HOMEWORK 5 - Due 12/17/2018 6PM.

Given a text document, create a binary tree where there is a node for each word that appears at least once in the text and the order of placing the words on the tree is such that left word < parent word < right word in the alphabetical order everywhere in the tree. The node also stores the positions where its word appears in the input text. For example, if word "the" appears in three places: (line 3, offset 10), (line 10, offset 2), and (line 129, offset 33), the line number and offset position (word order on the line) are stored. For example, (line 3, offset 10) means the 10th word on the third line of the text. The program also allows us to edit the text by adding or removing a word at a given position and removing all the occurrences of a given word. The updated text will be saved and output to stdout.

The node information is structured as follows:

```
struct Position {
        int line; // line number
        int offset; // 1, 2, 3, ...: 1st, 2nd, 3rd, ... word on the line
        struct Position *next;
};

struct TreeNode {
        char *word;
        struct Position *positions; // store positions where the word occurs
        struct TreeNode *parent;
        struct TreeNode *left;
        struct TreeNode *right;
};
```

We need to implement the following functions:

```
struct TreeNode *insertNode(struct TreeNode *tree, char word[], struct Position pos) {
        // insert word on a given line at a given offset position
        // if line exists but offset does not, add it as the last word on the line
        // if line does not exist, add it as the first word on a new line at the end
        // if a node for the word already exists, update the position information
        // else, add a new node as left (right) child of some parent node
        // to satisfy the tree order
        // update the tree (including updates of position information of other words)
        ...
}

struct TreeNode *removeWord(struct TreeNode *tree, char[] word) {
        // return same tree if word is not stored in tree
        // else, remove its node
        // update the tree (including updates of position information of other words)
        ...
}

struct TreeNode *removePosition(struct TreeNode *tree, struct Position pos) {
        // find the node that stores a word at this position
        // do nothing and return same tree if not found
        // else, remove this position
        // and remove node if there is no position left associated with the word
        // update the tree (including updates of position information of other words)
        ...
}
```

```
struct TreeNode *removeLine(struct TreeNode *tree, int line) {
        // do nothing and return same tree if line is not found
        // else, remove the entire line and all the corresponding nodes
        // update the tree (including updates of position information of words)
        ...
}

void output(struct TreeNode *tree) {
        // showing the text represented by the input tree
        ...
}


int main() {

        // read stdin and create the tree accordingly
        // perform a number of insert/remove operations according to stdin
        // show the updated text
        …
}
```

INPUT (stdin): A number of lines, a line "END", and the remaining text. Each line before the "END" line represent an insert or remove operation. The text after the "END" line forms the input text. For example, a test input can be as follows:

```
=====begin of stdin input ====
I hello 3 5
I hello 4 50
R world
R 20 1
RL 5
END
blah blah blah ...
blah blah blah ...
blah blah blah ...
blah blah blah ...
=====end of stdin input ====
```

Here, "I" means "insert" and "R" means "remove" and "RL" means removing a line. When two numbers "3 5" are shown, the first number is the line number (3) and the second number is the offset (5). Remember to stop reading stdin when you see the "END" line (not the EOF).

OUTPUT:

The new text resulted from applying the insert/remove operations on the input text. In the above example, the input text is

```
blah blah blah ...
blah blah blah ...
blah blah blah ...
blah blah blah ...
```

and so the insert/remove operations will be applied to this input text.