```
/*
LANG:C++
COMPILER:WCB
*/
//============================================================================//
// File: tree_basic.cpp
// The program creates the simple 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;

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

    TreeNode(int key) {
        this->key = key;
        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 {
        return NULL;    // Do nothing
    }
}

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->left != NULL && current->right != NULL) {
        TreeNode* replacer = findMax(current->left);
        current->key = replacer->key;
        remove(replacer->key, current->left);
    } else {
        TreeNode* temp = current;
        if (current->left != NULL)
            current = current->left;
        else
            current = current->right;

        delete temp;
    }
}

TreeNode* findN(int key, TreeNode* root) {
  if (root == NULL)
    return NULL;

  TreeNode* curr = root;
  while(curr != NULL) {
    if (curr->key == key)
      return curr;
    else if (key < curr->key)
      curr = curr->left;
    else if (key > curr->key)
      curr = curr->right;
  }
  return NULL;    // No match
}

TreeNode* findMaxN(TreeNode* root) {
  if (root == NULL)
    return NULL;
  else {
    TreeNode* curr = root;
    while (curr->right != NULL)
      curr = curr->right;
    return curr;
  }
}
```

```
void painful_remove(int key, TreeNode*& root) {
    if (root == NULL)   // Tree is empty, do nothing.
        return;         // Do nothing

    TreeNode* target = findN(key, root);
    if (target == NULL)     // no node with the specified key
        return;             // do nothing

    TreeNode* targetParent = target->parent;
    if (target->left == NULL && target->right == NULL) {// target is a leaf node
        if (targetParent != NULL) {               // Target is not the root node.
            if (key < targetParent->key)
                targetParent->left = NULL;
            else
                targetParent->right = NULL;
        } else
            root = NULL;    // Reset a tree.
        delete target;
    } else
    if (target->left != NULL && target->right != NULL) {
    // target has two children
    // Update target data by replacing the target's key value
    // 1. Find the replacer node
    TreeNode* replacer = findMaxN(target->left);
    // 2. Get the replacer key
    int replacerKey = replacer->key;
    // 3. Update links (do not replace the key yet. See the note of step 4 below.)
    TreeNode* replacerParent = replacer->parent;
    int parentKey = replacerParent->key;
    if (replacerKey < parentKey)
        replacerParent->left = replacer->left;
    else
        replacerParent->right = replacer->left;

    if (replacer->left != NULL)
        replacer->left->parent = replacer->parent;
    // 4. Replace the target key.
    // Note: key replacement is done here.  Otherwise, above comparison
    // (replacerKey < parentKey) may not be valid if the deleted node
    //   is the parent of the replacer node.
    target->key = replacerKey;
    delete replacer;
    } else {    // target has exactly one child, root may be updated
        TreeNode* child;
        if (target->left != NULL)
            child = target->left;
        else
            child = target->right;
        if (targetParent == NULL) {   // Root is deleted.
            root = child;
        } else {
            if (target->key < targetParent->key)
                targetParent->left = child;
            else
                targetParent->right = child;
        }
        child->parent = target->parent;
        delete target;
    }
}

void inorder(TreeNode* current) {
    if (current == NULL)
```

```
        return;
    else {
        inorder(current->left);
        cout << current->key << " ";
        inorder(current->right);
    }
}
void preorder(TreeNode* current) {
    if (current == NULL)
        return;
    else {
        cout << current->key << " ";
        preorder(current->left);
        preorder(current->right);
    }
}
void postorder(TreeNode* current) {
    if (current == NULL)
        return;
    else {
        postorder(current->left);
        postorder(current->right);
        cout << current->key << " ";
    }
}
int main() {
    char str[256];
    //FILE * pFile;
    int key;
    TreeNode* root = NULL;
    //pFile = fopen ("pain_test_data_10.txt", "r");
    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] == 'F') {
            //sscanf(str, "%s %d", dummy, &key);
            TreeNode* node = find(key, root);
            if (node == NULL)
                printf("\nN");
            else
                printf("\nY");
        } else if (str[0] == 'R') {
            //sscanf(str, "%s %d", dummy, &key);
            //remove(key, root);
            painful_remove(key, root);
        } else if (str[0] == 'P') {
            sscanf(str, "%s %d", dummy, &key);
            if (key == 1) {
                cout << "\n"; inorder(root);
            } else if (key == 2) {
                cout << "\n";  preorder(root);
            } else if (key == 3) {
                cout << "\n";  postorder(root);
            }
        }
    }
    return 0;
}
```