```c
//
// Created by Lenovo on 2022-04-24-下午 4:52.
// 作者：小象
// 版本：1.0
//

#include <stdio.h>
#include <malloc.h>

typedef struct BiTNode { // 二叉链表的定义
    char data; // 结点数据域
    struct BiTNode *lChild, *rChild; // 左右孩子指针
} BiTNode, *BiTree;

void InitBiTree(BiTree *tree); // 初始化
void CreateBiTree(BiTree *tree); // 创建二叉链表
BiTree InsertLeftNode(BiTree tree, char data); // 插入左子树
BiTree InsertRightNode(BiTree tree, char data); // 插入右子树
void PreOrderTraverse(BiTree tree); // 先序遍历二叉树
void InOrderTraverse(BiTree tree); // 中序遍历二叉树
void PostOrderTraverse(BiTree tree); // 后序遍历二叉树
void VisitNode(BiTree tree); // 访问二叉树结点
void DestroyBiTree(BiTree tree); // 使用后序遍历的方式销毁二叉树

int count = -1;

/**
 * <h2>二叉树实验一</h2>
 * <h3>三种遍历二叉树方法</h3>
 * @return 0
 */
int main() {

    // 定义一棵树
    BiTree tree;
    // 初始化
    InitBiTree(&tree);
    // 创建二叉树
    CreateBiTree(&tree);
    // 前序遍历
    printf("前序遍历：");
    PreOrderTraverse(tree);
    printf("\n");
    // 中序遍历
    printf("中序遍历：");
```

```
    InOrderTraverse(tree);
    printf("\n");
    // 后序遍历二叉树
    printf("后序遍历: ");
    PostOrderTraverse(tree);
    // 释放二叉树
    DestroyBiTree(tree);

    return 0;
}

void InitBiTree(BiTree *tree) {
    *tree = NULL; // 初始化为空
}

// 按先序次序输入二叉树中结点的值（一个字符），创建二叉链表表示的二叉树 tree
void CreateBiTree(BiTree *tree) {
    *tree = InsertLeftNode(*tree, 'A');
    *tree = InsertLeftNode(*tree, 'B');
    *tree = InsertLeftNode(*tree, 'E');
    (*tree)->lChild->lChild = InsertRightNode((*tree)->lChild->lChild, 'F');
    (*tree)->lChild = InsertRightNode((*tree)->lChild, 'C');
    (*tree)->lChild = InsertRightNode((*tree)->lChild, 'D');
    (*tree)->lChild->rChild->rChild = InsertLeftNode((*tree)->lChild->rChild->rChild, 'G');

}

// 左
BiTree InsertLeftNode(BiTree tree, char data) {
    if (tree != NULL) {
        tree->lChild = InsertLeftNode(tree->lChild, data);
    } else {
        tree = (BiTree) malloc(sizeof(BiTNode));
        if (tree == NULL) {
            return NULL;
        }
        tree->data = data;
        tree->lChild = tree->rChild = NULL;
    }
    return tree;
}

// 右
BiTree InsertRightNode(BiTree tree, char data) {
    if (tree != NULL) {
```

```
        tree->rChild = InsertRightNode(tree->rChild, data);
    } else {
        tree = (BiTree) malloc(sizeof(BiTNode));
        if (tree == NULL) {
            return NULL;
        }
        tree->data = data;
        tree->lChild = tree->rChild = NULL;
    }
    return tree;
}

// 先序遍历二叉树 tree 的递归算法
void PreOrderTraverse(BiTree tree) {
    if (tree != NULL) { // 若二叉树非空
        VisitNode(tree); // 访问根结点
        PreOrderTraverse(tree->lChild); // 先序遍历左子树
        PreOrderTraverse(tree->rChild); // 先序遍历右子树
    }
}

// 中序遍历二叉树 tree 的递归算法
void InOrderTraverse(BiTree tree) {
    if (tree != NULL) { // 若二叉树非空
        InOrderTraverse(tree->lChild); // 中序遍历左子树
        VisitNode(tree); // 访问根结点
        InOrderTraverse(tree->rChild); // 中序遍历右子树
    }
}

// 后序遍历二叉树 tree 的递归算法
void PostOrderTraverse(BiTree tree) {
    if (tree != NULL) { // 若二叉树非空
        PostOrderTraverse(tree->lChild); // 后序遍历左子树
        PostOrderTraverse(tree->rChild); // 后序遍历右子树
        VisitNode(tree); // 访问根结点
    }
}

// 输出二叉树结点值
void VisitNode(BiTree tree) {
    printf("%c ", tree->data);
}

// 释放二叉树
```

```c
void DestroyBiTree(BiTree tree) {
    if (tree != NULL) { // 若二叉树非空
        DestroyBiTree(tree->lChild); // 后序遍历左子树
        DestroyBiTree(tree->rChild); // 后序遍历右子树
        free(tree);
        tree = NULL;
    }
}
```