# Evaluando expresiones aritméticas

Este ejercicio consiste en implementar un intérprete de un lenguaje ficticio en el que solo existen asignaciones a variables. Cada una de estas asignaciones tiene la forma $x := e;$, donde $x$ es una variable y $e$ es una expresión que puede contener números enteros, variables, y operaciones de suma, resta y multiplicación. Los nombres de variables están formados por secuencias de letras minúsculas y números, comenzando por letra minúscula. Cada instrucción finaliza con un punto y coma (`;`). Por ejemplo, supongamos el siguiente programa:

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

Tras su ejecución, la variable `x` tiene el valor `3`, la variable `y` el valor `17`, y la variable `z` 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
```

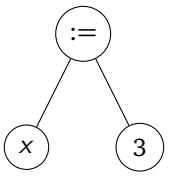
En este listado, las variables deben aparecer ordenadas alfabéticamente.

La primera de las tareas de un intérprete consiste en analizar sintácticamente el programa que se desea ejecutar, pero ese no es el objetivo de este curso. Por eso, en la plantilla se proporciona una función `parse` que ya realiza esta tarea:

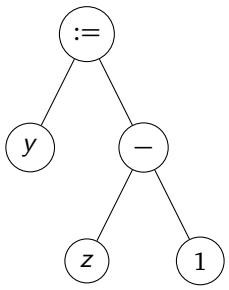
```
BinTree<std::string> parse(std::istream &in);
```

Esta función recibe un flujo de entrada (en nuestro caso siempre será `cin`), a partir del cual lee una instrucción, y devuelve un árbol binario con la representación sintáctica de la instrucción leída. A este árbol se le denomina AST (*Abstract Syntax Tree*). A continuación se muestran los AST correspondientes a algunas de las instrucciones mostradas anteriormente:

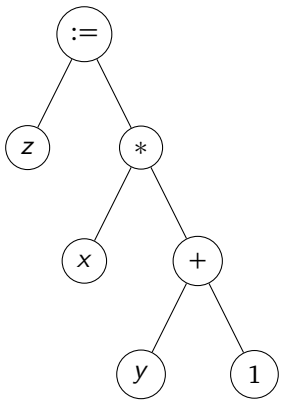
`x := 3;`




`y := z - 1;`




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



Como puedes ver, las hojas de este árbol contienen nombres de variables o números, mientras que los nodos internos contienen operadores `+`, `-`, `*` o `:=`. Cuando un nodo interno tiene un operador, sus hijos izquierdo y derecho contienen los operandos a los que se aplica. Además, como nuestro lenguaje solamente contiene instrucciones de asignación, el AST siempre tiene el operador `:=` 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.

Indica el coste de la solución en función del número *N* de variables, del número *M* de asignaciones de un programa, y del número máximo de nodos *S* que puede haber en el AST de una instrucción.

## Entrada

La entrada consta de una serie de casos de prueba. Cada uno de ellos representa 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 una 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 debe mostrar 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
---
```