实验 5 二叉树操作

一、 实验目的

1. 熟悉二叉树的二叉链表存储结构。
2. 掌握二叉树每一种操作的具体实现。
3. 学会利用递归方法编写对二叉树这种递归结构进行处理的算法。

二、 实验要求

1. 认真阅读并理解教材上相关操作函数。
2. 正确编写本程序并能上机运行。
3. 必须完成：
   1) 先序创建二叉树
   2) 三种遍历递归算法（先序、中序、后序）
   3) 求二叉树高度（深度）
   4) 求二叉树中结点总数
   5) 求二叉树中叶子数目
4. 选做：
   1) 三种遍历方法的非递归算法（先序、中序、后序）
   2) 层序遍历

可以自己重新编写程序，也可以将参考程序框架中剩余的操作函数补齐。

三 程序框架参考如下

注意：

- 函数调用时的函数名、实参与函数定义时的函数名、形参要能一一对应。
- 下列程序中函数的参数有引用参数，若不使用引用参数，而用指针参数的话，应该将主函数中函数调用的形式也改过来。

➢ **题目描述：**熟悉二叉树的二叉链表存储结构。掌握二叉树每一种操作的具体实现。学会利用递归方法编写对二叉树这种递归结构进行处理的算法。
构造一个二叉树，输入多行数字，每一行的含义如下： 输入 1，代表先序创建二叉树。**按先序次序输入，以#字符表示空树** 输入 2，代表先序递归遍历二叉树。 输入 3，代表中序递归遍历二叉树。 输入 4，代表后序递归遍历二叉树。 输入 5，代表求二叉树的深度。 输入 6，代表求二叉树的叶子结点个数。 输入 7，代表求二叉树的结点总数。 输入 0，代表退出。
输出：根据输入的不同数值和含义，输出所有答案。

输入样例：

```
1

AB#C##D##

2

3

4

5

6

7

0
```

输出样例：

```
A B C D

B C A D

C B D A

3

2

4
```

参考代码如下：

```
#include <iostream>
using namespace std;

typedef char ElemType;   //定义二叉树结点值的类型为字符型
typedef struct BiTNode
{
    //====补充代码====

}BiTNode, *BiTree;

int LEAFCOUNT=0;
int NODECOUNT=0;

//按先序次序输入，以特殊字符表示空树
void CreateBiTree(BiTree &T)
{
    //====补充代码====
```

```
}

//先序遍历
void PreOrderTraverse(BiTree T)
{
    //====补充代码====

}

//中序遍历
void InOrderTraverse(BiTree T)
{
    //====补充代码====

}

//后序遍历
void PostOrderTraverse(BiTree T)
{
    //====补充代码====

}

//求二叉树的深度
int BTDepth(BiTree T)
{
    //====补充代码====

}

//求二叉树的叶子数，使用全局变量
void Leaf(BiTree T)
{
    //====补充代码====

}

void NodeCount(BiTree T)//求二叉树的结点总数，使用全局变量
{
    //====补充代码====

}

int main()
{
```

```
BiTree T=NULL;
int select;
while(cin>>select)
{

    switch(select)
    {
        case 0:
            return 0;
        case 1:
            cin.clear();//接收
            CreateBiTree(T);
            break;
        case 2:
            if(T) {
                PreOrderTraverse(T);
                cout<<endl;
            }
            break;
        case 3:
            if(T) {
                InOrderTraverse(T);
                cout<<endl;
            }
            break;
        case 4:
            if(T) {
                PostOrderTraverse(T);
                cout<<endl;
            }
            break;
        case 5:
            cout<<BTDepth(T)<<endl;
            break;
        case 6:
            LEAFCOUNT=0;
            Leaf(T);
            cout<<LEAFCOUNT<<endl;
            break;
        case 7:
            NODECOUNT=0;
            NodeCount(T);
            cout<<NODECOUNT<<endl;
            break;
      }
}
```

```
    return 0;
}
```