Speculative execution (spectre)

Given how much Luca loves IT security, you should not be surprised to know that he wants to understand the recent flaw discovered in CPUs: Spectre![1]. The full paper describing the vulnerability is quite intricated and so he wants to proceed a step at a time.

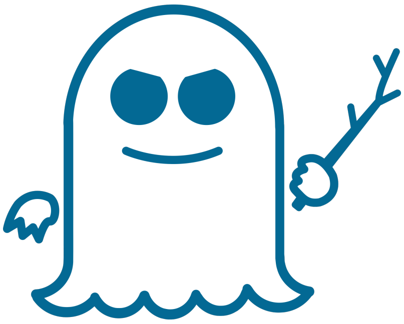

Figure 1: The official vulnerability's logo. Cool, isn't it?

Every respectable programmer knows that a CPU fundamentally executes instructions, one after another. But that is only one part of the story: modern CPUs aim at maximizing performace by executing multiple instructions at once, whenever possible.[2]

You are given a list of N instructions, each in the form

```
target_variable = operand1_variable OP operand2_variable
```

where OP is a mathematical operator among +, -, *, /. An instruction j can be executed immediately if it does not have to "wait" that one of its operand has to be computed (more precisely, a variable needs to be computed when it is the target of another instruction i with $i < j$).

In order to let Luca check that he has understood correctly this matter, compute for him the number of instructions that can be executed immediately!

> ☞ Among the attachments of this task you may find a template file spectre.* with a sample incomplete implementation.

Input

The first line contains the only integer N, the number of instructions. The following N lines contain each a single instruction, in the format specified above.

Output

You need to write a single line with an integer: the number of instructions that can be immediately executed.

[1] Spectre is probably one of the most profund vulnerabilities of this decade. More on https://spectreattack.com/

[2] The idea is called "pipelining", some details can be read at https://en.wikipedia.org/wiki/Instruction_pipelining

Constraints

- $1 \leq N \leq 1\,000\,000$.
- Every variable name is a string consisting in $1 - 10$ lower case ASCII letters.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** [**0 points**]: Examples.
- **Subtask 2** [**20 points**]: Every variable appears only once in the whole set of instructions.
- **Subtask 3** [**20 points**]: All variable names are single letters (a-z).
- **Subtask 4** [**10 points**]: $N \leq 10$.
- **Subtask 5** [**20 points**]: $N \leq 1\,000$.
- **Subtask 6** [**30 points**]: No additional limitations.

Examples

input.txt	output.txt

2
foo = bar + baz
i = foo * i
	1

5
width = length + deltax
height = length + deltay
area = width * height
halfarea = area / two
perimeter = length * four
	3

Explanation

In the **first sample case**, the first instruction can be executed immediately. The second instruction depends on the value of foo, which is computed in the first instruction; thus it *cannot* be executed at the moment.

In the **second sample case**, the first two instructions can be executed immediately. The third one depends on both the first and the second, thus cannot be executed. The fourth one depends on the third (which is on hold), thus neither can be executed. Finally, the last instruction can be executed out-of-order immediately.