# A Game of Falling Bricks!

Imagine playing a game where bricks of different shapes are dropped into the box. The dropped bricks do not undergo any angular displacement (no rotation & no sideways movement), they just fall down in a linear fashion; one after the other. There will be 7 different types of bricks, labeled A through G as indicated below.



Our game will be played with a box that is eight squares wide (numbered 0 to 7 from left to right). A shape dropped in a certain position falls until it strikes a previous piece, or hits the bottom. If an entire row of squares is filled with bricks (with no gaps between), that row ‘disappears’; and all the rows above this eliminated row drops down without any change in the row composition. The diagram below indicates this phenomenon. Here, an X indicates a square filled by a brick. No other adjustment (e.g., filling up of gaps) takes place due to this.



The objective of this exercise is that you’ll be given a series of brick drops and asked to show the resulting board.

## Input

There will be multiple test cases, each test case depicting a series of brick drops. Each test case will begin with an integer n (n < 100) on a line indicating the number of bricks to drop onto the box for that test case (n = 0 indicates end of input; i.e., end of all test cases). The next line will consist of n pairs, s c, separated by a single space, indicating that shape s will drop with its left-most square in column c (numbered 0 to 7 from left) and .



There could be several pairs of n & multiple s c pairs, each representing a test case. At the end of all the test cases, n will have a value of 0. Look at the following input:

6

B 2 A 0 A 0 E 2 A 5 G 4

2

G 4 G 0

8

F 0 F 1 F 2 F 3 F 4 F 5 F 6 F 7

3

D 0 B 1 D 2

0

The first number 6 indicates that the first test case has 6 bricks falling in the following order: brick of type B from column 2, followed by brick of type A from column 0, followed by type A brick from column 0, followed by type E brick from column 2, followed by type A brick from column 5 and lastly, type G brick from column 4. Your program will have to depict the resulting board after the fall of these 6 bricks. Once you do that, you need to clear the board and start processing the next input. In the above example, the number 2 indicates the next test case which has two bricks falling. So on and so forth, till you reach the end where n has a value of 0. In the above example, there are 4 test cases in all.

It is important to note that no shape will be illegally dropped ‘outside’ the box (for example, G 5 would be an illegal drop). Therefore, you do not have to validate such illegal inputs. However, the format of input is very strict as follows:

* Each n will start in a newline
* The line immediately after n will contain n pairs of s and c, each separated by a blank space
* There would be no empty lines (carriage returns) or tabs in the input
* There will be no input after a line where n’s value is 0

## Expected Output

For each test case, you will output a line indicating how many rows have squares filled in it and then print out the box composition. This should be repeated for each test case. The box should be printed as rows of ‘X’ (indicating a filled square) and ‘.’ (indicating an empty square). The top non-empty row in your resulting board is printed first. There should be no empty lines between outputs of test cases. The expected output for the sample input given above is mentioned here:

3

XX.X....

XXX..XX.

XXXX.XX.

0

0

5

..XX....

.XX.....

.XX.....

XX......

X.......

It may be noted that for an output of 0, there would be no rows of “X” and “.” (in the above example, two test cases gave an output of 0). Needless to say, the format of output expected is very strict. The rules are:

* For each input test case, you will output ‘n’ depicting the number of resulting rows in a newline
* The next ‘n’ lines immediately after ‘n’ will contain ‘X’ and ‘.’ depicting the actual resulting board composition (if ‘n’ is zero, there would be no such lines)
* There would be no spaces, tabs or other white spaces between the ‘X’ and the ‘.’
* The total number of ‘X’ and ‘.’ would always be 8 in any row
* There would be no empty lines (carriage returns) or tabs in the output

Since the solutions would be graded by an automated solution tester program, it is very important for your program to strictly adhere to the above output rules lest your solution may risk the possibility of being marked as incorrect.

## General Instructions

Here are a few general instructions for your program:

* The solution should be provided only in Java 1.4 or higher (core Java skills are sufficient for this program). You are expected to submit the “.java” file(s) to a particular email id.
* It is mandatory that the submitted files do not have any compilation problems.
* Your program should be executable in the command line (i.e., at least one of your “.java” files should have a “main” method).
* The file naming convention should be “<programFileName>\_<emp-id>.java”. E.g., fallingBricks\_12977.java.
* The test data should be read from the standard input and results (output) should be written to standard output. Do **not** use files for reading input and redirecting your output.
* Your solution will be tested and evaluated by an automated “solution tester” program. Hence, if you have not adhered to the above rules, your program may be rejected.
* The solution submitted is liable to rejection if
  + it is found to be copied from the internet or any other source.
  + the source code has compilation errors.
* In case of any dispute the decision of the judges would be final
* As soon as you solve the problem you should submit the source code by email to “\_DVCI Technical Competition”