

## Stock Market - Group Project

Generated by Doxygen 1.8.8

Fri Dec 8 2017 21:02:39

## Contents

1	Specification	2
2	Analysis	3
3	Design	4
4	HTML	5
5	Images/Features	13
6	Images/Features2	14
7	TestCases1	15
8	TestCases2	16
9	TestCases3	17
10	TestCases4	18
11	TestCases5	19
12	TestCase6	20

13	TestCase7	21
14	TestCase8	22
15	TestCase9	23
16	TestCase10	24
17	TestCase11	25
18	TestCase12	26
19	TestCase13	27
20	TestCase14	28
21	TestCase15	29
22	TestCase16	30
23	TestCase17	31
24	Class Index	32
24.1	Class List . . . . .	32

<b>25 File Index</b>	<b>33</b>
25.1 File List . . . . .	33
<b>26 Class Documentation</b>	<b>34</b>
26.1 HEAP< SOMETYPE > Class Template Reference . . . . .	34
26.1.1 Detailed Description . . . . .	36
26.1.2 Member Enumeration Documentation . . . . .	36
26.1.3 Constructor & Destructor Documentation . . . . .	37
26.1.4 Member Function Documentation . . . . .	38
26.1.5 Member Data Documentation . . . . .	43
26.2 Stock Class Reference . . . . .	44
26.2.1 Detailed Description . . . . .	45
26.2.2 Constructor & Destructor Documentation . . . . .	45
26.2.3 Member Data Documentation . . . . .	45
<b>27 File Documentation</b>	<b>47</b>
27.1 HEAP.H File Reference . . . . .	47
27.1.1 Enumeration Type Documentation . . . . .	49
27.1.2 Function Documentation . . . . .	49
27.2 loadFile.cpp File Reference . . . . .	50
27.2.1 Function Documentation . . . . .	51

---

27.3	project.cpp File Reference . . . . .	52
27.3.1	Function Documentation . . . . .	52
27.4	saveFile.cpp File Reference . . . . .	57
27.4.1	Function Documentation . . . . .	57
27.5	stock.dox File Reference . . . . .	59
27.6	Stock.h File Reference . . . . .	60
27.6.1	Function Documentation . . . . .	62
	<b>Index</b>	<b>64</b>

## 1 Specification

This is a stock program which is a system for trading in a dozen stocks. The system maintains the list of all active buy and sell orders for each stock that the customer has entered. In this system, the customer can log in, give his or her name, request quotes and place orders. An order holds the name of the customer, a buy or sell indicator, the stock symbol, the number of shares and a price limit. This system keeps all buy orders for each stock in a priority queue ranked by the bid price, and all sell orders in an inverted priority queue with the smallest "ask" price order on top.

## 2 Analysis

This program features around five different stocks with the user's option of auctioning for shares, bidding for buys, and asking for a price. Each stock will have a graph and show how much each stock is worth. When the program runs, it will present the user the market of five different stocks. Each stock has a unique value and can change as the user shares, buys, and asks. Each of the three functions have a button for the user to click to execute the function. Each time the user executes buy function, it will enter the value in a heap tree if it's not already in there. Also the user can buy that amount of shares for the price they are buying/bidding for. The shares function allows the user to enter how many shares he or she wants to buy or sell. Then the ask function lets the user enter the amount she or he wants for selling the amount of shares they are listing. The output of the function will display how many shares he or she have bought or sell for that stock. The user will then see how much they bought each share for and the commission they will be getting. Finally, it will print out the total worth of shares the user bought. Then it will display the latest bid, highest bid, and lowest bid. Also, the user is allowed to ask the market to display the latest bid, highest bid, and lowest bid or a specific stock. It will also display the bid size and ask size of how many shares they bought or sell. When the user adds the highest bid, it will move the heap tree around for the highest bid to be added onto the tree and replace the print out of highest bid when the user asks for details of the stock. The same applies for lowest bid. The heap tree will move around the values until the value has a spot and if the value was the lowest, it will display that value the next time the user asks for details of the stock i.e. highest,lowest, latest, shares amount, etc.

### 3 Design

When this program is launched, the user is given the option to choose from any company they would like do their stock exchange. They are given five different companies in a drop down bar to choose from. Once the user has chosen a company, they are given the option to either sell or buy. After the user has chosen one of those two options, the user is then, asked to give their name, shares amount, and price. After the user has entered the following, the program will then execute and insert the information into the heap tree and search through the heap tree. The heap tree will try to find a matching price that the user has input and it will execute the purchase. If there was no match, it will output "No seller/buyer" for the user. Also, if the user enter's the name market, it will purchase/sell the shares from the market. After every purchase and sell, it will display in the live graph and live table. The graph and table will update after every transaction.



## 4 HTML

```
<!DOCTYPE html>
<html>
<head>
<title>cs124 Stockz Lab</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script type="text/javascript">
    function newData() {
      var dataTable = document.getElementById("dataTable");
      var r = dataTable.insertRow(0);
      var c1 = dataTable.insertRow(0);

      var inputValue = document.getElementById("Stock");
      var strValue = inputValue.options[inputValue.selectedIndex].value;

      c1.innerHTML = "hello";
    }

    function CheckBrowser() {
      if('localStorage' in window && window ['localStorage'] !== null) {
        // we can use local storage
        console.log('local storage works');
        return true;
      }
      else {
        return false;
      }
    }

    function getStocks(){
      var stocks = new Array;
      var stocks_str= localStorage.getItem('stock');
      console.log(stocks_str);
      if (stocks_str !== null) {
```

```
        stocks = JSON.parse(stocks_str);
    }
    return stocks;
}

function getNames(){
    var names = new Array;
    var names_str= localStorage.getItem('name');
    console.log(names_str);
    if (names_str !== null) {
        names = JSON.parse(names_str);
    }
    return names;
}

function getShares(){
    var shares = new Array;
    var shares_str= localStorage.getItem('shares');
    console.log(shares_str);
    if (shares_str !== null) {
        shares = JSON.parse(shares_str);
    }
    return shares;
}

function getTypes(){
    var types = new Array;
    var types_str= localStorage.getItem('type');
    console.log(types_str);
    if (types_str !== null) {
        types = JSON.parse(types_str);
    }
    return types;
}

function getPrices(){
    var prices = new Array;
```

```
var prices_str= localStorage.getItem('price');
console.log(prices_str);
if (prices_str !== null) {
    prices = JSON.parse(prices_str);
}
return prices;
}

function add() {
    var stock = document.getElementById('Stock').value;

    var stocks = getStocks();
    stocks.push(stock);
    localStorage.setItem('stock', JSON.stringify(stocks));

    var name = document.getElementById('name').value;

    var names = getNames();
    names.push(name);
    localStorage.setItem('name', JSON.stringify(names));

    var share = document.getElementById('shares').value;

    var shares = getShares();
    var shareInt = parseInt(share);
    share = shareInt;
    shares.push(share);
    localStorage.setItem('shares', JSON.stringify(shares));

    var type = document.querySelector('input[name = "o"]:checked').value;

    var types = getTypes();
    types.push(type);
    localStorage.setItem('type', JSON.stringify(types));

    var price = document.getElementById('price').value;
    var priceInt = parseInt(price);
```

```
price = priceInt;
var prices = getPrices();
prices.push(price);
localStorage.setItem('price', JSON.stringify(prices));

addData();
show();
}

function show() {
    var stocks = getStocks();
    var names = getNames();
    var shares = getShares();
    var types = getTypes();
    var prices = getPrices();

    /*
    var html = "";
    for ( var i = 0; i <= stocks.length - 1; i++) {
        html += "<tr><td>" + stocks[i] + "</td><td>" + names[i] +
            "</td><td>" + shares[i] + "</td><td>" + types[i] +
            "</td><td>" + prices[i] + "</td></tr>";
    }
    document.getElementById('dataTable').innerHTML = html;
    */
    var table = document.getElementById("dataTable");

    var rowIndex = 1;
    for ( var i = 0; i <= stocks.length - 1; i++) {
        var r = table.insertRow(rowIndex);
        var c1 = r.insertCell(0);
        var c2 = r.insertCell(1);
        var c3 = r.insertCell(2);
        var c4 = r.insertCell(3);
        var c5 = r.insertCell(4);

        c1.innerHTML = stocks[i];
```

```
        c2.innerHTML = names[i];
        c3.innerHTML = shares[i];
        c4.innerHTML = types[i];
        c5.innerHTML = prices[i];
        rowIndex++;
    }
}

var dpB = []; // data points Buys
var dpS = []; // data points Sells

function pushData() {
    var types = getTypes();
    var prices = getPrices();
    var shares = getShares();
    for ( var i = dpB.length; i < prices.length; i++ ) {
        var sharesToInt = parseInt(shares[i]);
        var pricesToInt = parseInt(prices[i]);
        if (types[i] == "Buy") {
            dpB.push({
                x: sharesToInt,
                y: pricesToInt
            });
        }
        else if (types[i] == "Sell") {
            dpS.push({
                x: sharesToInt,
                y: pricesToInt
            });
        }
    }
}

function addData() {
    pushData();
    chart.options.data[0].dataPoints = dpB;
    chart.options.data[1].dataPoints = dpS;
}
```

```
        chart.render();
    }

    var chart;
    window.onload = function(){
        CheckBrowser();
        //localStorage.clear();
        chart = new CanvasJS.Chart("chartContainer",
            {
                title:{
                    text: "Amazon"
                },
                data: [
                    {
                        type: "bar",
                        dataPoints: dpB
                    },
                    {
                        type: "bar",
                        dataPoints: dpS
                    }
                ]
            });
        chart.render();
        show();
        addData();
    }

    //addData();
</script>
<script type="text/javascript" src="https://canvasjs.com/assets/script/canvasjs.min.js"></script>
<link rel="stylesheet" type="text/css" href="main.css">
</head>
<body>
<div class="home">
    <ul class="nav">
        <li><a href="">Home</a></li>
        <li><a href="">Profile</a></li>
```

```

        <li><a href="">Trades</a></li>
        <li><a href="">Logout</a></li>
    </ul>
    <h1>Stockz</h1>
    <hr>
    <div class="stockTable-1">
        <h1>EEEE-TRADE</h1>
        <div class="menu">
            <form name=ShoppingMenu action="/cgi-bin/stock">
                <div>
                    <label>Choose Stock:</label>
                    <select id="Stock" name="Stock">
                        <option value="Amazon">Amazon</option>
                        <option value="Nvidia">Nvidia</option>
                        <option value="BitCoin">BitCoin</option>
                        <option value="Ohlone">Ohlone</option>
                        <option value="Ganja4Lyfe">Ganja4Lyfe</option>
                    </select>
                </div>
                <input type="radio" id="Buy" name="o" value="Buy">Buy
                <input type="radio" id="Sell" name="o" value="Sell">Sell
            </div>
            <label>Name:</label><input type="text" id="name" name="n">
            <div>
                <label>Shares:</label><input type="text" id="shares" name="Z">
            </div>
            <div>
                <label>Price:</label><input type="text" id="price" name="p"><input type="submit" value="Go" onclick="add()
            </div>
        </div>
    </form>
    </div>
    <div class="stockTable">
        <div id="chartContainer" style="height: 360px; width: 100%;"></div>
    </div>
</div>
<div class="stockTable">

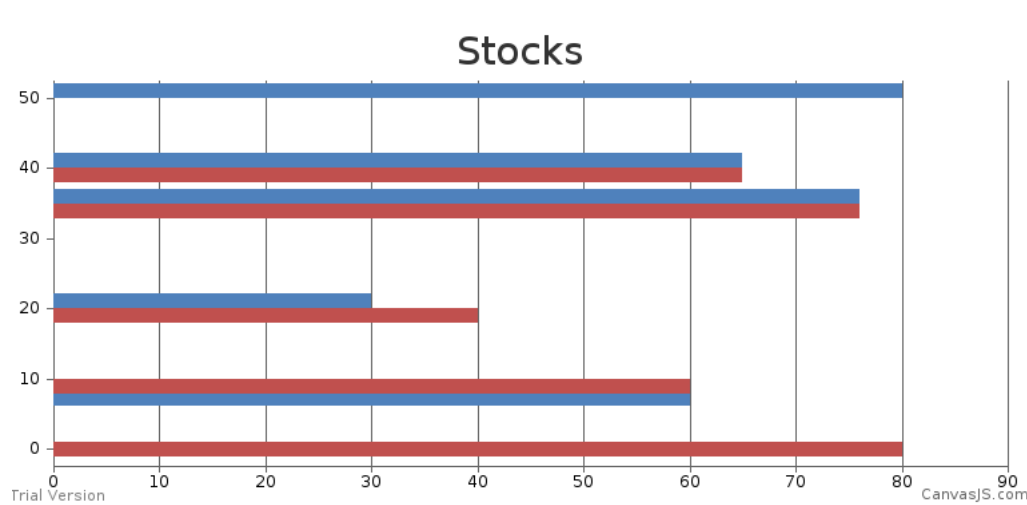
```

```
<h1>Stock Request</h1>
<table id="dataTable">
<tr>
  <th>Stock:</th>
  <th>Name:</th>
  <th>Shares:</th>
  <th>Type:</th>
  <th>Price:</th>
</tr>
</table>
</div>
</body>
</html>
```



## 5 Images/Features

In our HTML, we provided images and features such as a live table and live graph that uses JavaScript in order to update and show what is added to the Heap. Red bar for sell. Blue bar for buy.



## 6 Images/Features2

Live Table of Transaction Requests

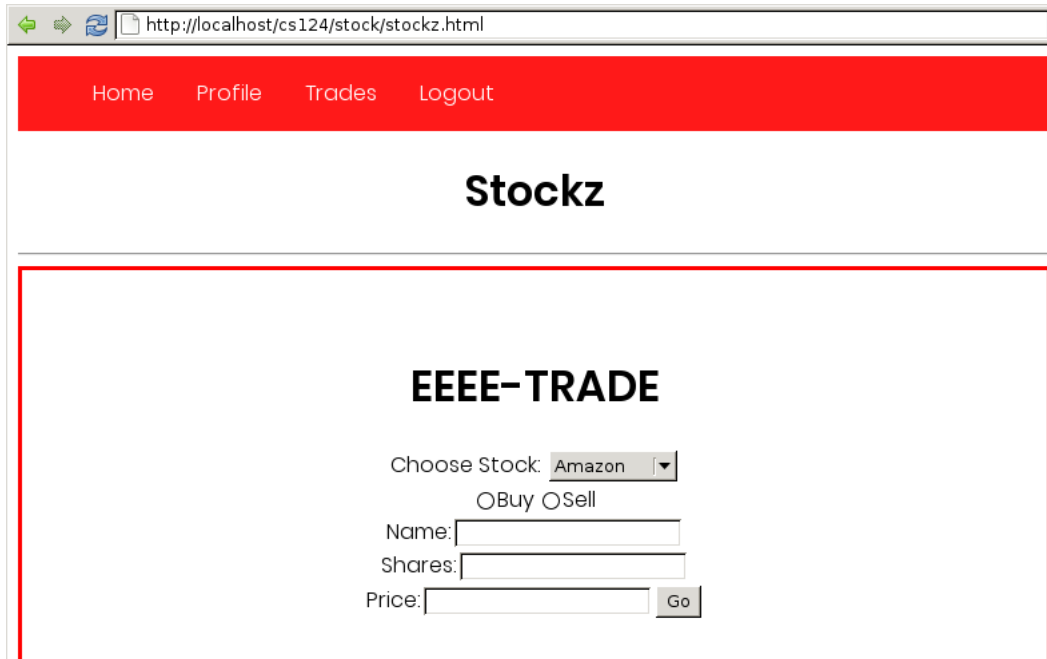


The screenshot shows a web browser window with the address bar displaying `http://localhost/cs124/stock/stockz.html`. The main content area features a table titled "Stock Request". The table has five columns: "Stock:", "Name:", "Shares:", "Type:", and "Price:". It contains five rows of transaction data.

Stock:	Name:	Shares:	Type:	Price:
Ganja4Lyfe	Kush	50	Sell	80
Ohlone	Druglord Peter	10	Sell	60
Ohlone	Fabio	6	Buy	60
Amazon	Ganja	20	Buy	30

## 7 TestCases1

Here is our home html page where one can input their name, how many shares, their prices, what stock they like and if they want to buy or sell.



The screenshot shows a web browser window with the address bar displaying `http://localhost/cs124/stock/stockz.html`. The page has a red navigation bar at the top with links: Home, Profile, Trades, and Logout. Below the navigation bar, the word "Stockz" is displayed in a large, bold, black font. A red rectangular border encloses the main content area, which contains the heading "EEEE-TRADE" in bold black text. Below the heading, there is a form with the following elements: a "Choose Stock:" label followed by a dropdown menu showing "Amazon"; two radio buttons labeled "Buy" and "Sell"; a "Name:" label followed by a text input field; a "Shares:" label followed by a text input field; a "Price:" label followed by a text input field; and a "Go" button.

Home Profile Trades Logout

# Stockz

## EEEE-TRADE

Choose Stock: Amazon ▼

☐ Buy ☐ Sell

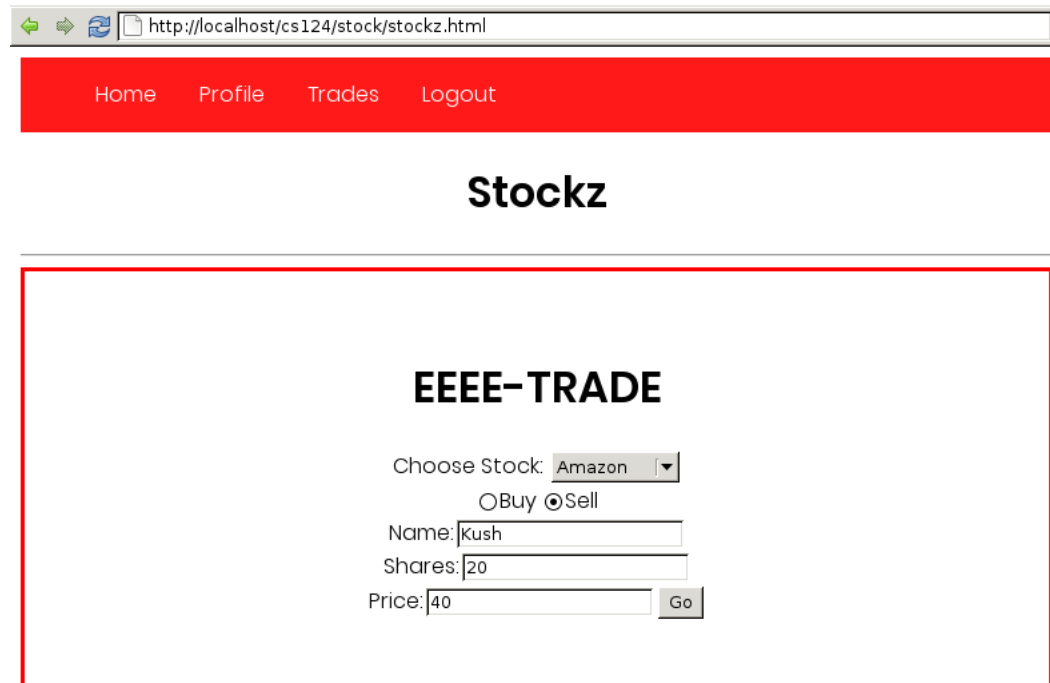
Name:

Shares:

Price:  Go

## 8 TestCases2

Let's say my buddy Kush wants to sell some of his Amazon stocks, but there are currently no buyers. We would expect that after clicking the go button. We would be redirected to an html page telling us if our transaction went through. Since no one is currently investing in Amazon, we should expect an output saying "no buyer found" and "a request pending".



http://localhost/cs124/stock/stockz.html

Home Profile Trades Logout

# Stockz

## EEEE-TRADE

Choose Stock: Amazon ▼

☐ Buy ☒ Sell

Name:

Shares:

Price:

## 9 TestCases3

As you can see, after submitting our request, we get our expected output telling us that our request has been added but sadly no one is currently buying.



## 10 TestCases4

Now don't worry Kush. Although, your request has yet to be fulfilled. My buddy Ganja thinks your prices are a bit too much for him and thinks he has a good offer for a lower price. Now when Ganja sends his request, since Kush is the only one selling Amazon stocks but for a higher price. We would expect Ganja to get a similar output but instead it would say that "there are no sellers found and your submission is pending".

---

### Stockz

---

#### EEEE-TRADE

Choose Stock:

☒ Buy ☐ Sell

Name:

Shares:

Price:

## 11 TestCases5

And as expected, Ganja's request has been added, but there is no one selling at that price.



## 12 TestCase6

Looking at the live graph, Kush sees Ganja's offer and he thinks its a fair deal. So Kush changes his offer to match Ganja's. Now when Kush hits the "go button", we should expect to see an output saying "there is a match found" and it should display the details of who you are exchanging with. In this case, we should see Ganja's info.

### Stockz

---

#### EEEE-TRADE

Choose Stock:

☐ Buy ☒ Sell

Name:

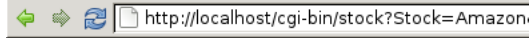
Shares:

Price:



## 13 TestCase7

Nice! The output is as we expected. We see who we are trading with and the shares they are receiving.



**Match found!**

**Stock: Amazon**

**Name: Ganja**

**Shares: 20**

**Type: Buy**

**Prices: 30**

## 14 TestCase8

Now let's test what happens when there is a partial buy. Here we have Druglord Peter selling 10 Ohlone shares for \$60. Now of course, we need a buyer.

### Stockz

---

#### EEEE-TRADE

Choose Stock:  ▼

☐ Buy ☒ Sell

Name:

Shares:

Price:

## 15 TestCase9

Here we have a buyer named Fabio and he only wants 6 Ohlone shares for \$60. When Fabio proceeds, a match would be found and transaction would occur, but ONLY for those 6 shares.

# Stockz

---

## EEEE-TRADE

Choose Stock:

☒ Buy ☐ Sell

Name:

Shares:

Price:

## 16 TestCase10

Awesome. As expected. We found a match and all information of the other party is displayed and so are the



**Match found!**

**Stock: Ohlone**

**Name: Druglord Peter**

**Shares:6**

**Type: Sell**

**Prices:60**

number shares bought.

## 17 TestCase11

Now let's test what happens where there is a partial sell. Here we have Kush again and now he's going to invest all his Amazon earnings on some Ganja4Lyfe stocks. He wants 50 shares for \$80. A great deal!

# EEEE-TRADE

Choose Stock:

☒ Buy ☐ Sell

Name:

Shares:

Price:

## 18 TestCase12

And now here we have his old partner Ganja. And he thinks the deal is terrible, but since Kush is such a good sport to sell him those Amazon stocks. He'll sell Kush 1 Ganja4Lyfe share. When Ganja submits this request, we should see that there is a match found and the other party's info and the number of stocks exchanged.

### EEEE-TRADE

Choose Stock:

☐ Buy ☒ Sell

Name:

Shares:

Price:

## 19 TestCase13

Beautiful. As expected, Ganja sells 1 share of Ganja4Lyfe to Kush for \$80.



**Match found!**

**Stock: Ganja4Lyfe**

**Name: Kush**

**Shares: 1**

**Type: Buy**

**Prices: 80**

## 20 TestCase14

Now let's try to do a market sell. We should see a successful transaction, since the market handles the request.

### EEEE-TRADE

Choose Stock:

☐ Buy ☒ Sell

Name:

Shares:

Price:



## 21 TestCase15

Nice!

---

---

# Match found!

**Stock: Nvidia**

**Name: Market**

**Shares:35**

**Type: Sell**

**Prices:76**

## 22 TestCase16

How about market buy? The same should happen.

### EEEE-TRADE

Choose Stock:  ▼

☒ Buy ☐ Sell

Name:

Shares:

Price:

## 23 TestCase17

Market buy works too!

---

# Match found!

**Stock: Nvidia**

**Name: Market**

**Shares: 40**

**Type: Buy**

**Prices: 65**

## 24 Class Index

### 24.1 Class List

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

#### **HEAP< SOMETYPE >**

This is a class **HEAP** which includes all function and storage needed to make the program and tree

34

#### **Stock**

This is a class **Stock** that creates minHeap and maxHeap which are able to be called throughout the program

44

## 25 File Index

### 25.1 File List

Here is a list of all files with brief descriptions:

<a href="#">HEAP.H</a>	<a href="#">47</a>
<a href="#">loadFile.cpp</a>	<a href="#">50</a>
<a href="#">project.cpp</a>	<a href="#">52</a>
<a href="#">saveFile.cpp</a>	<a href="#">57</a>
<a href="#">Stock.h</a>	<a href="#">60</a>

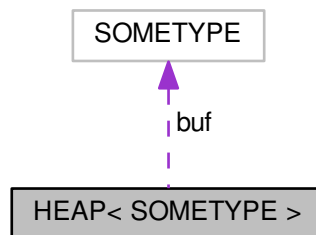
## 26 Class Documentation

### 26.1 HEAP< SOMETYPE > Class Template Reference

This is a class `HEAP` which includes all function and storage needed to make the program and tree.

```
#include <HEAP.H>
```

Collaboration diagram for `HEAP< SOMETYPE >`:



#### Public Types

- enum `heapType` { `MAX`, `MIN` }

### Public Member Functions

- `HEAP` (int a\_size, heapType)

*This is a constructor which build a heap tree of size a\_size.*

- `~HEAP` ()

*This is a destructor to destroy the heap trees.*

- bool `IsEmpty` ()

- bool `IsFull` ()

- `STATUS Insert` (SOMETYPE x)

*This is a insert function to add data into the heap trees.*

- `STATUS Remove` (SOMETYPE &x)

*This is a remove function to remove data from the heap trees.*

- void `printHeap` ()

*This is a function that prints out the heap tree.*

- void `addMaxHV` (int value)

- void `pushMaxHeapData` ()

*This is a push function that pushes information from maxHeap into a vector.*

- void `displayVector` ()

- void `tempMaxHeap` ()

*This is a function that adds the information from maxHeap into a temp vector.*

- void `tempMinHeap` ()

*This is a function that adds the information from minHeap into a temp vector.*

### Public Attributes

- SOMETYPE \* `buf`

- int `size`
- int `nNodes`
- `std::vector< int > minHV`
- `std::vector< int > maxHV`
- `std::vector< int > indexMinHV`
- `std::vector< int > indexMaxHV`
- int `tempMaxV`
- int `tempMinV`

### 26.1.1 Detailed Description

`template<class SOMETYPE>class HEAP< SOMETYPE >`

This is a class `HEAP` which includes all function and storage needed to make the program and tree.

### 26.1.2 Member Enumeration Documentation

26.1.2.1 `template<class SOMETYPE> enum HEAP::heapType`

Enumerator

***MAX***

***MIN***

26 {`MAX`, `MIN`};



### 26.1.3 Constructor & Destructor Documentation

26.1.3.1 `template<class SOMETYPE > HEAP< SOMETYPE >::HEAP ( int a_size, heapType ht )`

This is a constructor which build a heap tree of size *a\_size*.

**Parameters**

<i>a_size</i>	declares the size of the heap tree and which type of heap tree it will become
---------------	---

```

117 {
118     t = ht;
119     nNodes = 0;
120     buf = new SOMETYPE[a_size+1]; // +1 because buf[0] is not used
121     if (buf) size = a_size;
122     else size = 0;
123 }

```

**26.1.3.2 template<class SOMETYPE> HEAP< SOMETYPE >::~~HEAP ( )**

This is a destructor to destroy the heap trees.

```

134 {
135     delete [] buf;
136 }

```

**26.1.4 Member Function Documentation****26.1.4.1 template<class SOMETYPE> void HEAP< SOMETYPE >::addMaxHV ( int value )****26.1.4.2 template<class SOMETYPE> void HEAP< SOMETYPE >::displayVector ( )****26.1.4.3 template<class SOMETYPE> STATUS HEAP< SOMETYPE >::Insert ( SOMETYPE x )**

This is a insert function to add data into the heap trees.

## Parameters

<i>sometype</i>	which is basically an int of x
-----------------	--------------------------------

## Returns

returns OK

```
150 {
151     if (IsFull()) return FAILED;
152
153     nNodes++; // The last node of the heap is now vacant.
154
155     // Starting from the (vacant) last node, go from node i to
156     // its parent iParent and,
157
158     // if it is a max heap then as long as the parent is smaller
159     // than x, move the parent down:
160     // if it is a min heap then as long as the parent is
161     // larger than x, move the x up:
162
163     int i = nNodes;
164     int iParent;
165     while (i > 1) {
166         iParent = i/2;
167
168         if(t == MAX)
169         {
170             if (x <= buf[iParent]) break;
171         }
172         else // must be MIN
173         {
```

```
174         if( x >= buf[iParent]) break;
175     }
176     buf[i] = buf[iParent];
177     i = iParent;
178 }
179
180 // Insert x into the created vacancy:
181 buf[i] = x;
182
183 return OK;
184 }
```

**26.1.4.4** `template<class SOMETYPE> bool HEAP< SOMETYPE >::isEmpty ( ) [inline]`

```
45 {return (nNodes == 0);}
```

**26.1.4.5** `template<class SOMETYPE> bool HEAP< SOMETYPE >::isFull ( ) [inline]`

```
46 {return (nNodes == size);}
```

**26.1.4.6** `template<class SOMETYPE > void HEAP< SOMETYPE >::printHeap ( )`

This is a function that prints out the heap tree.

```
64 {
65     for(int i = 1; i < nNodes; i++)
66         std::cout << buf[i] << std::endl;
67 }
```

**26.1.4.7    template<class SOMETYPE > void HEAP< SOMETYPE >::pushMaxHeapData (    )**

This is a push function that pushes information from maxHeap into a vector.

```
103 {  
104     for(int i = 1; i < nNodes; i++)  
105         maxHV.push_back(buf[i]);  
106 }
```

**26.1.4.8    template<class SOMETYPE> STATUS HEAP< SOMETYPE >::Remove ( SOMETYPE & x )**

This is a remove function to remove data from the heap trees.

**Parameters**

<i>sometype</i>	which is basically an int reference of x
-----------------	--

**Returns**

returns OK

```
199 {  
200     if (IsEmpty()) return FAILED;  
201  
202     // Retrieve the top element:  
203  
204     x = buf[1];  
205  
206     // Starting from the vacant root, go from node iParent to its  
207     //   larger child i and, as long as that child  
208     //   is greater than the last element of the heap,
```

```

209     //  move that child up:
210
211     int iParent = 1;           // root
212     int i = 2;                // its left child
213     while (i <= nNodes) {
214         // Set i to the right child, i+1, if it
215         // exists and is larger:
216         if (i < nNodes && buf[i] < buf[i+1]) i++;
217         if (t == MIN)
218             if (i < nNodes && buf[i] > buf[i+1]) i++;
219         if (t == MAX)
220             if (i < nNodes && buf[i] < buf[i+1]) i++;
221         // Compare with the last node:
222         if (t == MAX && buf[i] <= buf[nNodes]) break;
223         if (t == MIN && buf[i] >= buf[nNodes]) break;
224         buf[iParent] = buf[i];    // Move the child up;
225         iParent = i;             // buf[iParent] is now vacant.
226         i *= 2;                 // i is set to its left child
227     }
228
229     // Move the last element into the created vacancy:
230     if (nNodes > 1) buf[iParent] = buf[nNodes];
231     nNodes--;
232
233     return OK;
234 }

```

#### 26.1.4.9 template<class SOMETYPE > void HEAP< SOMETYPE >::tempMaxHeap ( )

This is a function that adds the information from maxHeap into a temp vector.

```

83 {

```

```
84     for(int i = 1; i < nNodes; i++)
85         tempMaxV = buf[i];
86 }
```

#### 26.1.4.10 template<class SOMETYPE > void HEAP< SOMETYPE >::tempMinHeap ( )

This is a function that adds the information from minHeap into a temp vector.

```
73 {
74     for(int i = 1; i < nNodes; i++)
75         tempMinV = buf[i];
76 }
```

### 26.1.5 Member Data Documentation

26.1.5.1 template<class SOMETYPE> SOMETYPE\* HEAP< SOMETYPE >::buf

26.1.5.2 template<class SOMETYPE> std::vector<int> HEAP< SOMETYPE >::indexMaxHV

26.1.5.3 template<class SOMETYPE> std::vector<int> HEAP< SOMETYPE >::indexMinHV

26.1.5.4 template<class SOMETYPE> std::vector<int> HEAP< SOMETYPE >::maxHV

26.1.5.5 template<class SOMETYPE> std::vector<int> HEAP< SOMETYPE >::minHV

26.1.5.6 template<class SOMETYPE> int HEAP< SOMETYPE >::nNodes

26.1.5.7 template<class SOMETYPE> int HEAP< SOMETYPE >::size

26.1.5.8 template<class SOMETYPE> int HEAP< SOMETYPE >::tempMaxV

26.1.5.9 `template<class SOMETYPE> int HEAP< SOMETYPE >::tempMinV`

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

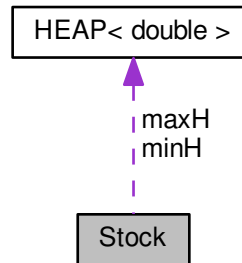
- [HEAP.H](#)

## 26.2 Stock Class Reference

This is a class [Stock](#) that creates minHeap and maxHeap which are able to be called throughout the program.

```
#include <Stock.h>
```

Collaboration diagram for Stock:





### Public Member Functions

- [Stock](#) ()

### Public Attributes

- [HEAP](#)< double > [minH](#)
- [HEAP](#)< double > [maxH](#)
- std::string [name](#)

#### 26.2.1 Detailed Description

This is a class [Stock](#) that creates minHeap and maxHeap which are able to be called throughout the program.

#### 26.2.2 Constructor & Destructor Documentation

##### 26.2.2.1 [Stock::Stock](#) ( ) [inline]

```
9           : minH(MAXHEAPSIZE, HEAP<double>::MIN) ,  
10           maxH(MAXHEAPSIZE, HEAP<double>::MAX) {}
```

#### 26.2.3 Member Data Documentation

##### 26.2.3.1 [HEAP](#)<double> [Stock::maxH](#)

##### 26.2.3.2 [HEAP](#)<double> [Stock::minH](#)

### 26.2.3.3 `std::string Stock::name`

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

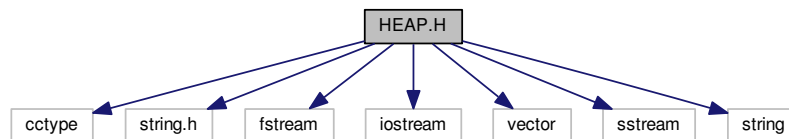
- [Stock.h](#)

## 27 File Documentation

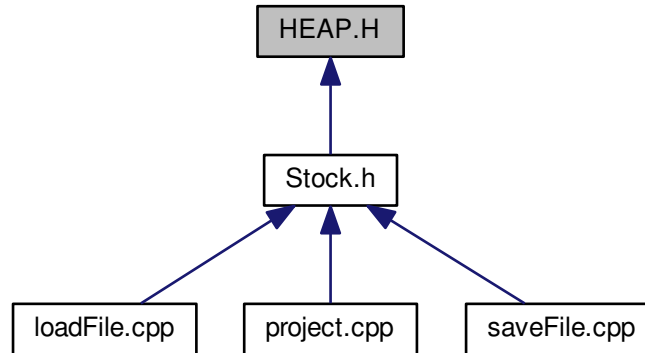
### 27.1 HEAP.H File Reference

```
#include <cctype>
#include <string.h>
#include <fstream>
#include <iostream>
#include <vector>
#include <sstream>
#include <string>
```

Include dependency graph for HEAP.H:



This graph shows which files directly or indirectly include this file:



## Classes

- class `HEAP< SOMETYPE >`

*This is a class `HEAP` which includes all function and storage needed to make the program and tree.*

## Enumerations

- enum **STATUS** { **OK**, **FAILED** }

*This is an enumeration that adds a status for OK and FAILED instead of true and false boolean.*

## Functions

- template<class SOMETYPE >  
void **HEAP**< SOMETYPE >::void **addMaxHV** (int value)

*This adds the value into the maxHeap vector.*

### 27.1.1 Enumeration Type Documentation

#### 27.1.1.1 enum **STATUS**

This is an enumeration that adds a status for OK and FAILED instead of true and false boolean.

#### Enumerator

**OK**

**FAILED**

```
18 {OK,FAILED};
```

### 27.1.2 Function Documentation

#### 27.1.2.1 template<class SOMETYPE > void **HEAP**<SOMETYPE>::void **addMaxHV** ( int *value* )

This adds the value into the maxHeap vector.

**Parameters**

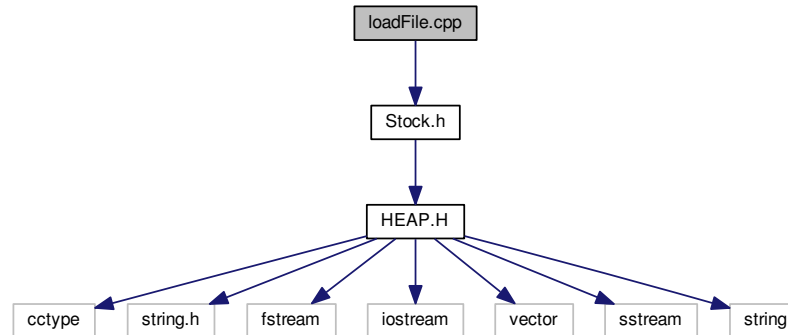
<i>a</i>	value of int to be added into maxHeap
----------	---------------------------------------

```
94 {  
95     maxHV.push_back (value);  
96 }
```

**27.2 loadFile.cpp File Reference**

```
#include "Stock.h"
```

Include dependency graph for loadFile.cpp:



## Functions

- void `loadFile` (std::string f, std::vector< double > &v)

*This is a function that loads a text file with information of the heap trees.*

### 27.2.1 Function Documentation

#### 27.2.1.1 void loadFile ( std::string f, std::vector< double > &v )

This is a function that loads a text file with information of the heap trees.

#### Parameters

<i>string</i>	to write in the file and a vector to reference the info that will be loaded from the file
---------------	---

#### Returns

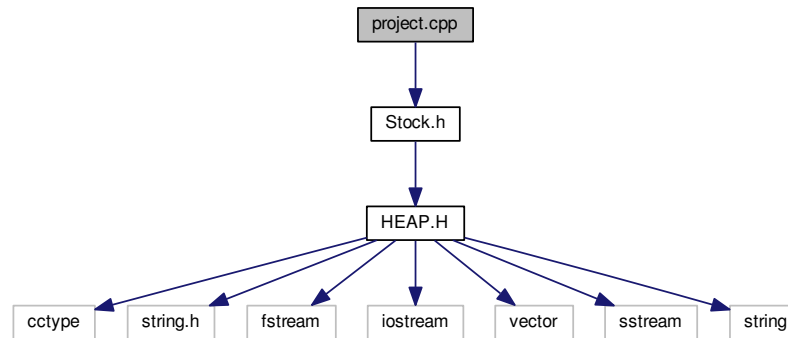
return information

```
3                                     {
4     std::ifstream ifs(f);
5     double x;
6     while(ifs >> x) {
7         //std::cout << "From file: " << x << std::endl;
8         v.push_back(x);
9     }
10    ifs.close();
11 }
```

### 27.3 project.cpp File Reference

```
#include "Stock.h"
```

Include dependency graph for project.cpp:



#### Functions

- int `main` ()

#### 27.3.1 Function Documentation



## 27.3.1.1 int main( )

```
3 {
4     Stock s;
5     std::vector<double> v;
6     std::vector<double> z;
7     std::string test = getenv("QUERY_STRING");
8     std::string delimiter = "&";
9     std::string stockName = test.substr(0, test.find(delimiter)); // extract StockName
10    test.erase(0, stockName.length()+1); // Remove Stock Info from string
11    stockName.erase(0,6);
12    std::string typeInput = test.substr(0, test.find(delimiter));
13    test.erase(0, typeInput.length()+1);
14    typeInput.erase(0,2);
15    std::string nameInput = test.substr(0, test.find(delimiter));
16    test.erase(0, nameInput.length()+1);
17    nameInput.erase(0,2);
18    std::string sharesInput = test.substr(0, test.find(delimiter));
19    test.erase(0, sharesInput.length());
20    sharesInput.erase(0,2);
21    std::string priceInput = test;
22    priceInput.erase(0,3);
23    /*
24    std::cout << priceInput << std::endl;
25    std::cout << "*****" << std::endl;
26    std::cout << stockName << std::endl;
27    std::cout << typeInput << std::endl;
28    std::cout << nameInput << std::endl;
29    std::cout << sharesInput << std::endl;
30    std::cout << priceInput << std::endl;
31    std::cout << "*****" << std::endl;
32    */
```

```

33     if (typeInput == "Buy") {
34         double priceDouble = std::stod(priceInput);
35         s.maxH.Insert(priceDouble);
36         s.maxH.Insert(priceDouble);
37         //s.maxH.printHeap();
38         double shareDouble = std::stod(sharesInput);
39         //std::cout << "priceDouble: " << priceDouble << std::endl;
40         //std::cout << "shareDouble: " << shareDouble << std::endl;
41         loadFile("/home/debian/cs124/stock/shares.json", z);
42         loadFile("/home/debian/cs124/stock/min.json", v);
43         //std::cout << v[0] << std::endl;
44         if (v.size() == 0 )
45             std::cout << "<html><h1>No seller found. Request pending...</h1></html>" << std::endl;
46         else if (v.size() != 0 ) {
47             int i;
48             for(i = 0; i < v.size(); i++) {
49                 if (priceDouble == v[i]) {
50                     std::cout << "<html><h1>Match found!</h1></html>" << std::endl;
51                     shareDouble = shareDouble;
52                     std::cout << "<html><h2>Stock: Nvidia</h2></html>" << std::endl;
53                     std::cout << "<html><h3>Name: Market</h3></html>" << std::endl;
54                     std::cout << "<html><h4>Shares:" << shareDouble << "</h4></html>" << std::endl;
55                     std::cout << "<html><h5>Type: Sell </h5></html>" << std::endl;
56                     std::cout << "<html><h6>Prices:" << priceDouble << "</h6></html>" << std::endl;
57                 }
58                 if(i == v.size()-1 && priceDouble != v[i])
59                     std::cout << "<html><h1>No seller found. Request pending...</h1></html>" << std::endl;
60             }
61         }
62         v.clear();
63         v.push_back(priceDouble);
64         z.clear();

```

```

65         z.push_back(shareDouble);
66         saveFile("/home/debian/cs124/stock/info.json", v);
67         saveFile("/home/debian/cs124/stock/shares.json", z);
68     }
69     else if (typeInput == "Sell") {
70         double priceDouble = std::stod(priceInput);
71         s.minH.Insert(priceDouble);
72         s.minH.Insert(priceDouble);
73         //s.minH.printHeap();
74         double shareDouble = std::stod(sharesInput);
75         //std::cout << "sharesInput: " << sharesInput << std::endl;
76         //std::cout << "priceDouble: " << priceDouble << std::endl;
77         loadFile("/home/debian/cs124/stock/shares.json", z);
78         loadFile("/home/debian/cs124/stock/info.json", v);
79         if (v.size() == 0 )
80             std::cout << "<html><h1>No seller found. Request pending...</h1></html>" << std::endl;
81         else if (v.size() != 0 ) {
82             int i;
83             for(i = 0; i < v.size(); i++) {
84                 if (priceDouble == v[i]) {
85                     shareDouble = shareDouble;
86                     if( shareDouble < 0)
87                         shareDouble = shareDouble * -1;
88                     std::cout << "<html><h1>Match found!</h1></html>" << std::endl;
89                     std::cout << "<html><h2>Stock: Nvidia<h2></html>" << std::endl;
90                     std::cout << "<html><h3>Name: Market</h3></html>" << std::endl;
91                     std::cout << "<html><h4>Shares:" << shareDouble << "</h4></html>" << std::endl;
92                     std::cout << "<html><h5>Type: Buy </h5></html>" << std::endl;
93                     std::cout << "<html><h6>Prices:" << priceDouble << "</h6></html>" << std::endl;
94                 }
95                 if(i == v.size()-1 && priceDouble != v[i])
96                     std::cout << "<html><h1>No buyer found. Request pending...</h1></html>" << std::endl;

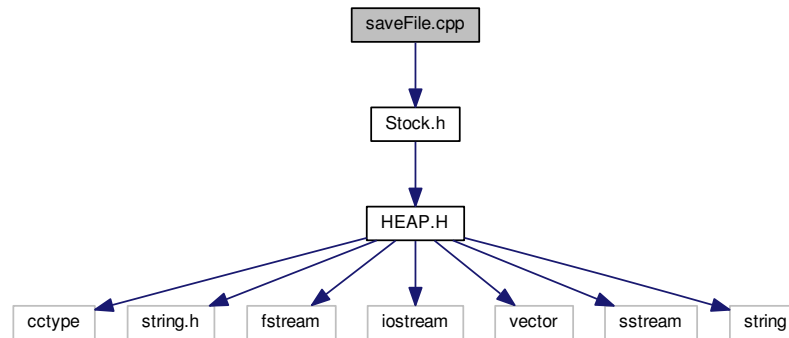
```

```
97         }
98     }
99     v.clear();
100     v.push_back(priceDouble);
101     z.clear();
102     z.push_back(shareDouble);
103     saveFile("/home/debian/cs124/stock/min.json", v);
104     saveFile("/home/debian/cs124/stock/shares.json", z);
105 }
106 }
```

## 27.4 saveFile.cpp File Reference

```
#include "Stock.h"
```

Include dependency graph for saveFile.cpp:



### Functions

- void [saveFile](#) (std::string f, std::vector< double > &v)

*This is a function that saves into a text file with the user's information.*

#### 27.4.1 Function Documentation

27.4.1.1 `void saveFile ( std::string f, std::vector< double > & v )`

This is a function that saves into a text file with the user's information.

## Parameters

<i>string</i>	to write in the file and a vector to reference the info that will be saved into the file
---------------	--

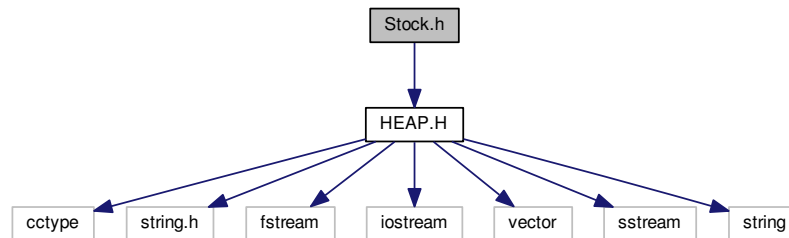
```
3                                     {
4     std::ofstream ofs;
5     ofs.open(f);
6     for(int i = 0; i < v.size(); i++) {
7         //std::cout << "Save File: " << v[i] << std::endl;
8         ofs << v[i];
9     }
10    ofs.close();
11 }
```

## 27.5 stock.dox File Reference

## 27.6 Stock.h File Reference

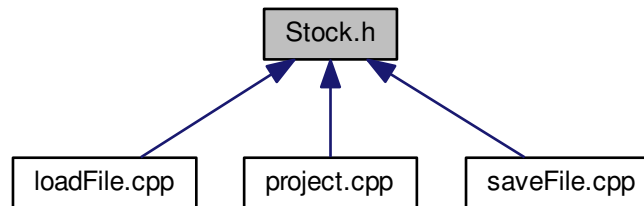
```
#include "HEAP.H"
```

Include dependency graph for Stock.h:





This graph shows which files directly or indirectly include this file:



### Classes

- class [Stock](#)

*This is a class [Stock](#) that creates minHeap and maxHeap which are able to be called throughout the program.*

### Functions

- void [loadFile](#) (std::string f, std::vector< double > &v)

*This is a function that loads a text file with information of the heap trees.*

- void [saveFile](#) (std::string f, std::vector< double > &v)

*This is a function that saves into a text file with the user's information.*

### 27.6.1 Function Documentation

#### 27.6.1.1 void loadFile ( std::string f, std::vector< double > & v )

This is a function that loads a text file with information of the heap trees.

##### Parameters

<i>string</i>	to write in the file and a vector to reference the info that will be loaded from the file
---------------	---

##### Returns

return information

```
3                                     {
4     std::ifstream ifs(f);
5     double x;
6     while(ifs >> x) {
7         //std::cout << "From file: " << x << std::endl;
8         v.push_back(x);
9     }
10    ifs.close();
11 }
```

#### 27.6.1.2 void saveFile ( std::string f, std::vector< double > & v )

This is a function that saves into a text file with the user's information.

## Parameters

<i>string</i>	to write in the file and a vector to reference the info that will be saved into the file
---------------	--

```
3                                     {
4     std::ofstream ofs;
5     ofs.open(f);
6     for(int i = 0; i < v.size(); i++) {
7         //std::cout << "Save File: " << v[i] << std::endl;
8         ofs << v[i];
9     }
10    ofs.close();
11 }
```

## Index

FAILED

HEAP.H, [49](#)

HEAP

MAX, [36](#)

MIN, [36](#)

HEAP.H

FAILED, [49](#)

OK, [49](#)

MAX

HEAP, [36](#)

MIN

HEAP, [36](#)

name

Stock, [45](#)

OK

HEAP.H, [49](#)

Stock, [44](#)

name, [45](#)

Stock, [45](#)