# 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** CS2106

**学　　号** U202115514

**姓　　名** 杨明欣

**指导教师** 周全

**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

线性表是最常用而且最简单的一种数据结构。简言之，一个线性表是 n 个数据元素的有限序列。线性表的存储结构分为顺序存储和链式存储。线性表的数据集合为 {a1,a2,⋯,an}，假设每个元素的类型均为 ElemType。其中，除第一个元素 a1 外，每一个元素有且只有一个直接前驱元素，除了最后一个元素 an 外，每一个元素有且只有一个直接后继元素。数据元素之间的关系是一对一的关系。

在较复杂的线性表中，一个数据元素可以由若干个数据项组成。在这种情况下，常把数据元素称为记录，含有大量记录的线性表又称为文件。

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。程序实现线性表的初始化、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。可以选择以文件的形式进行存储和加载，将生成的线性表存入到相应的文件中，也可以从文件中获取线性表进行操作。同时实现多线性表管理，完成多线性表的添加、删除、选择等操作。

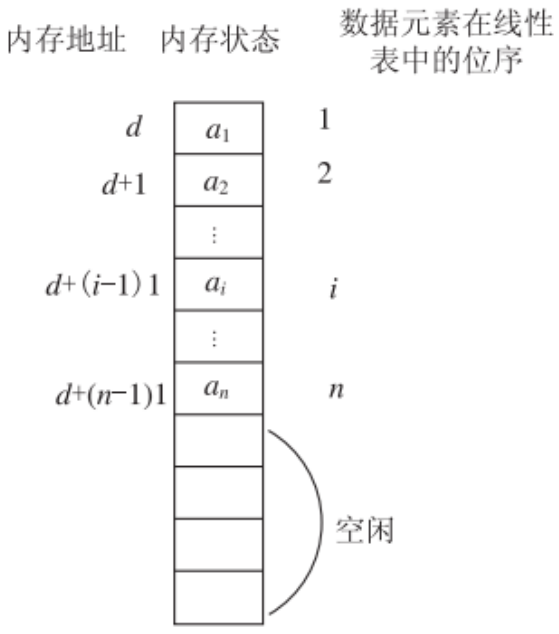


图 1-1 线性表存储结构

## 1.2 系统设计

### 1.2.1 头文件和预定义

1、头文件

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.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.
#define MAXlength 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（顺序结构）的定义
    ElemType * elem;
    int length;
    int listsize;
}SqList;
SqList L;

typedef struct{  //线性表的集合类型定义
    struct { char name[30];
        SqList L;
    } elem[11];
    int length;
```

```
15  }LISTS;
16  LISTS  Lists;        //线性表集合的定义Lists
```

### 1.2.2 基本功能函数

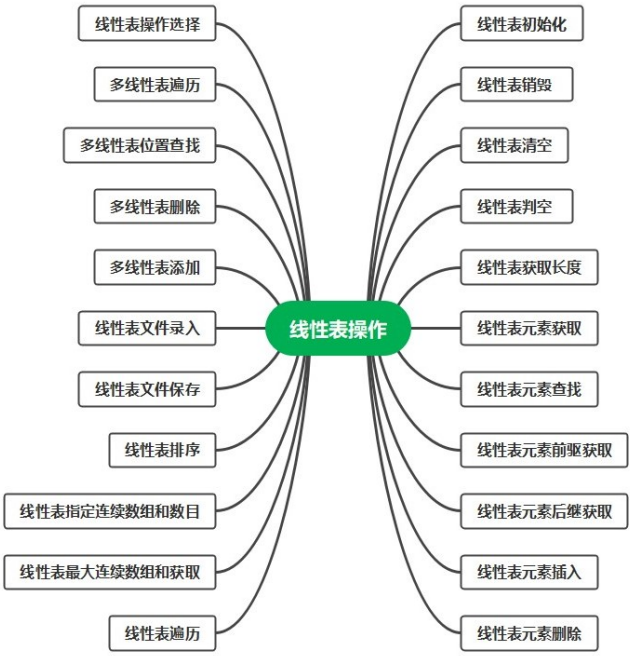


图 1-2　系统整体功能设计图

线性表的逻辑结构定义如下：

ADT List{

数据对象：D= $\{a_i|a_i \in ElemSet, i = 1, 2, \cdots, n, n{\geq}0\}$

数据关系：R1=$\{< a_{i-1}, a_i > |a_{i-1}, a_i \in D, i = 2, \cdots, n\}$

}

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) 遍历所有表：函数名称是 TraverseList(Lists,visit())；初始条件是 Lists 已存在；操作结果是对所有的表遍历并输出。
   (e) 选择表:函数名称是 SelectList (Lists, i);初始条件是 Lists 已存在,1≤i≤Lists.Length+1；操作结果是将 Lists 中逻辑索引为 i 的线性表选择为当前处理的线性表，以便后续再调用其他函数对该表进行操作。

### 1.2.4 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。

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

当前线性表名称和初始化情况。

输入 1～22 可以调用上述的 22 个函数，对线性表或多线性表进行操作；输入 0 时退出系统。

## 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，通过 system(”cls”) 语句实现清屏操作，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 0~22，其中 1~22 分别代表线性表的一个基本运算，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 数据结构设计

1. 线性表：线性表中用数组 elem 储存数据，定义 length 变量储存线性表长度，定义 listsize 变量表示线性表的大小。
2. 多线性表的管理表：定义 length 变量表示管理表的长度。定义 elem 数组顺序储存线性表及其名称。

### 1.3.3 函数思想及实现

**说明：无特殊说明情况下，每一函数均需判断线性表是否存在之后，再对线性表进行相应的操作。**

线性表基本功能函数的实现：

**1.** status InitList (SqList &L)

输入：线性表（引用参数）

输出：函数运行状态

函数思想描述：将线性表初始化过程写成函数，其中传入函数的参数是全局定义的结构体类型变量 L 的引用，在函数中，首先使用 malloc 函数为线性表分配 LISTSIZE 大小的连续内存空间，将首地址赋值给 L.elem，由于线性表的长度为 0，将 L.length 初始化为 0，即完成了线性表的初始化。

**2.** status DestroyList (SqList &L)

输入：线性表（引用参数）

输出：函数运行状态

函数思想描述：将销毁线性表的过程写成函数，其中将定义的结构体类型变量 L 的引用作为函数参数。在函数中，首先使用 free 函数释放掉以 L.elem 为首地址的连续内存空间，再将 L.length 重新赋值为 0，完成线性表的销毁。

**3.** status ClearList (SqList &L)

输入：线性表（引用参数）

输出：函数运行状态

函数思想描述：将清空线性表的过程写成函数，其中将定义的结构体类型变量 L 的引用作为函数参数。在函数中，无需对于 L.elem 内的元素进行处理，直接将 L.length 赋值为 0 即可，即完成了线性表的清空。

**4.** status ListEmpty (SqList L)

输入：线性表（传值调用）

输出：函数运行状态

函数思想描述：将求判断线性表是否为空写成函数，其中将定义的结构体类型变量 L 的值作为函数的参数，在函数中，如果线性表不存在，返回 INFEASIBLE。如果线性表存在，则判断如果 L.length 为 0，返回 FALSE，否则返回 TRUE。

**5.** int ListLength (SqList L)

输入：线性表（传值调用）

输出：整型变量（含义为表 L 中元素数目）

函数思想描述：将求表长过程写成函数，其中将定义的结构体类型变量 L 的引用作为函数的参数，在函数中，直接返回 L.length 即为所求线性表的表长。

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

输入：线性表（传值调用），整型变量（元素逻辑索引），类型变量（引用参数，用于存储元素）

输出：函数运行状态

函数思想描述：将获得线性表元素写成函数，其中函数的参数是结构体类型变量 L 以及数据元素的逻辑索引 i，首先需要判断输入的元素序号 i 的合法性（1≤i≤L.Length）。因为采取的是线性存储结构，所以直接通过访问数组元素的方式即 L.elem[i-1] 来获取元素存储到元素 e 中。

**7.** status LocateElem (SqList L, ElemType e, int (*compare)(SqList, int, ElemType))

输入：线性表（传值调用），类型变量（含义为待查找的元素），函数指针

输出：函数运行状态

函数思想描述：将查找线性表特定元素写成函数，其中函数的参数是定义的结构体类型变量 L 以及待查找的数据元素值，通过遍历线性表中的每一个元素查找值为 e 的元素, 即 compare(L,i,e) 为 1，返回该元素的逻辑索引（即位序），如果元素不存在，则返回 0。

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

输入：线性表（传值调用），类型变量（传值调用），类型变量（引用参数，用于存储元素）

输出：函数运行状态

函数思想描述：将获得前驱过程写成函数，函数的参数是结构体类型变量 L 以及特定数据元素的值，存储前驱元素的变量作为另一个参数。首先调用 LocateElem(L, e) 函数判断该线性表中特定数据元素的位序并转换为数组下标 i，判断数组下标是否合法 (1≤i≤L.Length-1)，则将元素的前驱赋值给 pre 变量，返回 OK；否则返回 ERROR。

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

输入：线性表（传值调用），类型变量（传值调用），类型变量（引用参数，用于存储元素）

输出：函数运行状态

函数思想描述：将获得后继过程写成函数，函数的参数是结构体类型变量 L 以及特定数据元素的值，存储后继元素的变量作为另一个参数。首先调用 LocateElem(L, e) 函数判断该线性表中特定数据元素的位序并转换为数组下标 i，判断数组下标是否合法 (0≤i≤L.Length-2)，则将元素的后继赋值给 pre 变量，返回 OK；否则返回 ERROR。

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

输入：线性表（传值调用），整型变量（元素位序），类型变量（含义为待插入的元素值）

输出：函数运行状态

函数思想描述：将插入元素的过程写成函数，函数的参数是结构体类型变量 L，插入元素的位序 i 和待插入的元素的值 e。在函数中，首先判断插入位置的合法性 (1≤i≤L.Length+1)。其次判断当前线性表存储空间是否已满，即判断 L.length 和 L.listsize，如果线性表已满，则使用 realloc 函数为线性表重新分配空间。插入元素时，先将位序为 i 后的元素向后移动一个内存单元，然后将 e 插入进去，返回 OK。

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

输入：线性表（传值调用），整型变量（元素位序），类型变量（引用参数，用于存储被删除元素）

输出：函数运行状态

函数思想描述：将删除线性表中元素写成函数，函数的参数是结构体类型变量 L，删除元素的位序 i 和存储被删除元素的变量。首先判断位序的合法性 (1≤i≤L.Length)，将被删除的元素赋值给 e，然后将位序为 i 后的元素向后移动一个内存单元，返回 OK。

**12.** status ListTraverse (SqList L，int (*visit)(SqList,int))

输入：线性表

输出：函数运行状态

函数思想描述：将遍历线性表写成一个函数，函数的参数是结构体类型变量 L，遍历数组中的每一个元素并调用 visit() 函数，如果线性表为空，则输出线性表为空。

附加功能函数的实现：

**13.** status MaxSubArray(SqList L)

输入：线性表（传值调用）

输出：整型变量（含义为连续数组和的最大值）

算法描述：函数通过将线性表队列化，用 max 记录前缀和，通过每次判断前缀是否为负，若为负，则从头开始依次出队列直至前缀为正，保证遍历每一组和为正的连续数组，通过 temp 记录连续数组和的最大值，最后返回 temp。

时间复杂度：O($n^2$)

空间复杂度：O(1)

**14.** status SubArrayNum(SqList L, int k)

输入：线性表（传值调用）、整型变量（含义为连续数组和）

输出：整型变量（线性表中连续数组和为 k 的连续数组数目）

函数思想描述：通过牺牲空间换取时间的算法思想，额外增添一前缀和数组，通过前缀和数组间的差值可以计算连续数组和，使用 count 计数连续数组和为 k 的数组数目。此算法可以将时间复杂度降到 O($n^2$)，具体算法流程图如下：

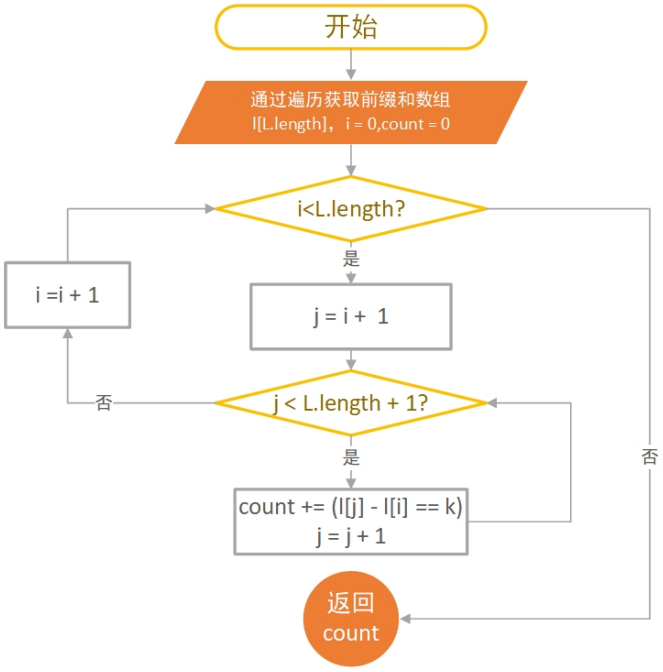


图 1-3　连续数组和数目程序流程图

时间复杂度：O($n^2$)

空间复杂度：O(n)

**15.** status sortList(SqList& L)

输入：线性表（引用参数）

输出：整型变量（线性表中连续数组和为 k 的连续数组数目）

函数思想描述：函数实现线性表排序。该函数通过归并排序算法将线性表进行排序。

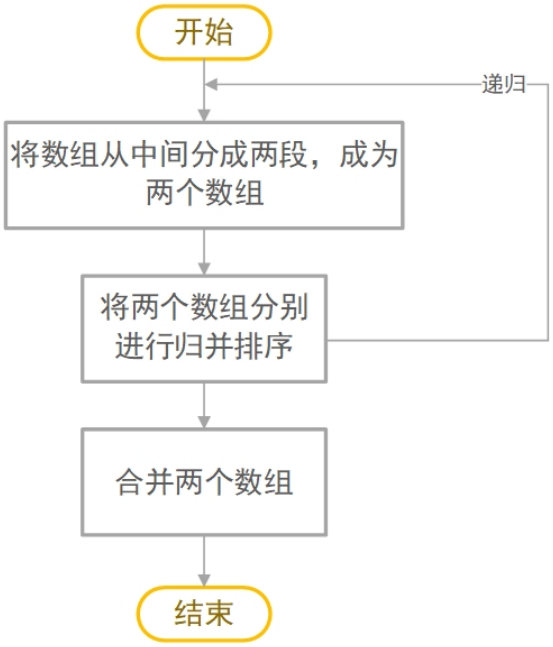


图 1-4　归并排序文字流程图

时间复杂度：O($nlogn$)

空间复杂度：O(n)

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

输入：线性表 (传值调用)，字符串变量（含义为文件名称）

输出：函数运行状态

函数思想描述：将保存线性表为文件写成函数，函数的参数是结构体类型变量 L 和文件名称 FileName。首先判断线性表是否存在，如果存在，然后判断文件内是否有内容，文件为空或不存在时则打开或创建文件，然后调用 fwrite 函数将表中的所有元素写入该文件中，之后关闭文件。

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

输入：线性表 (传值调用)，字符串变量（含义为文件名称）

输出：函数运行状态

函数思想描述：将读取文件中的线性表写成函数，函数的参数是结构体类型变量 L 和文件名称 FileName。首先判断 L 是否存在，如果线性表 L 存在，表示 L 中已经有数据，读入数据会覆盖原数据造成数据丢失，故只有 L 不存在时才可以继续操作。然后打开文件，调用 fread 函数将所有元素写入表中，之后关闭文件。

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

输入：顺序表数组，字符串变量（含义为线性表名称）

输出：函数运行状态

函数思想描述：将增加线性表写成函数，函数的参数是 LISTS 结构体类型变量和线性表名称 ListName。Lists 是一个以顺序表形式管理的线性表的集合，在集合中增加一个新的空线性表，并将表名称存储在该表的 name 分量当中。在添加线性表之前，应当判断表名是否唯一。

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

输入：顺序表数组，字符串变量（含义为线性表名称）

输出：函数运行状态

函数思想描述：将移除线性表写成函数，函数的参数是 LISTS 结构体类型变量和线性表名称 ListName。在 Lists 中查找名称为 ListName 的线性表，如果可以找到，则调用 DestroyList 函数将其删除。

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

输入：顺序表数组，字符串变量（含义为线性表名称）

输出：整型变量（线性表的位序）

函数思想描述：将查找线性表写成函数，函数的参数是 LISTS 结构类型变量和线性表名称 ListName。在 Lists 中查找名称为 ListName 的线性表，如果可以找到，返回线性表的逻辑索引，否则返回 0。

**21.** status TraverseList(LISTS Lists)

输入：顺序表数组

输出：函数运行状态

函数思想描述：函数的参数是 LISTS 结构类型变量。调用 TraverseList 函数遍历线性表集合中每一个线性表，输出每一个线性表的名称。

时间复杂度：O（mn）

**22.** status SelectList(LISTS Lists, int i)

输入：顺序表数组，整型变量（含义为线性表的逻辑索引）

输出：函数运行状态

函数思想描述：将选择线性表写成函数，函数的参数是 LISTS 结构类型变量和逻辑索引。将 Lists 中逻辑索引为 i 的线性表赋值给 L，后续可调用其它函数时将对此表进行操作。

## 1.4 系统测试

系统菜单整体布局如图：

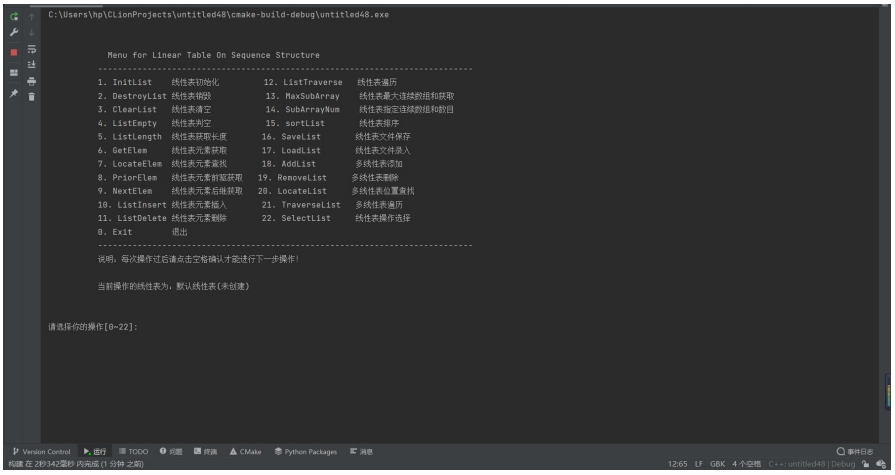


图 1-5 菜单

测试集如下：

测试集 1：线性表中存有元素 1 3 5 7 9

测试集 2：线性表中存有元素 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

测试集 3：线性表集合中有两个线性表 FirstList: 1 3 5 7 9;SecondList: 2 4 6 8 10

测试集 4：线性表中存有元素 -2 1 -3 4 -1 2 1 -5 4

### 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 |

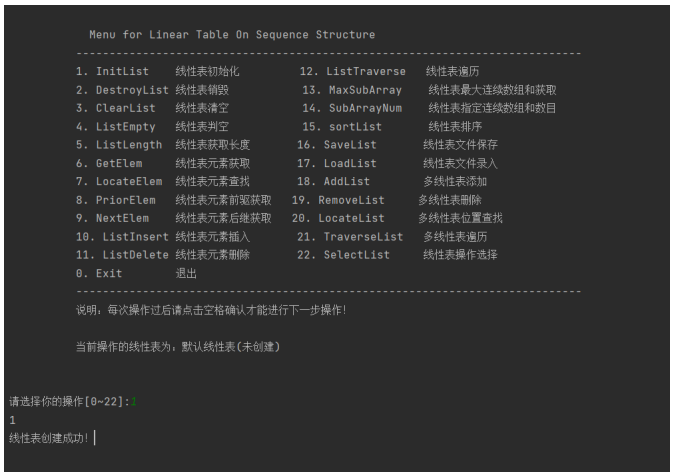


图 1-6　测试 1 运行结果

**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 |

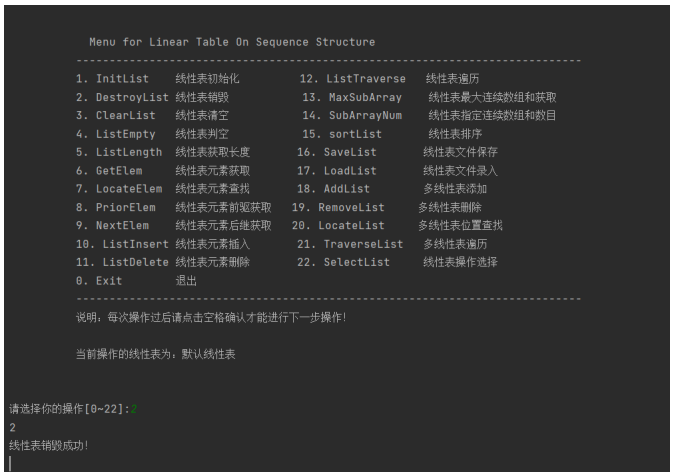


图 1-7　测试 2 运行结果

**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 |

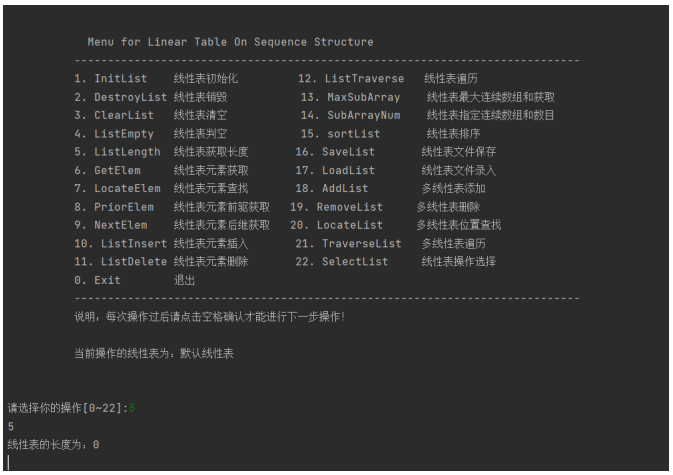


图 1-8　测试 3 运行结果

**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 |

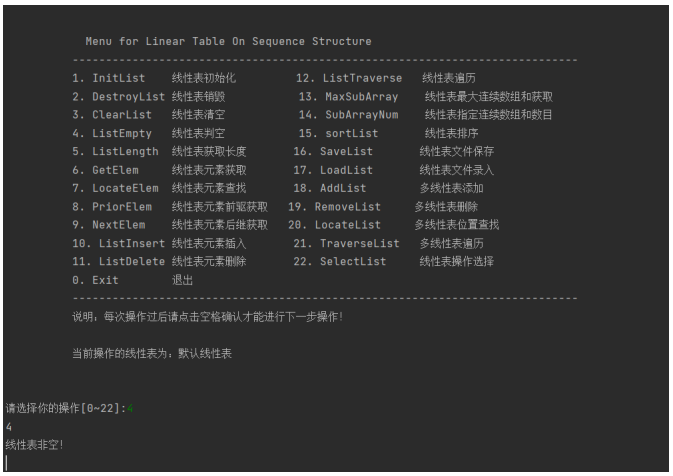


图 1-9　测试 3 运行结果

**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 5 | 一致 |


图 1-10　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 3，4，5：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 3 | 一致 |
| 2 | 6→5 | 线性表中第 5 个元素为 9 | 一致 |
| 3 | 6→-1 | 输入的逻辑索引不合法！ | 一致 |
| 4 | 6→0 | 输入的逻辑索引不合法！ | 一致 |
| 5 | 6→6 | 输入的逻辑索引不合法！ | 一致 |

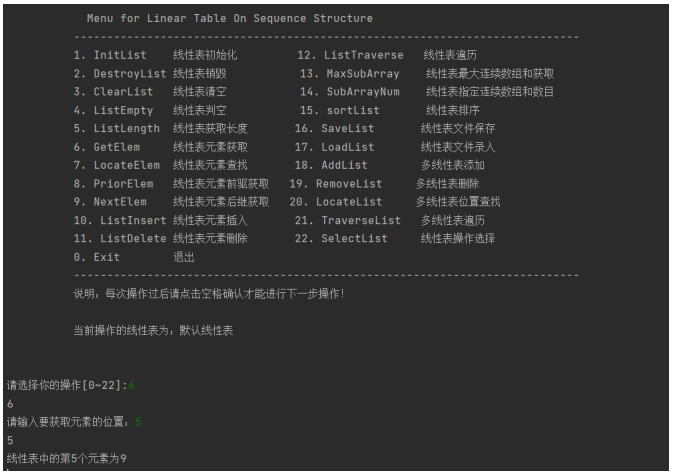


图 1-11　测试 2 运行结果

**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 1，2：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 |
| 2 | 7→9 | 该元素存在且元素逻辑索引为：5 | 一致 |
| 3 | 7→6 | 输入的元素不存在！ | 一致 |
| 4 | 7→-1 | 输入的元素不存在！ | 一致 |

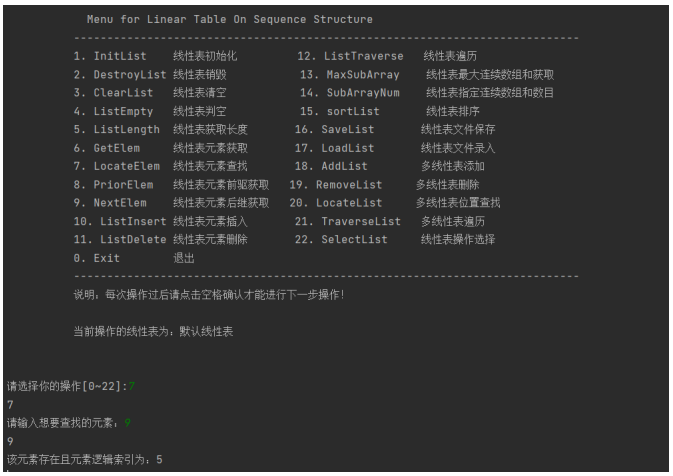


图 1-12　测试 2 运行结果

**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 1，2：将测试函数能否正确找到前驱；

测试 3：将测试函数能否正确判断第一个元素没有前驱；

测试 4：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→3 | 该元素存在且前驱元素为：1 | 一致 |
| 2 | 8→9 | 该元素存在且前驱元素为：7 | 一致 |
| 3 | 8→1 | 线性表中该元素不存在前驱！ | 一致 |
| 4 | 8→6 | 线性表中该元素不存在！ | 一致 |

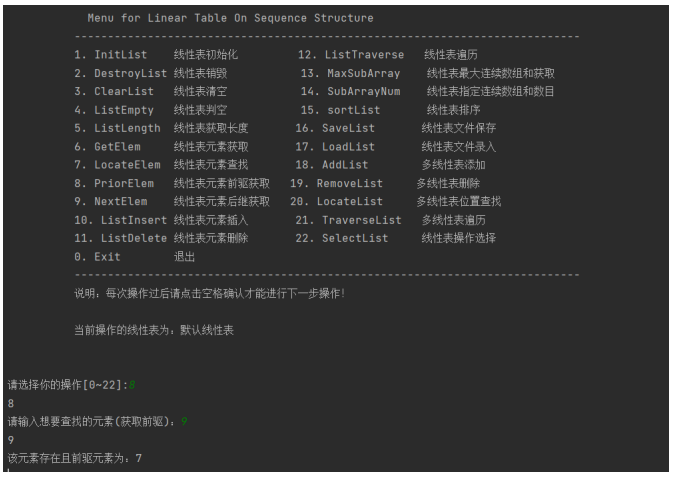


图 1-13　测试 2 运行结果

**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 1，2：将测试函数能否正确找到后继；

测试 3：将测试函数能否正确判断最后一个元素没有后继；

测试 4：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→3 | 线性表中该元素的后继为 5 | 一致 |
| 2 | 9→7 | 线性表中该元素的后继为 9 | 一致 |
| 3 | 9→9 | 线性表中该元素不存在后继！ | 一致 |
| 4 | 9→6 | 线性表中该元素不存在！ | 一致 |

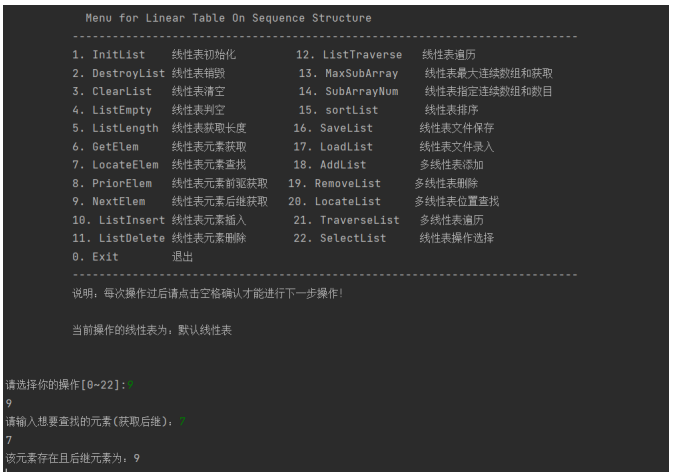


图 1-14　测试 2 运行结果

**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 3 5 7 9），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 4 在测试集 2 的基础上进行，测试当线性表已满状态时，能否正确插入元素；

测试 5 将在销毁线性表之后的情况下进行，测试函数能否正确判断线性表的存在性；

测试 6 将在清空线性表之后的情况下进行，测试函数能否正确判断非法的插入位置。

| 测试编号 | 测试输入 | 预期结果 | 实际运行结果 |
|---|---|---|---|
| 1 | 10→1<br>1→10→2<br>3→10→3<br>5→10→4<br>7→10→5 9 | 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 |
| 4 | 10→ 测试集<br>2→10→21 21 | 线性表插入成功！ | 一致 |
| 5 | 2→10→1 1 | 线性表不存在，插入失败！ | 一致 |
| 6 | 3→10→3 1 | 插入位置不合法，线性表插入失败！ | 一致 |

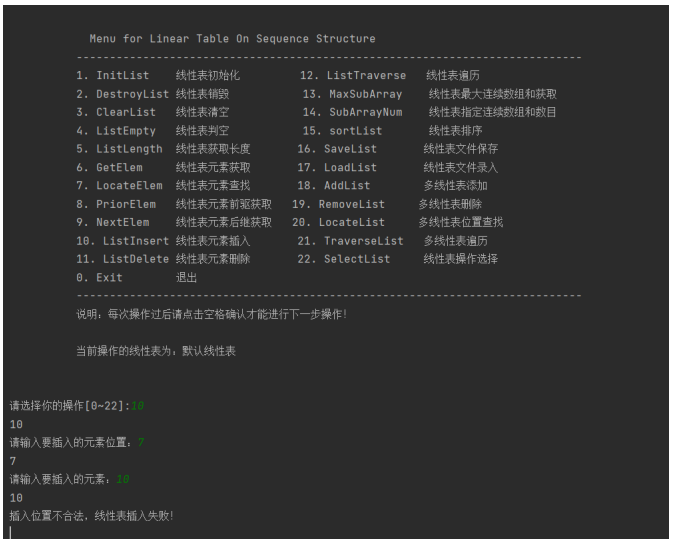


图 1-15　测试 2 运行结果

**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 3：在测试 2 的基础上完成，将测试函数能否正确删除元素，采用遍历和求表长的方式检验正确性；

| 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 5 7 9 | 一致 |
| 3 | 11→1→5 | 元素已删除！线性表的长度为 3！ | 一致 |

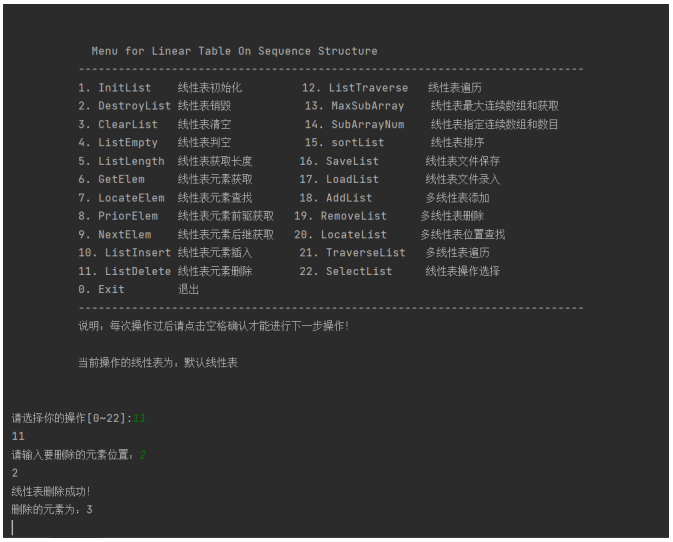


图 1-16　测试 2 运行结果

**12.** ListTraverse 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 3 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 | 12 | 线性表不存在！不能遍历 | 一致 |
| 2 | 3→12 | 线性表是空表 | 一致 |
| 3 | 12 | 1 3 5 7 9 | 一致 |

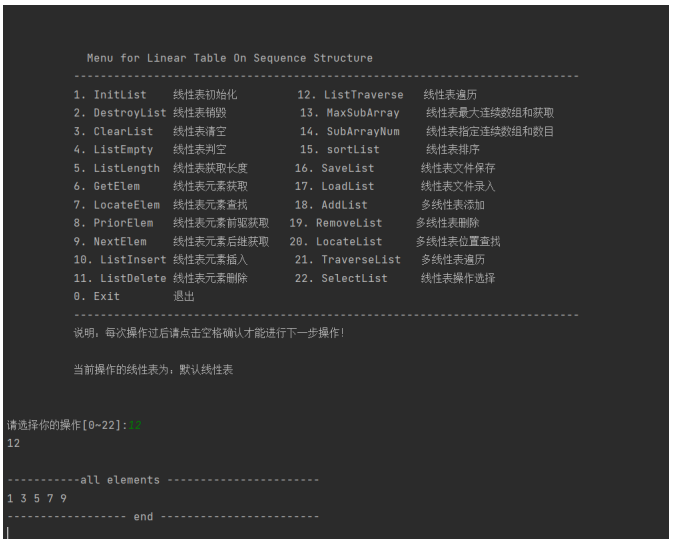


图 1-17　测试 3 运行结果

### 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 4 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 |

**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、3、4：在测试集 4 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：5 | 一致 |
| 3 | 14→5 | 和为数 5 的连续数组数目为：2 | 一致 |
| 4 | 14→7 | 和为数 7 的连续数组数目为：0 | 一致 |

**15.** sortList 测试

测试 1：在线性表为空的情况下进行，测试函数能否正确判断线性表为空；

测试 2：在线性表不存在的情况下进行，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 | 3→15 | 线性表是空表！ | 一致 |
| 2 | 2→15→12 | -5 -3 -2 -1 1 1 2 4 4 | 一致 |

**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 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：在线性表不存在的情况下进行，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 | 16→1.txt | 文件保存成功！ | 一致 |
| 2 | 16→1.txt | 该文件已有内容，不能读入！ | 一致 |
| 3 | 2→16→1.txt | 线性表不存在! 文件保存失败！ | 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 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→17→1.txt | 文件录入成功！遍历：1 3 5 7 9 | 一致 |
| 2 | 17→1.txt | 线性表存在! 文件录入失败！ | 一致 |

**18.** AddList 测试

测试 1：将测试函数能否正确添加线性表；

测试 2：在测试 1 的基础上进行，当线性表表名重复时，测试函数能否给出正确的判断。

测试 3：构建测试集 3，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→FirstList | FirstList 已成功添加！ | 一致 |
| 2 | 18→FirstList | 该名称的线性表已经存在！ | 一致 |
| 3 | 21 | FirstList SecondList | 一致 |

**19.** RemoveList 测试

本函数的测试在测试集 3 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→FirstList | FirstList 已成功删除！ | 一致 |
| 2 | 19→ThirdList | 线性表不存在！ | 一致 |

**20.** LocateList 测试

本函数的测试在测试集 3 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→FirstList | 该线性表的逻辑索引为：1 | 一致 |
| 2 | 20→ThirdList | 线性表查找失败！ | 一致 |

**21.** TraverseList 测试

测试 1：在测试集 3 的基础上进行，测试函数能否正确遍历各个线性表；

测试 2：在空集合的基础上进行，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 | （测试集 3）21 | FirstList SecondList | 一致 |
| 2 | 21 | 多线性表为空！ | 一致 |

**22.** SelectList 测试

输入要求：输入线性表集合中线性表对应的逻辑索引。

测试 1,2：在测试集 3 的情况下进行，测试函数能否正确判断非法的位序；

测试 3：在测试集 3 的情况下进行，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 | 22→0 | 线性表选取失败！ | 一致 |
| 1 | 22→3 | 线性表选取失败！ | 一致 |
| 2 | 22→1 | 线性表已选取成功！ | 一致 |

测试小结

22 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，没有测试线性表不存在的情况，同时对于附加功能函数没有具体给测试后的控制台界面。

## 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。其它有关类型和常量的定义和引用详见文献 [1] 的 p10。

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。基于顺序表实现的二叉树管理，其物理结构的参考设计如图 3-1 所示。

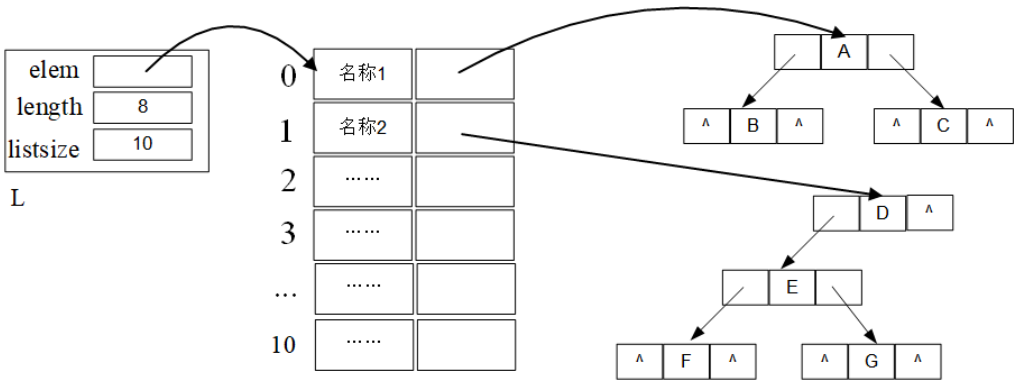


图 2-1 二叉树物理结构示意图

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

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.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 MAXlength 10
```

3、类型表达式

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

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

typedef struct{  //线性表的集合类型定义
    struct { char name[30];
        BiTree L;
    } elem[11];
```

```
17      int length;
18  }TREES;
19  TREES trees;        //线性表集合的定义TREES
```

### 2.2.2 基本功能函数

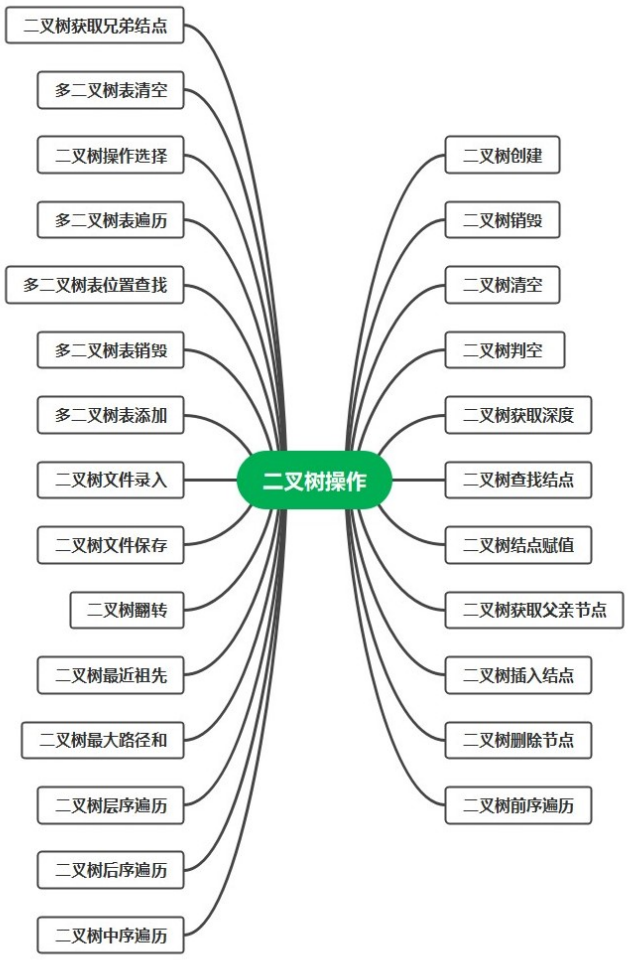


图 2-2 系统整体功能设计图

二叉树的逻辑结构如下：

ADT Tree {

数据对象 D: D 是具有相同特性的数据元素的集合。

数据关系 R: 若 D 为空集, 则称为空树;

若 D 仅含一个数据元素, 则 R 为空集, 否则 R={H}, H 是如下二元关系:

(1) 在 D 中存在惟一的称为根的数据元素 root, 它在关系 H 下无前驱;

(2) 若 D - {root} $\neq \Phi$ , 则存在 D - {root} 的一个划分 $D_1, D_2, \cdots, D_m$ $(m > 0)$ , 对任意 $j \neq k$ $(1 \leq j, k \leq m)$ 有 $D_j \cap D_k = \Phi$ , 且对任意的 i $(1 \leq i \leq m)$ , 惟一存在数据元素 $x_i \in D_i$ , 有 <root, $x_i$> $\in H$ ;

(3) 对应于 D-{root} 的划分, H - <root, $x_1$>,$\cdots$,<root, $x_0$> 有惟一的一个划分 $H_1, H_2, \cdots, H_m$ $(m > 0)$ , 对任意 $j \neq k$ $(1 \leq j, k \leq m)$ 有 $H_j \cap H_k = \Phi$ , 且对任意 i $(1 \leq i \leq m)$, $H_i$ 是 $D_i$ 上的二元关系, $(D_i, \{H_i\})$ 是一棵符合本定义的树, 称为根 root 的子树。

}

依据最小完备性和常用性相结合的原则，以函数形式定义了二叉树的创建二叉树、销毁二叉树、清空二叉树、判定空二叉树和求二叉树深度等 14 种基本运算。具体运算功能定义和说明如下：

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，如带空子树的二叉树前序遍历序列、或前序 + 中序、或后序 + 中序；操作结果是按 definition 构造二叉树 T；
   注：①要求 T 中各结点关键字具有唯一性。后面各操作的实现，也都要满足一棵二叉树中关键字的唯一性，不再赘述；②CreateBiTree 中根据 definition 生成 T，不应在 CreateBiTree 中输入二叉树的定义。
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 已存在，e 是和 T 中结点关键字类型相同的给定值；操作结果是返回查找到的结点指针，如无关键字为 e 的结点，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给定值；操作结果是关键字为 e 的结点赋值为 value；

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给定值；操作结果是返回关键字为 e 的结点的（左或右）兄弟结点指针。若关键字为 e 的结点无兄弟，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给定值，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；
   特殊情况，c 插入作为根结点，可以考虑 LR 为-1 时，作为根结点插入，原根结点作为 c 的右子树。
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给定值。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())；初始条件是二叉树 T 存在，Visit 是一个函数指针的形参（可使用该函数对结点操作）；操作结果：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。
    注：前序、中序和后序三种遍历算法，要求至少一个用非递归算法实现。
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（可使用该函数对结点操作）；操作结果是中序遍历 t，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（可使用该函数对结点操作）；操作结果是后序遍历 t，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 是对结点操作的应用函数；操作结果是层序遍历 t，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. 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(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。
   (b) 文件读出：函数名称是 LoadList(T,FileName)；初始条件是二叉树 T 不存在；操作结果是将文件 FileName 中的元素读到表 T 中。
5. 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、移除等功能。
   (a) 增加二叉树：函数名称是 AddList(trees, ListName)；初始条件是名称为 ListName 的二叉树不存在于二叉树集合中；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) 移除二叉树：函数名称是 RemoveList(trees, ListName)；初始条件是名称为 ListName 的二叉树存在于二叉树集合中；操作结果是将该线性表移除。
   (c) 查找二叉树：函数名称是 LocateList(trees, ListName)；初始条件是名称为 ListName 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 Lists 中的逻辑索引。
   (d) 遍历所有表：函数名称是 TraverseList(trees,visit())；初始条件是 trees 已存在；操作结果是对所有的表遍历并输出。
   (e) 选择表:函数名称是 SelectList (trees, i);初始条件是 trees 已存在,1≤i≤trees.Length+1；操作结果是将 treea 中逻辑索引为 i 的二叉树选择为当前处理的二叉树，以便后续再调用其他函数对该树进行操作。

### 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 1~26 可以调用上述的 26 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。

输入 1~ 26 可以调用上述的 26 个函数，对线性表或多线性表进行操作；输入 0 时退出系统。

## 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，通过 system(”cls”) 语句实现清屏操作，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 0~26，其中 1~26 分别代表二叉树的一个基本运算，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二叉树（未创建）进行操作，后续可增加新树并进行选择，实现森林的操作。

### 2.3.2 数据结构设计

1. 二叉树：二叉树中用特殊类型 data 储存数据，其中数据包含关键元素 key 和关键信息 others，定义 BiTNode 类型的 lchild 用于指向左子树结点，rchild 用于指向右子树结点。
2. 多二叉树的管理表：定义 length 变量表示管理二叉树表的长度。定义 elem 数组顺序储存二叉树及其名称。

### 2.3.3 函数思想及实现

**说明：无特殊说明情况下，每一函数均需判断二叉树是否存在之后，再对二叉树进行相应的操作。**

二叉树基本功能函数的实现：

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

输入：二叉树根结点，TELemType 类型数组

输出：函数运行状态

函数思想描述：将创建二叉树写成函数，其中传入函数的参数是二叉树根节点和 TELemType 类型数组。根据带空枝的二叉树先根遍历序列 definition 构造一棵二叉树，将根节点指针赋值给 T 并返回 OK，若关键字不唯一就返回 ERROR。在函数中，首先调用函数 checkKey 判断输入的关键字是否唯一。然后对于 definition 中的结点，若关键字为-1 则创建结束；若关键字为 0，则当前子树创建结束；若关键字大于 0，则创建当前结点，并递归调用本函数依次创建本结点的左子树和右子树，最后创建完好的二叉树。

**2.** status DestroyBiTree(BiTree &T)

输入：二叉树根结点

输出：函数运行状态

函数思想描述：将清空二叉树写成函数，其中传入函数的参数是二叉树根节点指针。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清空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：二叉树根结点

输出：函数运行状态

函数思想描述：将销毁二叉树写成函数，其中传入函数的参数是二叉树根节点指针。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清空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，操作基本与销毁相同，除非包含头结点指向二叉树的根。

**4.** status BiTreeEmpty(BiTree T)

输入：二叉树根结点

输出：函数运行状态

函数思想描述：判断树的根结点是否存在，如果结点为空，则返回 FALSE，否则说明树非空，返回 TRUE。

**5.** int BiTreeDepth(BiTree T)

输入：二叉树根结点

输出：整型变量（含义为二叉树深度）

函数思想概述：将求二叉树深度写成函数，其中传入函数的参数是二叉树根节点指针。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长度的最大值，作为二叉树的深度。

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

输入：二叉树根结点，KeyType 类型（查找结点的关键字）

输出：二叉树结点指针

函数思想描述：将查找二叉树中关键字为特定值的结点写成函数，其中传入函数的参数是二叉树根结点指针和查找结点的关键字。该函数使用递归思想，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：二叉树根结点，KeyType 类型（查找结点的关键字），TElemType 类型变量（存储要替代的结点信息）

输出：函数运行状态

函数思想描述：将实现结点赋值写成函数，其中传入函数的参数是二叉树根结点指针、查找结点的关键字、用以替换的结点值。在函数中，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：二叉树根结点，KeyType 类型（查找结点的关键字）

输出：二叉树结点指针

函数思想概述：将获得当前结点的兄弟结点写成函数。该函数使用了递归思想。在函数中，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：二叉树根结点，KeyType 类型（待插入结点的关键字），整型变量（标志插入左子树或右子树），TElemType 类型变量（存储要插入的结点信息）

输出：函数运行状态

函数思想概述：将插入结点写成函数。在函数中，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：二叉树根结点，KeyType 类型（待删除结点的关键字）

输出：函数运行状态

函数思想概述：将删除结点写成函数。e 是和 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：二叉树根结点，函数指针

输出：函数运行状态

函数思想概述：将先序遍历树写成函数。该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，则首先通过 visit 访问当前结点，然后递归调用本函数先序遍历当前结点的左子树和右子树。

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

输入：二叉树根结点，函数指针

输出：函数运行状态

函数思想概述：将中序遍历树写成函数。该函数用非递归形式实现，调用栈的数据结构和相应操作函数。在函数中，首先定义和初始化栈，并定义 p 为根节点指针。然后进行遍历左子树循环：若 p 不为空树，则将根指针进栈，此时若栈满返回 ERROR，若成功进栈则将 t 移向左子树。若该结点为空表示以栈顶元素为根节点的子树的左子树遍历结束。然后，若栈非空，则弹出栈，并访问该指针，再将该指针移向右子树，循环进行上述操作。此项工作一直循环到栈空且指针 p 为空，这表示无父节点未访问且无右子树未遍历。最终返回 OK。

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

输入：二叉树根结点，函数指针

输出：函数运行状态

函数思想概述：将后序遍历树写成函数。该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：二叉树根结点，函数指针

输出：函数运行状态

函数思想概述：将层序遍历树写成函数。该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：二叉树根结点

输出：整型变量（含义为最大路径和）

函数思想描述：函数实现返回最大路径和的功能。该函数通过不断递归到叶子节点到空树返回 ERROR，然后返回路径上的最大值，程序流程图如下：

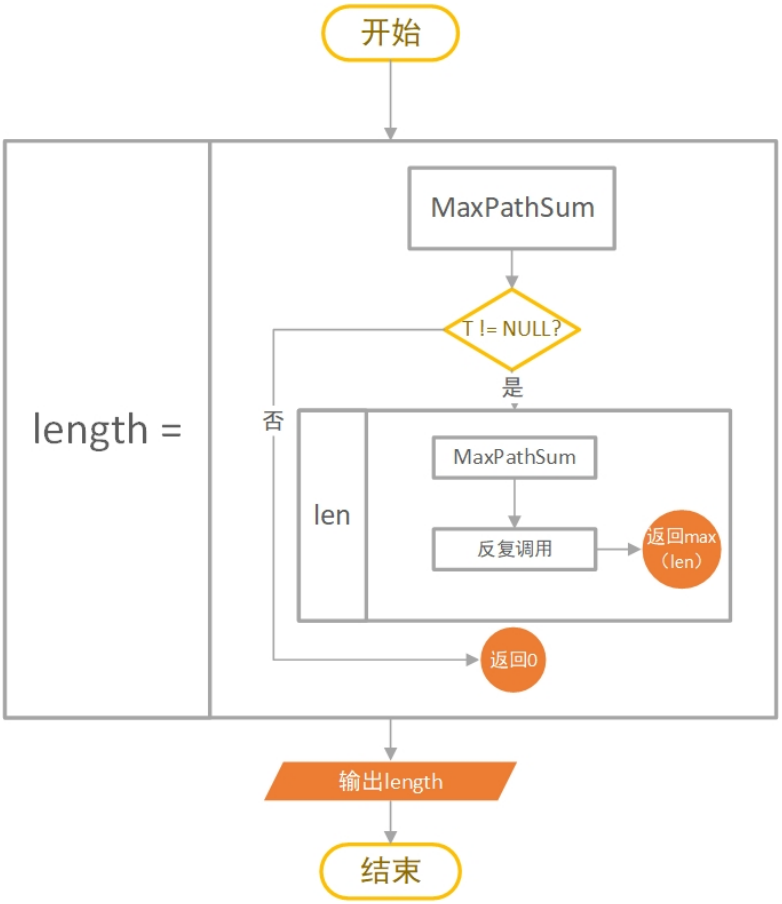


图 2-3　MaxPathSum 函数程序流程图

时间复杂度：O(nlogn)

空间复杂度：O(n)

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

输入：二叉树根结点，整型变量 1（查找结点的关键字），整型变量 2（查找结点的关键字）

输出：二叉树结点指针（含义为最近公共祖先）

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(n)

**17.** status InvertTree(BiTree &T)

输入：二叉树根结点

输出：函数运行状态

函数思想描述：将二叉树翻转写成函数。该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交换左右子树，然后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。最终得到翻转后的二叉树。

时间复杂度：O(n)

空间复杂度：O(n)

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

输入：二叉树根结点，字符串变量

输出：函数运行状态

函数思想描述：将把树写入文件写成函数，将二叉树的结点数据写入到文件 FileName 中。该函数采用了非递归思想，通过栈的数据结构模拟先序遍历带空枝遍历，并将遍历结果写入文件中。

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

输入：二叉树根结点，字符串变量

输出：函数运行状态

函数思想描述：将从文件中读取树写成函数，将文件 FileName 中的数据读取到空树中，该函数采用了递归思想，该函数是将读取的数据按照先序遍历带空枝遍历对空树进行赋值的。

**20.** status AddList(TREES &trees,char ListName[])

输入：结构体类型变量（森林），字符串类型（含义为待添加树的名称）

输出：函数运行状态

函数思想描述：将增加树写成函数，函数的参数是森林和树名称。trees 是一个以数组形式管理的森林，在数组尾部新增树，并将树名称存储在该树结点的 name 分量当中。当然，在添加二叉树之前，应当判断名称是否唯一。

**21.** status DestoryList(TREES &trees,char ListName[])

输入：结构体类型变量（森林），字符串类型（含义为待删除树的名称）

输出：函数运行状态

函数思想描述：将树写销毁树成函数，函数的参数是森林和树名称。在 trees 数组中查找名称为 ListName 的树，查找成功则将其删除，否则返回 ERROR。

**22.** int LocateList(TREES trees,char ListName[])

输入：结构体类型变量（森林），字符串类型（含义为待查找树的名称）

输出：整型变量（二叉树的位序）

函数思想描述：将查找二叉树写成函数，函数的参数是森林和树名称。在 trees 数组中查找名称为 ListName 的树，查找成功返回其位序。

**23.** status TraverseList(TREES trees)

输入：结构体类型变量（森林）

输出：函数运行状态

函数思想描述：将遍历森林写成函数，函数的参数为森林。顺序遍历森林中各个树即可。

**24.** status SelectList(TREES trees, int i)

输入：结构体类型变量（森林），整型变量（含义为逻辑索引）

输出：函数运行状态

函数思想描述：将选择二叉树写成函数，函数的参数为森林和逻辑索引。将 Trees 中逻辑索引为 i 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。

**25.** status ClearList(BiTree &T)

输入：二叉树根结点

输出：函数运行状态

函数思想描述：将清空在森林中的二叉树写成函数。直接调用 ClearBiTree 函数清空二叉树，保留二叉树的位置和名称。

**26.** BiTNode* GetFabling(BiTree T,KeyType e)

输入：二叉树根结点，KeyType 类型（查找结点的关键字）

输出：二叉树结点指针

函数思想概述：将获得当前结点的父亲结点写成函数。该函数使用了递归思想。在函数中，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，则找到父亲结点，返回父亲结点指针。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父亲节点；若返回值不为 NULL 则说明在某一子树中找到了指定节点的父亲结点，返回该结点的值。

## 2.4 系统测试

系统菜单整体布局如图：

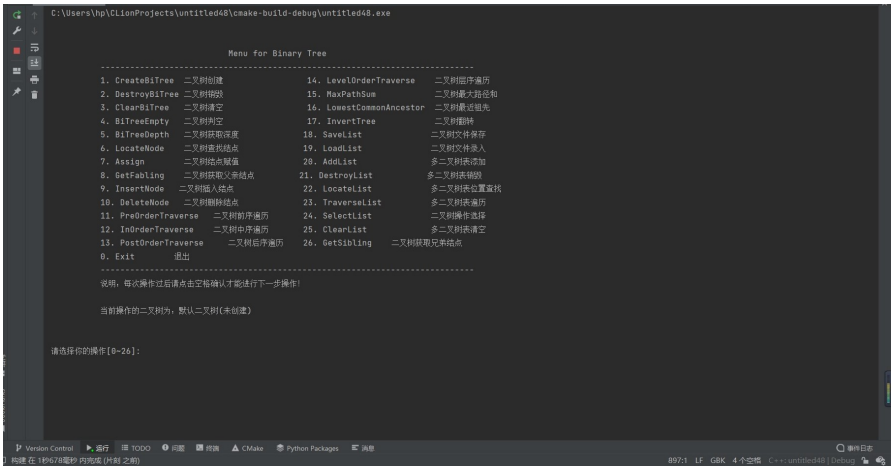


图 2-4 菜单

测试集如下：

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

测试集 2：森林中创建两个二叉树

FirstTree：1 a 2 b 0 null 0 null 3 c 4 d 0 null 0 null 5 e 0 null 0 null -1 null

SecondTree：1 a 2 b 0 null 3 c 4 d 0 null 5 e 0 null 0 null 0 null 0 null -1 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 2：将使用关键字重复的序列 1（1 a 2 b 0 null 0 null 3 c 4 d 0 null 0 null 3 e 0 null 0 null -1 null），测试函数能否给出判断；

测试 3：将输入根节点为空的序列（0 null -1 null），测试函数能否正确创建二叉树。

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！<br>先序遍历：1,a 2,b 3,c 4,d 5,e<br>中序遍历：2,b 1,a 4,d 3,c 5,e | 一致 |
| 2 | 1→ 序列 1 | 关键字不唯一！创建失败！ | 一致 |
| 3 | 1→ 序列 2 | 二叉树创建成功！ | 一致 |
| 4 | 1→ 测试集 1 | 该二叉树已存在！ | 一致 |

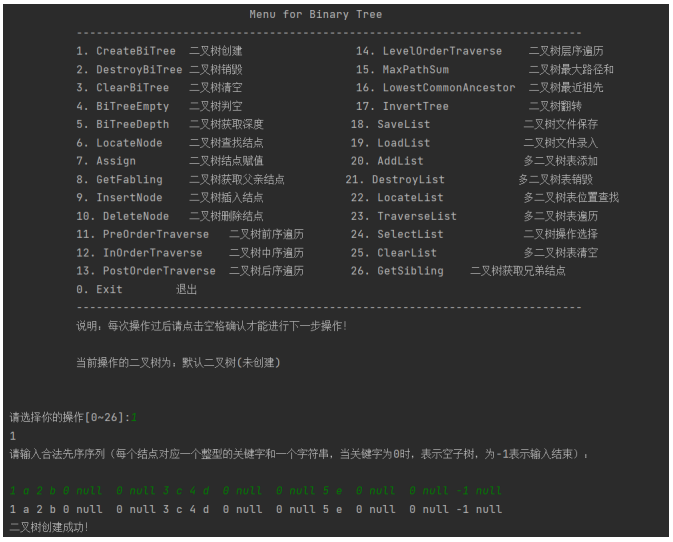


图 2-5 测试 3 运行结果

**2.** DestroyBiTree 测试

测试 1：将测试函数能否销毁不存在的二叉树；

测试 2：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。

| 测试编号 | 测试输入 | 预期结果 | 实际运行结果 |
|---|---|---|---|
| 1 | 2 | 二叉树为空！ | 一致 |
| 2 | 1→2 | 二叉树销毁成功！ | 一致 |

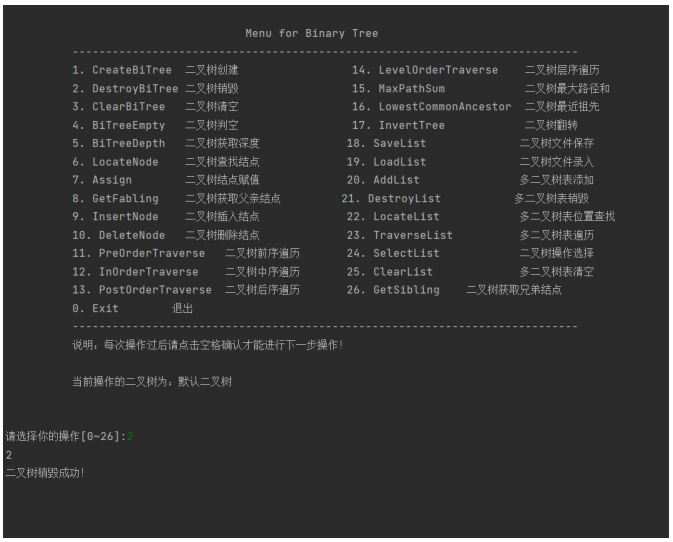


图 2-6　测试 2 运行结果

**3.** ClearBiTree 测试

测试 1：将测试函数能否清空不存在的二叉树；

测试 2：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。

| 测试编号 | 测试输入 | 预期结果 | 实际运行结果 |
|---|---|---|---|
| 1 | 3 | 二叉树为空！ | 一致 |
| 2 | 1→3 | 二叉树清空成功！ | 一致 |


图 2-7　测试 2 运行结果

**4.** BiTreeEmpty 测试

测试 1：测试函数是否能对空二叉树进行判空；

测试 2：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。

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

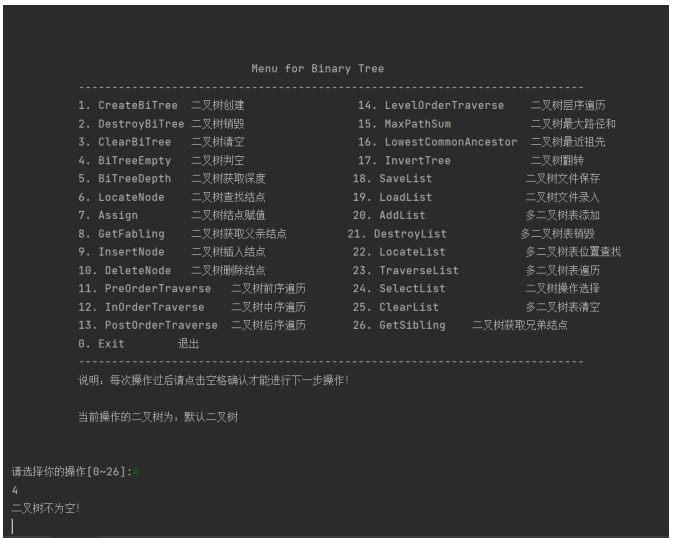


图 2-8　测试 2 运行结果

**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 |


图 2-9　测试 2 运行结果

**6.** LocateNode 测试

本函数的测试在测试集 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 |

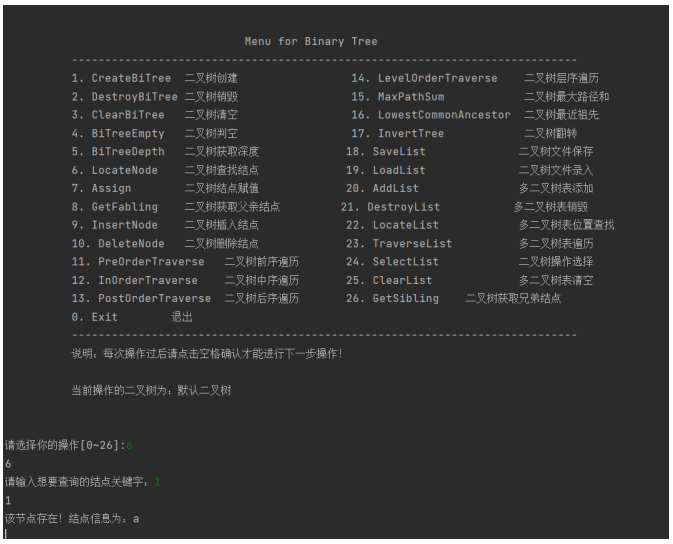


图 2-10　测试 1 运行结果

**7.** Assign 测试

本函数的测试在测试集 1 的情况下进行。

输入要求：先输入想要修改的结点关键字，再输入修改后的值，同时测试为顺序进行。

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！<br>先序遍历：1,a 2,b 3,c 4,d 6,f | 一致 |
| 2 | 7→1→2 g | 结点复制失败（请检查该关键字是否存在或者赋值关键字是否重复） | 一致 |
| 3 | 7→7→8 h | 结点复制失败（请检查该关键字是否存在或者赋值关键字是否重复） | 一致 |


图 2-11　测试 1 运行结果

**8.** GetSibling 测试

本函数的测试在测试集 1 的情况下进行。

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！<br>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！<br>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 |

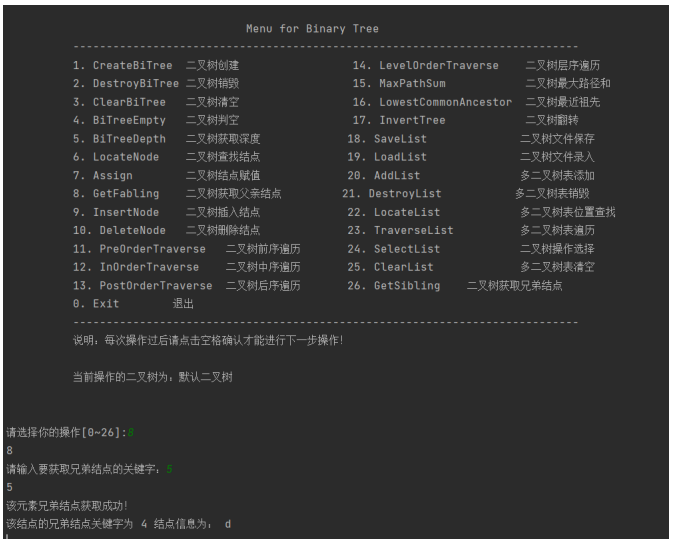


图 2-12　测试 1 运行结果

**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！<br>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！<br>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 |

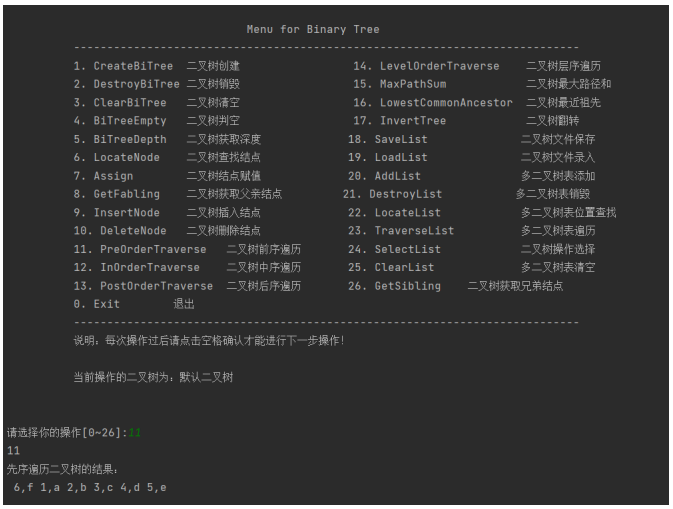


图 2-13　测试 1 运行结果

**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 |

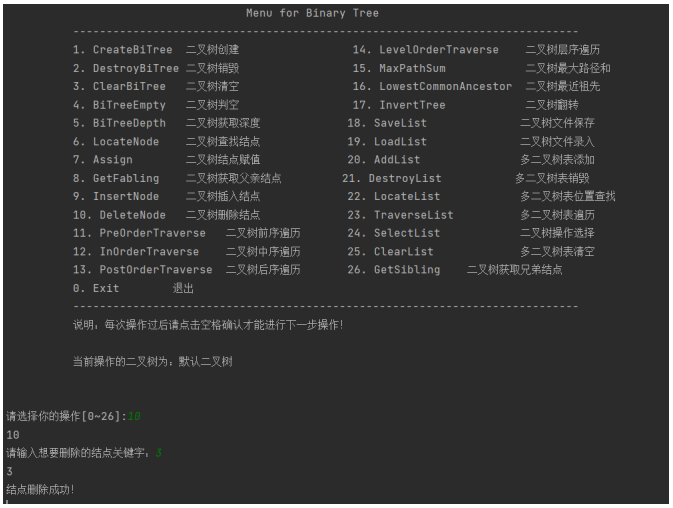


图 2-14　测试 1 运行结果

**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：<br>1,a 2,b 3,c 4,d 5,e | 一致 |

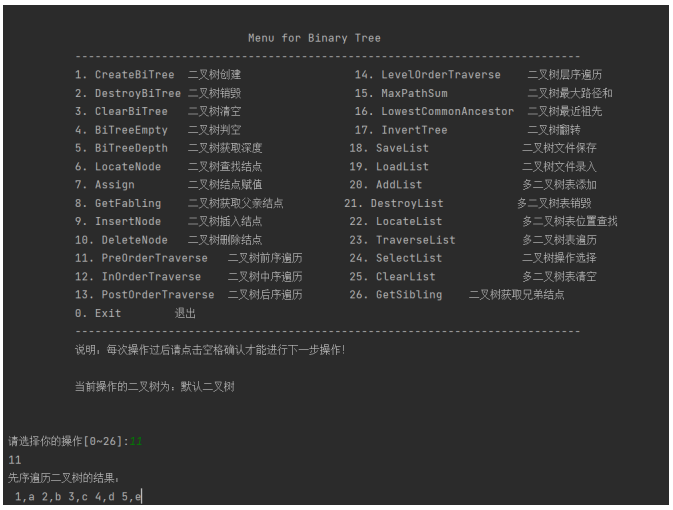


图 2-15　测试 2 运行结果

**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：<br>2,b 1,a 4,d 3,c 5,e | 一致 |

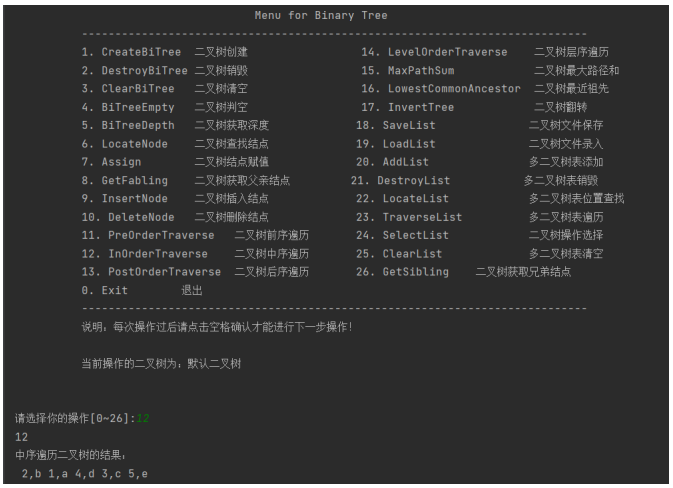


图 2-16　测试 2 运行结果

**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：<br>2,b 4,d 5,e 3,c 1,a | 一致 |

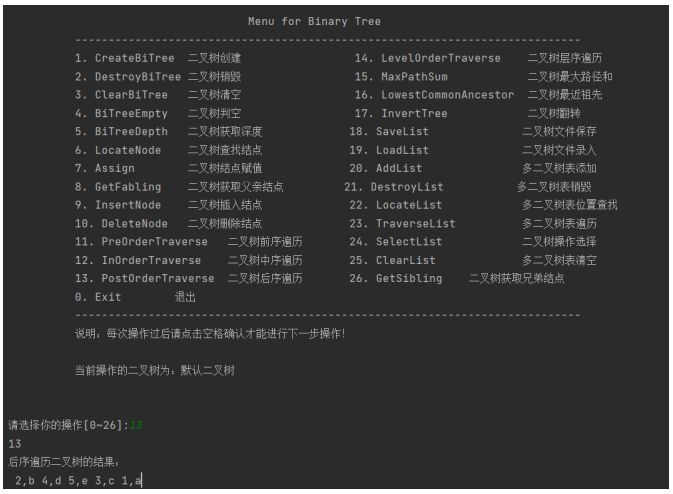


图 2-17　测试 2 运行结果

**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：<br>1,a 2,b 3,c 4,d 5,e | 一致 |

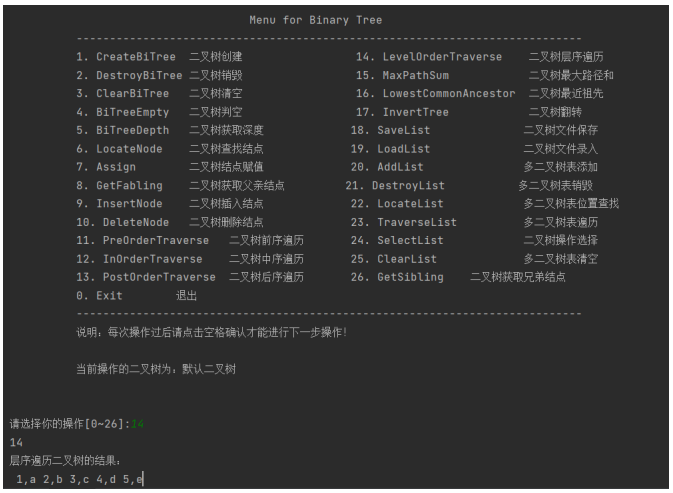


图 2-18　测试 2 运行结果

### 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 |

**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 |

**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！<br>层序遍历二叉树的结果：1,a 3,c 2,b 5,e 4,d | 一致 |

**18.** 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 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.** 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 |

**20.** AddList 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：构建测试集 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 |

**21.** DestoryList 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 |

**22.** 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→FirstTree | 该二叉树的逻辑索引为：1 | 一致 |
| 2 | 22→ThirdTree | 二叉树查找失败！ | 一致 |

**23.** TraverseList 测试

测试 1：在测试集 3 的基础上进行，测试函数能否正确遍历各个线性表；

测试 2：在空集合的基础上进行，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 | （测试集 2）23 | FirstList SecondList | 一致 |
| 2 | 23 | 多二叉树表为空！ | 一致 |

**24.** SelectList 测试

输入要求：输入线性表集合中线性表对应的逻辑索引。

测试 1,2：在测试集 3 的情况下进行，测试函数能否正确判断非法的位序；

测试 3：在测试集 3 的情况下进行，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→0 | 二叉树选取失败！ | 一致 |
| 1 | 24→3 | 二叉树选取失败！ | 一致 |
| 2 | 24→1 | 二叉树已选取成功！ | 一致 |

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。（25，26 函数属于辅助函数，对于测试不做要求。）

## 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] Larry Nyhoff. ADTs, Data Structures, and Problem Solving with C++. Second Edition,Calvin College,2005

[3] 殷立峰. Qt C++ 跨平台图形界面程序设计基础. 清华大学出版社,2014:192～197

[4] 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 <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.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
#define MAXlength 10
typedef struct{  //顺序表（顺序结构）的定义
    ElemType * elem;
    int length;
    int listsize;
}SqList;
SqList L;

typedef struct{  //线性表的集合类型定义
    struct { char name[30];
        SqList L;
    } elem[11];
    int length;
}LISTS;
LISTS Lists;       //线性表集合的定义Lists

status InitList(SqList& L)
// 线性表L不存在，构造一个空的线性表，返回OK，否则返回
```

```
    INFEASIBLE。
{
    if(L.elem) return INFEASIBLE;
    L.elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(
        ElemType));
    if(!L.elem)
        exit(OVERFLOW);
    L.length = 0;
    L.listsize = LIST_INIT_SIZE;
    return OK;

}

status DestroyList(SqList& L)
// 如果线性表L存在，销毁线性表L，释放数据元素的空间，返回OK，
    否则返回INFEASIBLE。
{
    if(!L.elem) return INFEASIBLE;
    free(L.elem);
    L.elem = NULL;
    return OK;

}

status ClearList(SqList& L)
// 如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回
    INFEASIBLE。
{
    if(!L.elem) return INFEASIBLE;
    L.length = 0;
    return OK;

}

status ListEmpty(SqList L)
```

```
65  // 如果线性表L存在，判断线性表L是否为空，空就返回TRUE，否则返
        回FALSE；如果线性表L不存在，返回INFEASIBLE。
66  {
67      if(!L.elem) return INFEASIBLE;
68      if(!L.length) return TRUE;
69      else return FALSE;
70
71  }
72
73  status ListLength(SqList L)
74  // 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
75  {
76      if(!L.elem) return INFEASIBLE;
77      return L.length;
78  }
79
80  status GetElem(SqList L,int i,ElemType &e)
81  // 如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK
        ；如果i不合法，返回ERROR；如果线性表L不存在，返回INFEASIBLE
        。
82  {
83      if(!L.elem) return INFEASIBLE;
84      if(i < 1 || i > L.length) return ERROR;
85      e = L.elem[i - 1];
86      return OK;
87
88  }
89
90  int compare(SqList L, int i, ElemType e){
91          if(L.elem[i] == e) return 1;
92          return 0;
93  }
94
95  int LocateElem(SqList L,ElemType e, int (*compare)(SqList, int
        , ElemType))
```

```
96  // 如果线性表L存在，查找元素e在线性表L中的位置序号并返回该序
        号；如果e不存在，返回0；当线性表L不存在时，返回INFEASIBLE
        （即-1）。
97  {
98      if(!L.elem) return INFEASIBLE;
99      for(int i = 0; i < L.length; i++){
100         if(compare(L, i, e)) return i + 1;
101     }
102     return 0;
103
104 }
105
106 status PriorElem(SqList L,ElemType e,ElemType &pre)
107 // 如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返
        回OK；如果没有前驱，返回ERROR；如果线性表L不存在，返回
        INFEASIBLE。
108 {
109     if(!L.elem) return INFEASIBLE;
110     int i = LocateElem(L, e, compare) - 1;
111     if(i <= 0 || i == L.length) return ERROR;
112     else {
113         pre = L.elem[i - 1];
114         return OK;
115     }
116
117 }
118
119 status NextElem(SqList L,ElemType e,ElemType &next)
120 // 如果线性表L存在，获取线性表L元素e的后继，保存在next中，返回
        OK；如果没有后继，返回ERROR；如果线性表L不存在，返回
        INFEASIBLE。
121 {
122     if(!L.elem) return INFEASIBLE;
123     int i = LocateElem(L, e, compare) - 1;
124     if(i >= L.length - 1) return ERROR;
```

```
125         else {
126             next = L.elem[i + 1];
127             return OK;
128         }
129 
130 }
131 
132 status ListInsert(SqList &L,int i,ElemType e)
133 // 如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回
        OK；当插入位置不正确时，返回ERROR；如果线性表L不存在，返回
        INFEASIBLE。
134 {
135     if(!L.elem) return INFEASIBLE;
136     if(i < 1 || i > L.length + 1) return ERROR;
137     if(L.length == L.listsize){
138         ElemType* newbase = (ElemType *)realloc(L.elem, (L.
                listsize + LISTINCREMENT) * sizeof(ElemType));
139         L.elem = newbase;
140         L.listsize += LISTINCREMENT;
141     }
142     int j;
143     for(j = L.length; j >= i - 1; j--)
144         L.elem[j + 1] = L.elem[j];
145     L.elem[j + 1] = e;
146     L.length++;
147     return OK;
148 
149 }
150 
151 status ListDelete(SqList &L,int i,ElemType &e)
152 // 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回
        OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回
        INFEASIBLE。
153 {
154     if(!L.elem) return INFEASIBLE;
```

```
155     if(i < 1 || i > L.length) return ERROR;
156     int j;
157     e = L.elem[i - 1];
158     for(j = i - 1; j <= L.length ; j++)
159         L.elem[j] = L.elem[j + 1];
160     L.length--;
161     return OK;
162
163 }
164
165 int visit(SqList L, int i){
166         printf("%d", L.elem[i]);
167 }
168
169 status ListTraverse(SqList L, int (*visit)(SqList, int))
170 // 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，
      返回OK；如果线性表L不存在，返回INFEASIBLE。
171 {
172     if(!L.elem) {
173         printf("线性表未创建！\n");
174         return INFEASIBLE;
175     }
176     printf("\n----------all elements ----------------------\
        n");
177     for(int i = 0; i < L.length; i++){
178         visit(L, i);
179         if(i != L.length - 1) printf(" ");
180     }
181     printf("\n----------------- end ----------------------\
        n");
182     return L.length;
183
184 }
185
186
```

```
187 status MaxSubArray(SqList L){
188 // 返回线性表中的连续数组和的最大值
189     int l = 0, r, max = 0, temp = L.elem[0];
190     for(r = 0; r < L.length; r++){
191         while(max < 0 && l <= r){
192             max -= L.elem[l++];
193         }
194         max += L.elem[r];
195         if(max > temp) temp = max;
196     }
197     return temp;
198 
199 }
200 
201 status SubArrayNum(SqList L, int k){
202 // 返回线性表中连续数组和为k的连续数组数目
203     int sum = 0, count = 0;
204     int *l = (int *)malloc(sizeof(int) * (L.length + 1));
205     l[0] = 0;
206     //获取前缀和
207     for(int i = 0; i < L.length; i++){
208         sum += L.elem[i];
209         l[i + 1] = sum;
210     }
211     //通过前缀和之间的差值计算连续数组和
212     for(int i = 0; i < L.length; i++)
213         for(int j = i + 1; j < L.length + 1; j++)
214             count += (l[j] - l[i] == k);
215     return count;
216 
217 }
218 
219 void merge(int *elem, int l, int r){
220         if(r - l <= 1){
221                 if(elem[l] > elem[r]){
```

```
222                         int t = elem[l];
223                         elem[l] = elem[r];
224                         elem[r] = t;
225                 }
226                 return;
227         }
228         int mid = (l + r) >> 1;
229         merge(elem, l, mid);
230         merge(elem, mid + 1, r);
231         int p1 = l, p2 = mid + 1, k = 0;
232         int *temp = (int *)malloc(sizeof(int)*(r - l + 1));
233         while(p1 <= mid || p2 <= r){
234                 if(p2 > r || (p1 <= mid && elem[p1] < elem[p2
                    ]))
235                 temp[k++] = elem[p1++];
236                 else
237                 temp[k++] = elem[p2++];
238         }
239         memcpy(elem + l, temp, sizeof(int) * (r - l + 1));
240         free(temp);
241         return;
242 }
243
244 status sortList(SqList& L){
245 // 如果线性表L存在，将线性表中的元素排序；如果线性表L不存在，
    返回INFEASIBLE。
246     if(!L.elem) {
247         printf("线性表未创建！\n");
248         return INFEASIBLE;
249     }
250     merge(L.elem, 0, L.length - 1);
251     return L.length;
252 }
253
254 status SaveList(SqList L,char FileName[])
```

```
255 // 如果线性表L存在，将线性表L的的元素写到FileName文件中，返回
    OK，否则返回INFEASIBLE。
256 {
257     if(!L.elem) return INFEASIBLE;
258     FILE *fp;
259     char ch;
260     if ((fp = fopen(FileName,"rb")) == NULL){
261         printf("File open error\n");
262         exit(-1);
263     }
264         ch = fgetc(fp);
265     if(ch != EOF){
266         printf("该文件已有内容，不能读入！\n");
267         return ERROR;
268         }
269     if ((fp = fopen(FileName,"wb")) == NULL){
270         printf("File open error\n");
271         exit(-1);
272     }
273
274     fwrite(L.elem, sizeof(ElemType), L.length, fp);
275     fclose(fp);
276     return OK;
277
278 }
279
280 status  LoadList(SqList &L,char FileName[])
281 // 如果线性表L不存在，将FileName文件中的数据读入到线性表L中，
    返回OK，否则返回INFEASIBLE。
282 {
283     if(L.elem) return INFEASIBLE;
284     FILE *fp;
285     if ((fp = fopen(FileName,"rb")) == NULL){
286         printf("File open error\n");
287         exit(-1);
```

```
288         }
289         L.elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(
                ElemType));
290         L.length = 0;
291         L.listsize = LIST_INIT_SIZE;
292         while(fread(&L.elem[L.length], sizeof(ElemType), 1, fp))
293             L.length++;
294         fclose(fp);
295         return OK;
296
297     }
298
299     status AddList(LISTS &Lists,char ListName[])
300     // 只需要在Lists中增加一个名称为ListName的空线性表，线性表数据
            又后台测试程序插入。
301     {
302             for(int i = 0; i < Lists.length; i++){
303                     if(!strcmp(Lists.elem[i].name,ListName))
                            return INFEASIBLE;
304             }
305         strcpy(Lists.elem[Lists.length].name,ListName);
306         Lists.elem[Lists.length].L.length=0;
307         Lists.elem[Lists.length].L.elem=(ElemType *)malloc(sizeof(
                ElemType)*LIST_INIT_SIZE);
308         Lists.elem[Lists.length].L.listsize=LIST_INIT_SIZE;
309         Lists.length++;
310         return OK;
311     }
312
313     status RemoveList(LISTS &Lists,char ListName[])
314     // Lists中删除一个名称为ListName的线性表
315     {
316         for(int i = 0; i < Lists.length; i++)
317             if(!strcmp(ListName, Lists.elem[i].name)){
318                     if(Lists.elem[i].L.elem)
```

```
319                     DestroyList(Lists.elem[i].L);
320                 for (int j = i; j < Lists.length - 1; j++)
321                     Lists.elem[j] = Lists.elem[j + 1];
322                 Lists.length--;
323                 return OK;
324             }
325         return ERROR;
326
327     }
328
329     int LocateList(LISTS Lists,char ListName[])
330     // 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，
        否则返回0
331     {
332         if(!Lists.elem) return INFEASIBLE;//疑问未解决
333         for(int i = 0; i < Lists.length; i++)
334             if(!strcmp(ListName, Lists.elem[i].name))
335                 return i + 1;
336         return 0;
337
338     }
339
340     status TraverseList(LISTS Lists){
341     // 如果多线性表不为空，依次显示多线性表的名称，每个名称间空一
        格，返回OK；如果多线性表为空，返回INFEASIBLE。
342         if(Lists.length == 0) return INFEASIBLE;
343         printf("\n----------all names ----------------------\n")
            ;
344         for(int i = 0; i < Lists.length; i++){
345             printf("%s",Lists.elem[i].name);
346             if(i != Lists.length - 1) printf(" ");
347         }
348         printf("\n------------------ end ------------------------\
            n");
349         return OK;
```

```
350 }
351
352 status SelectList(LISTS Lists, int i){
353 // 进行线性表的选择
354     if(Lists.length == 0) return INFEASIBLE;
355     if(i < 1 || i > Lists.length) return ERROR;
356     L = Lists.elem[i - 1].L;
357     return OK;
358 }
359
360 int main(){
361     int op=1;
362     int length, flag, temp, num = 0;
363     char FileName[100];
364     char Name[20];
365     Lists.length = 0;
366     while(op){
367         system("cls");
368         printf("\n\n");
369         printf("              Menu for Linear Table On Sequence
            Structure \n");
370         printf("
            ------------------------------------
371 ----------------------------------------\n");
372         printf("          1. InitList     线性表初始化
            12. ListTraverse    线性表遍历\n");
373         printf("          2. DestroyList 线性表销毁
            13. MaxSubArray     线性表最大连续数组和获取\n");
374         printf("          3. ClearList    线性表清空
            14. SubArrayNum     线性表指定连续数组和数目\n");
375         printf("          4. ListEmpty    线性表判空
            15. sortList        线性表排序\n");
376         printf("          5. ListLength   线性表获取长度
            16. SaveList        线性表文件保存\n");
377         printf("          6. GetElem      线性表元素获取
```

```
                17. LoadList          线性表文件录入\n");
378             printf("              7. LocateElem   线性表元素查找
                18. AddList           多线性表添加\n");
379             printf("              8. PriorElem    线性表元素前驱获取
                19. RemoveList        多线性表删除\n");
380             printf("              9. NextElem     线性表元素后继获取
                20. LocateList        多线性表位置查找\n");
381             printf("              10. ListInsert  线性表元素插入
                21. TraverseList      多线性表遍历\n");
382             printf("              11. ListDelete  线性表元素删除
                22. SelectList        线性表操作选择\n");
383             printf("              0. Exit         退出\n");
384             printf("
                ---------------------------------------
385 ------------------------------------\n");
386                     printf("          说明: 每次操作过后请点击空格
                            确认才能进行下一步操作! \n");
387                     printf("\n          当前操作的线性表为: ");
388                     if(num < 1|| num > Lists.length){
389                             if(num > Lists.length){
390                                     L.elem = NULL;
391                                     L.length = 0;
392                                     num = 0;
393                             }
394                             printf("默认线性表");
395                             if(!L.elem)
396                                     printf("(未创建)");
397                             printf("\n\n\n");
398                     }
399                     else
400                             printf("%s\n\n\n",Lists.elem[num - 1].
                                name);
401                     if(op > 22 || op < 0)
402                     printf("上一步命令出错! 请根据菜单正确输入! \n
                            \n\n");
```

```
403         printf("请选择你的操作[0~22]:");
404         scanf("%d",&op);
405
406         switch(op){
407             case 1:
408                 //printf("\n----IntiList功能待实现! \n");
409                 if(InitList(L) == OK) printf("线性表创建成功!
                        \n");
410                 else printf("线性表创建失败! \n");
411                 getchar();getchar();
412                 break;
413             case 2:
414                 //printf("\n----DestroyList功能待实现! \n");
415                 if(DestroyList(L) == OK) printf("线性表销毁成
                        功! \n");
416                 else printf("线性表销毁失败! \n");
417                 getchar();getchar();
418                 break;
419             case 3:
420                 //printf("\n----ClearList功能待实现! \n");
421                 if(ClearList(L) == OK) printf("线性表清空成
                        功! \n");
422                 else printf("线性表清空失败! \n");
423                 getchar();getchar();
424                 break;
425             case 4:
426                 //printf("\n----ListEmpty功能待实现! \n");
427                 if(ListEmpty(L) == OK) printf("线性表为空! \n"
                        );
428                 else printf("线性表非空! \n");
429                 getchar();getchar();
430                 break;
431             case 5:
432                 //printf("\n----ListLength功能待实现! \n");
433                 length = ListLength(L);
```

```
434                 if(length != INFEASIBLE) printf("线性表的长度
                        为：%d\n", length);
435                 else printf("线性表未创建!\n");
436                 getchar();getchar();
437                 break;
438             case 6:
439                 //printf("\n----GetElem功能待实现！\n");
440                 int x, y;
441                 printf("请输入要获取元素的位置：");
442                 scanf("%d",&x);
443                 flag = GetElem(L, x, y);
444                 if(flag == INFEASIBLE) printf("线性表未创建!\n
                        ");
445                 else if(flag == OK) printf("线性表中的第%d个元
                        素为%d\n", x, y);
446                 else printf("输入的逻辑索引不合法！\n");
447                 getchar();getchar();
448                 break;
449             case 7:
450                 //printf("\n----LocateElem功能待实现！\n");
451                 int a;
452                 printf("请输入想要查找的元素：");
453                 scanf("%d",&a);
454                 flag = LocateElem(L, a, compare);
455                 if(flag == INFEASIBLE) printf("线性表未创建!\n
                        ");
456                 else if(flag) printf("该元素存在且元素逻辑索引
                        为：%d\n", flag);
457                 else printf("该元素不存在！\n");
458                 getchar();getchar();
459                 break;
460             case 8:
461                 //printf("\n----PriorElem功能待实现！\n");
462                 printf("请输入想要查找的元素(获取前驱)：");
463                 scanf("%d",&a);
```

```
464                 flag = PriorElem(L, a, temp);
465                 if(flag == INFEASIBLE) printf("线性表未创建!\n
                        ");
466                 else if(flag == OK) printf("该元素存在且前驱元
                        素为: %d\n", temp);
467                 else printf("该元素不存在或不存在前驱! \n");
468                 getchar();getchar();
469                 break;
470             case 9:
471                 //printf("\n----NextElem功能待实现! \n");
472                 printf("请输入想要查找的元素(获取后继): ");
473                 scanf("%d",&a);
474                 flag = NextElem(L, a, temp);
475                 if(flag == INFEASIBLE) printf("线性表未创建!\n
                        ");
476                 else if(flag == OK) printf("该元素存在且后继元
                        素为: %d\n", temp);
477                 else printf("该元素不存在或不存在后继! \n");
478                 getchar();getchar();
479                 break;
480             case 10:
481                 //printf("\n----ListInsert功能待实现! \n");
482                 int i, e;
483                 printf("请输入要插入的元素位置: ");
484                 scanf("%d",&i);
485                 printf("请输入要插入的元素: ");
486                 scanf("%d",&e);
487                 flag = ListInsert(L, i, e);
488                 if(flag == OK) printf("线性表插入成功! \n");
489                 else if(flag == INFEASIBLE) printf("线性表不存
                        在, 插入失败! \n");
490                                     else printf("插入位置不合法,
                                            线性表插入失败! \n");
491                 getchar();getchar();
492                 break;
```

```
493             case 11:
494                 //printf("\n----ListDelete功能待实现! \n");
495                 printf("请输入要删除的元素位置: ");
496                 scanf("%d",&i);
497                 if(ListDelete(L, i, e) == OK){
498                         printf("线性表删除成功! \n");
499                         printf("删除的元素为: %d\n",e);
500                                 }
501                 else printf("线性表删除失败! \n");
502                 getchar();getchar();
503                 break;
504             case 12:
505                 //printf("\n----ListTraverse功能待实现! \n");
506                 if(!ListTraverse(L, visit)) printf("线性表是空
                        表! \n");
507                 getchar();getchar();
508                 break;
509                 //-2 1 -3 4 -1 2 1 -5 4
510             case 13:
511                 //printf("\n----MaxSubArray功能待实现! \n");
512                 if(!L.elem) printf("线性表未创建! \n");
513                 else if (ListEmpty(L)) printf("线性表为空! \n"
                        );
514                 else printf("最大子数组之和为: %d\n",
                        MaxSubArray(L));
515                 getchar();getchar();
516                 break;
517                 //6
518             case 14:
519                 //printf("\n----SubArrayNum功能待实现! \n");
520                 if(!L.elem){printf("线性表未创建! \n");getchar
                        ();getchar();break;}
521                 else if (ListEmpty(L)) {printf("线性表为空! \n
                        ");getchar();getchar();break;}
522                 printf("请输入寻找的连续数组的和: ");
```

```
523                 scanf("%d",&flag);
524                 printf("和为数%d的连续数组数目为：%d\n",flag,
                        SubArrayNum(L,flag));
525                 getchar();getchar();
526                 break;
527                 //3 5
528                 //5 2
529                 //15 0
530             case 15:
531                 //printf("\n----sortList功能待实现！\n");
532                 flag = sortList(L);
533                 if(!flag) printf("线性表是空表！\n");
534                 else if(flag != INFEASIBLE) printf("线性表排序
                        成功！\n");
535                 getchar();getchar();
536                 break;
537             case 16:
538                 //printf("\n----SaveList功能待实现！\n");
539                 printf("请输入要保存的文件名称：");
540                 scanf("%s",FileName);
541                 flag = SaveList(L, FileName);
542                 if(flag == INFEASIBLE) printf("线性表不存在!文
                        件保存失败！\n");
543                 else if(flag == ERROR);
544                 else printf("文件保存成功！\n");
545                 getchar();getchar();
546                 break;
547             case 17:
548                 //printf("\n----LoadList功能待实现！\n");
549                 printf("请输入要录入的文件名称：");
550                 scanf("%s",FileName);
551                 if(LoadList(L, FileName) == INFEASIBLE) printf
                        ("线性表存在!文件录入失败！\n");
552                 else printf("文件录入成功！\n");
553                 getchar();getchar();
```

```
554                 break;
555             case 18:
556                 //printf("\n----AddList功能待实现！\n");
557                 if(Lists.length == MAXlength) {
558                         printf("多线性表管理已满，请清除某些线
                            性表后再操作！\n");
559                         getchar();getchar();
560                         break;
561                                 }
562                 printf("请输入新增线性表的名称：");
563                 scanf("%s",Name);
564                 flag = AddList(Lists, Name);
565                 if(flag == INFEASIBLE) printf("该名称的线性表
                    已经存在！\n");
566                 else printf("%s已成功添加！\n",Name);
567                 getchar();getchar();
568                 break;
569             case 19:
570                 //printf("\n---RemoveList功能待实现！\n");
571                 if(Lists.length == 0) {
572                                         printf("多线性表管理已
                                            空，请添加某些线性
                                            表后再操作！\n");
573                                         getchar();getchar();
574                         break;
575                                 }
576                                 printf("请输入删除线性表的名
                                    称：");
577                 scanf("%s",Name);
578                 flag = RemoveList(Lists, Name);
579                 if(flag == OK) printf("%s已成功删除！\n",Name);
580                 else printf("线性表不存在！\n");
581                 getchar();getchar();
582                 break;
583             case 20:
```

```
584                     //printf("\n---LocateList功能待实现! \n");
585                     printf("请输入查找线性表的名称: ");
586                     scanf("%s",Name);
587                     if(LocateList(Lists, Name)) printf("该线性表的
                            逻辑索引为: %d\n", LocateList(Lists, Name))
                            ;
588                     else printf("线性表查找失败! \n");
589                     getchar();getchar();
590                     break;
591                 case 21:
592                     //printf("\n----TraverseList功能待实现! \n");
593                     if(TraverseList(Lists) == INFEASIBLE) printf("
                            多线性表为空! \n");
594                     getchar();getchar();
595                     break;
596                 case 22:
597                     //printf("\n----SelectList功能待实现! \n");
598                     printf("请选择要处理的线性表的逻辑索引: ");
599                     scanf("%d",&flag);
600                     if(SelectList(Lists, flag) == OK) {
601                             printf("线性表已选取成功! \n");
602                             num = flag;
603                                 }
604                     else printf("线性表选取失败! \n");
605                     getchar();getchar();
606                     break;
607                 case 0:
608                     break;
609             }//end of switch
610         }//end of while
611         printf("欢迎下次再使用本系统! \n");
612         return 0;
613 }//end of main()
```

# 5 附录 B 基于链式存储结构线性表实现的源程序

```
1  #include <stdio.h>
2  #include <malloc.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <windows.h>
6
7  #define TRUE 1
8  #define FALSE 0
9  #define OK 1
10 #define ERROR 0
11 #define error -3
12 #define INFEASIBLE -1
13 #define OVERFLOW -2
14 typedef int status;
15 typedef int ElemType; //数据元素类型定义
16 #define LIST_INIT_SIZE 100
17 #define LISTINCREMENT 10
18 #define MAXlength 10
19 typedef struct LNode{  //单链表（链式结构）结点的定义
20     ElemType data;
21     struct LNode *next;
22 }LNode,*LinkList;
23
24 LinkList L;
25
26 typedef struct{  //线性表的集合类型定义
27     struct { char name[30];
28         LinkList L;
29     } elem[11];
30     int length;
31 }LISTS;
32 LISTS Lists;       //线性表集合的定义Lists
33
```

```
34  status InitList(LinkList &L)
35  // 线性表L不存在，构造一个空的线性表，返回OK，否则返回
        INFEASIBLE。
36  {
37      if(L) return INFEASIBLE;
38      L = (LinkList)malloc(sizeof(LinkList));
39      L->next = NULL;
40      return OK;
41
42  }
43
44  status DestroyList(LinkList &L)
45  // 如果线性表L存在，销毁线性表L，释放数据元素的空间，返回OK，
        否则返回INFEASIBLE。
46  {
47      if(!L) return INFEASIBLE;
48      LinkList p = L, q = L->next;
49      while(q != NULL){
50          free(p);
51          p = q;
52          q = q->next;
53      }
54      free(p);
55      L = NULL;
56      return OK;
57  }
58
59
60  status ClearList(LinkList &L)
61  // 如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回
        INFEASIBLE。
62  {
63      if(!L) return INFEASIBLE;
64      if(!L->next) return ERROR;
65      LinkList p = L->next, q = p->next;
```

```
66      while(q != NULL){
67          free(p);
68          p = q;
69          q = q->next;
70      }
71      if(p) free(p);
72      L->next = NULL;
73      return OK;
74
75  }
76
77
78  status ListEmpty(LinkList L)
79  // 如果线性表L存在，判断线性表L是否为空，空就返回TRUE，否则返
        回FALSE；如果线性表L不存在，返回INFEASIBLE。
80  {
81      if(!L) return INFEASIBLE;
82      if(L->next) return FALSE;
83      else return TRUE;
84
85  }
86
87
88  int ListLength(LinkList L)
89  // 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
90  {
91      if(!L) return INFEASIBLE;
92      int length = 0;
93      LinkList t = L->next;
94      while(t) {
95          length++;
96          t = t->next;
97      }
98      return length;
99  }
```

```
100
101
102 status GetElem(LinkList L,int i,ElemType &e)
103 // 如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK
    ；如果i不合法，返回ERROR；如果线性表L不存在，返回INFEASIBLE
    。
104 {
105     if(!L) return INFEASIBLE;
106     LinkList p = L->next;
107     int j = 1;
108     while(p && j < i){
109         p = p->next;
110         j++;
111     }
112     if(!p || j > i) return ERROR;
113     e = p->data;
114     return OK;
115
116 }
117
118
119 status LocateElem(LinkList L,ElemType e)
120 // 如果线性表L存在，查找元素e在线性表L中的位置序号；如果e不存
    在，返回ERROR；当线性表L不存在时，返回INFEASIBLE。
121 {
122     if(!L) return INFEASIBLE;
123     int j = 1, flag = 0;
124     LinkList p = L->next;
125     while(p){
126         if(p->data == e){
127             flag = 1;
128             break;
129         }
130         p = p->next;
131         j++;
```

```
132     }
133     if(flag) return j;
134     else return ERROR;
135
136 }
137
138
139 status PriorElem(LinkList L,ElemType e,ElemType &pre)
140 // 如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返回OK；如果没有前驱，返回ERROR；如果线性表L不存在，返回INFEASIBLE。
141 {
142     if(!L) return INFEASIBLE;
143     LinkList p = L, q = L->next;
144     int flag = 0;
145     while(q != NULL){
146         if(q->data == e){
147             flag = 1;
148             pre = p->data;
149             break;
150         }
151         p = q;
152         q = q->next;
153     }
154     if(flag && p != L)
155         return OK;
156     if(!flag) return ERROR;
157     if(p == L) return error;
158
159 }
160
161 status NextElem(LinkList L,ElemType e,ElemType &next)
162 // 如果线性表L存在，获取线性表L元素e的后继，保存在next中，返回OK；如果没有后继，返回ERROR；如果线性表L不存在，返回INFEASIBLE。
```

```
163 {
164     if(!L) return INFEASIBLE;
165     LinkList q = L->next;
166     while(q != NULL){
167         if(q->data == e){
168             break;
169         }
170         q = q->next;
171     }
172     if(q && q->next){
173         next = q->next->data;
174         return OK;
175     }
176     if(!q) return ERROR;
177     if(!q->next) return error;
178 
179 }
180 
181 
182 status ListInsert(LinkList &L,int i,ElemType e)
183 // 如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回
    OK；当插入位置不正确时，返回ERROR；如果线性表L不存在，返回
    INFEASIBLE。
184 {
185     if(!L) return INFEASIBLE;
186     LinkList p = L;
187     int j = 1;
188     while(p && j < i){
189         p = p->next;
190         ++j;
191     }
192     if(!p || j > i) return ERROR;
193     LinkList s = (LinkList)malloc(sizeof(LinkList));
194     s->data = e;
195     s->next = p->next;
```

```
196       p->next = s;
197       return OK;
198
199   }
200
201
202   status ListDelete(LinkList &L,int i,ElemType &e)
203   // 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回
      OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回
      INFEASIBLE。
204   {
205       if(!L) return INFEASIBLE;
206       LinkList p = L;
207       int j = 1;
208       while(p && j < i){
209           p = p->next;
210           ++j;
211       }
212       if(!p || !p->next || j > i) return ERROR;
213       LinkList s = p->next;
214       e = s->data;
215       p->next = p->next->next;
216       free(s);
217       return OK;
218
219   }
220
221
222   status ListTraverse(LinkList L)
223   // 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，
      返回OK；如果线性表L不存在，返回INFEASIBLE。
224   {
225       if(!L) return INFEASIBLE;
226       LinkList p = L;
227       while(p->next){
```

```
228             printf("%d",p->next->data);
229             if(p->next->next != NULL)
230                 printf(" ");
231             p = p->next;
232         }
233         return OK;
234 
235 }
236 
237 
238 status MaxSubArray(LinkList L){
239 // 返回线性表中的连续数组和的最大值
240         LinkList p = L->next;
241         int i = 0;
242         int elem[LIST_INIT_SIZE];
243         for( ; p; p = p->next, i++){
244             elem[i] = p->data;
245         }
246         int l = 0, r, max = 0, temp = elem[0];
247         for(r = 0; r < i; r++){
248             while(max < 0 && l <= r){
249                 max -= elem[l++];
250             }
251             max += elem[r];
252             if(max > temp) temp = max;
253         }
254         return temp;
255 
256 }
257 
258 status SubArrayNum(LinkList L, int k){
259 // 返回线性表中连续数组和为k的连续数组数目
260         int sum = 0, count = 0;
261         int l[LIST_INIT_SIZE];
262         l[0] = 0;
```

```
263     int h;
264     LinkList p = L->next;
265     //获取前缀和
266     for(h = 0; p; p = p->next, h++){
267         sum += p->data;
268         l[h + 1] = sum;
269     }
270     //通过前缀和之间的差值计算连续数组和
271     for(int i = 0; i < h; i++)
272         for(int j = i + 1; j < h + 1; j++)
273             if(l[j] - l[i] == k)
274                 count++;
275     return count;
276
277 }
278
279 status sortList(LinkList &L){
280 //冒泡法进行单链表排序
281     if(!L) return INFEASIBLE;
282     if(!L->next) return ERROR;
283     LinkList pre = NULL, cur = NULL, next = NULL, end = NULL,
        temp = NULL;
284     while(L->next != end){
285         for(pre = L, cur = L->next, next = L->next->next; next
              != end ; pre = pre->next, cur = cur->next, next =
             next->next){
286             if(cur->data > next->data){
287                 cur->next = next->next;
288                 pre->next = next;
289                 next->next = cur;
290                 temp = cur;
291                 cur = next;
292                 next = temp;
293             }
294         }
```

```
295             end = cur;
296         }
297         return OK;
298     }
299
300     status SaveList(LinkList L,char FileName[])
301     // 如果线性表L存在，将线性表L的的元素写到FileName文件中，返回
        OK，否则返回INFEASIBLE。
302     {
303         if(!L) return INFEASIBLE;
304         FILE *fp;
305         int i;
306         char ch;
307         if ((fp = fopen(FileName,"wb")) == NULL){
308             printf("File open error\n");
309             exit(-1);
310         }
311         ch = fgetc(fp);
312         if(ch != EOF){
313             printf("该文件不能读入！\n");
314             return ERROR;
315         }
316         if ((fp = fopen(FileName, "wb")) == NULL) {
317             printf("File open error\n");
318             exit(-1);
319         }
320         LinkList p;
321         p = L->next;
322         while (p)
323         {
324             fwrite(&p->data,sizeof(ElemType), 1, fp);
325             p = p->next;
326         }
327         fclose(fp);
328         return OK;
```

```
329
330 }
331
332 status LoadList(LinkList &L,char FileName[])
333 // 如果线性表L不存在，将FileName文件中的数据读入到线性表L中，
    返回OK，否则返回INFEASIBLE。
334 {
335     if(L) return INFEASIBLE;
336     FILE *fp;
337     L=(LinkList)malloc(sizeof(LNode));
338     int temp;
339     if ((fp = fopen(FileName, "rb")) == NULL){
340         printf("File open error\n");
341         exit(-1);
342     }
343     LinkList t=L;
344     while (fread(&temp, sizeof(ElemType), 1, fp))
345     {
346         LinkList n = (LinkList)malloc(sizeof(LNode));
347         n->data = temp;
348         t->next = n;
349         t = n;
350         t->next = NULL;
351     }
352     fclose(fp);
353     return OK;
354
355 }
356
357 status AddList(LISTS &Lists,char ListName[])
358 // 只需要在Lists中增加一个名称为ListName的空线性表，线性表数据
    又后台测试程序插入。
359 {
360         for(int i = 0; i < Lists.length; i++){
361             if(!strcmp(Lists.elem[i].name,ListName))
```

```
                return INFEASIBLE;
362         }
363     strcpy(Lists.elem[Lists.length].name,ListName);
364     Lists.elem[Lists.length].L = NULL;
365     Lists.length++;
366     return OK;
367 }
368
369 status RemoveList(LISTS &Lists,char ListName[])
370 // Lists中删除一个名称为ListName的线性表
371 {
372     for(int i = 0; i < Lists.length; i++)
373         if(!strcmp(ListName, Lists.elem[i].name)){
374             if(Lists.elem[i].L)
375             DestroyList(Lists.elem[i].L);
376             for (int j = i; j < Lists.length - 1; j++)
377             Lists.elem[j] = Lists.elem[j + 1];
378             Lists.length--;
379             return OK;
380         }
381     return ERROR;
382
383 }
384
385 int LocateList(LISTS Lists,char ListName[])
386 // 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，
    否则返回0
387 {
388     if(!Lists.elem) return INFEASIBLE;//疑问未解决
389     for(int i = 0; i < Lists.length; i++)
390         if(!strcmp(ListName, Lists.elem[i].name))
391             return i + 1;
392     return 0;
393
394 }
```

```
395
396 status TraverseList(LISTS Lists){
397 // 如果多线性表不为空，依次显示多线性表的名称，每个名称间空一
    格，返回OK；如果多线性表为空，返回INFEASIBLE。
398     if(Lists.length == 0) return INFEASIBLE;
399     printf("\n----------all names ----------------------\n")
        ;
400     for(int i = 0; i < Lists.length; i++){
401         printf("%s",Lists.elem[i].name);
402         if(i != Lists.length - 1) printf(" ");
403     }
404     printf("\n----------------- end ----------------------\
        n");
405     return OK;
406 }
407
408 status SelectList(LISTS Lists, int i){
409 // 进行线性表的选择
410     if(Lists.length == 0) return INFEASIBLE;
411     if(i < 1 || i > Lists.length) return ERROR;
412     L = Lists.elem[i - 1].L;
413     return OK;
414 }
415
416 status reverseList(LinkList &L){
417 //迭代反转思想从头开始依次反转链表
418     if(!L) return INFEASIBLE;
419     if(!L->next) return ERROR;
420     LinkList p = NULL, q = L->next, r = L->next->next;
421     while(1){
422         q->next = p;
423         if(!r) break;
424         p = q;
425         q = r;
426         r = r->next;
```

```
427     }
428     L->next = q;
429     return OK;
430 }
431
432 status RemoveNthFromEnd(LinkList L, int n){
433 //移除倒数第n个元素
434     if(!L) return INFEASIBLE;
435     int length = ListLength(L);
436     if(n < 1 || n > length) return ERROR;
437     int pre = length - n + 1;
438     LinkList p = L, tmp = NULL;
439     while(--pre){
440         p = p->next;
441     }
442     tmp = p->next;
443     p->next = tmp->next;
444     free(tmp);
445     return OK;
446 }
447
448 status CreatList(LinkList L){
449         LinkList p = L;
450         while(1){
451                 LinkList s= (LinkList)malloc(sizeof(LinkList))
                        ;
452                 scanf("%d", &s->data);
453                 if(s->data == 0)
454                         break;
455                 p->next = s;
456                 p = s;
457                 s->next = NULL;
458         }
459         return OK;
460 }
```

```
461
462 int main(){
463     int op=1;
464     int length, flag, temp, num = 0;
465     char FileName[100];
466     char Name[20];
467     Lists.length = 0;
468     while(op){
469         system("cls");
470         printf("\n\n");
471         printf("              Menu for Linear Table On Chain
            Structure \n");
472         printf("          ----------------------------
473 ------------------------------------------------\n");
474         printf("          1. InitList     线性表初始化
            13. MaxSubArray       线性表最大连续数组和获取\n");
475         printf("          2. DestroyList 线性表销毁
            14. SubArrayNum       线性表指定连续数组和数目\n");
476         printf("          3. ClearList    线性表清空
            15. sortList          线性表排序\n");
477         printf("          4. ListEmpty    线性表判空
            16. SaveList          线性表文件保存\n");
478         printf("          5. ListLength   线性表获取长度
            17. LoadList          线性表文件录入\n");
479         printf("          6. GetElem      线性表元素获取
            18. AddList           多线性表添加\n");
480         printf("          7. LocateElem   线性表元素查找
            19. RemoveList        多线性表删除\n");
481         printf("          8. PriorElem    线性表元素前驱获取
            20. LocateList        多线性表位置查找\n");
482         printf("          9. NextElem     线性表元素后继获取
            21. TraverseList      多线性表遍历\n");
483         printf("          10. ListInsert 线性表元素插入
            22. SelectList        线性表操作选择\n");
484         printf("          11. ListDelete 线性表元素删除
```

```
            23. reverseList        线性表翻转\n");
485         printf("            12. ListTraverse    线性表遍历
            24. RemoveNthFromEnd 移除倒数元素\n");
486         printf("            0. Exit             退出\n");
487         printf("            附加功能：25. 线性表迅速输入！\n");
488         printf("            ------------------------------
489 ----------------------------------------\n");
490         printf("            说明：每次操作过后请点击空格确认才能
            进行下一步操作！\n");
491         printf("\n              当前操作的线性表为：");
492         if(num < 1|| num > Lists.length){
493             if(num > Lists.length){
494                 L = NULL;
495                 num = 0;
496             }
497             printf("默认线性表");
498             if(!L)
499                 printf("(未创建)");
500             printf("\n\n\n");
501         }
502         else{
503                 printf("%s",Lists.elem[num - 1].name);
504                 if(!L)
505                 printf("(未创建)");
506             printf("\n\n\n");
507                 }
508
509         if(op > 25 || op < 0)
510             printf("上一步命令出错！请根据菜单正确输入！\n\n\n
                ");
511         printf("请选择你的操作[0~22]:");
512         scanf("%d",&op);
513         switch(op){
514                 case 25:
515                                 if(!L) {
```

```
516                                         printf("线性表未创建!
                                                \n");
517                                         getchar();getchar();
518                         break;
519                                 }
520                                 printf("请输入: ");
521                                 CreatList(L);
522                                 getchar();getchar();
523             break;
524         case 1:
525             //printf("\n----IntiList功能待实现! \n");
526             if(InitList(L) == OK) printf("线性表创建成功!
                    \n");
527             else printf("线性表创建失败! \n");
528             getchar();getchar();
529             break;
530         case 2:
531             //printf("\n----DestroyList功能待实现! \n");
532             if(DestroyList(L) == OK) printf("线性表销毁成
                    功! \n");
533             else printf("线性表销毁失败! \n");
534             getchar();getchar();
535             break;
536         case 3:
537             //printf("\n----ClearList功能待实现! \n");
538             flag = ClearList(L);
539             if(flag == OK) printf("线性表清空成功! \n");
540             else if (flag == ERROR) printf("线性表清空失
                    败! \n");
541             else printf("线性表未创建! \n");
542             getchar();getchar();
543             break;
544         case 4:
545             //printf("\n----ListEmpty功能待实现! \n");
546             if(ListEmpty(L) == OK) printf("线性表为空! \n"
```

```
                    );
547                 else if(ListEmpty(L) == INFEASIBLE) printf("线
                        性表未创建！\n");
548                                 else printf("线性表非空！\n");
549                 getchar();getchar();
550                 break;
551             case 5:
552                 //printf("\n----ListLength功能待实现！\n");
553                 length = ListLength(L);
554                 if(length != INFEASIBLE) printf("线性表的长度
                        为：%d\n", length);
555                 else printf("线性表未创建!\n");
556                 getchar();getchar();
557                 break;
558             case 6:
559                 //printf("\n----GetElem功能待实现！\n");
560                 int x, y;
561                 printf("请输入要获取元素的位置：");
562                 scanf("%d",&x);
563                 flag = GetElem(L, x, y);
564                 if(flag == INFEASIBLE) printf("线性表未创建!\n
                        ");
565                 else if(flag == OK) printf("该元素为：%d\n", y
                        );
566                 else printf("位置不合法！\n");
567                 getchar();getchar();
568                 break;
569             case 7:
570                 //printf("\n----LocateElem功能待实现！\n");
571                 int a;
572                 printf("请输入想要查找的元素：");
573                 scanf("%d",&a);
574                 flag = LocateElem(L, a);
575                 if(flag == INFEASIBLE) printf("线性表未创建!\n
                        ");
```

```
576                 else if(flag) printf("该元素存在且元素逻辑索引
                        为：%d\n", flag);
577                 else printf("该元素不存在！\n");
578                 getchar();getchar();
579                 break;
580             case 8:
581                 //printf("\n----PriorElem功能待实现！\n");
582                 printf("请输入想要查找的元素(获取前驱)：");
583                 scanf("%d",&a);
584                 flag = PriorElem(L, a, temp);
585                 if(flag == INFEASIBLE) printf("线性表未创建!\n
                        ");
586                 else if(flag == OK) printf("该元素存在且前驱元
                        素为：%d\n", temp);
587                 else if(flag == ERROR)printf("该元素不存在！\n
                        ");
588                 else printf("该元素不存在前驱！\n");
589                 getchar();getchar();
590                 break;
591             case 9:
592                 //printf("\n----NextElem功能待实现！\n");
593                 printf("请输入想要查找的元素(获取后继)：");
594                 scanf("%d",&a);
595                 flag = NextElem(L, a, temp);
596                 if(flag == INFEASIBLE) printf("线性表未创建!\n
                        ");
597                 else if(flag == OK) printf("该元素存在且后继元
                        素为：%d\n", temp);
598                 else if(flag == ERROR) printf("该元素不存在！\
                        n");
599                 else printf("该元素不存在后继！\n");
600                 getchar();getchar();
601                 break;
602             case 10:
603                 //printf("\n----ListInsert功能待实现！\n");
```

```
604                 int i, e;
605                 printf("请输入要插入的元素位置: ");
606                 scanf("%d",&i);
607                 printf("请输入要插入的元素: ");
608                 scanf("%d",&e);
609                 if(ListInsert(L, i, e) == OK) printf("线性表插
                        入成功! \n");
610                 else printf("线性表插入失败! \n");
611                 getchar();getchar();
612                 break;
613             case 11:
614                 //printf("\n----ListDelete功能待实现! \n");
615                 printf("请输入要删除的元素位置: ");
616                 scanf("%d",&i);
617                 if(ListDelete(L, i, e) == OK){
618                     printf("线性表删除成功! \n");
619                     printf("删除的元素为: %d\n",e);
620                 }
621                 else printf("线性表删除失败! \n");
622                 getchar();getchar();
623                 break;
624             case 12:
625                 //printf("\n----ListTraverse功能待实现! \n");
626                 if(!ListTraverse(L)) printf("线性表是空表! \n"
                        );
627                 getchar();getchar();
628                 break;
629             case 13:
630                 //printf("\n----MaxSubArray功能待实现! \n");
631                 if(!L) printf("线性表未创建! \n");
632                 else if (ListEmpty(L)) printf("线性表为空! \n"
                        );
633                 else printf("最大子数组之和为: %d\n",
                        MaxSubArray(L));
634                 getchar();getchar();
```

```
635                 break;
636             case 14:
637                 //printf("\n----SubArrayNum功能待实现！\n");
638                 if(!L){printf("线性表未创建！\n");getchar();
                        getchar();break;}
639                 else if (ListEmpty(L)) {printf("线性表为空！\n
                        ");getchar();getchar();break;}
640                 printf("请输入寻找的连续数组的和：");
641                 scanf("%d",&flag);
642                 printf("和为数%d的连续数组数目为：%d\n",flag,
                        SubArrayNum(L,flag));
643                 getchar();getchar();
644                 break;
645             case 15:
646                 //printf("\n----sortList功能待实现！\n");
647                 flag = sortList(L);
648                 if(flag == ERROR) printf("线性表是空表！\n");
649                 else if(flag == INFEASIBLE) printf("线性表未创
                        建！\n");
650                 else printf("线性表排序成功！\n");
651                 getchar();getchar();
652                 break;
653             case 16:
654                 //printf("\n----SaveList功能待实现！\n");
655                 printf("请输入要保存的文件名称：");
656                 scanf("%s",FileName);
657                 flag = SaveList(L, FileName);
658                 if(flag == INFEASIBLE) printf("文件读入失败！\
                        n");
659                 else if(flag == ERROR);
660                 else printf("文件读入成功！\n");
661                 getchar();getchar();
662                 break;
663             case 17:
664                 //printf("\n----LoadList功能待实现！\n");
```

```
665                 printf("请输入要录入的文件名称: ");
666                 scanf("%s",FileName);
667                 if(LoadList(L, FileName) == INFEASIBLE) printf
                        ("文件录入失败! \n");
668                 else printf("文件录入成功! \n");
669                 getchar();getchar();
670                 break;
671             case 18:
672                 //printf("\n----AddList功能待实现! \n");
673                 if(Lists.length == MAXlength) {
674                     printf("多线性表管理已满, 请清除某些线性表
                            后再操作! \n");
675                     getchar();getchar();
676                     break;
677                 }
678                 printf("请输入新增线性表的名称: ");
679                 scanf("%s",Name);
680                 flag = AddList(Lists, Name);
681                 if(flag == INFEASIBLE) printf("该名称的线性表
                        已经存在! \n");
682                 else printf("%s已成功添加! \n",Name);
683                 getchar();getchar();
684                 break;
685             case 19:
686                 //printf("\n---RemoveList功能待实现! \n");
687                 if(Lists.length == 0) {
688                     printf("多线性表管理已空, 请添加某些线性表
                            后再操作! \n");
689                     getchar();getchar();
690                     break;
691                 }
692                 printf("请输入删除线性表的名称: ");
693                 scanf("%s",Name);
694                 flag = RemoveList(Lists, Name);
695                 if(flag == OK) printf("%s已成功删除! \n",Name);
```

```
696                 else printf("线性表不存在！\n");
697                 getchar();getchar();
698                 break;
699             case 20:
700                 //printf("\n---LocateList功能待实现！\n");
701                 printf("请输入查找线性表的名称：");
702                 scanf("%s",Name);
703                 if(LocateList(Lists, Name)) printf("该线性表的
                        逻辑索引为：%d\n", LocateList(Lists, Name))
                        ;
704                 else printf("线性表查找失败！\n");
705                 getchar();getchar();
706                 break;
707             case 21:
708                 //printf("\n----TraverseList功能待实现！\n");
709                 if(TraverseList(Lists) == INFEASIBLE) printf("
                        多线性表为空！\n");
710                 getchar();getchar();
711                 break;
712             case 22:
713                 //printf("\n----SelectList功能待实现！\n");
714                 printf("请选择要处理的线性表的逻辑索引：");
715                 scanf("%d",&flag);
716                 if(SelectList(Lists, flag) == OK) {
717                     printf("已选取成功！\n");
718                     num = flag;
719                 }
720                 else printf("选取失败！\n");
721                 getchar();getchar();
722                 break;
723             case 23:
724                 //printf("\n----reverseList功能待实现！\n");
725                 flag = reverseList(L);
726                 if(flag == INFEASIBLE) printf("线性表未创建！\
                        n");
```

```
727                     else if(flag == ERROR) printf("该线性表为空！\
                            n");
728                     else printf("链表反转成功！\n");
729                     getchar();getchar();
730                     break;
731                 case 24://printf("\n----RemoveNthFromEnd功能待实
                        现！\n");
732                     int n;
733                     printf("请输入想要删除倒数第几个元素：");
734                     scanf("%d",&n);
735                     flag = RemoveNthFromEnd(L, n);
736                     if(flag == INFEASIBLE) printf("线性表未创建！\
                            n");
737                     else if(flag == ERROR) printf("位置不合法！\n"
                            );
738                     else printf("删除成功！\n");
739                     getchar();getchar();
740                     break;
741                 case 0:
742                     break;
743             }//end of switch
744         }//end of while
745         printf("欢迎下次再使用本系统！\n");
746         return 0;
747 }//end of main()
```

# 6 附录 C 基于二叉链表二叉树实现的源程序

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

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
#define MAXlength 10

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

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

typedef struct{  //线性表的集合类型定义
    struct { char name[30];
        BiTree L;
    } elem[11];
    int length;
}TREES;
TREES trees;        //线性表集合的定义TREES

BiTree T;
```

```
34  int result = 0, count = 0;
35  BiTree BiTreestack[100];
36  int top;
37  BiTree BiTreequeue[100];
38  int l, r;
39  int i, k, flag1[1000], flag2 = 0;
40
41  status checkKey(TElemType definition[]) {
42      i = 0;
43      while(definition[i++].key != -1);
44      for (int j = 0; j < i; j++) {
45          for (int k = j + 1; k < i; k++) {
46              if (definition[k].key == definition[j].key &&
                    definition[k].key != 0)
47                  return 0;
48          }
49      }
50      return 1;
51  }
52
53  status CreateBiTree(BiTree& T, TElemType definition[]) {
54  //二叉树的创建，通过先序遍历的方式创建
55      if(result == 0) result = checkKey(definition);
56      if(result == 0) return ERROR;
57      if(definition[count].key != -1) {
58          if(definition[count].key != 0) {
59              T = (BiTree)malloc(sizeof(BiTNode));
60              T->data.key = definition[count].key;
61              strcpy(T->data.others, definition[count].others);
62              count++;
63              CreateBiTree(T->lchild, definition);
64              CreateBiTree(T->rchild, definition);
65          }
66          else {
67              T = NULL;
```

```
68                 count++;
69             }
70         }
71         return OK;
72
73 }
74
75 status ClearBiTree(BiTree &T)
76 //将二叉树设置成空，并删除所有结点，释放结点空间
77 {
78         if(!T) return OK;
79         ClearBiTree(T->lchild);
80         ClearBiTree(T->rchild);
81         free(T);
82         T = NULL;
83         return OK;
84
85 }
86
87 status DestroyBiTree(BiTree &T)
88 //将二叉树销毁
89 {
90         if(!T) return OK;
91         DestroyBiTree(T->lchild);
92         DestroyBiTree(T->rchild);
93         free(T);
94         T = NULL;
95         return OK;
96
97 }
98
99 int MAX(int a, int b){
100        if(a >= b) return a;
101        else return b;
102 }
```

```
103
104 int BiTreeDepth(BiTree T)
105 //求二叉树T的深度
106 {
107     if(!T) return  ERROR;
108     return MAX(BiTreeDepth(T->lchild), BiTreeDepth(T->rchild))
            + 1;
109
110 }
111
112
113
114 BiTNode* LocateNode(BiTree T,KeyType e)
115 //查找结点
116 {
117     if(!T) return NULL;
118     if(T->data.key == e) return T;
119     if(LocateNode(T->lchild, e))
120         return LocateNode(T->lchild, e);
121     else
122         return LocateNode(T->rchild, e);
123
124 }
125
126 status Assign(BiTree &T,KeyType e,TElemType value)
127 //实现结点赋值。
128 {
129     BiTree p = LocateNode(T, value.key);
130     BiTree t = LocateNode(T, e);
131     if(t == NULL || (value.key!= e && p != NULL)) return ERROR
            ;
132     t->data.key = value.key;
133     strcpy(t->data.others, value.others);
134     return OK;
135
```

```
136 }
137
138 BiTNode* GetSibling(BiTree T,KeyType e)
139 //实现获得兄弟结点
140 {
141     if(!T) return NULL;
142     if(T->lchild && T->lchild->data.key == e) return T->rchild
        ;
143     if(T->rchild && T->rchild->data.key == e) return T->lchild
        ;
144     if(GetSibling(T->lchild, e))
145         return GetSibling(T->lchild, e);
146     else
147         return GetSibling(T->rchild, e);
148
149 }
150
151 BiTNode* GetFabling(BiTree T,KeyType e)
152 //实现获得父亲结点
153 {
154     if(!T) return NULL;
155     if(T->lchild && T->lchild->data.key == e) return T;
156     if(T->rchild && T->rchild->data.key == e) return T;
157     if(GetFabling(T->lchild, e))
158         return GetFabling(T->lchild, e);
159     else
160         return GetFabling(T->rchild, e);
161
162 }
163
164 status InsertNode(BiTree &T,KeyType e,int LR,TElemType c)
165 //插入结点。
166 {
167     if(!T) return INFEASIBLE;
168     BiTree t = LocateNode(T, e);
```

```
169     BiTree p = (BiTree)malloc(sizeof(BiTNode));
170     p->data.key = c.key;
171     strcpy(p->data.others, c.others);
172     if(LocateNode(T, c.key)) return ERROR;
173     if(LR == -1){
174         p->rchild = T;
175         p->lchild = NULL;
176         T = p;
177         return OK;
178     }
179     if(!t) return ERROR;
180     if(LR == 0){
181         p->rchild = t->lchild;
182         p->lchild = NULL;
183         t->lchild = p;
184         return OK;
185     }
186     else if(LR == 1){
187         p->rchild = t->rchild;
188         p->lchild = NULL;
189         t->rchild = p;
190         return OK;
191     }
192     free(p);
193
194 }
195
196 status DeleteNode(BiTree &T,KeyType e)
197 //删除结点。
198 {
199     BiTree tmp = (BiTree)malloc(sizeof(BiTNode));
200     BiTree t = NULL, p = NULL, q = NULL, l = NULL;
201     int flag = 0;
202     tmp->lchild = T;
203     tmp->rchild = NULL;
```

```
204     t = GetFabling(tmp, e);
205     // printf("%d",t->data.key);
206     if(!t) return ERROR;
207     if(t->lchild && t->lchild->data.key == e) {q = t->lchild;
            flag = 1;}
208     else if(t->rchild && t->rchild->data.key == e) {q = t->
            rchild; flag = 0;}
209     if(!q->lchild && !q->rchild){
210         free(q);
211         if(flag) t->lchild = NULL;
212         else t->rchild = NULL;
213     }
214     else if(!q->lchild && q->rchild){
215         if(flag) t->lchild = q->rchild;
216         else t->rchild = q->rchild;
217         free(q);
218     }
219     else if(q->lchild && !q->rchild){
220         if(flag) t->lchild = q->lchild;
221         else t->rchild = q->lchild;
222         free(q);
223     }
224     else{
225         l = q->lchild;
226         while(1){
227             if(l){
228                 BiTreestack[top++] = l;
229                 l = l->lchild;
230             }
231             else{
232                 l = BiTreestack[--top];
233                 if(!l->rchild && !top)
234                     break;
235                 l = l->rchild;
236             }
```

```
237             }
238             l->rchild = q->rchild;
239             if(flag) t->lchild = q->lchild;
240             else t->rchild = q->lchild;
241             free(q);
242         }
243         T = tmp->lchild;
244         free(tmp);
245         return OK;
246 
247 }
248 
249 void visit(BiTree T)
250 {
251     printf("%d,%s",T->data.key,T->data.others);
252 }
253 
254 status PreOrderTraverse(BiTree T,void (*visit)(BiTree))
255 //先序遍历二叉树T
256 {
257     if(!T) return INFEASIBLE;
258     visit(T);
259     PreOrderTraverse(T->lchild, visit);
260     PreOrderTraverse(T->rchild, visit);
261     return OK;
262 
263 }
264 
265 status InOrderTraverse(BiTree T,void (*visit)(BiTree))
266 //中序遍历二叉树T
267 {
268     if(!T) return INFEASIBLE;
269     BiTree p = T;
270     while(p || top){
271         if(p){
```

```
272                 BiTreestack[top++] = p;
273                 p = p->lchild;
274             }
275             else{
276                 p = BiTreestack[--top];
277                 visit(p);
278                 p = p->rchild;
279             }
280         }
281 
282 }
283 
284 status PostOrderTraverse(BiTree T,void (*visit)(BiTree))
285 //后序遍历二叉树T
286 {
287     if(!T) return ERROR;
288     PostOrderTraverse(T->lchild, visit);
289     PostOrderTraverse(T->rchild, visit);
290     visit(T);
291     return OK;
292 
293 }
294 
295 status LevelOrderTraverse(BiTree T,void (*visit)(BiTree))
296 //按层遍历二叉树T
297 {
298     BiTree p = T;
299     BiTreequeue[0] = p;
300     l = 0, r = 1;
301     while(l != r){
302         p = BiTreequeue[l++];
303         visit(p);
304         if(p->lchild) BiTreequeue[r++] = p->lchild;
305         if(p->rchild) BiTreequeue[r++] = p->rchild;
306     }
```

```
307     return OK;
308
309 }
310
311 status SaveBiTree(BiTree T, char FileName[])
312 //将二叉树的结点数据写入到文件FileName中
313 {
314     if(!T) return INFEASIBLE;
315     FILE *fp;
316     char ch;
317     if ((fp = fopen(FileName,"wb")) == NULL)
318     {
319         printf("File open error!\n ");
320         return ERROR;
321     }
322     ch = fgetc(fp);
323     if(ch != EOF){
324         printf("该文件不能读入! \n");
325         return ERROR;
326     }
327     TElemType t;
328     t.key = 0;
329     BiTree s;
330     BiTreestack[top++] = T;
331     while(top)
332     {
333         s = BiTreestack[--top];
334         if(!s)
335         {
336             fwrite(&t, sizeof(TElemType), 1, fp);
337             continue;
338         }
339         BiTreestack[top++] = s->rchild;
340         BiTreestack[top++] = s->lchild;
341         fwrite(&s->data, sizeof(TElemType), 1, fp);
```

```
342         }
343         fclose(fp);
344         return OK;
345
346     }
347
348     status dfs(BiTree &T, TElemType definition[])
349     {
350         i++;
351         if(definition[i].key == -1) return OK;
352         if(definition[i].key == 0) T = NULL;
353         else
354         {
355             T = (BiTNode*)malloc(sizeof(BiTNode));
356             T->data = definition[i];
357             if(flag1[definition[i].key]) flag2 = 1;
358             flag1[T->data.key] = 1;
359             dfs(T->lchild, definition);
360             dfs(T->rchild, definition);
361         }
362         return OK;
363     }
364
365     status LoadBiTree(BiTree &T,  char FileName[])
366     //读入文件FileName的结点数据，创建二叉树
367     {
368         if(T) return INFEASIBLE;
369         FILE *fp;
370         if ((fp = fopen(FileName,"rb")) == NULL)
371         {
372             printf("File open error!\n");
373             return ERROR;
374         }
375         i = 0;
376         TElemType definition[100];
```

```
377     while(fread(&definition[i++], sizeof(TElemType), 1, fp));
378     definition[i].key = -1;
379     i = -1;
380     dfs(T, definition);
381     fclose(fp);
382     if(flag2) return ERROR;
383     return OK;
384
385 }
386
387 status BiTreeEmpty(BiTree T){
388     if(!T) return FALSE;
389     else return TRUE;
390 }
391
392 status AddList(TREES &trees,char ListName[])
393 // 需要在TREES中增加一个名称为ListName的空线性表
394 {
395         for(int i = 0; i < trees.length; i++){
396                 if(!strcmp(trees.elem[i].name,ListName))
                        return INFEASIBLE;
397         }
398     strcpy(trees.elem[trees.length].name,ListName);
399     trees.elem[trees.length].L = NULL;
400     trees.length++;
401     return OK;
402 }
403
404 status DestoryList(TREES &trees,char ListName[])
405 // TREES中删除一个名称为ListName的线性表
406 {
407     for(int i = 0; i < trees.length; i++)
408         if(!strcmp(ListName, trees.elem[i].name)){
409             if(trees.elem[i].L)
410                 DestroyBiTree(trees.elem[i].L);
```

```
411                 for (int j = i; j < trees.length - 1; j++)
412                     trees.elem[j] = trees.elem[j + 1];
413                 trees.length--;
414                 return OK;
415             }
416         return ERROR;
417
418     }
419
420     int LocateList(TREES trees,char ListName[])
421     // 在TREES中查找一个名称为ListName的线性表，成功返回逻辑序号，
            否则返回0
422     {
423         if(!trees.elem) return INFEASIBLE;//疑问未解决
424         for(int i = 0; i < trees.length; i++)
425             if(!strcmp(ListName, trees.elem[i].name))
426                 return i + 1;
427         return 0;
428
429     }
430
431     status TraverseList(TREES trees){
432     // 如果多线性表不为空，依次显示多线性表的名称，每个名称间空一
            格，返回OK；如果多线性表为空，返回INFEASIBLE。
433         if(trees.length == 0) return INFEASIBLE;
434         printf("\n----------all names ----------------------\n")
                ;
435         for(int i = 0; i < trees.length; i++){
436             printf("%s",trees.elem[i].name);
437             if(i != trees.length - 1) printf(" ");
438         }
439         printf("\n----------------- end ----------------------\
                n");
440         return OK;
441     }
```

```
442
443 status SelectList(TREES trees, int i){
444 // 进行线性表的选择
445     if(trees.length == 0) return INFEASIBLE;
446     if(i < 1 || i > trees.length) return ERROR;
447     T = trees.elem[i - 1].L;
448     return OK;
449 }
450
451 int MaxPathSum(BiTree T){
452 //返回最大路径和
453     if(!T) return  ERROR;
454     return MAX(MaxPathSum(T->lchild), MaxPathSum(T->rchild)) +
          T->data.key;
455 }
456
457 BiTree LowestCommonAncestor(BiTree T, int e1, int e2){
458 //返回公共祖先
459     if(!T || T->data.key == e1 || T->data.key == e2) return T;
460     BiTree l = LowestCommonAncestor(T->lchild, e1, e2);
461     BiTree r = LowestCommonAncestor(T->rchild, e1, e2);
462     if(l && !r) return l;
463     if(!l && r) return r;
464     if(!l && !r) return NULL;
465     if(l && r) return T;
466 }
467
468 status InvertTree(BiTree &T){
469 //二叉树翻转
470     BiTree p = T, t;
471     BiTreequeue[0] = p;
472     l = 0, r = 1;
473     while(l != r){
474         p = BiTreequeue[l++];
475         t = p->lchild;
```

```
476             p->lchild = p->rchild;
477             p->rchild = t;
478             if(p->lchild) BiTreequeue[r++] = p->lchild;
479             if(p->rchild) BiTreequeue[r++] = p->rchild;
480         }
481         return OK;
482     }
483
484     status ClearList(BiTree &T){
485             ClearBiTree(T);
486             return OK;
487     }
488
489     int main(){
490         int op=1;
491         int length, num = 0, LR;
492         int ans, e, ee;
493         char FileName[100];
494         char Name[20];
495         TElemType definition[100], value;
496         BiTree t = NULL;
497         trees.length = 0;
498         while(op){
499             system("cls");
500             printf("\n\n");
501             printf("                                    Menu for
                Binary Tree   \n");
502             printf("          ---------------------------------
503     ------------------------------------------\n");
504             printf("          1. CreateBiTree   二叉树创建
                                  14. LevelOrderTraverse     二叉树层
                序遍历\n");
505             printf("          2. DestroyBiTree  二叉树销毁
                                  15. MaxPathSum             二叉树最
                大路径和\n");
```

```
506         printf("          3. ClearBiTree    二叉树清空
                              16. LowestCommonAncestor  二叉树最
            近祖先\n");
507         printf("          4. BiTreeEmpty    二叉树判空
                              17. InvertTree            二叉树翻
            转\n");
508         printf("          5. BiTreeDepth    二叉树获取深度
                            18. SaveList                二叉树文件保
            存\n");
509         printf("          6. LocateNode     二叉树查找结点
                            19. LoadList                二叉树文件录
            入\n");
510         printf("          7. Assign         二叉树结点赋值
                            20. AddList                 多二叉树表添
            加\n");
511         printf("          8. GetFabling     二叉树获取父亲结点
                        21. DestroyList           多二叉树表销毁\n"
            );
512         printf("          9. InsertNode     二叉树插入结点
                             22. LocateList             多二叉树表位
            置查找\n");
513         printf("          10. DeleteNode    二叉树删除结点
                            23. TraverseList            多二叉树表遍
            历\n");
514         printf("          11. PreOrderTraverse    二叉树前序遍
            历        24. SelectList            二叉树操作选择\n
            ");
515         printf("          12. InOrderTraverse     二叉树中序遍
            历        25. ClearList             多二叉树表清空\n
            ");
516         printf("          13. PostOrderTraverse     二叉树后序
            遍历      26. GetSibling      二叉树获取兄弟结点\n");
517         printf("          0. Exit           退出\n");
518         printf("          ------------------------------------
519 ----------------------------------------\n");
```

```
520         printf("          说明：每次操作过后请点击空格确认才能
                进行下一步操作！\n");
521         printf("\n          当前操作的二叉树为：");
522         if(num < 1|| num > trees.length){
523             if(num > trees.length){
524                 T = NULL;
525                 num = 0;
526             }
527             printf("默认二叉树");
528             if(!T)
529                 printf("(未创建)");
530             printf("\n\n\n");
531         }
532         else{
533                 printf("%s",trees.elem[num - 1].name);
534                 if(!T)
535                 printf("(未创建)");
536             printf("\n\n\n");
537                 }
538         if(op > 26 || op < 0)
539             printf("上一步命令出错！请根据菜单正确输入！\n\n\n
                ");
540         printf("请选择你的操作[0~26]:");
541         scanf("%d",&op);
542         switch(op){
543             case 1:
544                 //printf("\n----CreateBiTree功能待实现！\n");
545                 if(T){
546                     printf("该二叉树已存在！\n");
547                     getchar();getchar();
548                     break;
549                 }
550                 i = 0;
551                 printf("请输入合法先序序列（每个结点对应一个整
                    型的关键字和一个字符串，当关键字为0时，表示
```

```
                空子树，为-1表示输入结束）：");
552             do {
553                 scanf("%d%s", &definition[i].key,
                        definition[i].others);
554             } while (definition[i++].key != -1);
555             ans = CreateBiTree(T,definition);
556             count = 0;
557             if(ans == ERROR) printf("关键字不唯一！创建失
                    败！\n");
558             else printf("二叉树创建成功！\n");
559             getchar();getchar();
560             break;
561         case 2:
562             //printf("\n----DestroyBiTree功能待实现！\n");
563             if(!T) {
564                 printf("二叉树为空！\n");
565                 getchar();getchar();
566                 break;
567             }
568             ans = DestroyBiTree(T);
569             if(ans == OK) printf("二叉树销毁成功！\n");
570             else printf("二叉树销毁失败！\n");
571             getchar();getchar();
572             break;
573         case 3:
574             //printf("\n----ClearBiTree功能待实现！\n");
575             ans = ClearBiTree(T);
576             if(ans == OK) printf("二叉树清空成功！\n");
577             else printf("二叉树清空失败！\n");
578             getchar();getchar();
579             break;
580             case 4:
581             //printf("\n----BiTreeEmpty功能待实现！\n");
582             ans = BiTreeEmpty(T);
583             if(ans == FALSE) printf("二叉树为空！\n");
```

```
584                 else printf("二叉树不为空！\n");
585                 getchar();getchar();
586                 break;
587             case 5:
588                 //printf("\n----BiTreeDepth功能待实现！\n");
589                 length = BiTreeDepth(T);
590                 if(length) printf("该二叉树的深度为%d！\n",
                        length);
591                 else printf("二叉树为空！\n");
592                 getchar();getchar();
593                 break;
594             case 6:
595                 //printf("\n----LocateNode功能待实现！\n");
596                 if(!T) {
597                     printf("二叉树为空！\n");
598                     getchar();getchar();
599                     break;
600                 }
601                 printf("请输入想要查询的结点关键字：");
602                 scanf("%d",&e);
603                 t = LocateNode(T, e);
604                 if(t == NULL) printf("该节点不存在！\n");
605                 else {
606                     printf("该节点存在！结点信息为：%s\n",t->
                            data.others);
607                 }
608                 t = NULL;
609                 getchar();getchar();
610                 break;
611             case 7:
612                 //printf("\n----Assign功能待实现！\n");
613                 if(!T) {
614                     printf("二叉树为空！\n");
615                     getchar();getchar();
616                     break;
```

```
617                 }
618                 printf("请输入想要赋值的节点关键字:");
619                 scanf("%d", &e);
620                 printf("请输入关键字: ");
621                 scanf("%d", &value.key);
622                 printf("请输入结点信息: ");
623                 scanf("%s", value.others);
624                 ans = Assign(T, e, value);
625                 if(ans == OK) printf("结点赋值成功!\n");
626                 else printf("结点复制失败（请检查该关键字是否
                        存在或者赋值关键字是否重复）!\n");
627                 getchar();getchar();
628                 break;
629             case 8:
630                 //printf("\n----GetSibling功能待实现! \n");
631                 if(!T) {
632                     printf("二叉树为空! \n");
633                     getchar();getchar();
634                     break;
635                 }
636                 printf("请输入要获取兄弟结点的关键字: ");
637                 scanf("%d",&e);
638                 t = GetSibling(T, e);
639                 if(t == NULL) printf("该结点不存在或不存在兄弟
                        结点!\n");
640                 else{
641                     printf("该元素兄弟结点获取成功! \n");
642                     printf("该结点的兄弟结点关键字为 %d 结点信
                            息为: %s\n", t->data.key, t->data.
                            others);
643                 }
644                 getchar();getchar();
645                 break;
646             case 9:
647                 //printf("\n----InsertNode功能待实现! \n");
```

```
648                 if(!T) {
649                     printf("二叉树为空！\n");
650                     getchar();getchar();
651                     break;
652                 }
653                 printf("请输入想要插入的结点关键字：");
654                 scanf("%d", &e);
655                 printf("请输入关键字：");
656                 scanf("%d", &value.key);
657                 printf("请输入结点信息：");
658                 scanf("%s", value.others);
659                 printf("请输入插入方式（LR为0或者1时作为关键字
                        为e的结点的左或右孩子结点，LR为-1时，作为根
                        结点插入，原根结点作为c的右子树）：");
660                 scanf("%d", &LR);
661                 ans = InsertNode(T, e, LR, value);
662                 if(ans == ERROR) printf("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）!\n
                        ");
663                 else printf("结点插入成功！\n");
664                 getchar();getchar();
665                 break;
666             case 10:
667                 //printf("\n----DeleteNode功能待实现！\n");
668                 if(!T) {
669                     printf("二叉树为空！\n");
670                     getchar();getchar();
671                     break;
672                 }
673                 printf("请输入想要删除的结点关键字：");
674                 scanf("%d", &e);
675                 ans = DeleteNode(T, e);
676                 if(ans == ERROR) printf("结点删除失败（请检查
                        该关键字是否存在）!\n");
677                 else printf("结点删除成功！\n");
```

```
678                 getchar();getchar();
679                 break;
680             case 11:
681                 //printf("\n----PreOrderTraverse功能待实现! \n
                        ");
682                 if(!T) {
683                     printf("二叉树为空! \n");
684                     getchar();getchar();
685                     break;
686                 }
687                 printf("先序遍历二叉树的结果: \n");
688                 PreOrderTraverse(T, visit);
689                 getchar();getchar();
690                 break;
691             case 12:
692                 //printf("\n----InOrderTraverse功能待实现! \n
                        ");
693                 if(!T) {
694                     printf("二叉树为空! \n");
695                     getchar();getchar();
696                     break;
697                 }
698                 printf("中序遍历二叉树的结果: \n");
699                 InOrderTraverse(T, visit);
700                 getchar();getchar();
701                 break;
702             case 13:
703                 //printf("\n----PostOrderTraverse功能待实现! \
                        n");
704                 if(!T) {
705                     printf("二叉树为空! \n");
706                     getchar();getchar();
707                     break;
708                 }
709                 printf("后序遍历二叉树的结果: \n");
```

```
710                 PostOrderTraverse(T, visit);
711                 getchar();getchar();
712                 break;
713             case 14:
714                 //printf("\n----LevelOrderTraverse功能待实现!
                        \n");
715                 if(!T) {
716                     printf("二叉树为空! \n");
717                     getchar();getchar();
718                     break;
719                 }
720                 printf("层序遍历二叉树的结果: \n");
721                 LevelOrderTraverse(T, visit);
722                 getchar();getchar();
723                 break;
724             case 15:
725                 //printf("\n----MaxPathSum功能待实现! \n");
726                 if(!T) {
727                     printf("二叉树为空! \n");
728                     getchar();getchar();
729                     break;
730                 }
731                 length = MaxPathSum(T);
732                 printf("根节点到叶子结点的最大路径和为: %d\n",
                        length);
733                 getchar();getchar();
734                 break;
735             case 16:
736                 //printf("\n----LowestCommonAncestor功能待实
                        现! \n");
737                 if(!T) {
738                     printf("二叉树为空! \n");
739                     getchar();getchar();
740                     break;
741                 }
```

```
742                 printf("请输入第一个结点: ");
743                 scanf("%d", &e);
744                 printf("请输入第二个结点: ");
745                 scanf("%d", &ee);
746                 t = LocateNode(T, e);
747                 if(!t){
748                     printf("第一个结点不存在! \n");
749                     getchar();getchar();
750                     break;
751                 }
752 //                  else if(t == T){
753 //                      printf("第一个结点为根节点, 不存在祖先!
        \n");
754 //                      t = NULL;
755 //                      getchar();getchar();
756 //                      break;
757 //                  }
758                 t = LocateNode(T, ee);
759                 if(!t){
760                     printf("第二个结点不存在! \n");
761                     getchar();getchar();
762                     break;
763                 }
764 //                  else if(t == T){
765 //                      printf("第二个结点为根节点, 不存在祖先!
        \n");
766 //                      t = NULL;
767 //                      getchar();getchar();
768 //                      break;
769 //                  }
770                 t = LowestCommonAncestor(T,e,ee);
771                 printf("两结点最近公共祖先的关键字为: %d,  结
                    点信息为: %s\n", t->data.key, t->data.
                    others);
772                 t = NULL;
```

```
773                 getchar();getchar();
774                 break;
775             case 17:
776                 //printf("\n----InvertTree功能待实现! \n");
777                 if(!T) {
778                     printf("二叉树为空! \n");
779                     getchar();getchar();
780                     break;
781                 }
782                 InvertTree(T);
783                 printf("二叉树翻转成功! \n");
784                 getchar();getchar();
785                 break;
786             case 18:
787                 //printf("\n----SaveList功能待实现! \n");
788                 printf("请输入要保存的文件名称: ");
789                 scanf("%s",FileName);
790                 ans = SaveBiTree(T, FileName);
791                 if(ans == INFEASIBLE) printf("二叉树不存在!文
                        件保存失败! \n");
792                 else if(ans == ERROR);
793                 else printf("文件保存成功! \n");
794                 getchar();getchar();
795                 break;
796             case 19:
797                 //printf("\n----LoadList功能待实现! \n");
798                 printf("请输入要录入的文件名称: ");
799                 scanf("%s", FileName);
800                 if(LoadBiTree(T, FileName) == INFEASIBLE)
                        printf("二叉树存在!文件录入失败\n");
801                 else printf("文件录入成功! \n");
802                 getchar();getchar();
803                 break;
804             case 20:
805                 //printf("\n----AddList功能待实现! \n");
```

```
806                 if(trees.length == MAXlength) {
807                     printf("多二叉树表管理已满，请清除某些二叉
                            树后再操作！\n");
808                     getchar();getchar();
809                     break;
810                 }
811                 printf("请输入新增二叉树的名称：");
812                 scanf("%s",Name);
813                 ans = AddList(trees, Name);
814                 if(ans == INFEASIBLE) printf("该名称的二叉树已
                        经存在!\n");
815                 else printf("%s已成功添加！\n",Name);
816                 getchar();getchar();
817                 break;
818             case 21:
819                 //printf("\n---DestoryList功能待实现！\n");
820                 if(trees.length == 0) {
821                     printf("多二叉树表管理已空，请添加某些二叉
                            树后再操作！\n");
822                     getchar();getchar();
823                     break;
824                 }
825                 printf("请输入销毁二叉树的名称：");
826                 scanf("%s",Name);
827                 ans = DestoryList(trees, Name);
828                 if(ans == OK)printf("%s已成功销毁！\n",Name);
829                 else printf("二叉树不存在！\n");
830                 getchar();getchar();
831                 break;
832             case 22:
833                 //printf("\n---LocateList功能待实现！\n");
834                 printf("请输入查找二叉树的名称：");
835                 scanf("%s",Name);
836                 if(LocateList(trees, Name)) printf("该二叉树的
                        逻辑索引为：%d\n", LocateList(trees, Name))
```

```
                ;
837             else printf("二叉树查找失败！\n");
838             getchar();getchar();
839             break;
840         case 23:
841             //printf("\n----TraverseList功能待实现！\n");
842             if(TraverseList(trees) == INFEASIBLE) printf("
                    多二叉树表为空！\n");
843             getchar();getchar();
844             break;
845         case 24:
846             //printf("\n----SelectList功能待实现！\n");
847             printf("请选择要处理的二叉树的逻辑索引：");
848             scanf("%d", &ans);
849             if(SelectList(trees, ans) == OK) {
850                 printf("二叉树已选取成功！\n");
851                 num = ans;
852             }
853             else printf("二叉树选取失败！\n");
854             getchar();getchar();
855             break;
856         case 25:
857             //printf("\n----ClearBiTree功能待实现！\n");
858             if(!T) {
859                 printf("二叉树为空！\n");
860                 getchar();getchar();
861                 break;
862             }
863             ClearList(T);
864             printf("二叉树清空成功！\n");
865             getchar();getchar();
866             break;
867                     case 26:
868             //printf("\n----GetFabling功能待实现！\n");
869             if(!T) {
```

```
870                         printf("二叉树为空！\n");
871                         getchar();getchar();
872                         break;
873                     }
874                     printf("请输入要获取父亲结点的关键字：");
875                     scanf("%d",&e);
876                     if(T->data.key == e){
877                         printf("该结点为根结点！");
878                     }
879                     t = GetFabling(T, e);
880                     if(t == NULL) printf("该结点不存在父亲节点!\n"
                            );
881                     else{
882                         printf("该元素父亲节点获取成功！\n");
883                         printf("该结点的父亲结点关键字为 %d 结点信
                                息为： %s", t->data.key, t->data.others
                                );
884                     }
885                     t = NULL;
886                     getchar();getchar();
887                     break;
888                 case 0:
889                     break;
890             }//end of switch
891         }//end of while
892         printf("欢迎下次再使用本系统！\n");
893         return 0;
894 }//end of main()
```

# 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
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 #define MAXlength 10
13
14 typedef int status;
15 typedef int KeyType;
16 typedef enum {DG,DN,UDG,UDN} GraphKind;
17 typedef struct {
18       KeyType  key;
19       char others[20];
20 } VertexType; //顶点类型定义
21
22
23 typedef struct ArcNode {          //表结点类型定义
24          int adjvex;               //顶点位置编号
25          struct ArcNode  *nextarc;          //下一个表结点指针
26 } ArcNode;
27
28 typedef struct VNode{                              //头结点及其数
       组类型定义
29          VertexType data;                  //顶点信息
30          ArcNode *firstarc;                 //指向第一条弧
31        } VNode, AdjList[MAX_VERTEX_NUM];
32
```

```
33  typedef  struct {  //邻接表的类型定义
34       AdjList vertices;                //头结点数组
35       int vexnum,arcnum;                //顶点数、弧数
36       GraphKind  kind;          //图的类型
37     } ALGraph;
38
39
40  typedef struct {//森林的定义
41       struct {
42            char name[30];
43            ALGraph G;
44       }elem[11];
45       int length;
46       int listsize;
47  }GRAPHS;
48  GRAPHS Graphs;
49
50  int num = 10;
51
52  status check1(VertexType V[]){
53  //判断结点集里是否有重复结点
54       int i = 0;
55       if(V[i].key == -1) return 1;
56       while(V[i].key != -1){
57            for(int j = 0; j < i; j++)
58                 if(V[j].key == V[i].key)
59                      return 1;
60            i++;
61       }
62       if(i > 20) return 1;
63       return 0;
64  }
65
66  void deleteVR(KeyType VR[][2], int i, int &num){
67       for(int k = i; k <= num; k++){
```

```
68          VR[k][0] = VR[k + 1][0];
69          VR[k][1] = VR[k + 1][1];
70      }
71      num--;
72      return;
73  }
74
75  status check2(VertexType V[], KeyType VR[][2]){
76  //判断是否有重复，错乱，多余的边
77      int flag = 0;
78      for(int i = 0; VR[i][0] != -1; i++){
79          for(int j = 0; V[j].key != -1; j++){
80              if(VR[i][0] == V[j].key){
81                  flag = 1;
82                  break;
83              }
84          }
85          if(!flag) return 1;
86          flag = 0;
87      }
88      for(int i = 0; VR[i][1] != -1; i++){
89          for(int j = 0; V[j].key != -1; j++){
90              if(VR[i][1] == V[j].key){
91                  flag = 1;
92                  break;
93              }
94          }
95          if(!flag) return 1;
96          flag = 0;
97      }
98      int i = 0, num = 0;
99      while(VR[num++][0] != -1);
100     while(VR[i][0] != -1){
101         if(VR[i][0] == VR[i][1]){
102             deleteVR(VR, i, num);
```

```
103                 continue;
104             }
105             for(int j = 0; j < i; j++){
106                 if(VR[j][0] == VR[i][0] && VR[j][1] == VR[i][1]){
107                     deleteVR(VR, i, num);
108                     i--;
109                     break;
110                 }
111                 if(VR[j][0] == VR[i][1] && VR[j][1] == VR[i][0]){
112                     deleteVR(VR, i, num);
113                     i--;
114                     break;
115                 }
116             }
117             i++;
118         }
119         return 0;
120     }
121
122     status CreateCraph(ALGraph &G,VertexType V[],KeyType VR[][2])
123     /*根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR
124     如果有相同的关键字，返回ERROR。*/
125     {
126         if(check1(V)) return ERROR;
127         if(check2(V, VR)) return ERROR;
128         G.kind = UDG;
129         G.vexnum = 0, G.arcnum = 0;
130         int m;
131         while(V[G.vexnum].key != -1){
132             G.vertices[G.vexnum].data.key = V[G.vexnum].key;
133             strcpy(G.vertices[G.vexnum].data.others, V[G.vexnum].
                    others);
134             G.vexnum++;
135         }
136         for(int i = 0; i < G.vexnum; i++){
```

```
137             G.vertices[i].firstarc = NULL;
138         }
139         while(VR[G.arcnum][0] != -1){
140             for(int i = 0; i < G.vexnum; i++){
141                 if(G.vertices[i].data.key == VR[G.arcnum][0]){
142                     for(m = 0; m < G.vexnum; m++){
143                         if(G.vertices[m].data.key == VR[G.arcnum
                                ][1])
144                             break;
145                     }
146                     ArcNode *t = (ArcNode *)malloc(sizeof(ArcNode)
                            );
147                     t->adjvex = m;
148                     t->nextarc = G.vertices[i].firstarc;
149                     G.vertices[i].firstarc = t;
150                 }
151                 if(G.vertices[i].data.key == VR[G.arcnum][1]){
152                     for(m = 0; m < G.vexnum; m++){
153                         if(G.vertices[m].data.key == VR[G.arcnum
                                ][0])
154                             break;
155                     }
156                     ArcNode *t = (ArcNode *)malloc(sizeof(ArcNode)
                            );
157                     t->adjvex = m;
158                     t->nextarc = G.vertices[i].firstarc;
159                     G.vertices[i].firstarc = t;
160                 }
161             }
162             G.arcnum++;
163         }
164         return OK;
165     }
166
167
```

```
168 status DestroyGraph(ALGraph &G)
169 /*销毁无向图G,删除G的全部顶点和边*/
170 {
171     int i;
172     ArcNode *p = NULL, *q = NULL;
173     for(i = 0; i < G.vexnum; i++){
174         p = G.vertices[i].firstarc;
175         if(!p) continue;
176         while(1){
177             if(!p) break;
178             q = p->nextarc;
179             free(p);
180             p = q;
181         }
182     }
183     G.vexnum = G.arcnum = 0;
184     return OK;
185 }
186
187
188 int LocateVex(ALGraph G,KeyType u)
189 //根据u在图G中查找顶点，查找成功返回位序，否则返回-1;
190 {
191     int i;
192     for(i = 0; i < G.vexnum; i++){
193         if(G.vertices[i].data.key == u)
194         return i;
195     }
196     return -1;
197
198 }
199
200
201
202 status PutVex(ALGraph &G,KeyType u,VertexType value)
```

```
203 //根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK
    ;
204 //如果查找失败或关键字不唯一，返回ERROR
205 {
206     int i;
207     for(i = 0; i < G.vexnum; i++)
208         if(G.vertices[i].data.key == value.key)
209             if(G.vertices[i].data.key != u)
210                 return ERROR;
211     for(i = 0; i < G.vexnum; i++){
212         if(G.vertices[i].data.key == u){
213         G.vertices[i].data.key = value.key;
214         strcpy(G.vertices[i].data.others, value.others);
215         return OK;
216         }
217     }
218     return ERROR;
219
220 }
221
222
223 int FirstAdjVex(ALGraph G,KeyType u)
224 //根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，
    否则返回-1;
225 {
226     int i;
227     for(i = 0; i < G.vexnum; i++){
228         if(G.vertices[i].data.key == u)
229             if(G.vertices[i].firstarc)
230                 return G.vertices[i].firstarc->adjvex;
231     }
232     return -1;
233
234 }
235
```

```
236 int NextAdjVex(ALGraph G,KeyType v,KeyType w)
237 //根据u在图G中查找顶点，查找成功返回顶点v的邻接顶点相对于w的下
    一邻接顶点的位序，查找失败返回-1;
238 {
239
240     int i;
241     ArcNode *p = NULL;
242     for(i = 0; i < G.vexnum; i++){
243         if(G.vertices[i].data.key == v){
244             if(G.vertices[i].firstarc){
245                 p = G.vertices[i].firstarc;
246                 while(p){
247                     if(G.vertices[p->adjvex].data.key == w)
248                         if(p->nextarc)
249                             return p->nextarc->adjvex;
250                     p = p->nextarc;
251                 }
252             }
253         }
254     }
255     return -1;
256
257 }
258
259 status InsertVex(ALGraph &G,VertexType v)
260 //在图G中插入顶点v，成功返回OK,否则返回ERROR
261 {
262     if(G.vexnum == MAX_VERTEX_NUM) return ERROR;
263     if(LocateVex(G, v.key) != -1) return ERROR;
264     G.vertices[G.vexnum++].data = v;
265     return OK;
266
267 }
268
269
```

```
270 status DeleteVex(ALGraph &G,KeyType v)
271 //在图G中删除关键字v对应的顶点以及相关的弧，成功返回OK,否则返
    回ERROR
272 {
273     int k;
274     if((k = LocateVex(G, v)) == -1) return ERROR;
275     if(G.vexnum == 1) return ERROR;
276     ArcNode *p = NULL, *q = NULL;
277     p = G.vertices[k].firstarc;
278     int count = 0;
279     int temp[100], t[100] = {0};
280     t[k] = 1;
281     while(1){
282         if(!p) break;
283         q = p->nextarc;
284         temp[count++] = p->adjvex;
285         t[p->adjvex] = 1;
286         free(p);
287         p = q;
288     }
289     for(int i = 0; i < count; i++){
290         p = G.vertices[temp[i]].firstarc;
291         if(G.vertices[p->adjvex].data.key == v) {
292             G.vertices[temp[i]].firstarc = G.vertices[temp[i
                ]].firstarc->nextarc;
293             free(p);
294             p = G.vertices[temp[i]].firstarc;
295             while(p){
296                 if(p->adjvex > k)
297                     p->adjvex--;
298                 p = p->nextarc;
299             }
300         }
301         else{
302             while(G.vertices[p->adjvex].data.key != v){
```

```
303                     q = p;
304                     if(q->adjvex > k)
305                         q->adjvex--;
306                     p = p->nextarc;
307                 }
308                 q->nextarc = p->nextarc;
309                 free(p);
310                 p = q->nextarc;
311                 while(p){
312                     if(p->adjvex > k)
313                         p->adjvex--;
314                     p = p->nextarc;
315                 }
316             }
317         }
318         for(int i = 0; i < G.vexnum; i++){
319             if(t[i]) continue;
320             p = G.vertices[i].firstarc;
321             while(p){
322                 if(p->adjvex > k)
323                     p->adjvex--;
324                 p = p->nextarc;
325             }
326         }
327         for(int j = k; j < G.vexnum - 1; j++){
328             G.vertices[j] = G.vertices[j + 1];
329         }
330         G.vexnum--;
331         G.arcnum = G.arcnum - count;
332         return OK;
333 
334 }
335 
336 
337 status InsertArc(ALGraph &G,KeyType v,KeyType w)
```

```
338 //在图G中增加弧<v,w>，成功返回OK,否则返回ERROR
339 {
340     int i, j;
341     ArcNode *p = NULL;
342     if((i = LocateVex(G, v)) == -1) return ERROR;
343     if((j = LocateVex(G, w)) == -1) return ERROR;
344     p = G.vertices[i].firstarc;
345     while(p){
346         if(p->adjvex == j)
347             return ERROR;
348         p = p->nextarc;
349     }
350     ArcNode *t = (ArcNode *)malloc(sizeof(ArcNode));
351     t->adjvex = j;
352     t->nextarc = G.vertices[i].firstarc;
353     G.vertices[i].firstarc = t;
354     ArcNode *tt = (ArcNode *)malloc(sizeof(ArcNode));
355     tt->adjvex = i;
356     tt->nextarc = G.vertices[j].firstarc;
357     G.vertices[j].firstarc = tt;
358     G.arcnum++;
359     return OK;
360
361 }
362
363 status DeleteArc(ALGraph &G,KeyType v,KeyType w)
364 //在图G中删除弧<v,w>，成功返回OK,否则返回ERROR
365 {
366     int i, j, flag = 0;
367     ArcNode *p = NULL, *q = NULL;
368     if((i = LocateVex(G, v)) == -1) return ERROR;
369     if((j = LocateVex(G, w)) == -1) return ERROR;
370     p = G.vertices[i].firstarc;
371     while(p){
372         if(p->adjvex == j){
```

```
373                 flag = 1;
374                 break;
375             }
376             p = p->nextarc;
377         }
378         if(!flag) return ERROR;
379         q = p = G.vertices[i].firstarc;
380         if(p->adjvex == j){
381             G.vertices[i].firstarc = p->nextarc;
382             free(p);
383         }
384         else{
385             while(p){
386                 if(p->adjvex == j){
387                     q->nextarc = p->nextarc;
388                     free(p);
389                     break;
390                 }
391                 q = p;
392                 p = p->nextarc;
393             }
394         }
395         q = p = G.vertices[j].firstarc;
396         if(p->adjvex == i){
397             G.vertices[j].firstarc = p->nextarc;
398             free(p);
399         }
400         else{
401             while(p){
402                 if(p->adjvex == i){
403                     q->nextarc = p->nextarc;
404                     free(p);
405                     break;
406                 }
407                 q = p;
```

```
408                 p = p->nextarc;
409             }
410         }
411         G.arcnum--;
412         return OK;
413 
414 }
415 
416 void DFS(ALGraph G, int *t, int v, void (*visit)(VertexType)){
417         ArcNode *p = NULL;
418         visit(G.vertices[v].data);
419         t[v] = 1;
420         p = G.vertices[v].firstarc;
421         while(p != NULL){
422             if(!t[p->adjvex]){
423                 DFS(G, t, p->adjvex, visit);
424             }
425             p = p->nextarc;
426         }
427 }
428 
429 status DFSTraverse(ALGraph &G,void (*visit)(VertexType))
430 //对图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数
    visit访问一次，且仅访问一次
431 {
432         if(G.vexnum == 0) return ERROR;
433         int t[100] = {0};
434         for(int i = 0; i < G.vexnum; i++){
435             if(!t[i])
436                 DFS(G, t, i, visit);
437         }
438         return OK;
439 
440 }
441 
```

```
442 void visit(VertexType v)
443 {
444     printf("%d %s",v.key,v.others);
445 }
446
447 status BFSTraverse(ALGraph &G,void (*visit)(VertexType))
448 //对图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数
    visit访问一次，且仅访问一次
449 {
450     int stack[100], top = 0, low = 0;
451     int t[100] = {0};
452     ArcNode *p = NULL;
453     for(int i = 0; i < G.vexnum; i++){
454         if(!t[i]){
455             stack[top++] = i;
456             p = G.vertices[i].firstarc;
457             visit(G.vertices[i].data);
458             t[i] = 1;
459             while(p || low != top){
460                 if(p){
461                     if(!t[p->adjvex]){
462                         visit(G.vertices[p->adjvex].data);
463                         t[p->adjvex] = 1;
464                         stack[top++] = p->adjvex;
465                     }
466                     p = p->nextarc;
467                     continue;
468                 }
469                 if(low != top){
470                     p = G.vertices[stack[low++]].firstarc;
471                 }
472             }
473         }
474     }
475
```

```
476 }
477
478 status SaveGraph(ALGraph G, char FileName[])
479 //将图的数据写入到文件FileName中
480 {
481     if(G.vexnum == 0) return INFEASIBLE;
482     int nu = -1;
483     FILE *fp;
484     if ((fp = fopen(FileName,"wb")) == NULL)
485     {
486         printf("File open error!\n");
487         return ERROR;
488     }
489     fwrite(&G.vexnum, sizeof(int), 1, fp);
490     fwrite(&G.arcnum, sizeof(int), 1, fp);
491
492     for(int i = 0; i < G.vexnum; i++)
493     {
494         fwrite(&G.vertices[i].data, sizeof(VertexType), 1, fp)
              ;
495         ArcNode* s = G.vertices[i].firstarc;
496         while(s)
497         {
498             fwrite(&s->adjvex, sizeof(int), 1, fp);
499             s = s->nextarc;
500         }
501         fwrite(&nu, sizeof(int), 1, fp);
502     }
503     fclose(fp);
504     return OK;
505
506 }
507
508 status LoadGraph(ALGraph &G, char FileName[]) //14
509 //读入文件FileName的图数据, 创建图的邻接表
```

```
510 {
511     if(G.vexnum != 0) return INFEASIBLE;
512     FILE *fp;
513     if ((fp = fopen(FileName,"rb")) == NULL)
514     {
515         printf("File open error!\n");
516         return ERROR;
517     }
518     fread(&G.vexnum, sizeof(int), 1, fp);
519     fread(&G.arcnum, sizeof(int), 1, fp);
520     G.kind = UDG;
521
522     for(int i = 0; i < G.vexnum ; i++)
523     {
524         fread(&G.vertices[i].data, sizeof(VertexType), 1, fp);
525         G.vertices[i].firstarc = NULL;
526         ArcNode* last = G.vertices[i].firstarc;
527         int arc, flag = 0;
528         while(fread(&arc, sizeof(int), 1, fp))
529         {
530             if(arc == -1) break;
531             if(flag == 0)
532             {
533                 ArcNode* s = (ArcNode*)malloc(sizeof(ArcNode))
                        ;
534                 flag = 1;
535                 s->adjvex = arc;
536                 s->nextarc = NULL;
537                 G.vertices[i].firstarc = s;
538                 last = s;
539                 continue;
540             }
541             ArcNode* s = (ArcNode*)malloc(sizeof(ArcNode));
542             s->adjvex = arc;
543             s->nextarc = NULL;
```

```
544                 last->nextarc = s;
545
546                 last = s;
547             }
548         }
549         fclose(fp);
550         return OK;
551
552 }
553
554 status AddList(GRAPHS& Graphs,char ListName[])
555 // 需要在Graphs中增加一个名称为ListName的空图
556 {
557             for(int i = 0; i < Graphs.length; i++){
558                     if(!strcmp(Graphs.elem[i].name,ListName))
                            return INFEASIBLE;
559             }
560         strcpy(Graphs.elem[Graphs.length].name,ListName);
561         Graphs.elem[Graphs.length].G.vexnum = 0;
562         Graphs.length++;
563         return OK;
564 }
565
566 status DestoryList(GRAPHS& Graphs,char ListName[])
567 // Graphs中删除一个名称为ListName的图
568 {
569         for(int i = 0; i < Graphs.length; i++)
570             if(!strcmp(ListName, Graphs.elem[i].name)){
571                 if(Graphs.elem[i].G.vexnum)
572                     DestroyGraph(Graphs.elem[i].G);
573                 for (int j = i; j < Graphs.length - 1; j++)
574                     Graphs.elem[j] = Graphs.elem[j + 1];
575                 Graphs.length--;
576                 return OK;
577             }
```

```
578         return ERROR;
579
580     }
581
582     int LocateList(GRAPHS& Graphs,char ListName[])
583     // 在Graphs中查找一个名称为ListName的图，成功返回逻辑序号，否
        则返回0
584     {
585         if(!Graphs.elem) return INFEASIBLE;//疑问未解决
586         for(int i = 0; i < Graphs.length; i++)
587             if(!strcmp(ListName, Graphs.elem[i].name))
588                 return i + 1;
589         return 0;
590
591     }
592
593     status TraverseList(GRAPHS& Graphs){
594     // 如果多图表不为空，依次显示多图表的名称，每个名称间空一格，
        返回OK；如果多图表为空，返回INFEASIBLE。
595         if(Graphs.length == 0) return INFEASIBLE;
596         printf("\n----------all names ----------------------\n")
            ;
597         for(int i = 0; i < Graphs.length; i++){
598             printf("%s",Graphs.elem[i].name);
599             if(i != Graphs.length - 1) printf(" ");
600         }
601         printf("\n----------------- end ----------------------\
            n");
602         return OK;
603     }
604
605     status SelectList(GRAPHS& Graphs, int i){
606     // 进行图的选择
607         if(Graphs.length == 0) return INFEASIBLE;
608         if(i < 1 || i > Graphs.length) return ERROR;
```

```
609         num = i - 1;
610         return OK;
611     }
612     //5 a 6 b 7 c 8 d 9 e 10 f 11 g -1 nil
613     //5 7 7 8 5 8 7 9 8 9 6 8 10 11 -1 -1
614
615     status VerticesSetLessThanK(ALGraph G, int v, int k){
616     //查找与给定结点距离为k的结点
617             int stack[100], top = 0, low = 0, count = 0 , num = 1;
618         int t[100] = {0}, b[100];
619         for(int i = 0; i < G.vexnum; i++)
620             b[i] = G.vexnum;
621         VNode p;
622         ArcNode *q = NULL;
623         int i = LocateVex(G, v);
624         p = G.vertices[i];
625         stack[top++] = i;
626         t[i] = 1;
627         if(k <= 1){
628             printf("请不要查找距离为1以下的结点! \n");
629             }
630         while(1){
631             if(low == num){
632                     count++;
633                     num = top;
634                     }
635                     if(count == k - 1) break;
636                     visit(p.data);
637                     q = p.firstarc;
638                     while(q){
639                             if(!t[q->adjvex]){
640                                     stack[top++] = q->adjvex;
641                                     t[q->adjvex] = 1;
642                             }
643                             q = q->nextarc;
```

```
644                     }
645                     if(low + 1 == top){
646                             break;
647                     }
648                     p = G.vertices[stack[++low]];
649             }
650             return OK;
651     }
652
653     void FindTarget(ALGraph G, int* t, int &target, int current,
        int i, int j){
654             if(i == j){
655                     target = (target > current) ? current : target
                        ;
656                     return;
657             }
658             current++;
659             t[i] = 1;
660             ArcNode *p = G.vertices[i].firstarc;
661             while(p){
662                     if(!t[p->adjvex]){
663                             t[p->adjvex] = 1;
664                             FindTarget(G, t, target, current, p->
                                adjvex, j);
665                             t[p->adjvex] = 0;
666                     }
667                     p = p->nextarc;
668             }
669     }
670
671     status ShortestPathLength(ALGraph G, int v, int w){
672     //查找最短路径
673             int t[100] = {0}, i, j, target = G.vexnum;
674             if((i = LocateVex(G, v)) == -1) return ERROR;
675         if((j = LocateVex(G, w)) == -1) return ERROR;
```

```
676         FindTarget(G, t, target, 0, i, j);
677         return target;
678 }
679
680 status ConnectedComponentsNums(ALGraph G){
681 //计算连通分支数目
682         int stack[100], top = 0, low = 0;
683     int t[100] = {0};
684     ArcNode *p = NULL;
685     int count = 0;
686     for(int i = 0; i < G.vexnum; i++){
687         if(!t[i]){
688             stack[top++] = i;
689             p = G.vertices[i].firstarc;
690             t[i] = 1;
691             while(p || low != top){
692                 if(p){
693                     if(!t[p->adjvex]){
694                         t[p->adjvex] = 1;
695                         stack[top++] = p->adjvex;
696                     }
697                     p = p->nextarc;
698                     continue;
699                 }
700                 if(low != top){
701                     p = G.vertices[stack[low++]].firstarc;
702                 }
703             }
704             count++;
705         }
706     }
707     return count;
708 }
709
710 int main(){
```

```
711     int op=1;
712     int i, e, ans, j;
713     VertexType V[100];
714         KeyType VR[100][2];
715     VertexType value;
716     char FileName[100], Name[100];
717     while(op){
718         system("cls");
719         printf("\n\n");
720         printf("                                    Menu for
            Graphy    \n");
721         printf("          ------------------------------
722 ------------------------------------------\n");
723         printf("          1. CreateCraph    创建图
                              12. BFSTraverse      广度优先搜索遍
            历\n");
724         printf("          2. DestroyGraph   销毁图
                              13. SaveGraph        图文件保存\n");
725         printf("          3. LocateVex      查找顶点
                             14. LoadGraph         图文件录入   \n");
726         printf("          4. PutVex         顶点赋值
                             15. AddList           多图表添加   \n");
727         printf("          5. FirstAdjVex    获得第一邻接点
                        16. DestroyList      多图表销毁     \n");
728         printf("          6. NextAdjVex     获得下一邻接点
                        17. LocateList       多图表位置查找     \n");
729         printf("          7. InsertVex      插入顶点
                             18. TraverseList      多图表遍历\n");
730         printf("          8. DeleteVex      删除顶点
                             19. SelectList        图操作选择\n");
731         printf("          9. InsertArc      插入弧
                               20. VerticesSetLessThanK      距离小
            于k的顶点集合\n");
732         printf("          10. DeleteArc     删除弧
                               21. ShortestPathLength        顶点
```

```
            间最短路径和长度\n");
733         printf("          11. DFSTraverse    深度优先搜索遍历
                22. ConnectedComponentsNums  图的连通分量\n");
734         printf("          0. Exit          退出\n");
735         printf("          ----------------------------------
736 ----------------------------------------\n");
737         printf("          说明: 每次操作过后请点击空格确认才能
            进行下一步操作! \n");
738         printf("\n            当前操作的二叉树为: ");
739         if(num < 1|| num > Graphs.length){
740             if(num > Graphs.length){
741                 Graphs.elem[num].G.vexnum = 0;
742                 num = 0;
743             }
744             printf("默认图");
745             if(Graphs.elem[num].G.vexnum == 0)
746                 printf("(未创建)");
747             printf("\n\n\n");
748         }
749         else{
750                 printf("%s",Graphs.elem[num - 1].name);
751                 if(Graphs.elem[num].G.vexnum == 0)
752                 printf("(未创建)");
753             printf("\n\n\n");
754                 }
755         if(op > 22 || op < 0)
756             printf("上一步命令出错! 请根据菜单正确输入! \n\n\n
                ");
757         printf("请选择你的操作[0~22]:");
758         scanf("%d",&op);
759         switch(op){
760             case 1:
761                 //printf("\n----CreateCraph功能待实现! \n");
762                 if(Graphs.elem[num].G.vexnum){
763                     printf("该图已存在! \n");
```

```
764                     getchar();getchar();
765                     break;
766                 }
767                 i = 0;
768                     printf("请输入顶点序列(-1 nil作为结束标志)
                        : ");
769                     do{
770                         scanf("%d%s", &V[i].key, V[i].others);
771                     }while (V[i++].key != -1);
772                     i = 0;
773                     printf("请输入关系对序列, 以-1 -1结束: ");
774                     do{
775                         scanf("%d%d", &VR[i][0], &VR[i][1]);
776                     }while (VR[i++][0] != -1);
777                     if (CreateCraph(Graphs.elem[num].G, V, VR)
                            == OK)
778                         printf("图创建成功! \n");
779                     else
780                         printf("图创建失败! \n");
781                 getchar();getchar();
782                 break;
783             case 2:
784                 //printf("\n----DestroyGraph功能待实现! \n");
785                 if(!Graphs.elem[num].G.vexnum) {
786                     printf("图为空! \n");
787                     getchar();getchar();
788                     break;
789                 }
790                 ans = DestroyGraph(Graphs.elem[num].G);
791                 if(ans == OK) printf("图销毁成功! \n");
792                 else printf("图销毁失败! \n");
793                 getchar();getchar();
794                 break;
795             case 3:
796                 //printf("\n----LocateVex功能待实现! \n");
```

```
797                 if(!Graphs.elem[num].G.vexnum) {
798                     printf("图为空！\n");
799                     getchar();getchar();
800                     break;
801                 }
802                                 printf("请输入想要查找的顶点关
                                        键字：");
803                     scanf("%d", &e);
804                     ans = LocateVex(Graphs.elem[num].G, e);
805                     if (ans != -1) printf("图中关键字为%d的顶
                            点的位序为%d\n", e, ans);
806                     else
807                         printf("图中不存在该顶点！\n");
808                 getchar();getchar();
809                 break;
810                 case 4:
811                 //printf("\n----PutVex功能待实现！\n");
812                 if(!Graphs.elem[num].G.vexnum) {
813                     printf("图为空！\n");
814                     getchar();getchar();
815                     break;
816                 }
817                                 printf("请输入想要修改的顶点的
                                        关键字：");
818                 scanf("%d", &e);
819                     printf("将其顶点值修改为：");
820                     scanf("%d %s", &value.key, value.others);
821                     ans = PutVex(Graphs.elem[num].G, e, value)
                            ;
822                     if (ans == ERROR)
823                         printf("赋值操作失败！\n");
824                     else if (ans == OK)
825                         printf("已将关键字为%d的顶点值修改为%d
                                ,%s\n", e, value.key, value.others)
                                ;
```

```
826                 getchar();getchar();
827                 break;
828             case 5:
829                 //printf("\n----BiTreeDepth功能待实现! \n");
830                 if(!Graphs.elem[num].G.vexnum) {
831                     printf("图为空! \n");
832                     getchar();getchar();
833                     break;
834                 }
835                                 printf("请输入想要查找其第一邻
                                    接点的顶点: ");
836                 scanf("%d", &e);
837                 ans = FirstAdjVex(Graphs.elem[num].G, e);
838                                 if (ans != -1)
839                     printf("顶点%d的第一邻接点的位序为%d\n
                        关键字为: %d关键信息为: %s\n", e,
                        ans, Graphs.elem[num].G.vertices[
                        ans].data.key, Graphs.elem[num].G.
                        vertices[ans].data.others);
840                 else
841                     printf("顶点%d没有第一邻接点! \n", e);
842                 getchar();getchar();
843                 break;
844             case 6:
845                 //printf("\n----NextAdjVex功能待实现! \n");
846                 if(!Graphs.elem[num].G.vexnum) {
847                     printf("图为空! \n");
848                     getchar();getchar();
849                     break;
850                 }
851                 printf("请输入两个顶点的关键字: ");
852                     scanf("%d %d", &e, &j);
853                     ans = NextAdjVex(Graphs.elem[num].G, e, j)
                        ;
854                     if (ans != -1)
```

```
855                         printf("顶点%d相对于顶点%d的下一个邻接
                                顶点位序为%d\n关键字为：%d关键信息
                                为：%s\n", e, j, ans, Graphs.elem[
                                num].G.vertices[ans].data.key,
                                Graphs.elem[num].G.vertices[ans].
                                data.others);
856                     else printf("无下一邻接顶点！\n");
857                 getchar();getchar();
858                 break;
859             case 7:
860                 //printf("\n----InsertVex功能待实现！\n");
861                 if(!Graphs.elem[num].G.vexnum) {
862                     printf("图为空！\n");
863                     getchar();getchar();
864                     break;
865                 }
866                 printf("请输入想要插入的关键字和关键信息：");
867                     scanf("%d %s", &value.key, value.others);
868                     ans = InsertVex(Graphs.elem[num].G, value)
                            ;
869                     if (ans == OK)
870                                         printf("顶点 %d %s 已
                                            成功插入图中\n",
                                            value.key, value.
                                            others);
871                         else if (ans == ERROR)
872                             printf("插入失败！\n");
873                 getchar();getchar();
874                 break;
875             case 8:
876                 //printf("\n----DeleteVex功能待实现！\n");
877                 if(!Graphs.elem[num].G.vexnum) {
878                     printf("图为空！\n");
879                     getchar();getchar();
880                     break;
```

```
881                 }
882                     printf("请输入想要删除的顶点的关键字: ");
883                     scanf("%d", &e);
884                     ans = DeleteVex(Graphs.elem[num].G, e);
885                     if (ans == OK)
886                         printf("关键字为%d的顶点已从图中删除\n
                                ", e);
887                     else if (ans == ERROR)
888                         printf("删除失败! \n");
889                 getchar();getchar();
890                 break;
891             case 9:
892                 //printf("\n----InsertArc功能待实现! \n");
893                 if(!Graphs.elem[num].G.vexnum) {
894                     printf("图为空! \n");
895                     getchar();getchar();
896                     break;
897                 }
898                 printf("请输入想要插入的弧: ");
899                     scanf("%d %d", &e, &j);
900                                 ans = InsertArc(Graphs.elem[
                                        num].G, e, j);
901                     if (ans == OK)
902                         printf("插入成功! \n");
903                     else if (ans == ERROR)
904                         printf("插入失败! \n");
905                 getchar();getchar();
906                 break;
907             case 10:
908                 //printf("\n----DeleteArc功能待实现! \n");
909                 if(!Graphs.elem[num].G.vexnum) {
910                     printf("图为空! \n");
911                     getchar();getchar();
912                     break;
913                 }
```

```
914                 printf("请输入要删除弧的两个端点: ");
915                 scanf("%d %d", &e, &j);
916                     ans = DeleteArc(Graphs.elem[num].G, e, j);
917                     if (ans == OK)
918                         printf("删除成功! \n");
919                     else if (ans == ERROR)
920                         printf("删除失败! \n");
921                 getchar();getchar();
922                 break;
923             case 11:
924                 //printf("\n----DFSTraverse功能待实现! \n");
925                 if(!Graphs.elem[num].G.vexnum) {
926                     printf("图为空! \n");
927                     getchar();getchar();
928                     break;
929                 }
930                 printf("深度优先搜索遍历: \n");
931                     DFSTraverse(Graphs.elem[num].G, visit);
932                     printf("\n");
933                 getchar();getchar();
934                 break;
935             case 12:
936                 //printf("\n----BFSTraverse功能待实现! \n");
937                 if(!Graphs.elem[num].G.vexnum) {
938                     printf("图为空! \n");
939                     getchar();getchar();
940                     break;
941                 }
942                 printf("广度优先搜索遍历: \n");
943                     BFSTraverse(Graphs.elem[num].G, visit);
944                     printf("\n");
945                 getchar();getchar();
946                 break;
947             case 13:
948                 //printf("\n----SaveList功能待实现! \n");
```

```
949                 printf("请输入要保存的文件名称: ");
950                 scanf("%s",FileName);
951                 ans = SaveGraph(Graphs.elem[num].G, FileName);
952                 if(ans == INFEASIBLE) printf("文件读入失败! \n
                        ");
953                 else if(ans == ERROR);
954                 else printf("文件读入成功! \n");
955                 getchar();getchar();
956                 break;
957             case 14:
958                 //printf("\n----LoadList功能待实现! \n");
959                 printf("请输入要录入的文件名称: ");
960                 scanf("%s", FileName);
961                 if(LoadGraph(Graphs.elem[num].G, FileName) ==
                        INFEASIBLE) printf("文件录入失败! \n");
962                 else printf("文件录入成功! \n");
963                 getchar();getchar();
964                 break;
965             case 15:
966                 //printf("\n----AddList功能待实现! \n");
967                 if(Graphs.length == MAXlength) {
968                     printf("多图表管理已满, 请清除某些图后再操
                            作! \n");
969                     getchar();getchar();
970                     break;
971                 }
972                 printf("请输入新增图的名称: ");
973                 scanf("%s",Name);
974                 ans = AddList(Graphs, Name);
975                 if(ans == INFEASIBLE) printf("该名称的图已经存
                        在!\n");
976                 else printf("%s已成功添加! \n",Name);
977                 getchar();getchar();
978                 break;
979             case 16:
```

```
980                 //printf("\n---DestoryList功能待实现！\n");
981                 if(Graphs.length == 0) {
982                     printf("多图表管理已空，请添加某些图后再操
                        作！\n");
983                     getchar();getchar();
984                     break;
985                 }
986                 printf("请输入销毁图的名称：");
987                 scanf("%s",Name);
988                 ans = DestoryList(Graphs, Name);
989                 if(ans == OK)printf("%s已成功销毁！\n",Name);
990                 else printf("图不存在！\n");
991                 getchar();getchar();
992                 break;
993             case 17:
994                 //printf("\n---LocateList功能待实现！\n");
995                 printf("请输入查找图的名称：");
996                 scanf("%s",Name);
997                 if(LocateList(Graphs, Name)) printf("该图的逻
                    辑索引为：%d\n", LocateList(Graphs, Name));
998                 else printf("图查找失败！\n");
999                 getchar();getchar();
1000                break;
1001            case 18:
1002                //printf("\n----TraverseList功能待实现！\n");
1003                if(TraverseList(Graphs) == INFEASIBLE) printf(
                    "多图表为空！\n");
1004                getchar();getchar();
1005                break;
1006            case 19:
1007                //printf("\n----SelectList功能待实现！\n");
1008                printf("请选择要处理的图的逻辑索引：");
1009                scanf("%d", &ans);
1010                if(SelectList(Graphs, ans) == OK) {
1011                    printf("已选取成功！\n");
```

```
1012                    num = ans;
1013                }
1014                else printf("选取失败！\n");
1015                getchar();getchar();
1016                break;
1017            case 20:
1018                //printf("\n----VerticesSetLessThanK功能待实
                        现！\n");
1019                if(!Graphs.elem[num].G.vexnum) {
1020                    printf("图为空！\n");
1021                    getchar();getchar();
1022                    break;
1023                }
1024                printf("请输入顶点和距离：");
1025                scanf("%d %d", &e, &j);
1026                VerticesSetLessThanK(Graphs.elem[num].G ,e,j);
1027                getchar();getchar();
1028                break;
1029            case 21:
1030                //printf("\n----ShortestPathLength功能待实现！
                        \n");
1031                if(!Graphs.elem[num].G.vexnum) {
1032                    printf("图为空！\n");
1033                    getchar();getchar();
1034                    break;
1035                }
1036                printf("请输入顶点v和顶点w：");
1037                scanf("%d %d", &e, &j);
1038                if(e == j){
1039                        printf("请不要输入两个相同的结点！\n")
                            ;
1040                            }
1041                            ans = ShortestPathLength(
                                    Graphs.elem[num].G,e,j);
1042                            if(ans == Graphs.elem[num].G.
```

```
                                        vexnum){
1043                                        printf("两者间不存在路
                                                径!\n");
1044                    getchar();getchar();
1045                    break;
1046                                    }
1047                                    if(ans == ERROR){
1048                                        printf("两顶点不都存
                                                在!\n");
1049                    getchar();getchar();
1050                    break;
1051                                    }
1052                printf("两节点之间的最短路径为:%d\n", ans);
1053                getchar();getchar();
1054                break;
1055            case 22:
1056                //printf("\n----ConnectedComponentsNums功能待
                        实现!\n");
1057                if(!Graphs.elem[num].G.vexnum) {
1058                    printf("图为空!\n");
1059                    getchar();getchar();
1060                    break;
1061                }
1062                ans = ConnectedComponentsNums(Graphs.elem[num
                        ].G);
1063                printf("连通分量包含%d个!\n", ans);
1064                getchar();getchar();
1065                break;
1066            case 0:
1067                break;
1068        }//end of switch
1069    }//end of while
1070    printf("欢迎下次再使用本系统!\n");
1071    return 0;
1072 }//end of main()
```