Abstract

This document describes a programming task to take an input file of order book feeds and produce price depth snapshots in a human readable format (example given in Task Description).

The first section, Order Book & Price Depth, details what an order book and a price depth are and how they relate to each other. This will be helpful to understand what it means to handle an order book feed and produce a price depth snapshot.

The second section, Task Description, defines the scope of the task and necessary details on what an order book is and what a price depth is. Knowledge of these structures is required to understand and build an appropriate solution.

The third section, Partial Implementation, describes the provided partial implementation of the programming task which implements the parsing of an input file.

The final section, Messages, is the reference material describing the set of possible messages within the input file. Information includes the data types and the order of fields for each message.

Order Book & Price Depth

For us to produce depth snapshots whenever we receive an order book feed we must understand what a price depth is.

A price depth is an aggregated view of the order book for a symbol. The aggregate is volume for a specific side and price. To illustrate, consider the below example of an order book for a symbol. Buys (bids) and sells (asks) are separate and each entry represents a unique order in the book - identified by Order ID. Bids are sorted in descending price order, asks are sorted in ascending price order.

Buy Orders (Bid)				Sell Orders (Ask)		
Order ID	**Volume**	**Price**		**Price**	**Volume**	**Order ID**
10200	150	101		102	5	10108
10444	50	101		103	10	10040
10232	300	100		103	10	10088
10333	50	99		104	100	10101
10215	150	99				
10010	100	98				

A price depth showing only 3 levels for the above order book would look like the following:

Bid			Ask	
Volume	**Price**		**Price**	**Volume**
200	101		102	5
300	100		103	20
200	99		104	100

We can see that orders at price 101, for example, are aggregated for a total of 200. There is only one entry per price point on bid or ask. Depending on market state, it’s possible for bid and ask prices to be in cross. That is, bid prices can be greater than ask prices.

Another way to view the price depth is as a ladder. The below is the same data, displayed differently. The result is the same, volume is aggregated by price, and bid or ask sits either side of the price.

Bid Volume	Price	Ask Volume
	104	100
	103	20
	102	5
200	101	
300	100	
200	99	

Task Description

In this exercise you're required to process an input stream of order book messages and output a price depth snapshot of the top N levels each time there is a visible change, where N is given as a command-line argument. This means that given symbols VC1 and VC2, in the event that we process an order book message that changes the depth snapshot VC1 we write the updated snapshot for VC1 but not VC2. You only need to publish a snapshot when it has changed from the last snapshot printed for that symbol, which means if you're displaying the top 5 levels and the 10th price level changes there should be no output.

The messages in the input file represent order book insertion, update, delete and executed. The structure of those messages and their interpretation is described in Messages.

Snapshots are to be output one-per-line in the following format:

```
sequenceNo, symbol, [(bidPrice1, bidVolume1), ...], [(askPrice1,
askVolume1), ...]
```

Where `sequenceNo` is taken from the corresponding field of the input message that caused the change.

The order of the bid and ask arrays should be strictly by competitiveness. Bids must have the highest bid price first, asks must have the lowest ask price first.

For example:

```
$ cat input1.stream | ./your_app 5
1, VC0, [(318800, 5000)], []
2, VC0, [(318800, 5000), (315000, 2986)], []
3, VC0, [(318800, 4709), (315000, 2986)], []
4, VC0, [(318800, 4709), (315000, 2986)], [(318900, 360)]
5, VC0, [(318800, 4709), (315000, 2986)], [(318900, 159)]
6, VC0, [(318800, 4709), (315000, 2986)], []
7, VC0, [(319000, 888), (318800, 4709)], []
8, VC0, [(319000, 221), (318800, 4709)], []
9, VC0, [(318800, 4709)], []
```

Partial Implementation

To aid in the development process, a partial implementation has been provided that will read from the standard input and decode the binary input format into data structures. Note that not all message types have been implemented.

Provided files:

`main`

Loops until all data in stdin is consumed, forwards data to `message` to decode and return the appropriate message type. Additionally prints the message in human readable format for debugging purposes.

`message`

Reads data from input buffer and decodes into appropriate data structure. Either returns a message or returns nothing on end of stream.

Messages

In this section we will provide details on the generic data types present in our input file and the structure of the messages that can be received. All messages are binary encoded and little-endian where appropriate.

The different data types we expose in our input file. For the messages described below we identify each value explicitly (i.e., ‘Message Type’) or as one of these types with an attributed length.

Type	**Size**	**Notes**
Numeric	1, 2, 4, or 8 Bytes	Unsigned little-endian binary encoded integer.
Alpha	Variable	Left justified, padded on the right with spaces.
Price	4 Bytes	Signed little-endian binary encoded integer. Fixed 4 decimal places, so the number must be divided by 100 for cents or 10000 for dollars. E.g., 318800 represents 3188 cents or 31.88 dollars. Assume that prices can be negative, though for very few bespoke cases. You may not observe them in stream.

Header

Each message comes with a header that defines a sequence number and message length to assist in processing. Each header will be of the form:

Field	**Value**	**Length**	**Notes**
Sequence No.	Number	4	Message counter to distinguish one message from another.
Message Size	Number	4	Message size in bytes, excluding this header.

Order Added

This message informs that an order has been added to the book. It is identified by the leading ‘A’ for message type as shown below.

Field	Value	Length	Description
Message Type	“A”	1	Order Added Message.
Symbol	Alpha	3	Uniquely identifies the symbol.
Order Id	Numeric	8	Identifies an order. Unique to a symbol and side (i.e., bid or ask).
Side	Alpha	1	Side that the order exists on. ‘B’ = Buy Order (a.k.a., Bid) ‘S’ = Sell Order (a.k.a., Ask)
Reserved	Alpha	3	Padding, all spaces.
Size	Numeric	8	Volume of the new order. Note that some orders can start at zero and be updated to non-zero.
Price	Price	4	Price of the order, see Data Types for more detail.
Reserved	Alpha	4	Padding, all spaces.

Order Updated

This message informs that an order has been updated to a new volume and price. Expect that one update message can change volume and price at the same time. It is identified by the leading ‘U’ for message type as shown below.

Field	**Value**	**Length**	**Description**
Message Type	“U”	1	Order Updated Message
Symbol	Alpha	3	Uniquely identifies the symbol.
Order Id	Numeric	8	Identifies an order. Unique to a symbol and side (i.e., bid or ask).
Side	Alpha	1	Side that the order exists on. ‘B’ = Buy Order (a.k.a., Bid) ‘S’ = Sell Order (a.k.a., Ask)
Reserved	Alpha	3	Padding, all spaces.
Size	Numeric	8	New volume of the order.
Price	Price	4	New price of the order. See Data Types for more detail.
Reserved	Alpha	4	Padding, all spaces.

Order Deleted

This message informs that an order has been deleted from the book. It is identified by the leading ‘D’ for message type as shown below. This means any remaining volume for the order is assumed to be removed from the book.

Field	**Value**	**Length**	**Description**
Message Type	“D”	1	Order Deleted Message
Symbol	Alpha	3	Uniquely identifies the symbol.
Order Id	Numeric	8	Identifies an order. Unique to a symbol and side (i.e., bid or ask).
Side	Alpha	1	Side that the order exists on. ‘B’ = Buy Order (a.k.a., Bid) ‘S’ = Sell Order (a.k.a., Ask)
Reserved	Alpha	3	Padding, all spaces.

Order Executed

This message informs that an order has traded, either partially or fully. It is identified by the leading ‘E’ for message type as shown below. When an order is executed the volume represents the amount traded and there is no corresponding delete message. If the order is fully traded it is assumed to be removed from the book.

Field	**Value**	**Length**	**Description**
Message Type	“E”	1	Order Executed Message
Symbol	Alpha	3	Uniquely identifies the symbol.
Order Id	Numeric	8	Identifies an order. Unique to a symbol and side (i.e., bid or ask).
Side	Alpha	1	Side that the order exists on. ‘B’ = Buy Order (a.k.a., Bid) ‘S’ = Sell Order (a.k.a., Ask)
Reserved	Alpha	3	Padding, all spaces.
Traded Quantity	Numeric	8	Amount traded, could be part or all of order volume.