Problem 7—Skewed Trees

Trees are particularly annoying to test pilot Hal Jordan. Frequently, the aircraft he is asked to test by his boss, Carol Ferris, are substandard and cannot be guaranteed to avoid the greenery. Hal is not actually afraid of trees—he was evidently born without fear and could never be called "yellow"—but they are annoying.

Hal's spent enough time observing trees to notice that some are more even than others. Some have branches evenly spread throughout the tree; others, though, seem weighted down on one side with a disproportionate amount of branches.

Hal knows nothing of computer science or data structures, but we do, and we know that the same thing happens in binary trees. A binary tree consists of nodes; each node contains a piece of information and the pieces of information can be sorted in some way. (In our example, the information is stored as ASCII strings and the strings are sorted in normal ASCII order.) One node is the "root" of tree and has no parent. All other nodes have a parent. Nodes may have a left child, a right child, both, or neither. In this manner, a binary tree is constructed. The root is at the top; the root's children are one level lower; the root's grandchildren are another level lower, and so forth. There is a unique path from the root down the tree to any other node. For any node in a binary tree, the node's information comes after all the information held in the node's left descendants in sorted order. Similarly, each node's information comes before all the information held in the node's right descendants.

It is possible to design a tree that is "balanced" meaning that every node has about the same number of left descendants as right descendants. It is also possible to design a tree that is "skewed" meaning that some nodes are grossly weighted down on one side or the other. For the purpose of data storage and retrieval, the less skewed a tree is, the better.

If a node has an equal number of left and right descendants or if those numbers differ by one, we say that the node has a skew of zero. Otherwise, the node's skew is one less than the difference between the left and right descendants; a node's skew is never negative. The skew of a tree is the sum of the skews of the nodes contained therein.

Given a binary tree, you are to compute its skew.

INPUT SPECIFICATION. Each data case consists of a series of strings, each one terminated by **<EOLN>**, with an extra **<EOLN>** following the last string in the data case. There is neither a specified maximum number of strings in any data case nor a specified maximum length of any string. The strings represent the nodes of a binary tree printed with a pre-order traversal, meaning that the root is printed first, then the left descendants are recursively printed with a pre-order traversal, then the right descendants are recursively printed with a pre-order traversal. No string appears more than once in any data case. An extra **<EOLN>** follows the last data case. This is not to be processed; it merely signifies the end of the input.

OUTPUT SPECIFICATION. The output cases should appear in the same order as their respective input cases. The output should be in the format "Case c: The skew factor is s" C is the case number; s is the skew of the input tree. Each output case should be followed by exactly one **<EOLN>**.

SAMPLE INPUT.

```
APPLE<EOLN>
BANANA<EOLN>
CANTELOUPE<EOLN>
DATE<EOLN>
<EOLN>
My,·it's·a·lovely·day!<EOLN>
Many·moons·make·man·wiser.<EOLN>
Mary,·Mary,·quite·contrary,·how·does·your·garden·grow?<EOLN>
Nanoo,·Nanoo!<EOLN>
Norman,·Coordinate!<EOLN>
<EOLN>
<EOLN>
<EOF>
```

SAMPLE OUTPUT.

```
Case·1:·The·skew·factor·is·3<EOLN>
Case·2:·The·skew·factor·is·0<EOLN>
<EOF>
```