3588 Logical Mazes

You have probably all seen maze problems where the maze is described using ASCII characters. For example, a maze might be described as:

```
.XX
...
XX.
```

where '.' represents a free space in the maze and 'X' represents a wall.

But there are other ways to represent mazes that are based on the logical structure of the maze instead of its appearance. One such way uses parentheses and underscores to represent the structure of a maze. These are used as follows:

- '_': a location occupied by a wall;
- '(*north−loc−desc*, *west−loc−desc*,*south−loc−desc*,*east−loc−desc*)': an empty location that has not been described previously. Within the set of parentheses will be four other descriptions, defining the neighbors of this empty location;
- '*': an empty location that has already been described.

Any space outside the boundaries of the maze will be represented as containing a wall, but these extra walls will not be shown in the final maze. So the description "(_,_,_,_)" describes an empty location with walls to its north, west, south, and east, or the maze:

```
.
```

(That's just a single empty location.)

A slightly more complicated maze is represented by the description "(_,_,(*,_,_(_,*,_,_)),_)". Here there are walls to the north and the west of the starting empty location. The cell to the south of the starting location is described by "(*,_,_(_,*,_,_))". The cell to the north is a cell that's previously been described. There are walls to the west and south. The cell to the east is defined by "(_,*,_,_)", an empty cell with walls to the north, south, and east and a previously described location to the west. All of this description ultimately gives the maze:

```
.X
..
```

You are to write a program that will translate from the parenthesized description of a maze to an ASCII version of the maze.

Input

The input to your program will be one or more data sets, each with a description of a maze, in the parenthesized version described above. The first line of each data set will be a pair of integers, '*row*' and '*column*', with $0 \leq row \leq 20$, $0 \leq column \leq 20$, giving the overall maze dimensions. The values row and column will only be equal in the maze indicating the end of input.

The second line of each data set will be a pair of integers '*startrow*' and '*startcolumn*', with $0 \leq startrow < row$, $0 \leq startcolumn < column$, giving the location initially being described, where (0, 0) is the upper left hand location of the maze. There will then be a syntactically valid description of the starting location and adjacent locations. There may be extra blanks and ends of line in the data.

The last input set will have the *row* and *column* equal to '0'. This set should not be processed.

Output

For each maze, first output a line with the number of the maze, formatted as in the sample output and starting with 1, then the ASCII version of the maze, using '`X`' to represent a wall and '`.`' to represent an empty location. You should assume all descriptions are valid and any space not described is a wall.

Sample Input

```
5 3
0 1
(_,_,_,_)
5 5
2 2
(_,_,(*,_,_(_,*,_,_)),_)
3 3
0 0
( _,
  _,
  ( *,
    _,
    _,
    ( _,
      *,
      _,
      ( _,
        *,
       (*,_,_,_)
       ,_)
      )
    ),
  _)
0 0
```

Sample Output

```
Maze number 1
X.X
XXX
XXX
XXX
XXX
Maze number 2
XXXXX
XXXXX
XX.XX
XX..X
XXXXX
Maze number 3
.XX
...
XX.
```