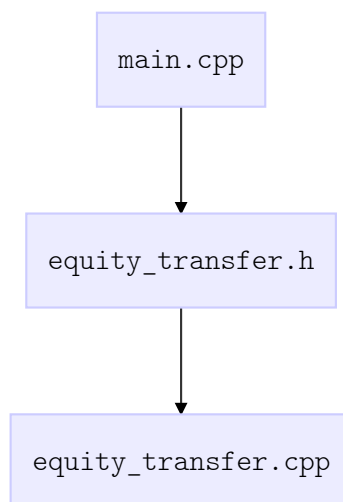


Project 4

Bingcheng HU

516021910219

File list



Appendix

main.cpp

```
1 | #include <iostream>
2 | #include <sstream>
3 | #include <getopt.h>
4 | #include "equity_transfer.h"
```

```

5
6 using namespace std;
7
8 int main(int argc, char *argv[]) {
9     std::ios::sync_with_stdio(false);
10    std::cin.tie(0);
11    // First get the opcode
12    bool verbose=false;
13    bool midpoint=false;
14    bool transfers=false;
15    bool median=false;
16    bool ttt=false;
17    int ID=0;
18    set<ttt_price *, ttt_equity> ttt_quity_set;
19    int c;
20    while (true) {
21        const option long_options[] = {
22            {"verbose", no_argument, NULL, 'v'},
23            {"median", no_argument, NULL, 'm'},
24            {"midpoint", no_argument, NULL, 'p'},
25            {"transfers", no_argument, NULL, 't'},
26            {"ttt", required_argument, NULL, 'g'},
27            {0, 0, 0, 0}
28        };
29        int option_index = 0;
30        c = getopt_long(argc, argv, "vmptg:", long_options, &option_index);
31        if (c == -1) break;
32        switch (c) {
33            case '?':
34                cerr<<"???"<<endl;
35                break;
36            default:
37                abort ();
38            case 'v':
39                verbose = true;
40                break;
41            case 'm':
42                median = true;
43                break;
44            case 'p':
45                midpoint = true;
46                break;
47            case 't':
48                transfers = true;
49                break;
50            case 'g':
51                ttt=true;
52                auto equity_temp=new ttt_price;
53                equity_temp->ID=ID;

```

```

54         equity_temp->equity_symbol=optarg;
55         equity_temp->timestamp_buy=-1;
56         equity_temp->timestamp_sell=-1;
57         equity_temp->price_buy=0;
58         equity_temp->price_sell=0;
59         ttt_quity_set.insert(equity_temp);
60         ID++;
61         break;
62
63     }
64 }
65 int timestamp_now=0;
66 int next_ID=0;
67 int TIMESTAMP=0;
68 string CLIENT_NAME;
69 string BUY_OR_SELL;
70 bool buy_signal=false;
71 string EQUITY_SYMBOL;
72 string limit_price;
73 int LIMIT_PRICE=0;
74 string quantity;
75 int QUANTITY=0;
76 int DURATION=0;
77 int count_amount=0;
78 int count=0;
79 int count_transfer=0;
80 int single_commission=0;
81 int total_commission=0;
82
83 map<string, equity_book> order_map;
84 set<equity *, equity_buy> *order_buy_set=NULLptr;
85 set<equity *, equity_sell> *order_sell_set=NULLptr;
86 map<string, client_equity *> client_map;
87 stringstream stream_temp;
88 while(!cin.eof()) {
89     string str;
90     getline(cin, str);
91     if(str.empty()) {
92         break;
93     }
94     stream_temp.clear();
95     stream_temp.str(str);
96
97     stream_temp>>TIMESTAMP>>CLIENT_NAME>>BUY_OR_SELL>>EQUITY_SYMBOL>>limit_price>>quantity>>
98     DURATION;;
99     LIMIT_PRICE=atoi(limit_price.substr(1, limit_price.length()).c_str());
100    QUANTITY=atoi(quantity.substr(1, quantity.length()).c_str());
101
102    if(BUY_OR_SELL=="BUY") {

```

```

101         buy_signal=true;
102     }
103     else if(BUY_OR_SELL=="SELL") {
104         buy_signal=false;
105     }
106     else {
107         exit(0);
108     }
109
110     auto client_equity_temp=new client_equity;
111     client_equity_temp->name=CLIENT_NAME;
112     client_equity_temp->buy_count=0;
113     client_equity_temp->sell_count=0;
114     client_equity_temp->net_count=0;
115     client_map.insert(make_pair(CLIENT_NAME, (client_equity_temp)));
116
117
118     for(auto tttEquity_equity_it=ttt_quity_set.begin();
119         tttEquity_equity_it!=ttt_quity_set.end(); ++tttEquity_equity_it) {
120
121         if((*tttEquity_equity_it)->equity_symbol==EQUITY_SYMBOL) {
122             auto ttt_equity_pt=(*tttEquity_equity_it);
123             if(buy_signal) {
124                 if(ttt_equity_pt->timestamp_buy==1) {
125                     ttt_equity_pt->buy_flag=false;
126                     break;
127                 }
128                 else if(ttt_equity_pt->timestamp_sell==1 || (ttt_equity_pt->
129 price_sell<LIMIT_PRICE && ttt_equity_pt->buy_flag==false)) {
130                     ttt_equity_pt->price_sell=LIMIT_PRICE;
131                     ttt_equity_pt->timestamp_sell=TIMESTAMP;
132                     ttt_equity_pt->price_earn_max=ttt_equity_pt->price_sell-
133 ttt_equity_pt->price_buy;
134                     ttt_quity_set.insert(ttt_equity_pt);
135                 }
136                 else if(ttt_equity_pt->buy_flag==true && (LIMIT_PRICE -
137 ttt_equity_pt->price_buy_temp)>ttt_equity_pt->price_earn_max){
138                     ttt_equity_pt->price_sell=LIMIT_PRICE;
139                     ttt_equity_pt->timestamp_sell=TIMESTAMP;
140                     ttt_equity_pt->timestamp_buy=ttt_equity_pt->timestamp_buy_temp;
141                     ttt_equity_pt->price_buy=ttt_equity_pt->price_buy_temp;
142                     ttt_equity_pt->price_earn_max=ttt_equity_pt->price_sell-
143 ttt_equity_pt->price_buy;
144                     ttt_equity_pt->buy_flag=false;
145                     ttt_quity_set.insert(ttt_equity_pt);
146                 }
147             }
148             else {
149                 if(ttt_equity_pt->timestamp_buy==1) {

```

```

146         ttt_equity_pt->price_buy=LIMIT_PRICE;
147         ttt_equity_pt->timestamp_buy=TIMESTAMP;
148         ttt_equity_pt->buy_flag=false;
149         ttt_equity_pt->price_earn_max=0;
150         ttt_quity_set.insert(ttt_equity_pt);
151     }
152     else if(ttt_equity_pt->price_buy>LIMIT_PRICE && ttt_equity_pt-
153 >timestamp_sell==1){
154         ttt_equity_pt->timestamp_buy=TIMESTAMP;
155         ttt_equity_pt->price_buy=LIMIT_PRICE;
156     }
157     else if(ttt_equity_pt->price_buy>LIMIT_PRICE){
158         ttt_equity_pt->buy_flag=true;
159         ttt_equity_pt->timestamp_buy_temp=TIMESTAMP;
160         ttt_equity_pt->price_buy_temp=LIMIT_PRICE;
161     }
162 }
163 }
164
165 if(TIMESTAMP!=timestamp_now) {
166     if(median) {
167         get_median(order_map, timestamp_now);
168     }
169     if(midpoint) {
170         get_midpoint(order_map, timestamp_now);
171     }
172     timestamp_now=TIMESTAMP;
173
174     get_expire(order_map, order_buy_set, order_sell_set, timestamp_now);
175
176 }
177
178 auto order_all_iterator=order_map.find(EQUITY_SYMBOL);
179 if(order_all_iterator==order_map.end()) {
180     equity_book equitybook_temp=equity_book();
181     equitybook_temp.EQUITY_SYMBOL=EQUITY_SYMBOL;
182     order_all_iterator=order_map.insert(make_pair(EQUITY_SYMBOL,
183 (equitybook_temp))).first;
184 }
185
186 if(buy_signal) {
187     deal_buy(order_map, client_map, timestamp_now, order_all_iterator, QUANTITY,
188 LIMIT_PRICE, next_ID,
189 transfers, CLIENT_NAME, verbose, EQUITY_SYMBOL, count_amount, count,
190 count_transfer,
191 single_commission, total_commission);
192 }
193 else {

```

```

191         deal_sell(order_map, client_map, timestamp_now, order_all_iterator, QUANTITY,
LIMIT_PRICE, next_ID,
192             transfers, CLIENT_NAME, verbose, EQUITY_SYMBOL, count_amount,
count, count_transfer,
193             single_commission, total_commission);
194     }
195     if(QUANTITY>0&&DURATION!=0) {
196         auto Order_temp=new equity;
197         Order_temp->ID=next_ID++;
198         Order_temp->PRICE=LIMIT_PRICE;
199         Order_temp->NAME=CLIENT_NAME;
200         Order_temp->AMOUNT=QUANTITY;
201         Order_temp->EXPIRE_TIME=(DURATION!=-1)?(timestamp_now+DURATION):-1;
202         if(buy_signal) {
203             order_buy_set=&(order_all_iterator->second.orderBuy);
204             order_buy_set->insert(Order_temp);
205         }
206         else {
207             order_sell_set=&(order_all_iterator->second.orderSell);
208             order_sell_set->insert(Order_temp);
209         }
210     }
211 }
212
213 if(median) {
214     get_median(order_map, timestamp_now);
215 }
216
217 if(midpoint) {
218     get_midpoint(order_map, timestamp_now);
219 }
220
221 final_print(count_amount, count, count_transfer, total_commission);
222
223 if(transfers) {
224     get_transfers(client_map);
225 }
226 if(ttt) {
227     for(auto tttEquity_record_it=ttt_quity_set.begin();
228         tttEquity_record_it!=ttt_quity_set.end(); tttEquity_record_it++) {
229         if((*tttEquity_record_it)->timestamp_buy<0||(*tttEquity_record_it)-
>timestamp_sell<0) {
230             cout<<"Time travelers would buy "<<(*tttEquity_record_it)-
>equity_symbol<<" at time: "
231                 <<-1<<" and sell it at time: "
232                 <<-1<<endl;
233             continue;
234         }
235         else {

```

```

236         cout<<"Time travelers would buy "<<(*tttEquity_record_it)-
>equity_symbol<<" at time: "
237         <<(*tttEquity_record_it)->timestamp_buy<<" and sell it at time: "
238         <<(*tttEquity_record_it)->timestamp_sell<<endl;
239     }
240 }
241 }
242 return 0;
243 }

```

equity_transfer.h

```

1  #ifndef EQUITY_TRANSFER_H
2  #define EQUITY_TRANSFER_H
3
4  #include <iostream>
5  #include <string>
6  #include <map>
7  #include <set>
8
9  using namespace std;
10
11 class equity {
12 public:
13     int ID;
14     string NAME;
15     int AMOUNT;
16     int PRICE;
17     int EXPIRE_TIME;
18 };
19
20 struct equity_buy {
21     bool operator()(const equity *a, const equity *b) const {
22         if(a->PRICE>b->PRICE) {
23             return true;
24         }
25         else if(a->PRICE==b->PRICE) {
26             return a->ID<b->ID;
27         }
28         else {
29             return false;
30         }
31     }
32 };
33
34 struct equity_sell {
35     bool operator()(const equity *a, const equity *b) const {
36         if(a->PRICE>b->PRICE) {
37             return false;

```

```

38         }
39         else if(a->PRICE==b->PRICE) {
40             return a->ID<b->ID;
41         }
42         else {
43             return true;
44         }
45     }
46 };
47
48 class equity_book {
49 public:
50     string EQUITY_SYMBOL;
51     set<equity *, equity_buy> orderBuy;
52     set<equity *, equity_sell> orderSell;
53     multiset<int> history;
54 };
55
56 class ttt_price {
57 public:
58     int ID;
59     string equity_symbol;
60     int timestamp_buy;
61     int timestamp_sell;
62     int price_buy;
63     int price_sell;
64     int price_buy_temp;
65     int timestamp_buy_temp;
66     int price_earn_max;
67     bool buy_flag;
68 };
69
70 struct ttt_equity {
71     bool operator()(const ttt_price *a, const ttt_price *b) const {
72         return a->ID<b->ID;
73     }
74 };
75
76 class client_equity {
77 public:
78     string name="";
79     int buy_count=0;
80     int sell_count=0;
81     int net_count=0;
82 };
83
84 void get_median(map<string, equity_book> &order_map, int timestamp_now);
85
86 void get_midpoint(map<string, equity_book> &order_list, int timestamp_now);

```



```

87
88 void get_transfers(map<string, client_equity *> &client_map);
89
90 void get_expire(map<string, equity_book> &order_list, set<equity *, equity_buy>
    *order_buy_set,
91                 set<equity *, equity_sell> *order_sell_set, int timestamp_now);
92
93 void deal_buy(map<string, equity_book> &order_list, map<string, client_equity *>
    &client_map, int timestamp_now,
94               map<string, equity_book>::iterator order_all_iterator, int &QUANTITY, int
    LIMIT_PRICE, int next_ID,
95               bool transfers, string &CLIENT_NAME, bool verbose, string &EQUITY_SYMBOL,
    int &NUMBER_OF_COMPLETED_TRADES,
96               int &count, int &NUMBER_OF_SHARES_TRADED, int &COMMISION_EARNINGS, int
    &MONEY_TRANSFERRED);
97
98 void deal_sell(map<string, equity_book> &order_map, map<string, client_equity *>
    &client_map, int timestamp_now,
99               map<string, equity_book>::iterator order_all_iterator, int &QUANTITY, int
    LIMIT_PRICE, int next_ID,
100               bool transfers, string &CLIENT_NAME, bool verbose, string &EQUITY_SYMBOL,
    int &count_num,
101               int &NUMBER_OF_COMPLETED_TRADES, int &NUMBER_OF_SHARES_TRADED, int
    &COMMISION_EARNINGS,
102               int &MONEY_TRANSFERRED);
103
104 void final_print(int NUMBER_OF_SHARES_TRADED, int NUMBER_OF_COMPLETED_TRADES, int
    MONEY_TRANSFERRED,
105                  int COMMISION_EARNINGS);
106
107 #endif

```

equity_transfer.cpp

```

1  #include "equity_transfer.h"
2
3  using namespace std;
4
5  void get_median(map<string, equity_book> &order_map, int timestamp_now) {
6      int median_price=0;
7      for(auto orderAll_it=order_map.begin(); orderAll_it!=order_map.end(); ++orderAll_it)
8      {
9          ssize_t size=orderAll_it->second.history.size();
10         if(size!=0) {
11             bool even=(size%2==0);
12             size/=2;
13             auto median_price_it=orderAll_it->second.history.begin();
14             for(auto i=0; i<size; ++i) {
15                 ++median_price_it;

```

```

15         }
16         if(!even) {
17             median_price=*median_price_it;
18         }
19         else {
20             median_price=((*median_price_it)+*(--median_price_it))/2;
21         }
22         cout<<"Median match price of "<<orderAll_it->second.EQUITY_SYMBOL<<" at time
" <<timestamp_now<<" is $"
23             <<median_price<<endl;
24     }
25     else {
26         continue;
27     }
28 }
29 }
30
31 void get_midpoint(map<string, equity_book> &order_list, int timestamp_now) {
32     for(auto orderAll_it=order_list.begin(); orderAll_it!=order_list.end();
++orderAll_it) {
33         if(orderAll_it->second.orderBuy.empty()||orderAll_it->second.orderSell.empty()) {
34             cout<<"Midpoint of "<<orderAll_it->second.EQUITY_SYMBOL<<" at time "
<<timestamp_now<<" is undefined"
35                 <<endl;
36             continue;
37         }
38         auto midpoint_price=
39             ((*orderAll_it->second.orderBuy.begin())->PRICE+(*orderAll_it->
>second.orderSell.begin())->PRICE)/2;
40         cout<<"Midpoint of "<<orderAll_it->second.EQUITY_SYMBOL<<" at time "
<<timestamp_now<<" is $"<<midpoint_price
41             <<endl;
42     }
43 }
44
45 void get_transfers(map<string, client_equity *> &client_map) {
46     for(auto clientAll_it=client_map.begin(); clientAll_it!=client_map.end();
++clientAll_it) {
47         // if(clientAll_it->second->buy_count == 0 && clientAll_it->second-
>sell_count==0) continue;
48         //cerr<<clientAll_it->second->buy_count <<"##"<< clientAll_it->second-
>sell_count<<endl;
49         cout<<clientAll_it->second->name<<" bought "<<clientAll_it->second->buy_count<<"
and sold "
50             <<clientAll_it->second->sell_count<<" for a net transfer of $"<<clientAll_it-
>second->net_count<<endl;
51     }
52 }
53

```

```

54 void get_expire(map<string, equity_book> &order_list, set<equity *, equity_buy>
    *order_buy_set,
55                 set<equity *, equity_sell> *order_sell_set, int timestamp_now) {
56     set<equity *, equity_sell>::iterator Sell_set_it;
57     set<equity *, equity_sell>::iterator Sell_set_it_temp;
58     set<equity *, equity_buy>::iterator buy_set_it;
59     set<equity *, equity_sell>::iterator buy_set_it_temp;
60     for(auto orderAll_it=order_list.begin(); orderAll_it!=order_list.end();
++orderAll_it) {
61         order_sell_set=&(orderAll_it->second.orderSell);
62         for(Sell_set_it=order_sell_set->begin(); Sell_set_it!=order_sell_set->end();) {
63             if((*Sell_set_it)->EXPIRE_TIME!=-1&&(*Sell_set_it)-
>EXPIRE_TIME<=timestamp_now) {
64                 Sell_set_it_temp=Sell_set_it;
65                 Sell_set_it=order_sell_set->erase(Sell_set_it_temp);
66             }
67             else {
68                 ++Sell_set_it;
69             }
70         }
71         order_buy_set=&(orderAll_it->second.orderBuy);
72         for(buy_set_it=order_buy_set->begin(); buy_set_it!=order_buy_set->end();) { //
bug here
73             if((*buy_set_it)->EXPIRE_TIME!=-1&&(*buy_set_it)->EXPIRE_TIME<=timestamp_now)
{
74                 buy_set_it_temp=buy_set_it;
75                 buy_set_it=order_buy_set->erase(buy_set_it_temp);
76             }
77             else {
78                 ++buy_set_it;
79             }
80         }
81     }
82 }
83
84 void deal_buy(map<string, equity_book> &order_list, map<string, client_equity *>
&client_map, int timestamp_now,
85               map<string, equity_book>::iterator order_all_iterator, int &QUANTITY, int
LIMIT_PRICE, int next_ID,
86               bool transfers, string &CLIENT_NAME, bool verbose, string &EQUITY_SYMBOL,
int &NUMBER_OF_COMPLETED_TRADES,
87               int &count, int &NUMBER_OF_SHARES_TRADED, int &COMMISSION_EARNINGS, int
&MONEY_TRANSFERRED) {
88     auto transact_price=0;
89
90     auto orderSell_ptr=&(order_all_iterator->second.orderSell);
91
92     while(QUANTITY>0&&!orderSell_ptr->empty()) {
93         if((*orderSell_ptr->begin())->PRICE<=LIMIT_PRICE) {

```

```

94     auto order_pt=*orderSell_ptr->begin();
95     if(order_pt->ID>=next_ID) {
96         transact_price=LIMIT_PRICE;
97     }
98     else {
99         transact_price=order_pt->PRICE;
100    }
101
102    auto equitybook_ptr=&(order_all_iterator->second);
103    equitybook_ptr->history.insert(transact_price);
104
105    if(order_pt->AMOUNT>QUANTITY) {
106        if(transfers) {
107            auto clientAll_it_1=client_map.find(CLIENT_NAME);
108            bool find_buyer!=(clientAll_it_1==client_map.end());
109
110            if(!find_buyer) {
111                auto client_record_temp=new client_equity;
112                client_record_temp->name=CLIENT_NAME;
113                client_record_temp->buy_count=QUANTITY;
114                client_record_temp->sell_count=0;
115                client_record_temp->net_count=QUANTITY*transact_price*(-1);
116                client_map.insert(make_pair(CLIENT_NAME, (client_record_temp)));
117            }
118            else {
119                auto client_map_pt_A=(clientAll_it_1->second);
120                client_map_pt_A->buy_count+=QUANTITY;
121                client_map_pt_A->net_count+=QUANTITY*transact_price*(-1);
122            }
123
124            auto client_map_it_B=client_map.find(order_pt->NAME);
125            bool find_seller!=(client_map_it_B==client_map.end());
126
127            if(!find_seller) {
128                auto client_record_temp=new client_equity;
129                client_record_temp->name=order_pt->NAME;
130                client_record_temp->buy_count=0;
131                client_record_temp->sell_count=QUANTITY;
132                client_record_temp->net_count=QUANTITY*transact_price;
133                client_map.insert(make_pair(order_pt->NAME,
134(client_record_temp)));
135            }
136            else {
137                auto clientAll_ptr_2=(client_map_it_B->second);
138                clientAll_ptr_2->sell_count+=QUANTITY;
139                clientAll_ptr_2->net_count+=QUANTITY*transact_price;
140            }
141        }
142        if(verbose) {

```

```

142         cout<<CLIENT_NAME<<" purchased "<<QUANTITY<<" shares of "
<<EQUITY_SYMBOL<<" from "<<order_pt->NAME
143         <<" for $"<<transact_price<<"/share"<<endl;
144     }
145     ++count;
146     NUMBER_OF_COMPLETED_TRADES+=QUANTITY;
147     NUMBER_OF_SHARES_TRADED+=transact_price*QUANTITY;
148     COMMISSION_EARNINGS=(transact_price*QUANTITY)/100;
149     MONEY_TRANSFERRED+=COMMISSION_EARNINGS*2;
150     order_pt->AMOUNT-=QUANTITY;
151     QUANTITY=0;
152 }
153 else if(order_pt->AMOUNT==QUANTITY) {
154     if(transfers) {
155         auto clientAll_it_1=client_map.find(CLIENT_NAME);
156         bool find_buyer=!(clientAll_it_1==client_map.end());
157
158         if(!find_buyer) {
159             auto client_record_temp=new client_equity;
160             client_record_temp->name=CLIENT_NAME;
161             client_record_temp->buy_count=QUANTITY;
162             client_record_temp->sell_count=0;
163             client_record_temp->net_count=QUANTITY*transact_price*(-1);
164             client_map.insert(make_pair(CLIENT_NAME, (client_record_temp)));
165         }
166         else {
167             auto client_map_pt_A=(clientAll_it_1->second);
168             client_map_pt_A->buy_count+=QUANTITY;
169             client_map_pt_A->net_count+=QUANTITY*transact_price*(-1);
170         }
171
172         auto client_map_it_B=client_map.find(order_pt->NAME);
173         bool find_seller=!(client_map_it_B==client_map.end());
174
175         if(!find_seller) {
176             auto client_record_temp=new client_equity;
177             client_record_temp->name=order_pt->NAME;
178             client_record_temp->buy_count=0;
179             client_record_temp->sell_count=QUANTITY;
180             client_record_temp->net_count=QUANTITY*transact_price;
181             client_map.insert(make_pair(order_pt->NAME,
182 (client_record_temp)));
183         }
184         else {
185             auto clientAll_ptr_2=(client_map_it_B->second);
186             clientAll_ptr_2->sell_count+=QUANTITY;
187             clientAll_ptr_2->net_count+=QUANTITY*transact_price;
188         }
189     }

```

```

189         if(verbose) {
190             cout<<CLIENT_NAME<<" purchased "<<QUANTITY<<" shares of "
191 <<EQUITY_SYMBOL<<" from "<<order_pt->NAME
192             <<" for $"<<transact_price<<"/share"<<endl;
193         }
194         ++count;
195         NUMBER_OF_COMPLETED_TRADES+=QUANTITY;
196         NUMBER_OF_SHARES_TRADED+=transact_price*QUANTITY;
197         COMMISSION_EARNINGS=(transact_price*QUANTITY)/100;
198         MONEY_TRANSFERRED+=COMMISSION_EARNINGS*2;
199         QUANTITY=0;
200         orderSell_ptr->erase(orderSell_ptr->begin());
201     }
202     else {
203         if(transfers) {
204             auto clientAll_it_1=client_map.find(CLIENT_NAME);
205             bool find_buyer!=(clientAll_it_1==client_map.end());
206
207             if(!find_buyer) {
208                 auto client_record_temp=new client_equity;
209                 client_record_temp->name=CLIENT_NAME;
210                 client_record_temp->buy_count=order_pt->AMOUNT;
211                 client_record_temp->sell_count=0;
212                 client_record_temp->net_count=order_pt->AMOUNT*transact_price*
213 (-1);
214                 client_map.insert(make_pair(CLIENT_NAME, (client_record_temp)));
215             }
216             else {
217                 auto clientAll_ptr=(clientAll_it_1->second);
218                 clientAll_ptr->buy_count+=order_pt->AMOUNT;
219                 clientAll_ptr->net_count+=order_pt->AMOUNT*transact_price*(-1);
220             }
221
222             auto client_map_it_B=client_map.find(order_pt->NAME);
223             bool find_seller!=(client_map_it_B==client_map.end());
224
225             if(!find_seller) {
226                 auto client_record_temp=new client_equity;
227                 client_record_temp->name=order_pt->NAME;
228                 client_record_temp->buy_count=0;
229                 client_record_temp->sell_count=order_pt->AMOUNT;
230                 client_record_temp->net_count=order_pt->AMOUNT*transact_price;
231                 client_map.insert(make_pair(order_pt->NAME,
232 (client_record_temp)));
233             }
234             else {
235                 auto clientAll_ptr_2=(client_map_it_B->second);
236                 clientAll_ptr_2->sell_count+=order_pt->AMOUNT;
237                 clientAll_ptr_2->net_count+=order_pt->AMOUNT*transact_price;

```

```

235         }
236     }
237     if(verbose) {
238         cout<<CLIENT_NAME<<" purchased "<<order_pt->AMOUNT<<" shares of "
<<EQUITY_SYMBOL<<" from "
239         <<order_pt->NAME<<" for $"<<transact_price<<"/share"<<endl;
240     }
241     NUMBER_OF_COMPLETED_TRADES+=order_pt->AMOUNT;
242     ++count;
243     NUMBER_OF_SHARES_TRADED+=transact_price*order_pt->AMOUNT;
244     COMMISSION_EARNINGS=transact_price*order_pt->AMOUNT/100;
245     MONEY_TRANSFERRED+=COMMISSION_EARNINGS*2;
246     QUANTITY-=order_pt->AMOUNT;
247     orderSell_ptr->erase(orderSell_ptr->begin());
248 }
249 }
250 else {
251     break;
252 }
253 }
254
255 }
256
257 void deal_sell(map<string, equity_book> &order_map, map<string, client_equity *>
&client_map, int timestamp_now,
258     map<string, equity_book>::iterator order_all_iterator, int &QUANTITY, int
LIMIT_PRICE, int next_ID,
259     bool transfers, string &CLIENT_NAME, bool verbose, string &EQUITY_SYMBOL,
int &count_num,
260     int &NUMBER_OF_COMPLETED_TRADES, int &NUMBER_OF_SHARES_TRADED, int
&COMMISSION_EARNINGS,
261     int &MONEY_TRANSFERRED) {
262     auto transact_price=0;
263     auto orderBuy_ptr=&(order_all_iterator->second.orderBuy);
264     while(QUANTITY>0&&!orderBuy_ptr->empty()) {
265         if((*orderBuy_ptr->begin())->PRICE>=LIMIT_PRICE) {
266             auto order_pt=*orderBuy_ptr->begin();
267             if(order_pt->ID>=next_ID) {
268                 transact_price=LIMIT_PRICE;
269             }
270             else {
271                 transact_price=order_pt->PRICE;
272             }
273             auto equitybook_ptr=&(order_all_iterator->second);
274             equitybook_ptr->history.insert(transact_price);
275             if(order_pt->AMOUNT>QUANTITY) {
276                 if(transfers) {
277                     auto clientAll_it_1=client_map.find(CLIENT_NAME);
278                     bool find_seller=!(clientAll_it_1==client_map.end());

```

```

279
280         if(!find_seller) {
281             auto client_record_temp=new client_equity;
282             client_record_temp->name=CLIENT_NAME;
283             client_record_temp->buy_count=0;
284             client_record_temp->sell_count=QUANTITY;
285             client_record_temp->net_count=QUANTITY*transact_price*(1);
286             client_map.insert(make_pair(CLIENT_NAME, (client_record_temp)));
287         }
288         else {
289             auto client_map_pt_A=(clientAll_it_1->second);
290             client_map_pt_A->sell_count+=QUANTITY;
291             client_map_pt_A->net_count+=QUANTITY*transact_price*(1);
292         }
293
294         auto client_map_it_B=client_map.find(order_pt->NAME);
295         bool find_buyer!=(client_map_it_B==client_map.end());
296
297         if(!find_buyer) {
298             auto client_record_temp=new client_equity;
299             client_record_temp->name=order_pt->NAME;
300             client_record_temp->buy_count=QUANTITY;
301             client_record_temp->sell_count=0;
302             client_record_temp->net_count=QUANTITY*transact_price*(-1);
303             client_map.insert(make_pair(order_pt->NAME,
304 (client_record_temp)));
305         }
306         else {
307             auto clientAll_ptr_2=(client_map_it_B->second);
308             clientAll_ptr_2->buy_count+=QUANTITY;
309             clientAll_ptr_2->net_count+=QUANTITY*transact_price*(-1);
310         }
311         if(verbose) {
312             cout<<order_pt->NAME<<" purchased "<<QUANTITY<<" shares of "
313 <<EQUITY_SYMBOL<<" from "<<CLIENT_NAME
314 <<" for $"<<transact_price<<"/share"<<endl;
315         }
316         count_num+=QUANTITY;
317         ++NUMBER_OF_COMPLETED_TRADES;
318         NUMBER_OF_SHARES_TRADED+=transact_price*QUANTITY;
319         COMMISSION_EARNINGS=transact_price*QUANTITY/100;
320         MONEY_TRANSFERRED+=COMMISSION_EARNINGS*2;
321         order_pt->AMOUNT-=QUANTITY;
322         QUANTITY=0;
323     }
324     else if(order_pt->AMOUNT==QUANTITY) {
325         if(transfers) {
326             auto clientAll_it_1=client_map.find(CLIENT_NAME);

```



```

326         bool find_seller=(clientAll_it_1==client_map.end());
327
328
329         if(!find_seller) {
330             auto client_record_temp=new client_equity;
331             client_record_temp->name=CLIENT_NAME;
332             client_record_temp->buy_count=0;
333             client_record_temp->sell_count=QUANTITY;
334             client_record_temp->net_count=QUANTITY*transact_price*(1);
335             client_map.insert(make_pair(CLIENT_NAME, (client_record_temp)));
336         }
337         else {
338             auto client_map_pt_A=(clientAll_it_1->second);
339             client_map_pt_A->sell_count+=QUANTITY;
340             client_map_pt_A->net_count+=QUANTITY*transact_price*(1);
341         }
342
343         auto client_map_it_B=client_map.find(order_pt->NAME);
344         bool find_buyer=(client_map_it_B==client_map.end());
345
346
347         if(!find_buyer) {
348             auto client_record_temp=new client_equity;
349             client_record_temp->name=order_pt->NAME;
350             client_record_temp->buy_count=QUANTITY;
351             client_record_temp->sell_count=0;
352             client_record_temp->net_count=QUANTITY*transact_price*(-1);
353             client_map.insert(make_pair(order_pt->NAME,
354 (client_record_temp)));
355         }
356         else {
357             auto clientAll_ptr_2=(client_map_it_B->second);
358             clientAll_ptr_2->buy_count+=QUANTITY;
359             clientAll_ptr_2->net_count+=QUANTITY*transact_price*(-1);
360         }
361         if(verbose) {
362
363             cout<<order_pt->NAME<<" purchased "<<QUANTITY<<" shares of "
364 <<EQUITY_SYMBOL<<" from "<<CLIENT_NAME
365             <<" for $"<<transact_price<<"/share"<<endl;
366         }
367         count_num+=QUANTITY;
368         ++NUMBER_OF_COMPLETED_TRADES;
369         COMMISSION_EARNINGS=transact_price*QUANTITY/100;
370         MONEY_TRANSFERRED+=COMMISSION_EARNINGS*2;
371         NUMBER_OF_SHARES_TRADED+=transact_price*QUANTITY;
372         QUANTITY=0;
373         orderBuy_ptr->erase(orderBuy_ptr->begin());

```

```

373     }
374     else {
375         if(transfers) {
376             auto clientAll_it_1=client_map.find(CLIENT_NAME);
377             bool find_seller!=(clientAll_it_1==client_map.end());
378
379             if(!find_seller) {
380                 auto client_record_temp=new client_equity;
381                 client_record_temp->name=CLIENT_NAME;
382                 client_record_temp->buy_count=0;
383                 client_record_temp->sell_count=order_pt->AMOUNT;
384                 client_record_temp->net_count=order_pt->AMOUNT*transact_price*
(1);
385
386                 client_map.insert(make_pair(CLIENT_NAME, (client_record_temp)));
387             }
388             else {
389                 auto client_map_pt_A=(clientAll_it_1->second);
390                 client_map_pt_A->sell_count+=order_pt->AMOUNT;
391                 client_map_pt_A->net_count+=order_pt->AMOUNT*transact_price*(1);
392             }
393
394             auto client_map_it_B=client_map.find(order_pt->NAME);
395             bool find_buyer!=(client_map_it_B==client_map.end());
396
397             if(!find_buyer) {
398                 auto client_record_temp=new client_equity;
399                 client_record_temp->name=order_pt->NAME;
400                 client_record_temp->buy_count=order_pt->AMOUNT;
401                 client_record_temp->sell_count=0;
402                 client_record_temp->net_count=order_pt->AMOUNT*transact_price*
(-1);
403
404                 client_map.insert(make_pair(order_pt->NAME,
(client_record_temp)));
405             }
406             else {
407                 auto clientAll_ptr_2=(client_map_it_B->second);
408                 clientAll_ptr_2->buy_count+=order_pt->AMOUNT;
409                 clientAll_ptr_2->net_count+=order_pt->AMOUNT*transact_price*(-1);
410             }
411         }
412         if(verbose) {
413             cout<<order_pt->NAME<<" purchased "<<order_pt->AMOUNT<<" shares of "
<<EQUITY_SYMBOL<<" from "
414
415             <<CLIENT_NAME<<" for $"<<transact_price<<"/share"<<endl;
416         }
417         count_num+=order_pt->AMOUNT;
418         ++NUMBER_OF_COMPLETED_TRADES;
419         NUMBER_OF_SHARES_TRADED+=transact_price*order_pt->AMOUNT;
420         COMMISSION_EARNINGS=transact_price*order_pt->AMOUNT/100;

```

```

418         MONEY_TRANSFERRED+=COMMISSION_EARNINGS*2;
419         QUANTITY-=order_pt->AMOUNT;
420         orderBuy_ptr->erase(orderBuy_ptr->begin());
421     }
422 }
423 else {
424     break;
425 }
426 }
427
428 }
429
430 void final_print(int NUMBER_OF_SHARES_TRADED, int NUMBER_OF_COMPLETED_TRADES, int
MONEY_TRANSFERRED,
431                 int COMMISSION_EARNINGS) {
432     cout<<"---End of Day---\n"<<"Commission Earnings: $"<<COMMISSION_EARNINGS
433     <<"\nTotal Amount of Money Transferred: $"<<MONEY_TRANSFERRED
434     <<"\nNumber of Completed Trades: "<<NUMBER_OF_COMPLETED_TRADES
435     <<"\nNumber of Shares Traded: "<<NUMBER_OF_SHARES_TRADED<<endl;
436 }

```