# 实验4 二叉树及其应用

**实验日期：2020年05月06日 20190521340 周吉瑞 报告日期：2020年05月06日**

### 一、实验目的

**1、掌握二叉树的基本特性**

**2、掌握二叉树的先序、中序、后序的递归遍历算法**

**3、通过求二叉树的深度、叶子结点数等算法，理解二叉树的基本特性**

### 二、实验预习

**复习以下概念**

1. **二叉树：**

**在计算机科学中，二叉树是每个结点最多有两个子树的树结构，通常子树被称作“左子树”和“右子树”。**

1. **递归遍历：**

**在二叉树中，通过自己对自己本身的调用，实现遍历二叉树的所有结点。**

### 三、实验内容和要求

**1、阅读并运行下面程序，根据输入写出运行结果，并画出二叉树的形态。**

#include<stdio.h>

#include<malloc.h>

**typedef struct BTNode{ /\*结点结构声明\*/**

**char data ; /\*结点数据\*/**

**struct BTNode \*lchild;**

**struct BTNode \*rchild ; /\*指针\*/**

**}\*BiTree;**

int createBiTree(BiTree &t){ /\* 先序遍历创建二叉树\*/

char s;

BiTree q;

s=getchar();

if(s=='.'){t=NULL; return 1;}

q=(BiTree)malloc(sizeof(struct BTNode));

if(q==NULL){printf("Memory alloc failure!"); return 0;}

q->data=s;

t=q;

createBiTree(q->lchild); /\*递归建立左子树\*/

createBiTree(q->rchild); /\*递归建立右子树\*/

return 1;

}

void PreOrder(BiTree p){ /\* 先序遍历二叉树\*/

if ( p!= NULL ) {

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

PreOrder( p->lchild ) ;

PreOrder( p->rchild) ;

}

}

void InOrder(BiTree p){ /\* 中序遍历二叉树\*/

if( p!= NULL ) {

InOrder( p->lchild ) ;

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

InOrder( p->rchild) ;

}

}

void PostOrder(BiTree p){ /\* 后序遍历二叉树\*/

if ( p!= NULL ) {

PostOrder( p->lchild ) ;

PostOrder( p->rchild) ;

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

}

}

void release(BiTree t){ /\*释放二叉树空间\*/

if(t!=NULL){

release(t->lchild);

release(t->rchild);

free(t);

}

}

/\*

int nodenum(BiTree p) //求二叉树结点数

{int n;

if (p == NULL)

return 0;

else

n = (nodenum(p->lchild) + nodenum(p->rchild) + 1);

return n;

}

int leafnum(BiTree p) //求二叉树叶子结点数

{int n0;

if(p==NULL) n0=0;

else

if(p->lchild==NULL && p->rchild==NULL)

n0= 1 ;

else

n0=leafnum(p->lchild)+ leafnum(p->rchild) ;

return n0 ;

}

int depth(BiTree p) //求二叉树深度

{int d1,d2;

if(p==NULL) return 0;

else

{d1= depth(p->lchild) ;

d2= depth(p->rchild) ;

if (d1 > d2)

return (d1 + 1);

else

return (d2 + 1);

}

}

\*/

int main(){

BiTree t=NULL;

printf("\nplease input data:(用.表示空树)");

createBiTree(t);

printf("\n\nPreOrder the tree is:");

PreOrder(t);

printf("\n\nInOrder the tree is:");

InOrder(t);

printf("\n\nPostOrder the tree is:");

PostOrder(t);

/\*

printf("\n该二叉树有%d个结点\n",nodenum(t));

printf("\n该二叉树有%d个叶子结点\n",leafnum(t));

printf("\n该二叉树的深度为%d\n",depth(t));

\*/

release(t);

return 0;

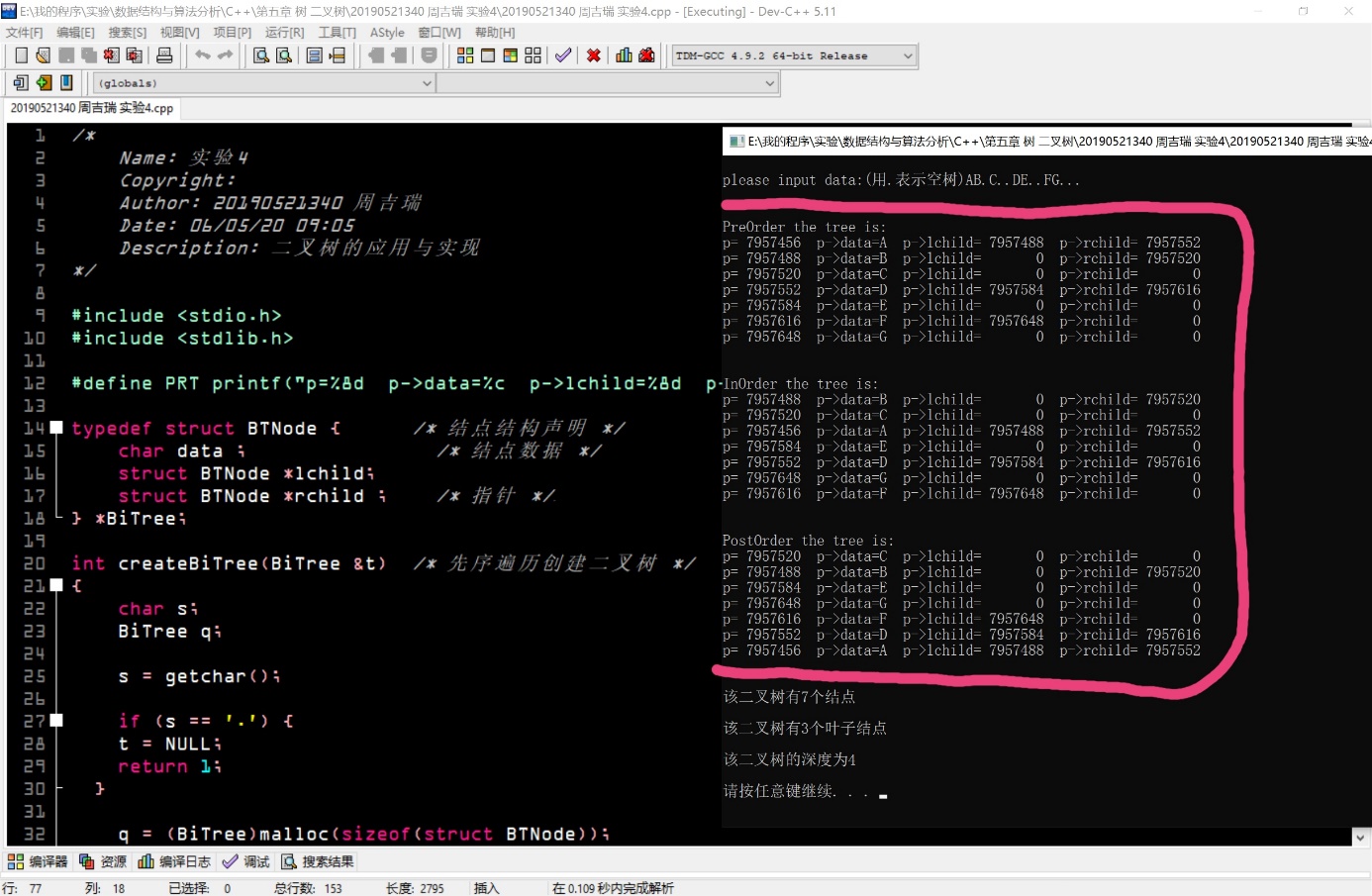
}

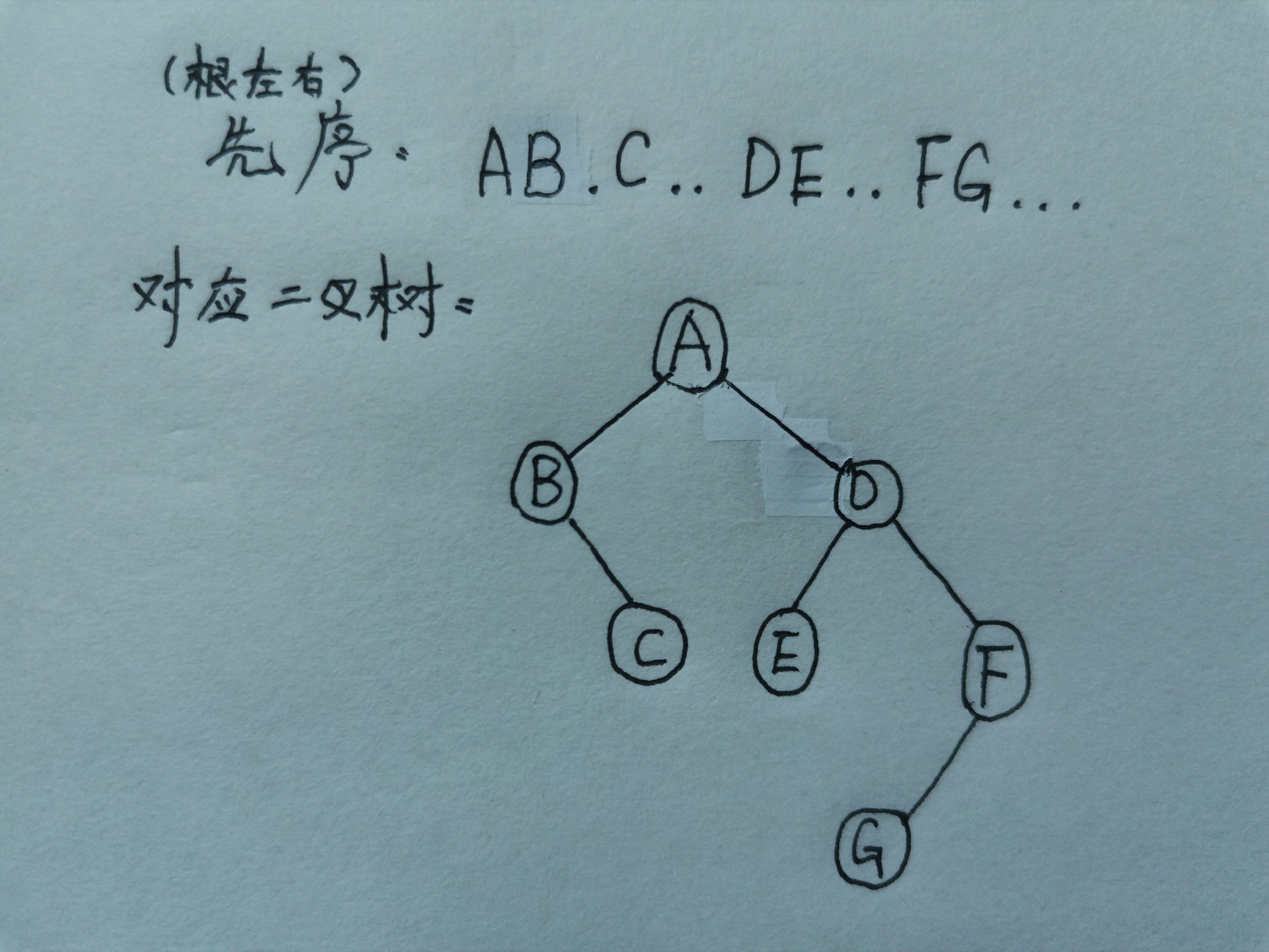
**运行程序**

**输入：**

**AB.C..DE..FG...**

**运行结果：**

****

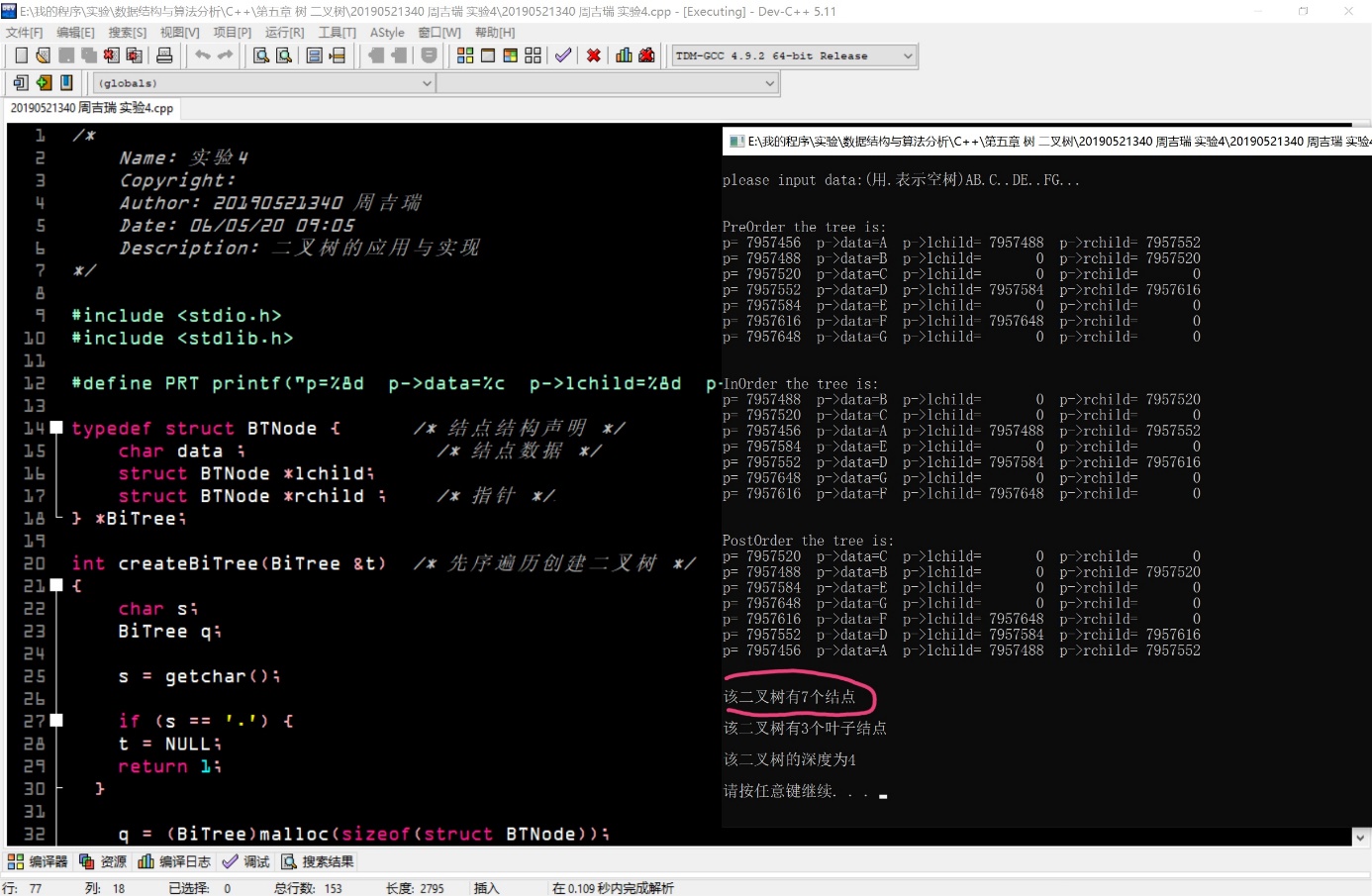
****

**2、在上题中补充完成求二叉树中求结点总数算法，并在主函数中补充相应的调用验证正确性。**

**输入：**

**AB.C..DE..FG...**

**运行结果：**

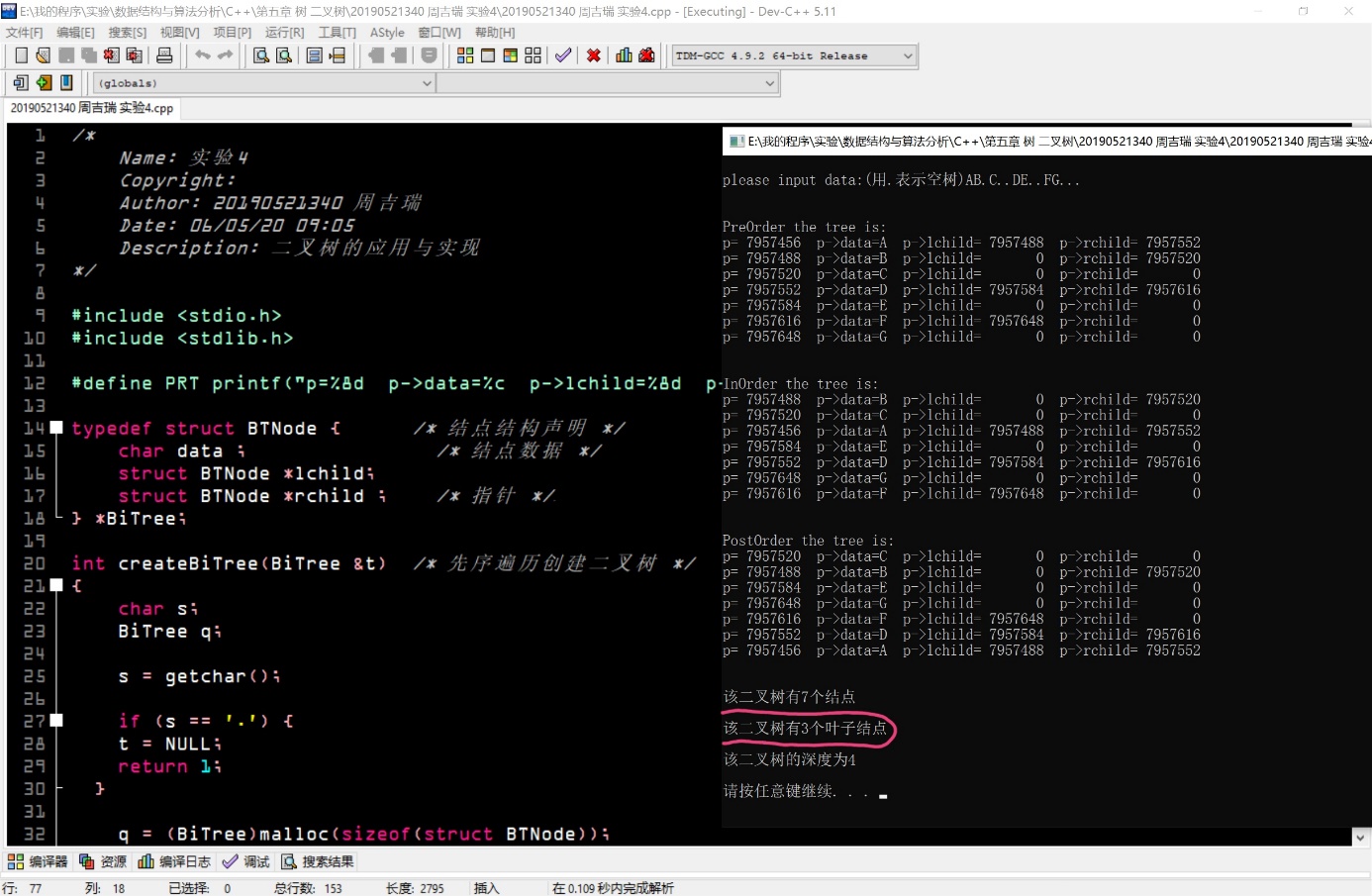
****

**3、在上题中补充完成求二叉树中求叶子结点总数算法，并在主函数中补充相应的调用验证正确性。**

**输入：**

**AB.C..DE..FG...**

**运行结果：**

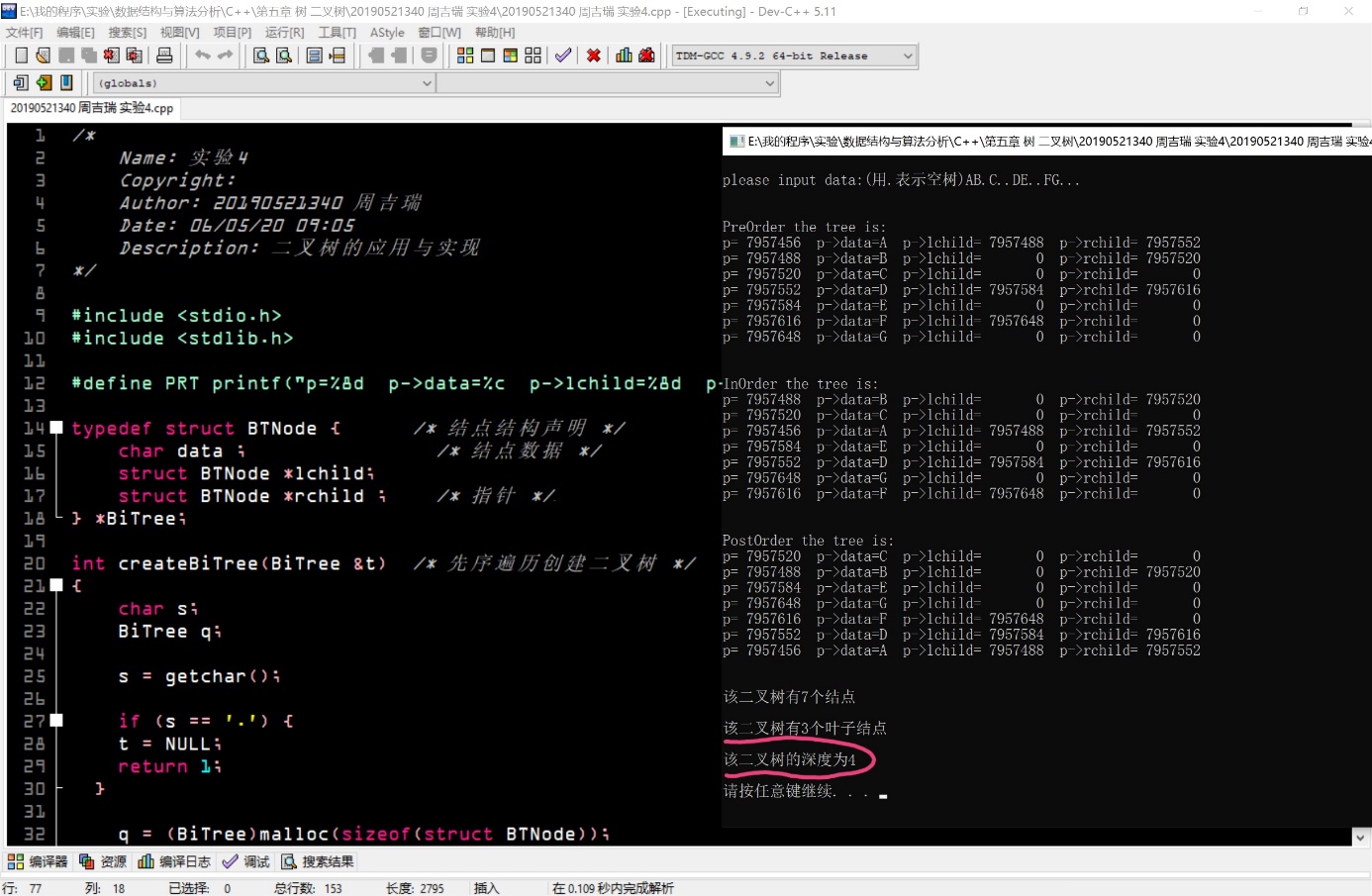
****

**4、在上题中补充完成求二叉树深度算法，并在主函数中补充相应的调用验证正确性。**

**输入：**

**AB.C..DE..FG...**

**运行结果：**

****

### 四、实验小结

**1.关键在于理解二叉树的：先序遍历（根、左、右），中序遍历（左、根、右），后序遍历（左、右、根）。**

**2.在实际代码编写中一定要注意各类“名称”是否对应，许多错误都发生在“名称”没有对应导致编译失败的情况。**

**3.对于二叉树而言，难点在于二叉树的性质，其中涉及许多的数学推导，所以要深刻理解方能掌握。**