Customs

Mr. Pigcow is the manager of the customs between Singapore and Yishun (which has recently seceded). A lot of Singaporeans want to get into Yishun to view the popular Point in the North so people are continuously joining the back of the queue. However, there are about 98,696 application forms to check through, so the customs officer can only process a new visitor at the rate of one a year, and as a result the queue is horrifyingly long.

Mr. Pigcow is worried that the customs officer is too ugly as they keep observing people at the back of the queue leaving the queue. They want to find out whether the looks of the customs officer is the cause of this problem. Thus, they want to know at any given time what is the number of people that can see the customs officer. A person can see the customs officer if everyone in the queue in front of him/her is shorter than him/her, or if he/she is the first person in the queue.

Not wanting to bother the existing customs officer from his work, Mr. Pigcow has asked you, a programmer with nothing better to do, to develop an algorithm to help him solve his problem.

Input

The first line of the input consists of a single integer, **Q (1 <= Q <= 100000)**, the number of queries you need to answer. It is then followed by **Q** lines, each containing a single query for you to answer. The queries follow the following specification:

Query Type	Input Format: `<QUERY_TYPE> <APPROPRIATE_PARAMETERS>`
1.	`join HEIGHT` A deluded person of height HEIGHT joins the back of the queue, in the hopes that he will be able to enter Yishun by the end of the century. Print the total number of people that can see the customs officer after this person joins the queue.
2.	`leave PEOPLE` Finally realizing that entering Yishun is practically impossible, the last PEOPLE people from the back of the queue leave. It is guaranteed that PEOPLE will be at least 1 and at most the current size of the queue. Print the total number of people that cansee the customs officer after these people leave the queue.

<Blank Space>

<End of Blank Space>

Output
Print the result of all the queries as described in the input format above. The last line of the output must contain a newline character. In the sample output below, the first line is left empty for better clarity of the sample. **No blank lines are to be printed in the actual output.**

Sample Input

```
12
join 200
leave 1
join 160
join 150
join 160
join 170
join 180
leave 1
join 165
leave 3
join 165
leave 3
```

Sample Output

```

1
0
1
1
1
2
3
2
2
1
2
0
```

Explanation
The queue after each of the queries is shown below. The shaded cells are the people who can see the front. The person with height 160 nearer to the back is blocked by the person with height 160 at the front.

`join 200`	**200**				
`leave 1`					
`join 160`	**160**				
`join 150`	**160**	150			
`join 160`	**160**	150	160		
`join 170`	**160**	150	160	**170**	
`join 180`	**160**	150	160	**170**	**180**
`leave 1`	**160**	150	160	**170**	
`join 165`	**160**	150	160	**170**	165
`leave 3`	**160**	150			
`join 165`	**160**	150	**165**		
`leave 3`					

<Blank Space>

<End of Blank Space>

Skeleton

You are given the skeleton file `Customs.java`. . You should see a non-empty file when opening it, otherwise you might be in the wrong directory. We also provide you with a simple implementation of Stack and Queue inside the skeleton file. **DO NOT modify these two classes.** You must use them as it is.

```java
/**
 * Name         :
 * Matric. No   :
 * PLab Acct.   :
 */

import java.util.*;

public class Customs {
    private void run() {
        //implement your "main" method here
    }

    public static void main(String[] args) {
        Customs newCustoms = new Customs();
        newCustoms.run();
    }
}
```

Stack<E>
+ push(E) : void
+ pop() : E
+ peek() : E
+ size() : int
+ isEmpty() : boolean

Queue<E>
+ enqueue(E) : void
+ dequeue() : E
+ peek() : E
+ size() : int
+ isEmpty() : boolean

Notes:

1. You should develop your program in the subdirectory `ex1` and use the skeleton java file provided. You should not create a new file or rename the file provided.
2. If your algorithm is different from the given skeleton, you are free to write a solution according to your own algorithm. You must use the given Stack and/or Queue in your algorithm to solve this problem. You are not allowed to use arrays, ArrayList, LinkedList, HashMap, etc. for this problem for any purpose. Failure to comply will result in 0 marks being given for the sit-in lab.
3. You are not allowed to violate the stack and/or queue properties, such as by traversing the stack and/or queue. **Use the given Stack and/or Queue and you should be fine and not fined.**
4. Please be reminded that the marking scheme is:

Input	: 10%
Output	: 10%
Correctness	: 50%
Programming Style	: 30% (awarded if you score **at least 20% from the above**):

   - Meaningful comments (pre- and post- conditions, comments inside the code): 10%
   - Modularity (modular programming, proper modifiers [public / private]): 10%
   - Proper Indentation: 5%
   - Meaningful Identifiers (for both method and variable names): 5%

   Compilation Error: Deduction of **50%** of the total marks obtained.