**STOCK TRADE SYSTEM**

Stock trade simulator implementation in C++

horizontal line

# Placeholder image

# Introduction

Elon Musk, Bill Gates and the Glazer brothers are considered as owners of Tesla Inc., Microsoft Corporation and Manchester United Football Club, but in reality, things are different. Musk, Gates and Glazers are stockholders and own a significant portion of the shares of these companies. But who is a stockholder? What is a stock? What are shares? How and where can you buy or sell them? Let's delve into this topic together.

## 

## Stock Exchanges

# Placeholder image

This is the **New York Stock Exchange** (NYSE, nicknamed "The Big Board"). NYSE is the largest stock exchange in the world. In stock exchanges stockbrokers and traders can buy and sell securities, such as shares of stock. A **stockbroker** is an individual or company that buys and sells stocks. **Stocks** consist of all the shares by which ownership of a corporation or company is divided. A **share** (sometimes referred to as stock or equity) is a unit of equity ownership in the stock of a corporation. A **stock trader,** also called a **stock investor**, is a person or company involved in trading equity securities and attempting to profit from the purchase and sale of those securities.

Following protests by fans, the Glazer brothers have decided to sell their shares in Manchester United and never come back again.They place an **order** to sell their ownership stake. Anyone else or company wants to buy shares of Man. Utd. and if their orders **match, exchange** occurs.

## 

## Orders

In the context of trading and finance, an **order** is a set of instructions provided by a trader to buy or sell a financial security (such as stocks, bonds, or commodities) through a broker or a trading platform. The order specifies important details, including:

* **Type of Order**: This indicates whether order is a limit, market, etc.
* **Quantity**: The number of shares or units of the security to buy or sell.
* **Price**: The specific price at which the investor wants the trade to be executed.
* **Duration**: How long the order remains active. It can be for the current trading session or until canceled.

Orders are essential for executing trades in financial markets and play a key role in determining when and at what price a trade will occur. Different types of orders, such as market orders, limit orders, and stop orders, allow investors to manage their trading strategies and risk.

### Some types of orders

* **Market Order**: A market order is an order to buy or sell a security immediately at the current market price. This type of order guarantees that the order will be executed but does not guarantee the price at which it will be executed.
* **Limit Order**: A limit order is an order to buy or sell a security at a specific price or better. It will only be executed at the specified price or a better one. If the limit price is not met, the order may not be executed.
* **Stop Order** (Stop-Loss Order): A stop order is an order to buy or sell a security once the market price reaches a specified "stop" price. It is often used as a risk management tool to limit potential losses.
* **Stop-Limit Orde**r: A stop-limit order combines aspects of a stop order and a limit order. It involves setting two prices: a "stop" price and a "limit" price. When the market price reaches the stop price, the order becomes a limit order to be executed at the limit price or better.

## Order Book

An order book is a real-time, continually updated list of buy and sell orders for a particular financial instrument, such as a stock, cryptocurrency, or commodity, on an exchange or trading platform. It provides a detailed view of the supply and demand for that instrument at various prices.

# **BINANCE TRADING**

# This is an Order Book. Red ones are Sell Orders and Green ones are Buy Orders. Sell orders are listed in decreasing order, buy orders in increasing order. You can choose your preferred order type and view the appropriate order book.

A **stock depository**, also known as a securities depository or central securities depository (CSD), is a financial institution or organization responsible for holding, maintaining, and facilitating the transfer of securities such as stocks in electronic or paper form. Stock depositories play a crucial role in modern financial markets by providing a centralized system for the safekeeping and efficient transfer of securities. They help reduce the risks associated with physical stock certificates and enable electronic trading and settlement of securities.

# Stock trade simulator implementation in C++

Let’s observe main parts of the code. Code consists of .cpp and .hpp files. Class definitions are provided in a header file, and the implementations are written in a separate source file with .cpp extension.

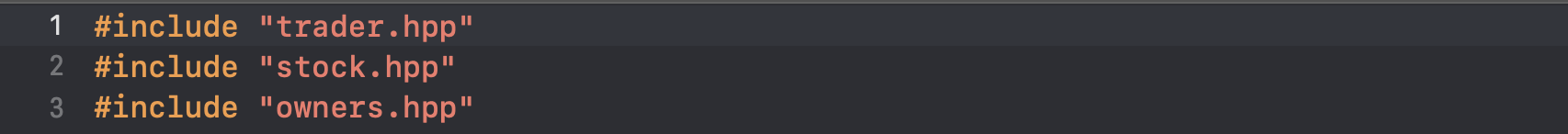
## Header Files - class definitions

* orders.hpp
* owners.hpp
* trader.hpp
* trade.hpp
* stock.hpp

## Source Files - implementations

* orders.cpp
* owners.cpp
* trader.cpp
* stock.cpp
* exchange.cpp

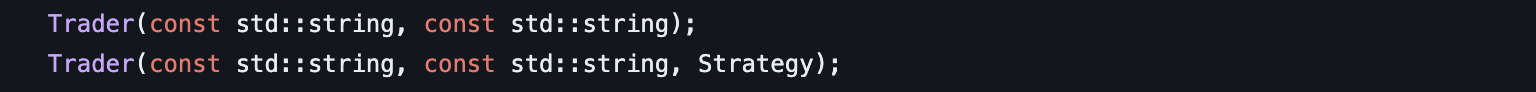
You need to include **includes.h** file into main.cpp and compile cpp files with main file



## Trader

### Bro’s name is Elon. He has a unique id and also an email "maskuerade@spacemail.com". He wants to buy some Twitter shares and needs to place an order. He should choose a strategy for creating order. Bro is a trader and must have funds (capital) to trade. Trade after trade he fills his portfolio. He can place either sell or buy orders, pay for shares or get money from selling. He can also top-up money. Ilon has lists of his current buy and sell orders.

### Creation

Trader has 2 constructors with parameters for setting name, email and strategy. Code also provides copy ctor and copy assignment and equality operators. Identificator of trader sets automatically without outer interaction. 

**Behavior**

* Two main methods of Trader are **place\_sell\_order** and **place\_buy\_order**. First one creates and pushes order to current\_sell\_orders, second one to current\_buy\_orders.
* **delete\_sell\_order** and **delete\_buy\_order** methods are called after execution or failure.
* **add\_to\_portfolio** methods add new stock to the trader's portfolio and **reduce\_quantity** method reduces the quantity of stock the trader owns or deletes it.
* **top\_up** and **pay** methods represent payment operations.
* Various **getters** and **setters**

**Stock**

Our bro Elon owns stock in Tesla, Inc. and now wants stock in Twitter. Each stock has its own unique symbol, name, trade history and order book.

* **AAPL** - Apple Inc.
* **AMZN** - Amazon.com, Inc.
* **TSLA** - Tesla inc.
* **NVDA** - NVIDIA Corp.

### Creation

### Stock has parameterized constructor, symbol and name as argument, copy constructor, copy assignment operator.

**Behavior**

### Code contains overloaded equality and comparison operators and some methods.

### getters - get\_symbol, get name

### rename - set new name to name :)

### add\_trade - add new trade to trade history

### add\_order - this function sends given order to the order book.

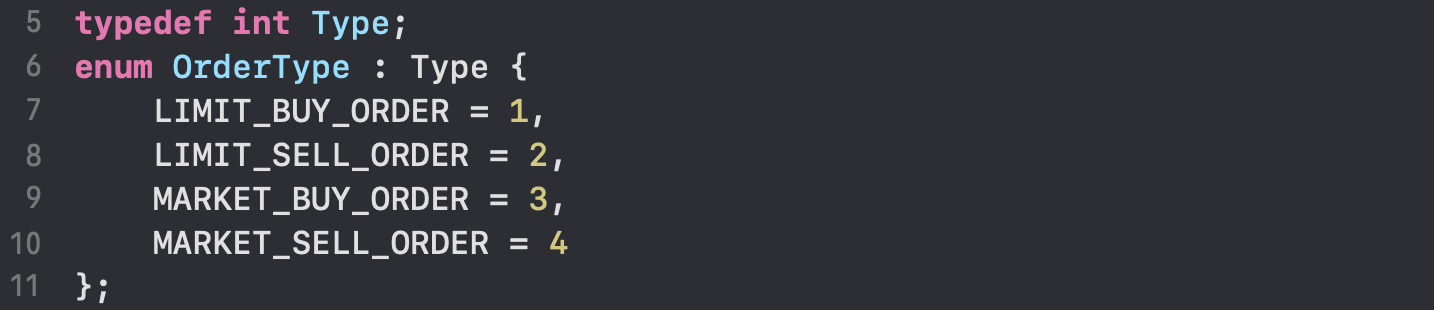
**Order**

Orderhas a unique **id**, **trader** who placed the order, **stock** the trader wants to buy/sell, **quantity** and **price**. 

Order is a single mother with four children.

* **LimitBuyOrder**
* **LimitSellOrder**
* **MarketSellOrder**
* **MarketBuyOrder**

Subclasses also have Type variable **ordertype**.



**Constructors, operators, getters**

Parameterized constructor that takes reference to Trader object, reference to Stock, quantity, and price, is here to construct traders order. It is called in order placing methods. Copy constructor, copy assignment and comparison operators are also implemented.

Order has many **getters** to get id, trader, stock, quantity, price, and virtual getter for type.

**Owners**

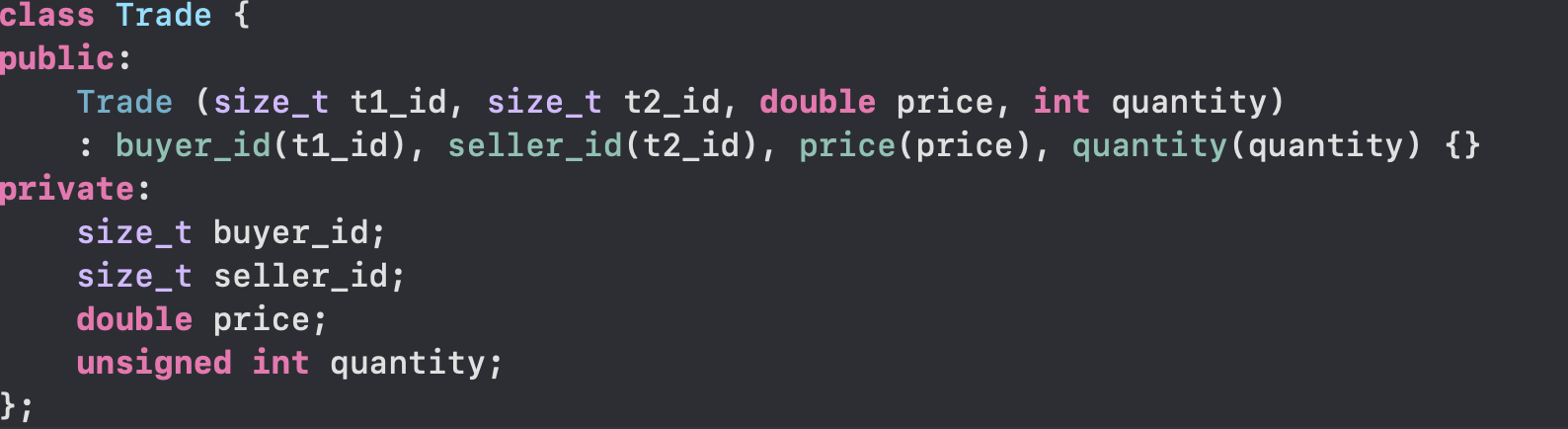
Owners is a representation of a stock depository in my simulator of the stock trade system. It is implemented as a **singleton class**. It can be considered as a database. Data structure, used to store ownership, is a **hash table**.

****

Trader id is mapping into another map, where keys are Stock objects, and they are mapping into stock quantity the trader owns. StockHash is a hash function for Stock object hashing. Owners class has 2 methods **add\_record** and **get\_stock\_quantity**. First one updates the database with adding new owners, second one is simple getter to get stock quantity.

**Trade**

Trade is a very simple class with 4 fields and constructor with parameters only. It is used only in Stock’s adding trade method to construct trade and add to trade history.



**Order Book**

OrderBook is a nested class in Stock. It contains two main sets for buy and sell orders. Sets are constructed using comparators, for ascending and descending order. Method **matching** is the most important part of the whole code. It takes an Order and checks its type (type of order is enum, called OrderType). Code supports only limit and market orders. I think SOLID principles are violated here, as if programmers decide to modify code and add new order types, they should add if cases in this function. Maybe implementation should check type, create such order and call polymorphic function with concrete order type. In the case of a market buy order, the system checks whether there are any sell orders for the stock. If the sell orders have a quantity greater than or equal to the quantity the buyer wants to purchase, and if the trader has sufficient funds, an **exchange** can take place. If there isn't enough money available or there are no matching sell orders, the matching process fails. The same logic applies to market sell orders. For limit orders I want to write endless programs that can run until the servers are not turned off. :) Orders should be added continuously, in a multithreaded system.

**exchange function**

The exchange function is the culmination of the system, its last part.

* Stock goes from seller to buyer.
* Seller stock quantity is reduced.
* Buyer stock quantity updates.
* Money goes from buyer to seller.
* Buyer’s funds decrease.
* Seller’s funds increase.
* Depository ( Owners database ) updates.
* Order is removed from trader’s current order lists
* And after exchange this executed trade should be added to stock trade history.

This was my vision on this topic with uncomplete code. Some parts differ from ones in real life, but I also don't work for Binance.

<https://github.com/arthur535/TradeStockSystem/tree/main> Arthur Mirakyan, 2023