Service Engineer Coding Exercise

To introduce PrestoQ to your coding style and technique, we'd like you to do a small coding exercise. This is not a *coding test* where we give you a score or run it through a suite of tests for correctness.

The purpose of this exercise is for you to be able to show us some code you wrote so we can discuss it and the decisions you made: like the discussions that typically happen during a code review or pull request. Our hope is that this discussion will provide both of us with an understanding of your skills, talents and what it'll be like to work at PrestoQ. We believe this better reflects what engineers do than asking you to code common patterns on a white board.

Requirements

Author a program that parses a flat file that contains product information into a list of **Product** objects.

- The input file format is defined in the section below
- Do not use any flat file format parsing libraries
    - You may use native or third party basic string utility functions like trim, pad, format, etc.
    - We know that you'd probably leverage an existing library if you needed to parse a flat file for a product but this is an exercise.
- Use one of the following modern, statically typed languages: Java, C#, Kotlin, F#
- Use a GitHub repository for source control
    - Email the link to the GitHub repository when you are finished
    - The readme.md file should include (at a minimum)
        - Instructions for running the project
        - Link to the latest Continuous Integration build/release
        - Any other information you would normally include when you author a readme
    - This code is your intellectual property
        - License it however you'd like (MIT, Apache or Public Domain are great choices)
        - Ensure the license you choose allows PrestoQ employees to read, build, and execute the code for free
- Set up a continuous integration environment using a free, publicly accessible service
    - We recommend AppVeyor (https://www.appveyor.com/)
    - Configure the CI environment to build directly from your GitHub project when there's a commit to master
    - Ensure that the GitHub readme file is automatically updated with the result of the CI build (AppVeyor has an easy badge that you can embed in your readme.md)

Input File Format

An example file is included in this GitHub repository.

The file is in an ASCII-encoded file flat file (fixed width) format with the following layout:

Start	End [Inclusive]	Name	Type
1	8	Product ID	Number
10	68	Product Description	String

70	77	Regular Singular Price	Currency
79	86	Promotional Singular Price	Currency
88	95	Regular Split Price	Currency
97	104	Promotional Split Price	Currency
106	113	Regular For X	Number
115	122	Promotional For X	Number
124	132	Flags	Flags
134	142	Product Size	String

Types

- **Number -** an integer value 8-digits long, zero left-padded
- **String** - ASCII encoded string, space right-padded
- **Currency** - US dollar value, where last two digits represent cents. The leading zero will be replaced with a dash if the value is negative
- **Flag** - Y/N

Pricing information

- Prices can either be a singular price per unit (e.g. $1.00) or a split price (e.g. 2 for $0.99). Only one price per pricing level will exist. The other will be all 0's, which indicates there is no price.
- If a price is split pricing, the Calculator Price is **Split Price / For X**
- You can be guaranteed that the input file will follow these rules – consider it a contract that the producer will always abide by. No error checking is required.

Flags

- If Flag #3 is set, this is a per-weight item
- If Flag #5 is set, the item is taxable. Tax rate is always 7.775%

Output File Requirements

A UTF-8 encoded, JSON formatted file with an array of Product objects that contain the following properties:

- Product ID
- Product Description
- Regular Display Price (English-readable string)
- Regular Calculator Price (price the calculator should use, rounded to 4 decimal places, half-down)
- Promotional Display Price, if it exists
- Promotional Calculator Price, if it exists
- Unit of Measure ("Each" or "Pound"). Weighted items are per pound
- Product size
- Tax rate

What We're Looking For

Remember, the purpose of this exercise is to have a conversation about your code that is typical of the conversations you'd have at PrestoQ during a pull request. PrestoQ values clean, SOLID, maintainable, testable, self-descriptive code with unit tests.

If you have any questions, please don't hesitate to email ben@prestoq.com