Nature of the Game

We want to understand how you think as a software engineer, 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. Generate solution from ChatGPT or github auto-pilot or any AI platforms are prohibited.

3. We are really, really interested in your object oriented or functional design skills, so please craft the most beautiful code you can.

4. We're also 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.

5. You must 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. Your solution **must** build+run on Linux. If you don't have access to a Linux dev machine, you can easily set one up using Docker.

6. 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.

7. Please **do not** check in binaries, class files, jars, libraries or output from the build process.

8. Please write comprehensive unit tests/specs. For object-oriented solutions, it's a huge plus if you test drive your code. Submitting your solution with only a

functional suite will result in a rejection.

9. Please create your solution inside the `warehouse_rack` 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.

10. For your submission to pass the automated tests, please update the Unix executable scripts `bin/setup` and `bin/warehouse_rack` 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/warehouse_rack` 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.

11. Please ensure that you follow the syntax and formatting of both the input and output samples. The zip file you have been sent includes the same automated functional test suite we use. This is to help you validate the correctness of your program. You can run it by invoking `bin/run_functional_tests`. **IMPORTANT:** To make the functional specs work correctly, some setup is needed. Instructions to set up the functional suite can be found under `functional_spec/README.md`.

12. 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 warehouse rack that can hold up to 'n' products 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 racking system that allows my warehouse operator to use my warehouse rack without human intervention.

When a product checked-in to my warehouse rack, I want to have a notification issued to the operator. The notification issuing process includes us documenting the stock keeping unit (SKU) number, product expired date and allocating an available slot to the product before sending a notification to the operator (we assume that our operators are nice enough to always put in the slots allocated to respected product). The product should be allocated a nearest rack slot. When operator checked-out a product which then marks the slot they were using as being available.

Due to warehouse regulations, the system should provide me with the ability to find out:

- SKU numbers of all products of a particular expired date.

- Slot number in which a product with a given SKU number is racked.

- Slot numbers of all slots where a product of a particular expired date is racked.

We interact with the system via a simple set of commands which produce a specific output. Please 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/warehouse_rack file_inputs.txt
```

Input (contents of file):

```
create_warehouse_rack 6
rack ZG11AQA 2024-02-28
rack SD92349WW 2024-02-28
rack ZG748WDG 2024-03-15
rack KA887YHJ 2024-01-21
rack KA888YHH 2024-04-01
rack DG8789YH 2024-03-15
rack_out 4
status
rack DL654ASA 2024-02-28
rack DO123UJU 2024-02-28
sku_numbers_for_product_with_exp_date 2024-02-28
slot_numbers_for_product_with_exp_date 2024-02-28
slot_number_for_sku_number DG8789YH
slot_number_for_sku_number AB8875WF
```

Output (to STDOUT):

```
Created a warehouse rack 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.  SKU No.         Exp Date
1         ZG11AQA         2024-02-28
2         SD92349WW       2024-02-28
3         ZG748WDG        2024-03-15
5         KA888YHH        2024-04-01
6         DG8789YH        2024-03-15
Allocated slot number: 4
Sorry, rack is full
ZG11AQA, SD92349WW, DL654ASA
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/warehouse_rack
```

Assuming a warehouse rack 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_rack 6
Created a warehouse rack with 6 slots

$ rack ZG11AQA 2024-02-28
Allocated slot number: 1


$ rack SD92349WW 2024-02-28
Allocated slot number: 2

$ rack ZG748WDG 2024-03-15
```

```
Allocated slot number: 3

$ rack KA887YHJ 2024-01-21
Allocated slot number: 4

$ rack KA888YHH 2024-04-01
Allocated slot number: 5

$ rack DG8789YH 2024-03-15
Allocated slot number: 6

$ rack_out 4
Slot number 4 is free

$ status
Slot No.  SKU No.         Exp Date
1         ZG11AQA         2024-02-28
2         SD92349WW       2024-02-28
3         ZG748WDG        2024-03-15
5         KA888YHH        2024-04-01
6         DG8789YH        2024-03-15


$ rack DL654ASA 2024-02-28
Allocated slot number: 4

$ rack DO123UJU 2024-02-28
Sorry, rack is full

$ sku_numbers_for_product_with_exp_date 2024-02-28
ZG11AQA, SD92349WW, DL654ASA

$ slot_numbers_for_product_with_exp_date 2024-02-28
1, 2, 4

$ slot_number_for_sku_number DG8789YH
6

$ slot_number_for_sku_number AB8875WF
Not found

$ exit
```