COMP226 Assignment 1: Limit Order Book Pricer

Continuous Assessment Number	1 (of 2)
Weighting	10%
Assignment Circulated	10:00 Tuesday 19 February 2019
Deadline	12:00 Thursday 7 March 2019
Submission Mode	Electronic only http://www.csc.liv.ac.uk/cgi-bin/submit.pl
Learning Outcomes Assessed	This assignment will address the following learning outcomes: • Have an understanding of market microstructure and its impact on trading. • Understand the spectrum of computer-based trading applications and techniques, from profit-seeking trading strategies to execution algorithms. • Understand the benchmarks used to evaluate execution algorithms.
Purpose of Assessment	The goal of this assignment is to implement a limit order book pricer
Marking Criteria	Correctness (80%); Readability (20%). See the end of this document.
Submission necessary in order to satisfy module requirements	No
Late Submission Penalty	Standard UoL policy; **note that no resubmissions after the deadline will be considered**.
Expected time taken	Roughly 8 hours

Please submit a single file called x3xx.R, where x3xx is your CS username (which you use to log on to the submission system; yours might look quite different).

This file must contain a function called `pricer`, whose form and functionality is described below. The file may also contain other functions used by `pricer`, **but the function** `pricer` **should not be called within the file**.

Warning

Your code will be put through the department's automatic plagiarism and collusion detection system. Student's found to have plagiarized or colluded will likely receive a mark of zero. Do not discuss or show your work to other students. Several students were found to have plagiarised in previous years and this had serious consequences for their studies (two students had their studies terminated and left without a degree).

To aid with marking, your code will be put through a number of automatic tests, so you must ensure that it is written as specified below.

Learning Outcomes

The syllabus for COMP226 can be found here:

http://intranet.csc.liv.ac.uk/teaching/modules/module.php?code=COMP226

This assignment will address the following two **learning outcomes**:

- Have an understanding of market microstructure and its impact on trading.
- Understand the spectrum of computer-based trading applications and techniques, from profit-seeking trading strategies to execution algorithms.
- Understand the benchmarks used to evaluate execution algorithms.

Marking

The marking will be as follows:

- 80% for **correctness**; this will be determined primarily by automatic tests. If your code fails many of these then a mark will be decided via manual testing of your code.
- 20% for **readability** of your code; first and foremost you should ensure that your code so is correct. The next most important thing is that it can be understood by others; to aid its readability your are encouraged to include clear informative comments.

Background

In this exercise you will write an R program to analyse the log file of a limit order book market.

The log file contains messages that describe changes to the book. Each message either

- adds an order to the book, or
- reduces the size of an order in the book (possibly removing the order entirely).

Problem

The function `pricer` should take three arguments as follows

```
pricer <- function(infile, outfile, targetsize)
```

`pricer` should write its output to the file `outfile`. The file `infile` contains the messages of the limit order book log file, as described in more detail below.

As the book is modified (according to the messages, which should be processed in order), `pricer` calculates the

- total expense you would incur if you bought `targetsize` shares (by taking as many asks as necessary, lowest first), and
- the total income you would receive if you sold `targetsize` shares (by hitting as many bids as necessary, highest first).

Each time the income or expense changes, it prints the changed value to `outfile` in the format described below.

Input Format

The market data log contains one message per line (terminated by a single linefeed character, '`\n`'), and each message is a series of fields separated by spaces.

An "Add Order to Book" message looks like this:

```
timestamp 'A' order-id side price size
```

Where all fields are variables except 'A', which is a constant (examples below).

Field	Meaning
`timestamp`	The time when this message was generated by the market, as milliseconds since midnight.
`A`	A literal string identifying this as an "Add Order to Book" message.
`order-id`	A unique string that subsequent "Reduce Order" messages will use to modify this order.
`side`	A '`B`' if this is a buy order (a bid), and a '`S`' if this is a sell order (an ask).
`price`	The limit price of this order.
`size`	The size in shares of this order, when it was initially sent to the market by some stock trader.

A "Reduce Order" message looks like this:

```
timestamp 'R' order-id size
```

Where all fields are variables except 'R', which is a constant (examples below).

Field	Meaning
`timestamp`	The time when this message was generated by the market, as milliseconds since midnight.
`R`	A literal string identifying this as an "Reduce Order" message.
`order-id`	The unique string that identifies the order to be reduced.
`size`	The amount by which to reduce the size of the order. This is *not* the new size of the order. If `size` is equal to or greater than the existing size of the order, the order is removed from the book.

The log file messages are sorted by `timestamp`.

Output Format

`pricer`'s output consists of one message per line in this format:

```
timestamp action total
```

Field	Meaning
`timestamp`	The timestamp from the input message that caused this output message to be generated.
`action`	A string: '`B`' if this message contains the new expense to buy `targetsize` shares, and '`S`' if this message contains the new income for selling `targetsize` shares.
`total`	The total expense (if `action` is '`B`') to buy `targetsize` shares, or the total income (if `action` is '`S`') for selling `targetsize` shares. If the book does not contain `targetsize` shares in the appropriate type of order (asks for expense; bids for income), the `total` field contains the string '`NA`'.

If `pricer` encounters an error in an input message, it prints a warning to the R console and goes to the next message.

Example Input and Output

Here is an example run of `pricer` with a `targetsize` of 200. Input messages are in the left column. Notes and output messages are in the right column.

Standard Input	Standard Output/Notes
`28800538 A b S 44.26 100`	No output yet because neither the bids nor the asks in the book have a total of 200 shares yet.
`28800562 A c B 44.10 100`	Still not enough shares on either side of the book.
`28800744 R b 100`	This reduces order '`b`' to zero shares, which removes it from the book, so now the book contains no asks. But there's still no change to the total income or expense on 200 shares.

`28800758 A d B 44.18 157`	The bid sizes now total 257, which is more than the target size of 200. To sell 200 shares, you would first hit the bid at 44.18 for 157 shares, spending $6936.26. Then you would hit the bid at 44.10 for the remaining 43 shares, spending another $1896.30. Your total income would be $8832.56, so `pricer` emits this message: `28800758 S 8832.56`
`28800773 A e S 44.38 100`	The book now contains a single ask of size 100, which is still not enough to change the target size expense from '`NA`'.
`28800796 R d 157`	This removes bid '`d`' from the book, leaving just one bid with a size of 100 on the book, so the income from selling changes to '`NA`': `28800796 S NA`
`28800812 A f B 44.18 157`	This new bid brings the total bid size back over 200, so the selling income is no longer '`NA`': `28800812 S 8832.56`
`28800974 A g S 44.27 100`	This ask brings the total ask size up to 200, exactly the target size. The total expense for buying 200 shares would be 100 * $44.27 + 100 * $44.38: `28800974 B 8865.00`
`28800975 R e 100`	Removing ask '`e`' from the book leaves less than 200 shares on the ask side, so the buying expense changes back to '`NA`': `28800975 B NA`
`28812071 R f 100`	Reducing bid '`f`' by 100 shares leaves only 157 shares on the bid side, so the selling income changes to '`NA`': `28812071 S NA`
`28813129 A h B 43.68 50`	This new bid makes it possible to sell 200 shares: 57 at $44.18, 100 at $44.10, and the last 43 at $43.68. `28813129 S 8806.50`
`28813300 R f 57`	This removes bid '`f`' from the book, so it is no longer possible to sell 200 shares: `28813300 S NA`
`28813830 A i S 44.18 100`	This ask makes it possible to buy 200 shares again: 100 at $44.18 and 100 at $44.27. `28813830 B 8845.00`
`28814087 A j S 44.18 1000`	This ask has the same price as an existing ask, and these two asks are tied for the best asking price. This means you could now buy all 200 shares at $44.18 instead of buying half of them at $44.27, so the buying expense decreases: `28814087 B 8836.00`
`28814834 R c 100`	This leaves only 50 shares on the bid side (all in order '`h`'), so it is still not possible to sell 200 shares. The selling income is therefore unchanged from '`NA`' and `pricer` prints no output message.

`28814864 A k B 44.09 100`	Only 150 shares on the bid side, so no output needed.
`28815774 R k 100`	Back to 50 shares on the bid side; still no output needed.
`28815804 A l B 44.07 175`	There are now more than 200 shares on the bid side. You could sell 175 shares at $44.07 each, and the remaining 25 shares at $43.68 each: `28815804 S 8804.25`
`28815937 R j 1000`	After ask '`j`' is removed from the book, you can still buy 200 shares: 100 at $44.18 each, and 100 at the worse price of $44.27. `28815937 B 8845.00`
`28816245 A m S 44.22 100`	Since $44.22 is a better price than $44.27, the buying expense decreases: `28816245 B 8840.00`

Note that the book initially contains no orders, and that the buying expense and selling income are both considered to start at '`NA`'. Since `pricer` only produces output when the income or expense changes, it does not print anything until the total size of all bids or the total size of all asks meets or exceeds `targetsize`.

Test Data

You can download three sample input files here:

https://www2.csc.liv.ac.uk/~rahul/teaching/comp226/assess.html

Your program needs to work with any `targetsize`.

In case you want to test your implementation on the example input data from above, here it is in an easy-to-cut-and-paste format:

```
28800538 A b S 44.26 100
28800562 A c B 44.10 100
28800744 R b 100
28800758 A d B 44.18 157
28800773 A e S 44.38 100
28800796 R d 157
28800812 A f B 44.18 157
28800974 A g S 44.27 100
28800975 R e 100
28812071 R f 100
28813129 A h B 43.68 50
28813300 R f 57
28813830 A i S 44.18 100
28814087 A j S 44.18 1000
28814834 R c 100
28814864 A k B 44.09 100
28815774 R k 100
28815804 A l B 44.07 175
28815937 R j 1000
28816245 A m S 44.22 100
```

And here is the corresponding output:

```
28800758 S 8832.56
28800796 S NA
28800812 S 8832.56
28800974 B 8865.00
28800975 B NA
28812071 S NA
28813129 S 8806.50
28813300 S NA
28813830 B 8845.00
28814087 B 8836.00
28815804 S 8804.25
28815937 B 8845.00
28816245 B 8840.00
```

Hints

We did some relevant things in the worksheet for Slides 4. **For example, we computed the average price of a market order**; the total price was obviously part of that calculation.

Some further guidance follows.

Data Input

There are many ways to read and write data in R.

`read.table` reads in a file and returns a `data.frame`. If you are reading in strings and want them to stay as strings (rather than become factors) you should add to the arguments `stringsAsFactors=FALSE`. There are variants of `read.table` that you may want to check out, like `read.csv` and `read.csv2`.

Also the data in the assignment does not all have the same number of fields per line. Thus you want to use the `fill=TRUE` argument.

An alternative to reading in the whole input file is to read it line by line. For that open a connection using `file`. For example,

```
con  <- file("FILENAME", open = "r")

 while (length(oneLine <- readLines(con, n = 1, warn = FALSE)) > 0) {
     # do stuff, for example
     print(oneLine)
 }

 close(con)
```

The line by line approach will be slow. For a very large file it may not be possible to read the whole file into memory in which case one would typically use an incremental approach but reading in chunks rather than lines (`read.table` allows that).

> ***Note***
>
> One problem you have to solve for Assignment 1 is that there are two types of message with different fields. There are different ways to approach this, and it's left to you to decide what to do.

Why don't you start right a way and load some of the sample data for Assignment 1 into R.

Data Output

There is a function `write.table`, however for Assignment 1 one you may just want to write each output line as it is produced (rather than store the output and only write it to a file in the end).

One way to write the output incrementally is with `cat`.

```
> price <- 100
> cat(file="test.out","The price is", price, "\n")
```

Now look in your working directory and you will find the file `test.out`.

```
> file.show("test.out")

The price is 100

(END)
```

For Assignment 1 you may want to append to a file, which `cat` does as follows:

```
> price <- 101
> cat(file="test.out",append=TRUE,"The price is now", price, "\n")
```

```
> file.show("test.out")

The price is 100
The price is now 101

(END)
```

By the way, `cat` will write to the console if a file is not provided to it as an argument.

One other function you will need is `format`. The best way to check that your program for Assignment 1 is correct is to run it on the sample input and compare your output with the sample output; the easiest way to do this is with a text file comparison tool such a `diff` on Linux/Unix systems. For diff tools on other platforms see

http://stackoverflow.com/questions/12625/best-diff-tool

Anyway, use these diff tools to make your output **identical** to the sample output, and `format` can help you do that. For example,

```
> prices <- c(800.5, 800.25, 800)

> cat(prices, '\n')
800.5 800.25 800

> cat(format(prices,nsmall=2),'\n')
800.50 800.25 800.00
```

Calling the function pricer

You **need to make sure** that when your source code is sourced with something like

```
source('x3xx.R')
```

that is gives no errors, **and**, will then allow the automatic tests to call pricer somewhat like what follows.

```
pricer(infile="PATH_TO_INPUT",outpfile="PATH_TO_OUTPUT",targetsize=50)
```

Do not include the call to pricer in the file you submit.

That's it. Good luck, and if you have questions please ask in the practicals or in the lectures.

MAKE SURE YOU READ THIS DOCUMENT CAREFULLY - ALMOST ALL THE INFORMATION YOU NEED TO COMPLETE THIS ASSIGNMENT IS EITHER HERE IN THIS DOCUMENT OR IN THE SLIDES OR WORKSHEETS.