327 Evaluating Simple C Expressions

The task in this problem is to evaluate a sequence of simple C expressions, buy you need not know C to solve the problem! Each of the expressions will appear on a line by itself and will contain no more than 80 characters. The expressions to be evaluated will contain only simple integer variables and a limited set of operators; there will be no constants in the expressions. There are 26 variables which may appear in our simple expressions, namely those with the names `a` through `z` (lower-case letters only). At the beginning of evaluation of each expression, these 26 variables will have the integer values 1 through 26, respectively (that is, `a = 1`, `b = 2`, ..., `n = 14`, `o = 15`, ..., `z = 26`). Each variable will appear at most once in an expression, and many variables may not be used at all.

The operators that may appear in expressions include the binary (two-operand) `+` and `-`, with the usual interpretation. Thus the expression `a + c - d + b` has the value 2 (computed as `1 + 3 - 4 + 2`). The only other operators that may appear in expressions are `++` and `--`. These are unary (one-operand) operators, and may appear before or after any variable. When the `++` operator appears before a variable, that variable's value is incremented (by one) before the variable's value is used in determining the value of the entire expression. Thus the value of the expression `++c - b` is 2, with `c` being incremented to 4 prior to evaluating the entire expression. When the `++` operator appears after a variable, that variable is incremented (again, by one) after its value is used to determine the value of the entire expression. Thus the value of the expression `c++ - b` is 1, but `c` is incremented after the complete expression is evaluated; its value will still be 4. The `--` operator can also be used before or after a variable to decrement (by one) the variable; its placement before or after the variable has the same signficance as for the `++` operator. Thus the expression `--c + b--` has the value 4, with variables `c` and `b` having the values 2 and 1 following the evaluation of the expression.

Here's another, more algorithmic, approach to explaining the `++` and `--` operators. We'll consider only the `++` operator, for brevity:

1. Identify each variable that has a `++` operator before it. Write a simple assignment statement that increments the value of each such variable, and remove the `++` operator from before that variable in the expression.

2. In a similar manner, identify each variable that has a `++` operator after it. Write a simple assignment statement that increments the value of each of these, and remove the `++` operator from after that variable in the expression.

3. Now the expression has no `++` operators before or after any variables. Write the statement that evaluates the remaining expression after those statements written in step 1, and before those written in step 2.

4. Execute the statements generated in step 1, then those generated in step 3, and finally the one generated in step 2, in that order.

Using this approach, evaluating the expression `++a + b++` is equivalent to computing

- `a = a + 1` (from step 1 of the algorithm)
- `expression = a + b` (from step 3)
- `b = b + 1` (from step 2)

where expression would receive the value of the complete expression.

Input and Output

Your program is to read expressions, one per line, until the end of the file is reached. Display each expression exactly as it was read, then display the value of the entire expression, and on separate lines, the value of each variable after the expression was evaluated. Do not display the value of variables that were not used in the expression. The samples shown below illustrate the desired *exact* output format.

Blanks are to be ignored in evaluating expressions, and you are assured that ambiguous expressions like `a+++b` (ambiguous because it could be treated as `a++ + b` or `a + ++b`) will not appear in the input. Likewise, `++` or `--` operators will never appear both before and after a single variable. Thus expressions like `++a++` will not be in the input data.

Sample Input

```
a + b
b - z
a+b--+c++
c+f--+--a
   f-- + c-- + d-++e
```

Sample Output

```
Expression: a + b
    value = 3
    a = 1
    b = 2
Expression: b - z
    value = -24
    b = 2
    z = 26
Expression: a+b--+c++
    value = 6
    a = 1
    b = 1
    c = 4
Expression: c+f--+--a
    value = 9
    a = 0
    c = 3
    f = 5
Expression:    f-- + c-- + d-++e
    value = 7
    c = 2
    d = 4
    e = 6
    f = 5
```