El código espagueti

En los albores de la informática, los lenguajes de programación no siempre venían acompañados de entornos integrados de desarrollo, como *Visual Studio*, que facilitaran la tarea de escribir programas. En lugar de ello, había que enfrentarse a una pantalla de línea de comandos, en la que se introducían las instrucciones del programa de manera secuencial, a menudo numeradas, y donde el flujo del programa se dirigía mediante instrucciones de salto, como la famosa `GOTO`. En este contexto, surgieron prácticas como el llamado *código espagueti*, donde las instrucciones se entrelazaban de manera compleja y poco estructurada, reflejando la naturaleza desordenada de la programación en aquellos tiempos.

Este enfoque primitivo, aunque rudimentario en comparación con los IDEs contemporáneos, sentó las bases de lo que hoy conocemos como desarrollo de software, marcando el inicio de una larga evolución hacia herramientas más cómodas y eficientes.

En este ejercicio implementaremos un intérprete para un pequeño lenguaje que solo tiene dos tipos de instrucciones: `INCR` y `GOTO`. Las instrucciones del programa están numeradas, de modo que el intérprete ejecuta las instrucciones en orden ascendente según su número, a menos que haya una instrucción `GOTO` que obligue a alterar ese orden. A continuación se describe lo que hace cada una de las instrucciones:

- `INCR var`

  Incrementa en una unidad el valor de la variable `var` indicada, y salta a la siguiente instrucción del programa. Si no hay otra instrucción después del `INCR`, la ejecución finaliza. Suponemos que, inicialmente, el valor de todas las variables es `0`.

- `GOTO num`

  Salta a la instrucción cuyo número es `num`. Si no existe ninguna instrucción con ese número, el intérprete produce un error.

Siguiendo la antigua usanza, nuestro intérprete funciona mediante una línea de comandos, en la que el programador introduce, una a una, cada instrucción del programa. Por ejemplo:

```
» 10 INCR a
» 20 GOTO 50
» 30 INCR b
» 40 GOTO 20
» 50 INCR a
```

Para ejecutar el programa introducido hasta el momento, podemos utilizar `RUN` desde la línea de comandos. Este comando recibe como parámetro un número `N`, que es el máximo número de instrucciones a ejecutar. La ejecución del programa se detiene si se llegan a ejecutar `N` instrucciones. De este modo, al poner un tope en el número de instrucciones ejecutadas, podemos asegurar que la ejecución terminará aunque el programa tenga bucles infinitos.

Al finalizar la ejecución, el intérprete imprime el valor de todas las variables cuyo valor sea distinto de `0`. Por ejemplo:

```
» RUN 100
a = 2          ← Mostramos en color azul la salida del intérprete,
OK               para distinguirla de la entrada introducida en la línea de comandos.
» RUN 2
a = 1
OK
```

En el primer `RUN`, se empieza ejecutando la instrucción 10, luego la 20, y desde ahí se salta a la 50, tras la cual finaliza el programa. Por tanto, la variable a se ha incrementado dos veces, y su valor final es 2. Por el contrario, en el segundo `RUN` solamente se ejecutan las instrucciones 10 y 20, porque con esta última se alcanza el tope máximo de instrucciones a ejecutar. Observa que las variables se *«resetean»* con cada `RUN`.

Es posible seguir añadiendo instrucciones el programa tras haberlo ejecutado. Para ello basta con introducir la línea con la instrucción correspondiente:

```
» 15 GOTO 30        ← añade una nueva instrucción entre la 10 y la 20
» RUN 100
a = 2
b = 1
OK
```

Al ejecutar el programa resultante se ejecutarán las instrucciones 10, 15, 30, 40, 20 y 50, en ese orden. También pueden modificarse las instrucciones ya existentes:

```
» 30 INCR a         ← modifica la instrucción 30
» RUN 100
a = 3
OK
```

Si la ejecución de un programa falla debido a que se ejecuta un `GOTO` que salta a una instrucción inexistente, el intérprete imprime `ERROR`.

```
» 60 GOTO 24        ← añadimos al final del programa un GOTO a una instrucción que no existe
» RUN 100
ERROR
```

Para salir del intérprete utilizamos el comando `BYE`, que finaliza la sesión.

```
» BYE
```

Entrada

La entrada contiene varios casos de prueba. Cada caso de prueba contiene los comandos que un programador introduce en el intérprete en una sesión de uso. Los comandos tienen la forma "`N instrucción`" para añadir o modificar una instrucción con el número `N`, o "`RUN M`" para ejecutar el programa actual con un tope de `M` instrucciones. Se cumple que $1 \leq N \leq 10^9$ y $0 \leq M \leq 10^5$. Cada sesión finaliza con el comando `BYE`.

Los nombres de variables utilizadas en `INCR` pueden están formadas por letras y números, y no tienen espacios.

Salida

Para cada caso de prueba debe imprimirse la salida que mostrará el intérprete tras la ejecución de cada comando `RUN`. Si la ejecución tiene éxito, debe imprimirse una línea por cada variable que tenga un valor distinto de 0. Cada línea tiene la forma `variable = valor`. Las variables deben imprimirse en orden lexicográfico. Tras imprimir los valores de las variables, debe imprimirse `OK`. Por el contrario, si la ejecución falla, debe imprimirse `ERROR`. En este caso, no se imprimen los valores de las variables.

Al final de cada caso de prueba, debe imprimirse una línea con tres guiones (`---`).

Entrada de ejemplo

```
10 INCR a
20 GOTO 50
30 INCR b
40 GOTO 20
50 INCR a
RUN 100
RUN 2
15 GOTO 30
RUN 100
30 INCR a
RUN 100
60 GOTO 24
RUN 100
BYE
10 INCR xs
20 INCR vs
30 INCR vs
RUN 2000
25 INCR xs
RUN 2000
50 GOTO 10
RUN 50
BYE
10 GOTO 10
RUN 40
BYE
```

Salida de ejemplo

```
a = 2
OK
a = 1
OK
a = 2
b = 1
OK
a = 3
OK
ERROR
---
vs = 2
xs = 1
OK
vs = 2
xs = 2
OK
vs = 20
xs = 20
OK
---
OK
---
```

Créditos

Autor: Manuel Montenegro