Backend Engineer Take Home Test

Create a program that gets taxi fares in Golang

Specification

(i) Overview

1. The base fare is 400 yen for up to 1 km.
2. Up to 10 km, 40 yen is added every 400 meters.
3. Over 10km, 40 yen is added every 350 meters.

This taxi is equipped with the following two meters. Only one of the most recent real values is recorded on these meters.

- Distance Meter
- Fare Meter

(ii) Input Format

Distance meter records are sent line by line for standard input in the following format.

00:00:00.000 0.0

00:01:00.123 480.9

00:02:00.125 1141.2

00:03:00.100 1800.8

The specifications of the distance meter are as follows.

- Space-separated first half is elapsed time (Max 99:99:99.999), second half is mileage.(the unit is meters, Max: 99999999.9)
- It keeps only the latest values.
- It calculates and creates output of the mileage per minute, but an error of less than 1 second occurs.

(iii) Error Definition

Error occurs under the following conditions.

- Not under the format of, hh:mm:ss.fff<SPACE>xxxxxxxx.f<LF>, but under an improper format.
- Blank Line
- When the past time has been sent.
- The interval between records is more than 5 minutes apart.
- When there are less than two lines of data.
- When the total mileage is 0.0m.

(iv) Output

Display the current fare as an integer on the fare meter (standard output). 12345

Standard output displays nothing for incorrect inputs that do not meet specifications, the exit code ends with a value other than 0.

What to submit

Submission needs to meet the following conditions for grading.

1. Be able to run on Debian Linux.
2. The files are zipped together into one file.
3. For programming languages that require compilation, include a set of README with build instructions along with the source code.
4. Include test code.
5. Logs should be output in appropriate timing and places.

For programming languages that require compilation, make a set of README that summarizes how to build with source code.

Hiring Criteria

- Is the core function test written?
- Is the test coverage at 70%?
- Are error cases taken into account as well as normal cases?
- Is there a log for each important part and is monitoring taken into account?
- Are logs written in important parts?
- Are logs structured in JSON or other format to facilitate later investigation?
- Based on the abstraction or documentation, is maintainability taken into account to facilitate the additional functions?
- Is the selection of Node.js libraries reasonable?
- Are computational costs taken into account?
- Is the time zone taken into account?
- Is the execution environment taken into account enough to run anywhere?
- Does it take the general directory structure of projects?
- Basic knowledge of classes
- Naming of classes (e.g. whether he/she uses properly camel case or snake case)
- Property naming
- Naming of methods (The ability to name object orientation and things properly)
- Appropriate comments for later readers of the code
- 抽象化、ドキュメンテーションから機能追加が容易になるメンテナンス性は考慮されているか ／ is maintainability considered, which make it easy to add function from abstaction and documentation?