D Tetris (`tetris.{c,cc,java}`)

D.1 Description

Tetris is a classic puzzle game where the goal is to drop tetrominoes into a board in such a way that rows are completely filled. Developing automated Tetris players is an interesting topic, but not suited for a programming contest. In this problem you will deal with an automated player for a much simpler version of Tetris.

The rules for our simplified Tetris are as follows:

- The board is 4 columns wide, 4 rows tall.
- There are four pieces, numbered as follows:

Piece 0	Piece 1	Piece 2	Piece 3
`*` `**`	`*` `*` `*`	`*` `*`	`*`

- The player receives one piece at a time. The player must decide how many times to rotate the piece 90 degrees clockwise, which column to drop the piece in, and then drop the piece straight down from above the board. The piece must fit entirely within the columns of the board. Note that unlike in Tetris, the piece may not be moved left or right while "in transit" downward.
- After a piece has been dropped, each row that is completely filled with pieces is removed, and the rows above it (including those above the top (4th) row of the board) are shifted downward grouped by row. (There is no "gravity" effect; individual filled cells do not move downward to fill "gaps." This can cause pieces to remain suspended in midair, such as in the second case in the sample run below.)
- If, after the drop and row removal, any part of a piece lies above the top (4th) row of the board, the player loses. Otherwise, the player receives the next piece.

Your job in this problem is to provide pieces to an automated player with the intent of forcing it to lose. It turns out that if the board is empty, this task is impossible if the player is perfect; however, there exist nonempty starting boards such that it is possible to force a player to lose. After each piece you provide, the player will respond with the location and orientation of the piece, information that you may use to choose the next piece to provide.

D.2 Input/Output

This is an interactive problem, so the input will be dependent on your output. Please read the following specification carefully.

There are multiple test cases. Each test case will be dealt with as follows:

The first 4 lines given have 4 characters each, representing the starting state of the board. An asterisk (*) denotes a filled grid square, and a dash (-) denotes an empty grid square. No line will consist solely of asterisks; that is, no line will begin completely filled.

If it is always possible to keep playing forever given that starting state, print -1 on a single line, which will conclude the test case. Otherwise, print the number of the piece p_i to give to the judge player on a single line. The judge will respond with a single line with two numbers c_i and k_i, separated by a space. c_i = the (0-indexed) leftmost column occupied by the piece, and k_i = the number of times to rotate the piece 90

degrees clockwise. After that, repeat the process by printing another piece number. Continue to do so until the judge responds with $c_i = k_i = -1$, indicating that the judge is unable to place the piece without causing the board to overflow beyond 4 rows. If the judge returns nonnegative values for c_i and k_i, it is guaranteed that placing the piece in the given position will not overflow the board beyond 4 rows after placement and filled row removal. This will conclude the test case.

After the last test case will be 4 lines with 4 asterisks each. This case should not be processed.

IMPORTANT: After each line you print, make sure to flush the output stream.

For C++, use	`cout << flush;`
For C, use	`fflush(stdout);`
For Java, use	`System.out.flush();`

For example:

Input to your program (from judge)	`----` `----` `----` `----`
Output of your program (to judge)	`-1`

Input to your program (from judge)	`-*--`
	`*---`
	`-*--`
	`*---`

Output of your program (to judge)	`1`
Input to your program (from judge)	`2 0`
Output of your program (to judge)	`1`
Input to your program (from judge)	`3 0`
Output of your program (to judge)	`1`

Input to your program (from judge)	`-1 -1`
	`****`
	`****`
	`****`
	`****`

Warning: Your solution will be judged as having exceeded its time limit if it attempts to cause the judge to lose on a board where the judge can play forever. Your solution will be judged as a wrong answer if it produces unexpected output, such as an invalid piece number or a -1 flag after already providing some pieces for the test case.