Optiver OA

1. Risk Limits

Working at Optiver, you quickly get used to the idea of having millions of dollars worth of orders "in the market" at any given time. But that doesn't mean we aren't aware of the risks! A misbehaving trading process could lose a lot of money within a few minutes. That is why every order gets checked against a set of risk limits before being sent to the market.

Your task is to write a simple version of such a limit-checking system. The system processes orders, consisting of an instrument identifier, a timestamp (given in milliseconds since midnight), an order volume (always a positive integer) and an order price (always a positive double). You can assume that orders arrive in increasing timestamp order.

The system also processes risk limits. For every instrument, the following limits are defined:

- If the value (volume * price) of the order is above a given threshold, your program should output MAX_VAL_LIMIT followed by the instrument name, e.g. MAX_VAL_LIMIT ABC.
- If the total volume of all orders in a ten-second period exceeds a certain threshold, your program should output MAX_VOL_10S_LIMIT and the instrument name.
- If the total value of all orders in a one-second period would exceed a certain threshold, your program should output MAX_VAL_1S_LIMIT, again followed the by instrument name.

For every order, your program should print only the first limit breach it encounters. If multiple limit breaches are present in a single order, they should be checked in the order given above.

We can't place orders without having limits defined for them; if this happens, your program should output NO_LIMITS and the instrument name. Also note that limits may change throughout the day, and the latest version received should be used for checking any orders received afterwards.

Function Description
Your task is to implement a class which provides the methods *AddLimit* and *ProcessOrder*.

▸ **Input Format For Custom Testing**

▾ **Sample Case 0**

Sample Input For Custom Testing

```
LIMIT ABC 100.0 1000 100.0
ORDER ABC 10000000 10 8.0
```

2. Crab Consortium

While walking along the beach early one morning, you come across a consortium of tiny crabs. When they see you coming, all the crabs immediately stop moving, hoping that you won't notice them. As you watch you observe that once every few seconds exactly one crab will take a single surreptitious step either to the left or right.

Function Description
Complete the function *nth* that takes:

- an integer *N*
- an integer *C* which is the number of crabs - crabs are numbered from *1* to *C*;
- an integer array *S* of length *C* describing the starting horizontal position of each crab; and
- a 2-dimensional integer array *M* with two columns describing crab movements.

Each row of the matrix describes a single crab movement. The first column contains the number of the crab that moved, and the second column will either be *-1* indicating the crab moved left, or *+1* indicating the crab moved right. For example, the row `[5, -1]` indicates that crab number 5 moved to the left.

Your function should return a list of integers describing which crab was in the *N'th* position after each crab movement. The leftmost crab is in first position and the rightmost crab is in last (i.e., *C'th*) position. If two or more crabs are in the same horizontal position, higher numbered crabs come before lower numbered ones.

Constraints

- $0 \le C \le 10^5$
- $0 \le N \le C$
- $-100 \le S[i] \le 100$
- The number of crab movements (i.e., the number of rows in the 2-dimensional array *M*) will be between *0* and 10^5

▸ **Input Format For Custom Testing**

▾ **Sample Case 0**

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
1       →   N
2       →   C
2       →   S size n = 2
0       →   S = [0, 1]
1
```

Crab number 1 is now in first position, so the result for this move is 1. The final row is `[1, 1]` indicating that crab number 1 moves to the right:

```
Position: | 0 | 1 |
          |---|---|
Crabs:    |   | 2 |
          |   | 1 |
```

Both crabs are again in the same horizontal position, so crab number 2 is in first position.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
2       →   N
3       →   C
3       →   S size n = 3
-2      →   S = [-2, -1, 0]
-1
0
4       →   M[] size n = 4 (rows)
2       →   M[j][] size n = 2 (columns)
2 1     →   M = [[2, 1], [1, 1], [2, -1], [3, -1]]
1 1
2 -1
3 -1
```

Sample Output

```
3
3
1
2
```

Explanation

There are three crabs numbered 1, 2 and 3 which begin at positions `[-2, 1, 0]` :

```
Position: | -2 | -1 | 0 |
          |----|----|---|
Crabs:    |    |    | 3 |
          |    |  2 |   |
          |  1 |    |   |
```

We want to know which crab is in second position after each move. The first row of *M* is `[2, +1]` indicating that crab number 2 moves to the right:

```
Position: | -2 | -1 | 0 |
          |----|----|---|
Crabs:    |    |    | 3 |
          |    |    | 2 |
          |  1 |    |   |
```