Detecting the Missing Ship

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 mebibytes

This is an interactive problem.

The oceanic expedition of the research ship was going well, but at one point the scientists entered an area where they lost contact with the shore. The first issue is that they cannot transmit the research data back to the mainland. Unfortunately, they will not be able to proceed and will have to return. But the second issue is that due to the loss of communication, there is no reliable way to get back! The group of scientists who remained on the shore has already sent a drone that will find the location of the ship and then guide it back to shore.

The area of the ocean being studied is represented as a table of $n \times n$ cells. The ship occupies $k > 1$ consecutive cells either vertically or horizontally. The rows of the table are numbered from top to bottom from 1 to n, and the columns are numbered from left to right from 1 to n. With one query, the drone can choose a cell (r, c). If the ship has at least one cell in the r-th row **or** in the c-th column, then the search is successfully concluded. Otherwise, after the query, the ship may either stay in place or move one cell in either direction depending on its orientation. The ship cannot leave the studied area of the ocean; all cells of the ship must remain within the table.

An example for $n = 5$, $k = 2$, and checking the drone's position $(r, c) = (1, 2)$ is shown in Figure 1. The arrows indicate where the ship can move after the query.

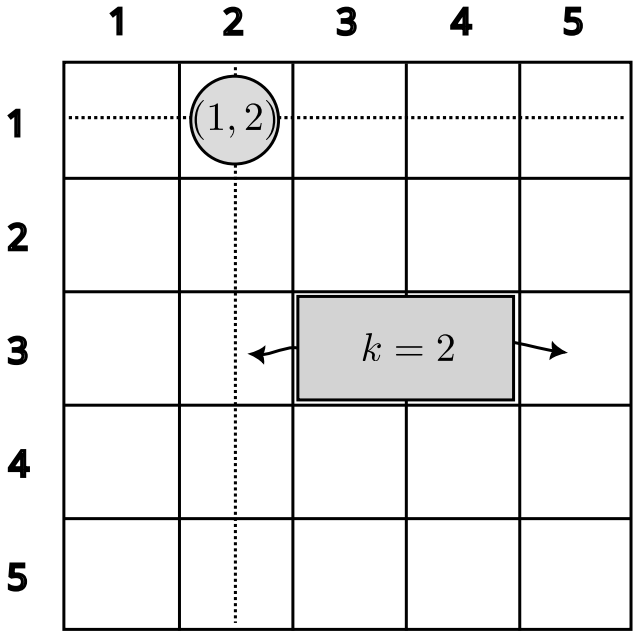

Figure 1. Example of a move

The goal is to find the ship within the minimum possible number of moves by hitting a row or column that definitely contains at least one cell of the ship.

Input

The first line of the input data contains two integers n and k ($3 \le n \le 10$, $2 \le k \le n$), specifying the size of the table and the length of the ship.

Interaction Protocol

To make a query, output two integers r and c ($1 \le r, c \le n$). The response to the query will be written

into the standard input stream: you will receive the number 0 if the ship has not yet been found, or the number 1 if it has been found. As soon as you read the number 1, the program must be terminated immediately. If you read the number 0, you may make a new query.

Please note that the interactor is **adaptive** and may not necessarily have a fixed strategy of ship movements.

If you make more queries than the minimum necessary to guarantee finding the ship, or if the query coordinates are outside the studied area of the ocean, your program will be judged as `Wrong Answer`, and the response to the query will be the number -1. Upon reading the response -1, the program must be terminated immediately. If your solution does not terminate immediately after finding the ship, or if it terminates before finding the ship, or if it continues execution after reading -1, it will be considered incorrect, and the verdict may be undefined.

After each query, do not forget to output a newline and flush the buffer to the standard output stream. You can use the following functions:

- `fflush(stdout)` or `std::cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see the documentation for other languages.

Example

standard input	standard output
3 2 1	2 2

Note

In the example, we make one query to cell $(2, 2)$ that allows us to find the ship in one move regardless of its location. In response to this move, the number 1 is read, after which the program terminates because it has found the ship.