UVa Online Judge

11124 Troubles for Modern Days Problemsetters

For some programming problems it is necessary to create big input files, to be able to make sure that efficient programs will finish within the time limit, but that inefficient programs will not. This is especially troublesome as processor speeds increase, but I/O devices can't keep up: for many programming problems the execution time is substancially dependent on efficiently reading the input data. This is, of course, an undesirable situation because a contestant in a programming competition should focus on algorithm efficiency, not on hacking with low-level I/O functions, which may lack in some languages.

One way to circumvent these troubles, is to let the contestants generate their own input. To illustrate this, we take a simple problem:

> Given a list of upto 1000000 numbers, not in any particular order, determine the i-th element in the list if the list were sorted in non-descending order.

It is clear that there are several approaches to this problem, some more efficient than others, and we don't want to let the runtime be determined by the time it takes to read the list.

To make it easier for the problemsetter to work with large lists, we will define three list commands:

1. `NList`: a normal list of values. It is invoked by the command `NList`$(n, a_1, \ldots, a_n)$, where n is the number of elements, and the a-s the elements themselves.
2. `IList`: a list of incremented values. It is invoked by the command `IList`(n, s, i), where n is the number of elements, s the value of the first element, and i the increment to get the subsequent values. The increment can be positive, negative or zero.
3. `RList`: a list of random values. It is invoked by the command `RList`(n, l, h, s), where n is the number of elements, l the lowest possible generated value, h the highest possible generated value and s the starting seed for a random number generator.

The random number generator is called consecutively for all n values of the `RList`. In this problem it is very crude and defined as follows:

- The limits, l and h, are signed 32-bit integers; the seed s is an unsigned 32-bit integer.
- To generate a random number, first update the seed by multiplying it with 17, then add 11 and then take the lower 32 bits as the new seed;
- Then take the remainder of the division of the new seed by $h - l + 1$, and add that to l to get the resulting random number.

List commands can be combined to create more challenging input. This is done by placing a plus-sign ('+') between list commands. Examples:

- `NList`(4,1,2,3,4) generates the list: 1 2 3 4.
- `IList`(3,100,10) generates the list: 100 110 120.
- `RList`(5,-200,300,1234567890) generates the list: -112 200 91 228 -11.
- `IList`(5,0,1)+IList(3,6,0)+IList(5,5,-1) generates the list: 0 1 2 3 4 5 6 6 6 5 4 3 2 1 0.

Input

The input consists of several cases, each on two consecutive lines.

The first line contains the number i, the index of the number we seek in the list defined in the next line, if that list were sorted in non-descending order. The first element of the sorted list has index 1. The index will be within the bounds of the defined list.

The second line of each case defines a list of input numbers for the problem, in terms of the three given list commands (`NList`, `IList` and `RList`). If there is more than one list command on a line, then they are separated by plusses. No spaces are used and all list commands will produce lists of 32-bit signed integers. The total number of numbers generated in one line is between 1 and 1000000 inclusive. There will be no more than 32 list commands on one line, and a line is not longer than 1000 characters.

The input is terminated by the number `0` (zero).

Output

For each case in the input, state the case number (starting from 1) and the element we seek, in the format shown in the sample output below.

Nota Bene: Although the problem is "simple", you will need an efficient algorithm to solve it in time!

Sample Input

```
4
NList(10,-9,6,-4,-3,6,501,7,6,6,-10000)
13
IList(5,0,1)+IList(3,6,0)+IList(5,5,-1)
1
IList(1000000,-90,0)
123456
RList(1000000,-2000000000,2000000000,0)
200000
IList(50001,-25000,1)+RList(500000,-25000,25000,3333333333)+NList(4,0,0,0,0)
0
```

Sample Output

```
Case 1: -3
Case 2: 6
Case 3: -90
Case 4: -1735543272
Case 5: -6808
```