Problem: Filler

Input file:	`standard input`
Output file:	`standard output`
Time limit:	10 seconds for 10 moves
Memory limit:	64 megabytes

In this problem you are to play well-known game "Filler". In this case we will consider a field of size 20×20. Every cell of the field initially is painted in one of 10 colors. Initially first player *owns* the cell in top-left corner and all cells, which are accessible from top-left by moving to adjacent cells of the same color. Here two cells are considered adjacent if they share a side. Second player *owns* the cell in bottom-right corner (and all cells accessible from there in the same way).

At every move player can choose any color (except the color of opponent's source cell, top-left or bottom-right respectively) and paint all cells he *owns* into that color. After that player begins to *own* all cells which are accessible by moves to adjacent cells of the same color from his initial cell. The game ends when every cell is owned by one of the players or after 400 moves (which os these happens earlier).

Input

Before every move your program will receive current field state in standard input. You will get 20 lines with 20 numbers in each. After that you will receive an empty line. Every number a_{ij} denotes the color of one cell (colors are numbered from 1 to 10). The field will be rotated in such way, that your initial cell will be top-left cell (with coordinates $(0, 0)$).

Output

At every move you should output the number of ther color, in which you want to paint your cells. Output an empty line after each move. Don't forget to flush the output buffer after each move. See sample programs for further clarification.

Also please note that your program should not terminate by itself. You can make a dummy read operation after last move to hang without consuming CPU time.

Example clarifies output for one move.

Examples

standard input
8 2 4 9 2 8 7 1 6 4 1 7 6 2 9 4 7 7 8 6 6 6 1 2 5 5 1 5 1 5 6 7 5 4 5 7 6 3 5 6 4 1 1 3 1 2 1 7 2 2 8 8 6 8 3 10 1 6 10 1 9 8 4 1 6 6 7 1 3 5 5 2 3 5 9 8 6 1 5 5 1 3 10 9 2 6 10 4 9 8 3 6 7 8 3 2 3 2 9 1 9 7 2 8 4 5 5 10 5 5 3 5 8 7 9 9 7 1 3 5 5 1 6 5 1 1 6 8 7 3 4 10 3 10 7 7 6 1 10 6 6 8 1 6 10 10 5 1 9 3 1 2 4 8 9 7 1 9 3 6 5 5 7 6 3 2 8 7 10 4 3 9 1 2 5 1 4 6 4 10 7 7 3 6 7 9 9 1 3 2 5 1 7 7 7 4 6 6 5 1 8 6 8 4 8 5 10 9 5 5 7 7 3 7 6 6 8 10 8 3 5 5 10 8 10 1 8 8 5 6 8 2 5 3 3 6 7 9 2 2 6 9 10 3 4 3 7 9 2 1 10 8 7 9 4 2 3 1 8 9 4 8 6 1 1 1 2 4 2 5 7 5 1 8 7 6 5 9 5 2 4 6 7 10 3 6 4 6 2 6 2 10 8 4 10 1 10 6 5 5 4 9 4 9 4 6 1 7 10 7 1 3 2 10 8 3 1 9 8 9 6 9 8 10 8 8 2 3 2 1 7 6 2 7 6 5 8 1 10 3 10 7 9 3 5 10 4 10 10 8 9 5 8 6 6 4 8 7 10 6 8 4 7 7 1 5 3 4 8 3 3 10 5 6 7 10 2 4 6 2 1 5 9 3 6 10 9 4 2 1 8 7 6 6 3 7 3 5 3 7
standard output
6

Sample program for C++:

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cstdlib>

using namespace std;

const int SZ = 20;
int field[SZ][SZ];

int main()
{
    srand((unsigned)time(NULL));
    int i, j;
    while (1)
    {
        for (i = 0; i < SZ; i++)
        {
            for (j = 0; j < SZ; j++)
            {
                scanf("%d", &field[i][j]);
            }
        }
        int newc = rand() % 10 + 1;
        while (newc == field[SZ - 1][SZ - 1]) newc = rand() % 10 + 1;
        printf("%d\n\n", newc);
        fflush(stdout);
    }
    // to prevent early program termination
    scanf("%d", &i);
    return 0;
}
```

Sample program for Pascal:

```
const SZ = 20;
var field : array[1..SZ, 1..SZ] of integer;
i, j : integer;
newc : integer;
begin
randomize;
while true do begin
        for i := 1 to SZ do begin
         for j := 1 to SZ do begin
         read(field[i, j]);
         end;
        end;
        newc := trunc(random() * 1000) mod 10 + 1;
        while (newc = field[SZ, SZ]) do newc := trunc(random() * 1000) mod 10 + 1;
        writeln(newc);
        writeln;
        flush(output);
    end;
    read(i);
end.
```