#include <iostream>

#include <stack>

#include <queue>

using namespace std;

struct BitreeNode

{

int data;

struct BitreeNode \*lchild, \*rchild;

};

int inx = 0;

void CreateBiTree(BitreeNode\* &t, int data[]) {

//按先序输入二叉树中结点的值（一个字符），空格字符代表空树，

//构造二叉树表表示二叉树T。

int value = data[inx++];

if (value == 0) t = nullptr;

else {

t = new BitreeNode;//产生新的子树

t->data = value;

CreateBiTree(t->lchild, data);//递归创建左子树

CreateBiTree(t->rchild, data);//递归创建右子树

}

}//CreateTree

void InitTreeNode(BitreeNode &t, int data, BitreeNode \*lchild, BitreeNode \*rchild)

{

t.data = data;

t.lchild = lchild;

t.rchild = rchild;

}

//前序

void PreOrder(BitreeNode \*t)

{

if (t != nullptr)

{

cout << t->data << endl;

PreOrder(t->lchild);

PreOrder(t->rchild);

}

}

//中序

void Inorder(BitreeNode \*t)

{

if (t != nullptr)

{

Inorder(t->lchild);

cout << t->data << endl;

Inorder(t->rchild);

}

}

//后序

void PostOrder(BitreeNode \*t)

{

if (t != nullptr)

{

PostOrder(t->lchild);

PostOrder(t->rchild);

cout << t->data << endl;

}

}

//前序非递归

void PreOrder2(BitreeNode \*t)

{

stack<BitreeNode \*> s;

BitreeNode \*p = t;

while (p != nullptr || !s.empty())

{

if (p != nullptr)

{

cout << p->data << endl;//根

s.push(p);

p = p->lchild;//左

}

else

{

p = s.top();

p = p->rchild;//右

s.pop();

}

}

}

//中序非递归

void Inorder2(BitreeNode \*t)

{

stack<BitreeNode\*> s;

BitreeNode \*p = t;

while (p != nullptr || !s.empty())

{

if (p != nullptr)

{

s.push(p);

p = p->lchild;//左

}

else

{

p = s.top();

cout << p->data << endl;//根·

s.pop();

p = p->rchild;//右

}

}

}

//后序非递归

void PostOrder2(BitreeNode \*t)

{

stack<BitreeNode \*> s;

BitreeNode \*p = t;

BitreeNode \*tmp = nullptr;

//tmp 指针是为了保存节点上一次访问的状态

while (p != nullptr || !s.empty())

{

if (p != nullptr)

{

s.push(p);

p = p->lchild;//左

}

else

{

//每次取栈定元素

//判断栈顶元素的右孩子是否为空，

//如果不为空，查看之前访问的节点是不是该栈顶元素的右孩子

p = s.top();

if (p->rchild != nullptr && p->rchild != tmp)

{

p = p->rchild;//右

}

else

{

cout << p->data << endl;//根

s.pop();

//每次访问节点之后，需要将缓存上一次访问的节点，并且将指针置空

tmp = p;

p = nullptr;

}

}

}

}

//层次遍历

void LevelOrder(BitreeNode \*t)

{

queue<BitreeNode \*> q;

BitreeNode \*p;

q.push(t);

while (!q.empty())

{

p = q.front();

q.pop();

cout << p->data << endl;

if (p->lchild != nullptr)

q.push(p->lchild);

if (p->rchild != nullptr)

q.push(p->rchild);

}

}

//二叉树翻转递归

void invert(BitreeNode \*root)//反转二叉树

{

if (root == nullptr)

return;

BitreeNode \*temp = root->lchild;

root->lchild = root->rchild;

root->rchild = temp;

invert(root->lchild);

invert(root->rchild);

}

//二叉树翻转非递归

void invert\_loop(BitreeNode\* root)

{

if (root == nullptr)

return;

queue<BitreeNode\*> q;

q.push(root);

while (!q.empty()) {

BitreeNode \*node = q.front();

q.pop();

BitreeNode \*tmp = node->lchild;

node->lchild = node->rchild;

node->rchild = tmp;

if (node->lchild) q.push(node->lchild);

if (node->rchild) q.push(node->rchild);

}

}

//遍历是二叉树各种操作的基础

//1、已知树求结点的双亲结点：使用前、中、后序遍历

//2、求结点的孩子结点：使用前、中、后序遍历

//3、求二叉树的深度：使用层次遍历

//4、求二叉树的叶子结点个数：使用层次遍历

//5、判断两颗二叉树是否相同：依次遍历每个节点进行比较

int main()

{

//BitreeNode t1, t2, t3, t4, t5, t6, t7;

//InitTreeNode(t4, 4, nullptr, nullptr);

//InitTreeNode(t5, 5, nullptr, nullptr);

//InitTreeNode(t6, 6, nullptr, nullptr);

//InitTreeNode(t7, 7, nullptr, nullptr);

//InitTreeNode(t2, 2, &t4, &t5);

//InitTreeNode(t3, 3, &t6, &t7);

//InitTreeNode(t1, 1, &t2, &t3);

//LevelOrder(&t1);

BitreeNode \*tree;

int data[] = {1,2,3,0,0,4,0,0,5,6,0,0,7,0,0};

CreateBiTree(tree, data);

//PostOrder2(tree);

//LevelOrder(tree);

invert(tree);

return 0;

}