COMP105 Assignment 2: Higher order functions

Assessment Number	2 (of 4)
Weighting	25%
Assignment Date Circulated	Wednesday the 10th of November (week 7)
Deadline	Wednesday the 24th of November (week 9) at 12:00 midday
Submission Mode	Electronic only
Learning outcome assessed	• Apply common functional programming idioms such as map, filter, and fold. • Write programs using a functional programming language.
Marking criteria	Correctness: 100%
Submission necessary in order to satisfy Module requirements?	No
Late Submission Penalty	UoL standard penalty applies

Important: Please read all of the instructions carefully before starting the assignment.

The assignment. In this assignment you will implement various functions that deal with the transaction history of stock portfolio, with a focus on applying higher-order programming techniques. In part A, you will implement functions using `map` and `filter`. In part B, you will implement a function using `foldr`. If a function in part A or part B tells you to use a function, then will need to use that function in order to obtain full marks. Otherwise, you are **free to use whatever techniques or functions that you like.**

Parts A and B combined are worth 65% of the marks for the assignment, which is equivalent to a 2:1 classification. Part C should be attempted by ambitious students who want to push for 100%. There are no restrictions at all for the code that you write in part C. Use whatever technique you see fit.

The template file. There is a template file called `Assignment2.hs`, which is available for download from the Canvas course, and your solutions must be written into that file. The template file also includes some test data, which will be explained later. As usual, **do not modify any of the type annotations in the template file.**

Marking. Marks will be awarded for correct function implementations only. No marks will be gained or lost for coding style. If you have a non-working function that you would like to be looked at by a human marker, then include the following comment above the function.

```
-- PLEASE READ
```

Please make sure that the comment is *exactly* as shown above including capitalization. Functions that do not include this comment will be marked by the automated marker only. If you have multiple non-working functions that you would like to be looked at, then please include the comment above each one. Do not include the comment above working functions – no extra feedback will be given for those other than what is produced by the automated marker.

Submission. Submit `"Assignment2.hs"` (please do not change the filename) to the assessment task submission system:

`https://sam.csc.liv.ac.uk/COMP/Submissions.pl`

Since your functions will be called from the automated marker, please **make sure that your code compiles without errors before submitting.** There will be a 5% penalty for any submission that fails to compile, or for any submission where the type annotations have been modified. You can use the checker file (see the section at the end of this document) to check whether your submission will get this penalty.

If you have some code that does not compile, but you would like the marker to see it, then comment out the code, explain in another comment what the code is supposed to do, and include the `PLEASE READ` comment following the instructions above. Remember that you will need to reinstate the stub implementation after you comment out your code, or the file will still fail to compile.

Academic Integrity. The university takes academic integrity very seriously. The work that you submit for this assessment must be your own. You should not *collude* with another student, meaning that you should not see another COMP105 student's submission, or copy any part of it. You should also not *plagiarise* anyone else's work, meaning that code written by someone else that you find online, or anywhere else, should not be submitted as part of the assignment.

Deadline. The deadline for this task is:

Wednesday the 24th of November (week 9) at 12:00 midday

The standard university late penalty will be applied: 5% of the *total marks available* for the assessment shall be deducted from the assessment mark for each 24 hour period after the submission deadline. Late submissions will no longer be accepted after five calendar days have passed.

Transaction Logs

In this assignment, 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:

```
[
    ('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.

For your convenience, the template file includes the type definition

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

which means that all transaction logs have the type `[Transaction]`. The example log given above appears as `test_log` in the template file.

Part A (worth 30%)

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.

Warning: be very careful about the formatting of your output for this question. If you get it wrong, then you will likely also lose marks in other Part A questions. The checker includes test cases for both buy and sell transactions, which you can use to verify your outputs.

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 should 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"]
```

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 should 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 (worth 35%)

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` × `price`, and
- the amount of money we gain when we sell is also `units` × `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 should 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
```

Warning: be careful about the formatting of your strings for this question. If you get it wrong, you may lose a substantial amount of marks even if your calculated profits are correct. You can use the checker to test if your output has the correct format.

Part C (worth 35%)

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 list of strings, 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`. For your convenience, the trade log above appears in the template file as `test_str_log`.

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 that is used in the transaction log has a corresponding price in the price database.

For convenience, the template includes the following type definition:

```
type Prices = [(String, [Int])]
```

The example price database given above appears in the template file as `test_prices`.

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"
```

The Checker

You can download `Checker2.hs` from the Canvas course. This file is intended to help you check whether your code will run successfully with the automated marker. To run the checker, follow these instructions.

1. Place `Checker2.hs` and your completed `Assignment2.hs` into the *same folder*. The checker will not work if it is not in the same folder as your submission.
2. You have two options to run the checker.

(a) Navigate to the folder in Windows explorer, and double-click on `Checker2.hs` to open the file into ghci.

(b) In a ghci instance, first use the `:cd` command to change to the directory containing the checker, and then run `:load Checker2.hs`

You **cannot** load the checker directly with something like `:l M:\Checker2.hs`, because you must first change the directory so that ghci can also find `Assignment2.hs`. Both of the methods above will do this.

If the checker compiles successfully, then you are guaranteed to not receive the 5% penalty for non-compilation. You can run the `compileCheck` in ghci to confirm that the checker has been compiled successfully.

You can also run the `test` function to automatically test your code using some simple test cases. Passing these test cases does not guarantee any marks in the assessment, since different test cases will be used to test your code after submission. You are encouraged to test your functions on a wide variety of inputs to ensure that they are correct.

FAQ

Can I use this library function?

Yes. Unlike Assignment 1, there are no functions that are prohibited. However, for Question 2 you are required to use `map`, for Question 4 you are required to use `filter`, and for Question 7 you are required to use `foldr`. Other than that, you may use whatever technique you see fit.

What should `test_log`, `test_str_log`, and `test_prices` be when I submit the file?

It doesn't matter what values you have chosen for these when you submit, as they will not be used by the marking program. These values are included in the file to help you test your functions, but your functions should work even if all of them are removed entirely from the file.

A common mistake is to hard code one of these values into your function. This will cause you to think that your function works if you only ever test it on the example inputs. So be careful, and make sure that you test your code on other logs (all parts) and price databases (in part C).

What assumptions can I make about the input in part C?

You can assume that the trade log follows the specification laid out in part C. So you can assume that each line contains exactly four words, that the first word is always `"BUY"` or `"SELL"`, and that the second and fourth words are both valid integers. You can also assume that there is exactly one space between each word in the trade log, and that there is no leading or trailing white space.

For the price database, you can assume that every stock named in the trade log also appears in the price database, and you can assume that every day mentioned in the trade log has a corresponding price listed in the price database.

If any of these assumptions are violated, then your function can do anything: crash, run forever, or give a nonsensical answer.

Can I create helper functions?

Yes. You can create any helper functions that you like. You will probably need to create some helper functions in part C, but you are also allowed to create them for parts A and B if you wish.

The `test` function in the checker reports that some of my functions are incorrect, throw exceptions, or timeout. Will I receive the penalty?

No. The 5% penalty is only for code that *doesn't compile* when the checker is loaded into ghci. If you can successfully load the checker into ghci, then you will not receive a penalty. You are likely to receive fewer marks for a question if your function doesn't work, however.

The checker doesn't compile with an error like `Could not find module 'Assignment2'`

There are three things to check:

- Are you following the instructions on how to run the checker? Note that just doing something like `:load M:\Checker2.hs` or similar **will not work**, because you need to change the directory first.
- Is your file name *exactly* `Assignment2.hs`? If it is misnamed, the compiler won't be able to find it.
- Did you alter or remove the `module Assignment2 ...` line at the top of the template? This line needs to be in place unaltered for the checker to load it in.

If you are still having difficulty, then send an email to get help.

My `Assignment2.hs` file compiles, but the checker does not compile and shows one of the assignment functions in the error. What should I do?

The most likely culprits are:

- You removed or altered the `module Assignment2 ...` line at the top of the template.
- You removed or altered one of the type annotations in the template file.

Make sure that these are all present. If they are, then your source file should also fail to compile, so you can fix that error. If you are still having difficulty, then send an email to get help.