Project 6: Algorithmic Trading

Due Wednesday 12 November at 11pm

Background

As everyone knows, the Battle Creek Commodity Exchange (BCCE) is the premier trading forum for speculating on the ups and downs in the dynamic cereal marketplace. Bowing to popular demand from investors, the BCCE has introduced a new capability for index trading. The Breakfast Opportunities Hedge Fund (a financial trading firm specializing in tasty morning commodities) has asked you to develop a BCCE market simulation so they can try out their ideas for trading strategies in these new markets.

The BCCE trading floor specializes in the exchange of contracts for delivery of popular cereal brands at a specified date. One share of the security entitles the bearer to one kilobowl of the associated cereal. In our simulation, we focus attention on the two most popular contracts, for the cereals *Sugar Bombs*™ (ticker symbol: 'B'), and *Flakey Pops*™ (ticker symbol: 'F'). There is also a new index contract (ticker symbol: 'I') which is essentially equivalent to having one kilobowl of *each* of the cereals in the index—in this case both Sugar Bombs and Flakey Pops.

Simulating the BCCE

For the simulation, the hedge fund generates a sequence of orders that flow into the market. Each order is defined by a ticker symbol ('B', 'F', or 'I'), a quantity (either 1 for buying one unit, or –1 for selling one unit), and a price. Your **first task** is to implement the processing of these orders.

The sequence of input orders is specified in a file, the name of which is provided as input. Each line of the file contains one order, specified by the three elements above, in order. The file may contain any number of orders. All prices are positive integers, expressed in units we call *ticks*.

Your BCCE simulation will process the orders in the given sequence. Maintain the state of trading in each security separately. Processing an order has two possible outcomes:

1. If an order matches an existing order for that security, the two orders will *transact*, at the price of the existing order.
2. If there is no match, the new order gets added to the *order book* for that security.

The order book consists of two lists, one for buy orders (quantity 1) and one for sell orders (quantity –1). When a new buy order comes in at price p, the market checks whether there is an existing sell order at price p or lower. If there are several, the lowest sell order wins. If there are several orders at the same price, the one received first gets priority. The new buy transacts with this best existing sell, at the price of the existing sell. The process is analogous for a new sell order: check for a compatible existing buy, and if there are many pick the highest/earliest, and transact at the price of the existing buy.

Note that the state of an order book can be summarized by two numbers: the highest outstanding offer to buy (called the “bid quote”), and the lowest outstanding offer to sell (called the “ask quote”). The bid quote tells a trader the price it could get by selling in the current state, and the ask quote communicates the price at which one could currently buy. The combination of these two numbers is called the *bid-ask spread*. If there are no buy offers in the order book, the bid quote is undefined, and if there are no sell offers the ask quote is undefined.

You may find it convenient to maintain the buy and sell lists of the order book in order by price: descending (highest first) for buys, and ascending for sells.

When a transaction occurs, you need to report this event on the ticker tape, which for this project goes to the standard output stream. The ticker output has one line for each transaction, with the ticker symbol, a colon, one space, and the price. For example, if there is a transaction for Flakey Pops at price 55, the line in the ticker output would be:

```
F: 55
```

For example, suppose that order input file comprises the following orders:

```
B  1 10
B  1 12
B -1 20
B -1 17
B  1 14
B -1 13
B  1 16
B  1 18
```

Notice that in this example, all the orders are for security 'B'. This lets us focus on a single market. None of the first five orders result in any transactions, since all of the buy prices are below all the sell prices. After these orders have been processed, the order book would be as follows:

BUY: [14 12 10]
SELL: [17 20]

The sixth order is an offer to sell at a price of 13. This *does* result in a transaction, since there is an outstanding buy offer above this price. The trade occurs at price 14 (the price of the existing buy offer), and that value is removed from the BUY list. The next buy offer (price 16) does not transact, so gets added to the list. The last one (price 18), however, transacts with the sell offer at a price of 17. The final result output to the ticker would be:

```
B: 14
B: 17
```

At the end of processing, the order book would be:

BUY: [16 12 10]
SELL: [20]

Remember that in the actual order data there will generally be interleaved orders for securities 'F' and 'I' as well, so you will need to maintain three order books: two lists for each.

Trading Strategies

Your **second task** is to support the testing of hedge-fund trading strategies. For this project we consider only simple strategies, which consider the bid quote and ask quote for each security, and decide what orders to submit based on this information. Specifically, we are to implement two strategies: index arbitrage, and market making. We assume that traders can buy or sell any amount of the security; you do not need to worry about checking for positive cash balances or holdings of any security prior to trading.

Index Arbitrage

The *index arbitrage* (IA) strategy the hedge fund wants you to implement would choose to buy the index and sell the individual cereals, whenever the price for the index is less than the total you could get by selling the two cereals. Conversely, if there is an opportunity to buy the cereals at a combined price less than at what you could sell the index, IA would do that. Otherwise, IA refrains from trading.

For example, suppose the price quotes are as follows.

- B: bid 16, ask 20
- F: bid 35, ask 39
- I: bid 48, ask 50

Then IA could make a profit of one tick by selling the two cereals and buying the index.

To implement the IA strategy, your program must include the logic to make the decision above, as well as to execute any trades decided. To execute a trade, your program should submit orders to the market implemented in Task 1. For example to buy a unit of security X, you would submit an order to market X to buy one unit at or above the current ask quote. (Since the orders match if and only if the buy is greater or equal to the sell, it does not matter what exact price the new order is, as long as it satisfies this condition.) For example, in the situation above, IA could submit the following orders:

```
B -1 16
F -1 35
I 1 50
```

All of these will transact at the price of standing orders, yielding a profit of one tick.

Market Making

In a *market making* (MM) strategy, the MM submits both buy and sell bids, in an effort to make profits off the *spread*, or difference between them. In the version we consider, the MM has a threshold, and submits orders if and only if the current spread in the price-quote is defined, and meets or exceeds the threshold. If so, MM places orders to buy and sell within the quote, taking into account any existing orders it may have at that market.

Specifically, if the trigger condition is met, the MM submits bids as follows.

- If the MM has a current buy order at the bid quote, it leaves that alone. If it has a current buy order below the bid quote, it withdraws that and replaces it with a buy order at one tick above the bid quote. If it has no current buy order, it places one at one tick above the bid quote.
- Its behavior for sell orders mirrors that for buys. A current sell order at the ask quote is left alone. A current sell order above the ask is withdrawn, and in this case or if it has no sell order it places one at one tick below the ask.

For example, suppose that MM has a threshold of 4, and encounters the following situations, defined by its own outstanding orders and price quotes for security B.

1. Own orders: buy (none), sell (none); quote: bid 16, ask 20. MM submits orders to buy at 17 and sell at 19.
2. Own orders: buy 17, sell 19; quote: bid 17, ask 19. No action.
3. Own orders: buy 17, sell (none); quote: bid 17, ask 24. MM places sell order at 23.
4. Own orders: buy 17, sell 23; quote: bid 18, ask 23. MM withdraws current buy, replaces with buy at 19.
5. Own orders: buy (none), sell 23; quote: bid 16, ask 22. MM places buy at 17, withdraws current sell, replaces with sell at 21.
6. Own orders: buy 17, sell 21; quote: bid 18, ask 20. No action.

Strategy Implementation

You are to implement both the IA and MM strategies, and evaluate them with the input order stream. The processing proceeds by executing the following steps in sequence, as long as orders remain in the input.

1. Read in one order, and submit it to the market for the corresponding security, updating the order book as appropriate and reporting any resulting transaction to the ticker output.
2. Run the IA strategy. This entails obtaining price quotes from each market, and submitting appropriate orders if an arbitrage opportunity is found. To execute an arbitrage, the IA must submit to the markets in the following sequence: B, F, I. After executing the arbitrage, your program should print the following to the ticker output (standard output stream):

   `Index Arbitrage: profit =` *profit_amount*

3. Run the MM strategy. For each market, get the price quotes and submit orders as dictated by the strategy (the sequence does not matter). Whenever any transactions involve the market maker, add an annotation to the ticker to reflect that fact. For example, if a transaction on security B involved a sell order from MM, the ticker line would read:

   ```
   B: 17 (MM sell)
   ```

When the end of the order input stream is reached, you are to report the cumulative results for each strategy. For IA, the relevant statistic is final cash holdings:

```
IA final holdings: cash = final_cash_balance
```

The summary for MM is a bit more complicated, since it takes into account any holdings (positive or negative) of the securities. Your report should take the following form:

```
MM final holdings: cash = final_cash_balance
Security B: B_holdings units, value = B_value
Security F: F_holdings units, value = F_value
Security I: I_holdings units, value = I_value
Total profit is total_profit.
```

where the value of each security is based on the price of the last transaction. For example, if the MM ends up with –6 units of B (i.e., it sold 6 units more than it bought), and the final price of B was 100 tickets, then *B_value* is –600. Total profit is the sum of cash balance and value of the three securities.

Input and Output

Your program will be given two lines on the standard input stream. The first gives a string with the filename of the order input file. The second gives an integer (≥ 3) with the spread threshold to be used by the MM strategy.

Output requirements are described above. Example output traces will be posted.

Tips and Coding Requirements

As always, we recommend employing procedural and data abstraction to the extent possible. We recommend developing a class for markets, and *require* implementation of a base class for trading strategies, with derived classes for index arbitrage (IA) and market making (MM). You should use inheritance in a nontrivial way, with substantial functionality in the base as well as derived classes.

Grading

10 points for submitting something that compiles.

10 points for employing proper style, including the specific requirements for this assignment (above), and the general course C++ style guidelines.

35 points for correctness on a variety of test inputs.