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 CS2410

学　　号 U202414825

姓　　名 徐越扬

指导教师 郑渤龙

报告日期 2022 年 6 月 12 日

计算机科学与技术学院

目　　录

1 基于顺序存储结构的线性表实现

1.1 问题描述

线性表（Linear List）是最基本、最常用的一种数据结构，用来存储具有线性关系的一组数据元素。它是具有相同数据类型的 n 个数据元素的有限序列，每个元素有唯一的前驱和后继（第一个元素除外没有前驱，最后一个元素除外没有后继）。线性表有两种主要的存储方式：顺序存储结构（数组），链式存储结构（链表）。前者是把线性表的元素存储在一块连续的内存空间中，而后者包含每个元素的存储数据和指向下一个元素的指针。在优缺点方面：前者可以快速随机的访问元素，但是插入删除效率较低。后者插入删除效率高，但是必须从头访问。不同的情形下两者的使用不同。

本实验要求构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备以及执行结果的显示，并给出正确的操作提示。该程序实现了线性表的初始化、销毁线性表、清空线性表、线性表判空、求线性表表长、获得元素等基本功能，以及最大连续子数组和、和为 K 的子数组、顺序表排序等附加功能。可以以文件的形式进行存储和加载。同时实现多线性表管理，完成多线性表的添加、删除、定位，查找和展示等操作。

1.2 系统设计

1.2.1 头文件和预定义

1、头文件

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
```

2、预定义常量

```
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
```

```
#define INFEASIBLE -1
#define OVERFLOW -2
#define LIST_INIT_SIZE 100
#define LISTINCREMENT 10
```

3、类型表达式

```
typedef int status;
typedef int ElemType; //数据元素类型定义
typedef struct{  //顺序表的定义
    ElemType * elem;
    int length;
    int listsize;
}SqList;
typedef struct{  //线性表的集合类型定义
    struct {
        char name[30];
        SqList L;
    } elem[10];
    int length;
    int listsize;
}LISTS;
```

1.2.2 基本功能函数

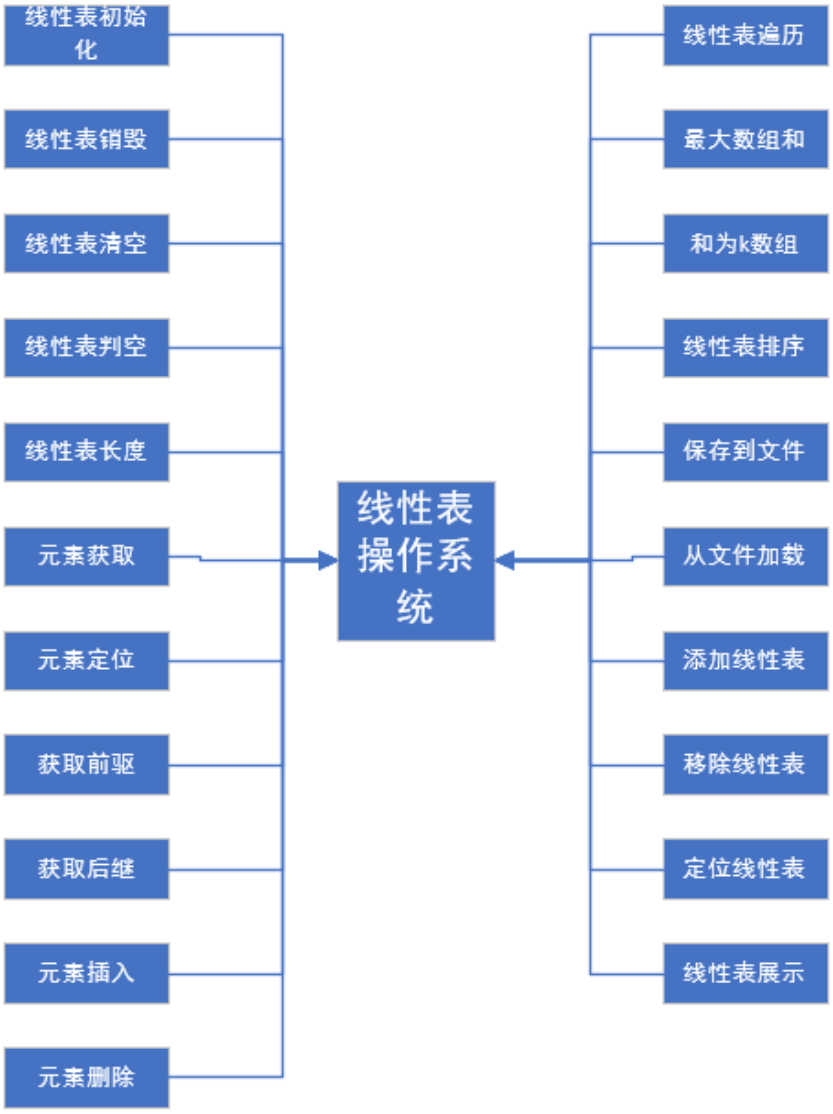

图 1-1 系统整体功能图

依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等 12 种基本运算，具体运算功能定义如下：

1. 初始化表：函数名称是 InitList(L)；初始条件是线性表 L 不存在；操作结果是构造一个空的线性表；
2. 销毁表：函数名称是 DestroyList(L)；初始条件是线性表 L 已存在；操作结果是销毁线性表 L；

3. 清空表：函数名称是 ClearList(L)；初始条件是线性表 L 已存在；操作结果是将 L 重置为空表；
4. 判定空表：函数名称是 ListEmpty(L)；初始条件是线性表 L 已存在；操作结果是若 L 为空表则返回 TRUE, 否则返回 FALSE；
5. 求表长：函数名称是 ListLength(L)；初始条件是线性表已存在；操作结果是返回 L 中数据元素的个数；
6. 获得元素：函数名称是 GetElem(L,i,e)；初始条件是线性表已存在，同时需要满足 1≤i≤ListLength(L)；操作结果是用 e 返回 L 中第 i 个数据元素的值；
7. 查找元素：函数名称是 LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回 L 中第 1 个与 e 满足关系 compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为 0；
8. 获得前驱：函数名称是 PriorElem(L,cur_e,pre_e)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是第一个，则用 pre_e 返回它的前驱，否则操作失败，pre_e 无定义；
9. 获得后继：函数名称是 NextElem(L,cur_e,next_e)；初始条件是线性表 L 已存在；操作结果是若 cur_e 是 L 的数据元素，且不是最后一个，则用 next_e 返回它的后继，否则操作失败，next_e 无定义；
10. 插入元素：函数名称是 ListInsert(L,i,e)；初始条件是线性表 L 已存在，同时需要满足 1≤i≤ListLength(L)+1；操作结果是在 L 的第 i 个位置之前插入新的数据元素 e。
11. 删除元素：函数名称是 ListDelete(L,i,e)；初始条件是线性表 L 已存在且非空，1≤i≤ListLength(L)；操作结果：删除 L 的第 i 个数据元素，用 e 返回其值；
12. 遍历表：函数名称是 ListTraverse(L,visit())，初始条件是线性表 L 已存在；操作结果是依次对 L 的每个数据元素调用函数 visit()。

1.2.3 附加功能函数

1. 最大连续子数组和：函数名称是 MaxSubArray(L); 初始条件是线性表 L 已存在且非空，请找出一个具有最大和的连续子数组（子数组最少包含一个元素），操作结果是其最大和；
2. 和为 K 的子数组：函数名称是 SubArrayNum(L,k); 初始条件是线性表 L 已

存在且非空, 操作结果是该数组中和为 k 的连续子数组的个数；

3. 顺序表排序：函数名称是 sortList(L)；初始条件是线性表 L 已存在；操作结果是将 L 由小到大排序；
4. 实现线性表的文件形式保存：其中，①需要设计文件数据记录格式，以高效保存线性表数据逻辑结构 (D,R) 的完整信息；②需要设计线性表文件保存和加载操作合理模式。
   (a) 文件写入：函数名称是 SaveList(L,FileName)；初始条件是线性表 L 已存在；操作结果是将 L 的元素写到名称为 FileName 的文件中。
   (b) 文件读出：函数名称是 LoadList(L,FileName)；初始条件是线性表 L 不存在；操作结果是将文件 FileName 中的元素读到表 L 中。
5. 实现多个线性表管理：设计相应的数据结构管理多个线性表的查找、添加、移除等功能。
   (a) 增加线性表：函数名称是 AddList(Lists, ListName)；初始条件是名称为 ListName 的线性表不存在于线性表集合中；操作结果是在 Lists 中创建一个名称为 ListName 的初始化好的线性表。
   (b) 移除线性表：函数名称是 RemoveList(Lists, ListName)；初始条件是名称为 ListName 的线性表存在于线性表集合中；操作结果是将该线性表移除。
   (c) 查找线性表：函数名称是 LocateList(Lists, ListName)；初始条件是名称为 ListName 的线性表存在于线性表集合中；操作结果是返回该线性表在 Lists 中的逻辑索引。
   (d) 选择表：函数名称是 ShowAllLists(Lists)；初始条件是 Lists 已存在。操作是将所有已经存储的线性表依次打印出来。

1.3 系统实现

1.3.1 演示系统框架

系统主体通过 while 循环实现多次选择，通过 op 获取用户的选择，通过 switch 语句根据用户选择实现具体功能，保证界面整洁和满足用户体验感。

具体实现为将菜单和功能实现写入到 while 循环中，用 op 获取用户的选择，op 初始化为 1，以便第一次能进入循环。进入循环后，用户输入选择 0~21，其

中 1~21 分别代表线性表的操作，在主函数中通过 switch 语句对应到相应的函数功能，执行完该功能后通过 break 跳出 switch 语句，继续执行 while 循环，直至用户输入 0 退出当前演示系统。系统初进入时默认对默认线性表（未创建）进行操作，后续可增加新表并进行选择，实现多线性表操作。

1.3.2 函数思想及实现

线性表基本功能函数的实现：

1. status InitList (SqList &L)

输入：线性表 L

输出：线性表初始化状态

函数思想描述：初始化函数调用全局定义的线性表 L，首先进行判空，如果为空，则对线性表的元素进行 malloc 分配内存，并将长度置为 0，返回 OK；如果不为空，则初始化失败，返回 INFEASIBLE；

2. status DestroyList (SqList &L)

输入：线性表 L

输出：线性表销毁状态

函数思想描述：销毁函数首先对 L 进行判空，如果为空，则无法销毁，返回 INFEASIBLE；如果不为空，则释放 L 的 elem 数组的内存，并将长度等置为 0，返回 OK；

3.status ClearList (SqList &L)

输入：线性表 L

输出：线性表清空状态

函数思想描述：清空函数首先对 L 判空，如果为空，则返回 INFEASIBLE；如果不为空，则将其长度置为 0，各元素置为 0，返回 OK；

4.status ListEmpty (SqList L)

输入：线性表 L

输出：线性表是否为空

函数思想描述：判空函数首先对 L 判空，如果为空，则返回 INFEASIBLE；如果元素长度为 0，则为空，返回 TRUE；否则返回 FALSE；

5.int ListLength (SqList L)

输入：线性表 L

输出：线性表的长度

函数思想描述：求线性表的长度函数首先对 L 判空，如果为空，则返回 INFEASIBLE；否则返回线性表 L 的长度；

6.status GetElem (SqList L, int i, ElemType &e)

输入：线性表 L，元素位序 i，元素值 e；

输出：元素获取的状态

函数思想描述：元素获取函数首先对 L 判空，如果为空，则返回 INFEASIBLE；当 i 小于线性表长度时，对位序为 i 的元素赋值为 e，返回 OK；否则返回 ERROR；

7.int LocateElem (SqList L, ElemType e)

输入：线性表 L，元素值 e

输出：定位元素的状态

函数思想描述：元素定位函数首先对 L 判空，如果为空，则返回 INFEASIBLE；然后对线性表内的元素进行查找，找到则返回元素的位序；否则返回 ERROR。

8.status PriorElem (SqList L, ElemType e, ElemType &pre)

输入：线性表 L，元素值 e，待赋值元素 pre

输出：查找前驱元素的状态

函数思想描述：获取前驱函数首先对 L 判空，如果为空，则返回 INFEASIBLE；然后对线性表元素进行遍历，如果找到等于 e 的元素，且其位序大于 0，则将位序为 i-1 的元素值赋值给 pre，返回 OK；否则返回 ERROR。

9.status NextElem (SqList L, ElemType e, ElemType &next)

输入：线性表 L，元素值 e，待赋值元素 next

输出：查找后继元素的状态

函数思想描述：获取后继元素函数首先对 L 判空，如果为空，则返回 INFEASIBLE；然后付线性表元素进行遍历，如果找到等于 e 的元素，且位序小于线性表长度-1，则将位序为 i+1 的元素值赋值给 next，返回 OK；否则返回 ERROR。

10.status ListInsert (SqList L, int i, ElemType e)

输入：线性表 L，位序 i，元素值 e

输出：元素插入状态

函数思想描述：元素插入函数首先对 L 判空，如果为空，则返回 INFEASIBLE；在 i 大于等于 1 且小于等于表长的情况下，将第 i 个元素后的元素后移，并将第 i 个位置的元素赋值为 e，返回 OK；如果线性表的长度已达最大，则重新分配更

多内存；其他情况返回 ERROR。

11.status ListDelete (SqList L, int i , ElemType &e)

输入：线性表 L，位序 i，元素值 e

输出：元素删除状态

函数思想描述：删除元素函数首先对 L 判空，如果为空，则返回 INFEASIBLE；当 i 在 1 到表长的范围内，将位序为 i-1 的元素值赋值给 e，并将后面的元素依次前移，表长减一，返回 OK；否则返回 ERROR。

12.status ListTraverse (SqList L)

输入：线性表 L

输出：线性表遍历状态

函数思想描述：线性表遍历函数对线性表的元素进行遍历，返回 OK；否则返回 ERROR。

附加功能函数的实现：

13. ElemType MaxSubArray(SqList L)

输入：线性表 L

输出：最大和的值

函数思想描述：求最大和函数首先对 L 判空，如果为空，则返回 INFEASIBLE；在线性表长度不为 0 的情况下，假定最大值为第零位元素，对后面的元素依次判断，如果 sum 大于 0，则 sum 等于 sum+ 第 i 个元素，否则等于第 i 个元素；当 sum 大于 maxsum 时将其赋值给 maxsum；遍历结束后返回 maxsum，即为最大和。

14.status SubArrayNum(SqList L, int k)

输入：线性表 L，值总和 k

输出：合为 k 的子数组的个数

函数思想描述：求和为 k 子数组的函数首先对 L 判空，如果为空，则返回 INFEASIBLE；首先置数目 count 为 0，然后遍历每一个元素，对其后面的元素依次相加直到最后，如果和为 k，则 count 加一，最终返回子数组个数 count。

15.void sortList(SqList& L)

输入：线性表 L

输出：void

函数思想描述：使用冒泡排序对线性表的元素进行从小到大的排序

16.status SaveList (SqList L, char FileName [])

输入：线性表 L，文件名 FileName

输出：文件保存的状态

函数思想描述：文件保存函数首先对 L 判空，如果为空，则返回 INFEASIBLE；然后以“r”形式读取文件，如果 fp 不为空，则关闭文件进行“w”写入，以达到覆盖原有内容的目的；然后将线性表的元素依次写入到文件中，关闭文件返回 OK；否则返回 ERROR。

17.status LoadList (SqList L, char FileName [])

输入：线性表 L，文件名 FileName

输出：文件载入的状态

函数思想描述：文件载入函数首先对 L 判空，如果为空，则返回 INFEASIBLE；对文件以“r”只读方式打开，首先为 L 的元素分配内存，然后读取文件中的值到 elem 数组中，关闭文件返回 OK；否则返回 ERROR。

18.status AddList (LISTS &Lists, char ListName [])

输入：顺序表数组 Lists，表名 ListName

输出：添加线性表的状态

函数思想描述：添加线性表的函数首先将表的数量加一，将表名赋值给最新的表的表名，初始化一个新的表 l，并将该表传递给新表，返回 OK；否则返回 ERROR。

19.status RemoveList (LISTS List, char ListName [])

输入：顺序表数组 Lists，数组名 ListName

输出：线性表移除状态

函数思想描述：线性表移除函数对表名进行查找，如果查找成功，则销毁该表，并对其后的线性表进行前移操作，返回 OK；否则返回 ERROR。

20.int LocateList (LISTS Lists, char ListName [])

输入：顺序表数组 Lists，文件名 ListName

输出：线性表的位置

函数思想描述：位置查找函数遍历所有线性表，如果找到与指定名称相同的表，则返回其位置，否则返回 ERROR。

21.void TraverseList(LISTS Lists)

输入：顺序表数组 Lists

输出：void

函数思想描述：展示函数遍历所有线性表，并一一输出他们，无返回值。

1.4 系统测试

系统菜单整体布局如图：

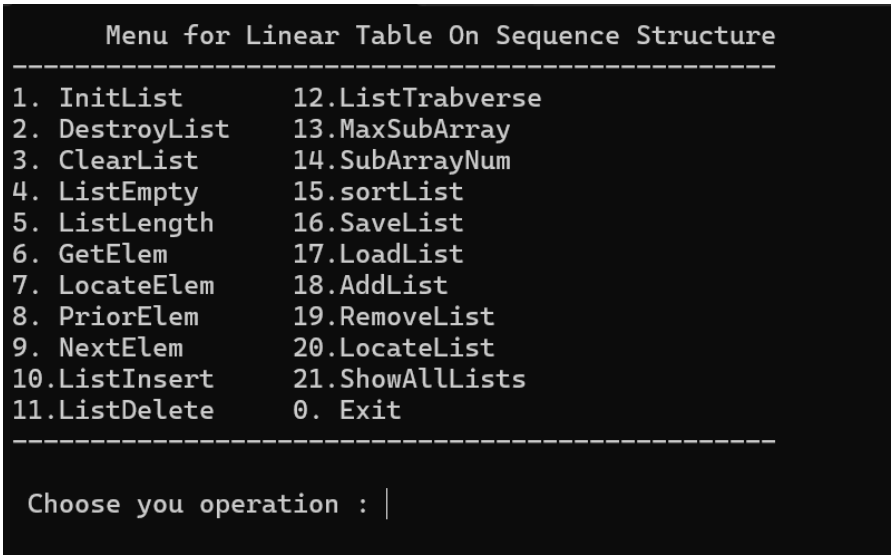

图 1-2 菜单

测试集如下：

测试集 1：线性表中存有元素 1 2 3

测试集 2：线性表集合中有两个线性表表一: 1 3 5 ; 表二: 2 4 6

1.4.1 基本功能函数测试

1.InitList 测试

测试 1：测试函数是否能成功创建线性表；

测试 2：测试当线性表已经存在时，函数是否能再次创建线性表。

测试编号	测试输入	预期结果	实际运行结果
1	1	线性表创建成功	一致
2	1→1	线性表创建失败	一致

```
 Choose you operation : 1
Linear table was successfully created !

 Choose you operation : 1
Linear table creation failed !
```

图 1-3　测试 1 运行结果

2.DestroyList 测试

测试 1：测试函数是否能对不存在的线性表进行销毁；

测试 2：测试函数是否能对已经存在的线性表进行销毁；

测试 3：将测试在销毁线性表之后检测能否重新创建线性表。

测试编号	测试输入	预期结果	实际运行结果
1	2	线性表不存在，销毁失败	一致
2	1→2	线性表销毁成功	一致
3	1→2→1	线性表销毁成功；线性表创建成功	一致

```
 Choose you operation : 1
Linear table was successfully created !

 Choose you operation : 2
Linear table was successfully destroyed !

 Choose you operation : 1
Linear table was successfully created !

 Choose you operation : 10
Please choose your position and number to insert : 1 2
The number is successfully inserted !

 Choose you operation : 2
Linear table was successfully destroyed !

 Choose you operation : 1
Linear table was successfully created !
```

图 1-4　测试 2 运行结果

3.ClearList 测试

测试 1：测试函数是否能对不存在的线性表进行清空；

测试 2：测试函数是否能对已经存在的线性表进行清空；

测试 3：在测试集 1 的情况下进行，在调用 ClearList 之后，通过求线性表的表长，来判断线性表中的元素是否确实被清空。

测试编号	测试输入	预期结果	实际运行结果
1	3	线性表不存在，清空失败	一致
2	1→3	线性表清空成功	一致
3	1→ 测试集 1→3→5	线性表长度为 0	一致

```
 Choose you operation : 1
Linear table was successfully created !

 Choose you operation : 3
Linear table was successfully cleared !

 Choose you operation : 2
Linear table was successfully destroyed !

 Choose you operation : 3
Linear table clear failed !

 Choose you operation : 1
Linear table was successfully created !

 Choose you operation : 3
Linear table was successfully cleared !

 Choose you operation : 5
The length of this linear table is 0 !
```

图 1-5　测试 3 运行结果

4.ListEmpty 测试

测试 1：测试函数能否对不存在的线性表判空；

测试 2：测试函数能否正确判断空线性表；

测试 3：在测试集 1 的情况下进行，测试函数能否正确判断空线性表。

测试编号	测试输入	预期结果	实际运行结果
1	4	线性表不存在，判空失败	一致
2	1→4	线性表为空	一致
3	1→ 测试集 1→4	线性表非空	一致

```
 Choose you operation : 1
Linear table was successfully created !

 Choose you operation : 4
The linear table is empty !
```

图 1-6　测试 4 运行结果

5.ListLength 测试

测试 1：测试函数能否对不存在的线性表求长；

测试 2：测试函数能否正确求出空线性表的长度；

测试 3：在测试集 1 的情况下进行，测试函数能否正确求出线性表的长度。

测试编号	测试输入	预期结果	实际运行结果
1	5	线性表不存在，求长失败	一致
2	1→5	线性表长度为 0	一致
3	1→ 测试集 1→5	线性表长度为 3	一致

```
 Choose you operation : 1
Linear table was successfully created !

 Choose you operation : 5
The length of this linear table is 0 !

 Choose you operation : 10
Please choose your position and number to insert : 1 1
The number is successfully inserted !

 Choose you operation : 10
Please choose your position and number to insert : 2 2
The number is successfully inserted !

 Choose you operation : 10
Please choose your position and number to insert : 3 3
The number is successfully inserted !

 Choose you operation : 5
The length of this linear table is 3 !
```

图 1-7　测试 1 运行结果

6.GetElem 测试

此函数的所有测试将在测试集 1 的情况下进行。

测试 1，2：将测试函数能否正确找到元素；

测试 3，4：将测试函数能否正确识别非法的位序。

测试编号	测试输入	预期结果	实际运行结果
1	6→2	线性表中第 2 个元素为 2	一致
2	6→5	线性表中第 3 个元素为 3	一致
3	6→-1	输入的逻辑索引不合法！	一致
4	6→5	输入的逻辑索引不合法！	一致

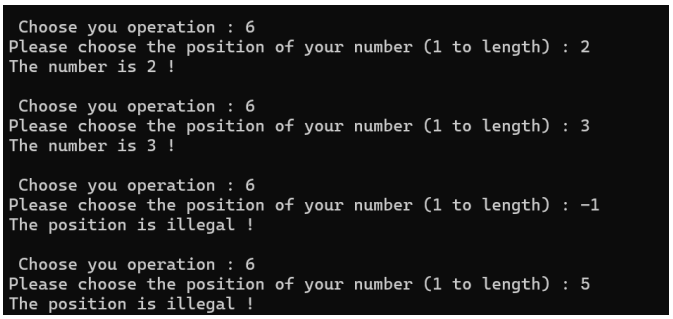

图 1-8　测试 2 运行结果

7.LocateElem 测试

此函数的所有测试将在测试集 1 的情况下进行。

测试 1：将测试函数能否正确找到位序；

测试 3，4：将测试函数能否正确识别不在线性表中的元素。

测试编号	测试输入	预期结果	实际运行结果
1	7→1	该元素存在且元素逻辑索引为：1	一致
3	7→5	输入的元素不存在！	一致
4	7→-1	输入的元素不存在！	一致

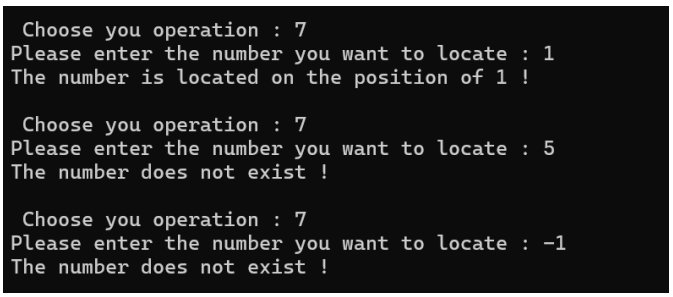

图 1-9　测试 2 运行结果

8.PriorElem 测试

此函数的所有测试将在测试集 1 的情况下进行。

测试 1：将测试函数能否正确找到前驱；

测试 2：将测试函数能否正确判断第一个元素没有前驱；

测试 3：将测试函数能否正确判断不在线性表中的元素没有前驱。

测试编号	测试输入	预期结果	实际运行结果
1	8→2	该元素存在且前驱元素为：1	一致
2	8→1	线性表中该元素不存在前驱！	一致
3	8→4	线性表中该元素不存在！	一致

```
 Choose you operation : 8
Please enter your number : 2
The prior number is 1 !

 Choose you operation : 8
Please enter your number : 1
This number does not have a prior number in the table !

 Choose you operation : 8
Please enter your number : 4
This number does not have a prior number in the table !
```

图 1-10　测试 2 运行结果

9.NextElem 测试

此函数的所有测试将在测试集 1 的情况下进行。

测试 1：将测试函数能否正确找到后继；

测试 2：将测试函数能否正确判断最后一个元素没有后继；

测试 3：将测试函数能否正确判断不在线性表中的元素没有后继。

测试编号	测试输入	预期结果	实际运行结果
1	9→2	线性表中该元素的后继为 3	一致
2	9→3	线性表中该元素不存在后继！	一致
3	9→4	线性表中该元素不存在！	一致

```
 Choose you operation : 9
Please enter your number : 2
The next number is 3 !

 Choose you operation : 9
Please enter your number : 3
This number does not have a next number in the table !

 Choose you operation : 9
Please enter your number : 4
This number does not have a next number in the table !
```

图 1-11　测试 2 运行结果

10. ListInsert 测试

输入要求：依次输入插入元素位置和插入元素。

测试 1 在空线性表的情况下进行，通过反复调用函数来构建测试集 1（1 2 3），将通过遍历线性表和求表长来检验插入是否正确；

测试 2，3 将在测试 1 的基础上进行，测试函数能否正确判断线性表两端非法的插入位置；

测试编号	测试输入	预期结果	实际运行结果
1	10→1 1→10→2 2→10→3 3	线性表插入成功！	一致
2	10→7	插入位置不合法，线性表插入失败！	一致
3	10→0	插入位置不合法，线性表插入失败！	一致

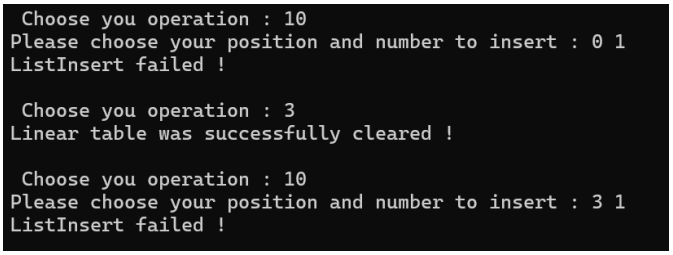

图 1-12　测试 2 运行结果

11. ListDelete 测试

输入要求：输入删除的序号

此函数的所有测试将在测试集 1 的情况下进行，进行一项测试后，不恢复至测试集 1 的状态。

测试 1：将测试函数能否正确判断非法的序号；

测试 2：将测试函数能否正确删除元素，采用遍历的方式检验正确性；

测试编号	测试输入	预期结果	实际运行结果
1	11→0	删除位置不合法！	一致
2	11→2→2	元素已删除！遍历后的结果为 1 3	一致

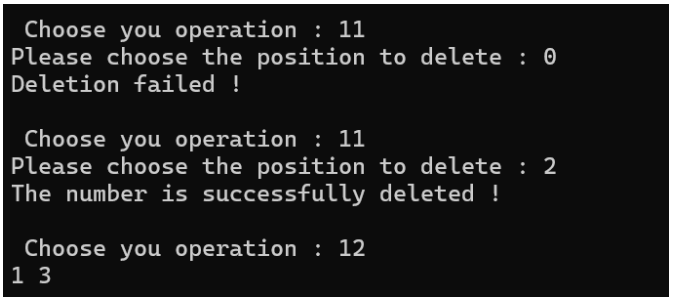

图 1-13　测试 2 运行结果

12. ListTraverse 测试

测试 1 在线性表是空表的情况下进行，测试函数能否处理空表的情况；

测试 2 在测试集 1 的情况下进行，测试函数能否正确遍历线性表。

测试编号	测试输入	预期结果	实际运行结果
1	3→12	线性表是空表	一致
2	12	1 2 3	一致

```
 Choose you operation : 12
1 2 3

 Choose you operation : 3
Linear table was successfully cleared !

 Choose you operation : 12
```

图 1-14　测试 2 运行结果

1.4.2　附加功能测试

13. MaxSubArray 测试

测试 1：在没有创建线性表的情况下进行，测试函数能否正确判断线性表的存在性；

测试 2：在测试集 1 的情况下进行，测试函数能否实现求最大连续子数组和的功能。

测试编号	测试输入	预期结果	实际运行结果
1	2→13	线性表未创建！	一致
2	13	最大子数组之和为：6	一致

```
 Choose you operation : 13
MaxSum is 6

 Choose you operation : 3
Linear table was successfully cleared !

 Choose you operation : 13
MaxSum is 0
```

图 1-15　测试 2 运行结果

14. SubArrayNum 测试

测试 1：在没有创建线性表的情况下进行，测试函数能否正确判断线性表的

存在性；

测试 2：在测试集 1 的情况下进行，测试函数能否实现计数和为 K 的子数组的功能。

测试编号	测试输入	预期结果	实际运行结果
1	2→14	线性表未创建！	一致
2	14→3	和为数 3 的连续数组数目为：2	一致

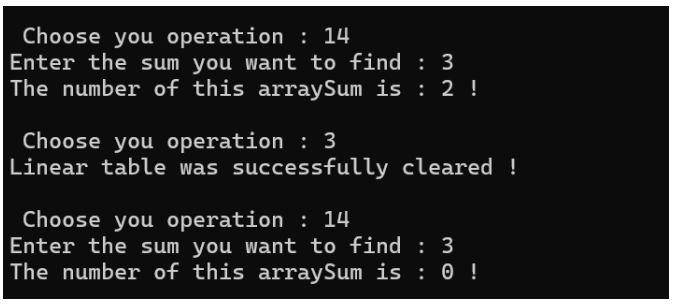

图 1-16　测试 2 运行结果

15.sortList 测试

测试 1：在线性表存在的情况下进行，测试函数能否正确排序。

测试编号	测试输入	预期结果	实际运行结果
1	12→15→12	1 2 3	一致

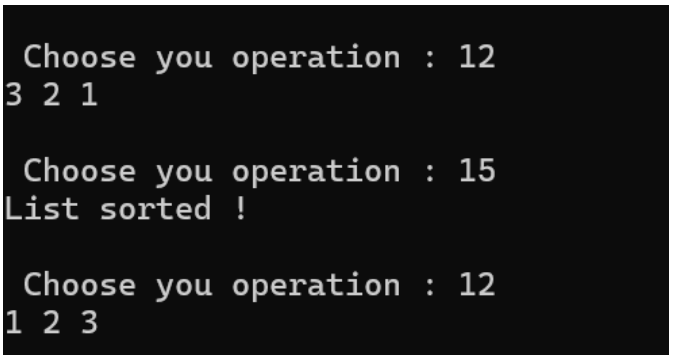

图 1-17　测试 2 运行结果

16. SaveList 测试

测试 1：在测试集 1 的情况下进行，测试函数能否正常进行写文件操作；

测试编号	测试输入	预期结果	实际运行结果
1	16→list1.txt	文件保存成功！	一致

17. LoadList 测试

本函数的测试都在文件 1 中已存有测试集 1 的情况下进行。

测试 1：在线性表不存在的情况下进行，测试函数能否正确进行读文件操作，采用遍历线性表的方式检验正确性。

测试编号	测试输入	预期结果	实际运行结果
1	17→list1.txt	文件录入成功！遍历：1 2 3	一致

18. AddList 测试

测试 1：构建测试集 2，测试线性表能否正确添加，通过遍历各个表检验正确性。

测试编号	测试输入	预期结果	实际运行结果
1	18	表一表二	一致

```
 Choose you operation : 18
Please enter your listname : 表一
Please enter your elements amount and members : 3 1 2 3
List added !

 Choose you operation : 18
Please enter your listname : 表二
Please enter your elements amount and members : 3 3 2 1
List added !

 Choose you operation : 21
表一 1 2 3
表二 3 2 1
```

图 1-18　测试 2 运行结果

19. RemoveList 测试

本函数的测试在测试集 2 的基础上进行。

测试 1：将测试函数能否正确删除线性表，采用遍历各线性表的方式判断正确性；

测试 2：在测试 1 的基础上进行，尝试删除一个不在集合中的线性表，测试函数能否给出正确判断。

测试编号	测试输入	预期结果	实际运行结果
1	19→ 表一	表一已成功删除！	一致
2	19→ 表三	线性表不存在！	一致

```
 Choose you operation : 19
Enter the listname to remove : 表一
List removed !

 Choose you operation : 21
表二 3 2 1

 Choose you operation : 19
Enter the listname to remove : 表三
Cant remove the list !
```

图 1-19　测试 2 运行结果

20. LocateList 测试

本函数的测试在测试集 2 的基础上进行。

测试 1：将测试函数能否正确定位线性表；

测试 2：将尝试查找一个不在集合中的线性表，测试函数能否给出正确判断。

测试编号	测试输入	预期结果	实际运行结果
1	20→ 表一	该线性表的逻辑索引为：1	一致
2	20→ 表三	线性表查找失败！	一致

```
 Choose you operation : 20
Enter the name to locate : 表一
The position is : 1

 Choose you operation : 20
Enter the name to locate : 表三
The list does not exist !
```

图 1-20　测试 2 运行结果

21. TraverseList 测试

测试 1：在测试集 2 的基础上进行，测试函数能否正确遍历各个线性表；

测试编号	测试输入	预期结果	实际运行结果
1	（测试集 3）21	表一表二	一致

测试小结

21 个函数基本符合了测试要求，在正常和异常用例的条件下均可以正常运行。

1.5 实验小结

本次实验让我加深了对线性表的概念、基本运算的理解，掌握了线性表的基本运算的实现，熟练了线性表的逻辑结构和物理结构的关系。

在编写程序和测试的过程中，遇到了诸多问题，例如如何设计多线性表操作，如何保证能够单独对某一线性表进行基本操作。解决方案是将其完整赋值给主函数中的全局变量，保证了集合中表的独立性，可以不受主函数中操作的影响，表之间可以分立进行。

在今后的学习过程当中应该更多地从数据结构的角度去分析如何进行数据的存储、读取和处理，如何设计便于存储的数据结构，同时应具有开放性思维，设计高性能的算法，以达到更简便地解决实际问题的目的。同时，以后还需要多加练习，以达到熟能生巧的效果。

2 基于二叉链表的二叉树实现

2.1 问题描述

采用二叉链表作为二叉树的物理结构，实现基本运算。ElemType 为数据元素的类型名，具体含义可自行定义，但要求二叉树结点类型为结构型，至少包含二个部分，一个是能唯一标识一个结点的关键字（类似于学号或职工号），另一个是其它部分。

要求构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

演示系统可选择实现二叉树的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存二叉树数据逻辑结构 (D,R) 的完整信息；②需要设计二叉树文件保存和加载操作合理模式。附录 B 提供了文件存取的方法。演示系统可选择实现多个二叉树管理。可采用线性表的方式管理多个二叉树，线性表中的每个数据元素为一个二叉树的基本属性，至少应包含有二叉树的名称。

演示系统的源程序应按照代码规范增加注释和排版，目标程序务必是可以独立于 IDE 运行的 EXE 文件。

2.2 系统设计

2.2.1 头文件和预定义

1、头文件

```
1 #include "stdio.h"
2 #include "stdlib.h"
3 #include <malloc.h>
4 #include <string.h>
```

2、预定义常量

```
1 #define TRUE 1
2 #define FALSE 0
3 #define OK 1
4 #define ERROR 0
5 #define INFEASIBLE -1
```

```
6 #define OVERFLOW -2
7 #define MAXlength 10
```

3、类型表达式

```
1  typedef int status;
2  typedef int KeyType;
3  typedef struct {
4      KeyType key;
5      char others[20];
6  } TElemType; // 二叉树结点类型定义
7  typedef struct BiTNode { // 二叉链表结点的定义
8      TElemType data;
9      struct BiTNode *lchild, *rchild;
10 } BiTNode, *BiTree;
11 typedef struct {
12     struct {
13         char name[20];
14         BiTree T;
15     } elem[10];
16     int amount;
17 } Trees;
18 typedef struct {
19     int pos;
20     TElemType data;
21 } DEF;
```

2.2.2 基本功能函数

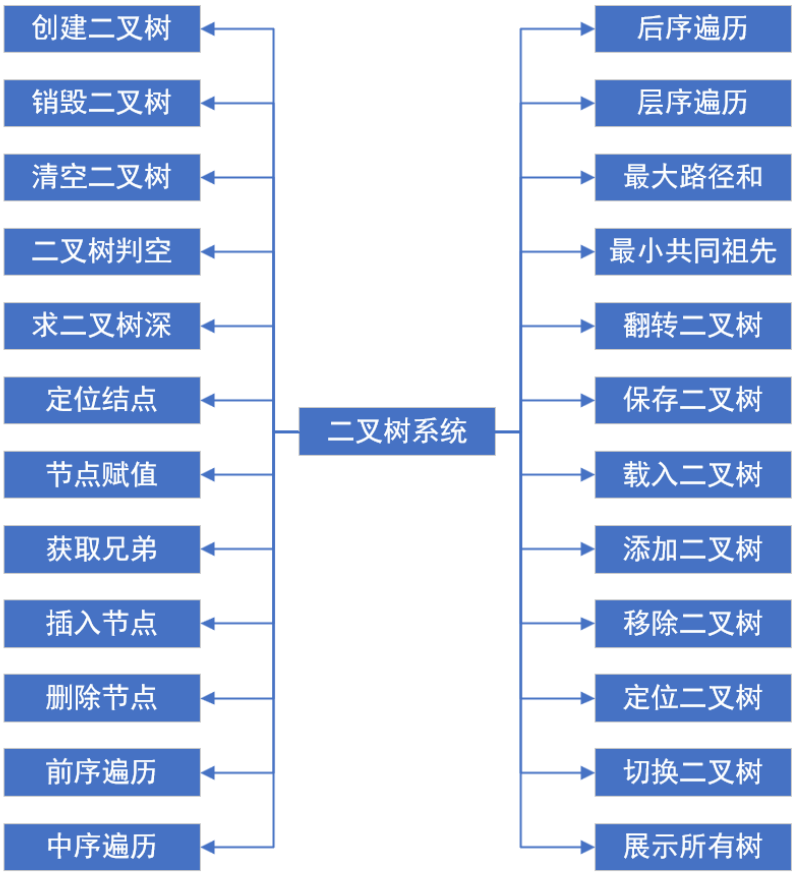

图 2-1 系统整体功能设计图

1. 创建二叉树：函数名称是 CreateBiTree(T,definition)；初始条件是 definition 给出二叉树 T 的定义；操作结果是按 definition 构造二叉树 T；
2. 销毁二叉树：函数名称是 DestroyBiTree(T)；初始条件是二叉树 T 已存在；操作结果是销毁二叉树 T；
3. 清空二叉树：函数名称是 ClearBiTree (T)；初始条件是二叉树 T 存在；操作结果是将二叉树 T 清空；
4. 判定空二叉树：函数名称是 BiTreeEmpty(T)；初始条件是二叉树 T 存在；操作结果是若 T 为空二叉树则返回 TRUE，否则返回 FALSE；
5. 求二叉树深度：函数名称是 BiTreeDepth(T)；初始条件是二叉树 T 存在；操作结果是返回 T 的深度；
6. 查找结点：函数名称是 LocateNode(T,e)；初始条件是二叉树 T 存在，e 是和

T 中结点关键字类型相同的值；操作结果是返回查找到的结点指针，否则返回 NULL；

7. 结点赋值：函数名称是 Assign(T,e,value)；初始条件是二叉树 T 已存在，e 是和 T 中结点关键字类型相同的值；操作结果是关键字为 e 的结点赋值为 value，返回 OK，否则返回 FALSE；
8. 获得兄弟结点：函数名称是 GetSibling(T,e)；初始条件是二叉树 T 存在，e 是和 T 中结点关键字类型相同的值；操作结果是返回关键字为 e 的结点的兄弟结点指针。否则返回 NULL；
9. 插入结点：函数名称是 InsertNode(T,e,LR,c)；初始条件是二叉树 T 存在，e 是和 T 中结点关键字类型相同的值，LR 为 0 或 1，c 是待插入结点；操作结果是根据 LR 为 0 或者 1，插入结点 c 到 T 中，作为关键字为 e 的结点的左或右孩子结点，结点 e 的原有左子树或右子树则为结点 c 的右子树；当 LR 为-1 时新增节点作为根节点，原树作为该节点的右子树，操作成功返回 OK，否则返回 FALSE。
10. 删除结点：函数名称是 DeleteNode(T,e)；初始条件是二叉树 T 存在，e 是和 T 中结点关键字类型相同的值。操作结果是删除 T 中关键字为 e 的结点；同时，如果关键字为 e 的结点度为 0，删除即可；如关键字为 e 的结点度为 1，用关键字为 e 的结点孩子代替被删除的 e 位置；如关键字为 e 的结点度为 2，用 e 的左孩子代替被删除的 e 位置，e 的右子树作为 e 的左子树中最右结点的右子树；
11. 前序遍历：函数名称是 PreOrderTraverse(T,Visit())；（本系统前序遍历采用非递归算法）初始条件是二叉树 T 存在，Visit 是一个函数指针的形参；操作结果是先序遍历，对每个结点调用函数 Visit 一次且一次，一旦调用失败，则操作失败。
12. 中序遍历：函数名称是 InOrderTraverse(T,Visit())；初始条件是二叉树 T 存在，Visit 是一个函数指针的形参；操作结果是中序遍历，对每个结点调用函数 Visit 一次且一次，一旦调用失败，则操作失败；
13. 后序遍历：函数名称是 PostOrderTraverse(T,Visit())；初始条件是二叉树 T 存在，Visit 是一个函数指针的形参；操作结果是后序遍历，对每个结点调用函数 Visit 一次且一次，一旦调用失败，则操作失败。
14. 按层遍历：函数名称是 LevelOrderTraverse(T,Visit())；初始条件是二叉树 T

存在，Visit 是一个函数指针的形参；操作结果是层序遍历，对每个结点调用函数 Visit 一次且一次，一旦调用失败，则操作失败。

2.2.3 附加功能函数

1. 最大路径和：函数名称是 MaxPathSum(T)，初始条件是二叉树 T 存在；操作结果是返回根节点到叶子结点的最大路径和；
2. 最近公共祖先：函数名称是 LowestCommonAncestor(T,e1,e2)；初始条件是二叉树 T 存在；操作结果是该二叉树中 e1 节点和 e2 节点的最近公共祖先；
3. 翻转二叉树：函数名称是 InvertTree(T)，初始条件是线性表 L 已存在；操作结果是将 T 翻转，使其所有节点的左右节点互换；
4. 文件写入：函数名称是 SaveBiTree(T,FileName)；初始条件是二叉树 T 已存在；操作结果是将 T 的元素写到名称为 FileName 的文件中。
5. 文件读出：函数名称是 LoadBiTree(T,FileName)；初始条件是二叉树 T 不存在；操作结果是将文件 FileName 中的内容载入到新表 T 中。
6. 实现多个二叉树管理：设计相应的数据结构管理多个二叉树的查找、添加、移除、切换、展示功能。
   (a) 增加二叉树：函数名称是 AddBiTree(trees, TreeName)；初始条件是名称为 TreeName 的二叉树不存在于二叉树集合中；操作结果是在 trees 中创建一个名称为 TreeName 的二叉树。
   (b) 移除二叉树：函数名称是 RemoveBiTree(trees, TreeName)；初始条件是名称为 TreeName 的二叉树存在于二叉树集合中；操作结果是将该二叉树移除。
   (c) 查找二叉树：函数名称是 LocateBiTree(trees, TreeName)；初始条件是名称为 TreeName 的二叉树存在于二叉树集合中；操作结果是返回该二叉树在 trees 中的逻辑索引。
   (d) 切换二叉树：函数名称是 SwitchTrees(trees,TreeName,T)；初始条件是名称为 TreeName 的二叉树已存在于二叉树集合中；操作结果是将该树调转为当前操作的树 T。
   (e) 遍历所有表：函数名称是 ShowAllTrees(trees)；初始条件是 trees 已存在；操作结果是对所有的二叉树依次进行前序和中序遍历。

2.2.4 演示系统

构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。演示系统可选择实现二叉树的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存二叉树数据逻辑结构 (D,[R]) 的完整信息；②需要设计二叉树文件保存和加载操作合理模式。演示系统实现了多个二叉树管理。输入 1~24 可以调用上述的 24 个函数，对二叉树或二叉树集合进行操作；输入 0 时退出系统。

该演示系统具有完备性，包含每一功能的中英文说明和注意事项，同时显示当前二叉树名称和初始化情况。

下面是该系统的初始界面

```
 Menu for BiTree Structure
-------------------------------------------------
1. CreateBiTree        13.PostOrderTraverse
2. DestroyBiTree       14.LevelOrderTraverse
3. ClearBiTree         15.MaxPathSum
4. BiTreeEmpty         16.LowestCommonAncestor
5. BiTreeDepth         17.InvertTree
6. LocateNode          18.SaveBiTree
7. Assign              19.LoadBiTree
8. GetSibling          20.AddTree
9. InsertNode          21.RemoveTree
10.DeleteNode          22.LocateTree
11.PreOrderTraverse    23.SwitchTrees
12.InOrderTraverse     24.ShowAllTrees
0. Exit-------------------------------------------------

 Choose your operation :
```

图 2-2 系统整体界面

2.3 系统实现

2.3.1 演示系统框架

系统主体通过 while 循环实现多次选择，通过 op 获取用户的选择，通过 switch 语句根据用户选择实现具体功能。

具体实现为将菜单和功能实现写入到 while 循环中，用 op 获取用户的选择，op 初始化为 1，以便第一次能进入循环。进入循环后，用户输入选择 0~24，其中 1~24 分别代表二叉树的一个基本操作，在主函数中通过 switch 语句对应到相应的函数功能，执行完该功能后通过 break 跳出 switch 语句，继续执行 while 循

环，直至用户输入 0 退出当前演示系统。

2.3.2 函数思想及实现

二叉树基本功能函数的实现：

1. status CreateBiTree(BiTree &T, TElemType definition[])

输入：二叉树 T，TELemType 类型数组

输出：创建二叉树的状态

函数思想描述：创建二叉树函数，传入二叉树和 TELemType 类型数组。根据 defination 的位置判断奇偶来确定左右子树，最后将 T 赋值根节点，从而构建二叉树。

2. status DestroyBiTree(BiTree &T)

输入：二叉树 T

输出：二叉树销毁状态

函数思想描述：销毁二叉树函数，将二叉树设置成空，并删除所有结点，释放结点空间。在函数中，如果当前结点有左子树或右子树，则递归调用本函数依次销毁当前结点的左子树和右子树；如果当前结点的左右子树都为空指针时释放当前结点的存储空间，并将当前结点的指针设置为 NULL。

3.status ClearBiTree(BiTree &T)

输入：二叉树 T

输出：二叉树清空状态

函数思想描述：清空二叉树函数，将二叉树设置成空。在函数中，如果当前结点有左子树或右子树，则递归调用本函数依次清空当前结点的左子树和右子树。

4.status BiTreeEmpty(BiTree T)

输入：二叉树 T

输出：二叉树是否为空

函数思想描述：判断树是否为空，如果根结点为空，则返回 TRUE，否则说明树非空，返回 FALSE。

5.int BiTreeDepth(BiTree T)

输入：二叉树 T

输出：二叉树的深度

函数思想概述：求二叉树深度函数，依次遍历二叉树的左右结点，遇到空结点则返回 0，不断比较获取二叉树到叶子结点的深度的较大值，作为二叉树的深度，最终结果即为二叉树深度。

6.BiTNode *LocateNode(BiTree T, KeyType e)

输入：二叉树 T，KeyType e

输出：二叉树结点指针

函数思想描述：查找二叉树中关键字结点函数，使用递归思想，若当前结点为目标结点，则返回当前结点的地址值；若当前结点指针为空，则说明此子树都没有所找结点，返回 NULL 指针；否则依次递归查找左右子树中是否有目标结点，并记录在左右子树查找的返回值。整个函数的返回值若为 NULL，则说明整棵树没有目标结点，否则返回目标结点。

7.status Assign(BiTree &T, KeyType e, TElemType value)

输入：二叉树 T，KeyType e，TElemType value

输出：赋值状态

函数思想描述：结点赋值函数，首先通过 LocateNode 函数查找用以替换的结点的关键字与除被替换结点以外的其他结点的关键字是否有重复，若重复则返回 ERROR。若无重复则调用 LocateNode 函数查找被替换结点，若未找到则返回 ERROR；若找到，则对该结点的关键字和关键信息进行赋值，并返回 OK。

8.BiTNode *GetSibling(BiTree T, KeyType e)

输入：二叉树 T，KeyType e

输出：二叉树兄弟结点指针

函数思想概述：获得当前结点的兄弟结点函数，使用了递归思想，若当前结点为空则说明为空树，若当前结点无左右子树则说明当前结点不符合条件，返回 NULL；若当前结点有左右子树，则若左右子树其一结点关键字为所求，且另一子树存在，则找到兄弟结点，返回兄弟结点指针，若另一子树不存在，则该节点没有兄弟结点，返回 NULL；若当前结点的左右子树都不为指定节点，则递归调用此函数在左右子树中查找，并记录返回结果。整个函数若最终返回值为 NULL 则说明该树没有此结点或此结点无兄弟节点；若返回值不为 NULL 则说明在某一子树中找到了指定节点的兄弟结点，返回该结点的值。

9.status InsertNode(BiTree &T, KeyType e, int LR, TElemType c)

输入：二叉树 T，KeyType e，整型变量 LR ，TElemType c

输出：结点插入状态

函数思想概述：插入结点函数，首先通过 LocateNode 函数判断待插入结点在树中有无关键字重复，若有则返回 ERROR，若无重复则调用 LocateNode 函数查找被插入结点，然后根据 LR 为 0 或者 1，插入结点 c 到 T 中，作为关键字为 e 的结点的左或右孩子结点，结点 e 的原有左子树或右子树则为结点 c 的右子树，返回 OK。如果插入失败，返回 ERROR。特别地，当 LR 为-1 时，作为根结点插入，原根结点作为 c 的右子树，最后返回 OK。

10.status DeleteNode(BiTree &T, KeyType e)

输入：二叉树 T，KeyType e

输出：节点删除状态

函数思想概述：删除结点函数，e 是 T 中结点关键字类型相同的给定值。删除 T 中关键字为 e 的结点；如果关键字为 e 的结点度为 0，删除即可；如关键字为 e 的结点度为 1，用关键字为 e 的结点孩子代替被删除的 e 位置；如关键字为 e 的结点度为 2，用 e 的左孩子代替被删除的 e 位置，e 的右子树作为 e 的左子树中最右结点的右子树。成功删除结点后返回 OK，否则返回 ERROR。

11.status PreOrderTraverse(BiTree T, void (*visit)(BiTree))

输入：二叉树 T，函数指针 *visit

输出：前序遍历状态

函数思想概述：先序遍历函数，采用了非递归算法。在函数中，若当前结点为空，则返回 ERROR；若当前结点不为空，则首先通过 visit 访问当前结点，然后利用栈的特性依次对左右子树结点进栈出栈实现遍历。

12.status InOrderTraverse(BiTree T, void (*visit)(BiTree))

输入：二叉树 T，函数指针 *visit

输出：中序遍历状态

函数思想概述：中序遍历函数，用递归算法实现。在函数中，若当前结点为空，则返回 ERROR；若当前结点不为空，则首先递归调用本函数依次中序遍历当前结点的左子树，根节点和右子树。

13.status PostOrderTraverse(BiTree T, void (*visit)(BiTree))

输入：二叉树 T，函数指针 *visit

输出：后序遍历状态

函数思想概述：后序遍历函数，采用了递归思想。在函数中，若当前结点为

空，则返回 ERROR；若当前结点不为空，则首先递归调用本函数依次后序遍历当前结点的左子树和右子树，然后访问当前结点。

14.status LevelOrderTraverse(BiTree T, void (*visit)(BiTree))

输入：二叉树 T，函数指针 *visit

输出：层序遍历状态

函数思想概述：层序遍历函数，用非递归形式实现，调用队列数据结构和相应操作函数。在函数中，首先定义并初始化队列，然后定义 p 为根节点，并将根结点进队列。然后开始循环，该循环在队列为空时结束：首先将根节点出队列，若此节点不为空则访问该结点并依次将该结点的左右子树进队列，该循环结束则说明已经遍历完所有结点。在队列中，根节点下一层的子树先进队列。再依此顺序将左右子树的下一层子树再进队列，以此类推，直到最后一层子树。所以在每一层内的遍历顺序为从左向右。

附加功能函数的实现：

15.int MaxPathSum(BiTree T)

输入：二叉树 T

输出：最大路径和

函数思想描述：最大路径和函数，通过不断递归到叶子节点到空树，比较左右子节点的大小，选择较大值直到无法继续，然后返回路径上的最大值。

16. BiTree LowestCommonAncestor(BiTree T, int e1, int e2)

输入：二叉树 T，整型变量 e1，整型变量 e2

输出：二叉树结点：最近公共祖先

函数思想描述：递归查找左子树和右子树，若查找到目标结点，则返回该结点的指针，若查找到空树，则返回 NULL。然后判断若左结点的返回值和右节点的返回值均不为 NULL，则返回该结点，若左结点和右节点存在一个结点不为空，则返回该结点；否则返回 NULL。最终返回的结点为两结点的最近公共祖先。

时间复杂度：O(nlogn)

空间复杂度：O(1)

17.status InvertTree(BiTree &T)

输入：二叉树 T

输出：函数运行状态

函数思想描述：将二叉树翻转写成函数，以递归形式实现，依次交换左右子

树直至无法交换，最终得到翻转后的二叉树。

时间复杂度：O(nlogn)

空间复杂度：O(1)

18.status SaveBiTree(BiTree T, char FileName[])

输入：二叉树 T，字符串变量 FileName

输出：文件保存状态

函数思想描述：文件保存函数，将二叉树的结点数据写入到文件 FileName 中。该函数借助 SaveBiTreeHelper 函数实现递归算法遍历，并将遍历结果写入文件中。

19.status LoadBiTree(BiTree &T, char FileName[])

输入：二叉树 T，字符串变量 FileName

输出：文件载入状态

函数思想描述：文件载入函数，将文件 FileName 中的数据读取到空树中，该函数借助 LoadBiTreeHelper 函数实现递归算法遍历，并以先序遍历的顺序写入空树中。

20.status AddBiTree(TREES &trees,char TreeName[])

输入：结构体类型变量 trees，字符串类型 TreeName

输出：二叉树添加状态

函数思想描述：增加树函数，在 trees 数组尾部新增树，并将树名称存储在该树结点的 name 分量当中。在添加二叉树之前，判断名称是否唯一，如果树的数量超过设定数目也会返回 ERROR。

21.status RemoveTree(TREES &trees,char TreeName[])

输入：结构体类型变量 trees，字符串类型 TreeName

输出：销毁树的状态

函数思想描述：销毁树函数，在 trees 数组中查找名称为 TreeName 的树，查找成功则将其删除，否则返回 ERROR。

22.int LocateTree(TREES trees,char TreeName[])

输入：结构体类型变量 trees，字符串类型 TreeName

输出：树的位次

函数思想描述：查找二叉树函数，在 trees 数组中查找名称为 ListName 的树，查找成功返回其位序，否则返回 ERROR。

23.status SwitchTrees(TREES trees, char TreeName[], BiTree &T)

输入：结构体类型变量 trees，字符串类型 TreeName，当前操作数 T

输出：二叉树切换状态

函数思想描述：切换二叉树函数，将 trees 中名称为 TreeName 的树赋值给主函数中的树 T，后续可调用其他函数将对此二叉树进行操作。

24.status ShowAllTrees(TREES trees)

输入：结构体类型变量 trees

输出：展示函数状态

函数思想描述：遍历森林函数，依次对每个树进行前序和中序遍历。

2.4 系统测试

测试集如下：

测试集 1：1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null

2.4.1 基本功能函数测试

1..CreateBiTree 测试

测试 1：将使用测试集 1，测试函数能否正确构造二叉树，通过先序遍历和中序遍历的方法来检验正确性；

测试 4 在测试集 1 的基础上测试能否判断已有二叉树。

（注：为了排版方便，测试用例在上文中给出）

测试编号	测试输入	预期结果	实际运行结果
1	1→ 测试集 1	二叉树创建成功！ 先序遍历：1,a 2,b 3,c 4,d 5,e 中序遍历：2,b 1,a 4,d 3,c 5,e	一致

图 2-3 测试 3 运行结果

2.DestroyBiTree 测试

测试 1：在测试集 1 的情况下进行，测试函数能否销毁已存在的二叉树。

测试 2：将测试函数能否销毁不存在的二叉树。

测试编号	测试输入	预期结果	实际运行结果
1	2	二叉树销毁成功！	一致
2	2	二叉树不存在！	一致

```
 Choose your operation : 2
Destroy successfully

 Choose your operation : 2
The BiTree does not exist
```

图 2-4　测试 3 运行结果

3.ClearBiTree 测试

测试 1：在测试集 1 的情况下进行，测试函数能否清空已存在的二叉树。

测试 2：将测试函数能否清空不存在的二叉树。

测试编号	测试输入	预期结果	实际运行结果
1	3	二叉树清空成功！	一致
2	2→3	二叉树不存在！	一致

```
 Choose your operation : 3
Clear successfully

 Choose your operation : 2
Destroy successfully

 Choose your operation : 3
The BiTree does not exist
```

图 2-5　测试 3 运行结果

4.BiTreeEmpty 测试

测试 1：在测试集 1 的基础上，测试函数是否能对非空二叉树进行判空。

测试 2：测试函数是否能对空二叉树进行判空。

测试编号	测试输入	预期结果	实际运行结果
1	4	二叉树不为空！	一致
2	2→4	二叉树为空！	一致

```
 Choose your operation : 4
The BiTree is not empty

 Choose your operation : 2
Destroy successfully

 Choose your operation : 4
The BiTree is empty
```

图 2-6　测试 3 运行结果

5.BiTreeDepth 测试

测试 1：测试函数能否对空二叉树求深度；

测试 2：在测试集 1 的基础上，测试函数能否正确求得二叉树的深度。

测试编号	测试输入	预期结果	实际运行结果
1	5	二叉树为空！	一致
2	1→5	该二叉树的深度为 3！	一致

```
 Choose your operation : 5
The depth of this BiTree is 3

 Choose your operation : 2
Destroy successfully

 Choose your operation : 5
The depth of this BiTree is 0
```

图 2-7　测试 3 运行结果

6.LocateNode 测试

测试 1：将测试函数能否正确找到头结点并输出其值；

测试 2：将测试函数能否正确找到一般结点并输出其值；

测试 3：将测试函数能否正确判断结点关键字不在二叉树中。

测试编号	测试输入	预期结果	实际运行结果
1	6→1	该节点存在！结点信息为：a	一致
2	6→3	该节点存在！结点信息为：c	一致
3	6→6	该节点不存在！	一致

```
 Choose your operation : 6
Enter the key node : 1
The node is 1 a

 Choose your operation : 6
Enter the key node : 3
The node is 3 c

 Choose your operation : 6
Enter the key node : 6
Couldn't find the node with key 6
```

图 2-8　测试 3 运行结果

7.Assign 测试

测试 1：将测试在修改结点值时，函数能否输出正确结果（通过先序遍历结果判断）；

测试 2：将测试在修改结点值，但修改后关键字重复的情况下，函数能否给出判断；

测试 3：将测试所修改结点不在二叉树中的情况下，函数能否给出判断。

测试编号	测试输入	预期结果	实际运行结果
1	7→5→6 f	结点赋值成功！ 先序遍历：1,a 2,b 3,c 4,d 6,f	一致
2	7→1→2 g	结点赋值失败	一致
3	7→7→8 h	结点赋值失败	一致

```
 Choose your operation : 7
Enter the key and the message of value(key and others) :5 6 f
Assign successfully

 Choose your operation : 11
1,a 2,b 3,c 4,d 6,f

 Choose your operation : 7
Enter the key and the message of value(key and others) :1 2 g
The value's key has already exists

 Choose your operation : 7
Enter the key and the message of value(key and others) :7 8 h
Assign failed
```

图 2-9　测试 3 运行结果

8.GetSibling 测试

测试 1,2：将输入一组互为兄弟的结点，测试函数能否输出正确结果；

测试 3：将输入没有兄弟结点的结点，测试函数能否输出正确结果；

测试 4：将输入一个不在二叉树中的结点，测试函数能否给出判断。

测试编号	测试输入	预期结果	实际运行结果
1	8→5	该元素兄弟结点获取成功！ 该结点的兄弟结点关键字为 4 结点信息为：d	一致
2	8→4	该元素兄弟结点获取成功！ 该结点的兄弟结点关键字为 5 结点信息为：e	一致
3	8→1	该结点不存在或不存在兄弟结点!	一致
4	8→6	该结点不存在或不存在兄弟结点!	一致

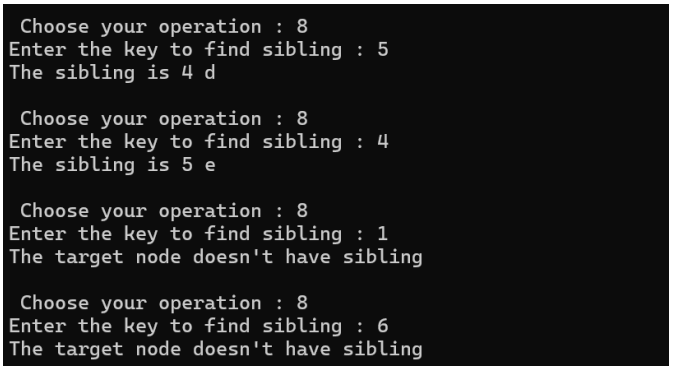

图 2-10　测试运行结果

9.

InsertNode 测试

输入要求：首先输入结点父亲的关键字，再输入插入要求（左孩子 (0)/右孩子 (1)/根节点 (-1)，如插入结点作为根节点，则无需考虑结点父亲的值），最后输入插入结点的值。

测试 1：在测试集 1 的情况下进行，将新插入结点（6 f）作为根节点（-1），测试函数能否输出正确结果，通过前序遍历检验正确性；

测试 2：在测试集 1 的情况下进行，在根节点（1 a）的左孩子插入结点（6 f），测试函数能否输出正确结果，通过前序遍历检验正确性；

测试 3：在测试集 1 的情况下进行，在根节点（1 a）的右孩子插入结点（6 f），测试函数能否输出正确结果，通过前序遍历检验正确性；

测试 4：在测试集 1 的情况下进行，尝试在根节点处插入一个关键字重复的结点（2 b），测试函数能否给出正确判断；

测试 5：在测试集 1 的情况下进行，尝试插入一个结点（6 f），测试当输入的父亲结点（6）不存在于二叉树中时，函数能否给出正确的判断。

测试编号	测试输入	预期结果	实际运行结果
1	9→1→6 f -1	结点插入成功！ 前序遍历：6,f 1,a 2,b 3,c 4,d 5,e	一致
2	9→1→6 f 0	结点插入成功！ 前序遍历：1,a 6,f 2,b 3,c 4,d 5,e	一致
3	9→1→2 b 1	结点插入失败（请检查该关键字是否存在或者插入关键字是否重复）！	一致
3	9→6→6 f 1	结点插入失败（请检查该关键字是否存在或者插入关键字是否重复）！	一致

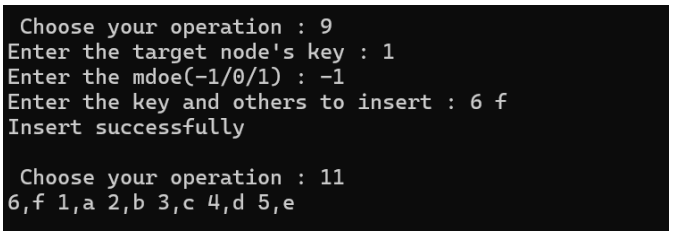

图 2-11　测试运行结果

10.

DeleteNode 测试

测试 1：在测试集 1 的情况下进行，测试函数能否正确删除度为 2 的根结点，通过层序遍历的方式来检验正确性；

测试 2：在测试 1 的基础上进行，测试函数能否正确删除度为 0 的叶子结点，通过层序遍历的方式来检验正确性；

测试 3：在测试 2 的基础上进行，测试函数能否正确删除度为 1 的结点，通过层序遍历的方式来检验正确性。

测试编号	测试输入	预期结果	实际运行结果
1	10→3	结点删除成功！层序遍历：1,a 2,b 4,d 5,e	一致
2	10→2	结点删除成功！层序遍历：1,a 4,d 5,e	一致
3	10→4	结点删除成功！层序遍历：1,a 5,e	一致

```
 Choose your operation : 10
Enter the key node to delete : 3
Delete successfully

 Choose your operation : 14
1,a 2,b 4,d 5,e

 Choose your operation : 10
Enter the key node to delete : 2
Delete successfully

 Choose your operation : 14
1,a 4,d 5,e

 Choose your operation : 10
Enter the key node to delete : 4
Delete successfully

 Choose your operation : 14
1,a 5,e
```

图 2-12　测试运行结果

11.

PreOrderTraverse 测试

测试 1：将测试函数是否能对空树做出判断；

测试 2：在测试集 1 的情况下进行，测试函数能否输出正确结果。

测试编号	测试输入	预期结果	实际运行结果
1	11	二叉树为空！	一致
2	11	先序遍历二叉树的结果： 1,a 2,b 3,c 4,d 5,e	一致

```
 Choose your operation : 11
1,a 2,b 3,c 4,d 5,e

 Choose your operation : 2
Destroy successfully

 Choose your operation : 11
Error
```

图 2-13　测试运行结果

12.

InOrderTraverse 测试

测试 1：将测试函数是否能对空树做出判断；

测试 2：在测试集 1 的情况下进行，测试函数能否输出正确结果。

测试编号	测试输入	预期结果	实际运行结果
1	12	二叉树为空！	一致
2	12	中序遍历二叉树的结果： 2,b 1,a 4,d 3,c 5,e	一致

```
 Choose your operation : 12
2,b 1,a 4,d 3,c 5,e

 Choose your operation : 2
Destroy successfully

 Choose your operation : 12
Error
```

图 2-14　测试运行结果

13.

PostOrderTraverse 测试

测试 1：将测试函数是否能对空树做出判断；

测试 2：在测试集 1 的情况下进行，测试函数能否输出正确结果。

测试编号	测试输入	预期结果	实际运行结果
1	13	二叉树为空！	一致
2	13	后序遍历二叉树的结果： 2,b 4,d 5,e 3,c 1,a	一致

```
 Choose your operation : 13
2,b 4,d 5,e 3,c 1,a

 Choose your operation : 2
Destroy successfully

 Choose your operation : 13
Error
```

图 2-15　测试运行结果

14.

LevelOrderTraverse 测试

测试 1：将测试函数是否能对空树做出判断；

测试 2：在测试集 1 的情况下进行，测试函数能否输出正确结果。

测试编号	测试输入	预期结果	实际运行结果
1	14	二叉树为空！	一致
2	14	层序遍历二叉树的结果： 1,a 2,b 3,c 4,d 5,e	一致

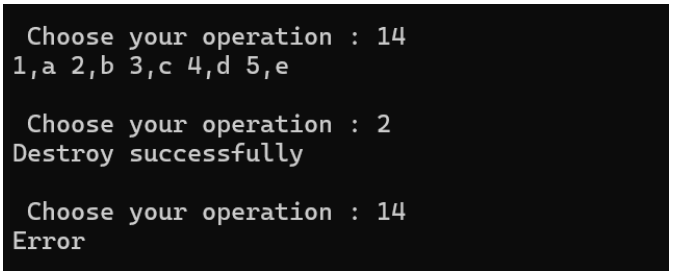

图 2-16　测试运行结果

2.4.2　附加功能测试

15.

MaxPathSum 测试

测试 1：在二叉树为空的情况下进行，测试函数能否给出正确判断；

测试 2：在测试集 1 的情况下进行，测试函数能否正常返回最大路径和。

测试编号	测试输入	预期结果	实际运行结果
1	15	二叉树为空！	一致
2	15	根节点到叶子结点的最大路径和为：9	一致

```
 Choose your operation : 15
The max path sum is 9

 Choose your operation : 2
Destroy successfully

 Choose your operation : 15
The BiTree doesn't exist
```

图 2-17　测试运行结果

16.

LowestCommonAncestor 测试

测试 1：在二叉树为空的情况下进行，测试函数能否给出正确判断；

测试 2：在测试集 1 的情况下进行，两结点均存在测试能否给出根结点；

测试 3：在测试集 1 的情况下进行，测试第一个结点不存在时能否给出正确的判断；

测试 4：在测试集 1 的情况下进行，测试第二个结点不存在时能否给出正确的判断。

测试编号	测试输入	预期结果	实际运行结果
1	16	二叉树为空！	一致
2	16→2 4	两结点最近公共祖先的关键字为：1，结点信息为：a	一致
3	16→6 4	第一个结点不存在！	一致
4	16→2 6	第二个结点不存在！	一致

```
 Choose your operation : 16
Enter the key of e1 and e2 : 2 4
The lowest common ancestor is 1 a

 Choose your operation : 16
Enter the key of e1 and e2 : 6 4
One or two node is not found

 Choose your operation : 16
Enter the key of e1 and e2 : 2 6
One or two node is not found
```

图 2-18　测试运行结果

17.

InvertTree 测试

测试 1：在二叉树为空的情况下进行，测试函数能否给出正确判断；

测试 2：在测试集 1 的情况下，测试函数能否正确反转二叉树，通过层序遍历测试函数实现正确性。

测试编号	测试输入	预期结果	实际运行结果
1	17	二叉树为空！	一致
2	17→14	二叉树翻转成功！ 层序遍历二叉树的结果：1,a 3,c 2,b 5,e 4,d	一致

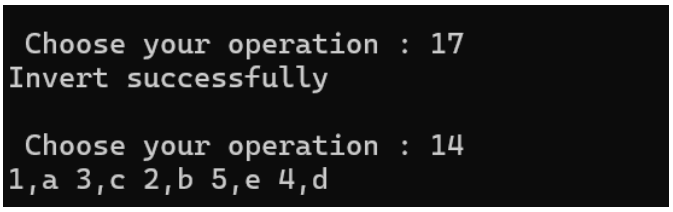

图 2-19　测试运行结果

18.

SaveTree 测试

测试 1：在测试集 1 的情况下进行，测试函数能否正常进行写文件操作；

测试 2：在文件已经有内容时，测试函数是否能够判断文件不能覆盖；

测试 3：在二叉树为空的情况下进行，测试函数能否给出正确判断。

测试编号	测试输入	预期结果	实际运行结果
1	18→1.txt	文件保存成功！	一致
2	18→1.txt	该文件已有内容，不能读入！	一致
3	2→18→1.txt	二叉树不存在! 文件保存失败！	一致

19.

LoadTree 测试

测试 1：在二叉树不存在的情况下进行，测试函数能否正确进行读文件操作，采用遍历二叉树的方式检验正确性。

测试 2：在线性表存在的情况下进行，测试函数能否给出正确判断。

测试编号	测试输入	预期结果	实际运行结果
1	2→19→1.txt	文件录入成功！	一致
2	19→1.txt	二叉树存在! 文件录入失败	一致

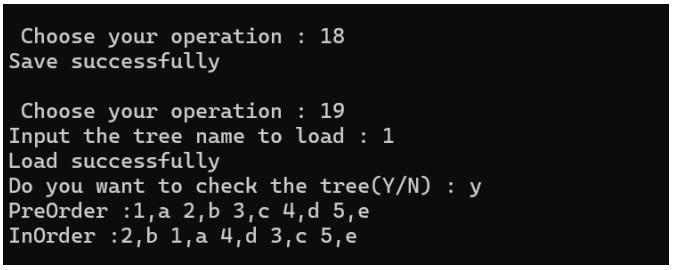

图 2-20　测试运行结果

20.

AddTree 测试

测试 1：将测试函数能否正确添加二叉树；

测试 2：在测试 1 的基础上进行，当二叉树名称重复时，测试函数能否给出正确的判断。

测试 3：测试二叉树能否正确添加，通过遍历森林检验正确性。

测试编号	测试输入	预期结果	实际运行结果
1	20→FirstTree	FirstTree 已成功添加！	一致
2	20→FirstTree	该名称的线性表已经存在！	一致
3	213	FirstTree SecondTree	一致

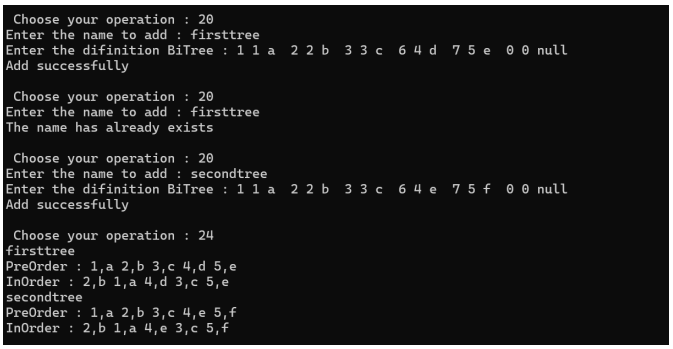

图 2-21　测试运行结果

21.

RemoveTree 测试

测试 1：将测试函数能否正确销毁二叉树，采用遍历森林的方式判断正确性；

测试 2：在测试集 2 的情况下进行，尝试销毁一个不在集合中的二叉树，测试函数能否给出正确判断。

测试编号	测试输入	预期结果	实际运行结果
1	21→FirstTree	FirstTree 已成功销毁！	一致
2	21→ThirdTree	二叉树不存在！	一致

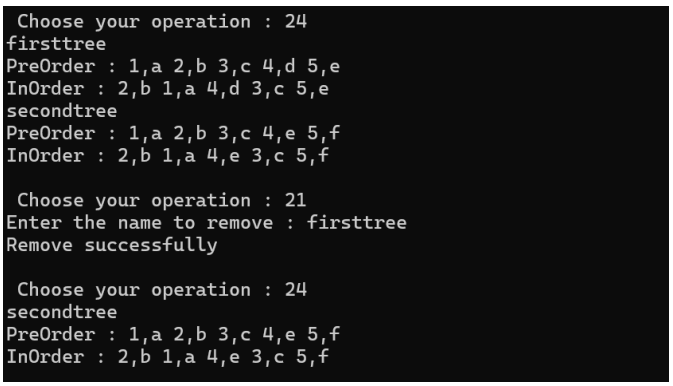

图 2-22　测试运行结果

22.

LocateTree 测试

本实验基于上个测试删除 firsttree 操作

测试 1：将测试函数能否正确定位二叉树；

测试 2：将尝试查找一个不在集合中的二叉树，测试函数能否给出正确判断。

测试编号	测试输入	预期结果	实际运行结果
1	22→SecondTree	该二叉树的逻辑索引为：1	一致
2	22→FirstTree	二叉树查找失败！	一致

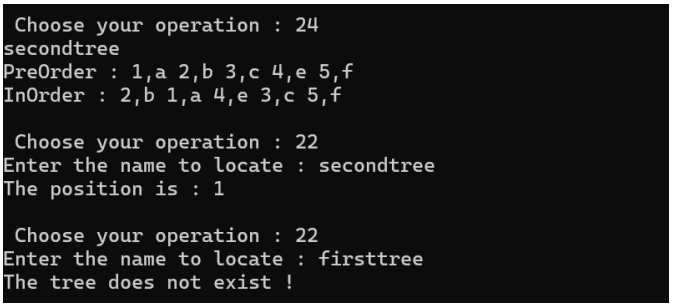

图 2-23　测试运行结果

23.

SwitchTrees 测试

测试 1：测试函数能否正确判断非法的名称；

测试 2：测试函数能否正确地实现选取线性表操作。

测试编号	测试输入	预期结果	实际运行结果
1	24→thirdtree	二叉树切换失败！	一致
2	24→firsttree	二叉树切换成功！	一致

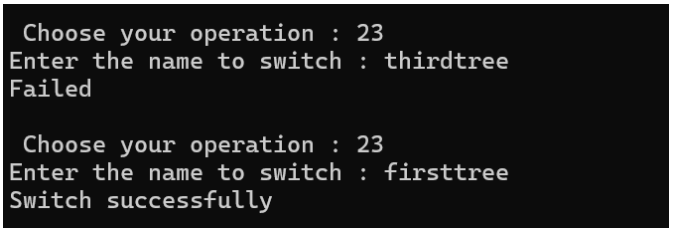

图 2-24　测试运行结果

23.

ShowAllTrees 测试

测试 1：测试函数能否正确遍历各个线性表；

测试编号	测试输入	预期结果	实际运行结果
1	23	FirstList SecondList	一致

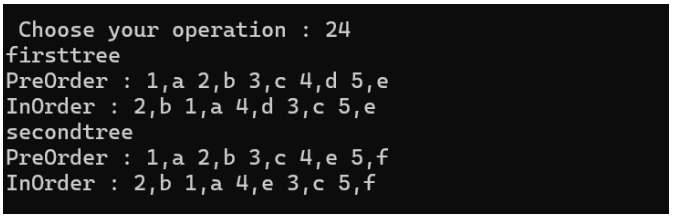

图 2-25　测试运行结果

测试小结

24 个函数基本符合了测试要求，在正常和异常用例的条件下均可以正常运行。需要注意的是，在对某些函数进行测试时，出于篇幅限制，没有测试二叉树不存在的情况，同时对于附加功能函数没有具体给测试后的控制台界面。

2.5 实验小结

这次实验使用链式存储结构实现二叉树，让我更加清楚了二叉树的物理结构、数据结构类型、基本操作及实现。认识到二叉树的存储结构与线性存储结构的不同。这次实验中使用顺序表来管理多树（森林），提高了本次实验的程序的功能性。

在本次实验过程中，多个函数使用递归调用自身的形式编写函数，同时通过设置全局变量的方式解决递归过程中变量的初始化和迭代问题。

除此以外，在用非递归方式实现遍历时，使用了栈的存储结构和相应操作，在实现层序遍历时，使用了队列的存储结构和相应操作。通过不同数据结构的运用，使得问题的解决更加便利。

同时，基于二叉链表的二叉树的实验的难度较前两次都有所提升，极大地提升了我的编程水平。

本次实验使我加深了对二叉树的概念、基本运算的理解，掌握了二叉树的基本运算的实现。熟练了二叉树的逻辑结构和物理结构的关系。在今后的学习过程当中应该更多地从数据结构的角度去分析如何进行数据的存储、读取和处理，以达到更简便地解决实际问题的目的。

3 课程的收获和建议

通过本学期的数据结构实验课，我收获了很多知识，非常感谢老师和助教的帮助，同时包括我自己的努力，让我在数据结构方面收获很多，编程能力和思维能力有了很大的提高。

3.1 基于顺序存储结构的线性表实现

通过实验达到：（1）加深对线性表的概念、基本运算的理解；（2）熟练掌握线性表的逻辑结构与物理结构的关系；（3）物理结构采用顺序表，熟练掌握顺序表基本运算的实现。在实验过程中，对于线性表的本质有了更加深入的思考与理解，提升了思维能力。

3.2 基于链式存储结构的线性表实现

通过实验达到：（1）加深对线性表的概念、基本运算的理解；（2）熟练掌握线性表的逻辑结构与物理结构的关系；（3）物理结构采用单链表，熟练掌握线性表的基本运算的实现。在实验过程中，清晰了顺序存储结构和单链表存储结构之间的区别与联系，对于线性表有了深刻的体会，同时能够灵活应用。

3.3 基于二叉链表的二叉树实现

通过实验达到：（1）加深对二叉树的概念、基本运算的理解；（2）熟练掌握二叉树的逻辑结构与物理结构的关系；（3）以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。在实验过程中，对于二叉树特殊的存储结构能够很好地应用，同时对于递归操作有了更加深刻的理解。

3.4 基于邻接表的图实现

通过实验达到：（1）加深对图的概念、基本运算的理解；（2）熟练掌握图的逻辑结构与物理结构的关系；（3）以邻接表作为物理结构，熟练掌握图基本运算的实现。在实验过程中，学会使用图的邻接表创建无向图，同时由于图的复杂性，对于心理素质的锻炼起到了很大的效果。

参考文献

[1] 严蔚敏等. 数据结构（C 语言版）. 清华大学出版社

[2] 严蔚敏等. 数据结构题集（C 语言版）. 清华大学出版社

4 附录 A 基于顺序存储结构线性表实现的源程序

```
#include <malloc.h>
#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
typedef int status;
typedef int ElemType;
#define LIST_INIT_SIZE 100
#define LISTINCREMENT 10
typedef struct {     //顺序表（顺序存储结构）结点的定义
    ElemType *elem;
    int length;
    int listsize;
} SqList;
typedef struct {     //线性表（顺序存储结构）结点的定义
    struct {
        char name[30];
        SqList L;
    } elem[10];
    int length;
    int listsize;
} LISTS;
```

```
28  status InitList(SqList &L) {// 初始化顺序表，分配空间
29      if (L.elem != NULL)
30          return INFEASIBLE;
31      L.elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(
            ElemType));
32      L.length = 0;
33      L.listsize = LIST_INIT_SIZE;
34      return OK;
35  }
36  status DestroyList(SqList &L) {// 销毁顺序表，释放内存
37      if (L.elem == NULL)
38          return INFEASIBLE;
39      else {
40          free(L.elem);
41          L.elem = NULL;
42          L.length = 0;
43          L.listsize = 0;
44          return OK;
45      }
46  }
47  status ClearList(SqList &L) {// 清空顺序表内容但不释放空间
48      if (L.elem == NULL)
49          return INFEASIBLE;
50      L.length = 0;
51      return OK;
52  }
53  status ListEmpty(SqList L) {// 判断顺序表是否为空
54      if (L.elem == NULL)
55          return INFEASIBLE;
56      if (L.length == 0)
57          return TRUE;
58      else
59          return FALSE;
60  }
61  status ListLength(SqList L) {// 获取顺序表长度
```

```
62      if (L.elem == NULL)
63          return INFEASIBLE;
64      return L.length;
65  }
66  status GetElem(SqList L, int i, ElemType &e) {// 获取顺序表第i
        个元素
67      if (L.elem == NULL)
68          return INFEASIBLE;
69      if (i >= 0 && i <= L.length) {
70          if (i == 0)
71              return ERROR;
72          e = L.elem[i - 1];
73          return OK;
74      } else
75          return ERROR;
76  }
77  int LocateElem(SqList L, ElemType e) {// 查找元素e在线性表中的
        位置
78      if (L.elem == NULL)
79          return INFEASIBLE;
80      for (int i = 0; i < L.length; i++)
81          if (L.elem[i] == e)
82              return i + 1;
83      return ERROR;
84  }
85  status PriorElem(SqList L, ElemType e, ElemType &pre) {// 获取
        元素e的前驱
86      if (L.elem == NULL)
87          return INFEASIBLE;
88      for (int i = 0; i < L.length; i++) {
89          if (L.elem[i] == e && i > 0) {
90              pre = L.elem[i - 1];
91              return OK;
92          }
93      }
```

```
94      return ERROR;
95  }
96  status NextElem(SqList L, ElemType e, ElemType &next) {// 获取
        元素e的后继
97      if (L.elem == NULL)
98          return INFEASIBLE;
99      for (int i = 0; i < L.length; i++) {
100         if (L.elem[i] == e && i < L.length - 1) {
101             next = L.elem[i + 1];
102             return OK;
103         }
104     }
105     return ERROR;
106 }
107 status ListInsert(SqList &L, int i, ElemType e) {// 在第i个位
        置插入元素e
108     if (L.elem == NULL)
109         return INFEASIBLE;
110     if (i < 1 || i > L.length + 1)
111         return ERROR;
112     if (L.length >= L.listsize) {
113         ElemType *newElem = (ElemType *)realloc(L.elem, (L.
                listsize + LISTINCREMENT) * sizeof(ElemType));
114         if (!newElem)
115             return OVERFLOW;
116         L.elem = newElem;
117         L.listsize += LISTINCREMENT;
118     }
119     for (int k = L.length - 1; k >= i - 1; k--)
120         L.elem[k + 1] = L.elem[k];
121     L.elem[i - 1] = e;
122     L.length++;
123     return OK;
124 }
125 status ListDelete(SqList &L, int i, ElemType &e) {// 删除第i个
```

```
        元素，并用e返回其值
126     if (L.elem == NULL)
127         return INFEASIBLE;
128     if (i < 1 || i > L.length)
129         return ERROR;
130     e = L.elem[i - 1];
131     for (int k = i - 1; k < L.length - 1; k++)
132         L.elem[k] = L.elem[k + 1];
133     L.length--;
134     return OK;
135 }
136 status ListTrabverse(SqList L) {// 遍历顺序表并输出所有元素
137     int i;
138     for (i = 0; i < L.length; i++)
139         printf("%d ", L.elem[i]);
140     printf("\n");
141     return OK;
142 }
143 ElemType MaxSubArray(SqList L) {// 求最大连续子数组和
144     if (L.elem == NULL)
145         return ERROR;
146     if (L.length) {
147         ElemType maxSum = L.elem[0];
148         ElemType sum = L.elem[0];
149         for (int i = 1; i < L.length; i++) {
150             sum = (sum + L.elem[i] > L.elem[i]) ? sum + L.elem
                    [i] : L.elem[i];
151             if (sum > maxSum)
152                 maxSum = sum;
153         }
154         return maxSum;
155     }
156 }
157 int SubArrayNum(SqList L, ElemType k) {// 求和为k的子数组个数
158     if (L.elem == NULL)
```

```
159             return ERROR;
160         int count = 0;
161         for (int i = 0; i < L.length; i++) {
162             int sum = 0;
163             for (int j = i; j < L.length; j++) {
164                 sum += L.elem[j];
165                 if (sum == k)
166                     count++;
167             }
168         }
169         return count;
170     }
171     void sortList(SqList L) {// 冒泡排序顺序表
172         if (L.elem == NULL)
173             printf("No number to sort !\n");
174         for (int i = 0; i < L.length; i++)
175             for (int j = 0; j < L.length - i - 1; j++)
176                 if (L.elem[j] > L.elem[j + 1]) {
177                     int temp = L.elem[j + 1];
178                     L.elem[j + 1] = L.elem[j];
179                     L.elem[j] = temp;
180                 }
181     }
182     status SaveList(SqList L, char FileName[]) {// 保存顺序表到文
        件
183         if (L.elem == NULL)
184             return INFEASIBLE;
185         FILE *fp;
186         fp = fopen(FileName, "r");
187         if (fp) {
188             fclose(fp);
189             fp = fopen(FileName, "w");
190         } else {
191             fp = fopen(FileName, "w");
192         }
```

```
193     if (!fp)
194         return ERROR;
195     for (int i = 0; i < L.length; i++)
196         fprintf(fp, "%d ", L.elem[i]);
197     fclose(fp);
198     return OK;
199 }
200 status LoadList(SqList &L, char FileName[]) {// 从文件加载顺序
        表
201     if (L.elem != NULL)
202         return INFEASIBLE;
203     FILE *fp;
204     fp = fopen(FileName, "r");
205     if (!fp)
206         return ERROR;
207     L.elem = (ElemType *)malloc(sizeof(ElemType) *
           LIST_INIT_SIZE);
208     if (!L.elem) {
209         fclose(fp);
210         return OVERFLOW;
211     }
212     L.length = 0;
213     L.listsize = LIST_INIT_SIZE;
214     ElemType e;
215     while (fscanf(fp, "%d", &e) == 1) {
216         L.elem[L.length++] = e;
217     }
218
219     fclose(fp);
220     return OK;
221 }
222 status AddList(LISTS &Lists, char ListName[]) {// 向LISTS结构
        中添加新顺序表
223     Lists.length++;
224     strcpy(Lists.elem[Lists.length - 1].name, ListName);
```

```
225     SqList l;
226     l.elem = NULL;
227     InitList(l);
228     Lists.elem[Lists.length - 1].L = l;
229     return OK;
230 }
231 status RemoveList(LISTS &Lists, char ListName[]) {// 参数:
    Lists为顺序表集合, ListName为要移除的顺序表名
232     int k = 0;
233     int isfound = 0;
234     for (int i = 0; i < Lists.length; i++) {
235         if (strcmp(Lists.elem[i].name, ListName) == 0) {// 查
              找要删除的顺序表
236             DestroyList(Lists.elem[i].L); // 释放该顺序表内存
237             k = i;
238             isfound = 1;
239         }
240     }
241     if (isfound) {
242         for (int j = k; j < Lists.length - 1; j++)
243             Lists.elem[j] = Lists.elem[j + 1];// 如果找到则后
                续顺序表前移
244         Lists.length--;
245         return OK;
246     }
247     return ERROR;
248 }
249 int LocateList(LISTS Lists, char ListName[]) {// 查找LISTS结构
    中顺序表的位置
250     for (int i = 0; i < Lists.length; i++) {
251         if (strcmp(Lists.elem[i].name, ListName) == 0)
252             return i + 1;// 返回顺序表在集合中的下标+1, 未找到
                返回ERROR
253     }
254     return ERROR;
```

```
255 }
256 void ShowAllLists(LISTS lists) {// 显示所有顺序表及其内容
257     for (int i = 0; i < lists.length; i++) {
258         printf("%s ", lists.elem[i].name);
259         for (int j = 0; j < lists.elem[i].L.length; j++)
260             printf("%d ", lists.elem[i].L.elem[j]);// 依次输出
                每个顺序表的名字和所有元素
261         printf("\n");
262     }
263 }
264 #include "def.h"
265 #include "system1Func.h"
266 int main() {
267     SqList L;
268     L.elem = NULL;
269     LISTS lists;
270     lists.length = 0;
271     lists.listsize = LIST_INIT_SIZE;
272     int op = 1;
273     printf("      Menu for Linear Table On Sequence Structure\
        n"
274         "-------------------------------------------------\
            n"
275         "1. InitList        12.ListTrabverse\n"
276         "2. DestroyList     13.MaxSubArray\n"
277         "3. ClearList       14.SubArrayNum\n"
278         "4. ListEmpty       15.sortList\n"
279         "5. ListLength      16.SaveList\n"
280         "6. GetElem         17.LoadList\n"
281         "7. LocateElem      18.AddList\n"
282         "8. PriorElem       19.RemoveList\n"
283         "9. NextElem        20.LocateList\n"
284         "10.ListInsert      21.ShowAllLists\n"
285         "11.ListDelete      0. Exit\n"
286         "-------------------------------------------------\
```

```
            n");
287     while (op) {
288         printf("\n Choose you operation : ");
289         scanf("%d", &op);
290         switch (op) {
291         case 1:
292             if (InitList(L) == OK)
293                 printf("Linear table was successfully created
                        !\n");
294             else
295                 printf("Linear table creation failed !\n");
296             getchar();
297             break;
298         case 2:
299             if (DestroyList(L) == OK)
300                 printf("Linear table was successfully
                        destroyed !\n");
301             else
302                 printf("Linear table destruction failed !\n");
303             getchar();
304             break;
305         case 3:
306             if (ClearList(L) == OK)
307                 printf("Linear table was successfully cleared
                        !\n");
308             else
309                 printf("Linear table clear failed !\n");
310             getchar();
311             break;
312         case 4:
313             if (ListEmpty(L) == TRUE)
314                 printf("The linear table is empty !\n");
315             else
316                 printf("The linear table is not empty !\n");
317             getchar();
```

```
318             break;
319         case 5:
320             if (ListLength(L) != INFEASIBLE)
321                 printf("The length of this linear table is %d
                        !\n", ListLength(L));
322             getchar();
323             break;
324         case 6:
325             printf("Please choose the position of your number
                    (1 to length) : ");
326             int pos;
327             ElemType res;
328             scanf("%d", &pos);
329             if (GetElem(L, pos, res) == OK)
330                 printf("The number is %d !\n", res);
331             else
332                 printf("The position is illegal !\n");
333             getchar();
334             break;
335         case 7:
336             printf("Please enter the number you want to locate
                    : ");
337             ElemType loc;
338             scanf("%d", &loc);
339             if (LocateElem(L, loc))
340                 printf("The number is located on the position
                        of %d !\n", LocateElem(L, loc));
341             else
342                 printf("The number does not exist !\n");
343             getchar();
344             break;
345         case 8:
346             printf("Please enter your number : ");
347             ElemType num, pre;
348             scanf("%d", &num);
```

```
349             if (PriorElem(L, num, pre) == OK)
350                 printf("The prior number is %d !\n", pre);
351             else
352                 printf("This number does not have a prior
                        number in the table !\n");
353             getchar();
354             break;
355         case 9:
356             printf("Please enter your number : ");
357             ElemType numD, next;
358             scanf("%d", &numD);
359             if (NextElem(L, numD, next) == OK)
360                 printf("The next number is %d !\n", next);
361             else
362                 printf("This number does not have a next
                        number in the table !\n");
363             getchar();
364             break;
365         case 10:
366             printf("Please choose your position and number to
                    insert : ");
367             int pos1;
368             ElemType num1;
369             scanf("%d%d", &pos1, &num1);
370             if (ListInsert(L, pos1, num1) == OK)
371                 printf("The number is successfully inserted !\
                        n");
372             else
373                 printf("ListInsert failed !\n");
374             getchar();
375             break;
376         case 11:
377             printf("Please choose the position to delete : ");
378             int pos2;
379             ElemType num2;
```

```
380             scanf("%d", &pos2);
381             if (ListDelete(L, pos2, num2) == OK)
382                 printf("The number is successfully deleted !\n
                        ");
383             else
384                 printf("Deletion failed !\n");
385             getchar();
386             break;
387         case 12:
388             if (!ListTrabverse(L))
389                 printf("The list is empty \n");
390             getchar();
391             break;
392         case 13:
393             if (L.elem == NULL)
394                 printf("No element to sum !\n");
395             else
396                 printf("MaxSum is %d \n", MaxSubArray(L));
397             getchar();
398             break;
399         case 14:
400             printf("Enter the sum you want to find : ");
401             ElemType k;
402             scanf("%d", &k);
403             if (L.elem == NULL)
404                 printf("No element to sum !\n");
405             else
406                 printf("The number of this arraySum is : %d !\
                        n", SubArrayNum(L, k));
407             getchar();
408             break;
409         case 15:
410             if (L.elem == NULL)
411                 printf("No elem to sort!\n");
412             else {
```

```
413                 sortList(L);
414                 printf("List sorted !\n");
415             }
416             getchar();
417             break;
418         case 16: {
419             char filename[] = "c:\\Users\\12841\\Desktop\\none
                    \\2025springDS\\experiments\\ex1\\list1.txt";
420             if (SaveList(L, filename) == OK)
421                 printf("Saved successfully !\n");
422             else
423                 printf("Saving failed\n");
424             getchar();
425             break;
426         }
427         case 17: {
428             char filename[] = "c:\\Users\\12841\\Desktop\\none
                    \\2025springDS\\experiments\\ex1\\list1.txt";
429             SqList l;
430             l.elem = NULL;
431             if (l.elem != NULL)
432                 printf("List is not empty !\n");
433             if (LoadList(l, filename) == OK)
434                 printf("Load successfully !\n");
435             else
436                 printf("Loading failed !\n");
437             printf("Do you want to check the list (Y/N) : ");
438             char opt;
439             getchar();
440             scanf("%c", &opt);
441             if (opt == 'Y' || opt == 'y')
442                 ListTrabverse(l);
443             else
444                 break;
445             getchar();
```

```
446             break;
447         }
448         case 18: {
449             printf("Please enter your listname : ");
450             char name[100];
451             scanf("%s", name);
452             if (AddList(lists , name) == OK) {
453                 printf("Please enter your elements amount and
                        members : ");
454                 int len;
455                 scanf("%d", &len);
456                 lists.elem[lists.length - 1].L.length = len;
457                 for (int i = 0; i < len; i++)
458                     scanf("%d", &lists.elem[lists.length - 1].
                            L.elem[i]);
459                 printf("List added !\n");
460             }
461             else
462                 printf("Failed !\n");
463             getchar();
464             break;
465         }
466         case 19:
467             printf("Enter the listname to remove : ");
468             char name2[100];
469             scanf("%s", name2);
470             if (RemoveList(lists , name2) == OK)
471                 printf("List removed !\n");
472             else
473                 printf("Cant remove the list !\n");
474             getchar();
475             break;
476         case 20:
477             printf("Enter the name to locate : ");
478             char name3[100];
```

```
479                 scanf("%s", name3);
480                 if (LocateList(lists, name3))
481                     printf("The position is : %d \n", LocateList(
                            lists, name3));
482                 else
483                     printf("The list does not exist !\n");
484                 getchar();
485                 break;
486             case 21:
487                 ShowAllLists(lists);
488                 getchar();
489                 break;
490             case 0:
491                 break;
492             }
493             // printf("Press enter to continue ...");
494             // getchar();
495         }
496         printf("Welcome to the system next time !\n");
497         return 0;
498     }
```

5 附录 B 基于链式存储结构线性表实现的源程序

```
#include <malloc.h>
#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
typedef int status;
typedef int ElemType; // 数据元素类型定义
#define LIST_INIT_SIZE 100
#define LISTINCREMENT 10
typedef int ElemType;
typedef struct LNode { // 单链表（链式结构）结点的定义
    ElemType data;
    struct LNode *next;
} LNode, *LinkList;
typedef struct {
    struct {
        char name[30];
        LinkList Li;
    } elem[10];
    int length;
    int listsize;
} LISTS;
status InitList(LinkList &L) {
    if (L != NULL)
        return INFEASIBLE;                    // 已初始化则返回不可
            行
    L = (LinkList)malloc(sizeof(LNode)); // 分配头结点
    L->next = NULL;                          // 初始化为空链表
```

```
33      return OK;
34  }
35  status DestroyList(LinkList &L) {
36      if (L == NULL)
37          return INFEASIBLE; // 未初始化
38      LinkList p = L, q;
39      while (p) {
40          q = p->next;
41          free(p); // 释放每个结点
42          p = q;
43      }
44      L = NULL; // 头指针置空
45      return OK;
46  }
47  status ClearList(LinkList &L) {
48      if (L == NULL)
49          return INFEASIBLE;
50      LinkList p = L->next, q;
51      while (p) {
52          q = p->next;
53          p->data = 0;
54          p->next = NULL;
55          p = q;
56      }
57      L->next = NULL;
58      return OK;
59  }
60  status ListEmpty(LinkList L) {
61      if (L == NULL)
62          return INFEASIBLE;
63      if (L != NULL && L->next == NULL)
64          return OK;
65      else if (L->next != NULL)
66          return ERROR;
67  }
```

```
68  int ListLength(LinkList L) {
69      if (L == NULL)
70          return INFEASIBLE;
71      LinkList p = L->next;
72      int len = 0;
73      while (p) {
74          len++;
75          p = p->next;
76      }
77      return len;
78  }
79  status GetElem(LinkList L, int i, ElemType &e) {
80      if (L == NULL)
81          return INFEASIBLE;
82      if (i < 1)
83          return ERROR;
84      LinkList p = L->next;
85      int len = 0;
86      while (p) {
87          len++;
88          i--;
89          if (i == 0) {
90              e = p->data;
91              return OK;
92          }
93          p = p->next;
94      }
95      if (i != 0)
96          return ERROR;
97  }
98  status LocateElem(LinkList L, ElemType e, char mode) {
99      if (L == NULL)
100         return INFEASIBLE;
101     int i = 1;
102     LinkList p = L->next;
```

```
103     while (p) {
104         if (mode == '>') {
105             if (p->data > e)
106                 return i;
107         } else if (mode == '<') {
108             if (p->data < e)
109                 return i;
110         } else if (mode == '=') {
111             if (p->data == e)
112                 return i;
113         } else {
114             printf("The mode is illegal \n");
115             break;
116         }
117         i++;
118         p = p->next;
119     }
120     return ERROR;
121 }
122 status PriorElem(LinkList L, ElemType e, ElemType &pre) {
123     if (L == NULL)
124         return INFEASIBLE;
125     LinkList p = L->next, q = L;
126     while (p) {
127         if (e == p->data) {
128             if (p == L->next)
129                 return ERROR;
130             pre = q->data;
131             return OK;
132         }
133         q = p;
134         p = p->next;
135     }
136     return ERROR;
137 }
```

```
138 status NextElem(LinkList L, ElemType e, ElemType &next) {
139     if (L == NULL)
140         return INFEASIBLE;
141     LinkList p = L->next;
142     while (p) {
143         if (p->next == NULL)
144             return ERROR;
145         if (p->data == e) {
146             next = p->next->data;
147             return OK;
148         }
149         p = p->next;
150     }
151     return ERROR;
152 }
153 status ListInsert(LinkList &L, int i, ElemType e) {
154     if (L == NULL)
155         return INFEASIBLE;
156     LinkList q = L;
157     while (q) {
158         i--;
159         if (!i) {
160             LinkList s = (LinkList)malloc(sizeof(LNode));
161             s->data = e;
162             s->next = q->next;
163             q->next = s;
164             return OK;
165         }
166         q = q->next;
167     }
168     return ERROR;
169 }
170 status ListDelete(LinkList &L, int i, ElemType &e) {
171     if (L == NULL)
172         return INFEASIBLE;
```

```
173     LinkList p = L, q = L->next;
174     while (q) {
175         i--;
176         if (!i) {
177             p->next = q->next;
178             e = q->data;
179             free(q);
180             return OK;
181         }
182         p = q;
183         q = q->next;
184     }
185     return ERROR;
186 }
187 status ListTraverse(LinkList L) {
188     if (L == NULL)
189         return INFEASIBLE;
190     LinkList p = L->next;
191     while (p) {
192         printf("%d ", p->data);
193         p = p->next;
194     }
195     printf("\n");
196     return OK;
197 }
198 status ReverseList(LinkList L) {
199     if (L == NULL)
200         return INFEASIBLE; // 链表未初始化
201     if (L->next == NULL)
202         return OK; // 空链表无需反转
203     LinkList p = L->next, q;
204     L->next = NULL; // 断开原链表
205     while (p) {
206         q = p->next;        // 暂存下一个结点
207         p->next = L->next; // 头插法反转
```

```
208             L->next = p;          // 新头结点
209             p = q;                // 继续下一个
210         }
211         return OK;
212     }
213     status RemoveNthFromEnd(LinkList L, int n) {
214         if (L == NULL || L->next == NULL)
215             return INFEASIBLE;    // 空链表
216         int len = ListLength(L); // 获取长度
217         ElemType res;
218         if (n > len || n < 1)
219             return ERROR; // n非法
220         if (ListDelete(L, len + 1 - n, res) == OK)
221             return OK; // 删除倒数第n个
222     }
223     void sortList(LinkList L) {
224         LinkList head = L->next, p = head->next, q;
225         head->next = NULL; // 断开原链表
226         while (p) {
227             q = p->next; // 暂存下一个
228             LinkList pre = L, cur = L->next;
229             while (cur && cur->data < p->data) {
230                 pre = cur;
231                 cur = cur->next;
232             }
233             p->next = cur; // 插入到合适位置
234             pre->next = p;
235             p = q;
236         }
237     }
238     status SaveList(LinkList L, char FileName[]) {
239         if (L == NULL)
240             return INFEASIBLE; // 未初始化
241         FILE *fp = fopen(FileName, "r");
242         if (fp) {
```

```
243             fclose(fp);
244             fp = fopen(FileName, "w");
245         } else {
246             fp = fopen(FileName, "w");
247         }
248         if (!fp)
249             return ERROR; // 打开失败
250         LinkList p = L->next;
251         while (p) {
252             fprintf(fp, "%d ", p->data); // 写入数据
253             p = p->next;
254         }
255         fclose(fp);
256         return OK;
257     }
258     status LoadList(LinkList &L, char FileName[]) {
259         if (L != NULL)
260             return INFEASIBLE; // 已初始化
261         FILE *fp = fopen(FileName, "r");
262         if (!fp)
263             return ERROR; // 文件不存在
264         LinkList tail;
265         L = (LinkList)malloc(sizeof(LinkList)); // 分配头结点
266         tail = L;
267         ElemType e;
268         while (fscanf(fp, "%d", &e) == 1) {
269             LinkList node = (LinkList)malloc(sizeof(LinkList)); //
                    新结点
270             tail->next = node;
271             node->data = e;
272             tail = tail->next;
273             tail->next = NULL;
274         }
275         fclose(fp);
276         return OK;
```

```
277 }
278 status AddList(LISTS &Lists, char ListName[]) {
279     for (int i = 0; i < Lists.length; i++)
280         if (strcmp(Lists.elem[i].name, ListName) == 0)
281             return ERROR; // 名称重复
282     Lists.length++;
283     strcpy(Lists.elem[Lists.length - 1].name, ListName); // 赋
        名
284     LinkList l;
285     l = NULL;
286     InitList(l); // 初始化链表
287     Lists.elem[Lists.length - 1].Li = l;
288     return OK;
289 }
290 status RemoveList(LISTS &Lists, char ListName[]) {
291     int k = 0;
292     int isfound = 0;
293     for (int i = 0; i < Lists.length; i++) {
294         if (strcmp(Lists.elem[i].name, ListName) == 0) {
295             DestroyList(Lists.elem[i].Li); // 释放链表
296             k = i;
297             isfound = 1;
298         }
299     }
300     if (isfound) {
301         for (int j = k; j < Lists.length - 1; j++)
302             Lists.elem[j] = Lists.elem[j + 1]; // 前移
303         Lists.length--;
304         return OK;
305     }
306     return ERROR;
307 }
308 int LocateList(LISTS Lists, char ListName[]) {
309     for (int i = 0; i < Lists.length; i++) {
310         if (strcmp(Lists.elem[i].name, ListName) == 0)
```

```
311                 return i + 1; // 返回下标+1
312         }
313         return ERROR;
314     }
315     status SwitchList(LISTS Lists, char ListName[], LinkList &L) {
316         for (int i = 0; i < Lists.length; i++) {
317             if (strcmp(ListName, Lists.elem[i].name) == 0) {
318                 L = Lists.elem[i].Li; // 切换当前链表
319                 return OK;
320             }
321         }
322         return ERROR;
323     }
324     void ShowAllLists(LISTS lists) {
325         for (int i = 0; i < lists.length; i++) {
326             printf("%s ", lists.elem[i].name); // 输出名字
327             ListTraverse(lists.elem[i].Li);     // 输出内容
328             printf("\n");
329         }
330     }
331     #include "def2.h"
332     #include "system2Func.h"
333     int main() {
334         LinkList L;                          // 当前操作的链表指针
335         L = NULL;                            // 初始化为空
336         LISTS lists;                         // 链表集合
337         lists.length = 0;                    // 集合长度初始化
338         lists.listsize = LIST_INIT_SIZE;     // 集合容量初始化
339         int op = 1;                          // 操作选项
340         printf("Menu for Linear List on Linked Structure\n"
341                 "-------------------------------------------------\
                    n"
342                 "1. InitList        12.ListTraverse\n"
343                 "2. DestroyList     13.ReverseList\n"
344                 "3. ClearList       14.RemoveNthFromEnd\n"
```

```
345             "4. ListEmpty       15. SortList\n"
346             "5. ListLength      16. SaveList\n"
347             "6. GetElem         17. LoadList\n"
348             "7. LocateElem      18. AddList\n"
349             "8. PriorElem       19. RemoveList\n"
350             "9. NextElem        20. LocateList\n"
351             "10. ListInsert     21. SwitchList\n"
352             "11. ListDelete     22. ShowAllLists\n"
353             "0. Exit\n"
354             "-------------------------------------------------\
                n");
355     while (op) {
356         printf("\n Choose you operation : ");
357         scanf("%d", &op); // 读取操作选项
358         switch (op) {
359         case 1:
360             if (InitList(L) == OK)
361                 printf("Linear list was successfully created \
                    n"); // 初始化链表
362             else
363                 printf("Linear list creation failed \n");
364             getchar();
365             break;
366         case 2:
367             if (DestroyList(L) == OK)
368                 printf("Linear list was successfully destroyed
                        \n"); // 销毁链表
369             else
370                 printf("Linear list destruction failed \n");
371             getchar();
372             break;
373         case 3:
374             if (ClearList(L) == OK)
375                 printf("Linear list was successfully cleared \
                    n"); // 清空链表
```

```
376             else
377                 printf("Linear list clear failed \n");
378             getchar();
379             break;
380         case 4:
381             if (ListEmpty(L) == TRUE)
382                 printf("The linear list is empty \n"); // 判断
                        链表是否为空
383             else
384                 printf("The linear list is not empty \n");
385             getchar();
386             break;
387         case 5:
388             if (ListLength(L) != INFEASIBLE)
389                 printf("The length of this linear list is %d \
                        n", ListLength(L)); // 输出链表长度
390             getchar();
391             break;
392         case 6: {
393             printf("Please choose the position of your number
                    (1 to length) : ");
394             int pos;
395             ElemType res;
396             scanf("%d", &pos);
397             if (GetElem(L, pos, res) == OK)
398                 printf("The number is %d \n", res); // 获取指
                        定位置元素
399             else
400                 printf("The position is illegal \n");
401             getchar();
402             break;
403         }
404         case 7: {
405             printf("Please enter the number you want to locate
                    in comparison : ");
```

```
406             ElemType loc;
407             scanf("%d", &loc);
408             getchar();
409             printf("Choose one comparison mode ( > = < ) :");
410             char mode;
411             scanf("%c", &mode);
412             if (LocateElem(L, loc, mode))
413                 printf("The first number %c %d is located on
                        the position of %d \n", mode, loc,
414                             LocateElem(L, loc, mode)); // 查找第一
                                个满足条件的位置
415             else
416                 printf("The number does not exist \n");
417             getchar();
418             break;
419         }
420         case 8: {
421             printf("Please enter your number : ");
422             ElemType num, pre;
423             scanf("%d", &num);
424             if (PriorElem(L, num, pre) == OK)
425                 printf("The prior number is %d \n", pre); //
                        查找前驱
426             else
427                 printf("This number does not have a prior
                        number in the list \n");
428             getchar();
429             break;
430         }
431         case 9: {
432             printf("Please enter your number : ");
433             ElemType numD, next;
434             scanf("%d", &numD);
435             if (NextElem(L, numD, next) == OK)
436                 printf("The next number is %d \n", next); //
```

```
                    查找后继
437             else
438                 printf("This number does not have a next
                        number in the list \n");
439             getchar();
440             break;
441         }
442         case 10: {
443             printf("Please choose your position and number to
                    insert : ");
444             int pos1;
445             ElemType num1;
446             scanf("%d%d", &pos1, &num1);
447             if (ListInsert(L, pos1, num1) == OK)
448                 printf("The number is successfully inserted \n
                        "); // 插入元素
449             else
450                 printf("ListInsert failed \n");
451             getchar();
452             break;
453         }
454         case 11:
455             printf("Please choose the position to delete : ");
456             int pos2;
457             ElemType num2;
458             scanf("%d", &pos2);
459             if (ListDelete(L, pos2, num2) == OK)
460                 printf("The number is successfully deleted \n"
                        ); // 删除元素
461             else
462                 printf("Deletion failed \n");
463             getchar();
464             break;
465         case 12:
466             if (!ListTraverse(L))
```

```
467                 printf("The list is empty \n"); // 遍历链表
468             getchar();
469             break;
470         case 13:
471             if (ReverseList(L) == OK)
472                 printf("The list is reversed \n"); // 反转链表
473             else
474                 printf("ERROR \n");
475             getchar();
476             break;
477         case 14: {
478             int n;
479             printf("Enter the rank from the end : ");
480             scanf("%d", &n);
481             status resFromEnd = RemoveNthFromEnd(L, n);
482             if (resFromEnd == OK)
483                 printf("Successfully removed \n"); // 删除倒数
                        第n个
484             else if (resFromEnd == INFEASIBLE)
485                 printf("The list doesn't exist \n");
486             else
487                 printf("The rank is illegal \n");
488             getchar();
489             break;
490         }
491         case 15:
492             if (L == NULL)
493                 printf("No elem to sort\n");
494             else if (L->next == NULL)
495                 printf("Can't sort empty list \n");
496             else {
497                 sortList(L);
498                 printf("List sorted \n"); // 排序链表
499             }
500             getchar();
```

```
501             break;
502         case 16: {
503             char filename[] = "c:\\Users\\12841\\Desktop\\none
                    \\2025springDS\\experiments\\ex2\\list2.txt";
504             if (SaveList(L, filename) == OK)
505                 printf("Saved successfully \n"); // 保存链表到
                        文件
506             else
507                 printf("Saving failed\n");
508             getchar();
509             break;
510         }
511         case 17: {
512             char filename[] = "c:\\Users\\12841\\Desktop\\none
                    \\2025springDS\\experiments\\ex2\\list2.txt";
513             LinkList l = NULL;
514             if (l != NULL)
515                 printf("List is not empty \n");
516             if (LoadList(l, filename) == OK)
517                 printf("Load successfully \n"); // 从文件加载
                        链表
518             else
519                 printf("Loading failed \n");
520             printf("Do you want to check the list (Y/N) : ");
521             char opt;
522             getchar();
523             scanf("%c", &opt);
524             if (opt == 'Y' || opt == 'y')
525                 ListTraverse(l); // 加载后遍历
526             else
527                 break;
528             getchar();
529             break;
530         }
531         case 18: {
```

```
532             printf("Please enter your listname : ");
533             char name[100];
534             scanf("%s", name);
535             if (AddList(lists, name) == OK) {
536                 printf("List added \n");                    // 添加新
                        链表到集合
537                 L = lists.elem[lists.length - 1].Li; // 切换到
                        新链表
538             } else
539                 printf("Failed \n");
540             getchar();
541             break;
542         }
543         case 19:
544             printf("Enter the listname to remove : ");
545             char name2[100];
546             scanf("%s", name2);
547             if (RemoveList(lists, name2) == OK)
548                 printf("List removed \n"); // 移除链表
549             else
550                 printf("Can't remove the list \n");
551             getchar();
552             break;
553         case 20: {
554             printf("Enter the name to locate : ");
555             char name3[100];
556             scanf("%s", name3);
557             if (LocateList(lists, name3))
558                 printf("The position is : %d \n", LocateList(
                        lists, name3)); // 查找链表在集合中的位置
559             else
560                 printf("The list does not exist !\n");
561             getchar();
562             break;
563         }
```

```
564             case 21: {
565                 printf("Enter the name to switch : ");
566                 char name3[100];
567                 scanf("%s", name3);
568                 if (SwitchList(lists, name3, L) == OK)
569                     printf("Switched successfully \n"); // 切换当
                        前链表
570                 else
571                     printf("The list does not exist \n");
572                 getchar();
573                 break;
574             }
575             case 22:
576                 ShowAllLists(lists); // 显示所有链表及内容
577                 getchar();
578                 break;
579             case 0:
580                 break;
581             } // end of switch
582         } // end of while
583         printf("Welcome to the system next time \n"); // 退出提示
584         return 0;
585 }
```

6 附录 C 基于二叉链表二叉树实现的源程序

```
1  #include "stdio.h"
2  #include "stdlib.h"
3  #include <malloc.h>
4  #include <string.h>
5  #define TRUE 1
6  #define FALSE 0
7  #define OK 1
8  #define ERROR 0
9  #define INFEASIBLE -1
10 #define OVERFLOW -2
11 #define MAXTREENUM 10
12 typedef int status;
13 typedef int KeyType;
14 typedef struct {
15     KeyType key;
16     char others[20];
17 } TElemType; // 二叉树结点类型定义
18 typedef struct BiTNode { // 二叉链表结点的定义
19     TElemType data;
20     struct BiTNode *lchild, *rchild;
21 } BiTNode, *BiTree;
22 typedef struct {
23     struct {
24         char name[20];
25         BiTree T;
26     } elem[10];
27     int amount;
28 } Trees;
29 typedef struct {
30     int pos;
31     TElemType data;
32 } DEF;
33 status CreateBiTree(BiTree &T, DEF definition[]) {
```

```
34      int i = 0, j;
35      static BiTNode *p[100];
36      while (j = definition[i].pos) {
37          p[j] = (BiTNode *)malloc(sizeof(BiTNode));
38          p[j]->data = definition[i].data;
39          p[j]->lchild = NULL;
40          p[j]->rchild = NULL;
41          if (j != 1) {
42              if (j % 2)
43                  p[j / 2]->rchild = p[j];
44              else
45                  p[j / 2]->lchild = p[j];
46          }
47          i++;
48      }
49      T = p[1];
50      return OK;
51  }
52  status DestroyBiTree(BiTree &T) {
53      if (!T)
54          return ERROR;
55      if (T->lchild)
56          DestroyBiTree(T->lchild);
57      if (T->rchild)
58          DestroyBiTree(T->rchild);
59      free(T);
60      T = NULL;
61      return OK;
62  }
63  status ClearBiTree(BiTree &T) {
64      if (!T)
65          return ERROR;
66      if (T->lchild)
67          ClearBiTree(T->lchild);
68      if (T->rchild)
```

```
69          ClearBiTree(T->rchild);
70      T->data.key = 0;
71      strcpy(T->data.others, "NULL");
72      return OK;
73  }
74  status BiTreeEmpty(BiTree T) {
75      if (!T)
76          return TRUE;
77      else
78          return FALSE;
79  }
80  int BiTreeDepth(BiTree T) {
81      if (!T)
82          return 0;
83      int left = BiTreeDepth(T->lchild);
84      int right = BiTreeDepth(T->rchild);
85      return (left > right) ? (left + 1) : (right + 1);
86  }
87  BiTNode *LocateNode(BiTree T, KeyType e) {
88      if (!T)
89          return NULL;
90      if (T->data.key == e)
91          return T;
92      BiTNode *left = LocateNode(T->lchild, e);
93      if (left)
94          return left;
95      return LocateNode(T->rchild, e);
96  }
97  status Assign(BiTree &T, KeyType e, TElemType value) {
98      BiTNode *target = LocateNode(T, e);
99      BiTNode *err = LocateNode(T, value.key);
100     if (!target)
101         return ERROR;
102     if (err != target && err != NULL)
103         return INFEASIBLE;
```

```
104     target->data = value;
105     return OK;
106 }
107 BiTNode *GetSibling(BiTree T, KeyType e) {
108     if (!T || !T->lchild || !T->rchild)
109         return NULL;
110     if (T->lchild && T->lchild->data.key == e)
111         return T->rchild;
112     if (T->rchild && T->rchild->data.key == e)
113         return T->lchild;
114     BiTNode *tmp = GetSibling(T->lchild, e);
115     if (tmp)
116         return tmp;
117     return GetSibling(T->rchild, e);
118 }
119 status InsertNode(BiTree &T, KeyType e, int LR, TElemType c) {
120     if (LocateNode(T, c.key))
121         return ERROR;
122     if (!LocateNode(T, e))
123         return ERROR;
124     BiTNode *ins = (BiTNode *)malloc(sizeof(BiTNode));
125     ins->data = c;
126     ins->lchild = NULL;
127     ins->rchild = NULL;
128
129     BiTNode *tmp = LocateNode(T, e);
130     if (LR == -1) {
131         ins->rchild = T;
132         ins->lchild = NULL;
133         T = ins;
134     }
135     if (LR == 0) {
136         ins->rchild = tmp->lchild;
137         tmp->lchild = ins;
138     }
```

```
139     if (LR == 1) {
140         ins->rchild = tmp->rchild;
141         tmp->rchild = ins;
142     }
143     return OK;
144 }
145 status DeleteNode(BiTree &T, KeyType e) {
146     if (!T)
147         return ERROR;
148     if (T->data.key == e) {
149         BiTree tmp = T;
150         if (!T->lchild && !T->rchild) {
151             free(T);
152             T = NULL;
153         } else if (!T->lchild && T->rchild) {
154             T = T->rchild;
155             free(tmp);
156         } else if (!T->rchild && T->lchild) {
157             T = T->lchild;
158             free(tmp);
159         } else {
160             BiTree l = T->lchild;
161             BiTree r = T->rchild;
162             BiTree mr = T->lchild;
163             while (mr->rchild)
164                 mr = mr->rchild;
165             mr->rchild = r;
166             T = l;
167             free(tmp);
168         }
169         return OK;
170     }
171     if (DeleteNode(T->lchild, e) == OK)
172         return OK;
173     if (DeleteNode(T->rchild, e) == OK)
```

```
174         return OK;
175 }
176 void visit(BiTree T) {
177     printf("%d,%s ", T->data.key, T->data.others);
178 }
179 status PreOrderTraverse(BiTree &T, void (*visit)(BiTree)) {
180     if (!T)
181         return ERROR;
182     BiTree stack[100];
183     int top = 0;
184     BiTree p = T;
185     while (p || top > 0) {
186         if (p) {
187             visit(p);
188             stack[top++] = p;
189             p = p->lchild;
190         } else {
191             p = stack[--top];
192             p = p->rchild;
193         }
194     }
195     return OK;
196 }
197 status InOrderTraverse(BiTree &T, void (*visit)(BiTree)) {
198     if (!T)
199         return ERROR;
200     InOrderTraverse(T->lchild, visit);
201     visit(T);
202     InOrderTraverse(T->rchild, visit);
203     return OK;
204 }
205 status PostOrderTraverse(BiTree &T, void (*visit)(BiTree)) {
206     if (!T)
207         return ERROR;
208     PostOrderTraverse(T->lchild, visit);
```

```
209     PostOrderTraverse(T->rchild, visit);
210     visit(T);
211     return OK;
212 }
213 status LevelOrderTraverse(BiTree &T, void (*visit)(BiTree)) {
214     if (!T)
215         return ERROR;
216     BiTree queue[100];
217     int front = 0, rear = 0;
218     queue[rear++] = T;
219     while (front < rear) {
220         BiTree cur = queue[front++];
221         visit(cur);
222         if (cur->lchild)
223             queue[rear++] = cur->lchild;
224         if (cur->rchild)
225             queue[rear++] = cur->rchild;
226     }
227     return OK;
228 }
229 int MaxPathSum(BiTree T) {
230     if (!T)
231         return 0;
232     if (!T->lchild && !T->rchild)
233         return T->data.key;
234     int leftSum = MaxPathSum(T->lchild);
235     int rightSum = MaxPathSum(T->rchild);
236     return T->data.key + ((leftSum > rightSum) ? leftSum :
        rightSum);
237 }
238 BiTNode *LowestCommonAncestor(BiTree T, BiTNode *e1, BiTNode *
    e2) {
239     if (!T)
240         return T;
241     if (T->data.key == e1->data.key || T->data.key == e2->data
```

```
            .key)
242             return T;
243         BiTNode *left = LowestCommonAncestor(T->lchild, e1, e2);
244         BiTNode *right = LowestCommonAncestor(T->rchild, e1, e2);
245         if (left && right)
246             return T;
247         return left ? left : right;
248     }
249     status InvertTree(BiTree &T) {
250         if (!T)
251             return ERROR;
252         BiTree tmp = T->lchild;
253         T->lchild = T->rchild;
254         T->rchild = tmp;
255         InvertTree(T->lchild);
256         InvertTree(T->rchild);
257         return OK;
258     }
259     status SaveBiTreeHelper(BiTree &T, FILE *fp) {
260         if (!fp)
261             return ERROR;
262         if (T) {
263             fprintf(fp, "%d %s\n", T->data.key, T->data.others);
264             SaveBiTreeHelper(T->lchild, fp);
265             SaveBiTreeHelper(T->rchild, fp);
266         } else
267             fprintf(fp, "0 NULL\n");
268         return OK;
269     }
270     status LoadBiTreeHelper(BiTree &T, FILE *fp) {
271         if (!fp)
272             return ERROR;
273         int key;
274         char others[20];
275         if (fscanf(fp, "%d %s", &key, others) != 2) {
```

```
276             fclose(fp);
277             fp = NULL;
278             return ERROR;
279         }
280         if (key == 0 && strcmp(others, "NULL") == 0) {
281             T = NULL;
282             return OK;
283         }
284         T = (BiTree)malloc(sizeof(BiTNode));
285         if (!T)
286             return OVERFLOW;
287         T->data.key = key;
288         strcpy(T->data.others, others);
289         LoadBiTreeHelper(T->lchild, fp);
290         LoadBiTreeHelper(T->rchild, fp);
291         return OK;
292     }
293     status SaveBiTree(BiTree &T, char FileName[]) {
294         FILE *fp = fopen(FileName, "r");
295         if (fp) {
296             fclose(fp);
297             fp = fopen(FileName, "w");
298         } else {
299             fp = fopen(FileName, "w");
300         }
301         if (!fp)
302             return ERROR;
303         SaveBiTreeHelper(T, fp);
304         fclose(fp);
305         return OK;
306     }
307     status LoadBiTree(BiTree &T, char FileName[]) {
308         static FILE *fp = NULL;
309         if (!fp) {
310             fp = fopen(FileName, "r");
```

```
311             if (!fp)
312                 return ERROR;
313         }
314         LoadBiTreeHelper(T, fp);
315         return OK;
316     }
317     status AddTree(Trees &trees, char TreeName[]) {
318         if (trees.amount > MAXTREENUM)
319             return ERROR; // 超过最大树数量
320         for (int i = 0; i < trees.amount; i++)
321             if (strcmp(trees.elem[i].name, TreeName) == 0) {
322                 printf("The name has already exists \n");
323                 return ERROR; // 名称已存在
324             }
325         strcpy(trees.elem[trees.amount].name, TreeName); // 复制树
                名
326         trees.elem[trees.amount].T = NULL;                    // 初始化
                树指针
327         trees.amount++;
328         return OK;
329     }
330     status RemoveTree(Trees &trees, char TreeName[]) {
331         for (int i = 0; i < trees.amount; i++) {
332             if (strcmp(trees.elem[i].name, TreeName) == 0) {
333                 DestroyBiTree(trees.elem[i].T); // 释放树内存
334                 for (int j = i; j < trees.amount - 1; j++)
335                     trees.elem[j] = trees.elem[j + 1]; // 后续前移
336                 trees.amount--;
337                 return OK;
338             }
339         }
340         return ERROR;
341     }
342     int LocateTree(Trees trees, char TreeName[]) {
343         for (int i = 0; i < trees.amount; i++) {
```

```
344             if (strcmp(trees.elem[i].name, TreeName) == 0)
345                 return i + 1; // 返回下标+1
346         }
347         return ERROR;
348     }
349     status SwitchTrees(Trees trees, char TreeName[], BiTree &T) {
350         for (int i = 0; i < trees.amount; i++) {
351             if (strcmp(trees.elem[i].name, TreeName) == 0) {
352                 T = trees.elem[i].T; // 切换当前树
353                 return OK;
354             }
355         }
356         return ERROR;
357     }
358     void ShowAllTrees(Trees trees) {
359         for (int i = 0; i < trees.amount; i++) {
360             printf("%s", trees.elem[i].name); // 输出树名
361             printf("\nPreOrder : ");
362             PreOrderTraverse(trees.elem[i].T, visit); // 先序遍历
363             printf("\nInOrder : ");
364             InOrderTraverse(trees.elem[i].T, visit); // 中序遍历
365             printf("\n");
366         }
367     }
368     #include "def3.h"
369     #include "system3Func.h"
370     int main() {
371         BiTree T = NULL;
372         Trees trees;
373         trees.amount = 0;
374         char filename[] = "c:\\Users\\12841\\Desktop\\none\\2025
              springDS\\experiments\\ex3\\bitree.txt";
375         int op = 1;
376         printf(" Menu for BiTree Structure\n"
377                "-------------------------------------------------\
```

```
                n"
378             "1. CreateBiTree        13.PostOrderTraverse\n"
379             "2. DestroyBiTree       14.LevelOrderTraverse\n"
380             "3. ClearBiTree         15.MaxPathSum\n"
381             "4. BiTreeEmpty         16.LowestCommonAncestor \n"
382             "5. BiTreeDepth         17.InvertTree\n"
383             "6. LocateNode          18.SaveBiTree \n"
384             "7. Assign              19.LoadBiTree\n"
385             "8. GetSibling          20.AddTree\n"
386             "9. InsertNode          21.RemoveTree\n"
387             "10.DeleteNode          22.LocateTree\n"
388             "11.PreOrderTraverse    23.SwitchTrees\n"
389             "12.InOrderTraverse     24.ShowAllTrees\n"
390             "0. Exit"
391             "-------------------------------------------------\
                n");
392     while (op) {
393         printf("\n Choose your operation : ");
394         scanf("%d", &op);
395         switch (op) {
396         case 0:
397             break;
398         case 1: {
399             DEF difinition[100];
400             int count = 0;
401             printf("Enter the difinition BiTree : ");
402             do {
403                 scanf("%d%d%s", &difinition[count].pos, &
                        difinition[count].data.key, difinition[
                        count].data.others);
404             } while (difinition[count++].pos); // 输入树的定
                义, 直到pos为0
405             if (CreateBiTree(T, difinition) == OK)
406                 printf("BiTree create successfully \n"); // 创
                    建二叉树
```

```
407                 else
408                     printf("Failed \n");
409                 break;
410             }
411         case 2:
412             if (T == NULL) {
413                 printf("The BiTree does not exist \n"); // 树
                        不存在
414                 break;
415             }
416             if (DestroyBiTree(T) == OK)
417                 printf("Destroy successfully \n"); // 销毁树
418             else
419                 printf("Failed \n");
420             break;
421         case 3:
422             if (T == NULL) {
423                 printf("The BiTree does not exist \n"); // 树
                        不存在
424                 break;
425             }
426             if (ClearBiTree(T) == OK)
427                 printf("Clear successfully \n"); // 清空树内容
428             else
429                 printf("Failed \n");
430             break;
431         case 4:
432             if (BiTreeEmpty(T) == TRUE)
433                 printf("The BiTree is empty \n"); // 判断树是
                        否为空
434             else
435                 printf("The BiTree is not empty \n");
436             break;
437         case 5: {
438             int depth = BiTreeDepth(T);
```

```
439             printf("The depth of this BiTree is %d \n", depth)
                    ; // 输出树深度
440             break;
441         }
442         case 6: {
443             BiTNode *tmp;
444             KeyType e;
445             printf("Enter the key node : ");
446             scanf("%d", &e);
447             getchar();
448             tmp = LocateNode(T, e);
449             if (tmp != NULL)
450                 printf("The node is %d %s \n", tmp->data.key,
                        tmp->data.others); // 查找节点
451             else
452                 printf("Couldn't find the node with key %d \n"
                        , e);
453             break;
454         }
455         case 7: {
456             KeyType e;
457             TElemType value;
458             printf("Enter the key and the message of value(key
                    and others) :");
459             scanf("%d%d%s", &e, &value.key, value.others);
460             getchar();
461             status as = Assign(T, e, value);
462             if (as == OK)
463                 printf("Assign successfully \n"); // 赋值节点
464             else if (as == INFEASIBLE)
465                 printf("The value's key has already exists \n"
                        );
466             else
467                 printf("Assign failed \n");
468             break;
```

```
469         }
470         case 8: {
471             KeyType e;
472             printf("Enter the key to find sibling : ");
473             scanf("%d", &e);
474             getchar();
475             BiTNode *p = GetSibling(T, e);
476             if (p)
477                 printf("The sibling is %d %s \n", p->data.key,
                        p->data.others); // 查找兄弟节点
478             else
479                 printf("The target node doesn't have sibling \
                        n");
480             break;
481         }
482         case 9: {
483             KeyType e;
484             int mode;
485             TElemType c;
486             printf("Enter the target node's key : ");
487             scanf("%d", &e);
488             getchar();
489             printf("Enter the mdoe(-1/0/1) : ");
490             scanf("%d", &mode);
491             getchar();
492             printf("Enter the key and others to insert : ");
493             scanf("%d%s", &c.key, c.others);
494             getchar();
495             if (InsertNode(T, e, mode, c) == OK)
496                 printf("Insert successfully \n"); // 插入节点
497             else
498                 printf("Failed \n");
499             break;
500         }
501         case 10: {
```

```
502             KeyType e;
503             printf("Enter the key node to delete : ");
504             scanf("%d", &e);
505             getchar();
506             if (DeleteNode(T, e) == OK)
507                 printf("Delete successfully \n"); // 删除节点
508             else
509                 printf("Failed \n");
510             break;
511         }
512         case 11:
513             if (!PreOrderTraverse(T, visit))
514                 printf("Error \n"); // 先序遍历
515             else
516                 printf("\n");
517             break;
518         case 12:
519             if (!InOrderTraverse(T, visit))
520                 printf("Error \n"); // 中序遍历
521             else
522                 printf("\n");
523             break;
524         case 13:
525             if (!PostOrderTraverse(T, visit))
526                 printf("Error \n"); // 后序遍历
527             else
528                 printf("\n");
529             break;
530         case 14:
531             if (!LevelOrderTraverse(T, visit))
532                 printf("Error \n"); // 层序遍历
533             else
534                 printf("\n");
535             break;
536         case 15:
```

```
537             if (!T) {
538                 printf("The BiTree doesn't exist \n"); // 树不
                        存在
539                 break;
540             }
541             printf("The max path sum is %d \n", MaxPathSum(T))
                    ; // 最大路径和
542             break;
543         case 16: {
544             if (!T) {
545                 printf("The BiTree doesn't exist \n"); // 树不
                        存在
546                 break;
547             }
548             int key1, key2;
549             printf("Enter the key of e1 and e2 : ");
550             scanf("%d%d", &key1, &key2);
551             getchar();
552             BiTNode *e1 = LocateNode(T, key1);
553             BiTNode *e2 = LocateNode(T, key2);
554             if (!e1 || !e2) {
555                 printf("One or two node is not found \n"); //
                        节点不存在
556                 break;
557             }
558             BiTNode *p = LowestCommonAncestor(T, e1, e2);
559             if (p)
560                 printf("The lowest common ancestor is %d %s \n
                        ", p->data.key, p->data.others); // 最近公
                        共祖先
561             else
562                 printf("No common ancestor found \n");
563             break;
564         }
565         case 17:
```

```
566             if (InvertTree(T) == OK)
567                 printf("Invert successfully \n"); // 翻转树
568             else
569                 printf("Failed \n");
570             break;
571         case 18:
572             if (SaveBiTree(T, filename) == OK)
573                 printf("Save successfully \n"); // 保存树到文
                        件
574             else
575                 printf("Error \n");
576             break;
577         case 19: {
578             char name[20];
579             printf("Input the tree name to load : ");
580             scanf("%s", name);
581             getchar();
582             BiTree t = NULL;
583             if (LoadBiTree(t, filename) == OK)
584                 printf("Load successfully \n"); // 加载树
585             else
586                 printf("Error \n");
587             printf("Do you want to check the tree(Y/N) : ");
588             char opt;
589             scanf("%c", &opt);
590             getchar();
591             if (opt == 'Y' || opt == 'y') {
592                 printf("PreOrder :");
593                 PreOrderTraverse(t, visit);
594                 printf("\nInOrder :");
595                 InOrderTraverse(t, visit);
596             }
597             trees.elem[trees.amount].T = t;
598             strcpy(trees.elem[trees.amount++].name, name);
599             T = t;
```

```
600             printf("\n");
601             break;
602         }
603         case 20: {
604             printf("Enter the name to add : ");
605             char name[20];
606             scanf("%s", name);
607             getchar();
608             if (AddTree(trees, name) == OK) {
609                 DEF difinition[100];
610                 int count = 0;
611                 printf("Enter the difinition BiTree : ");
612                 do {
613                     scanf("%d%d%s", &difinition[count].pos, &
                            difinition[count].data.key, difinition[
                            count].data.others);
614                 } while (difinition[count++].pos); // 输入树定
                    义
615                 if (CreateBiTree(trees.elem[trees.amount - 1].
                    T, difinition) == OK) {
616                     printf("Add successfully \n"); // 添加树
617                     T = trees.elem[trees.amount - 1].T;
618                 } else
619                     printf("Error \n");
620             }
621             break;
622         }
623         case 21: {
624             printf("Enter the name to remove : ");
625             char name[20];
626             scanf("%s", name);
627             getchar();
628             if (RemoveTree(trees, name) == OK)
629                 printf("Remove successfully\n"); // 移除树
630             else
```

```
631                 printf("Failed \n");
632             break;
633         }
634         case 22: {
635             printf("Enter the name to locate : ");
636             char name3[100];
637             scanf("%s", name3);
638             if (LocateTree(trees, name3))
639                 printf("The position is : %d \n", LocateTree(
                        trees, name3)); // 查找树在集合中的位置
640             else
641                 printf("The tree does not exist !\n");
642             getchar();
643             break;
644         }
645         case 23: {
646             printf("Enter the name to switch : ");
647             char name[20];
648             scanf("%s", name);
649             getchar();
650             if (SwitchTrees(trees, name, T) == OK)
651                 printf("Switch successfully \n"); // 切换当前
                        树
652             else
653                 printf("Failed \n");
654             break;
655         }
656         case 24:
657             ShowAllTrees(trees); // 显示所有树及遍历
658             break;
659         default:
660             break;
661         }
662     }
663     printf("Welcome to the system next time \n"); // 退出提示
```

```
664 }
```

7 附录 D 基于邻接表图实现的源程序

```
1  #include "stdio.h"
2  #include "stdlib.h"
3  #include <string.h>
4  #include <malloc.h>
5  #define TRUE 1
6  #define FALSE 0
7  #define OK 1
8  #define ERROR 0
9  #define INFEASIBLE -1
10 #define OVERFLOW -2
11 #define MAX_VERTEX_NUM 20
12 typedef int status;
13 typedef int KeyType;
14 typedef enum {
15         DG,DN,UDG,UDN
16 } GraphKind;
17 typedef struct {
18     KeyType key;
19     char others[20];
20 } VertexType; // 顶点类型定义
21 typedef struct ArcNode {      // 表结点类型定义
22     int adjvex;               // 顶点位置编号
23     struct ArcNode *nextarc; // 下一个表结点指针
24 } ArcNode;
25 typedef struct VNode { // 头结点及其数组类型定义
26     VertexType data;    // 顶点信息
27     ArcNode *firstarc; // 指向第一条弧
28 } VNode, AdjList[MAX_VERTEX_NUM];
29 typedef struct {          // 邻接表的类型定义
30     AdjList vertices;    // 头结点数组
31     int vexnum, arcnum; // 顶点数、弧数
32     GraphKind kind;      // 图的类型
33 } ALGraph;
```

```
34 typedef struct {
35     int amount;
36     struct {
37         char name[20];
38         ALGraph G;
39     } elem[20];
40 } Graphs;
41 status CreateGraph(ALGraph &G, VertexType V[], KeyType VR
       [][2]) {
42     int i = 0;
43     while (V[i].key != -1)
44         i++;
45     G.vexnum = i;
46     if (G.vexnum == 0 || G.vexnum > MAX_VERTEX_NUM)
47         return ERROR;
48     for (int i = 0; i < G.vexnum; i++)
49         for (int j = i + 1; j < G.vexnum; j++)
50             if (V[i].key == V[j].key)
51                 return ERROR;
52     for (int i = 0; i < G.vexnum; i++) {
53         G.vertices[i].data = V[i];
54         G.vertices[i].firstarc = NULL;
55     }
56     i = 0;
57     while (VR[i][0] != -1) {
58         int v1 = -1, v2 = -1;
59         for (int j = 0; j < G.vexnum; j++) {
60             if (G.vertices[j].data.key == VR[i][0])
61                 v1 = j;
62             if (G.vertices[j].data.key == VR[i][1])
63                 v2 = j;
64         }
65         if (v1 == -1 || v2 == -1)
66             return ERROR;
67         ArcNode *arc12 = (ArcNode *)malloc(sizeof(ArcNode));
```

```
68          if (!arc12)
69              return ERROR;
70          arc12->adjvex = v2;
71          arc12->nextarc = G.vertices[v1].firstarc;
72          G.vertices[v1].firstarc = arc12;
73          ArcNode *arc21 = (ArcNode *)malloc(sizeof(ArcNode));
74          arc21->adjvex = v1;
75          arc21->nextarc = G.vertices[v2].firstarc;
76          G.vertices[v2].firstarc = arc21;
77          i++;
78      }
79      G.arcnum = i;
80      return OK;
81  }
82  status DestroyGraph(ALGraph &G) {
83      if (G.vexnum <= 0)
84          return ERROR;
85      for (int i = 0; i < G.vexnum; i++) {
86          ArcNode *p = G.vertices[i].firstarc;
87          while (p) {
88              ArcNode *tmp = p;
89              p = p->nextarc;
90              free(tmp);
91              G.vertices[i].firstarc = p;
92          }
93          G.vertices[i].firstarc = NULL;
94      }
95      G.arcnum = 0;
96      G.vexnum = 0;
97      return OK;
98  }
99  int LocateVex(ALGraph G, KeyType u) {
100     for (int i = 0; i < G.vexnum; i++) {
101         if (G.vertices[i].data.key == u)
102             return i;
```

```
103     }
104     return -1;
105 }
106 status PutVex(ALGraph &G, KeyType u, VertexType value) {
107     if (G.vexnum <= 0)
108         return ERROR;
109     int find = 0;
110     for (int i = 0; i < G.vexnum; i++) {
111         if (G.vertices[i].data.key == value.key)
112             return ERROR;
113     }
114     for (int i = 0; i < G.vexnum; i++) {
115         if (G.vertices[i].data.key == u) {
116             G.vertices[i].data.key = value.key;
117             strcpy(G.vertices[i].data.others, value.others);
118             find = 1;
119         }
120     }
121     if (!find)
122         return ERROR;
123     return OK;
124 }
125 int FirstAdjVex(ALGraph &G, KeyType u) {
126     for (int i = 0; i < G.vexnum; i++) {
127         if (G.vertices[i].data.key == u) {
128             if (G.vertices[i].firstarc == NULL)
129                 return -1;
130             return G.vertices[i].firstarc->adjvex;
131         }
132     }
133     return -1;
134 }
135 int NextAdjVex(ALGraph &G, KeyType v, KeyType w) {
136     int vpos = -1, wpos = -1;
137     for (int i = 0; i < G.vexnum; i++) {
```

```
138             if (G.vertices[i].data.key == v)
139                 vpos = i;
140             if (G.vertices[i].data.key == w)
141                 wpos = i;
142         }
143         if (vpos == -1 || wpos == -1)
144             return -1;
145         ArcNode *p = G.vertices[vpos].firstarc;
146         while (p != NULL) {
147             if (p->adjvex == wpos) {
148                 if (!p->nextarc)
149                     return -1;
150                 return p->nextarc->adjvex;
151             }
152             p = p->nextarc;
153         }
154         return -1;
155     }
156     status InsertVex(ALGraph &G, VertexType v) {
157         if (G.vexnum >= MAX_VERTEX_NUM)
158             return ERROR;
159         for (int i = 0; i < G.vexnum; i++)
160             if (G.vertices[i].data.key == v.key)
161                 return ERROR;
162         G.vertices[G.vexnum].data = v;
163         G.vertices[G.vexnum].firstarc = NULL;
164         G.vexnum += 1;
165         return OK;
166     }
167     status DeleteVex(ALGraph &G, KeyType v) {
168         int find = 0, pos = -1;
169         int del = 0;
170         ArcNode *p = NULL;
171         for (int i = 0; i < G.vexnum; i++)
172             if (G.vertices[i].data.key == v) {
```

```
173                 find = 1;
174                 pos = i;
175                 break;
176             }
177         if (!find)
178             return ERROR;
179         if (G.vexnum == 1)
180             return ERROR;
181         p = G.vertices[pos].firstarc;
182         while (p) {
183             ArcNode *tmp = p;
184             p = p->nextarc;
185             free(tmp);
186             del++;
187         }
188         for (int i = 0; i < G.vexnum; i++) {
189             if (i == pos)
190                 continue;
191             p = G.vertices[i].firstarc;
192             if (p && p->adjvex == pos) {
193                 G.vertices[i].firstarc = p->nextarc;
194                 free(p);
195                 del++;
196                 p = G.vertices[i].firstarc;
197             }
198             while (p && p->nextarc) {
199                 if (p->nextarc->adjvex == pos) {
200                     ArcNode *tmp = p->nextarc;
201                     p->nextarc = p->nextarc->nextarc;
202                     free(tmp);
203                     del++;
204                 } else
205                     p = p->nextarc;
206             }
207         }
```

```
208     for (int i = pos; i < G.vexnum - 1; i++) {
209         G.vertices[i].data = G.vertices[i + 1].data;
210         G.vertices[i].firstarc = G.vertices[i + 1].firstarc;
211     }
212     G.vexnum--;
213     for (int i = 0; i < G.vexnum; i++) {
214         p = G.vertices[i].firstarc;
215         while (p) {
216             if (p->adjvex > pos)
217                 p->adjvex--;
218             p = p->nextarc;
219         }
220     }
221     G.arcnum -= del / 2;
222     return OK;
223 }
224 status InsertArc(ALGraph &G, KeyType v, KeyType w) {
225     int vpos = -1, wpos = -1;
226     for (int i = 0; i < G.vexnum; i++) {
227         if (G.vertices[i].data.key == v)
228             vpos = i;
229         if (G.vertices[i].data.key == w)
230             wpos = i;
231     }
232     if (vpos == -1 || wpos == -1)
233         return ERROR;
234     ArcNode *p = G.vertices[vpos].firstarc;
235     while (p) {
236         if (p->adjvex == wpos)
237             return ERROR;
238         p = p->nextarc;
239     }
240     ArcNode *vw = (ArcNode *)malloc(sizeof(ArcNode));
241     ArcNode *wv = (ArcNode *)malloc(sizeof(ArcNode));
242     vw->nextarc = G.vertices[vpos].firstarc;
```

```
243     G.vertices[vpos].firstarc = vw;
244     vw->adjvex = wpos;
245     wv->nextarc = G.vertices[wpos].firstarc;
246     G.vertices[wpos].firstarc = wv;
247     wv->adjvex = vpos;
248     G.arcnum += 1;
249     return OK;
250 }
251 status DeleteArc(ALGraph &G, KeyType v, KeyType w) {
252     int vpos = -1, wpos = -1;
253     for (int i = 0; i < G.vexnum; i++) {
254         if (G.vertices[i].data.key == v)
255             vpos = i;
256         if (G.vertices[i].data.key == w)
257             wpos = i;
258     }
259     if (vpos == -1 || wpos == -1)
260         return ERROR;
261     ArcNode *p = G.vertices[vpos].firstarc;
262     int find = 0;
263     while (p) {
264         if (p->adjvex == wpos) {
265             find = 1;
266             break;
267         }
268         p = p->nextarc;
269     }
270     if (!find)
271         return ERROR;
272     p = G.vertices[vpos].firstarc;
273     if (p->adjvex == wpos) {
274         G.vertices[vpos].firstarc = p->nextarc;
275         free(p);
276     };
277     p = G.vertices[vpos].firstarc;
```

```
278     while (p && p->nextarc) {
279         if (p->nextarc->adjvex == wpos) {
280             ArcNode *tmp = p->nextarc;
281             p->nextarc = p->nextarc->nextarc;
282             free(tmp);
283             break;
284         }
285         p = p->nextarc;
286     }
287     p = G.vertices[wpos].firstarc;
288     while (p && p->nextarc) {
289         if (p->nextarc->adjvex == vpos) {
290             ArcNode *tmp = p->nextarc;
291             p->nextarc = p->nextarc->nextarc;
292             free(tmp);
293             break;
294         }
295         p = p->nextarc;
296     }
297     G.arcnum -= 1;
298     return OK;
299 }
300 void visit(VertexType v) {
301     printf("%d %s", v.key, v.others);
302 }
303 status DFS(ALGraph &G, int v, int visited[], void (*visit)(
        VertexType)) {
304     visited[v] = 1;
305     visit(G.vertices[v].data);
306     ArcNode *p = G.vertices[v].firstarc;
307     while (p) {
308         if (!visited[p->adjvex])
309             DFS(G, p->adjvex, visited, visit);
310         p = p->nextarc;
311     }
```

```
312     return OK;
313 }
314 status DFSTraverse(ALGraph &G, void (*visit)(VertexType)) {
315     static int visited[MAX_VERTEX_NUM] = {0};
316     for (int i = 0; i < G.vexnum; i++)
317         visited[i] = 0;
318     for (int i = 0; i < G.vexnum; i++) {
319         if (!visited[i])
320             DFS(G, i, visited, visit);
321     }
322     return OK;
323 }
324 status BFSTraverse(ALGraph &G, void (*visit)(VertexType)) {
325     int visited[MAX_VERTEX_NUM] = {0};
326     int queue[MAX_VERTEX_NUM];
327     int front = 0, rear = 0;
328     for (int i = 0; i < G.vexnum; i++) {
329         if (!visited[i]) {
330             visited[i] = 1;
331             visit(G.vertices[i].data);
332             queue[rear++] = i;
333             while (front < rear) {
334                 int v = queue[front++];
335                 ArcNode *p = G.vertices[v].firstarc;
336                 while (p) {
337                     if (!visited[p->adjvex]) {
338                         visited[p->adjvex] = 1;
339                         visit(G.vertices[p->adjvex].data);
340                         queue[rear++] = p->adjvex;
341                     }
342                     p = p->nextarc;
343                 }
344             }
345         }
346     }
```

```
347     return OK;
348 }
349 status VerticesSetLessThanK(ALGraph G, VertexType v, int k) {
350     if (k < 0)
351         return ERROR;
352     int *visited = (int *)malloc(G.vexnum * sizeof(int));
353     int *d = (int *)malloc(G.vexnum * sizeof(int));
354     if (!visited || !d)
355         return ERROR;
356     for (int i = 0; i < G.vexnum; i++) {
357         visited[i] = 0;
358         d[i] = -1;
359     }
360     int pos = LocateVex(G, v.key);
361     if (pos == -1)
362         return ERROR;
363     d[pos] = 0;
364     int queue[MAX_VERTEX_NUM];
365     memset(queue, -1, MAX_VERTEX_NUM * sizeof(int));
366     int front = 0, rear = 0;
367     queue[rear++] = pos;
368     visited[pos] = 1;
369     while (front < rear) {
370         int cur = queue[front++];
371         ArcNode *p = G.vertices[cur].firstarc;
372         while (p) {
373             if (!visited[p->adjvex]) {
374                 d[p->adjvex] = d[cur] + 1;
375                 visited[p->adjvex] = 1;
376                 queue[rear++] = p->adjvex;
377             }
378             p = p->nextarc;
379         }
380     }
381     for (int i = 0; i < G.vexnum; i++) {
```

```
382             if (d[i] >= 0 && d[i] < k) {
383                 printf("(%d,%s) ", G.vertices[i].data.key, G.
                        vertices[i].data.others);
384             }
385         }
386         printf("\n");
387         return OK;
388     }
389     int ShortestPathLength(ALGraph G, VertexType v, VertexType w)
        {
390         int pos_v = LocateVex(G, v.key);
391         int pos_w = LocateVex(G, w.key);
392         if (pos_v == -1 || pos_w == -1)
393             return -1;
394         int *visited = (int *)malloc(G.vexnum * sizeof(int));
395         int *dist = (int *)malloc(G.vexnum * sizeof(int));
396         int *parent = (int *)malloc(G.vexnum * sizeof(int));
397         if (!visited || !dist)
398             return -1;
399         for (int i = 0; i < G.vexnum; i++) {
400             visited[i] = 0;
401             dist[i] = -1;
402             parent[i] = -1;
403         }
404         int *queue = (int *)malloc(G.vexnum * sizeof(int));
405         if (!queue) {
406             free(visited);
407             free(dist);
408             free(parent);
409             return -1;
410         }
411         int front = 0, rear = 0;
412         queue[rear++] = pos_v;
413         visited[pos_v] = 1;
414         dist[pos_v] = 0;
```

```
415     while (front < rear) {
416         int curr = queue[front++];
417         if (curr == pos_w) {
418             int path[MAX_VERTEX_NUM];
419             int pathLen = 0;
420             for (int i = curr; i != -1; i = parent[i])
421                 path[pathLen++] = i;
422             for (int i = pathLen - 1; i >= 0; i--) {
423                 printf("(%d,%s)", G.vertices[path[i]].data.key
                        , G.vertices[path[i]].data.others); // 输出
                        路径节点
424                 if (i > 0)
425                     printf("->"); // 路径箭头
426             }
427             printf("\nlength is %d \n", dist[curr]); // 输出路
                    径长度
428             free(visited);
429             free(dist);
430             free(queue);
431             free(parent);
432             return dist[curr];
433         }
434         ArcNode *p = G.vertices[curr].firstarc;
435         while (p) {
436             int next = p->adjvex;
437             if (!visited[next]) {
438                 visited[next] = 1;
439                 dist[next] = dist[curr] + 1;
440                 parent[next] = curr;
441                 queue[rear++] = next;
442             }
443             p = p->nextarc;
444         }
445     }
446     free(visited);
```

```
447         free(dist);
448         free(queue);
449         free(parent);
450         return -1;
451     }
452     void DFS_C(ALGraph &G, int v, int visited[]) {
453         visited[v] = 1; // 标记已访问
454         ArcNode *p = G.vertices[v].firstarc;
455         while (p) {
456             if (!visited[p->adjvex])
457                 DFS_C(G, p->adjvex, visited); // 递归访问
458             p = p->nextarc;
459         }
460     }
461     int ConnectedComponentNums(ALGraph &G) {
462         if (G.vexnum <= 0)
463             return 0; // 空图
464         int *visited = (int *)malloc(G.vexnum * sizeof(int));
465         if (!visited)
466             return 0;
467         for (int i = 0; i < G.vexnum; i++)
468             visited[i] = 0;
469         int count = 0;
470         for (int i = 0; i < G.vexnum; i++) {
471             if (!visited[i]) {
472                 DFS_C(G, i, visited); // 统计连通分量
473                 count++;
474             }
475         }
476         free(visited);
477         return count;
478     }
479     status SaveGraph(ALGraph G, char FileName[]) {
480         FILE *fp = fopen(FileName, "w");
481         if (!fp)
```

```
482             return ERROR;                   // 文件打开失败
483         fprintf(fp, "%d\n", G.vexnum); // 写入顶点数
484         for (int i = 0; i < G.vexnum; i++) {
485             fprintf(fp, "%d %s ", G.vertices[i].data.key, G.
                    vertices[i].data.others); // 写入顶点信息
486             ArcNode *p = G.vertices[i].firstarc;
487             while (p) {
488                 fprintf(fp, "%d ", p->adjvex); // 写入邻接点
489                 p = p->nextarc;
490             }
491             fprintf(fp, "-1\n"); // 邻接表结束
492         }
493         fclose(fp);
494         return OK;
495     }
496     status LoadGraph(ALGraph &G, char FileName[]) {
497         FILE *fp = fopen(FileName, "r");
498         KeyType m;
499         if (!fp)
500             return ERROR;                   // 文件打开失败
501         fscanf(fp, "%d", &G.vexnum); // 读取顶点数
502         for (int i = 0; i < G.vexnum; i++) {
503             fscanf(fp, "%d %s", &G.vertices[i].data.key, G.
                    vertices[i].data.others); // 读取顶点信息
504             G.vertices[i].firstarc = NULL;
505             ArcNode *tail = NULL;
506             while (fscanf(fp, "%d", &m) == 1 && m != -1) {
507                 ArcNode *p = (ArcNode *)malloc(sizeof(ArcNode));
508                 p->adjvex = m;
509                 p->nextarc = NULL;
510                 if (G.vertices[i].firstarc == NULL)
511                     G.vertices[i].firstarc = p; // 第一个邻接点
512                 else
513                     tail->nextarc = p; // 追加邻接点
514                 tail = p;
```

```
515             }
516         }
517         fclose(fp);
518         return OK;
519     }
520     status AddGraph(Graphs &graphs, char GraphName[]) {
521         for (int i = 0; i < graphs.amount; i++)
522             if (strcmp(graphs.elem[i].name, GraphName) == 0)
523                 return ERROR;                                    // 名
                        称重复
524         strcpy(graphs.elem[graphs.amount].name, GraphName); // 复
            制名称
525         graphs.elem[graphs.amount].G.vexnum = 0;
526         graphs.elem[graphs.amount].G.arcnum = 0;
527         graphs.amount++;
528         return OK;
529     }
530     status RemoveGraph(Graphs &graphs, char GraphName[]) {
531         if (graphs.amount <= 0)
532             return ERROR; // 没有图
533         for (int i = 0; i < graphs.amount; i++) {
534             if (strcmp(graphs.elem[i].name, GraphName) == 0) {
535                 DestroyGraph(graphs.elem[i].G); // 释放图内存
536                 for (int j = i; j < graphs.amount - 1; j++)
537                     graphs.elem[j] = graphs.elem[j + 1]; // 前移
538                 graphs.amount--;
539                 return OK;
540             }
541         }
542         DestroyGraph(graphs.elem[graphs.amount].G);
543         return ERROR;
544     }
545     int LocateGraph(Graphs &graphs, char GraphName[]) {
546         for (int i = 0; i < graphs.amount; i++) {
547             if (strcmp(graphs.elem[i].name, GraphName) == 0)
```

```
548                 return i + 1; // 返回下标+1
549         }
550         return ERROR;
551     }
552     status SwitchGraph(Graphs &graphs, char GraphName[], ALGraph G
        ) {
553         for (int i = 0; i < graphs.amount; i++) {
554             if (strcmp(graphs.elem[i].name, GraphName) == 0) {
555                 G = graphs.elem[i].G; // 切换当前图
556                 return OK;
557             }
558         }
559         return ERROR;
560     }
561     void ShowAllGraphs(Graphs graphs) {
562         for (int i = 0; i < graphs.amount; i++) {
563             printf("%s\n", graphs.elem[i].name); // 输出图名
564             for (int j = 0; j < graphs.elem[i].G.vexnum; j++) {
565                 printf("%d %s ", graphs.elem[i].G.vertices[j].data
                        .key,
566                        graphs.elem[i].G.vertices[j].data.others);
                           // 输出顶点
567                 ArcNode *p = graphs.elem[i].G.vertices[j].firstarc
                        ;
568                 while (p) {
569                     printf("%d ", p->adjvex); // 输出邻接点
570                     p = p->nextarc;
571                 }
572                 printf("\n");
573             }
574             printf("\n");
575         }
576     }
577     #include "def4.h"
578     #include "system4Func.h"
```

```
579 int main() {
580     ALGraph G;                          // 当前操作的图
581     Graphs graphs;                      // 图集合
582     graphs.amount = 0;                  // 图集合数量初始化
583     char filename[] = "./graph.txt"; // 默认文件名
584     int op = 1;                         // 操作选项
585     printf("Menu for Graph Structure\n"
586            "-------------------------------------------------\
               n"
587            "1.CreateGraph              2.DestroyGraph       \n
               "
588            "3.LocateVex                4.PutVex             \n
               "
589            "5.FirstAdjVex              6.NextAdjVex         \n
               "
590            "7.InsertVex                8.DeleteVex          \n
               "
591            "9.InsertArc                10.DeleteArc         \n
               "
592            "11.DFSTraverse             12.BFSTraverse       \n
               "
593            "13.VerticesSetLessThanK    14.ShortestPathLength\n
               "
594            "15.ConnectedComponentNums  16.SaveGraph         \n
               "
595            "17.LoadGraph               18.AddGraph          \n
               "
596            "19.RemoveGraph             20.LocateGraph       \n
               "
597            "21.SwitchGraph             22.ShowAllGraphs     \n
               "
598            "0.Exit\n"
599            "-------------------------------------------------\
               n");
600     while (op) {
```

```
601         printf("\n Please Choose your operation : ");
602         scanf("%d", &op); // 读取操作选项
603         switch (op) {
604         case 0:
605             break;
606         case 1: {
607             VertexType V[100];
608             KeyType VR[100][2];
609             printf("Input the name :");
610             scanf("%s", graphs.elem[graphs.amount].name); //
                    输入图名
611             G.vexnum = 0, G.arcnum = 0;
612             printf("Input the vexs and keys definition :");
613             while (scanf("%d%s", &V[G.vexnum].key, V[G.vexnum
                    ].others)) {
614                 if (V[G.vexnum].key == -1)
615                     break;
616                 G.vexnum++;
617             }
618             while (scanf("%d%d", &VR[G.arcnum][0], &VR[G.
                    arcnum][1])) {
619                 if (VR[G.arcnum][0] == -1)
620                     break;
621                 G.arcnum++;
622             }
623
624             if (CreateGraph(G, V, VR) == OK) {
625                 graphs.elem[graphs.amount].G = G; // 保存图到
                        集合
626                 printf("Graph create successfully \n");
627             } else
628                 printf("Failed \n");
629
630             break;
631         }
```

```
632         case 2: {
633             if (G.vexnum <= 0) {
634                 printf("The graph does not exist \n"); // 图不
                        存在
635                 break;
636             }
637             if (DestroyGraph(G) == OK)
638                 printf("Destroy succeddfully \n"); // 销毁图
639             else
640                 printf("Failed \n");
641             break;
642         }
643         case 3: {
644             KeyType u;
645             printf("Input the key of the vex :");
646             scanf("%d", &u);
647             getchar();
648             int p = LocateVex(G, u);
649             if (p >= 0 && p < G.vexnum)
650                 printf("The vex is (%d,%s) \n", G.vertices[p].
                        data.key, G.vertices[p].data.others); // 查
                        找顶点
651             else
652                 printf("Failed \n");
653             break;
654         }
655         case 4: {
656             KeyType u;
657             VertexType value;
658             printf("Input the key :");
659             scanf("%d", &u);
660             getchar();
661             int find = 0;
662             for (int i = 0; i < G.vexnum; i++) {
663                 if (u == G.vertices[i].data.key) {
```

```
664                     find = 1;
665                 }
666             }
667             if (find == 0) {
668                 printf("The key does not exist \n"); // 顶点不
                        存在
669                 break;
670             }
671             printf("Input the value key and others :");
672             scanf("%d%s", &value.key, value.others);
673             getchar();
674             status p = PutVex(G, u, value);
675             if (p == OK)
676                 printf("Put vex successfully \n"); // 修改顶点
                        信息
677             else
678                 printf("Failed \n");
679             break;
680         }
681         case 5: {
682             KeyType u;
683             printf("Input the key :");
684             scanf("%d", &u);
685             getchar();
686             int p = FirstAdjVex(G, u);
687             if (p != -1)
688                 printf("The first adjVex of u is (%d,%s) \n",
                        G.vertices[p].data.key,
689                         G.vertices[p].data.others); // 查找第一
                            个邻接点
690             else
691                 printf("Failed \n");
692             break;
693         }
694         case 6: {
```

```
695             KeyType v, w;
696             printf("Input the key of v and w :");
697             scanf("%d%d", &v, &w);
698             getchar();
699             int p = NextAdjVex(G, v, w);
700             if (p == -1)
701                 printf("Failed \n");
702             else
703                 printf("The next vex of w to v is (%d,%s) \n",
                        G.vertices[p].data.key,
704                         G.vertices[p].data.others); // 查找下一
                            个邻接点
705             break;
706         }
707         case 7: {
708             VertexType v;
709             printf("Input the key and others of v to insert :"
                    );
710             scanf("%d%s", &v.key, v.others);
711             getchar();
712             if (InsertVex(G, v) == OK)
713                 printf("Insert vex successfully \n"); // 插入
                        顶点
714             else
715                 printf("Failed \n");
716             break;
717         }
718         case 8: {
719             KeyType v;
720             printf("Input the key to delete :");
721             scanf("%d", &v);
722             getchar();
723             if (DeleteVex(G, v) == OK)
724                 printf("Delete vex successfully \n"); // 删除
                        顶点
```

```
725             else
726                 printf("Failed \n");
727             break;
728         }
729         case 9: {
730             KeyType v, w;
731             printf("Input the key of v w to insert :");
732             scanf("%d%d", &v, &w);
733             getchar();
734             if (InsertArc(G, v, w) == OK)
735                 printf("Insert arc successfully \n"); // 插入
                        边
736             else
737                 printf("Failed \n");
738             break;
739         }
740         case 10: {
741             KeyType v, w;
742             printf("Input the key of v w arc to delete :");
743             scanf("%d%d", &v, &w);
744             getchar();
745             if (DeleteArc(G, v, w) == OK)
746                 printf("Delete arc successfully \n"); // 删除
                        边
747             else
748                 printf("Falied \n");
749             break;
750         }
751         case 11: {
752             if (!DFSTraverse(G, visit))
753                 printf("Failed \n"); // 深度优先遍历
754             else
755                 printf("\n");
756             break;
757         }
```

```
758         case 12: {
759             if (!BFSTraverse(G, visit))
760                 printf("Failed \n"); // 广度优先遍历
761             else
762                 printf("\n");
763             break;
764         }
765         case 13: {
766             if (G.vexnum <= 0)
767                 printf("The graph does not exist \n");
768             else {
769                 VertexType v;
770                 int k;
771                 printf("Input the key:");
772                 scanf("%d", &v.key);
773                 getchar();
774                 int find = 0;
775                 for (int i = 0; i < G.vexnum; i++) {
776                     if (v.key == G.vertices[i].data.key) {
777                         find = 1;
778                     }
779                 }
780                 if (find == 0) {
781                     printf("The key does not exist \n"); // 顶
                            点不存在
782                     break;
783                 }
784                 printf("Input the distance :");
785                 scanf("%d", &k);
786                 status s = VerticesSetLessThanK(G, v, k);
787                 if (s == ERROR)
788                     printf("Failed \n");
789             }
790             break;
791         }
```

```
792         case 14: {
793             VertexType v, w;
794             printf("Input the key of v w :");
795             scanf("%d%d", &v.key, &w.key);
796             getchar();
797             int pathlen = ShortestPathLength(G, v, w);
798             if (pathlen == -1)
799                 printf("No path exists from %d to %d\n", v.key
                        , w.key); // 无路径
800             break;
801         }
802         case 15: {
803             if (G.vexnum <= 0)
804                 printf("The graph does not exist \n");
805             else {
806                 int count = ConnectedComponentNums(G);
807                 if (count > 0)
808                     printf("The number of connected component
                            is %d \n", count); // 连通分量数
809                 else
810                     printf("Failed \n");
811             }
812             break;
813         }
814         case 16: {
815             if (SaveGraph(G, filename) == OK)
816                 printf("Save successfully \n"); // 保存图到文
                        件
817             else
818                 printf("Failed \n");
819             break;
820         }
821         case 17: {
822             char name[20];
823             printf("Input the graph name to load : ");
```

```
824             scanf("%s", name);
825             getchar();
826             int find = 0;
827             for (int i = 0; i < graphs.amount; i++) {
828                 if (strcmp(name, graphs.elem[i].name) == 0) {
829                     printf("The name has already existed \n");
830                     find = 1;
831                 }
832             }
833             if (find == 1)
834                 break;
835             ALGraph g;
836             if (LoadGraph(g, filename) == OK)
837                 printf("Load successfully \n"); // 加载图
838             else
839                 printf("Failed \n");
840             printf("Do you want to check the graph(Y/N) : ");
841             char opt;
842             scanf("%c", &opt);
843             getchar();
844             if (opt == 'Y' || opt == 'y') {
845                 for (int i = 0; i < g.vexnum; i++) {
846                     printf("%d %s ", g.vertices[i].data.key, g
                            .vertices[i].data.others);
847                     ArcNode *p = g.vertices[i].firstarc;
848                     while (p) {
849                         printf("%d ", p->adjvex);
850                         p = p->nextarc;
851                     }
852                     printf("\n");
853                 }
854             }
855             graphs.elem[graphs.amount].G = g;
856             strcpy(graphs.elem[graphs.amount++].name, name);
857             G = g;
```

```
858             printf("\n");
859             break;
860         }
861         case 18: {
862             printf("Input the name to add : ");
863             char name[20];
864             scanf("%s", name);
865             getchar();
866             int find = 0;
867             for (int i = 0; i < graphs.amount; i++) {
868                 if (strcmp(name, graphs.elem[i].name) == 0) {
869                     printf("The name has already existed \n");
870                     find = 1;
871                 }
872             }
873             if (find == 1)
874                 break;
875             if (AddGraph(graphs, name) == OK) {
876                 VertexType V[100];
877                 KeyType VR[100][2];
878                 int vexnum = 0, arcnum = 0;
879                 printf("Input the vexs and keys definition :")
                        ;
880                 while (scanf("%d%s", &V[vexnum].key, V[vexnum
                        ].others)) {
881                     if (V[vexnum].key == -1)
882                         break;
883                     vexnum++;
884                 }
885                 while (scanf("%d%d", &VR[arcnum][0], &VR[
                        arcnum][1])) {
886                     if (VR[arcnum][0] == -1)
887                         break;
888                     arcnum++;
889                 }
```

```
890                 if (CreateGraph(graphs.elem[graphs.amount -
                        1].G, V, VR) == OK) {
891                     printf("Add successfully \n"); // 添加图
892                     G = graphs.elem[graphs.amount - 1].G;
893                 } else
894                     printf("Failed \n");
895             }
896             break;
897         }
898         case 19: {
899             printf("Input the name to remove : ");
900             char name[20];
901             scanf("%s", name);
902             getchar();
903             if (RemoveGraph(graphs, name) == OK)
904                 printf("Remove successfully \n"); // 移除图
905             else
906                 printf("Failed \n");
907             break;
908         }
909         case 20: {
910             printf("Input the name to locate : ");
911             char name[20];
912             scanf("%s", name);
913             getchar();
914             int pos = LocateGraph(graphs, name);
915             if (pos)
916                 printf("The position is %d \n ", pos); // 查找
                        图在集合中的位置
917             else
918                 printf("The graph does not exist \n");
919             break;
920         }
921         case 21: {
922             printf("Input the name to switch :");
```

```
923                 char name[20];
924                 scanf("%s", name);
925                 getchar();
926                 if (SwitchGraph(graphs, name, G) == OK)
927                     printf("Switch successfully \n"); // 切换当前
                        图
928                 else
929                     printf("Failed \n");
930                 break;
931             }
932             case 22: {
933                 ShowAllGraphs(graphs); // 显示所有图及内容
934                 break;
935             }
936             default:
937                 break;
938             }
939         }
940         printf("Welcome to the system next time \n"); // 退出提示
941 }
```