```
/*
LANG:C++
COMPILER:WCB
*/
//============================================================================/
/
// File: trie_basic.cpp
// The program implements basic operations of dictionary by using a trie.
//
// 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;

    TrieNode* parent;
    TrieNode* link[26];

    TrieNode(char* word) {
        this->key = word[0];
        end = false;
        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);
bool find(char* word, TrieNode* root);
void printPrefixWord(char* word, TrieNode* root);

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

    if (key < 65 || key > 90)   // Invalid character
        return -1;  // To indicate an invalid character
    else
        return (key - 65); // Translate an uppercase character to a proper slot
}

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;
            } 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) {
            printf("%s ", extendPrefix.c_str());
        }

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

// As the root node is special, this function is created to handle the root.
// Then, it delegates the rest of the work to the printAllWords2 function.
void printAllWords(TrieNode* root) {
    if (root == NULL)
        return;
    else {
        char* prefix = "";
        for (int i = 0; i < 26; ++i) {
            printAllWords2(root->link[i], prefix);
        }
    }
}

bool find2(char* word, TrieNode* subtree) {
    char key = word[0];
    if (subtree == NULL) {
        return false;
```

```
    } else {
        char nextKey = word[1];
        if (nextKey == '\0') {  // This is the last character
            if (subtree->end == true)
                return true;
            else
                return false;
        } else {
            int slot = getSlot(nextKey);
            if (slot == -1)
                return false;
            else
                return find2(word+1, subtree->link[slot]);
        }
    }
}

// As the root node is special, this function is created to handle the root.
// Then, it delegates the rest of the work to the find2 function.
bool find(char* word, TrieNode* root) {
    char key = word[0];
    if (key == '\0' || root == NULL) {
        return false;
    } else {
        int slot = getSlot(key);
        return find2(word, root->link[slot]);
    }
}

//========== Helper functions for printing prefixed words.
=====================
TrieNode* getPrefixNode2(char* prefix, TrieNode* subtree) {
    if (subtree == NULL) {
        return NULL;
    } else {
        char nextKey = prefix[1];
        if (nextKey == '\0') {  // This is the last character of the prefix
            return subtree;
        } else {
            int slot = getSlot(nextKey);
            if (slot == -1)
                return NULL;
            else
                return getPrefixNode2(prefix+1, subtree->link[slot]);
        }
    }
}

TrieNode* getPrefixNode(char* prefix, TrieNode* root) {
    char key = prefix[0];
    if (key == '\0' || root == NULL) {
        return false;
    } else {
        int slot = getSlot(key);
        return getPrefixNode2(prefix, root->link[slot]);
    }
}

void printPrefix(TrieNode* subtree, const char* prefix) {
    if (subtree == NULL)
        return;
    else {
        // Check if prefix is also a complete word
```

```
        if (subtree->end == true) {
            printf("%s ", prefix);
        }

        for (int i = 0; i < 26; ++i) {
            printAllWords2(subtree->link[i], prefix);
        }
    }
}
//===== End of Helper functions for printing prefixed words. =====

void printPrefixWord(char* prefix, TrieNode* root) {
    TrieNode* prefixNode = getPrefixNode(prefix, root);
    if (prefixNode == NULL) {
        printf("\nN");
        return;
    } else {
            printf("\nY ");
    }
    printPrefix(prefixNode, prefix);
}

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

    TrieNode* root = NULL;
    int numb;
       cin >> numb;
    //pFile = fopen (argv[1], "r");
    //pFile = fopen("trie_basic_test_data_2.txt", "r");
    while( numb-- > 0) {
        char word[256];
        char dummy[256];
        scanf("%s %s", str, word);

        if (str[0] == 'I') {
            //sscanf(str, "%s %s", dummy, word);
            insert(word, root);
            //cout << "Insert " << word << endl;
        } else if (str[0] == 'F') {
            //sscanf(str, "%s %s", dummy, word);
            bool result = find(word, root);
            if (result)
                cout << "\nY";
            else
                cout << "\nN";
        } else if (str[0] == 'P') {
            printf("\n");
            printAllWords(root);
        } else if (str[0] == 'Q') {
            //sscanf(str, "%s %s", dummy, word);
            printPrefixWord(word, root);
        }
    }

    return 0;
}
```