CM2005 Object Oriented Programming

Coursework 1 Report

LIM WEE KIAT

Table of Contents

Introduction

Advisorbot is a command-line program that can perform various tasks to help cryptocurrency investors analyse the data available on the exchange. Advisorbot will respond to commands entered by the user. So that users can consider whether it is suitable for bid or ask. First and foremost, there is a set of commands that will be responded to by the advisorbot which are :

- **help**
- **help <cmd>**
- **prod**
- **min**
- **max**
- **avg**
- **predict**
- **time**
- **step**
- **printstats**

'help' command

The purpose of the 'help' command is to list all available commands. Users can use this command to understand which available commands that the advisorbot can respond to.

'help <cmd>' command

The purpose of the 'help <cmd>' command is to output the help for the specified command. Users can use the 'help avg' command to check the 'avg' command format in order to reduce errors.

'prod' command

The purpose of the 'prod' command is to list all the available products that are in the exchange data.

'min' command

The purpose of the 'min' command is to find the minimum bid or ask for the product in the current time step. The 'min' command format is 'min product bid/ask'. For example, if users want to find the minimum bid for BTC/USDT in the current time step, they can enter the command such as 'min BTC/USDT bid'.

‘**max’ command**

The purpose of the ‘max’ command is to find the maximum bid or ask for the product in the current time step. The ‘max’ command format is ‘max product bid/ask’. For example, if users want to find the maximum bid for ETH/BTC in the current time step, they can enter the command such as ‘max ETH/BTC bid’.

‘**avg’ command**

The purpose of the ‘avg’ command is to compute the average ask or bid for the product over the number of time steps. The ‘avg’ command format is ‘avg product bid/ask timesteps’. For example, if users want to compute the average ask for the BTC/USDT over 10 timesteps, they can enter the command such as ‘avg BTC/USDT ask 10’.

‘**predict’ command**

The purpose of the ‘predict’ command is to predict the max or min ask or bid for the product for the next time step. The ‘predict’ command format is ‘predict max/min product ask/bid’. For example, if users want to predict the max bid for the ETH/BTC for the next time step, they can enter the command such as ‘predict max ETH/BTC bid’.

‘**time’ command**

The purpose of the ‘time’ command is to state current time in dataset, which timeframe are we looking at.

‘**step’ command**

The purpose of the ‘step’ command is move to next time step.

‘**printstats’ command**

The purpose of the ‘printstats’ command is to show how many successful matching transaction in current time steps.

Table reporting

Commands	Achieved or Not
C1: help	Achieved
C2: help cmd	Achieved
C3: prod	Achieved
C4: min	Achieved
C5: max	Achieved
C6: avg	Achieved
C7: predict	Not
C8: time	Achieved
C9: step	Achieved
C10: printstats	Achieved

Command parsing code

For the command parsing code, I use the tokenise function to take the command string as input and split the elements of the command.

```
32  std::vector<std::string> CSVReader::tokenise(std::string csvLine, char separator) {
33      std::vector<std::string>tokens;
34      signed int start, end;
35      std::string token;
36      start = csvLine.find_first_not_of(separator, 0);
37      do {
38          end = csvLine.find_first_of(separator, start);
39          if (start == csvLine.length() || start == end) break;
40          if (end >= 0) token = csvLine.substr(start, end - start);
41          else token = csvLine.substr(start, csvLine.length() - start);
42          tokens.push_back(token);
43          start = end + 1;
44      } while (end > 0);
45
46      return tokens;
47  }
```

Figure 1.1 CSVReader.cpp (line 32 - 47)

As shown in figure 1.1, tokenise function have two arguments which is csvLine and separator. So that I can use this function to extracts the elements of command.

First and foremost, I need to create a vector of strings called tokens and call the tokenise function, and pass two arguments which are userOption and ‘ ’. This is because when users enter their command, I can split the command into tokens to calculate the size of the tokens and validate each token. After that, I used the if-else loop to separate based on the size of the tokens. If the size of the token is equal to 1, there is another if-else loop to identify the tokens.

```cpp
void MerkelMain::processUserOption(std::string userOption) {
    std::vector <std::string> tokens = CSVReader::tokenise(userOption, ' ');
    try {

        if (tokens.size() == 1)
        {
            if (tokens[0] == "help")
            {
                printHelp();
            }

            else if (tokens[0] == "prod")
            {
                printProducts();
            }

            else if (tokens[0] == "time")
            {
                printCurrentTime();
            }

            else if (tokens[0] == "step")
            {
                gotoNextStep();
            }
            else if (tokens[0] == "printstats")
            {
                printMatchStats();
            }

            else
            {
                std::cout << "Invalid command. Please enter help to choose available commands." << std::endl;
            }
        }
```

Figure 1.2 MerkelMain.cpp (line 248 - 282)

As shown in figure 1.2, I can identify whether each token is equal to the command string. If not, it will return an error message to the user. For example, when the user enters the ‘help’ command, the advisorbot will print the help menu to the user. In this case, we know that the ‘help/prod/time/step/printstats’ command just has only one token which is tokens[0] and it will call each function based on the specified command.

```cpp
//list all available command
void MerkelMain::printHelp()
{
    std::cout << "====================================";
    std::cout << "\n The available commands are: " << std::endl;
    std::cout << "====================================";
    std::cout << "\n >> help \n >> help <cmd> \n >> prod \n >> min \n >> max \n >> avg \n >> predict\n >> time \n >> step \n >> printstats" << std::endl;
    std::cout << "====================================" << std::endl;
}
```

Figure 1.3 MerkelMain.cpp (printHelp())

```cpp
//move to next time step
void MerkelMain::gotoNextStep()
{
    currentTime = orderBook.getNextTime(currentTime);
    std::cout << "Now at " << currentTime << std::endl;
}
```

Figure 1.4 MerkelMain.cpp (gotoNextStep())

```
218     //shows how many successful matching transaction
219     void MerkelMain::printMatchStats() {
220         for (std::string p : orderBook.getKnownProducts())
221         {
222             std::cout << "==============================" << std::endl;
223             std::cout << "Matching " << p << std::endl;
224             std::vector<OrderBookEntry> sales = orderBook.matchAskToBids(p, currentTime);
225             std::cout << "Total sales: " << sales.size() << std::endl;
226             for (OrderBookEntry& sale : sales)
227             {
228                 std::cout << "Sale price: " << sale.price << " Amount " << sale.amount << std::endl;
229                 //std::cout << "==============================" << std::endl << std::endl;
230             }
231 
232             std::cout << "==============================" << std::endl;
233         }
234     }
```

Figure 1.5 MerkelMain.cpp (printMatchStats())

```
96      //list available products
97      void MerkelMain::printProducts()
98      {
99          int i = 1;
100         std::cout << "============= " << std::endl;
101         std::cout << "  Products " << std::endl;
102         std::cout << "============= " << std::endl;
103         for (std::string const& p : orderBook.getKnownProducts()) {
104             std::cout << i<<". " << p << std::endl;
105             i++;
106         }
107     }
```

Figure 1.6 MerkelMain.cpp (printProducts())

```
236     //state current time in dataset,i.e which timeframe are we looking at
237     void MerkelMain::printCurrentTime() {
238         std::cout << "Current time is : " << currentTime << std::endl;
239     }
240 
```

Figure 1.7 MerkelMain.cpp (printCurrentTime())

Besides, if the size of the token is equal to 2, there is the if-else loop to identify the tokens. As we know, the tokens[0] is our command which is ‘help’, and the tokens[1] is the specific command which is the ‘avg/min/max/predict’ command. If the command is invalid, the advisor bot will return an error message. So that, users can use help <avg/min/max/predict> to execute the command that they need.

```
284         else if (tokens.size() == 2)
285         {
286             if (tokens[0] == "help")
287             {
288                 processHelpCommand(userOption);
289             }
290 
291             else
292             {
293                 std::cout << "Invalid command. Please enter help to choose available commands." << std::endl;
294             }
295         }
```

Figure 1.8 MerkelMain.cpp (line 284 - 295)

As shown in figure 1.8, I can identify whether the tokens[0] is equal to the ‘help’ command. If not, it will return an error message to the user. If the tokens[0] is equal to ‘help’, it will call the processHelpCommand function.

```
42  //output help for the specific command
43  void MerkelMain::processHelpCommand(std::string userOption) {
44      std::vector <std::string> tokens = CSVReader::tokenise(userOption, ' ');
45      try
46      {
47          if (tokens[1] == "min")
48          {
49              std::cout << "================" << std::endl;
50              std::cout << "  min command" << std::endl;
51              std::cout << "================" << std::endl;
52              std::cout << "Command: >> min product bid/ask" << std::endl;
53              std::cout << "Example: >> min ETH/BTC ask " << std::endl;
54              std::cout << "Output : -> The min ask for ETH/BTC is 1.0" << std::endl;
55          }
56 
57          else if (tokens[1] == "max")
58          {
59              std::cout << "================" << std::endl;
60              std::cout << "  max command" << std::endl;
61              std::cout << "================" << std::endl;
62              std::cout << "Command: >> max product bid/ask" << std::endl;
63              std::cout << "Example: >> max ETH/BTC ask " << std::endl;
64              std::cout << "Output : -> The max ask for ETH/BTC is 1.0" << std::endl;
65          }
66 
67          else if (tokens[1] == "avg")
68          {
69              std::cout << "================" << std::endl;
70              std::cout << "  avg command" << std::endl;
71              std::cout << "================" << std::endl;
72              std::cout << "Command: >> avg product bid/ask timesteps" << std::endl;
73              std::cout << "Example: >> avg ETH/BTC ask 10 " << std::endl;
74              std::cout << "Output : -> The average ETH/BTC ask price over the last 10 timesteps was 1.0" << std::endl;
75          }
76 
77          else if (tokens[1] == "predict")
78          {
79              std::cout << "================" << std::endl;
80              std::cout << " predict command" << std::endl;
81              std::cout << "================" << std::endl;
82              std::cout << "Command: >> predict max/min product bid/ask" << std::endl;
83              std::cout << "Example: >> predict max ETH/BTC bid " << std::endl;
84              std::cout << "Output : -> The average ETH/BTC ask price over the last 10 timesteps was 1.0" << std::endl;
85          }
86 
87          else
88          {
89              std::cout << "Invalid command. Please enter help to choose available commands." << std::endl;
90          }
91      }catch (const  std::exception& e){
92          std::cout << "MerkelMain::processHelpCommand Error.Please check again." << std::endl;
93      }
94  }
```

Figure 1.9 MerkelMain.cpp (line 43 - 94)

As shown in figure 1.9, I need to validate the tokens[1] by using if-else loop. If the tokens[1] is invalid, it will return an error message. Inside the function, I used the exception handling(try and catch) to prevent errors. For example, users enter ‘help avg’ command, the advisorbot will print out all the message about the avg command.

Furthermore, if the size of the token is equal to 3, there is the if-else loop to identify the tokens. If the tokens[0] is equal to min or max, it will call the function. If tokens[0] is invalid, it will return an error message as shown in line 297 and 313 in figure 2.0.

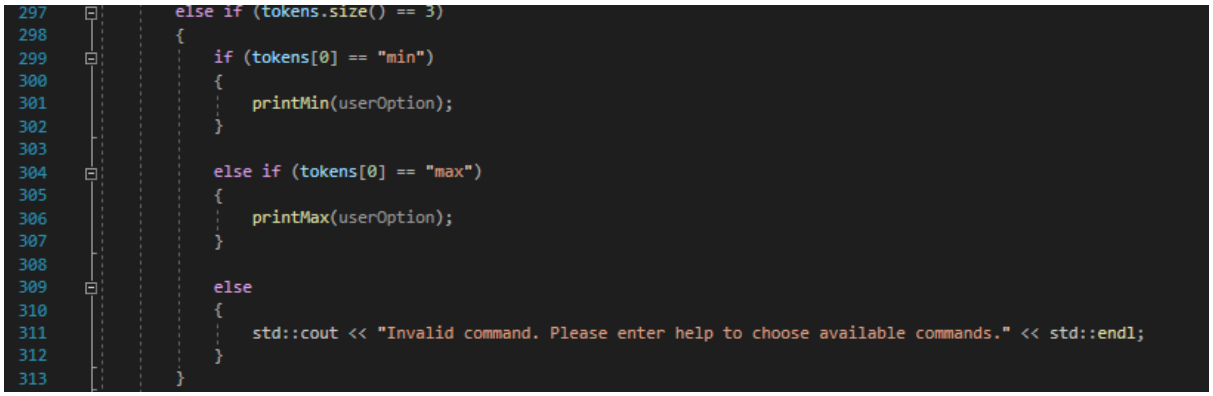


Figure 2.0 MerkelMain.cpp (line 297 - 313)

As we know, the tokens[0] is our command which is ‘min/max’, the tokens[1] is the products, and the tokens[2] is bid or ask. After we validate the tokens[0] is equal to ‘min’, I need to validate the tokens[2] by using if-else loop.If the tokens[2] is invalid, it will return error message. If the tokens[2] is equal to ‘ask/bid’, I created a vector and call the getOrder function to retrieve the tokens[1] data (products), and use the getLowPrice function to return a minimum ask/bid price for the products as shown in line 109 and 138 in figure 2.1.

Figure 2.1 MerkelMain.cpp (line 109 and 138)

On the other hand, if the tokens[0] is equal to ‘max’, it also need to validate the tokens[2] by using the if-else loop similar to the printMin function. If the tokens[2] is invalid, it will return error message. If the tokens[2] is equal to ‘ask/bid’, I created a vector and call the getOrder function to retrieve the tokens[1] data (products), and use the getHighPrice function to return a maximum ask/bid price for the products as shown in line 140 and 168 in figure 2.2.

```
140  //find maximum bid or ask for product in current time step
141  void MerkelMain::printMax(std::string userOption) {
142      // Tokens[0]:command, Tokens[1]:Product, Tokens[2]:Bid/Ask
143      std::vector <std::string> tokens = CSVReader::tokenise(userOption, ' ');
144
145      try
146      {
147          if (tokens[2] == "ask")
148          {
149              std::vector <OrderBookEntry> entries = orderBook.getOrders(OrderBookType::ask, tokens[1], currentTime);
150              std::cout << "The max " << tokens[2] << " for " << tokens[1] << " is " << orderBook.getHighPrice(entries) << std::endl;
151          }
152
153          else if (tokens[2] == "bid")
154          {
155              std::vector <OrderBookEntry> entries = orderBook.getOrders(OrderBookType::bid, tokens[1], currentTime);
156              std::cout << "The max " << tokens[2] << " for " << tokens[1] << " is " << orderBook.getHighPrice(entries) << std::endl;
157          }
158
159          else
160          {
161              std::cout << "Invalid command. Please enter help to choose available commands." << std::endl;
162          }
163      }
164      catch (const std::exception& e)
165      {
166          std::cout << "MerkelMain::printMax Error." << std::endl;
167      }
168  }
```

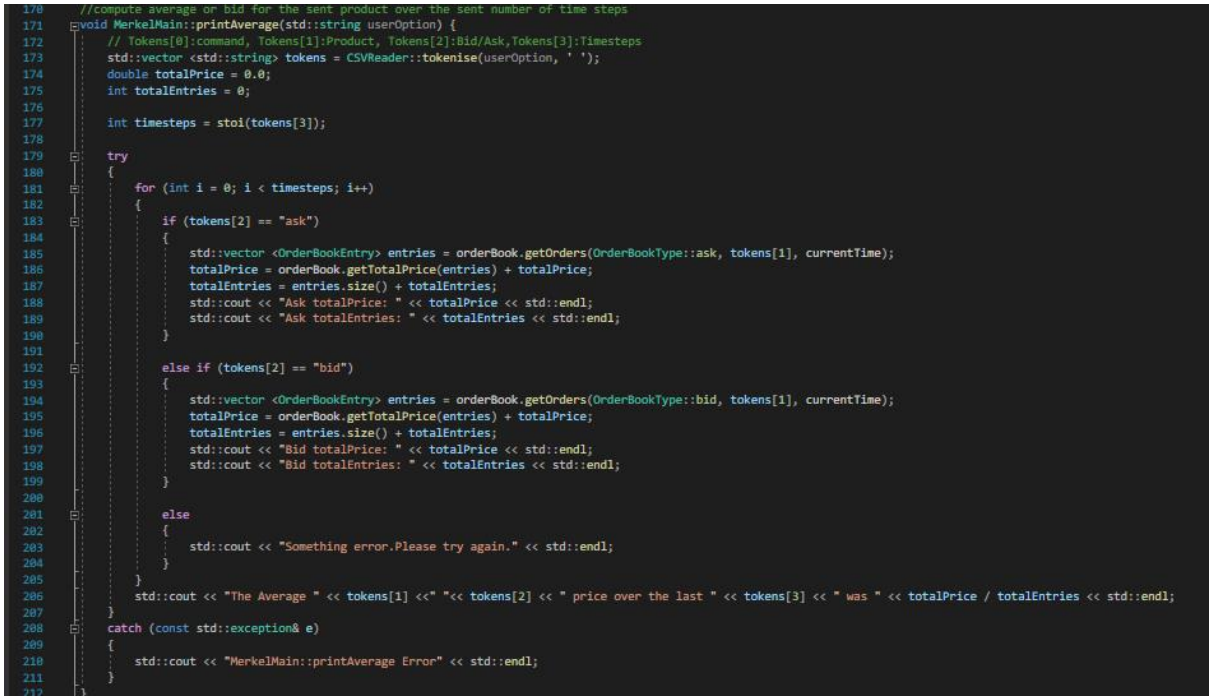
Figure 2.2 MerkelMain.cpp (line 140 and 168)

If the size of the token is equal to 4, there is the if-else loop to identify the tokens. So the first step of verification is to see if tokens[0] is equal to ‘avg’ or ‘predict’ command. If not, it will return an error message as shown in line 315 and 331 in figure 2.3.

```
315          else if (tokens.size() == 4)
316          {
317              if (tokens[0] == "avg")
318              {
319                  printAverage(userOption);
320              }
321
322              else if (tokens[0] == "predict")
323              {
324                  printPredict();
325              }
326
327              else
328              {
329                  std::cout << "Invalid command. Please enter help to choose available commands." << std::endl;
330              }
331          }
```

Figure 2.3 MerkelMain.cpp (line 140 and 168)

If the tokens[0] is equal to ‘avg’, I have create three variables which are totalPrice, totalEntries, and timestamps in the printAverage function. For the timestamps variable, I have converted the string to int type by using stoi(). After that, I created a for loop to calculate the average bid/ask price over the sent timesteps. Inside the for loop, I created the if-else loop to determine whether the tokens [2] is ask or bid. If the tokens[2] is invalid, it will return error message. Otherwise, it will call the getOrder function and do some calculation for the total price and total entries. After done the calculation, the advisorbot will return the average bid/ask price for the sent product over the sent number of the time steps as shown in line 171 and 212 in figure 2.4.



```
170   //compute average or bid for the sent product over the sent number of time steps
171   void MerkelMain::printAverage(std::string userOption) {
172       // Tokens[0]:command, Tokens[1]:Product, Tokens[2]:Bid/Ask,Tokens[3]:Timesteps
173       std::vector <std::string> tokens = CSVReader::tokenise(userOption, ' ');
174       double totalPrice = 0.0;
175       int totalEntries = 0;
176
177       int timesteps = stoi(tokens[3]);
178
179       try
180       {
181           for (int i = 0; i < timesteps; i++)
182           {
183               if (tokens[2] == "ask")
184               {
185                   std::vector <OrderBookEntry> entries = orderBook.getOrders(OrderBookType::ask, tokens[1], currentTime);
186                   totalPrice = orderBook.getTotalPrice(entries) + totalPrice;
187                   totalEntries = entries.size() + totalEntries;
188                   std::cout << "Ask totalPrice: " << totalPrice << std::endl;
189                   std::cout << "Ask totalEntries: " << totalEntries << std::endl;
190               }
191
192               else if (tokens[2] == "bid")
193               {
194                   std::vector <OrderBookEntry> entries = orderBook.getOrders(OrderBookType::bid, tokens[1], currentTime);
195                   totalPrice = orderBook.getTotalPrice(entries) + totalPrice;
196                   totalEntries = entries.size() + totalEntries;
197                   std::cout << "Bid totalPrice: " << totalPrice << std::endl;
198                   std::cout << "Bid totalEntries: " << totalEntries << std::endl;
199               }
200
201               else
202               {
203                   std::cout << "Something error.Please try again." << std::endl;
204               }
205           }
206           std::cout << "The Average " << tokens[1] <<" "<< tokens[2] << " price over the last " << tokens[3] << " was " << totalPrice / totalEntries << std::endl;
207       }
208       catch (const std::exception& e)
209       {
210           std::cout << "MerkelMain::printAverage Error" << std::endl;
211       }
212   }
```

Figure 2.4 MerkelMain.cpp (line 171 and 212)

Custom command(printstats)

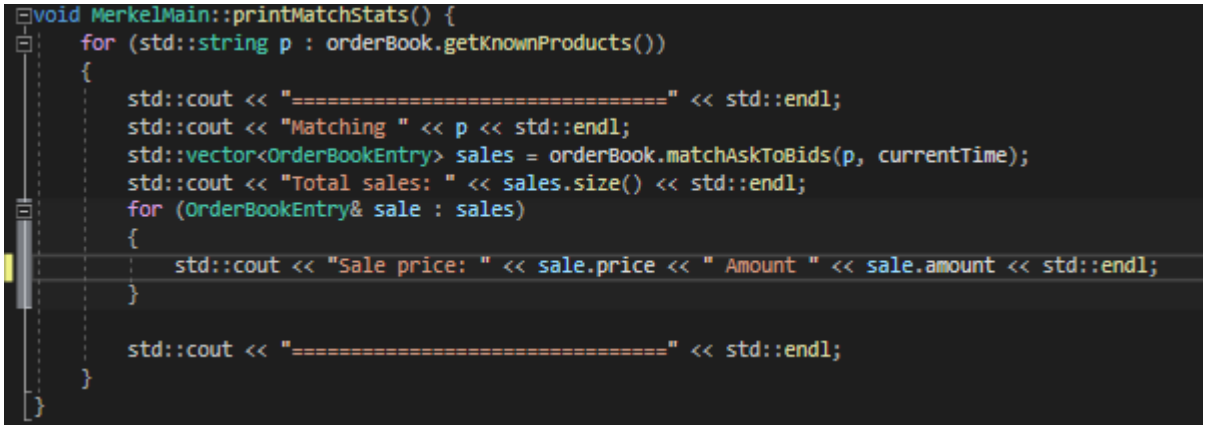
I have implemented the print stats command in the advisorbot. The print stats command is used to show how many successful matching transactions are in the current timestamp and it will display the minimum or maximum of all the products price for bid or ask.

First and foremost, I had created a function called 'matchAskToBids' inside the OrderBook.cpp and it's going to return a vector of order book entries. So I will use the order book entries to generate my sales. After that, I had created a vector of the asks and bids and pass the getOrder function that we have. I also have sorted the lowest of asks and the highest of bids. Besides, there has a nested loop to check the bid price and ask price. For example, if the bid price is greater than the ask price, it will run another if-else loop to check the bid amount is equal to or greater than or less than the ask amount as shown in line 94 and 139 in figure 2.5.

```
94  std::vector<OrderBookEntry> OrderBook::matchAskToBids(std::string product, std::string timestamp) {
95
96      std::vector<OrderBookEntry>asks = getOrders(OrderBookType::ask, product, timestamp);
97      std::vector<OrderBookEntry>bids = getOrders(OrderBookType::bid, product, timestamp);
98      std::vector<OrderBookEntry>sales;
99      std::sort(asks.begin(),asks.end(),OrderBookEntry::compareByPriceAsc);
100     std::sort(bids.begin(),bids.end(),OrderBookEntry::compareByPriceDesc);
101
102     std::cout << "max ask : " << asks[asks.size() - 1].price << std::endl;
103     std::cout << "min ask : " << asks[0].price << std::endl;
104     std::cout << "max bid : " << bids[0].price << std::endl;
105     std::cout << "min bid : " << bids[bids.size() - 1].price << std::endl;
106
107     for (OrderBookEntry& ask : asks) {
108         for (OrderBookEntry& bid : bids) {
109
110             if (bid.price >= ask.price) {
111                 std::cout << "bid price is right " << std::endl;
112                 OrderBookEntry sale{ask.price,0,timestamp,product,OrderBookType::sale};
113
114                 if (bid.amount = ask.amount) {
115                     sale.amount = ask.amount;
116                     sales.push_back(sale);
117                     bid.amount = 0;
118                     break;
119                 }
120
121                 if (bid.amount > ask.amount) {
122                     sale.amount = ask.amount;
123                     sales.push_back(sale);
124                     bid.amount = bid.amount - ask.amount;
125                     break;
126                 }
128                 if (bid.amount < ask.amount) {
129                     sale.amount = bid.amount;
130                     sales.push_back(sale);
131                     ask.amount = ask.amount - bid.amount;
132                     bid.amount = 0;
133                     continue;
134                 }
135             }
136         }
137     }
138     return sales;
139 }
```

Figure 2.5 OrderBook.cpp (line 94 and 139)

Furthermore, I had created the printMatchStats function in the MerkelMain.cpp. Inside the function, I had used the for loop to load all the products and pass the matchAskToBids function in order to print out each matching order of the products as shown in figure 2.6.



```
void MerkelMain::printMatchStats() {
    for (std::string p : orderBook.getKnownProducts())
    {
        std::cout << "==============================" << std::endl;
        std::cout << "Matching " << p << std::endl;
        std::vector<OrderBookEntry> sales = orderBook.matchAskToBids(p, currentTime);
        std::cout << "Total sales: " << sales.size() << std::endl;
        for (OrderBookEntry& sale : sales)
        {
            std::cout << "Sale price: " << sale.price << " Amount " << sale.amount << std::endl;
        }

        std::cout << "==============================" << std::endl;
    }
}
```

Figure 2.6 printMatchStats function

Optimise the exchange code

When we run the program, it needs a few times to read the exchange data and waste a lot of time. To optimise the exchange code, I have an idea is use the timestamp to reduce the processing time. Now the program is read the whole CSV file line by line and it will need a lot of processing time. So we can change the program to read the CSV file timestamps by timestamps. For example, we can load the first 20 timestamps in the advisorbot system, and continue read the other 20 timestamps when users enter the ‘step’ command. In this case, the exchange data can be run faster and didn’t affect user satisfication. But unfortunately, I can’t implement my idea into the program due to always getting errors. I think this idea is feasible, but my own coding knowledge is not enough, which led to this failure. I will try to keep learning and try again until I succeed.

Testing(Output)

Help command

```
=======================
Welcome to Advisorbot
=======================
Please enter a command, or help for a list of commands :help
====================================
 The available commands are:
====================================
 >> help
 >> help <cmd>
 >> prod
 >> min
 >> max
 >> avg
 >> predict
 >> time
 >> step
 >> printstats
====================================
```

Help avg command

```
=======================
Welcome to Advisorbot
=======================
Please enter a command, or help for a list of commands :help avg
=================
  avg command
=================
Command: >> avg product bid/ask timesteps
Example: >> avg ETH/BTC ask 10
Output : -> The average ETH/BTC ask price over the last 10 timesteps was 1.0
```

Help max command

```
=======================
Welcome to Advisorbot
=======================
Please enter a command, or help for a list of commands :help max
=================
  max command
=================
Command: >> max product bid/ask
Example: >> max ETH/BTC ask
Output : -> The max ask for ETH/BTC is 1.0
```

Help min command

```
=======================
Welcome to Advisorbot
=======================
Please enter a command, or help for a list of commands :help min
=================
  min command
=================
Command: >> min product bid/ask
Example: >> min ETH/BTC ask
Output : -> The min ask for ETH/BTC is 1.0
```

Help predict command

```
=======================
Welcome to Advisorbot
=======================
Please enter a command, or help for a list of commands :help predict
=================
 predict command
=================
Command: >> predict max/min product bid/ask
Example: >> predict max ETH/BTC bid
Output : -> The average ETH/BTC ask price over the last 10 timesteps was 1.0
-----------------------
```

Prod command

```
=======================
Welcome to Advisorbot
=======================
Please enter a command, or help for a list of commands :prod
=============
  Products
=============
1. BTC/USDT
2. DOGE/BTC
3. DOGE/USDT
4. ETH/BTC
5. ETH/USDT
```

Avg command

```
=======================
Welcome to Advisorbot
=======================
Please enter a command, or help for a list of commands :avg ETH/BTC ask 10
Ask totalPrice: 1.27837
Ask totalEntries: 50
Ask totalPrice: 2.55673
Ask totalEntries: 100
Ask totalPrice: 3.8351
Ask totalEntries: 150
Ask totalPrice: 5.11347
Ask totalEntries: 200
Ask totalPrice: 6.39183
Ask totalEntries: 250
Ask totalPrice: 7.6702
Ask totalEntries: 300
Ask totalPrice: 8.94857
Ask totalEntries: 350
Ask totalPrice: 10.2269
Ask totalEntries: 400
Ask totalPrice: 11.5053
Ask totalEntries: 450
Ask totalPrice: 12.7837
Ask totalEntries: 500
The Average ETH/BTC ask price over the last 10 was 0.0255673
```

Min command

```
=======================
Welcome to Advisorbot
=======================
Please enter a command, or help for a list of commands :max ETH/BTC bid
The max bid for ETH/BTC is 0.0248274
```

Max command

```
=======================
Welcome to Advisorbot
=======================
Please enter a command, or help for a list of commands :min ETH/BTC ask
The min ask for ETH/BTC is 0.0248467
```

Time command

```
========================
Welcome to Advisorbot
========================
Please enter a command, or help for a list of commands :time
Current time is : 2020/06/01 11:57:35.334211
```

Step command

```
========================
Welcome to Advisorbot
========================
Please enter a command, or help for a list of commands :step
Now at 2020/06/01 11:57:40.339453
```

Printstats command

```
Welcome to Advisorbot
======================
Please enter a command, or help for a list of commands :printstats
================================
Matching BTC/USDT
max ask : 9590.91
min ask : 9544.37
max bid : 9543.19
min bid : 9495
Total sales: 0
================================
================================
Matching DOGE/BTC
max ask : 7.6e-07
min ask : 2.7e-07
max bid : 2.6e-07
min bid : 1e-08
Total sales: 0
================================
================================
Matching DOGE/USDT
max ask : 0.00276
min ask : 0.00257726
max bid : 0.00255828
min bid : 0.002425
Total sales: 0
================================
================================
Matching ETH/BTC
max ask : 0.0252069
min ask : 0.0248467
max bid : 0.0248394
min bid : 0.02459
Total sales: 0
================================
================================
Matching ETH/USDT
max ask : 240.953
min ask : 237.293
max bid : 237.153
min bid : 233.842
Total sales: 0
================================
```