IN3043 Functional Programming
Resit coursework

This resit coursework is due at **4pm on Friday 11th August 2023**. As with all modules, this deadline is hard. (See "Submission" below for the procedure for applying for extensions.)

The aim of this coursework is to allow you to demonstrate effective use of the Haskell language, containers and higher-order functions to construct solutions that are correct, clear and concise. This task does not require any material presented after session 6, and I do not expect to see it in your solutions.

The task

You are to write a number of functions that perform queries for an order system.

Type definitions

We will be dealing with customers and products, described by the type synonyms

```
type Customer = String
type Product = String
```

An order of some quantity of a product by a customer is described by the type containing a customer name, product name and a number of these products:

```
data Order = Order Customer Product Int
    deriving (Show)
```

You may assume that the number ordered is greater than zero. Lists of orders may contain multiple orders by the same customer of the same product (not necessarily adjacent to each other).

The prices of products are given by a value of type

```
type PriceList = Map Product Double
```

You may assume that the number (the price of the product) is greater than zero.

Functions to be written

You are asked to implement four functions operating on these types. They are assigned equal marks, but you will find that later ones are more difficult than earlier ones.

The functions to be defined are:

```
customers :: [Order] -> [Customer]
```

a list (without repetitions) of all the customers that have made orders.

`bill :: [Order] -> PriceList -> [(Customer, Double)]`

a list of customers who ordered products in the price list, together with the total value of the products they ordered.

`unavailable :: [Order] -> PriceList -> [(Customer, [Product])]`

a list of customers who ordered products that were not in the price list, together with a list of those products (without repetitions).

`bill_discount :: [Order] -> PriceList -> [(Customer, Double)]`

like **`bill`**, but applying a "buy one, get one free" discounting policy, i.e. if a customer orders 4 of a given product, they pay for 2; if they order 5, they pay for 3.

For each part, you should add a brief comment explaining the strategy you have used, *not* a translation of the code into English.

There are many different ways of implementing these functions, so I am not expecting to see identical or strongly similar answers.

File organization

I have supplied a skeletal module `Billing.hs`. You must not change the type definitions or function declarations in that file (I will rely on them for automatic testing), but you will want to modify the equations currently defining each function as **`undefined`** to your own definitions. You can add whatever else you like.

Marking criteria

As noted above, each function is weighted equally. Within each part, a correct solution will achieve at least 60% of the mark, regardless of style. To earn a mark in the first class range (70% or more), a solution should make effective use of the language and containers to achieve clarity and avoid unnecessary repetition and complexity.

Submission

You should submit your version of the file `Billing.hs` via Moodle by the due date.

If you believe that you have extenuating circumstances that justify an extension, you should

- submit your work to `smcse-extension@city.ac.uk` within one week of the deadline (i.e. by 4pm Friday 18th August), including your student number, course, module code and the original due date, **and**
- apply for an extension through the Extenuating Circumstances process.

As not all claims for extenuating circumstances meet the criteria, you can also submit on Moodle what you have by the deadline, as a fallback in case your claim is rejected.

Caution

You must not discuss the details of this coursework with other students, nor can you ever share your code (in any form) with another student. Any cases of suspected copying or sharing of answers will be treated as academic misconduct. Note that giving your code to another student is also academic misconduct.