# Stock Market - Group Project

Generated by Doxygen 1.8.8

Fri Dec 8 2017 21:18:00

# Contents

# 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 lastest bid, highest bid, and lowest bid. Also, the user is allowed to ask the market to display the lastest 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;
```

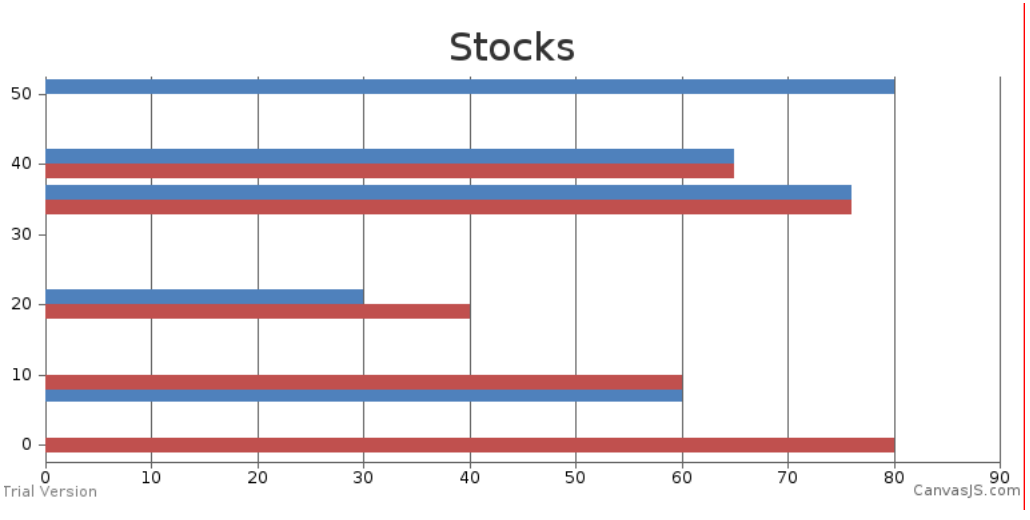
```
  chart.render();
}

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.



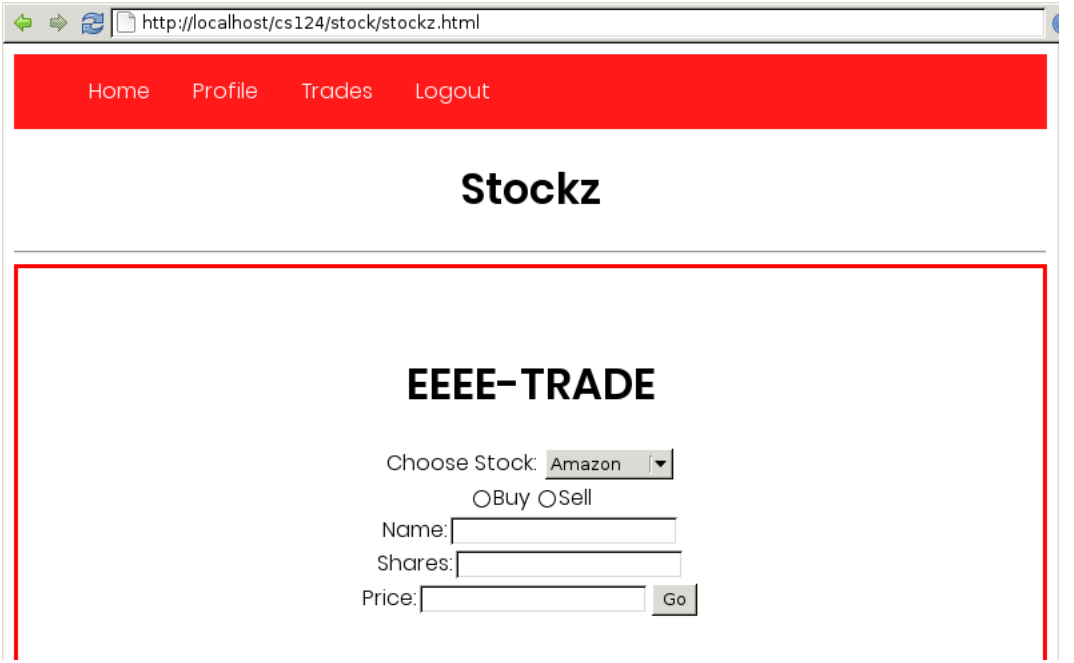
# 6 Images/Features2

Live Table of Transaction Requests


| 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 |

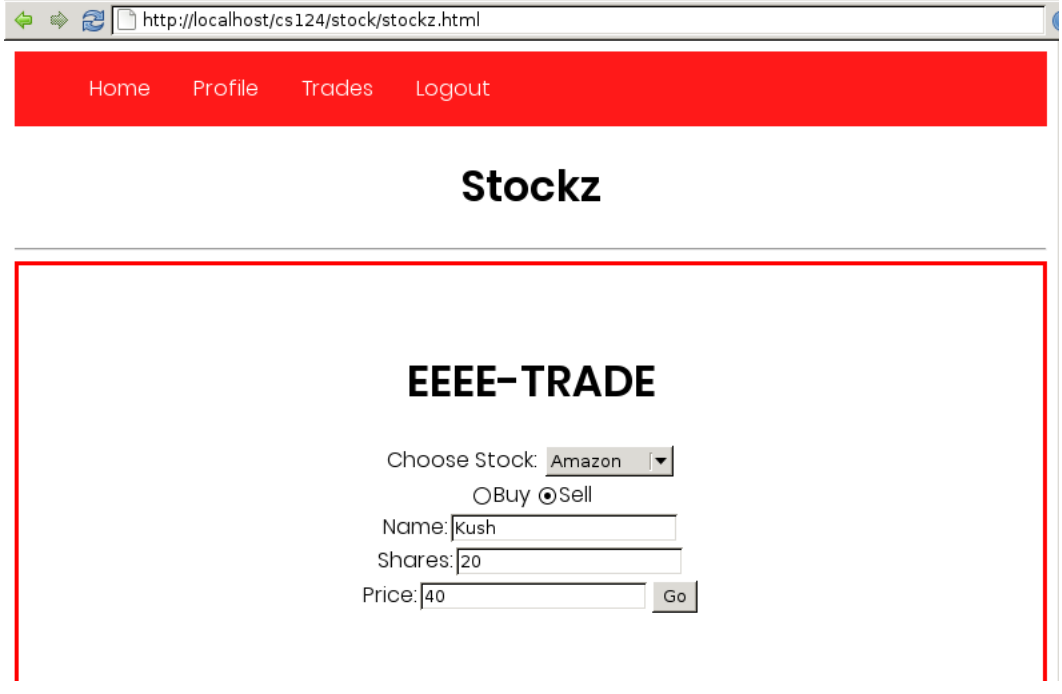
# 7 TestCases1

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.



# 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".

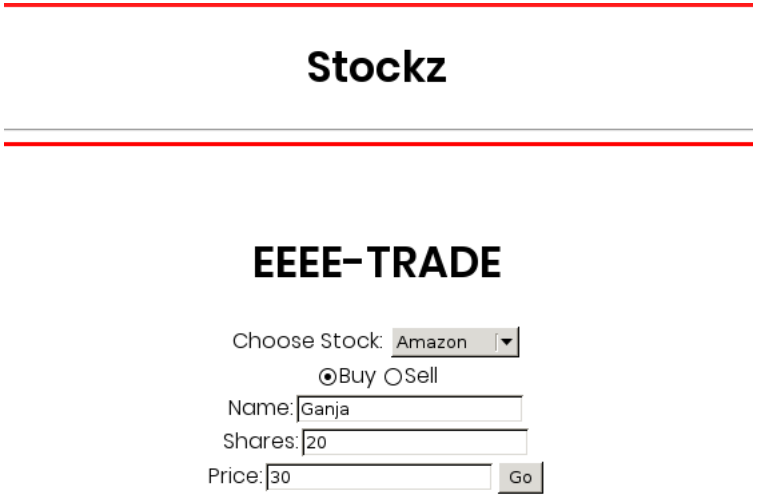


# 9 TestCases3

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 fufilled. 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".

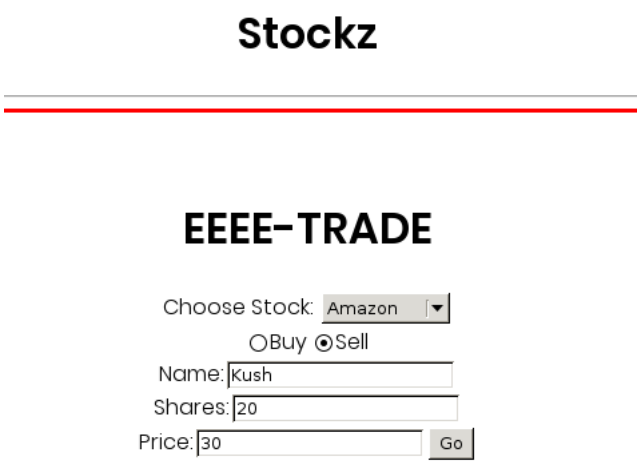


# 11 TestCases5

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

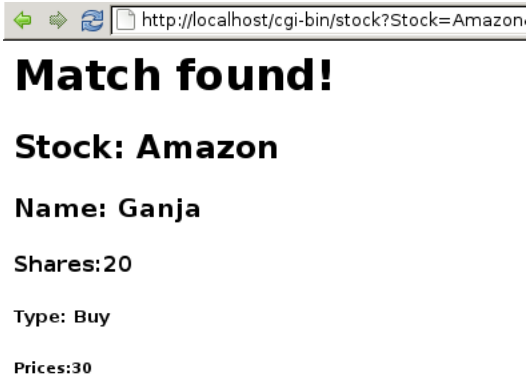
## 12 TestCase6

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.



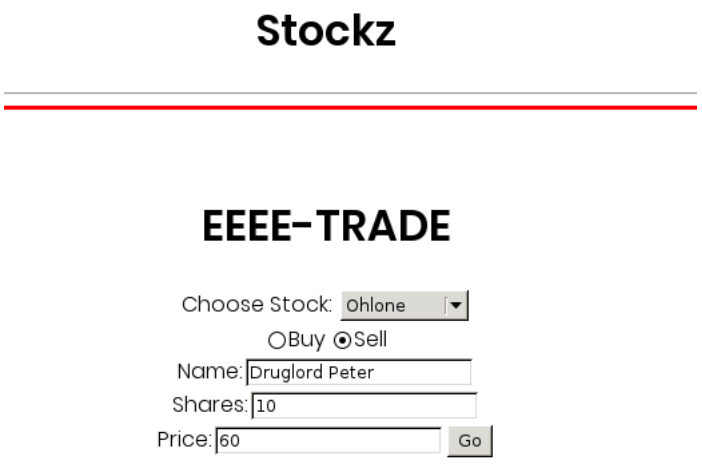
# 13 TestCase7

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



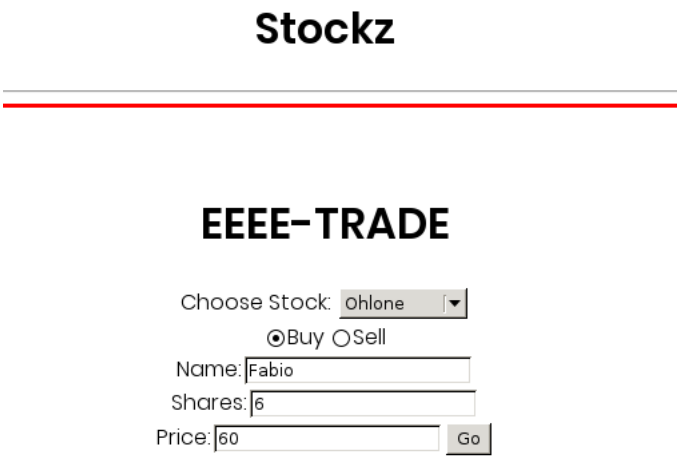
## 14 TestCase8

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.



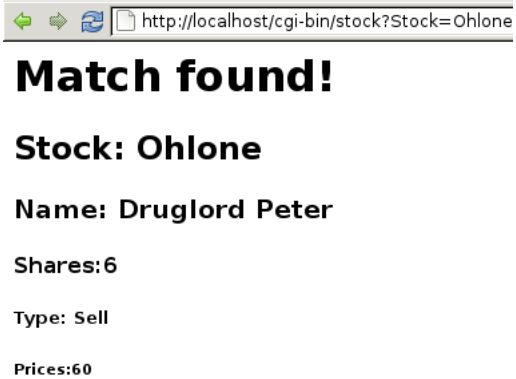
# 15 TestCase9

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.



## 16 TestCase10

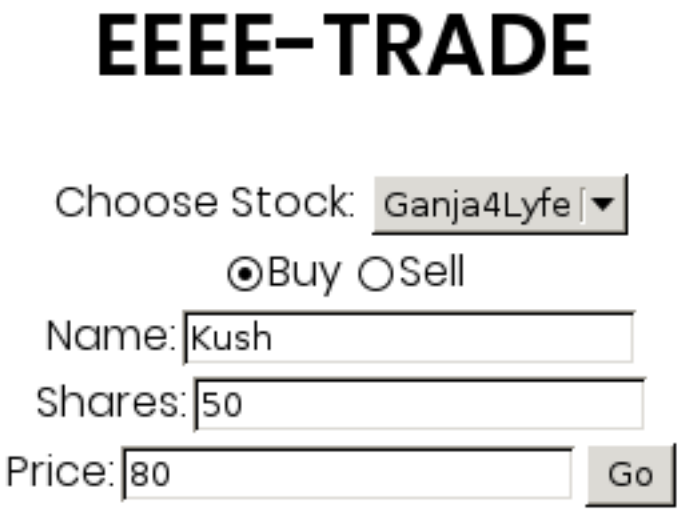
Awesome. As expected. We found a match and all information of the other party is displayed and so are the




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!



## 18 TestCase12

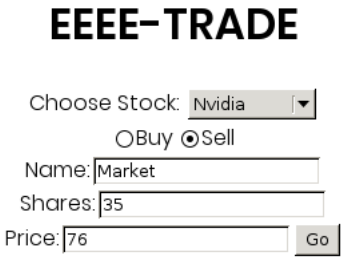
And now here we have his old partner Ganja. And he thinks the deal is terrible, but since Kush is such a good sport to sell him those Amazon stocks. He'll sell Kush 1 Ganja4Lyfe share. When Ganja submits this request, we should see that there is a match found and the other party's info and the number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

# 19 TestCase13

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

# 20 TestCase14

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



# 21 TestCase15

Nice!

---

# Match found!

## Stock: Nvidia

### Name: Market

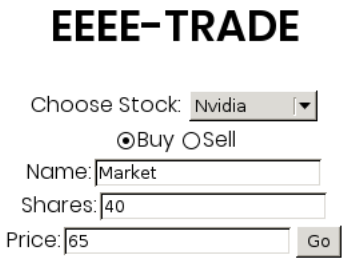
Shares:35

Type: Sell

Prices:76

## 22 TestCase16

How about market buy? The same should happen.



# 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:

# 25 File Index

## 25.1 File List

Here is a list of all files with brief descriptions:

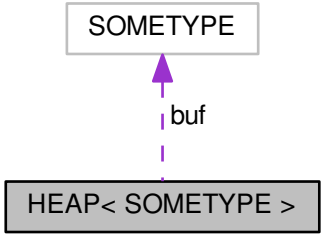
# 26 Class Documentation

## 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 funcction 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**

| | |
|---|---|
| *a_size* | declares the size of the heap tree |
| *ht* | heapType which type of heap tree it will become |

```
{
    t = ht;
    nNodes = 0;
    buf = new SOMETYPE[a_size+1]; // +1 because buf[0] is not used
    if (buf) size = a_size;
    else size = 0;
}
```

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

This is a destructor to destroy the heap trees.

```
{
    delete [] buf;
}
```

### 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**

| | |
|---|---|
| *x* | sometype which is basically an int of x |

**Returns**

returns OK

```
{
    if (IsFull()) return FAILED;

    nNodes++;  // The last node of the heap is now vacant.

    // Starting from the (vacant) last node, go from node i to
    //   its parent iParent and,

    // if it is a max heap then as long as the parent is smaller
    //  than x, move the parent down:
    // if it is a min heap then as long as the parent is
    //  larger than x, move the x up:

    int i = nNodes;
    int iParent;
    while (i > 1) {
        iParent = i/2;

        if(t == MAX)
        {
            if (x <= buf[iParent]) break;
        }
        else // must be MIN
        {
```

```
            if( x >= buf[iParent]) break;
        }
        buf[i] = buf[iParent];
        i = iParent;
    }

    // Insert x into the created vacancy:
    buf[i] = x;

    return OK;
}
```

#### 26.1.4.4 template<class SOMETYPE> bool HEAP< SOMETYPE >::IsEmpty ( ) [inline]

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

#### 26.1.4.5 template<class SOMETYPE> bool HEAP< SOMETYPE >::IsFull ( ) [inline]

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

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

This is a function that prints out the heap tree.

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

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

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

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

#### 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**

| | |
|---|---|
| *x* | sometype which is basically an int reference of x |

**Returns**

returns OK

```
{
    if (IsEmpty()) return FAILED;

    // Retrieve the top element:

    x = buf[1];

    // Starting from the vacant root, go from node iParent to its
    //   larger child i and, as long as that child
    //   is greater than the last element of the heap,
```

```
    //   move that child up:

    int iParent = 1;        // root
    int i = 2;              // its left child
    while (i <= nNodes) {
        // Set i to the right child, i+1, if it
        //   exists and is larger:
        if (i < nNodes && buf[i] < buf[i+1]) i++;
        if (t == MIN)
            if(i < nNodes && buf[i] > buf[i+1]) i++;
        if (t == MAX)
            if(i < nNodes && buf[i] < buf[i+1]) i++;
        // Compare with the last node:
        if (t == MAX && buf[i] <= buf[nNodes]) break;
        if (t == MIN && buf[i] >= buf[nNodes]) break;
        buf[iParent] = buf[i];    // Move the child up;
        iParent = i;              // buf[iParent] is now vacant.
        i *= 2;                   // i is set to its left child
    }

    // Move the last element into the created vacancy:
    if (nNodes > 1) buf[iParent] = buf[nNodes];
    nNodes--;

    return OK;
}
```

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

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

```
{
```

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

#### 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.

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

### 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:

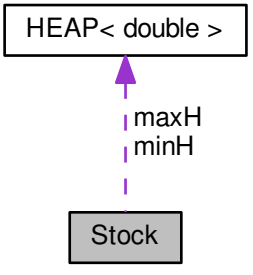
- HEAP.H

## 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]

```
: minH(MAXHEAPSIZE,HEAP<double>::MIN),
  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:

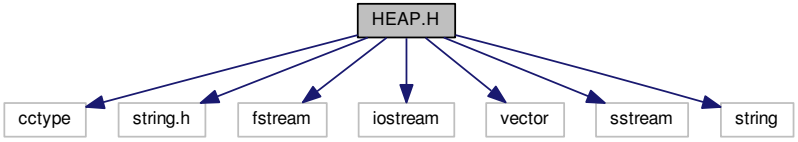
- Stock.h

# 27 File Documentation

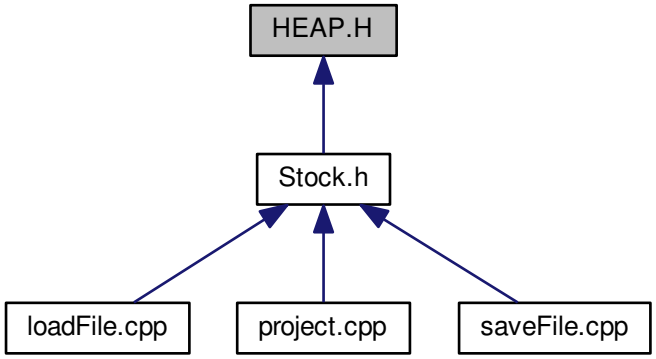
## 27.1 HEAP.H File Reference

```
#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**

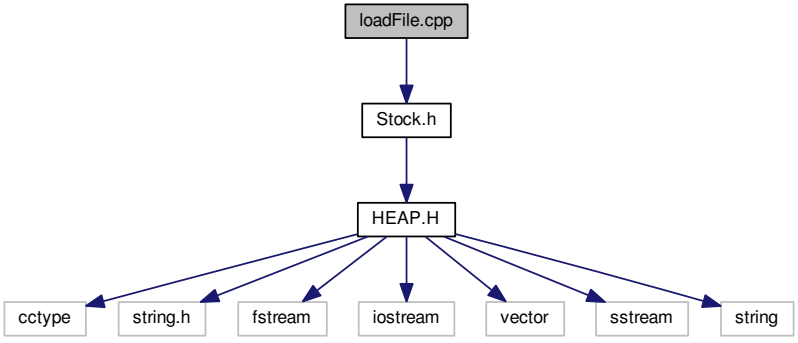
| | |
|---|---|
| *value* | a value of int to be added into maxHeap |

```
{
    maxHV.push_back(value);
}
```

## 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**

| | |
|---|---|
| *f* | string to write in the file |
| *v* | vector which references the info that will be loaded from the file |

**Returns**

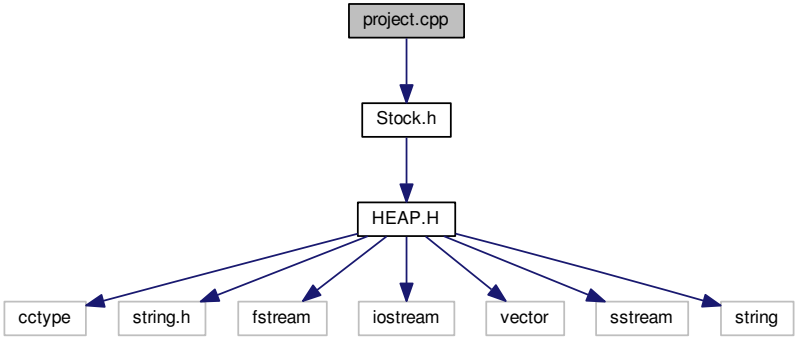
return information

```
{
    std::ifstream ifs(f);
    double x;
    while(ifs >> x) {
        //std::cout << "From file: " << x << std::endl;
        v.push_back(x);
    }
     ifs.close();
}
```

## 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 ( )

```
{
    Stock s;
    std::vector<double> v;
    std::vector<double> z;
    std::string test = getenv("QUERY_STRING");
    std::string delimiter = "&";
    std::string stockName = test.substr(0, test.find(delimiter)); // extract StockName
     test.erase(0, stockName.length()+1); // Remove Stock Info from string
     stockName.erase(0,6);
     std::string typeInput = test.substr(0, test.find(delimiter));
     test.erase(0, typeInput.length()+1);
     typeInput.erase(0,2);
     std::string nameInput = test.substr(0, test.find(delimiter));
     test.erase(0, nameInput.length()+1);
     nameInput.erase(0,2);
     std::string sharesInput = test.substr(0, test.find(delimiter));
     test.erase(0, sharesInput.length());
     sharesInput.erase(0,2);
     std::string priceInput = test;
     priceInput.erase(0,3);
     /*
     std::cout << priceInput << std::endl;
     std::cout << "****************" << std::endl;
     std::cout << stockName << std::endl;
     std::cout << typeInput << std::endl;
     std::cout << nameInput << std::endl;
     std::cout << sharesInput << std::endl;
     std::cout << priceInput << std::endl;
     std::cout << "**************" << std::endl;
     */
```

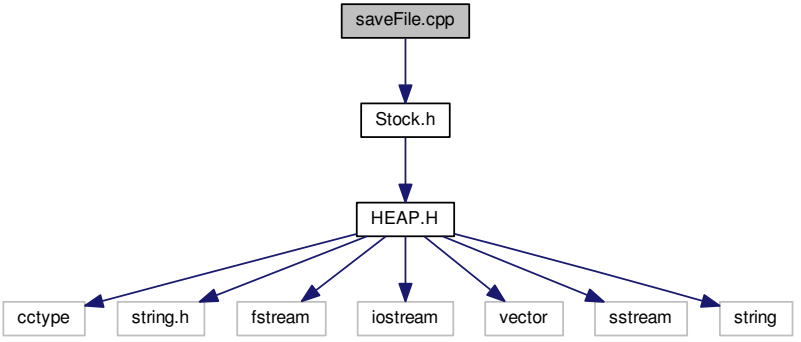
```
    if (typeInput == "Buy") {
        double priceDouble = std::stod(priceInput);
        s.maxH.Insert(priceDouble);
        s.maxH.Insert(priceDouble);
        //s.maxH.printHeap();
        double shareDouble = std::stod(sharesInput);
        //std::cout << "priceDouble: " << priceDouble << std::endl;
        //std::cout << "shareDouble: " <<  shareDouble << std::endl;
        loadFile("/home/debian/cs124/stock/shares.json", z);
        loadFile("/home/debian/cs124/stock/min.json", v);
        //std::cout << v[0] << std::endl;
        if (v.size() == 0 )
            std::cout << "<html><h1>No seller found. Request pending...</h1></html>" << std::en
        else if(v.size() != 0 ) {
            int i;
            for(i = 0; i < v.size(); i++) {
                if (priceDouble == v[i]) {
                    std::cout << "<html><h1>Match found!</h1></html>" << std::endl;
                    shareDouble = shareDouble;
                    std::cout << "<html><h2>Stock: Nvidia<h2></html>" << std::endl;
                    std::cout << "<html><h3>Name: Market</h3></html>" << std::endl;
                    std::cout << "<html><h4>Shares:" << shareDouble << "</h4></html>" << std::e
                    std::cout << "<html><h5>Type: Sell </h5></html>" << std::endl;
                    std::cout << "<html><h6>Prices:" << priceDouble << "</h6></html>" << std::e
                }
                if(i == v.size()-1 && priceDouble != v[i])
                    std::cout << "<html><h1>No seller found. Request pending...</h1></html" <<
                }
        }
        v.clear();
        v.push_back(priceDouble);
        z.clear();
```

```
        z.push_back(shareDouble);
        saveFile("/home/debian/cs124/stock/info.json", v);
        saveFile("/home/debian/cs124/stock/shares.json", z);
    }
    else if (typeInput == "Sell") {
        double priceDouble = std::stod(priceInput);
        s.minH.Insert(priceDouble);
        s.minH.Insert(priceDouble);
        //s.minH.printHeap();
        double shareDouble = std::stod(sharesInput);
        //std::cout << "sharesInput: " << sharesInput << std::endl;
        //std::cout << "priceDouble: " << priceDouble << std::endl;
        loadFile("/home/debian/cs124/stock/shares.json", z);
        loadFile("/home/debian/cs124/stock/info.json", v);
        if (v.size() == 0 )
            std::cout << "<html><h1>No seller found. Request pending...</h1></html>" << std::endl;
        else if(v.size() != 0 ) {
            int i;
            for(i = 0; i < v.size(); i++) {
                if (priceDouble == v[i]) {
                    shareDouble = shareDouble;
                    if( shareDouble < 0)
                        shareDouble = shareDouble * -1;
                    std::cout << "<html><h1>Match found!</h1></html>" << std::endl;
                    std::cout << "<html><h2>Stock: Nvidia<h2></html>" << std::endl;
                    std::cout << "<html><h3>Name: Market</h3></html>" << std::endl;
                    std::cout << "<html><h4>Shares:" << shareDouble << "</h4></html>" << std::endl;
                    std::cout << "<html><h5>Type: Buy </h5></html>" << std::endl;
                    std::cout << "<html><h6>Prices:" << priceDouble << "</h6></html>" << std::endl;
                }
                if(i == v.size()-1 && priceDouble != v[i])
                    std::cout << "<html><h1>No buyer found. Request pending...</h1></html>" << std::en
```

```
                }
        }
        v.clear();
         v.push_back(priceDouble);
         z.clear();
         z.push_back(shareDouble);
         saveFile("/home/debian/cs124/stock/min.json", v);
         saveFile("/home/debian/cs124/stock/shares.json", z);
         }
}
```

## 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**

| | |
|---|---|
| *f* | string to write in the file |
| *v* | vector which references the info that will be saved into the file |

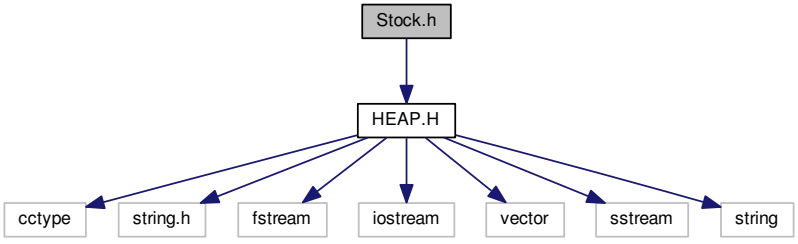
```
{
    std::ofstream ofs;
    ofs.open(f);
    for(int i = 0; i < v.size(); i++) {
        //std::cout << "Save File: " << v[i] << std::endl;
        ofs << v[i];
    }
     ofs.close();
}
```

## 27.5 stock.dox File Reference

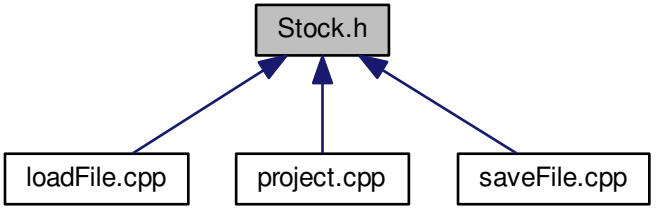
## 27.6 Stock.h File Reference

```
#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**

| | |
|---|---|
| *f* | string to write in the file |
| *v* | vector which references 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**

| | |
|---|---|
| *f* | string to write in the file |
| *v* | vector which references the info that will be saved into the file |

```
{
    std::ofstream ofs;
    ofs.open(f);
    for(int i = 0; i < v.size(); i++) {
        //std::cout << "Save File: " << v[i] << std::endl;
        ofs << v[i];
    }
     ofs.close();
}
```

# Index