#include<iostream>

#include<string>

using namespace std;

//int lr=0,rr=0;

struct avlnode

{

int value;

avlnode \*left;

avlnode \*right;

int height;

};

class avl

{

private:

public:

avlnode \*root;

avlnode \*temp;

//////////////// CONSTRUCTOR ////////////

avl()

{

temp = root = NULL;

}

/////////////// HEIGHT /////////////////

int height(avlnode \*temp)

{

if (temp == NULL)

{

return 0;

}

else

{

int lr = height(temp->left);

int rr = height(temp->right);

if (lr > rr)

{

return (lr + 1);

}

else

{

return (rr + 1);

}

}

}

/\*int height(avlnode \*ro)

{

if(ro->left==NULL && ro->right==NULL)

{

return 0;

}

else

{

if(ro->left!=NULL)

lr=height(ro->left);

if(ro->right!=NULL)

rr=height(ro->right);

return max(lr,rr)+1;

}

}\*/

//////////////// LEFT ROTATION ////////////

void right\_right(avlnode \*temp)

{

avlnode \*temp1 = new avlnode;

temp1->value = temp->value;

temp1->right = temp->right->left;

temp1->left = temp->left;

temp->value = temp->right->value;

temp->left = temp1;

avlnode \*replace = new avlnode;

replace = temp->right;

temp->right = temp->right->right;

delete replace;

replace = NULL;

return;

}

/////////////// RIGHT ROTATION ///////////

void left\_left(avlnode \*temp)

{

avlnode \*temp1 = new avlnode;

temp1->value = temp->value;

temp1->left = temp->left->right;

temp1->right = temp->right;

temp->value = temp->left->value;

temp->right = temp1;

avlnode \*replace = new avlnode;

replace = temp->left;

temp->left = temp->left->left;

delete replace;

replace = NULL;

return;

}

////////////////// LEFT RIGHT ROTATION /////////////////

void right\_left(avlnode \*temp)

{

avlnode \*temp1 = temp->right;

temp->right = temp1->left;

temp1->left = temp1->left->right;

temp->right->right = temp1;

right\_right(temp);

}

//////////////// RIGHT LEFT ROTATION /////////////

void left\_right(avlnode \*temp)

{

avlnode \*temp2 = temp->left;

temp->left = temp2->right;

temp2->right = temp->left->left;

temp->left->left = temp2;

left\_left(temp);

}

///////////// ROTATIONS /////////////

void rotation(avlnode \*temp , int x)

{

int theight = 0;

int lheight = height(temp->left);

int rheight = height(temp->right);

theight = lheight - rheight;

if (theight <= -2)

{

if (temp->value < x && temp->right->value < x)

right\_right(temp);

else

right\_left(temp);

}

else if (theight >= 2)

{

if (temp->value > x && temp->left->value > x)

left\_left(temp);

else

left\_right(temp);

}

}

//////////// INSERTION ////////////////

void insert(int x , avlnode \*temp)

{

if (root == NULL)

{

root = new avlnode;

root->value = x;

root->left = NULL;

root->right = NULL;

}

else if (temp->value > x)

{

if (temp->left == NULL)

{

temp->left = new avlnode;

temp = temp->left;

temp->value = x;

temp->left = NULL;

temp->right = NULL;

return;

}

else

{

insert(x , temp->left);

rotation(temp , x);

}

}

else if (temp->value < x)

{

if (temp->right == NULL)

{

temp->right = new avlnode;

temp = temp->right;

temp->value = x;

temp->left = NULL;

temp->right = NULL;

return;

}

else

{

insert(x , temp->right);

rotation(temp , x);

}

}

}

/////////////////////// BALANCING /////////////////////

avlnode \*balance(avlnode \* & t )

{

if ( t == NULL)

cout << "Tree is empty" << endl;

if(height(t->left) - height(t->right) > 1)

{

if (height(t->left->left) >= height(t->left->right))

right\_right(t);

else

right\_left(t);

}

else if (height(t->right) - height(t->left) > 1)

{

if (height(t->right->right) >= height(t->right->left))

left\_left (t);

else

left\_right(t);

}

t->height = max(height(t->left) , height(t->right)) + 1;

return t;

}

/////////////// MINIMUM //////////////////

int minimum(avlnode \*temp)

{

if (temp == NULL)

{

return NULL;

}

else if (temp->left == NULL)

{

return temp->value;

}

else

return minimum(temp->left);

}

///////////////// DELETION ////////////////

avlnode \*DeleteNode(int x, avlnode \*t )

{

if(t == NULL)

{

cout << "Tree is empty" << endl;

}

else if (x < t->value)

{

t->left = DeleteNode(x, t->left);

}

else if ( t->value < x )

{

t->right = DeleteNode(x, t->right);

}

else if (t->left != NULL && t->right != NULL)

{

t->value = minimum(t->right);//->value;

t->right = DeleteNode(t->value, t->right);

}

else

{

avlnode \*TempNode= t;

if( t->left == NULL && t->right!=NULL) t = t->right;

if( t->right == NULL && t->left!=NULL) t = t->left;

delete TempNode;

}

balance(t);

return t;

}

/////////////// PRINTING ///////////////

void print(avlnode \*temp)

{

if (temp == NULL)

{

return;

}

print(temp->left);

cout << temp->value << " ";

print(temp->right);

}

};

void main()

{

avl a;

int x = 0;

int t = 0;

int d;

cout << "Enter Number Of Nodes = " << endl;

cin >> t;

cout << endl;

for (int i = 0; i <= t; i++)

{

cout << "Enter the value:" << endl;

cin >> x;

a.insert(x , a.root);

}

cout << endl;

a.print(a.root);

cout << endl;

//a.balance(a.root);

for (int i = 0; i < 10; i++)

{

cout << "enter value to be deleted = " << endl;

cin >> d;

a.DeleteNode(d,a.root);

}

a.print(a.root);

system ("pause");

}