IN3043 Functional Programming
Coursework (resit)

There is a single coursework in this module, counting for 30% of the overall module mark. This coursework is due at **5pm on Sunday 4th December**. 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 inventory 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

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

You may assume that the quantity 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).

A delivery to the supplier of some quantity of a product is described by the type

```
data Delivery = Delivery Product Double
    deriving (Show)
```

Again you may assume that the quantity is greater than zero. Lists of deliveries may include multiple deliveries of the same product (not necessarily adjacent to each other).

Functions to be written

You are asked to implement five 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:

```
numProducts :: [Order] -> [(Customer, Int)]
```

a list of all customers who submitted an order, with the number of different products each of them ordered.

```
productQuantities :: [Order] -> [(Product, Double)]
```

a list of all products that have been ordered, with the total quantity of each.

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

the list of customers and products for which the customer has ordered more than half the total quantity for that product.

```
shortfall :: [Order] -> [Delivery] -> [(Product, Double)]
```

a list of products for which the total quantity ordered exceeds the total quantity delivered, with the difference in quantity.

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 `Orders.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 replace the equations 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 `Orders.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 5pm Sunday 11th December), 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.

Help and feedback

Please ask general questions about the coursework on the Moodle discussion board. Queries by email will be answered on that board, so that everyone gets the same information.

We can view and discuss draft solutions in the weekly lab session. You can also use my drop-in hours or make an appointment by email to see me at other times.

Sample solutions will be published on Moodle after the deadline. Marks and comments on your code will be returned via Moodle.

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.