```c
/*
 * João Paulo Batista Ferreira
 * 2009113274
 * Algoritmos e Estruturas de Dados - TP3 exB
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

typedef struct item {
    char word[256];
    unsigned int counter;
}item;

typedef struct node {
    item info;
    int height;
    struct node *right;
    struct node *left;
} node;

typedef node *nodePtr;

void eraseTree(nodePtr leaf)
{
    if (leaf == NULL)
        return;
    eraseTree(leaf->left);
    eraseTree(leaf->right);
    free(leaf);
}

int max(int a, int b)
{
    return a > b ? a : b;
}

int getHeight(nodePtr leaf)
{
    return max( (leaf->left ? leaf->left->height : 0), (leaf->right ? leaf->right->height : 0
    ) );
}

int difference(nodePtr a)
{
    return (a->left ? a->left->height : 0) - (a->right ? a->right->height : 0);
}

void visita (nodePtr leaf)
{
    printf("%s %d\n", leaf->info.word, leaf->info.counter);
}

void emOrdem (nodePtr leaf)
{
```

```
    if (leaf != NULL)
    {
        emOrdem(leaf->left);
        visita (leaf);
        emOrdem(leaf->right);
    }
}

nodePtr rotationLL(nodePtr leaf)
{
    nodePtr aux;

    aux = leaf->left;
    leaf->left = aux->right;
    aux->right = leaf;

    leaf->height = getHeight(leaf)+1;
    aux->height = getHeight(aux)+1;

    r++;

    return aux;
}

nodePtr rotationRR(nodePtr leaf)
{
    nodePtr aux;

    aux = leaf->right;
    leaf->right = aux->left;
    aux->left = leaf;

    leaf->height =  getHeight(leaf)+1;
    aux->height = getHeight(aux)+1;

    r++;

    return aux;
}

nodePtr rotationLR(nodePtr leaf)
{
    leaf->left = rotationRR(leaf->left);
    return rotationLL (leaf);
}

nodePtr rotationRL(nodePtr leaf)
{
    leaf->right = rotationLL(leaf->right);
    return rotationRR (leaf);
}

nodePtr insert (nodePtr leaf, char* word)
{
    if (leaf == NULL)
    {
        leaf = (nodePtr) malloc (sizeof(node));
```

```
        strcpy(leaf->info.word, word);
        leaf->info.counter = 1;
        leaf->height = 1;
        leaf->left = NULL;
        leaf->right = NULL;

        return leaf;
    }

    if (strcmp(word,leaf->info.word ) < 0)
    {
        t++;
        leaf->left = insert (leaf->left, word);
    }

    else if (strcmp(word,leaf->info.word) > 0)
    {
        t++;
        leaf->right = insert (leaf->right, word);
    }

    else
        leaf->info.counter++;

    if (difference(leaf) == 2)
    {
        if (difference(leaf->left) == 1)
            leaf = rotationLL(leaf);
        else
            leaf = rotationLR(leaf);
    }
    else if (difference(leaf) == -2)
    {
        if (difference(leaf->right) == 1)
            leaf = rotationRL(leaf);
        else
            leaf = rotationRR(leaf);
    }

    leaf->height = getHeight(leaf)+1;
    return leaf;
}

nodePtr worker(char *word, nodePtr tree)
{
    int i, size = strlen(word);

    for (i = 0; i < size; i++)
        word[i] = tolower(word[i]);

    tree = insert(tree, word);

    return tree;
}

int main(int argc, char **argv)
```

```c
{
    char word[256];
    nodePtr tree = NULL;

    while( (scanf("%s", word)) != EOF )
        tree = worker (word, tree);

    emOrdem(tree);

    eraseTree(tree);

    return 0;
}
```