ECOO '20 P4 - Ring

Time limit: 20.0s **Memory limit:** 256M

Aside from learning music, Mimi has decided to learn a new programming language! Ring is a simple programming language used to perform simple calculations. A Ring program keeps a single integer `X` in memory, initially set to zero, and repeatedly performs addition, subtraction, and multiplication operations on it.

Formally, the language has six types of syntax:

Syntax	Description
`ADD Y`	Add `Y` to `X`
`SUB Y`	Subtract `Y` from `X`
`MULT Y`	Multiply `X` by `Y`
`FUN F`	Begin the definition of a function with the unique name `F`
`END`	End of the current function definition
`CALL F`	Call function `F`

Function definitions can be nested, in which case `END` represents the end of the most recent function definition that has not already been ended. For function calls, the function `F` must have been defined on a previous line and a function cannot call itself (otherwise, infinite recursion would happen).

For example, the result of the following program is 5:

```
FUN INCREMENT
ADD 1
END
CALL INCREMENT
MULT 2
ADD 3
```

Mimi has written a few programs in Ring, but she finds that her interpreter takes an eternity to execute them. Can you help Mimi determine the results of her Ring programs?

Input Specification

The first line begins with a single integer T $(1 \leq T \leq 10)$, the number of test cases. T test cases follow.

Each test case begins with one integer N $(1 \leq N \leq 10^5)$, the number of instructions in the program.
The next N lines each contain an instruction in the format described above.
All integers will be non-negative and less than or equal to 10^9. Function names will be upper-case and at most 10 letters long.

For the first three cases, there are no function definitions.
For the next two cases, there is at most one function declaration.

Output Specification

For each test case, print the result of the program, modulo $1\,000\,000\,007$.

Note: "X modulo Y" is defined as the positive remainder of X divided by Y.

Sample Input

```
2
3
ADD 1
MULT 1000000000
ADD 7
6
FUN INCREMENT
ADD 1
END
CALL INCREMENT
MULT 2
ADD 3
```

Sample Output

```
0
5
```

Note: you do NOT need to pass the sample to pass some of the cases.