```
/*
LANG:C++
COMPILER:WCB
*/
//============================================================================/
/
// File: top_hit_word.cpp
// The program finds a word (or a group of words) that appear most often
//   in input data.
//
// Author: Pinyo Taeprasartsit
// Copyright: 2011 Pinyo Taeprasartsit
// Modified by Ratchadaporn Kanawong
// Last update: March 2013
//============================================================================/

#include <iostream>
#include <stdio.h>
#include <string.h>

using namespace std;

class TrieNode {
public:
    bool end;
    char key;
    int count;

    TrieNode* parent;
    TrieNode* link[26];

    TrieNode(char* word) {
        this->key = word[0];
        end = false;
        count = 0;
        parent = NULL;
        for (int i = 0; i < 26; ++i)
            link[i] = NULL;
    };
};

void insert(char* word, TrieNode*& root);
void insert2(char* word, TrieNode*& subtree, TrieNode* parent);
int getSlot(char key);
void printAllWords(TrieNode* root);

int topFreq;
int numTop;
string topList[100];

// We can also use this function to check validity of a character.
int getSlot(char key) {
    if (key >= 97)
        key -= 32;  // convert to uppercase.

    if (key < 65 || key > 90)
        return -1;  // invalid slot
    else
        return (key - 65);
}
```

```
void insert(char* word, TrieNode*& root) {
    if (root == NULL)
        root = new TrieNode("*");

    int slot = getSlot(word[0]);
    if (slot == -1)
        return;

    insert2(word, root->link[slot], root);
}

void insert2(char* word, TrieNode*& subtree, TrieNode* parent) {
    char key = word[0];
    if (key == '\0') {
        parent->end = true;
    } else {
        int slot = getSlot(word[0]);
        if (slot == -1)
            return;     // invalid character
        else {
            if (subtree == NULL) { // need new node
                subtree = new TrieNode(word);
                subtree->parent = parent;
            }

            if (word[1] == '\0') {
                subtree->end = true;
                subtree->count += 1;
            } else {
                int nextSlot = getSlot(word[1]);
                insert2(word+1, subtree->link[nextSlot], subtree);
            }
        }
    }
}

void printAllWords2(TrieNode* subtree, const char* prefix) {
    if (subtree == NULL)
        return;
    else {
        string extendPrefix(prefix);
        extendPrefix.push_back(subtree->key);
        if (subtree->end == true) {
            if (subtree->count > topFreq) {     // New top hit found
                topFreq = subtree->count;
                numTop = 1;
                topList[numTop-1] = extendPrefix;
            } else if (subtree->count == topFreq) {
                numTop += 1;
                topList[numTop-1] = extendPrefix;
            }
        }

        for (int i = 0; i < 26; ++i) {
            printAllWords2(subtree->link[i], extendPrefix.c_str());
        }
    }
}

void printAllWords(TrieNode* root) {
    if (root == NULL)
        return;
    else {
```

```
        char* prefix = "";
        for (int i = 0; i < 26; ++i) {
            printAllWords2(root->link[i], prefix);
        }
    }
}

int main(int argc, char* argv[])
{
    //cout << "Hello Top-Hit Word" << endl;
    char str[256];
    //FILE * pFile;

    topFreq = 0;
    numTop = 0;

    TrieNode* root = NULL;
       int numb;
       cin >> numb;
       //cout << endl << numb << endl;
    //pFile = fopen (argv[1], "r");
    while( numb-- > 0) {
        char word[256];
        char dummy[256];
        scanf("%s %s", dummy, word);
        if (dummy[0] == 'I') {
            //sscanf(str, "%s %s", dummy, word);
            insert(word, root);
        }
    }

    // Print top list
    printAllWords(root);
    cout << topFreq << endl;
    cout << numTop << endl;
    for (int i = 0; i < numTop; ++i) {
        cout << topList[i] << endl;
    }

    return 0;
}
```