Project: Order Matching Engine

Manual

Team Members

- Swetha Shanmugam (UNI: ss6357)
- Zhejian Jin (UNI:zj2324)
- Fenglei Gu (UNI: fg2546)

Reader

The Reader class reads bytes from the stream and parses messages.

Members

```
std::string fileName;    // the data source file name
std::ifstream file;      // ifstream of the data
unsigned count = 0;
char message[64];        // buffer for the message
bool validFile = false;
time_t start;            // used for recording the time
```

Functions

```
Reader(std::string fileName); // constructor
virtual ~Reader();            // destructor
Message createMessage();      // Creates a Message object from the file
stream and returns a message to the BookConstructor class.
virtual void readBytesIntoMessage(const long &); // read n bytes from the
file
virtual void skipBytes(const long &); // skip n bytes from the stream
```

BookBuilder

The BookBuilder class is used for building central order books and matching orders.

Members:

```
Message message; // object for parsed message
CentralOrderBook centralBook; // object for central OrderBook
Reader message_reader; // object for reader that is used for parsing
Writer parserWriter;  // object for writer that writes parsed results
time_t totalTime;    // used for record total time taken
std::vector<std::string> SymbolFilters =
    { "AAPL", "MSFT", "TSLA", "AMZN"}; // used for filtering the messages
int totalAdd = 0;
int totalDelete = 0;
```

Functions:

```
BookBuilder(const std::string &inputMessagePath,
             const std::string &outputMessageCSV); // constructor
~BookBuilder(); // destructor
void start();  // start of the building central order books and matching
orders process
void next();  // thet function that iterates through the data file to find
the next message and
void updateBook(); // the function that builds central order books and
matches orders
```

Order Side

The OrderSide enum represents if the order is a BUY order or SELL order.

```
enum class OrderSide : unsigned char {
    BUY,
    SELL
}
```

Order Type

The OrderType enum represents the type of order.

```
enum class OrderType : unsigned char {
    LIMIT,
    MARKET,
    STOP,
    STOP_LIMIT
};
```

Order

The Order class represents an order object.

Members:

`unsigned order_id` - The order ID of the order
`unsigned owner_id` - Owner of the order
`unsigned quantity` - The quantity of the stock
`unsigned quote` - The price/quote of the order
`unsigned stop_price` - The stop_price in case of a STOP/STOP_LIMIT order
`OrderSide order_side` - The order side
`OrderType order_type` - The order type
`char all_or_none` - Indicates if the order has to be completely executed or not. By default it is 0.
`std::chrono::time_point<std::chrono::system_clock> timestamp` - timestamp of added order

Functions

Below are getters to fetch private members of the order class.

```
unsigned get_id()const
unsigned get_owner()const
unsigned get_quantity()const
unsigned get_quote()
unsigned get_stop_price()
auto get_side()const
auto get_type()const
std::chrono::time_point<std::chrono::system_clock> get_time()const
```

`char isAON() const` - Returns if the order is All or None, returns 1 if yes else 0
`bool isBuy()const` - Returns a bool indicating if the order is a buy side order
`void reduce_quantity(unsigned qty)` - Reduces order quantity by qty
`void set_type(OrderType type)` - Sets the order type to the type passed
`void set_quote(unsigned qt)` - Sets the order quote to qt

Central Order Book

The `CentralOrderBook` maintains the order book for multiple stocks. It internally stores the order books in an unordered map.

Members:

```
std::unordered_map<string, OrderBook> order_book_map;
std::unordered_map<unsigned int, string> order_ticket_map;
```

The `order_book_map` is a map of the stock symbol to the `OrderBook` for that stock. The `order_ticket_map` is a map of order ID to the symbol name.

Functions:

```
StatusCode add_symbol(string symbol);
```

Add a symbol (a stock, e.g : Apple) to the `order_book_map`. This function creates a new `OrderBook` class for the new symbol.

Parameters:
- symbol - A string representing the stock symbol

Return Value:
Returns the `StatusCode` of the operation.
- Returns `OK` if the operation is successful.
- Returns `SYMBOL_EXISTS` if the symbol already exists in the order map.

```
StatusCode add_order(string symbol, Order& order)
```

Add an order for a particular stock symbol to the order book.

Parameters:
- symbol - A string representing the stock symbol
- order - The order object to be added

Return Value:
Returns the `StatusCode` of the operation.
- Returns `OK` if the operation is successful.
- Returns `ORDER_EXISTS` if the order already exists

```
std::optional<Order> get_order(unsigned int order_id)
```

Fetch an `Order` object given the order_id

Parameters:
- order_id - The order ID of the order

Return Value:
Returns the `Order` object if the order exists, else returns an empty value.

```
StatusCode delete_order(unsigned int order_id)
```

Delete an order given the order_id

Parameters:
- order_id - The order ID of the order

Return Value:
Returns the `StatusCode` of the operation.
- Returns `OK` if the operation is successful.
- Returns `ORDER_EXISTS` if the order already exists

```
std::pair<StatusCode, unsigned> best_ask(string symbol) const
```

Returns the best ask/sell price for a particular stock symbol.

Parameters:
- symbol - A string representing the stock symbol

Return Value:
Returns a pair of the `StatusCode` and best ask price.
- Returns (`OK`, price) if the operation is successful.

- Returns `(SYMBOL_NOT_EXISTS,std::numeric_limits<unsigned>::max())` if the symbol doesn't exist in the order map.
- Returns `(OK,std::numeric_limits<unsigned>::max())`, if there are currently no sell orders in the order book.

```
std::pair<StatusCode, unsigned> best_bid(string symbol) const
```

Returns the best bid/buy price for a particular stock symbol.

Parameters:
- symbol - A string representing the stock symbol

Return Value:
Returns a pair of the StatusCode and best bid price.
- Returns (`OK`, price) if the operation is successful.
- Returns `(SYMBOL_NOT_EXISTS,0)` if the symbol doesn't exist in the order map.
- Returns `(OK,0)` if there are currently no buy orders in the order book.

Orderbook

Members:

`unsigned last_buy_price` - Last buy matching price
`unsigned last_sell_price` - Last sell matching price
`std::string company` - stock symbol
`std::unordered_map<unsigned, std::list<Order>> buypool` - Map of price level to list of buy orders at that price level
`std::unordered_map<unsigned, std::list<Order>> sellpool` - Map of price level to list of sell orders at that price level
`std::unordered_map<unsigned, std::list<Order>> stop_buy_pool` - Map of price level to list of stop-buy orders at that price level
`std::unordered_map<unsigned, std::list<Order>> stop_sell_pool` - Map of price level to list of stop-sell orders at that price level
`std::set<unsigned, std::less<unsigned>> sellprices` - Sorted set of sell price levels
`std::set<unsigned, std::less<unsigned>> stop_buy_prices` - Sorted set of stop buy price levels
`std::set<unsigned, std::greater<unsigned>> buyprices` - Sorted set of buy price levels
`std::set<unsigned, std::greater<unsigned>> stop_sell_prices` - Sorted set of stop-sell price levels
`std::unordered_map<unsigned, OrderInfo> order_map` - Hash map of order ID to OrderInfo

Functions:

```
StatusCode add_order(Order& order)
```

Add an order to the order book.

Parameters:
- order - The order object to be added

Return Value:
Returns the `StatusCode` of the operation.
- Returns `OK` if the operation is successful.
- Returns `ORDER_EXISTS` if the order already exists

```
std::optional<Order> get_order(unsigned int order_id)
```

Fetch an `Order` object given the order_id

Parameters:
- order_id - The order ID of the order

Return Value:
Returns the `Order` object if the order exists, else returns an empty value.

```
StatusCode delete_order(unsigned int order_id)
```

Delete an order given the order_id

Parameters:
- order_id - The order ID of the order

Return Value:
Returns the `StatusCode` of the operation.
- Returns `OK` if the operation is successful.
- Returns `ORDER_EXISTS` if the order already exists

```
std::pair<StatusCode, unsigned> best_ask() const
```

Returns the best ask/sell price in the order book

Return Value:
Returns a pair of the `StatusCode` and best ask price.

- Returns (`OK`, price) if the operation is successful.
- Returns `(SYMBOL_NOT_EXISTS,std::numeric_limits<unsigned>::max())` if the symbol doesn't exist in the order map.
- Returns `(OK,std::numeric_limits<unsigned>::max()),` if there are currently no sell orders in the order book.

```
std::pair<StatusCode, unsigned> best_bid() const
```

Returns the best bid/buy price in the order book.

Return Value:
Returns a pair of the StatusCode and best bid price.

- Returns (`OK`, price) if the operation is successful.
- Returns `(SYMBOL_NOT_EXISTS,0)` if the symbol doesn't exist in the order map.
- Returns `(OK,0)` if there are currently no buy orders in the order book.

```
void match(Order& order, bool isMarket)
```

Match an `Order` object

Parameters:

- order - passed by reference
- isMarket - bool, whether the order is market order.