**Ve 281 Project 4**

Dong Jing

515370910182

main.h

#ifndef PROJECT\_4\_MARKET\_H

#define PROJECT\_4\_MARKET\_H

struct client{

int time, type, price, quantity, duration, order;

std::string name, symbol;

};

struct compare{

bool operator()(client a, client b)

{

if (a.type==0)//buy

if (a.price<b.price) return true;

else if (a.price>b.price) return false;

else return a.order>b.order;

else if (a.price>b.price) return true;

else if (a.price<b.price) return false;

else return a.order>b.order;

}

};

struct transfer{

int number\_buy, number\_sell, net;

};

struct traveler{

std::string symbol;

int buy\_time, sell\_time, buy\_price, sell\_price, order, lowest\_time, lowest\_price;

};

typedef std::priority\_queue<client, std::vector<client>, compare> client\_heap;

typedef std::map<std::string, client\_heap> type\_map;

typedef std::map<std::string, std::priority\_queue<int> > trade\_max;

typedef std::map<std::string, std::priority\_queue<int, std::vector<int>, std::greater<int>>> trade\_min;

typedef std::map<std::string, transfer> map\_transfer;

typedef std::map<int, traveler> map\_traveler;

typedef std::map<std::string, traveler> map\_string\_traveler;

#endif // PROJECT\_4\_MARKET\_H

main.cpp

#include <iostream>

#include <unordered\_map>

#include <string>

#include <sstream>

#include <queue>

#include <map>

#include <getopt.h>

#include <ctime>

#include <set>

#include "main.h"

using namespace std;

void Median(trade\_max &m1, trade\_min &m2, int current\_time)

{

auto it1=m1.begin();

auto it2=m2.begin();

int median=-1;

while (it1!=m1.end())

{

median=-1;

if ((!it1->second.empty())||(!it2->second.empty()))

if (it1->second.size()==it2->second.size())

median=(it1->second.top()+it2->second.top())/2;

else if (it1->second.size()<it2->second.size())

median=it2->second.top();

else

median=it1->second.top();

if (median!=-1)

cout<<"Median match price of "<<it1->first<<" at time "<<current\_time<<" is $"<<median<<endl;

it1++;

it2++;

}

}

void Midpoint(type\_map &buy, type\_map &sell, int current\_time)

{

auto it1=buy.begin();

auto it2=sell.begin();

int midpoint=-1;

while (it1!=buy.end())

{

while (!it1->second.empty())

{

if (it1->second.top().duration==-1) break;

if (it1->second.top().time+it1->second.top().duration>current\_time) break;

it1->second.pop();

}

while (!it2->second.empty())

{

if (it2->second.top().duration==-1) break;

if (it2->second.top().time+it2->second.top().duration>current\_time) break;

it2->second.pop();

}

if (!it1->second.empty() && !it2->second.empty())

cout<<"Midpoint of "<<it1->first<<" at time "<<current\_time<<" is $"<<(it1->second.top().price+it2->second.top().price)/2<<endl;

else

cout<<"Midpoint of "<<it1->first<<" at time "<<current\_time<<" is undefined\n";

it1++;

it2++;

}

}

int main(int argc, char \*argv[])

{

char opt;

char \*short\_opts=(char \*)"vmptg:";

struct option long\_opts[]={

{"verbose", no\_argument, NULL, 'v'},

{"median", no\_argument, NULL, 'm'},

{"midpoint", no\_argument, NULL, 'p'},

{"transfers", no\_argument, NULL, 't'},

{"ttt", required\_argument, NULL, 'g'},

{0, 0, 0, 0}

};

int fee, num\_shares, num\_trades, amount;

fee=0; amount=0; num\_trades=0; num\_shares=0;

bool verbose,median,midpoint,transfers,ttt;

verbose=false;

median=false;

midpoint=false;

transfers=false;

ttt=false;

int n=1;

int price,quantity;

traveler temp;

temp.buy\_time=-1;

temp.sell\_time=-1;

temp.buy\_price=-1;

temp.sell\_price=-1;

temp.lowest\_time=-1;

temp.lowest\_price=-1;

map\_traveler t;

map\_string\_traveler t2;

while ((opt=getopt\_long(argc,argv,short\_opts,long\_opts,NULL))!=-1)

{

switch (opt)

{

case 'v': verbose=true;break;

case 'm': median=true;break;

case 'p': midpoint=true;break;

case 't': transfers=true;break;

case 'g':

{

ttt=true;

temp.symbol=optarg;

temp.order=n;

t.insert(make\_pair(n,temp));

t2.insert(make\_pair(optarg,temp));

n++;

break;

}

default: cout<<"Wrong Arguments!\n";exit(0);

}

}

string str;

istringstream iStream;

string goal;

int duration;

char c;

client tem;

client tem1;

int current\_time=0;

trade\_max m1;

trade\_max::iterator it\_max;

priority\_queue<int> no\_max;

trade\_min m2;

trade\_min::iterator it\_min;

priority\_queue<int, vector<int>, greater<int>> no\_min;

type\_map Buy;

type\_map Sell;

client\_heap none;

map\_transfer trans;

type\_map::iterator it;

map\_transfer::iterator it\_trans;

transfer notrans;

notrans.number\_sell=0;

notrans.number\_buy=0;

notrans.net=0;

int order=0;

map\_traveler::iterator it\_traveler;

map\_string\_traveler::iterator it\_st;

while (getline(cin,str))

{

if (str=="exit") break;

iStream.str(str);

iStream>>tem.time>>tem.name>>goal>>tem.symbol>>c>>tem.price>>c>>tem.quantity>>tem.duration;

iStream.clear();

tem.order=order;

order++;

if (trans.find(tem.name)==trans.end())

trans.insert(make\_pair(tem.name,notrans));

if (goal=="BUY") tem.type=0;

else tem.type=1;

if (median)

{

if (m1.find(tem.symbol)==m1.end())

m1.insert(make\_pair(tem.symbol,no\_max));

if (m2.find(tem.symbol)==m2.end())

m2.insert(make\_pair(tem.symbol,no\_min));

}

if (tem.time!=current\_time)

{

if (median) Median(m1,m2,current\_time);

if (midpoint) Midpoint(Buy,Sell,current\_time);

current\_time=tem.time;

}

if (tem.type==0)//buy

{

if (ttt)

{

it\_st=t2.find(tem.symbol);

if (it\_st!=t2.end())

{

if ((it\_st->second.buy\_time!=-1) && (tem.price>it\_st->second.sell\_price))

{

it\_st->second.sell\_price=tem.price;

it\_st->second.sell\_time=current\_time;

t[it\_st->second.order]=it\_st->second;

}

if ((it\_st->second.lowest\_time!=-1) && (tem.price-it\_st->second.lowest\_price>it\_st->second.sell\_price-it\_st->second.buy\_price))

{

it\_st->second.sell\_price=tem.price;

it\_st->second.sell\_time=current\_time;

it\_st->second.buy\_time=it\_st->second.lowest\_time;

it\_st->second.buy\_price=it\_st->second.lowest\_price;

it\_st->second.lowest\_price=-1;

it\_st->second.lowest\_time=-1;

t[it\_st->second.order]=it\_st->second;

}

}

}

if (Buy.find(tem.symbol)==Buy.end())

{

Buy.insert(make\_pair(tem.symbol, none));

Sell.insert(make\_pair(tem.symbol, none));

it=Buy.find(tem.symbol);

if (tem.duration!=0)

it->second.push(tem);

}

else

{

it=Sell.find(tem.symbol);

if (!it->second.empty())

{

while (it->second.top().price<=tem.price)

{

if ((it->second.top().time+it->second.top().duration<=current\_time)&&(it->second.top().duration!=-1))

{

it->second.pop();

}

else

{

if (it->second.top().quantity>=tem.quantity)

{

if (verbose)

{

cout<<tem.name<<" purchased "<<tem.quantity<<" shares of "<<tem.symbol<<" from "<<it->second.top().name<<" for $"<<it->second.top().price<<"/share\n";

}

fee=fee+tem.quantity\*it->second.top().price/100\*2;

amount=amount+tem.quantity\*it->second.top().price;

num\_trades=num\_trades+1;

num\_shares=num\_shares+tem.quantity;

tem1=it->second.top();

it->second.pop();

tem1.quantity=tem1.quantity-tem.quantity;

price=tem1.price;

quantity=tem.quantity;

if (tem1.quantity>=0) it->second.push(tem1);

it\_trans=trans.find(it->second.top().name);

it\_trans->second.number\_sell=it\_trans->second.number\_sell+quantity;

it\_trans->second.net=it\_trans->second.net+price\*quantity;

it\_trans=trans.find(tem.name);

it\_trans->second.number\_buy=it\_trans->second.number\_buy+quantity;

it\_trans->second.net=it\_trans->second.net-price\*quantity;

tem.quantity=0;

if (median)

{

it\_max=m1.find(tem.symbol);

it\_min=m2.find(tem.symbol);

if (it\_max->second.empty()||it\_max->second.top()>=price) it\_max->second.push(price);

else it\_min->second.push(price);

while (it\_max->second.size()>it\_min->second.size()+1)

{

it\_min->second.push(it\_max->second.top());

it\_max->second.pop();

}

while (it\_min->second.size()>it\_max->second.size()+1)

{

it\_max->second.push(it\_min->second.top());

it\_min->second.pop();

}

}

if (it->second.top().quantity==0) it->second.pop();

break;

}

else

{

if (verbose)

{

cout<<tem.name<<" purchased "<<it->second.top().quantity<<" shares of "<<tem.symbol<<" from "<<it->second.top().name<<" for $"<<it->second.top().price<<"/share\n";

}

fee=fee+it->second.top().quantity\*it->second.top().price/100\*2;

amount=amount+it->second.top().quantity\*it->second.top().price;

num\_trades=num\_trades+1;

num\_shares=num\_shares+it->second.top().quantity;

price=it->second.top().price;

quantity=it->second.top().quantity;

it\_trans=trans.find(it->second.top().name);

it\_trans->second.number\_sell=it\_trans->second.number\_sell+quantity;

it\_trans->second.net=it\_trans->second.net+price\*quantity;

it\_trans=trans.find(tem.name);

it\_trans->second.number\_buy=it\_trans->second.number\_buy+quantity;

it\_trans->second.net=it\_trans->second.net-price\*quantity;

tem.quantity=tem.quantity-quantity;

if (median)

{

it\_max=m1.find(tem.symbol);

it\_min=m2.find(tem.symbol);

if (it\_max->second.empty()||it\_max->second.top()>=price) it\_max->second.push(price);

else it\_min->second.push(price);

while (it\_max->second.size()>it\_min->second.size()+1)

{

it\_min->second.push(it\_max->second.top());

it\_max->second.pop();

}

while (it\_min->second.size()>it\_max->second.size()+1)

{

it\_max->second.push(it\_min->second.top());

it\_min->second.pop();

}

}

it->second.pop();

}

}

if (it->second.empty()) break;

}

if ((tem.quantity>0)&&(tem.duration!=0))

{

it=Buy.find(tem.symbol);

it->second.push(tem);

}

}

else

{

if (tem.duration!=0)

{

it=Buy.find(tem.symbol);

it->second.push(tem);

}

}

}

}

else //sell

{

if (ttt)

{

it\_st=t2.find(tem.symbol);

if (it\_st!=t2.end())

{

if (it\_st->second.buy\_time==-1)

{

it\_st->second.buy\_price=tem.price;

it\_st->second.buy\_time=current\_time;

t[it\_st->second.order]=it\_st->second;

}

else

{

if ((tem.price<it\_st->second.buy\_price) && (it\_st->second.sell\_price==-1))

{

it\_st->second.buy\_price=tem.price;

it\_st->second.buy\_time=current\_time;

t[it\_st->second.order]=it\_st->second;

}

if ((tem.price<it\_st->second.buy\_price) && ((it\_st->second.lowest\_price==-1)||(it\_st->second.lowest\_price>tem.price)))

{

it\_st->second.lowest\_price=tem.price;

it\_st->second.lowest\_time=current\_time;

t[it\_st->second.order]=it\_st->second;

}

}

}

}

if (Sell.find(tem.symbol)==Sell.end())

{

Buy.insert(make\_pair(tem.symbol, none));

Sell.insert(make\_pair(tem.symbol, none));

it=Sell.find(tem.symbol);

if (tem.duration!=0);

it->second.push(tem);

}

else

{

it=Buy.find(tem.symbol);

if (!it->second.empty())

{

while (it->second.top().price>=tem.price)

{

if ((it->second.top().duration!=-1)&&(it->second.top().time+it->second.top().duration<=current\_time))

it->second.pop();

else

{

if (it->second.top().quantity>=tem.quantity)

{

if (verbose)

{

cout<<it->second.top().name<<" purchased "<<tem.quantity<<" shares of "<<tem.symbol<<" from "<<tem.name<<" for $"<<it->second.top().price<<"/share\n";

}

price=it->second.top().price;

quantity=tem.quantity;

fee=fee+tem.quantity\*it->second.top().price/100\*2;

amount=amount+tem.quantity\*it->second.top().price;

num\_trades++;

num\_shares=num\_shares+tem.quantity;

tem1=it->second.top();

it->second.pop();

tem1.quantity=tem1.quantity-tem.quantity;

if (tem1.quantity>=0) it->second.push(tem1);

it\_trans=trans.find(it->second.top().name);

it\_trans->second.number\_buy=it\_trans->second.number\_buy+quantity;

it\_trans->second.net=it\_trans->second.net-price\*quantity;

it\_trans=trans.find(tem.name);

it\_trans->second.number\_sell=it\_trans->second.number\_sell+quantity;

it\_trans->second.net=it\_trans->second.net+price\*quantity;

tem.quantity=0;

if (median)

{

it\_max=m1.find(tem.symbol);

it\_min=m2.find(tem.symbol);

if (it\_max->second.empty()||it\_max->second.top()>=price) it\_max->second.push(price);

else it\_min->second.push(price);

while (it\_max->second.size()>it\_min->second.size()+1)

{

it\_min->second.push(it\_max->second.top());

it\_max->second.pop();

}

while (it\_min->second.size()>it\_max->second.size()+1)

{

it\_max->second.push(it\_min->second.top());

it\_min->second.pop();

}

}

if (it->second.top().quantity==0) it->second.pop();

break;

}

else

{

if (verbose)

{

cout<<it->second.top().name<<" purchased "<<it->second.top().quantity<<" shares of "<<tem.symbol<<" from "<<tem.name<<" for $"<<it->second.top().price<<"/share\n";

}

price=it->second.top().price;

quantity=it->second.top().quantity;

fee=fee+it->second.top().quantity\*it->second.top().price/100\*2;

amount=amount+it->second.top().quantity\*it->second.top().price;

num\_trades++;

num\_shares=num\_shares+it->second.top().quantity;

it\_trans=trans.find(it->second.top().name);

it\_trans->second.number\_buy=it\_trans->second.number\_buy+quantity;

it\_trans->second.net=it\_trans->second.net-price\*quantity;

it\_trans=trans.find(tem.name);

it\_trans->second.number\_sell=it\_trans->second.number\_sell+quantity;

it\_trans->second.net=it\_trans->second.net+price\*quantity;

tem.quantity=tem.quantity-quantity;

if (median)

{

it\_max=m1.find(tem.symbol);

it\_min=m2.find(tem.symbol);

if (it\_max->second.empty()||it\_max->second.top()>=price) it\_max->second.push(price);

else it\_min->second.push(price);

while (it\_max->second.size()>it\_min->second.size()+1)

{

it\_min->second.push(it\_max->second.top());

it\_max->second.pop();

}

while (it\_min->second.size()>it\_max->second.size()+1)

{

it\_max->second.push(it\_min->second.top());

it\_min->second.pop();

}

}

it->second.pop();

}

}

if (it->second.empty()) break;

}

if ((tem.quantity>0)&&(tem.duration!=0))

{

it=Sell.find(tem.symbol);

it->second.push(tem);

}

}

else

{

if (tem.duration!=0)

{

it=Sell.find(tem.symbol);

it->second.push(tem);

}

}

}

}

}

if (median) Median(m1,m2,current\_time);

if (midpoint) Midpoint(Buy,Sell,current\_time);

cout<<"---End of Day---\n";

cout<<"Commission Earnings: $"<<fee<<endl;

cout<<"Total Amount of Money Transferred: $"<<amount<<endl;

cout<<"Number of Completed Trades: "<<num\_trades<<endl;

cout<<"Number of Shares Traded: "<<num\_shares<<endl;

if (transfers)

{

it\_trans=trans.begin();

while (it\_trans!=trans.end())

{

cout<<it\_trans->first<<" bought "<<it\_trans->second.number\_buy<<" and sold "<<it\_trans->second.number\_sell<<" for a net transfer of $"<<it\_trans->second.net<<endl;

it\_trans++;

}

}

if (ttt)

{

for (int i=1;i<n;i++)

{

if ((t[i].buy\_time!=-1)&&(t[i].sell\_time!=-1))

cout<<"Time travelers would buy "<<t[i].symbol<<" at time: "<<t[i].buy\_time<<" and sell it at time: "<<t[i].sell\_time<<endl;

else

cout<<"Time travelers would buy "<<t[i].symbol<<" at time: "<<-1<<" and sell it at time: "<<-1<<endl;

}

}

}

Makefile

all: main

main: main.o

g++ -o main main.o

main.o: main.cpp

g++ -c main.cpp -std=c++11

clean: rm -f main\*.o