COMP105: Programming Paradigms
Lab Sheet 7

Filename: Lab7.hs

This is a challenge lab on higher order functions, which means that the questions in this lab are focused on building a useful program.

Restrictions. Like Lab 4, this lab comes with a set of restrictions that will be checked by Codegrade. This lab is focused on the use of higher-order functions, which means that you must **not** use recursion or list comprehensions to solve any of the problems. Some questions require you to use certain functions, and Codegrade will refuse to mark your code if you do not do so. Other than these restrictions, you can do whatever you like.

Transaction logs. In this lab sheet, you will implement functions that deal with the transaction history of a portfolio of stocks. Each transaction is represented by a tuple. For example:

```
('B', 100, 1104,  "VTI",  1)
```

The elements of this tuple are:

- The first element is a character that is either `'B'` or `'S'`. This represents whether the transaction was *buying* or *selling*.
- The second element is an integer representing how many *units* were bought or sold.
- The third element is an integer representing the *price per unit* that we bought or sold at.
- The fourth element represents the *stock* that we bought.
- The final element represents the *day* that the transaction took place on.

So our example transaction says that we bought 100 units of `VTI` on day 1, and we paid £1104 per unit. So the total amount that we spent was $100 \times £1104 = £110400$.

A transaction log is a list of transactions. For example:

```
test_log =
    [
        ('B', 100, 1104,  "VTI",  1),
        ('B', 200,   36, "ONEQ",  3),
```

```
        ('B',  50, 1223,  "VTI",  5),
        ('S', 150, 1240,  "VTI",  9),
        ('B', 100,  229, "IWRD", 10),
        ('S', 200,   32, "ONEQ", 11),
        ('S', 100,  210, "IWRD", 12)
    ]
```

This gives a list of transactions in the order that they were executed. Note that there is a mix of buy and sell transactions, and various stocks are bought and sold. Note also that at the end of the log we do not own any stocks, eg., we bought `100 + 50 = 150` units of `VTI`, and then sold `150` units before the end of the log. You can assume that this is the case for all transaction logs.

The questions in this lab sheet assume that you have the `test_log` shown above and the following type definition in your file.

```
type Transaction = (Char, Int, Int, String, Int)
```

Don't copy these from the PDF, because you will end up with indentation problems and parse errors. You can copy these definitions from `Code.hs`, which is available on Canvas.

Part A

In Part A, the goal is to build a program to report the transactions for a particular stock in a human-readable format. For example, for our test data set, the transactions on the stock `VTI` will be printed as:

```
Bought 100 units of VTI for 1104 pounds each on day 1
Bought 50 units of VTI for 1223 pounds each on day 5
Sold 150 units of VTI for 1240 pounds each on day 9
```

Question 1. Write a function `transaction_to_string :: Transaction -> String` that converts a *single* transaction in the format shown above. For example:

```
ghci> transaction_to_string ('B', 100, 1104,  "VTI",  1)
"Bought 100 units of VTI for 1104 pounds each on day 1"

ghci> transaction_to_string ('S', 200, 32, "ONEQ", 11)
"Sold 200 units of ONEQ for 32 pounds each on day 11"
```

The format is

```
[Action] [Units] units of [Stock] for [Price] pounds each on day [Day]
```

where `[Action]` is either `Bought` or `Sold`, and each other parameter is determined by the corresponding element of the transaction.

Hint: you will need to use pattern matching to break the tuple down into its individual elements, and remember that the `show` function can be used to turn any type into a string.

Question 2. Write a function `trade_report_list :: [Transaction] -> [String]` that takes a list of transactions, and converts each transaction into a string. Your solution must use the `map` function. For example:

```
ghci> trade_report_list (take 3 test_log)
["Bought 100 units of VTI for 1104 pounds each on day 1","Bought 200 units of
ONEQ for 36 pounds each on day 3","Bought 50 units of VTI for 1223 pounds each
on day 5"]
```

Here we are calling `trade_report_list` on `take 3 test_log`, where `test_log` is the log shown earlier. You should copy `test_log` into your file from `Code.hs` (which can be found on Canvas).

Question 3. Write a function `stock_test :: String -> Transaction -> Bool` that takes a stock and a transaction log, and returns `True` if the transaction trades that stock, and `False` otherwise. For example:

```
ghci> stock_test "VTI"  ('B', 100, 1104,  "VTI",  1)
True

ghci> stock_test "ONEQ" ('B', 100, 1104,  "VTI",  1)
False
```

Question 4. Write a function `get_trades :: String -> [Transaction] -> [Transaction]` that takes a stock and a transaction log, and returns all trades in the transaction log that trade the given stock. Your solution must use the `filter` function. For example:

```
ghci> get_trades "VTI" test_log
[('B',100,1104,"VTI",1),('B',50,1223,"VTI",5),('S',150,1240,"VTI",9)]

ghci> get_trades "ONEQ" test_log
[('B',200,36,"ONEQ",3),('S',200,32,"ONEQ",11)]
```

Question 5. Write a function `trade_report :: String -> [Transaction] -> String` that takes a stock and a transaction log, and returns a string containing the human-readable version of the log. For example:

```
ghci> trade_report "VTI" test_log
"Bought 100 units of VTI for 1104 pounds each on day 1\nBought 50 units of VTI for 1223
pounds each on day 5\nSold 150 units of VTI for 1240 pounds each on day 9\n"
```

Remember that `\n` is the *new line* character, which causes a new line to be generated when your string is printed. You can test your function by printing the string with the `putStr` function like so:

```
ghci> putStr (trade_report "ONEQ" test_log)

Bought 200 units of ONEQ for 36 pounds each on day 3
Sold 200 units of ONEQ for 32 pounds each on day 11
```

Hint: remember the `unlines` function that we saw in the lectures.

Part B

In Part B, the goal is to build a function to tell the user how much *profit* or *loss* was made on each stock. For our test data set, this report will be

```
VTI: 14450
ONEQ: -800
IWRD: -1900
```

which indicates that we made `14450` pounds from our trades on `VTI`, we lost `800` pounds from our trades on `ONEQ`, and we lost `1900` pounds from our trades on `IWRD`. Remember that

- the amount of money we spend when we buy is `units` $\times$ `price`, and
- the amount of money we gain when we sell is also `units` $\times$ `price`.

So to work out the profit or loss from a particular stock, we add up the amount of money we gained by selling, and then we subtract the money that we spent when we bought. The calculation for `VTI` is

$$-(100 \times 1104) - (50 \times 1223) + (150 \times 1240) = 14450$$

Question 6 Write a function `update_money :: Transaction -> Int -> Int` that takes a transaction and the current amount of money that you have, and returns the amount of money that you have after the transaction. So if the transaction buys a stock the amount of money you have will decrease, and if the transaction sells a stock the amount of money you have will increase. For example:

```
ghci> update_money ('B',  1, 10,  "VTI",  5) 100
90

ghci> update_money ('S',  2, 10,  "VTI",  5) 100
120
```

In the first example we spent 10 pounds, while in the second example we gained 20 pounds.

Question 7 Write a function `profit :: [Transaction] -> String -> Int` that takes a transaction log and the name of a stock, and returns the total amount of profit or loss made for that stock. Your solution must use the `foldr` function. For example:

```
ghci> profit test_log "VTI"
14450

ghci> profit test_log "ONEQ"
-800
```

Question 8 Write a function `profit_report :: [String] -> [Transaction] -> String` that takes a list of stocks and a transaction log, and returns the human-readable string containing the profit and loss report. Specifically, for each stock in the input list, the report should include the line

`STOCK: PROFIT`

where `STOCK` is the name of the stock, and `PROFIT` is the amount of profit made. The stocks should appear in the order in which they are listed in the input. For example:

```
ghci> profit_report ["VTI", "ONEQ"] test_log
"VTI: 14450\nONEQ: -800\n"

ghci> profit_report ["VTI"] test_log
"VTI: 14450\n"
```

Once again, you can test your function using `putStr` like so:

```
ghci> putStr (profit_report ["VTI", "ONEQ", "IWRD"] test_log)

VTI: 14450
ONEQ: -800
IWRD: -1900
```

(*) Part C

In Part C the objective is to again produce a profit report, in the same format as Part B, but this time the input is given in a less convenient format. The trade log will be given as a plain text string, such as:

```
BUY 100 VTI 1
BUY 200 ONEQ 3
BUY 50 VTI 5
SELL 150 VTI 9
BUY 100 IWRD 10
SELL 200 ONEQ 11
SELL 100 IWRD 12
```

Each line has the following format

- The first word is either `BUY` or `SELL`.
- The second word is an integer giving the number of units that were bought or sold.
- The third word is the stock.
- The fourth word is the day on which the trade took place.

You may assume that there is exactly one space between each word. Each line ends with the newline character `\n`.

Note that the prices do not appear in the transactions. These are instead given as a price database of type `[(String, [Int])]`. For example:

```
[
    ("VTI", [1689, 1785, 1772, 1765, 1739, 1725, 1615, 1683, 1655, 1725,
             1703, 1726, 1725, 1742, 1707, 1688, 1697, 1688, 1675]),
    ("ONEQ", [201, 203, 199, 199, 193, 189, 189, 183, 185, 190, 186,
              182, 186, 182, 182, 186, 183, 179, 178]),
    ("IWRD", [207, 211, 213, 221, 221, 222, 221, 218, 226, 234, 229,
              229, 228, 222, 218, 223, 222, 218, 214])
]
```

Each element of the list is a tuple. The first element of the tuple gives the name of the stock. The second element of the tuple gives a list of prices for that stock. The first element of the list is the price on day one, the second element is the price on day two, and so on. So the price of `VTI` on day two is `1785`. You may assume that each day that is used in the transaction log has a corresponding price in the price database.

You can find the examples above as `test_str_log` and `test_prices` in `Code.hs` on Canvas. You should copy those into your file for testing purposes.

Question 9. Write a function `complex_profit_report :: String -> Prices -> String` that takes a transaction log and a price database, and returns a profit and loss report in the same format as used in Question 8. The report should contain one line for each stock in the price database in the order in which they are listed. For example:

```
ghci> complex_profit_report test_str_log test_prices
"VTI: -7600\nONEQ: -2600\nIWRD: -500\n"
```