It is a bit complicated to describe what is “Reverse polish notation” (in short RPN). If you already knew about RPN hopefully you can recall them after looking at some examples:

Normal Form	Reverse Polish Notation
1 + 2	1 2 +
(1 + 3) * 6	1 3 + 6 *
(1 + 3) * (7 - 2)	1 2 + 7 2 - *

If you still do not remember please go through the hint section for some detailed explanation of RPN.

For this problem we will consider only one variable ‘`a`’ and only one binary operator ‘`+`’. So some valid RPN will be:

`a` which means: `a`
`aa+` which means: `a + a`
`aa+a+` which means: `(a + a) + a`
`aaa++` which means: `a + (a + a)`

And some invalid RPN can be: ‘`aaa`’, ‘`+aa`’, ‘`a+a`’, etc.

It might seem a bit confusing since all the variables are same, so you can not map which variable went where but for our problem it does not matter. We only care if the RPN is valid or not.

You will be given an RPN, you need to make it valid in minimum number of moves. In a move you can do one of the following operations:

1. Insert one a at any place you wish.
2. Insert one + at any place you wish.
3. Swap any two adjacent characters in the RPN.

You can apply the operations at any order you wish, that means, you can apply operation 3 with some newly added character by operation 1 or 2.

Input

First line of the input is number of test cases T ($1 \le T \le 3000$). Hence T lines follow each containing RPN for the case. The length of the RPN will be at least 1 and at most 100000. The RPN will be consisted of only ‘`+`’ and ‘`a`’. Sum of the lengths of all the input strings will not exceed 3000000.

Output

For each test case output ‘`Case` C`:` *ans*’ where C is the case number and *ans* is the answer for the corresponding case.

Hint:

Copied from: `http://www-stone.ch.cam.ac.uk/documentation/rrf/rpn.html`

Reverse Polish Notation is a way of expressing arithmetic expressions that avoids the use of brackets to define priorities for evaluation of operators. In ordinary notation, one might write

(3 + 5) * (7 - 2)

and the brackets tell us that we have to add 3 to 5, then subtract 2 from 7, and multiply the two results together. In RPN, the numbers and operators are listed one after another, and an operator always acts on the most recent numbers in the list. The numbers can be thought of as forming a stack, like a pile of plates. The most recent number goes on the top of the stack. An operator takes the appropriate number of arguments from the top of the stack and replaces them by the result of the operation. In this notation the above expression would be

3 5 + 7 2 - *

Reading from left to right, this is interpreted as follows:

1. Push 3 onto the stack.
2. Push 5 onto the stack. Reading from the bottom, the stack now contains (3, 5).
3. Apply the ‘+’ operation: take the top two numbers off the stack, add them together, and put the result back on the stack. The stack now contains just the number 8.
4. Push 7 onto the stack.
5. Push 2 onto the stack. It now contains (8, 7, 2).
6. Apply the ‘-’ operation: take the top two numbers off the stack, subtract the top one from the one below, and put the result back on the stack. The stack now contains (8, 5).
7. Apply the ‘*’ operation: take the top two numbers off the stack, multiply them together, and put the result back on the stack. The stack now contains just the number 40.

Sample Input

```
4
a
a+a
+aa
aa++++a
```

Sample Output

```
Case 1: 0
Case 2: 1
Case 3: 2
Case 4: 3
```