
```

Sample Output

```
Case 1: >RR
Case 2: R>R
Case 3: RR>RLLLLL>R
```