7294 Interpreter

Wouldn't it be cool if your favorite language had feature XYZ? Well, your professor has started you on your journey to design your own language with your own features. Perhaps you'll even find time to include blackjack, among other things, in the language. Until then, your first class assignment is to write an interpreter for a simple programming language.

Data Types

The language has a single data type, a 32-bit signed integer.

Operators and statements that generally operate on booleans (`!`, `&&`, `||`, `if`, and `while`) treat 0 (zero) as false and all other values as true.

Operators that generally result in a boolean (`!`, `<`, `<=`, `>`, `>=`, `==`, `!=`, `&&`, and `||`) return 0 (zero) for false and 1 (one) for true.

Variables

Each variable name is a single lowercase letter between '`a`' and '`z`'. All variables are implicitly initialized to 0 (zero) at the beginning of each program.

Whitespace

Each simple statement (`set` or `print`) or part of a compound statement (`if`, `else`, `end if`, `while`, `end while`) is on its own line.

There are no blank lines.

Any amount of whitespace (space or tab) may occur before or after any token (operator, variable name, keyword, constant, etc.). It will not occur within a token. It is only guaranteed to be present between 2 alphanumeric tokens.

Statements

A statement consists of any of the following:

- If (with else):

```
if expression
      statements
else
      statements
end if
```

  Run the zero or more statements in the first block if *expression* evaluates to true (non-zero). Otherwise, run the zero or more statements in the second block.

- If (without else):

```
if expression
      statements
end if
```

  Run the zero or more statements in the block if *expression* evaluates to true (non-zero). Otherwise, skip the block.

- While:

  `while` *expression*
  statements
  `end while`

  Evaluate *expression*. If it is true (non-zero), run the zero or more statements in the block and return to the start of the while statement to re-evaluate the expression. Otherwise, skip the block.

- Assignment:

  `set` *name* = *expression*

  Set the variable named name to the result of expression.

- Output:

  `print` *expression*

  Write the result of expression to stdout on its own line.

Expressions

An expression consists of a variable name, integer value (sequence of digits with a value between 0 and $2^{31} - 1$ inclusive), or any of the operations listed below.

Some operators have a higher precedence than others; $1 + 2 * 3$ is equivalent to $1 + (2 * 3)$, though $1 * 2 + 3$ is equivalent to $(1 * 2) + 3$.

Within a single precedence level, binary operators group left-to-right, and unary operators group right-to-left. For example, $1 + 2 - 3$ is equivalent to $(1 + 2) - 3$, and $! - x$ is equivalent to $!(-x)$.

Precedence	Operator	Description
7	`()`	grouping
6	`-`	unary minus (additive inverse)
	`!`	logical negation (not)
5	`*`	multiplication
	`/`	integer division
	`%`	modulo (remainder)
4	`+`	addition
	`-`	subtraction
3	`<`	less than
	`<=`	less than or equal
	`>`	greater than
	`>=`	greater than or equal
2	`==`	equal
	`!=`	not equal
1	`&&`	logical and
0	`||`	logical or

All final and intermediate values will fit in a 32-bit signed integer; no overflow detection is necessary. Division (both `/` and `%`) by 0 will not occur.

Input

Input consists of 1 or more programs. Each begins with a line containing the number of lines N, $(1 \leq N \leq 50)$ in that program. The following N lines contain the program to run. No line is longer than 100 characters. Input is terminated when N equals 0 (zero).

Output

Output consists only of the result of the print statements in the programs as described above.

Sample Input

```
11
    set n = 22
    set s = 1
    while n &gt; 1
        if n % 2 == 0
            set n = n / 2
        else
            set n = 3 * n + 1
        end if
        set s = s + 1
    end while
    print s
1
    print ((2*4-6/3)*(3*5+8/4))-(2+3)+1/2*0
0
```

Sample Output

```
16
97
```