#include<iostream>

using namespace std;

class TreeNode {

int value;

TreeNode\* left;

TreeNode\* right;

public:

TreeNode(int value) {

this->value = value;

}

void setValue(int v) {

value = v;

}

int getValue() {

return value;

}

void setLeft(TreeNode\* l) {

left = l;

}

TreeNode\* getLeft() {

return left;

}

void setRight(TreeNode\* r) {

right = r;

}

TreeNode\* getRight() {

return right;

}

};

class Tree {

TreeNode\* root;

public:

Tree() {

root = NULL;

}

bool isEmpty() {

return (root == NULL);

}

void insertNode(int value) {

if (isEmpty()) {

TreeNode\* temp = new TreeNode(value);

root = temp;

}

else {

TreeNode\* parent = NULL;

TreeNode\* temp2 = root;

while (temp2 != NULL) {

if (temp2->getValue() == value) {

cout << "Duplicate Value" << endl;

break;

}

else if (value < temp2->getValue()) {

parent = temp2;

temp2 = temp2->getLeft();

}

else if (value > temp2->getValue()) {

parent = temp2;

temp2 = temp2->getRight();

}

}

TreeNode\* newNode = new TreeNode(value);

if (parent->getValue() > value) {

parent->setLeft(newNode);

}

else {

parent->setRight(newNode);

}

}

}

void inorder(TreeNode\* node) {

if (node != NULL) {

inorder(node->getLeft());

cout << node->getValue() << endl;

inorder(node->getRight());

}

}

void display() {

inorder(root);

}

void removeNode(int toRemove) {

TreeNode\* target = root;

TreeNode\* parent = NULL;

while (target != NULL) {

if (target->getValue() == toRemove) {

break;

}

else if (target->getValue() > toRemove) {

parent = target;

target = target->getLeft();

}

else {

parent = target;

target = target->getRight();

}

}

if (target == NULL) {

cout << "Not Found" << endl;

}

else {

if (target->getLeft() == NULL && target->getRight() == NULL) {

// Case 1: No children

if (parent->getLeft() == target) {

parent->setLeft(NULL);

}

else {

parent->setRight(NULL);

}

delete target;

}

else if (target->getLeft() == NULL && target->getRight() != NULL) {

// Case 2: One child (right)

if (parent->getLeft() == target) {

parent->setLeft(target->getRight());

}

else {

parent->setRight(target->getRight());

}

delete target;

}

else if (target->getLeft() != NULL && target->getRight() == NULL) {

// Case 2: One child (left)

if (parent->getLeft() == target) {

parent->setLeft(target->getLeft());

}

else {

parent->setRight(target->getLeft());

}

delete target;

}

else {

// Case 3: Two children

TreeNode\* successor = findSuccessor(target->getRight());

int successorValue = successor->getValue();

removeNode(successorValue);

target->setValue(successorValue);

}

}

}

TreeNode\* findSuccessor(TreeNode\* node) {

while (node->getLeft() != NULL) {

node = node->getLeft();

}

return node;

}

};

int main() {

Tree tree1;

tree1.insertNode(500);

tree1.insertNode(300);

tree1.insertNode(505);

tree1.insertNode(507);

tree1.insertNode(150);

tree1.insertNode(305);

cout << "Original Tree:" << endl;

tree1.display();

tree1.removeNode(505);

cout << "After Deleting 505:" << endl;

tree1.display();

return 0;

}