#include <stdio.h>

#include <stdlib.h>

#include <time.h>

typedef struct Node {

int data;

long k;

struct Node \*left,\*right,\*parent;

} NODE ;

typedef struct {

NODE \*head;

} BST;

NODE \* Search(NODE \*root, int x);

void Insert(BST \*t, int x);

void Delete(BST \*t, int x);

void Init(BST \*t, int x);

void dfs(NODE \*root);

int main()

{

int i,j,k;

BST t;

NODE \*node;

srand((unsigned)time(**NULL**));

t.head = **NULL**;

Init(&t, 10);

Insert(&t, 5);

Insert(&t, 3);

Insert(&t, 2);

Insert(&t, 6);

Insert(&t, 11);

Delete(&t, 6);

node = Search(t.head, 5);

if (node)

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%d**\n**",node->data);

dfs(t.head);

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("**\n**");

system("pause");

return 0;

}

NODE \* Search(NODE \*root, int x)

{

while (root && root->data!=x)

{

if (root->data>x)

root = root->left;

else

root = root->right;

}

return root;

}

void Insert(BST \*t, int x)

{

NODE \*node,\*child,\*parent,\*root;

root = t->head;

node = root;

child = root;

while (node && child)

{

if (node->data==x)

child = **NULL**;

else

{

parent = node;

if (node->data>x)

node = node->left;

else

node = node->right;

}

}

if (child)

{

child = (NODE \*)malloc(sizeof(NODE));

child->left = child->right = **NULL**;

child->k = rand();

child->data = x;

child->parent = parent;

if (parent->data>x)

parent->left = child;

else

parent->right = child;

node = child;

while (node->parent->parent && node->k<node->parent->k)

{

parent = node->parent;

if (node->data<parent->data)

{

parent->left = node->right;

if (node->right)

{

parent->left->parent = parent;

}

node->right = parent;

if (parent->data<parent->parent->data)

parent->parent->left = node;

else

parent->parent->right = node;

}

else

{

parent->right = node->left;

if (node->left)

{

parent->right->parent = parent;

}

node->left = parent;

if (parent->data<parent->parent->data)

{

parent->parent->left = node;

}

else

{

parent->parent->right = node;

}

}

node->parent = parent->parent;

parent->parent = node;

}

if (node->parent==root && node->k<root->k)

{

if (node->data<root->data)

{

root->left = node->right;

if (node->right)

root->left->parent = root;

node->right = root;

}

else

{

root->right = node->left;

if (node->left)

root->right->parent = root;

node->left = root;

}

root->parent = node;

node->parent = **NULL**;

t->head = node;

}

}

return ;

}

void Delete(BST \*t,int x)

{

NODE \*node,\*parent,\*node1,\*parent1,\*root;

int temp;

root = t->head;

node = root;

while (node && node->data!=x)

{

parent = node;

if (node->data>x)

node = node->left;

else

node = node->right;

}

if (!node)

return;

if (node!=root)

{

node->k = 1000000;

while (node->left || node->right)

{

if (node->left)

{

node1 = node->left;

node->left = node1->right;

if (node1->right)

node1->right->parent = node;

}

else

{

node1 = node->right;

node->right = node1->left;

if (node1->left)

node1->left->parent = node;

}

if (node->parent->data>node->data)

node->parent->left = node1;

else

node->parent->right = node1;

node1->parent = node->parent;

node->parent = node1;

}

if (node->data<node->parent->data)

node->parent->left = **NULL**;

else

node->parent->right = **NULL**;

free(node);

}

return ;

}

void Init(BST \*t, int x)

{

t->head = (NODE \*)malloc(sizeof(NODE));

t->head->left = t->head->right = **NULL**;

t->head->data = x;

t->head->k = rand();

t->head->parent = **NULL**;

return ;

}

void dfs(NODE \*root)

{

if (root)

{

dfs(root->left);

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%d ",root->data);

dfs(root->right);

}

return ;

}