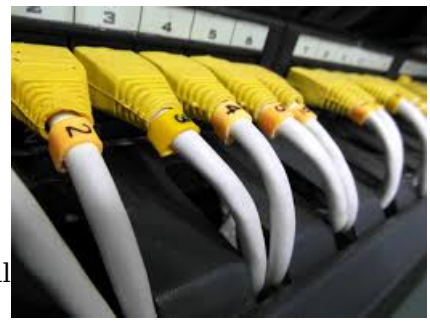
Problem A: Load Balance



The IT team of FreeWebServer, Inc. is in trouble with the load of its server, as it can not handle all the requests that it receives. In order to solve the problem they are buying a new set of servers and they want to use a fair way of dividing the work among all machines. In particular they are opting for a Weighted Round Robin load balance strategy.

The company is buying N servers, numbered from 1 to N, each one with an associated ratio R_i, indicating its capacity. Initially all servers are turned on and the first requests will go to server 1, until it gets as many requests as its ratio. When that happens, the next request will go to the server 2, which will also keep receiving requests until it meets its ratio. This pattern will go on in a round-robin fashion, with server i+1 being served right after server i, and server 1 being served after server N. Requests will arrive in groups. When a new group of requests arrives, the load balancing will continue exactly from the last server that received requests and with the ratio reflecting the amount of requests it already received. This means, for instance, that if a server i has received k requests (with $k<R_i$), in the next set of requests it will still be able to receive $R_i - k$ requests.

For energy saving purposes, a company may switch on and off any server. When a server is turned off, it will not be able to receive any requests and the round-robin scheme will skip it continuing to the next server. When a server i is turned on again, it will be able to receive requests again and its ratio will be reset to the initial value, meaning that it will again be able to receive R_i before continuing to the next server.

Can you help in simulating this strategy for several scenarios? Two detailed examples are given in the sample input and output sections.

Task

Help the IT team to simulate the behaviour of described load balancing strategy and determine how many requests each server will process during that scenario.

Input

The first line of the input identifies the number N of servers. The second line has the ratios R_i for each server (separated by a space). The third line contains the number of lines M of input for the simulation. The remaining M lines may have one of two formats: i) "S i" indicating that the server is switching its status (if it was on, if goes off, and vice versa); ii) an integer C indicating the number of requests that are made in a group of incoming requests. You can be sure that when there are requests, there will always be at least one server turned on.

Output

The output consists of exactly **N** lines, each one with the number of requests that is received by the corresponding server.

Constraints

$1 \leq N \leq 1{,}000$	Number of servers
$1 \leq R_i \leq 1{,}000$	Ratio of a server
$1 \leq M \leq 100$	Number of lines for the simulation
$1 \leq C \leq 1{,}000{,}000$	Number of calls made

Input Example 1

```
3
2 1 3
1
11
```

Output Example 1

```
4
2
5
```

Explanation of Example 1

#i represents server *i*, and [C_1, C_2, C_3] represents the already received requests per server:

Event: 11 requests arrive	
#1 gets 2 req.	[2,0,0]
#2 gets 1 req.	[2,1,0]
#3 gets 3 req.	[2,1,3]
#1 gets 2 req.	[4,1,3]
#2 gets 1 req.	[4,2,3]
#3 gets the last 3 req.	[4,2,5]

Input Example 2

```
4
4 1 5 2
13
7
4
S 4
S 3
2
S 1
S 1
9
S 1
S 1
5
S 3
7
```

Output Example 2

```
20
3
10
1
```

Explanation of Example 2

#i represents server *i*, and [C_1, C_2, C_3, C_4] represents the already received requests per server:

Event: 7 requests arrive	
#1 gets 4 req.	[4,0,0,0]
#2 gets 1 req.	[4,1,0,0]
#3 gets 2 req. (still has space for 3 more)	[4,1,2,0]
Event: 4 requests arrive	
#3 gets 3 req.	[4,1,5,0]
#4 gets 1 req. (still has space for 1 more)	[4,1,5,1]
Event: #3 and #4 are turned off	
Event: 2 requests arrive	
Current server #4 is skipped	
#1 gets the 2 req. (still has space for 2 more)	[6,1,5,1]
Event: #1 is turned off	
Event: #1 is turned on (its capacity is reset)	
Event: 9 requests arrive	
#1 gets 4 req.	[10,1,5,1]
#2 gets 1 req.	[10,2,5,1]
#3 and #4 are skipped	
#1 gets 4 req.	[14,2,5,1]
Event: #1 is turned off	
Event: #1 is turned on (its capacity is reset)	
Event: 5 request arrives	
#1 gets 4 req.	[18,2,5,1]
#2 gets 1 req.	[18,3,5,1]
Event: #3 is turned on (its capacity is reset)	
Event: 7 requests arrive	
#3 gets 5 req.	[18,3,10,1]
#4 is skipped	
#1 gets 2 req. (still has space for 2 more)	[20,3,10,1]
