100/x

Order Book Coding Assignment

An exchange allows the buyers and sellers of a product to discover each other and trade. Buyers and sellers (traders) submit orders to the exchange and the exchange applies simple rules to determine if a trade has occurred. The dominant kind of exchange is a central limit order book (CLOB) where orders are matched using 'price time priority'.

When placing an order, traders specify if they wish to buy or sell, the limit price ie. worst possible price they will trade at, and the volume (number of shares) they wish to trade. On our exchange trades only occur during the processing of a newly posted order, and happen immediately, which is known as 'continuous trading'.

Matching example

As orders arrive at the exchange, the are considered for aggressive matching first against the opposite side of the book. Once this completes, any remaining order volume will rest on their own side of the book. Consider 3 orders have been submitted to the exchange, in the following order:

- Buy 1000 @ 99
- Buy 1200 @ 98
- Buy 500 @ 99

As there are no Sell orders yet, they rest on the order book as follows (note Buy for 98 is lowest priority):

Bids (buying)		**Asks (selling)**	
Volume	**Price (p)**	**Price (p)**	**Volume**
1,000 500 1,200	99 99 98		

Price time priority refers to the order in which orders in the book are eligible to be matched during the aggressive phase. Orders are first matched in order of price (most aggressive to least aggressive), then by arrival time into the book (oldest to newest). A **Sell** order is now submitted, with a limit price that does not cross with any of the existing resting orders:

- Sell 2000 @ 101

Bids (buying)		**Asks (selling)**	
Volume	**Price (p)**	**Price (p)**	**Volume**
1,000 500 1,200	99 99 98	101	2,000

A **Sell** order is now submitted that is aggressively-priced:

- Sell 2000 @ 95

Order Book Coding Assignment

This triggers a matching event as there are orders on the **Buy** side that match with the new **Sell** order. The orders are matched in price time priority (first by price, then by arrival time into the book) i.e.

- Buy 1000 @ 99 is matched first (as it is the oldest order at the highest price level)
- Buy 500 @ 99 is matched second (as it is at the highest price level and arrived after the BUY 1000 @ 99 order)
- Buy 500 @ 98 is matched third (as it is at a lower price. This partially fills the resting order of 1200, leaving 700 in the order book)

Bids (buying)		**Asks (selling)**	
Volume	**Price (p)**	**Price (p)**	**Volume**
700	98	101	2,000

100/x

Order Book Coding Assignment

A: Limit order handling

The assignment is to produce executable code that will accept orders from standard input, and to emit to standard output the trades as they occur. Once standard input ends, the program should print the final contents of the order book.

Order inputs will be given as a comma separated values, one order per line of the input, delimited by a new line character. The fields are: `order-id, side, price, volume`. Side will have a value of 'B' for **Buy** or 'S' for **Sell**. Price and volume will both be integers. `order-id` should be handled as a string.

Example 1

In this example no buyer or seller is willing the pay the opposing price so no trades occur.

```
$ cat test1.txt
10000,B,98,25500
10005,S,105,20000
10001,S,100,500
10002,S,100,10000
10003,B,99,50000
10004,S,103,100
$ ./exchange < test1.txt
     50,000     99 |    100         500
     25,500     98 |    100      10,000
                   |    103         100
                   |    105      20,000
$
```

Which represents the following order book:

Bids (buying)		Asks (selling)	
Volume	**Price (p)**	**Price (p)**	**Volume**
50,000	99	100	500
25,500	98	100	10,000
		103	100
		105	20,000

Example 2

If an order is then submitted to Buy 16000 @ 105p, it will fill completely against the resting orders, producing the following output:

```
$ cp test1.txt test2.txt
$ echo "10006,B,105,16000" >> test2.txt
$ ./exchange < test2.txt
trade 10006,10001,100,500
trade 10006,10002,100,10000
trade 10006,10004,103,100
trade 10006,10005,105,5400
   50,000     99 |    105    14,600
   25,500     98 |
```

```
$
```

Trade output must indicate the aggressing `order-id`, the resting `order-id`, the `price` of the match and the `volume` traded, followed by a newline.

The order book output should be formatted to a fixed width using the following template:

```
Buyers                Sellers
000,000,000 000000 | 000000 000,000,000
```

Please note that once submitted, orders are not modified by further input. There is no need to maintain more than one order book, all orders are for the same product.

BitMEX will test your application using a variety of inputs.

B: Handling Iceberg Orders (*Bonus/Optional*)

The input format now supports an additional column, `visible-quantity` as the fifth value to indicate an iceberg order.

The implementation of Iceberg orders should follow the supplemented document '**SETSmm and Iceberg Orders**' published by the London Stock Exchange sections 4.2 - 4.2.3.2. The relevant documentation on Iceberg orders begins on page 22

The output format is unchanged.

```
$ cat test_ice2.txt
10000,B,98,25500
10005,S,101,20000
10002,S,100,10000
10001,S,100,7500
10003,B,99,50000
ice1,B,100,100000,10000
$ ./exchange < test_ice2.txt
trade ice1,10002,100,10000
trade ice1,10001,100,7500
   10,000    100 |    101    20,000
   50,000     99 |
   25,500     98 |
$
```

The order book display shows the currently visible portion of the iceberg order `ice1`.