```
/*
LANG:C++
COMPILER:WCB
*/
//=============================================================================/
/
// File: Augment.cpp
// The program finds a frequency of each integer key by using a binary search
//   tree.
//
// 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 TreeNode {
public:
    int key;
    int count;  // New variable for data-structure augmentation

    TreeNode* parent;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int key) {
        this->key = key;
        count = 1;
        parent = left = right = NULL;
    };
};

TreeNode* insert(int key, TreeNode*& current, TreeNode* parent) {
    if (current == NULL) {
        current = new TreeNode(key);
        current->parent = parent;
        return current;
    }
    else if (key < current->key) {
        return insert(key, current->left, current);
    } else if (key > current->key) {
        return insert(key, current->right, current);
    } else {
        current->count += 1;  // Increase counter of an existing node
        return current; // Return an existing node
    }
}

TreeNode* find(int key, TreeNode* current) {
    if (current == NULL)
        return NULL;
    else if (key < current->key)
        return find(key, current->left);
    else if (key > current->key)
        return find(key, current->right);
    else //if (key == current->key)
        return current;
```

```
}

TreeNode* findMax(TreeNode* current) {
    if (current == NULL)
        return NULL;
    else if (current->right == NULL)
        return current;
    else
        return findMax(current->right);
}

TreeNode* findMin(TreeNode* current) {
    if (current == NULL)
        return NULL;
    else if (current->left == NULL)
        return current;
    else
        return findMin(current->left);
}

void remove(int key, TreeNode*& current) {
    if (current == NULL)
        return;     // No match node, do nothing
    else if (key < current->key)
        return remove(key, current->left);
    else if (key > current->key)
        return remove(key, current->right);
    else {
        if (current->count > 1) {
            current->count -= 1;
        }
        else if (current->left != NULL && current->right != NULL) {
            TreeNode* replacer = findMax(current->left);
            current->key = replacer->key;
            current->count = replacer->count;

            // IMPORTANT: To ensure that the replacer node will be physically
deleted
            replacer->count = 1;
            remove(replacer->key, current->left);
        } else {
            TreeNode* temp = current;
            if (current->left != NULL)
                current = current->left;
            else
                current = current->right;

            delete temp;
        }
    }
}

void inorder(TreeNode* current) {
    if (current == NULL)
        return;
    else {
        inorder(current->left);
        printf("%d(%d) ", current->key, current->count);
        inorder(current->right);
    }
}

void preorder(TreeNode* current) {
    if (current == NULL)
        return;
    else {
        printf("%d(%d) ", current->key, current->count);
        preorder(current->left);
        preorder(current->right);
    }
}

void postorder(TreeNode* current) {
    if (current == NULL)
        return;
    else {
        postorder(current->left);
        postorder(current->right);
        printf("%d(%d) ", current->key, current->count);
    }
}

int main()
{
    //cout << "Hello Key Counter" << endl;
    int key;
    char str[256]="Start";
    //FILE * pFile;
    TreeNode* root = NULL;
    //pFile = fopen ("counter_test_data_10.txt", "r");

    //printf("%s\n", str);
    while( str[0] != 'X' ) {
        scanf("%s %d", str, &key);
        char dummy[256];

        if (str[0] == 'I') {
            //sscanf(str, "%s %d", dummy, &key);
            insert(key, root, NULL);
        } else if (str[0] == 'P') {
            //sscanf(str, "%s %d", dummy, &key);
            if (key == 1) {
                printf("\n");
                inorder(root);
            } else if (key == 2) {
                printf("\n");
                preorder(root);
            } else {
                printf("\n");
                postorder(root);
            }
        } else if (str[0] == 'R') {
            //sscanf(str, "%s %d", dummy, &key);
            remove(key, root);
        } else if (str[0] == 'F') {
            //sscanf(str, "%s %d", dummy, &key);
            TreeNode* node = find(key, root);
            if (node != NULL) {
                printf("\nY %d", node->count);
            } else {    // If the pointer is NULL, the node is not in the tree.
                printf("\nN");
            }
        }
    }
    return 0;
}
```