实验4内容：二叉树的遍历

基本内容：

建立以左右孩子链接结构表示的二叉树，实现二叉树的先序、中序、后序的递归和非递归方式遍历，分层遍历、统计树的高度。

算法描述：

1. 先序遍历：

递归：输出节点--递归遍历左子树--递归遍历右子树

非递归：创建栈--根节点入栈--弹出栈顶节点并输出--右先进左后进，直到栈空。

1. 中序和后序大致如上
2. 层次遍历：

创建队列，根节点入队--弹出队头节点，输出，左右节点入队--循环上一步直到队列为空。

1. 统计数的高度：

递归计算左子树和右子树的高度，取较大值然后+1即为树的高度。

代码：

#define \_CRT\_SECURE\_NO\_WARNI

#include<stdio.h>

#include<stdlib.h>

//定义节点

typedef struct node {

int data;

struct node\* l;

struct node\* r;

}treenode;

//创建节点

treenode\* creatnode(int data) {

treenode\* node = (treenode\*)malloc(sizeof(treenode));

node->data = data;

node->l = NULL;

node->r = NULL;

return node;

}

//递归先序，根左右

void dgxianxu(treenode\* root) {

if (root != NULL) {

printf("%d", root->data);

dgxianxu(root->l);

dgxianxu(root->r);

}

}

//非递归先序

void fdgxianxu(treenode\* root) {

if (root == NULL) {

return;

}

treenode\* s[100];//队列

int t = 0;

s[t++] = root;

while (t > 0) {

treenode\* node = s[--t];//栈顶元素出栈

printf("%d", node->data);

if (node->r != NULL) {

s[t++] = node->r;

}

if (node->l != NULL) {

s[t++] = node->l;

}

}

}

//递归中序，左根右

void dgzhongxu(treenode\* root) {

if (root != NULL) {

dgzhongxu(root->l);

printf("%d", root->data);

dgzhongxu(root->r);

}

}

//非递归中序

void fdgzhongxu(treenode\* root) {

if (root == NULL) {

return;

}

treenode\* s[100];

int t = 0;

treenode\* node = root;

while (t > 0 || node != NULL) {

while (node != NULL) {

s[t++] = node;

node = node->l;

}

node = s[--t];//栈顶元素出栈

printf("%d", node->data);

node = node->r;

}

}

//递归后序，左右根

void dghouxu(treenode\* root) {

if (root != NULL) {

dghouxu(root->l);

dghouxu(root->r);

printf("%d", root->data);

}

}

//非递归后序

void fdghouxu(treenode\* root) {

if (root == NULL) {

return;

}

treenode\* s1[100];

treenode\* s2[100];

int t1 = 0, t2 = 0;

s1[t1++] = root;

while (t1 > 0) {

treenode\* node = s1[--t1];

s2[t2++] = node;

if (node->l != NULL) {

s1[t1++] = node->l;

}

if (node->r != NULL) {

s1[t1++] = node->r;

}

}

while (t2 > 0) {

treenode\* node = s2[--t2];

printf("%d", node->data);

}

}

void fencengbianli(treenode\* root) {

if (root == NULL) {

return;

}

treenode\* queue[100];//队列

int front = 0, rear = 0;

queue[rear++] = root;

while (front < rear) {

treenode\* node = queue[front++];

printf("%d", node->data);

if (node->l != NULL) {

queue[rear++] = node->l;

}

if (node->r != NULL) {

queue[rear++] = node->r;

}

}

}

int high(treenode\* root) {

if (root == NULL) {

return 0;

}

int lefthigh = high(root->l);

int righthigh = high(root->r);

return (lefthigh > righthigh ? lefthigh + 1 : righthigh + 1);

}

int main() {

treenode\* root = creatnode(1);

root->l = creatnode(2);

root->r = creatnode(3);

root->l->l = creatnode(4);

root->l->r = creatnode(5);

root->r->l = creatnode(6);

root->r->r = creatnode(7);

//递归先序

printf(" 递归先序：");

dgxianxu(root);

printf("\n");

//非递归先序

printf("非递归先序：");

fdgxianxu(root);

printf("\n");

//递归中序

printf("递归中序：");

dgzhongxu(root);

printf("\n");

//非递归中序

printf("非递归后序：");

fdgzhongxu(root);

printf("\n");

//递归后序

printf("递归后序：");

dghouxu(root);

printf("\n");

//非递归后序

printf("非递归后序：");

fdghouxu(root);

printf("\n");

printf("层次遍历：");

fencengbianli(root);

printf("\n");

int h = 0;

h = high(root);

printf("树的高度：%d",h);

return 0;

}

无输入

输出： 递归先序：1 2 4 5 6 7 3 8 9 10 11

非递归先序：1 2 4 5 6 7 3 8 9 10 11

递归中序：2 6 5 7 4 1 9 10 8 11 3

非递归后序：2 6 5 7 4 1 9 10 8 11 3

递归后序：6 7 5 4 2 10 9 11 8 3 1

非递归后序：6 7 5 4 2 10 9 11 8 3 1

层次遍历：1 2 3 4 8 5 9 11 6 7 10

树的高度：5

反思：这段代码是在代码中创建树结构的，而不是通过文件或者键盘输入，可以改进这一点。