

**学生实验实习报告册**

|  |  |
| --- | --- |
| 学年学期： | 2022 -2023学年🞎春🗹秋学期 |
| 课程名称： | 数据结构实验 |
| 学生学院： | 软件工程 |
| 专业班级： | 13002106 |
| 学生学号： | 2021214257 |
| 学生姓名： | 张俊涛 |
| 联系电话： | 17784298305 |

**重庆邮电大学教务处制**

## 实验一、二叉树的基本算法

### 实验目的

1．理解二叉树的逻辑结构；

2．理解二叉树的存储结构特点，掌握二叉树的存储分配要点；

3．掌握二叉树的基本操作及递归实现，深刻领会二叉树遍历操作的非递归实现。

### 二、使用仪器、材料

Vs2019,笔记本电脑

### 三、实验内容

1. 二叉链表的存储结构

（1）二叉链表结点的存储结构：（私有）数据封装数据元素data、指向左孩子结点指针lchild、指向右孩子结点指针rchild。

（2）二叉链表的存储结构：数据（仅）封装二叉树根结点的指针root。

2. 二叉链表的基本操作

（1）初始化二叉链表：理解扩展二叉树及其前序遍历序列；

（2）前序遍历二叉链表（递归实现）；

（3）中序遍历二叉链表（递归实现）；

（4）后序遍历二叉链表（递归实现）；

（5）层序遍历二叉链表；

（6）销毁二叉链表（递归实现）。

### 四、实验步骤及过程记录

对于二叉树，我采用了头文件和源文件分开写的思想。先定义各自的结构体如下，树：struct tree

{

int data;

tree\* Lnext;

tree\* Rnext;

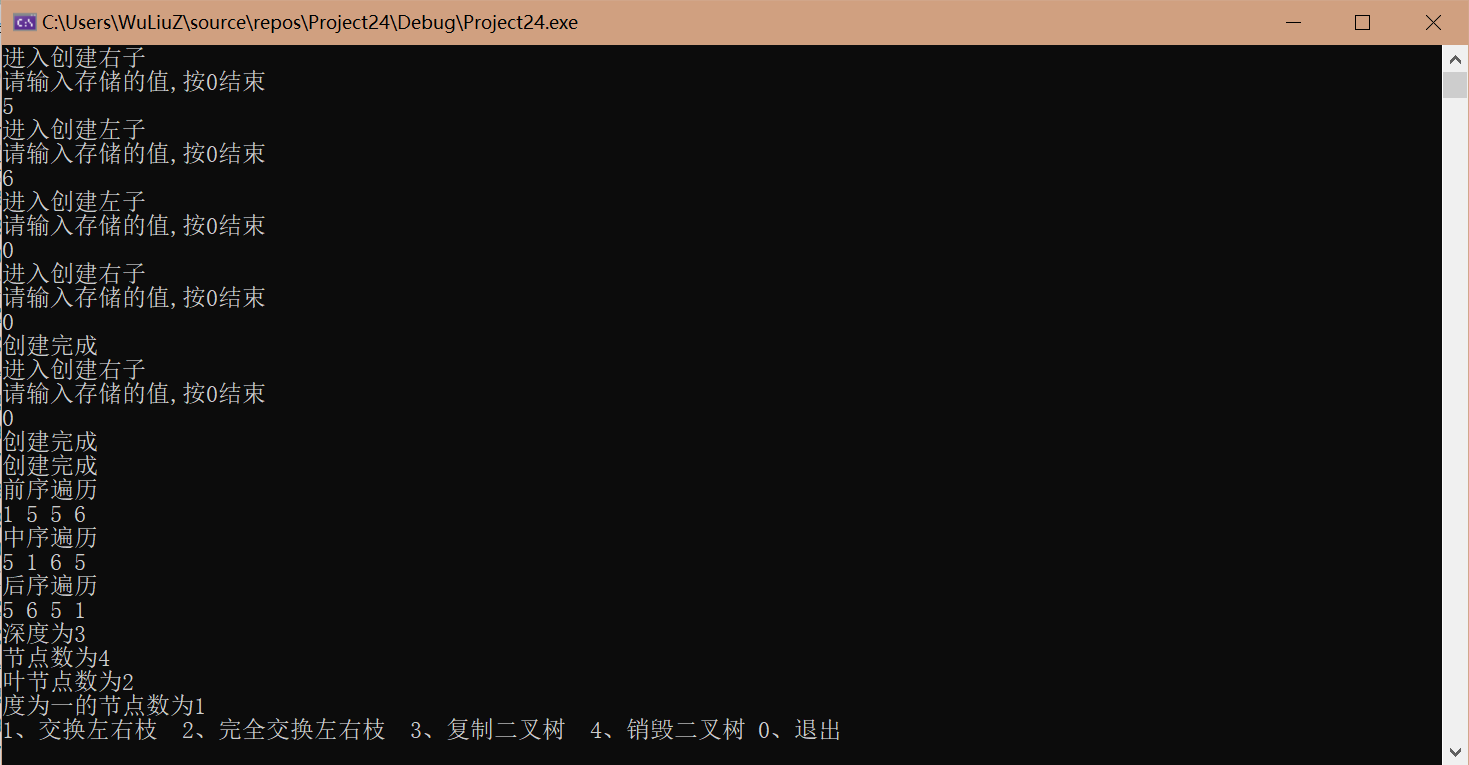
};

定义出结构体之后，将他们放在各自的头文件中然后再于头文件中书写实现各自功能如：初始化、前中后遍历序遍历、层序遍历、销毁。

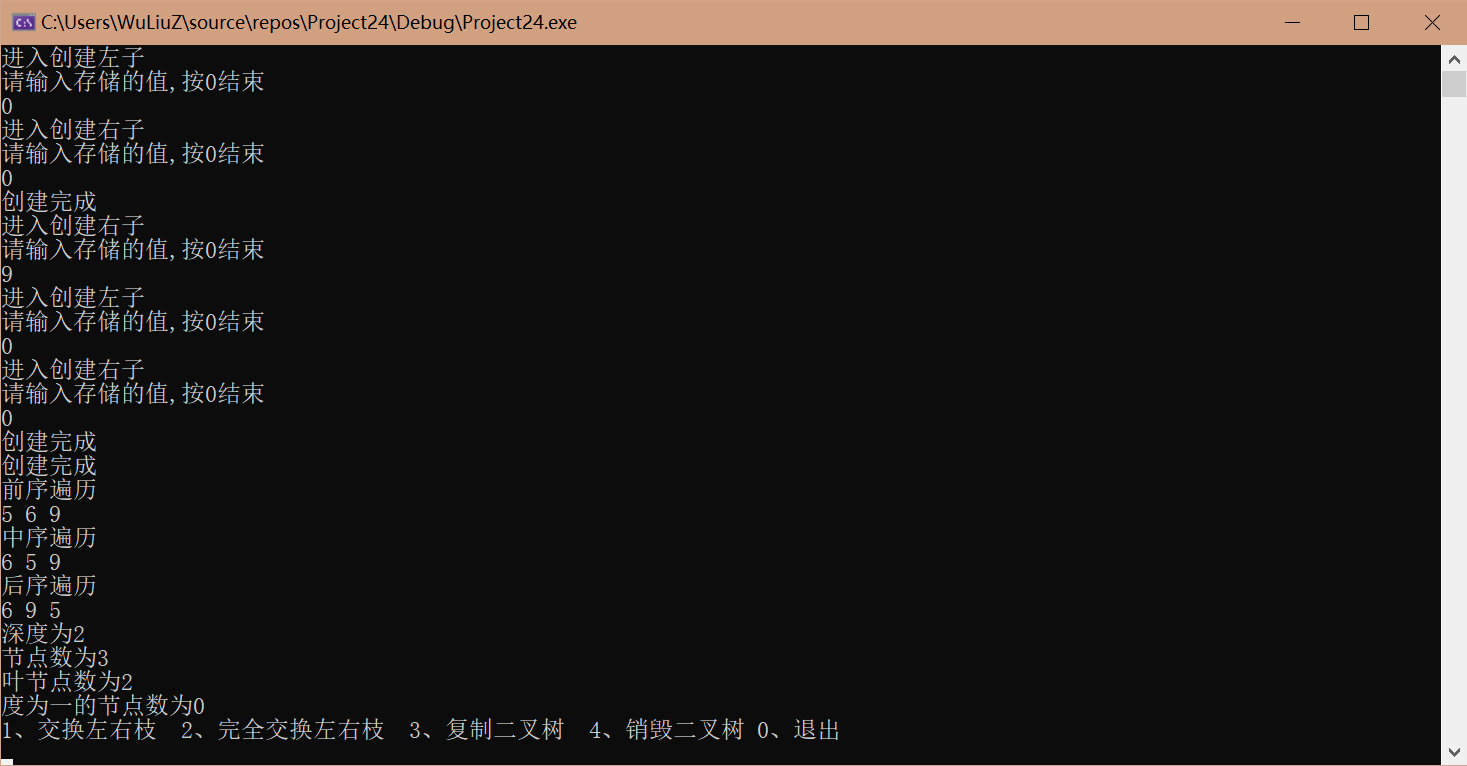
然后再于源文件中写出主函数，并调试二叉树的功能：





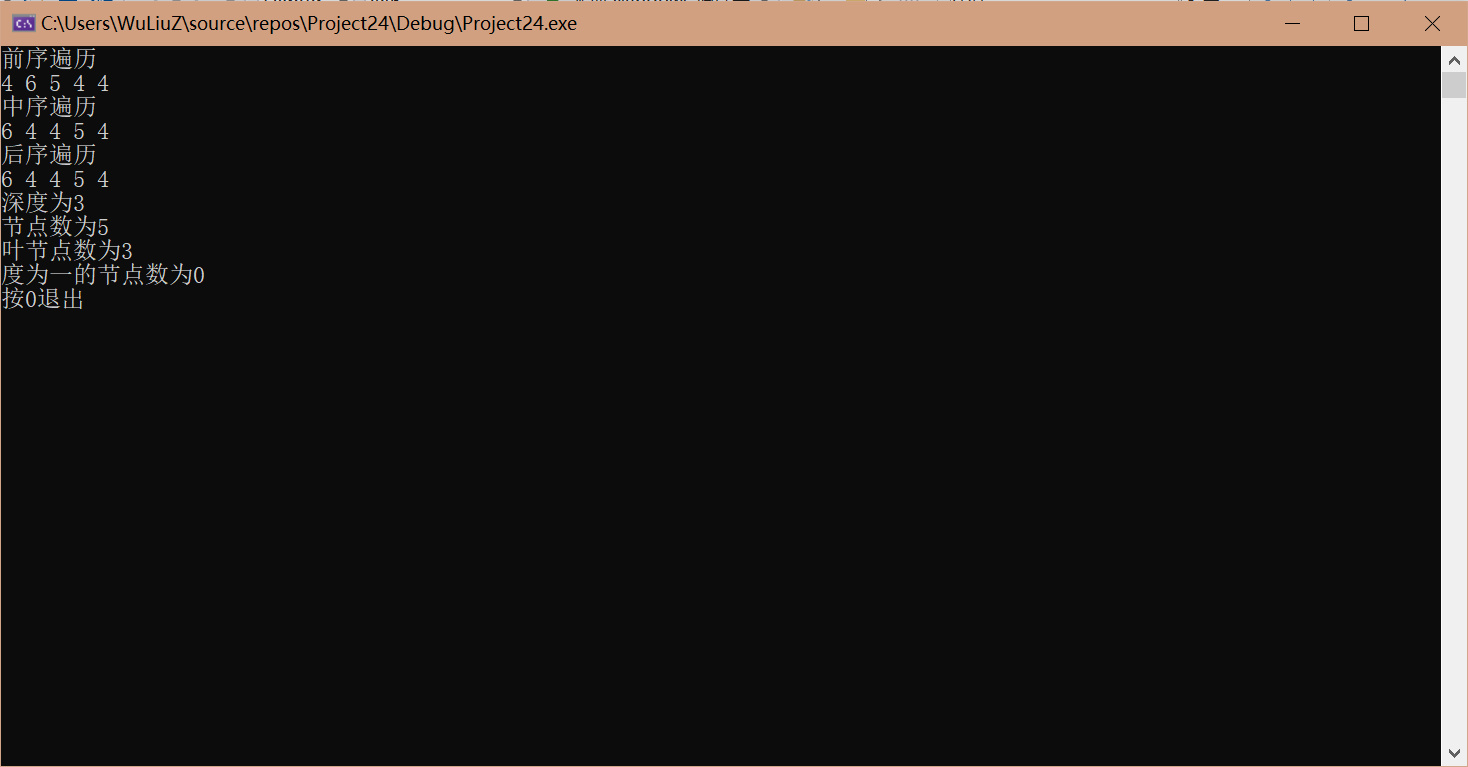


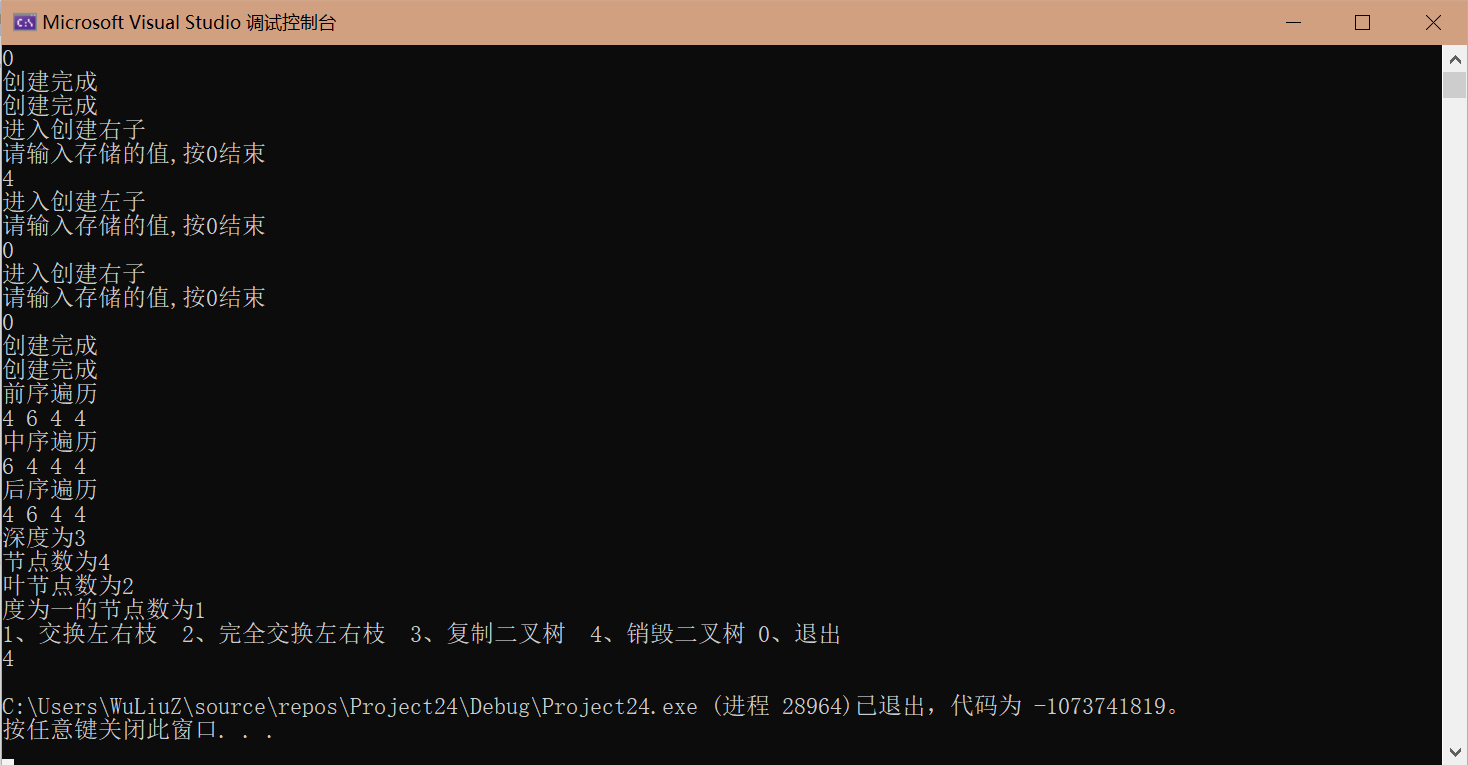












对于二叉树的时间复杂度和算法思想如下：

1. 递归实现：O(n)

2. 非递归实现：

（1）前序遍历:O(n)

step1：只要当前指针不空，访问当前指针所指结点中数据元素后将当前指针入栈，然后将当前指针修改为其左孩子指针；

step2：重复step1直到遇到空指针；

step3：栈空则结束；栈不空，从栈中弹出指针，将其右孩子指针作为当前指针，转step1。

（2）中序遍历:O(n)

step1：只要当前指针不空，将当前指针入栈，后然后将当前指针修改为其左孩子指针；

step2：重复step1直到遇到空指针；

step3：栈空则结束；栈不空，从栈中弹出指针，访问该指针所指结点中数据元素，然后将其右孩子指针作为当前指针，转step1。

（3）后序遍历：O(n)

栈结构中增加flag域

step1：只要当前指针不空，将当前指针入栈（flag置1），然后将当前指针修改为其左孩子指针；

step2：重复step1直到遇到空指针；

step3：栈空则结束；若栈不空

step3.1从栈中弹出指针，若flag为1，将flag改为2后将当前指针重新入栈，然后将其右孩子指针作为当前指针，转step1；

step3.2若flag为2，访问该指针所指结点中数据元素，转step3

### 五、实验收获与体会

通过二叉树基本操作实验，我更加理解二叉树的逻辑结构，以及二叉树的存储结构特点，巩固和掌握了二叉树的存储分配要点，此外还理解掌握二叉树的基本操作及递归实现，深刻体会二叉树遍历操作的非递归实现。

程序附件清单主函数：

#include <iostream>

#include"Doubletree.h"

using namespace std;

void main()

{

//tree\* s = new tree;

tree\* s = (tree\*)malloc(sizeof(tree));

tree\* a; //声明

cout << "注意：输入数字必须为整数！"<<endl;

CreateTree(a);

cout << "前序遍历" << endl;

Qian(a);

cout << endl;//分行

cout << "中序遍历" << endl;

Zhong(a);

cout << endl;

cout << "后序遍历" << endl;

Hou(a);

cout << endl;

cout << "深度为" << ShenDu(a) << endl;

cout << "节点数为" << JieDian(a) << endl;

cout << "叶节点数为" << LeafPTree(a) << endl;

cout << "度为一的节点数为" << DTree(a) << endl;

int select; //选择判断

int flag1 = 1;

cout << "1、交换左右枝 2、完全交换左右枝 3、复制二叉树 4、销毁二叉树 0、退出" << endl;

cin >> select;

while (flag1)

{

switch (select)

{

case 1:

system("CLS");//清屏操作

cout << "交换左右枝" << endl;

Exchange(a);

cout << "前序遍历" << endl;

Qian(a);

cout << endl;

cout << "中序遍历" << endl;

Zhong(a);

cout << endl;

cout << "后序遍历" << endl;

Hou(a);

cout << endl;

cout<<”层序遍历<<endl;

Layer(a, JieDian(a))”;

cout << "深度为" << ShenDu(a) << endl;

cout << "节点数为" << JieDian(a) << endl;

cout << "叶节点数为" << LeafPTree(a) << endl;

cout << "度为一的节点数为" << DTree(a) << endl;

break;

case 2:

system("CLS");//清屏操作

cout << "完全交换左右枝" << endl;

AllExchange(a);

cout << "前序遍历" << endl;

Qian(a);

cout << endl;

cout << "中序遍历" << endl;

Zhong(a);

cout << endl;

cout << "后序遍历" << endl;

Hou(a);

cout << endl;

cout<<”层序遍历<<endl;

Layer(a, JieDian(a))”;

cout << "深度为" << ShenDu(a) << endl;

cout << "节点数为" << JieDian(a) << endl;

cout << "叶节点数为" << LeafPTree(a) << endl;

cout << "度为一的节点数为" << DTree(a) << endl;

break;

case 3:

Copy(a, s);

system("CLS");//清屏操作

cout << "前序遍历" << endl;

Qian(s);

cout << endl;

cout << "中序遍历" << endl;

Zhong(s);

cout << endl;

cout << "后序遍历" << endl;

Hou(s);

cout << endl;

cout<<”层序遍历<<endl;

Layer(a, JieDian(a));

cout << "深度为" << ShenDu(s) << endl;

cout << "节点数为" << JieDian(s) << endl;

cout << "叶节点数为" << LeafPTree(s) << endl;

cout << "度为一的节点数为" << DTree(s) << endl;

break;

case 4:

Destroy(a);

cout << "销毁完毕！";

default:

cout << "退出" << endl;

break;

}

cout << "按0退出" << endl;

cin >> flag1;

}

}

头文件：

#pragma once

#ifndef DOUBLETREE

#define DOUBLETREE

struct tree

{

int data;

tree\* Lnext;

tree\* Rnext;

};

void CreateTree(tree\*& t);

void Qian(tree\* t);

void Zhong(tree\* t);

void Hou(tree\* t);

int ShenDu(tree\* t);

int JieDian(tree\* t);

int LeafPTree(tree\* t);

int DTree(tree\* t);

void Exchange(tree\* t);

void AllExchange(tree\* t);

tree\* Copy(tree\* S, tree\* s);

void Destroy(tree\* S);

void Layer(tree\* t,int a) ;

#endif

实现文件；

#include <iostream>

#include"Doubletree.h"

using namespace std;

//创建节点

void CreateTree(tree\*& t)

{

t = (tree\*)malloc(sizeof(tree));

int e;

cout << "请输入存储的值,按0结束" << endl;

cin >> e;

if (e == 0)

{

t = NULL;

return;

}

t->data = e;

cout << "进入创建左子" << endl;

CreateTree(t->Lnext);

cout << "进入创建右子" << endl;

CreateTree(t->Rnext);

cout << "创建完成" << endl;

return;

}

//前序遍历

void Qian(tree\* t)

{

if (t == NULL)

{

return;

}

cout << t->data << " ";

Qian(t->Lnext);

Qian(t->Rnext);

return;

}

//中序遍历

void Zhong(tree\* t)

{

if (t == NULL)

{

return;

}

Zhong(t->Lnext);

cout << t->data << " ";

Zhong(t->Rnext);

return;

}

//后序遍历

void Hou(tree\* t)

{

if (t == NULL)

{

return;

}

Hou(t->Lnext);

Hou(t->Rnext);

cout << t->data << " ";

return;

}

//深度计算

int ShenDu(tree\* t)

{

if (t == NULL)

{

return 0;

}

int L = ShenDu(t->Lnext);

int R = ShenDu(t->Rnext);

if (L >= R)

{

return L + 1;

}

else if (R > L)

{

return R + 1;

}

}

//节点

int JieDian(tree\* t)

{

if (t == NULL)

{

return 0;

}

int m = JieDian(t->Lnext);

m += JieDian(t->Rnext);

return m + 1;

}

//叶节点

int LeafPTree(tree\* t)

{

if (t == NULL)

{

return 0;

}

int m = LeafPTree(t->Lnext);

m += LeafPTree(t->Rnext);

if (t->Lnext == NULL && t->Rnext == NULL)

{

m = 1;

}

return m;

}

//度为一的节点数

void Layer(tree\* t, int a)

{

tree\*\* s = new tree \* [a];

int head = 0;

int tear = 0;

int\* s1 = new int[a];

s[tear] = t;

tear++;

while ((tear - head) != 0)

{

if (s[head]->Lnext)

{

s[tear] = s[head]->Lnext;

tear++;

}

if (s[head]->Rnext)

{

s[tear] = s[head]->Rnext;

tear++;

}

s1[head] = s[head]->data;

head++;

}

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

{

cout << s1[i] << " ";

}

}

int DTree(tree\* t)

{

if (t == NULL)

{

return 0;

}

int m = DTree(t->Lnext);

m += DTree(t->Rnext);

if ((t->Lnext != NULL && t->Rnext == NULL) || (t->Lnext == NULL && t->Rnext != NULL))

{

m = 1;

}

return m;

}

//交换左右枝

void Exchange(tree\* t)

{

tree\* p = t->Lnext;

t->Lnext = t->Rnext;

t->Rnext = p;

}

//完全交换左右枝

void AllExchange(tree\* t)

{

if (t == NULL)

{

return;

}

tree\* p = t->Lnext;

t->Lnext = t->Rnext;

t->Rnext = p;

AllExchange(t->Lnext);

AllExchange(t->Rnext);

}

//左右枝的枝的枝的......再交换

tree\* Copy(tree\* S,tree\* s) {

if (S == NULL) {

cout << "原树为空" << endl;

return NULL;

}

s->data = S->data;

s->Lnext = S->Lnext;

s->Rnext = S->Rnext;

Copy(S->Lnext, s->Lnext);

Copy(S->Rnext, s->Rnext);

}

//复制树

void Destroy(tree\* S) {

S->data = 0;

free (S);

Destroy(S->Lnext);

Destroy(S->Rnext);

}

//递归销毁树

头文件；

#pragma once

#ifndef DOUBLETREE

#define DOUBLETREE

struct tree

{

int data;

tree\* Lnext;

tree\* Rnext;

};

void CreateTree(tree\*& t);

void Qian(tree\* t);

void Zhong(tree\* t);

void Hou(tree\* t);

int ShenDu(tree\* t);

int JieDian(tree\* t);

int LeafPTree(tree\* t);

int DTree(tree\* t);

void Exchange(tree\* t);

void AllExchange(tree\* t);

tree\* Copy(tree\* S, tree\* s);

void Destroy(tree\* S);

void Layer(tree\*t,int a);

#endif