Central Limit Order Book

A Central Limit Order Book is a trading method used by most exchanges globally. It is a transparent system that matches customer orders on a “price time priority” basis. The highest (“best”) bid order and the lowest (“cheapest”) offer order constitutes the best market or “the touch” in a given security. Customers can routinely cross the bid/ask spread to achieve low cost execution.

Whenever an order comes into the system, the order book’s matching logic will check if the order can be matched, and modify the order book accordingly. Order execution only happens during this logic, and it happens immediately.

Price level:
When several orders contain the same price, they are referred to as a price level. For example several sell orders all have the same price, then they can all fulfilled a buy order at the same price.

Order Queue:
When there are several orders with the same price, the matching of the orders happens in a FIFO manner, meaning the first order that occupies a price level will be matched first.

Order Amendment:
When an order quantity is ameneed down, it's order queue position does not change. When a order’s quanaityt is amended up, it will be moved to the end of the queue.

Tick Size:
Tick size usually refers to the minimum price movement of a trading instrument. For example, if a stock has a tick size of 0.5, its price can only move up or down only by n*0.5 each time.

Project Requirements:

Implement a limit order book that support the following functions:

1. Limit order
    a. User is able to send buy or sell limit orders.
    b. The program should be able to identify if the order price has the correct tick size. If the order does not have the correct tick size, the program rejects the order by not doing anything.
2. Cancel
    a. User is able to cancel an order by order id before the order is executed. When the order is canceled it will no longer be inside the order book. If the order is already executed, the program rejects the cancel request by not doing anything.
3. Amend

    a. User is able to change the quantity (not the price) of an existing order.
    b. When the order quantity is increased (amend up), the order moves to the back of the queue.
    c. When the order quantity is decreased (amend down), the order remains in its current queue position.
4. Query
    a. Query level. User can check the order book at each level using query command to see **price**, **quantity, and number of items in this level.**
    b. Query order. User can check the order' data, including **order status** (open, partially filled, fully filled, cancelled), **leave quantity** (the quantity that has not yet filled), **queue position** (0 for front of the queue)

These functions should be accompanied with proper test cases.

Command Format:

Limit order:
1. "order 1001 buy 100 12.30"
2. "order 1002 sell 100 12.20"
3. "order 1003 buy 200 12.40"
4. "order 1004 buy 300 12.15"

Cancel order:
1. "cancel 1003"

Amend order:
1. "amend 1004 600" will try to change order 1005 from buy 300 shares to buy 600 shares and move it to the back of the queue.

Query:
1. "q level ask 0" will print the price and total quantity of the best ask level in a comma separated format this "ask, 0, 12.15, 100".
2. "q level bid 1" will print the price and total quantity of the second bid level. E.g. "bid, 1, 12.10, 200"
3. "q order 1004" will print the order status of order_id 1004. The queue position should be the back of the queue if the order is amended up.

Project setup

Please host the project code in Github as a public repo so that we can easily look at it.
Unit Test with boost library (1.71, the current version)
Build using Cmake (cmake >= 3.10)
System: Mac (Mojave) or Ubuntu (18.04)
Compiler: clang or g++

The project should have at least two binary outputs:

1. order_book
2. order_book_test

“order_book” receives input from stdin
“order_book < input.txt” will run the program with the input.txt as the input.
“order_book_test” will run the unit tests.

Please make the input processing logic as simple as possible, assume all input is going to be in the perfect format.

Time frame

Preferably the practise can be done in a few hours (2~5). If you prefer to make it better and therefore want to spend more time on it, let us know and we will review the project with that in mind.

Definition of Done:

1. Successfully compile and produce two executable files.
2. A runnable order_book program, one txt files for it as input data, and expected output to compare with.
3. A runnable order_book_test program that can run all the test cases.
4. A public Github repo with all the code.