Hey You, a part of Beat the Q Pty Ltd
Level 2, 102 - 106 Commonwealth St.
Surry Hills, NSW 2010
Ph: 1800 220 153

Full-stack take home exercise

The aim of this exercise is to show off your technical skills and for us to understand your experience level with various tools and technologies. Please treat this exercise as if you were going to deploy it into a production environment, so consider things like code structure, testability, responsive design etc… Feel free to use any existing libraries, tools and/or frameworks to complete the task and cite / reference these in the `README.md`.

Background

Nigel (an avid $C_8H_{10}N_4O_2$ fuelled engineer) has been tinkering behind his coffee shop and has hacked together a robotic controller and plans to use it to program his fleet of robots to deliver coffee around his store to his customers. As the Robots will be operating in a real-world environment, each Robot is programmed and will execute their commands in parallel. Eventually he wants to roll this out to all his stores worldwide but needs help writing a simulation program that will validate his robotic fleet controller.

Part 1 - the basic simulation

Nigel wants a simulator for this robotic controller built such that he can test various scenarios and commands before going to worldwide rollout. Write a program that will execute a simulation given the following input format.

Input Format

The first line contains two space separated integers, which indicate the maximum size of the shop. The shop is partitioned into a grid, and indexed according to Matrix Convention (see Appendix 1).

The rest of the input is information for programming the Robots. Each robot consumes two lines of input.

- The first Robot input line contains three space separated characters
  - X, Y - Integers representing the coordinate of the starting position of the Robot
  - H - Character representing one of the four cardinal compass points (**N**orth, **S**outh, **E**ast, **W**est) for the starting heading of the Robot.
- The second Robot input line contains the commands for moving the Robot, it is a string of letters made up of the possible letters
  - '**L**' - The robot will turn 90 degrees left, without moving from its current location
  - '**R**' - The robot will turn 90 degrees right, without moving from its current location
  - '**M**' - The robot will move forward one grid point, maintaining the same heading.

Once all input has been processed, the program executes and the simulation will run. The robots execute their commands in 'parallel'. That is, they all execute their first command 'at the same time'. If there are no commands left for a robot, that robot will just stay in its last position until simulation ends.

Output format

The output is the final state of the Robots in the shop after they have executed all their commands.

Unlike the digital world, Robot are physical and must not be permitted to bump into, or run over each other - your program should detect this and instead output a message; “Robot Collision Detected”

Sample Input

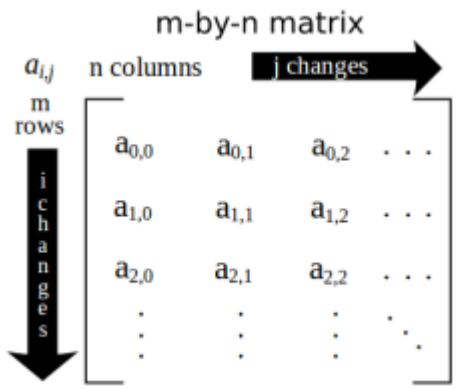
	Line-by-line annotation
5 5 1 2 N LMLMLMLMM 3 3 E MLMLMRMRMRRM	Grid is size 5 x 5 1st Robot’s initial position (1,2) and heading (N) 1st Robot’s set of commands 2nd Robot’s initial position (3,3) and heading (E) 2nd Robot’s set of commands

Sample Output

	Line-by-line annotation
1 1 N 3 1 W	1st Robot’s final position and heading 2nd Robot’s final position and heading

Appendix 1 - Matrix Convention

Note: If a discrepancy exists between this and the linked source, this document has priority.



The board follows the matrix convention of Mathematics.
A board of size m x n, has top-left indexed (0, 0) and bottom-right indexed (m-1, n-1).
The row index increases from top to bottom, and the column index increases from left to right.

The square directly NORTH from (x,y) is (x, y-1).
The square directly EAST from (x,y) is (x+1, y)

Image source: https://www.hackerrank.com/scoring/board-convention

Part 2 - API for world-wide rollout

Nigel is ready for a world-wide rollout and wants to let his franchise owners also run simulations themselves. He decides that he needs an API for his franchise owners to use so they can run simulations remotely. Below is a list of endpoint requirements. Use appropriate HTTP response codes.

POST /shop

Creates a new shop, with the required dimensions

Request	Response

{
width: int
height: int
}
	{
	id: int
	width: int
	height: int
	}

GET /shop/:id

Retrieves a shop by its id. The list of Robots is in the same order that they were added.

Request	Response

	{
	id: int
	width: int
	height: int
	robots: [
	{ < robot > },
	]
	}

DELETE /shop/:id

Deletes the shop, ALL Robots attached are also deleted

Request	Response

	{
	status: “ok”
	}

POST /shop/:id/robot

Creates a new robot in the shop

Request	Response

{
x: int
y: int
heading: char
commands: string
}
	{
	x: int
	y: int
	heading: char
	commands: string
	}

PUT /shop/:id/robot/:rid

Updates the properties of the Robot

Request	Response

{
x: int
y: int
heading: char
commands: string
}
	{
	x: int
	y: int
	heading: char
	commands: string
	}

DELETE /shop/:id/robot/:rid

Deletes the robot from the lawn

Request	Response

	{
	status: “ok”
	}

POST /shop/:id/execute

Runs a simulation in the shop using the Robots. The Robots have their positions updated and the shop is an accurate representation of the final state. If any errors occur, these should be returned to the client. The list of robots is in the same order that they were added to the Shop.

Request	**Response**

	`{`
	` id: int`
	` width: int`
	` height: int`
	` robots: [`
	`  { < robot > },`
	` ]`
	`}`

You can decide on the error conditions and response payload (if any). These should be based on the assumptions you made in part 1.

Bonus - deploying the application

It's boring if no-one can use your application. So time to deploy it to somewhere useful :)

- Deploy the application to a hosting platform such as Heroku or AWS. Include a URL in the `README.md`
- Have unit tests running using a continuous integration (CI) tool such as Travis. Display the status in the `README.md` file in your repository.

To submit your solution

We recommend using a public or private repository such as Bitbucket or GitHub and share the repository with sekmun@heyyou.com.au or send us a link.