GINKGO LLC CODE TEST

The following material is designed by Ginkgo LLC (the Company). The candidate (You) shall not disclose or publish the material to any other individual, entity or online sources. This document is protected by your signed Ginkgo LLC NDA.

NOTE: The following problem is designed to assess your proficiency in performance driven programming and critical thinking in data structures and basic algorithms. While the subject is related to what the Company does, you should not misinterpret this as your primary type of work to perform at the Company. In reality the Company covers a much greater scope of modern computing and data analysis, such as statistics, graphics, signal processing, computer vision, machine learning, market connectivity, hardware-software co-design, trading operations, and so on. Have fun computing!

Simple Order Book

Order book is a data structure that maintains the price and size information for each side, BUY and SELL, for a given financial security. In the financial market, participants can send an order to buy or sell a security with a certain price and a size to the exchange. Order book is used to keep the current buy and sell orders and match the orders crossing the other side at the exchange. However, in this simple order book implementation, matching operation is not considered.

In the C++ implementation of the order book, you are required to implement the following OrderBook class with five public member functions to update and retrieve the order book.

```
class OrderBook {
public :
void add(int order_id, char side, double price, int size);
void modify(int order_id, int new_size);
void remove(int order_id);
double get_price(char side, int level);
int get_size(char side, int level);
};
```

There are three order book update functions, add(), modify(), and delete(). The argument **side** can be either ‘B’ or ‘S’ representing BUY and SELL respectively.
(1) void add(int order_id, char side, double price, int size)
Add a new order with a unique order id, its side, price, and order size.

(2) void modify(int order_id, int new_size)
Modify the order size of an existing order to the new size.
(3) void remove(int order_id)
Delete the existing order from the order book

note: Order ids are unique and not reused. When all orders at a certain price level is gone, the level has to be removed.

There are two order book retrieval functions, get_price() and get_size(). In both functions, the second argument, **level**, can be any positive number starting from 1. Top level, 1, represents the lowest price for the SELL side and the highest price for the BUY side respectively. Accordingly, level 2 represents the second lowest price for the SELL side and the second highest price for the BUY side.

(4) double get_price(char side, int level)
Return the price for the given side and level in the order book.
(5) int get_size(char side, int level)
Return the aggregate size of all the orders in the level for the given side of the order book.

Note that the above OrderBook class provides the minimum interface function declarations. You need to complete the implementation of five member functions by adding necessary data structures and any additional private member functions as needed.

Input Data Format
Input data will be provided by a file that contains one of the following five possible formats in each line of the file.
(1) add order_id side price size
(2) modify order_id new_size
(3) remove order_id
(4) get price
(5) get size
Double slash, ie. //, indicates a comment and may be ignored by the program

Here are some examples:

Input file:

```
add 1 B 45.2 100
modify 1 50
get price B 1 //this returns 45.2
add 2 S 51.4 200
```

```
add 3 B 45.1 100
get size S 1 //this returns 200
add 4 S 51.2 300
add 5 S 51.2 200
remove 3
get price B 1 // this returns 45.2
get size B 1 //this returns 50
get price S 1//this returns 51.2
get size S 1 //this  returns 500
```

Running your executable on this file should print out results:

```
45.2
200
45.2
50
51.2
500
```

Input file:

```
add 1 B 22.5 100
add 2 S 37.8 250
add 3 B 24.7 150
get price B 1 //this returns 24.7
get price B 2 //this returns 22.5
modify 3 50
add 4 S 35.1 250
add 5 S 37.8 150
get price S 1//this returns 35.1
remove 3
get size S 1 //this returns 250
get size S 2 //this returns 400.
remove 5
add 6 S 37.8 150
add 7 S 37.6 350
add 8 B 24.7 200
get size B 1 //this returns 200
get price S 2//this returns 37.6
modify 8 150
add 9 S 35.1 200
add 10 B 22.5 350
get size B 2 //this returns 450
get price S 3//this returns 37.8
```

Running your executable on this file should print out results:

```
24.7
22.5
35.1
250
400
200
37.6
450
37.8
```

Restrictions

Please, DO NOT USE boost or any additional libraries to complete this coding problem. You are allowed to use C++ STL containers if needed. Your program will be compiled and tested using gcc without linking with any additional libraries.

Evaluation Criteria

Your program will be tested against a testing data file consisting of lines of three input formats.

Evaluation of the program will be done by;

(1) Can the code be compiled?

(2) Correctness of the output.

(3) Performance of the program will be measured by its execution time against a large sample data set.

(4) Clarity of the code.

Put all of your writing and results in a single PDF file format. This conversion is part of the test; only this PDF file is used to evaluate your performance.