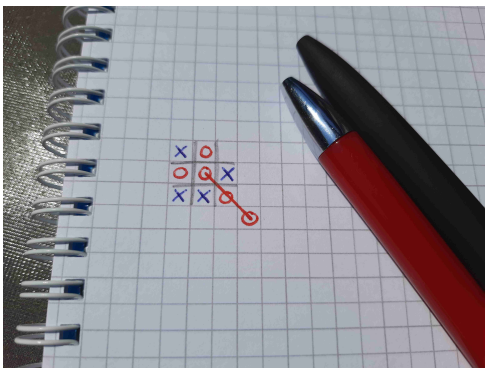
Problem C: Crosses and Circles

Time limit: 1 second



An unusual game of tic-tac-toe.

Your friend watched some video about Connect Four and memorized the winning strategy for the starting player. What a loser! Whatever they are saying sounds about as relevant to your life as this paragraph is to the problem statement. Apparently, someone found a weak solution to Connect Four which can be compressed to just 150 kB and requires no game tree search.

To get them off their high horse, you challenge them to a real game. You pull out a piece of A4 graph paper and explain: "We will take turns picking any unpicked cell. The first player to get three consecutive cells in any row, column, or diagonal wins. I will start." Your friend is barely listening, but this is not just regular tic-tac-toe: you are playing on the whole paper. Convert your winning advantage.

Interaction

This is an interactive problem. Your submission will be run against an *interactor*, which reads from the standard output of your submission and writes to the standard input of your submission. This interaction needs to follow a specific protocol:

There is no initial input as you make the first move.

On each of your turns, print one line with two integers r and c ($1 \leq r \leq 59$, $1 \leq c \leq 42$), the row and column of the cell you pick. The cell must not have been picked before.

After each of your moves, the interactor will respond with one line with two integers r' and c' ($0 \leq r' \leq 59$, $0 \leq c' \leq 42$).

- If $r' = c' = 0$, the interaction is over. This happens if you won on your last move, made an illegal move, or your friend can win with their next move. Your program should terminate immediately.
- Otherwise, $1 \leq r' \leq 59$ and $1 \leq c' \leq 42$. This means that your friend picked the cell in row r' and column c'. It is guaranteed that neither player has picked this cell before. It is now your turn again, and the interaction continues from there.

The interactor is adaptive, so your friend's moves will depend on all previous moves.

Make sure you **flush** the standard output after every output. For example, you can use `fflush(stdout)` in C++, `System.out.flush()` in Java, `sys.stdout.flush()` in Python, `std::io::stdout().flush()` in Rust, and `hFlush stdout` in Haskell.

A *testing tool* is provided to help you develop your solution.

Sample Interaction 1

Read	Write
	`9 7`
`10 6`	
	`9 6`
`9 8`	

```
8 7
10 7
10 8
8 6
11 9
0 0
```