UM-SJTU JOINT INSTITUTE
Data Structures and Algorithms
(VP281)

Programming Assignment

Programming Assignment Four
Electronic Trading

Name: Pan Chongdan
ID: 516370910121

Date: November 30, 2018

1 Introduction

The programming assignment asks us to write a program to help facilitate the trading of equities on an electronic exchange market.

2 Code Appendix

The appendix shows the cpp code for main project and time comparison. The following part is for the main algorithm

```cpp
#ifndef MARKET_H
#define MARKET_H
#include <iostream>
#include <sstream>
#include <string>
#include <map>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;
class client{
    public:
        client(string n,int b,int s){
            name=n;
            buyamount=0;
            sellamount=0;
            net=0;
        }
        string name;
        int buyamount;
        int sellamount;
        int net;
};
class order{
    public:
        string name;
        int expiretime;
        int price;
        int quantity;
        int id;
        int duration;

        order(string n,int E,int P,int Q,int D,int ID){
            name=n;
            expiretime=E;
            price=P;
            quantity=Q;
            duration=D;
            id=ID;
            if(D==-1)expiretime=-1;
```

```
            if(D==-1)expiretime=-1;
        };
        struct buyer_order_compare{
            bool operator()(order* a,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 seller_order_compare{
            bool operator()(order* a,order* b)const{
                if(a->price>b->price)return true;
                else if(a->price==b->price)return a->id>b->id;
                else return false;
            }
        };
};
class stock{
    public:
        vector<int>tradeprice;
        string name;
        int ttt=0,selltime,sellprice,buyprice,buytime,init=0;
        priority_queue<order*,vector<order*>,order::buyer_order_compare> buyer_orders;
        priority_queue<order*,vector<order*>,order::seller_order_compare> seller_orders;
        stock(string n,int T,int i){
            name=n;
            ttt=T;
            selltime=-1;
            sellprice=-1;
            buytime=-1;
            buyprice=-1;
            init=i;
        }
        ~stock(){
            while(!buyer_orders.empty()){
                auto victim=buyer_orders.top();
                delete victim;
                buyer_orders.pop();
            }
            while(!seller_orders.empty()){
```

```
80                  auto victim=seller_orders.top();
81                  delete victim;
82                  seller_orders.pop();
83              }
84          }
85          void addorder(order* newer,string buy_or_sell){
86              if(buy_or_sell=="BUY")buyer_orders.push(newer);
87              else seller_orders.push(newer);
88          };
89  };
90  class market{
91      private:
92          bool verbose=0;
93          bool median=0;
94          bool midpoint=0;
95          bool transfers=0;
96          int current_timestamp=0;
97          int ID=0;
98          int earning=0;
99          int transfer=0;
100         int shareamount=0;
101         int transfernumber=0;
102         map<string, stock*>stocks;
103         map<string, client>clients;
104         vector<string>ttt;
105     public:
106         market(bool v, bool m, bool p, bool t,vector<string>T){
107             verbose=v;
108             median=m;
109             midpoint=p;
110             transfers=t;
111             ID=0;
112             earning=0;
113             transfer=0;
114             shareamount=0;
115             transfernumber=0;
116             ttt=T;
117             for(auto it=ttt.begin();it!=ttt.end();++it){
118                 auto newstock=new stock(*it,1,0);
```

The following is for binary heap.

```
119                 stocks.insert({*it,newstock});
120             }
121         };
122         ~market(){
123 
124         };
125         void trade(order*buyer_order,order*seller_order,int amount,string equity_symbol,bool v,stock*
126             int price;
127             if(buyer_order->id<seller_order->id)price=buyer_order->price;
128             else price=seller_order->price;
129 
130             buyer_order->quantity-=amount;
131             seller_order->quantity-=amount;
132             if(v)cout<<buyer_order->name<<" purchased "<<amount<<" shares of "<<equity_symbol<<" from
133             shareamount+=amount;
134             transfer+=amount*price;
135             earning+=(amount*price)/100+(amount*price)/100;
136             transfernumber++;
137             auto clientit=clients.find(buyer_order->name);
138             clientit->second.buyamount+=amount;
139             clientit->second.net-=amount*price;
140             clientit=clients.find(seller_order->name);
141             clientit->second.sellamount+=amount;
142             clientit->second.net+=amount*price;
143             tmp->tradeprice.push_back(price);//determine the median
144 
145         };
146 
147         void work(string &inputbook){
148             stringstream input;
149 
150             int timestamp,price,quantity,duration,amount;
151             string client_name,buy_or_sell,equity_symbol,Price,Quantity;
152 
153             input.clear();
154             input.str(inputbook);
155             input>>timestamp>>client_name>>buy_or_sell>>equity_symbol>>Price>>Quantity>>duration;
156             price=strtol(Price.c_str()+1,NULL,10);
157             quantity=strtol(Quantity.c_str()+1,NULL,10);
```

```
158
159
160
161             auto newer=new order(client_name,timestamp+duration,price,quantity,duration,ID);//define t
162             auto stockit=stocks.find(equity_symbol);
163
164             client newclient(client_name,0,0);
165             auto clientit=clients.find(client_name);
166             if(clientit==clients.end())clients.insert({client_name, newclient});//add clients
167             //matching
168
169             if(timestamp!=current_timestamp)print(timestamp);
170
171             if(stockit!=stocks.end()){//the stock exists, try to match the order first
172                 stock* tmp=stockit->second;
173                 order* seller_order;
174                 order* buyer_order;
175                 tmp->init=1;
176
177
178                 if(buy_or_sell=="BUY"){//find the seller_order
179
180                     if(newer->price>tmp->sellprice||tmp->sellprice==-1){
181                         tmp->sellprice=newer->price;
182                         tmp->selltime=timestamp;
183
184                     }
185                     while(!tmp->seller_orders.empty()){
186                         seller_order=tmp->seller_orders.top();
187                         if(seller_order->expiretime<=current_timestamp&&seller_order->expiretime!=-1){
188                             tmp->seller_orders.pop();
189                             delete seller_order;
190                             continue;
191                         }
192                         if(seller_order->price>newer->price)break;//the trade won't happen
193                         else {
194
195                             if(seller_order->quantity<newer->quantity){//buyer_order wants more
196                                 amount=seller_order->quantity;
```

```
197                                 tmp->seller_orders.pop();
198                                 /*while(1){
199                                     if(tmp->seller_orders.empty())break;
200                                     auto top=tmp->seller_orders.top();
201                                     if(top->expiretime<=current_timestamp&&top->expiretime!=-1){
202                                         tmp->seller_orders.pop();
203                                         delete top;
204                                         continue;
205                                     }
206                                     if(top->name==seller_order->name&&top->price==seller_order->price)
207                                         if(top->quantity+amount<=newer->quantity){
208                                             amount=amount+top->quantity;
209                                             tmp->seller_orders.pop();
210                                             delete top;
211                                             continue;
212                                         }
213                                         else {
214                                             top->quantity=top->quantity-newer->quantity+amount;
215                                             amount=newer->quantity;
216                                             break;
217                                         }
218                                     }
219                                     else break;
220                                 }*/
221                                 trade(newer,seller_order,amount,tmp->name,verbose,tmp);
222                                 delete seller_order;
223                             }
224                             else {//buyer_order wants less
225                                 amount=newer->quantity;
226                                 trade(newer,seller_order,amount,tmp->name,verbose,tmp);
227                                 if(seller_order->quantity==newer->quantity){
228                                     tmp->seller_orders.pop();
229                                     delete seller_order;
230                                 }
231                             }
232                         }
233                         if(newer->quantity==0)break;
234                     }
235                     if(newer->quantity!=0&&newer->duration!=0)tmp->addorder(newer,buy_or_sell); //read
```

The following is for unsorted heap.

```
236                 else delete newer;
237
238             }
239             else {//find the buyer_order
240
241                 if(newer->price<tmp->buyprice||tmp->buyprice==-1){
242                     tmp->buyprice=newer->price;
243                     tmp->buytime=timestamp;
244                 }
245
246                 while(!tmp->buyer_orders.empty()){
247                     buyer_order=tmp->buyer_orders.top();
248                     if(buyer_order->expiretime<=current_timestamp&&buyer_order->expiretime!=-1){
249                         tmp->buyer_orders.pop();
250                         delete buyer_order;
251                         continue;
252                     }
253                     if(buyer_order->price<newer->price)break;
254                     else {
255                         if(buyer_order->quantity<newer->quantity){//buyer_order wants less
256                             amount=buyer_order->quantity;
257                             tmp->buyer_orders.pop();
258                             /*while(1){
259                                 if(tmp->buyer_orders.empty())break;
260                                 auto top=tmp->buyer_orders.top();
261                                 if(top->expiretime<=current_timestamp&&top->expiretime!=-1){
262                                     tmp->buyer_orders.pop();
263                                     delete top;
264                                     continue;
265                                 }
266                                 if(top->name==buyer_order->name&&top->price==buyer_order->price){
267                                     if(top->quantity+amount<=newer->quantity){
268                                         amount=amount+top->quantity;
269                                         tmp->buyer_orders.pop();
270                                         delete top;
271                                         continue;
272                                     }
273                                     else {
274                                         top->quantity=top->quantity-newer->quantity+amount;
```

```
                                    amount=newer->quantity;
                                    break;
                                }
                            }
                            else break;
                        }*/
                        trade(buyer_order,newer,amount,tmp->name,verbose,tmp);
                        delete buyer_order;
                    }
                    else {//buyer_order wants more
                        amount=newer->quantity;
                        trade(buyer_order,newer,amount,tmp->name,verbose,tmp);
                        if(buyer_order->quantity==newer->quantity){
                            tmp->buyer_orders.pop();
                            delete buyer_order;

                        }

                    }
                }
                if(newer->quantity==0)break;
            }
            if(newer->quantity!=0&&newer->duration!=0)tmp->addorder(newer,buy_or_sell); //read
            else delete newer;
        }
    }
    else {//it can't be matched so it need to be inserted into the book
        stock* newstock;
        if(find(ttt.begin(),ttt.end(),equity_symbol)!=ttt.end())newstock=new stock(equity_symb
        else newstock=new stock(equity_symbol,0,1);
        stocks.insert({equity_symbol,newstock});
        if(buy_or_sell=="BUY"){
            newstock->sellprice=newer->price;
            newstock->selltime=timestamp;
        }
        else {
            newstock->buyprice=newer->price;
            newstock->buytime=timestamp;
        }
```

```
314                 if(newer->duration!=0)newstock->addorder(newer,buy_or_sell);//read the order
315                 else delete newer;
316             }
317             ID++;
318         };
319         void print(int timestamp){
320             int medianprice;
321             if(median)for(auto it=stocks.begin();it!=stocks.end();++it){
322                 int n=it->second->tradeprice.size();
323                 if(n==0||it->second->init==0)continue;
324                 else {
325                     sort(it->second->tradeprice.begin(),it->second->tradeprice.end());
326                     if(n%2==1)medianprice=it->second->tradeprice[n/2];
327                     else medianprice=(it->second->tradeprice[n/2]+it->second->tradeprice[n/2-1])/2;
328                 }
329                 cout<<"Median match price of "<<it->second->name<<" at time "<<current_timestamp<<" is
330             }
331             if(midpoint)for(auto it=stocks.begin();it!=stocks.end();++it){
332                 if(it->second->init==0)continue;
333                 int highest,lowest;
334                 cout<<"Midpoint of "<<it->second->name<<" at time "<<current_timestamp<<" is ";
335                 if(it->second->buyer_orders.empty())highest=-1;
336                 else highest=it->second->buyer_orders.top()->price;
337 
338                 if(it->second->seller_orders.empty())lowest=-1;
339                 else lowest=it->second->seller_orders.top()->price;
340                 if(highest==-1||lowest==-1)cout<<"undefined"<<endl;
341                 else cout<<"$"<<(highest+lowest)/2<<endl;
342 
343             }
344 
345             current_timestamp=timestamp;
346             for(auto it=stocks.begin();it!=stocks.end();++it){//delete expired order
347                 while(!it->second->buyer_orders.empty()){
348                     auto buyer_order=it->second->buyer_orders.top();
349                     if(buyer_order->expiretime<=current_timestamp&&buyer_order->expiretime!=-1){
350                         it->second->buyer_orders.pop();
351                         delete buyer_order;
352                     }
```

The following is for fibonacci heap.

```
353                     else break;
354                 }
355                 while(!it->second->seller_orders.empty()){
356                     auto seller_order=it->second->seller_orders.top();
357                     if(seller_order->expiretime<=current_timestamp&&seller_order->expiretime!=-1){
358                         it->second->seller_orders.pop();
359                         delete seller_order;
360                     }
361                     else break;
362                 }
363             }
364         }
365         void summary(){
366             for(auto it=stocks.begin();it!=stocks.end();++it){//delete expired order
367                 while(!it->second->buyer_orders.empty()){
368                     auto buyer_order=it->second->buyer_orders.top();
369                     if(buyer_order->expiretime<=current_timestamp&&buyer_order->expiretime!=-1){
370                         it->second->buyer_orders.pop();
371                         delete buyer_order;
372                     }
373                     else break;
374                 }
375                 while(!it->second->seller_orders.empty()){
376                     auto seller_order=it->second->seller_orders.top();
377                     if(seller_order->expiretime<=current_timestamp&&seller_order->expiretime!=-1){
378                         it->second->seller_orders.pop();
379                         delete seller_order;
380                     }
381                     else break;
382                 }
383             }
384             print(current_timestamp);
385             cout<<"---End of Day---"<<endl;
386             cout<<"Commission Earnings: $"<<earning<<endl;
387             cout<<"Total Amount of Money Transferred: $"<<transfer<<endl;
388             cout<<"Number of Completed Trades: "<<transfernumber<<endl;
389             cout<<"Number of Shares Traded: "<<shareamount<<endl;
390             for(auto it=clients.begin();it!=clients.end();++it){
391                 if(transfers)cout<<it->second.name<<" bought "<<it->second.buyamount<<" and sold "<<it
```

```
392
393             }
394             for(auto it=stocks.begin();it!=stocks.end();++it){
395                 if(it->second->ttt==1)cout<<"Time travelers would buy "<<it->second->name<<" at time:
396                 delete it->second;
397             }
398
399         };
400 };
401 #endif
```