VE281 Project Four Report

Liu Yihao 515370910207

1 Appendix

1.1 The project files

1.1.1 Median.h

```
//
// Created by liu on 17-11-8.
//

#ifndef PROJECT_MEDIAN_H
#define PROJECT_MEDIAN_H

#include <queue>
#include <functional>

template<typename TYPE>
struct MedianAverage {
    std::plus<TYPE> plus = std::plus<TYPE>();
    std::divides<TYPE> divides = std::divides<TYPE>();
    const TYPE operator()(const TYPE &a, const TYPE &b) const {
        return divides(plus(a, b), TYPE(2));
    }
};

template<typename TYPE, typename AVERAGE = MedianAverage<TYPE>, typename COMP = std::less<TYPE>>
class Median {
public:
    typedef unsigned long size_type;
private:
    size_type size = 0;
    std::priority_queue<TYPE, std::vector<TYPE>, COMP> maxQueue;
    std::priority_queue<TYPE, std::vector<TYPE>, std::binary_negate<COMP> > minQueue;
    AVERAGE average;
public:
    explicit Median(AVERAGE average = AVERAGE(), COMP compare = COMP()) :
            average(average), maxQueue(compare),
            minQueue(std::binary_negate<COMP>(compare)) {}

```

```
33      void add(TYPE val) {
34          if (size % 2) {
35              if (val >= maxQueue.top()) {
36                  minQueue.push(std::move(val));
37              } else {
38                  TYPE temp = maxQueue.top();
39                  maxQueue.pop();
40                  minQueue.push(std::move(temp));
41                  maxQueue.push(std::move(val));
42              }
43          } else {
44              if (maxQueue.empty()) {
45                  maxQueue.push(std::move(val));
46              } else if (val <= minQueue.top()) {
47                  maxQueue.push(std::move(val));
48              } else {
49                  TYPE temp = minQueue.top();
50                  minQueue.pop();
51                  maxQueue.push(std::move(temp));
52                  minQueue.push(std::move(val));
53              }
54          }
55          ++size;
56      };
57
58      TYPE get() const {
59          if (size % 2) {
60              return maxQueue.top();
61          } else {
62              return average(maxQueue.top(), minQueue.top());
63          }
64      }
65
66      bool empty() const {
67          return size == 0;
68      }
69
70  };
71
72  #endif //PROJECT_MEDIAN_H
```

1.1.2 Stock.h

```
1   //
2   // Created by liu on 2017/9/16.
3   //
4
5   #ifndef PROJECT_STOCK_H
6   #define PROJECT_STOCK_H
7
```

```cpp
#include <string>
#include <queue>
#include <exception>
#include <set>
#include <unordered_map>
#include "Median.h"
#include "Client.h"

class Stock {
public:
    struct trade_t {
        Client *client;
        size_t timestamp, id, price, quantity;
        Stock *stock;
        bool isSell;
    };
private:
    struct trade_ptr_compare_buy {
        bool operator()(const trade_t *a, const trade_t *b) const {
            if (a->price == b->price) return a->id < b->id;
            return a->price > b->price;
        }
    };

    struct trade_ptr_compare_sell {
        bool operator()(const trade_t *a, const trade_t *b) const {
            if (a->price == b->price) return a->id < b->id;
            return a->price < b->price;
        }
    };

    std::string _name;

    std::set<trade_t *, trade_ptr_compare_buy> _buySet;
    std::set<trade_t *, trade_ptr_compare_sell> _sellSet;
    std::vector<trade_t *> _timeTraveler;

    Median<size_t> _median;

public:
    struct stock_ptr_compare {
        bool operator()(const Stock *a, const Stock *b) const {
            return a->_name < b->_name;
        }
    };

    explicit Stock(const std::string &name);

    ~Stock();
```

```
57
58      const std::string &name() const;
59
60      void addBuy(Client *client, size_t id, size_t price, size_t quantity, int
        ↪  expire, size_t timestamp, bool verbose);
61
62      void addSell(Client *client, size_t id, size_t price, size_t quantity, int
        ↪  expire, size_t timestamp, bool verbose);
63
64      bool matchBuy(trade_t *buy, bool verbose);
65
66      bool matchSell(trade_t *sell, bool verbose);
67
68      trade_t *getHighestBuy() const;
69
70      trade_t *getLowestSell() const;
71
72      void removeHighestBuy();
73
74      void removeLowestSell();
75
76      void removeExpiredTrade(trade_t *trade);
77
78      void printMedian(size_t timestamp) const;
79
80      void printMidPoint(size_t timestamp) const;
81
82      void printTimeTraveler() const;
83  };
84
85
86  #endif //PROJECT_STOCK_H
```

1.1.3 Stock.cpp

```
1   //
2   // Created by liu on 2017/9/16.
3   //
4
5   #include <iostream>
6   #include "Stock.h"
7   #include "Market.h"
8
9   Stock::Stock(const std::string &name) {
10      this->_name = name;
11  }
12
13  Stock::~Stock() {
14      for (auto &item : _timeTraveler) {
15          delete item;
```

```
16      }
17  }
18
19  const std::string &Stock::name() const {
20      return this->_name;
21  }
22
23  void
24  Stock::addBuy(Client *client, size_t id, size_t price, size_t quantity, int
    ↪  expire, size_t timestamp, bool verbose) {
25      auto trade = new trade_t{client, timestamp, id, price, quantity, this,
        ↪  false};
26      _timeTraveler.push_back(trade);
27      while (matchBuy(trade, verbose)) {}
28      if (expire != 0 && trade->quantity > 0) {
29          this->_buySet.insert(trade);
30          if (expire > 0) {
31              Market::getInstance().addExpiringTrade(trade, (size_t) expire);
32          }
33      }
34  }
35
36  void
37  Stock::addSell(Client *client, size_t id, size_t price, size_t quantity, int
    ↪  expire, size_t timestamp, bool verbose) {
38      auto trade = new trade_t{client, timestamp, id, price, quantity, this, true};
39      _timeTraveler.push_back(trade);
40      while (matchSell(trade, verbose)) {}
41      if (expire != 0 && trade->quantity > 0) {
42          this->_sellSet.insert(trade);
43          if (expire > 0) {
44              Market::getInstance().addExpiringTrade(trade, (size_t) expire);
45          }
46      }
47  }
48
49  bool Stock::matchBuy(trade_t *buy, bool verbose) {
50      if (buy->quantity == 0) return false;
51      auto sell = getLowestSell();
52      if (!sell) return false;
53      if (buy->price < sell->price) return false;
54      size_t price = sell->price;
55      size_t quantity;
56      if (buy->quantity > sell->quantity) {
57          quantity = sell->quantity;
58          buy->quantity -= sell->quantity;
59          this->removeLowestSell();
60      } else if (buy->quantity < sell->quantity) {
61          quantity = buy->quantity;
```

```
62          sell->quantity -= buy->quantity;
63          buy->quantity = 0;
64      } else {
65          quantity = sell->quantity;
66          buy->quantity = 0;
67          this->removeLowestSell();
68      }
69      _median.add(price);
70      buy->client->buy(quantity, price);
71      sell->client->sell(quantity, price);
72      Market::getInstance().trade(quantity, price);
73      if (verbose) {
74          std::cout << buy->client->name() << " purchased " << quantity << " shares
            ↪  of " << this->_name << " from "
75                    << sell->client->name() << " for $" << price << "/share" <<
                       ↪  std::endl;
76      }
77      return true;
78  }
79
80  bool Stock::matchSell(trade_t *sell, bool verbose) {
81      if (sell->quantity == 0) return false;
82      auto buy = getHighestBuy();
83      if (!buy) return false;
84      if (buy->price < sell->price) return false;
85      size_t price = buy->price;
86      size_t quantity;
87      if (buy->quantity > sell->quantity) {
88          quantity = sell->quantity;
89          buy->quantity -= sell->quantity;
90          sell->quantity = 0;
91      } else if (buy->quantity < sell->quantity) {
92          quantity = buy->quantity;
93          sell->quantity -= buy->quantity;
94          this->removeHighestBuy();
95      } else {
96          quantity = sell->quantity;
97          sell->quantity = 0;
98          this->removeHighestBuy();
99      }
100     _median.add(price);
101     buy->client->buy(quantity, price);
102     sell->client->sell(quantity, price);
103     Market::getInstance().trade(quantity, price);
104     if (verbose) {
105         std::cout << buy->client->name() << " purchased " << quantity << " shares
            ↪  of " << this->_name << " from "
106                   << sell->client->name() << " for $" << price << "/share" <<
                      ↪  std::endl;
```

```
107         }
108         return true;
109     }
110
111     Stock::trade_t *Stock::getHighestBuy() const {
112         if (_buySet.empty()) return nullptr;
113         return *(_buySet.begin());
114     }
115
116     Stock::trade_t *Stock::getLowestSell() const {
117         if (_sellSet.empty()) return nullptr;
118         return *(_sellSet.begin());
119     }
120
121     void Stock::removeHighestBuy() {
122         if (!_buySet.empty()) _buySet.erase(_buySet.begin());
123     }
124
125     void Stock::removeLowestSell() {
126         if (!_sellSet.empty()) _sellSet.erase(_sellSet.begin());
127     }
128
129     void Stock::removeExpiredTrade(trade_t *trade) {
130         if (trade->isSell) {
131             _sellSet.erase(trade);
132         } else {
133             _buySet.erase(trade);
134         }
135     }
136
137     void Stock::printMedian(size_t timestamp) const {
138         if (_median.empty()) return;
139         std::cout << "Median match price of " << _name << " at time " << timestamp <<
            ↪  " is $" << _median.get() << std::endl;
140     }
141
142     void Stock::printMidPoint(size_t timestamp) const {
143         std::cout << "Midpoint of " << _name << " at time " << timestamp << " is ";
144         auto buy = getHighestBuy();
145         auto sell = getLowestSell();
146         if (buy && sell) {
147             std::cout << "$" << (buy->price + sell->price) / 2;
148         } else {
149             std::cout << "undefined";
150         }
151         std::cout << std::endl;
152     }
153
154     void Stock::printTimeTraveler() const {
```

```
155     int sellPrice = -1, sellTime = -1, buyTime = -1, profit = 0;
156     int minSell = -1, minSellTime = -1;
157     for (const auto &trade : _timeTraveler) {
158         if (trade->isSell) {
159             if (minSell < 0 || minSell > trade->price) {
160                 minSell = (int) trade->price;
161                 minSellTime = (int) trade->timestamp;
162             }
163         } else {
164             if (minSell < 0) continue;
165             int diff = (int) trade->price - minSell;
166             if (sellPrice == -1 || diff > profit) {
167                 sellPrice = minSell;
168                 sellTime = minSellTime;
169                 buyTime = (int) trade->timestamp;
170                 profit = diff;
171             }
172         }
173     }
174     // variable name error, won't fix
175     std::cout << "Time travelers would buy " << _name << " at time: " << sellTime
176               << " and sell it at time: " << buyTime << std::endl;
177 }
```

1.1.4 Client.h

```
1  //
2  // Created by liu on 17-9-17.
3  //
4
5  #ifndef PROJECT_CLIENT_H
6  #define PROJECT_CLIENT_H
7
8  #include <string>
9
10 class Client
11 {
12 private:
13     std::string _name;
14     int _transfer = 0;
15     int _buyAmount = 0, _sellAmount = 0;
16 public:
17     struct client_ptr_compare {
18         bool operator()(const Client *a, const Client *b) const {
19             return a->_name < b->_name;
20         }
21     };
22
23     explicit Client(const std::string &name);
24
```

```
25      const std::string &name() const;
26
27      void buy(size_t quantity, size_t price);
28
29      void sell(size_t quantity, size_t price);
30
31      void printTransfer() const;
32  };
33
34
35  #endif //PROJECT_CLIENT_H
```

1.1.5 Client.cpp

```
1   //
2   // Created by liu on 17-9-17.
3   //
4
5   #include <iostream>
6   #include "Client.h"
7
8   Client::Client(const std::string &name) {
9       this->_name = name;
10  }
11
12  const std::string &Client::name() const {
13      return this->_name;
14  }
15
16  void Client::buy(size_t quantity, size_t price) {
17      _buyAmount += quantity;
18      _transfer -= quantity * price;
19  }
20
21  void Client::sell(size_t quantity, size_t price) {
22      _sellAmount += quantity;
23      _transfer += quantity * price;
24  }
25
26  void Client::printTransfer() const {
27      std::cout << _name << " bought " << _buyAmount << " and sold "
28                << _sellAmount << " for a net transfer of $"
29                << _transfer << std::endl;
30  }
```

1.1.6 Market.h

```
1   //
2   // Created by liu on 2017/9/16.
3   //
```

```
4
5   #ifndef PROJECT_MARKET_H
6   #define PROJECT_MARKET_H
7
8   #include <map>
9   #include <string>
10  #include <sstream>
11  #include "Stock.h"
12  #include "Client.h"
13
14  class Market {
15  private:
16      std::set<Stock *, Stock::stock_ptr_compare> stocks;
17      std::unordered_map<std::string, Stock *> stocksMap;
18
19      std::set<Client *, Client::client_ptr_compare> clients;
20      std::unordered_map<std::string, Client *> clientsMap;
21
22      std::multimap<size_t, Stock::trade_t *> _expireMap;
23      std::stringstream ss;
24      size_t timestamp = 0, tradeNum = 0;
25      size_t commission = 0, transferMoney = 0;
26      size_t tradeComplete = 0, shareTrade = 0;
27      bool verbose = false;
28      bool median = false;
29      bool midpoint = false;
30      bool transfers = false;
31      std::vector<std::string> timeTravelers;
32      Market() = default;
33  public:
34      ~Market();
35
36      static Market &getInstance();
37
38      void initOptions(bool verbose, bool median, bool midpoint, bool transfers);
39
40      void initTimeTraveler(std::string name);
41
42      Client *getClient(const std::string &name);
43
44      Stock *getStock(const std::string &name);
45
46      void readLine(const std::string &line);
47
48      void trade(size_t quantity, size_t price);
49
50      void addExpiringTrade(Stock::trade_t *trade, size_t expire);
51
52      void removeExpiredTrade(size_t timestamp);
```

```
53
54      void printTickSummary() const;
55
56      void nextTick(size_t newTimestamp);
57
58      void printDaySummary();
59  };
60
61
62  #endif //PROJECT_MARKET_H
```

1.1.7 Market.cpp

```
1   //
2   // Created by liu on 2017/9/16.
3   //
4
5   #include <iostream>
6   #include "Market.h"
7
8   Market::~Market() {
9       for (auto &stock:this->stocks) {
10          delete stock;
11      }
12      for (auto &client:this->clients) {
13          delete client;
14      }
15  }
16
17  Market &Market::getInstance() {
18      static Market market;
19      return market;
20  }
21
22  void Market::initOptions(bool verbose, bool median, bool midpoint, bool
    ↪  transfers) {
23      this->verbose = verbose;
24      this->median = median;
25      this->midpoint = midpoint;
26      this->transfers = transfers;
27  }
28
29  void Market::initTimeTraveler(std::string name) {
30      timeTravelers.push_back(std::move(name));
31  }
32
33  Client *Market::getClient(const std::string &name) {
34      auto it = this->clientsMap.find(name);
35      if (it == this->clientsMap.end()) {
36          auto client = new Client(name);
```

```
37          this->clients.insert(client);
38          it = this->clientsMap.insert({name, client}).first;
39      }
40      return it->second;
41  }
42
43  Stock *Market::getStock(const std::string &name) {
44      auto it = this->stocksMap.find(name);
45      if (it == this->stocksMap.end()) {
46          auto stock = new Stock(name);
47          this->stocks.insert(stock);
48          it = this->stocksMap.insert({name, stock}).first;
49      }
50      return it->second;
51  }
52
53  void Market::readLine(const std::string &line) {
54      this->ss.clear();
55      this->ss.str(line);
56      size_t timestamp, price, quantity;
57      int duration;
58      std::string clientName, action, stockName, priceStr, quantityStr;
59      ss >> timestamp >> clientName >> action >> stockName >> priceStr >>
        ↪  quantityStr >> duration;
60
61      if (timestamp > this->timestamp) nextTick(timestamp);
62
63      price = strtoul(priceStr.c_str() + 1, NULL, 10);
64      quantity = strtoul(quantityStr.c_str() + 1, NULL, 10);
65
66      auto client = this->getClient(clientName);
67      auto stock = this->getStock(stockName);
68
69      /**
70       * The expire time of stock
71       * -1 : forever
72       * 0  : instant
73       * >0 : expire time
74       */
75      int expire = duration > 0 ? (int) (timestamp + duration) : duration;
76      if (action == "SELL") {
77          stock->addSell(client, tradeNum++, price, quantity, expire,
            ↪  this->timestamp, verbose);
78      } else {
79          stock->addBuy(client, tradeNum++, price, quantity, expire,
            ↪  this->timestamp, verbose);
80      }
81  }
82
```

```
83  void Market::trade(size_t quantity, size_t price) {
84      ++tradeComplete;
85      shareTrade += quantity;
86      auto temp = quantity * price;
87      transferMoney += temp;
88      commission += (temp / 100) * 2;
89  }
90
91  void Market::addExpiringTrade(Stock::trade_t *trade, size_t expire) {
92      this->_expireMap.emplace(expire, trade);
93  }
94
95  void Market::removeExpiredTrade(size_t timestamp) {
96      auto end = _expireMap.upper_bound(timestamp);
97      for (auto it = _expireMap.begin(); it != end; ++it) {
98          auto trade = it->second;
99          trade->stock->removeExpiredTrade(trade);
100     }
101     _expireMap.erase(_expireMap.begin(), end);
102 }
103
104 void Market::printTickSummary() const {
105     if (median) {
106         for (auto &stock:this->stocks) {
107             stock->printMedian(this->timestamp);
108         }
109     }
110     if (midpoint) {
111         for (auto &stock:this->stocks) {
112             stock->printMidPoint(this->timestamp);
113         }
114     }
115 }
116
117 void Market::nextTick(size_t newTimestamp) {
118     printTickSummary();
119     removeExpiredTrade(newTimestamp);
120     this->timestamp = newTimestamp;
121 }
122
123 void Market::printDaySummary() {
124     std::cout << "---End of Day---" << std::endl;
125     std::cout << "Commission Earnings: $" << commission << std::endl;
126     std::cout << "Total Amount of Money Transferred: $" << transferMoney
127               << std::endl;
128     std::cout << "Number of Completed Trades: " << tradeComplete << std::endl;
129     std::cout << "Number of Shares Traded: " << shareTrade << std::endl;
130     if (transfers) {
131         for (auto &client:this->clients) {
```

```
132                 client->printTransfer();
133             }
134         }
135         for (auto &name: this->timeTravelers) {
136             this->getStock(name)->printTimeTraveler();
137         }
138     }
```

1.1.8 main.cpp

```
1   //
2   // Created by liu on 2017/9/16.
3   //
4
5   #include <string>
6   #include <iostream>
7   #include <fstream>
8   #include <getopt.h>
9   #include "Market.h"
10
11  int main(int argc, char *argv[]) {
12      //std::ifstream fin("test.txt");
13      //std::cin.rdbuf(fin.rdbuf());
14
15      Market &market = Market::getInstance();
16
17      bool verbose = false;
18      bool median = false;
19      bool midpoint = false;
20      bool transfers = false;
21
22      std::string impl;
23      while (true) {
24          const option long_options[] = {
25                  {"verbose",   no_argument,       NULL, 'v'},
26                  {"median",    no_argument,       NULL, 'm'},
27                  {"midpoint",  no_argument,       NULL, 'p'},
28                  {"transfers", no_argument,       NULL, 't'},
29                  {"ttt",       required_argument, NULL, 'g'},
30                  {0, 0, 0,                              0}
31          };
32          int c = getopt_long(argc, argv, "vmptg:", long_options, NULL);
33          if (c == -1) break;
34          switch (c) {
35          case 'v':
36              verbose = true;
37              break;
38          case 'm':
39              median = true;
40              break;
```

```
41          case 'p':
42              midpoint = true;
43              break;
44          case 't':
45              transfers = true;
46              break;
47          case 'g':
48              market.initTimeTraveler(optarg);
49              break;
50          default:
51              break;
52          }
53      }
54
55      market.initOptions(verbose, median, midpoint, transfers);
56
57      std::string str;
58      while (!std::cin.eof()) {
59          std::getline(std::cin, str);
60          if (!str.empty()) market.readLine(str);
61      }
62      market.printTickSummary();
63      market.printDaySummary();
64      return 0;
65  }
```

1.1.9 Makefile

```
1  all: main.cpp Client.cpp Market.cpp Stock.cpp
2      g++ -std=c++11 -o main main.cpp Client.cpp Market.cpp Stock.cpp
```