Final Team Project - Stock Exchange Lab

Generated by Doxygen 1.8.8

Fri Dec 8 2017 22:33:41

Contents

1 Specification

This is a stock exchange program where a user is able to buy and sell stocks, and ask for a quote from three companies, those being Google, Facebook, and Amazon. For every company, the program keeps track of the last selling/buying price, highest selling/buying price, lowest selling/buying price, top current number of shares being sold along with their price and top current number of shares willing to be bought and at what price. If the user wants to buy/sell stocks, they can enter their own price or buy/sell at market price. The user then also enters the number of shares they wish to buy/sell and their name. Market price becomes the current “bid” or “ask” price. If there is no current “bid” or “ask” price, the last sale price is taken instead. If the buying price is greater than the selling price or vice versa, the price the user chose will become the market price. If the user is selling or buying more shares than people are willing to purchase or sell, a partial sale will occur. The price at which the transaction is happening still matches, simply the number of shares available for the transaction doesn’t match. In other words, there is a deficit of shares. In this case, the seller/buyer sells/buys as many shares as they can and the remaining shares are pending for selling/buying with the user is informed of this information. If a transaction is successful, then the user will see a report of their purchase, whether it was a perfect transaction (shares and price match) or a partial sale. This program also allows the user to ask for a quote for any of the three companies. A quote contains the following information: highest price of stock sold, the lowest price sold, the last sale price, the top current number of shares being sold along with their price, top current number of shares willing to be bought and at what price.

2 HTML Code

This is our HTML code that displays the first HTML page the user goes on to.

```
<html>
<head>
<title>Stock Exchange Lab</title>  <!--Tab title -->
    <link href="https://fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet">
<style type="text/css">                  /*CSS to make the HTML form look pretty */
body{
background-image: url(rick-tap-110126.jpg);
background-size: cover;
background-repeat: no-repeat;
}

input, select{
            color: #2c3e50;
            font-size: 1vw;
padding-top: 0.5vw;
            padding-bottom: 0.45vw;
            padding-right: 0.3vw;
            padding-left: 0.3vw;
}

input[type=submit]{
font-family: 'Open Sans', Century Gothic, Helvetica, Geneva, sans-serif;
            font-size: 1.9vw;
}

div{
background-color: rgba(189, 195, 199, .9);
display: inherit;
margin: auto;
width: 40%;
padding: 13px;
border-radius: 5px;
```

```
font-size: 20px;
            font-size: 2vw;
font-family: 'Open Sans', Century Gothic, Helvetica, Geneva, sans-serif;
}

        img{
            max-width:100%;
        }

        table{
width: auto;
table-layout: fixed;
}

        .tableH{
            background-color: rgba(60,136,126,0.6);
            border: 1px solid rgba(60,136,126,1);
padding: 15px 30px;
text-align: center;
        }

th, .dblue{
background-color: rgba(48,48,48,0.5);
padding: 15px 30px;
text-align: center;
font-weight: 500;
font-size: 12px;
color: #fff;
text-transform: uppercase;
}

.dblue{
padding: 8px;
            border: 1px solid rgba(48,48,48,0.7);
}

.lblue{
```

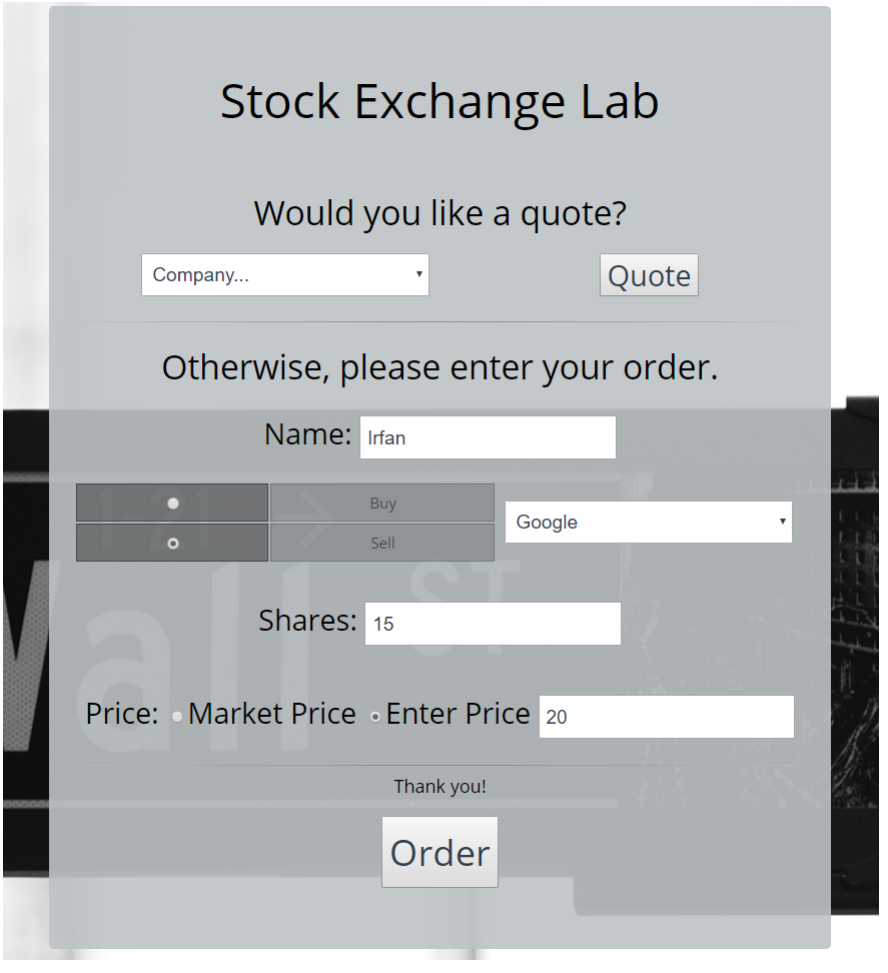
```
background-color: rgba(105,105,105,0.4);
height:3px;
overflow-x:auto;
margin-top: 0px;
border: 1px solid rgba(105,105,105,0.7);
padding: 8px;
text-align: center;
vertical-align: middle;
font-weight: 300;
font-size: 15px;
color: #37474f;
border-bottom: solid 1px rgba(85,119,136,0.7);
}

td{
height:3px;
overflow-x:auto;
margin-top: 0px;
padding: 8px;
text-align: center;
vertical-align: middle;
font-weight: 300;
font-size: 15px;
}

        hr{
border: 0;
height: 1px;
background-image: -webkit-linear-gradient(left, rgba(189, 195, 199, .6), #8c8b8b, rgba(189, 195, 199, .6));
background-image: -moz-linear-gradient(left, rgba(189, 195, 199, .6), #8c8b8b, rgba(189, 195, 199, .6));
background-image: -ms-linear-gradient(left, rgba(189, 195, 199, .6), #8c8b8b, rgba(189, 195, 199, .6));
background-image: -o-linear-gradient(left, rgba(189, 195, 199, .6), #8c8b8b, rgba(189, 195, 199, .6));
}
</style>
</head>

<body>
```

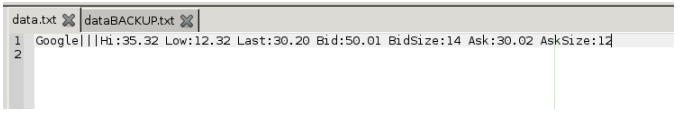
```
<div>  <!--HTML Form -->
<form action="/cgi-bin/final" method="GET">  <!--Forward user's input to the cgi file -->
<center>
<p style="font-size: 2.5vw; padding-bottom: 0.5%">Stock Exchange Lab</p>
<table>
            <tr><td colspan="4" style="font-size: 1.75vw; margin-top: 3px; margin-bottom: 15px; padding-top: 
            <td colspan="2">
                <select name="company" style="width: 15vw">
                    <option value="" disabled selected>Company...</option>
                    <option value="Google">Google</option>
                    <option value="Apple">Apple</option>
                    <option value="Facebook">Facebook</option>
                </select>
            <td ><input style="font-size:1.5vw; padding-top:0vw; padding-bottom:0vw" name="formType" type="sub
            </td></tr>
            <tr><td colspan="4"><hr></td></tr>
            <tr><td colspan="4" style="font-size: 1.75vw; margin-top: 3px; margin-bottom: 15px; padding-top: 
            <tr><td colspan="4" style="font-size: 1.5vw; padding-top: 2%; padding-bottom: 3%;">Name: <input ty
            <tr><td class="dblue"><input name="action" value="Buy" type="radio"></td> <td class="lblue">Buy</t
            <td rowspan="2" style="width: 1.2vw">
                <select name="company" style="width: 15vw">
                    <option value="" disabled selected>Company...</option>
                    <option value="Google">Google</option>
                    <option value="Apple">Apple</option>
                    <option value="Facebook">Facebook</option>
                </select>
            </td></tr>
            <tr><td class="dblue"><input name="action" value="Sell" type="radio"></td> <td class="lblue">Sell<
            <tr><td colspan="4" style="font-size: 1.5vw; padding-top: 5%">Shares: <input type="number" min="0"
            <tr><td colspan="4" style="font-size: 1.5vw; padding-top: 5%">Price:
                <input name="price" value="Market" type="radio">Market Price
                <input name="price" type="radio">Enter Price <input type="number" min="0" step="0.01" name="en
            </td></tr>
            <tr><td colspan="4"><hr><p style="font-size: 18px; margin-top: 2px;"> Thank you! </td></tr></p>
            <tr><td colspan="4"><input name="formType" type="submit" value="Order"></td></tr>
        </table>
</center>
```

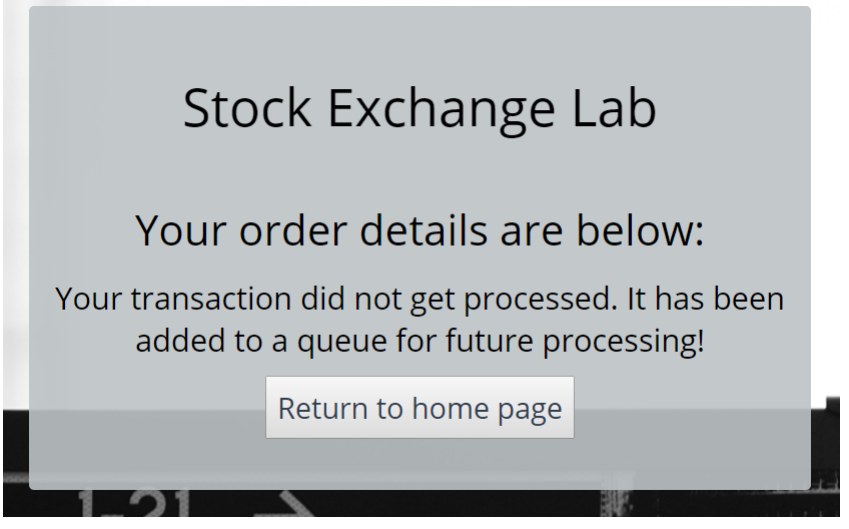
```
</form>
</div>
<!--
    <br>
    <div>
        <center>
        <p style="font-size: 1.8vw">Data Sample</p>
        <img src="/images/graph.png">
        </center>
    </div>
-->
</body>
</html>
```

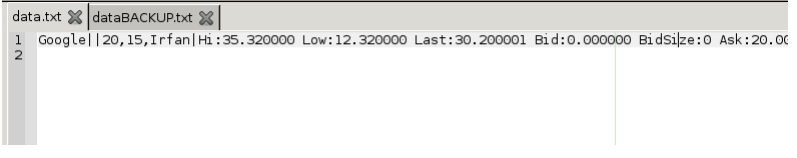
3 Test Part 1 (Seller/No Buyer)

This test shows the senario where a seller has arrived to sell stocks but no one is available to buy said stocks. The first image shows the submittion form where 15 stocks are being sold for $20. The second image shows how both heaps for the Google stock are empty. This is denoted by the empty spots between the '|' (Google| empty | empty | ...). The third image tells the user how their order could not be processed and that it will be added to the queue for future processing. This is the expected result. Since no one is available to buy the stocks, the user's order should be placed onto the selling heap and their order will therefore not be processed. This is exactly what happened when checking the fourth image. The selling heap (which is between the second and third '|') now has the user's order which includes their price, shares, and name (in that order).


Stock Exchange Lab
Would you like a quote?
Company...
Quote
Otherwise, please enter your order.
Name:
Irfan
Buy
Sell
Google
Shares:
15
Price:
Market Price
Enter Price
20
Thank you!
Order


data.txt
dataBACKUP.txt
1 Google|||Hi:35.32 Low:12.32 Last:30.20 Bid:50.01 BidSize:14 Ask:30.02 AskSize:12
2

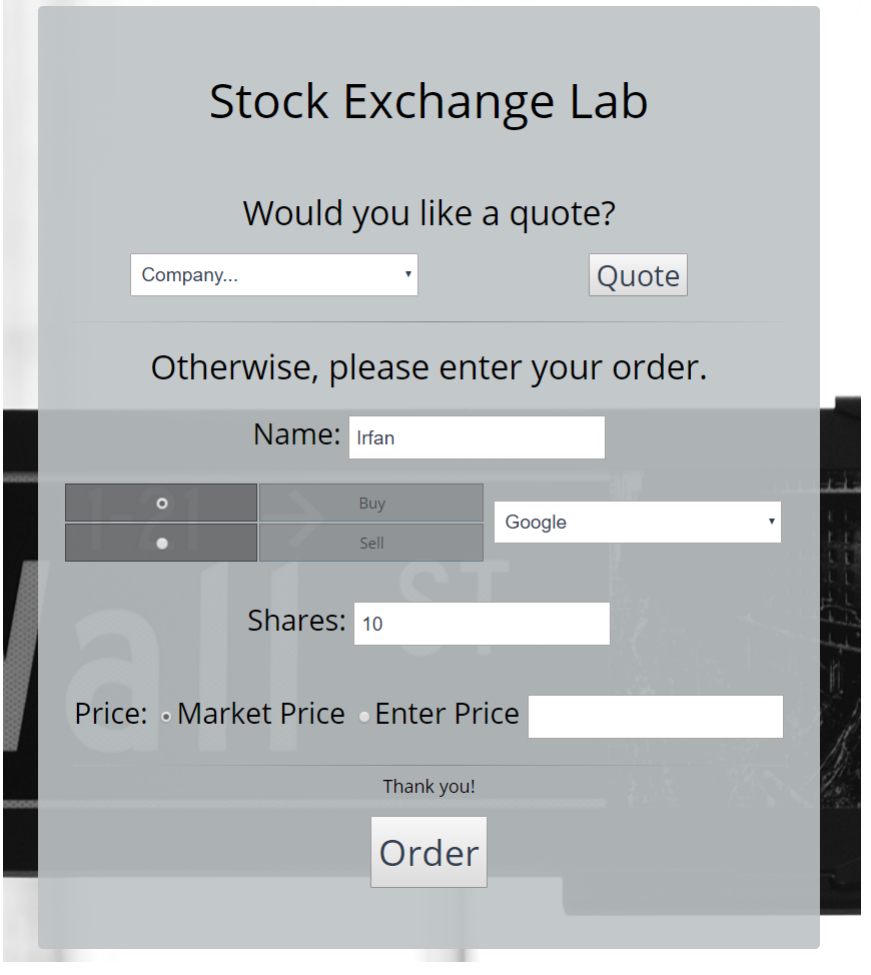

Stock Exchange Lab
Your order details are below:
Your transaction did not get processed. It has been added to a queue for future processing!
Return to home page


data.txt
dataBACKUP.txt
1 Google||20,15,Irfan|Hi:35.320000 Low:12.320000 Last:30.200001 Bid:0.000000 BidSize:0 Ask:20.0
2

4 Test Part 2 (Buyer/No Seller)

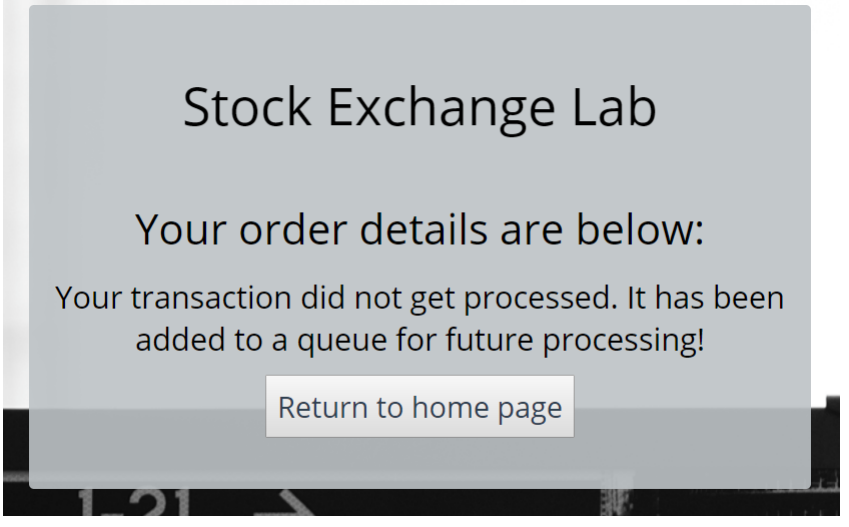
This test shows the senario where a buyer has arrived to buy stocks but no one is available to sell said stocks. The first image shows the submittion form where 10 stocks are being bought at market price. The second image shows how both heaps for the Google stock are empty. This is denoted by the empty spots between the '|' (Google| empty | empty | ...). The third image tells the user how their order could not be processed and that it will be added to the queue for future processing. This is the expected result. Since no one is available to sell the stocks, the user's order should be placed onto the buying heap and their order will therefore not be processed. This is exactly what happened when checking the fourth image. The buying heap (which is between the first and second '|') now has the user's order which includes their price, shares, and name (in that order).

This test also demonstrates the market price feature. The user in the first image picked the 'Market Price' option. If we look in the second image, since the heaps are completely empty, 'Market Price' in this case would fall back on whatever the last sale price was. This happens to be $30.20 as the second image shows. Looking at the fourth image now, we see that the user's price was indeed set to $30.20. Thus, the market price feature works as well.


Stock Exchange Lab
Would you like a quote?
Company...
Quote
Otherwise, please enter your order.
Name:
Irfan
Buy
Sell
Google
Shares:
10
Price:
Market Price
Enter Price
Thank you!
Order

data.txt | dataBACKUP.txt

```
1 Google|||Hi:35.32 Low:12.32 Last:30.20 Bid:50.01 BidSize:14 Ask:30.02 AskSize:12
2
```



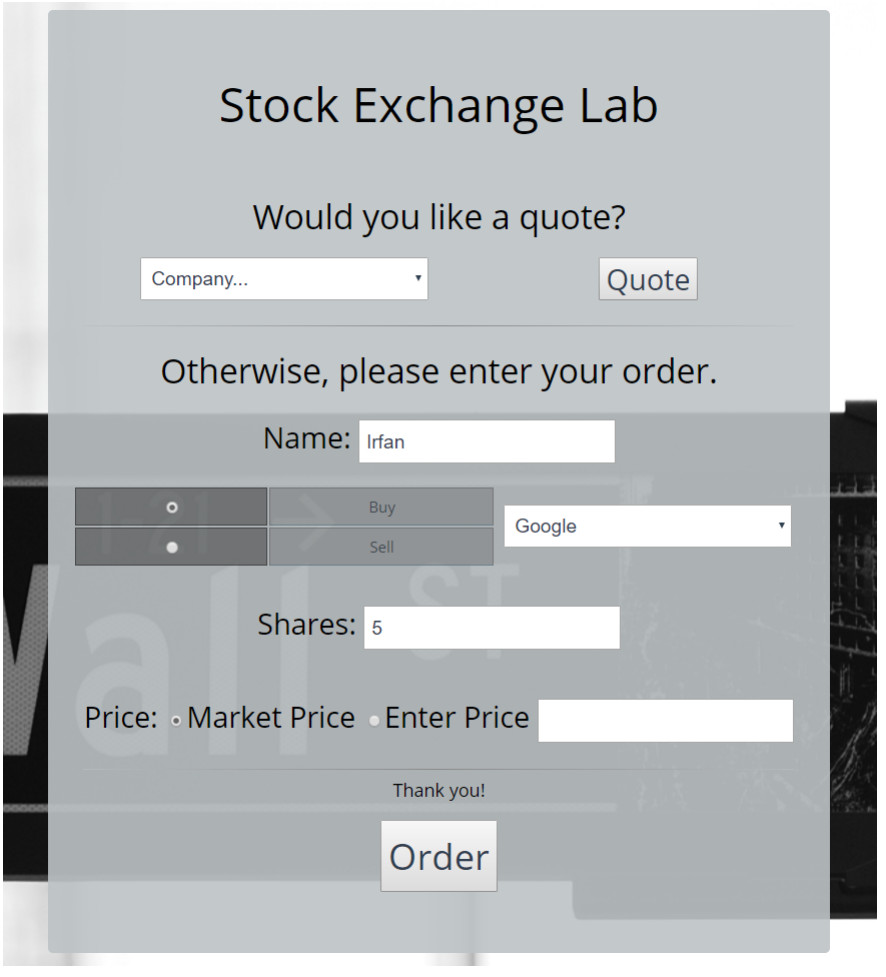
data.txt | dataBACKUP.txt

```
1 Google|30.2,10,Irfan||Hi:35.320000 Low:12.320000 Last:30.200001 Bid:30.200001 BidSize:10 Ask:0.00
2
```

5 Test Part 3 (Buyer/Seller Match - Perfect Transaction)

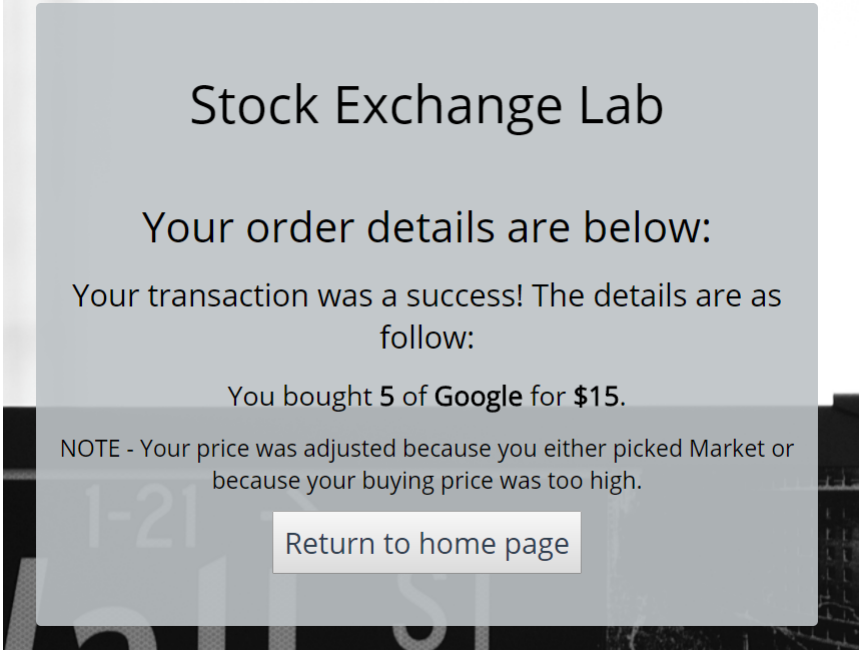
This test shows the senario where there is both a buyer and a seller available for a transaction. Furthermore, this is a perfect transaction which means not only do the prices match but also the number of shares being bought/sold. The first image shows the submission form. The user wants to buy 5 shares of Google stock at Market Price. The second image shows the two heaps before the transaction is made. The buy heap shows two people's orders already in line. The sell heap also has two people waiting in line to sell stock. After the transaction occures, we expect the buy heap to remian unchanged while the sell heap will no longer have Robert's order since it will be sold to Irfan. This information should be confirmed in the results page as well to the user. The third image shows the result page the user sees. As expected, the transaction was a complete success. The user bought 5 Google shares like they wanted. Looking at the fourth image, we see the expected heap changes as well. The buy heap remained unchanged while Robert was removed from the sell heap.

This test also shows the other Market Price senario. In this case, there was a seller on the sell heap and therefore, when Market Price is chosen, the user's price should be set to the price present in the root node instead of falling back on the last price data. Comparing the second and third image, this is exactly what happened. The second image shows Robert selling for $15 and in the third image, we see the prompt say 'You bought 5 of Google for $15' as expected.


Stock Exchange Lab
Would you like a quote?
Company...
Quote
Otherwise, please enter your order.
Name:
Irfan
Buy
Sell
Google
Shares:
5
Price:
Market Price
Enter Price
Thank you!
Order

data.txt | dataBACKUP.txt

```
1 Google|14,13,Ted 10,12,Bob|15,5,Robert 30,43,Mark|Hi:35.320000 Low:12.320000 Last:30.200001 Bid:
2
```

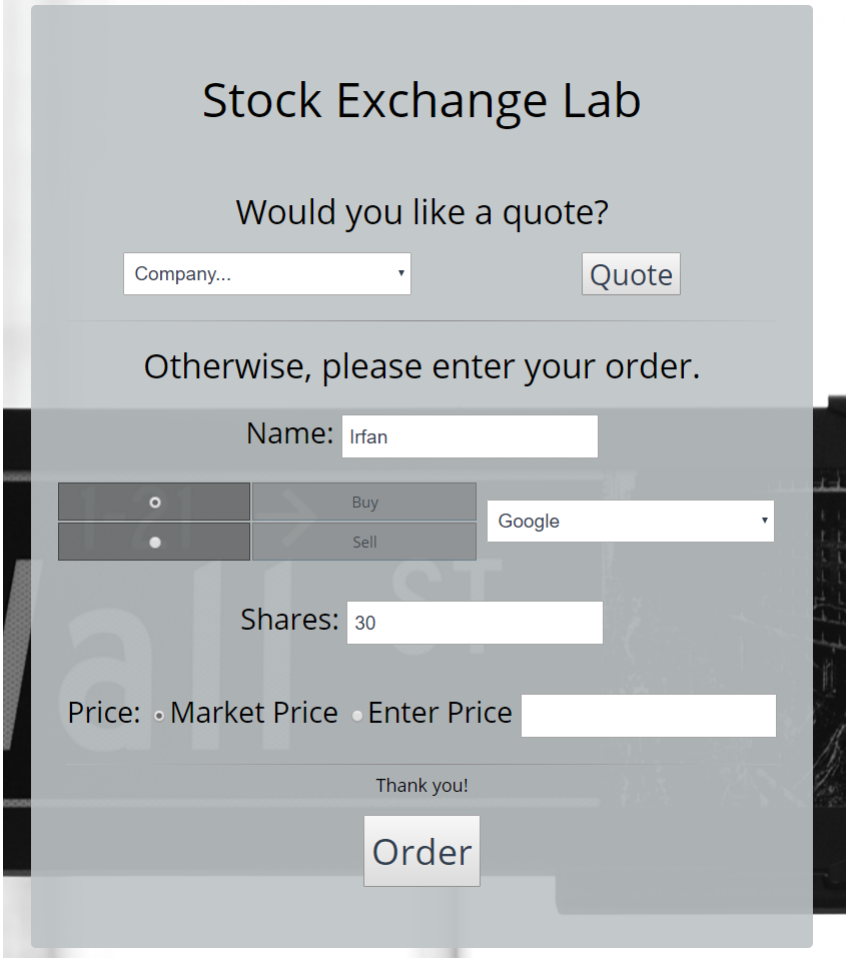


data.txt | dataBACKUP.txt

```
1 Google|14,13,Ted 10,12,Bob|30,43,Mark|Hi:35.320000 Low:12.320000 Last:15.000000 Bid:14.000000 Bi
2
```

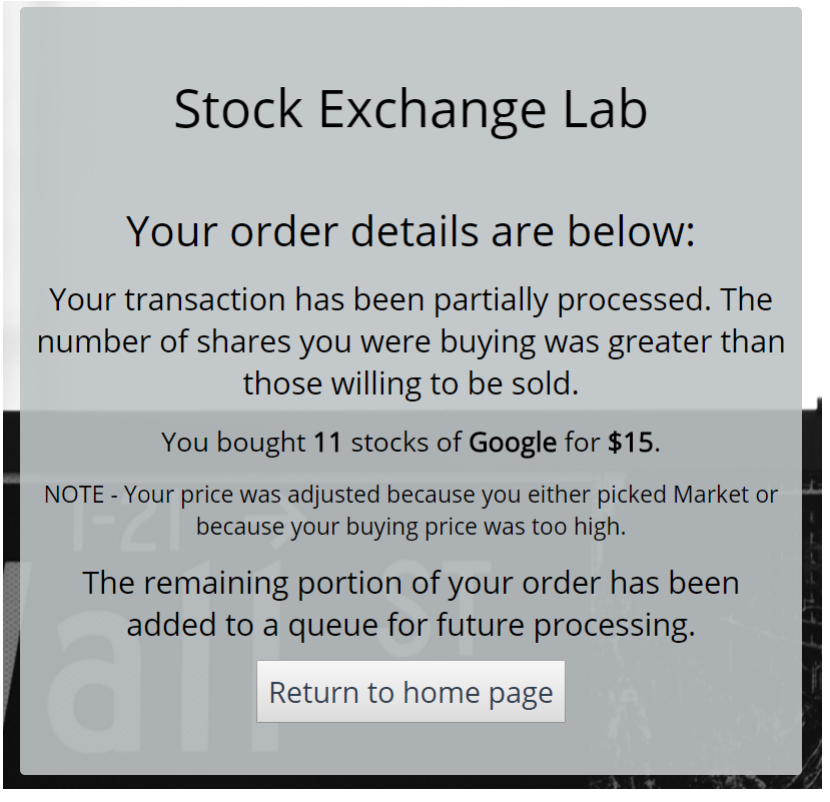
6 Test Part 4 (Partial Buy)

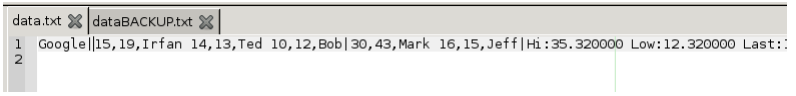
This test shows the senario where a partial buy occurs. When a partial buy happens, the price matches but the number of shares do not. The user is trying to buy more stocks than are being sold at any given price. When this happens, however many stocks can be bought are bought and the remaining portion of the order is added to the buy heap. The user should then be notified of these developments. The first image shows the submission form where the user is buying 30 shares of Google at Market Price. The second image shows the heaps before the transaction occurs. Clearly, Robert is only selling 11 shares of Google. Thus, the user should see a display stating that a partial buy has occured. The third image shows the result page. Here the user is notified that the transaction was partially procced because they were buying more shares than were being sold. It says they bought 11 stocks which is expected. Finally, the display says the remaining portion of the order has been added to the queue as expected. Looking at the fourth image, we see that the remaining portion of the user's order of 19 shares has indeed been added to the buy heap as expected.


Stock Exchange Lab
Would you like a quote?
Company...
Quote
Otherwise, please enter your order.
Name:
Irfan
Buy
Sell
Google
Shares:
30
Price:
Market Price
Enter Price
Thank you!
Order

data.txt | dataBACKUP.txt

```
1  Google|14,13,Ted 10,12,Bob|15,11,Robert 16,15,Jeff 30,43,Mark|Hi:35.320000 Low:12.320000 Last:
2
```

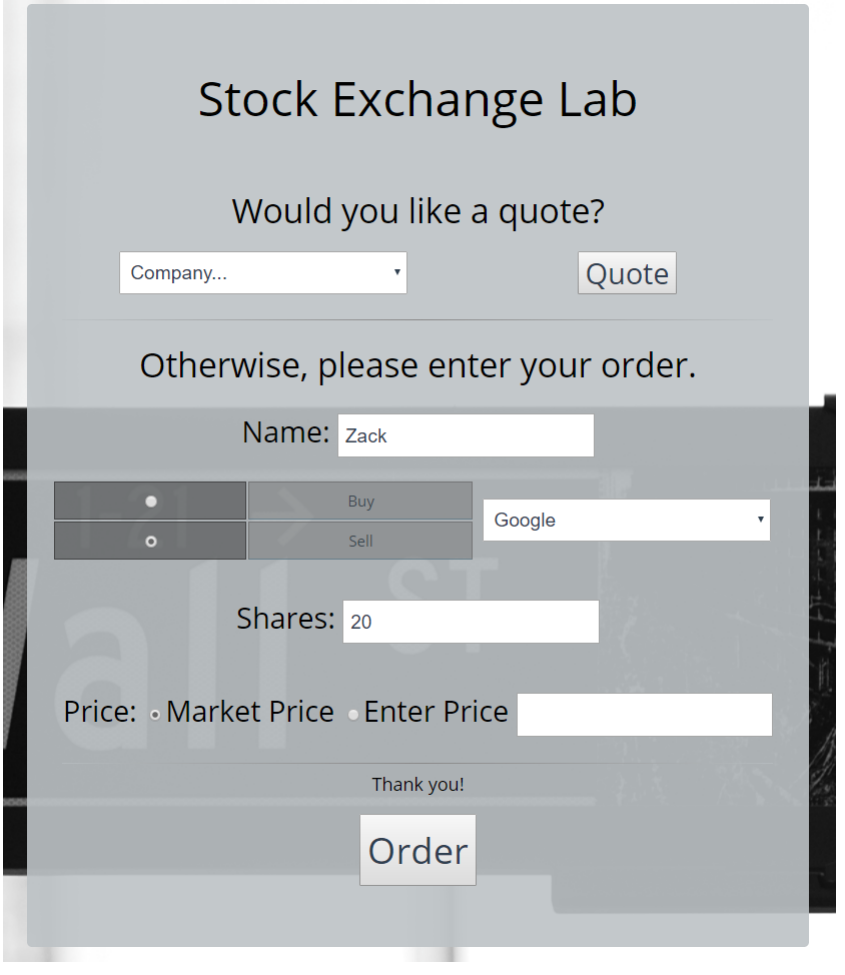


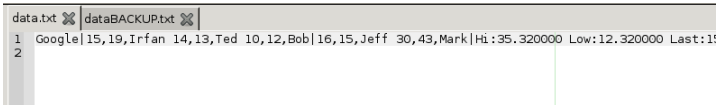

data.txt
dataBACKUP.txt
1 Google|15,19,Irfan 14,13,Ted 10,12,Bob|30,43,Mark 16,15,Jeff|Hi:35.320000 Low:12.320000 Last:
2

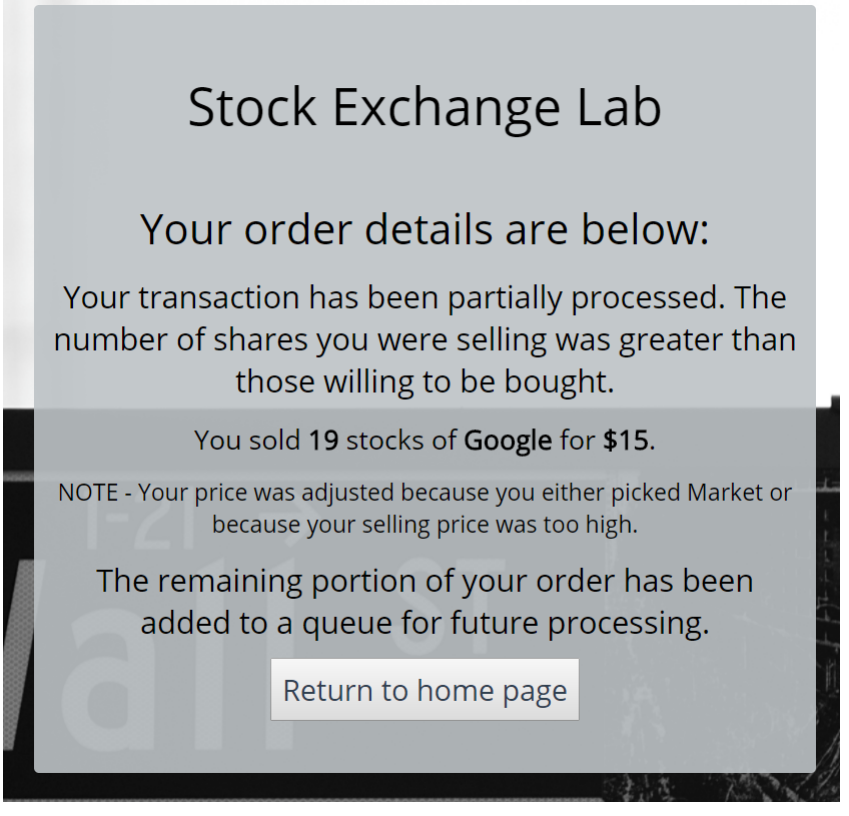
7 Test Part 5 (Partial Sell)

This test shows the senario where a partial sell occurs. When a partial sell happens, the price matches but the number of shares do not. The user is trying to sell more stocks than are being bought at any given price. When this happens, however many stocks can be sold are sold and the remaining portion of the order is added to the sell heap. The user should then be notified of these developments. The first image shows the submission form where the user is selling 20 shares of Google at Market Price. The second image shows the heaps before the transaction occurs. Clearly, Irfan is only buying 19 shares of Google. Thus, the user should see a display stating that a partial sell has occured. The third image shows the result page. Here the user is notified that the transaction was partially procced because they were selling more shares than were being bought. It says they sold 19 stocks which is expected. Finally, the display says the remaining portion of the order has been added to the queue as expected. Looking at the fourth image, we see that the remaining portion of the user's order of 1 share has indeed been added to the sell heap as expected.

This test case also demonstrates that the Market Price function works when the user is selling stocks also. From the second image, we see that when the user picks Market Price, their price should be set based off the buy heap's root node. In this case, that would mean a price of $15. In the third image, we see that the user did indeed sell stocks for $15.


Stock Exchange Lab
Would you like a quote?
Company...
Quote
Otherwise, please enter your order.
Name:
Zack
Buy
Sell
Google
Shares:
20
Price:
Market Price
Enter Price
Thank you!
Order

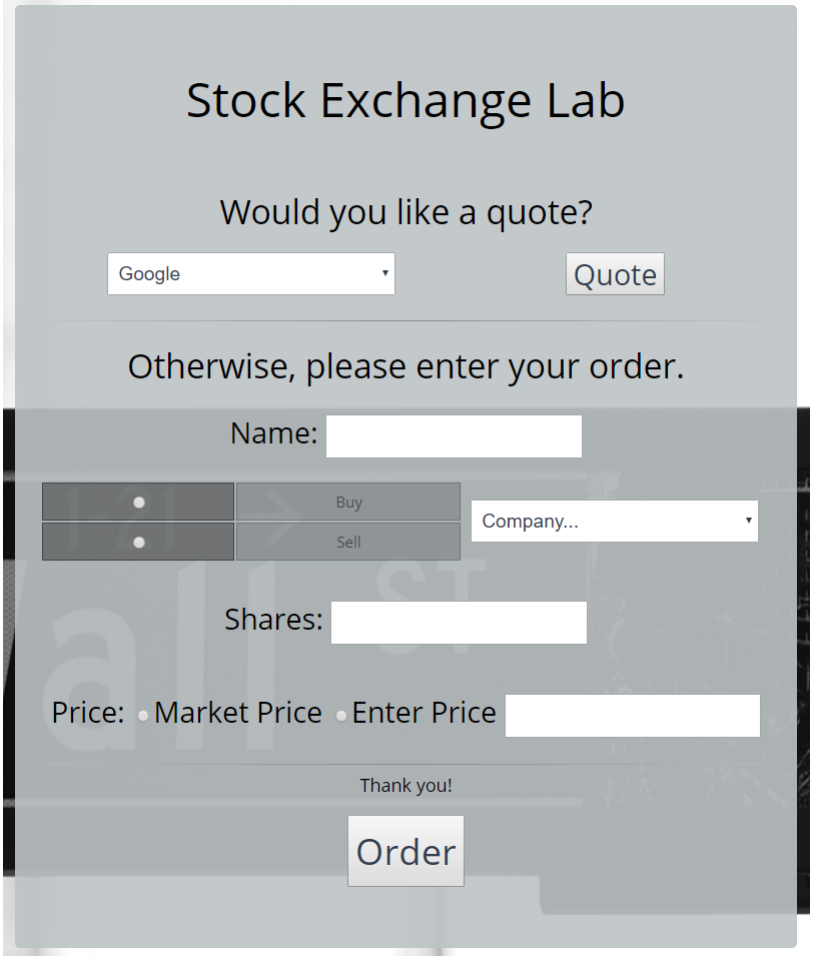

data.txt
dataBACKUP.txt
1 Google|15,19,Irfan 14,13,Ted 10,12,Bob|16,15,Jeff 30,43,Mark|Hi:35.320000 Low:12.320000 Last:1
2

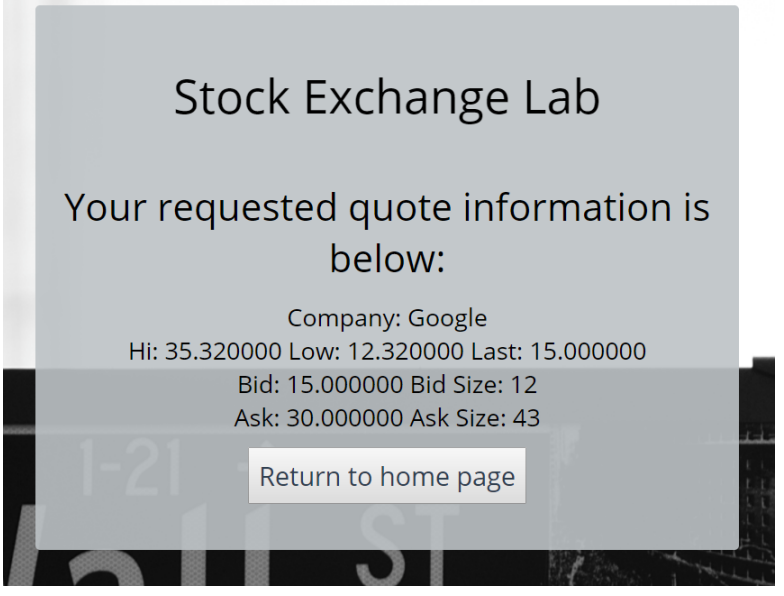

Stock Exchange Lab
Your order details are below:
Your transaction has been partially processed. The number of shares you were selling was greater than those willing to be bought.
You sold 19 stocks of Google for $15.
NOTE - Your price was adjusted because you either picked Market or because your selling price was too high.
The remaining portion of your order has been added to a queue for future processing.
Return to home page


data.txt
dataBACKUP.txt
1 Google|10,12,Bob 14,13,Ted|15,1,Zack 16,15,Jeff 30,43,Mark|Hi:35.320000 Low:12.3200
2

8 Test Part 6 (Quote)

This test shows the process of getting a Quote. When the user asks for a quote, they should get information about the highest, lowest, and last sale price. They should also get information on the top selling bid and top buying bid like the number of shares and price involved. In the first image, we see the user select Google for the company in the 'Quote' section. Then, in the second image, the user sees the quote information for Google. All the categories stated above are listed. Finally, the third image, confirms that the information displayed in the second image is correct.


Stock Exchange Lab
Would you like a quote?
Google
Quote
Otherwise, please enter your order.
Name:
Buy
Sell
Company...
Shares:
Price:
Market Price
Enter Price
Thank you!
Order



lab.dox | lab.h | data.txt

```
1 k|Hi:35.320000 Low:12.320000 Last:15.000000 Bid:15.000000 BidSize:12 Ask:30.000000 AskSize:43
2
```

9 Class Index

9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

10 File Index

10.1 File List

Here is a list of all files with brief descriptions:

11 Class Documentation

11.1 Heap Class Reference

```
#include <lab.h>
```

Public Member Functions

- Heap ()
- Heap (string heapType)
- ~Heap ()
- void setType (string type)
- string getType ()
- bool isEmpty ()
- bool isFull ()
- bool Insert (Order x)
- bool Remove (Order &x)
- Order peak ()

11.1.1 Constructor & Destructor Documentation

11.1.1.1 Heap::Heap ( ) [inline]

```
60          {
61              nNodes = 0;
62              size = 100;
63              buf = new Order[size];
64          }
```

11.1.1.2 Heap::Heap (string *heapType*) [inline]

```
66          {
67              heapType = heapType;
68              nNodes = 0;
69              size = 100;
70          }
```

11.1.1.3 Heap::~Heap ( )

```
4 {
5     //Destructor
6
7     //delete [] buf;
8 }
```

11.1.2 Member Function Documentation

11.1.2.1 string Heap::getType ( ) [inline]

```
77          {
78              return heapType;
79          }
```

11.1.2.2 bool Heap::Insert (Order *x*)

```
11 {
12     if (isFull())
13     {
14         return false;
```

```
15      }
16
17      nNodes++; //The last node of the heap is now vacant
18
19      //Starting from the (vacant) last node, go from node i to
20      //its parent iParent and, as long as the parent is smaller
21      //than x, move the parent down (MAX HEAP):
22
23      int i = nNodes;
24      int iParent;
25      while (i > 1)
26      {
27          iParent = i/2;
28          if(heapType == "buy")
29          {
30              if (x.price <= buf[iParent].price)
31              {
32                  break;
33              }
34          }
35          if(heapType == "sell")
36          {
37              if (x.price >= buf[iParent].price)
38              {
39                  break;
40              }
41          }
42          buf[i] = buf[iParent];  //Move the parent down
43          i = iParent;            //buf[i] is now vacant
44      }
45
46      //Insert x into the created vacancy:
```

```
47      buf[i] = x;
48
49      return true;
50  }
```

11.1.2.3 bool Heap::isEmpty ( ) [inline]

```
81          {
82              return bool(nNodes == 0);
83          }
```

11.1.2.4 bool Heap::isFull ( ) [inline]

```
85          {
86              return bool(nNodes == size);
87          }
```

11.1.2.5 Order Heap::peak ( ) [inline]

```
91          {
92              return buf[1];
93          }
```

11.1.2.6 bool Heap::Remove (Order & *x*)

```
53  {
54      //Removes the largest element (from the root of the heap).
55      //Retruns true if successful, false if the heap is empty.
56
57      if (isEmpty())
```

```
58      {
59          return false;
60      }
61
62      //Retreat top element:
63      x = buf[1];
64
65      //Starting from the vacant root, go from node iParent to its
66      //larger child i and, as long as the child is greater than the last
67      //element of the heap, move that child up:
68
69      int iParent = 1; //root
70      int i = 2; //its left child
71      while (i <= nNodes)
72      {
73          if (heapType == "buy")
74          {
75              //Set i to the right child, i+1, if it exists and is larger:
76              if (i < nNodes && buf[i].price < buf[i+1].price)
77              {
78                  i++;
79              }
80
81              //Compare with the last node:
82              if (buf[i].price <= buf[nNodes].price)
83              {
84                  break;
85              }
86          }
87          else if (heapType == "sell")
88          {
89              //Set i to the right child, i+1, if it exists and is larger:
```

```
90              if (i < nNodes && buf[i].price > buf[i+1].price)
91              {
92                  i++;
93              }
94
95              //Compare with the last node:
96              if (buf[i].price >= buf[nNodes].price)
97              {
98                  break;
99              }
100         }
101
102
103
104         buf[iParent] = buf[i]; //Move the child up;
105         iParent = 1;            //buf[iParent] is now vacant.
106         i *= 2;                 //i is set to its left child
107     }
108
109     //Move the last element into the created vacancy:
110     if (nNodes > 1)
111     {
112         buf[iParent] = buf[nNodes];
113     }
114
115     nNodes--;
116
117     return true;
118 }
```

11.1.2.7 void Heap::setType (string *type*) [inline]

```
73          {
74              heapType = type;
75          }
```

The documentation for this class was generated from the following files:

- lab.h
- lab.cpp

11.2 Order Struct Reference

This struct only contains key information about the user's order. That is, it includes their name, number of shares they want to buy/sell, and the price. This data would be taken from the Url struct. This struct is also the thing that our heaps are made of.

`#include <lab.h>`

Public Attributes

- string name
- int shares
- float price

11.2.1 Detailed Description

This struct only contains key information about the user's order. That is, it includes their name, number of shares they want to buy/sell, and the price. This data would be taken from the Url struct. This struct is also the thing that

our heaps are made of.

11.2.2 Member Data Documentation

11.2.2.1 string Order::name

11.2.2.2 float Order::price

11.2.2.3 int Order::shares

The documentation for this struct was generated from the following file:

- lab.h

11.3 Result Struct Reference

This struct stores some key information about the status of the transaction in the case that a user places an order. It stores information like whether the transaction went through or if it failed, whether the transaction ended up being a partial transaction or not, and whether the user's price was adjusted.

`#include <lab.h>`

Public Attributes

- bool success = false
- bool partial = false
- bool priceAdjust = false
- int shares = 0
- float price

11.3.1 Detailed Description

This struct stores some key information about the status of the transaction in the case that a user places an order. It stores information like whether the transaction went through or if it failed, whether the transaction ended up being a partial transaction or not, and whether the user's price was adjusted.

11.3.2 Member Data Documentation

11.3.2.1 bool Result::partial = false

11.3.2.2 float Result::price

11.3.2.3 bool Result::priceAdjust = false

11.3.2.4 int Result::shares = 0

11.3.2.5 bool Result::success = false

The documentation for this struct was generated from the following file:

- lab.h

11.4 Stock Class Reference

This class is our Stock class. It represents our companies within our program. Each object of this class in a company within our stock market. It contains two heaps that are used for managing and priortizing the orders the user's submit. One of the heaps is the buy heap and the other heap is the sell heap. The class also contains some other member variables to hold data such as the highest/lowest sale price, the last sale price, etc.

```
#include <lab.h>
```

Public Member Functions

- Stock ()
- void setData (string company, string hiSale, string lowSale, string lastSale, string currentBid, string bidSize, string currentAsk, string askSize)

  This function simply sets all the non-heap member variables of the Stock class to the appropriate value as passed in by parameter.
- string getData ()

  This function is responsible for returning the non-heap member variables as a string. It's used mainly for the writeData() function.
- string getDataHTML ()

  This function is responsible for returning a string of the non-heap member variables with HTML formatting. It is used by the resultPage() function if the user requested a quote.
- void insertHeap (Heap heap, string heapType)

  This function is responsible for assigning the heap passed in as a parameter to the correct member variable heap within the Stock object. Since we have two heaps, this function allows for a way to differeniate between the two heaps. It's mainly used for the readData() function.
- string getCompany ()

  This function returns the company member variable.
- Heap & getHeap (string type)

  This function is used to return one of the member heaps' address.
- float getLastSale ()

  This function returns the lastSale member variable.
- void updateData (float a)

  This function updates the highest, lowest, and last sale prices.
- void updateCurrent ()

  This function is used for updating the member variables related to the root nodes of both heap.

11.4.1 Detailed Description

This class is our Stock class. It represents our companies within our program. Each object of this class in a company within our stock market. It contains two heaps that are used for managing and priortizing the orders the user's submit. One of the heaps is the buy heap and the other heap is the sell heap. The class also contains some other member variables to hold data such as the highest/lowest sale price, the last sale price, etc.

11.4.2 Constructor & Destructor Documentation

11.4.2.1 Stock::Stock ( ) [inline]

```
112 : sellHeap("sell") , buyHeap("buy") { /*EMPTY CONSTRUCTOR BODY*/}
```

11.4.3 Member Function Documentation

11.4.3.1 string Stock::getCompany ( ) [inline]

This function returns the company member variable.

Returns

Returns a string containing the company member variable's value

```
147         {
148             return company;
149         }
```

11.4.3.2 string Stock::getData ( ) [inline]

This function is responsible for returning the non-heap member variables as a string. It's used mainly for the writeData() function.

Returns

Returns a string that contains the non-heap member variables

```
122         {
123             return ("Hi:" + to_string(hiSale) + " Low:" + to_string(lowSale) + " Last:" + to_
      lastSale) + " Bid:" + to_string(currentBid) + " BidSize:" + to_string(bidSize) + " Ask:" +
      " AskSize:" + to_string(askSize));
124         }
```

11.4.3.3 string Stock::getDataHTML ( ) [inline]

This function is responsible for returning a string of the non-heap member variables with HTML formatting. It is used by the resultPage() function if the user requested a quote.

Returns

Returns a string containing the non-heap member variables with HTML formatting.

```
135         {
136             return ("Company: " + company + "<br>Hi: " + to_string(hiSale) + " Low: " + to_st
      + " Last: " + to_string(lastSale) + "<br>Bid: " + to_string(currentBid) + " Bid Size: " + t
       + "<br>Ask: " + to_string(currentAsk) + " Ask Size: " + to_string(askSize));
137         }
```

11.4.3.4 Heap& Stock::getHeap (string *type*) [inline]

This function is used to return one of the member heaps' address.

Parameters

type	This variable contains the type of heap that needs to be returned, either the buy or sell heap.

Returns

Returns the appropriate heap by reference.

```
159         {
160             if (type == "buy")
161             {
162                 return buyHeap;
163             }
164             else if (type == "sell")
165             {
166                 return sellHeap;
167             }
168         }
```

11.4.3.5 float Stock::getLastSale ( ) [inline]

This function returns the lastSale member variable.

Returns

Returns the lastSale member variable

```
176         {
177             return lastSale;
178         }
```

11.4.3.6 void Stock::insertHeap (Heap *heap,* string *heapType*)

This function is responsible for assigning the heap passed in as a parameter to the correct member variable heap within the Stock object. Since we have two heaps, this function allows for a way to differeniate between the two heaps. It's mainly used for the readData() function.

Parameters

heap	The heap that needs to be equated to one of the member variable heaps.
heapType	This tells us if the heap being passed in is a sell heap or a buy heap.

```
130 {
131     if(heapType == "sell")
132     {
133         sellHeap = heap;
134     }
135     else if(heapType == "buy")
136     {
137         buyHeap = heap;
138     }
139 }
```

11.4.3.7 void Stock::setData (string *pCompany,* string *pHiSale,* string *pLowSale,* string *pLastSale,* string *pCurrentBid,* string *pBidSize,* string *pCurrentAsk,* string *pAskSize*)

This function simply sets all the non-heap member variables of the Stock class to the appropriate value as passed in by parameter.

Parameters

pCompany	This is the company the Stock object represents.
pHiSale	This is the highest sale price.
pLowSale	This is the lowest sale price.
pLastSale	This is the last sale price.
pCurrentBid	This is the root node's price amount of the buy heap.
pBidSize	This is the root node's share amount of the buy heap.
pCurrentAsk	This is the root node's price amount of the sell heap.
pAskSize	This is the root node's share amount of the sell heap.

```
155 {
156     company = pCompany;
157     hiSale = stof(pHiSale);
158     lowSale = stof(pLowSale);
159     lastSale = stof(pLastSale);
160     currentBid = stof(pCurrentBid);
161     bidSize = stoi(pBidSize);
162     currentAsk = stof(pCurrentAsk);
163     askSize = stoi(pAskSize);
164 }
```

11.4.3.8 void Stock::updateCurrent ( ) [inline]

This function is used for updating the member variables related to the root nodes of both heap.

```
204         {
205             currentBid = buyHeap.peak().price;
206             bidSize = buyHeap.peak().shares;
207             currentAsk = sellHeap.peak().price;
```

```
208             askSize = sellHeap.peak().shares;
209         }
```

11.4.3.9 void Stock::updateData (float *a*) [inline]

This function updates the highest, lowest, and last sale prices.

Parameters

a	This is the price of the last sale passed in which is used for updating the member variables.

```
187         {
188             lastSale = a;
189             if (lastSale > hiSale)
190             {
191                 hiSale = lastSale;
192             }
193             else if (lastSale < lowSale)
194             {
195                 lowSale = lastSale;
196             }
197         }
```

The documentation for this class was generated from the following files:

- lab.h
- lab.cpp

11.5 Url Struct Reference

This struct's function is to store the data passed in from the HTML url into member variables so they can be easily accessed. It stores things like whether the user is asking for a quote or placing an order (formType), what company they are interested in, information on shares, price, and whether they want to buy/sell.

`#include <lab.h>`

Public Attributes

- string action
- string company
- string shares
- string price
- string enterPrice
- string formType
- string name

11.5.1 Detailed Description

This struct's function is to store the data passed in from the HTML url into member variables so they can be easily accessed. It stores things like whether the user is asking for a quote or placing an order (formType), what company they are interested in, information on shares, price, and whether they want to buy/sell.

11.5.2 Member Data Documentation

11.5.2.1 string Url::action

11.5.2.2 string Url::company

11.5.2.3 string Url::enterPrice

11.5.2.4 string Url::formType

11.5.2.5 string Url::name

11.5.2.6 string Url::price

11.5.2.7 string Url::shares

The documentation for this struct was generated from the following file:

- lab.h

12 File Documentation

12.1 findCompany.cpp File Reference

```
#include "lab.h"
```

Functions

- int findCompany (vector< Stock > stock, string company)

  This function simply finds and returns the index at which one of the companies in the vector is located.

12.1.1 Function Documentation

12.1.1.1 int findCompany (vector< Stock > *stock*, string *company*)

This function simply finds and returns the index at which one of the companies in the vector is located.

Parameters

stock	This is the vector which the function will loop through to find the company in question.
company	This is the company being searched for.

```
4 {
5     for (int i = 0; i < stock.size(); i++)
6     {
7         if(stock[i].getCompany() == company)
8         {
```

```
9               return i;
10          }
11      }
12  }
```

12.2 lab.cpp File Reference

```
#include "lab.h"
```

12.3 lab.dox File Reference

12.4 lab.h File Reference

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <vector>
#include <fstream>
```

Classes

- struct Url

  This struct's function is to store the data passed in from the HTML url into member variables so they can be easily accessed. It stores things like whether the user is asking for a quote or placing an order (formType), what company they are interested in, information on shares, price, and whether they want to buy/sell.

- struct Result

  This struct stores some key information about the status of the transaction in the case that a user places an order. It stores information like whether the transaction went through or if it failed, whether the transaction ended up being a partial transaction or not, and whether the user's price was adjusted.

- struct Order

  This struct only contains key information about the user's order. That is, it includes their name, number of shares they want to buy/sell, and the price. This data would be taken from the Url struct. This struct is also the thing that our heaps are made of.

- class Heap
- class Stock

  This class is our Stock class. It represents our companies within our program. Each object of this class in a company within our stock market. It contains two heaps that are used for managing and priortizing the orders the user's submit. One of the heaps is the buy heap and the other heap is the sell heap. The class also contains some other member variables to hold data such as the highest/lowest sale price, the last sale price, etc.

Functions

- void readForm (Url &url)

  This function is responsible for parsing the user's input which is received by the program as a STRING_QUERY. It takes the information sent by the browser and parses out each piece of relavent data and stores it in our Url struct so the data can be used later in our program.

- void readData (vector< Stock > &stockMarket)

  This function is responsible for reading the data stored in the data.txt text file. That file stores the data between program executions since all data is lost once the program ends. Thus, it's stored in the data.txt file. Therefore, this function's purpose is to read the data back into the program from the data file at the start of the program. Each line of the file represents a Stock object. There are four different sections in each line, each of which are seperated by the '|' symbol. The function parses up until a '|' symbol and stores the data it read into the appropriate variable. Certain sections are further parsed to seperate groups of data. For example, the two heaps contain Order structs in them. Each of these are seperated by a space. Therefore, the function would also have to parse within the heaps,

separating the multiple Order structs. Therefore, this function parses on multiple levels to seperate and read all the relavent data back into the program.

- void writeData (vector< Stock > &stockMarket)

  This function is responsible for storing all the data within our program. Once our program ends, all the data it uses will be lost so unless it is stored in a text file, the next time the program runs, it will start from scratch and this is not something we want. Therefore, this function is responsible for writing all the data within our program to a text file. When doing so, it simply goes through each Stock object within the vector that is passed in as a parameter and writes its member variables. This means it writes things like the company name, the highest/lowest/last sale price, the various Order structs stored within our heaps, etc to the file.

- void transaction (Url url, vector< Stock > &stock, Result &result)

  This function is responsible for handling a transaction. If the user decides to place an order, this is the function that gets called. It takes in the user's inputed data and the stock market that contains all our different companies and walks through the apporpiate course of action. It first takes the user's data and creates an Order struct which it inputs into the correct heap of the correct company in question. Then, it determines if a transaction is possible based on if the price member of the Order struct of the root nodes of the heaps match. If this happens, it then looks at the shares to seperate a partial sale from a perfect sale and processes the transaction. The status of the transaction, things like if it was a success, if the sale was a partial sale, etc are also stored in another struct that was passed in as a parameter.

- int findCompany (vector< Stock > stock, string company)

  This function simply finds and returns the index at which one of the companies in the vector is located.

- void resultPage (Url url, Result result, string quote)

  This function is responsible for printing out the HTML code to display the results webpage. It displays the correct result based on what the user initially inputed. It looks at the initial input and determines whether to display text related to quotes or a transaction and then finds that info from the other parameters passed in and displays said info correctly.

12.4.1 Function Documentation

12.4.1.1 int findCompany (vector< Stock > *stock*, string *company*)

This function simply finds and returns the index at which one of the companies in the vector is located.

Parameters

stock	This is the vector which the function will loop through to find the company in question.
company	This is the company being searched for.

```
4 {
5     for (int i = 0; i < stock.size(); i++)
6     {
7         if(stock[i].getCompany() == company)
8         {
9             return i;
10         }
11     }
12 }
```

12.4.1.2 void readData (vector< Stock > & *stockMarket*)

This function is responsible for reading the data stored in the data.txt text file. That file stores the data between program executions since all data is lost once the program ends. Thus, it's stored in the data.txt file. Therefore, this function's purpose is to read the data back into the program from the data file at the start of the program. Each line of the file represents a Stock object. There are four different sections in each line, each of which are seperated by the '|' symbol. The function parses up until a '|' symbol and stores the data it read into the appropriate variable. Certain sections are further parsed to seperate groups of data. For example, the two heaps contain Order structs in them. Each of these are seperated by a space. Therefore, the function would also have to parse within the heaps, separating the multiple Order structs. Therefore, this function parses on multiple levels to seperate and read all the relavent data back into the program.

Parameters

stockMarket	This is a vector that is supposed to contain our multiple Stock objects (each of which represents a company). This vector is also passed in by reference. This is where all the parsed data will be stored in.

```
4 {
5     //Pass in vector (by reference) to hold data, each index has stock object which itself has l
6     //and thus we hold all the data
7     //Create a stock and heap object and add it into our vector
8     //
9     //Sample data.txt line:
10     //company|BUYHEAP(MAXHEAP) Amount,Shares,Name Amount,Shares,Name Amount,Shares,Name|SELLHEA
       Amount,Shares,Name Amount,Shares,Name|DATA Hi:Amount Low:Amount Last:Amount Bid:Amount Bids
       Ask:Amount AskSize:Amount
11
12     //cout << "Test 1" << endl;
13
14     ifstream ifs("/home/debian/cs-124/final/data.txt");
15     if(!ifs)
16     {
17         cout << "ERROR - File failed to open." << endl;
18     }
19
20     string line;
21     while(getline(ifs, line))
22     {
23         //cout << "Test 2" << endl;
24         string heap, order, tempString;
25         string company, hiSale, lowSale, lastSale, currentBid, bidSize, currentAsk, askSize;
26         Heap tempBuyHeap;
27         Heap tempSellHeap;
```

```
28          Stock tempStock;
29
30          //cout << "Test 3" << endl;
31
32          stringstream ss(line);
33
34          getline(ss, company, '|');
35
36          //cout << "Test 4" << endl;
37
38          for(int i = 0; i < 2; i++)
39          {
40              //cout << "Test 5" << endl;
41
42              getline(ss, heap, '|');
43
44              if (i == 0)
45              {
46                  tempBuyHeap.setType("buy");
47              }
48              else if (i == 1)
49              {
50                  tempSellHeap.setType("sell");
51              }
52
53              //cout << "Test 6" << endl;
54
55              stringstream ssHeap(heap);
56              while(getline(ssHeap, order, ' '))
57              {
58                  //cout << "Test 6.1" << endl;
59                  Order tempOrder;
```

```
60                  string temp;
61
62                  //cout << "Test 6.2" << endl;
63
64                  stringstream ssOrder(order);
65                  getline(ssOrder, temp, ',');
66                  tempOrder.price = stof(temp);
67                  getline(ssOrder, temp, ',');
68                  tempOrder.shares = stoi(temp);
69                  getline(ssOrder, tempOrder.name);
70
71                  //cout << "Test 6.3" << endl;
72                  //cout << tempHeap.getType() << endl;
73
74                  //cout << tempOrder.shares << tempOrder.price << tempOrder.name << endl;
75
76                  if (i == 0)
77                  {
78                      tempBuyHeap.Insert(tempOrder);
79                  }
80                  else if (i == 1)
81                  {
82                      tempSellHeap.Insert(tempOrder);
83                  }
84
85                  //cout << "Test 7" << endl;
86              }
87
88              if(i == 0)
89              {
90                  tempStock.insertHeap(tempBuyHeap, "buy"); //Store heap in tempStock before
    overriding for min heap
```

```
91              }
92              else if (i == 1)
93              {
94                  tempStock.insertHeap(tempSellHeap, "sell"); //Store heap in tempStock before
    overriding for min heap
95              }
96          }
97
98          //cout << "Test 8" << endl;
99
100          while (ss)
101          {
102              //cout << "Test 9" << endl;
103
104              getline(ss, tempString, ':');
105
106              if (tempString == "Hi")
107              {
108                  getline(ss, tempString, ' ');
109                  hiSale = tempString;
110              }
111              else if (tempString == "Low")
112              {
113                  getline(ss, tempString, ' ');
114                  lowSale = tempString;
115              }
116              else if (tempString == "Last")
117              {
118                  getline(ss, tempString, ' ');
119                  lastSale = tempString;
120              }
121              else if (tempString == "Bid")
```

```
122             {
123                 getline(ss, tempString, ' ');
124                 currentBid = tempString;
125             }
126             else if (tempString == "BidSize")
127             {
128                 getline(ss, tempString, ' ');
129                 bidSize = tempString;
130             }
131             else if (tempString == "Ask")
132             {
133                 getline(ss, tempString, ' ');
134                 currentAsk = tempString;
135             }
136             else if (tempString == "AskSize")
137             {
138                 getline(ss, tempString, ' ');
139                 askSize = tempString;
140             }
141         }
142 
143         //cout << "Test 10" << endl;
144 
145         tempStock.setData(company, hiSale, lowSale, lastSale, currentBid, bidSize, currentAsk
    askSize);
146         stockMarket.push_back(tempStock);
147 
148         //cout << "Test 11" << endl;
149     }
150 
151     //cout << "Finished getting data." << endl;
152     ifs.close();
```

```
153 }
```

12.4.1.3 void readForm (Url & *url*)

This function is responsible for parsing the user's input which is received by the program as a STRING_QUERY. It takes the information sent by the browser and parses out each piece of relavent data and stores it in our Url struct so the data can be used later in our program.

Parameters

url	This member variable is the Url struct that our data will be stored it. It is also passed in by reference.

```
4 {
5     string s = getenv("QUERY_STRING"); //EX: s may be "option=Encode&Message=Hello+there" after this li
6     string tempString;
7
8     stringstream ss;
9
10     ss << s;
11
12     while(ss)
13     {
14         getline(ss, tempString, '='); //Go up till =
15
16         if(tempString == "action")  //If that's an option go up till &
17         {
18             getline(ss, url.action, '&');
19         }
20         else if (tempString == "company")
21         {
22             getline(ss, url.company, '&');
```

```
23          }
24          else if (tempString == "shares")
25          {
26              getline(ss, url.shares, '&');
27          }
28          else if (tempString == "price")
29          {
30              getline(ss, url.price, '&');
31          }
32          else if (tempString == "enter_price")
33          {
34              getline(ss, url.enterPrice, '&');
35          }
36          else if (tempString == "formType")
37          {
38              getline(ss, url.formType, '&');
39          }
40          else if (tempString == "name")
41          {
42              getline(ss, url.name, '&');
43          }
44
45          //~ else if(tempString == "Message") //If that's a message, in a loop go up till +
46          //~ {                                  //to parse entire message
47              //~ while(getline(ss, tempString, '+'))
48              //~ {
49                  //~ if(tempString == "%2F")
50                  //~ {
51                      //~ pMessage += "/ ";
52                  //~ }
53                  //~ else
54                  //~ {
```

```
55                      //~ pMessage += tempString + " ";
56                  //~ }
57              //~ }
58              //~ pMessage = pMessage.substr(0, pMessage.length() - 1);
59          //~ }
60      }
61  }
```

12.4.1.4 void resultPage (Url *url*, Result *result*, string *quote*)

This function is responsible for printing out the HTML code to display the results webpage. It displays the correct result based on what the user initially inputed. It looks at the initial input and determines whether to display text related to quotes or a transaction and then finds that info from the other parameters passed in and displays said info correctly.

Parameters

url	This is the user's input stored in a Url struct
result	This is the Result struct that contains info on the transaction status from the transaction function
quote	This variable contains a string with the quote data in case the user asked for a quote instead of submitting an order

```
4  {
5      cout << "<html>"
6      "    <head>"
7      "            <title>Stock Exchange Lab</title>  <!--Tab title -->"
8      "            <link href=\"https://fonts.googleapis.com/css?family=Open+Sans\" rel=\"stylesheet\">"
9      "            <script>"
10      "                function goBack(){"
11      "                    window.history.back()"
```

```
12      "                 }"
13      "             </script>"
14      "             <style type=\"text/css\">         "         /*CSS to make the HTML form look p
15      "                 body{"
16      "                     background-image: url(/cs-124/final/rick-tap-110126.jpg);"
17      "                     background-size: cover;"
18      "                     background-repeat: no-repeat;"
19      "                 }"
20
21      "                 input{"
22      "                     padding: 10px 10px;"
23      "                     color: #2c3e50;"
24      "                 }"
25
26      "                 input[type=submit]{"
27      "                     font-family: 'Open Sans', Century Gothic, Helvetica, Geneva, sans-ser
28      "                     font-size: 1.5vw;"
29      "                 }"
30
31      "                 div{"
32      "                     background-color: rgba(189, 195, 199, .9);"
33      "                     display: inherit;"
34      "                     margin: auto;"
35      "                     width: 40%;"
36      "                     padding: 13px;"
37      "                     border-radius: 5px;"
38      "                     font-size: 20px;"
39      "                     font-size: 2vw;"
40      "                     font-family: 'Open Sans', Century Gothic, Helvetica, Geneva, sans-ser
41      "                 }"
42
43      "                 img{"
```

```
44      "                   max-width:100%;"
45      "               }"
46
47      "               table{"
48      "                   width: auto; "
49      "                   table-layout: fixed;"
50      "               }"
51
52      "               .tableH{"
53      "                   background-color: rgba(60,136,126,0.6);"
54      "                   border: 1px solid rgba(60,136,126,1);"
55      "                   padding: 15px 30px;"
56      "                   text-align: center;"
57      "               }"
58
59      "               th, .dblue{"
60      "                   background-color: rgba(34,68,85,0.5);"
61      "                   padding: 15px 30px; "
62      "                   text-align: center; "
63      "                   font-weight: 500; "
64      "                   font-size: 12px;"
65      "                   color: #fff; "
66      "                   text-transform: uppercase; "
67      "               }"
68
69      "               .dblue{ "
70      "                   padding: 8px;            "
71      "                   border: 1px solid rgba(34,68,85,0.7);"
72      "               } "
73
74      "               .lblue{"
75      "                   background-color: rgba(119,170,170,0.5); "
```

```
76      "                    height:3px; "
77      "                    overflow-x:auto; "
78      "                    margin-top: 0px; "
79      "                    border: 1px solid rgba(85,119,136,0.7);"
80      "                    padding: 8px;"
81      "                    text-align: center; "
82      "                    vertical-align: middle; "
83      "                    font-weight: 300;"
84      "                    font-size: 15px; "
85      "                    color: #37474f;"
86      "                    border-bottom: solid 1px rgba(85,119,136,0.7); "
87      "                } "
88
89      "                td{ "
90      "                    height:3px; "
91      "                    overflow-x:auto; "
92      "                    margin-top: 0px;"
93      "                    padding: 8px; "
94      "                    text-align: center; "
95      "                    vertical-align: middle; "
96      "                    font-weight: 300; "
97      "                    font-size: 15px; "
98      "                }"
99
100      "                 hr{"
101      "                     border: 0;"
102      "                     height: 1px;"
103      "                     background-image: -webkit-linear-gradient(left, rgba(189, 195, 199,
       rgba(189, 195, 199, .6));"
104      "                     background-image: -moz-linear-gradient(left, rgba(189, 195, 199, .6)
       rgba(189, 195, 199, .6));"
105      "                     background-image: -ms-linear-gradient(left, rgba(189, 195, 199, .6),
```

```
      rgba(189, 195, 199, .6));"
106    "                    background-image: -o-linear-gradient(left, rgba(189, 195, 199, .6), #8c8b8b,
      rgba(189, 195, 199, .6));"
107    "                }"
108    "            </style>"
109    "        </head>"
110
111    "        <body>    "
112    "            <div>  <!--HTML Form -->"
113    "            <form action=\"/cs-124/final/stockOriginal.html\" method=\"GET\">  <!--Forward user'
      input to the cgi file -->"
114    "                <center>"
115    "                   <p style=\"font-size: 2.7vw; padding-bottom: 0.5%\">Stock Exchange Lab</p>";
116
117     if (url.formType == "Quote")
118     {
119         cout << "<p style=\"font-size: 2.2vw; margin-top: 3px; margin-bottom: 15px; padding-top: 0.5%
      >Your requested quote information is below:</p>" << endl;
120         cout << "<p style=\"font-size: 1.4vw; margin-top: 3px; margin-bottom: 15px; padding-top: 0.5%
      < quote << "</p>";
121
122     }
123     else if (url.formType == "Order")
124     {
125         cout << "<p style=\"font-size: 2.2vw; margin-top: 3px; margin-bottom: 15px; padding-top: 0.5%
      >Your order details are below:</p>" << endl;
126
127         if (result.success)
128         {
129             if (result.partial)
130             {
131                 cout << "<p style=\"font-size: 1.6vw; margin-top: 3px; margin-bottom: 15px; padding-t
```

```
      0.5%\">Your transaction has been partially processed. ";
132                 if (url.action == "Buy")
133                 {
134                     cout << "The number of shares you were buying was greater than those will
      sold.</p>" << endl;
135                     cout << "<p style=\"font-size: 1.4vw; margin-top: 3px; margin-bottom: 15px
      padding-top: 0.5%\">You bought <b>" << (stoi(url.shares) - result.shares) << "</b> stocks o
     company << "</b> for <b>$" << result.price << "</b>.</p>" << endl;
136                     if (result.priceAdjust)
137                     {
138                         cout << "<p style=\"font-size: 1.2vw; margin-top: 3px; margin-bottom:
      padding-top: 0.5%\">NOTE - Your price was adjusted because you either picked Market or beca
      was too high. </p>" << endl;
139                     }
140                     cout << "<p style=\"font-size: 1.6vw; margin-top: 3px; margin-bottom: 15px
      padding-top: 0.5%\">The remaining portion of your order has been added to a queue for futur
141                 }
142                 else if (url.action == "Sell")
143                 {
144                     cout << "The number of shares you were selling was greater than those will
      bought.</p>" << endl;
145                     cout << "<p style=\"font-size: 1.4vw; margin-top: 3px; margin-bottom: 15px
      padding-top: 0.5%\">You sold <b>" << (stoi(url.shares) - result.shares) << "</b> stocks of
     company << "</b> for <b>$" << result.price << "</b>.</p>" << endl;
146                     if (result.priceAdjust)
147                     {
148                         cout << "<p style=\"font-size: 1.2vw; margin-top: 3px; margin-bottom:
      padding-top: 0.5%\">NOTE - Your price was adjusted because you either picked Market or beca
      was too high. </p>" << endl;
149                     }
150                     cout << "<p style=\"font-size: 1.6vw; margin-top: 3px; margin-bottom: 15px
      padding-top: 0.5%\">The remaining portion of your order has been added to a queue for futur
```

```
151                 }
152             }
153             else
154             {
155                 cout << "<p style=\"font-size: 1.6vw; margin-top: 3px; margin-bottom: 15px; padding-t
    0.5%\">Your transaction was a success! The details are as follow:</p>" << endl;
156                 if (url.action == "Buy")
157                 {
158                     cout << "<p style=\"font-size: 1.4vw; margin-top: 3px; margin-bottom: 15px;
    padding-top: 0.5%\">You bought <b>" << url.shares << "</b> stocks of <b>" << url.
    company << "</b> for <b>$" << result.price << "</b>.</p>" << endl;
159                     if (result.priceAdjust)
160                     {
161                         cout << "<p style=\"font-size: 1.2vw; margin-top: 3px; margin-bottom: 15px;
    padding-top: 0.5%\">NOTE - Your price was adjusted because you either picked Market or because you
    was too high. </p>" << endl;
162                     }
163                 }
164                 else if (url.action == "Sell")
165                 {
166                     cout << "<p style=\"font-size: 1.4vw; margin-top: 3px; margin-bottom: 15px;
    padding-top: 0.5%\">You sold <b>" << url.shares << "</b> stocks of <b>" << url.company << "</b> fo
    << result.price << "</b>.</p>" << endl;
167                     if (result.priceAdjust)
168                     {
169                         cout << "<p style=\"font-size: 1.2vw; margin-top: 3px; margin-bottom: 15px;
    padding-top: 0.5%\">NOTE - Your price was adjusted because you either picked Market or because you
    was too high. </p>" << endl;
170                     }
171                 }
172             }
173         }
```

```
174         else
175         {
176             cout << "<p style=\"font-size: 1.6vw; margin-top: 3px; margin-bottom: 15px; paddi
      >Your transaction did not get processed. It has been added to a queue for future processing
177         }
178 
179     }
180 
181 
182 //    "                <p style=\"font-size: 2.5vw; padding-bottom: 0.5%\">Stock Exchange Lab<
183 //    "                <table>"
184 //    "                    <tr><td colspan=\"4\" style=\"font-size: 1.7vw; margin-top: 3px; ma
      15px; padding-top: 0.5%\">Your Message & Translation</td></tr>"
185 //    "                    <tr><td class=\"tableH\" style=\"font-size: 22px;  border-collapse:
      collapse;\">Message</td> <td class=\"tableH\" style=\"font-size: 22px; margin-top: 3px; mar
      border-collapse: collapse;\">"
186 
187 //    /*<< message << */"</td>";
188 
189 //   //cout <<
190 //    "                    <tr><td class=\"dblue\"> Translation </td> <td class=\"lblue\">"
191 //    /*<< translation << */"</td>";
192 
193 
194     cout <<
195 //    "                    <tr><td colspan=\"4\"><hr><p style=\"font-size: 18px; margin-top: 7
      you! </td></tr></p>"
196 //    "                    <tr><td colspan=\"4\"><input class=\"button\" type=\"submit\" value=
      home page\"></td></tr>"
197 //    "                </table>     "
198     "                  <input class=\"button\" type=\"submit\" value=\"Return to home page\"
199     "               </center>"
```

```
200     "            </form>"
201     "            </div>"
202     "        </body>"
203     "        </html>";
204
205 }
```

12.4.1.5 void transaction (Url *url*, vector< Stock > & *stock*, Result & *result*)

This function is responsible for handling a transaction. If the user decides to place an order, this is the function that gets called. It takes in the user's inputed data and the stock market that contains all our different companies and walks through the apporpiate course of action. It first takes the user's data and creates an Order struct which it inputs into the correct heap of the correct company in question. Then, it determines if a transaction is possible based on if the price member of the Order struct of the root nodes of the heaps match. If this happens, it then looks at the shares to seperate a partial sale from a perfect sale and processes the transaction. The status of the transaction, things like if it was a success, if the sale was a partial sale, etc are also stored in another struct that was passed in as a parameter.

Parameters

url	This struct contains the parsed user input which was received from the HTML page url.
stock	This is our stock vector that contains the Stock objects of all the different companies.
result	This is the struct the transaction status information will be stored in.

```
4 {
5     int i = findCompany(stock, url.company);
6
7     if(url.action == "Buy")
8     {
9         Order buy;
10         buy.name = url.name;
```

```
11          buy.shares = stoi(url.shares);
12          if (url.price != "Market")
13          {
14              buy.price = stof(url.enterPrice);
15          }
16          result.price = buy.price;
17          //use market price if the buying price is high than the lowest selling price
18          //in the sellheap
19          if (url.price == "Market" && stock[i].getHeap("sell").isEmpty())
20          {
21              buy.price = stock[i].getLastSale();
22              result.priceAdjust = true;
23              result.price = buy.price;
24          }
25          else if(url.price == "Market" || (!stock[i].getHeap("sell").isEmpty() && buy.
    price > stock[i].getHeap("sell").peak().price))
26          {
27              buy.price = stock[i].getHeap("sell").peak().price;
28              result.priceAdjust = true;
29              result.price = buy.price;
30          }
31
32          stock[i].getHeap("buy").Insert(buy);
33      }
34      else if (url.action == "Sell")
35      {
36          Order sell;
37          sell.name = url.name;
38          sell.shares = stoi(url.shares);
39          if (url.price != "Market")
40          {
41              sell.price = stof(url.enterPrice);
```

```
42          }
43          result.price = sell.price;
44          if(url.price == "Market" && stock[i].getHeap("buy").isEmpty())
45          {
46              sell.price = stock[i].getLastSale();
47              result.priceAdjust = true;
48              result.price = sell.price;
49          }
50
51          else if(url.price == "Market" || (!stock[i].getHeap("buy").isEmpty() && sell.
    price > stock[i].getHeap("buy").peak().price))
52          {
53              sell.price = stock[i].getHeap("buy").peak().price;
54              result.priceAdjust = true;
55              result.price = sell.price;
56          }
57
58          stock[i].getHeap("sell").Insert(sell);
59      }
60
61      if((!stock[i].getHeap("buy").isEmpty() && !stock[i].getHeap("sell").isEmpty()) && stock[i].getHeap
    ").peak().price == stock[i].getHeap("sell").peak().price)
62      {
63          Order buy, sell;
64          //Remove both top nodes of buy and sell
65          stock[i].getHeap("buy").Remove(buy);
66          stock[i].getHeap("sell").Remove(sell);
67          //compare the share
68          //if buy has larger share, push the rest buy share back
69          //if sell has larger share, push the rest sell share back
70          //if equal, don't push back
71          if(buy.shares > sell.shares)
```

```
72          {
73              buy.shares -= sell.shares;
74              stock[i].getHeap("buy").Insert(buy);
75              stock[i].updateData(buy.price);
76
77
78              if (buy.name == url.name)
79              {
80                  result.success = true;
81                  result.partial = true;
82                  if (url.action == "Buy")
83                  {
84                      result.shares = buy.shares;
85                  }
86                  else if (url.action == "Sell")
87                  {
88                      result.shares = 0;
89                  }
90              }
91              else if (sell.name == url.name)
92              {
93                  result.success = true;
94                  result.partial = true;
95                  if (url.action == "Buy")
96                  {
97                      result.shares = buy.shares;
98                  }
99                  else if (url.action == "Sell")
100                  {
101                      result.shares = 0;
102                  }
103              }
```

```
104
105         }
106         else if(buy.shares < sell.shares)
107         {
108             sell.shares -= buy.shares;
109             stock[i].getHeap("sell").Insert(sell);
110             stock[i].updateData(sell.price);
111             result.partial = true;
112
113             if (buy.name == url.name)
114             {
115                 result.success = true;
116                 result.partial = true;
117                 if (url.action == "Buy")
118                 {
119                     result.shares = 0;
120                 }
121                 else if (url.action == "Sell")
122                 {
123                     result.shares = sell.shares;
124                 }
125             }
126             else if (sell.name == url.name)
127             {
128                 result.success = true;
129                 result.partial = true;
130                 if (url.action == "Buy")
131                 {
132                     result.shares = 0;
133                 }
134                 else if (url.action == "Sell")
135                 {
```

```
136                     result.shares = sell.shares;
137                 }
138             }
139         }
140         else
141         {
142             stock[i].updateData(buy.price);
143             result.success = true;
144         }
145     }
146 
147     stock[i].updateCurrent();
148 }
```

12.4.1.6 void writeData (vector< Stock > & *stockMarket*)

This function is responsible for storing all the data within our program. Once our program ends, all the data it uses will be lost so unless it is stored in a text file, the next time the program runs, it will start from scratch and this is not something we want. Therefore, this function is responsible for writing all the data within our program to a text file. When doing so, it simply goes through each Stock object within the vector that is passed in as a parameter and writes its member variables. This means it writes things like the company name, the highest/lowest/last sale price, the various Order structs stored within our heaps, etc to the file.

Parameters

stockMarket	This is the vector of data we need to write to the file so it can be stored for future program executions.

```
4 {
5     //Go through each index of the vector; at each index, there is a Stock
6     //object; write all the information stored in all the member variables
```

```
7     //into the file, all the quote related data and the two heaps using
8     //the following format:
9     //company|BUYHEAP(MAXHEAP) Amount,Shares,Name Amount,Shares,Name Amount,Shares,Name|SELLHEAP(MINHEA
       Amount,Shares,Name Amount,Shares,Name|DATA Hi:Amount Low:Amount Last:Amount Bid:Amount BidSize:Amo
       Ask:Amount AskSize:Amount
10
11     ofstream ofs("/home/debian/cs-124/final/data.txt");
12     for (int i = 0; i < stockMarket.size(); i++)
13     {
14         ofs << stockMarket[i].getCompany() << "|";
15
16         //DOES IT MATTER WHAT ORDER WE WRITE THE HEAP TO THE FILE?
17         //SINCE WHEN WE READ IT BACK IN AND INSERT WOULDN'T IT REHEAP ANYWAYS?
18         Heap heap = stockMarket[i].getHeap("buy");
19         Order order;
20         long pos;
21
22         if (!heap.isEmpty())
23         {
24             while(heap.Remove(order))
25             {
26                 ofs << order.price << "," << order.shares << "," << order.
      name << " ";
27             }
28
29             pos = ofs.tellp();
30             ofs.seekp(pos-1);
31             ofs << "|";
32         }
33         else
34         {
35             ofs << "|";
```

```
36          }
37
38          heap = stockMarket[i].getHeap("sell");
39
40          if (!heap.isEmpty())
41          {
42              while(heap.Remove(order))
43              {
44                  ofs << order.price << "," << order.shares << "," << order.
    name << " ";
45              }
46
47              pos = ofs.tellp();
48              ofs.seekp(pos-1);
49              ofs << "|";
50          }
51          else
52          {
53              ofs << "|";
54          }
55
56          ofs << fixed << setprecision(2) << stockMarket[i].getData();
57          ofs << '\n';
58      }
59
60      ofs.close();
61  }
```

12.5 main.cpp File Reference

```
#include "lab.h"
```

Functions

- int main ()

  This function is the main function. It first calls the readForm() function to parse the HTML url input. Then, it calls the readData() function to read the data from the text file. Then, it processes the user's request, saves the programs data to the file and finally calls the resultPage() function to display the results webpage.

12.5.1 Function Documentation

12.5.1.1 int main ( )

This function is the main function. It first calls the readForm() function to parse the HTML url input. Then, it calls the readData() function to read the data from the text file. Then, it processes the user's request, saves the programs data to the file and finally calls the resultPage() function to display the results webpage.

Parameters

pOption	This variable, which is passed in by reference, will hold the option the user chose, whether encode or decode.

pMessage	This variable, which is also passed in by reference, will hold the user's message.

```
15 {
16     Url url; //Struct that will contain parsed url data
17     Result result; //Struct that contains the transaction status
18     readForm(url); //Populating the struct with the HTML URL data
19     string quote;
20 
21     vector<Stock> stockMarket;
22 
23     //cout << "Test 1" << endl;
24     readData(stockMarket);
25     if (url.formType == "Quote")
26     {
27         quote = stockMarket[findCompany(stockMarket, url.company)].getDataHTML();
28     }
29     else if (url.formType == "Order")
30     {
31         transaction(url, stockMarket, result);
32     }
33 
34     //cout << "Test 2" << endl;
35     writeData(stockMarket);
36     //cout << "Test 3" << endl;
37     resultPage(url, result, quote);
38 
39     /*cout << "Name: " << url.name << endl;
40     cout << "Action: " << url.action << endl;
41     cout << "Company: " << url.company << endl;
42     cout << "Shares: " << url.shares << endl;
43     cout << "Price: " << url.price << endl;
44     cout << "Enter Price: " << url.enterPrice << endl;
```

```
45      cout << "Form Type: " << url.formType << endl;
46      */
47  }
```

12.6 readData.cpp File Reference

```
#include "lab.h"
```

Functions

- void readData (vector< Stock > &stockMarket)

  This function is responsible for reading the data stored in the data.txt text file. That file stores the data between program executions since all data is lost once the program ends. Thus, it's stored in the data.txt file. Therefore, this function's purpose is to read the data back into the program from the data file at the start of the program. Each line of the file represents a Stock object. There are four different sections in each line, each of which are seperated by the '|' symbol. The function parses up until a '|' symbol and stores the data it read into the appropriate variable. Certain sections are further parsed to seperate groups of data. For example, the two heaps contain Order structs in them. Each of these are seperated by a space. Therefore, the function would also have to parse within the heaps, separating the multiple Order structs. Therefore, this function parses on multiple levels to seperate and read all the relavent data back into the program.

12.6.1 Function Documentation

12.6.1.1 void readData (vector< Stock > & *stockMarket*)

This function is responsible for reading the data stored in the data.txt text file. That file stores the data between program executions since all data is lost once the program ends. Thus, it's stored in the data.txt file. Therefore,

this function's purpose is to read the data back into the program from the data file at the start of the program. Each line of the file represents a Stock object. There are four different sections in each line, each of which are seperated by the '|' symbol. The function parses up until a '|' symbol and stores the data it read into the appropriate variable. Certain sections are further parsed to seperate groups of data. For example, the two heaps contain Order structs in them. Each of these are seperated by a space. Therefore, the function would also have to parse within the heaps, separating the multiple Order structs. Therefore, this function parses on multiple levels to seperate and read all the relavent data back into the program.

Parameters

stockMarket	This is a vector that is supposed to contain our multiple Stock objects (each of which represents a company). This vector is also passed in by reference. This is where all the parsed data will be stored in.

```
4 {
5     //Pass in vector (by reference) to hold data, each index has stock object which itself has l
6     //and thus we hold all the data
7     //Create a stock and heap object and add it into our vector
8     //
9     //Sample data.txt line:
10     //company|BUYHEAP(MAXHEAP) Amount,Shares,Name Amount,Shares,Name Amount,Shares,Name|SELLHEA
       Amount,Shares,Name Amount,Shares,Name|DATA Hi:Amount Low:Amount Last:Amount Bid:Amount Bids
       Ask:Amount AskSize:Amount
11
12     //cout << "Test 1" << endl;
13
14     ifstream ifs("/home/debian/cs-124/final/data.txt");
15     if(!ifs)
16     {
17         cout << "ERROR - File failed to open." << endl;
18     }
19
```

```
20      string line;
21      while(getline(ifs, line))
22      {
23          //cout << "Test 2" << endl;
24          string heap, order, tempString;
25          string company, hiSale, lowSale, lastSale, currentBid, bidSize, currentAsk, askSize;
26          Heap tempBuyHeap;
27          Heap tempSellHeap;
28          Stock tempStock;
29
30          //cout << "Test 3" << endl;
31
32          stringstream ss(line);
33
34          getline(ss, company, '|');
35
36          //cout << "Test 4" << endl;
37
38          for(int i = 0; i < 2; i++)
39          {
40              //cout << "Test 5" << endl;
41
42              getline(ss, heap, '|');
43
44              if (i == 0)
45              {
46                  tempBuyHeap.setType("buy");
47              }
48              else if (i == 1)
49              {
50                  tempSellHeap.setType("sell");
51              }
```

```
52
53              //cout << "Test 6" << endl;
54
55              stringstream ssHeap(heap);
56              while(getline(ssHeap, order, ' '))
57              {
58                  //cout << "Test 6.1" << endl;
59                  Order tempOrder;
60                  string temp;
61
62                  //cout << "Test 6.2" << endl;
63
64                  stringstream ssOrder(order);
65                  getline(ssOrder, temp, ',');
66                  tempOrder.price = stof(temp);
67                  getline(ssOrder, temp, ',');
68                  tempOrder.shares = stoi(temp);
69                  getline(ssOrder, tempOrder.name);
70
71                  //cout << "Test 6.3" << endl;
72                  //cout << tempHeap.getType() << endl;
73
74                  //cout << tempOrder.shares << tempOrder.price << tempOrder.name << endl;
75
76                  if (i == 0)
77                  {
78                      tempBuyHeap.Insert(tempOrder);
79                  }
80                  else if (i == 1)
81                  {
82                      tempSellHeap.Insert(tempOrder);
83                  }
```

```
84
85                  //cout << "Test 7" << endl;
86              }
87
88              if(i == 0)
89              {
90                  tempStock.insertHeap(tempBuyHeap, "buy"); //Store heap in tempStock before
    overriding for min heap
91              }
92              else if (i == 1)
93              {
94                  tempStock.insertHeap(tempSellHeap, "sell"); //Store heap in tempStock before
    overriding for min heap
95              }
96          }
97
98          //cout << "Test 8" << endl;
99
100          while (ss)
101          {
102              //cout << "Test 9" << endl;
103
104              getline(ss, tempString, ':');
105
106              if (tempString == "Hi")
107              {
108                  getline(ss, tempString, ' ');
109                  hiSale = tempString;
110              }
111              else if (tempString == "Low")
112              {
113                  getline(ss, tempString, ' ');
```

```
114                 lowSale = tempString;
115             }
116             else if (tempString == "Last")
117             {
118                 getline(ss, tempString, ' ');
119                 lastSale = tempString;
120             }
121             else if (tempString == "Bid")
122             {
123                 getline(ss, tempString, ' ');
124                 currentBid = tempString;
125             }
126             else if (tempString == "BidSize")
127             {
128                 getline(ss, tempString, ' ');
129                 bidSize = tempString;
130             }
131             else if (tempString == "Ask")
132             {
133                 getline(ss, tempString, ' ');
134                 currentAsk = tempString;
135             }
136             else if (tempString == "AskSize")
137             {
138                 getline(ss, tempString, ' ');
139                 askSize = tempString;
140             }
141         }
142 
143         //cout << "Test 10" << endl;
144 
145         tempStock.setData(company, hiSale, lowSale, lastSale, currentBid, bidSize, currentAsk
```

```
      askSize);
146         stockMarket.push_back(tempStock);
147
148         //cout << "Test 11" << endl;
149     }
150
151     //cout << "Finished getting data." << endl;
152     ifs.close();
153 }
```

12.7 readForm.cpp File Reference

```
#include "lab.h"
```

Functions

- void readForm (Url &url)

  This function is responsible for parsing the user's input which is received by the program as a STRING_QUERY. It takes the information sent by the browser and parses out each piece of relavent data and stores it in our Url struct so the data can be used later in our program.

12.7.1 Function Documentation

12.7.1.1 void readForm (Url & *url*)

This function is responsible for parsing the user's input which is received by the program as a STRING_QUERY. It takes the information sent by the browser and parses out each piece of relavent data and stores it in our Url struct

so the data can be used later in our program.

Parameters

url	This member variable is the Url struct that our data will be stored it. It is also passed in by reference.

```
4 {
5     string s = getenv("QUERY_STRING"); //EX: s may be "option=Encode&Message=Hello+there" after this li
6     string tempString;
7
8     stringstream ss;
9
10     ss << s;
11
12     while(ss)
13     {
14         getline(ss, tempString, '='); //Go up till =
15
16         if(tempString == "action")  //If that's an option go up till &
17         {
18             getline(ss, url.action, '&');
19         }
20         else if (tempString == "company")
21         {
22             getline(ss, url.company, '&');
23         }
24         else if (tempString == "shares")
25         {
26             getline(ss, url.shares, '&');
27         }
28         else if (tempString == "price")
29         {
30             getline(ss, url.price, '&');
```

```
31          }
32          else if (tempString == "enter_price")
33          {
34              getline(ss, url.enterPrice, '&');
35          }
36          else if (tempString == "formType")
37          {
38              getline(ss, url.formType, '&');
39          }
40          else if (tempString == "name")
41          {
42              getline(ss, url.name, '&');
43          }
44
45          //~ else if(tempString == "Message") //If that's a message, in a loop go up till +
46          //~ {                                          //to parse entire message
47              //~ while(getline(ss, tempString, '+'))
48              //~ {
49                  //~ if(tempString == "%2F")
50                  //~ {
51                      //~ pMessage += "/ ";
52                  //~ }
53                  //~ else
54                  //~ {
55                      //~ pMessage += tempString + " ";
56                  //~ }
57              //~ }
58              //~ pMessage = pMessage.substr(0, pMessage.length() - 1);
59          //~ }
60      }
61  }
```

12.8 resultPage.cpp File Reference

```
#include "lab.h"
```

Functions

- void resultPage (Url url, Result result, string quote)

  This function is responsible for printing out the HTML code to display the results webpage. It displays the correct result based on what the user initially inputed. It looks at the initial input and determines whether to display text related to quotes or a transaction and then finds that info from the other parameters passed in and displays said info correctly.

12.8.1 Function Documentation

12.8.1.1 void resultPage (Url *url*, Result *result*, string *quote*)

This function is responsible for printing out the HTML code to display the results webpage. It displays the correct result based on what the user initially inputed. It looks at the initial input and determines whether to display text related to quotes or a transaction and then finds that info from the other parameters passed in and displays said info correctly.

Parameters

url	This is the user's input stored in a Url struct

result	This is the Result struct that contains info on the transaction status from the transaction function
quote	This variable contains a string with the quote data in case the user asked for a quote instead of submitting an order

```
4 {
5     cout << "<html>"
6     "    <head>"
7     "            <title>Stock Exchange Lab</title>  <!--Tab title -->"
8     "            <link href=\"https://fonts.googleapis.com/css?family=Open+Sans\" rel=\"styleshe
9     "            <script>"
10     "                function goBack(){"
11     "                    window.history.back()"
12     "                 }"
13     "            </script>"
14     "            <style type=\"text/css\">         "         /*CSS to make the HTML form look p
15     "                body{"
16     "                    background-image: url(/cs-124/final/rick-tap-110126.jpg);"
17     "                    background-size: cover;"
18     "                    background-repeat: no-repeat;"
19     "                }"
20
21     "                input{"
22     "                    padding: 10px 10px;"
23     "                    color: #2c3e50;"
24     "                }"
25
26     "                input[type=submit]{"
27     "                    font-family: 'Open Sans', Century Gothic, Helvetica, Geneva, sans-seri
28     "                    font-size: 1.5vw;"
29     "                }"
```

```
30
31      "                div{"
32      "                    background-color: rgba(189, 195, 199, .9);"
33      "                    display: inherit;"
34      "                    margin: auto;"
35      "                    width: 40%;"
36      "                    padding: 13px;"
37      "                    border-radius: 5px;"
38      "                    font-size: 20px;"
39      "                    font-size: 2vw;"
40      "                    font-family: 'Open Sans', Century Gothic, Helvetica, Geneva, sans-serif;"
41      "                }"
42
43      "                img{"
44      "                    max-width:100%;"
45      "                }"
46
47      "                table{"
48      "                    width: auto; "
49      "                    table-layout: fixed;"
50      "                }"
51
52      "                .tableH{"
53      "                    background-color: rgba(60,136,126,0.6);"
54      "                    border: 1px solid rgba(60,136,126,1);"
55      "                    padding: 15px 30px;"
56      "                    text-align: center;"
57      "                }"
58
59      "                th, .dblue{"
60      "                    background-color: rgba(34,68,85,0.5);"
61      "                    padding: 15px 30px; "
```

```
62      "                   text-align: center; "
63      "                   font-weight: 500; "
64      "                   font-size: 12px;"
65      "                   color: #fff; "
66      "                   text-transform: uppercase; "
67      "               }"
68
69      "               .dblue{ "
70      "                   padding: 8px;            "
71      "                   border: 1px solid rgba(34,68,85,0.7);"
72      "               } "
73
74      "               .lblue{"
75      "                   background-color: rgba(119,170,170,0.5); "
76      "                   height:3px; "
77      "                   overflow-x:auto; "
78      "                   margin-top: 0px; "
79      "                   border: 1px solid rgba(85,119,136,0.7);"
80      "                   padding: 8px;"
81      "                   text-align: center; "
82      "                   vertical-align: middle; "
83      "                   font-weight: 300;"
84      "                   font-size: 15px; "
85      "                   color: #37474f;"
86      "                   border-bottom: solid 1px rgba(85,119,136,0.7); "
87      "               } "
88
89      "               td{ "
90      "                   height:3px; "
91      "                   overflow-x:auto; "
92      "                   margin-top: 0px;"
93      "                   padding: 8px; "
```

```
94      "                    text-align: center; "
95      "                    vertical-align: middle; "
96      "                    font-weight: 300; "
97      "                    font-size: 15px; "
98      "                }"
99
100      "                hr{"
101      "                    border: 0;"
102      "                    height: 1px;"
103      "                    background-image: -webkit-linear-gradient(left, rgba(189, 195, 199, .6), #8
       rgba(189, 195, 199, .6));"
104      "                    background-image: -moz-linear-gradient(left, rgba(189, 195, 199, .6), #8c8b8
       rgba(189, 195, 199, .6));"
105      "                    background-image: -ms-linear-gradient(left, rgba(189, 195, 199, .6), #8c8b8b
       rgba(189, 195, 199, .6));"
106      "                    background-image: -o-linear-gradient(left, rgba(189, 195, 199, .6), #8c8b8b,
       rgba(189, 195, 199, .6));"
107      "                }"
108      "            </style>"
109      "        </head>"
110
111      "        <body>    "
112      "            <div>  <!--HTML Form -->"
113      "            <form action=\"/cs-124/final/stockOriginal.html\" method=\"GET\">  <!--Forward user'
       input to the cgi file -->"
114      "                <center>"
115      "                    <p style=\"font-size: 2.7vw; padding-bottom: 0.5%\">Stock Exchange Lab</p>";
116
117     if (url.formType == "Quote")
118     {
119         cout << "<p style=\"font-size: 2.2vw; margin-top: 3px; margin-bottom: 15px; padding-top: 0.5%
       >Your requested quote information is below:</p>" << endl;
```

```
120         cout << "<p style=\"font-size: 1.4vw; margin-top: 3px; margin-bottom: 15px; padding-to
    < quote << "</p>";
121
122     }
123     else if (url.formType == "Order")
124     {
125         cout << "<p style=\"font-size: 2.2vw; margin-top: 3px; margin-bottom: 15px; padding-to
    >Your order details are below:</p>" << endl;
126
127         if (result.success)
128         {
129             if (result.partial)
130             {
131                 cout << "<p style=\"font-size: 1.6vw; margin-top: 3px; margin-bottom: 15px; pa
     0.5%\">Your transaction has been partially processed. ";
132                 if (url.action == "Buy")
133                 {
134                     cout << "The number of shares you were buying was greater than those will
     sold.</p>" << endl;
135                     cout << "<p style=\"font-size: 1.4vw; margin-top: 3px; margin-bottom: 15px
     padding-top: 0.5%\">You bought <b>" << (stoi(url.shares) - result.shares) << "</b> stocks o
    company << "</b> for <b>$" << result.price << "</b>.</p>" << endl;
136                     if (result.priceAdjust)
137                     {
138                         cout << "<p style=\"font-size: 1.2vw; margin-top: 3px; margin-bottom:
     padding-top: 0.5%\">NOTE - Your price was adjusted because you either picked Market or beca
     was too high. </p>" << endl;
139                     }
140                     cout << "<p style=\"font-size: 1.6vw; margin-top: 3px; margin-bottom: 15px
     padding-top: 0.5%\">The remaining portion of your order has been added to a queue for futur
141                 }
142                 else if (url.action == "Sell")
```

```
143                 {
144                     cout << "The number of shares you were selling was greater than those willing to
    bought.</p>" << endl;
145                     cout << "<p style=\"font-size: 1.4vw; margin-top: 3px; margin-bottom: 15px;
    padding-top: 0.5%\">You sold <b>" << (stoi(url.shares) - result.shares) << "</b> stocks of <b>" <<
    company << "</b> for <b>$" << result.price << "</b>.</p>" << endl;
146                     if (result.priceAdjust)
147                     {
148                         cout << "<p style=\"font-size: 1.2vw; margin-top: 3px; margin-bottom: 15px;
    padding-top: 0.5%\">NOTE - Your price was adjusted because you either picked Market or because you
    was too high. </p>" << endl;
149                     }
150                     cout << "<p style=\"font-size: 1.6vw; margin-top: 3px; margin-bottom: 15px;
    padding-top: 0.5%\">The remaining portion of your order has been added to a queue for future proce
151                 }
152             }
153             else
154             {
155                 cout << "<p style=\"font-size: 1.6vw; margin-top: 3px; margin-bottom: 15px; padding-t
    0.5%\">Your transaction was a success! The details are as follow:</p>" << endl;
156                 if (url.action == "Buy")
157                 {
158                     cout << "<p style=\"font-size: 1.4vw; margin-top: 3px; margin-bottom: 15px;
    padding-top: 0.5%\">You bought <b>" << url.shares << "</b> stocks of <b>" << url.
    company << "</b> for <b>$" << result.price << "</b>.</p>" << endl;
159                     if (result.priceAdjust)
160                     {
161                         cout << "<p style=\"font-size: 1.2vw; margin-top: 3px; margin-bottom: 15px;
    padding-top: 0.5%\">NOTE - Your price was adjusted because you either picked Market or because you
    was too high. </p>" << endl;
162                     }
163                 }
```

```
164                 else if (url.action == "Sell")
165                 {
166                     cout << "<p style=\"font-size: 1.4vw; margin-top: 3px; margin-bottom: 15px
       padding-top: 0.5%\">You sold <b>" << url.shares << "</b> stocks of <b>" << url.company << "
      << result.price << "</b>.</p>" << endl;
167                     if (result.priceAdjust)
168                     {
169                         cout << "<p style=\"font-size: 1.2vw; margin-top: 3px; margin-bottom:
       padding-top: 0.5%\">NOTE - Your price was adjusted because you either picked Market or beca
       was too high. </p>" << endl;
170                     }
171                 }
172             }
173         }
174         else
175         {
176             cout << "<p style=\"font-size: 1.6vw; margin-top: 3px; margin-bottom: 15px; paddi
      >Your transaction did not get processed. It has been added to a queue for future processing
177         }
178 
179     }
180 
181 
182 //    "                <p style=\"font-size: 2.5vw; padding-bottom: 0.5%\">Stock Exchange Lab<
183 //    "                <table>"
184 //    "                    <tr><td colspan=\"4\" style=\"font-size: 1.7vw; margin-top: 3px; ma
       15px; padding-top: 0.5%\">Your Message & Translation</td></tr>"
185 //    "                    <tr><td class=\"tableH\" style=\"font-size: 22px;  border-collapse:
       collapse;\">Message</td> <td class=\"tableH\" style=\"font-size: 22px; margin-top: 3px; mar
       border-collapse: collapse;\">"
186 
187 //    /*<< message << */"</td>";
```

```
188
189 //   //cout <<
190 //    "                    <tr><td class=\"dblue\"> Translation </td> <td class=\"lblue\">"
191 //    /*<< translation << */"</td>";
192
193
194     cout <<
195 //    "                    <tr><td colspan=\"4\"><hr><p style=\"font-size: 18px; margin-top: 7px;\">
      you! </td></tr></p>"
196 //    "                    <tr><td colspan=\"4\"><input class=\"button\" type=\"submit\" value=\"Retur
      home page\"></td></tr>"
197 //    "                </table>     "
198     "                  <input class=\"button\" type=\"submit\" value=\"Return to home page\">"
199     "                </center>"
200     "            </form>"
201     "            </div>"
202     "        </body>"
203     "        </html>";
204
205 }
```

12.9 transaction.cpp File Reference

```
#include "lab.h"
```

Functions

- void transaction (Url url, vector< Stock > &stock, Result &result)

  This function is responsible for handling a transaction. If the user decides to place an order, this is the function

that gets called. It takes in the user's inputed data and the stock market that contains all our different companies and walks through the apporpiate course of action. It first takes the user's data and creates an Order struct which it inputs into the correct heap of the correct company in question. Then, it determines if a transaction is possible based on if the price member of the Order struct of the root nodes of the heaps match. If this happens, it then looks at the shares to seperate a partial sale from a perfect sale and processes the transaction. The status of the transaction, things like if it was a success, if the sale was a partial sale, etc are also stored in another struct that was passed in as a parameter.

12.9.1 Function Documentation

12.9.1.1 void transaction (Url *url*, vector< Stock > & *stock*, Result & *result*)

This function is responsible for handling a transaction. If the user decides to place an order, this is the function that gets called. It takes in the user's inputed data and the stock market that contains all our different companies and walks through the apporpiate course of action. It first takes the user's data and creates an Order struct which it inputs into the correct heap of the correct company in question. Then, it determines if a transaction is possible based on if the price member of the Order struct of the root nodes of the heaps match. If this happens, it then looks at the shares to seperate a partial sale from a perfect sale and processes the transaction. The status of the transaction, things like if it was a success, if the sale was a partial sale, etc are also stored in another struct that was passed in as a parameter.

Parameters

url	This struct contains the parsed user input which was received from the HTML page url.
stock	This is our stock vector that contains the Stock objects of all the different companies.
result	This is the struct the transaction status information will be stored in.

```
4 {
5     int i = findCompany(stock, url.company);
6
7     if(url.action == "Buy")
```

```
8     {
9         Order buy;
10         buy.name = url.name;
11         buy.shares = stoi(url.shares);
12         if (url.price != "Market")
13         {
14             buy.price = stof(url.enterPrice);
15         }
16         result.price = buy.price;
17         //use market price if the buying price is high than the lowest selling price
18         //in the sellheap
19         if (url.price == "Market" && stock[i].getHeap("sell").isEmpty())
20         {
21             buy.price = stock[i].getLastSale();
22             result.priceAdjust = true;
23             result.price = buy.price;
24         }
25         else if(url.price == "Market" || (!stock[i].getHeap("sell").isEmpty() && buy.
    price > stock[i].getHeap("sell").peak().price))
26         {
27             buy.price = stock[i].getHeap("sell").peak().price;
28             result.priceAdjust = true;
29             result.price = buy.price;
30         }
31
32         stock[i].getHeap("buy").Insert(buy);
33     }
34     else if (url.action == "Sell")
35     {
36         Order sell;
37         sell.name = url.name;
38         sell.shares = stoi(url.shares);
```

```
39          if (url.price != "Market")
40          {
41              sell.price = stof(url.enterPrice);
42          }
43          result.price = sell.price;
44          if(url.price == "Market" && stock[i].getHeap("buy").isEmpty())
45          {
46              sell.price = stock[i].getLastSale();
47              result.priceAdjust = true;
48              result.price = sell.price;
49          }
50
51          else if(url.price == "Market" || (!stock[i].getHeap("buy").isEmpty() && sell.
    price > stock[i].getHeap("buy").peak().price))
52          {
53              sell.price = stock[i].getHeap("buy").peak().price;
54              result.priceAdjust = true;
55              result.price = sell.price;
56          }
57
58          stock[i].getHeap("sell").Insert(sell);
59      }
60
61      if((!stock[i].getHeap("buy").isEmpty() && !stock[i].getHeap("sell").isEmpty()) && stock[i]
    ").peak().price == stock[i].getHeap("sell").peak().price)
62      {
63          Order buy, sell;
64          //Remove both top nodes of buy and sell
65          stock[i].getHeap("buy").Remove(buy);
66          stock[i].getHeap("sell").Remove(sell);
67          //compare the share
68          //if buy has larger share, push the rest buy share back
```

```
69          //if sell has larger share, push the rest sell share back
70          //if equal, don't push back
71          if(buy.shares > sell.shares)
72          {
73              buy.shares -= sell.shares;
74              stock[i].getHeap("buy").Insert(buy);
75              stock[i].updateData(buy.price);
76
77
78              if (buy.name == url.name)
79              {
80                  result.success = true;
81                  result.partial = true;
82                  if (url.action == "Buy")
83                  {
84                      result.shares = buy.shares;
85                  }
86                  else if (url.action == "Sell")
87                  {
88                      result.shares = 0;
89                  }
90              }
91              else if (sell.name == url.name)
92              {
93                  result.success = true;
94                  result.partial = true;
95                  if (url.action == "Buy")
96                  {
97                      result.shares = buy.shares;
98                  }
99                  else if (url.action == "Sell")
100                  {
```

```
101                     result.shares = 0;
102                 }
103             }
104 
105         }
106         else if(buy.shares < sell.shares)
107         {
108             sell.shares -= buy.shares;
109             stock[i].getHeap("sell").Insert(sell);
110             stock[i].updateData(sell.price);
111             result.partial = true;
112 
113             if (buy.name == url.name)
114             {
115                 result.success = true;
116                 result.partial = true;
117                 if (url.action == "Buy")
118                 {
119                     result.shares = 0;
120                 }
121                 else if (url.action == "Sell")
122                 {
123                     result.shares = sell.shares;
124                 }
125             }
126             else if (sell.name == url.name)
127             {
128                 result.success = true;
129                 result.partial = true;
130                 if (url.action == "Buy")
131                 {
132                     result.shares = 0;
```

```
133                 }
134                 else if (url.action == "Sell")
135                 {
136                     result.shares = sell.shares;
137                 }
138             }
139         }
140         else
141         {
142             stock[i].updateData(buy.price);
143             result.success = true;
144         }
145     }
146 
147     stock[i].updateCurrent();
148 }
```

12.10 writeData.cpp File Reference

```
#include "lab.h"
```

Functions

- void writeData (vector< Stock > &stockMarket)

  This function is responsible for storing all the data within our program. Once our program ends, all the data it uses will be lost so unless it is stored in a text file, the next time the program runs, it will start from scratch and this is not something we want. Therefore, this function is responsible for writing all the data within our program to a text file. When doing so, it simply goes through each Stock object within the vector that is passed in as a parameter and

writes its member variables. This means it writes things like the company name, the highest/lowest/last sale price, the various Order structs stored within our heaps, etc to the file.

12.10.1 Function Documentation

12.10.1.1 void writeData (vector< Stock > & *stockMarket*)

This function is responsible for storing all the data within our program. Once our program ends, all the data it uses will be lost so unless it is stored in a text file, the next time the program runs, it will start from scratch and this is not something we want. Therefore, this function is responsible for writing all the data within our program to a text file. When doing so, it simply goes through each Stock object within the vector that is passed in as a parameter and writes its member variables. This means it writes things like the company name, the highest/lowest/last sale price, the various Order structs stored within our heaps, etc to the file.

Parameters

stockMarket	This is the vector of data we need to write to the file so it can be stored for future program executions.

```
4 {
5     //Go through each index of the vector; at each index, there is a Stock
6     //object; write all the information stored in all the member variables
7     //into the file, all the quote related data and the two heaps using
8     //the following format:
9     //company|BUYHEAP(MAXHEAP) Amount,Shares,Name Amount,Shares,Name Amount,Shares,Name|SELLHEAP
       Amount,Shares,Name Amount,Shares,Name|DATA Hi:Amount Low:Amount Last:Amount Bid:Amount Bid
       Ask:Amount AskSize:Amount
10
11     ofstream ofs("/home/debian/cs-124/final/data.txt");
12     for (int i = 0; i < stockMarket.size(); i++)
13     {
```

```
14          ofs << stockMarket[i].getCompany() << "|";
15
16          //DOES IT MATTER WHAT ORDER WE WRITE THE HEAP TO THE FILE?
17          //SINCE WHEN WE READ IT BACK IN AND INSERT WOULDN'T IT REHEAP ANYWAYS?
18          Heap heap = stockMarket[i].getHeap("buy");
19          Order order;
20          long pos;
21
22          if (!heap.isEmpty())
23          {
24              while(heap.Remove(order))
25              {
26                  ofs << order.price << "," << order.shares << "," << order.
    name << " ";
27              }
28
29              pos = ofs.tellp();
30              ofs.seekp(pos-1);
31              ofs << "|";
32          }
33          else
34          {
35              ofs << "|";
36          }
37
38          heap = stockMarket[i].getHeap("sell");
39
40          if (!heap.isEmpty())
41          {
42              while(heap.Remove(order))
43              {
44                  ofs << order.price << "," << order.shares << "," << order.
```

```
      name << " ";
45              }
46
47              pos = ofs.tellp();
48              ofs.seekp(pos-1);
49              ofs << "|";
50          }
51          else
52          {
53              ofs << "|";
54          }
55
56          ofs << fixed << setprecision(2) << stockMarket[i].getData();
57          ofs << '\n';
58      }
59
60      ofs.close();
61  }
```

Index