.NET - Martian Robots

The problem

The surface of Mars can be modelled by a rectangular grid around which robots are able to move according to instructions provided from Earth. You are to write a program that determines each sequence of robot positions and reports the final position of the robot.

A robot position consists of a grid coordinate (a pair of integers: x-coordinate followed by y-coordinate) and an orientation (N, S, E, W for north, south, east, and west). A robot instruction is a string of the letters "L", "R", and "F" which represent, respectively, the instructions:

- Left: the robot turns left 90 degrees and remains on the current grid point.
- Right: the robot turns right 90 degrees and remains on the current grid point.
- Forward: the robot moves forward one grid point in the direction of the current orientation and maintains the same orientation.

The direction North corresponds to the direction from grid point (x, y) to grid point (x, y+1).

There is also a possibility that additional command types may be required in the future and provision should be made for this.

Since the grid is rectangular and bounded (...yes Mars is a strange planet), a robot that moves "off" an edge of the grid is lost forever. However, lost robots leave a robot "scent" that prohibits future robots from dropping off the world at the same grid point. The scent is left at the last grid position the robot occupied before disappearing over the edge. An instruction to move "off" the world from a grid point from which a robot has been previously lost is simply ignored by the current robot.

The input

The first line of input is the upper-right coordinates of the rectangular world, the lower-left coordinates are assumed to be 0, 0.

The remaining input consists of a sequence of robot positions and instructions (two lines per robot). A position consists of two integers specifying the initial coordinates of the robot and an orientation (N, S, E, W), all separated by whitespace on one line. A robot instruction is a string of the letters "L", "R", and "F" on one line.

Each robot is processed sequentially, i.e., finishes executing the robot instructions before the next robot begins execution.

The maximum value for any coordinate is 50.

All instruction strings will be less than 100 characters in length.

The output

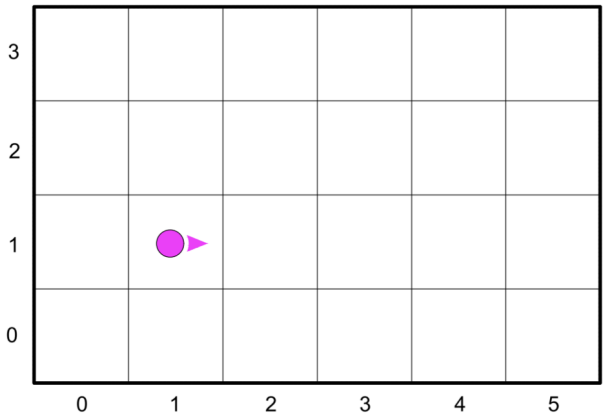
For each robot position/instruction in the input, the output should indicate the final grid position and orientation of the robot. If a robot falls off the edge of the grid the word "LOST" should be printed after the position and orientation.

Sample input

```
5 3
1 1 E
RFRFRFRF
3 2 N
FRRFLLFFRRFLL
0 3 W
LLFFFRFLFL
```

Sample output

```
1 1 E
3 3 N LOST
4 2 N
```



Graphic visualisation of the grid and the first robot of the sample input (Grid 5,3 - Robot 1,1, E)

Minimum requirements

- The app should properly work with at least the example input.
- The app should be hosted in a **GitHub public repository**.
    - It should have a README.md describing the product and explaining how to run it.

Features hints and nice to have

The solution to the described problem should be the core of the business logic implemented by your **.NET** app but Guidesmiths is not setting any limits regarding the functionalities and complexity of the product that you are going to implement, this is something that we will leave up to you. For this reason here are a couple of hints and ideas:

- Libraries and third party code. You can use as many libraries as you want.
- Testing. You can go for:
    - Unit / Integration / End2End / all of them.
    - Mocha / Jest / Any other interesting testing framework.
- Application running (execute the process over a given input). You can go for:
    - Simple script approach.
    - REST API approach.
    - CLI approach.
    - All of the above.
- Persistence layer.
    - You can stores somewhere a lot of different info:

        - Inputs, output, etc.
        - You can even decide to store extra information calculated from the outcome of the execution (e.g. lost robots, explored surface, etc.)

    - You can go for any of the following approaches

        - Simple .json / .txt / ... file.
        - Relational / Non-Relational DB.
        - Any other interesting approach.

- Insight:

    - You can expose some interesting information using API endpoints

        - On the fly calculated information
        - Previously stored information (see above e.g. lost robots, explored surface, etc.)

- Shipping. You can go for:

    - Docker.
    - Deploy it somewhere (heroku / your kubernetes cluster / any other interesting place).