2134 Virtual Computer

James recently purchased a new palm-top computer at a discount store. To get a really good price, he decided to have the free operating system, "JameSIX", installed instead of the more popular and expensive "Windwoes" operating system. One drawback to the free OS is that there are many bugs, but James is a savvy computer programmer and thinks it worth the risk.

A few days later, James is attempting to play "Sweeper" when all of a sudden the screen flashes and some strange output appears. Angered by this interruption to his favorite game, James decides to use his vast programming knowledge and attempt to debug the program in the free debugger, JDB.

James has found a suspicious piece of code. He knows the values of all variables, and he is attempting to determine if the code fragment was the one that produced the output that interrupted his high-score attempt.

The "JameSIX" Assembly language in which the program was written has a very simple set of commands:

Command	Explanation
`START` $< progname >$	The first statement of a program, where $< progname >$ is the program's integer identifier Example: `START 21`
`SET` $< x > < value >$	Will set variable $< x >$'s value to $< value >$ Example: `SET` a `9` is equivalent to $a = 9$ Note: $< x > < y >$ can only be integers.
`STORE` $< x > < y >$	Will set variable $< x >$'s value to the value of variable $< y >$ Example: `STORE` b a is equivalent to $b = a$
`MULT` $< x > < y >$	Equivalent to variable $< x >$ = variable $< x >$ * variable $< y >$ Example: `MULT` b a is equivalent to $b = b * a$
`SUB` $< x > < y >$	Equivalent to variable $< x >$ = variable $< x >$ - variable $< y >$ Example: `SUB` b c is equivalent to $b = b - c$
`DIV` $< x > < y >$	Equivalent to variable $< x >$ = floor(variable $< x >$ / variable $< y >$) except in the case where variable $< y >$'s value is `0`. If variable $< y >$'s value is equal to `0`, then the statement has no effect on the value of variable $< x >$ or variable $< y >$ or any other variable within the program (i.e. $< x > = < x >$). Example: `DIV` b c is equivalent to $b = b - c$
`ADD` $< x > < y >$	Equivalent to variable $< x >$ = variable $< x >$ + variable $< y >$ Example: `ADD` b c is equivalent to $b = b + c$
`PRINT` $< x >$	Prints the value of the variable $< x >$ followed by a carriage-return Example: `PRINT` b
`GOTO` $< n >$	Execution moves to the nth line of the program. The first line is line number 1. Example: `GOTO 10`
`END`	This statement ends the current program. Example: `END`

And one conditional command (non-simple):

Command	Explanation

`IF` $< x >$ `IS` $<$ *value* $>$ `THEN` $<$ *simple_command* $>$	Where $<$ *simple_command* $>$ is any of the previously mentioned simple commands. $<$ *simple_command* $>$ is only executed if the value of the variable $< x >$ is equivalent to the integer $<$ *value* $>$ (this is not a variable).
	Example: `IF` b `IS 3 THEN ADD` b c
	is equivalent to
	`if b = 3 then`
	`b = b + c`
	`endif`

NOTE: All variables are of type integer.

Input

Input will consist of a non-zero number of 'programs', each of which are completely independent of one another. Each program will conform to the following:

1. The first statement (line 1 of each program) will be '`START` $<$ *progname* $>$' (See above for clarification on `START` command).
2. The last statement will be '`END`'.
3. All variables will be initialized with a '`SET`' statement before being used in any other statement.
4. Program ID numbers will be unique.
5. Line numbering resets each time a new program begins (each '`START`' statement).
6. All variables are single lowercase letters (`a`, `b`, `c`, ..., `z`).
7. All variable values will be within the following range throughout the program; $-1000 \leq a \leq 1000$ where a is the variable i.e. '`ADD` x y' will never result in a value outside of the range.

Arguments to individual statements will always be separated by exactly one space, and there will not be any blank lines in the input file.

Output

The output will be the results of the programs in the input file. For a given program, you will print to the output file:

1. A single line containing '`START` $<$ *progname* $>$' exactly like the first line of the program when execution of a new program begins
2. A single line with the value currently stored in the variable $< x >$ when a '`PRINT` $< x >$' statement is executed.

There will be no blank lines separating output sets.

Sample Input

```
START 1
SET b 10
PRINT b
END
START 2
SET b 10
SET a 1
SET c 0
STORE c a
MULT c b
PRINT c
END
START 10
SET a 1
SET b 2
SET c 3
SET d 4
SET e 5
STORE e a
ADD a a
PRINT a
MULT c b
IF e IS 8 GOTO 13
GOTO 7
PRINT c
END
```

Sample Output

```
START 1
10
START 2
10
START 10
2
4
8
16
48
```