Problem: Snake - 2

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

Many people can remember the game called "Snake which is widely known in chinese game consoles as well as on cell phones some well-known manufacturer. In this problem you are to write a program which plays this game,

The game field is a matrix of 30×30 cells. Every cell can contain a part of one of two snakes, a wall or an apple. Every snake occupies several cells, numbered starting from one. At every moves snake chooses some moving direction. After that first cell of the snake moves in that direction and every other cell is moved to the place of previous cell. If the first cell of the snake moves to a cell with an apple, snake becomes one cell longer. New cell is added to the place, where the last cell of the snake was before moving.

Apples appear on the field randomly, approximately one apple for 10 moves (here move is a move of one snake and response move of the other).

Initially every snake consists of one cell and resides in the cell $(4, 4)$ or $(25, 25)$ (rows and columns are numbered from zero).

If a snake moves into a cell occupied by any snake, or into a cell occupied by wall, or outside the field, that snake is considered a loser. If after snake's move it length reaches 250 cells, the snake is considered a winner.

Input

You will receive current field state before every move in stdin. You will receive 30 lines with 30 numbers in each. After that you will receive an empty line. Every number a_{ij} corresponds to one cell. If $0 < a_{ij} \leq 250$, then there's your snake's cell number a_{ij} in that cell. If $a_{ij} < 0$, then there's your opponent snake's cell with number $-a_{ij}$ in that cell. If $a_{ij} = 1000$, then there's an apple in that cell. If $a_{ij} = -1000$, there's a wall in that cell. Otherwise $a_{ij} = 0$ and the cell is empty.

Output

You should output one direction for each move. Direction is denoted by one letter: `U` — up, `L` — left, `D` — down, `R` — right. Output an empty line after each move. Don't forget to flush output buffer after each move. See sample programs for further clarification.

Examples

standard input

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 3 2 1 1000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -2 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -3 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -6 -5 -4 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -7 -8 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

standard output

```
R
```

In this example moving snake will become one cell longer after move.

Sample program for C++:

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

using namespace std;

const int SZ = 30;
int field[SZ][SZ];
const char dir[5] = "LURD";

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]);
            }
        }
        printf("%c\n\n", dir[rand() % 4]);
        fflush(stdout);
    }
    return 0;
}
```

Sample program for Pascal:

```
const SZ = 30;
const dir : array[0..3] of char = ('L', 'R', 'U', 'D');
var field : array[1..SZ, 1..SZ] of integer;
i, j : 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;
        writeln(dir[trunc(random() * 1000) mod 4]);
        writeln;
        flush(output);
    end;
end.
```