Robot simulator

Program structure

The program consists of two classes:

- TableTop
- Robot

and some helper functions defined in:

- Direction
- Coordinate

A brief explanation of each is found further down. Each pair of *.cpp* and *.hpp* files also come with a test suite in the same folder.

The outline of the program is the following:

1. The user inputs a text file that should contain commands for the robot to execute
2. If the file can't be opened the program exits
3. If the file can be opened, the program goes on to initialize one instance of the `TableTop` class and one instance of the `Robot` class
4. The program then reads one line at a time from the provided text file
5. If the line doesn't start with a '#' it is passed to the instance of the `Robot` class, along with a reference to the `TableTop`

The current structure does not implement any form of relationship between the `TableTop` class and the `Robot` class. Apart from the instance of the `TableTop` class being passed to the robot to allow for the robot to get feedback about any coordinates it might want to move to, if they are valid or not.

An alternative implementation that could make sense logically, is that, since the robot is moving around on the table, it could be seen as an instance on the table. And therefore the `TableTop` class could contain any instances of the `Robot` class.

Note: it wasn't mentioned in the description, but I decided to add functionality to add comments in the text files. This to make it possible to write what output is expected for a provided input file. The program will simply echo any line that starts with a '#'.

TableTop

This class is a very small class and only contains information about the size of the table and a function that determines if a coordinate is on the table or not. It's purpose is to prevent a table to be created with a negative size and for the robot to end up on any coordinates that are outside the boundaries of the table.

Robot

This class is where the majority of the functionality exists. The entry point for this class is the *performAction* function that takes a string (that should contain an argument), a reference to the `TableTop` class, and tries to execute the command contained in the string. Since there is quite a bit of code in this class, it won't all be explain in detail here. It better to read the documentation for each function more details about what each function does.

Coordinate

This file contains a definition of the `Point2D` struct which is intended to be used to group a coordinate pair, x and y, together. Along with functionality to convert a string to an integer.

Direction

This file contains a definition of the enum class `Direction` which is intended to be used to represent the direction of the robot. Having an enum to represent the direction makes for easier comparisons than by storing it as a string. It also contains some functions to rotate between different directions, convert a string to a direction and a direction to a string.

Compiling and running

To compile and run the program, C++11 and g++ is required.

To compile the program, run this from the root folder:

```
./compile_project.sh
```

If for some reason it isn't possible to run the beforementioned script, run this in a bash terminal:

```
chmod +x compile_project.sh
```

The program is available to run by executing the file *main.exe* from a command prompt. Since the test framework is integrated into the same main file as the program, one can execute both the tests and the program with the same file.

Running tests

```
# Running main.exe withouth any arguments will execute the test suites
main.exe

# Passing '--help' will show possible parameters to the test framework
main.exe --help
```

Running program

```
# To disable the tests, the flag '--no-run' should be passed.
# In addition, to run the program a path to a txt file is required
```

```
main.exe --no-run TestInputFiles/test_1.txt
```

Some additional test vectors can be found in the folder **TestInputFiles**.

Test framework

The test framework used in this small project is one called **doctest** and is available on github here: doctest on github.

It was chosen for the simple reason that it is a lightweight framework which requires minimal set up. The entirety of the framework is contained in the file *doctest.hpp* in the root folder.