

Week 6 Practical Class Exercises

Supermarket Docket Example

Objectives

This week's practical exercises are based on the *Supermarket Docket* program discussed in lectures last week. You are to complete the definition of several of the functions in the `Docket` module. We will also add a `Main` module to complete the program, managing the input of a sequence of barcodes and the output of the finished docket.

Exercise 1 (Docket Module)

Several functions in the `Docket` module are yet to be defined. Test each of them individually before continuing.

1. Given a number of cents, say 1023, the dollars and cents parts are respectively given by `1023 `div` 100` and `1023 `rem` 100`. Using this fact, and the `show` function, define a function of type:

```
formatCents :: Price -> String
```

so that, for example:

```
formatCents 1023 ==> "10.23"
```

and

```
formatCents 205 ==> "2.05".
```

2. Using the `formatCents` function, define a function to format one line of a bill:

```
formatLine  :: (Name,Price) -> String
```

such that, for example,

```
formatLine ("Rice, 5kg" , 499) ==>
    "Rice, 5kg.....4.99\n"
```

Recall that `'\n'` is the newline character, that the list append operator (`++`) can be used to join two strings together (since a string is just a list of characters), and that the function `length` returns the length of a string. There is a function `replicate` in the standard prelude which will build a list of a specified length by repeating an element:

```
replicate 4 'a' ==> "aaaa"
```

3. Using the `formatLine` function, define:

```
formatLines :: [(Name,Price)] -> String
```

which applies `formatLine` to each `(Name,Price)` pair and joins the results together.

That should be enough to format the main body of the bill. All we have left to do is include the total and the heading.

4. Define a function:

```
makeTotal :: BillType -> Price
```

which takes a list of `(Name,Price)` pairs and returns the total of the prices. For instance,

```
makeTotal [("...",449),("...",213)] ==> 662
```

Exercise 2 (Input-Output)

To complete the Supermarket Docket *program*, we need to include a `Main` module, with a `main` function to perform the IO interactions with the user.

In the final installed system, the sequence of barcodes would come from a barcode scanner. To simulate that process in this exercise, the sequence of barcodes are to be typed in at the keyboard, one barcode per line. The sequence is to be terminated by a blank line. An example interaction is as follows:

```
Enter barcodes, one per line.
```

```
Hit <return> to terminate.
```

```
1112
```

```
1111
```

```
3216
```

```
5823
```

```
1234
```

```
4567
```

```
Alonzo's Mega-Mart
```

```
Mars Bar.....1.60
```

```
Tim Tams.....3.79
```

```
Ayam Chili Sauce.....3.50
```

```
Ice Cream.....5.79
```

```
Chianti, 1lt.....17.95
```

```
Unknown Item.....0.00
```

```
Total.....$32.63
```

Hint: look at the lecture notes on IO, particularly the examples to read and sum a list of numbers entered at the keyboard.

You should test your program with `GHCi`. When it is working, compile your program with the command: `ghc --make Main.hs -o docket` (assuming your `Main` module is in `Main.hs`). This will produce an executable called `docket` which you can run by typing `./docket` at the shell prompt.