C09: El lenguaje *Calculón++*

Estructuras de Datos
Facultad de Informática - UCM

Este ejercicio debe entregarse a través del problema `C09` de *DOMjudge*. La fecha tope de entrega es el **3 de mayo de 2020**.

La entrega consiste en un único fichero `.cpp` que se subirá a *DOMjudge*. Puedes subir tantos intentos como quieras. Se tendrá en cuenta el último intento con el veredicto `CORRECT` que se haya realizado antes de la hora de entrega por parte de alguno de los miembros del grupo.

No olvides poner el nombre de los componentes del grupo en el fichero `.cpp`. Solo es necesario que uno de los componentes del grupo realice la entrega.

Evaluación: Este ejercicio se puntuará de 0 a 10. Para poder obtener una calificación superior a 0 es necesario obtener un veredicto `CORRECT`.

Este ejercicio consiste en implementar un intérprete de un lenguaje ficticio (*Calculón++*) que contiene expresiones aritméticas con números enteros y variables. En *Calculón++* solo existe un tipo de instrucción: la asignación, que tiene la forma $x := e$, donde x es una variable y e es una expresión que puede contener números, variables, y operaciones de suma, resta y multiplicación. Los nombres de variables están formados por letras minúsculas y números, comenzando siempre en letra minúscula. Cada instrucción finaliza con un punto y coma (`;`). A continuación se muestra un programa de ejemplo:

```
x := 3;
y := 5;
z := x * (y + 1);
y := z - 1;
```

Tras la ejecución de este programa, la variable `x` tiene el valor 3, la variable `y` tiene el valor 17, y la variable `z` toma el valor 18. El intérprete debe escribir el estado final de las variables que aparecen en el programa, con el siguiente formato:

```
x = 3
y = 17
z = 18
```

En este listado, las variables deben aparecer ordenadas alfabéticamente.

La primera de las tareas de un intérprete consiste en analizar sintácticamente el programa que se desea ejecutar. En este caso, no es necesario hacer esta tarea. En la plantilla que encontrarás en el Campus Virtual se proporciona una función `parse`, definida del siguiente modo:

```
BinTree<std::string> parse(std::istream &in);
```

Esta función recibe un flujo de entrada (en nuestro caso siempre será `cin`), a partir del cual lee una instrucción de *Calculón++*. La función `parse` devuelve un árbol binario con la representación sintáctica de la instrucción leída. A este árbol se le denomina AST (*Abstract Syntax Tree*). Las hojas de este árbol pueden ser nombres de variables o números, mientras que los nodos internos pueden contener operadores de la forma `+`, `-`, `*` o `:=`. Si un nodo interno tiene un operador, los hijos izquierdo y derecho representan sus operandos. A modo de ejemplo, la siguiente tabla muestra el AST correspondiente a cada una de las instrucciones del ejemplo mostrado anteriormente:

Instrucción	AST
`x := 3;`	`((. x .) := (. 3 .))`
`y := 5;`	`((. y .) := (. 5 .))`
`z := x * (y + 1);`	`((. z .) := ((. x .) * ((. y .) + (. 1 .))))`
`y := z - 1;`	`((. y .) := ((. z .) - (. 1 .)))`

Como puedes ver, el AST siempre tiene la cadena `:=` en su raíz. A la izquierda tiene una hoja con el nombre de una variable, y a la derecha contiene un subárbol con una expresión aritmética.

El objetivo del ejercicio es, por tanto, realizar un programa que lea una serie de bloques de instrucciones escritas en *Calculón*++. Para cada bloque debe imprimir el valor contenido en cada una de las variables tras la ejecución del mismo.

Entrada

La entrada consta de una serie de casos de prueba, cada uno de ellos representando un programa. Cada caso de prueba comienza con un número N, que indica el número de instrucciones de las que se compone el programa, seguido de N líneas con las instrucciones propiamente dichas.

Se garantiza que todas las instrucciones son sintácticamente correctas, y que no se accede a ninguna variable `x` cuyo valor no haya sido asignado previamente. Es decir, ningún programa lee variables sin inicializar. Los números que aparecen en cada expresión aritmética son siempre positivos, aunque el resultado de evaluar dicha expresión puede ser un número comprendido entre -10^8 y 10^8.

La entrada finaliza con un programa de 0 instrucciones, que no se procesa.

Salida

Para cada caso se escribirá el valor final de cada una de las variables involucradas en el programa correspondiente. Para ello se escribirán una serie de líneas de la forma `x = `n, donde `x` es una variable y `n` es un número entero. El listado de variables se mostrará ordenado de manera creciente según el nombre de la variable, utilizando la relación de orden `<` entre cadenas de C++ (esto es, el orden lexicográfico). Al final de cada caso de prueba debe imprimirse una única línea con tres guiones.

Entrada de ejemplo

```
3
x := 3;
y := 5;
z := x + y;
2
b := 5 + 6 * 3;
a := b - 1;
3
x := 1;
x := x * 2;
x := x * 2;
0
```

Salida de ejemplo

```
x = 3
y = 5
z = 8
---
a = 22
b = 23
---
x = 4
---
```