Introduction: Suppose John intends to buy 100 shares of Global Widget Corp (GWC), and the maximum price he is ready to pay per share is $99. This is a **Limit Order**, since he has specified the limiting criteria – the maximum quantity he needs and the maximum price he is willing to pay. A limit order to buy is called a **Bid**. Similarly, if Sally wishes to sell 200 shares of GWC and the minimum price she is willing to accept per share is $101, she would send a limit order to sell, which is called an **Offer**. Matching engine accepts limit orders continuously and these form the **Order Book**.

Normal Priority Rules:

If John & Sally's orders from the above example are the only orders, no trade can happen, since John is only willing to pay up to $99, but Sally would only accept a minimum of $101. Now, Forest comes in with a limit order to buy 100 shares for $101. Bueno! We can have a trade here - Forest buys 100 shares from Sally for $101. After the trade, Sally's offer remains on the order book, but with 100 shares outstanding.

But, what if the limit price on Forest's bid was $102? The trade would still be conducted at $101, even though Sally would be happier to sell at $102. The trade price is determined at the limit price of the order that came earlier to the order book. A trade is immediately conducted upon receipt of a limit order whenever possible, after which the quantity on the interacting bids or offers on the order book is reduced by the quantity traded.

Since multiple orders can reside in a same price, the matching engine uses **Price - Time - Size** priority for conducting trades. When an incoming sell (buy) order can trade with multiple available bids (offers), the following hierarchy is used.

1. Price: Bids (Offers) with higher (lower) price trade first,
2. Time: If price is equal, bids (offers) sent earlier to the matching engine would trade earlier,
3. Size: If there are multiple bids (offers) at the same price and time, they are traded in descending order of size.

Auction Period Rules:

However, during an auction period, no trades happen immediately even if John, Sally & Forest's orders can match each other. Rather, everyone's orders accumulate in Order Book and wait for the simultaneous match at the end of the auction. Then, at what price should the match occur? To maximize everyone's benefit, the matching engine should calculate a price that maximizes the total transaction amount and match as many orders as possible.

For example, John intends to buy 100 shares at $99, Sally wishes to sell 200 shares at $101, Forest wishes to buy 100 shares at $102, Wood wishes to sell 50 shares at $100.

1. If the match price is $99. No trade occurs. Only John accepts this price to buy, no else accepts this price to sell.
2. Suppose that the match price is $100. Forest accepts this price to buy (since it is <= his price $102) and Wood accepts this price to sell (since it is >= his price $100). Yet, only 50 shares can be traded.
3. Suppose that the match price is $101. Forest accepts this price to buy (since it is <= his price $102) and both Wood and Sally accept this price to sell (since it is >= his price $101/100). Now all of Forest's 100 shares can be traded.
4. When the match price of $102, Forest, Wood, and Sally will accept this price and all of Forest's 100 shares can be traded.
5. Finally, how about $103? Sadly, nobody wants to buy at that price.

Thus, the estimated match price is $101 or $102, (or any price in between $101 and $102). Among them, let's pick $102 ($102 * 100 = $10200 amount) as its total transaction amount is bigger than $101 ($101 * 100 = $10100 amount). Note that, we want to maximize the total transaction amount, neither the total transaction share, nor the number of people involved in all transactions. In the rare case that the total transaction amounts are same, use the maximum of match prices among potential solutions.

In summary, the matching engine does not process the request during an auction period immediately. Instead, when an incoming sell (buy) order comes, the matching engine waits and calculates the expected matching price continuously. At the end of the auction, all matches happen simultaneously at the calculated matching price which maximizes total transaction amount.

Note that the matching engine uses **Price - Size - Time** priority for conducting trades, since the time of all incoming sell (buy) orders are treated equality during an auction period, except among orders whose price and size are same.

Inventory: When Kaylee buys 100 shares from Joy, Kaylee's inventory goes Long by 100 while Joy's inventory goes Short by 100.

Consider a sequence of Sally's trades: Buy 30, Sell 100, Sell 60, Buy 100

Total Long positions: 30 + 100 = 130; Total Short Positions: 100 + 60 = 160; Net Position: Short 30

Dataset: orders.csv contains a list of limit orders sent to the matching engine during a trading day. The entries have the following format: ID, party, price, quantity, timestamp, side

Example: 673, River, 100.42, 200, 100044, BUY

This means River placed an order at timestamp 100044 to buy 200 shares at price $100.42

About 10,000 lines or 1MB data will be given for your development. About 1,000,000 lines or 100MB data will be used to verify your code.

Problem 1: What is everyone's net position at the end of the day if normal priority rule is used?

Problem 2: What is the expected matching price which maximizes the total transaction amount and everyone's net position at the end of auction if auction priority rule is used?

Problem 3 (extra points): Along with everyone's net position, please analyze the following:

- historical net position from the beginning to the end
- profit and loss
- total volume
- max, min, avg position

Output: Your program needs output two values per person for the Problem 1:

- <X> = L or S depending upon each person being net long or short respectively;
- <num> is the absolute value of net position.

And the expected matched price for the Problem 2.

Apart from the correct answer, you should contain a source code and tell us how to run the program.

Requirements: please implement an **online** algorithm which processes the dataset line by line, not an **offline** algorithm which processes the entire dataset simultaneously.

You may use your preferred language, such as C++, as well as available standard libraries (stl, boost, etc). Please include other necessary files, such as makefile or project file.

Finally, we are looking for a **clear, efficient & concise** code. We believe that a qualified candidate should have good coding style as well.