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 CS2301

学　　号 U202315413

姓　　名 刘仁鹏

指导教师 李剑军

报告日期 2024 年 06 月 16 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

顺序表[1] 是在计算机内存中以数组的形式保存的线性表，线性表的顺序存储是指用一组地址连续的存储单元依次存储线性表中的各个元素、使得线性表中在逻辑结构上相邻的数据元素存储在相邻的物理存储单元中，即通过数据元素物理存储的相邻关系来反映数据元素之间逻辑上的相邻关系，采用顺序存储结构的线性表通常称为顺序表。顺序表是将表中的结点依次存放在计算机内存中一组地址连续的存储单元中。

本实验将构造顺序表，并且在控制台上呈现功能演示系统：通过输入数字 0 至 18 实现各个功能，包括初始化表、销毁表、清空表、判定空表、求表长和获得元素等。

1.1 问题描述

1) 加深对线性表的概念、基本运算的理解；
2) 掌握线性表的顺序存储结构（顺序表）的含义与实现方法；
3) 熟练掌握线性表在顺序存储结构上的插入、删除、查找等操作。

1.2 系统设计

本系统提供一个顺序存储的线性表。菜单可供选择的操作有：初始化表、销毁表、清空表、判空表，求表长、获得元素、查找元素、获得前驱元素、获得后继元素、插入元素、删除元素、遍历表、线性表的文件形式保存、最大连续子数组和、和为 K 的子数组、顺序表排序、多线性表管理等功能。

头文件、常量、类型的定义如下：

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

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

```
#define  ERROR 0
#define  INFEASTABLE -1
#define  OVERFLOW -2

typedef  int   status ;
typedef  int  ElemType;

#define  LIST_INIT_SIZE 100
#define  LISTINCREMENT 10
typedef   struct {
      ElemType * elem;
       int   length ;
       int    listsize  ;
}SqList;
typedef   struct
{
        struct  { char  name[30];
                    SqList  L;
         } elem[10];
         int   length ;
         int    listsize  ;
 }LISTS;
```

1.3 系统实现

各个函数的具体定义及功能如下：

1) 初始化表：实现线性表的初始化。

   函数定义：status InitList(SqList &L);

   功能说明：若线性表不存在，则构造一个空的线性表，返回 OK；否则（即线性表存在）返回 INFEASIBLE。

2) 销毁表：实现顺序表的销毁。

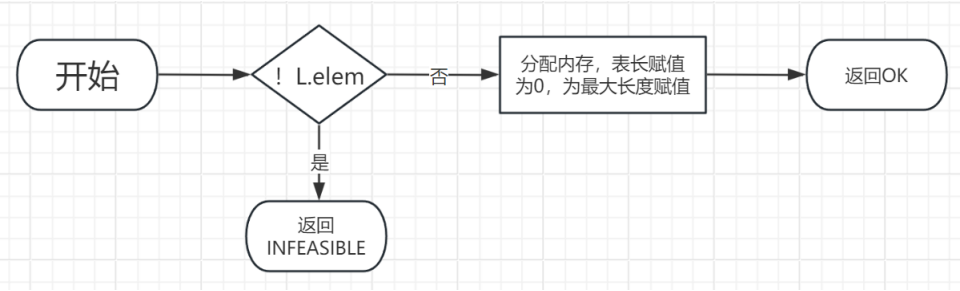

图 1-1　初始化表

函数定义：status DestroyList(SqList &L);

功能说明：若线性表存在，则释放线性表的空间，使线性表成为未初始化状态，返回 OK；否则（即线性表未初始化）不能进行销毁操作，返回 INFEASIBLE。

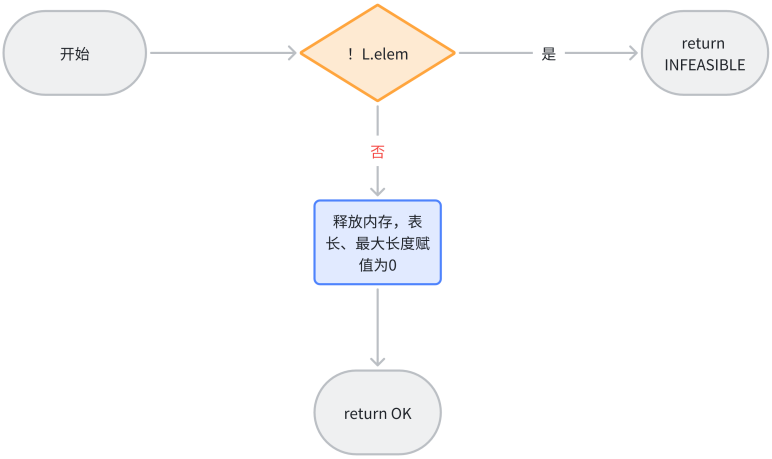

图 1-2　销毁表

3) 清空表：实现线性表的清空。

函数定义：status ClearList(SqList &L);

功能说明：若线性表存在，则清空线性表，返回 OK；否则（即线性表不存在）返回 INFEASIBLE。

4) 判定空表：实现线性表是否为空的判断。

函数定义：status ListEmpty(SqList L);

功能说明：若线性表不存在，则返回 INFEASIBLE；若线性表存在且长度为 0，则返回 TRUE；若线性表存在且长度不为 0，则返回 FALSE。

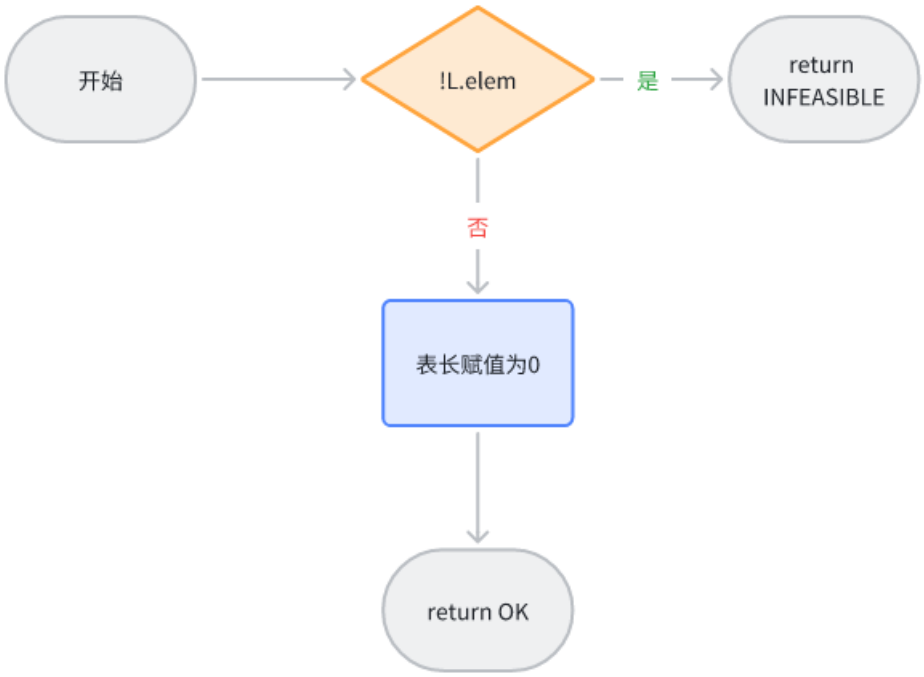

图 1-3　清空表

5) 求表长：实现线性表长度的获取。

函数定义：status ListLength(SqList L);

功能说明：若线性表存在，则返回线性表的长度；否则（即线性表不存在）返回 INFEASIBLE。

6) 获得元素：实现线性表第 i 个元素的获取。

函数定义：status GetElem(SqList L,int i,ElemType &e);

功能说明：若线性表存在，当 i<1 或 i 超过线性表的长度时返回 ERROR，反

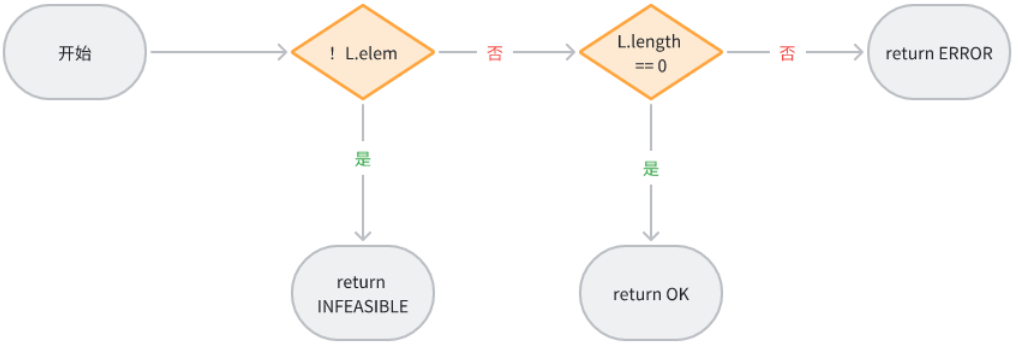

图 1-4　判定空表

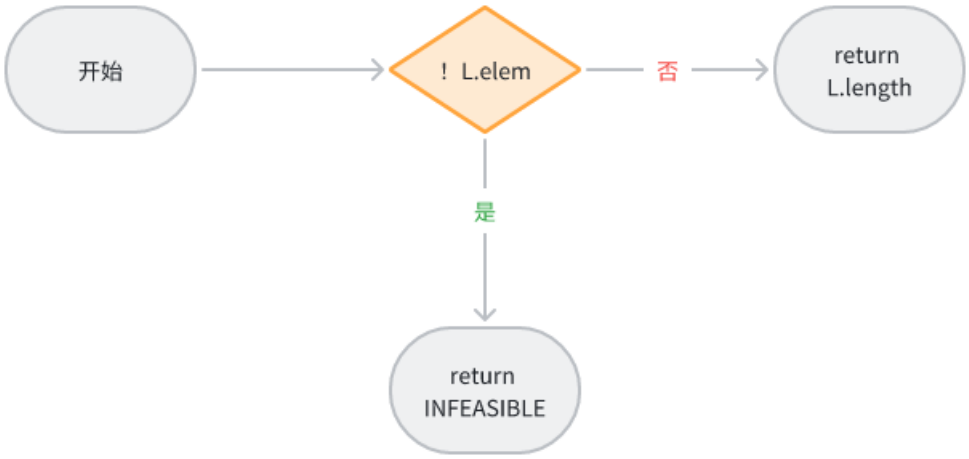

图 1-5　求表长

之 i 值合理，则将线性表第 i 个元素值赋给 e，返回 OK；否则（即线性表不存在）返回 INFEASIBLE。

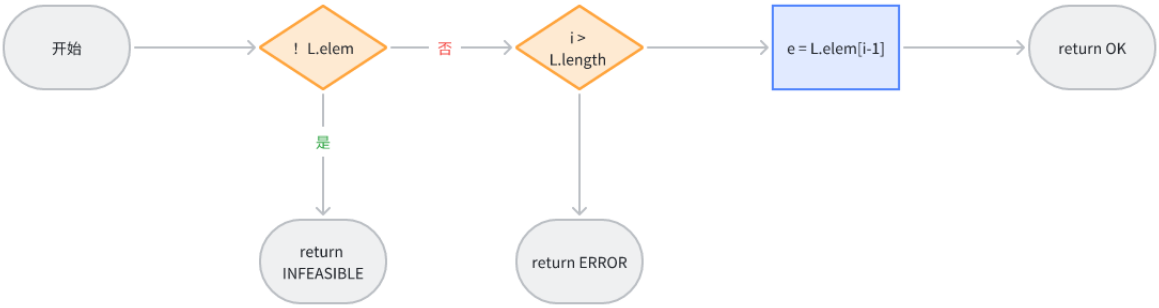

图 1-6　获得元素

7) 查找指定元素：实现线性表中指定元素的查找。

函数定义：status LocateElem(SqList L,ElemType e);

功能说明：若线性表存在但未找到指定元素 e，则返回 ERROR；若线性表存在且找到指定元素 e，则返回元素逻辑序号 i；否则（即线性表不存在）返回 INFEASIBLE。

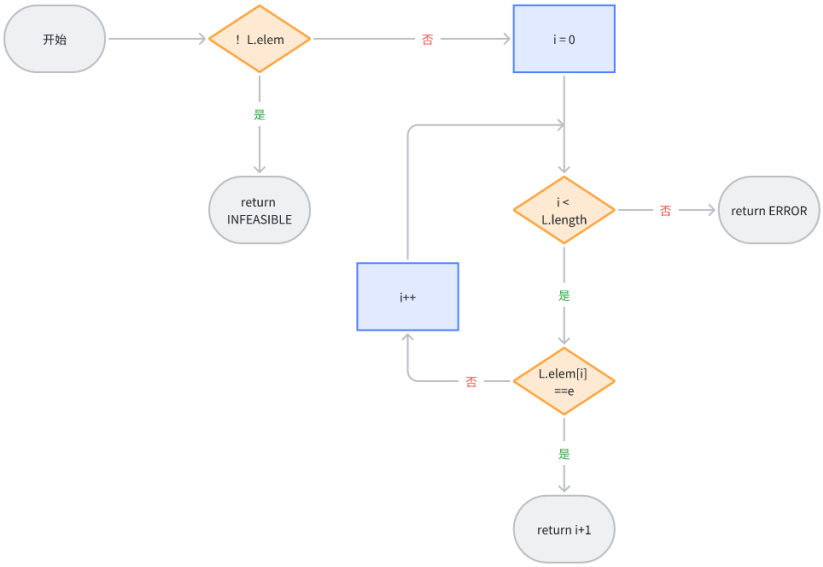

图 1-7　查找指定元素

8) 获得前驱元素：实现线性表指定元素前驱的获取。

函数定义：status PriorElem(SqList L,ElemType e,ElemType &pre);

功能说明：若线性表存在但未找到指定元素 e 的前驱，则返回 ERROR；若线性表存在且找到指定元素 e 的前驱，则将值赋给 pre，返回 OK；否则（即线性表不存在）返回 INFEASIBLE。

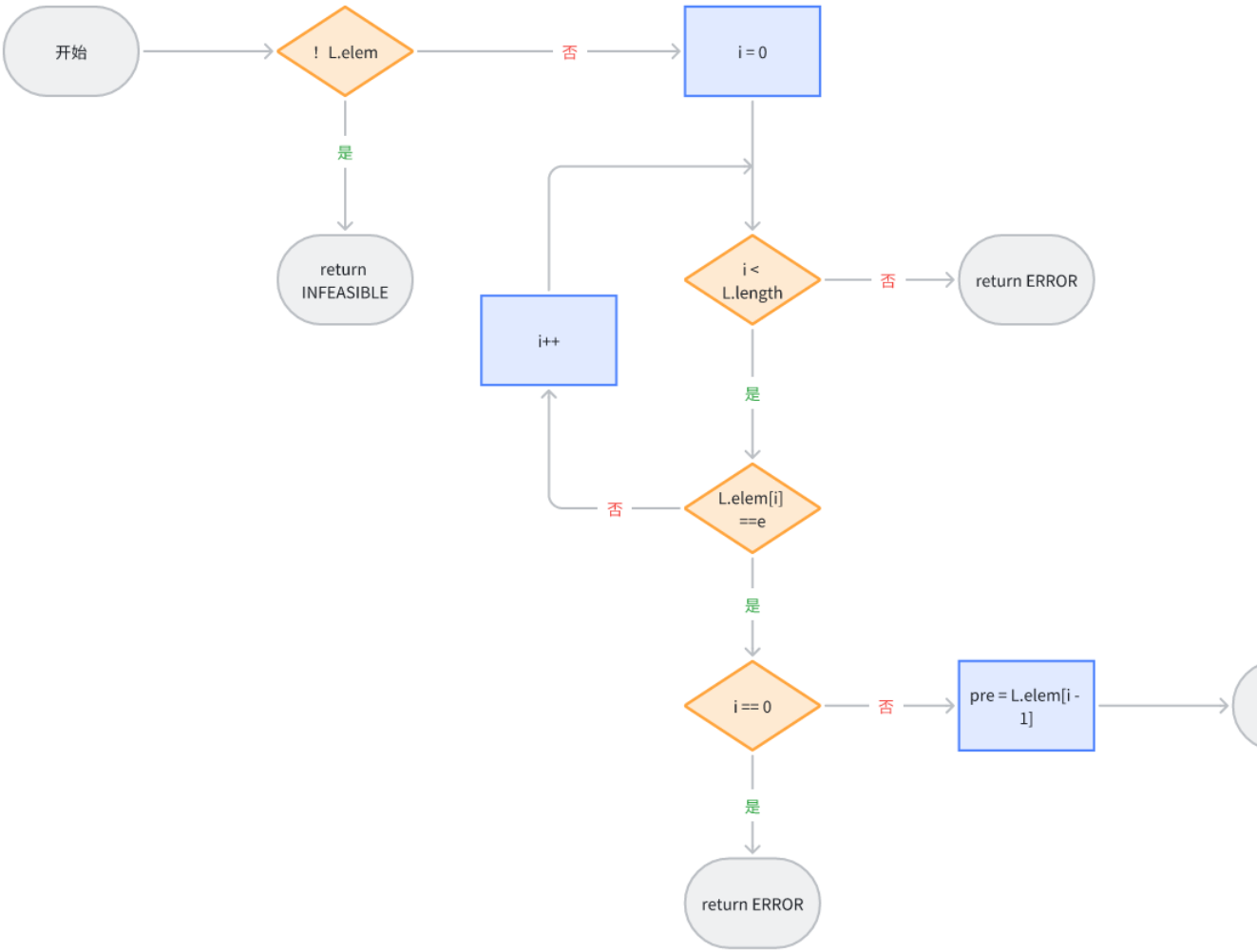

图 1-8　获得前驱元素

9) 获得后继元素：实现线性表指定元素后继的获取。

函数定义：status NextElem(SqList L,ElemType e,ElemType &next);

功能说明：若线性表存在但未找到指定元素 e 的后继，则返回 ERROR；若线性表存在且找到指定元素 e 的后继，则将值赋给 next，返回 OK；否则（即线性表不存在）返回 INFEASIBLE。

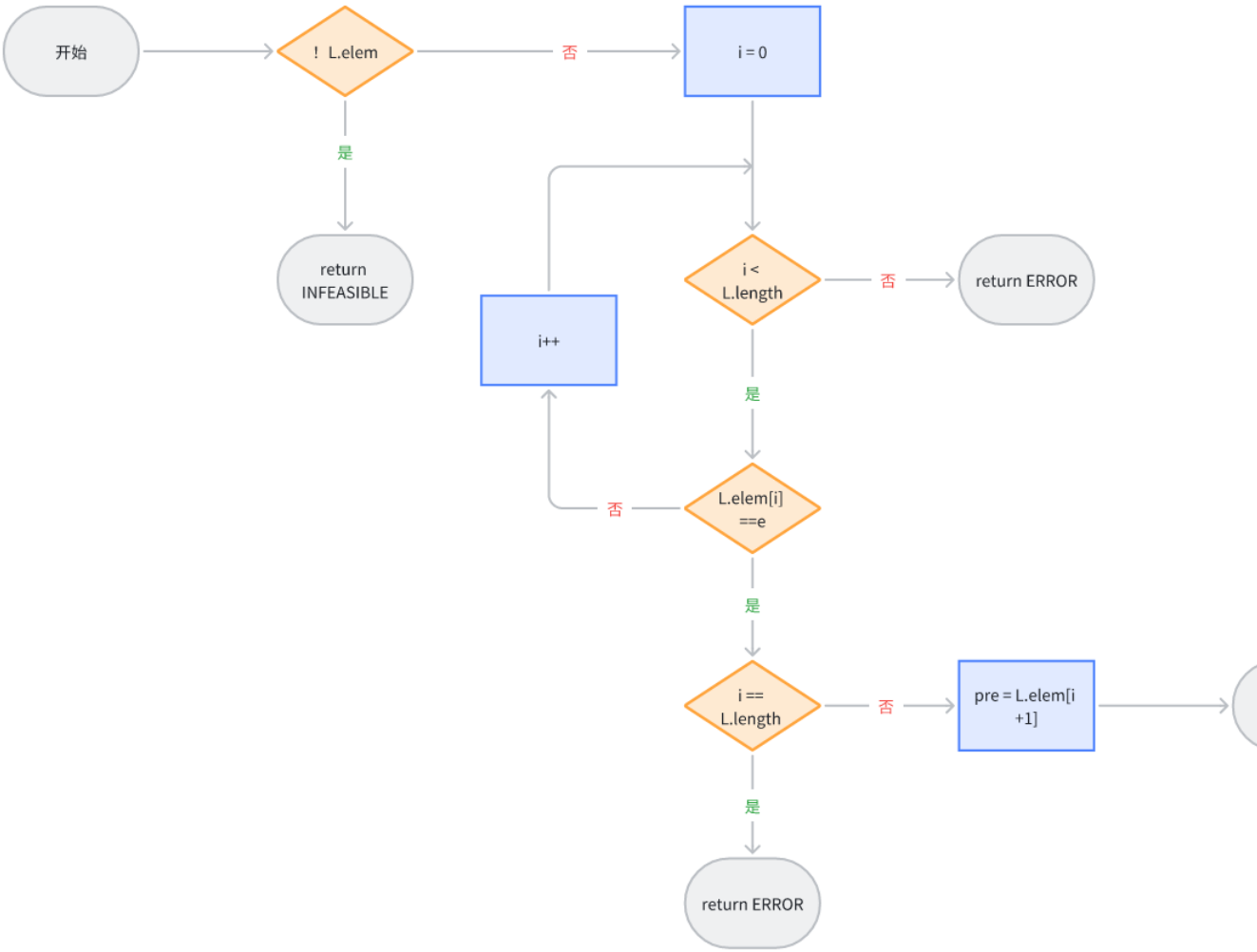

图 1-9　获得后继元素

10) 插入元素：实现顺序表指定元素的插入操作。

函数定义：status ListInsert(SqList &L,int i,ElemType e);

功能说明：若线性表存在，当 i<1 或 i 超过线性表的长度时返回 ERROR，反之 i 值合理，则在线性表的第 i 个元素前插入新的元素 e，返回 OK；否则（即线性表不存在）返回 INFEASIBLE。

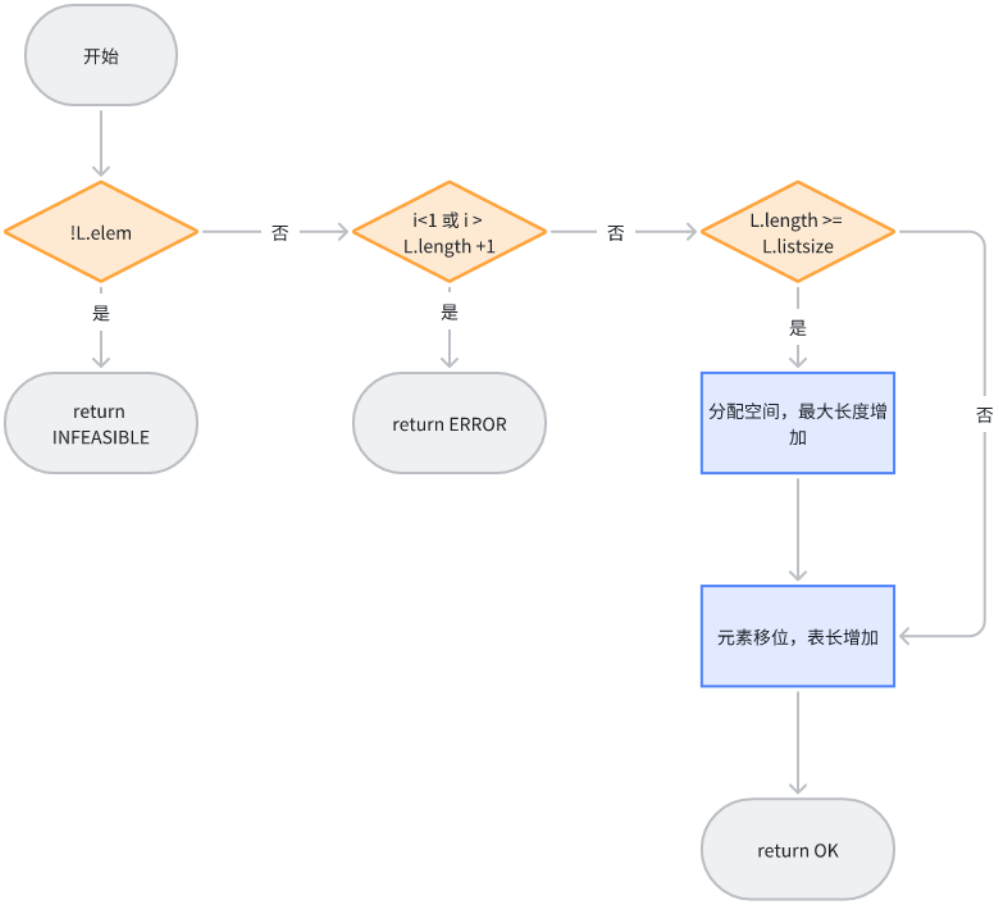

图 1-10　插入元素

11) 删除元素：实现线性表指定元素的删除操作。

函数定义：status ListDelete(SqList &L,int i,ElemType &e);

功能说明：若线性表存在，当 i<1 或 i 超过线性表的长度时返回 ERROR，反之 i 值合理，则删除线性表的第 i 个元素（将值赋给 e），返回 OK；否则（即线性表不存在）返回 INFEASIBLE。

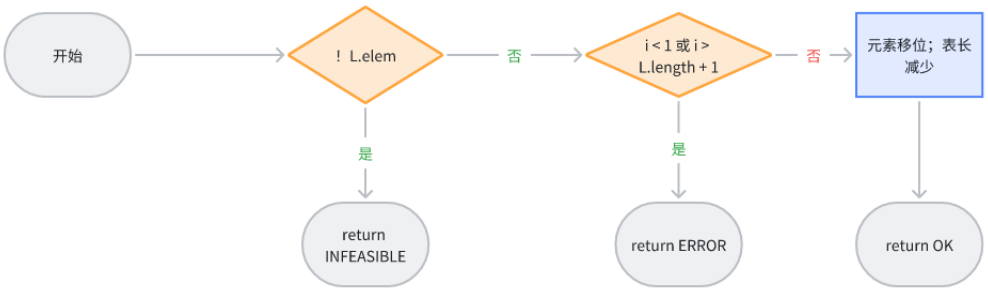

图 1-11　删除元素

12) 遍历表：实现线性表的遍历操作。

函数定义：status ListTraverse(SqList L);

功能说明：若线性表存在，则输出线性表的每一个元素 (以空格隔开)，并返回 OK；否则（即线性表不存在）返回 INFEASIBLE。

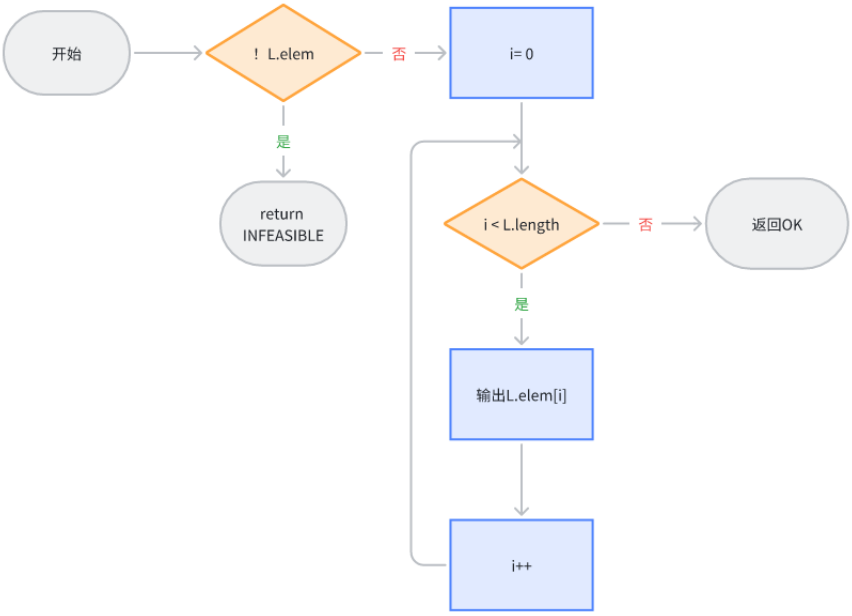

图 1-12　遍历表

13) 读写文件：实现将线性表的数据元素进行读写文件操作。

<1> 写入文件的函数定义：status SaveList(SqList L,char FileName[]);

功能说明：若线性表存在，则将线性表 L 的全部元素写入到文件名为 FileName 的文件中，返回 OK；否则（即线性表不存在）返回 INFEASIBLE。

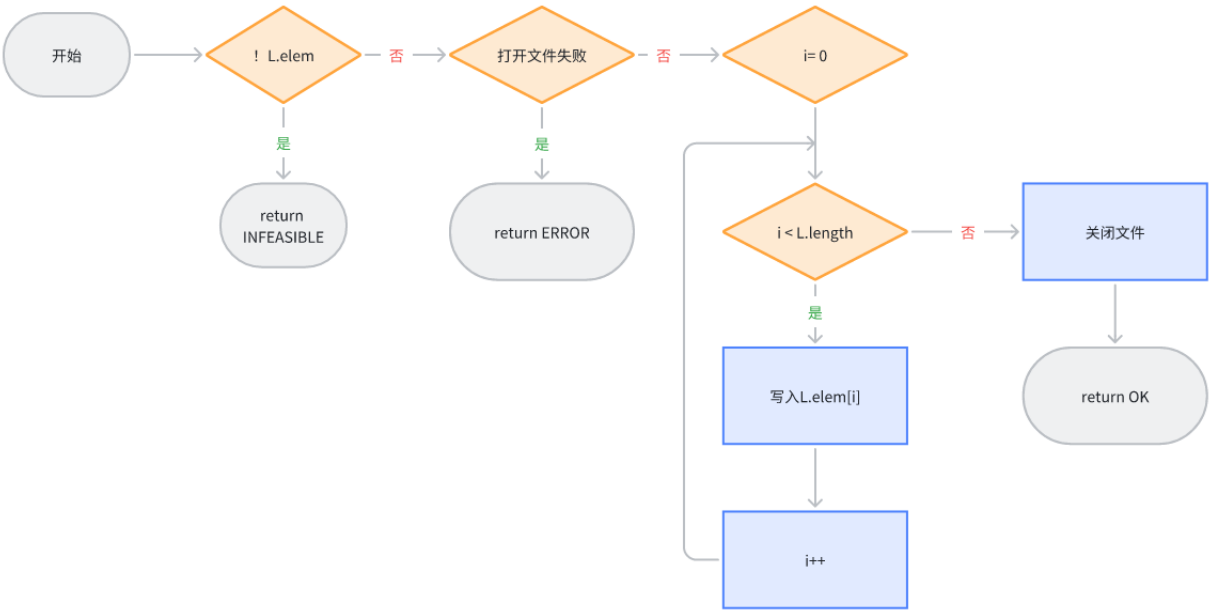

图 1-13　写入文件

<2> 读取文件的函数定义：status LoadList(SqList &L,char FileName[]);

功能说明：若线性表存在，返回 INFEASIBLE；否则（即线性表存在）则读取文件 FileName 创建线性表 L，返回 OK。

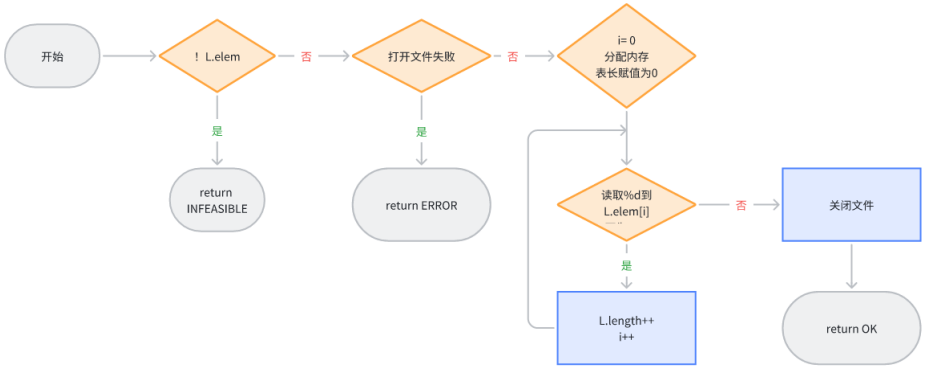

图 1-14　读取文件

14) 多线性表管理——增加一个新线性表：实现在线性表的集合中增加一个新的线性表。

函数定义：status AddList(LISTS &Lists,char ListName[]);

功能说明：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。增加成功返回 OK，否则（即 List 集合元素已满）返回 ERROR。

15) 多线性表管理——移除一个新线性表：实现在线性表的集合中移除一个指定的线性表。

函数定义：status RemoveList(LISTS &Lists,char ListName[]);

功能说明：Lists 是一个以顺序表形式管理的线性表的集合，在集合中查找名称为 ListName 的线性表，有则删除并将 Lists 集合中后面的线性表向前移动，返回 OK；否则返回 ERROR。

16) 多线性表管理——查找线性表：实现在线性表的集合中查找一个指定的线性表。

函数定义：int LocateList(LISTS Lists,char ListName[]);

功能说明：Lists 是一个以顺序表形式管理的线性表的集合，在集合中查找名称为 ListName 的线性表，有则返回线性表的逻辑序号；无则返回 0。

*** 程序源代码见：附录 A 基于顺序存储结构线性表实现的源程序（点击可跳转）**。

1.4 系统测试

程序在控制台上打印功能面板，提示用户输入数字 0 至 18 进行对应的线性表相关操作。**(表格中的每次输入都会影响该表格中的下次输出结果！)**

1) 初始化表、销毁表、清空表、判定空表

这是线性表最最基本的功能，销毁、清空和判空时均要判断线性表是否存在。

表 1-1　初始化表、销毁表、清空表、判定空表

输入	解释	实际输出
2	未初始化 + 销毁表	线性表销毁失败!
3	未初始化 + 清空表	线性表清空失败!
4	未初始化 + 判空表	线性表判空失败!
1	初始化	线性表创建成功!
3	已初始化 + 清空表	线性表清空成功!
4	已初始化 + 判空表	线性表是空的!
2	已初始化 + 销毁表	线性表销毁成功!

2) 求表长、插入元素、删除元素、遍历表、获得元素以及其前驱后继

这些也是线性表比较基本的功能，关乎对线性表中元素的各种操作。

表 1-2　求表长、插入元素、删除元素、遍历表、获得元素以及其前驱后继

输入	解释	实际输出
10 2 1	未初始化 + 在第 1 个位置插入元素 2	插入失败
11 1	未初始化 + 删除第 1 个元素	删除失败
12	未初始化 + 遍历表	线性表是空表!
1	初始化	线性表创建成功!
12	已初始化 + 遍历表	(线性表为空表，故遍历没有输出)
10 1 2	已初始化 + 在第 1 个位置插入元素 2	插入成功
10 1 3	已初始化 + 在第 1 个位置插入元素 3	插入成功，线性表现在的长度是 2
10 1 4	已初始化 + 在第 1 个位置插入元素 4	插入成功，线性表现在的长度是 3
12	已初始化 + 遍历表	4 3 2
6 1	已初始化 + 获取第 1 个元素的值	线性表中第 1 个元素的值是 4
7 3	已初始化 + 查找元素值为 3 的最小序号	线性表中元素值为 3 的最小序号是 2
8 3	已初始化 + 查找元素值为 3 的前驱元素	线性表中元素值为 3 的前驱元素是 4
8 4	已初始化 + 查找元素值为 4 的前驱元素	未找到值为 4 的元素 或该元素没有前驱元素

9 3	已初始化 + 查找元素值为 3 的后继元素	线性表中元素值为 3 的最小序号是 2
11 4	已初始化 + 删除第 4 个元素	错误：序号小于 1 或者超过了线性表的长度 3
11 2	已初始化 + 删除第 2 个元素	删除成功，线性表现在的长度是 2
12	已初始化 + 删除了元素 + 遍历表	4 2

3) 读写文件

将线性表的全部元素写入文件，以及将文件中的数据读取出到线性表中。

表 1-3　读写文件

输入	解释	实际输出
13 2	未初始化 + 写入文件	线性表未初始化, 不能写入文件
1	初始化	线性表初始化成功
10 1 2	已初始化 + 在第 1 个位置插入元素 2	插入成功
10 1 3	已初始化 + 在第 1 个位置插入元素 3	插入成功
10 1 4	已初始化 + 在第 1 个位置插入元素 4	插入成功
13 2	已初始化 + 写入文件	写入文件成功
13 1	已初始化 + 读取文件	线性表已初始化, 不能读取文件 (不能覆盖线性表)
2	已初始化 + 销毁表	线性表销毁成功
13 1	已初始化 + 读取文件	读取文件成功
12	已读取文件 + 遍历表	4 3 2

1.5　实验小结

本章节的内容是线性表的练习，控制台输出的基础框架已经给出，所以只要补充所要求的函数具体实现即可。由于自己在上学期的俱乐部项目中制作过控制台交互程序，故在这次的实践中也更加得心应手。

唯一有些生疏的是 C 语言的读写文件操作，因此又复习了《C 语言程序设计》中对应章节。

总之，本章节属于数据结构中相对基础的一章，学好这一部分也为掌握后面的知识打下坚实基础。

2 基于二叉链表的二叉树实现

二叉树[2] 是指树中结点的度不大于 2 的有序树，它是一种最简单且最重要的树。二叉树的递归定义为：二叉树是一棵空树，或者是一棵由一个根结点和两棵互不相交的，分别称作根的左子树和右子树组成的非空树；左子树和右子树又同样都是二叉树。

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。

本实验将构造二叉树，并且在控制台上呈现功能演示系统：通过输入数字 0 至 14 实现各个功能，包括初始化树、清空树、求深度和获得兄弟结点等。

2.1 问题描述

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。

2.2 系统设计

本系统提供一个基于二叉链表的二叉树。菜单可供选择的操作有：创建二叉树、清空二叉树、求二叉树深度、查找结点、结点赋值、获得兄弟结点、插入结点、删除结点、前序遍历、中序遍历、后序遍历、按层遍历、实现线性表的文件形式保存等共计 14 种功能。

头文件、常量、类型的定义如下：

```
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
```

```
#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;
int   result  = 0,  count  = 0;
BiTree  BiTreestack [100];
int  top;
BiTree  BiTreequeue[100];
int  l,  r;
```

```
int i, k, flag1[1000], flag2 = 0;

status checkKey(TElemType definition[]) {
    i = 0;
    while(definition[i++].key != -1);
    for (int j = 0; j < i; j++) {
        for (int k = j + 1; k < i; k++) {
            if (definition[k].key == definition[j].key && definition
                [k].key != 0)
                return 0;
        }
    }
    return 1;
}
```

2.3 系统实现

各个函数的具体定义及功能如下：

1) 创建二叉树：实现二叉树的初始化。

函数定义：status CreateBiTree(BiTree& T, TElemType definition[]);

功能说明：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。

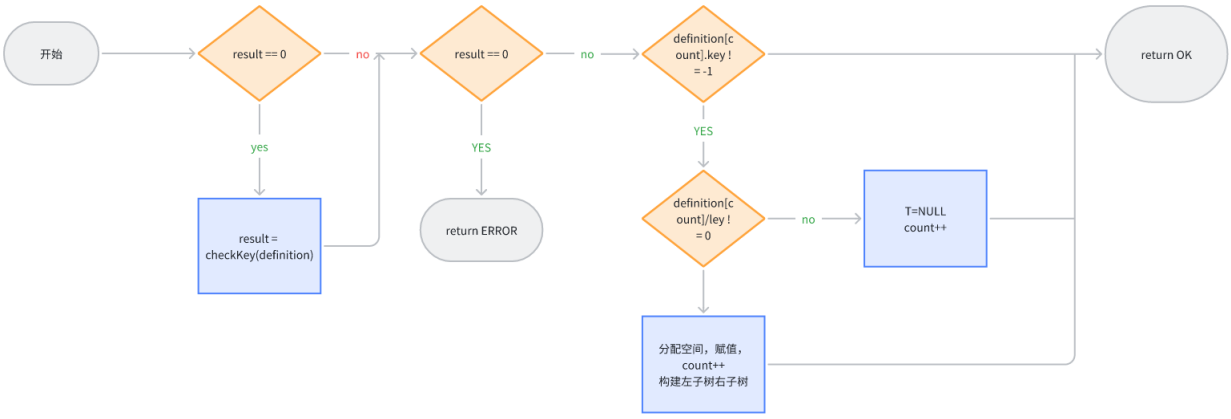

图 2-1　创建二叉树

2) 清空二叉树：实现二叉树的清空。

函数定义：status ClearBiTree(BiTree& T);

功能说明：初始条件是二叉树 T 存在；操作结果是将二叉树 T 清空。

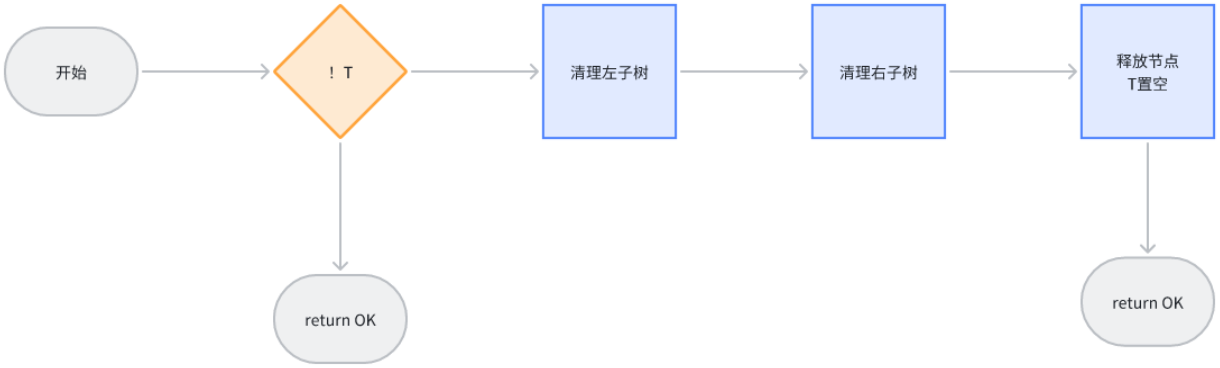

图 2-2　清空二叉树

3) 求二叉树深度

函数定义：int BiTreeDepth(BiTree T);

功能说明：初始条件是二叉树 T 存在；操作结果是返回 T 的深度。

4) 查找结点

函数定义：BiTNode* LocateNode(BiTree T, KeyType e);

功能说明：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。

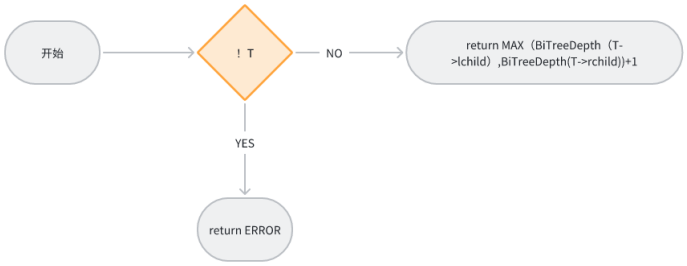

图 2-3　求二叉树深度

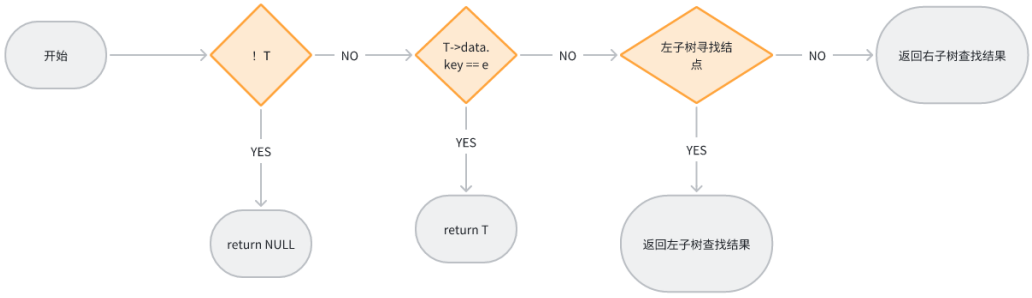

图 2-4　查找结点

5) 结点赋值。

函数定义：status Assign(BiTree& T, KeyType e, TElemType value);

功能说明：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。

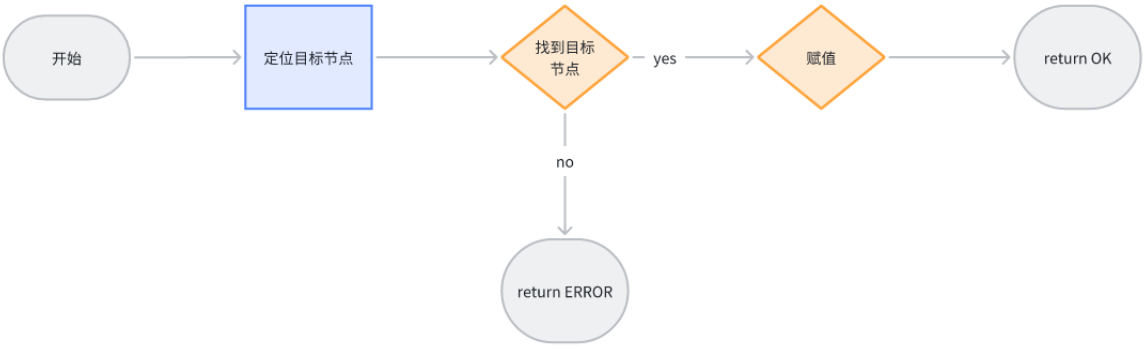

图 2-5　结点赋值

6) 获得兄弟结点

函数定义：BiTNode* GetSibling(BiTree T, KeyType e);

功能说明：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。

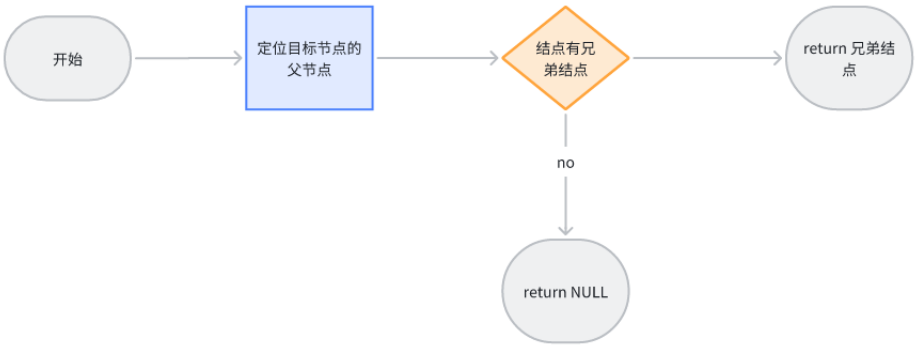

图 2-6　获得兄弟结点

7) 插入结点

函数定义：status InsertNode(BiTree& T, KeyType e, int LR, TElemType c);

功能说明：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。

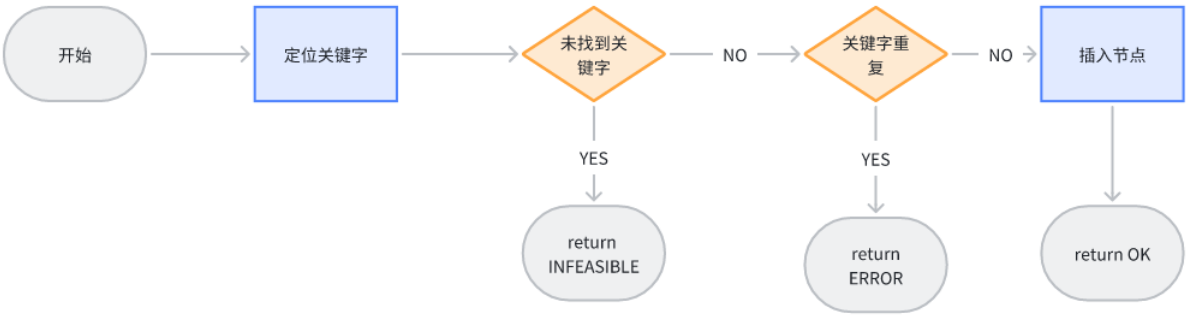

图 2-7　插入结点

8) 删除结点

函数定义：status DeleteNode(BiTree& T, KeyType e);

功能说明：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。

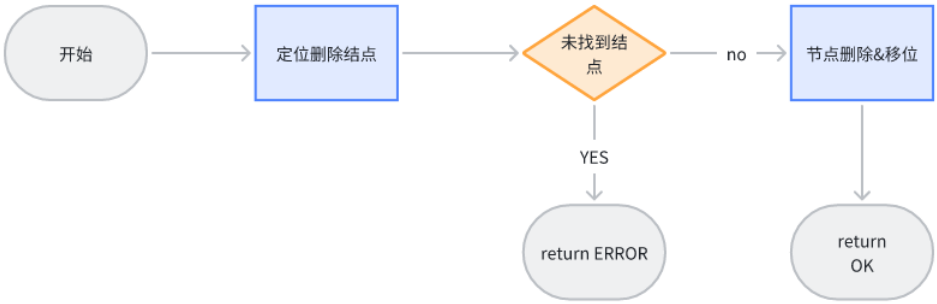

图 2-8　删除结点

9) 前序遍历

函数定义：status PreOrderTraverse(BiTree T, void (*visit)(BiTree));

功能说明：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。

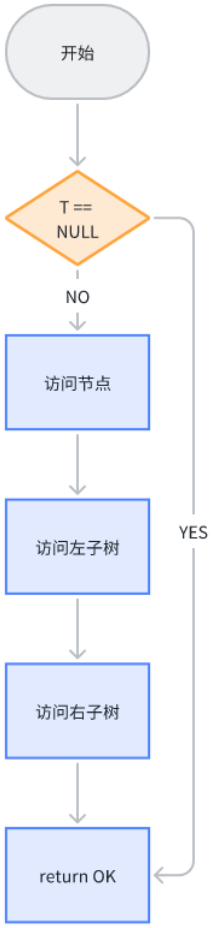

图 2-9　前序遍历

10) 中序遍历

函数定义：status InOrderTraverse(BiTree T, void (*visit)(BiTree));

功能说明：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。

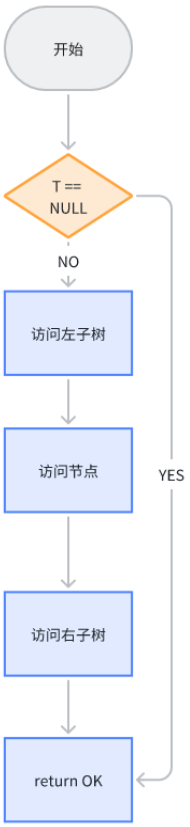

图 2-10　中序遍历

11) 后序遍历

函数定义：status PostOrderTraverse(BiTree T, void (*visit)(BiTree));

功能说明：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。

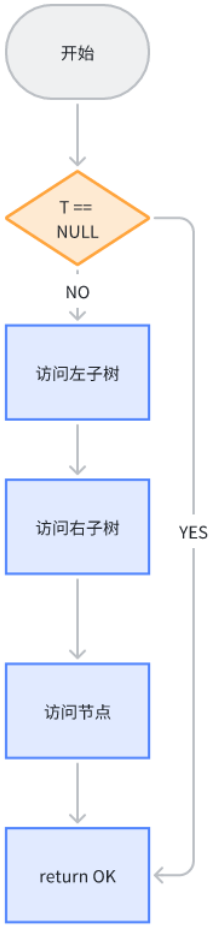

图 2-11　后序遍历

12) 按层遍历

函数定义：status LevelOrderTraverse(BiTree T, void (*visit)(BiTree));

功能说明：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。

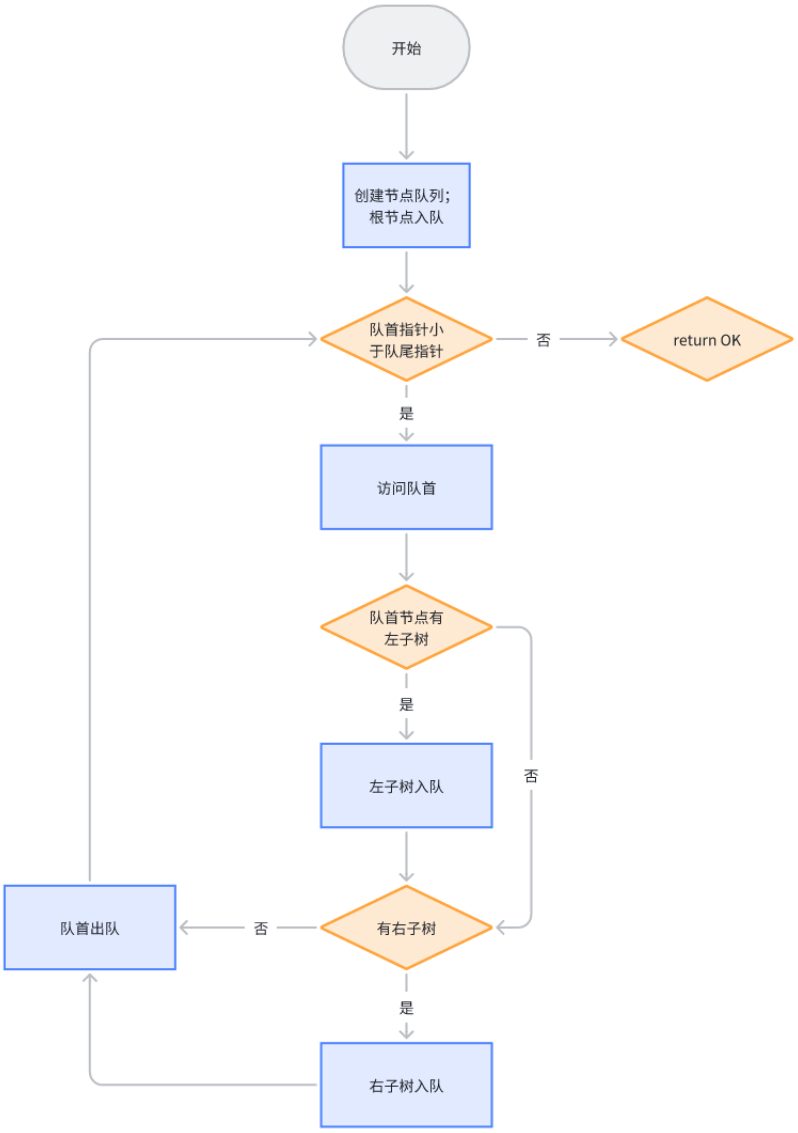

图 2-12　按层遍历

13) 读写文件：实现将二叉树的数据元素进行读写文件操作。

<1> 写入文件的函数定义：status SaveBiTree(BiTree T, char FileName[]);

功能说明：若二叉树存在，则将二叉树 T 的全部元素写入到文件名为 FileName 的文件中，返回 OK；否则（即二叉树不存在）返回 INFEASIBLE。

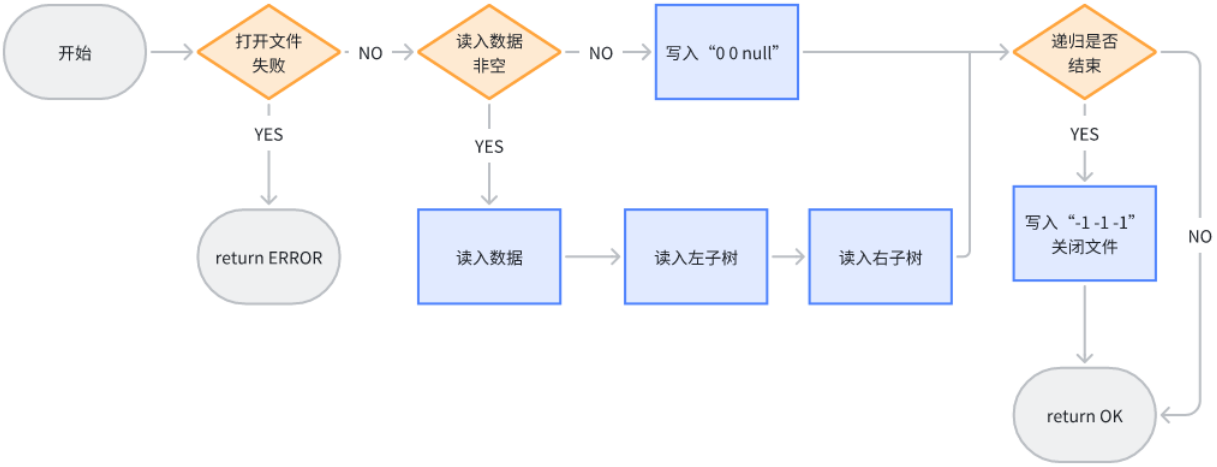

图 2-13　写入文件

<2> 读取文件的函数定义：status LoadBiTree(BiTree& T, char FileName[]);

功能说明：若二叉树存在，返回 INFEASIBLE；否则（即二叉树存在）则读取文件 FileName 创建二叉树 T，返回 OK。

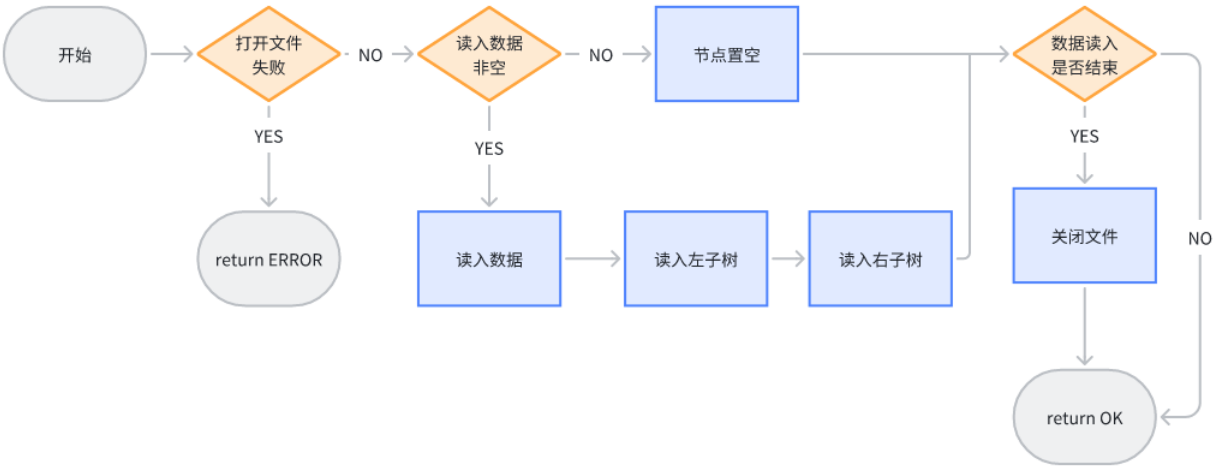

图 2-14　读取文件

*** 程序源代码见：附录 C 基于二叉链表二叉树实现的源程序（点击可跳转）。**

2.4　系统测试

程序在控制台上打印功能面板，提示用户输入数字 0 至 25 进行对应的线性表相关操作。**(表格中的每次输入都会影响该表格中的下次输出结果！)**

默认创建的二叉树如图所示。

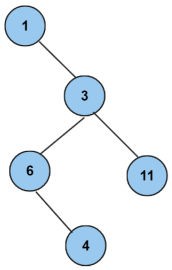

图 2-15　初始二叉树

1) 创建二叉树、销毁二叉树、清空二叉树、判空二叉树

这是二叉树最最基本的功能，销毁、清空和判空时均要判断二叉树是否存在。

表 2-1　创建二叉树、销毁二叉树、清空二叉树、判空二叉树

输入	解释	实际输出
2	未创建 + 销毁二叉树	二叉树为空!
3	未创建 + 清空二叉树	二叉树为空!
4	未创建 + 判空二叉树	二叉树为空!
1	创建二叉树	请继续输入合法先序序列
4	已创建 + 判空二叉树	二叉树不是空树
3	已创建 + 清空二叉树	二叉树清空成功
4	已清空二叉树 + 判空二叉树	二叉树是空树
2	已清空二叉树 + 销毁二叉树	二叉树销毁成功

2) 求二叉树深度、查找结点（根据关键字）、结点赋值（根据关键字）、获取兄弟结点（根据关键字）

这些也是二叉树深比较基本的功能，关乎对二叉树深中元素的各种获取，暂时不包括插入和删除操作。

表 2-2　求二叉树深度、查找结点（根据关键字）、结点赋值（根据关键字）、获取兄弟结点（根据关键字）

输入	解释	实际输出
5	未创建 + 求深度	二叉树为空!

6	未创建 + 试图查找结点	二叉树为空!
7	未创建 + 试图给结点赋值	二叉树为空!
8	未创建 + 试图获取兄弟结点	二叉树为空!
1	创建二叉树	请继续输入合法先序序列
5	已创建 + 求深度	答案是 4
6 1	已创建 + 查找关键字为 1 的结点	查找成功! 关键字为 1 的结点数据为 1,a
6 2	已创建 + 查找关键字为 2 的结点	查找失败! 未找到关键字为 2 的结点
7 1 2 new	已创建 + 将关键字 为 1 的结点修改为 2,new	赋值成功!
7 1 3 world	已修改结点 + 试图将关键字 为 1 的结点修改为 3,world	赋值失败! 未找到关键字为 1 的结点
7 3	已修改结点 + 查找 关键字为 2 的结点	查找成功! 关键字为 1 的结点数据为 2,new
8 6	已创建 + 获取 关键字为 6 的兄弟结点	查找成功! 关键字为 6 的兄弟结点数据为 11,x
8 4	已创建 + 获取 关键字为 1 的兄弟结点	查找失败! 未找到关键字为 1 的结点

3) 插入、删除结点，遍历树

插入和删除结点也是二叉树中比较常规且频繁的操作，需要的参数还包括插入后原子树的位置。遍历二叉树分为前序遍历、中序遍历、后序遍历和按层遍历四种方式。

表 2-3　插入、删除结点，遍历树

输入	解释	实际输出
9	未创建 + 试图插入结点	二叉树为空!
1	创建二叉树	请继续输入合法先序序列

11	已创建 + 前序遍历	1,a | 3,c | 6,f | 4,d | 11,x |
12	已创建 + 中序遍历	1,a | 6,f | 4,d | 3,c | 11,x |
13	已创建 + 后序遍历	4,d | 6,f | 11,x | 3,c | 1,a |
14	已创建 + 按层遍历	1,a | 3,c | 6,f | 11,x | 4,d |
9 0 -1 1 one	已创建 + 试图把 1,one 插入作为根结点	插入失败! 关键字 1 与已有结点重复
9 0 -1 2 two	已创建 + 把 2,two 插入作为根结点	插入成功!
11	已插入结点 2+ 前序遍历	2,two | 1,a | 3,c | 6,f | 4,d | 11,x |
10 1	已创建 + 删除关键字为 1 的结点	删除成功!
11	已插入结点 2 + 删除结点 1+ 前序遍历	2,two | 3,c | 6,f | 4,d | 11,x |

4) 读写文件

将二叉树的全部元素写入文件，以及将文件中的数据读取出到二叉树中。

表 2-4　读写文件

输入	解释	实际输出
15 2	未创建 + 写入文件	二叉树为空!
1	创建二叉树	请继续输入合法先序序列
9 0 -1 2 two	已创建 + 把 2,two 插入作为根结点	插入成功!
15 2	已创建 + 写入文件	写入文件成功

2	销毁二叉树	二叉树销毁成功
1	创建二叉树	二叉树创建成功
14	已创建 + 按层遍历	1,a | 3,c | 6,f | 11,x | 4,d |
15 1	已创建 + 读取文件	二叉树已初始化, 不能读取文件 (不能覆盖二叉树)
2	销毁二叉树	二叉树销毁成功
15 1	已销毁 + 读取文件	读取文件成功
14	已读取文件 + 按层遍历	2,two | 1,a | 3,c | 6,f | 11,x | 4,d |

2.5 实验小结

本章节的内容是二叉树的练习，控制台输出的基础框架已经给出，所以只要补充所要求的函数具体实现即可。由于自己在上学期的俱乐部项目中制作过控制台交互程序，故在这次的实践中也更加得心应手。

在这个章节有些生疏的是 C 语言的结构体和链表操作，因此又复习了《C 语言程序设计》中对应章节。

总之，本章节不仅让我加深了对二叉树概念的理解，也学会了利用二叉树实现一些特定数据结构的管理和维护，意识到数据结构的重要性和其巨大应用价值。

3 课程的收获和建议

在学习数据结构的过程中，我深刻认识到了数据结构的重要性和应用价值。数据结构是计算机科学中的基础课程之一，它不仅仅是一些概念的堆砌，更是计算机程序设计的基础，对编写高效稳定的程序和系统非常重要。

通过学习，我深刻理解了数据结构的基本概念和原理，包括数组、链表、栈、队列、树、图等。这些数据结构在现实生活中有广泛的应用，比如图像处理、网络交互、数据库等各个领域。

总的来说，数据结构是计算机科学中非常重要的一个课程，它深刻影响着科技的发展和应用。在学习数据结构时，我认为需要注重理论的学习同时也需要注重实际应用，通过程序实践和项目开发加深对数据结构的理解和掌握。同时也需要不断地思考和探索，不断学习新的数据结构，以应对不断变化的应用场景。

3.1 基于顺序存储结构的线性表实现

在这一章节，我掌握了基于数组实现线性表的基本概念和操作。在这个过程中，我深刻认识到了线性表作为一种基础数据结构的重要性和应用价值。通过该章节的学习，我不仅学到了线性表的基本概念和操作，还学会了如何灵活地使用它构建更加复杂的数据结构和解决实际问题。

在现实生活中，我们可以通过线性表来进行数据的存储和管理，比如对学生信息、图书信息、库存信息等的维护和管理。同时，线性表也在编程中得到了广泛的应用，比如在操作系统调度算法中、文件操作中等等。

总的来说，通过学习这一章节，我强化了个人对线性表的理解，同时也为我后续的学习和实践提供了不少的启示。我建议在学习该章节时，需要结合实际场景灵活运用，同时也需勇于尝试和探索，挖掘出线性表在实际应用中的潜力。

3.2 基于链式存储结构的线性表实现

这一章节的学习内容与上一章节颇为相似，我不仅掌握了链表这种重要的数据结构的基本概念和操作，也深刻认识到了链表与顺序表的异同以及应用场景。通过该章节的学习，我不仅学到了链表的各种基本操作和实现方式，还学到了链表的各种变形结构和相关算法。

在现实生活中，链表的应用也非常广泛，比如对大量数据的存储和管理、游戏引擎的场景切换、追踪路径等等都采用了链表这种数据结构。同时，链表的特性也为算法和数据结构的研究提供了很多思路和启示。

总的来说，通过学习这一章节，我更加深入地了解了链表这种数据结构的本质和应用，对程序设计和数据处理的能力也有所提高。建议在学习该章节时，需注重对链表各种操作的理解和实践，同时也要持续学习和推进对链表的创新研究，探索出更多的应用场景和方法。

3.3 基于二叉链表的二叉树实现

在这一章节，我深入了解了二叉树这种非线性结构的基本原理和实现方式。通过此章节的学习，我不仅学会了如何定义二叉树节点和二叉树的基本操作，还学会了如何进行中序、前序和后序遍历等常见的二叉树操作。同时，我也通过实践学习了如何利用递归和栈等方式处理二叉树操作。

在现实生活中，二叉树的运用非常广泛，比如在搜索引擎中处理大量关键词信息、数据库中处理大量数据等等，都需要运用到二叉树这种数据结构。

总的来说，通过学习这一章节，我深化了对二叉树这种非线性结构的理解和应用。我的建议是，在学习该章节时，需要注重对二叉树各种遍历方式和操作的理解，多做实际练习和项目实践，才能更好地掌握二叉树这种数据结构的应用价值。

3.4 基于邻接表的图实现

在这一章节，我深入了解了图这种非线性结构的基本原理和实现方式。通过该章节的学习，我不仅学会了定义和构建图，还学习了基于邻接表实现图的方法和相关操作。同时，我还通过本章节的实践学习了如何深度遍历和广度遍历图等常见的图操作。

在现实生活中，图的应用也非常广泛，比如社交网络中的人际关系、电子地图中的位置信息、交通路线中的节点，都采用了图这种数据结构。

总的来说，通过学习这一章节，我更深入地了解了图这种非线性结构的本质和应用。我的建议是，在学习该章节时，需要注重对邻接表的理解和实践，多探索该数据结构在图操作中的应用场景和优化方法，以拓展自己的思路和应用能

力。同时，也需要不断加强编程实践，努力提高自己的算法和数据结构水平。

参考文献

[1] 百度百科：顺序表

https://baike.baidu.com/item/%E9%A1%BA%E5%BA%8F%E8%A1%A8/9664274

[2] 百度百科：二叉树

https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%A0%91?fromModule=lemma_search-box

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 <malloc.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASTABLE -1
#define OVERFLOW -2

typedef int status;
typedef int ElemType;

#define LIST_INIT_SIZE 100
#define LISTINCREMENT 10
typedef struct {
    ElemType *elem;
    int length;
    int listsize;
}SqList;
typedef struct
{
    struct { char name[30];
             SqList L;
    } elem[10];
    int length;
    int listsize;
}LISTS;
```

```
30
31  status    InitList  (SqList& L);
32  status    DestroyList (SqList& L);
33  status    ClearList (SqList&L);
34  status   ListEmpty(SqList  L);
35  int   ListLength(SqList  L);
36  status   GetElem(SqList L, int   i ,ElemType& e);
37  status   LocateElem(SqList L,ElemType e);
38  status   PriorElem(SqList  L,ElemType cur,ElemType& pre_e);
39  status   NextElem(SqList L,ElemType cur,ElemType& next_e);
40  status    ListInsert  (SqList&L,int  i ,ElemType e);
41  status    ListDelete (SqList&L,int  i ,ElemType& e);
42  status    ListTrabverse (SqList  L);
43  status     SaveList(SqList  L,char  FileName[]);
44  status     LoadList(SqList  &L,char FileName[]);
45  status   AddList(LISTS &Lists,char  ListName[]);
46  int   LocateList(LISTS Lists ,char  ListName[]);
47  status   RemoveList(LISTS &Lists,char ListName[]);
48  status   MaxSubArray(SqList&L);
49  status   SubArrayNum(SqList&L,int k);
50  status    SortList (SqList&L);
51  status   GetList(LISTS&Lists,SqList &L,int  num);
52  int   main(void)
53  {
54      SqList  L;
55      LISTS List;
56      List . length  = 0;
57      List .  listsize   = 10;
58      int  op = 1  ,  num = 0 ,  e  = 0,pre_e  = 0,next_e  = 0, result  = 0,
            num2= 0;
59      char  filename [30];
```

```
60      DestroyList(L);
61      while(op)
62      {
63          system("cls");
64          printf ("\n\n");
65          printf ("␣␣␣␣␣␣Menu␣for␣Linear␣Table␣On␣Sequence␣Structure
                ␣\n");
66          printf ("
                ————————————————————————————————————————————\
                n");
67          printf ("␣␣␣␣␣␣␣␣␣␣1.␣InitList\t\t2. DestroyList\n");
68          printf ("␣␣␣␣␣␣␣␣␣␣3.␣ClearList\t\t4. ListEmpty\n");
69          printf ("␣␣␣␣␣␣␣␣␣␣5.␣ListLength\t\t6. GetElem\n");
70          printf ("␣␣␣␣␣␣␣␣␣␣7.␣LocateElem\t\t8.PriorElem\n");
71          printf ("␣␣␣␣␣␣␣␣␣␣9.␣NextElem\t\t10. ListInsert \n");
72          printf ("␣␣␣␣␣␣␣␣␣␣11.␣ListDelete\t12. ListTrabverse \n");
73          printf ("␣␣␣␣␣␣␣␣␣␣13.␣MaxSubArray\t14.SubArrayNum\n");
74          printf ("␣␣␣␣␣␣␣␣␣␣15.␣SortList\t\t16. SaveList\n");
75          printf ("␣␣␣␣␣␣␣␣␣␣17.␣LoadList\t\t18.EnterListManagerMode\
                n");
76          printf ("␣␣␣␣␣␣␣␣␣␣0.␣Exit\n");
77          printf ("
                ————————————————————————————————————————————\
                n");
78          printf ("␣␣␣␣请选择你的操作[0~18]:");
79          scanf("%d",&op);
80          switch(op)
81          {
82              case 1:
83                  if( InitList (L)==OK)
84                      printf ("线性表创建成功！\n");
```

```
85                  else
86                      printf (”线性表创建失败！\n”);
87                  getchar () ;
88                  getchar () ;
89                  break;
90              case  2:
91                  if ( DestroyList (L)==OK)
92                      printf (”线性表销毁成功！\n”);
93                  else
94                      printf (”线性表销毁失败！\n”);
95                  getchar () ;
96                  getchar () ;
97                  break;
98              case  3:
99                  if ( ClearList (L)==OK)
100                     printf (”线性表清空成功！\n”);
101                 else
102                     printf (”线性表清空失败！\n”);
103                 getchar () ;
104                 getchar () ;
105                 break;
106             case  4:
107                 if (( result  = ListEmpty(L))==TRUE)
108                     printf (”线性表是空的！\n”);
109                 else  if ( result  ==FALSE)
110                     printf (”线性表不是空的！\n”);
111                 else
112                     printf (”线性表判空失败！\n”);
113                 getchar () ;
114                 getchar () ;
115                 break;
```

```
116             case  5:
117                 if (ListLength(L)==INFEASTABLE)
118                     printf (”获取长度失败！\n”);
119                 else
120                     printf (”线性表长度为%d\n”,ListLength(L));
121                 getchar () ;
122                 getchar () ;
123                 break;
124             case  6:
125                 printf (”请输入你想获取第几个元素:\n”);
126                 scanf(”%d”,&num);
127                 if (GetElem(L,num,e)==INFEASTABLE)
128                     printf (”获取元素失败！\n”);
129                 else
130                     printf (”线性表第%d个元素的值为%d\n”,num,e);
131                 getchar () ;
132                 getchar () ;
133                 break;
134             case  7:
135                 printf (”请输入你想查找的元素:\n”);
136                 scanf(”%d”,&e);
137                 if (( result  = LocateElem(L,e))==INFEASTABLE)
138                     printf (”查找元素失败！\n”);
139                 else  if  ( result  == 0)
140                     printf (”表里没有这个元素哦！\n”);
141                 else
142                     printf (”这个元素是表里面的第%d个元素\n”,
                        LocateElem(L,e));
143                 getchar () ;
144                 getchar () ;
145                 break;
```

```
146             case  8:
147                 printf (”请输入你想获得前驱的元素： \n”);
148                 scanf(”%d”,&e);
149                 if (( result =PriorElem(L,e,pre_e))==INFEASTABLE)
150                     printf (”获取失败了！ \n”);
151                 else  if( result  == ERROR)
152                     printf (”是第一个，或者根本没找到\n”);
153                 else
154                     printf (”是%d哦!\n”,pre_e);
155                 getchar () ;
156                 getchar () ;
157                 break;
158             case  9:
159                 printf (”请输入你想获得后继的元素:\n”);
160                 scanf(”%d”,&e);
161                 if (( result  = NextElem(L,e,next_e)  )==
                        INFEASTABLE)
162                     printf (”获取失败\n”);
163                 else  if  ( result ==ERROR)
164                     printf (”最后一个，或者根本没找到\n”);
165                 else
166                     printf (”是%d哦！ \n”,next_e);
167                 getchar () ;
168                 getchar () ;
169                 break;
170             case  10:
171                 printf (”请输入你想插入的位置及元素:\n”);
172                 scanf(”%d␣%d”,&num,&e);
173                 if (( result = ListInsert (L,num,e))==INFEASTABLE)
174                     printf (”插入失败\n”);
175                 else  if( result ==ERROR)
```

```
176                     printf (”给的位置不合法\n”);
177                 else
178                 {
179                     printf (”插入成功！ \n”);
180 //                  printf (”现在这个列表是这样的:\n”);
181 //                  ListTrabverse (L);
182 //                  测试用临时代码
183                 }
184                 getchar () ;
185                 getchar () ;
186                 break;
187             case  11:
188                 printf (”请输入你想删除的元素的序号： \n”);
189                 scanf(”%d”,&num);
190                 if (( result  =  ListDelete (L,num,e))==INFEASTABLE)
191                     printf (”删除失败\n”);
192                 else  if ( result  == ERROR)
193                     printf (”找不到\n”);
194                 else
195                     printf (”删除了%d\n”,e);
196                 getchar () ;
197                 getchar () ;
198                 break;
199             case  12:
200                 if ( ListTrabverse (L)==INFEASTABLE)
201                     printf (”线性表是空表！ \n”);
202                 getchar () ;
203                 getchar () ;
204                 break;
205             case  13:
206                 if (( result  = MaxSubArray(L))==INFEASTABLE)
```

```
207                 printf (”失败\n”);
208             else
209                 printf (”%d\n”, result );
210             getchar ();
211             getchar ();
212             break;
213         case  14:
214             printf (”请输入你的和： \n”);
215             scanf(”%d”,&num);
216             if (( result  = SubArrayNum(L,num))==INFEASTABLE
                    )
217                 printf (”失败\n”);
218             else
219                 printf (”%d\n”, result );
220             getchar ();
221             getchar ();
222             break;
223         case  15:
224             if (( result  =  SortList (L))==INFEASTABLE)
225                 printf (”失败\n”);
226             else  if( result  == ERROR)
227                 printf (”表为空\n”);
228             else
229                 printf (”整理完毕\n”);
230             getchar ();
231             getchar ();
232             break;
233         case  16:
234             printf (”请输入文件名:\n”);
235             scanf(”%s”,filename);
236             if (( result  = SaveList(L,filename))  == OK)printf(”保
```

```
                存完成\n");
237             else   printf ("保存失败\n");
238             getchar () ;
239             getchar () ;
240             break;
241         case  17:
242             printf ("请输入文件名:\n");
243             scanf("%s",filename);
244             if (( result  = LoadList(L,filename))==OK)printf("加载
                完成\n");
245             else   printf ("加载失败\n");
246             getchar () ;
247             getchar () ;
248             break;
249         case  18:
250             printf ("
                ————————————————————————————————————————\
                n");
251             printf ("␣␣␣␣␣␣␣␣␣␣␣␣已进入多线性表管理模式\n");
252             printf ("
                ————————————————————————————————————————\
                n");
253             printf ("␣␣␣␣␣␣␣␣␣␣1.␣LocateList\t\t2.RemoveList\n");
254             printf ("␣␣␣␣␣␣␣␣␣␣3.␣ClearList\t\t4.ListEmpty\n");
255             printf ("␣␣␣␣␣␣␣␣␣␣5.␣ListLength\t\t6.GetElem\n");
256             printf ("␣␣␣␣␣␣␣␣␣␣7.␣LocateElem\t\t8.PriorElem\n");
257             printf ("␣␣␣␣␣␣␣␣␣␣9.␣NextElem\t\t10. ListInsert \n");
258             printf ("␣␣␣␣␣␣␣␣␣␣11.␣ListDelete\t12. ListTrabverse \
                n");
259             printf ("␣␣␣␣␣␣␣␣␣␣13.␣MaxSubArray\t14.
                SubArrayNum\n");
```

```
260                 printf ("␣␣␣␣␣␣␣␣␣␣15.␣SortList\t\t16.SaveList\n");
261                 printf ("␣␣␣␣␣␣␣␣␣␣17.␣LoadList\t\t18.AddList\n");
262                 printf ("␣␣␣␣␣␣␣␣␣␣0.␣Quit\n");
263                 printf ("
                        ————————————————————————————————————————————\
                    n");
264                 printf ("␣␣␣␣请选择你的操作[0~18]:");
265                 scanf("%d",&op);
266                 switch(op)
267                 {
268                     case 1:
269                         printf ("请输入线性表名字: \n");
270                         scanf("%s",filename);
271                         if (( result = LocateList( List ,filename))!= 0)
                                printf ("是第%d个\n",result);
272                         else  printf ("没找到\n");
273                         getchar ();
274                         getchar ();
275                         break;
276                     case 2:
277                         printf ("请输入你想操作的线性表序号:\n");
278                         scanf("%d",&num);
279                         if (RemoveList(List, List .elem[num-1].name)
                            ==OK)
280                             printf ("线性表移除成功! \n");
281                         else
282                             printf ("线性表移除失败! \n");
283                         getchar ();
284                         getchar ();
285                         break;
286                     case 3:
```

```
287                 printf (”请输入你想操作的线性表序号:\n”);
288                 scanf(”%d”,&num);
289                 if ( ClearList ( List .elem[num−1].L)==OK)
290                     printf (”线性表清空成功！ \n”);
291                 else
292                     printf (”线性表清空失败！ \n”);
293                 getchar () ;
294                 getchar () ;
295                 break;
296             case  4:
297                 printf (”请输入你想操作的线性表序号:\n”);
298                 scanf(”%d”,&num);
299                 if (( result  = ListEmpty(List .elem[num−1].L))
                        ==TRUE)
300                     printf (”线性表是空的！ \n”);
301                 else  if ( result  ==FALSE)
302                     printf (”线性表不是空的！ \n”);
303                 else
304                     printf (”线性表判空失败！ \n”);
305                 getchar () ;
306                 getchar () ;
307                 break;
308             case  5:
309                 printf (”请输入你想操作的线性表序号:\n”);
310                 scanf(”%d”,&num);
311                 if (ListLength( List .elem[num−1].L)==
                        INFEASTABLE)
312                     printf (”获取长度失败！ \n”);
313                 else
314                     printf (”线性表长度为%d\n”,ListLength(L
                            ));
```

```
315                 getchar () ;
316                 getchar () ;
317                 break;
318             case  6:
319                 printf (”请输入你想操作的线性表序号:\n”);
320                 scanf(”%d”,&num);
321                 printf (”请输入你想获取第几个元素:\n”);
322                 scanf(”%d”,&num2);
323                 if (GetElem(List.elem[num−1].L,num2,e)==
                        INFEASTABLE)
324                     printf (”获取元素失败！\n”);
325                 else
326                     printf (”线性表第%d个元素的值为%d\n”,
                            num2,e);
327                 getchar () ;
328                 getchar () ;
329                 break;
330             case  7:
331                 printf (”请输入你想操作的线性表序号:\n”);
332                 scanf(”%d”,&num);
333                 printf (”请输入你想查找的元素:\n”);
334                 scanf(”%d”,&e);
335                 if (( result  = LocateElem(List.elem[num−1].L,
                        e))==INFEASTABLE)
336                     printf (”查找元素失败！\n”);
337                 else  if  ( result  == 0)
338                     printf (”表里没有这个元素哦！\n”);
339                 else
340                     printf (”这个元素是表里面的第%d个元素
                            \n”,LocateElem(List.elem[num−1].L,e))
                            ;
```

```
341                 getchar () ;
342                 getchar () ;
343                 break;
344             case  8:
345                 printf (”请输入你想操作的线性表序号:\n”);
346                 scanf(”%d”,&num);
347                 printf (”请输入你想获得前驱的元素： \n”);
348                 scanf(”%d”,&e);
349                 if (( result =PriorElem(List .elem[num−1].L,e,
                        pre_e))==INFEASTABLE)
350                     printf (”获取失败了！ \n”);
351                 else  if( result  == ERROR)
352                     printf (”是第一个，或者根本没找到\n”);
353                 else
354                     printf (”是%d哦!\n”,pre_e);
355                 getchar () ;
356                 getchar () ;
357                 break;
358             case  9:
359                 printf (”请输入你想操作的线性表序号:\n”);
360                 scanf(”%d”,&num);
361                 printf (”请输入你想获得后继的元素:\n”);
362                 scanf(”%d”,&e);
363                 if (( result  = NextElem(List.elem[num−1].L,e,
                        next_e) )== INFEASTABLE)
364                     printf (”获取失败\n”);
365                 else  if  ( result ==ERROR)
366                     printf (”最后一个，或者根本没找到\n”);
367                 else
368                     printf (”是%d哦！ \n”,next_e);
369                 getchar () ;
```

```
370                     getchar () ;
371                     break;
372                 case  10:
373                     printf (”请输入你想操作的线性表序号:\n”);
374                     scanf(”%d”,&num);
375                     printf (”请输入你想插入的位置及元素:\n”);
376                     scanf(”%d␣%d”,&num2,&e);
377                     if (( result = ListInsert ( List .elem[num−1].L,
                            num2,e))==INFEASTABLE)
378                         printf (”插入失败\n”);
379                     else  if( result ==ERROR)
380                         printf (”给的位置不合法\n”);
381                     else
382                     {
383                         printf (”插入成功！ \n”);
384 //                      printf (”现在这个列表是这样的:\n”);
385 //                      ListTrabverse (L);
386 //                      测试用临时代码
387                     }
388                     getchar () ;
389                     getchar () ;
390                     break;
391                 case  11:
392                     printf (”请输入你想操作的线性表序号:\n”);
393                     scanf(”%d”,&num);
394                     printf (”请输入你想删除的元素的序号： \n”);
395                     scanf(”%d”,&num2);
396                     if (( result  =  ListDelete ( List .elem[num−1].L,
                            num2,e))==INFEASTABLE)
397                         printf (”删除失败\n”);
398                     else  if( result  == ERROR)
```

```
399                 printf (”找不到\n”);
400             else
401                 printf (”删除了%d\n”,e);
402             getchar () ;
403             getchar () ;
404             break;
405         case  12:
406             printf (”请输入你想操作的线性表序号:\n”);
407             scanf(”%d”,&num);
408             if ( ListTrabverse ( List .elem[num−1].L)==
                    INFEASTABLE)
409                 printf (”线性表是空表！ \n”);
410             getchar () ;
411             getchar () ;
412             break;
413         case  13:
414             printf (”请输入你想操作的线性表序号:\n”);
415             scanf(”%d”,&num);
416             if (( result  = MaxSubArray(List.elem[num−1].
                    L))==INFEASTABLE)
417                 printf (”失败\n”);
418             else
419                 printf (”%d\n”, result ) ;
420             getchar () ;
421             getchar () ;
422             break;
423         case  14:
424             printf (”请输入你想操作的线性表序号:\n”);
425             scanf(”%d”,&num);
426             printf (”请输入你的和： \n”);
427             scanf(”%d”,&num2);
```

```
428                 if (( result  = SubArrayNum(List.elem[num−1].
                        L,num))==INFEASTABLE)
429                     printf (”失败\n”);
430                 else
431                     printf (”%d\n”, result ) ;
432                 getchar () ;
433                 getchar () ;
434                 break;
435             case  15:
436                 printf (”请输入你想操作的线性表序号:\n”);
437                 scanf(”%d”,&num);
438                 if (( result  =  SortList ( List . elem[num−1].L))
                        ==INFEASTABLE)
439                     printf (”失败\n”);
440                 else  if ( result  == ERROR)
441                     printf (”表为空\n”);
442                 else
443                     printf (”整理完毕\n”);
444                 getchar () ;
445                 getchar () ;
446                 break;
447             case  16:
448                 printf (”请输入你想操作的线性表序号:\n”);
449                 scanf(”%d”,&num);
450                 printf (”请输入文件名:\n”);
451                 scanf(”%s”,filename) ;
452                 if (( result  = SaveList( List . elem[num−1].L,
                        filename))  == OK)printf(”保存完成\n”);
453                 else   printf (”保存失败\n”);
454                 getchar () ;
455                 getchar () ;
```

```
456                         break;
457                     case  17:
458                         printf (”请输入你想操作的线性表序号:\n”);
459                         scanf(”%d”,&num);
460                         printf (”请输入文件名:\n”);
461                         scanf(”%s”,filename);
462                         if (( result  = LoadList( List .elem[num-1].L,
                                filename))==OK)printf(”加载完成\n”);
463                         else   printf (”加载失败\n”);
464                         getchar () ;
465                         getchar () ;
466                         break;
467                     case  18:
468                         printf (”请输入线性表名字:\n”);
469                         scanf(”%s”,filename);
470                         if (( result  = AddList( List , filename ))==OK)
                                printf(”保存完毕\n”);
471                         else   printf (”保存失败\n”);
472                         getchar () ;
473                         getchar () ;
474                         break;
475                     case  0:
476                         break;
477                 }
478 //              case  18:
479 //                  printf (”请输入线性表名字:\n”);
480 //                  scanf(”%s”,filename);
481 //                  if (( result  = AddList( List , filename ))==OK)printf(”保
    存完毕\n”);
482 //                  else   printf (”保存失败\n”);
483 //                  getchar () ;
```

```
484 //                  getchar();
485 //                  break;
486 //              case 19:
487 //                  printf("请输入线性表名字: \n");
488 //                  scanf("%s",filename);
489 //                  if((result = LocateList(List,filename))!= 0) printf
    ("是第%d个\n",result);
490 //                  else printf("没找到\n");
491 //                  getchar();
492 //                  getchar();
493 //                  break;
494 //              case 20:
495 //                  printf("请输入线性表名字: \n");
496 //                  scanf("%s",filename);
497 //                  if((result = RemoveList(List,filename)) == OK)
    printf("删除完毕\n");
498 //                  else printf("删除失败\n");
499 //                  getchar();
500 //                  getchar();
501 //                  break;
502 //              case 21:
503 //                  printf("请输入要加载的线性表的序号: \n");
504 //                  scanf("%d",&num);
505 //                  if((result = GetList(List,L,num))== OK) printf("加
    载完毕\n");
506 //                  else printf("加载失败\n");
507 //                  getchar();
508 //                  getchar();
509 //                  break;
510                 case 0:
511                     break;
```

```
512             }
513         }
514         printf (”\t欢迎下次再使用本系统！ \n”);
515 }
516 status    InitList (SqList& L)
517 {
518         if (L.elem != NULL)
519         {
520             return  INFEASTABLE;
521         }
522         else
523         {
524             L.elem = (ElemType *)malloc(sizeof(ElemType)*
                    LIST_INIT_SIZE);
525             L.length = 0;
526             L. listsize  = LIST_INIT_SIZE;
527         }
528         return  OK;
529 }
530 status   DestroyList (SqList& L)
531 {
532         if (L.elem == NULL)
533         {
534             return  INFEASTABLE;
535         }
536         else
537         {
538             free (L.elem);
539             L.elem = NULL;//置空
540             L.length = 0 ;
541             L. listsize  = 0;
```

```
542         }
543         return  OK;
544 }
545 status   ClearList (SqList& L)
546 {
547
548         if (L.elem == NULL)
549                 return  INFEASTABLE;
550         else
551         {
552                 free (L.elem);
553                 L.elem = (ElemType *)malloc(sizeof(ElemType)*
                        LIST_INIT_SIZE);//分配空间
554                 L.length  = 0;
555                 L. listsize   = 0;
556         }
557         return  OK;
558 }
559 status  ListEmpty(SqList  L)
560 {
561         if (L.elem !=  NULL)
562         {
563                 if (L.length  == 0) // 看length就行
564                         return  TRUE;
565                 return  FALSE;
566         }
567         else
568                 return  INFEASTABLE;
569 }
570 status  ListLength(SqList  L)
571 {
```

```
572     if(L.elem != NULL)
573         return L.length; //返回length就行
574     else
575         return INFEASTABLE;
576 }
577 status GetElem(SqList L,int i,ElemType &e)
578 {
579     if(L.elem != NULL)
580     {
581         if(i>L.length || i < 1)
582             return ERROR;
583         e = L.elem[i-1];
584         return OK;
585     }
586     else
587         return INFEASTABLE;
588 }
589 int LocateElem(SqList L,ElemType e)
590 {
591     if(L.elem != NULL)
592     {
593         for(int i = 0; i < L.length;i++)
594         {
595             if(L.elem[i] == e) // 确定e的位置
596                 return i+1;
597         }
598         return 0;
599     }
600     else
601         return INFEASTABLE;
602 }
```

```
603 status  PriorElem(SqList  L,ElemType e,ElemType &pre)
604 {
605     if(L.elem != NULL)
606     {
607         for(int  i = 0;  i < L.length;i++)
608         {
609             if(L.elem[i] == e)//遍历找到e
610             {
611                 if(i == 0)
612                     return  ERROR;
613                 pre = L.elem[i-1];
614                 return  OK;
615             }
616         }
617         return  ERROR;
618     }
619     else
620         return  INFEASTABLE;
621 }
622 status  NextElem(SqList L,ElemType e,ElemType &next)
623 {
624     if(L.elem != NULL)
625     {
626         for(int  i = 0;  i < L.length;i++)
627         {
628             if(L.elem[i] == e)  //遍历找到e
629             {
630                 if(i == L.length-1)
631                     return  ERROR;
632                 next = L.elem[i+1];
633                 return  OK;
```

```
634                 }
635             }
636             return  ERROR;
637         }
638         else
639             return  INFEASTABLE;
640 }
641 status    ListInsert (SqList  &L,int  i,ElemType e)
642 {
643         if(L.elem != NULL)
644         {
645             if(i>L.length +1|| i<1)
646                 return  ERROR;
647             ElemType * temp = (ElemType *)malloc(sizeof(ElemType)*(
                    LIST_INIT_SIZE+1));//留出空间
648             for( int  j  = 0;  j  < i-1 ;  j++)
649                 temp[j ]  = L.elem[j ];
650             temp[i-1] = e;   //插入e
651             for( int  j  = i-1 ;  j  < L.length ;j++)
652                 temp[j+1] = L.elem[j ];
653             L.length  ++;
654             L.  listsize   ++;
655             free(L.elem);
656             L.elem = temp;
657             return  OK;
658         }
659         else
660             return  INFEASTABLE;
661 }
662 status    ListDelete (SqList  &L,int  i,ElemType &e)
663 {
```

```
664     if(L.elem != NULL)
665     {
666         if(i>L.length || i<1)
667             return ERROR;
668         ElemType * temp = (ElemType *)malloc(sizeof(ElemType)*(
                LIST_INIT_SIZE-1)); //创造临时线性表
669         for(int j = 0; j < i-1 ; j++)
670             temp[j] = L.elem[j];
671         e = L.elem[i-1]; //剔除要去掉的元素，同时存到e里面
672         for(int j = i-1 ; j < L.length-1;j++)
673             temp[j] = L.elem[j+1];
674         L.length --;
675         L. listsize --;
676         free(L.elem);
677         L.elem = temp;
678         return OK;
679     }
680     else
681         return INFEASTABLE;
682 }
683 status ListTrabverse(SqList L)
684 {
685     bool flag = false;
686     if(L.elem != NULL)
687     {
688         printf("\n------------all elements
                ------------------------\n");
689         for(int i = 0; i < L.length; i++)//遍历输出
690         {
691             if(flag)
692                 printf(" ");
```

```
693                 else
694                     flag  =  true ; // flag用来标识第一个，第一个前面不打
                        空格
695                 printf (”%d”,L.elem[i]) ;
696             }
697             printf (”\n————————————————␣end␣
                ————————————————————\n”);
698             return  OK;
699         }
700         else
701             return  INFEASTABLE;
702 }
703 status    SaveList(SqList  L,char  FileName[])
704 {
705     if(L.elem !=  NULL)
706     {
707         FILE  *fp;  //保存为二进制文件
708         fp=fopen(FileName,”wb”);
709         fwrite (L.elem, sizeof (ElemType),L.length , fp) ;
710         fclose (fp) ;
711         return  OK;
712     }
713     else
714         return  INFEASTABLE;
715 }
716 status    LoadList(SqList  &L,char FileName[])
717 {
718     if(L.elem == NULL)
719     {
720         FILE *fp;
721         L.elem=(ElemType *) malloc(sizeof(ElemType)*
```

```
                LIST_INIT_SIZE);
722             L. listsize   = LIST_INIT_SIZE;
723             L.length=0;
724             fp=fopen(FileName,"rb");
725             while( fread (L.elem+L.length, sizeof (ElemType),1,fp)) // 快读
726                 L.length++;
727             return  OK;
728         }
729         else
730             return  INFEASTABLE;
731 }
732 status  AddList(LISTS &Lists,char  ListName[])
733 {
734         if ( Lists . length<10)
735         {
736             int  i  =  0;
737             while(ListName[i]!='\0')  // 输入名字
738             {
739                 Lists .elem[ Lists . length ].name[i]  =  ListName[i];
740                 i++;
741             }
742             Lists .elem[ Lists . length ].L.elem = NULL;
743             InitList  ( Lists .elem[ Lists . length ].L); // 初始化
744             Lists . length++;
745             return  OK;
746         }
747         return  ERROR;
748 }
749 status  RemoveList(LISTS &Lists,char ListName[])
750 {
751         bool  flag  =  true ;
```

```
752     int  j  = 0;
753     for( int  i  = 0;  i  < Lists . length ;  i++)
754     {
755         flag  = true ;
756         j  = 0;
757         while(ListName[j]!='\0')  // 比对名字
758         {
759             if( Lists .elem[i ]. name[j]  != ListName[j ])
760                 flag  = false ;
761             j++;
762         }
763         if( flag ) // 如果找到了就删除
764         {
765             for( int  j  = i ;j  < Lists . length ;j++)
766                 Lists .elem[j ]  = Lists .elem[j +1];
767             Lists . length--;
768             return  OK;
769         }
770     }
771     return  ERROR;//没找到报错
772 }
773 int  LocateList (LISTS Lists , char  ListName[])
774 {
775     bool  flag  = true ;
776     int  j  = 0;
777     for( int  i  = 0;  i  < Lists . length ;  i++)
778     {
779         flag  = true ;
780         j  = 0;
781         while(ListName[j]!='\0') // 比对顺序表名字
782         {
```

```
783                 if( Lists .elem[i ].name[j]  !=  ListName[j ])
784                     flag  =  false ;
785                 j++;
786             }
787             if( flag )
788             {
789                 return  i+1;  //记住返回的是逻辑上的位置，不是数组里
                      的序号
790             }
791         }
792         return  0;
793 }
794 status  MaxSubArray(SqList&L)
795 {
796         int  res=-114514,sum = 0;
797         int  e  =  0;
798         if (L.elem == NULL || L. length  == 0)
799             return  INFEASTABLE;
800         for( int  i=0;i<L.length ; i++)  //遍历所有可能，寻找最大
801         {
802             for( int  j=i+1;j<=L.length ; j++)
803             {
804                 sum=0;
805                 for( int  k=i+1;k<=j;k++)
806                 {
807                     GetElem(L,k,e);
808                     sum += e;
809                 }
810                 res= res  > sum ?  res  :  sum;
811             }
812         }
```

```
813     return  res;
814 }
815 status  SubArrayNum(SqList&L,int k)
816 {
817     if(L.elem == NULL)
818         return  INFEASTABLE;
819     int  e = 0,cnt = 0,sum = 0;
820     for(int  i=0;i<L.length;i++) //遍历寻找值相等的子数组，用cnt记
        录个数
821     {
822         for(int  j=i+1;j<=L.length;j++)
823         {
824             sum=0;
825             for(int  k=i+1;k<=j;k++)
826             {
827                 GetElem(L,k,e);
828                 sum += e;
829             }
830             if(sum == k)
831                 cnt ++;
832         }
833     }
834     return  cnt;
835 }
836 status  SortList(SqList&L)
837 {
838     if(L.elem != NULL)
839     {
840         if(L.length == 0)
841             return  ERROR;
842         else
```

```
843         { //冒泡排序
844             int max = L.elem[0],maxNum = 0,temp = 0;
845             for(int i = 0 ; i < L.length; i ++)
846             {
847                 max = L.elem[0];
848                 maxNum = 0;
849                 for(int j = 0 ; j < L.length - i;j++)
850                 {
851                     if(max < L.elem[j])
852                     {
853                         max = L.elem[j];
854                         maxNum = j;
855                     }
856                 }
857                 temp = L.elem[L.length-1-i];
858                 L.elem[L.length-1-i] = max;
859                 L.elem[maxNum] = temp;
860             }
861         }
862         return OK;
863     }
864     return INFEASTABLE;
865 }
```

5 附录 B 基于链式存储结构线性表实现的源程序

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

#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define error -3
#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 LNode{ //单链表（链式结构）结点的定义
    ElemType data;
    struct LNode *next;
}LNode,*LinkList;

LinkList L;

typedef struct {  //线性表的集合类型定义
    struct { char name[30];
        LinkList L;
    } 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  sortList (LinkList &L){
239 // 冒泡法进行单链表排序
240         if (!L) return INFEASIBLE;
241         if (!L->next) return ERROR;
242         LinkList pre = NULL, cur = NULL, next = NULL, end = NULL,
                temp = NULL;
243         while(L->next != end){
244             for(pre = L, cur = L->next, next = L->next->next; next !=
                    end ; pre = pre->next, cur = cur->next, next = next->
                    next){
245                 if(cur->data > next->data){
246                     cur->next = next->next;
247                     pre->next = next;
248                     next->next = cur;
249                     temp = cur;
250                     cur = next;
251                     next = temp;
252                 }
253             }
254             end = cur;
255         }
256         return OK;
257 }
258
```

```
259 status SaveList(LinkList L,char FileName[])
260 // 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。
261 {
262     if(!L) return INFEASIBLE;
263     FILE *fp;
264     int i;
265     char ch;
266     if ((fp = fopen(FileName,"wb")) == NULL){
267         printf ("File open error\n");
268         exit(-1);
269     }
270     ch = fgetc(fp);
271     if(ch != EOF){
272         printf ("该文件不能读入！ \n");
273         return ERROR;
274     }
275     if ((fp = fopen(FileName, "wb")) == NULL) {
276         printf ("File open error\n");
277         exit(-1);
278     }
279     LinkList p;
280     p = L->next;
281     while (p)
282     {
283         fwrite(&p->data,sizeof(ElemType), 1, fp);
284         p = p->next;
285     }
286     fclose(fp);
287     return OK;
288
```

```
289 }
290
291 status  LoadList(LinkList  &L,char FileName[])
292 // 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。
293 {
294     if(L)  return  INFEASIBLE;
295     FILE  *fp;
296     L=(LinkList)malloc( sizeof(LNode));
297     int  temp;
298     if  ((fp  =  fopen(FileName, "rb"))  == NULL){
299         printf ("File␣open␣error\n␣");
300         exit(-1);
301     }
302     LinkList  t=L;
303     while  (fread(&temp,  sizeof(ElemType), 1,  fp))
304     {
305         LinkList  n  =  (LinkList)malloc( sizeof(LNode));
306         n->data = temp;
307         t->next = n;
308         t  = n;
309         t->next = NULL;
310     }
311     fclose(fp);
312     return  OK;
313
314 }
315
316 status  AddList(LISTS &Lists,char  ListName[])
317 // 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。
```

```
318 {
319     for(int i = 0; i < Lists.length; i++){
320         if(!strcmp(Lists.elem[i].name,ListName)) return
                INFEASIBLE;
321     }
322     strcpy(Lists.elem[Lists.length].name,ListName);
323     Lists.elem[Lists.length].L = NULL;
324     InitList (Lists.elem[Lists.length].L);
325     Lists.length++;
326     return OK;
327 }
328
329 status RemoveList(LISTS &Lists,char ListName[])
330 // Lists中删除一个名称为ListName的线性表
331 {
332     for(int i = 0; i < Lists.length; i++)
333         if(!strcmp(ListName, Lists.elem[i].name)){
334             if(Lists.elem[i].L)
335                 DestroyList(Lists.elem[i].L);
336             for (int j = i; j < Lists.length - 1; j++)
337                 Lists.elem[j] = Lists.elem[j + 1];
338             Lists.length--;
339             return OK;
340         }
341     return ERROR;
342
343 }
344
345 int LocateList(LISTS Lists,char ListName[])
346 // 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
```

```
347 {
348     if (! Lists .elem)  return  INFEASIBLE;//疑问未解决
349     for ( int  i  = 0;  i  < Lists . length ;  i++)
350         if (! strcmp(ListName,  Lists .elem[i ]. name))
351             return  i  + 1;
352     return  0;
353
354 }
355
356 status   TraverseList (LISTS Lists ){
357 //  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。
358     if ( Lists . length  == 0)  return  INFEASIBLE;
359     printf (”\n——————————all␣names␣————————————————————\
          n”);
360     for ( int  i  = 0;  i  < Lists . length ;  i++){
361         printf (”%s”,Lists .elem[i ]. name);
362         if (i  !=  Lists . length  − 1)  printf (”␣”);
363     }
364     printf (”\n————————————————————␣end␣
          ————————————————————————\n”);
365     return  OK;
366 }
367
368 status   SelectList (LISTS Lists ,  int  i ){
369 //  进行线性表的选择
370     if ( Lists . length  == 0)  return  INFEASIBLE;
371     if (i  < 1  ||  i  > Lists . length )  return  ERROR;
372     L =  Lists .elem[i  − 1]. L;
373     return  OK;
374 }
```

```
375
376 status   reverseList (LinkList  &L){
377 // 迭代反转思想从头开始依次反转链表
378     if (!L)  return  INFEASIBLE;
379     if (!L->next) return  ERROR;
380     LinkList  p = NULL, q = L->next, r = L->next->next;
381     while(1){
382         q->next = p;
383         if (!r)  break;
384         p = q;
385         q = r;
386         r = r->next;
387     }
388     L->next = q;
389     return  OK;
390 }
391
392 status  RemoveNthFromEnd(LinkList L, int n){
393 // 移除倒数第n个元素
394     if (!L)  return  INFEASIBLE;
395     int  length = ListLength(L);
396     if(n < 1 || n > length) return  ERROR;
397     int  pre = length - n + 1;
398     LinkList  p = L, tmp = NULL;
399     while(--pre){
400         p = p->next;
401     }
402     tmp = p->next;
403     p->next = tmp->next;
404     free(tmp);
405     return  OK;
```

```
406 }
407
408 status  CreatList(LinkList  L){
409     LinkList  p = L;
410     while(1){
411         LinkList  s= (LinkList)malloc( sizeof(LinkList));
412         scanf("%d", &s->data);
413         if(s->data == 0)
414             break;
415         p->next = s;
416         p = s;
417         s->next = NULL;
418     }
419     return  OK;
420 }
421
422 int  main(){
423     int  op=1;
424     int  length, flag, temp, num = 0;
425     char  FileName[100];
426     char  Name[20];
427     Lists.length = 0;
428     while(op){
429         system("cls");
430         printf ("\n\n");
431         printf ("␣␣␣␣␣␣␣␣␣␣␣␣Menu␣for␣Linear␣Table␣On␣Chain␣
                Structure␣\n");
432         printf ("␣␣␣␣␣␣␣␣␣␣
                ————————————————————————————————————————————————————————————
                n");
433         printf ("␣␣␣␣␣␣␣␣␣␣1.␣InitList␣␣␣␣线性表初始化␣␣␣␣␣␣␣␣␣13.
```

```
␣sortList␣␣␣␣␣␣␣␣␣线性表排序\n");
434 printf ("␣␣␣␣␣␣␣␣␣␣2.␣DestroyList␣线性表销毁␣␣␣␣␣␣␣␣␣␣
    14.␣SaveList␣␣␣␣␣␣␣␣␣线性表文件保存\n");
435 printf ("␣␣␣␣␣␣␣␣␣␣3.␣ClearList␣␣␣线性表清空␣␣␣␣␣␣␣␣␣␣
    15.␣LoadList␣␣␣␣␣␣␣␣␣线性表文件录入\n");
436 printf ("␣␣␣␣␣␣␣␣␣␣4.␣ListEmpty␣␣␣线性表判空␣␣␣␣␣␣␣␣␣␣
    16.␣AddList␣␣␣␣␣␣␣␣␣␣多线性表添加\n");
437 printf ("␣␣␣␣␣␣␣␣␣␣5.␣ListLength␣␣线性表获取长度␣␣␣␣␣␣␣
    17.␣RemoveList␣␣␣␣␣␣␣多线性表删除\n");
438 printf ("␣␣␣␣␣␣␣␣␣␣6.␣GetElem␣␣␣␣␣线性表元素获取␣␣␣␣␣␣␣
    18.␣LocateList␣␣␣␣␣␣␣多线性表位置查找\n");
439 printf ("␣␣␣␣␣␣␣␣␣␣7.␣LocateElem␣␣线性表元素查找␣␣␣␣␣␣␣
    19.␣TraverseList␣␣␣␣␣多线性表遍历\n");
440 printf ("␣␣␣␣␣␣␣␣␣␣8.␣PriorElem␣␣␣线性表元素前驱获取␣␣␣
    20.␣SelectList␣␣␣␣␣␣␣线性表操作选择\n");
441 printf ("␣␣␣␣␣␣␣␣␣␣9.␣NextElem␣␣␣␣线性表元素后继获取␣␣␣
    21.␣reverseList␣␣␣␣␣␣线性表翻转\n");
442 printf ("␣␣␣␣␣␣␣␣␣␣10.␣ListInsert␣线性表元素插入␣␣␣␣␣␣␣22.
    ␣RemoveNthFromEnd␣移除倒数元素\n");
443 printf ("␣␣␣␣␣␣␣␣␣␣11.␣ListDelete␣线性表元素删除␣\n");
444 printf ("␣␣␣␣␣␣␣␣␣␣12.␣ListTraverse␣␣␣线性表遍历␣\n");
445 printf ("␣␣␣␣␣␣␣␣␣␣0.␣Exit␣␣␣␣␣␣␣␣退出\n");
446 printf ("␣␣␣␣␣␣␣␣␣␣
    ————————————————————————————————————————————————————————————————————————————————
    n");
447 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");
448 printf ("\n␣␣␣␣␣␣␣␣␣␣当前操作的线性表为：");
449 if(num < 1|| num > Lists.length){
450     if(num > Lists.length){
451         L = NULL;
```

```
452                 num = 0;
453             }
454             printf (”默认线性表”);
455             if (!L)
456                 printf (”(未创建)”);
457             printf (”\n\n\n”);
458         }
459         else {
460             printf (”%s”,Lists .elem[num − 1].name);
461             if (!L)
462                 printf (”(未创建)”);
463             printf (”\n\n\n”);
464         }
465
466         if(op > 24 || op < 0)
467             printf (”上一步命令出错！请根据菜单正确输入！\n\n\n”);
468         printf (”请选择你的操作[0~22]:”);
469         scanf(”%d”,&op);
470         switch(op){
471             case 1:
472                 // printf (”\n———IntiList功能待实现！\n”);
473                 if( InitList (L) == OK) printf(”线性表创建成功！\n”);
474                 else printf (”线性表创建失败！\n”);
475                 getchar () ; getchar () ;
476                 break;
477             case 2:
478                 // printf (”\n———DestroyList功能待实现！\n”);
479                 if( DestroyList (L) == OK) printf(”线性表销毁成功！\
                    n”);
480                 else printf (”线性表销毁失败！\n”);
481                 getchar () ; getchar () ;
```

```
482                 break;
483             case  3:
484                 // printf (”\n————ClearList功能待实现！\n”);
485                 flag  =  ClearList (L);
486                 if( flag  == OK) printf(”线性表清空成功！\n”);
487                 else  if  ( flag  == ERROR) printf(”线性表清空失败！\n
                        ”);
488                 else   printf (”线性表未创建！\n”);
489                 getchar () ; getchar () ;
490                 break;
491             case  4:
492                 // printf (”\n————ListEmpty功能待实现！\n”);
493                 if(ListEmpty(L) == OK) printf(”线性表为空！\n”);
494                 else  if(ListEmpty(L) == INFEASIBLE) printf(”线性表
                        未创建！\n”);
495                 else   printf (”线性表非空！\n”);
496                 getchar () ; getchar () ;
497                 break;
498             case  5:
499                 // printf (”\n————ListLength功能待实现！\n”);
500                 length  =  ListLength(L);
501                 if( length  !=  INFEASIBLE) printf(”线性表的长度为:
                        %d\n”, length);
502                 else   printf (”线性表未创建!\n”);
503                 getchar () ; getchar () ;
504                 break;
505             case  6:
506                 // printf (”\n————GetElem功能待实现！\n”);
507                 int  x，y;
508                 printf (”请输入要获取元素的位置：”);
509                 scanf(”%d”,&x);
```

```
510                 flag  = GetElem(L, x,  y);
511                 if( flag  == INFEASIBLE) printf("线性表未创建!\n");
512                 else  if( flag  == OK) printf("该元素为:  %d\n", y);
513                 else  printf ("位置不合法!  \n");
514                 getchar () ; getchar () ;
515                 break;
516             case  7:
517                 // printf ("\n----LocateElem功能待实现!  \n");
518                 int  a;
519                 printf ("请输入想要查找的元素:  ");
520                 scanf("%d",&a);
521                 flag  = LocateElem(L, a) ;
522                 if( flag  == INFEASIBLE) printf("线性表未创建!\n");
523                 else  if( flag )  printf ("该元素存在且元素逻辑索引
                        为:  %d\n", flag);
524                 else  printf ("该元素不存在!  \n");
525                 getchar () ; getchar () ;
526                 break;
527             case  8:
528                 // printf ("\n----PriorElem功能待实现!  \n");
529                 printf ("请输入想要查找的元素(获取前驱):  ");
530                 scanf("%d",&a);
531                 flag  = PriorElem(L,  a,  temp);
532                 if( flag  == INFEASIBLE) printf("线性表未创建!\n");
533                 else  if( flag  == OK) printf("该元素存在且前驱元素
                        为:  %d\n", temp);
534                 else  if( flag  == ERROR)printf("该元素不存在!  \n");
535                 else  printf ("该元素不存在前驱!  \n");
536                 getchar () ; getchar () ;
537                 break;
538             case  9:
```

```
539                 // printf ("\n----NextElem功能待实现！\n");
540                 printf ("请输入想要查找的元素(获取后继)：");
541                 scanf("%d",&a);
542                 flag = NextElem(L, a, temp);
543                 if( flag == INFEASIBLE) printf("线性表未创建!\n");
544                 else if( flag == OK) printf("该元素存在且后继元素
                        为：%d\n", temp);
545                 else if( flag == ERROR) printf("该元素不存在！\n");
546                 else printf ("该元素不存在后继！\n");
547                 getchar () ; getchar () ;
548                 break;
549             case 10:
550                 // printf ("\n----ListInsert功能待实现！\n");
551                 int i, e;
552                 printf ("请输入要插入的元素位置：");
553                 scanf("%d",&i);
554                 printf ("请输入要插入的元素：");
555                 scanf("%d",&e);
556                 if( ListInsert (L, i, e) == OK) printf("线性表插入成
                        功！\n");
557                 else printf ("线性表插入失败！\n");
558                 getchar () ; getchar () ;
559                 break;
560             case 11:
561                 // printf ("\n----ListDelete功能待实现！\n");
562                 printf ("请输入要删除的元素位置：");
563                 scanf("%d",&i);
564                 if( ListDelete (L, i, e) == OK){
565                     printf ("线性表删除成功！\n");
566                     printf ("删除的元素为：%d\n",e);
567                 }
```

```
568                     else   printf (”线性表删除失败！\n”);
569                     getchar () ; getchar () ;
570                     break;
571                 case  12:
572                     // printf (”\n————ListTraverse功能待实现！\n”);
573                     if (! ListTraverse (L))   printf (”线性表是空表！\n”);
574                     getchar () ; getchar () ;
575                     break;
576 //                  case  13:
577 //                      // printf (”\n————MaxSubArray功能待实现！\n”);
578 //                      if (!L)  printf (”线性表未创建！\n”);
579 //                      else   if  (ListEmpty(L))  printf (”线性表为空！\n”);
580 //                      else   printf (”最大子数组之和为：%d\n”,
    MaxSubArray(L));
581 //                      getchar () ; getchar () ;
582 //                      break;
583 //                  case  12:
584 //                      // printf (”\n————SubArrayNum功能待实现！\n”);
585 //                      if (!L){ printf (”线性表未创建！\n”);getchar() ;
    getchar () ;break;}
586 //                      else   if  (ListEmpty(L))  { printf (”线性表为空！\n”);
    getchar() ; getchar () ;break;}
587 //                      printf (”请输入寻找的连续数组的和：”);
588 //                      scanf(”%d”,&flag);
589 //                      printf (”和为数%d的连续数组数目为：%d\n”,flag,
    SubArrayNum(L,flag));
590 //                      getchar () ; getchar () ;
591 //                      break;
592                 case  13:
593                     // printf (”\n————sortList功能待实现！\n”);
594                     flag  =  sortList (L);
```

```
595             if( flag  == ERROR) printf("线性表是空表！\n");
596             else  if( flag  == INFEASIBLE) printf("线性表未创
                    建！\n");
597             else  printf ("线性表排序成功！\n");
598             getchar () ; getchar () ;
599             break;
600         case  14:
601             // printf ("\n----SaveList功能待实现！\n");
602             printf ("请输入要保存的文件名称：");
603             scanf("%s",FileName);
604             flag  =  SaveList(L,  FileName);
605             if( flag  == INFEASIBLE) printf("文件读入失败！\n");
606             else  if( flag  == ERROR);
607             else  printf ("文件读入成功！\n");
608             getchar () ; getchar () ;
609             break;
610         case  15:
611             // printf ("\n----LoadList功能待实现！\n");
612             printf ("请输入要录入的文件名称：");
613             scanf("%s",FileName);
614             if(LoadList(L,  FileName) == INFEASIBLE) printf("文
                    件录入失败！\n");
615             else  printf ("文件录入成功！\n");
616             getchar () ; getchar () ;
617             break;
618         case  16:
619             // printf ("\n----AddList功能待实现！\n");
620             if( Lists . length  == MAXlength) {
621                 printf ("多线性表管理已满，请清除某些线性表后
                        再操作！\n");
622                 getchar () ; getchar () ;
```

```
623                 break;
624             }
625             printf (”请输入新增线性表的名称：”);
626             scanf(”%s”,Name);
627             flag = AddList( Lists , Name);
628             if ( flag == INFEASIBLE) printf(”该名称的线性表已经
                    存在！\n”);
629             else printf (”%s已成功添加！\n”,Name);
630             getchar () ; getchar () ;
631             break;
632         case 17:
633             // printf (”\n——RemoveList功能待实现！\n”);
634             if ( Lists . length == 0) {
635                 printf (”多线性表管理已空，请添加某些线性表后
                        再操作！\n”);
636                 getchar () ; getchar () ;
637                 break;
638             }
639             printf (”请输入删除线性表的名称：”);
640             scanf(”%s”,Name);
641             flag = RemoveList(Lists, Name);
642             if ( flag == OK)printf(”%s已成功删除！\n”,Name);
643             else printf (”线性表不存在！\n”);
644             getchar () ; getchar () ;
645             break;
646         case 18:
647             // printf (”\n——LocateList功能待实现！\n”);
648             printf (”请输入查找线性表的名称：”);
649             scanf(”%s”,Name);
650             if ( LocateList ( Lists , Name)) printf (”该线性表的逻辑
                    索引为：%d\n”, LocateList(Lists, Name));
```

```
651                 else   printf (”线性表查找失败！\n”);
652                 getchar () ; getchar () ;
653                 break;
654             case  19:
655                 // printf (”\n————TraverseList功能待实现！\n”);
656                 if ( TraverseList ( Lists ) == INFEASIBLE) printf(”多线
                        性表为空！\n”);
657                 getchar () ; getchar () ;
658                 break;
659             case  20:
660                 // printf (”\n————SelectList功能待实现！\n”);
661                 printf (”请选择要处理的线性表的逻辑索引：”);
662                 scanf(”%d”,&flag);
663                 if ( SelectList ( Lists ,  flag ) == OK) {
664                     printf (”已选取成功！\n”);
665                     num = flag ;
666                 }
667                 else   printf (”选取失败！\n”);
668                 getchar () ; getchar () ;
669                 break;
670             case  21:
671                 // printf (”\n————reverseList功能待实现！\n”);
672                 flag  =  reverseList (L);
673                 if ( flag  == INFEASIBLE) printf(”线性表未创建！\n”);
674                 else  if ( flag  == ERROR) printf(”该线性表为空！\n”);
675                 else   printf (”链表反转成功！\n”);
676                 getchar () ; getchar () ;
677                 break;
678             case  22: // printf (”\n————RemoveNthFromEnd功能待实
                    现！\n”);
679                 int  n;
```

```
680                     printf (”请输入想要删除倒数第几个元素： ”);
681                     scanf(”%d”,&n);
682                     flag  = RemoveNthFromEnd(L, n);
683                     if( flag  == INFEASIBLE) printf(”线性表未创建！ \n”);
684                     else  if( flag  == ERROR) printf(”位置不合法！ \n”);
685                     else  printf (”删除成功！ \n”);
686                     getchar () ; getchar () ;
687                     break;
688                 case  0:
689                     break;
690             }
691         }
692         printf (”欢迎下次再使用本系统！ \n”);
693         return  0;
694 }
```

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 ;
```

```
30  }TREES;
31  TREES trees;          //线性表集合的定义TREES
32
33  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  status  DestroyBiTree(BiTree &T)
87  // 将二叉树销毁
88  {
89          if (!T) return OK;
90          DestroyBiTree(T->lchild);
```

```
91      DestroyBiTree(T->rchild);
92      free(T);
93      T = NULL;
94      return OK;
95
96  }
97
98  int MAX(int a, int b){
99      if(a >= b) return a;
100     else return b;
101 }
102
103 int BiTreeDepth(BiTree T)
104 //求二叉树T的深度
105 {
106     if(!T) return ERROR;
107     return MAX(BiTreeDepth(T->lchild), BiTreeDepth(T->rchild)) + 1;
108
109 }
110
111
112
113 BiTNode* LocateNode(BiTree T,KeyType e)
114 //查找结点
115 {
116     if(!T) return NULL;
117     if(T->data.key == e) return T;
118     if(LocateNode(T->lchild, e))
119         return LocateNode(T->lchild, e);
120     else
121         return LocateNode(T->rchild, e);
```

```
122
123 }
124
125 status  Assign(BiTree &T,KeyType e,TElemType value)
126 //实现结点赋值。
127 {
128     BiTree p = LocateNode(T, value.key);
129     BiTree t = LocateNode(T, e);
130     if(t == NULL || (value.key!= e && p != NULL)) return ERROR;
131     t->data.key = value.key;
132     strcpy(t->data.others, value.others);
133     return OK;
134
135 }
136
137 BiTNode* GetSibling(BiTree T,KeyType e)
138 //实现获得兄弟结点
139 {
140     if(!T) return NULL;
141     if(T->lchild && T->lchild->data.key == e) return T->rchild;
142     if(T->rchild && T->rchild->data.key == e) return T->lchild;
143     if(GetSibling(T->lchild, e))
144         return GetSibling(T->lchild, e);
145     else
146         return GetSibling(T->rchild, e);
147
148 }
149
150 BiTNode* GetFabling(BiTree T,KeyType e)
151 //实现获得父亲结点
152 {
```

```
153     if(!T) return NULL;
154     if(T->lchild && T->lchild->data.key == e) return T;
155     if(T->rchild && T->rchild->data.key == e) return T;
156     if(GetFabling(T->lchild, e))
157         return GetFabling(T->lchild, e);
158     else
159         return GetFabling(T->rchild, e);
160
161 }
162
163 status InsertNode(BiTree &T,KeyType e,int LR,TElemType c)
164 //插入结点。
165 {
166     if(!T) return INFEASIBLE;
167     BiTree t = LocateNode(T, e);
168     BiTree p = (BiTree)malloc( sizeof(BiTNode));
169     p->data.key = c.key;
170     strcpy(p->data.others, c.others);
171     if(LocateNode(T, c.key)) return ERROR;
172     if(LR == -1){
173         p->rchild = T;
174         p->lchild = NULL;
175         T = p;
176         return OK;
177     }
178     if(!t) return ERROR;
179     if(LR == 0){
180         p->rchild = t->lchild;
181         p->lchild = NULL;
182         t->lchild = p;
183         return OK;
```

```
184         }
185         else  if(LR == 1){
186             p->rchild  =  t->rchild;
187             p->lchild  =  NULL;
188             t->rchild  =  p;
189             return  OK;
190         }
191         free(p);
192
193     }
194
195     status  DeleteNode(BiTree &T,KeyType e)
196     // 删除结点。
197     {
198         BiTree  tmp =  (BiTree)malloc( sizeof(BiTNode));
199         BiTree  t  =  NULL, p = NULL, q = NULL, l = NULL;
200         int  flag  =  0;
201         tmp->lchild  =  T;
202         tmp->rchild  =  NULL;
203         t  =  GetFabling(tmp,  e);
204         //   printf("%d",t->data.key);
205         if(!t)  return  ERROR;
206         if(t->lchild  && t->lchild->data.key == e)  {q  =  t->lchild;  flag
                =  1;}
207         else  if(t->rchild  && t->rchild->data.key == e)  {q  =  t->rchild;
                flag  =  0;}
208         if(!q->lchild  && !q->rchild){
209             free(q);
210             if( flag )  t->lchild  =  NULL;
211             else  t->rchild  =  NULL;
212         }
```

```
213     else  if(!q->lchild  && q->rchild){
214         if(flag)  t->lchild  = q->rchild;
215         else  t->rchild  = q->rchild;
216         free(q);
217     }
218     else  if(q->lchild  && !q->rchild){
219         if(flag)  t->lchild  = q->lchild;
220         else  t->rchild  = q->lchild;
221         free(q);
222     }
223     else{
224         l  = q->lchild;
225         while(1){
226             if(1){
227                 BiTreestack[top++] = l;
228                 l  = l->lchild;
229             }
230             else{
231                 l  = BiTreestack[--top];
232                 if(!l->rchild  && !top)
233                     break;
234                 l  = l->rchild;
235             }
236         }
237         l->rchild  = q->rchild;
238         if(flag)  t->lchild  = q->lchild;
239         else  t->rchild  = q->lchild;
240         free(q);
241     }
242     T = tmp->lchild;
243     free(tmp);
```

```
244         return  OK;
245
246 }
247
248 void   visit (BiTree  T)
249 {
250         printf (”␣%d,%s”,T−>data.key,T−>data.others);
251 }
252
253  status   PreOrderTraverse(BiTree  T,void  (∗ visit )(BiTree))
254  // 先序遍历二叉树T
255  {
256         if (!T)  return   INFEASIBLE;
257         visit (T);
258        PreOrderTraverse(T−>lchild,   visit );
259        PreOrderTraverse(T−>rchild,   visit );
260         return  OK;
261
262  }
263
264  status   InOrderTraverse(BiTree  T,void  (∗ visit )(BiTree))
265  // 中序遍历二叉树T
266  {
267         if (!T)  return   INFEASIBLE;
268        BiTree  p  =  T;
269        while(p   ||   top){
270              if(p){
271                    BiTreestack[top++] = p;
272                    p  =  p−>lchild;
273              }
274              else {
```

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

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

```
337             }
338             BiTreestack[top++] = s->rchild;
339             BiTreestack[top++] = s->lchild;
340             fwrite(&s->data, sizeof(TElemType), 1, fp);
341         }
342         fclose(fp);
343         return OK;
344
345 }
346
347 status  dfs(BiTree &T, TElemType definition[])
348 {
349     i++;
350     if( definition[i].key == -1) return OK;
351     if( definition[i].key == 0) T = NULL;
352     else
353     {
354         T = (BiTNode*)malloc(sizeof(BiTNode));
355         T->data = definition[i];
356         if(flag1[definition[i].key]) flag2 = 1;
357         flag1[T->data.key] = 1;
358         dfs(T->lchild, definition);
359         dfs(T->rchild, definition);
360     }
361     return OK;
362 }
363
364 status  LoadBiTree(BiTree &T, char FileName[])
365 // 读入文件FileName的结点数据，创建二叉树
366 {
367     if(T) return INFEASIBLE;
```

```
368     FILE *fp;
369     if ((fp = fopen(FileName,"rb")) == NULL)
370     {
371         printf ("File open error!\n");
372         return ERROR;
373     }
374     i = 0;
375     TElemType definition[100];
376     while(fread(&definition[i++], sizeof(TElemType), 1, fp));
377      definition[i].key = -1;
378     i = -1;
379     dfs(T, definition);
380      fclose(fp);
381     if(flag2) return ERROR;
382     return OK;
383
384 }
385
386 status BiTreeEmpty(BiTree T){
387     if(!T) return FALSE;
388     else return TRUE;
389 }
390
391 status AddList(TREES &trees,char ListName[])
392 // 需要在TREES中增加一个名称为ListName的空线性表
393 {
394     TElemType definition[100];
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     i = 0;
401     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）： ”);
402     do {
403         scanf(”%d%s”, &definition[i ]. key,  definition [i ]. others );
404     } while ( definition [i++].key != −1);
405     CreateBiTree( trees .elem[ trees . length ]. L, definition );
406     count = 0;
407     trees . length++;
408     return OK;
409 }
410
411 status  DestoryList (TREES &trees,char ListName[])
412 // TREES中删除一个名称为ListName的线性表
413 {
414     for ( int i = 0; i < trees . length ; i++)
415         if (! strcmp(ListName, trees .elem[i ]. name)){
416             if ( trees .elem[i ]. L)
417                 DestroyBiTree( trees .elem[i ]. L);
418             for ( int j = i ; j < trees . length − 1; j++)
419                 trees .elem[j ] = trees .elem[j + 1];
420             trees . length−−;
421             return OK;
422         }
423     return ERROR;
424
425 }
426
```

```
427 int  LocateList(TREES trees,char  ListName[])
428 // 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
429 {
430     if(! trees.elem)  return  INFEASIBLE;//疑问未解决
431     for(int  i = 0; i < trees.length; i++)
432         if(!strcmp(ListName, trees.elem[i].name))
433             return  i + 1;
434     return  0;
435
436 }
437
438 status  TraverseList(TREES trees){
439 // 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。
440     if(trees.length == 0)  return  INFEASIBLE;
441     printf("\n------------all␣names␣----------------------\
          n");
442     for(int  i = 0; i < trees.length; i++){
443         printf("%s",trees.elem[i].name);
444         if(i != trees.length - 1)  printf("␣");
445     }
446     printf("\n--------------------␣end␣
          ----------------------\n");
447     return  OK;
448 }
449
450 status  SelectList(TREES trees, int  i){
451 // 进行线性表的选择
452     if(trees.length == 0)  return  INFEASIBLE;
453     if(i < 1 || i > trees.length)  return  ERROR;
```

```
454     T = trees.elem[i - 1].L;
455     return OK;
456 }
457
458 int MaxPathSum(BiTree T){
459 //返回最大路径和
460     if(!T) return ERROR;
461     return MAX(MaxPathSum(T->lchild), MaxPathSum(T->rchild)) + T
            ->data.key;
462 }
463
464 BiTree LowestCommonAncestor(BiTree T, int e1, int e2){
465 //返回公共祖先
466     if(!T || T->data.key == e1 || T->data.key == e2) return T;
467     BiTree l = LowestCommonAncestor(T->lchild, e1, e2);
468     BiTree r = LowestCommonAncestor(T->rchild, e1, e2);
469     if(l && !r) return l;
470     if(!l && r) return r;
471     if(!l && !r) return NULL;
472     if(l && r) return T;
473 }
474
475 status InvertTree(BiTree &T){
476 //二叉树翻转
477     BiTree p = T, t;
478     BiTreequeue[0] = p;
479     l = 0, r = 1;
480     while(l != r){
481         p = BiTreequeue[l++];
482         t = p->lchild;
483         p->lchild = p->rchild;
```

```
            p->rchild = t;
            if(p->lchild) BiTreequeue[r++] = p->lchild;
            if(p->rchild) BiTreequeue[r++] = p->rchild;
        }
        return OK;
}

status ClearList(BiTree &T){
        ClearBiTree(T);
        return OK;
}

int main(){
        int op=1;
        int length, num = 0, LR;
        int ans, e, ee;
        char FileName[100];
        char Name[20];
        TElemType definition[100], value;
        BiTree t = NULL;
        trees.length = 0;
        while(op){
            system("cls");
            printf("\n\n");
            printf("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Menu␣for␣
                Binary␣Tree␣␣\n");
            printf("␣␣␣␣␣␣␣␣␣␣
                ------------------------------------------------------------------------
                n");
            printf("␣␣␣␣␣␣␣␣␣␣1.␣CreateBiTree␣␣二叉树创建␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣␣␣␣␣14.␣LevelOrderTraverse␣␣␣␣二叉树层序遍历\n");
```

```
511     printf ("␣␣␣␣␣␣␣␣␣␣2.␣DestroyBiTree␣二叉树销毁␣␣␣␣␣␣␣␣␣␣
            ␣␣␣␣␣␣␣15.␣MaxPathSum␣␣␣␣␣␣␣␣␣␣␣␣二叉树最大路径和\
            n");
512     printf ("␣␣␣␣␣␣␣␣␣␣3.␣ClearBiTree␣␣␣二叉树清空␣␣␣␣␣␣␣␣␣␣
            ␣␣␣␣␣␣␣16.␣LowestCommonAncestor␣␣二叉树最近祖先\n")
            ;
513     printf ("␣␣␣␣␣␣␣␣␣␣4.␣BiTreeEmpty␣␣␣二叉树判空␣␣␣␣␣␣␣␣␣
            ␣␣␣␣␣␣␣␣17.␣InvertTree␣␣␣␣␣␣␣␣␣␣␣␣二叉树翻转\n");
514     printf ("␣␣␣␣␣␣␣␣␣␣5.␣BiTreeDepth␣␣␣二叉树获取深度␣␣␣␣␣␣
            ␣␣␣␣␣␣␣18.␣SaveList␣␣␣␣␣␣␣␣␣␣␣␣␣␣二叉树文件保存\n");
515     printf ("␣␣␣␣␣␣␣␣␣␣6.␣LocateNode␣␣␣␣二叉树查找结点␣␣␣␣␣
            ␣␣␣␣␣␣␣␣19.␣LoadList␣␣␣␣␣␣␣␣␣␣␣␣␣␣二叉树文件录入\n")
            ;
516     printf ("␣␣␣␣␣␣␣␣␣␣7.␣Assign␣␣␣␣␣␣␣␣二叉树结点赋值␣␣␣␣␣␣
            ␣␣␣␣␣␣␣20.␣AddList␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣多二叉树表添加\n");
517     printf ("␣␣␣␣␣␣␣␣␣␣8.␣GetSibling␣␣␣␣二叉树获取兄弟结点␣␣␣
            ␣␣␣␣␣␣␣21.␣DestroyList␣␣␣␣␣␣␣␣␣␣␣多二叉树表销毁\n");
518     printf ("␣␣␣␣␣␣␣␣␣␣9.␣InsertNode␣␣␣␣二叉树插入结点␣␣␣␣␣␣
            ␣␣␣␣␣␣␣22.␣LocateList␣␣␣␣␣␣␣␣␣␣␣␣多二叉树表位置查找\
            n");
519     printf ("␣␣␣␣␣␣␣␣␣␣10.␣DeleteNode␣␣␣二叉树删除结点␣␣␣␣␣␣
            ␣␣␣␣␣␣␣23.␣TraverseList␣␣␣␣␣␣␣␣␣多二叉树表遍历\n");
520     printf ("␣␣␣␣␣␣␣␣␣␣11.␣PreOrderTraverse␣␣␣二叉树前序遍历␣
            ␣␣␣␣␣␣24.␣SelectList␣␣␣␣␣␣␣␣␣␣␣␣二叉树操作选择\n");
521     printf ("␣␣␣␣␣␣␣␣␣␣12.␣InOrderTraverse␣␣␣␣二叉树中序遍历␣
            ␣␣␣␣␣␣25.␣ClearList␣␣␣␣␣␣␣␣␣␣␣␣多二叉树表清空\n");
522     printf ("␣␣␣␣␣␣␣␣␣␣13.␣PostOrderTraverse␣␣二叉树后序遍历␣"
            );
523     printf ("␣␣␣␣␣␣␣␣␣␣0.␣Exit␣␣␣␣␣␣␣␣退出\n");
524     printf ("␣␣␣␣␣␣␣␣␣␣
            ----------------------------------------------------------------------
```

```
        n");
525     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");
526     printf ("\n␣␣␣␣␣␣␣␣␣␣当前操作的二叉树为：");
527     if(num < 1|| num > trees.length){
528         if(num > trees.length){
529             T = NULL;
530             num = 0;
531         }
532         printf ("默认二叉树");
533         if(!T)
534             printf ("(未创建)");
535         printf ("\n\n\n");
536     }
537     else{
538         printf ("%s",trees.elem[num - 1].name);
539         if(!T)
540             printf ("(未创建)");
541         printf ("\n\n\n");
542     }
543     if(op > 25 || op < 0)
544         printf ("上一步命令出错！请根据菜单正确输入！\n\n\n");
545     printf ("请选择你的操作[0~26]:");
546     scanf("%d",&op);
547     switch(op){
548         case 1:
549             // printf ("\n----CreateBiTree功能待实现！\n");
550             if(T){
551                 printf ("该二叉树已存在！\n");
552                 getchar(); getchar();
553                 break;
```

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

```
581             if(ans == OK) printf("二叉树清空成功！\n");
582             else  printf ("二叉树清空失败！\n");
583             getchar () ; getchar () ;
584             break;
585         case 4:
586             // printf ("\n----BiTreeEmpty功能待实现！\n");
587             ans = BiTreeEmpty(T);
588             if(ans == FALSE) printf("二叉树为空！\n");
589             else  printf ("二叉树不为空！\n");
590             getchar () ; getchar () ;
591             break;
592         case 5:
593             // printf ("\n----BiTreeDepth功能待实现！\n");
594             length = BiTreeDepth(T);
595             if(length)  printf ("该二叉树的深度为%d！\n", length
                    );
596             else  printf ("二叉树为空！\n");
597             getchar () ; getchar () ;
598             break;
599         case 6:
600             // printf ("\n----LocateNode功能待实现！\n");
601             if(!T) {
602                 printf ("二叉树为空！\n");
603                 getchar () ; getchar () ;
604                 break;
605             }
606             printf ("请输入想要查询的结点关键字： ");
607             scanf("%d",&e);
608             t = LocateNode(T, e);
609             if(t == NULL) printf("该节点不存在！\n");
610             else {
```

```
611                 printf ("该节点存在！结点信息为：%s\n",t->data.
                        others);
612             }
613             t = NULL;
614             getchar () ; getchar () ;
615             break;
616         case 7:
617             // printf ("\n----Assign功能待实现！\n");
618             if (!T) {
619                 printf ("二叉树为空！\n");
620                 getchar () ; getchar () ;
621                 break;
622             }
623             printf ("请输入想要赋值的节点关键字:");
624             scanf("%d", &e);
625             printf ("请输入关键字：");
626             scanf("%d", &value.key);
627             printf ("请输入结点信息：");
628             scanf("%s", value . others ) ;
629             ans = Assign(T, e, value) ;
630             if (ans == OK) printf("结点赋值成功!\n");
631             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");
632             getchar () ; getchar () ;
633             break;
634         case 8:
635             // printf ("\n----GetSibling功能待实现！\n");
636             if (!T) {
637                 printf ("二叉树为空！\n");
638                 getchar () ; getchar () ;
639                 break;
```

```
640                 }
641                 printf ("请输入要获取兄弟结点的关键字：");
642                 scanf("%d",&e);
643                 t = GetSibling(T, e);
644                 if(t == NULL) printf("该结点不存在或不存在兄弟结
                        点!\n");
645                 else {
646                     printf ("该元素兄弟结点获取成功！\n");
647                     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);
648                 }
649                 getchar () ; getchar () ;
650                 break;
651             case 9:
652                 // printf ("\n----InsertNode功能待实现！\n");
653                 if (!T) {
654                     printf ("二叉树为空！\n");
655                     getchar () ; getchar () ;
656                     break;
657                 }
658                 printf ("请输入想要插入的结点关键字：");
659                 scanf("%d", &e);
660                 printf ("请输入关键字：");
661                 scanf("%d", &value.key);
662                 printf ("请输入结点信息：");
663                 scanf("%s", value . others ) ;
664                 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）：");
665                 scanf("%d", &LR);
666                 ans = InsertNode(T, e, LR, value);
```

```
667                 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");
668                 else  printf ("结点插入成功！\n");
669                 getchar(); getchar();
670                 break;
671             case 10:
672                 // printf ("\n----DeleteNode功能待实现！\n");
673                 if(!T) {
674                     printf ("二叉树为空！\n");
675                     getchar(); getchar();
676                     break;
677                 }
678                 printf ("请输入想要删除的结点关键字：");
679                 scanf("%d", &e);
680                 ans = DeleteNode(T, e);
681                 if(ans == ERROR) printf("结点删除失败（请检查该
                        关键字是否存在）!\n");
682                 else  printf ("结点删除成功！\n");
683                 getchar(); getchar();
684                 break;
685             case 11:
686                 // printf ("\n----PreOrderTraverse功能待实现！\n");
687                 if(!T) {
688                     printf ("二叉树为空！\n");
689                     getchar(); getchar();
690                     break;
691                 }
692                 printf ("先序遍历二叉树的结果：\n");
693                 PreOrderTraverse(T, visit);
694                 getchar(); getchar();
695                 break;
```

```
696             case 12:
697                 // printf ("\n----InOrderTraverse功能待实现！\n");
698                 if (!T) {
699                     printf ("二叉树为空！\n");
700                     getchar () ; getchar () ;
701                     break;
702                 }
703                 printf ("中序遍历二叉树的结果：\n");
704                 InOrderTraverse(T, visit ) ;
705                 getchar () ; getchar () ;
706                 break;
707             case 13:
708                 // printf ("\n----PostOrderTraverse功能待实现！\n");
709                 if (!T) {
710                     printf ("二叉树为空！\n");
711                     getchar () ; getchar () ;
712                     break;
713                 }
714                 printf ("后序遍历二叉树的结果：\n");
715                 PostOrderTraverse(T, visit ) ;
716                 getchar () ; getchar () ;
717                 break;
718             case 14:
719                 // printf ("\n----LevelOrderTraverse功能待实现！\n")
                    ;
720                 if (!T) {
721                     printf ("二叉树为空！\n");
722                     getchar () ; getchar () ;
723                     break;
724                 }
725                 printf ("层序遍历二叉树的结果：\n");
```

```
726                 LevelOrderTraverse(T,  visit );
727                 getchar () ; getchar () ;
728                 break;
729             case  15:
730                 // printf (”\n----MaxPathSum功能待实现！\n”);
731                 if (!T) {
732                     printf (”二叉树为空！\n”);
733                     getchar () ; getchar () ;
734                     break;
735                 }
736                 length  = MaxPathSum(T);
737                 printf (”根节点到叶子结点的最大路径和为：%d\n”,
                        length);
738                 getchar () ; getchar () ;
739                 break;
740             case  16:
741                 // printf (”\n----LowestCommonAncestor功能待实
                        现！\n”);
742                 if (!T) {
743                     printf (”二叉树为空！\n”);
744                     getchar () ; getchar () ;
745                     break;
746                 }
747                 printf (”请输入第一个结点：”);
748                 scanf(”%d”, &e);
749                 printf (”请输入第二个结点：”);
750                 scanf(”%d”, &ee);
751                 t  = LocateNode(T, e) ;
752                 if (!t){
753                     printf (”第一个结点不存在！\n”);
754                     getchar () ; getchar () ;
```

```
755                     break;
756                 }
757 //                  else  if(t == T){
758 //                      printf("第一个结点为根节点，不存在祖先！\n
    ");
759 //                      t = NULL;
760 //                      getchar(); getchar();
761 //                      break;
762 //                  }
763                 t = LocateNode(T, ee);
764                 if(!t){
765                     printf("第二个结点不存在！\n");
766                     getchar(); getchar();
767                     break;
768                 }
769 //                  else  if(t == T){
770 //                      printf("第二个结点为根节点，不存在祖先！\n
    ");
771 //                      t = NULL;
772 //                      getchar(); getchar();
773 //                      break;
774 //                  }
775                 t = LowestCommonAncestor(T,e,ee);
776                 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);
777                 t = NULL;
778                 getchar(); getchar();
779                 break;
780             case 17:
781                 // printf("\n———InvertTree功能待实现！\n");
782                 if(!T) {
```

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

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

```
840                 scanf(”%s”,Name);
841                 if( LocateList( trees , Name)) printf(”该二叉树的逻辑
                        索引为: %d\n”, LocateList(trees, Name));
842                 else  printf (”二叉树查找失败! \n”);
843                 getchar () ; getchar () ;
844                 break;
845             case 23:
846                 // printf (”\n----TraverseList功能待实现! \n”);
847                 if( TraverseList ( trees ) == INFEASIBLE) printf(”多二
                        叉树表为空! \n”);
848                 getchar () ; getchar () ;
849                 break;
850             case 24:
851                 // printf (”\n----SelectList功能待实现! \n”);
852                 printf (”请选择要处理的二叉树的逻辑索引: ”);
853                 scanf(”%d”, &ans);
854                 if( SelectList ( trees , ans) == OK) {
855                     printf (”二叉树已选取成功! \n”);
856                     num = ans;
857                 }
858                 else  printf (”二叉树选取失败! \n”);
859                 getchar () ; getchar () ;
860                 break;
861             case 25:
862                 // printf (”\n----ClearBiTree功能待实现! \n”);
863                 if (!T) {
864                     printf (”二叉树为空! \n”);
865                     getchar () ; getchar () ;
866                     break;
867                 }
868                 ClearList (T);
```

```
869                     printf ("二叉树清空成功！\n");
870                     getchar () ; getchar () ;
871                     break;
872 //              case  26:
873 //                  // printf ("\n----GetFabling功能待实现！\n");
874 //                  if (!T) {
875 //                      printf ("二叉树为空！\n");
876 //                      getchar () ; getchar () ;
877 //                      break;
878 //                  }
879 //                  printf ("请输入要获取父亲结点的关键字：");
880 //                  scanf("%d",&e);
881 //                  if (T->data.key == e){
882 //                      printf ("该结点为根结点！");
883 //                  }
884 //                  t = GetFabling(T, e);
885 //                  if (t == NULL) printf("该结点不存在父亲节点!\n");
886 //                  else {
887 //                      printf ("该元素父亲节点获取成功！\n");
888 //                      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);
889 //                  }
890 //                  t = NULL;
891 //                  getchar () ; getchar () ;
892 //                  break;
893             case  0:
894                 break;
895         } // end of switch
896     } // end of while
897     printf ("欢迎下次再使用本系统！\n");
898     return  0;
```

```
899 } //end of main()
```

7 附录 D 基于邻接表图实现的源程序

```
#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 MAX_VERTEX_NUM 20
#define MAXlength 10

typedef int status ;
typedef int KeyType;
typedef enum {DG,DN,UDG,UDN} GraphKind;
typedef struct {
    KeyType key;
    char others [20];
} VertexType; // 顶点类型定义


typedef struct ArcNode {          // 表结点类型定义
    int adjvex;                   // 顶点位置编号
    struct ArcNode *nextarc;      // 下一个表结点指针
} ArcNode;

typedef struct VNode{             // 头结点及其数组类型定义
    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
```

```
}


status  InsertArc(ALGraph &G,KeyType v,KeyType w)
//在图G中增加弧<v,w>，成功返回OK,否则返回ERROR
{
    int i, j;
    ArcNode *p = NULL;
    if((i = LocateVex(G, v)) == -1) return ERROR;
    if((j = LocateVex(G, w)) == -1) return ERROR;
    p = G.vertices[i].firstarc;
    while(p){
        if(p->adjvex == j)
            return ERROR;
        p = p->nextarc;
    }
    ArcNode *t = (ArcNode *)malloc(sizeof(ArcNode));
    t->adjvex = j;
    t->nextarc = G.vertices[i].firstarc;
    G.vertices[i].firstarc = t;
    ArcNode *tt = (ArcNode *)malloc(sizeof(ArcNode));
    tt->adjvex = i;
    tt->nextarc = G.vertices[j].firstarc;
    G.vertices[j].firstarc = tt;
    G.arcnum++;
    return OK;

}

status  DeleteArc(ALGraph &G,KeyType v,KeyType w)
//在图G中删除弧<v,w>，成功返回OK,否则返回ERROR
```

```
{
    int i, j, flag = 0;
    ArcNode *p = NULL, *q = NULL;
    if ((i = LocateVex(G, v)) == -1) return ERROR;
    if ((j = LocateVex(G, w)) == -1) return ERROR;
    p = G.vertices[i].firstarc;
    while(p){
        if(p->adjvex == j){
            flag = 1;
            break;
        }
        p = p->nextarc;
    }
    if (!flag) return ERROR;
    q = p = G.vertices[i].firstarc;
    if(p->adjvex == j){
        G.vertices[i].firstarc = p->nextarc;
        free(p);
    }
    else{
        while(p){
            if(p->adjvex == j){
                q->nextarc = p->nextarc;
                free(p);
                break;
            }
            q = p;
            p = p->nextarc;
        }
    }
    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 ;
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;
```

```
                    continue;
                }
                if(low != top){
                    p = G.vertices[stack[low++]].firstarc ;
                }
            }
            count++;
        }
    }
    return count;
}

int main(){
    int op=1;
    int i, e, ans, j;
    VertexType V[100];
    KeyType VR[100][2];
    VertexType value;
    char FileName[100], Name[100];
    while(op){
        system(”cls”);
        printf(”\n\n”);
        printf(”␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Menu␣for␣
            Graphy␣␣\n”);
        printf(”␣␣␣␣␣␣␣␣␣␣
            ——————————————————————————————————————————————————
            n”);
        printf(”␣␣␣␣␣␣␣␣␣␣1.␣CreateCraph␣␣␣创建图␣␣␣␣␣␣␣␣␣␣␣␣␣␣
            ␣␣␣12.␣BFSTraverse␣␣␣␣␣广度优先搜索遍历\n”);
        printf(”␣␣␣␣␣␣␣␣␣␣2.␣DestroyGraph␣␣销毁图␣␣␣␣␣␣␣␣␣␣␣␣␣␣
            ␣␣␣13.␣SaveGraph␣␣␣␣␣␣␣图文件保存\n”);
```

```
724         printf ("␣␣␣␣␣␣␣␣␣␣3.␣LocateVex␣␣␣␣␣查找顶点␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣14.␣LoadGraph␣␣␣␣␣␣␣␣图文件录入␣␣␣\n");
725         printf ("␣␣␣␣␣␣␣␣␣␣4.␣PutVex␣␣␣␣␣␣␣␣顶点赋值␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣␣15.␣AddList␣␣␣␣␣␣␣␣多图表添加␣␣␣\n");
726         printf ("␣␣␣␣␣␣␣␣␣␣5.␣FirstAdjVex␣␣␣获得第一邻接点␣␣␣␣␣␣
                ␣␣␣16.␣DestroyList␣␣␣␣␣多图表销毁␣␣␣␣\n");
727         printf ("␣␣␣␣␣␣␣␣␣␣6.␣NextAdjVex␣␣␣␣获得下一邻接点␣␣␣␣␣
                ␣␣␣␣17.␣LocateList␣␣␣␣␣␣多图表位置查找␣␣␣␣\n");
728         printf ("␣␣␣␣␣␣␣␣␣␣7.␣InsertVex␣␣␣␣␣插入顶点␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣18.␣TraverseList␣␣␣␣多图表遍历\n");
729         printf ("␣␣␣␣␣␣␣␣␣␣8.␣DeleteVex␣␣␣␣␣删除顶点␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣19.␣SelectList␣␣␣␣␣␣图操作选择\n");
730         printf ("␣␣␣␣␣␣␣␣␣␣9.␣InsertArc␣␣␣␣␣插入弧␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣20.␣VerticesSetLessThanK␣␣␣␣␣距离小于k的顶点集合\n
                ");
731         printf ("␣␣␣␣␣␣␣␣␣␣10.␣DeleteArc␣␣␣␣删除弧␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                ␣␣␣21.␣ShortestPathLength␣␣␣␣␣␣␣顶点间最短路径和长度\
                n");
732         printf ("␣␣␣␣␣␣␣␣␣␣11.␣DFSTraverse␣␣␣深度优先搜索遍历␣␣␣
                ␣␣␣22.␣ConnectedComponentsNums␣␣图的连通分量\n");
733         printf ("␣␣␣␣␣␣␣␣␣␣0.␣Exit␣␣␣␣␣␣␣␣退出\n");
734         printf ("␣␣␣␣␣␣␣␣␣␣
                ----------------------------------------------------------------------------
                n");
735         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");
736         printf ("\n␣␣␣␣␣␣␣␣␣␣当前操作的图为：");
737         if(num < 1|| num > Graphs.length){
738             if(num > Graphs.length){
739                 Graphs.elem[num].G.vexnum = 0;
740                 num = 0;
```

```
741                }
742                printf (”默认图”);
743                if(Graphs.elem[num].G.vexnum == 0)
744                    printf (”(未创建)”);
745                printf (”\n\n\n”);
746            }
747            else{
748                printf (”%s”,Graphs.elem[num − 1].name);
749                if(Graphs.elem[num].G.vexnum == 0)
750                    printf (”(未创建)”);
751                printf (”\n\n\n”);
752            }
753            if(op > 22 || op < 0)
754                printf (”上一步命令出错！请根据菜单正确输入！\n\n\n”);
755            printf (”请选择你的操作[0~22]:”);
756            scanf(”%d”,&op);
757            switch(op){
758                case 1:
759                    // printf (”\n————CreateCraph功能待实现！\n”);
760                    if(Graphs.elem[num].G.vexnum){
761                        printf (”该图已存在！\n”);
762                        getchar(); getchar();
763                        break;
764                    }
765                    i = 0;
766                    printf (”请输入顶点序列(−1␣nil作为结束标志)：”);
767                    do{
768                        scanf(”%d%s”, &V[i].key, V[i]. others);
769                    }while (V[i++].key != −1);
770                    i = 0;
771                    printf (”请输入关系对序列，以−1␣−1结束：”);
```

```
772                 do{
773                     scanf("%d%d", &VR[i][0], &VR[i][1]);
774                 }while (VR[i++][0] != -1);
775                 if (CreateCraph(Graphs.elem[num].G, V, VR) == OK)
776                     printf ("图创建成功！\n");
777                 else
778                     printf ("图创建失败！\n");
779                 getchar () ; getchar () ;
780                 break;
781             case 2:
782                 // printf ("\n----DestroyGraph功能待实现！\n");
783                 if (!Graphs.elem[num].G.vexnum) {
784                     printf ("图为空！\n");
785                     getchar () ; getchar () ;
786                     break;
787                 }
788                 ans = DestroyGraph(Graphs.elem[num].G);
789                 if(ans == OK) printf("图销毁成功！\n");
790                 else  printf ("图销毁失败！\n");
791                 getchar () ; getchar () ;
792                 break;
793             case 3:
794                 // printf ("\n----LocateVex功能待实现！\n");
795                 if (!Graphs.elem[num].G.vexnum) {
796                     printf ("图为空！\n");
797                     getchar () ; getchar () ;
798                     break;
799                 }
800                 printf ("请输入想要查找的顶点关键字：");
801                 scanf("%d", &e);
802                 ans = LocateVex(Graphs.elem[num].G, e);
```

```
803                 if (ans != -1) printf ("图中关键字为%d的顶点的位
                        序为%d\n", e, ans);
804                 else
805                     printf ("图中不存在该顶点！\n");
806                 getchar () ; getchar () ;
807                 break;
808             case 4:
809                 // printf ("\n----PutVex功能待实现！\n");
810                 if (!Graphs.elem[num].G.vexnum) {
811                     printf ("图为空！\n");
812                     getchar () ; getchar () ;
813                     break;
814                 }
815                 printf ("请输入想要修改的顶点的关键字：");
816                 scanf("%d", &e);
817                 printf ("将其顶点值修改为：");
818                 scanf("%d␣%s", &value.key, value. others ) ;
819                 ans = PutVex(Graphs.elem[num].G, e, value) ;
820                 if (ans == ERROR)
821                     printf ("赋值操作失败！\n");
822                 else if (ans == OK)
823                     printf ("已将关键字为%d的顶点值修改为%d,%s\n
                            ", e, value.key, value.others);
824                 getchar () ; getchar () ;
825                 break;
826             case 5:
827                 // printf ("\n-----BiTreeDepth功能待实现！\n");
828                 if (!Graphs.elem[num].G.vexnum) {
829                     printf ("图为空！\n");
830                     getchar () ; getchar () ;
831                     break;
```

```
832                 }
833                 printf (”请输入想要查找其第一邻接点的顶点：”);
834                 scanf(”%d”, &e);
835                 ans = FirstAdjVex(Graphs.elem[num].G, e);
836                 if (ans != -1)
837                     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 );
838                 else
839                     printf (”顶点%d没有第一邻接点！\n”, e);
840                 getchar (); getchar ();
841                 break;
842             case 6:
843                 // printf (”\n----NextAdjVex功能待实现！\n”);
844                 if (!Graphs.elem[num].G.vexnum) {
845                     printf (”图为空！\n”);
846                     getchar (); getchar ();
847                     break;
848                 }
849                 printf (”请输入两个顶点的关键字：”);
850                 scanf(”%d␣%d”, &e, &j);
851                 ans = NextAdjVex(Graphs.elem[num].G, e, j);
852                 if (ans != -1)
853                     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);
854                 else   printf (”无下一邻接顶点！\n”);
855                 getchar (); getchar ();
```

```
856                 break;
857             case 7:
858                 // printf ("\n----InsertVex功能待实现！\n");
859                 if (!Graphs.elem[num].G.vexnum) {
860                     printf ("图为空！\n");
861                     getchar () ; getchar () ;
862                     break;
863                 }
864                 printf ("请输入想要插入的关键字和关键信息：");
865                 scanf("%d␣%s", &value.key, value. others ) ;
866                 ans = InsertVex (Graphs.elem[num].G, value) ;
867                 if (ans == OK)
868                     printf ("顶点␣%d␣%s␣已成功插入图中\n", value.
                        key, value.others);
869                 else if (ans == ERROR)
870                     printf ("插入失败！\n");
871                 getchar () ; getchar () ;
872                 break;
873             case 8:
874                 // printf ("\n----DeleteVex功能待实现！\n");
875                 if (!Graphs.elem[num].G.vexnum) {
876                     printf ("图为空！\n");
877                     getchar () ; getchar () ;
878                     break;
879                 }
880                 printf ("请输入想要删除的顶点的关键字：");
881                 scanf("%d", &e);
882                 ans = DeleteVex(Graphs.elem[num].G, e) ;
883                 if (ans == OK)
884                     printf ("关键字为%d的顶点已从图中删除\n", e);
885                 else if (ans == ERROR)
```

```
886                 printf (”删除失败！\n”);
887             getchar () ; getchar () ;
888             break;
889         case 9:
890             // printf (”\n————InsertArc功能待实现！\n”);
891             if (!Graphs.elem[num].G.vexnum) {
892                 printf (”图为空！\n”);
893                 getchar () ; getchar () ;
894                 break;
895             }
896             printf (”请输入想要插入的弧：”);
897             scanf(”%d␣%d”, &e, &j);
898             ans = InsertArc (Graphs.elem[num].G, e, j);
899             if (ans == OK)
900                 printf (”插入成功！\n”);
901             else if (ans == ERROR)
902                 printf (”插入失败！\n”);
903             getchar () ; getchar () ;
904             break;
905         case 10:
906             // printf (”\n————DeleteArc功能待实现！\n”);
907             if (!Graphs.elem[num].G.vexnum) {
908                 printf (”图为空！\n”);
909                 getchar () ; getchar () ;
910                 break;
911             }
912             printf (”请输入要删除弧的两个端点：”);
913             scanf(”%d␣%d”, &e, &j);
914             ans = DeleteArc(Graphs.elem[num].G, e, j);
915             if (ans == OK)
916                 printf (”删除成功！\n”);
```

```
917                 else  if  (ans  == ERROR)
918                       printf (”删除失败！\n”);
919                 getchar () ; getchar () ;
920                 break;
921           case  11:
922                 // printf (”\n————DFSTraverse功能待实现！\n”);
923                 if (! Graphs.elem[num].G.vexnum) {
924                       printf (”图为空！\n”);
925                       getchar () ; getchar () ;
926                       break;
927                 }
928                 printf (”深度优先搜索遍历：\n”);
929                 DFSTraverse(Graphs.elem[num].G, visit ) ;
930                 printf (”\n”) ;
931                 getchar () ; getchar () ;
932                 break;
933           case  12:
934                 // printf (”\n————BFSTraverse功能待实现！\n”);
935                 if (! Graphs.elem[num].G.vexnum) {
936                       printf (”图为空！\n”);
937                       getchar () ; getchar () ;
938                       break;
939                 }
940                 printf (”广度优先搜索遍历：\n”);
941                 BFSTraverse(Graphs.elem[num].G, visit ) ;
942                 printf (”\n”) ;
943                 getchar () ; getchar () ;
944                 break;
945           case  13:
946                 // printf (”\n————SaveList功能待实现！\n”);
947                 printf (”请输入要保存的文件名称：”);
```

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

```
976                 break;
977             case 16:
978                 // printf ("\n---DestoryList功能待实现！\n");
979                 if(Graphs.length == 0) {
980                     printf ("多图表管理已空，请添加某些图后再操
                        作！\n");
981                     getchar () ; getchar () ;
982                     break;
983                 }
984                 printf ("请输入销毁图的名称：");
985                 scanf("%s",Name);
986                 ans = DestoryList(Graphs, Name);
987                 if(ans == OK)printf("%s已成功销毁！\n",Name);
988                 else  printf ("图不存在！\n");
989                 getchar () ; getchar () ;
990                 break;
991             case 17:
992                 // printf ("\n---LocateList功能待实现！\n");
993                 printf ("请输入查找图的名称：");
994                 scanf("%s",Name);
995                 if( LocateList(Graphs, Name)) printf ("该图的逻辑索
                        引为：%d\n", LocateList(Graphs, Name));
996                 else  printf ("图查找失败！\n");
997                 getchar () ; getchar () ;
998                 break;
999             case 18:
1000                // printf ("\n----TraverseList功能待实现！\n");
1001                if( TraverseList (Graphs) == INFEASIBLE) printf("多
                        图表为空！\n");
1002                getchar () ; getchar () ;
1003                break;
```

```
1004            case 19:
1005                // printf ("\n-----SelectList功能待实现！\n");
1006                printf ("请选择要处理的图的逻辑索引：");
1007                scanf("%d", &ans);
1008                if( SelectList (Graphs, ans) == OK) {
1009                    printf ("已选取成功！\n");
1010                }
1011                else  printf ("选取失败！\n");
1012                getchar () ; getchar () ;
1013                break;
1014            case 20:
1015                // printf ("\n-----VerticesSetLessThanK功能待实现！\
                        n");
1016                if (!Graphs.elem[num].G.vexnum) {
1017                    printf ("图为空！\n");
1018                    getchar () ; getchar () ;
1019                    break;
1020                }
1021                printf ("请输入顶点和距离：");
1022                scanf("%d␣%d", &e, &j);
1023                VerticesSetLessThanK(Graphs.elem[num].G ,e, j ) ;
1024                getchar () ; getchar () ;
1025                break;
1026            case 21:
1027                // printf ("\n-----ShortestPathLength功能待实现！\n");
1028                if (!Graphs.elem[num].G.vexnum) {
1029                    printf ("图为空！\n");
1030                    getchar () ; getchar () ;
1031                    break;
1032                }
1033                printf ("请输入顶点v和顶点w：");
```

```
1034                scanf("%d␣%d", &e, &j);
1035                if(e == j){
1036                    printf("请不要输入两个相同的结点！\n");
1037                }
1038                ans = ShortestPathLength(Graphs.elem[num].G,e,j);
1039                if(ans == Graphs.elem[num].G.vexnum){
1040                    printf("两者间不存在路径！\n");
1041                    getchar(); getchar();
1042                    break;
1043                }
1044                if(ans == ERROR){
1045                    printf("两顶点不都存在！\n");
1046                    getchar(); getchar();
1047                    break;
1048                }
1049                printf("两节点之间的最短路径为：%d\n", ans);
1050                getchar(); getchar();
1051                break;
1052            case 22:
1053                // printf("\n-----ConnectedComponentsNums功能待实
                        现！\n");
1054                if(!Graphs.elem[num].G.vexnum) {
1055                    printf("图为空！\n");
1056                    getchar(); getchar();
1057                    break;
1058                }
1059                ans = ConnectedComponentsNums(Graphs.elem[num].G
                        );
1060                printf("连通分量包含%d个！\n", ans);
1061                getchar(); getchar();
1062                break;
```

```
1063                case  0:
1064                    break;
1065            } // end of switch
1066        } // end of while
1067        printf (”欢迎下次再使用本系统！\n”);
1068        return  0;
1069    } // end of main()
```