实验五 二叉树

### (一) 实验目的

1. 学习掌握二叉树的建立、遍历等基本操作
2. 提高对树型结构的综合应用能力
3. 进一步掌握递归算法
4. 进一步掌握堆栈的应用
5. 进一步掌握队列的应用

### (二) 实验内容

**1. 熟悉二叉树的构建及遍历操作**

[**题目1**：构造二叉链表表示的二叉树：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；再输出三种遍历序列。本题只给出部分代码,请补全内容。]

#include <stdio.h>

#include <malloc.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int Status;

typedef char ElemType;

typedef struct BiTNode{

ElemType data;

struct BiTNode \*lchild,\*rchild;//左右孩子指针

} BiTNode,\*BiTree;

BiTree 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)))) return ERROR;

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 T;

} // CreateBiTree

Status PrintElement( ElemType e ) { // 输出元素e的值

printf("%c", e );

return OK;

}// PrintElement

Status PreOrderTraverse( BiTree T, Status(\*Visit)(ElemType) ) {

// 算法6.1

// 采用二叉链表存储结构，Visit是对数据元素操作的应用函数，

// 先序遍历二叉树T的递归算法，对每个数据元素调用函数Visit。

// 最简单的Visit函数是：

// Status PrintElement( ElemType e ) { // 输出元素e的值

// printf( e ); // 实用时，加上格式串

// return OK;

// }

// 调用实例：PreOrderTraverse(T, PrintElement);

if (T) {

if (Visit(T->data))

if (PreOrderTraverse(T->lchild, Visit))

if (PreOrderTraverse(T->rchild, Visit)) return OK;

return ERROR;

} else return OK;

} // PreOrderTraverse

Status InOrderTraverse( BiTree T, Status(\*Visit)(ElemType) ) {

// 中序遍历二叉树T的递归算法，对每个数据元素调用函数Visit。

if (T) {

if (InOrderTraverse(T->lchild, Visit))

if (Visit(T->data))

if (InOrderTraverse(T->rchild, Visit)) return OK;

return ERROR;

} else return OK;

} // InOrderTraverse

Status PostOrderTraverse( BiTree T, Status(\*Visit)(ElemType) ) {

// 中序遍历二叉树T的递归算法，对每个数据元素调用函数Visit。

if (T) {

if (PostOrderTraverse(T->lchild, Visit))

if (PostOrderTraverse(T->rchild, Visit))

if (Visit(T->data)) return OK;

return ERROR;

} else return OK;

} // PostOrderTraverse

void main() //主函数

{

//补充代码

}//main

**测试样例格式说明：**

[键盘输入]

第一行：输入一棵二叉树的先序遍历序列

[正确输出］

第一行：二叉树的先序遍历序列

第二行：二叉树的中序遍历序列

第三行：二叉树的后序遍历序列

**测试样例：**

[第一组自测数据]

[键盘输入]

ABC####↙

[正确输出]

ABC

BAC //CBA才对

BCA //CBA才对

[第二组自测数据]

[键盘输入]

ABD#E##FG###C#H##↙

[正确输出]

ABDEFGCH

CEBGFACH

EDGFBHCA

**完整代码:**

#include<stdio.h>

#include<malloc.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int Status;

typedef char ElemType;

typedef struct BiTNode

{

ElemType data;

struct BiTNode \*lchild,\*rchild;

}BiTNode,\*BiTree;

BiTree CreateBiTree(BiTree &T)

{

char ch;

scanf("%c",&ch);

if(ch=='#')

T=NULL;

else

{

if(!(T=(BiTNode\*)malloc(sizeof(BiTNode))))

return ERROR;

T->data=ch;

CreateBiTree(T->lchild);

CreateBiTree(T->rchild);

}

return T;

}

Status Visit(ElemType e)

{

if(e!='#')

{

printf("%c",e);

return OK;

}

else

return FALSE;

}

Status PreOrderTraverse(BiTree T)

{

if(T)

{

if(Visit(T->data))

if(PreOrderTraverse(T->lchild))

if(PreOrderTraverse(T->rchild))

return OK;

return ERROR;

}

else

return OK;

}

Status InOrderTraverse(BiTree T)

{

if(T)

{

if(InOrderTraverse(T->lchild))

if(Visit(T->data))

if(InOrderTraverse(T->rchild))

return OK;

return ERROR;

}

else

return OK;

}

Status PostOrderTraverse(BiTree T)

{

if(T)

{

if(PostOrderTraverse(T->lchild))

if(PostOrderTraverse(T->rchild))

if(Visit(T->data))

return OK;

return ERROR;

}

else

return OK;

}

int main()

{

BiTree T;

CreateBiTree(T);

PreOrderTraverse(T);

printf("\n");

InOrderTraverse(T);

printf("\n");

PostOrderTraverse(T);

printf("\n");

return 0;

}