*Cryptocurrency Market*

For this project, you will be implementing a rudimentary market for cryptocurrencies. A class, name "Exchange", has public member functions that allow for the deposit and withdrawl of various assets (e.g. Bitcoin, Etherum and US Dollars). Orders to buy and sell assets at a specified price can also be made and the exchange will perform trades as possible to fulfill orders. Lastly, the exchange has a member function dedicated to printing out the current status of the exchange (i.e. what is in each user’s portfolio, what orders have been opened and filled, what trades have taken place, and what the current prices are for the assets being traded.

The starter code has three header files and 4 implementation files. Some of these files have content in them already, but you are welcome to add, remove or change any part of the code given. Each test case will compile the implementation files and include the exchange.hpp and/or utility.hpp if it makes use of a class declared within. The useraccount files are not directly tested in the test cases, but they can be optionally used to store and manipulate information associated with a user’s account (as helper class(es) for the exchange). The provided main file is never used by any test cases, but it provides an example use of the exchange for your own testing purposes.

**Exchange Member Functions:**

* MakeDeposit: this function should take a username (e.g. “Nahum”), an asset (e.g. “BTC”), and an amount (e.g. 100). The function does not return anything, but should store the information that the deposit occurred to a user’s portfolio. A portfolio is a record of the assets a user possesses on the exchange that are not involved in any open orders. In the example, Nahum is depositing 100 BTC (Bitcoins) into the exchange.
* PrintUserPortfolios: this function takes a reference to a std::ostream and should print out each user’s portfolio (ordered alphabetically by user’s name, and then by asset). Remember that assets that are involved in an open order should not be part of a user’s portfolio. Please see the test cases for exact formatting. Most test cases involving printing will print to std::cout (for easier output reading), as well as, to a std::ostringstream (for test case validation).
* MakeWithdrawl: this function has the same parameters as MakeDeposit, but instead of depositing funds in the exchange, the user is removing previously deposited funds. The function returns true if and only if the withdrawal was successful. A withdrawal can only be unsuccessful if there are insufficient assets in a user’s portfolio.
* AddOrder: This is the most involved member function here. All of the code needed to implement this functionality should not be in this single function but allocated to many helper functions and classes. This function takes an Order (declared in utility.hpp) and returns true if the order was successfully placed. The only reason an order could not be placed is if there were insufficient assets to cover the trade. For instance, if you want to sell 10 bitcoins you must have at least 10 bitcoins in your portfolio. Additionally, if you want to buy 10 bitcoins for 100 USD (US Dollars), you need to have at least 1000 dollars in your portfolio to pay for such an order. All orders (buy or sell) are priced in USD. When an order is made, the funds needed to cover the order are removed from the portfolio (to prevent double spending). The order is then considered open and may participate market taker trades and then market maker trades (details on trading provided below).
* PrintUsersOrders: For each user (ordered alphabetically), this function prints the open and filled trades (ordered chronologically as each trade was opened/filled) to the referenced std::ostream. See test cases for exact formatting.
* PrintTradeHistory: Prints a record of each trade (chronologically) that occurred to the provided reference to std::ostream. See test cases for exact formatting.
* PrintBidAskSpread: For each asset (ordered alphabetically), prints the bid ask spread dictated by the open orders to the reference to std::ostream. See test cases for exact formatting.

**How trading works:**

First, please read <https://academy.binance.com/en/articles/what-are-makers-and-takers>

And read <https://academy.binance.com/en/glossary/bid-ask-spread>

When an order is placed (for example, I want to buy 10 BTC for 100 USD), this is a taker order. This means that the exchange will try to immediately fulfill my order by using existing open orders for the asset (BTC) on the other side of the trade (Sells). If the lowest priced sell is (sell 5 BTC for 200 USD), then no trade can take place because I only want to spend 100 USD per BTC, and no one is willing to sell for that low a price. If no trades can take place, my order becomes a market maker, meaning it is recorded by the exchange and waits until someone makes a sell order for 100 USD or less.

However, lets imagine that Dolson had a preexisting sell market maker order to sell 5 BTC for 50 USD. In this circumstance, my buy taker order would be partially fulfilled by a trade. And the price is determined by the taker order. So 5 BTC from Dolson would be transfered to me and I would give Dolson 500 USD (5 \* 100 USD). If no other sell market order existed with a price less than or equal to $100, then the rest of my order (buy 5 BTC for 100 USD) would be added as an open market maker.

There many be 0, one, or many taker trades that occur when an order is placed. And a taker order can be a buy or a sell order. If the taker is a buy order, the sell with the lowest price (and under the taker’s buy price) will be the order that participates in a trade. If the taker is a sell, then the highest buy (and must be higher than the taker’s sell price) is the best order and will participate in the trade. After a trade takes place, the relevant assets are deposited in the appropriate user’s portfolio.

Additionally, by examining the open market maker orders you can determine the bid-ask spread for an asset. This is effectively the highest price people are willing to buy the asset for and the lowest price people are willing to sell the asset for. These two values must always be different (else a mutually agreeable trade should have already occurred.