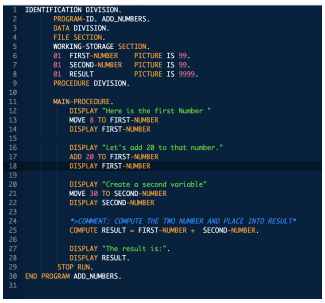
Problema número 786

Se buscan programadores de COBOL

Tiempo máximo: 1,000 s Memoria máxima: 8192 KiB

http://www.aceptaelreto.com/problem/statement.php?id=786



```
IDENTIFICATION DIVISION.
       PROGRAM-ID. ADD_NUMBERS.
       DATA DIVISION.
       FILE SECTION.
       WORKING-STORAGE SECTION.
       01  FIRST-NUMBER    PICTURE IS 99.
       01  SECOND-NUMBER   PICTURE IS 99.
       01  RESULT          PICTURE IS 9999.
       PROCEDURE DIVISION.

       MAIN-PROCEDURE.
           DISPLAY "Here is the first Number "
           MOVE 8 TO FIRST-NUMBER
           DISPLAY FIRST-NUMBER

           DISPLAY "Let's add 20 to that number."
           ADD 20 TO FIRST-NUMBER
           DISPLAY FIRST-NUMBER

           DISPLAY "Create a second variable"
           MOVE 30 TO SECOND-NUMBER
           DISPLAY SECOND-NUMBER

           *>COMMENT: COMPUTE THE TWO NUMBER AND PLACE INTO RESULT*
           COMPUTE RESULT = FIRST-NUMBER +  SECOND-NUMBER.

           DISPLAY "The result is:".
           DISPLAY RESULT.
           STOP RUN.
END PROGRAM ADD_NUMBERS.
```

Alba Cil Illa ha sido desde siempre una amante de los lenguajes de programación esotéricos más enrevesados. Ha pasado innumerables horas "jugando" con *Brainfuck*, *Befunge* o *Whitespace*.

Aunque nunca lo ha hecho con ánimo de sacar partido a esos conocimientos más allá de ir vacilando a sus conocidos (al fin y al cabo esos lenguajes no tienen ninguna utilidad práctica) es posible que eso esté a punto de cambiar. Ha visto una oferta de trabajo que busca programadores en COBOL y ha pensado que, por muy complicado que sea, seguro que lo podrá aprender en poco tiempo y ser contratada. Su habilidad con *Brainfuck* seguro que le sirve para interiorizar rápidamente cómo funciona COBOL. Pero la realidad es que ese antiguo lenguaje de medidados del siglo XX parece más esotérico que los lenguajes que ha utilizado en los últimos tiempos. ¡Lo de usar lenguaje natural para sumar es de lo más desconcertante!

Para ayudarla a aprenderlo crearemos un intérprete de un subconjunto muy reducido de COBOL en el que tendremos asignación de valores a variables, sumas y escritura de variables por pantalla. Para hacerlo más sencillo nos tomaremos, además, ciertas "licencias" que no se corresponden con el lenguaje original.

Nuestros programas estarán compuestos de órdenes, cada una en una línea. Todas las órdenes terminan con un carácter punto ('`.`'), y todos los programas terminan con la orden `STOP`.

Las tres órdenes que admitiremos (además de la de final de programa) son `MOVE` para dar un valor numérico a una variable, `ADD` para hacer operaciones de suma y `DISPLAY` para escribir el valor de las variables.

En nuestro caso las variables serán palabras compuestas por como mucho 5 letras minúsculas del alfabeto inglés. Aunque COBOL necesitaba declarar esas variables en una sección de código independiente, en nuestro caso se declararán automáticamente en su primera aparición en una orden `MOVE`.

La orden `MOVE` tiene la forma `MOVE [valor] TO [variable]` donde `[valor]` es un número no negativo menor que 2×10^9 y `[variable]` es el nombre de una variable. La operación crea la variable si no existía aún y le da el valor indicado.

La orden `DISPLAY` tiene, tras la palabra clave, una lista con los nombres de las variables que hay que escribir. Aparecerán separadas por espacios y asumiremos que todas ellas han sido declaradas para almacenar como mucho 5 dígitos decimales por lo que si tienen que guardar un valor que supere ese número de dígitos, los más significativos se perderán.

Por último, la orden `ADD` tiene dos variantes:

- `ADD var1 ... varN TO destVar1 ... destVarM`: hace la suma de todas las variables de la lista inicial, `var1`...`varN` (como mucho 10), y suma el resultado a las variables `destVar1`...`destVarM`. Podría ocurrir que la misma variable apareciera varias veces en ambas listas.

- `ADD var1 ... varN TO otherVar GIVING destVar1 ... destVarM`: hace la suma de todas las variables `var1`...`varN` (también 10 como mucho) y `otherVar` y el resultado lo *asigna* a las `destVar1`...`destVarM` (en este caso, `otherVar` *no* se modifica). Igual que con la primera variante, la misma variable puede aparecer varias veces.

Entrada

La entrada comienza con una línea con el número de programas que vendrán a continuación.

Cada programa es una secuencia de líneas con las órdenes a ejecutar, que terminan con `STOP`. Cada orden sigue el formato explicado más arriba, donde todas las palabras y números están separados únicamente por un espacio. Todas las líneas terminan con un punto ('`.`') pegado a la última palabra de la orden.

Salida

Cada orden `DISPLAY` de los programas generará una línea con el valor de las variables indicadas separadas por espacios. Recuerda que las variables se asumen declaradas como `9(5)` que significa que son capaces de guardar 5 caracteres numéricos.

Tras cada programa se escribirá una línea con tres guiones (`---`).

Entrada de ejemplo

```
1
MOVE 10 TO a.
MOVE 20 TO b.
ADD a TO b.
DISPLAY a b.
MOVE 0 TO c.
ADD a b TO c.
DISPLAY a b c.
MOVE 100100 TO d.
DISPLAY d.
ADD a b TO c d.
DISPLAY a b c d.
ADD c d TO a GIVING b d.
DISPLAY a b c d.
ADD a TO c c.
DISPLAY c.
STOP.
```

Salida de ejemplo

```
10 30
10 30 40
100
10 30 80 140
10 230 80 230
100
---
```

Autor: Marco Antonio Gómez Martín.

Revisores: Pedro Pablo Gómez Martín y Alberto Verdejo.