Problem H: Peter's Calculator

Source file: calculator.c / calculator.p
Input file: calculator.in
Output file: calculator.out

Unfortunately, Peter's Calculator broke down last week. Now Peter is left with his computer, which has no calculator application, and paper and pencil, which is too tiresome for an engineer. As one of Peter's friends, you are asked to write him a calculator application. After talking to him, you figure out the following:

- Peter does only integer arithmetic. The operations he needs are addition, subtraction and multiplication.
- He would like to use an arbitrary number of variables whose names are not longer than 50 characters.
- His main way of doing calculations are to type in a few formulas and to assign them to variables. Some formulas are complicated expressions, which can refer to yet undefined variables, while other formulas consist of a single number. Then Peter asks for the value of some variables, i.e. he evaluates the formulas.
- Peters wants to redefine some variables and then to reevaluate formulas that depend on these variables.

The input strictly adheres to the following syntax (given in EBNF):

```
file = line { line } <EOF>.
line = [ assignment | print |
       reset ] <CR>.
assignment = var ":=" expression.
print = "PRINT" var.
reset = "RESET".
expression = term { addop term }.
term = factor { mulop factor }.
factor = "(" expression ")" |
       var | number.
addop = "+" | "-".
mulop = "*".
```

In the Extended Backus-Naur Formalism (EBNF), `A = B C` declares that the grammatical construct `A` consists of a `B` followed by a `C`. `A = B | C` means that `A` consists of a `B` or, alternatively, of a `C`. `A = [ B ]` defines construct `A` to be either a `B` or nothing and `A = { B }` tells you that `A` consists of the concatenation of any number of `B`s (including none).

The production `var` stands for the name of a variable, which starts with a letter followed by up to 49 letters or digits. Letters may be uppercase or lowercase. The production `number` stands for a integer number. The precise syntax for these productions are given below. The case of letters is important for both variables and statements.

```
var = letter { letter | digit }.
number = [ "-" ] digit { digit }.
letter = "A" | "B" | … | "Z" |
       "a" | "b" | … | "z".
digit = "0" | "1" | … | "8" | "9".
```

Between the parts of a grammatical construct but not within the names of variables or integer numbers, any number of spaces may appear. <EOF> stands for the end of the input file and <CR> stands for the new-line character. All lines in the input file are shorter than 200 characters.

The value of a variable is said to be undefined:

- if it has not yet been defined or it refers to a variable, which has not yet been defined;
- if the definition of the variable contains a cycle.

Your are to write a program that implements Peter's calculator. It should store all variable definitions and for each "PRINT" statement evaluate the specified variable based on the latest variable definitions. If your program encounters a "RESET" statement, it

should delete all stored variables so that all variables become undefined.

Input

The input file contains calculations adhering to the syntax given above. Each line contains either an assignment to a variable, a "PRINT" statement, a "RESET" statement or nothing.

Output

For each "PRINT" statement found in the input file, your program should output a line containing the numerical value of the specified variable or the word "UNDEF" if the variable is undefined.

Example

Input

```
a := b + c
b := 3
c := 5
PRINT d
PRINT a
b := 8
PRINT a
RESET
PRINT a
```

Output

```
UNDEF
8
13
UNDEF
```