Tax-free Stock Market

In this project, you are asked to build a toy system mimicking a tax-free stock market. The system supports a few common stock transactions. We will first introduce how our toy stock market works. Read the description carefully since the rules might be different from your experience in the real world.

Mechanism of tax-free stock market

First, we have stocks and users in the market. Each stock has many shares, and its price is determined by the buying and selling orders from users. Specifically, a stock includes two list of orders: the buying orders and the selling orders. A user who holds shares of a certain stock can post an order trying to sell his/her shares at some price. Similarly, a user with enough cash can also post a buying order with a target price and the amount of shares. For example, if Alice holds 100 shares of stock A, she could post a selling order under stock A as "sell 20 shares at price $\$100$". On the other hand, if Bob wants to buy some shares, he could post a buying order under stock A as "buy 10 shares at price $\$90$". Notice that for a particular stock, the price in a buying order must be smaller than that in any selling order, otherwise the buyer could just consume the selling orders without posting a buying order.

Each stock maintains the buying orders and selling orders in two separate lists. Each list is sorted based on price. The buying orders are sorted from the most expensive to the cheapest, while the selling orders are sorted from the cheapest to the most expensive. And the price of a stock is defined as the price of the cheapest selling order at the moment. In particular, if the list of selling order is empty, the price of the stock is 0.

When a stock is added to the market, the market account (with usr_id 0) initially holds all shares of such stock and it post all these shares in one selling order under the new stock with the initial price.

Our market also implements the "circuit breaker" mechanism to make sure that the daily price fluctuation is within 10% for each stock. Specifically, if current_price > start_price_of_the_day $\times 1.1$ or
current_price < start_price_of_the_day $\times 0.9$, the stock will "freeze" for the rest of the day, i.e., the stock accepts no transactions. In particular, if the start price of a stock is 0, the stock will never freeze on that day.

For a user account, it maintains a cash balance and the personal buying/selling orders ever posted. A user can perform the following two basic transactions :

- buy/sell k shares of stock A at maximum price p. For example, suppose Alice wants to buy 30 shares of stock A at maximum price $\$25$. And suppose stock A currently has three orders under its selling order list: 1.(share : 10, price : $\$10$), 2.(share : 15, price $\$20$), 3.(share : 100, price $\$30$). In this situation, Alice would first consume order 1 for 10 shares (trading with the owner of order 1 at price $\$10$) and then consume order 2 for 15 shares (trading with the owner of order 2 at price $\$20$). At this point, Alice still want 5 more shares but she cannot find matching orders in the selling list. Therefore, she will proceed to the next transaction type:
- add a buying/selling order to stock A. Continuing with the above example, Alice will post a buying order as (share : 5, price 25). Note again that the price in a buying order must be smaller than that in any selling order. A user could also simply add a buying/selling order (i.e. only do transaction 2).

Since the market is tax-free, all transactions are bi-directional, and we must be able to `undo` any of the transactions, i.e., return to the state before the **last user transaction**. The user transactions are considered **atomic**, meaning that either it is executed as a entirety or it appears to be never happend. You are only required to implement "undo" for the above two user transactions.

Framework

We have provided a code skeleton for you in `stock.h`. You are allowed to change anything except for class `Market`, which has the interface you are supposed to implement. Read the `stock.h` for more details.

```
void newday(); // begin a new day
void add_usr(int usr_id, double remain); // add a user
void add_stock(int stock_id, int num_share, double price); // add a stock
bool add_order(int usr_id, int stock_id, int num_share, double price, bool order_type); // add an order, return 1 for success, 0 for fail
int usr_buy(int usr_id, int stock_id, int num_share, double price); // buy a stock, return the amount of shares successfully traded, -1 for Abort
int usr_sell(int usr_id, int stock_id, int num_share, double price); // sell a stock, return the amount of shares successfully traded, -1 for Abort
pair<int, double> best_stock(); // the stock with highest price, (stock_id, price)
pair<int, double> worst_stock(); // the stock with lowest price, (stock_id, price)
double evaluate(int usr_id); // the value of an account (the money it holds suppose it could successfully sell all selling orders, neglect the buying orders and shares not for selling)
void undo(); // undo
```

Grading

This project is worth 30 points directly to your final grade:

- Correctness: we will evaluate your program with 5 test cases, 4 points each. Unlike the homework, we will only release one test case to help you debug. To get full score, you should test your program extensively, especially with corner cases.
- Performance: 3 points. Grading rubric: **TBD**. Don't worry too much here. As long as your program is not absurdly slow, you will get full points.
- White box inspection: 5 points. We will read your code and grade based on code structure and style. The TA's won't be too strict on this. Example situations where you might get points deducted here:
  - Completely ignoring the OOP framework;
  - Meaningless variable names, for example,

    ```
    class A;
    class B;
    void foo(int _, double __, A a, B b);
    ```

  - Using `public` everywhere;
  - Some other behaviors severely violating the principles of OOP.

- Please write a report for this project. In the report, you should briefly describe your work. The report worth 2 points.

Please finish all the code individually . Plagiarism of any kind would be dealt seriously.