Australian National University

# ASSIGNMENT COVER SHEET

This coversheet should be filled electronically when possible and attached as the front page of your report in PDF format.

**ANU College of Engineering and Computer Science**
Australian National University
Canberra ACT 0200 Australia
**www.anu.edu.au**

| | |
|---|---|
| Student ID | u6361796 |
| For group assignments, list each student's ID | |
| Course Code | comp1100 |
| Course Name | programming as problem solving |
| Assignment number | 2 |
| Assignment Topic | stock market |
| Lecturer | Katya Lebedeva |
| Tutor | Debashish Chakraborty |
| Tutorial (day and time) | Monday 11am-1pm |
| Word count | 2751 |
| Due Date | Friday Week 7 (September 22) |
| Date Submitted Friday | Week 7 (September 22) |
| Extension Granted | no |

I declare that this work:

- ✓ upholds the principles of academic integrity, as defined in the University Academic Misconduct Rules;
- ✓ is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the course outline and/or Wattle site;
- ✓ is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- ✓ gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- ✓ in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

**Signatures**

For group assignments, each student must sign.

Wenbo Du

# Report

U6361796
Wenbo Du

## Abstract

The goal of code implement is increase total wealth using various methods on a stock market. The following report will be divided into 3 parts: function explanations, results and conclusion. The first part focuses on the explanation of different function in Market.hs, Simulate.hs and Order.hs. And the argument of why makeOrder function is non-trivial will be included too. The second part will discuss the result I got. Conclusion will include the drawback of my algorithm and further possible improvement.

## Function explanations

Market.hs:

- Calculatewealth:

  Take Portfolio and [StockHistory] as input and output the total wealth you have. Taking Portfolio, the function exacts the "cash" in Portfolio. Taking [StockHistory], the function calculates the total current value of your stocks. This is achieved in function currentValue: this function gets a list that contain some cash. Each cash represents the total value of a single stock you have: The quantity of stocks product the latest price (the first element of a list of prices of the stock which can be got in the function getStockPrice). Sum built-in function add all single cash to get the total value of your stocks. After that, we can get the total value of stocks (remaining cash + total value of stocks).

- getStockPrice:

  The function gets the latest price for each stock. The first step is get filter certain stock. This can be achieved in function getStock. In a list of stockHistory, this function filters the elements (tuple of stock name the history price) by the given stock name. Once we get the tuple including stock name and its history price, we just exact the first element of history price causes that represents the current price of stocks.

- excuteOrder:
  The function takes 3 inputs: a tuple of cash and holdings, histories and orders. As defined, the type signature for the tuple is Double and a list of tuples including stock and quantity. The type signature for them is string and Integers. For histories, the type signature is a list of "StockHistories", and each element in the list is the tuple which includes Stock(string) and a list of quantity(integer). For orders, according to the file Type.hs, it is a list of "Order Stock Quantity ", and Order before Stock is just constructor. The function uses case for orders and guards expression. Furthermore, the function used where expression the compose some different function together. With different input, the goal is refresh and get a new Portfolio for traders. The situations can be divided into case and guards to discuss.

**Case 1**: if the traders do not make any order ([input orders is []):

(1) if the trader borrow cash in Portfolio (cash <0), trader's debt will increase because of the negative interest, which is calculated in the function "interest". This function takes two Double inputs (one is loanRate defined in Market.hs and the other is Cash defined in Type.hs) and output their product as interest for debt. Meanwhile, the status of stocks holding do not change.

(2) If the trader has some cash(otherwise), the interest rate change to CashRate, the interest will be traders' cash product CashRate. In the output, cash in Portfolio will change to cash with some interest. Again, stockholdings will no change.

**Case 2**: If the traders sell or buy stocks (input orders is "Order s q: xs"):

(1) the order will be skipped in the following conditions (the output Portfolio will be the same as the input one, which is represent in the function skipOrder):

<1> the stock name is "XJO".

<2>short selling is invalid(invalidShort).

That is, ordering negative value of stocks (q <0,in the function isSelling) , and satisfied the following conditions: cost, which is quantity of stocks ordering product the price of stock(in getStockPrice function) with commission fee (in the function commission: cost multiply ssCommssion) is high than current wealth. Current wealth is calculated in the function Calculatewealth.

<3>long selling is invalid. The process is similar as <2>.

(2) short selling condition 1(isShortingReg):
Function quantityHeld is used to calculate the quantity of certain stock in original Portfolio (guards expression): first, check if the stock ordered is in the stock holding lists (use notElem function to check whether the stock in input orders exists in the original Portfolio.) If exists, the function calculates the quantity of this stocks by using "filter" and "fst" to get the exact stock name, then use head map, snd functions to get the quantity. When trader orders negative value of stocks and the Portfolio do not have positive numbers of this certain of stock (isSelling<0 &&quantityHeld<=0, also the orders cannot satisfy the condition of (1)), both cash and stocks holding in Portfolio changes. The output of cash is the previous cash minus the total cost a of ordering (cost+ commission fee), and the output of stockholdings will add the stock ordered (in the function updateHoldings, and the default input of the function is (s,q) and holdings) . Then change the input "orders" to the next element of list orders (xs)to make a recursion.

(3) short selling condition 2(isShortingHeld):
This consider when trader want to do some short selling but he or she already have this stock (positive value). To do this, the trader first need to do some regular sell until he or she do not have this kind of stocks anymore. the function regularSellOrder sell all the stocks trader has (also use quantotyHeld function the calculate the total quantity of a certain stocks). Then use the function shortSellOrder to do short selling: for example, if the trader wants to sell 20 stock" A" and originally he has 5 "A", 5 of them will be seen as regular order, and 15 of them will be seen as short selling. The commission fee is different between 2 kinds of orders so they need to be treated separately. The output of this part refreshes the cash and holdings. Also, this function includes recursion to excute next order.

(4) Long-term selling:
If the order does not satisfy the situation above, it is treated as a long-term ordering. The cash is changed to cash minus total cost. Total cost include cost to selling stocks and commission fee. The cost here is negative value (means earn some money). Updateholdings and recursion is included too.

(5) Long-term buying(otherwise):
Another situation buying. The cost here is positive value so the only difference between (5) and (4) is plus/minus operation.

Simulate.hs:

This function composes by several functions:

- unfoldPrices:

the function takes a list of Price (type double) and output a list of lists contained price. If input is [1,2,3] from the use foldr, at first, the input of the lambda is 3 and [] (p=3, up= []) , from the case expression , up==[], so the output will be [[3]] ([p]=[3], [p]:[]=[[3]]).Then the input of the lambda function is [[3]] and 2(p=2, up=[[3]]). Because [[3]]/= [], the output will following the second case, (head [[3]] = [3] ,2: head [[3]] = [2,3], (2: head [[3]]):[[3]]=[[2,3],[3]]).When the input of lambda is 1 , accordingly, the output is [[1,2,3],[2,3],[3]]. On a certain day, the function input the day's price history, the output every previous day's price history and form a list from the latest day to the first (the latest days' price history is the head of the list, and the first day's history is the last element of the list).

- distribute:

the function takes Stock(string) and a nested list of prices (double) as input, and output a list of stockHistory. Using map function, the function can insert stock name to every element in a list and form a list of tuples (every element is a tuple, and its first element is stock name, second element is price history).

- convert:

It input a list, element of the list is tuple. The first element of tuple will be applied to the function distribute as the first argument. The second element of tuple we be applied to be the second argument. Then, it used the form (x: f xs) to build a recursion, so every element will be applied too.

- unfoldHistoriesAux:

The function makes a reclusive use of function UnfoldPrices: It take a list of stockhistory, for each element(tuple), apply function unfoldPrices to the second element of the tuple and form a list contain all days' stocks status (format (stock, [price historyDay1, price historyDay2...]). Again, use (x: f x) to make recursion and from a list contain different stocks available status.

- unfoldHistories:

The function takes a list of stockHistory (only one day, one element), and output all available stockHistory before(include) that day. For example, if the input is [("A", [1,2,3]), ("B", [3,4,5])][(Firstly, apply the function unfoldHistories to input, output is [("a",[[1.0,2.0,3.0],[2.0,3.0],[3.0]]),("b",[[3.0,4.0,5.0],[4.0,5.0],[5.0]])] . Then apply convert to [("a",[[1.0,2.0,3.0],[2.0,3.0],[3.0]]),("b",[[3.0,4.0,5.0],[4.0,5.0],[5.0]]) , the output is [[("a",[1.0,2.0,3.0]),("a",[2.0,3.0]),("a",[3.0])],[("b",[3.0,4.0,5.0]),("b",[4.0,5.0]),("b",[5.0])]]. Finally, apply built-in function transpose to above list. The output [[("a",[1.0,2.0,3.0]),("b",[3.0,4.0,5.0])],[("a",[2.0,3.0]),("b",[4.0,5.0])],[("a",[3.0]),("b",[5.0])]] . Now, in the nested, every element represents all tradeable stocks and their history price. The first element is the latest day and the last element is the first tradeable day.

- Simulate:

The function takes Portfolio and a list of Stockhistory as input, and output the final Portfolio. Foldl function is used to "separate every day" to analyse. Lambda function applies to every element (every day) in the list. Every day, with the days' all stocks historys, the excuteOrder function with filter the stocks the trader wants to buy (with makeOrder function) and determine whether the order is valid, then refresh the Portfolio after the trade. The process will continue until element in list (every day) is executed.

Order.hs:

This function includes 3 main part to consider: when to start order, how many stocks to order, what stock to buy or sell.

>When to start:

In the function timeObserve, it output 20 which means after 20 days the trader started to make order (when the length of price history is 20)

>what stock to buy or sell:

This part divide into 2 strategies.

(1) Cross

The first one is named "Cross" (Golden Cross and Death Cross): it is derived from technical analysis.

<1>

the function countAverage calculate the moving average (count the mean of price on every several days). It inputs a list x and y(Int), and output a list of a (a is a number and is fractional). y means how many days' moving average the function will calculate. The function expresses in guards. When the length of x is bigger than y, use "take" to get the first y element of x and count their average (sum /length). the use of

(*100) and (/100) is to get the round number on two decimal places, otherwise the function is too time-consuming. Use the from x:(f xs) to count the remaining price until the length of x is smaller than y, in this situation, just count the average of them all. As in the doctest, in the case of empty list, the countAverage should be [].

<2>

The function averageCross determine whether there is cross exist. There are 2 kinds of cross. One is golden cross; the shorter moving average forms a crossover up through the larger moving average. In technical analysis, this indicates a possible bull market. The other is death cross, the larger moving average forms a crossover up through a smaller moving average and this possibly indicate a bear market. Here, the function just compares two different moving average (5 days' and 20 days'), if the head of 5 days' moving average form is larger than 20 days' and the second element of 5 days' moving average is smaller or equal than 20 days', cross appears (golden cross or death cross). In function, if the import is [5,5,5,5,4,4.01], that indicates a Golden Cross, so the output is string "All Trade Sell".

(2) Round Numbers:

Round numbert tend to be important point since psychological importance." Buyers will often purchase large amounts of stock once the price starts to fall toward a major round number, which makes it more difficult for shares to fall below that level ("All trader purchase" in function allTradeOrder). On the other hand, sellers start to sell off a stock as it moves toward a round number peak, making it difficult to move past the upper level("All trader sell" in function allTradeOrder).( http://www.investopedia.com/university/technical/techanalysis4.asp#ixzz4tJRLw8Vg) Here, round number is Integer, and when the prices of stock is near round number(abs(price- nearest round number <= 0.1 in the function psychologicalRoundNumber) , the trader will order some stocks. Also, to make the round number is significance enough, I make a constraint the interval of every order for round number must be bigger than 10 days in the function nearest10Days. This time we need to consider empty list again, when the input of allTradeOrder is an empty list, I defined is as "No Influence".

>How many stock to order:

This part I used Sharpe Ratio to calculate the risk of certain stock. The larger Sharpe Ratio indicates safer investment. The function SharpeRatio is used the calculate a stocks' Sharpe Ratio.

dailyReturn:

it inputs price history(list) and output the change of daily price(list). For example, the stock "ABC "has stock history [1,1.5,2,2.5]. Its daily return will be [0.5,0.33,0.2].

returnedExpected:

It will the sum of daily return list. On the example above, the output is 1.02.

dailyFreeRate:

It is the daily interest (3%/365).

standardDev:

The function is used to calculate standard deviation. To get it, the first step is to count mean and this is indicated in the function mean." fromIntegral length(x)/1" convert type Int (length) to a(Fractional). In the function standardDev, for every element use map to minus mean, then map again to power every element. Eventually, sum all elements get their square and divide it by the square of length(x). Then it gets the standardard deviation of a list of history.

SharpeRatio:

The function uses function described above, and get the Sharpe Ratio of a list of prices. (format: sharpeRatio((returnExpected x)-dailyFreeRiskRate*convertedLength)/(standardDev x)).For example, if the input is [1.1,1.2,1.3] , the output should be 1.1103845096768414(in doctest) .

calculateRisk and investQuantity:

As mentioned, the larger Sharpe Ratio indicates lower risk. The function calculateRisk and investQuantity determine the quantity of stocks the invest according to different Sharpe Ratio.

makeOrder:

Compose all other functions in Order.hs and use case and guards to discuss different situations. First, consider when there is no available stock history. If not, use case to consider [stockHistory] ((s,p):xs).

According to different outputs through function allTraderOrder and averageCross, the function determines what stock to buy. The function inverstQuantity here is the calculate how many stocks to buy or sell with different risks.

## Results

During the process of writing code, there are some error, exception and warning I have encountered:

1> *** Exception: Prelude.head: empty list. That means I did not consider the condition that head built-in function is applied an empty list. Also, the similar message encounter when I tried the use minimum or maximum function to an empty list. To solve it, made a guard that count [] separately

2> This binding for `x' shadows the existing binding. This mean I used overlap symbol to express different argument. To fix it, change one of them to avoid collision.

3> Pattern match(es) are non-exhaustive. This warning indicated that I did not consider all possibility. Usually, I did not consider an empty list.

Determine better performance:

To make a better strategy. I covered some article on Investopedia, mainly technical analysis including simple moving average, golden cross and death cross. Furthermore, to calculate the risk, I used Sharpe Ratio which can be found on Investopedia too. The only thing I change according to my performance on scoreboard is the output of investQuantity: I change the output to get better performance on most scoreboard.

Function effectiveness:

Previously I did not use round built-in function at all. But I find without the use of round the process of counting moving average is too time-consuming. I used round and (*100) (/100) the get the round number with two decimal number.

## Conclusion

I function perform is good on scoreboard in long term (the simulation day is bigger than 1000). But when the simulation day is short, my function performance is not good. I think I can do something short selling since that is more profitable. By this assignment, I get a better understanding of programming use. All we learn is not for exam or other kinds of practice, the core of programming is real world application. Although basic practice is important, it is better to use Haskell solving real world problems. By solving real world problems, our programming skill can be improved faster.