**实验4 二叉树**

学号：2022217587 姓名：党存远 专业：物联网工程

|  |  |  |  |
| --- | --- | --- | --- |
| **“数据结构课程实验”验收细则** | | | |
| **成绩等级** | **具体表现** | **教师评分**  **（百分制）** | |
| A（100-90] | 1）能较快完成实验；2）编码正确；3）能完整正确地回答教师提问； | □ |  |
| B（90-80] | 1）能够在规定时间内完成实验；2）编码正确；3）能正确地回答教师提问； | □ |  |
| C（80-70] | 1）能够在规定时间内完成实验； | □ |  |
| D（70-60] | 1）虽完成实验但未能在规定时间内完成验收的； | □ |  |
| E（<60） | 1）未能进行验收的； | □ |  |

**4.1 实验目的**

掌握二叉树的动态链表存储结构及表示。

掌握二叉树的三种遍历算法（递归和非递归两类）。

运用二叉树三种遍历的方法求解有关问题。

**4.2 实验要求**

设计二叉树的二叉链表存储结构，编写算法实现下列问题的求解。

<1>打印出二叉树的三种遍历序列。

<2>设计算法按中序次序输出二叉树中各结点的值及其所对应的层次数。

<3>求二叉树的高度。

<4>求二叉树的结点数。

<5>求二叉树的叶子结点数。

<6>求二叉树的度为2的结点数。

<7>键盘输入一个元素x，求其父节点、兄弟结点、子结点的值，不存在时给出相应提示信息。对兄弟结点和孩子结点，存在时要明确指出是左兄弟、左孩子、右兄弟或右孩子。

<8>键盘输入一个元素x，求其在树中的层次，不存在时给出相应提示信息。

<9>将按顺序方式存储在数组中的二叉树转换为二叉链表形式。（数组中要扩展为完全二叉树）。

<10>交换二叉树中每个结点的左右孩子指针的值。（即：左子树变为右子树，右子树变为左子树）。

（下面为选做实验，有兴趣的同学完成）

<11>复制一棵二叉树T到T1。

<12>输出二叉树从每个叶子结点到根结点的路径（经历的结点）。

<13>对二叉链表表示的二叉树，按从上到下，从左到右打印结点值，即按层次打印。（提示：需要使用队列）

<14>对二叉链表表示的二叉树，求2个结点最近的共同祖先。

实验测试数据基本要求：

<15>求二叉树中一条最长的路径长度（边数），并输出路径上的个结点值。

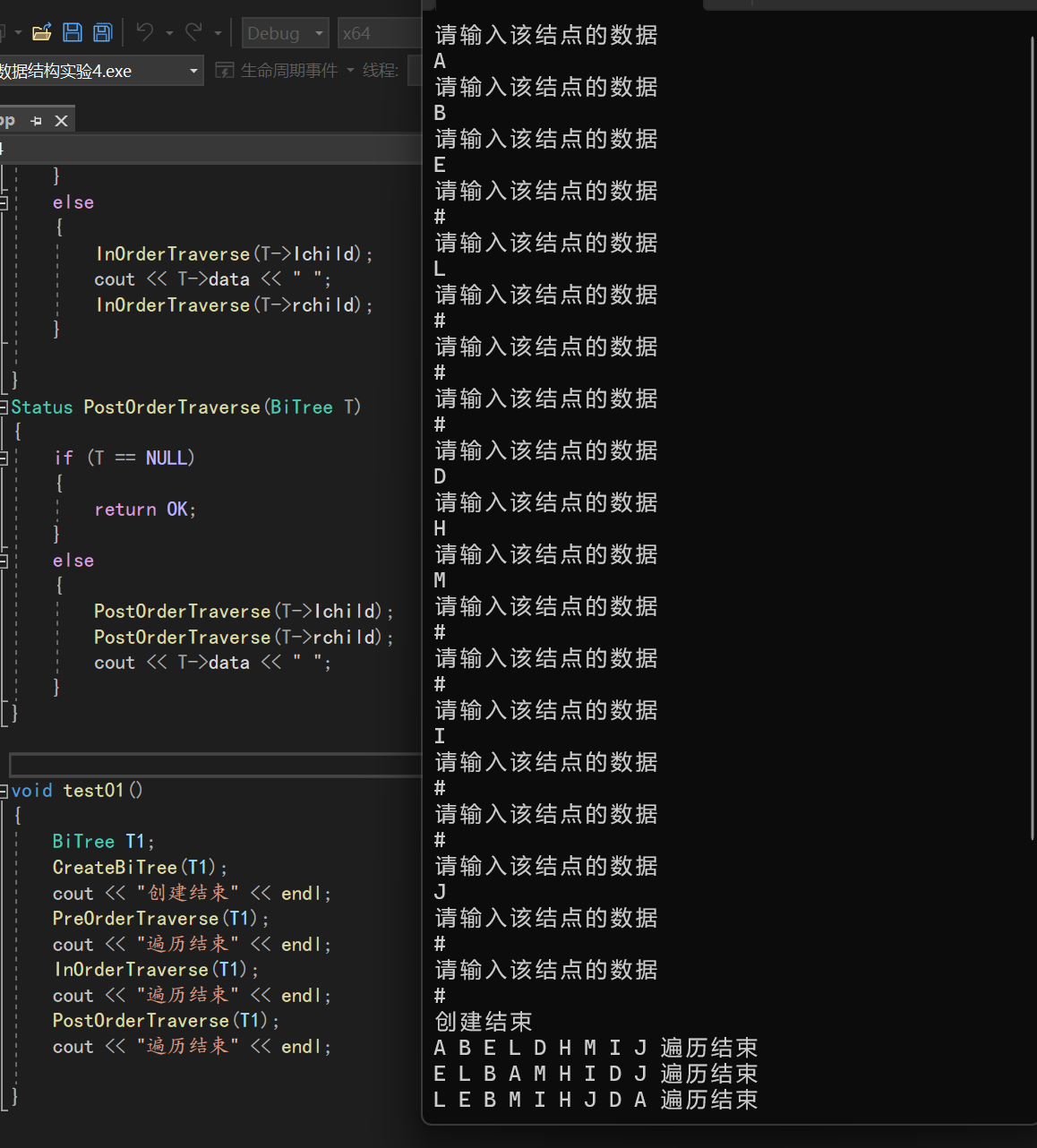
实验测试数据基本要求：

**4.3 实验数据要求**

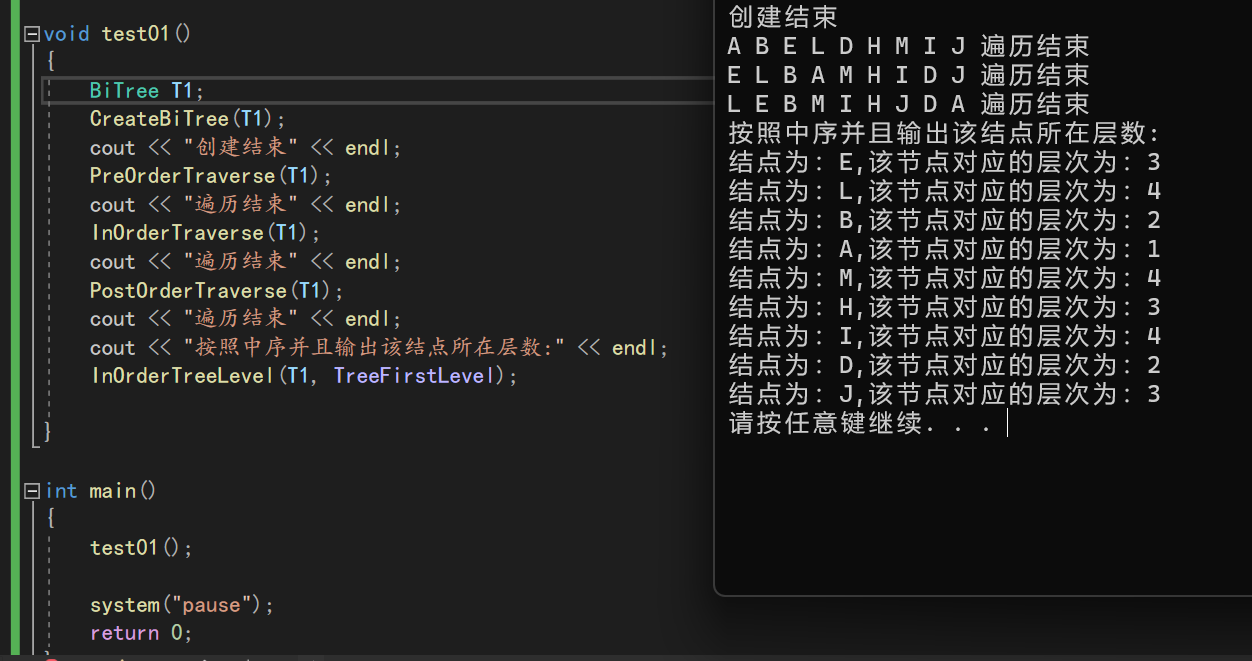
自我编写测试样例，要求每个功能函数的测试样例不少于两组

**4.4 运行结果截图及说明**

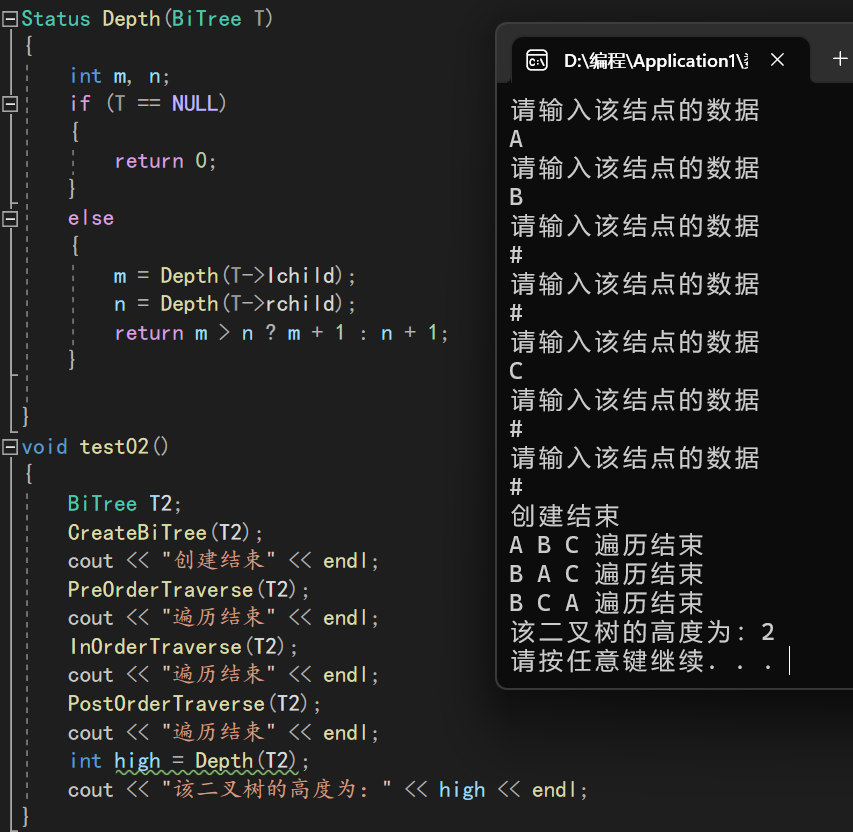
<1>打印出二叉树的三种遍历序列：

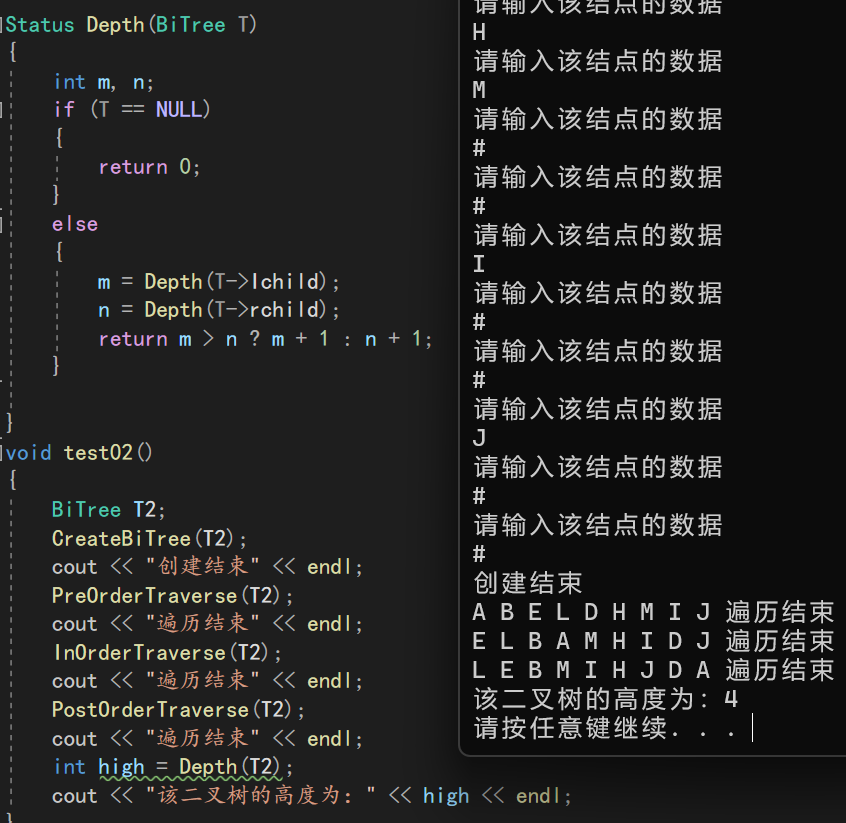


<2>设计算法按中序次序输出二叉树中各结点的值及其所对应的层次数：

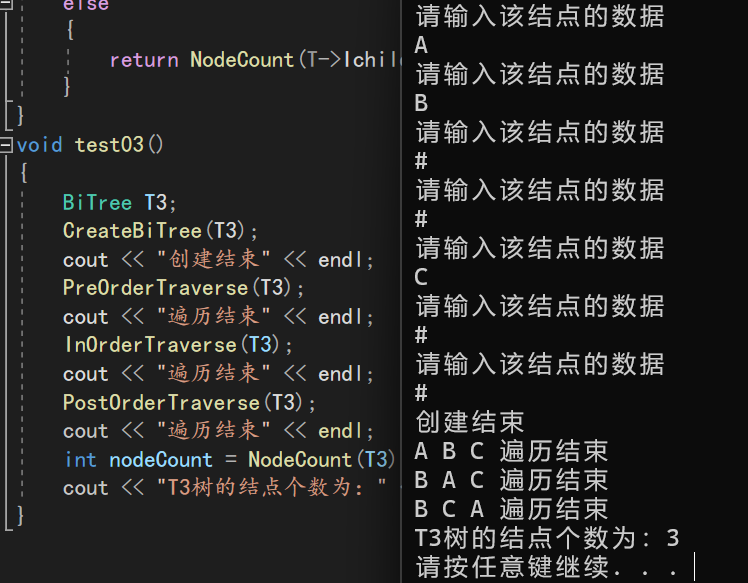


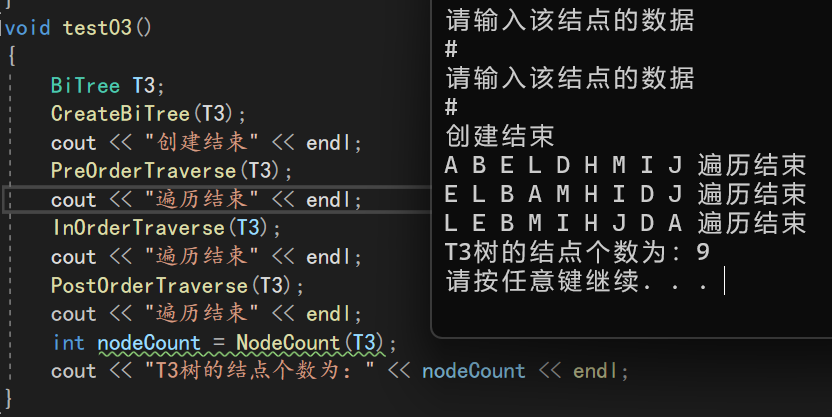
<3>求二叉树的高度：



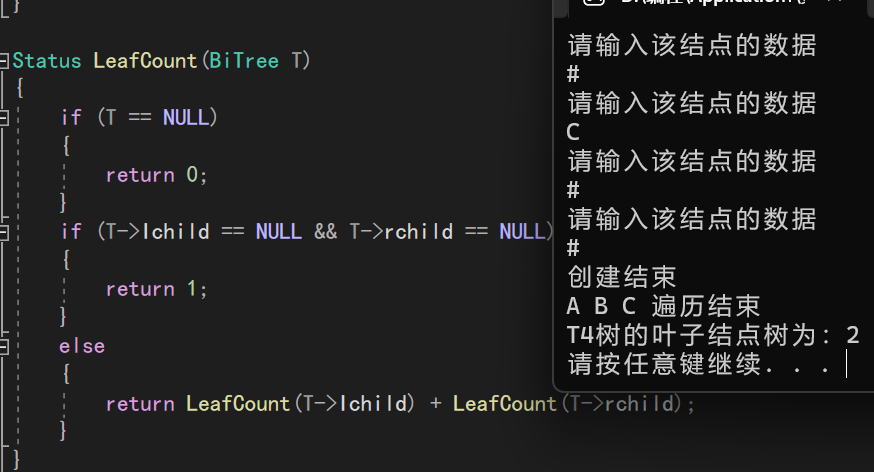


<4>求二叉树的结点数：



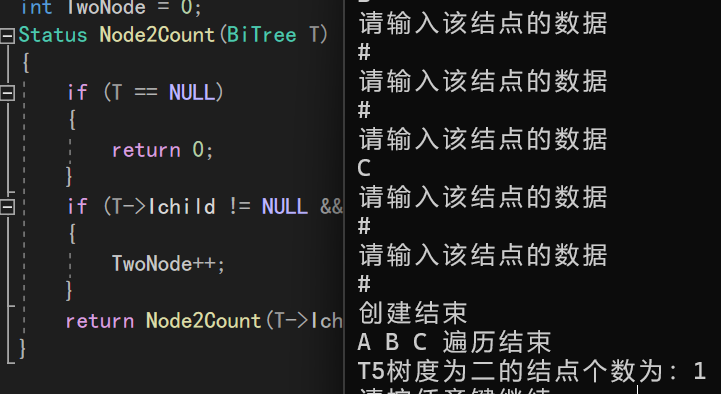


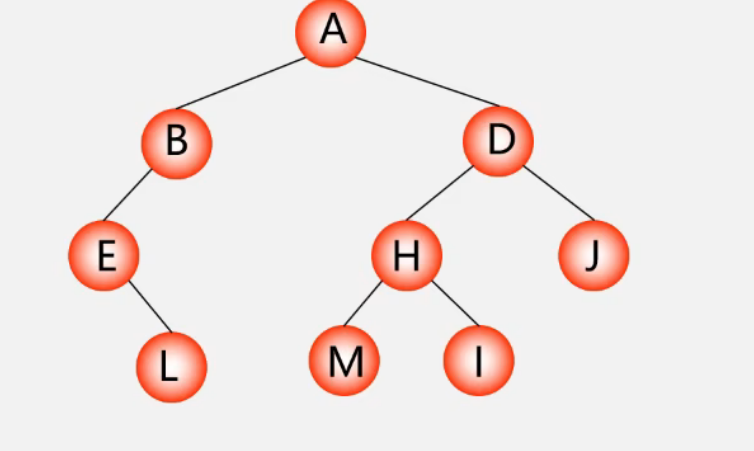
<5>求二叉树的叶子结点数：



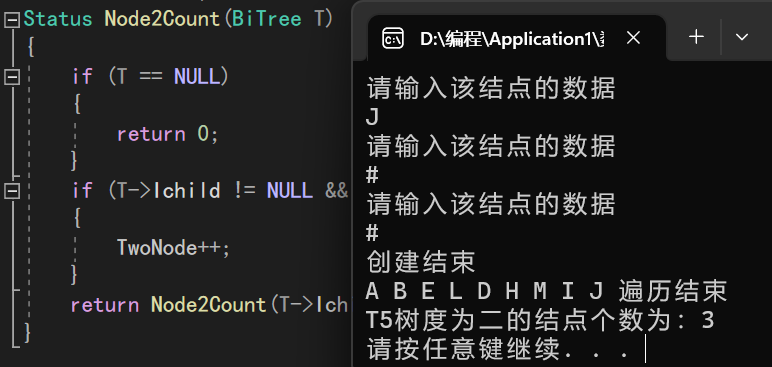


<6>求二叉树的度为2的结点数：

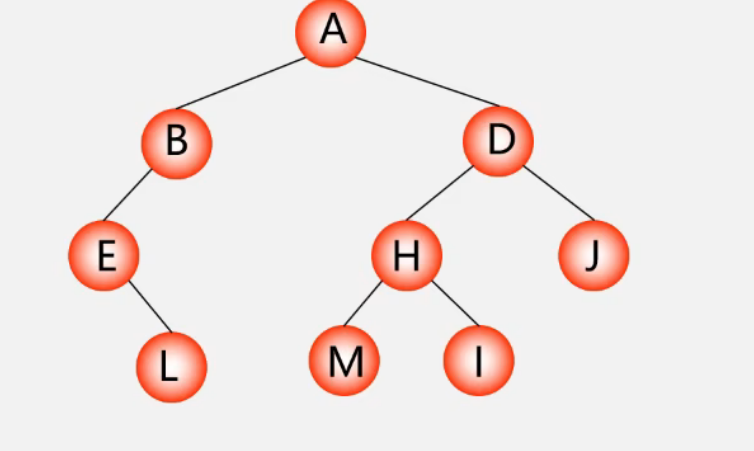




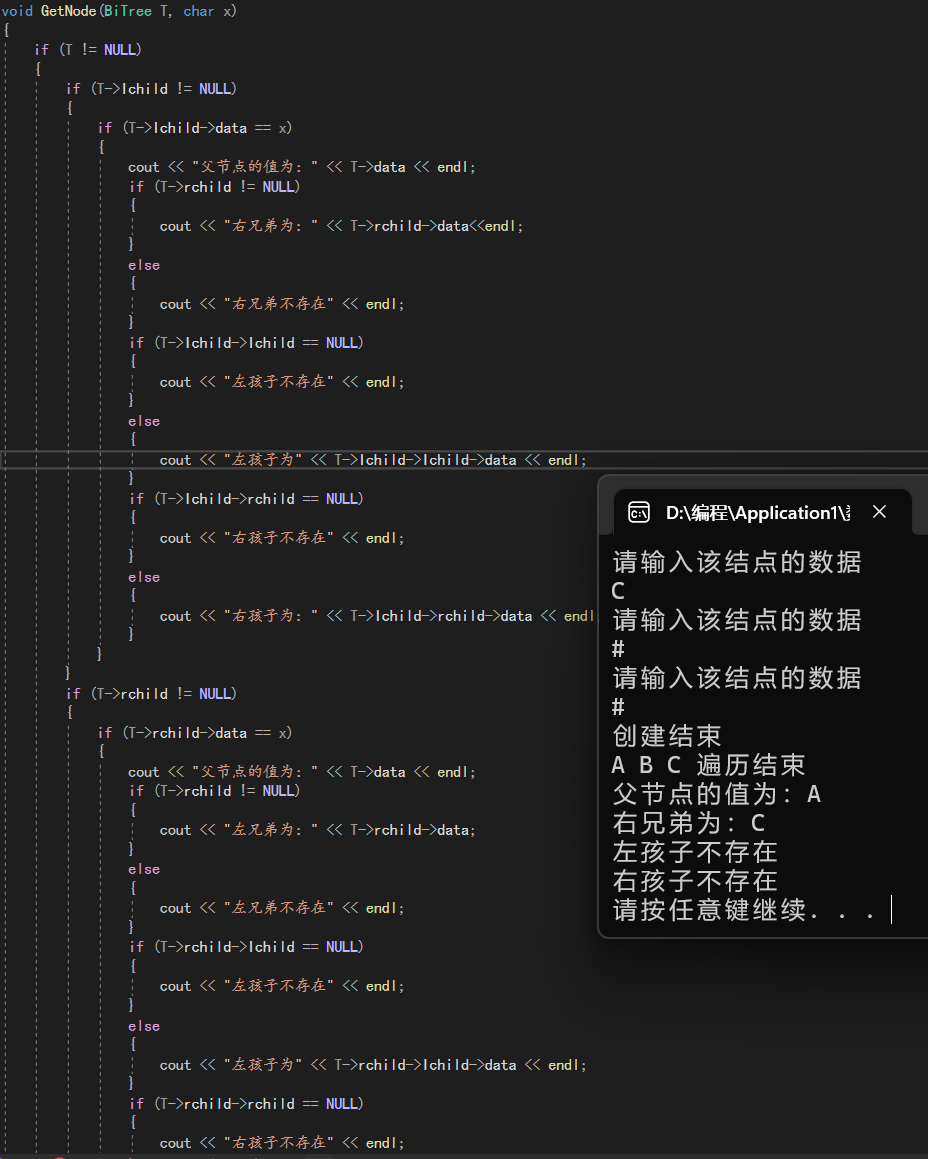
验证度为2的结点数为3

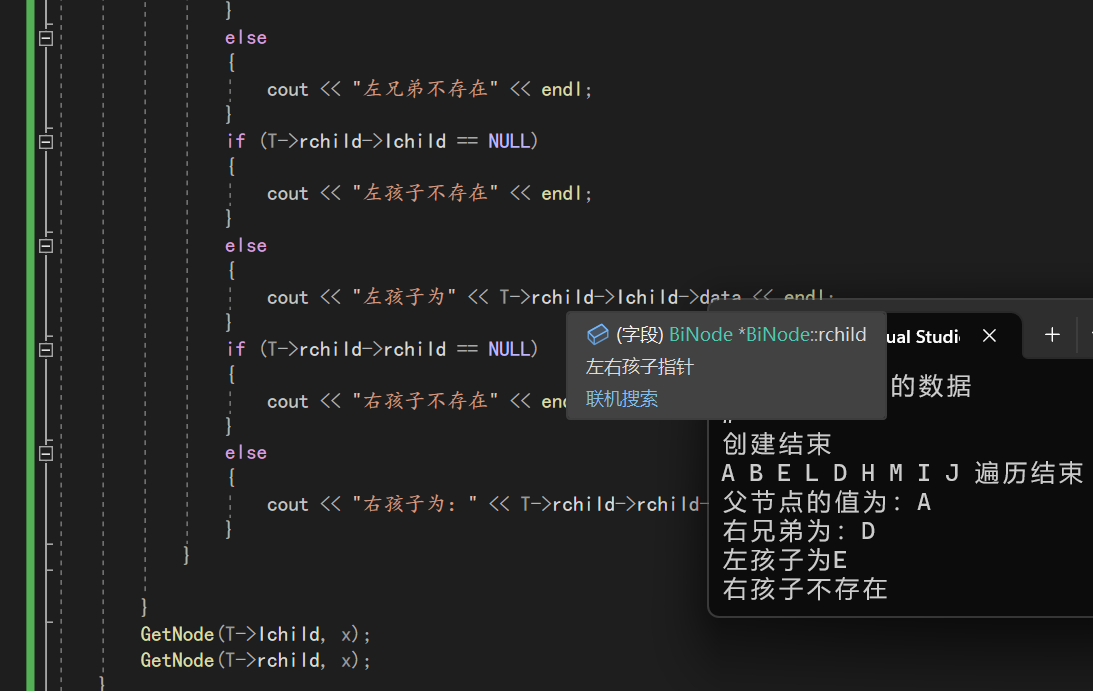


<7>键盘输入一个元素x，求其父节点、兄弟结点、子结点的值，不存在时给出相应提示信息。对兄弟结点和孩子结点，存在时要明确指出是左兄弟、左孩子、右兄弟或右孩子：



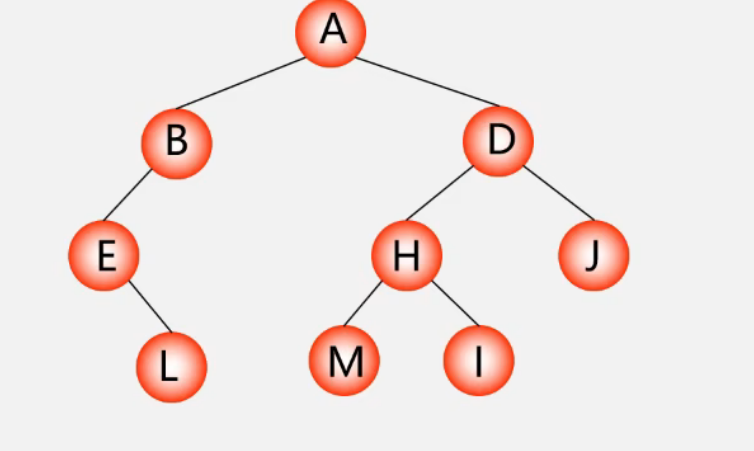
验证数据二中B结点的父亲兄弟孩子





<8>键盘输入一个元素x，求其在树中的层次，不存在时给出相应提示信息：





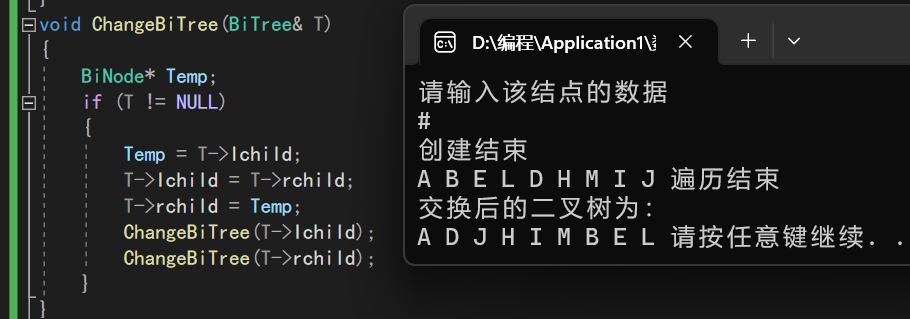
验证结果如图：



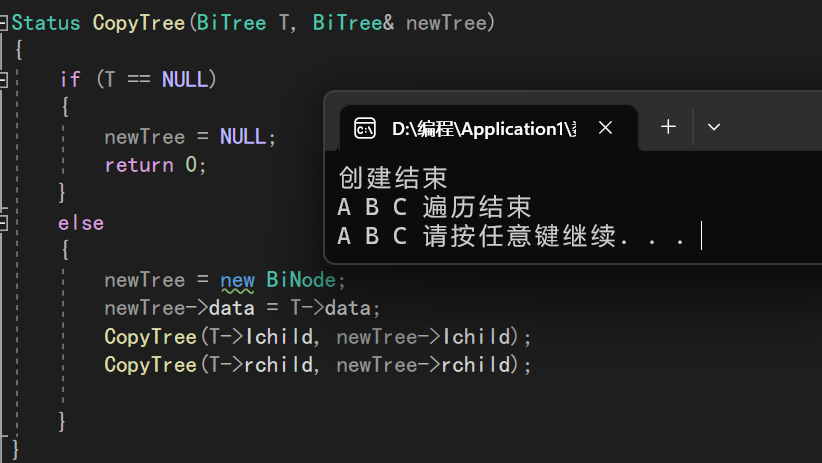
<9>将按顺序方式存储在数组中的二叉树转换为二叉链表形式。（数组中要扩展为完全二叉树）：



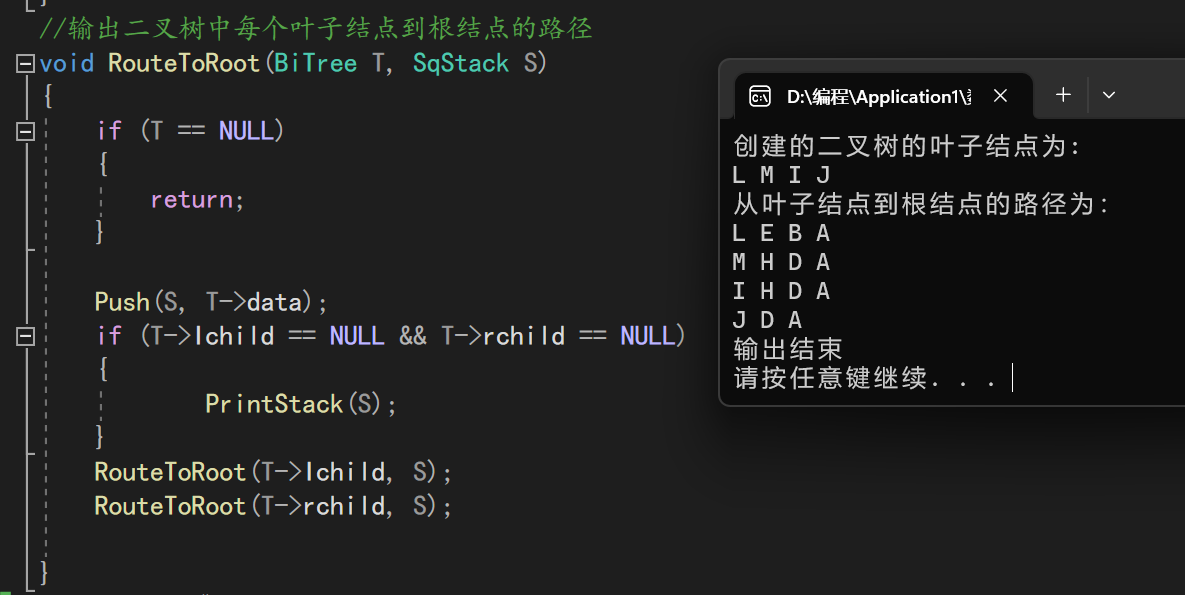
<10>交换二叉树中每个结点的左右孩子指针的值。（即：左子树变为右子树，右子树变为左子树）：



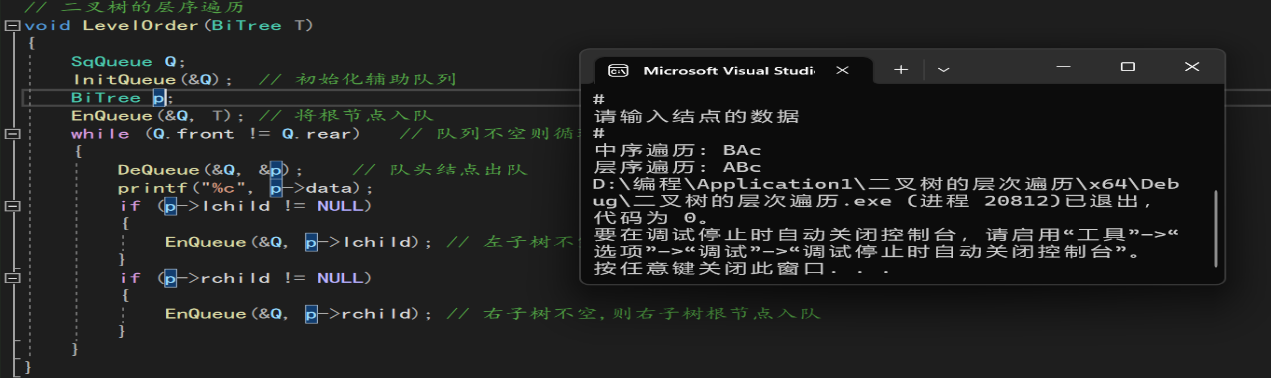
<11>复制一棵二叉树T到T1：



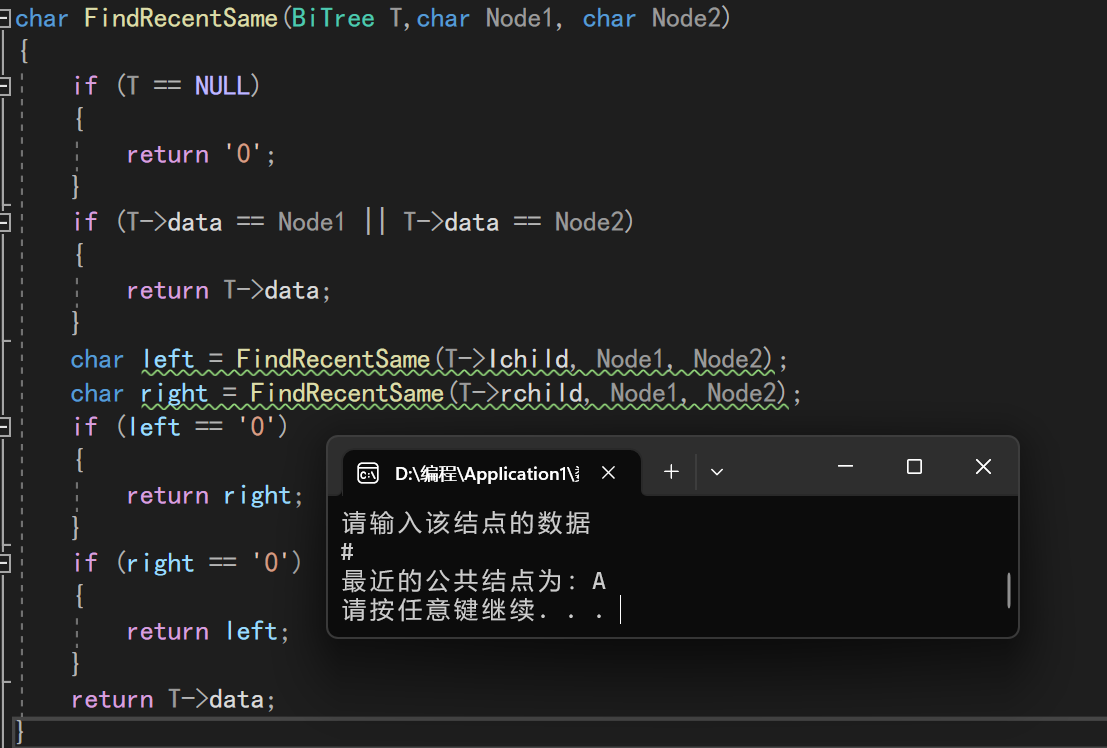
<12>输出二叉树从每个叶子结点到根结点的路径（经历的结点）：



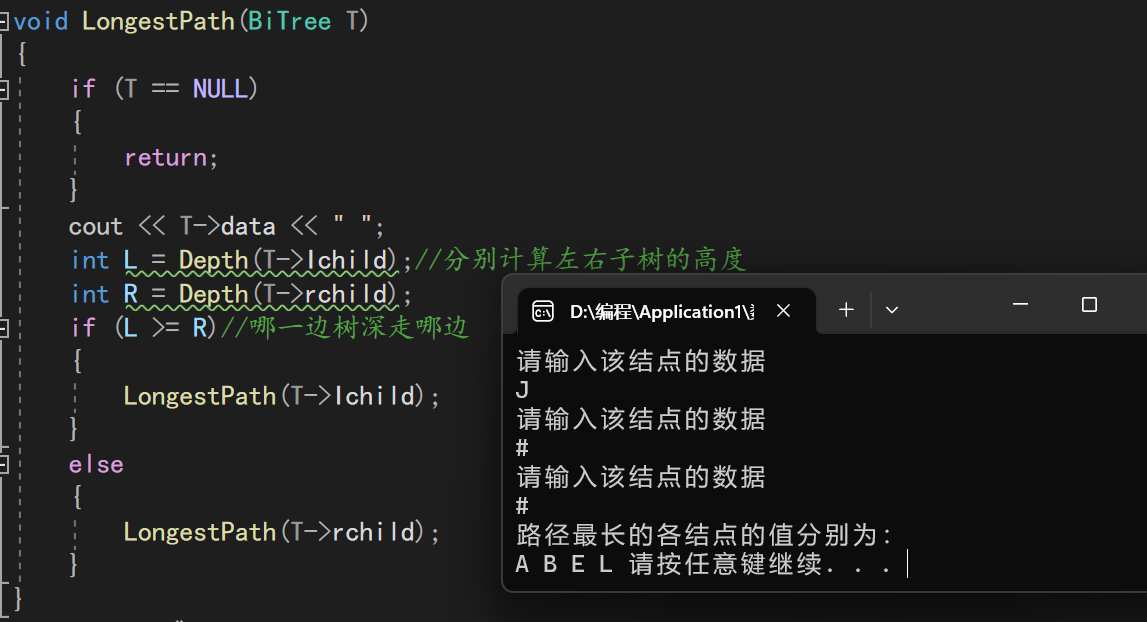
<13>对二叉链表表示的二叉树，按从上到下，从左到右打印结点值，即按层次打印：



<14>对二叉链表表示的二叉树，求2个结点最近的共同祖先：



<15>求二叉树中一条最长的路径长度（边数），并输出路径上的个结点值：



**4.5 附源代码**

<1>打印出二叉树的三种遍历序列：

Status CreateBiTree(BiTree& T)

{

char ch = '0';

cout << "请输入该结点的数据" << endl;

cin >> ch;

if (ch == '#')

{

T = NULL;

}

else

{

T = new BiNode;

if (T != NULL)

{

T->data = ch;

}

else

{

cout << "结点创建失败" << endl;

exit(OVERFLOW);

}

CreateBiTree(T->Ichild);

CreateBiTree(T->rchild);

}

return OK;

}

Status PreOrderTraverse(BiTree T)//先序遍历

{

if (T == NULL)

{

return OK;

}

else

{

cout << T->data << " ";

PreOrderTraverse(T->Ichild);

PreOrderTraverse(T->rchild);

}

}

Status InOrderTraverse(BiTree T)

{

if (T == NULL)

{

return OK;

}

else

{

InOrderTraverse(T->Ichild);

cout << T->data << " ";

InOrderTraverse(T->rchild);

}

}

Status PostOrderTraverse(BiTree T)

{

if (T == NULL)

{

return OK;

}

else

{

PostOrderTraverse(T->Ichild);

PostOrderTraverse(T->rchild);

cout << T->data << " ";

}

}

<2>设计算法按中序次序输出二叉树中各结点的值及其所对应的层次数：

Status InOrderTreeLevel(BiTree T, int n)//n表示第一个结点所对应的层次

{

if (T != NULL)

{

InOrderTreeLevel(T->Ichild, n + 1);

cout << "结点为：" << T->data << ",该节点对应的层次为：" << n << endl;

InOrderTreeLevel(T->rchild, n + 1);

}

return OK;

}

<3>求二叉树的高度：

Status Depth(BiTree T)

{

int m, n;

if (T == NULL)

{

return 0;

}

else

{

m = Depth(T->Ichild);

n = Depth(T->rchild);

return m > n ? m + 1 : n + 1;

}

}

void test02()

{

BiTree T2;

CreateBiTree(T2);

cout << "创建结束" << endl;

PreOrderTraverse(T2);

cout << "遍历结束" << endl;

InOrderTraverse(T2);

cout << "遍历结束" << endl;

PostOrderTraverse(T2);

cout << "遍历结束" << endl;

int high = Depth(T2);

cout << "该二叉树的高度为：" << high << endl;

}

<4>求二叉树的结点数：

Status NodeCount(BiTree T)

{

if (T == NULL)

{

return 0;

}

else

{

return NodeCount(T->Ichild) + NodeCount(T->rchild) + 1;

}

}

void test03()

{

BiTree T3;

CreateBiTree(T3);

cout << "创建结束" << endl;

PreOrderTraverse(T3);

cout << "遍历结束" << endl;

InOrderTraverse(T3);

cout << "遍历结束" << endl;

PostOrderTraverse(T3);

cout << "遍历结束" << endl;

int nodeCount = NodeCount(T3);

cout << "T3树的结点个数为：" << nodeCount << endl;

}

<5>求二叉树的叶子结点数：

Status LeafCount(BiTree T)

{

if (T == NULL)

{

return 0;

}

if (T->Ichild == NULL && T->rchild == NULL)

{

return 1;

}

else

{

return LeafCount(T->Ichild) + LeafCount(T->rchild);

}

}

void test04()

{

BiTree T4;

CreateBiTree(T4);

cout << "创建结束" << endl;

PreOrderTraverse(T4);

cout << "遍历结束" << endl;

cout << "T4树的叶子结点树为：" << LeafCount(T4) << endl;

}

<6>求二叉树的度为2的结点数：

int TwoNode = 0;

Status Node2Count(BiTree T)

{

if (T == NULL)

{

return 0;

}

if (T->Ichild != NULL && T->rchild != NULL)

{

TwoNode++;

}

return Node2Count(T->Ichild) + Node2Count(T->rchild);

}

void test05()

{

BiTree T5;

CreateBiTree(T5);

cout << "创建结束" << endl;

PreOrderTraverse(T5);

cout << "遍历结束" << endl;

Node2Count(T5);

cout << "T5树度为二的结点个数为：" << TwoNode << endl;

}

<7>键盘输入一个元素x，求其父节点、兄弟结点、子结点的值，不存在时给出相应提示信息。对兄弟结点和孩子结点，存在时要明确指出是左兄弟、左孩子、右兄弟或右孩子：

void GetNode(BiTree T, char x)

{

if (T != NULL)

{

if (T->Ichild != NULL)

{

if (T->Ichild->data == x)

{

cout << "父节点的值为：" << T->data << endl;

if (T->rchild != NULL)

{

cout << "右兄弟为：" << T->rchild->data<<endl;

}

else

{

cout << "右兄弟不存在" << endl;

}

if (T->Ichild->Ichild == NULL)

{

cout << "左孩子不存在" << endl;

}

else

{

cout << "左孩子为" << T->Ichild->Ichild->data << endl;

}

if (T->Ichild->rchild == NULL)

{

cout << "右孩子不存在" << endl;

}

else

{

cout << "右孩子为：" << T->Ichild->rchild->data << endl;

}

}

}

if (T->rchild != NULL)

{

if (T->rchild->data == x)

{

cout << "父节点的值为：" << T->data << endl;

if (T->rchild != NULL)

{

cout << "左兄弟为：" << T->rchild->data;

}

else

{

cout << "左兄弟不存在" << endl;

}

if (T->rchild->Ichild == NULL)

{

cout << "左孩子不存在" << endl;

}

else

{

cout << "左孩子为" << T->rchild->Ichild->data << endl;

}

if (T->rchild->rchild == NULL)

{

cout << "右孩子不存在" << endl;

}

else

{

cout << "右孩子为：" << T->rchild->rchild->data << endl;

}

}

}

GetNode(T->Ichild, x);

GetNode(T->rchild, x);

}

}

<8>键盘输入一个元素x，求其在树中的层次，不存在时给出相应提示信息：

void GetNodeLevel(BiTree T, char x,int &flag,int Level=1)

{

if (T != NULL)

{

if (T->data == x)

{

cout << x << "所在的层数为" << Level << endl;

flag = 1;

}

GetNodeLevel(T->Ichild, x,flag, Level + 1);

GetNodeLevel(T->rchild, x,flag, Level + 1);

}

}

<9>将按顺序方式存储在数组中的二叉树转换为二叉链表形式。（数组中要扩展为完全二叉树）：

void ChangeArray(BiTree& T, Elemtype a[],int num,int i)//num为二叉树结点最后位置的有效编号，i为该节点在二叉树上的编号

{

if (i <= num && a[i] != '\0')

{

T = new BiNode;

if (a[i] != '0')

{

T->data = a[i];

}

T->Ichild = NULL;

T->rchild == NULL;

ChangeArray(T->Ichild, a, num, 2 \* i);

ChangeArray(T->rchild, a, num, 2 \* i + 1);

}

}

<10>交换二叉树中每个结点的左右孩子指针的值。（即：左子树变为右子树，右子树变为左子树）：

void ChangeBiTree(BiTree& T)

{

BiNode\* Temp;

if (T != NULL)

{

Temp = T->Ichild;

T->Ichild = T->rchild;

T->rchild = Temp;

ChangeBiTree(T->Ichild);

ChangeBiTree(T->rchild);

}

}

<11>复制一棵二叉树T到T1：

Status CopyTree(BiTree T, BiTree& newTree)

{

if (T == NULL)

{

newTree = NULL;

return 0;

}

else

{

newTree = new BiNode;

newTree->data = T->data;

CopyTree(T->Ichild, newTree->Ichild);

CopyTree(T->rchild, newTree->rchild);

}

}

<12>输出二叉树从每个叶子结点到根结点的路径（经历的结点）：

//输出二叉树中每个叶子结点到根结点的路径

void RouteToRoot(BiTree T, SqStack S)

{

if (T == NULL)

{

return;

}

Push(S, T->data);

if (T->Ichild == NULL && T->rchild == NULL)

{

PrintStack(S);

}

RouteToRoot(T->Ichild, S);

RouteToRoot(T->rchild, S);

}

<13>对二叉链表表示的二叉树，按从上到下，从左到右打印结点值，即按层次打印：

// 二叉树的层序遍历

void LevelOrder(BiTree T)

{

SqQueue Q;

InitQueue(&Q); // 初始化辅助队列

BiTree p;//表示出队的结点

EnQueue(&Q, T); // 将根节点入队

while (Q.front != Q.rear) // 队列不空则循环

{

DeQueue(&Q, &p); // 队头结点出队，出队哪一个结点，判断哪一个结点的左右子树

printf("%c", p->data);

if (p->lchild != NULL)

{

EnQueue(&Q, p->lchild); // 左子树不空,则左子树根节点入队

}

if (p->rchild != NULL)

{

EnQueue(&Q, p->rchild); // 右子树不空,则右子树根节点入队

}

}

}

<14>对二叉链表表示的二叉树，求2个结点最近的共同祖先：

char FindRecentSame(BiTree T,char Node1, char Node2)

{

if (T == NULL)

{

return '0';

}

if (T->data == Node1 || T->data == Node2)

{

return T->data;

}

char left = FindRecentSame(T->Ichild, Node1, Node2);

char right = FindRecentSame(T->rchild, Node1, Node2);

if (left == '0')

{

return right;

}

if (right == '0')

{

return left;

}

return T->data;

}

<15>求二叉树中一条最长的路径长度（边数），并输出路径上的个结点值：

void LongestPath(BiTree T)

{

if (T == NULL)

{

return;

}

cout << T->data << " ";

int L = Depth(T->Ichild);//分别计算左右子树的高度

int R = Depth(T->rchild);

if (L >= R)//哪一边树深走哪边

{

LongestPath(T->Ichild);

}

else

{

LongestPath(T->rchild);

}

**4.6 调试过程中出现的bug总结**

1、根据先序序列创建二叉树时一定要注意‘#’的输入不可以出现错误。

2、中序遍历二叉树并且输出结点所在层数的关键在于每次递归调用时，层数要加1，每次传入参数都是当前层的层数。

3、计算二叉树高度时，如果T为空则返回0,不为空递归调用分别 计算该根结点的左子树和右子树的高度并且进行比较，返回较高的子树+1

4、计算二叉树的结点数时，使用递归思想依次计算左子树和右子树的结点数，之后返回左子树的结点数+右子树的结点数+1

5、计算叶子结点数时，对叶子结点的条件进行判断，若是叶子结点则直接返回1，否则返回该结点的左子树的叶子节点数+右子树的叶子结点数

6、在计算度为2的结点数时注意要设置一个全局变量来统计度为2的结点的个数

7、在判断该结点的双亲结点，左孩子，右孩子时，首先直接传入该结点的父亲节点，因为是单项传递，如果直接传入该节点那么无法找寻到该结点的双亲结点。

8、获取某个结点的层次时，当T不为空时判断该节点是否为所查找的结点，若不是，则level++，递归的调用继续查找，同时设置标志flag，查找到该节点与查不到该节点时提示不同的信息

9、将左右子树进行互换时采用递归的思想，先将该结点的左子树和右子树进行互换，之后递归的调用将该节点左子树的左右子树和右子树的左右子树进行互换。

10、复制二叉树时，若树为空，将需要复制的二叉树也标记为空，若树不为空则创建结点复制该结点，之后采用递归的思想复制该结点的左子树和右子树。

11、利用栈的思想，首先从头到尾依次访问并且将结点存储到栈中，如果该结点是叶子结点则将栈中的所有元素进行输出。之后利用递归的思想访问输出该结点的左子树的叶子节点以及右子树的叶子结点的路径。

12、查找该两个结点最近的祖先时，如果没有找到则返回‘0’，如果该结点是需要查找的两个结点则返回当前结点，如果不是则在其左右子树上寻找，如果在左子树上找到返回这个值，如果在右子树上找到返回这个值，若在左右子树上都没找到那么该结点就是这两个结点最近的祖先。

13、直接输出该结点，之后调用求树高度的函数，该树的左子树树高则从左边输出该树的路径，若右边高则从右边走输出路径。

14、在进行二叉树的层次遍历时，首先将该结点入队，判断该结点的左右子树，若左子树不为空则将左子树入队，若右子树不为空则将右子树入队，之后将该结点出队并且输出，当队列不为空时重复上述操作直到队为空，注意：哪一个结点出队判断哪一个结点的左右子树