APL is an array programming language that uses a notation invented by Ken Iverson in 1957. In this problem we consider only a small subset of the language which we call *apl* (that is, *small APL*).

Each *apl* expression appears on a line by itself and each expression has a value, which is displayed immediately after the expression is entered. Operators in *apl* do not have precedence like those in C, C++, or Java, but instead are applied right to left. However, parentheses may be used to control evaluation order. Similarly, operands for binary operators are evaluated in right to left order. Here are some examples of *apl* expressions.

`var = 1 2 3`	Store the vector `1 2 3` in var, replacing its previous value. The value of the expression is `1 2 3`. The left operand of the `=` operator must be a variable.
`var + 4`	Display the value of `var` with 4 added to each of its elements (result: `5 6 7`); the stored version of var is not modified.
`- / var`	Display the value of `var` as if a `-` operator had been inserted between each of its elements on each row (result: `2`). If `var` has two dimensions, the result is a vector. If `var` has three dimensions, the result is a two-dimensional array. `* /` and `+ /` have analogous behaviors.
`iota 5`	Generate a vector with the values `1 2 3 4 5`.
`2 2 rho 1 2 3 4`	Reshape the vector `1 2 3 4` into a 2 by 2 array; `1` and `2` are in the first row, and `3` and `4` are in the second row.
`2 2 rho 1 2 3 4 5 6`	Same result as above.
`2 3 rho 1 2 3 4`	Another reshaping, yielding a first row with `1 2 3` and a second row with `4 1 2`; if the right argument does not have a sufficient number of elements, then the elements of the right operand are reused starting from the beginning, in row-major order.
`2 drop iota 5`	Result: `3 4 5`. Drops the two leading elements from `iota 5`.
`1 2 * 3 4`	Result: `3 8`. Illustrates element-wise multiplication. Operands must be conformable - either they have the same shape, or at least one must be a one-element vector (see second example).
`( ( a = 1 ) drop 1 2 3 ) - 5`	Result: `-3 -2`. Illustrates use of parentheses.
`a + ( a = 5 ) + a + ( a = 6 )`	Result: 22. Illustrates evaluation order

In this problem you are to write an interpreter for *apl*. Integers in the input are non-negative and less than 10^4. All computed integer values (including intermediate values) have absolute values less than 10^4. The number of entries in any matrix is always less than or equal to 10^4. Variable names consist of one to three alphabetic lowercase characters, and the names `iota`, `rho`, and `drop` are always interpreted as operators. Exactly one space separates elements of statements (constants, variables, operators, and parentheses).

Constants in the input are vectors. All intermediate values are one, two, or three-dimensional arrays with positive dimensions. This restricts some operand ranges: "`2 0 rho 1 2 3`", "`2 3 2 1 rho 5`", and "`3 drop iota 3`" are illegal. The only arithmetic operators provided are `+` (addition), `-` (subtraction), and `*` (multiplication). Their operands are conformable as illustrated in the examples. Observe that "`1 1 rho 1`" and "`1 rho 1`" have different shapes. The operand for `iota` evaluates to a one-element positive vector. The left operand of `drop` evaluates to a one-element non-negative vector and its right operand evaluates to a vector. Both operands of `rho` evaluate to vectors.

Input

The input contains several test cases, each on a line by itself. The values of variables assigned in one test case are available for use in following test cases. No expression exceeds 80 characters in length, including space characters. No test case produces an invalid result (for example, an empty vector).

The last test case is followed by a line containing the single character '`#`'.

Output

For each test case, display a line containing the case number and the input line. Then, starting on the next line, display the result of evaluating the expression. Vectors display as a single line of integers; $m \times n$ arrays display as m lines of n values, and $m \times n \times p$ arrays display as m arrays of size $n \times p$, with a blank line separating the $n \times p$ arrays. Values on the same line should be separated by white space as shown in the sample output.

Sample Input

```
var = 1 2 3
var + 4
- / var
iota 5
2 2 rho 1 2 3 4
2 3 rho 1 2 3 4
2 drop iota 4
1 2 * 3 4
( ( a = 1 ) drop 1 2 3 ) - 5
a + ( a = 5 ) + a + ( a = 6 )
( 2 2 rho 2 drop iota 6 ) + 100
1 2 3 + 4 5 6
2 3 rho 1 2 3 4 5 + 1 2 3 4 5
+ / 2 3 4 rho iota 2 * 3 * 4
( 2 4 5 rho iota 2 * 4 * 5 ) - 99
#
```

Sample Output

```
Case 1: var = 1 2 3
1 2 3
Case 2: var + 4
5 6 7
Case 3: - / var
2
Case 4: iota 5
1 2 3 4 5
Case 5: 2 2 rho 1 2 3 4
1 2
3 4
Case 6: 2 3 rho 1 2 3 4
1 2 3
4 1 2
Case 7: 2 drop iota 4
3 4
Case 8: 1 2 * 3 4
3 8
Case 9: ( ( a = 1 ) drop 1 2 3 ) - 5
-3 -2
Case 10: a + ( a = 5 ) + a + ( a = 6 )
22
Case 11: ( 2 2 rho 2 drop iota 6 ) + 100
103 104
105 106
Case 12: 1 2 3 + 4 5 6
5 7 9
Case 13: 2 3 rho 1 2 3 4 5 + 1 2 3 4 5
2 4 6
8 10 2
Case 14: + / 2 3 4 rho iota 2 * 3 * 4
10 26 42
58 74 90
Case 15: ( 2 4 5 rho iota 2 * 4 * 5 ) - 99
-98 -97 -96 -95 -94
-93 -92 -91 -90 -89
-88 -87 -86 -85 -84
-83 -82 -81 -80 -79

-78 -77 -76 -75 -74
-73 -72 -71 -70 -69
-68 -67 -66 -65 -64
-63 -62 -61 -60 -59
```