

[Sell/Buy Order MessageBook \rightarrow OrderBook] program (Developed for WTL Capital Pte. Ltd. interview) ©2017

Sumanta Bose
sumanta001@e.ntu.edu.sg

July 26, 2017

1 Problem Statement

Given a sequence of messages, construct an in-memory representation of the current state of the order book. An example MessageBook is of the form:

```
A,100600,S,1,1081
A,100601,B,9,1006
A,100602,B,30,981
A,100603,S,10,1056
A,100604,B,10,950
A,100605,S,2,1031
A,100606,B,1,1006
X,100604,B,10,950
A,100607,S,5,1031
```

with messages in the format **action, orderID, side, quantity, price** (e.g., A,123,B,9,1006), where

- action = A (add), X (remove), M (modify)
- orderID = unique 64 bit unsigned long long to identify each order used to reference existing orders for remove/modify
- side = B (buy), S (sell)
- quantity = positive integer indicating maximum quantity to buy/sell
- price = integer indicating max price at which to buy/min price to sell

After every message, write the state of order book (best sell price and best buy price) in this format:
[price,S,order qty of first sell order, order qty of second sell order, ..]

[price,B,order qty of first buy order, order qty of second buy order, ..]

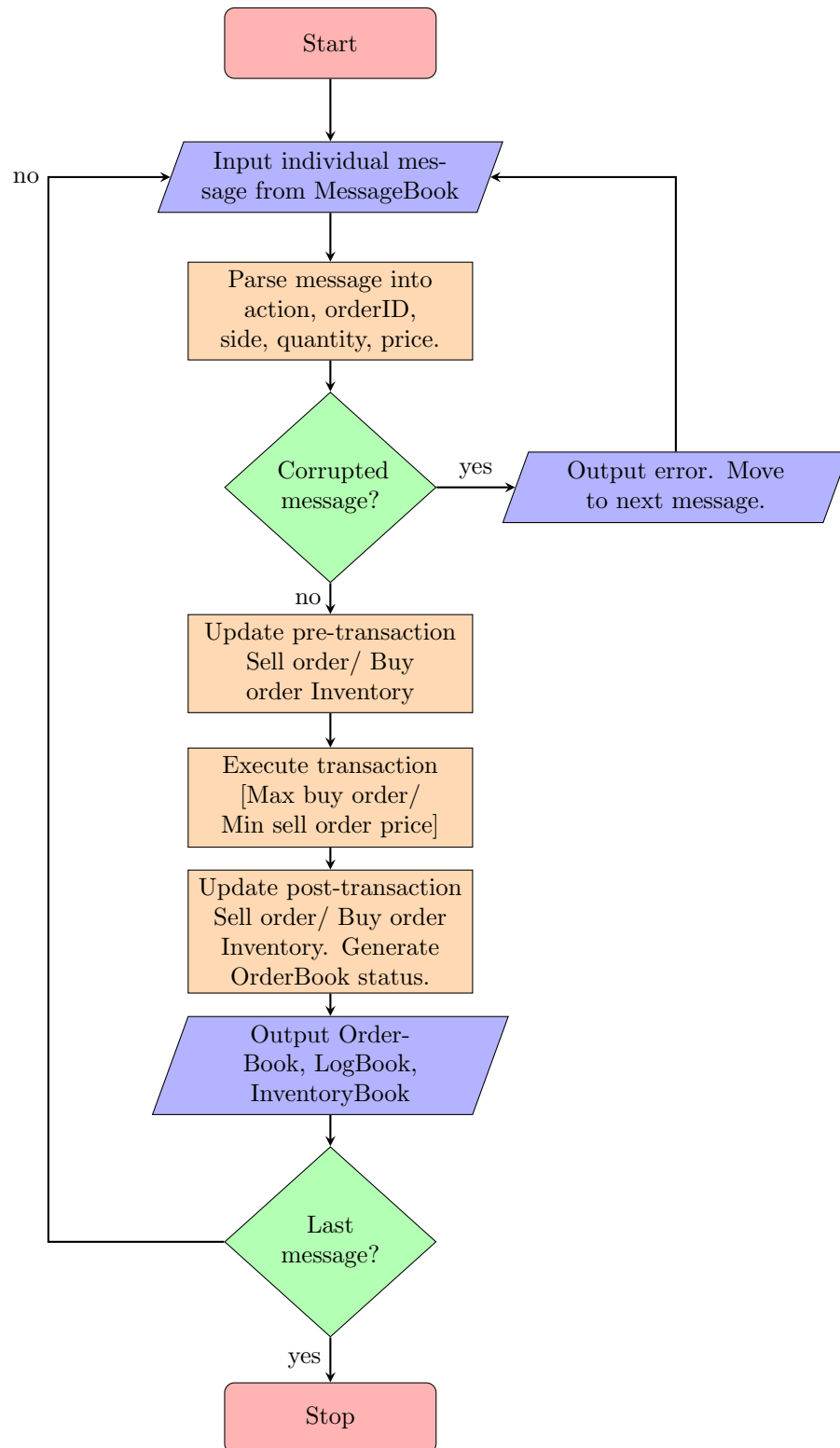
The previous example above would yield the output below:

```
[1081,S,1] [NAN]
[1081,S,1] [1006,B,9]
[1081,S,1] [1006,B,9]
[1056,S,10] [1006,B,9]
[1056,S,10] [1006,B,9]
[1031,S,2] [1006,B,9]
[1031,S,2] [1006,B,9,1]
[1031,S,2] [1006,B,9,1]
[1031,S,2,5] [1006,B,9,1]
[1031,S,5] [1006,B,9,1]
[1031,S,4] [1006,B,9,1]
```

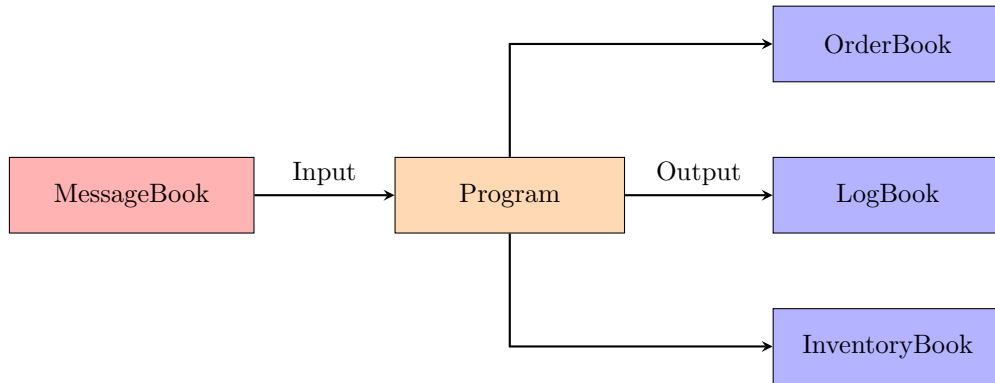
2 Approach and Solution

2.1 Algorithmic Flowchart

The flowchart below describes broadly the algorithmic approach adopted to solve the problem.



2.2 Input and Output Files



We have one input file (MessageBook) and three output files (OrderBook, LogBook and InventoryBook) generated from the program's execution.

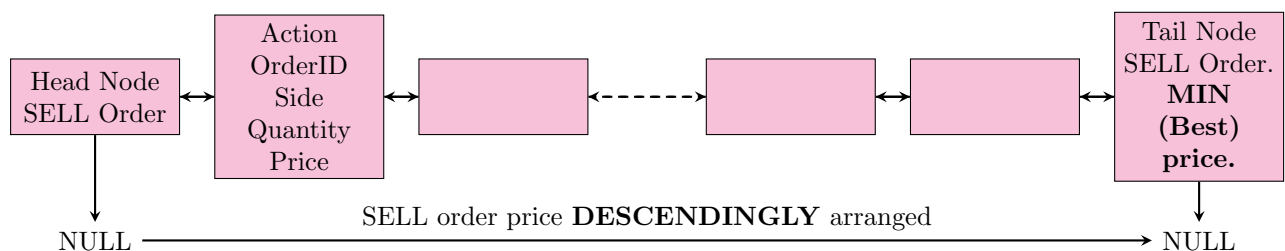
- MessageBook : Contains input message feed. A mix of sell and buy orders.
- OrderBook: Contains updated OrderBook entry after every transaction is done.
- LogBook: Contains log record of every incoming message, status of OrderBook before and after the transaction, any profit incurred and remarks in case of any message errors.
- InventoryBook: Contains a final inventory of all the un-bought sell order and un-sold buy orders after all messages are processed.

3 C++ Implementation

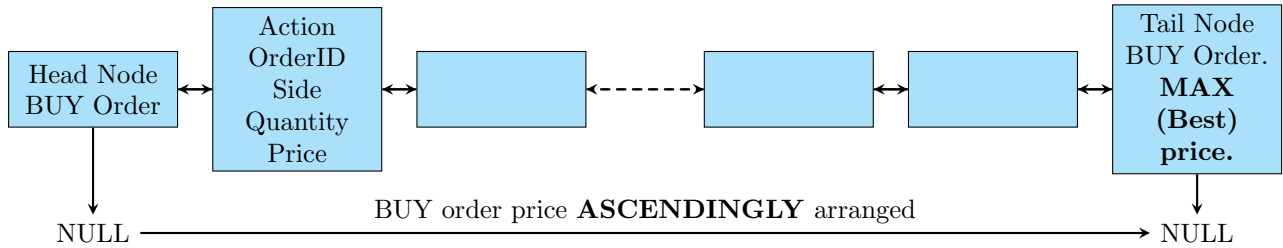
3.1 Framework

'Sell orders' and 'Buy orders' from the MessageBook are used to build their inventory which is implemented as a **bidirectional linked list** in C++. The inventory is considered as an **object** of a **class** with **private members** being the message attributes such as 'action', 'orderID', etc, and **public functions** being the actions, such a reading the MessageBook, updating inventory, executing transactions, creating OrderBook, LogBook and InventoryBook outputs, etc.

The following illustration shows the **bidirectional linked list** of sell orders, which are arranged in **descending** order of price from **Head node** to **Tail node**. When a new sell order comes in, the list is traversed using **pointers** and the order node inserted in the appropriate location. In case of order with same price, time stamp priority is used.



Similar to the **bidirectional linked list** of sell orders, the following illustration shows that of the buy orders, which are arranged in **ascending** order of price from **Head node** to **Tail node**. When a new buy order comes in, the list is traversed using **pointers** and the order node inserted in the appropriate location. In case of order with same price, time stamp priority is used.



- **Tail node** contains the **BEST** prices for transaction. For sell order, it has the **MIN** price and for buy orders it has the **MAX** price.
- ALL transactions begin at the **Tail node**, and subsequently moves up towards the **Head node** as the transactions proceeds.
- After a transaction is complete, the involved nodes gets deleted (if all quantities are over), or updated (if any quantity is left over).
- Transaction will continue to happen if the **MAX** price of buy orders is more or equal to the **MIN** price of sell orders.
- Transaction will stop when the **MAX** price of buy orders is less than the **MIN** price of sell orders.

3.2 Dependencies

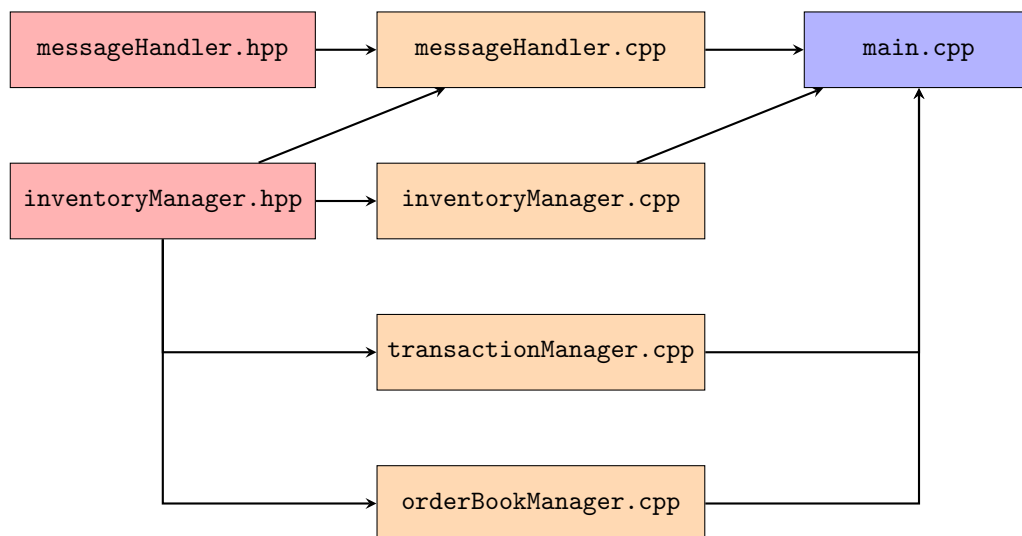
We have two header files and their roles are:

- **messageHandler.hpp** : Contains declarations of data structures and functions used to read and process encapsulated messages
- **inventoryManager.hpp** : Contains declarations of data structures and functions used to deal with bidirectional linked list to store sell orders and buy orders.

The various data structures and functions used in the program are defined in four C++ files:

- **messageHandler.cpp** : Contains functions to read and quality-test the messages.
- **inventoryManager.cpp** : Contains functions to build, modify and manage inventory, and update the InventoryBook (output).
- **transactionManager.cpp** : Contains functions to execute transaction and update the inventory.
- **orderBookManager.cpp** : Contains functions to extract the best sell order and buy order prices from the inventory to update the orderBook and LogBook (output).

Finally the dependency of **main.cpp** (the main code) on the two header files and four C++ files are described in the node chart below.



3.3 Illustration

Consider the following MessageBook as an example input to the program.

```
A,1000,S,1,120
A,1001,S,4,115
A,1002,B,3,108
A,1003,S,5,105
A,1004,S,6,105
A,1005,B,5,110
A,1006,B,2,113
A,1007,B,6,118
```

The program produces the following LogBook output. There are six columns. They are:

- Message serial number
- Message
- OrderBook status after receiving message but **before** the transaction
- OrderBook status **after** completing the sell/buy order transaction, shown in red. **This is same as the *OrderBook* output.**
- Profit made from the transaction
- Remark (this is used in case of error in message, as we shall subsequently see).

Msg#	Message	OB before trnx	OB after trnx	Trnx profit	Remark
#1	A,1000,S,1,120	[120,S,1] [NAN]	[120,S,1] [NAN]	0	Okay done.
#2	A,1001,S,4,115	[115,S,4] [NAN]	[115,S,4] [NAN]	0	Okay done.
#3	A,1002,B,3,108	[115,S,4] [108,B,3]	[115,S,4] [108,B,3]	0	Okay done.
#4	A,1003,S,5,105	[105,S,5] [108,B,3]	[105,S,2] [NAN]	9	Okay done.
#5	A,1004,S,6,105	[105,S,2,6] [NAN]	[105,S,2,6] [NAN]	0	Okay done.
#6	A,1005,B,5,110	[105,S,2,6] [110,B,5]	[105,S,3] [NAN]	25	Okay done.
#7	A,1006,B,2,113	[105,S,3] [113,B,2]	[105,S,1] [NAN]	16	Okay done.
#8	A,1007,B,6,118	[105,S,1] [118,B,6]	[120,S,1] [118,B,1]	25	Okay done.
TOTAL PROFIT = 75					

And the program produces the following InventoryBook output. At the end of all transactions one sell order at price 120 and one buy order at price 118 is left in the inventory.

Final inventory of un-bought sell orders at end of all transactions:
(Lowest sell price first)

Sell Order 1	OrderID = 1000	Quantity = 1	Price = 120
--------------	----------------	--------------	-------------

Final inventory of un-sold buy orders at end of all transactions:
(Highest buy price first)

Buy Node 1	OrderID = 1007	Quantity = 1	Price = 118
------------	----------------	--------------	-------------

3.4 Exception Handling

Now, consider the following MessageBook as an example input to the program. **This contains several corrupted messages.**

```
A,1001,S,10,100
G,1002,S,12,110
A,1003,H,15,120
A,1004,B,-5,130
X,1005,B,10,-20
A,SALT,S,10,140
M,1006,S,HI,140
A,1007,B,25,BOY
A,SALT,S,40,BOY
M,B002,S,-2,150
A,2001,S,10,100,R
A,2002,S,10,100,R,S,T
Q,2003,S,10,-50,F1
A,5001,S,50,500
A,5001,B,60,600
M,5002,S,80,800
A,8001,S,10,100
X,8002,S,10,100
X,8001,B,10,100
X,8001,S,20,100
X,8001,S,10,200
```

The program identifies the corrupted messages and in the LogBook outputs the following remarks for each message.

Msg#	Message	Remark
#1	A,1001,S,10,100	Okay done.
#2	G,1002,S,12,110	Invalid Action. Message ignored.
#3	A,1003,H,15,120	Invalid Side. Message ignored.
#4	A,1004,B,-5,130	Invalid Quantity. Message ignored.
#5	X,1005,B,10,-20	Invalid Price. Message ignored.
#6	A,SALT,S,10,140	Invalid Order ID. Message ignored.
#7	M,1006,S,HI,140	Invalid Quantity. Message ignored.
#8	A,1007,B,25,BOY	Invalid Price. Message ignored.
#9	A,SALT,S,40,BOY	Invalid Order ID. Invalid Price. Message ignored.
#10	M,B002,S,-2,150	Invalid Order ID. Invalid Quantity. Message ignored.
#11	A,2001,S,10,100,R	Invalid message length. Message ignored.
#12	A,2002,S,10,100,R,S,T	Invalid message length. Message ignored.
#13	Q,2003,S,10,-50,F1	Invalid Action & Price. Invalid message length. Message ignored.
#14	A,5001,S,50,500	Okay done.
#15	A,5001,B,60,600	Duplicate Order ID. Message ignored.
#16	M,5002,S,80,800	Modify Error: Order ID or Side mismatch.
#17	A,8001,S,10,100	Okay done.
#18	X,8002,S,10,100	Delete Error: Order ID or Side or Quantity or Price mismatch.
#19	X,8001,B,10,100	Delete Error: Order ID or Side or Quantity or Price mismatch.
#20	X,8001,S,20,100	Delete Error: Order ID or Side or Quantity or Price mismatch.
#21	X,8001,S,10,200	Delete Error: Order ID or Side or Quantity or Price mismatch.


4 Runtime Environment

4.1 Files and Directories

ls command in our program root directory will show the following.

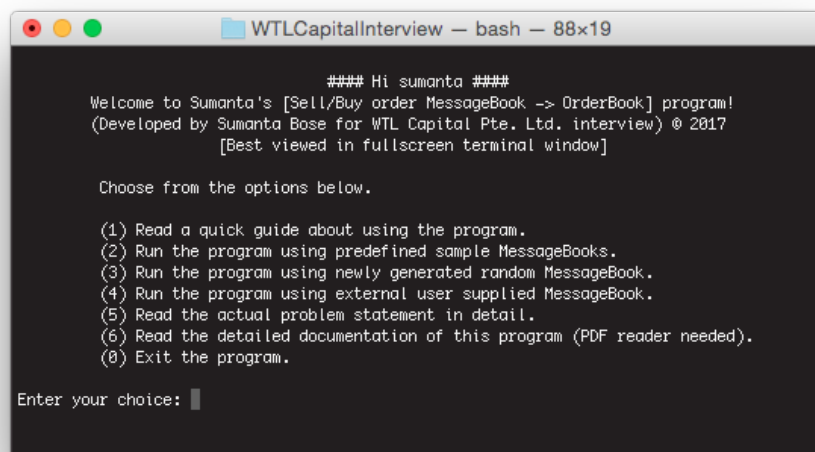
```
run.sh
ReadMe.pdf
doc/
lib/
results/
sample/
src/
```

It has two files and five directories. They are:

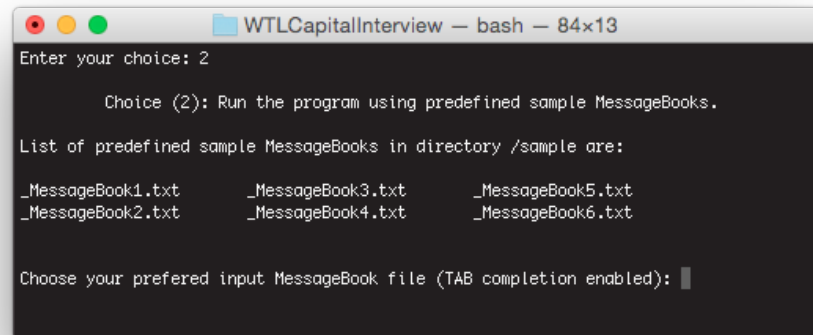
- **run.sh**: This will invoke the user interface of our program. To execute our program run the following command in the terminal.

- **ReadMe.pdf**: This document that you're currently reading.
- **doc/** : Contains data files essential to the program. Do not delete or modify this.
- **lib/** : Contains program and script files essential to the program. Do not delete or modify this.
- **results/** : Contains/ Used to save projects using the program.
- **sample/** : Contains sample input MessageBook for users.
- **src/** : Contains the C++ header files, source code and Makefile used to run the program.

4.2 User Interface

When `bash run.sh` is executed in the terminal, the following user interface opens up. It provides the user with six options as shown.



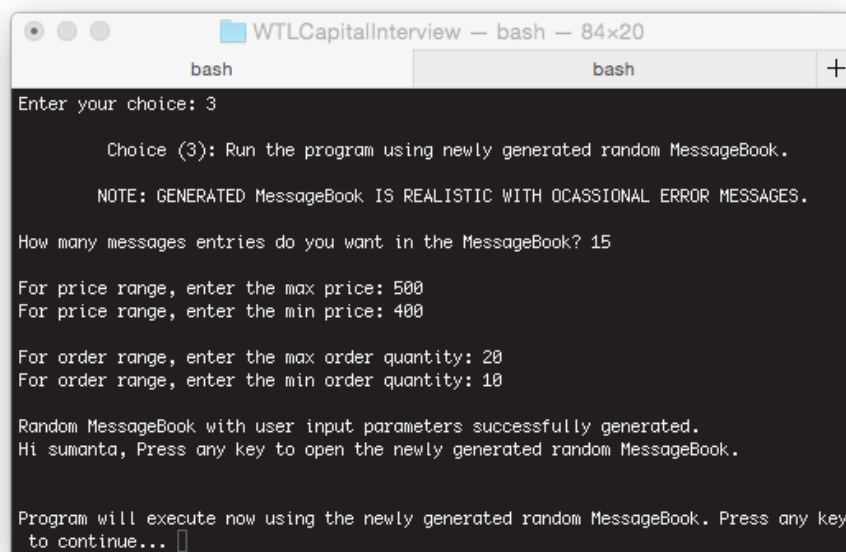
Choice (2) provides the user with six predefined MessageBook for running the program. The interface for choice (2) is shown below.

A terminal window titled "WTLCapitalInterview - bash - 84x13". The prompt is "Enter your choice: 2". The output shows "Choice (2): Run the program using predefined sample MessageBooks." followed by "List of predefined sample MessageBooks in directory /sample are:". Below this, six files are listed in two rows: "_MessageBook1.txt", "_MessageBook3.txt", "_MessageBook5.txt" in the first row, and "_MessageBook2.txt", "_MessageBook4.txt", "_MessageBook6.txt" in the second row. At the bottom, it says "Choose your preferred input MessageBook file (TAB completion enabled):" with a cursor.

Choice (3) allows the user to create a randomly generated custom MessageBook. The user can choose:

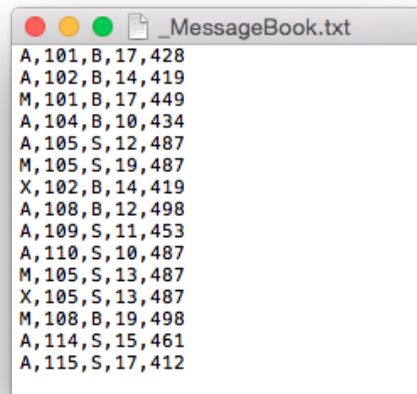
- Size of MessageBook i.e number of messages
- Maximum and minimum range of the sell/buy order prices
- Maximum and minimum range of the sell/buy order quantities

The program will automatically generate a random MessageBook according to the user's inputs.¹ The interface for choice (3) is shown below.

A terminal window titled "WTLCapitalInterview - bash - 84x20" with two tabs labeled "bash". The prompt is "Enter your choice: 3". The output shows "Choice (3): Run the program using newly generated random MessageBook." followed by "NOTE: GENERATED MessageBook IS REALISTIC WITH OCCASSIONAL ERROR MESSAGES." Then it asks "How many messages entries do you want in the MessageBook? 15". Next, it asks for price range: "For price range, enter the max price: 500" and "For price range, enter the min price: 400". Then it asks for order range: "For order range, enter the max order quantity: 20" and "For order range, enter the min order quantity: 10". The output then says "Random MessageBook with user input parameters successfully generated." and "Hi sumanta, Press any key to open the newly generated random MessageBook." At the bottom, it says "Program will execute now using the newly generated random MessageBook. Press any key to continue..." with a cursor.

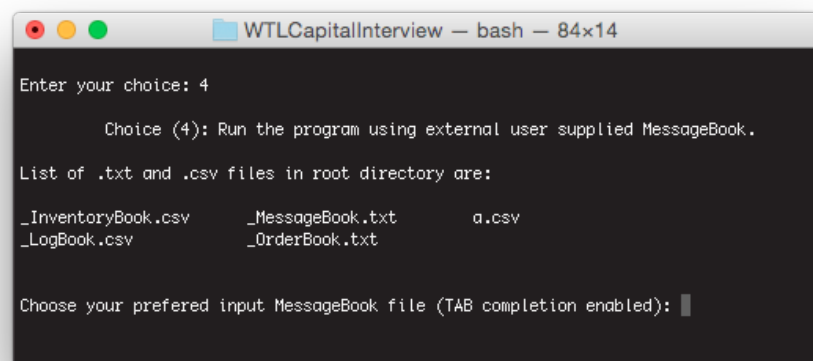
¹Note: Sell and Buy have 50% probability each. Add, Modify and Delete actions have 60%, 20% and 20% probability of occurrence. Also for realistic purposes, occasionally the messages have errors. This is intentional.

For a choice of the user inputs shown in the previous figure, a randomly generated MessageBook is shown below.



```
A,101,B,17,428
A,102,B,14,419
M,101,B,17,449
A,104,B,10,434
A,105,S,12,487
M,105,S,19,487
X,102,B,14,419
A,108,B,12,498
A,109,S,11,453
A,110,S,10,487
M,105,S,13,487
X,105,S,13,487
M,108,B,19,498
A,114,S,15,461
A,115,S,17,412
```

Furthermore, choice (4) allows the users to externally supply their own MessageBook. Our program will display a list of .txt and .csv files in the program directory. It can work on both .txt and .csv files. The interface for choice (4) is shown below.



```
WTLCapitalInterview - bash - 84x14
Enter your choice: 4
Choice (4): Run the program using external user supplied MessageBook.
List of .txt and .csv files in root directory are:
_InventoryBook.csv    _MessageBook.txt    a.csv
_LogBook.csv          _OrderBook.txt
Choose your preferred input MessageBook file (TAB completion enabled):
```

Finally after running the program and generating outputs, the user has the option to save all the files as a project. The saving option interface is shown below. All projects will be saved in numbered directories inside **results/**.

```
WTLCapitalInterview — bash — 84x37
Program was successfully executed.
#####
All input and output files are currently in the root directory.
A summary of the program execution is given below.

Input file:
(1) _MessageBook.txt : Contains input message feed.
                        A mix of sell and buy orders.

Output files:
(1) _OrderBook.txt : Contains updated OrderBook entry
                    after every transaction is done.

(2) _LogBook.csv : Contains Log of every incoming
                 message, status of OrderBook
                 before and after the transaction,
                 any profit incurred and remarks
                 in case of any message errors.

(3) _InventoryBook.csv : Contains a final inventory of
                        all un-bought sell orders and
                        un-sold buy orders after all
                        messages are processed.

#####

OrderBook and LogBook will open now for viewing.

Hi sumanta, Press any key to open the OrderBook and LogBook.

Would you like to save the current set of
(MessageBook, OrderBook, LogBook & InventoryBook) as a project? Choose [y/n]: y

Ok. Saving project...
The project is saved in directory results/project1.

Do you wish to (1) Go back to main menu, or (0) Exit program ? : █
```

Afterword:

Thanks for using my [Sell/Buy order MessageBook → OrderBook] program.
(Developed by Sumanta Bose for WTL Capital Pte. Ltd. interview) ©2017
sumanta001@e.ntu.edu.sg