The Starship Problem

You're the chief engineer of the USS FreeFlyer, Starfleet's fastest starship. Your job is to execute a series of maneuvers to move the FreeFlyer to its destination, which can be expressed as an [x,y] coordinate in a two-dimensional grid. Your starship can process the following commands:

Command	Action
`L`	Rotate the FreeFlyer left (counter-clockwise) by 90 degrees. This command reorients the vehicle without changing its velocity vector. This action takes 1 unit of time, during which the FreeFlyer continues to cruise along its previous velocity vector.
`R`	Rotate the FreeFlyer right (clockwise) by 90 degrees. This command reorients the vehicle without changing its velocity vector. This action takes 1 unit of time, during which the FreeFlyer continues to cruise along its previous velocity vector.
`T`	Fire thrusters. This is an instantaneous acceleration that increases the FreeFlyer's velocity by 1 unit of velocity in the direction the FreeFlyer is facing. This action occurs immediately and then the FreeFlyer "cruises" for 1 unit of time.
`C`	Cruise. The FreeFlyer cruises for 1 unit of time, during which its position updates based on its velocity.

Definitions

- **Position**: The USS FreeFlyer always starts at the origin of the grid, `P = [0,0]`. Use the starship's velocity to compute its updated position at each time step. Remember, the change in position = velocity * time, and every command takes 1 time unit.
- **Velocity**: The USS FreeFlyer always starts at rest with a velocity of `V = [0,0]`. The velocity is updated instantaneously when thrusters are applied. The thrusters add 1 unit of velocity to the horizontal or vertical component of the velocity vector, based on the starship's current orientation. For example, if the FreeFlyer is pointed in the positive x-direction and fires its thrusters, `[1,0]` would be added to the velocity. If the FreeFlyer is pointed in the negative y-direction and fires its thrusters, `[0,-1]` would be added to the velocity vector.
- **Orientation**: The USS FreeFlyer always starts pointing in the positive x-direction, `O = [1,0]`. The orientation should be used to determine how the velocity changes when thrust is applied. Note that the orientation is not coupled to the starship's motion - the starship can be moving in one direction while pointed in a different direction. Note that the only valid values of orientation are `[1,0]`, `[0,1]`, `[-1,0]`, `[0,-1]`.

Example maneuver plans

Here are some example maneuver plans and their result:

Maneuver Plan	Final Position, Velocity, and Orientation	Description

`TLLT`	`P = [3,0]`
	`V = [0,0]`
	`O = [-1,0]`
		FreeFlyer thrusts in the positive x-direction, turns around and thrusts in the negative x-direction, causing it to stop.

`TLLTRTRRCCCT`	`P = [3,6]`
	`V = [0,0]`
	`O = [0,-1]`
		FreeFlyer moves in the positive x-direction 3 units and then in the positive y-direction by 6 units. Note that the FreeFlyer cruises towards the end of the maneuver plan before stopping.

`LTLLTTRRT`	`P = [0,0]`
	`V = [0,0]`
	`O = [0,1]`
		FreeFlyer moves in the position y-direction, turns around, and returns to the origin.

Problem statement

Write a program that reads a maneuver plan from a single line of standard input and then outputs to standard output the final position, velocity and orientation in the following format:

```
[Px,Py]
[Vx,Vy]
[Ox,Oy]
```

where `[Px,Py]`, `[Vx,Vy]`, and `[Ox,Oy]` are the final position, velocity, and orientation, respectively. All values should be integers.

We will attempt to compile and/or execute your program in Windows (via PowerShell) and/or Linux command-line environments by piping a maneuver plan to your program. For example:

```
echo TLLT | ./program
[3,0]
[0,0]
[-1,0]
```

What we look for

- **Language**. Please choose between C, C++, C#, Python, and Java. The team has a preference for solutions in C++, but feel free to use the language you are most comfortable with.
- **Style**. We prefer clean, conventional code that leverages object-oriented programming.
- **Documentation**. Please include a `README` file that outlines how to build and/or execute your program.
- **Software dependencies**. We prefer a solution that does not require dependencies. If your program does contain external software/library dependencies, please make note of it in the `README`.
- **Packaging**. Please e-mail us a tar.gz or .zip file containing your source code and documentation.
- **Discussion**. Please be prepared to talk about your solution and to discuss how the inverse problem would be solved: How would you write a program to generate a maneuver plan to navigate the USS FreeFlyer to a specified target coordinate?