Equity Order Matching

locked

Introduction

In the stock trading world, the order book is the place where all active orders (both Buy and Sell) are maintained in certain priority for the purpose of matching buy and sell orders. Orders (both Buy and Sell Sides) can be of different types, for the purposes of this challenge please consider following order types:

- `Market` : A Market order is an order to buy or sell a symbol that must be matched at best price on the other side.
  - Market Buy: Amount x, where x is current lowest selling price available in the order book.
  - Market Sell: Amount x, where x is current highest buying price available in the order book.
- `Limit` : A Limit order is an order to buy or sell a symbol that must be matched at a specified price p or better.
  - Limit Buy: Amount x, where x is selling price and $x \leq p$, eventually.
  - Limit Sell: Amount x, where x is buying price and $x \geq p$, eventually.
- `IOC` : An immediate or cancel order (IOC) is an order to buy or sell a symbol that must be matched at a specified capped price p in the next matching cycle, and any portion of the order that cannot be matched will be automatically canceled.
  - IOC Limit Buy: Amount x, where x is selling price and $x \leq p$, now.
  - IOC Limit Sell: Amount x, where x is buying price and $x \geq p$, now.

Task

In this challenge you should design a matcher engine which performs matching between buy and sell sides of the order book. The matcher receives an input stream consisting of buy and sell orders and after receiving a match command, it should attempt to match any outstanding orders at that point in time. The matcher must ensure that buy order with highest price is matched with sell order with the lowest price. In case of multiple orders have the same price on given side (buy or sell), the matcher should pick order on first come first serve based on timestamp. A buy order for a symbol X at price P can be matched with any sell order on the same symbol X at prices less than or equal to P. Thus, a new buy or sell order may be matched with one or more of outstanding sell or buy orders.

Commands Description

The input command will have the following components, also note that some of the components are optional:

- `Action` : `N` (New), `A` (Amend), `X` (Cancel), `M` (Match) or `Q` (Query).
- `OrderID` : An integer value in the range $[1,\ 2^{63} - 1]$ inclusive.
- `Timestamp` : An integer value, typically milliseconds since epoch.
- `Symbol` : Varying length string containing only alphabets.
- `OrderType` : `M` (Market), `L` (Limit) or `I` (IOC).

- `Side` : `B` (Buy) or `S` (Sell)
- `Price` : A float value, 0.00 if `OrderType` is `M` (Market). The price is given exactly to two places of decimal.
- `Quantity` : An integer value in the range $[1,\ 2^{63}-1]$ inclusive.

The input command structure depends on the `Action` component, i.e., there are following five types of input commands:

- The `N` (New) command means that a new order is being requested to be entered into the matching book. The matcher should reject the incoming order in case of duplicate order Id or invalid input fields. The command is described as:
  `N,<OrderID>,<Timestamp>,<Symbol>,<OrderType>,<Side>,<Price>,<Quantity>`

  The matcher should output one of the following:

  - `<OrderID> - Accept`
  - `<OrderID> - Reject - 303 - Invalid order details`

  For example, consider the following commands:

  ```
  N,2,0000002,XYZ,L,B,104.53,100
  N,3,0000002,XYZ,L,B,104.53,100.3
  ```

  The matcher should output the following:

  ```
  2 - Accept
  3 - Reject - 303 - Invalid order details
  ```

  Note that, 100.3 is not an integer, so order 3 is rejected.

- The `A` (Amend) command means an existing order is being requested to be updated as per details in this command. A valid amend command should have quantity and/or price amended, any other field update should result into error code.

  In case of quantity amend down, the amended order should not lose existing priority in the matching book. In all other amend requests, the priority of amended order in matching book may vary according to the latest price and timestamp. Also note that partial amends should be supported. In case of a quantity amend request on a partially executed order, it should be accepted. If the quantity in the amend request is less than or equal to the currently matched quantity, then order should be considered closed. In scenarios where order is fully matched or already cancelled, new amend requests should be rejected. The command is described as:
  `A,<OrderID>,<Timestamp>,<Symbol>,<OrderType>,<Side>,<Price>,<Quantity>`

  The matcher should output one of the following:

  - `<OrderID> - AmendAccept`
  - `<OrderID> - AmendReject - 101 - Invalid amendment details`
  - `<OrderID> - AmendReject - 404 - Order does not exist`

  For example, consider the following commands:

  ```
  A,2,0000001,XYZ,L,B,103.53,150
  A,2,0000001,XYZ,L,S,103.53,150
  ```

  The matcher should output the following:

  ```
  2 - AmendAccept
  2 - AmendReject - 101 - Invalid amendment details
  ```

Note that, we are trying to update the `OrderType` from buy to sell, so amend order `2` is rejected.

- The `X` (Cancel) command means an existing order is being requested to be canceled. Also note that partial cancels should be supported. In scenarios where order is fully matched or already cancelled, new cancel requests should be rejected. The command is described as:
`X,<OrderID>,<Timestamp>`

  The matcher should output one of the following:

  - `<OrderID> - CancelAccept`
  - `<OrderID> - CancelReject - 404 - Order does not exist`

  For example, consider the following commands:

```
X,1,0000001
X,2,0000002
X,2,0000002
```

  The matcher should output the following:

```
1 - CancelAccept
2 - CancelAccept
2 - CancelReject - 404 - Order does not exist
```

- The `M` (Match) command means that the existing orders in the matching book should be matched now. The `Symbol` is an optional parameter if specified would mean matching should be done only for the specified symbol else it should be done for all the symbols in alphabetical order. The command is described by one of the following formats:

  - `M,<Timestamp>`
  - `M,<Timestamp>,<Symbol>`

  The matching result has three components which must be separated by a pipe (`|`):

  - `Symbol` : This represents the matched symbol.
  - `MatchedBuy` : This represents the information of matched buy. It should have `<OrderID>,<OrderType>,<Buy Qty Matched>,<Buy Price Matched>` format.
  - `MatchedSell` : This represents the information of matched sell. It should have `<Sell Price Matched>,<Sell Qty Matched>,<OrderType>,<OrderID>` format.

  The matcher should output all resulting matches in the following format:
`<Symbol>|<MatchedBuy>|<MatchedSell>`

  In case of no matches, there shouldn't be any output. For example, consider the following commands:

```
N,1,0000001,ALN,L,B,60.90,100
N,11,0000002,XYZ,L,B,60.90,200
N,110,0000003,XYZ,L,S,60.90,100
N,112,0000003,XYZ,L,S,60.90,120
N,10,0000006,ALN,L,S,60.90,100
M,00010
M,00010,ALN
```

  The matcher should output the following:

```
ALN|1,L,100,60.90|60.90,100,L,10
XYZ|11,L,100,60.90|60.90,100,L,110
XYZ|11,L,100,60.90|60.90,100,L,112
```

Note that, there is no output for the match command `M,00010,ALN` because after the match command `M,00010` there are no buy or sell orders to be matched.

Also, consider the following commands:

```
N,1,0000001,ALN,L,B,60.90,100
N,11,0000002,XYZ,L,B,60.90,200
N,110,0000003,XYZ,L,S,60.90,100
N,112,0000003,XYZ,L,S,60.90,120
N,10,0000006,ALN,L,S,60.90,100
M,00010,ALN
```

The matcher should output the following:

```
ALN|1,L,100,60.90|60.90,100,L,10
```

- This `Q` (Query) is a request to print the state of matching book. `Symbol` and `Timestamp` are optional parameters. If only symbol is specified would mean state of the matching book should be printed for the given symbol. If only timestamp is specified would mean state of the matching book(s) should be printed at the given timestamp for all symbols in alphabetical order. Otherwise, state of the matching book for the given symbol and timestamp should be printed. If no optional parameters are specified, then matcher should print state of the matching book(s) as of now for all symbols in alphabetical order. The command is described by one of the following formats:
  - `Q`
  - `Q,<Symbol>`
  - `Q,<Timestamp>`
  - `Q,<Timestamp>,<Symbol>`
  - `Q,<Symbol>,<Timestamp>`

  The state of matching book should be described by the following three components which must be separated by a pipe (`|`):

  - `Symbol` : This represents the symbol processed for the state of matching book.
  - `BuyState` : This represent the buy state of the matching book. It should have `<OrderID>,<OrderType>,<Buy Qty>,<Buy Price>` format. If there are no corresponding buy orders, this field should be left blank.
  - `SellState` : This represent the sell state of the matching book. It should have `<Sell Price>,<Sell Qty>,<OrderType>,<OrderID>` format. If there are no corresponding sell orders, this field should be left blank.

  The matcher should output the top five buy and sell entries as the state of the matching book for the given (available) symbols in the following format:
  `<Symbol>|<BuyState>|<SellState>`

  In case of Market orders, price should be printed as 0.00. For example, consider the following commands:

```
N,1,0000001,ALN,L,B,60.90,100
N,13,0000002,ALN,L,B,60.90,100
N,10,0000003,ALN,L,S,60.90,100
N,12,0000004,ALN,L,S,60.90,100
N,11,0000005,ALB,L,S,60.90,100
N,14,0000006,ALB,L,S,62.90,101
```

```
N,16,0000007,ALB,L,S,63.90,102
N,18,0000008,ALB,L,S,64.90,103
N,20,0000009,ALB,L,S,65.90,104
Q,0000003
Q,ALB
Q,ALN,0000002
Q,0000002,ALN
Q
```

The matcher should print the following:

```
ALN|1,L,100,60.90|60.90,100,L,10
ALN|13,L,100,60.90|
ALB||60.90,100,L,11
ALB||62.90,101,L,14
ALB||63.90,102,L,16
ALB||64.90,103,L,18
ALB||65.90,104,L,20
ALN|1,L,100,60.90|
ALN|13,L,100,60.90|
ALN|1,L,100,60.90|
ALN|13,L,100,60.90|
ALB||60.90,100,L,11
ALB||62.90,101,L,14
ALB||63.90,102,L,16
ALB||64.90,103,L,18
ALB||65.90,104,L,20
ALN|1,L,100,60.90|60.90,100,L,10
ALN|13,L,100,60.90|60.90,100,L,12
```

Note that:

- The query command `Q,0000003` should return the first two lines of the output.
- The query command `Q,ALB` should return the next five lines of the output.
- The query command `Q,ALN,0000002` should return the next two lines of the output.
- The query command `Q,0000002,ALN` should return the next two lines of the output.
- The query command `Q` should return the last seven lines of the output.

Note that:

- If the value of `OrderType` is other than `M` (Market), `L` (Limit), or `I` (IOC), then the matcher should reject the command and output the valid rejection message.
- If the value of `Side` is other than `B` (Buy) or `S` (Sell), then the matcher should reject the command and output the valid rejection message.
- For query command, in case of partially executed order only remaining quantity should be printed.
- We might run additional test cases offline apart from the test cases setup in hackerrank.

Test Cases Distribution

- The first test case, **`Test Case #0`** is sample test case.
- The second test case, **`Test Case #1`** has `N` (New) commands only.
- The third test case, **`Test Case #3`** has `N` (New), `A` (Amend), and `M` (Match) commands only.
- The fourth test case, **`Test Case #4`** has `N` (New), `M` (Match), and `I` (IOC) `OrderType` commands only.
- Rest of the test cases test combination of different commands.

Input Format

The first line of the input contains an integer n describing the total number of commands. Each of the n subsequent line contains a

comma-separated string describing the command.

The commands are read by the code stub in the editor below and passed to `processQueries` function as parameter. You should complete the function and return the result of each command as a string array. Note that, any sort of look-ahead is not allowed and any submission utilizing look-ahead will not be considered for evaluation.

Constraints

- Price and quantity can not be negative.
- Any input command having lower timestamp than the previous command should be rejected, except query command.
- Orders that are fully matched or canceled do not further participate in matching.

Output Format

The code stub in the editor below prints the result of each query on a new line.

Sample Input 0

```
12
N,1,0000001,AB,L,B,104.53,100
N,2,0000002,AB,L,S,105.53,100
N,3,0000003,AB,L,B,104.53,90
M,0000004
N,4,0000005,AB,L,S,104.43,80
A,2,0000006,AB,L,S,104.42,100
Q
M,0000008
N,5,0000009,AB,L,S,105.53,120
X,3,0000010
N,6,0000011,XYZ,L,B,1214.82,2568
Q
```

Sample Output 0

```
1 - Accept
2 - Accept
3 - Accept
4 - Accept
2 - AmendAccept
AB|1,L,100,104.53|104.42,100,L,2
AB|3,L,90,104.53|104.43,80,L,4
AB|1,L,100,104.42|104.42,100,L,2
AB|3,L,80,104.43|104.43,80,L,4
5 - Accept
3 - CancelAccept
6 - Accept
AB||105.53,120,L,5
XYZ|6,L,2568,1214.82|
```

```c
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();


/*
 * Complete the function below.
 * Please store the size of the string array to be returned in result_size pointer. For example,
 * char a[2][6] = {"hello", "world"};
 * *result_size = 2;
 * return a;
 *
 */
char** processQueries(int queries_size, char** queries, int* result_size) {
    // Write your code here.


}


int main()
{
    FILE *f = fopen(getenv("OUTPUT_PATH"), "w");

    char* queries_size_endptr;
    char* queries_size_str = readline();
    int queries_size = strtol(queries_size_str, &queries_size_endptr, 10);

    if (queries_size_endptr == queries_size_str || *queries_size_endptr != '\0') { exit(EXIT_FAILURE); }

    char* queries[queries_size];
    for (int queries_i = 0; queries_i < queries_size; queries_i++) {
        char* queries_item = readline();

        queries[queries_i] = queries_item;
    }

    int response_size;
    char** response = processQueries(queries_size, queries, &response_size);

    for (int response_itr = 0; response_itr < response_size; response_itr++) {
        fprintf(f, "%s", response[response_itr]);

        if (response_itr != response_size - 1) {
            fprintf(f, "\n");
        }
    }

    fprintf(f, "\n");


    fclose(f);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;
    char* data = malloc(alloc_length);

    while (true) {
```

```
69          char* cursor = data + data_length;
70          char* line = fgets(cursor, alloc_length - data_length, stdin);
71
72          if (!line) { break; }
73
74          data_length += strlen(cursor);
75
76          if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') { break; }
77
78          size_t new_length = alloc_length << 1;
79          data = realloc(data, new_length);
80
81          if (!data) { break; }
82
83          alloc_length = new_length;
84      }
85
86      if (data[data_length - 1] == '\n') {
87          data[data_length - 1] = '\0';
88      }
89
90      data = realloc(data, data_length);
91
92      return data;
93  }
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code