Stock Lab

In this project, we created a primative stock market that implements minimum and maximum heaps. Users will be able to buy and sell stocks and see what the market price is for each stock in a General User Interface. A window in the GUI will also list the previous transactions done.

```
#ifndef LABH
#define LABH

#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Button.H>
#include <FL/Fl_Input.H>
#include <FL/Fl_Text_Display.H>
#include <FL/Fl_Text_Editor.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Choice.H>
#include <vector>
#include <algorithm>
#include <iomanip>
#include "HeapMing.h"
#include "Order.h"
#include "Log.h"
#include "StockExchange.h"
using namespace std;
```

- include files for all modules

Stock Lab: lab.h Cont

```
template<typename T>
string to_string(const T& x)
{
    std::ostringstream os;
    os << x;
    return os.str();
}

const int WIDTH = 600;
const int HEIGHT = 600;
const string NAMES[]= {"GOOG", "INTL", "AAPL"};
extern bool wantToBuy;

void buyPressed_cb(Fl_Widget*);
void sellPressed_cb(Fl_Widget*);
void placeOrderPressed_cb(Fl_Widget*);

void createAllWidgets();
void hideMenuScreen();
void showMenuScreen();
void showInputScreen();
void hideInputScreen();
void showInputScreen();
```

- template for converting integers to string in the transaction log
- constants such as window width and height
- function declarations

Stock Lab: lab.h Cont

```
extern Fl_Double_Window window;
extern Fl_Choice* stockNameInput;
extern Fl_Input* numberSharesInput;
extern Fl_Input* priceInput;
extern Fl_Box* stockInfo1;
extern Fl_Box* stockInfo2;
extern Fl_Box* stockInfo3;
extern Fl_Box* stockInfo4;
extern Fl_Box* stockInfo5;
extern Fl_Box* stockInfo6;
extern Fl_Button* sellB;
extern Fl_Button* buyB;
extern Fl_Button* placeOrderB;
extern Fl_Text_Buffer* historyBuffer;
extern Fl_Text_Display* history;

extern StockExchange GOOGse;
extern StockExchange INTLse;
extern StockExchange AAPLse;
extern vector<Log> logsV;
extern string hist;
```

- extern statements for global variables
- the company objects are made global, so are the vector of logs and hist string

```
#ifndef HEAP_H
# define HEAP_H
template <class T>
class        Heap
{
        public:
                Heap(void) {}
                Heap(int size);
                ~Heap(void);

                int                getSize(void) { return _size; }
                int                getnNodes(void) { return _nNodes; }
                bool        isEmpty(void) { return (_nNodes == 0); }
                bool        isFull(void) { return (_nNodes == _size); }
                bool        insert(T item);
                bool        remove(T &item);
                bool        peek(T &item);
                virtual bool        compare(T &lhs, T &rhs) = 0;
        protected:
                int                _size;
                int                _nNodes;
                T                *_buf;
};
```

- in this part of the heap header file we are creating the Heap class

Stock Lab: HeapMing.h Cont

```
template <class T>
Heap<T>::Heap(int size)
{
        _buf = new T[size + 1];
        _size = (_buf) ? size : 0;
        _nNodes = 0;
}
template <class T>
Heap<T>::~Heap(void)
{
        delete[] _buf;
}
template <class T>
bool        Heap<T>::insert(T item)
{
        if (isFull())
                return false;
        int        i = ++_nNodes;
        for (int iParent = 0; i > 1; i = iParent)
        {
                iParent = i / 2;
                if (!compare(_buf[iParent], item))
                        break ;
                _buf[i] = _buf[iParent];
        }
        _buf[i] = item;
        return true;
}
```

In lines 25 through 52, we define the following member functions.

- Heap constructer
- Heap destructer
- Heap Insert

Stock Lab: HeapMing.h Cont

```
template <class T>
bool        Heap<T>::remove(T &item)
{
        if (isEmpty())
                return false;
        item = _buf[1];
        int iParent = 1;
        for (int i = 2; i <= _nNodes; i *= 2)
        {
                if (i < _nNodes && compare(_buf[i], _buf[i + 1]))
                        i++;
                if (!compare(_buf[_nNodes], _buf[i]))
                        break ;
                _buf[iParent] = _buf[i];
                iParent = i;
        }
        _buf[iParent] = _buf[_nNodes--];
        return true;
}

template <class T>
bool        Heap<T>::peek(T &item)
{
        if (isEmpty())
                return false;
        item = _buf[1];
        return true;
}
```

In lines 52 through 82, we define the following member functions.

- Heap remove
- Heap peek

Stock Lab: HeapMing.h Cont

```
template <class T>
class        MaxHeap : public Heap<T>
{
        public:
                MaxHeap(int size) : Heap<T>::Heap(size) {};
                ~MaxHeap(void) {};

                bool        compare(T &lhs, T &rhs) { return (lhs < rhs); }
};

template <class T>
class        MinHeap : public Heap<T>
{
        public:
                MinHeap(int size) : Heap<T>::Heap(size) {};
                ~MinHeap(void) {};

                bool        compare(T &lhs, T &rhs) { return (lhs > rhs); }
};
```

In lines 83 through 101, we define the following member functions.

- MinHeap
- MaxHeap

Stock Lab: main.cpp

In our main.cpp file, we will first call the function to create all the widgets on one window, then show the menu screen, and finally call the FL::run command to start up the GUI.

```
#include "lab.h"

StockExchange GOOGse(5);
StockExchange INTLse(5);
StockExchange AAPLse(5);
vector<Log> logsV; //Log vector is also global so I
                   //can reference it in Place order pressed
string hist = "Transaction Log\n";

Fl_Double_Window window(WIDTH, HEIGHT);
bool wantToBuy = true;

int main()
{
    // Logo provided by Mingun Inc.
        std::cout <<
        "/ ********************************** \\\n"
        "* CS124 - STOCK EXCHANGE       MADE BY. *\n"
        "* Dean Huang  : Gevorg Keshishian       *\n"
        "* Mingyun Kim : Muhammed Hassan Mahmood *\n"
        "\\ ********************************** /\n"
        << std::endl;
    //Open cool stuff
    createAllWidgets();
    showMenuScreen();
    window.label("Stock Market");
    window.show();

    GOOGse.populate("GOOG");//Populate the 3 companies
    INTLse.populate("INTL");
    AAPLse.populate("AAPL");

    Fl::run();
}
```

```
1  #include "lab.h"
2
3  Fl_Box* stockInfo1;
4  Fl_Box* stockInfo2;
5  Fl_Box* stockInfo3;
6  Fl_Box* stockInfo4;
7  Fl_Box* stockInfo5;
8  Fl_Box* stockInfo6;
9  Fl_Button* buyB;
10 Fl_Button* sellB;
11 Fl_Button* placeOrderB;
12 Fl_Text_Buffer* historyBuffer;
13 Fl_Text_Display* history;
14 Fl_Choice* stockNameInput;
15 Fl_Input* numberSharesInput;
16 Fl_Input* priceInput;
17
18 void createAllWidgets()
19 {
20     Order peek;
21     window.label("Stock Market");
22     window.show();
```

In this file, we will be creating all the widgets on the GUI. From lines 3 to 16, we declare all the widgets. On line 18, we begin the createAllWidgets function and show the window to begin drawing.

Stock Lab: createAllWidgets.cpp Cont

```
stockInfo1 = new Fl_Box(0, 0, WIDTH/3, HEIGHT/8);
stockInfo1->label("GOOG");
stockInfo1->hide();
stockInfo1->box(FL_BORDER_BOX);
window.add(stockInfo1);

stockInfo2 = new Fl_Box(WIDTH/3, 0, WIDTH/3, HEIGHT/8);
stockInfo2->label("INTL");
stockInfo2->box(FL_BORDER_BOX);
stockInfo2->hide();
window.add(stockInfo2);

stockInfo3 = new Fl_Box(2*WIDTH/3, 0, WIDTH/3, HEIGHT/8);
stockInfo3->label("AAPL");
stockInfo3->box(FL_BORDER_BOX);
stockInfo3->hide();
window.add(stockInfo3);


stockInfo4 = new Fl_Box(0, HEIGHT/8, WIDTH/3, HEIGHT/8);
stockInfo4->label("Known for Search Engine");
stockInfo4->box(FL_BORDER_BOX);
stockInfo4->hide();
window.add(stockInfo4);

stockInfo5 = new Fl_Box(WIDTH/3, HEIGHT/8, WIDTH/3, HEIGHT/8);
stockInfo5->label("Known for Computer Chips");
stockInfo5->box(FL_BORDER_BOX);
stockInfo5->hide();
window.add(stockInfo5);

stockInfo6 = new Fl_Box(2*WIDTH/3, HEIGHT/8, WIDTH/3, HEIGHT/8);
stockInfo6->label("Known for the iPhone");
stockInfo6->box(FL_BORDER_BOX);
stockInfo6->hide();
window.add(stockInfo6);
```

In this section of the cpp file, we create all the boxes which display the stocks and their market prices, and general information about the company as well

Stock Lab: createAllWidgets.cpp Cont

```
buyB = new Fl_Button(9*WIDTH/40, 5*HEIGHT/16, WIDTH/5, HEIGHT/8);
buyB->label("BUY");
buyB->labelsize(24);
buyB->callback(buyPressed_cb);
buyB->hide();
window.add(buyB);
sellB = new Fl_Button(23*WIDTH/40, 5*HEIGHT/16, WIDTH/5, HEIGHT/8);
sellB->label("SELL");
sellB->labelsize(24);
sellB->callback(sellPressed_cb);
sellB->hide();
window.add(sellB);

historyBuffer = new Fl_Text_Buffer();
history = new Fl_Text_Display(0, HEIGHT/2, WIDTH, HEIGHT/2);
history->buffer(historyBuffer);
history->hide();
window.add(history);
```

In this section of the cpp file, we are creating the widgets for the buy and sell buttons, as well as the history section, which shows previous buy and sell orders.

```
stockNameInput = new Fl_Choice(WIDTH/4, HEIGHT/8, WIDTH/2, HEIGHT/20, "Stock name");
for(int i = 0; i < 3;i++) {stockNameInput->add(NAMES[i].c_str());}
stockNameInput->hide();
window.add(stockNameInput);
numberSharesInput = new Fl_Input(WIDTH/4, 2*HEIGHT/8, WIDTH/2, HEIGHT/20, "Num Shares");
numberSharesInput->hide();
window.add(numberSharesInput);
priceInput = new Fl_Input(WIDTH/4, 3*HEIGHT/8, WIDTH/2, HEIGHT/20, "Price");
priceInput->hide();
window.add(priceInput);

placeOrderB = new Fl_Button(WIDTH/4, HEIGHT/2, WIDTH/4, HEIGHT/10);
placeOrderB->label("Place Order");
placeOrderB->callback(placeOrderPressed_cb);
placeOrderB->hide();
window.add(placeOrderB);
```

In this section of the cpp file, we are creating the widgets that make the buy and sell orders possible, specifically for the drop down menus and the choices that the user is given, and finally the button for confirming the order.

This is the most important class in this program Stock exchange is basically a company, and inside the company there is a max heap for buyers and min heap for sellers The remove and insert will depend on whether you're a buyer or seller. Buyers touch the max heap and Sellers touch the min heap. The function bodies are just one layer down into the Heap's functions.

```
#ifndef STOCK_EXCHANGE_H
# define STOCK_EXCHANGE_H

class        StockExchange
{
        public:
                StockExchange(void) {}
                StockExchange(int size);
                ~StockExchange(void);

                void        populate(std::string symbol);
                bool        saveSeller(std::string file);
                bool        saveBuyer(std::string file);
                bool        loadSeller(std::string file);
                bool        loadBuyer(std::string file);

                bool        saveLog(std::string file);
                bool        loadLog(std::string file);

                int                getSize(void) { return _size; }
                double        getTotal(void) { return _total; }
                void        setTotal(double total) { _total = total; }
                void        addTotal(double price) { _total += price; }
                void        subTotal(double price) { _total -= price; }
```

```
        int             getSellernNodes(void) { return _seller->getnNodes(); }
        bool        insertSeller(Order item) { return _seller->insert(item); }
        bool        removeSeller(Order &item) { return _seller->remove(item); }
        bool        peekSeller(Order &item);

        int             getBuyernNodes(void) { return _buyer->getnNodes(); }
        bool        insertBuyer(Order item) { return _buyer->insert(item); }
        bool        removeBuyer(Order &item) { return _buyer->remove(item); }
        bool        peekBuyer(Order &item);

    private:
        MinHeap<Order>          *_seller;
        MaxHeap<Order>          *_buyer;
        std::vector<Log>    _log;
        int                             _size;
        double                      _total;
};

#endif
```

This is the most important class in this program Stock exchange is basically a company, and inside the company there is a max heap for buyers and min heap for sellers The remove and insert will depend on whether you're a buyer or seller. Buyers touch the max heap and Sellers touch the min heap. The function bodies are just one layer down into the Heap's functions.

```
#ifndef ORDER_H
# define ORDER_H

enum e_indicator
{
        BUY = 0,
        SELL
};

class        Order
{
private:
                std::string        _name;
                std::string        _symbol;
                e_indicator        _indicator;
                int                        _share;
                double                _price;
public:
                Order(void) {};

        Order(std::string name, std::string symbol, e_indicator indicator, int share, double price)
```

Order is simple A buyer and Seller both make "orders" It is basically 'I want company A, 500 shares, for 20 dollars. ' Something like that

Stock Lab: Order.h Cont

```
        :_name(name),
        _symbol(symbol),
        _indicator(indicator),
        _share(share),
        _price(price)
        {
        }

                ~Order(void) {}


                std::string        getCustomer(void) { return _name; }
                std::string        getSymbol(void) { return _symbol; }
                e_indicator        getIndicator(void) { return _indicator; }
        void        setShare(int s){ _share = s;}
                int                          getShare(void) { return _share; }
                double                getPrice(void) { return _price; }

                bool        operator<(Order &rhs) { return (this->_price < rhs._price); }
                bool        operator>(Order &rhs) { return (this->_price > rhs._price); }
};



#endif
```

Order is simple A buyer and Seller both make "orders" It is basically 'I want company A, 500 shares, for 20 dollars. ' Something like that

```
#ifndef LOG_H
# define LOG_H
# include "lab.h"
class Log
{
        public:
        //Default constructor basically
                Log(std::string name, std::string symbol, e_indicator indicator,
                        int share, int price)
                :        _name(name), _symbol(symbol), _indicator(indicator),
                        _share(share), _price(price) {}

                ~Log(void) {}

                void        setName(std::string name) { _name = name; }
                void        setSymbol(std::string symbol) { _symbol = symbol; }
                void        setIndicator(e_indicator ind) { _indicator = ind; }
                void        setShare(int share) { _share = share; }
                void        setPrice(double price) { _price = price; }

                std::string        getName(void) { return _name; }
                std::string        getSymbol(void) { return _symbol; }
                e_indicator        getIndicator(void) { return _indicator; }
                int                        getShare(void) { return _share; }
                int                getPrice(void) { return _price; }

        private:
                std::string        _name;
                std::string        _symbol;
                e_indicator        _indicator;
                int                        _share;
                int                _price;
};
```

The log exists to keep track of every successful transaction Only successful transactions wille be recorded in the vector of logs.

```
#include "lab.h"
void hideInputScreen()
{
    placeOrderB->hide();
    stockNameInput->hide();
    numberSharesInput->hide();
    priceInput->hide();
}
```

- This function is responsible for closing the order window after it is used

```
#include "lab.h"
void hideMenuScreen()
{
    stockInfo1->hide();
    stockInfo2->hide();
    stockInfo3->hide();
    stockInfo4->hide();
    stockInfo5->hide();
```

- This function is responsible for closing the menu widgets after they are used

Stock Lab: showInputScreen.cpp

```
#include "lab.h"
void showInputScreen()
{
    placeOrderB->show();
    stockNameInput->show();
    numberSharesInput->show();
    priceInput->show();
}
```

- This function is responsible for opening the input widgets so they can be used

Stock Lab: showMenuScreen.cpp

```
#include "lab.h"
void showMenuScreen()
{
    stockInfo1->show();
    stockInfo2->show();
    stockInfo3->show();
    stockInfo4->show();
    stockInfo5->show();
    stockInfo6->show();
    sellB->show();
    buyB->show();

    //Here, the updated history string will be put into menu's history display
    historyBuffer->text(hist.c_str());
    history->buffer(historyBuffer);
    history->show();
}
```

- This function is responsible for opening the menu widgets

```
1  #include "lab.h"
2  StockExchange::StockExchange(int size)
3  {
4          _seller = new MinHeap<Order>(size);
5          _buyer = new MaxHeap<Order>(size);
6          _size = size;
7          _total = 0;
8  }
9
10 StockExchange::~StockExchange(void)
11 {
12         delete _seller;
13         delete _buyer;
14 }
15
16 bool        StockExchange::peekSeller(Order &item)
17 {
18         if (_seller->peek(item))
19                 return true;
20         else
21                 return false;
22 }
23 bool        StockExchange::peekBuyer(Order &item)
24 {
25         if (_buyer->peek(item))
26                 return true;
27         else
28                 return false;
29 }
```

This file contains function definitions for all of the functions for the StockExchange class. Lines 1 through 29 contains the following functions:

- Constructor
- Destructor
- PeekSeller
- PeekBuyer

Stock Lab: StockExchange.cpp Cont

Lines 31 through 61 contains the following functions:

- Populate: fills the heap with random data
- SaveSeller

```
31  void        StockExchange::populate(std::string symbol)
32  {
33          for (int i = 0; i < _size; i++)
34          {
35                  _buyer->insert(Order("Buyer", symbol, BUY, rand() % 3000,
36                                       static_cast<double>(rand() % 10000) / 100));
37                  _seller->insert(Order("Seller", symbol, SELL, rand() % 3000,
38                                       static_cast<double>(rand() % 10000) / 100));
39          }
40  }
41  bool        StockExchange::saveSeller(std::string file)
42  {
43          std::fstream        f(file.c_str());
44          Order        tmp;
45
46          if (!f)
47                  return false;
48          f << _seller->getnNodes() << std::endl;
49          while (_seller->remove(tmp))
50          {
51                  f <<
52                  tmp.getCustomer() << "\t" <<
53                  tmp.getSymbol() << "\t" <<
54                  (tmp.getIndicator()) ? "SELL\t" : "BUY\t";
55                  f << tmp.getShare();
56                  f << "\t";
57                  f << tmp.getPrice() << std::endl;
58          }
59          return true;
60  }
```

```
62  bool        StockExchange::saveBuyer(std::string file)
63  {
64          std::fstream        f(file.c_str());
65          Order       tmp;
66
67          if (!f)
68                  return false;
69          f << _buyer->getSize() << std::endl;
70          while (_buyer->remove(tmp))
71          {
72                  f <<
73                  tmp.getCustomer() << "\t" <<
74                  tmp.getSymbol() << "\t" <<
75                  (tmp.getIndicator()) ? "SELL\t" : "BUY\t";
76                  f << tmp.getShare();
77                  f << "\t";
78                  f << tmp.getPrice() << std::endl;
79          }
80          return true;
81  }
```

Lines 62 through 81 contains the SaveBuyer function

Lines 83 through 105 contains the LoadSeller function

```
83  bool        StockExchange::loadSeller(std::string file)
84  {
85          std::fstream        f(file.c_str());
86
87          if (!f)
88                  return false;
89          int         size;
90          f >> size;
91          if (!(_seller = new MinHeap<Order> (size)))
92                  return false;
93
94          std::string         name, symbol, inds;
95          e_indicator         ind;
96          int                         share;
97          double                 price;
98
99          while (f.is_open())
100         {
101                 f >> name >> symbol >> inds >> share >> price;
102                 ind = (inds == "SELL") ? SELL : BUY;
103                 _seller->insert(Order(name, symbol, ind, share, price));
104         }
105 }
```

Lines 107 through 129 contains the LoadBuyer function

```
107 bool        StockExchange::loadBuyer(std::string file)
108 {
109         std::fstream        f(file.c_str());
110
111         if (!f)
112                 return false;
113         int        size;
114         f >> size;
115         if (!(_buyer = new MaxHeap<Order> (size)))
116                 return false;
117
118         std::string        name, symbol, inds;
119         e_indicator        ind;
120         int                        share;
121         double                price;
122
123         while (f.is_open())
124         {
125                 f >> name >> symbol >> inds >> share >> price;
126                 ind = (inds == "SELL") ? SELL : BUY;
127                 _buyer->insert(Order(name, symbol, ind, share, price));
128         }
129 }
```

Lines 131 through 148 contains the SaveLog function

```
131 bool        StockExchange::saveLog(std::string file)
132 {
133         std::fstream        f(file.c_str());
134 
135         if (!f)
136                 return false;
137 
138         for (size_t i = 0; i < _log.size(); i++)
139         {
140                 f <<
141                 _log[i].getName() << "\t" <<
142                 _log[i].getSymbol() << "\t" <<
143                 (_log[i].getIndicator() == SELL) ? "SELL\t" : "BUY\t";
144                 f << _log[i].getShare();
145                 f << "\t" <<
146                 _log[i].getPrice() << "\t" << std::endl;
147         }
148 }
```

Stock Lab: StockExchange.cpp Cont

```
150 bool        StockExchange::loadLog(std::string file)
151 {
152         std::fstream        f(file.c_str());
153
154         if (!f)
155                 return false;
156
157         std::string        name, symbol, inds;
158         e_indicator        ind;
159         int                        share;
160         double                price;
161
162         while (f.is_open())
163         {
164                 f >> name >> symbol >> inds >> share >> price;
165                 ind = (inds == "SELL") ? SELL : BUY;
166                 _log.push_back(Log(name, symbol, ind, share, price));
167                 _total += price;
168         }
169         return true;
170 }
```

Lines 150 through 170 contains the LoadLog function