

# 《数据结构与算法实验》第 9 次实验

学院：

专业：

年级：

姓名：

学号：

日期： 2022 年 5 月 21 日

## 第一部分 验证实验

### 一、 实验目的

1. 掌握二叉树的逻辑结构和二叉链表存储结构。
2. 验证二叉树的二叉链表存储及遍历操作，实现遍历操作递归和非递归算法。
3. 掌握树的逻辑结构和孩子兄弟存储结构，并验证其存储结构及遍历操作。

### 二、 实验内容

1. 二叉树的实现 1（参考实验书 p207）
  - 建立一棵含有  $n$  个结点的二叉树，采用二叉链表存储。
  - 输出前序、中序、后序、层序遍历该二叉树的遍历结果。
2. 树的实现（参考实验书 p210）
  - 采用孩子兄弟表示法建立一棵树。
  - 基于树的孩子兄弟表示法实现前序和后序遍历树的操作。
3. 二叉树的实现 2（参考教材 p125）
  - 在第一个验证实验的基础上，将递归算法转化为非递归算法。

### 三、 设计编码

#### 1. 二叉树的实现 1

(1) BiTree.h

```
#ifndef BiTree_H
#define BiTree_H

struct BiNode
{
```

```
    char data;
    BiNode *lchild, *rchild;
};

class BiTree
{
public:
    BiTree(){root=Creat(root);}
    ~BiTree(){Release(root);}
    void PreOrder(){PreOrder(root);}
    void InOrder(){InOrder(root);}
    void PostOrder(){PostOrder(root);}
private:
    BiNode *root;
    BiNode *Creat(BiNode *bt);
    void Release(BiNode *bt);
    void PreOrder(BiNode *bt);
    void InOrder(BiNode *bt);
    void PostOrder(BiNode *bt);
};

#endif
```

## (2)BiTree.cpp

```
#include <iostream>
#include <cstdio>
using namespace std;
#include "BiTree.h"

BiNode *BiTree::Creat(BiNode *bt)
{
    char ch;
    cout<<"请输入创建一棵二叉树的结点数据"<<endl;
    cin>>ch;
    if(ch == '#') return NULL;
    else
    {
        bt = new BiNode;
        bt -> data = ch;
        bt -> lchild = Creat(bt -> lchild);
        bt -> rchild = Creat(bt -> rchild);
    }
    return bt;
}
```

```
}

void BiTree::Release(BiNode *bt)
{
    if (bt != NULL)
    {
        Release(bt -> lchild);
        Release(bt -> rchild);
        delete bt;
    }
}

void BiTree::PreOrder(BiNode *bt)
{
    if (bt == NULL) return;
    else
    {
        cout<<bt->data<<" ";
        PreOrder(bt->lchild);
        PreOrder(bt->rchild);
    }
}

void BiTree::InOrder(BiNode *bt)
{
    if (bt == NULL) return;
    else
    {
        InOrder(bt->lchild);
        cout<<bt->data<<" ";
        InOrder(bt->rchild);
    }
}

void BiTree::PostOrder(BiNode *bt)
{
    if(bt==NULL) return ;
    else
    {
        PostOrder(bt->lchild);
        PostOrder(bt->rchild);
        cout<<bt->data<<" ";
    }
}
```

## (3) BiTree\_main.cpp

```
#include <iostream>
using namespace std;
#include "BiTree.h"

int main()
{
    BiTree T;
    cout<<endl;
    cout<<"——前序遍历——"<<endl;
    T.PreOrder();
    cout<<endl;
    cout<<"——中序遍历——"<<endl;
    T.InOrder();
    cout<<endl;
    cout<<"——后序遍历——"<<endl;
    T.PostOrder();
    cout<<endl;
}
```

## 2. 树的实现

## (1) Tree.h

```
#ifndef Tree_H
#define Tree_H
const int Max=20;

struct TNode
{
    char data;
    TNode *firstchild,*rightsib;
};

//以下是树类 tree 的声明
class Tree
{
public:
    Tree();
    ~Tree();
    void PreOrder();
}
```

```
        void PostOrder();
    private:
        TNode *root;
        void Release(TNode *bt);
        void PreOrder(TNode *bt);
        void PostOrder(TNode *bt);
};

#endif
```

## (2) Tree.cpp

```
#include <iostream>
using namespace std;
#include "Tree.h"

Tree::Tree()
{
    TNode *Q[Max] = {NULL};
    int front = -1, rear = -1;
    char ch1 = '#', ch2 = '#';
    TNode *p = NULL, *q = NULL;
    cout<<"请输入根结点: ";
    cin>>ch1;
    p = new TNode;
    p -> data = ch1;
    p -> firstchild = p -> rightsib = NULL;
    root = p;
    Q[ ++rear ] = p;
    cout<<"请输入结点对, 以空格分隔: ";
    fflush(stdin);
    ch1 = getchar();
    getchar();
    ch2 = getchar();
    while (ch1 != '#' || ch2 != '#')
    {
        p = new TNode;
        p -> data = ch2;
        p -> firstchild = p -> rightsib = NULL;
        Q[++rear] = p;
        while (front != rear)
        {
            q = Q[front+1];
```

```
        if (q -> data != ch1) front++;
        else
        {
            if(q -> firstchild == NULL)
                q -> firstchild = p;
            else
            {
                while (q -> rightsib != NULL)
                    {q = q -> rightsib;}
                q -> rightsib = p;
            }
            break;
        }
    }
    cout<<"请输入结点对，以空格分隔： ";
    fflush(stdin);
    ch1 = getchar();
    getchar();
    ch2 = getchar();
}

Tree::~Tree()
{
    Release(root);
}

void Tree::PreOrder()
{
    PreOrder(root);
}

void Tree::PostOrder()
{
    PostOrder(root);
}

void Tree::Release(TNode *bt)
{
    if (bt == NULL) return;
    else
    {
        Release(bt -> firstchild);
        Release(bt -> rightsib);
    }
}
```

```
        delete bt;
    }
}

void Tree::PreOrder(TNode *bt)
{
    if (bt == NULL) return;
    else
    {
        cout<<bt -> data;
        PreOrder(bt -> firstchild);
        PreOrder(bt -> rightsib);
    }
}

void Tree::PostOrder(TNode *bt)
{
    if (bt == NULL) return;
    else
    {
        PostOrder(bt -> firstchild);
        cout<<bt -> data;
        PostOrder(bt -> rightsib);
    }
}
```

### (3) Tree\_main.cpp

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
#include "Tree.h"

int main()
{
    Tree t1;
    t1.PreOrder();
    cout<<endl;
    t1.PostOrder();
    cout<<endl;
    return 0;
}
```

### 3. 二叉树的实现 2

(1)Bitree.h

```
#ifndef Bitree_H
#define Bitree_H
const int MaxSize=20;

struct BiNode
{
    char data;
    BiNode *lchild,*rchild;
};

struct element
{
    BiNode * ptr;
    int flag;
};

//以下是树类 tree 的声明
class Bitree
{
public:
    Bitree();
    ~Bitree();
    void PreOrder();
    void PostOrder();
    void LeverOrder();
    BiNode *getRoot();
    void Release(BiNode *bt);
    void PreOrder(BiNode *bt);
    void InOrder(BiNode *bt);
    void PostOrder(BiNode *bt);
private:
    BiNode * root;
    BiNode * Creat(BiNode *bt);
    //void Release(BiNode *bt);
};

#endif
```



## (2)Bitree.cpp

```
#include <iostream>
using namespace std;
#include "Bitree.h"

Bitree::Bitree()
{
    root = Creat(root);
}

Bitree::~~Bitree()
{
    Release(root);
}

void Bitree::PreOrder()
{
    PreOrder(root);
}

void Bitree::PostOrder()
{
    PostOrder(root);
}

BiNode *Bitree::Creat(BiNode *bt)
{
    char ch;
    cout<<"请输入创建一棵二叉树的结点数据"<<endl;
    cin>>ch;
    if (ch == '#')
    {
        return NULL;
    }
    else
    {
        bt = new BiNode;
        bt -> data = ch;
        bt -> lchild = Creat(bt -> lchild);
        bt -> rchild = Creat(bt -> rchild);
    }
    return bt;
}
```

```
void Bitree::Release(BiNode *bt)
{
    if (bt != NULL)
    {
        Release(bt -> lchild);
        Release(bt -> rchild);
        delete bt;
    }
}

void Bitree::LeverOrder()
{
    int front = -1, rear = -1;
    BiNode *Q[MaxSize], *q;
    if (root == NULL) return;
    else
    {
        Q[++rear] = root;
        while (front != rear)
        {
            q = Q[++front];
            cout<<q->data<<" ";
            if (q->lchild!=NULL) Q[++rear]=q->lchild;
            if (q->rchild!=NULL) Q[++rear]=q->rchild;
        }
    }
}

BiNode *Bitree::getRoot()
{
    return root;
}

void Bitree::PreOrder(BiNode *bt)
{
    int top = -1;
    BiNode *s[MaxSize] = {NULL};
    while (bt != NULL || top != -1)
    {
        while (bt != NULL)
        {
            cout<<bt -> data;
            s[++top] = bt;
        }
    }
}
```

```
        bt = bt -> lchild;
    }
    if (top != -1)
    {
        bt = s[top--];
        bt = bt -> rchild;
    }
}

void Bitree::InOrder(BiNode *bt)
{
    int top = -1;
    BiNode *s[MaxSize] = {NULL};
    while (bt != NULL || top != -1)
    {
        while (bt != NULL)
        {
            s[++top] = bt;
            bt = bt->lchild;
        }
        if (top != -1)
        {
            bt = s[top--];
            cout<<bt -> data;
            bt = bt -> rchild;
        }
    }
}

void Bitree::PostOrder(BiNode *bt)
{
    int top = -1;
    element s[MaxSize] = {NULL,1};
    while (bt != NULL || top != -1)
    {
        while (bt != NULL)
        {
            top++;
            s[top].ptr = bt;
            s[top].flag = 1;
            bt = bt -> lchild;
        }
        while (top != -1 && s[top].flag == 1)
```

```
    {
        bt = s[top--].ptr;
        cout<<bt ->data;
    }
    if (top != -1)
    {
        s[top].flag = 2;
        bt = s[top].ptr -> rchild;
    }
    else break;
}
}
```

### (3) Bitree\_main.cpp

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
#include "Bitree.h"

int main()
{
    Bitree T;
    BiNode *root = T.getRoot();
    cout<<"——前序遍历——"<<endl;
    T.PreOrder(root);
    cout<<endl;
    cout<<"——中序遍历——"<<endl;
    T.InOrder(root);
    cout<<endl;
    cout<<"——后序遍历——"<<endl;
    T.PostOrder(root);
    cout<<endl;
    return 0;
}
```

## 四、 运行与测试

### 1. 二叉树的实现 1 2

```

请输入创建一棵二叉树的结点数据
A
请输入创建一棵二叉树的结点数据
B
请输入创建一棵二叉树的结点数据
E
请输入创建一棵二叉树的结点数据
#
请输入创建一棵二叉树的结点数据
#
请输入创建一棵二叉树的结点数据
D
请输入创建一棵二叉树的结点数据
#
请输入创建一棵二叉树的结点数据
#
请输入创建一棵二叉树的结点数据
C
请输入创建一棵二叉树的结点数据
#
请输入创建一棵二叉树的结点数据
#

```

```

-----前序遍历-----
A B E D C
-----中序遍历-----
E B D A C
-----后序遍历-----
E D B C A

```

## 2. 树的实现

```

请输入根结点: A
请输入结点对, 以空格分隔: A B
请输入结点对, 以空格分隔: A C
请输入结点对, 以空格分隔: B E
请输入结点对, 以空格分隔: B D
请输入结点对, 以空格分隔: # #
ABEDC
EBDAC

```

```

-----
Process exited after 14.34 seconds with return value 0
请按任意键继续. . .

```

## 第二部分 设计实验

### 一、实验目的

通过共计 13 个实验，实现根据前序遍历和中序遍历建立二叉树、对二叉树的复制、两棵

二叉树相似的判断、求结点与根结点的路径、求叶子个数与叶子、求总结点数、求深度、求后序遍历的逆序、求双亲、删除子树、交换左右子树、顺序存储的遍历、求某结点的第  $i$  个孩子。

## 二、实验内容

### 1. 根据前序遍历和中序遍历建立二叉树：参考实验书 p70

这个过程是一个递归过程，其基本思想是：先根据前序序列的第一个元素建立根结点；然后在中序序列中找到该元素，确定根结点的左、右子树的中序序列；再在前序序列中确定左、右子树的前序序列；最后由左子树的前序序列与中序序列建立左子树，由右子树的前序序列与中序序列建立右子树。

### 2. 复制一棵二叉树：参考实验书 p71

复制二叉树是在计算机中已经存在一棵二叉树，要求按原二叉树的结构重新生成一棵二叉树，其实质就是按照原二叉树的二叉链表另建立一个新的二叉链表。复制是在遍历过程中，将“访问”操作定义为“生成二叉树的一个结点”。

### 3. 判断两棵二叉树是否相似：参考实验书 p72

判断两棵二叉树是否相似，所谓两棵二叉树相似，是指要么它们都为空或都只有一个根结点，要么它们的左右子树均相似分析：依题意，得到如下判定两棵二叉树  $s$  和  $t$  是否相似的递归函数 Like：

- (1) 若 NULL，则  $s$  和  $t$  相似，即  $\text{Like}(s)$
- (2) 若  $s$  和  $t$  中有一个为 NULL，另一个不为 NULL，则  $s$  和  $t$  不相似， $\text{Like}(s, t)=0$
- (3) 进一步判断  $s$  的左子树和  $t$  的左子树、 $s$  的右子树和  $t$  的右子树是否相似

### 4. 求路径：参考实验书 p72

- 假设二叉树采用二叉链表存储， $p$  所指为任一指定结点，编写算法求从根结点到  $p$  所指结点之间的路径。

- 本题采用非递归后序遍历二叉树，当后序遍历访问到  $p$  所指结点时，此时栈中所有结点均为  $p$  所指结点的祖先，由这些祖先便构成了一条从根结点到  $p$  所指结点之间的路径。

### 5. 求二叉树中叶子结点的个数和打印叶子结点：参考实验书 p80, p214

本算法的要求与前序遍历算法既有相同之处，又有不同之处。相同之处是打印次序均为前序，不同之处是此处不是打印每个结点的值，而是打印出其中的叶子结即为有条件打印。为此，将前序遍历算法中的访问操作改为条件打印即可。在打印的时候顺便计数即可。

### 6. 求二叉树节点个数：参考实验书 p80

本算法不是要打印每个结点的值，而是求出结点的个数。所以可将遍历算法中的“访问”操作改为“计数操作”，将结点的数目累加到一个全局变量中，每个结点累加次即完成了结点个数的求解。

### 7. 求二叉树的深度：参考实验书 p80

当二叉树为空时，深度为 0；若二叉树不为空，深度应是其左右子树深度的最大值加 1，而其左右子树深度的求解又可通过递归调用本算法来完成。

### 8. 输出二叉树的后续遍历序列的逆序：参考实验书 p81

要想得到后序的逆序，只要按照后序遍历相反的顺序即可。即先访问根结点，再遍历根结点的右子树，最后遍历根结点的左子树。要注意和前序遍历的区别。

### 9. 求二叉树中结点 x 的双亲：参考实验书 p81

以二叉链表为存储结构对二叉链表进行遍历，在遍历过程中查找结点 x 并记载其双亲。

### 10. 在二叉树中删除以值 x 为根节点的子树：参考实验书 p82

以二叉链表为存储结构对二叉链表进行遍历，在遍历过程中查找结点 x 并记载其双亲，然后将结点 x 的双亲结点中指向 x 的指针置空，再把这个树删除。

### 11. 交换二叉树中所有结点的左右子树：参考实验书 p83

对二叉树进行后序遍历，在遍历过程中访问某结点时交换该结点的左右子树。

### 12. 顺序存储结构的前序遍历：参考实验书 p82

一棵具有  $n$  个结点的二叉树采用顺序存储结构，编写算法对该二叉树进行前序遍历。

按照题目要求，设置一个工作栈以完成对该树的非递归算法，思路如下：

- (1) 每访问一个结点，将此结点压栈，查看此结点是否有左子树。若有，则访问左子重复执行该过程直到左子树为空；
- (2) 从栈弹出一个结点，如果此结点有右孩子结点，则执行步骤(1)，否则执行步骤(3)；
- (3) 如果栈为空，则算法结束，否则重复执行步骤(2)

### 13. 求树中结点 $x$ 的第 $i$ 个孩子：参考实验书 p83

以孩子兄弟表示法做存储结构。

先在链表中进行遍历，在遍历过程中找到等于  $r$  的结点，然后由此结点的最左孩子域 `firstchild` 找到值为  $x$  结点的第一个孩子，再沿右兄弟域 `rightsib` 找到值为  $x$  结点的第  $i$  个孩子并返回指向这个孩子的指针。

## 三、 设计编码

由于编码的重复性较高，因此部分实验的实现过程进行合并，具体参见编码说明。

### 1. T1/2/8-Bitree.h

```
#ifndef Bitree_H
#define Bitree_H
const int MaxSize=100;

struct BiNode
{
    char data;
    BiNode *lchild, *rchild;
};

class Bitree
{
private:
    BiNode *root;
    char pin[MaxSize];
    char pre[MaxSize];
};
```



```
public:
    Bitree();
    Bitree(BiNode *bt);
    ~Bitree();
    BiNode * getRoot();
    BiNode * Creat(BiNode *bt);
    void isEmpty();
    void PreOrder();
    void InOrder();
    void PostOrder();
    void LeverOrder();
    void Release(BiNode *bt);
    void PreOrder(BiNode *bt);
    void InOrder(BiNode *bt);
    void PostOrder(BiNode *bt);
    BiNode * interface();
    BiNode * create(BiNode * &root, int i1, int i2, int k);
    int pos(char, char[], int, int);
};

#endif
```

### 1. T1/2/8-Bitree.cpp

```
#include <iostream>
#include "Bitree.h"
using namespace std;

Bitree::Bitree()
{
    root = NULL;
}

Bitree::Bitree(BiNode *bt)
{
    root = bt;
}

Bitree::~Bitree()
{
    Release(root);
}

void Bitree::PreOrder()
```

```
{
    PreOrder(root);
}

void Bitree::InOrder()
{
    InOrder(root);
}

void Bitree::PostOrder()
{
    PostOrder(root);
}

BiNode *Bitree::getRoot()
{
    return root;
}

void Bitree::isEmpty()
{
    if (root == NULL)
    {
        cout<<"empty"<<endl;
    }
    else
    {
        cout<<"not empty"<<endl;
    }
}

BiNode *Bitree::Creat(BiNode *bt)
{
    char ch;
    cout<<"请输入创建一棵二叉树的结点数据: "<<endl;
    cin>>ch;
    if (ch == '#')
    {
        return NULL;
    }
    else
    {
        bt = new BiNode;
        bt -> data = ch;
    }
}
```

```
        bt -> lchild = Creat(bt -> lchild);
        bt -> rchild = Creat(bt -> rchild);
    }
    return bt;
}

void Bitree::Release(BiNode *bt)
{
    if (bt != NULL)
    {
        Release(bt -> lchild);
        Release(bt -> rchild);
        delete bt;
    }
}

void Bitree::PreOrder(BiNode *bt)
{
    if(bt == NULL) return;
    else
    {
        cout<<bt->data<<" ";
        PreOrder(bt -> lchild);
        PreOrder(bt -> rchild);
    }
}

void Bitree::InOrder(BiNode *bt)
{
    if (bt==NULL)
    {
        return;
    }
    else {
        InOrder(bt->lchild);
        cout<<bt->data<<" ";
        InOrder(bt->rchild);
    }
}

void Bitree::PostOrder(BiNode *bt)
{
    if (bt == NULL) return;
    else
```

```
{
    PostOrder(bt -> lchild);
    PostOrder(bt -> rchild);
    cout<<bt->data<<" ";
}
}

void Bitree::LeverOrder()
{
    int front=-1,rear=-1;
    BiNode *Q[MaxSize],*q;
    if (root == NULL) return;
    else
    {
        Q[++rear] = root;
        while (front != rear)
        {
            q = Q[++front];
            cout<<q->data<<" ";
            if (q -> lchild != NULL)
                Q[++rear] = q->lchild;
            if (q -> rchild != NULL)
                Q[++rear] = q->rchild;
        }
    }
}

BiNode *Bitree::interface()
{
    int n;
    cout << "输入序列长度\n";
    cin >> n;
    cout << "输入前序序列\n";
    for (int i = 0; i < n ; i++)
    {
        cin >> pre[i];
    }
    cout << "输入中序序列\n";
    for (int i = 0; i < n; i++)
    {
        cin >> pin[i];
    }
    root = create(root, 0, 0, n);
    return root;
}
```

```
}

BiNode *Bitree::create(BiNode * &root, int i1, int i2, int k)
{
    //i1 为前序起始下标, i2 为中序起始下标, k 为序列长度
    if (k <= 0)
    {
        root = NULL;
    }
    else
    {
        root = new BiNode;
        root -> data = pre[i1];
        int m = pos(pre[i1], pin, i2, k);
        int leftlen = m - i2;
        int rightlen = k - (leftlen + 1);
        if (k >= 2)
        {
            while (true)
            {
                int m_next = pos(pre[i1], pin, m+1, rightlen);
                if (m_next != -1)
                {
                    int check = pos(pre[i1], pre, i1, leftlen);
                    if (check != -1) //如果找得到
                    {
                        m = m_next;
                        leftlen = m - i2;
                        rightlen = k - (leftlen + 1);
                    }
                    else
                    {
                        break;
                    }
                }
                else break;
            }
            create(root->lchild, i1 + 1, i2, leftlen);
            create(root->rchild, i1 + leftlen + 1, m + 1, rightlen);
        }
        return root;
    }
}
```

```
int Bitree::pos(char data, char pin[], int i2, int k)
{
    // 查找值为 data, 前序序列 pre, 从第 i2 位开始查找, 查找长度为 k
    for (int i = i2; i < i2 + k ; i++)
    {
        if (pin[i] == data)
            return i;
    }
    return -1;
}

BiNode *CopyTree(BiNode *root)
{
    if(root==NULL) return NULL;
    else
    {
        BiNode *newltre,*newrtre,*newnode;
        newltre=CopyTree(root->lchild);
        newrtre=CopyTree(root->rchild);
        newnode=new BiNode;
        newnode->data=root->data;
        newnode->lchild=newltre;
        newnode->rchild=newrtre;
        return newnode;
    }
}

void PostOrder(BiNode * root)
{
    if (root != NULL)
    {
        cout<<root->data;
        PostOrder(root->rchild);
        PostOrder(root->lchild);
    }
}
```

### 1. T1/2/8-Bitree\_main.cpp

```
#include <iostream>
#include "Bitree.cpp"
using namespace std;

int main()
```

```

{
    Bitree T;
    T.isEmpty();
    T.interface();
    T.isEmpty();
    cout<<"-----preorder----- ";<<endl;T.PreOrder();
    cout<<endl;
    cout<<"-----inorder----- ";<<endl;T.InOrder();
    cout<<endl;
    cout<<"-----postorder----- ";<<endl;T.PostOrder();
    cout<<endl;
    cout<<"-----leverorder----- ";<<endl;T.LeverOrder();
    cout<<endl;

    cout<<"-----COPY----- ";<<endl;
    BiNode * Troot = new BiNode;
    Troot = CopyTree(T.getRoot());
    Bitree Tc(Troot);
    cout<<"-----preorder----- ";<<endl;Tc.PreOrder();
    cout<<endl;
    cout<<"-----inorder----- ";<<endl;Tc.InOrder();
    cout<<endl;
    cout<<"-----postorder----- ";<<endl;Tc.PostOrder();
    cout<<endl;
    cout<<"-----leverorder----- ";<<endl;Tc.LeverOrder();
    cout<<endl;

    cout<<"-----REVERSE----- ";<<endl;
    PostOrder(T.getRoot());
    return 0;
}

```

## 2. T3-similar-Bitree.h

```

#ifndef Bitree_H
#define Bitree_H
const int MaxSize=100;
struct BiNode
{
    char data;
    BiNode *lchild, *rchild;
};

class Bitree

```

```
{
public:
    Bitree();
    Bitree(BiNode * bt);
    ~Bitree();
    BiNode * getRoot();
    BiNode * Creat(BiNode *bt);
    void isEmpty();
    void PreOrder();
    void InOrder();
    void PostOrder();
    void LeverOrder();
    void Release(BiNode *bt);
    void PreOrder(BiNode *bt);
    void InOrder(BiNode *bt);
    void PostOrder(BiNode *bt);
private:
    BiNode * root;
};
#endif
```

## 2. T3-similar-Bitree.cpp

```
#include <iostream>
using namespace std;
#include "Bitree.h"
Bitree::Bitree()
{
    root = Creat(root);
}

Bitree::Bitree(BiNode *bt)
{
    root = bt;
}

Bitree::~~Bitree()
{
    Release(root);
}

void Bitree::isEmpty()
{
    if (root == NULL)
```



```
{
    cout<<"empty"<<endl;
}
else
{
    cout<<"not empty"<<endl;
}
}

BiNode *Bitree::getRoot()
{
    return root;
}

void Bitree::PreOrder()
{
    PreOrder(root);
}

void Bitree::InOrder()
{
    InOrder(root);
}

void Bitree::PostOrder()
{
    PostOrder(root);
}

BiNode *Bitree::Creat(BiNode *bt)
{
    char ch;
    cout<<"请输入创建一棵二叉树的结点数据"<<endl;
    cin>>ch;
    if (ch=='#') return NULL;
    else
    {
        bt = new BiNode;
        bt -> data=ch;
        bt -> lchild = Creat(bt -> lchild);
        bt -> rchild = Creat(bt -> rchild);
    }
    return bt;
}
```

```
void Bitree::Release(BiNode *bt)
{
    if (bt != NULL)
    {
        Release(bt -> lchild);    //释放左子树
        Release(bt -> rchild);    //释放右子树
        delete bt;
    }
}

void Bitree::PreOrder(BiNode *bt)
{
    if(bt==NULL) return;
    else
    {
        cout<<bt -> data<<" ";
        PreOrder(bt -> lchild);
        PreOrder(bt -> rchild);
    }
}

void Bitree::InOrder(BiNode *bt)
{
    if (bt==NULL) return;
    else
    {
        InOrder(bt -> lchild);
        cout<<bt -> data<<" ";
        InOrder(bt -> rchild);
    }
}

void Bitree::PostOrder(BiNode *bt)
{
    if (bt==NULL) return;
    else
    {
        PostOrder(bt -> lchild);
        PostOrder(bt -> rchild);
        cout<<bt -> data<<" ";
    }
}
```

```
void Bitree::LeverOrder( )
{
    int front = -1, rear = -1;
    BiNode *Q[MaxSize], *q;
    if (root == NULL) return;
    else
    {
        Q[++rear] = root;
        while (front != rear)
        {
            q=Q[++front];
            cout<<q -> data<<" ";
            if (q -> lchild != NULL)
                Q[++rear] = q -> lchild;
            if (q -> rchild != NULL)
                Q[++rear] = q -> rchild;
        }
    }
}

int Like(BiNode * s, BiNode * t)
{
    if (s == NULL && t == NULL) return 1;
    else if ((s == NULL && t != NULL)|| (s != NULL && t == NULL))
        return 0;
    else
    {
        int same = Like(s -> lchild, t -> rchild);
        if(same) same = Like(s -> rchild, t -> rchild);
        return same;
    }
}
```

## 2. T3-similar-Like.cpp

```
#include <iostream>
using namespace std;
#include "Bitree.cpp"

int main()
{
    Bitree T; //创建一棵树
    Bitree R;
    if (Like(T.getRoot(), R.getRoot())) cout<<"similar"<<endl;
```

```
else cout<<"not similar"<<endl;
return 0;
}
```

### 3. T4-path-Bitree.h

```
#ifndef Bitree_H
#define Bitree_H
const int MaxSize=100;
struct BiNode
{
    char data;
    BiNode *lchild, *rchild;
};

class Bitree
{
public:
    Bitree();
    Bitree(BiNode * bt);
    ~Bitree();
    BiNode * getRoot();
    BiNode * Creat(BiNode *bt);
    void isEmpty();
    void PreOrder();
    void InOrder();
    void PostOrder();
    void LeverOrder();
    void Release(BiNode *bt);
    void PreOrder(BiNode *bt);
    void InOrder(BiNode *bt);
    void PostOrder(BiNode *bt);
    BiNode * getleftleaf(BiNode * root);
    void helper(BiNode * node, int dpt, int &max_dpt, BiNode* &res);
private:
    BiNode * root;
};
#endif
```

### 3. T4-path-Bitree.cpp

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

```
#include "Bitree.h"
Bitree::Bitree()
{
    root = Creat(root);
}

Bitree::Bitree(BiNode *bt)
{
    root = bt;
}

Bitree::~~Bitree()
{
    Release(root);
}

void Bitree::isEmpty()
{
    if (root == NULL)
    {
        cout<<"empty"<<endl;
    }
    else
    {
        cout<<"not empty"<<endl;
    }
}

BiNode *Bitree::getRoot()
{
    return root;
}

void Bitree::PreOrder()
{
    PreOrder(root);
}

void Bitree::InOrder()
{
    InOrder(root);
}

void Bitree::PostOrder()
```

```
{
    PostOrder(root);
}

BiNode *Bitree::Creat(BiNode *bt)
{
    char ch;
    cout<<"请输入创建一棵二叉树的结点数据"<<endl;
    cin>>ch;
    if (ch=='#') return NULL;
    else
    {
        bt = new BiNode;
        bt -> data=ch;
        bt -> lchild = Creat(bt -> lchild);
        bt -> rchild = Creat(bt -> rchild);
    }
    return bt;
}

void Bitree::Release(BiNode *bt)
{
    if (bt != NULL)
    {
        Release(bt -> lchild); //释放左子树
        Release(bt -> rchild); //释放右子树
        delete bt;
    }
}

void Bitree::PreOrder(BiNode *bt)
{
    if(bt==NULL) return;
    else
    {
        cout<<bt -> data<<" ";
        PreOrder(bt -> lchild);
        PreOrder(bt -> rchild);
    }
}

void Bitree::InOrder(BiNode *bt)
{
    if (bt==NULL) return;
```

```
        else
        {
            InOrder(bt -> lchild);
            cout<<bt -> data<<" ";
            InOrder(bt -> rchild);
        }
    }

void Bitree::PostOrder(BiNode *bt)
{
    if (bt==NULL) return;
    else
    {
        PostOrder(bt -> lchild);
        PostOrder(bt -> rchild);
        cout<<bt -> data<<" ";
    }
}

void Bitree::LeverOrder( )
{
    int front = -1, rear = -1;
    BiNode *Q[MaxSize], *q;
    if (root == NULL) return;
    else
    {
        Q[++rear] = root;
        while (front != rear)
        {
            q=Q[++front];
            cout<<q -> data<<" ";
            if (q -> lchild != NULL)
                Q[++rear] = q -> lchild;
            if (q -> rchild != NULL)
                Q[++rear] = q -> rchild;
        }
    }
}

BiNode *Bitree::getleftleaf(BiNode * root)
{
    int max_dpt = 1;
    BiNode * res = root;
    helper(root, 1, max_dpt, res);
}
```

```
        return res;
    }

void Bitree::helper(BiNode *node, int dpt, int &max_dpt, BiNode *&res)
{
    if (!node) return;
    if (dpt > max_dpt)
    {
        max_dpt = dpt;
        res = node;
    }
    helper(node->lchild, dpt+1, max_dpt, res);
    helper(node->rchild, dpt+1, max_dpt, res);
}

void Path(BiNode * root, BiNode * p)
{
    BiNode * stack[MaxSize] = {NULL};
    int tag[MaxSize] = {0};
    int top = -1;
    BiNode * T = root;
    while (T != NULL || top != -1)
    {
        while (T != NULL)
        {
            top++;
            stack[top] = T;
            tag[top] = 0;
            T = T->lchild;
        }
        while (top != -1 && tag[top] == 1)
        {
            T = stack[top];
            if (T == p)
            {
                for (int i = 0; i < top; i++)
                {
                    cout<<stack[i]->data;
                    if (i == top - 1)
                    {
                        cout<<p->data;
                        return;
                    }
                }
            }
            tag[top] = 1;
            T = T->rchild;
        }
    }
}
```



```
        }
    }
    break;
}
else top--;
}
if (top != -1)
{
    T = stack[top]->rchild;
    tag[top] = 1;
}
else return;
}
}
```

### 3. T4-path-Bitree\_main.cpp

```
#include <iostream>
using namespace std;
#include "Bitree.cpp"

int main()
{
    Bitree T;
    BiNode * root = T.getRoot();
    BiNode * test = T.getleftleaf(root);
    Path(root, test);
    return 0;
}
```

### 4. T5/6/7-Bitree.h

```
#ifndef Bitree_H
#define Bitree_H
const int MaxSize=100;
struct BiNode
{
    char data;
    BiNode *lchild, *rchild;
};

class Bitree
{
```

```
public:
    Bitree();
    Bitree(BiNode * bt);
    ~Bitree();
    BiNode * getRoot();
    BiNode * Creat(BiNode *bt);
    void isEmpty();
    void PreOrder();
    void InOrder();
    void PostOrder();
    void LeverOrder();
    void Release(BiNode *bt);
    void PreOrder(BiNode *bt);
    void InOrder(BiNode *bt);
    void PostOrder(BiNode *bt);
    void countLeaves(BiNode * root, int &num);
    int countLeaves();
    int countNode();
private:
    BiNode * root;
    void countNode(BiNode * root, int &num);
};
#endif
```

#### 4. T5/6/7-Bitree.cpp

```
#include <iostream>
using namespace std;
#include "Bitree.h"
Bitree::Bitree()
{
    root = Creat(root);
}

Bitree::Bitree(BiNode *bt)
{
    root = bt;
}

Bitree::~Bitree()
{
    Release(root);
}
```

```
void Bitree::isEmpty()
{
    if (root == NULL)
    {
        cout<<"empty"<<endl;
    }
    else
    {
        cout<<"not empty"<<endl;
    }
}

BiNode *Bitree::getRoot()
{
    return root;
}

void Bitree::PreOrder()
{
    PreOrder(root);
}

void Bitree::InOrder()
{
    InOrder(root);
}

void Bitree::PostOrder()
{
    PostOrder(root);
}

BiNode *Bitree::Creat(BiNode *bt)
{
    char ch;
    cout<<"请输入创建一棵二叉树的结点数据"<<endl;
    cin>>ch;
    if (ch=='#') return NULL;
    else
    {
        bt = new BiNode;
        bt -> data=ch;
        bt -> lchild = Creat(bt -> lchild);
        bt -> rchild = Creat(bt -> rchild);
    }
}
```

```
    }  
    return bt;  
}  
void Bitree::Release(BiNode *bt)  
{  
    if (bt != NULL)  
    {  
        Release(bt -> lchild);    //释放左子树  
        Release(bt -> rchild);    //释放右子树  
        delete bt;  
    }  
}  
  
void Bitree::PreOrder(BiNode *bt)  
{  
    if(bt==NULL) return;  
    else  
    {  
        cout<<bt -> data<<" ";  
        PreOrder(bt -> lchild);  
        PreOrder(bt -> rchild);  
    }  
}  
  
void Bitree::InOrder(BiNode *bt)  
{  
    if (bt==NULL) return;  
    else  
    {  
        InOrder(bt -> lchild);  
        cout<<bt -> data<<" ";  
        InOrder(bt -> rchild);  
    }  
}  
  
void Bitree::PostOrder(BiNode *bt)  
{  
    if (bt==NULL) return;  
    else  
    {  
        PostOrder(bt -> lchild);  
        PostOrder(bt -> rchild);  
        cout<<bt -> data<<" ";  
    }  
}
```

```
    }  
}  
  
void Bitree::LeverOrder( )  
{  
    int front = -1, rear = -1;  
    BiNode *Q[MaxSize], *q;  
    if (root == NULL) return;  
    else  
    {  
        Q[++rear] = root;  
        while (front != rear)  
        {  
            q=Q[++front];  
            cout<<q -> data<<" ";  
            if (q -> lchild != NULL)  
                Q[++rear] = q -> lchild;  
            if (q -> rchild != NULL)  
                Q[++rear] = q -> rchild;  
        }  
    }  
}  
  
int Bitree::countLeaves()  
{  
    int num = 0;  
    if (root == NULL) return 0;  
    else if (root != NULL)  
    {  
        countLeaves(root, num);  
    }  
    else  
        num++;  
    return num;  
}  
  
void Bitree::countLeaves(BiNode* root, int &num) // 用引用的形式参数传递  
{  
    //前置条件: root 不是空指针  
    if (root->lchild == NULL && root->rchild == NULL)  
    {  
        num++;  
    }  
    else
```

```
{
    if(root -> lchild != NULL)
    {
        countLeaves(root -> lchild, num);
    }
    if (root -> rchild != NULL)
    {
        countLeaves(root -> rchild, num);
    }
}
}

void leavesPrint(BiNode * root)
{
    if (root != NULL)
    {
        if (!root->lchild && !root->rchild)
        {
            cout<<root->data;
        }
        leavesPrint(root -> lchild);
        leavesPrint(root -> rchild);
    }
}

BiNode *CopyTree(BiNode *root)
{
    if(root==NULL) return NULL;
    else
    {
        BiNode *newltre,*newrtre,*newnode;
        newltre=CopyTree(root->lchild);
        newrtre=CopyTree(root->rchild);
        newnode=new BiNode;
        newnode->data=root->data;
        newnode->lchild=newltre;
        newnode->rchild=newrtre;
        return newnode;
    }
}

int Bitree::countNode()
{
    int num = 0;
```

```
    countNode(root, num);
    return num;
}

void Bitree::countNode(BiNode *root, int &num)
{
    if (root)
    {
        num++;
        countNode(root->lchild, num);
        countNode(root->rchild, num);
    }
}

int max(int a, int b)
{
    if (a > b) return a;
    else return b;
}

int Depth(BiNode * root)
{
    if (root == NULL) return 0;
    else
    {
        int h1 = Depth(root->lchild);
        int hr = Depth(root->rchild);
        return max(h1, hr) + 1;
    }
}
```

#### 4. T5/6/7-Bitree\_main.cpp

```
#include <iostream>
using namespace std;
#include "Bitree.cpp"

int main()
{
    Bitree T; //创建一棵树
    cout<<"-----前序遍历----- "<<endl;
    T.PreOrder();
    cout<<endl;
    cout<<"-----中序遍历----- "<<endl;
```

```
T.InOrder();
cout<<endl;
cout<<"-----后序遍历----- "<<endl;
T.PostOrder();
cout<<endl;
cout<<"-----层序遍历----- "<<endl;
T.LeverOrder();
cout<<endl;
int n = T.countLeaves();
cout<<"叶子数为: "<<n<<endl;
cout<<"叶子结点为: ";leavesPrint(T.getRoot());
cout<<endl;
int n1 = T.countNode();
cout<<"节点数为: "<<n1<<endl;
int n2 = Depth(T.getRoot());
cout<<"深度为: "<<n2<<endl;
return 0;
}
```

## 5. T9/10/11-Bitree.h

```
#ifndef Bitree_H
#define Bitree_H
const int MaxSize=100;
struct BiNode //二叉树的结点结构
{
    char data;
    BiNode *lchild, *rchild;
};

class Bitree
{
public:
    Bitree();
    Bitree(BiNode * bt);
    ~Bitree();
    BiNode * getRoot();
    BiNode * Creat(BiNode *bt);
    void isEmpty();
    void PreOrder();
    void InOrder();
    void PostOrder();
    void LeverOrder();
    void Release(BiNode *bt);
};
```



```
void PreOrder(BiNode *bt);
void InOrder(BiNode *bt);
void PostOrder(BiNode *bt);
void info(BiNode * bt);
void del(char x);
private:
    BiNode * root;
    void Delete(BiNode * root, char x, BiNode * &p);
};
#endif
```

## 5. T9/10/11-Bitree.cpp

```
#include <iostream>
using namespace std;
#include "Bitree.h"
Bitree::Bitree()
{
    root = Creat(root);
}

Bitree::Bitree(BiNode *bt)
{
    root = bt;
}

Bitree::~~Bitree()
{
    Release(root);
}

void Bitree::PreOrder()
{
    PreOrder(root);
}

void Bitree::InOrder()
{
    InOrder(root);
}

void Bitree::PostOrder()
{
    PostOrder(root);
}
```

```
}

BiNode *Bitree::getRoot()
{
    return root;
}

BiNode *Bitree::Creat(BiNode *bt)
{
    char ch;
    cout<<"请输入创建一棵二叉树的结点数据"<<endl;
    cin>>ch;
    if (ch=='#') return NULL;
    else
    {
        bt = new BiNode;
        bt -> data=ch;
        bt -> lchild = Creat(bt -> lchild);
        bt -> rchild = Creat(bt -> rchild);
    }
    return bt;
}

void Bitree::Release(BiNode *bt)
{
    if (bt != NULL)
    {
        Release(bt -> lchild); //释放左子树
        Release(bt -> rchild); //释放右子树
        delete bt;
    }
}

void Bitree::PreOrder(BiNode *bt)
{
    if(bt == NULL) return;
    else
    {
        cout<<bt -> data<<" ";
        PreOrder(bt -> lchild);
        PreOrder(bt -> rchild);
    }
}
```

```
void Bitree::InOrder(BiNode *bt)
{
    if (bt == NULL) return;
    else
    {
        InOrder(bt -> lchild);
        cout<<bt -> data<<" ";
        InOrder(bt -> rchild);
    }
}

void Bitree::PostOrder(BiNode *bt)
{
    if (bt == NULL) return;
    else
    {
        PostOrder(bt -> lchild);
        PostOrder(bt -> rchild);
        cout<<bt -> data<<" ";
    }
}

void Bitree::LeverOrder( )
{
    int front = -1, rear = -1;
    BiNode *Q[MaxSize], *q;
    if (root==NULL) return;
    else
    {
        Q[++rear]=root;
        while (front != rear)
        {
            q = Q[++front];
            cout<<q -> data<<" ";
            if (q -> lchild != NULL)
                Q[++rear]=q -> lchild;
            if (q -> rchild != NULL)
                Q[++rear] = q -> rchild;
        }
    }
}

BiNode *CopyTree(BiNode *root)
```

```
{
    if(root==NULL) return NULL;
    else
    {
        BiNode *newltre,*newrtre,*newnode;
        newltre=CopyTree(root->lchild);
        newrtre=CopyTree(root->rchild);
        newnode=new BiNode;
        newnode->data=root->data;
        newnode->lchild=newltre;
        newnode->rchild=newrtre;
        return newnode;
    }
}

void Parent(BiNode * root, char x, BiNode * &p)
{
    if (root != NULL)
    {
        if (root->data == x) p = NULL;
        else
        {
            p = root;
            Parent(root -> lchild, x, p);
            if(p == NULL) // 左子树没有找到
            {
                Parent(root -> rchild, x, p);
            }
        }
    }
}

void Bitree::info(BiNode *bt)
{
    if (bt) cout<<"该节点: "<<bt -> data;
    else cout<<"空节点 ";
}

void Bitree::del(char x)
{
    BiNode* p = NULL;
    Delete(root, x, p);
}
```

```
void Bitree::Delete(BiNode *bt, char x, BiNode *&p)
{
    if (bt != NULL)
    {
        if (bt->data == x)
        {
            if (p == NULL) bt = NULL;
            else if (p->lchild == bt) p->lchild = NULL;
            else p->rchild = NULL;
        }
        else
        {
            p = bt;
            Delete(bt->lchild, x, p);
            Delete(bt->rchild, x, p);
        }
    }
}

void Exchange(BiNode * root)
{
    if (root != NULL)
    {
        Exchange(root->lchild);
        Exchange(root->rchild);
        BiNode * temp = NULL;
        temp = root->lchild;
        root->lchild = root->rchild;
        root->rchild = temp;
    }
}
```

## 5. T9/10/11-main.cpp

```
#include <iostream>
using namespace std;
#include "Bitree.cpp"

int main()
{
    Bitree T; //创建一棵树
    BiNode * p = NULL;
    cout<<"-----前序遍历----- "<<endl;T.PreOrder();
    cout<<endl;
```

```
cout<<"-----中序遍历----- "<<endl;T.InOrder();
cout<<endl;
cout<<"-----后序遍历----- "<<endl;T.PostOrder();
cout<<endl;
cout<<"-----层序遍历----- "<<endl;T.LeverOrder();
//T12
cout<<endl;
char test = '\0';
cout<<"输入查找节点: "<<endl;
cin>>test;
Parent(T.getRoot(), test, p);T.info(p);
cout<<"为双亲节点."<<endl;
//T14
cout<<endl;
Exchange(T.getRoot());
cout<<"-----前序遍历----- "<<endl;T.PreOrder();
cout<<endl;
cout<<"-----中序遍历----- "<<endl;T.InOrder();
cout<<endl;
cout<<"-----后序遍历----- "<<endl;T.PostOrder();
cout<<endl;
cout<<"-----层序遍历----- "<<endl;T.LeverOrder();
cout<<endl;
//T13
cout<<endl;
T.del('D');
T.InOrder();
return 0;
}
```

## 6. T12-main.cpp

```
#include <iostream>
using namespace std;
const int MaxSize = 100;
void PreOrder(char * A, int n)
{
    int S[MaxSize] = {0};
    int top = -1;
    int i = 1;
    cout<<A[i - 1];
    S[++top] = i;
    int j = 2*i;
    while (A[j - 1] != '#' || top != -1)
```

```
{
    while (j <= n && (A[j - 1] != '#'))
    {
        cout<<A[j - 1];
        S[++top] = j;
        i = j;
        j = 2*i;
    }
    i = S[top--];
    j = 2 * i + 1;
}
}

int main()
{
    int n;
    cin>>n;
    char seq[n] = {'#'};
    cin>>seq;
    PreOrder(seq, n);
    return 0;
}
```

## 7. T13-Tree.h

```
#ifndef Tree_H
#define Tree_H
const int Max = 20;
struct TNode
{
    char data;
    TNode *firstchild, *rightsib;
};

class Tree
{
public:
    Tree();
    ~Tree();           //析构函数，释放各结点的存储空间
    void PreOrder();
    void PostOrder();
    TNode * getRoot();
    void info(TNode * bt);
};
```

```
private:
    TNode *root;
    void PreOrder(TNode *bt);
    void PostOrder(TNode *bt);
    void Release(TNode *bt);
};
#endif
```

## 7. T13-Tree.cpp

```
#include <iostream>
using namespace std;
#include "Tree.h"

Tree::Tree()
{
    TNode *Q[Max] = {NULL};
    char ch1 = '#', ch2 = '#';
    int front = -1, rear = -1;
    TNode *p = NULL, *q = NULL;
    cout<<"请输入根结点: ";
    cin>>ch1;
    p = new TNode; p -> data = ch1;
    p -> firstchild = p -> rightsib = NULL;
    root = p;
    Q[++rear] = p;
    cout<<"请输入结点对, 以空格分隔: ";
    fflush(stdin);
    ch1 = getchar(); getchar(); ch2 = getchar();
    while (ch1 != '#' || ch2 != '#')
    {
        p = new TNode; p -> data = ch2;
        p -> firstchild = p -> rightsib = NULL;
        Q[++rear] = p;
        while (front < rear)
        {
            q = Q[front + 1];
            if (q -> data != ch1)
                front++;
            else
            {
                if (q -> firstchild == NULL)
                    q -> firstchild = p;
            }
        }
    }
}
```



```
        else
        {
            q = q -> firstchild;
            while (q -> rightsib != NULL)
            {
                q = q -> rightsib;
            }
            q -> rightsib = p;
        }
        break;
    }
}

cout<<"请输入结点对，以空格分隔：";
fflush(stdin); // 再次清空键盘缓冲区
ch1 = getchar(); getchar(); ch2 = getchar();
}

Tree::~Tree()
{
    Release(root);
}

void Tree::PreOrder()
{
    PreOrder(root);
}

void Tree::PostOrder()
{
    PostOrder(root);
}

TNode *Tree::getRoot()
{
    return root;
}

void Tree::Release(TNode *bt)
{
    if (bt == NULL) return;
    else
    {
        Release(bt->firstchild);
    }
}
```

```
        Release(bt->rightsib);
        delete bt;
    }
}

void Tree::PreOrder(TNode *bt)
{
    if (bt == NULL) return;
    else
    {
        cout<<bt->data;
        PreOrder(bt->firstchild);
        PreOrder(bt->rightsib);
    }
}

void Tree::PostOrder(TNode *bt)
{
    if (bt == NULL) return;
    else
    {
        PostOrder(bt->firstchild);

        cout<<bt->data;
        PostOrder(bt->rightsib);
    }
}

TNode * Search(TNode * root, char x, int i)
{
    TNode * p = NULL;
    if (root->data == x)
    {
        int j = 1;
        p = root -> firstchild;
        while (p != NULL && j < i)
        {
            j++;
            p = p -> rightsib;
        }
        if (p != NULL) return p;
        else return NULL;
    }
    Search(root->firstchild, x, i);
}
```

```
    if (p != NULL) Search(root->rightsib, x, i);
}

void Tree::info(TNode *bt)
{
    if (bt) cout<<"该节点: "<<bt->data<<endl;
    else cout<<"不存在."<<endl;
}
```

## 7. T13-Tree\_main.cpp

```
#include <iostream>
using namespace std;
#include "Tree.cpp"

int main()
{
    Tree t1;
    t1.PreOrder();
    cout<<endl;
    t1.PostOrder();
    cout<<endl;
    char test = '\0';
    int num = 0;
    cout<<"输入查找节点: ";
    cin>>test;
    cout<<"输入查找位置: ";
    cin>>num;
    TNode* ans = Search(t1.getRoot(), test, num);
    t1.info(ans);
    return 0;
}
```

#### 四、运行与测试

##### 1. T1/2/8

```
empty
输入序列长度
8
输入前序序列
ABCDEFGH
输入中序序列
CDBAFEHG
not empty
-----前序遍历-----
A B C D E F G H
-----中序遍历-----
C D B A F E H G
-----后序遍历-----
D C B F H G E A
-----层序遍历-----
A B E C F G D H
-----COPY-----
-----前序遍历-----
A B C D E F G H
-----中序遍历-----
C D B A F E H G
-----后序遍历-----
D C B F H G E A
-----层序遍历-----
A B E C F G D H
请按任意键继续. . .
```

## 2. T3-similar

```
请输入创建一棵二叉树的结点数据
ABC####
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
AB#D##C##
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
not similar
-----
Process exited after 22.93 seconds with return value 0
请按任意键继续. . .
```

## 3. T4-path

```
请输入创建一棵二叉树的结点数据
ABE##D##C##
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
ABE
-----
Process exited after 18.78 seconds with return value 0
请按任意键继续. . .
```

## 4. T5/6/7

```

请输入创建一棵二叉树的结点数据
ABE##D##C##
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
-----前序遍历-----
A B E D C
-----中序遍历-----
E B D A C
-----后序遍历-----
E D B C A
-----层序遍历-----
A B C E D
叶子数为：3
叶子结点为：EDC
节点数为：5
深度为：3
请按任意键继续. . .

```

## 5. T9/10/11

```

请输入创建一棵二叉树的结点数据
ABE##D##C##
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
请输入创建一棵二叉树的结点数据
-----前序遍历-----
A B E D C
-----中序遍历-----
E B D A C

```

```
-----后序遍历-----
E D B C A
-----层序遍历-----
A B C E D
输入查找节点:
D
该节点: E为双亲节点.

-----EXCHANGE-----
-----前序遍历-----
A C B D E
-----中序遍历-----
C A D B E
-----后序遍历-----
C D E B A
-----层序遍历-----
A C B D E

-----DELETE-----
C A B E
-----
Process exited after 16.13 seconds with return value 0
请按任意键继续. . .
```

## 6. T12

```
15
ABC#DE###F##G##
ABDFCEG
-----
Process exited after 34.32 seconds with return value 0
请按任意键继续. . .
```

## 7. T13

```
请输入根结点: A
请输入结点对, 以空格分隔: A B
请输入结点对, 以空格分隔: B D
请输入结点对, 以空格分隔: B E
请输入结点对, 以空格分隔: B F
请输入结点对, 以空格分隔: E I
请输入结点对, 以空格分隔: A C
请输入结点对, 以空格分隔: C G
请输入结点对, 以空格分隔: C H
请输入结点对, 以空格分隔: # #
```

```
ABDEIF
```

```
DIEFBA
```

```
输入查找节点: B
```

```
输入查找位置: 3
```

```
该节点: F
```

```
-----
Process exited after 34.1 seconds with return value 0
请按任意键继续. . .
```

## 五、总结与心得

前面的几个设计实验可以总结为在二叉树的基础上实现一些操作，其实编程的时候应该把这几个实验的代码写到一个项目中，这样调试或者打印输出结果的时候会方便很多。后面的几个实验在编程时吸取了教训，尽量把重复率较高的程序题目写在一起，一定程度上节约了时间，同时也达到了训练的目的。

二叉树的建立我使用的书上的前序遍历（带 #）的方法来建立，用得是逐个输入，实际上也可以直接读入一串，再逐个存储建立二叉树。无论是哪种编译器均支持这样输入和建立二叉树（树）。

倒数第二题为了方便起见依旧使用键盘键入 n，然后根据 n 来建立数组的方式，将所有代码写在同一个程序里，因此这一题只能在 DEV C++ 环境下运行。如果想要在 VSCode 环境下运行则需要将代码修改成键盘键入 n 后通过 new 建立临时数组空间，再进行接下来的操作。