here comes a simulation of BitCoin trading, which mimic the equity trading

A stream of orders. Each on separate line and identified by its line number, starting at one. An order message contains the following space-delimited fields:
. type (string): define the type of the order, Say, market, limit, stop,cancel,
. side (string): either "buy" or "sell", unless ignored for specific order types
. value1(integer): varies by order type
. value2(decimal): varies by order type

1. Market order
order executed immediately at the best available prices, executed to the extent that opposing order are available, and any unfilled amount is dropped.
Order format:
. type market
. side buy or sell
. value1 a positive number of BTC to trade
. value2 ignored
Example:
market buy 100 0.00
market sell 50 0.00

2. Limit order
order can only executed at specified price limit. If a limit order can not be filled in its entirety. any unfilled amount is booked for subsequent execution.
Order format
. type limit
. side buy or sell
. value1 a positive number of BTC to trade
. value2 a positive price limit at which they can be traded.
Buy orders may executed at or below this limit,
sell orders may executed at or above it.
Example:
limit buy 10 100.10
limit sell 5 110.20

3. Stop order
order that only triggers when a subsequent trade occurs at a price that satisfies a given threshold.
A sell-side stop order (aka. stop-loss) triggers when the price is at or below the threshold.
A buy-side stop order triggers when the price is at or above the threshold.
If multiple stop orders are triggered at any given time, the oldest one executed first. Once triggered. A stop order executes as a market order. Triggered stop orders must execute before any subsequent incoming orders.
Order format
. type stop
. side buy or sell
. value1 a positive number of BTC to trade
. value2 a positive threshold price at which the oder should triggered

4. Cancel order
Cancels a previous order. May cancel a partially filled order, and cancels any unfilled portion in its entirety. Canceling a non-existent, fully executed or previous canceled order is a no-op.
Order format
. type cancel
. side ignored
. value1 a number of order to cancel
. value2 ignored
Example:
cancel none 3 0.00

Example:

```
stop buy 20 90.00
stop sell 35 45.00
```

Executing order

An order is executed by matching it with one or more opposing orders, subject to following rules

. If multiple orders are available that can match a given order, they should first match by price limit.
  When matching a sell order, match against the highest buy order first;   When matching a buy order, match against the lowest sell order first;
  if multiple have the same price limit, match the oldest among them first

. Whenever an order is matched with an opposing order, a trade occurs. The amount of BTC traded is the maximum accommodated by both. The price at which they trade is the limit of the opposing order. Say, when an incoming sell limit order of 5 BTC at $99 is matched against a buy limit order of 10 BTC at $100, 5 BTC will be traded at $100 each.

. Execution stops when the oder is filled in its entirety or no further matching order are available. An order must stop executing before any other order (triggered or incoming) executed.

Note that processing and execution of orders is done in an online manner: by the time k+1 is processed, order k should have been processed and possibly executed.

Output

Every time two orders are matched. emit a single output message with the following format:

```
match <taker> <maker> <volume> <price>
```

where

. taker (integer): the number of order being executed
. maker (integer): the number of the opposing order that was matched
. volume (integer): the number of BTC traded
. price (decimal): the price that they were traded, with 2 decimals

Example input

Input

ordid	ordtyp	side	vol	pri
1	L	B	10	99.00
2	L	B	15	100.00
3	L	B	3	100.50
4	L	S	5	100.00
5	L	B	5	99.50
6	S	S	3	99.49
7	C	na	2	0.00
8	M	S	6	0.00

the final output is

	taker	maker	vol	pri
Match	4	3	3	100.50
Match	4	2	2	100.00
Match	8	5	5	99.50
Match	8	1	1	99.00
Match	6	1	3	99.00

When ordid 4 comes, it will find the highest and oldest order of Buy side, ie. ordid 3, then it will find the ordid 2 to fulfill. So the execution records as rightmost shows.

ordid	ordtyp	side	vol	pri
1	L	B	10	99.00
2	L	B	15	100.00
3	L	B	3	100.50
4	***L***	***S***	***5***	***100.00***

ordid	ordtyp	side	vol	pri
1	L	B	10	99.00
2	L	B	13	100.00
~~3~~	~~L~~	~~B~~	~~3~~	~~100.50~~
4	***L***	***S***	***5***	***100.00***

	taker	maker	vol	pri
Match	4	3	3	100.50
Match	4	2	2	100.00

here comes the ordid 6, 7, ordid 6 is a stop sell order its price is 99.49, the current market price of buy is 99.50, so ordid 6 is not triggered, 7 will cancel ordid 2,

ordid	ordtyp	side	vol	pri
1	L	B	10	99.00
2	L	B	13	100.00
5	L	B	5	99.50
6	S	S	3	99.49
7	***C***	***na***	***2***	***0.00***

ordid	ordtyp	side	vol	pri
1	L	B	10	99.00
~~2~~	~~L~~	~~B~~	~~13~~	~~100.00~~
5	L	B	5	99.50
6	S	S	3	99.49

here comes the ordid 8, which match the ordid 5, so current highest price changed from 99.50 to 99.00

ordid	ordtyp	side	vol	pri
1	L	B	10	99.00
5	L	B	5	99.50
6	S	S	3	99.49
8	***M***	***S***	***6***	***0.00***

ordid	ordtyp	side	vol	pri
1	L	B	9	99.00
~~5~~	~~L~~	~~B~~	~~5~~	~~99.50~~
6	S	S	3	99.49
8	***M***	***S***	***6***	***0.00***

	taker	maker	vol	pri
Match	4	3	3	100.50
Match	4	2	2	100.00
Match	8	5	5	99.50
Match	8	1	1	99.00

now the buy side market price is 99.00 < ordid 6’s triggered price 99.49, so ordid 6 is triggered, and turned into a market order.

ordid	ordtyp	side	vol	pri
1	L	B	9	99.00
6	***S***	***S***	***3***	***99.49***

ordid	ordtyp	side	vol	pri
1	L	B	6	99.00
~~***6***~~	~~***M***~~	~~***S***~~	~~***3***~~	~~***0.00***~~

	taker	maker	vol	pri
Match	4	3	3	100.50
Match	4	2	2	100.00
Match	8	5	5	99.50
Match	8	1	1	99.00
Match	6	1	3	99.00

Functionality points

1. when read order from csv file, the id should be able to automatically increased. - done
2. Orders collection is vector, deque, or list?   - done
3. Limit order
   For buy side, need a heap to store the limit buy, the **largest** price and **smallest** ordid(based on the same price) will be on the peak of heap.
   For sell side, need a heap to store the limit sell, the **smallest** price and **smallest** ordid(based on the same price) will be on the peak of heap.

When market price of buy side change, it needs to check the stop sell order collection, map<price, ordid>, to reap a range of stop order and turned them into market order, then, execute or drop   -completion 50%, the snippet of code is ready, can be insert

4. Market order. Because it is executed immediately, so, it will not store, instead, execute or drop

5. Stop order, should store in map<double, int> (price, ordid) when not triggered.
   when a stop sell order comes, it will check the market price of buyside, ie. the peak price of the heap limit buy, if market price is lower or equal than its threshold, it will trigger, turn to market order
   when a stop buy order comes, it will check the market price of sell side, ie. the peak price of the heap limit sell, if market price is larger or equal than its threshold, it will trigger, turn to market order
   if stop order not triggered, need a storage for it. Use map<double, int> (price, ordid) to play as a tree storage

6.

   if we have two stop sell order comes, and the current market price of buy side is 99.50, so 6 and 7 will not triggered

ordid	ordtyp	side	vol	pri
6	*S*	*S*	*3*	*99.49*
7	*S*	*S*	*4*	*99.30*

   then, the market price is dropping to 99.00, 6, 7 are triggered, and 6 will executed first, because it is older.

7. Cancel order
   stored in a set named cancel-limit-set
   if the order it cancels is a limit order. either for buy or for sell, need to go to the heap, where limit order stored, to delete the ordid.
   But consider the heap is a chunk of memory, so, the cancel order, if it is for limit order, should be store in a set cancel-limit-set.

   if the order it cancel is a market order.

3. When a (l, s, v, p) (Limit Sell order with volume vol, and price p) comes, it will go the heap of buy side,

```
while the peak price >= p
    while qty of peak <= v
        pop peak
```

```
                    if peak.ordid  not in cancel order set
                          v -= qty of peak # only do this when the order is not canceled.
```
4. When a (m, s, v) (Market Sell order with quantity v, price ignored) comes, it will go the heap of buy side.
5. A function getMarketPrice(heap, side) will return the price.
6. A cancel order can go to the order collection, to delete the items. So the order collection should be list or deque,
   and the canceled order should be removed from heap it belongs to. Heap need to be able to search via ordid

```
    {
         so when the canceld order not in the heap peak, just let it go.
         When heap pop out, we need to check if the canceled ordid is on the peak or not, if yes, pop again
    }
```
   need to maintain a cancel orders collection, it should be a set

7.