Parking lot problem

Nature of the Game

We want to understand how you think as a programmer, and the level of craft you bring to bear when building software.

Of course, the ideal would be a real-world problem, with real scale, but that isn't practical as it would take too much time. So instead, we have a dead simple, high school level problem that we want you to solve *as though* it was a real-world problem.

Please note that not following the instructions below will result in an automated rejection. Taking longer than the time allocated will negatively affect our evaluation of your submission.

Rules of the Game

1. You have two full days to implement a solution.
2. We're interested in understanding how you make assumptions when building software. If a particular workflow or boundary condition is not defined in the problem statement below, what you do is your choice.
3. We're also interested in your object oriented or functional design skills.
4. You have to solve the problem in any object oriented or functional language **without using any external libraries** to the core language except for a testing library for TDD, if you're writing tests. Your solution **must** build+run on Linux/Mac. If you don't have access to a Linux/Mac dev machine, you can easily set one up using Docker.
5. Please use Git for version control. We expect you to send us a **standard zip or tarball** of your source code when you're done that includes Git metadata (the .git folder) in the tarball so we can look at your commit logs and understand how your solution evolved. Frequent commits are a huge plus.
6. Please **do not** check in binaries, class files, jars, libraries or output from the build process.
7. Please write unit tests/specs for additional points. For object-oriented solutions, it's a huge plus if you test drive your code.

8. Please create your solution inside the `parking_lot` directory. Your codebase should have the same level of structure and organization as any mature open source project including coding conventions, directory structure and build approach (make, gradle etc.) and a README.md with clear instructions.
9. For your submission to pass the automated tests, please update the Unix executable scripts `bin/setup` and `bin/parking_lot` in the bin directory of the project root**.** `bin/setup` should install dependencies and/or compile the code and then run your unit test suite. `bin/parking_lot` runs the program itself. It takes an input file as an argument and prints the output on STDOUT. Please see the examples below. Please note that these files are Unix executable files and should run on Unix.
10. Please do not make either your solution or this problem statement publicly available by, for example, using GitHub or Bitbucket or by posting this problem to a blog or forum.

Problem Statement

I own a parking lot that can hold up to 'n' cars at any given point in time. Each slot is given a number starting at 1 increasing with increasing distance from the entry point in steps of one. I want to create an automated ticketing system that allows my customers to use my parking lot without human intervention.

When a car enters my parking lot, we scan the license plate of the car and get a six-character alphanumeric code, (ex. 514KZE) to uniquely identify the car (ID). I want to have a ticket issued to the driver. The ticket issuing process includes us documenting the ID and the color of the car and allocating an available parking slot to the car before actually handing over a ticket to the driver (we assume that our customers are nice enough to always park in the slots allocated to them). The customer should be allocated a parking slot which is nearest to the entry. At the exit the customer returns the ticket which then marks the slot they were using as being available.

Due to government regulation, the system should provide me with the ability to find

out:

- IDs of all cars of a particular color.
- Slot number in which a car with a given ID is parked.
- Slot numbers of all slots where a car of a particular color is parked.

We interact with the system via a simple set of commands which produce a specific output. Please take a look at the example below, which includes all the commands you need to support - they're self-explanatory. The system should allow input in two ways. Just to clarify, the same codebase should support both modes of input - we don't want two distinct submissions.

1) It should provide us with an interactive command prompt-based shell where commands can be typed in

2) It should accept a filename as a parameter at the command prompt and read the commands from that file

Example: File

To install all dependencies, compile and run tests:

```
$ bin/setup
```

To run the code so it accepts input from a file:

```
$ bin/parking_lot file_inputs.txt
```

Input (contents of file):

```
create_parking_lot 6
park EUS687 White
park 510IBD White
park 6TRJ24 Black
park EK3333 Red
park IYTE32 Blue
park MNG728 Black
leave 4
status
park AU7367 White
park 999AAA White
ids_for_cars_with_color White
slot_numbers_for_cars_with_color White
slot_number_for_id MNG728
slot_number_for_id 045BKR
```

Output (to STDOUT):

```
Created a parking lot with 6 slots
Allocated slot number: 1
Allocated slot number: 2
Allocated slot number: 3
Allocated slot number: 4
Allocated slot number: 5
Allocated slot number: 6
Slot number 4 is free
Slot No.  ID          Color
1         EUS687      White
2         510IBD      White
3         6TRJ24      Black
5         IYTE32      Blue
6         MNG728      Black
Allocated slot number: 4
Sorry, parking lot is full
EUS687, 510IBD, AU7367
1, 2, 4
6
Not found
```

Example: Interactive

To install all dependencies, compile and run tests:

```
$ bin/setup
```

To run the program and launch the shell:

```
$ bin/parking_lot
```

Assuming a parking lot with 6 slots, the following commands should be run in sequence by typing them in at a prompt and should produce output as described below the command. Note that `exit` terminates the process and returns control to the shell.

```
$ create_parking_lot 6
Created a parking lot with 6 slots

$ park EUS687 White
Allocated slot number: 1

$ park 510IBD White
Allocated slot number: 2
```

```
$ park 6TRJ24 Black
Allocated slot number: 3

$ park EK3333 Red
Allocated slot number: 4

$ park IYTE32 Blue
Allocated slot number: 5

$ park MNG728 Black
Allocated slot number: 6

$ leave 4
Slot number 4 is free

$ status
Slot No.  ID           Color
1         EUS687       White
2         510IBD       White
3         6TRJ24       Black
5         IYTE32       Blue
6         MNG728       Black


$ park AU7367 White
Allocated slot number: 4

$ park 999AAA White
Sorry, parking lot is full

$ ids_for_cars_with_color White
EUS687, 510IBD, AU7367

$ slot_numbers_for_cars_with_color White
1, 2, 4

$ slot_number_for_id MNG728
6

$ slot_number_for_id 045BKR
Not found

$ exit
```