To run the tests, you can call the grader directly on one test case like so:

```
$ cd path/to/grader
$ ./grader ../cs3211-a1-.../engine < tests/testcasename.in

$ # You can also test the Go engine
$ ./grader ../cs3211-a2-.../engine < tests/testcasename.in

$ # Note that if your engine deadlocks or does not flush output, the grader
will wait forever
$ # A detailed explanation of the grader output can be found at the bottom of
this README.
```

If you encounter any issues with the grader, please ask on Canvas Discussions.

Detailed grader output explanation

The grader runs in 2 phases.

Phase 1: Engine output collection

In the first phase, it sends commands to the engine and collects all output from the engine stdout until no more output is expected. The engine's stdout and stderr are echoed to the grader's stdout (with prefixes "Engine stderr" and "Engine stdout"). Note that the engine's stderr is ignored for grading purposes, and it's only here for debugging convenience. Some errors are caught in this phase, such as output parse errors, or if obvious invariants are broken (zero quantity, sending messages about an order that does not exist, incorrect buy/sell price, etc.)

If an error occurs here, the reason is printed and the grader immediately terminates.

```
$ ./grader engines/engine-A0149796E-build-original-
4b080ab19bf3255f15d3199522d43706c3ddf1ac < tests/basic-exec-reject-cancel.in
Engine stderr closed
Engine stdout: B 125 GOOG 2705 30 14930384012 14930384097
Engine stdout: E 125 126 1 2705 30 14930384209 14930384215
Engine stdout: S 126 GOOG 2705 0 14930384209 14930384229
Output thread exception: got output for an order that should not exist (it has
fully executed, or it was not sent yet)
One or more threads failed
```

In this case, order 126 is fully executed, so "S 126 ..." was unexpected.

Phase 2: Validity checking

In the second phase, we try to check if the engine's output was valid. The grader contains a single threaded order book, and as each output line is checked, the assignment's requirements are checked first, and if the output line is valid, we will update the state of the grader's order book, and then proceed to check the next output line.

Here is some sample grader output on a student's engine:

```
Checking: B 0 GOOG 2700 1 20664306272 20664306302
Checking: X 0 A 20664306386 20664306396
Checking: B 2 GOOG 2700 1 20664306273 20664306465
Checking: X 2 A 20664306510 20664306513
...
Checking: B 2214 GOOG 2700 1 20664366606 20664366610
Checking: B 2256 GOOG 2700 1 20664366617 20664366620
Checking: X 2214 A 20664366629 20664366642
Checking: B 2238 GOOG 2700 1 20664366608 20664366643
Checking: E 2238 2189 1 2700 1 20664366631 20664366645
```

If our checks fail, then the reason for failure will be printed. For example:

```
added buy to book when a matchable sell exists
```

The output that triggered the error will be the last message printed (by `checking`) before the grader terminates, so that should be the one that you focus on.

The full list of the phase 2 errors is here:

- "resting order does not exist"
  - For an execute order, the resting order id printed does not and will never exist
- "new order does not exist"
  - For an execute order, the new order id printed does not and will never exist
- "resting order not in book"
  - For an execute order, the resting order id is valid, but not in the order book (it was not added yet, or it was fully filled)
- "new order not active"
  - For an execute order, the active order is not active (already added to the book, fully filled, or not active yet)
- "matched orders for different instruments"
  - For an execute order, the instruments for the new and resting orders are different

- "resting order filled more than initial quantity"
  - For an execute order, the count was too high, and caused the resting order's quantity to drop below 0
- "accepted cancel for order not in book"
  - The engine accepted the cancel when the order was either fully filled, or not added to the book yet
- "rejected cancel for order in book"
  - The engine rejected the cancel when the order was added to the book and still resting
- "cancel order does not exist"
  - The cancel order id does not and will never exist
- "incorrect price"
  - The price printed is not correct. For buy/sell output lines, the price should match the request. For execute output lines, the price should match the resting order.
- "incorrect instrument"
  - We sent a buy/sell request for some instrument, but the instrument name printed by the engine was some other instrument
- "incorrect side"
  - We sent a buy request, but a sell output line was printed (or vice versa)
- "booking fully filled order"
  - The active order already has 0 quantity, so it should not be added to the book
- "booking inactive order"
  - An order was added to the order book

Grader testcase format

The input format for the grader is as follows:

```
# lines starting with # are ignored
# blank lines are ignored

# at the start of the testcase, specify the number of client threads to create
10
# each thread has an ID, assigned sequentially from 0

# all commands can be prefixed by thread ID
# specify IDs like:
# - single ID: 0
# - comma-separated: 0,1
# - range (incl on both ends): 0-2
# - combination: 0,2,4-7
# if not prefixed, then all threads will execute the command

# note: commands are NOT executed in lockstep
# this input sequence is parsed into separate input streams for each thread

# sync all threads (barrier)
.

# sync threads 0, 1, 7, 8 and 9
0,1,7-9 .

# thread 0 connect to server (open)
0 o

# disconnect
0 x

# sleep (ms)
0 s 1000

# send command
# same syntax as our client
# id = 123, symbol = GOOG, price 99999, count 11111
0 C 123
0 B 123 GOOG 99999 11111
0 S 123 GOOG 99999 11111
# note: because buy/sell need a unique order ID and because of the restriction
on cancel (same client)
# it does not make sense to have more than one thread execute the same command

# wait for an active order to be either fully matched (without going onto the
book) or put onto the book
0 w 123
```