華中科技大學

课 程 实 验 报 告

课程名称： 数据结构实验

专业班级 CS2202

学　　号 U202215378

姓　　名 冯瑞琦

指导教师 李丹

报告日期 2022 年 6 月 10 日

计算机科学与技术学院

目　　录

1　基于二叉树结构的二叉链表实现 1

1.1　问题描述 1

1.2　系统设计 2

1.3　系统实现 3

1.4　系统测试 12

1.5　实验小结 19

2　附录 A 基于二叉树结构的二叉链表实现的源程序 21

1 基于二叉树结构的二叉链表实现

1.1 问题描述

采用二叉树作为物理结构，构造一个具有菜单的功能演示系统。在主程序中显示完成函数调用所用的数字菜单，选择菜单数字调用相应的函数。操作时系统给出适当的输入提示，并显示调用结果。演示系统可进行多二叉树管理。设计二叉树文件保存和加载操作，可将生成的二叉树保存到文件中，也可以从文件中读取二叉树进行操作。

1.1.1 基本操作

1) CreateBiTree(BiTree T,TElemType definition[])

   操作结果: 根据带空枝的二叉树先根遍历序列 definition 构造一棵二叉树，将根节点指针赋值给 T 并返回 OK，如果有相同的关键字，返回 ERROR。

2) ClearBiTree(BiTree T))

   初始条件: 二叉树 T 已存在。

   操作结果: 将二叉树设置成空，并删除所有结点，释放结点空间。

3) BiTreeDepth(BiTree T)

   初始条件: 二叉树 T 已存在。

   操作结果: 返回二叉树 T 的深度。

4) LocateNode(BiTree T,KeyType e)

   初始条件: 二叉树 T 已存在。

   操作结果: 查找结点 e。若结点存在，返回该结点，否则返回 NULL。

5) Assign(BiTree T,KeyType e,TElemType value)

   初始条件: 二叉树 T 已存在。

   操作结果: 将 value 的值赋给结点 e，若未找到结点 e 则返回 ERROR。

6) GetSibling(BiTree T,KeyType e)

   初始条件: 二叉树 T 已存在。

   操作结果: 返回结点 e 的兄弟节点。

7) InsertNode(BiTree T,KeyType e,int LR,TElemType c)

初始条件: 二叉树 T 已存在。

操作结果: 根据 LR 的值，在结点 e 的左子树或右子树插入节点 c。

8) DeleteNode(BiTree T,KeyType e)

初始条件: 二叉树 T 已存在。

操作结果: 删除结点 e。

9) PreOrderTraverse(BiTree T,void (*visit)(BiTree))

初始条件: 二叉树 T 已存在。

操作结果: 先序遍历二叉树 T。

10) InOrderTraverse(BiTree T,void (*visit)(BiTree))

初始条件: 二叉树 T 已存在。

操作结果: 中序遍历二叉树 T。

11) PostOrderTraverse(BiTree T,void (*visit)(BiTree))

初始条件: 二叉树 T 已存在。

操作结果: 后序遍历二叉树 T。

12) LevelOrderTraverse(BiTree T,void (*visit)(BiTree))

初始条件: 二叉树 T 已存在。

操作结果: 按层遍历二叉树 T。

1.2 系统设计

1.2.1 数据存储结构与形式

二叉树的物理数据结构如下:

```
typedef struct {
KeyType  key;
char others[20];
} TElemType; //二叉树结点类型定义

typedef struct BiTNode { //二叉链表结点的定义
```

```
TElemType  data;
struct BiTNode *lchild,*rchild;
} BiTNode, *BiTree;
```

若要实现多二叉树管理，则定义一个二叉链表的结构数组，对二叉链表编号进行操作。

1.2.2 系统总体框架

通过 while 循环与菜单界面，用户通过选择菜单中的选项实现交互，使用 op 变量获取用户选择选项值 (op 初始化值为 1，以便第一次能进入循环)。

进入循环后系统首先显示功能菜单，然后用户输入选择 0-21, 其中 1-13 分别代表二叉链表的一个基本运算，15 与 16 选项分别是二叉链表的文件保存，17 是多线性表管理中选择操作的线性表的序号，14、18-21 是附加功能。在主函数中通过 switch 语句对应到相应的函数功能，执行完该功能后 break 跳出 switch 语句，继续执行 while 循环，直至用户输入 0 退出当前菜单系统。

系统总体框架如下图：

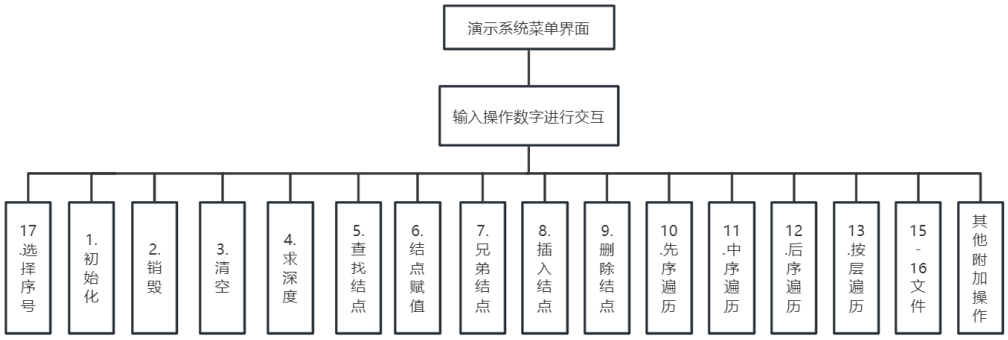

图 1-1 系统总体框架

1.3 系统实现

1.3.1 编程环境及运行环境描述

编程环境：采用 Dev-C++5.11 软件编写。

运行环境：微软 Windows 10 系统。

1.3.2 头文件及预定义常量说明

```
//头文件
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//预定义常量
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define treesnum 100
#define _CRT_SECURE_NO_WARNINGS 1
#define LIST_INIT_SIZE 100
#define LISTINCREMENT  10
#define root T
#define name others
#define filename Filename
#define L T
#define NODENUMBER 100//结点数最大值

//数据元素类型定义
typedef int ElemType;
typedef int status;
typedef int KeyType;
```

1.3.3 算法设计与实现

1) 初始化二叉树 status CreateBiTree(BTN *T)

   算法实现: 使用 malloc 函数分配连续内存空间，输入关键字和内容，连接形成二叉链表。

2) 销毁二叉树 status DestroyBiTree(BTN *T)

   算法实现: 使用 free 函数释放掉以 T 为首地址的连续内存空间，并将 T 置为 NULL。

3) 清空二叉树 status ClearBiTree(BTN *T)

   算法实现: 用递归释放 T 的左右孩子，直至 T 被清空。

4) 二叉树深度 status BiTreeDepth(BiTree T)

   算法实现: 使用递归, 得到左右子树的深度, 并比较大小, 之后返回最大的深度。

5) 查找结点 BiTNode* LocateNode(BiTNode *T,int e)

   算法实现: 当 T 不为空时，依次比较当前结点和 e，若相等则找到结点，返回该结点，若直到 T 为空都没有找到，则返回 NULL。

6) 结点赋值 status Assign(BiTree T,KeyType e,TElemType value)

   算法实现: 调用上一个函数 BiTNode* LocateNode(BiTNode *T,int e)，若未找到 e，返回 ERROR，若找到了则将 value 的值赋给 e。

7) 兄弟结点 BTN* GetSibling(BTN *T,int e)

   算法实现: 先寻找关键字为 e 的结点的双亲，若没有找到，则该结点也不会有兄弟结点，返回 NULL，若找到了，e 为双亲的左孩子，则返回双亲的右孩子，否则返回双亲的左孩子。

8) 插入结点 status InsertNode(BTN *T,int e,int LR,BTN *c)

   算法实现: 先用 LocateNode 判断关键字为 e 的结点是否存在，若不存在则返回 ERROR，若 LR 为 0 且 e 没有左孩子，则 c 为 e 的左孩子，否则将 e 原本的左孩子接到 e 的右孩子上，将 c 赋为 e 的左孩子，若 LR 等于 1 同理，让 c 成为 e 的右孩子；若 LR 既不为 1 也不为 0，提醒 LR 的值必须为 1 或 0，返回 ERROR。

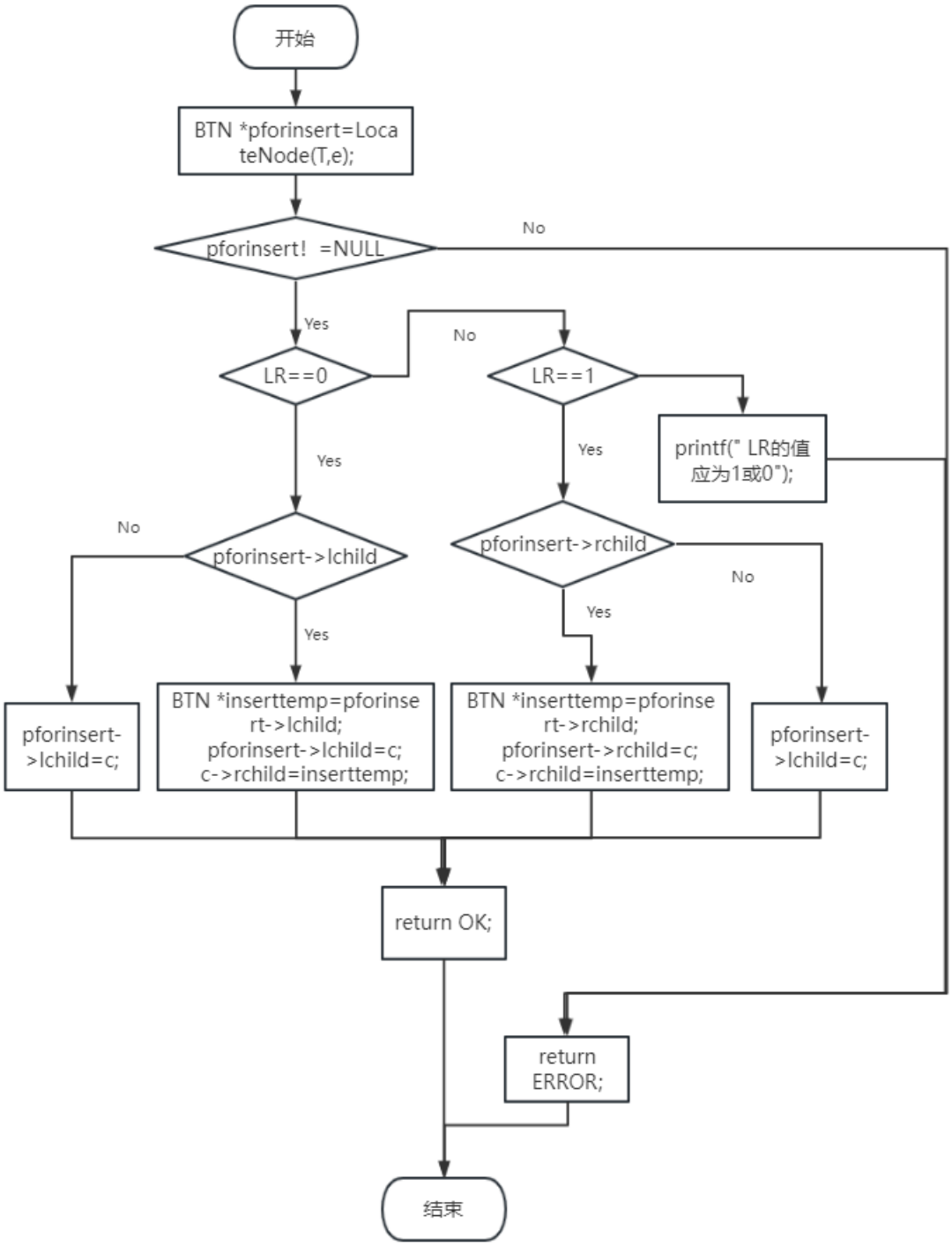

图 1-2　插入结点

9) 删除结点 status DeleteNode(BiTree T,KeyType e)

算法实现: 先用 LocateNode 判断关键字为 e 的结点是否存在，若不存咋则返回 ERROR。然后分情况讨论。分为 e 是叶子（度为 0），结点 e 不是根节点

且度为 1，结点 e 不是根节点且度为 2，e 是根节点。

10) 先序遍历 status PreOrderTraverse(BTN *T)

算法实现: 构建指针栈，空间换时间，若栈满，返回 OVERFLOW，当 top 或 T 中有一个不为 0 或空，当前结点入栈。

11) 中序遍历 status InOrderTraverse(BTN *T)

算法实现: 使用递归，先对左孩子调用函数，然后输出当前结点，再对右孩子调用函数。

12) 后序遍历 status PostOrderTraverse(BTN *T)

算法实现: 使用递归，先对左孩子调用函数，然后对右孩子调用函数，最后输出当前结点。

13) 按层遍历 status LevelOrderTraverse(BTN *T)

算法实现: 先用 BiTreeDepth 获得 T 的层数，然后运用 for 循环一层一层遍历。

14) 树的图形结构 status TreeDisplay(BiTree T,int depth,status (* visit)(char e))

算法实现: 代码如下

```
status TreeDisplay(BiTree T,int depth,status (* visit)(char e)) {
if(!T) {
printf("\n");
return OK;
}
int i=0;
for(; i<depth; i++)
printf("                ");
Dis_visit(*(T -> data.others));
printf("\n");
if(T->lchild||T->rchild) {
TreeDisplay(T->lchild,depth+1,Dis_visit);
TreeDisplay(T->rchild,depth+1,Dis_visit);
}
```

```
return OK;
}
```

15) 读取文件 status ReadBiTree(BTN *T)

算法实现: 代码如下

```
void read(BTN *&T,FILE *fp) {
int definition;
fscanf(fp," %d ",&definition);
if(!definition) T=NULL;
else {
T=(BTN*)malloc(sizeof(BTN));
T->data.key=definition;

fscanf(fp," %s ",&T->data.others);
read(T->lchild,fp);
read(T->rchild,fp);
}
}

status ReadBiTree(BTN *&T) {
FILE *fp;
char filename[100];
printf("                输入要读取的文件名: ");
scanf("%s",&filename);
getchar();
if ((fp=fopen(filename,"rb"))==NULL) {
printf("                读取发生错误\n");
return ERROR;
}
read(T,fp);

fclose(fp);
```

```
return OK;
}
```

16) 保存文件 status SaveBiTree(BTN *T)

算法实现: 代码如下

```
status SaveBiTree(BTN *T) { //非递归先序储存
FILE *fp;
BTN *pforsave=T;
BTN *stack[NODENUMBER];
int top=0;
char filename[100];
printf("输入要存入的文件名: ");
scanf("%s",&filename);

if((fp=fopen(filename,"wb"))==NULL) {
printf("存储发生错误\n");
return ERROR;
}

do {
while(pforsave) {
if(top==NODENUMBER-1) return OVERFLOW;//栈满
stack[top++]=pforsave;//当前结点入栈
fprintf(fp," %d ",pforsave->data.key);
            //采用 “ %d "是lr=3为了在读取时前后两个数据不会混淆
fprintf(fp," %s ",pforsave->data.name);
pforsave=pforsave->lchild;
}
fprintf(fp," %d ",0);
if(top) {
top--;
```

```
pforsave=stack[top]->rchild;
}
} while(top||pforsave);
fprintf(fp," %d ",0);
fclose(fp);
return OK;
}
```

17) 选择操作的二叉树 LocateList

算法实现: 输入数字 i，在第 i 个二叉树上操作。

18) 最大路径和 status MaxPathSum(BiTree T)

算法实现: 代码如下

```
int MPS(BiTree T)
{

if(!T) return 0;
int l=MPS(T->lchild);
int r=MPS(T->rchild);
return max(l,r)+T->data.key;
}
status MaxPathSum(BiTree T)
//最大路径和
{
int ans=MPS(T);
return ans;
}
```

19) 最近公共祖先 BiTNode* LowestCommonAncestor(BiTree T,KeyType e1,KeyType e2)

算法实现: 代码如下

```
BiTNode* LowestCommonAncestor(BiTree T,KeyType e1,KeyType e2)
//最近公共祖先
{
if(LocateNode(T->lchild,e1))
    {
if(LocateNode(T->lchild,e2))
            return LowestCommonAncestor(T->lchild,e1,e2);
return T;
}
else
{
if(LocateNode(T->rchild,e2))
            return LowestCommonAncestor(T->rchild,e1,e2);
return T;
}
}
```

20) 翻转二叉树 status InvertTree(BiTree T)

算法实现: 代码如下

```
status InvertTree(BiTree &T)
//翻转二叉树
{
if(!T)  return OK;
BiTNode*t;
t=T->lchild;
T->lchild=T->rchild;
T->rchild=t;
InvertTree(T->lchild);
InvertTree(T->rchild);
return 1;
}
```

21) 判空 status BiTreeEmpty(BTN *T)

算法实现: 若二叉树为空，返回 TRUE，不为空返回 FALSE。

1.4 系统测试

当程序开始运行时，会进入菜单演示界面。

```
                              二 叉 树 的 基 本 操 作
*********************************************************************************
*            1. CreateBiTree(创建二叉树)       8.InsertNode(插入结点)           *
*            2. DestroyBiTree (销毁二叉树)     9.DeleteNode(删除结点)           *
*            3. ClearBiTree (清空二叉树)       10. PreOrderTraverse(先序遍历)   *
*            4. BiTreeDepth(二叉树深度)        11. InOrderTraverse(中序遍历)    *
*            5. LocateNode(查找结点)           12. PostOrderTraverse(后序遍历)  *
*            6. Assign(结点赋值)               13. LevelOrderTraverse(按层遍历) *
*            7. GetSibling (兄弟结点)          14. TreeDisplay(树的图形结构)    *
*            17.ChooseTree (选择在哪个树操作)  15. ReadBiTree    (读取文件)    *
*            0. Exit                           16. SaveBiTree    (保存文件)    *
*            18. MaxPathSum   (最大路径和)     19. LowestCommonAncestor (最近公共祖先) *
*            20. InvertTree   (翻转二叉树)     21. BiTreeEmpty(判空)            *
*********************************************************************************
请选择你的操作[0~21]:
```

图 1-3 菜单演示界面

输入数字 17，系统提示选择要操作的二叉树的序号（如果不选择默认在 0 号二叉树上操作）二叉链表的输入可以通过功能 1 初始化，也可通过功能 15 从文件中读入，读入时要先输入文件名。

```
请选择你的操作[0~21]:
17
请输入要在第几个树操作:
 *只支持在100个树上操作*
2
正在对第2个树进行操作
```

图 1-4 选择要操作的线性表的序号

接下来展示各功能的测试结果。

(a) 初始化

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

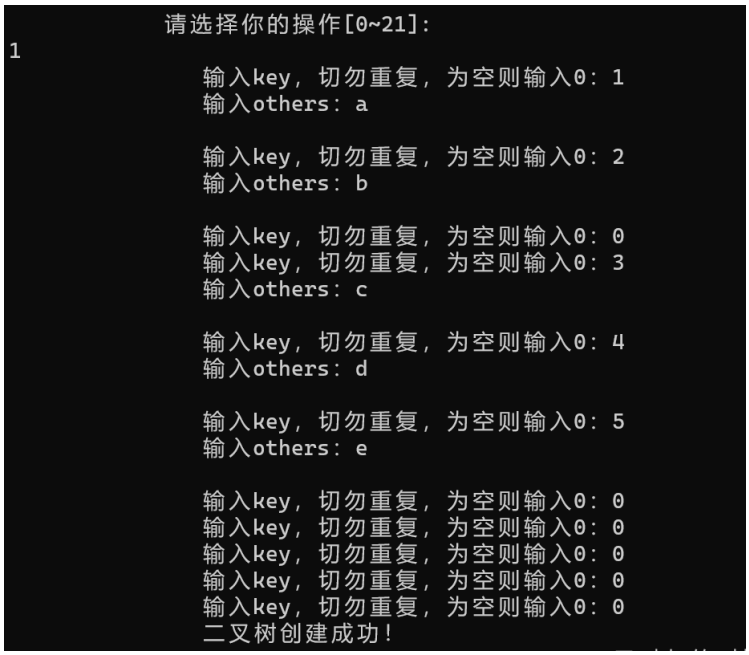

图 1-5　初始化

(b) 销毁二叉树

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

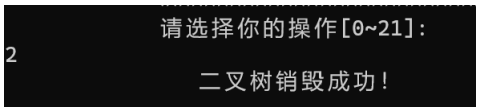

图 1-6　销毁

(c) 清空表

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

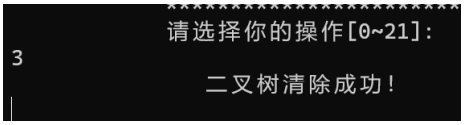

图 1-7　清空

(d) 求深度

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

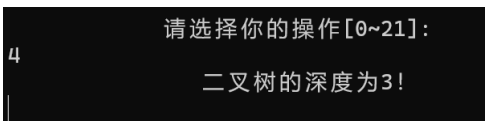

图 1-8　求深度

测试用例：0 0（即空二叉树），结果图如下。

```
请选择你的操作[0~21]:
4

二叉树的深度为0!
```

图 1-9　求深度

(e) 查找结点

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，查找关键字为 3 的结点，结果图如下。

```
请选择你的操作[0~21]:
5

请输入你要查找结点的关键字：3
已找到，结点信息如下：
key: 3,others: c
```

图 1-10　查找结点

(f) 结点赋值

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，将关键字为 5 的结点赋值为 11 x，赋值后前序遍历，结果图如下。

```
请选择你的操作[0~21]:
6

请输入你要赋值的结点的关键字：5
请输入：11 x
赋值完成，现在该结点信息如下：
key: 11,others: x

                                二 叉 树 的 基 本 操 作
*********************************************************************************************
*            1. CreateBiTree(创建二叉树)       8.InsertNode(插入结点)                       *
*            2. DestroyBiTree（销毁二叉树）    9.DeleteNode(删除结点)                       *
*            3. ClearBiTree（清空二叉树）      10. PreOrderTraverse(先序遍历)               *
*            4. BiTreeDepth(二叉树深度)        11. InOrderTraverse(中序遍历)                *
*            5. LocateNode(查找结点)           12. PostOrderTraverse(后序遍历)              *
*            6. Assign(结点赋值)               13. LevelOrderTraverse(按层遍历)             *
*            7. GetSibling（兄弟结点）         14. TreeDisplay(树的图形结构)                *
*            17.ChooseTree（选择在哪个树操作） 15. ReadBiTree    （读取文件）              *
*            0. Exit                           16. SaveBiTree    （保存文件）              *
*            18. MaxPathSum   （最大路径和）   19. LowestCommonAncestor（最近公共祖先）    *
*            20. InvertTree   （翻转二叉树）   21. BiTreeEmpty(判空)                        *
*********************************************************************************************
请选择你的操作[0~21]:
10

key: 1,others: a
key: 2,others: b
key: 3,others: c
key: 4,others: d
key: 11,others: x
前序遍历完成!
```

图 1-11　结点赋值

(g) 兄弟结点测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，获取关键字为 2 的结点的兄弟结点，结果图如下。

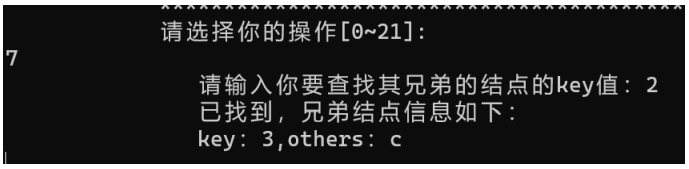

图 1-12　兄弟结点

(h) 插入结点

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，在关键字为 4 的结点的左孩子插入关键字为 8，others 为 f 的结点，之后先序遍历，结果图如下。

图 1-13　插入结点

(i) 删除结点

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，删除关键字为 3 的结点后先序遍历，结果图如下。

```
请选择你的操作[0~21]:
9

   请输入你要删除的结点的关键字:
3

   删除成功!

                              二 叉 树 的 基 本 操 作
**************************************************************************************
*            1. CreateBiTree(创建二叉树)       8.InsertNode(插入结点)                 *
*            2. DestroyBiTree (销毁二叉树)     9.DeleteNode(删除结点)                 *
*            3. ClearBiTree (清空二叉树)       10. PreOrderTraverse(先序遍历)         *
*            4. BiTreeDepth(二叉树深度)        11. InOrderTraverse(中序遍历)          *
*            5. LocateNode(查找结点)           12. PostOrderTraverse(后序遍历)        *
*            6. Assign(结点赋值)               13. LevelOrderTraverse(按层遍历)       *
*            7. GetSibling (兄弟结点)          14. TreeDisplay(树的图形结构)          *
*            17.ChooseTree (选择在哪个树操作)  15. ReadBiTree    (读取文件)          *
*            0. Exit                           16. SaveBiTree    (保存文件)          *
*            18. MaxPathSum   (最大路径和)     19. LowestCommonAncestor (最近公共祖先)  *
*            20. InvertTree   (翻转二叉树)     21. BiTreeEmpty(判空)                  *
**************************************************************************************
请选择你的操作[0~21]:
10

   key: 1,others: a
   key: 2,others: b
   key: 4,others: d
```

图 1-14　删除结点

(j) 先序遍历

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

```
    请选择你的操作[0~18]:4
线性表不存在!
```

图 1-15　先序遍历

(k) 中序遍历测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

```
            请选择你的操作[0~21]:
11
               key: 2,others: b
               key: 1,others: a
               key: 4,others: d
               key: 3,others: c
               key: 5,others: e
               中序遍历完成!
```

图 1-16　中序遍历

(l) 后序遍历

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果如下图。

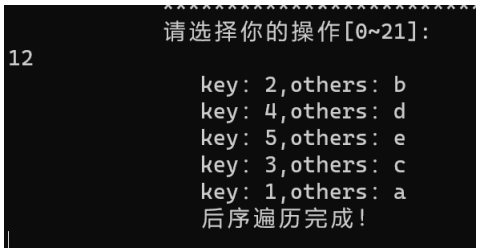

图 1-17　后序遍历

(m) 按层遍历

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

图 1-18　按层遍历

(n) 树的图形结构

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

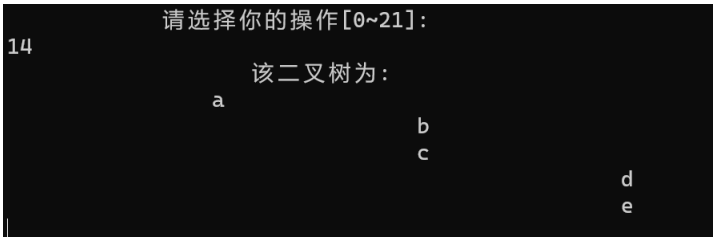

图 1-19　树的图形结构

(o) 读取文件

测试用例：文件名 tree，文件内容为 1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，读取后遍历，结果图如下。

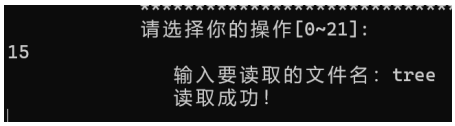

图 1-20　读取文件

(p) 写入文件

测试用例：测试用例为 1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，写入的文件名为

ok, 结果图如下。

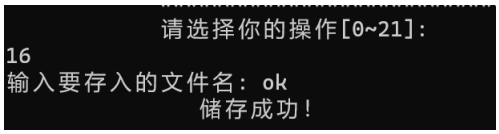

图 1-21　写入文件

(q) 选择在第几个二叉树上操作

测试用例：选择在第二个二叉树上操作，结果图如下。

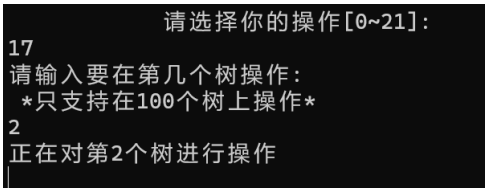

图 1-22　选择二叉树

(r) 最大路径和

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

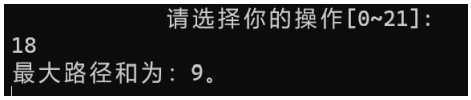

图 1-23　最大路径和

(s) 最近公共祖先

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

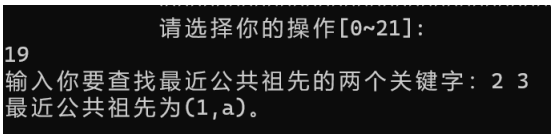

图 1-24　最近公共祖先

(t) 翻转二叉树测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

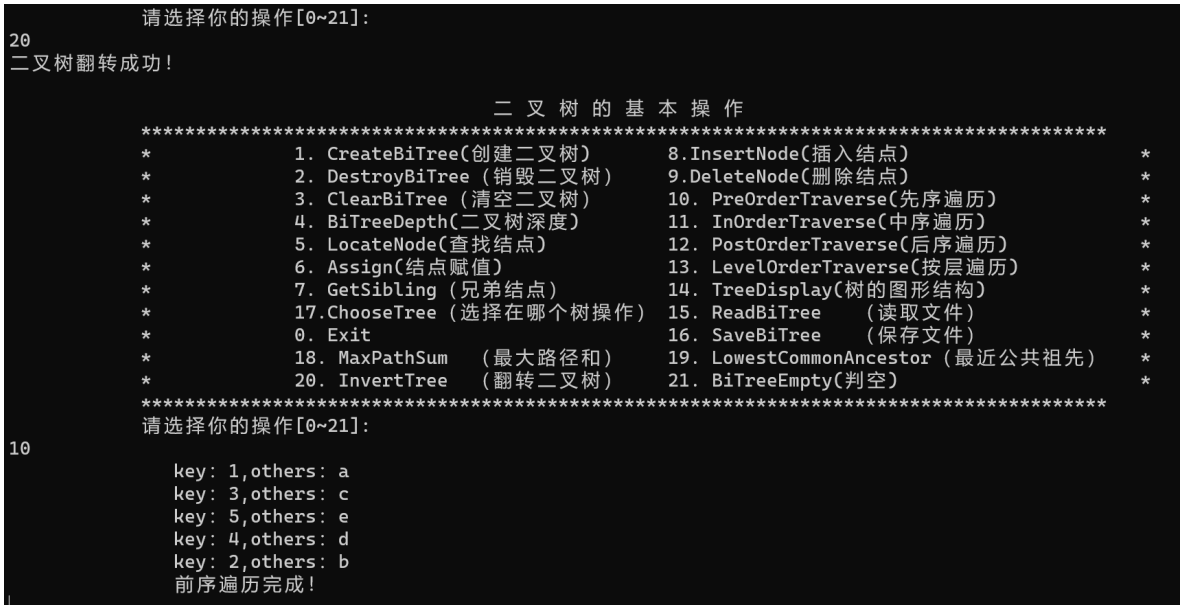

图 1-25　翻转二叉树

(u) 判空

测试用例：1 a 2 b 0 0 3 c 4 d 0 0 5 e 0 0，结果图如下。

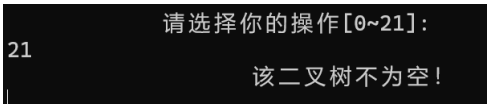

图 1-26　判空

测试用例：0 0（空二叉树），结果图如下。

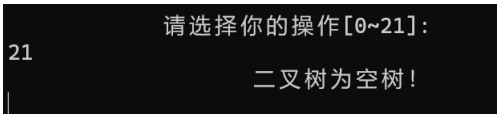

图 1-27　判空

1.5　实验小结

在本次实验中，我使用的语言为 C 语言，通过代码的设计和编写完成了二叉链表的创建和一系列操作。在实验过程中，我理解了二叉树的物理存储结构的编写和实现原理，学会了二叉树的初始化、遍历、查找等基本操作。

在调试中，我碰到的第一个错误是细节上拼写的错误。从老师给的模板上复制来的框架内容和预定义常量部分和实际的正确拼写有一点小小的偏差，

比如 INFEASIBLE 拼成了 INFEASABLE，一字之差，就会导致程序无法运行，这提醒我了代码的严谨性是至关重要的。

实验过程中我遇到的最大的困难是关于文件的处理、二叉树的结点赋值以及结点插入。上个学期学习 C 语言时文件部分我掌握得不太牢固，导致编写与文件有关的函数时比较生疏。但是通过在网上查找资料和不断试错，我掌握了关于文件打开、关闭、读入等基本操作，还尝试用栈进行文件读入。二叉树结点的赋值一开始我没有完全实现功能，只能对结点的内容进行赋值而无法改变结点的关键字，在反复修改函数及其调用后，实现了给关键字赋值的功能。插入结点的难点在于需要详细的分类讨论，分为 e 是叶子（度为 0），结点 e 不是根节点且度为 1，结点 e 不是根节点且度为 2，e 是根节点的情况，如果没有正确的分类，就会导致一部分结果的偏差，这对严谨的逻辑思维能力有着较高要求。

总体来说，通过这次实验，我对文件处理和栈的运用以及递归函数的编写都有了更深的了解，补上了以前的知识漏洞，也对多函数的模块化处理以及菜单显示有了直观的认识，还意识到了逻辑严谨的重要性，很有收获。

2 附录 A 基于二叉树结构的二叉链表实现的源程序

```
\noindent
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define treesnum 100
#define _CRT_SECURE_NO_WARNINGS 1
#define LIST_INIT_SIZE 100
#define LISTINCREMENT  10
#define root T
#define name others
#define filename Filename
#define L T
#define NODENUMBER 100//结点数最大值

typedef int ElemType; //数据元素类型定义
typedef int status;
typedef int KeyType;
typedef struct {
KeyType  key;
char others[20];
```

```
} TElemType; //二叉树结点类型定义

typedef struct BiTNode { //二叉链表结点的定义
TElemType  data;
struct BiTNode *lchild,*rchild;
} BiTNode, *BiTree;
typedef BiTNode BTN;



status Dis_visit(char e);
void visit(BTN *T);
status InitBiTree(BiTree *T);
status CreateBiTree(BTN *&T);
status DestroyBiTree(BTN *&T);
status ClearBiTree(BTN *&T);
status BiTreeEmpty(BTN *T);
int BiTreeDepth(BTN *T);
BTN* LocateNode(BTN *T,int e);//e是关键字
status Assign(BiTree &T,KeyType e,TElemType value);
BTN* GetParent(BTN *T,int e);//找兄弟先找双亲
BTN* GetSibling(BTN *T,int e);
status InsertNode(BTN *&T,int e,int LR,BTN *&c);
status DeleteNode(BiTree &T,KeyType e);
BiTree Getfather(BiTree T, KeyType e);
status PreOrderTraverse(BTN *T);//已简化, 非递归
status InOrderTraverse(BTN *T);
status PostOrderTraverse(BTN *T);
void OneLevel(BTN *T,int h);//遍历一层
status LevelOrderTraverse(BTN *T);
void read(BTN *&T,FILE *fp);
status ReadBiTree(BTN *&T);//读文件
```

```
status SaveBiTree(BTN *T);
status TreeDisplay(BiTree T,int depth,status (* visit)(char e));
status MaxPathSum(BiTree T);
BiTNode* LowestCommonAncestor(BiTree T,KeyType e1,KeyType e2);
status InvertTree(BiTree &T);
int MPS(BiTree T);

int max(int x,int y)
{
    return x>y?x:y;
}

BiTree T[treesnum];
int main()
{

ElemType op=1,j = 0,k=0, lr = 3;
KeyType key,key1,key2;
BiTNode* t;
BiTNode *pforassign=(BTN *)malloc(sizeof(BTN));
BTN *pforsib=(BTN *)malloc(sizeof(BTN));
BTN *pforloc=(BTN *)malloc(sizeof(BTN));
BTN *pforins=(BTN *)malloc(sizeof(BTN));
for(int i = 0; i < treesnum ; i++)
T[i] = NULL;
j = 0;
char Filename[100];
while(op){
printf("                                        二 叉 树 的 基 本 操 作 \n
printf("          ************************************************************
printf("          *       1. CreateBiTree(创建二叉树)       8.InsertNode(插入
```

```
printf("            *       2. DestroyBiTree (销毁二叉树)     9.DeleteNode(删
printf("            *       3. ClearBiTree (清空二叉树)       10. PreOrderTrav
printf("            *       4. BiTreeDepth(二叉树深度)        11. InOrderTrave
printf("            *       5. LocateNode(查找结点)           12. PostOrderTra
printf("            *       6. Assign(结点赋值)               13. LevelOrderTr
printf("            *       7. GetSibling (兄弟结点)          14. TreeDisplay(
printf("            *      17.ChooseTree (选择在哪个树操作)  15. ReadBiTree
printf("            *      0. Exit                           16. SaveBiTree
printf("         *   18. MaxPathSum   (最大路径和)     19. LowestCommonAncesto
printf("         *   20. InvertTree   (翻转二叉树)     21. BiTreeEmpty(判空)
printf("            ************************************************************
printf("            请选择你的操作[0~21]:\n");
scanf("%d", &op);
switch (op) {
case 0:
break;

case 1:
if(CreateBiTree(T[k])) printf("                二叉树创建成功! \n");
else printf("               二叉树创建失败! \n");
break;

case 2:
if (DestroyBiTree(T[k]) == OK) printf("                二叉树销毁成功! \n");
else printf("               二叉树销毁失败! \n");
getchar();
getchar();
break;

case 3:
if (DestroyBiTree(T[k]) == OK) printf("                二叉树清除成功! \n");
```

```
getchar();
getchar();
break;

case 4:
printf("                二叉树的深度为%d! \n", BiTreeDepth(T[k]));
getchar();
getchar();
break;

case 5:
int keyforlocate;
printf("                请输入你要查找结点的关键字: ");
scanf("%d",&keyforlocate);
pforloc = LocateNode(T[k],keyforlocate);
if(pforloc) {
printf("                已找到, 结点信息如下: \n");
visit(pforloc);
} else printf("                未找到! \n");
getchar();
getchar();
break;

case 6:
pforassign->lchild=NULL;
pforassign->rchild=NULL;//保证安全
printf("                请输入你要赋值的结点的关键字: ");
scanf("%d",&pforassign->data.key);
TElemType e;
printf("                请输入: ");
scanf("%d %s",&e.key,e.others);
```

```
if(Assign(T[k],pforassign->data.key,e)) {
printf("                赋值完成，现在该结点信息如下：\n");
visit(LocateNode(T[k],e.key));
} else printf("                未找到！\n");
getchar();
getchar();
break;

case 7:
int keyforsib;
printf("                请输入你要查找其兄弟的结点的key值：");
scanf("%d",&keyforsib);
pforsib = GetSibling(T[k],keyforsib);
if(pforsib) {
printf("                已找到，兄弟结点信息如下：\n");
visit(pforsib);
} else printf("                未找到！\n");
getchar();
getchar();
break;

case 8:
int keyforins;
BTN * pForInsert;
printf("                请输入你要插入的结点key值(0表示根节点)：");
scanf("%d",&keyforins);
if(keyforins == 0) {
pForInsert = (BiTNode*)malloc(sizeof(BiTNode));
printf("                请输入你要赋值的结点的key值：");
scanf("%d",&pForInsert -> data.key);
printf("                请输入others：");
```

```
scanf("%s",pForInsert -> data.others);//新结点生成
pForInsert -> rchild = T[k];
T[k] = pForInsert;
break;
}
lr=3;
while(lr != 0 && lr != 1) {
printf("                请输入0 (左孩子) 或1 (右孩子): ");
scanf("%d",&lr);
}
pforins=(BTN *)malloc(sizeof(BTN));
pforins -> lchild = NULL;
pforins -> rchild = NULL;//保证安全
printf("                请输入你要赋值的结点的key值: ");
scanf("%d",&pforins->data.key);
printf("                请输入others: ");
scanf("%s",&pforins->data.others);
if(InsertNode(T[k],keyforins,lr,pforins)==OK) printf("                插入完成!
else printf("                插入失败! \n");
getchar();
getchar();
break;

case 9:
int keyfordel;
printf("                请输入你要删除的结点的关键字: ");
scanf("%d",&keyfordel);
if(DeleteNode(T[k],keyfordel)==OK) printf("                删除成功! \n");
else printf("                删除失败! ");
getchar();
getchar();
```

```
break;

case 10:
if(PreOrderTraverse(T[k])==OK) printf("                前序遍历完成! \n");
else printf("               前序遍历失败! \n");
getchar();
getchar();
break;

case 11:
if(InOrderTraverse(T[k])==OK) printf("                中序遍历完成! \n");
else printf("               中序遍历失败! \n");
getchar();
getchar();
break;

case 12:
if(PostOrderTraverse(T[k])==OK) printf("                后序遍历完成! \n");
else printf("               后序遍历失败! \n");
getchar();
getchar();
break;

case 13:
if(LevelOrderTraverse(T[k])==OK)
printf("               按层遍历完成! \n");
else printf("               按层遍历失败! \n");
getchar();
getchar();
break;
```

```
case 14:
if (!T[k]) {
printf("                    二叉树不存在! \n");
getchar();
getchar();
break;
}
if(BiTreeEmpty(T[k]) == OK)
printf("                    二叉树为空树! \n");
else {
printf("                    该二叉树为:\n");
TreeDisplay(T[k],1, Dis_visit);
}
getchar();
getchar();
break;

case 15:
if(ReadBiTree(T[k])==OK) printf("                读取成功! \n");
else printf("                读取失败! \n");
getchar();
break;

case 16:
if(SaveBiTree(T[k])==OK) printf("                储存成功! \n");
else printf("                储存失败! \n");
getchar();
break;

case 17://选择在哪个树进行操作
            printf("请输入要在第几个树操作:\n ");
```

```
            printf("*只支持在%d个树上操作*\n",treesnum);
  scanf("%d",&k);
  printf("正在对第%d个树进行操作\n",k);
if((k<0)||(k>99))
{
printf("请选择正确范围! \n");
k=0;
}
getchar(); getchar();
        break;


   case 18:
   if(!T[k])  printf("二叉树不存在! \n");
              else
              {
                  printf("最大路径和为: %d。\n",MaxPathSum(T[k]));
              }
   getchar();getchar();
   break;
   case 19:

   if(!T[k]) printf("二叉树不存在! \n");
              else
              {
                  printf("输入你要查找最近公共祖先的两个关键字: ");
                  scanf("%d%d",&key1,&key2);
                  t=LowestCommonAncestor(T[k],key1,key2);
                  if(t)printf("最近公共祖先为(%d,%s)。\n",t->data.key,t->dat
                  else printf("查找失败! \n");
              }
```

```
        getchar();getchar();
        break;
        case 20:
        if(!T[k]) printf("二叉树不存在! \n");
                  else
                  {
                      if(InvertTree(T[k]))  printf("二叉树翻转成功! \n");
                      else printf("二叉树翻转失败! \n");
                  }
        getchar();getchar();
        break;

        case 21:
if(BiTreeEmpty(T[k]) == OK)
printf("                    二叉树为空树! \n");
else {
printf("                    该二叉树不为空! \n");
}
getchar();
getchar();
break;
default://确保输入的数字有意义
printf("                    请输入[0~21]实现菜单功能");
break;

}//end of switch
}//end of while
}//end of main

status Dis_visit(char e) {
printf("%c",e);//依次调用该函数, 用来打印
```

```
}


void visit(BTN *T) {
printf("                key: %d,others: %s\n",T->data.key,T->data.others);
}

/**
 * 函数名称: InitBiTree
 * 初始条件: 二叉树T不存在
 * 操作结果: 构造空树二叉树T
 *
 */
status InitBiTree(BiTree *T) {
*T = (BiTree)malloc(sizeof(BiTNode));
(*T) -> lchild = NULL;
(*T) -> rchild = NULL;
strcpy((*T) -> data.others,"#");
(*T) -> data.key = 0;//初始化二叉树,将左右指针指向空

return OK;
}


/**
 * 函数名称: DestroyBiTree
 * 初始条件: 二叉树T已存在
 * 操作结果: 销毁二叉树T
 *
 */
status DestroyBiTree(BTN *&T) {
```

```
free(T);
T = NULL;
return OK;
}

status ClearBiTree(BTN *&T) {
if(T) {
DestroyBiTree(T->lchild);
DestroyBiTree(T->rchild);
free(T);
}
InitBiTree(&T);
return OK;
}


/**
 * 函数名称: CreateBiTree
 * 初始条件: 二叉树T已存在
 * 操作结果: 创建二叉树
 *取材于课本,依次输入节点值, 并用key标记节点
 */
status CreateBiTree(BTN *&T) {
int definition;
printf("                输入key, 切勿重复, 为空则输入0: ");
scanf("%d",&definition);
getchar();
if(!definition) {
T=NULL;
} else {
```

```
T=(BTN*)malloc(sizeof(BTN));
T->data.key=definition;
printf("                输入others: ");
scanf("%s",&T->data.others);
getchar();
getchar();
CreateBiTree(T->lchild);
CreateBiTree(T->rchild);
}
return OK;
}
```

```
/**
 * 函数名称: BiTreeEmpty
 * 初始条件: 无
 * 操作结果: 二叉树判空
 *
 */
status BiTreeEmpty(BTN *T) {
if(T) return FALSE;
else return TRUE;
}
```

```
/**
```

```
 * 函数名称: BiTreeDepth
 * 初始条件: 二叉树T已存在
 * 操作结果: 返回T的深度
 *
 */
status BiTreeDepth(BiTree T) {
int depth = 0;
if(T) {
int lchilddepth = BiTreeDepth(T->lchild);
int rchilddepth = BiTreeDepth(T->rchild);
depth = (lchilddepth>=rchilddepth?(lchilddepth+1):(rchilddepth+1));
}//使用递归,得到左右子树的深度,并比较大小,之后返回最大的深度
return depth;
}


BiTNode* LocateNode(BiTNode *T,int e) {
if (!T) return NULL;

BTN *pForLocate=NULL;
if(T->data.key==e) {
pForLocate=T;
return pForLocate;
} else {
pForLocate=LocateNode(T->lchild,e);
if(pForLocate)  return pForLocate;
else {
pForLocate=LocateNode(T->rchild,e);
if(pForLocate)  return pForLocate;
}
}
```

```
return NULL;
}


status Assign(BiTree &T,KeyType e,TElemType value)
//实现结点赋值。此题允许通过增加其它函数辅助实现本关任务
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    if(!T) return ERROR;
    if(LocateNode(T,value.key)&&e!=value.key)//未找到关键字
            return ERROR;
    if(T->data.key==e) {
        T->data.key=value.key;
        strcpy(T->data.others,value.others);
        return OK;
    }
    while(T){
        if(Assign(T->lchild,e,value))
            return Assign(T->lchild,e,value);
        else return Assign(T->rchild,e,value);
    }

    /********** End **********/
}


BTN* GetParent(BTN *T,int e) {
if (!T) return NULL;
BTN *pForParent=NULL;
```

```
if(!T->lchild && !T->rchild) return NULL;
else if(!T->lchild) {
if(T->rchild->data.key==e) {
pForParent=T;
return pForParent;
}
} else if(!T->rchild) {
if(T->lchild->data.key==e) {
pForParent=T;
return pForParent;
}
} else if(T->lchild->data.key==e||T->rchild->data.key==e) {
pForParent=T;
return pForParent;
}

pForParent=GetParent(T->lchild,e);
if(pForParent)  return pForParent;
pForParent=GetParent(T->rchild,e);
if(pForParent)  return pForParent;

return NULL;
}

BTN* GetSibling(BTN *T,int e) {
BTN *pforsib = GetParent(T,e);//找双亲
if (!pforsib) return NULL;
else if(!pforsib->lchild||!pforsib->rchild) return NULL;
else if(pforsib->lchild->data.key==e) return pforsib->rchild;
else if(pforsib->rchild->data.key==e) return pforsib->lchild;
else return NULL;//看起来无用，保证安全。
```

```
}

status InsertNode(BTN *&T,int e,int LR,BTN *&c) {
BTN *pforinsert=LocateNode(T,e);
if(!pforinsert) return ERROR;//关键字e的结点e不存在
else if(LR==0) {
if(!pforinsert->lchild) {
pforinsert->lchild=c;
} else {
BTN *inserttemp=pforinsert->lchild;
pforinsert->lchild=c;
c->rchild=inserttemp;
}
return OK;
} else if(LR==1) {
if(!pforinsert->rchild) {
pforinsert->rchild=c;
} else {
BTN *inserttemp=pforinsert->rchild;
pforinsert->rchild=c;
c->rchild=inserttemp;
}
return OK;
} else {
printf("                LR的值应为1或0");
return ERROR;
}

}

BiTree Getfather(BiTree T, KeyType e){
```

```
    if(T->lchild!=NULL){
        if(T->lchild->data.key == e)
            return T;
        else{
            BiTree left = Getfather(T->lchild,e);
            if(left) return left;
        }
    }
    if(T->rchild!=NULL){
        if(T->rchild->data.key == e)
            return T;
        else{
            BiTree right = Getfather(T->rchild,e);
            if(right) return right;
        }
    }

}

status DeleteNode(BiTree &T,KeyType e)
//删除结点。此题允许通过增加其它函数辅助实现本关任务
{
    // 请在这里补充代码，完成本关任务
    /********** Begin *********/
    BiTree loce = LocateNode(T, e);
    if (!loce) return ERROR;

    BiTree pl = NULL, pr = NULL, parent;
    parent = Getfather(T, e);

    if (!loce->lchild && !loce->rchild && parent) // e是叶子（度为0）
```

```
{
    if (parent->lchild == loce)
        parent->lchild = NULL;
    else parent->rchild = NULL;
    free(loce);
    loce = NULL;
    return OK;
}

if (loce != T) {//e不是根节点
    if (!parent) return ERROR;
    if (parent->lchild == loce)
        pl = loce;
    else  pr = loce;

    if (loce->lchild && !loce->rchild) //结点e不是根节点且度为1
    {
        if (pl)
            parent->lchild = loce->lchild; // parent为空指针的情况在上面已
        else if (pr)
            parent->rchild = loce->lchild;
        free(loce);
        loce = NULL;
    }
    else if (loce->rchild && !loce->lchild) //结点e不是根节点且度为1
    {
        if (pl)
            parent->lchild = loce->rchild;
        else if (pr)
            parent->rchild = loce->rchild;
        free(loce);
```

```
            loce = NULL;
        }
        else if (loce->rchild && loce->lchild) //结点e不是根节点且度为2
        {
            if (parent->lchild == loce)
                parent->lchild = loce->lchild;
            if (parent->rchild == loce)
                parent->rchild = loce->lchild;
            BiTree temp1 = loce->lchild, temp2 = temp1->rchild;
            while (temp2) {
                temp1 = temp2;
                temp2 = temp2->rchild;
            }
            temp1->rchild = loce->rchild;
            free(loce);
            loce = NULL;
        }
    }


    else { //e是根节点
        if (loce->lchild == NULL && loce->rchild) // deg == 1
            T = loce->rchild;
        else if (loce->rchild == NULL && loce->lchild)
            T = loce->lchild;
        else if (!loce->rchild && !loce->lchild) {
            free(loce);
            return OK;
        }
        else // deg == 2
        {
```

```
            BiTree temp1 = loce->lchild, temp2 = temp1->rchild;
            while (temp2) //找左子树的最右结点
            {
                temp1 = temp2;
                temp2 = temp2->rchild;
            }
            temp1->rchild = loce->rchild;
            T = loce->lchild;
            free(loce);
            loce = NULL;
        }
        return OK;
    }

    return OK;
    /********** End **********/
}
status PreOrderTraverse(BTN *T) { //非递归
BTN *stack[NODENUMBER];//构建指针栈，空间换时间
int top=0;//栈顶
BTN *pforPreT=T;
do {
while(pforPreT) {
if(top==NODENUMBER-1) return OVERFLOW;//栈满
stack[top++]=pforPreT;//当前结点入栈
visit(pforPreT);
pforPreT=pforPreT->lchild;
}
if(top) {
top--;
pforPreT=stack[top]->rchild;
```

```
}
} while(top||pforPreT);
return OK;
}

status InOrderTraverse(BTN *T) {
if(T) {
InOrderTraverse(T->lchild);
visit(T);
InOrderTraverse(T->rchild);
}
return OK;
}

status PostOrderTraverse(BTN *T) {
if(T) {
PostOrderTraverse(T->lchild);
PostOrderTraverse(T->rchild);
visit(T);
}
return OK;
}

void OneLevel(BTN *T,int h) {
if(T) {
if(h==1) visit(T);
else {
OneLevel(T->lchild,h-1);
OneLevel(T->rchild,h-1);
}
}
```

```
}

status LevelOrderTraverse(BTN *T) {
if(!T) return ERROR;

int d=BiTreeDepth(T);
int i;
for(i=1; i<=d; i++) {
OneLevel(T,i);
}
return OK;
}

status TreeDisplay(BiTree T,int depth,status (* visit)(char e)) {
if(!T) {
printf("\n");
return OK;
}
int i=0;
for(; i<depth; i++)
printf("                ");
Dis_visit(*(T -> data.others));
printf("\n");
if(T->lchild||T->rchild) {
TreeDisplay(T->lchild,depth+1,Dis_visit);
TreeDisplay(T->rchild,depth+1,Dis_visit);
}
return OK;
}

void read(BTN *&T,FILE *fp) {
```

```
int definition;
fscanf(fp," %d ",&definition);
if(!definition) T=NULL;
else {
T=(BTN*)malloc(sizeof(BTN));
T->data.key=definition;

fscanf(fp," %s ",&T->data.others);
read(T->lchild,fp);
read(T->rchild,fp);
}
}

status ReadBiTree(BTN *&T) {
FILE *fp;
char filename[100];
printf("                输入要读取的文件名: ");
scanf("%s",&filename);
getchar();
if ((fp=fopen(filename,"rb"))==NULL) {
printf("                读取发生错误\n");
return ERROR;
}
read(T,fp);

fclose(fp);

return OK;
}

status SaveBiTree(BTN *T) { //非递归先序储存
```

```
FILE *fp;
BTN *pforsave=T;
BTN *stack[NODENUMBER];
int top=0;
char filename[100];
printf("输入要存入的文件名: ");
scanf("%s",&filename);

if((fp=fopen(filename,"wb"))==NULL) {
printf("存储发生错误\n");
return ERROR;
}

do {
while(pforsave) {
if(top==NODENUMBER-1) return OVERFLOW;//栈满
stack[top++]=pforsave;//当前结点入栈
fprintf(fp," %d ",pforsave->data.key);//采用 “ %d "是lr=3为了在读取时前后两个数
fprintf(fp," %s ",pforsave->data.name);
pforsave=pforsave->lchild;
}
fprintf(fp," %d ",0);
if(top) {
top--;
pforsave=stack[top]->rchild;
}
} while(top||pforsave);
fprintf(fp," %d ",0);
fclose(fp);
return OK;
}
```

```
int MPS(BiTree T)
{

if(!T) return 0;
int l=MPS(T->lchild);
int r=MPS(T->rchild);
return max(l,r)+T->data.key;
}
status MaxPathSum(BiTree T)
//最大路径和
{
int ans=MPS(T);
return ans;
}

BiTNode* LowestCommonAncestor(BiTree T,KeyType e1,KeyType e2)
//最近公共祖先
{
if(LocateNode(T->lchild,e1))
    {
if(LocateNode(T->lchild,e2))  return LowestCommonAncestor(T->lchild,e1,e2);
return T;
}
else
{
if(LocateNode(T->rchild,e2))  return LowestCommonAncestor(T->rchild,e1,e2);
return T;
}
}
status InvertTree(BiTree &T)
```

```
//翻转二叉树
{
if(!T)  return OK;
BiTNode*t;
t=T->lchild;
T->lchild=T->rchild;
T->rchild=t;
InvertTree(T->lchild);
InvertTree(T->rchild);
return 1;
}
```