Problem: Concert Ticket Exchange

Concert Ticket Exchange

Programming challenge description:

Concert tickets can be hard to come by, especially when going through the resale market. We would like to build our own concert ticket exchange system so that we can view available tickets in real-time. This system will listen to incoming ticket orders for a new concert venue, BT Arena. We'd like to build a system that listens to concert ticket orders and provides fans with this market information in the form of a price ladder.

Concert ticket orders arrive as a stream of events that can either add an order to our market data book or delete an order from our book. We also want to be able to delete all orders at given price level.

Add orders come with the following information:

- Action - `ADD`
- OrderId - unique identifier for order
- Artist - concert performer, such as "TaylorSwift" or "Drake"
- Price - price for the order
- Quantity - positive quantity is a buy order and negative quantity is a sell order

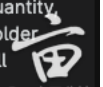
Delete orders come with the following information:

- Action - `DEL`
- Artist - concert performer, such as "TaylorSwift" or "Drake"
- OrderId - Unique identifier for the order to be deleted

Delete price levels come with the following information:

- Action - `DEL_PRICE`
- Artist - concert performer, such as "TaylorSwift" or "Drake"
- Price - price level to be deleted

We'd like to keep track of these events so that fans can request a market view in the form of a price ladder. A price ladder is made up of `n` price levels for buy orders and sell orders. A price level consists of the Price, BuyQuantity, and SellQuantity. A price level will either have a non-zero BuyQuantity or SellQuantity but not both. For buy orders, the best price is the highest price because that is the price a ticket holder would sell at. Conversely, the best price for sell orders is the lowest price. The streamed events will guarantee at least `n` price levels for both buys and sells.



输入：

Repeated number of order operations in the following format:

```
<ADD> <OrderId> <Artist> <Price> <Quantity>
<DEL> <OrderId> <Artist>
<DEL_PRICE> <Artist> <Price>
```

We can get orders for different artists in the same stream of operations. The last line of input will the artist and number of price levels that we would like to view in our ladder for both buy and sell orders.

```
<Artist> <NumberOfPriceLevels>
```

Input:
Repeated number of order operations in the following format:

`<ADD> <OrderId> <Artist> <Price> <Quantity>`
`<DEL> <OrderId> <Artist>`
`<DEL_PRICE> <Artist> <Price>`

We can get orders for different artists in the same stream of operations. The last line of input will the artist and number of price levels that we would like to view in our ladder for both buy and sell orders.

`<Artist> <NumberOfPriceLevels>`

Example Input:

```
ADD 1 TaylorSwift 100 10
ADD 2 TaylorSwift 101 -10
ADD 3 TaylorSwift 99 5
ADD 4 TaylorSwift 102 -5
ADD 5 TaylorSwift 100 2
ADD 6 Drake 95 2
DEL 1 TaylorSwift
TaylorSwift 2
```

Output:
The price ladder should begin with the artist name and should be followed by a repeated list of levels.

`<Artist>`
`<BuyQuantity> <Price> <SellQuantity>`

The levels should be sorted by price in descending order and contain `NumberOfPriceLevels` for buy and sell orders. Levels with 0 BuyQuantity and 0 SellQuantity should not be printed.

Example Output:

```
TaylorSwift
0 102 5
0 101 10
2 100 0
5 99 0
```

Test 1

测试 1

测试输入

```
ADD 1 TaylorSwift 100  10
ADD 2 TaylorSwift 101 -10
ADD 3 TaylorSwift 99   5
ADD 4 TaylorSwift 102 -5
ADD 5 TaylorSwift 100  2
DEL 1 TaylorSwift
TaylorSwift 2
```

预期输出

```
TaylorSwift
0 102 5
0 101 10
2 100 0
5 99 0
```

测试 2

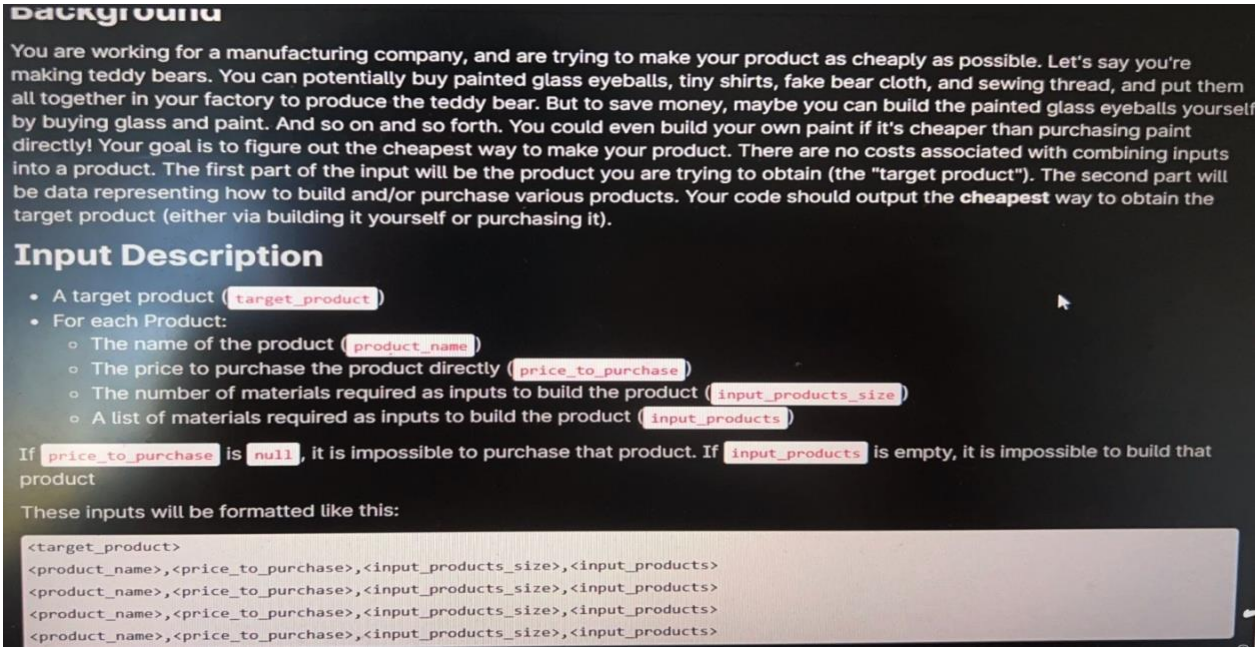
测试输入

```
ADD 1 TaylorSwift 100  10
ADD 2 TaylorSwift 101 -10
ADD 3 TaylorSwift 99   5
ADD 4 TaylorSwift 102 -5
ADD 5 TaylorSwift 98 2
ADD 6 TaylorSwift 99   5
DEL_PRICE TaylorSwift 100
TaylorSwift 1
```

预期输出

```
TaylorSwift
0 101 10
10 99 0
```

Problem statement : tedy bear wala



Background

You are working for a manufacturing company, and are trying to make your product as cheaply as possible. Let's say you're making teddy bears. You can potentially buy painted glass eyeballs, tiny shirts, fake bear cloth, and sewing thread, and put them all together in your factory to produce the teddy bear. But to save money, maybe you can build the painted glass eyeballs yourself by buying glass and paint. And so on and so forth. You could even build your own paint if it's cheaper than purchasing paint directly! Your goal is to figure out the cheapest way to make your product. There are no costs associated with combining inputs into a product. The first part of the input will be the product you are trying to obtain (the "target product"). The second part will be data representing how to build and/or purchase various products. Your code should output the **cheapest** way to obtain the target product (either via building it yourself or purchasing it).

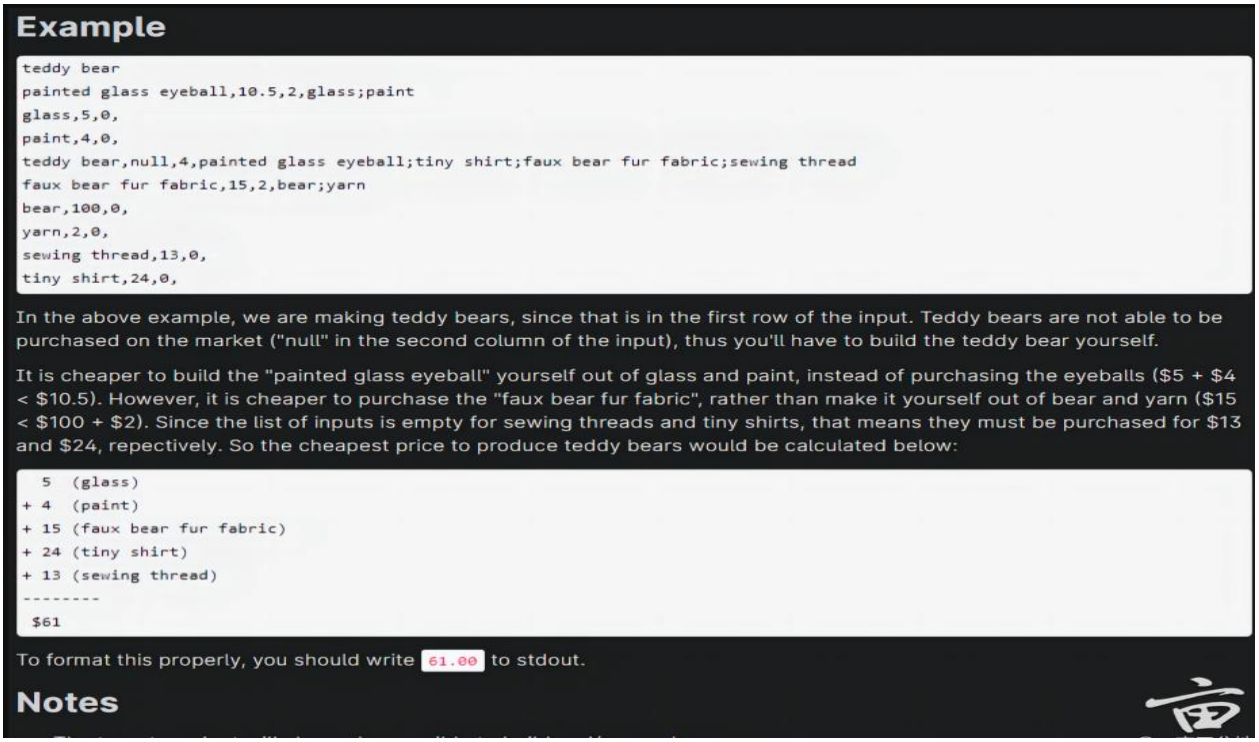
Input Description

- A target product (`target_product`)
- For each Product:
  - The name of the product (`product_name`)
  - The price to purchase the product directly (`price_to_purchase`)
  - The number of materials required as inputs to build the product (`input_products_size`)
  - A list of materials required as inputs to build the product (`input_products`)

If `price_to_purchase` is `null`, it is impossible to purchase that product. If `input_products` is empty, it is impossible to build that product

These inputs will be formatted like this:

```
<target_product>
<product_name>,<price_to_purchase>,<input_products_size>,<input_products>
<product_name>,<price_to_purchase>,<input_products_size>,<input_products>
<product_name>,<price_to_purchase>,<input_products_size>,<input_products>
<product_name>,<price_to_purchase>,<input_products_size>,<input_products>
```



Example

```
teddy bear
painted glass eyeball,10.5,2,glass;paint
glass,5,0,
paint,4,0,
teddy bear,null,4,painted glass eyeball;tiny shirt;faux bear fur fabric;sewing thread
faux bear fur fabric,15,2,bear;yarn
bear,100,0,
yarn,2,0,
sewing thread,13,0,
tiny shirt,24,0,
```

In the above example, we are making teddy bears, since that is in the first row of the input. Teddy bears are not able to be purchased on the market ("null" in the second column of the input), thus you'll have to build the teddy bear yourself.

It is cheaper to build the "painted glass eyeball" yourself out of glass and paint, instead of purchasing the eyeballs ($5 + $4 < $10.5). However, it is cheaper to purchase the "faux bear fur fabric", rather than make it yourself out of bear and yarn ($15 < $100 + $2). Since the list of inputs is empty for sewing threads and tiny shirts, that means they must be purchased for $13 and $24, repectively. So the cheapest price to produce teddy bears would be calculated below:

```
  5  (glass)
+ 4  (paint)
+ 15 (faux bear fur fabric)
+ 24 (tiny shirt)
+ 13 (sewing thread)
--------
 $61
```

To format this properly, you should write `61.00` to stdout.

Notes

Notes

- The target product will always be possible to build and/or purchase
- There are no "cycles" of inputs. For example, it would never be the case that product A is an input to product B, but product B is also an input to product A
- Some products are "raw inputs", and are unable to be produced. They must be purchased. They have an empty list of input products (in the above example, bear, yarn, sewing thread, and tiny shirt are all "raw inputs")
- Since the output is simply the cheapest price to produce the target product, it does not matter if there is a tie between the price to purchase the target product and the price to produce it
- It is possible to for a product to be an input to multiple other products
- If a product is in `input_products`, it is guaranteed to be listed as a product

输入：

- A target product
- For each Product:
  - The name of the product
  - The price to purchase the product directly
  - The number of products required as inputs to build the product
  - A list of products required as inputs to build the product

```
teddy bear
painted glass eyeball,10.5,2,glass;paint
glass,5,0,
paint,4,0,
teddy bear,null,4,painted glass eyeball;tiny shirt;faux bear fur fabric;sewing thread
faux bear fur fabric,15,2,bear;yarn
bear,100,0,
yarn,2,0,
sewing thread,13,0,
tiny shirt,24,0,
```


NOTE: You have been given skeleton code to help parse this input. You may alter this code as you wish, including not using it at all.

输出：

A number, formatted with two decimal places, that is the cheapest possible price to manufacture the target product

`61.00`

测试 1

测试输入

```
car
car,30000,5,seat;steering wheel;carpet;windshield;radio
radio,200,0,
steering wheel,null,3,leather;plastic;foam
seat,null,3,leather;foam;plastic
plastic,1300,0,
foam,7000,0,
leather,4000,0,
carpet,1000,1,plastic
windshield,5000,1,glass
glass,2000,1,sand
sand,0,0,
```

预期输出

```
25800.00
```

测试 2

测试输入

```
office chair
wheels,3.5,2,plastic;metal
cushioned seat,7.5,3,screws;padding;leather
screws,1.5,1,metal
arm rests,5,3,padding;plastic;leather
lumbar support,15,4,plastic;padding;leather;screws
plastic,2,0,
padding,3,0,
leather,4,0,
metal,1,0,
office chair,26.25,5,wheels;cushioned seat;screws;arm rests;lumbar support
```

预期输出

```
26.25
```

测试输入

```
sandwich
mayonnaise,1,0,
sand,0,0,
bread,3,3,yeast;water;flour
flour,1,0,
water,1,0,
yeast,1,0,
sandwich,10,6,mayonnaise;sand;bread;mozzarella;bacon;salt
bacon,3,1,pig
pig,1000,0,
salt,1,2,sea salt;iodine
iodine,40,0,
sea salt,.5,0,
mozzarella,3,0,
```

预期输出

```
10.00
```

测试 4

测试输入

```
teddy bear
painted glass eyeball,10.5,2,glass;paint
glass,5,0,
paint,4,0,
teddy bear,null,4,painted glass eyeball;tiny shirt;faux bear fur fabric;sewing thread
faux bear fur fabric,15,2,bear;yarn
bear,100,0,
yarn,2,0,
sewing thread,13,0,
tiny shirt,24,0,
```

预期输出

```
61.00
```

Problem: Trade Aggregrator

Trade Aggregator

Programming challenge description:

When a trader buys or sells an asset, that trade is said to have a certain amount of "edge" defined as the difference between the trade price and a theoretical value. Theoretical value is what the Trader or Firm thinks the asset is worth, and can change throughout the day.

$Edge_{buy} = TheoreticalValue - TradePrice \; Edge_{sell} = TradePrice - TheoreticalValue$

Traders typically want to execute trades that have positive edge, that is buying an asset for lower than the theoretical value, or selling an asset for higher than the theoretical value.

For example: TradePrice = 95, TheoreticalValue = 100, Edge = 100-95 = 5

As a Firm, we want to track which traders are making the best decisions, so we calculate a score for each trade.

$Score = SignOfEdge(Edge^2 * |Quantity|)$

Notice that negative edge will result in a negative score, indicating a bad trade.

Task

Given a series of trades, print the trader with the highest **cumulative** score after each trade.

Additional Details

A "Buy" trade will have positive quantity, and "Sell" trades will be denoted by negative (-) quantity.

Theoretical Values can be updated periodically. Your solution should use the most recent theoretical value when calculating edge. If an asset does not have an available theoretical value at the time of trade, assume Edge = 0 and save the trade price as the theoretical value. Theo Updates are not retroactive, so any existing trade for that asset should NOT be updated

Trades may happen with negative edge. They are valid and should be included in a traders cumulative score.

If more than one trader has the highest cumulative score, use lexicographical ordering of TraderName to break the tie.

Input:
A number trade lines in the following format:

Input:
A number trade lines in the following format:

`Trade <TraderName> <Asset> <Quantity> <Price>`

Interlaced with 1 or more theo update lines in the following format:

`TheoUpdate <Asset> <Value>`

Example:

```
TheoUpdate 1 100
Trade Alice 1 1 95
Trade Bob 1 1 94
Trade Alice 1 -1 107
```

Output:
A line of output for each trade in the input, in the format

`<TraderName> <Score>`

Example:

```
Alice 25
Bob 36
Alice 74
```

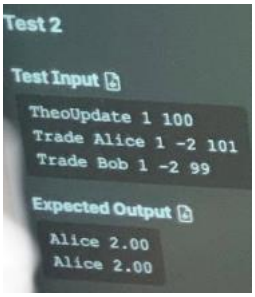
Test 1

Test Input

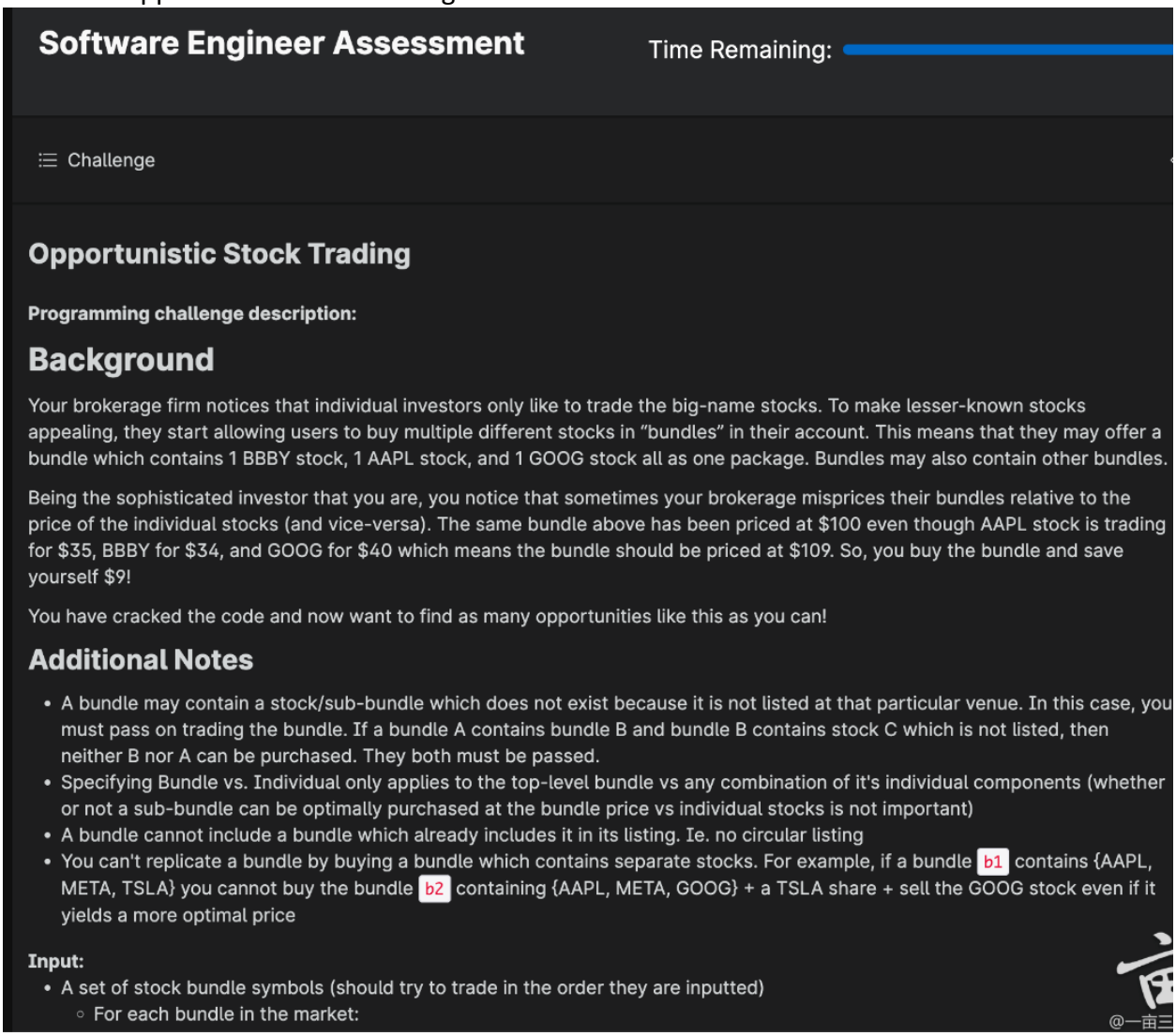
```
TheoUpdate 1 100
Trade Alice 1 1 95
Trade Bob 1 1 94
Trade Alice 1 -1 107
```

Expected Output

```
Alice 25.00
Bob 36.00
Alice 74.00
```

Problem: Opportunistic Stock Trading



Software Engineer Assessment

Time Remaining:

Challenge

Opportunistic Stock Trading

Programming challenge description:

Background

Your brokerage firm notices that individual investors only like to trade the big-name stocks. To make lesser-known stocks appealing, they start allowing users to buy multiple different stocks in "bundles" in their account. This means that they may offer a bundle which contains 1 BBBY stock, 1 AAPL stock, and 1 GOOG stock all as one package. Bundles may also contain other bundles.

Being the sophisticated investor that you are, you notice that sometimes your brokerage misprices their bundles relative to the price of the individual stocks (and vice-versa). The same bundle above has been priced at $100 even though AAPL stock is trading for $35, BBBY for $34, and GOOG for $40 which means the bundle should be priced at $109. So, you buy the bundle and save yourself $9!

You have cracked the code and now want to find as many opportunities like this as you can!

Additional Notes

- A bundle may contain a stock/sub-bundle which does not exist because it is not listed at that particular venue. In this case, you must pass on trading the bundle. If a bundle A contains bundle B and bundle B contains stock C which is not listed, then neither B nor A can be purchased. They both must be passed.
- Specifying Bundle vs. Individual only applies to the top-level bundle vs any combination of it's individual components (whether or not a sub-bundle can be optimally purchased at the bundle price vs individual stocks is not important)
- A bundle cannot include a bundle which already includes it in its listing. Ie. no circular listing
- You can't replicate a bundle by buying a bundle which contains separate stocks. For example, if a bundle `b1` contains {AAPL, META, TSLA} you cannot buy the bundle `b2` containing {AAPL, META, GOOG} + a TSLA share + sell the GOOG stock even if it yields a more optimal price

Input:

- A set of stock bundle symbols (should try to trade in the order they are inputted)
  - For each bundle in the market:

Software Engineer Assessment

Time Remaining:

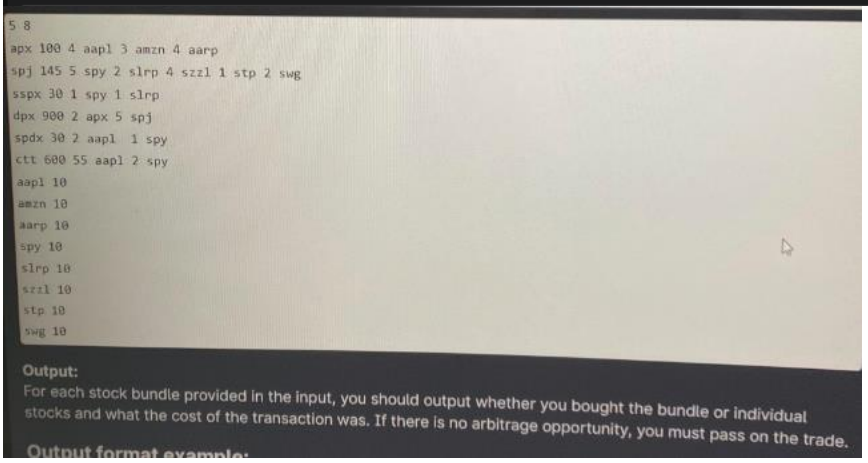
Challenge

Input:

- A set of stock bundle symbols (should try to trade in the order they are inputted)
  - For each bundle in the market:
    - The bundle symbol
    - The price of the bundle
    - The number of stocks/sub-bundles which make up the bundle
    - A list of {stock symbol, quantity} pairs which make up the bundle
  - For each individual stock in the market:
    - The product symbol
    - The price to purchase the product

Input format example:

```
<number_of_bundles_to_trade>, <number_of_stocks>
<bundle_symbol>,<price_to_purchase>,<num_stocks>,<{stock_symbol, quantity}>
<bundle_symbol>,<price_to_purchase>,<num_stocks>,<{stock_symbol, quantity}>
<stock_symbol>,<price_to_purchase>
<stock_symbol>,<price_to_purchase>
<stock_symbol>,<price_to_purchase>
```



Output:

For each stock bundle provided in the input, you should output whether you bought the bundle or individual stocks and what the cost of the transaction was. If there is no arbitrage opportunity, you must pass on the trade.

Output format example:

Output:
For each stock bundle provided in the input, you should output whether you bought the bundle or individual stocks and what the cost of the transaction was. If there is no arbitrage opportunity, you must pass on the trade.

Output format example:

```
<bundle_symbol>, <execute_or_pass>, <bundle_or_individual_stocks>, <total_price>
```

```
apx E B 100.0
spj E I 130.0
sspx E I 20.0
dpx E B 900.0
spdx P /* no arbitrage opportunity */
```

Test 1

Test Input

```
3 5
apx 99.5 3 aapl 4 amzn 3 aarp 4
spj 200.0 2 spy 3 swg 2
sspx 65 2 spy 1 aapl 1
aapl 10.5
amzn 12.0
aarp 5.0
spy 45.0
swg 33.5
```

Expected Output

```
apx E I 98.00
spj E B 200.00
sspx E I 55.50
```

Test 2

Test Input

```
2 4
adgb 5404.0 3 tadb 2 goog 2 meta 1
tadb 2520.0 2 bbby 5 hmny 4
goog 125.0
meta 114.0
bbby 225.0
hmny 350.75
```

Expected Output

```
adgb P
tadb E B 2520.00
```

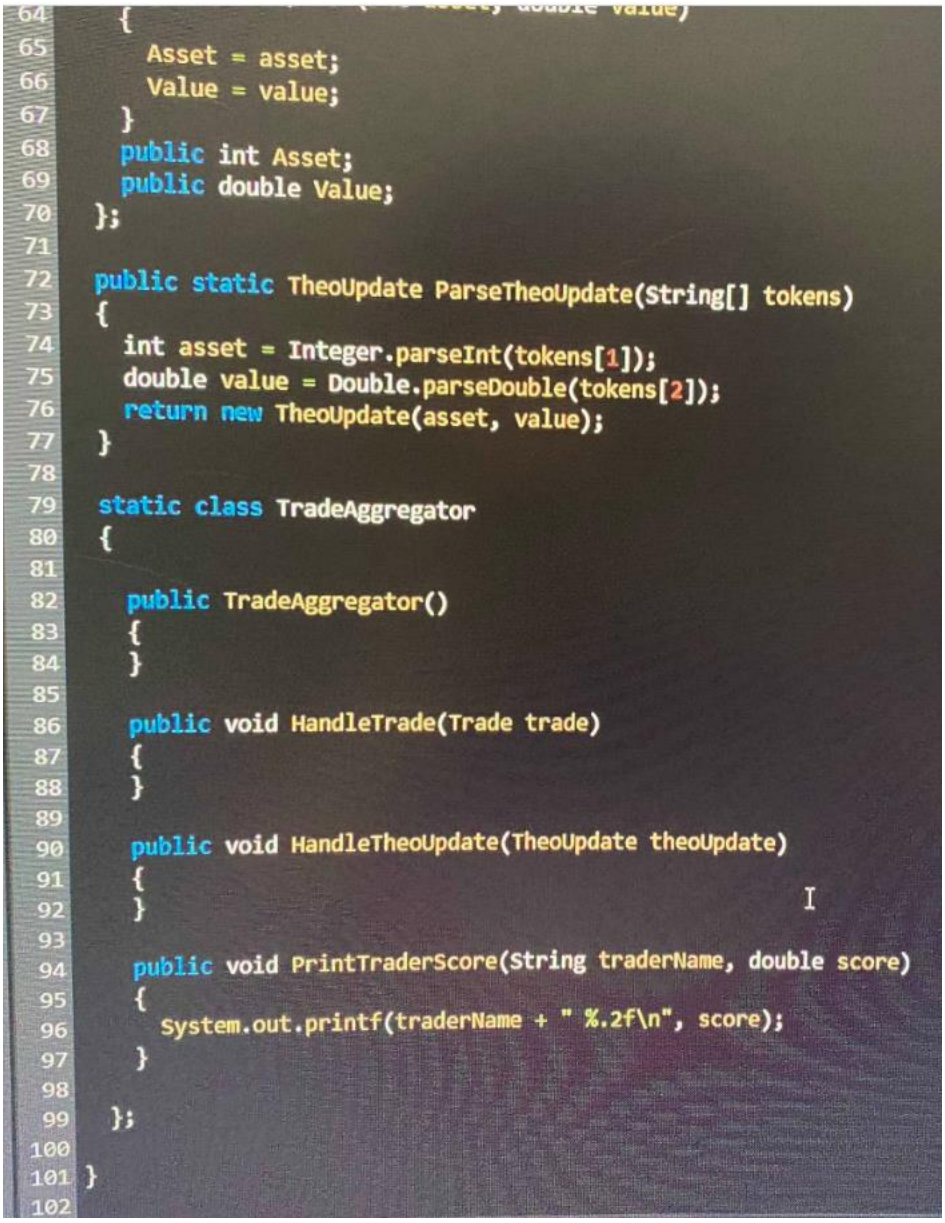
Test 3

Test Input

```
2 4
adgb 5402.0 3 tadb 2 goog 2 meta 1
tadb 2530.0 2 bbby 5 hmny 4
goog 125.0
meta 114.0
bbby 225.0
```

Expected Output

```
adgb P
tadb P
```



```
64      {                                                    double value)
65        Asset = asset;
66        Value = value;
67      }
68      public int Asset;
69      public double Value;
70    };
71
72    public static TheoUpdate ParseTheoUpdate(String[] tokens)
73    {
74      int asset = Integer.parseInt(tokens[1]);
75      double value = Double.parseDouble(tokens[2]);
76      return new TheoUpdate(asset, value);
77    }
78
79    static class TradeAggregator
80    {
81
82      public TradeAggregator()
83      {
84      }
85
86      public void HandleTrade(Trade trade)
87      {
88      }
89
90      public void HandleTheoUpdate(TheoUpdate theoUpdate)
91      {
92      }
93
94      public void PrintTraderScore(String traderName, double score)
95      {
96        System.out.printf(traderName + " %.2f\n", score);
97      }
98
99    };
100
101 }
102
```

```
35  }
36
37  static class Trade
38  {
39    public Trade(String trader, int asset, int quantity, double price)
40    {
41      Trader = trader;
42      Asset = asset;
43      Quantity = quantity;
44      Price = price;
45    }
46    public String Trader;
47    public int Asset;
48    public int Quantity;
49    public double Price;
50  };
51
52  public static Trade ParseTrade(String[] tokens)
53  {
54    String trader = tokens[1];
55    int asset = Integer.parseInt(tokens[2]);
56    int quantity = Integer.parseInt(tokens[3]);
57    double price = Double.parseDouble(tokens[4]);
58    return new Trade(trader, asset, quantity, price);
59  }
60
61  static class TheoUpdate
62  {
63    public TheoUpdate(int asset, double value)
64    {
65      Asset = asset;
66      Value = value;
67    }
68    public int Asset;
69    public double Value;
70  };
71
72  public static TheoUpdate ParseTheoUpdate(String[] tokens)
73  {
74    int asset = Integer.parseInt(tokens[1]);
75    double value = Double.parseDouble(tokens[2]);
76    return new TheoUpdate(asset, value);
```