## 11886 Infinite Dictionaries

A dictionary is a set of key-value pairs, for example: {'color':'red', 'price':2, 7:'test', 100:-100 }. As you can see, keys and values can be strings or integers. What's more, values can also be dictionaries or variable references. Here is the formal definition of terms that will be used soon:

```
key   ::=      INTEGER | STRING
value ::=      INTEGER | STRING | dict
pair  ::=      key ':' value
dict  ::=      '{' [pair (',' pair)*] '}'
var   ::=      'a'|'b'|'c'|...|'z'
slot  ::=      var('[' key ']')*
lvar  ::=      slot
rvar  ::=      slot | value
```

Here `('[' key ']')*` means zero or more subscripts, `[pair (',' pair)*]` means zero or more key-value pairs.

Strings are always enclosed by single quotes (`''`) and consists of up to 10 lower-case letters. Integers always have absolute values of no more than 1000. You can insert spaces anywhere, except inside strings or integers. For example, { 'a':-1} and {'a' : -1 } are the same, but {'a b':1} and {'a':-1} are both illegal.

Your task is to execute a series of commands and print the results. There are 3 kinds of commands:

***1. Assignment:*** `<lvar> = <rval>`

After assigning a slot to a slot (rather than a value), the left-hand slot will be holding a reference to the right-hand. For example, After executing the following commands, `b[1][0]` is 1, rather than 0:

```
a = {0:0}
b = {}
b[1] = a
a[0] = 1
```

Slots must be assigned before it is read or subscripted, and integers and strings cannot be subscripted. Consider the following comammd list:

```
c = {}
c[0] = 3
c[1] = c[0]
d[0] = 'i'
c = d
d = c[1]['a']
c[2][2] = 2
```

The first three commands are legal, but the next two are both illegal because slot d must be assigned before it is read or subscripted. The last three are also illegal.

***2. Length:*** `length(<slot>)`

Output the number of key-value pairs in the slot. Note that nested pairs are not counted. For example:

```
a = {0: {0:0, 1:1}}
length(a)
```

will output 1, not 3. In this command, it is guaranteed that <slot¿ is storing a dictionary, not a string or an integer.

*3. Infinity test:* `test(<slot>)`

If the slot can be subscripted indefinitely, output 1. Otherwise, output 0. For example, after executing the following command list:

```
d = {}
d[0] = d
```

Then `d` is infinite, since `d[0][0][0][0][0][0]...` is always `d`. In this command, it is guaranteed that <slot> is storing a dictionary, not a string or an integer.

## Input

The input contains at most 10000 lines of commands, each line will be non-empty and will contain no more than 300 characters. All the commands are legal.

## Output

Print the output (one line for each length/test command).

## Sample Input

```
c = {}
d = {'color': 'red', 'price': 2, 7: 'test', 100: -100}
length(d)
d[7] = {'this': 'is', 'a': 'book'}
length(d)
d[8] = {'this' : 'is', 'another' : {'a' : 'book', 'b': 'book2'} }
length(d)
c[7] = c
test(c)
test(d)
length(c)
d[0] = c
length(d)
test(d[0])
```

## Sample Output

```
4
4
5
1
0
```

```
1
6
1
```

**Notes:**

The term definitions in this problem use a modified BNF grammar notation that is described in the official documentation of the Python programming language. Here is the explanation of the notation, extracted from the documentation:

*Each rule begins with a name (which is the name defined by the rule) and ::=. A vertical bar (—) is used to separate alternatives; it is the least binding operator in this notation. A star (*) means zero or more repetitions of the preceding item; likewise, a plus (+) means one or more repetitions, and a phrase enclosed in square brackets ([ ]) means zero or one occurrences (in other words, the enclosed phrase is optional). The * and + operators bind as tightly as possible; parentheses are used for grouping. Literal strings are enclosed in quotes. White space is only meaningful to separate tokens. Rules are normally contained on a single line; rules with many alternatives may be formatted alternatively with each line after the first beginning with a vertical bar.*

---

**Problemsetter:** Rujia Liu, **Special Thanks:** Yiming Li