UM–SJTU Joint Institute

Data Structures and Algorithms
(VE281)

Assignment Report

Project 4

Name: Ji Xingyou ID: 515370910197
Date: 28 November 2017

Contents

```
#include <iostream>
#include <sstream>
#include <string>
#include <getopt.h>
#include <map>
#include <set>

using namespace std;

class Order {
public:
    int ID;
    string NAME;
    int AMOUNT;
    int PRICE;
    int EXPIRE_TIME;
};

struct comp_buy {
    bool operator()(const Order *a, const Order *b) const {
        if(a->PRICE>b->PRICE) {
            return true;
        }
        else if(a->PRICE==b->PRICE) {
            return a->ID<b->ID;
        }
        else {
            return false;
        }
    }
};

struct comp_sell {
    bool operator()(const Order *a, const Order *b) const {
        if(a->PRICE>b->PRICE) {
            return false;
        }
```

```
        else if(a->PRICE==b->PRICE) {
            return a->ID<b->ID;
        }
        else {
            return true;
        }
    }
};

class equitybook {
public:
    string EQUITY_SYMBOL;
    set<Order *, comp_buy> orderBuy;
    set<Order *, comp_sell> orderSell;
    multiset<int> history;
};

class ttt_equity {
public:
    int ID;
    string equity_symbol;
    int timestamp_buy;
    int timestamp_sell;
    int price_buy;
    int price_sell;
};

struct comp_ttt_equity {
    bool operator()(const ttt_equity *a, const ttt_equity *b) const {
        return a->ID<b->ID;
    }
};

class client_record {
public:
    string name="";
    int buy_count=0;
    int sell_count=0;
```

```
    int net_count=0;
};

struct comp_client_record {
    bool operator()(const client_record &a, const client_record &b) const {
        return a.name<b.name;
    }
};

////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

void do_median(map<string, equitybook> &orderAll, int current_timestamp);

void do_midpoint(map<string, equitybook> &orderAll, int current_timestamp);

void do_transfers(map<string, client_record *> &clientAll);

void do_expire(map<string, equitybook> &orderAll, set<Order *, comp_buy> *orderBuy_ptr,
               set<Order *, comp_sell> *orderSell_ptr, int current_timestamp);

void do_transact_buy(map<string, equitybook> &orderAll, map<string, client_record *> &clientA
                     map<string, equitybook>::iterator orderAll_it, int &QUANTITY, int LIMIT_
                     bool transfers, string &CLIENT_NAME, bool verbose, string &EQUITY_SYMBOL
                     int &count, int &count_transfer, int &single_commission, int &total_comm

void do_transact_sell(map<string, equitybook> &orderAll, map<string, client_record *> &client
                      map<string, equitybook>::iterator orderAll_it, int &QUANTITY, int LIMIT
                      bool transfers, string &CLIENT_NAME, bool verbose, string &EQUITY_SYMBO
                      int &count, int &count_transfer, int &single_commission, int &total_com

void do_end_of_day(int count_amount, int count, int count_transfer, int single_commission, in

////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

int main(int argc, char *argv[]) {
```

```
bool verbose=false;
bool median=false;
bool midpoint=false;
bool transfers=false;
bool ttt=false;

int ID=0;

int current_timestamp=0;
int next_ID=0;

int TIMESTAMP=0;
string CLIENT_NAME;
string BUY_OR_SELL;
bool buy_signal=false;
// bool sell_signal=false;
string EQUITY_SYMBOL;
string limit_price;
int LIMIT_PRICE=0;
string quantity;
int QUANTITY=0;
int DURATION=0;

int count_amount=0;
int count=0;
int count_transfer=0;
int single_commission=0;
int total_commission=0;

map<string, equitybook> orderAll;

set<Order *, comp_buy> *orderBuy_ptr=nullptr;
set<Order *, comp_sell> *orderSell_ptr=nullptr;

map<string, client_record *> clientAll;

set<ttt_equity *, comp_ttt_equity> tttEquity_record;
```

```
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    while(true) {
        static struct option long_options[]={{"verbose",    no_argument,       0, 'v'},
                                             {"median",     no_argument,       0, 'm'},
                                             {"midpoint",   no_argument,       0, 'p'},
                                             {"transfers",  no_argument,       0, 't'},
                                             {"ttt",        required_argument, 0, 'g'},
                                             {0, 0,                            0, 0}};
        int option_index=0;
        int c=getopt_long(argc, argv, "g:vmpt", long_options, &option_index);
        if(c==-1) {
            break;
        }
        if(c=='v') {
            verbose=true;
        }
        if(c=='m') {
            median=true;
        }
        if(c=='p') {
            midpoint=true;
        }
        if(c=='t') {
            transfers=true;
        }
        if(c=='g') {
            ttt=true;
            auto temp_equity=new ttt_equity;
            temp_equity->ID=ID;
            temp_equity->equity_symbol=optarg;
            temp_equity->timestamp_buy=-1;
            temp_equity->timestamp_sell=-1;
            temp_equity->price_buy=0;
            temp_equity->price_sell=0;
            tttEquity_record.insert(temp_equity);
            ID++;
```

```
            break;
        }
    }

    //int times_of_loop=0;

    stringstream ss;


    while(!cin.eof()) {
        string str;
        getline(cin, str);
        if(str.empty()) {
            break;
        }
        ss.clear();
        ss.str(str);
        ss>>TIMESTAMP>>CLIENT_NAME>>BUY_OR_SELL>>EQUITY_SYMBOL>>limit_price>>quantity>>DURATI
        //cout<<"this is "<<times_of_loop<<" loop"<<endl;
        //times_of_loop++;

        LIMIT_PRICE=atoi(limit_price.substr(1, limit_price.length()).c_str());
        QUANTITY=atoi(quantity.substr(1, quantity.length()).c_str());

        if(BUY_OR_SELL=="BUY") {
            buy_signal=true;
            //sell_signal=false;
        }
        else if(BUY_OR_SELL=="SELL") {
            buy_signal=false;
            //sell_signal=true;
        }
        else {
            exit(0);
        }

        auto client_record_temp=new client_record;
        client_record_temp->name=CLIENT_NAME;
```

```
        client_record_temp->buy_count=0;
        client_record_temp->sell_count=0;
        client_record_temp->net_count=0;
        clientAll.insert(make_pair(CLIENT_NAME, (client_record_temp)));



        for(auto tttEquity_record_it=tttEquity_record.begin();
            tttEquity_record_it!=tttEquity_record.end(); ++tttEquity_record_it) {

            //cout<<"ttt_equity now has the record of "<<tttEquity_record_it->ID<<endl;

            if((*tttEquity_record_it)->equity_symbol==EQUITY_SYMBOL) {
                auto tttEquity_ptr=(*tttEquity_record_it);
                if(!buy_signal) {
                    if(tttEquity_ptr->timestamp_buy==-1||tttEquity_ptr->price_buy>LIMIT_PRICE
                        tttEquity_ptr->price_buy=LIMIT_PRICE;
                        tttEquity_ptr->timestamp_buy=TIMESTAMP;

                        //cout<<"buy price = "<<tttEquity_ptr->price_buy<<endl;
                        //cout<<"buy time = "<<tttEquity_ptr->timestamp_buy<<endl;

                        tttEquity_record.insert(tttEquity_ptr);
                    }
                }
                else {
                    if(tttEquity_ptr->timestamp_buy==-1) {
                        break;
                    }
                    if(tttEquity_ptr->timestamp_sell==-1||tttEquity_ptr->price_sell<LIMIT_PRI
                        tttEquity_ptr->price_sell=LIMIT_PRICE;
                        tttEquity_ptr->timestamp_sell=TIMESTAMP;

                        //cout<<"sell price = "<<tttEquity_ptr->price_sell<<endl;
                        //cout<<"sell time = "<<tttEquity_ptr->timestamp_sell<<endl;

                        tttEquity_record.insert(tttEquity_ptr);
                    }
                }
```

```
        }
    }

    if(TIMESTAMP!=current_timestamp) {
        if(median) {
            do_median(orderAll, current_timestamp);
        }
        if(midpoint) {
            do_midpoint(orderAll, current_timestamp);
        }
        current_timestamp=TIMESTAMP;

        do_expire(orderAll, orderBuy_ptr, orderSell_ptr, current_timestamp);

    }

    auto orderAll_it=orderAll.find(EQUITY_SYMBOL);
    if(orderAll_it==orderAll.end()) {
        equitybook equitybook_temp=equitybook();
        equitybook_temp.EQUITY_SYMBOL=EQUITY_SYMBOL;
        orderAll_it=orderAll.insert(make_pair(EQUITY_SYMBOL, (equitybook_temp))).first;
    }

    if(buy_signal) {

        //cout<<"Let us do transact buy!"<<endl;

        do_transact_buy(orderAll, clientAll, current_timestamp, orderAll_it, QUANTITY, LI
                        transfers, CLIENT_NAME, verbose, EQUITY_SYMBOL, count_amount, cou
                        single_commission, total_commission);
    }
    else {

        //cout<<"Let us do transact sell!"<<endl;

        do_transact_sell(orderAll, clientAll, current_timestamp, orderAll_it, QUANTITY, L
                         transfers, CLIENT_NAME, verbose, EQUITY_SYMBOL, count_amount, co
                         single_commission, total_commission);
```

```
        }

        //cout<<boolalpha<<(orderAll_it==orderAll.end())<<endl;




        if(QUANTITY>0&&DURATION!=0) {
            auto Order_temp=new Order;
            Order_temp->ID=next_ID++;
            Order_temp->PRICE=LIMIT_PRICE;
            Order_temp->NAME=CLIENT_NAME;
            Order_temp->AMOUNT=QUANTITY;
            Order_temp->EXPIRE_TIME=(DURATION!=-1)?(current_timestamp+DURATION):-1;
            if(buy_signal) {
                orderBuy_ptr=&(orderAll_it->second.orderBuy);
                orderBuy_ptr->insert(Order_temp);
            }
            else {
                orderSell_ptr=&(orderAll_it->second.orderSell);
                orderSell_ptr->insert(Order_temp);
            }
        }
    }

    if(median) {
        do_median(orderAll, current_timestamp);
    }

    if(midpoint) {
        do_midpoint(orderAll, current_timestamp);
    }

    do_end_of_day(count_amount, count, count_transfer, single_commission, total_commission);

    if(transfers) {
        do_transfers(clientAll);
    }
```

```
    if(ttt) {
        for(auto tttEquity_record_it=tttEquity_record.begin();
            tttEquity_record_it!=tttEquity_record.end(); ++tttEquity_record_it) {
            if((*tttEquity_record_it)->timestamp_buy<0||(*tttEquity_record_it)->timestamp_sel
                continue;
            }
            else {
                cout<<"Time travelers would buy "<<(*tttEquity_record_it)->equity_symbol<<" a
                    <<(*tttEquity_record_it)->timestamp_buy<<" and sell it at time: "
                    <<(*tttEquity_record_it)->timestamp_sell<<endl;
            }
        }
    }
    return 0;
}

////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

void do_median(map<string, equitybook> &orderAll, int current_timestamp) {

    //cout<<"we are doing median!"<<endl;

    int median_price=0;
    for(auto orderAll_it=orderAll.begin(); orderAll_it!=orderAll.end(); ++orderAll_it) {
        ssize_t size=orderAll_it->second.history.size();

        //cout<<"current records: "<<orderAll_it->second.EQUITY_SYMBOL<<endl;
        //cout<<"current trade number = "<<size<<endl;

        if(size!=0) {
            bool even=(size%2==0);
            size/=2;
            auto median_price_it=orderAll_it->second.history.begin();
            for(auto i=0; i<size; ++i) {
                ++median_price_it;
            }
```

```
            if(!even) {
                median_price=*median_price_it;
            }
            else {
                median_price=((*median_price_it)+*(--median_price_it))/2;
            }
            cout<<"Median match price of "<<orderAll_it->second.EQUITY_SYMBOL<<" at time "<<c
                <<median_price<<endl;
        }
        else {
            continue;
        }
    }
}

void do_midpoint(map<string, equitybook> &orderAll, int current_timestamp) {

    //cout<<"we are doing midpoint!"<<endl;

    //cout<<"current time = "<<current_timestamp<<endl;

    //string equity_symbol_temp=orderAll.begin()->second.EQUITY_SYMBOL;

    for(auto orderAll_it=orderAll.begin(); orderAll_it!=orderAll.end(); ++orderAll_it) {

        /*
        cout<<orderAll_it->second.EQUITY_SYMBOL<<endl;

        if(orderAll_it->second.orderBuy.empty()) {
            cout<<"no buy record!"<<endl;
        }
        else {
            cout<<"buy price = "<<(*orderAll_it->second.orderBuy.begin()).PRICE<<endl;
        }

        if(orderAll_it->second.orderSell.empty()) {
            cout<<"no sell record!"<<endl;
        }
```

```
        else {
            cout<<"sell price = "<<(*orderAll_it->second.orderSell.begin()).PRICE<<endl;
        }
         */

        if(orderAll_it->second.orderBuy.empty()||orderAll_it->second.orderSell.empty()) {
            cout<<"Midpoint of "<<orderAll_it->second.EQUITY_SYMBOL<<" at time "<<current_time
                <<endl;
            continue;
        }

        auto midpoint_price=
                ((*orderAll_it->second.orderBuy.begin())->PRICE+(*orderAll_it->second.orderSe
        cout<<"Midpoint of "<<orderAll_it->second.EQUITY_SYMBOL<<" at time "<<current_timestan
            <<endl;
    }
}

void do_transfers(map<string, client_record *> &clientAll) {
    for(auto clientAll_it=clientAll.begin(); clientAll_it!=clientAll.end(); ++clientAll_it) {
        cout<<clientAll_it->second->name<<" bought "<<clientAll_it->second->buy_count<<" and
            <<clientAll_it->second->sell_count<<" for a net transfer of $"<<clientAll_it->sec
    }
}

void do_expire(map<string, equitybook> &orderAll, set<Order *, comp_buy> *orderBuy_ptr,
               set<Order *, comp_sell> *orderSell_ptr, int current_timestamp) {
    set<Order *, comp_sell>::iterator orderSell_it;
    set<Order *, comp_sell>::iterator orderSell_it_temp;
    set<Order *, comp_buy>::iterator orderBuy_it;
    set<Order *, comp_sell>::iterator orderBuy_it_temp;

    //int loop=0;

    for(auto orderAll_it=orderAll.begin(); orderAll_it!=orderAll.end(); ++orderAll_it) {

        orderSell_ptr=&(orderAll_it->second.orderSell);
```

```
        for(orderSell_it=orderSell_ptr->begin(); orderSell_it!=orderSell_ptr->end();) {

            //cout<<"loop number = "<<loop<<endl;

            if((*orderSell_it)->EXPIRE_TIME!=-1&&(*orderSell_it)->EXPIRE_TIME<=current_timesta
                orderSell_it_temp=orderSell_it;
                //cout<<"point to "<<(*orderSell_ptr->begin()).NAME<<endl;
                orderSell_it=orderSell_ptr->erase(orderSell_it_temp);
                //loop++;
                //cout<<"do I come here 1?"<<endl;
            }
            else {
                ++orderSell_it;
            }

        }

        orderBuy_ptr=&(orderAll_it->second.orderBuy);

        for(orderBuy_it=orderBuy_ptr->begin(); orderBuy_it!=orderBuy_ptr->end();) { // bug he

            //cout<<"loop number = "<<loop<<endl;

            if((*orderBuy_it)->EXPIRE_TIME!=-1&&(*orderBuy_it)->EXPIRE_TIME<=current_timestam
                orderBuy_it_temp=orderBuy_it;
                //cout<<"point to "<<(*orderBuy_ptr->begin()).NAME<<endl;
                orderBuy_it=orderBuy_ptr->erase(orderBuy_it_temp);
                //loop++;
                //cout<<"do I come here 1?"<<endl;
            }
            else {
                ++orderBuy_it;
            }
        }
    }
}

void do_transact_buy(map<string, equitybook> &orderAll, map<string, client_record *> &clientA
```

```
                map<string, equitybook>::iterator orderAll_it, int &QUANTITY, int LIMIT_
                bool transfers, string &CLIENT_NAME, bool verbose, string &EQUITY_SYMBOL
                int &count, int &count_transfer, int &single_commission, int &total_comm
auto transact_price=0;

auto orderSell_ptr=&(orderAll_it->second.orderSell);

while(QUANTITY>0&&!orderSell_ptr->empty()) {

    //cout<<"others willing to sell "<<(*orderSell_ptr->begin()).NAME<<" at "<<(*orderSe
    //cout<<"I am willing to pay "<<LIMIT_PRICE<<endl;

    if((*orderSell_ptr->begin())->PRICE<=LIMIT_PRICE) {
        auto Order_ptr=*orderSell_ptr->begin();
        if(Order_ptr->ID>=next_ID) {
            transact_price=LIMIT_PRICE;
        }
        else {
            transact_price=Order_ptr->PRICE;
        }

        auto equitybook_ptr=&(orderAll_it->second);
        equitybook_ptr->history.insert(transact_price);

        if(Order_ptr->AMOUNT>QUANTITY) {
            if(transfers) {
                auto clientAll_it_1=clientAll.find(CLIENT_NAME);
                bool find_buyer=!(clientAll_it_1==clientAll.end());

                if(!find_buyer) {
                    auto client_record_temp=new client_record;
                    client_record_temp->name=CLIENT_NAME;
                    client_record_temp->buy_count=QUANTITY;
                    client_record_temp->sell_count=0;
                    client_record_temp->net_count=QUANTITY*transact_price*(-1);
                    clientAll.insert(make_pair(CLIENT_NAME, (client_record_temp)));
                }
                else {
```

```
                auto clientAll_ptr_1=(clientAll_it_1->second);
                clientAll_ptr_1->buy_count+=QUANTITY;
                clientAll_ptr_1->net_count+=QUANTITY*transact_price*(-1);
            }

            auto clientAll_it_2=clientAll.find(Order_ptr->NAME);
            bool find_seller=!(clientAll_it_2==clientAll.end());

            if(!find_seller) {
                auto client_record_temp=new client_record;
                client_record_temp->name=Order_ptr->NAME;
                client_record_temp->buy_count=0;
                client_record_temp->sell_count=QUANTITY;
                client_record_temp->net_count=QUANTITY*transact_price;
                clientAll.insert(make_pair(Order_ptr->NAME, (client_record_temp)));
            }
            else {
                auto clientAll_ptr_2=(clientAll_it_2->second);
                clientAll_ptr_2->sell_count+=QUANTITY;
                clientAll_ptr_2->net_count+=QUANTITY*transact_price;
            }
            //find_buyer=false;
            //find_seller=false;
        }
        if(verbose) {
            cout<<CLIENT_NAME<<" purchased "<<QUANTITY<<" shares of "<<EQUITY_SYMBOL<
                <<" for $"<<transact_price<<"/share"<<endl;
        }
        ++count;
        count_amount+=QUANTITY;
        count_transfer+=transact_price*QUANTITY;
        single_commission=(transact_price*QUANTITY)/100;
        total_commission+=single_commission*2;
        Order_ptr->AMOUNT-=QUANTITY;
        QUANTITY=0;
    }
    else if(Order_ptr->AMOUNT==QUANTITY) {
        if(transfers) {
```

```
            auto clientAll_it_1=clientAll.find(CLIENT_NAME);
            bool find_buyer=!(clientAll_it_1==clientAll.end());

            if(!find_buyer) {
                auto client_record_temp=new client_record;
                client_record_temp->name=CLIENT_NAME;
                client_record_temp->buy_count=QUANTITY;
                client_record_temp->sell_count=0;
                client_record_temp->net_count=QUANTITY*transact_price*(-1);
                clientAll.insert(make_pair(CLIENT_NAME, (client_record_temp)));
            }
            else {
                auto clientAll_ptr_1=(clientAll_it_1->second);
                clientAll_ptr_1->buy_count+=QUANTITY;
                clientAll_ptr_1->net_count+=QUANTITY*transact_price*(-1);
            }

            auto clientAll_it_2=clientAll.find(Order_ptr->NAME);
            bool find_seller=!(clientAll_it_2==clientAll.end());

            if(!find_seller) {
                auto client_record_temp=new client_record;
                client_record_temp->name=Order_ptr->NAME;
                client_record_temp->buy_count=0;
                client_record_temp->sell_count=QUANTITY;
                client_record_temp->net_count=QUANTITY*transact_price;
                clientAll.insert(make_pair(Order_ptr->NAME, (client_record_temp)));
            }
            else {
                auto clientAll_ptr_2=(clientAll_it_2->second);
                clientAll_ptr_2->sell_count+=QUANTITY;
                clientAll_ptr_2->net_count+=QUANTITY*transact_price;
            }
            //find_buyer=false;
            //find_seller=false;
        }
        if(verbose) {
            cout<<CLIENT_NAME<<" purchased "<<QUANTITY<<" shares of "<<EQUITY_SYMBOL<
```

```
                    <<" for $"<<transact_price<<"/share"<<endl;
                }
                ++count;
                count_amount+=QUANTITY;
                count_transfer+=transact_price*QUANTITY;
                single_commission=(transact_price*QUANTITY)/100;
                total_commission+=single_commission*2;
                QUANTITY=0;
                orderSell_ptr->erase(orderSell_ptr->begin());
            }
            else {
                if(transfers) {
                    auto clientAll_it_1=clientAll.find(CLIENT_NAME);
                    bool find_buyer=!(clientAll_it_1==clientAll.end());

                    if(!find_buyer) {
                        auto client_record_temp=new client_record;
                        client_record_temp->name=CLIENT_NAME;
                        client_record_temp->buy_count=Order_ptr->AMOUNT;
                        client_record_temp->sell_count=0;
                        client_record_temp->net_count=Order_ptr->AMOUNT*transact_price*(-1);
                        clientAll.insert(make_pair(CLIENT_NAME, (client_record_temp)));
                    }
                    else {
                        auto clientAll_ptr=(clientAll_it_1->second);
                        clientAll_ptr->buy_count+=Order_ptr->AMOUNT;
                        clientAll_ptr->net_count+=Order_ptr->AMOUNT*transact_price*(-1);
                    }

                    auto clientAll_it_2=clientAll.find(Order_ptr->NAME);
                    bool find_seller=!(clientAll_it_2==clientAll.end());

                    if(!find_seller) {
                        auto client_record_temp=new client_record;
                        client_record_temp->name=Order_ptr->NAME;
                        client_record_temp->buy_count=0;
                        client_record_temp->sell_count=Order_ptr->AMOUNT;
                        client_record_temp->net_count=Order_ptr->AMOUNT*transact_price;
```

```
                        clientAll.insert(make_pair(Order_ptr->NAME, (client_record_temp)));
                    }
                    else {
                        auto clientAll_ptr_2=(clientAll_it_2->second);
                        clientAll_ptr_2->sell_count+=Order_ptr->AMOUNT;
                        clientAll_ptr_2->net_count+=Order_ptr->AMOUNT*transact_price;
                    }
                    //find_buyer=false;
                    //find_seller=false;
                }
                if(verbose) {
                    cout<<CLIENT_NAME<<" purchased "<<Order_ptr->AMOUNT<<" shares of "<<EQUIT
                        <<Order_ptr->NAME<<" for $"<<transact_price<<"/share"<<endl;
                }
                count_amount+=Order_ptr->AMOUNT;
                ++count;
                count_transfer+=transact_price*Order_ptr->AMOUNT;
                single_commission=transact_price*Order_ptr->AMOUNT/100;
                total_commission+=single_commission*2;
                QUANTITY-=Order_ptr->AMOUNT;
                orderSell_ptr->erase(orderSell_ptr->begin());
            }
        }
        else {

            //cout<<"trade failed!"<<endl;

            break;
        }
    }

}

void do_transact_sell(map<string, equitybook> &orderAll, map<string, client_record *> &client
                      map<string, equitybook>::iterator orderAll_it, int &QUANTITY, int LIMIT
                      bool transfers, string &CLIENT_NAME, bool verbose, string &EQUITY_SYMBO
                      int &count, int &count_transfer, int &single_commission, int &total_com
    auto transact_price=0;
```

```
auto orderBuy_ptr=&(orderAll_it->second.orderBuy);

/*
auto orderSell_ptr=&(orderAll_it->second.orderSell);

cout<<(*orderSell_ptr->begin())->NAME<<endl;
 */


/*
 if(orderBuy_ptr->empty()) {
     auto it_temp=orderSell_ptr->begin();
     auto Order_ptr=*it_temp;
     while(it_temp!=orderSell_ptr->end()) {
         auto client_record_temp=new client_record;
         client_record_temp->name=Order_ptr->NAME;
         client_record_temp->buy_count=0;
         client_record_temp->sell_count=Order_ptr->AMOUNT;
         client_record_temp->net_count=QUANTITY*transact_price*(1);
         clientAll.insert(make_pair(Order_ptr->NAME, (client_record_temp)));
         it_temp++;
     }
}
 */

while(QUANTITY>0&&!orderBuy_ptr->empty()) {

    //cout<<"others willing to buy "<<(*orderBuy_ptr->begin()).NAME<<" at "<<(*orderBuy_p
    //cout<<"I am willing to sell "<<LIMIT_PRICE<<endl;

    if((*orderBuy_ptr->begin())->PRICE>=LIMIT_PRICE) {
        auto Order_ptr=*orderBuy_ptr->begin();
        if(Order_ptr->ID>=next_ID) {
            transact_price=LIMIT_PRICE;
        }
        else {
            transact_price=Order_ptr->PRICE;
```

```
            }
            auto equitybook_ptr=&(orderAll_it->second);
            equitybook_ptr->history.insert(transact_price);
            if(Order_ptr->AMOUNT>QUANTITY) {
                if(transfers) {
                    auto clientAll_it_1=clientAll.find(CLIENT_NAME);
                    bool find_seller=!(clientAll_it_1==clientAll.end());

                    if(!find_seller) {
                        auto client_record_temp=new client_record;
                        client_record_temp->name=CLIENT_NAME;
                        client_record_temp->buy_count=0;
                        client_record_temp->sell_count=QUANTITY;
                        client_record_temp->net_count=QUANTITY*transact_price*(1);
                        clientAll.insert(make_pair(CLIENT_NAME, (client_record_temp)));
                    }
                    else {
                        auto clientAll_ptr_1=(clientAll_it_1->second);
                        clientAll_ptr_1->sell_count+=QUANTITY;
                        clientAll_ptr_1->net_count+=QUANTITY*transact_price*(1);
                    }

                    auto clientAll_it_2=clientAll.find(Order_ptr->NAME);
                    bool find_buyer=!(clientAll_it_2==clientAll.end());

                    if(!find_buyer) {
                        auto client_record_temp=new client_record;
                        client_record_temp->name=Order_ptr->NAME;
                        client_record_temp->buy_count=QUANTITY;
                        client_record_temp->sell_count=0;
                        client_record_temp->net_count=QUANTITY*transact_price*(-1);
                        clientAll.insert(make_pair(Order_ptr->NAME, (client_record_temp)));
                    }
                    else {
                        auto clientAll_ptr_2=(clientAll_it_2->second);
                        clientAll_ptr_2->buy_count+=QUANTITY;
                        clientAll_ptr_2->net_count+=QUANTITY*transact_price*(-1);
                    }
```

```
                //find_buyer=false;
                //find_seller=false;

                //Order_ptr->AMOUNT-=QUANTITY;

            }
            if(verbose) {
                //do_verbose_sell(Order_ptr, EQUITY_SYMBOL, CLIENT_NAME, transact_price);
                cout<<Order_ptr->NAME<<" purchased "<<QUANTITY<<" shares of "<<EQUITY_SYM
                    <<" for $"<<transact_price<<"/share"<<endl;
            }
            count_amount+=QUANTITY;
            ++count;
            count_transfer+=transact_price*QUANTITY;
            single_commission=transact_price*QUANTITY/100;
            total_commission+=single_commission*2;
            Order_ptr->AMOUNT-=QUANTITY;
            QUANTITY=0;
        }
        else if(Order_ptr->AMOUNT==QUANTITY) {
            if(transfers) {
                auto clientAll_it_1=clientAll.find(CLIENT_NAME);
                bool find_seller=!(clientAll_it_1==clientAll.end());


                if(!find_seller) {
                    auto client_record_temp=new client_record;
                    client_record_temp->name=CLIENT_NAME;
                    client_record_temp->buy_count=0;
                    client_record_temp->sell_count=QUANTITY;
                    client_record_temp->net_count=QUANTITY*transact_price*(1);
                    clientAll.insert(make_pair(CLIENT_NAME, (client_record_temp)));
                }
                else {
                    auto clientAll_ptr_1=(clientAll_it_1->second);
                    clientAll_ptr_1->sell_count+=QUANTITY;
                    clientAll_ptr_1->net_count+=QUANTITY*transact_price*(1);
                }
```

```
            auto clientAll_it_2=clientAll.find(Order_ptr->NAME);
            bool find_buyer=!(clientAll_it_2==clientAll.end());


            if(!find_buyer) {
                auto client_record_temp=new client_record;
                client_record_temp->name=Order_ptr->NAME;
                client_record_temp->buy_count=QUANTITY;
                client_record_temp->sell_count=0;
                client_record_temp->net_count=QUANTITY*transact_price*(-1);
                clientAll.insert(make_pair(Order_ptr->NAME, (client_record_temp)));
            }
            else {
                auto clientAll_ptr_2=(clientAll_it_2->second);
                clientAll_ptr_2->buy_count+=QUANTITY;
                clientAll_ptr_2->net_count+=QUANTITY*transact_price*(-1);
            }
            //find_buyer=false;
            //find_seller=false;
        }
        if(verbose) {

            cout<<Order_ptr->NAME<<" purchased "<<QUANTITY<<" shares of "<<EQUITY_SYM
                <<" for $"<<transact_price<<"/share"<<endl;
        }
        count_amount+=QUANTITY;
        ++count;
        count_transfer+=transact_price*QUANTITY;
        single_commission=transact_price*QUANTITY/100;
        total_commission+=single_commission*2;
        QUANTITY=0;
        orderBuy_ptr->erase(orderBuy_ptr->begin());
    }
    else {
        if(transfers) {
            auto clientAll_it_1=clientAll.find(CLIENT_NAME);
            bool find_seller=!(clientAll_it_1==clientAll.end());
```

```
            if(!find_seller) {
                auto client_record_temp=new client_record;
                client_record_temp->name=CLIENT_NAME;
                client_record_temp->buy_count=0;
                client_record_temp->sell_count=Order_ptr->AMOUNT;
                client_record_temp->net_count=Order_ptr->AMOUNT*transact_price*(1);
                clientAll.insert(make_pair(CLIENT_NAME, (client_record_temp)));
            }
            else {
                auto clientAll_ptr_1=(clientAll_it_1->second);
                clientAll_ptr_1->sell_count+=Order_ptr->AMOUNT;
                clientAll_ptr_1->net_count+=Order_ptr->AMOUNT*transact_price*(1);
            }

            auto clientAll_it_2=clientAll.find(Order_ptr->NAME);
            bool find_buyer=!(clientAll_it_2==clientAll.end());

            if(!find_buyer) {
                auto client_record_temp=new client_record;
                client_record_temp->name=Order_ptr->NAME;
                client_record_temp->buy_count=Order_ptr->AMOUNT;
                client_record_temp->sell_count=0;
                client_record_temp->net_count=Order_ptr->AMOUNT*transact_price*(-1);
                clientAll.insert(make_pair(Order_ptr->NAME, (client_record_temp)));
            }
            else {
                auto clientAll_ptr_2=(clientAll_it_2->second);
                clientAll_ptr_2->buy_count+=Order_ptr->AMOUNT;
                clientAll_ptr_2->net_count+=Order_ptr->AMOUNT*transact_price*(-1);
            }
            //find_buyer=false;
            //find_seller=false;
        }
        if(verbose) {
            cout<<Order_ptr->NAME<<" purchased "<<Order_ptr->AMOUNT<<" shares of "<<E
                <<CLIENT_NAME<<" for $"<<transact_price<<"/share"<<endl;
        }
```

```
                count_amount+=Order_ptr->AMOUNT;
                ++count;
                count_transfer+=transact_price*Order_ptr->AMOUNT;
                single_commission=transact_price*Order_ptr->AMOUNT/100;
                total_commission+=single_commission*2;
                QUANTITY-=Order_ptr->AMOUNT;
                orderBuy_ptr->erase(orderBuy_ptr->begin());
            }
        }
        else {

            //cout<<"trade failed!"<<endl;

            break;
        }
    }

}

void do_end_of_day(int count_amount, int count, int count_transfer, int single_commission, in
    cout<<"---End of Day---"<<endl;
    cout<<"Commission Earnings: $"<<total_commission<<endl;
    cout<<"Total Amount of Money Transferred: $"<<count_transfer<<endl;
    cout<<"Number of Completed Trades: "<<count<<endl;
    cout<<"Number of Shares Traded: "<<count_amount<<endl;
}
```