```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define OVERFLOW -2
typedef int Status;
typedef char TElemType;
typedef struct BiTNode
{     TElemType data;
      struct BiTNode *lchild, *rchild;     //左右孩子指针
}BiTNode, *BiTree;
int Nodenum=0;   //统计二叉树的结点个数
int Count=0;       //统计二叉树中叶子结点个数

Status CreateBiTree(BiTree &T) {   // 算法 6.4
  // 按先序次序输入二叉树中结点的值（一个字符），#字符表示空树，
  // 构造二叉链表表示的二叉树 T。
  char ch;
  scanf("%c",&ch);
  if (ch=='#') T = NULL;
  else {
    if (!(T = (BiTNode *)malloc(sizeof(BiTNode)))) exit(OVERFLOW);
    T->data = ch;                    // 生成根结点
    CreateBiTree(T->lchild);     // 构造左子树
    CreateBiTree(T->rchild);     // 构造右子树
  }
  return OK;
} // CreateBiTree

Status PreOrderTraverse(BiTree T) {
    // 算法 6.1
    // 采用二叉链表存储结构
    // 先序遍历二叉树 T 的递归算法
    if (T) {
          printf("%c", T->data);
          if (PreOrderTraverse(T->lchild))
              if (PreOrderTraverse(T->rchild)) return OK;
          return ERROR;
    } else return OK;
} // PreOrderTraverse
```

```
Status InOrderTraverse(BiTree T) {
    // 采用二叉链表存储结构
    // 中序遍历二叉树 T 的递归算法
     int m=0;
     if (T) {
          if (InOrderTraverse(T->lchild)) m=1;
          printf("%c", T->data);
          if (m)
               if (InOrderTraverse(T->rchild)) return OK;
          return ERROR;
    } else return OK;
} // InOrderTraverse

Status PostOrderTraverse(BiTree T) {
    // 采用二叉链表存储结构
    // 后序遍历二叉树 T 的递归算法
    if (T) {
          if (PostOrderTraverse(T->lchild))
               if (PostOrderTraverse(T->rchild)) {
                    printf("%c", T->data); return OK;
               }
          return ERROR;
    } else return OK;
} // PostOrderTraverse

// 用递归方法求二叉树的深度
int Depth(BiTree T)
{
     int leftDepth, rightDepth;
     if (T == NULL) return 0;
     else {
          leftDepth = Depth(T->lchild);
          rightDepth = Depth(T->rchild);
          if (leftDepth>=rightDepth) return leftDepth+1;
          else return rightDepth+1;
     }
}

Status NodeCount(BiTree T) {
     //统计二叉树的结点个数
     if (T) {
          Nodenum++;
          if (NodeCount(T->lchild))
               if (NodeCount(T->rchild)) return OK;
```

```
            return ERROR;
        } else return OK;
} // NodeCount

Status LeafCount(BiTree T) {
    //输出二叉树的叶子结点，并统计叶子结点个数
    if (T) {
        if (T->lchild == NULL && T->rchild == NULL) {
            printf("%c", T->data); Count++;
        }
        if (LeafCount(T->lchild))
            if (LeafCount(T->rchild)) return OK;
        return ERROR;
    } else return OK;
} // LeafCount

Status ExchangeBiTree(BiTree &T) {
    //交换二叉树中所有结点的左右子树
    BiTree p;
    if (T) {
        if (T->lchild || T->rchild) {
            p = T->lchild; T->lchild = T->rchild; T->rchild = p;
        }
        if (ExchangeBiTree(T->lchild))
            if (ExchangeBiTree(T->rchild)) return OK;
        return ERROR;
    } else return OK;
} // ExchangeBiTree

Status CopyBiTree(BiTree T, BiTree &B) {
    //复制二叉树
    if (T==NULL)
        B = NULL;
    else {
        if (!(B = (BiTNode *)malloc(sizeof(BiTNode)))) exit(OVERFLOW);
        B->data = T->data;                          //复制一个根节点*B
        CopyBiTree(T->lchild,B->lchild);    //递归复制左子树
        CopyBiTree(T->rchild,B->rchild);    //递归复制右子树
    }
    return OK;
} // CopyBiTree
```

```
Status DestroyTree(BiTree &T) {
  //销毁二叉树
  if (T) {
      DestroyTree(T->lchild);
      DestroyTree(T->rchild);
      free(T);
  }
  return OK;
} // DestroyTree

Status ClearTree(BiTree &T) {
  //置二叉树为空树
  if (T) {
      T = NULL;
  }
  return OK;
} // ClearTree

void main()
{
    BiTree T, B;
    printf("创建二叉树，按先序次序输入二叉树中结点的值：\n");
    CreateBiTree(T);
    NodeCount(T);
    printf("二叉树的结点个数为：%d\n", Nodenum);
    printf("二叉树的深度为：%d\n", Depth(T));
    printf("先序遍历二叉树，结果是：\n");
    PreOrderTraverse(T);
    printf("\n");
    printf("中序遍历二叉树，结果是：\n");
    InOrderTraverse(T);
    printf("\n");
    printf("后序遍历二叉树，结果是：\n");
    PostOrderTraverse(T);
    printf("\n");
    printf("输出二叉树的叶子结点：\n");
    LeafCount(T);
    printf("\n");
    printf("统计二叉树的叶子结点个数：%d\n", Count);
    printf("交换二叉树中所有结点的左右子树！\n");
    ExchangeBiTree(T);
    if (CopyBiTree(T, B)==OK) printf("成功复制二叉树 T 到二叉树 B！\n");
    if (DestroyTree(T)==OK) printf("成功销毁二叉树 T！\n");
    if (ClearTree(T)==OK) printf("将二叉树 T 置为空树！\n");
```

```
    printf("先序遍历二叉树 B，结果是：\n");
    PreOrderTraverse(B);
    printf("\n");
    printf("中序遍历二叉树 B，结果是：\n");
    InOrderTraverse(B);
    printf("\n");
    printf("后序遍历二叉树 B，结果是：\n");
    PostOrderTraverse(B);
    printf("\n");
}
```