南京信息工程大学__数据结构Ⅰ__实验(实习)报告

实验（实习）名称__二叉树的遍历__日期__2024.11.__得分_____指导教师_____

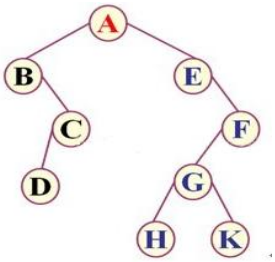
学院__计院__专业__计科_____

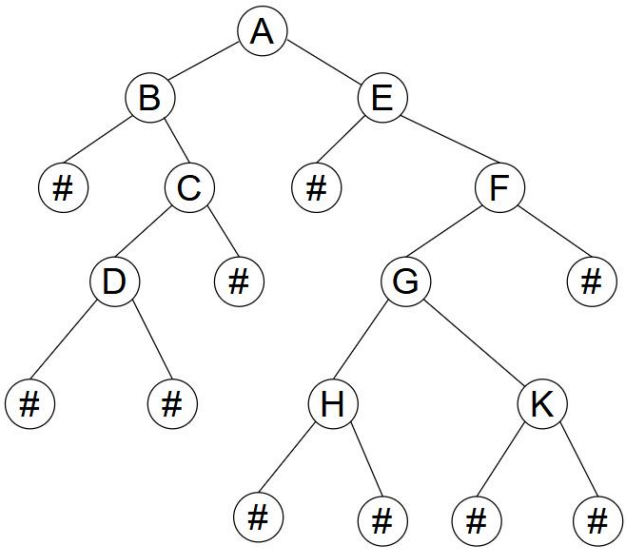
一、实验目的

1、了解二叉树的定义和递归的思想；
2、掌握二叉树的链式存储结构；
3、掌握建立二叉树的基本算法；
4、掌握二叉树遍历的基本操作，并对其进行简单应用。

二、实验内容与步骤

1、对下面给定的二叉树补足空结点，并画出补充后的二叉树（度为 1 的结点只需补 1 个空结点；度为 0 的结点需要补 2 个结点；度为 2 的结点不需要补空结点）。





2、写出对补充过空结点的二叉树进行先序遍历的结点序列（空结点用‘#’表示）。
AB#CD###E#FGH##K###

3、采用二叉链表存储结构（数据类型定义见教材 P127）根据上题得到的先序遍历结点序列，递归建立该二叉树（即实现教材算法 6.4）。

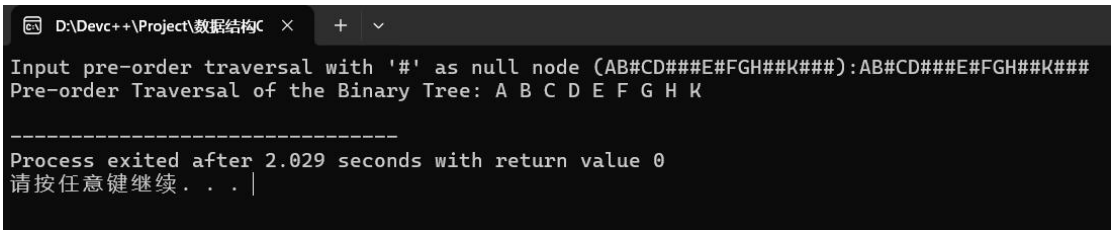
实验代码：

```
#include<iostream>
#include<cstdlib>
```

```
using namespace std;

typedef enum PointerTag { Link, Thread } PointerTag;
typedef char TElemType;

typedef struct BiThrNode {
	TElemType data;
	struct BiThrNode* lchild, * rchild;
	PointerTag LTag, RTag;
} BiThrNode, * BiThrTree;

// 构建二叉树的递归函数
void CreateBiTree(BiThrTree& T) {
   char ch;
   cin>>ch;
   if (ch=='#')
   {
           T=NULL; // 如果输入为'#'，表示空节点
   }
   else{
        T = (BiThrNode*)malloc(sizeof(BiThrNode)); // 动态分配节点内存
          if (!T) exit(EXIT_FAILURE); // 检查内存分配
          T->data = ch; // 保存节点数据
          T->LTag = Link;
          T->RTag = Link;
          CreateBiTree(T->lchild); // 递归创建左子树
          CreateBiTree(T->rchild); // 递归创建右子树
   }
}

// 先序遍历输出
void PreOrderTraverse(BiThrTree T) {
	if (T != NULL) {
		cout << T->data << " "; // 访问当前结点
		PreOrderTraverse(T->lchild); // 递归遍历左子树
		PreOrderTraverse(T->rchild); // 递归遍历右子树
	}
}
// 主函数
int main() {
	BiThrTree T = NULL;
	cout << "Input pre-order traversal with '#' as null node (AB#CD###E#FGH##K###):";
	CreateBiTree(T); // 构建二叉树
	cout << "Pre-order Traversal of the Binary Tree: ";
```

```
    PreOrderTraverse(T); //  先序遍历输出
    cout << endl;
    return 0;
}
```

运行结果：

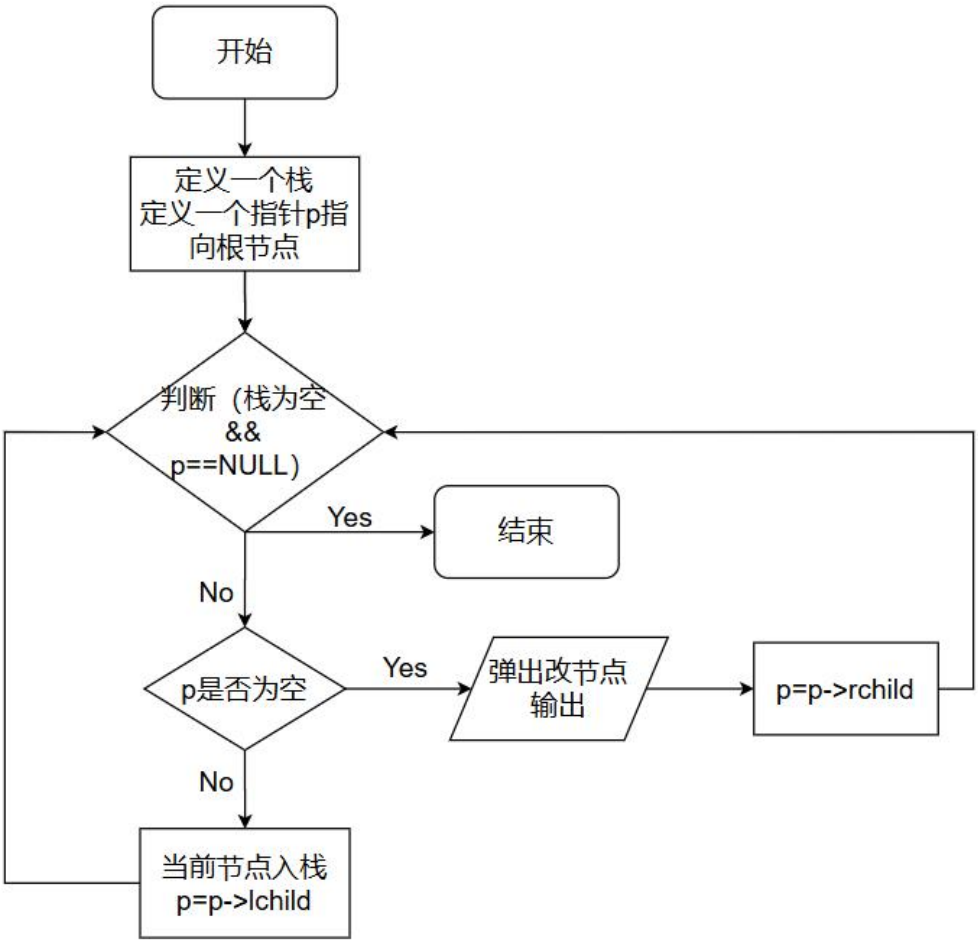


4、分别用递归和非递归算法中序遍历该二叉树，并输出中序遍历的结果（其中递归算法仿照教材算法 6.1、非递归算法见教材算法 6.3）。

（1）写出非递归遍历二叉树算法的基本思路

定义一个栈，用于存储二叉树的节点。定义一个当前指针 p，初始指向根节点。当当前节点 p 不为空时，将其入栈，并移动到左子树（即 p = p->lchild）。当当前节点为空时，从栈中弹出节点，访问该节点，将当前节点指向其右子树（即 p = p->rchild）。当栈为空且当前节点为 NULL 时，遍历结束。

（2）（在电脑里）画出非递归遍历算法的程序流程图



（3）完整的程序源代码（文本）及运行结果（截图）

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

typedef enum PointerTag { Link, Thread } PointerTag;
typedef char TElemType;

typedef struct BiThrNode {
    TElemType data;
    struct BiThrNode* lchild, * rchild;
    PointerTag LTag, RTag;
} BiThrNode, * BiThrTree;

// 构建二叉树的递归函数
void CreateBiTree(BiThrTree& T) {
  char ch;
  cin>>ch;
  if (ch=='#')
  {
        T=NULL; // 如果输入为'#'，表示空节点
  }
  else{
      T = (BiThrNode*)malloc(sizeof(BiThrNode)); // 动态分配节点内存
        if (!T) exit(EXIT_FAILURE); // 检查内存分配
        T->data = ch; // 保存节点数据
        T->LTag = Link;
        T->RTag = Link;
        CreateBiTree(T->lchild); // 递归创建左子树
        CreateBiTree(T->rchild); // 递归创建右子树
  }
}

// 中序遍历（递归）
void InOrderTraverseRecursive(BiThrTree T) {
    if (T != NULL) {
        InOrderTraverseRecursive(T->lchild); // 遍历左子树
        cout << T->data << " "; // 访问当前节点
        InOrderTraverseRecursive(T->rchild); // 遍历右子树
    }
}

// 中序遍历（非递归）
void InOrderTraverseNonRecursive(BiThrTree T) {
```

```
    stack<BiThrNode*> s; // 使用栈
    BiThrNode* p = T; // 当前节点指针

    while (p != NULL || !s.empty()) {
        while (p != NULL) { // 一直向左走，将节点压入栈
            s.push(p);
            p = p->lchild;
        }
        if (!s.empty()) {
            p = s.top(); // 取栈顶元素
            s.pop();
            cout << p->data << " "; // 访问节点
            p = p->rchild; // 转向右子树
        }
    }
}
// 主函数
int main() {
    BiThrTree T = NULL;

    cout << "Input pre-order traversal with '#' as null node (AB#CD###E#FGH##K###):" << endl;
    CreateBiTree(T); // 构建二叉树

    cout << "In-order Traversal (递归): ";
    InOrderTraverseRecursive(T); // 递归中序遍历
    cout << endl;

    cout << "In-order Traversal (非递归): ";
    InOrderTraverseNonRecursive(T); // 非递归中序遍历
    cout << endl;

    return 0;
}
```

三、实验心得（必写）

通过本次实验，我深入理解了二叉树的递归与非递归遍历方法，掌握了二叉链表的存储结构和基本操作。递归实现简单清晰，而非递归遍历通过栈的使用让我更加直观地理解了递

归的底层逻辑。本次实验加深了我对二叉树和算法实现的理解，收获颇丰。