

华中科技大学

课程实验报告

课程名称: 数据结构实验

专业班级 本硕博 2301

学 号 U202315752

姓 名 陈宇航

指导教师 向文

报告日期 2024 年 5 月 20 日

计算机科学与技术学院

前言

此页交代了该代码程序运行的操作系统，编译环境，编译语言等，避免由于不兼容导致的报错问题。

此外，操作演示系统时，为了防止各种错误，不要使用中文提示，避免因为输入问题导致的程序报错和乱码

运行的操作系统	Windows
编译环境	CLion
代码语言	C++

目 录

1 基于链式存储结构的线性表实现.....	1
1.1 问题描述	1
1.2 系统设计	1
1.3 系统实现	5
1.4 系统测试	14
1.5 实验小结	33
2 基于邻接表的图实现	35
2.1 问题描述	35
2.2 系统设计	35
2.3 系统实现	41
2.4 系统测试	49
2.5 实验小结	62
3 课程的收获和建议	63
3.1 基于顺序存储结构的线性表实现	63
3.2 基于链式存储结构的线性表实现	63
3.3 基于二叉链表的二叉树实现	64
3.4 基于邻接表的图实现	64
附录	66
附录 A 基于顺序存储结构线性表实现的源程序	66
附录 B 基于链式存储结构线性表实现的源程序	66
附录 C 基于二叉链表二叉树实现的源程序	66
附录 D 基于邻接表图实现的源程序	115

1 基于链式存储结构的线性表实现

1.1 问题描述

- 1) 构造一个具有菜单功能的演示系统，演示采用链式存储的物理结构的程序代码。
- 2) 在主程序中完成函数调用所需参值的准备和函数执行结果的显示。
- 3) 定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等基本运算对应的函数。并给出适当的操作提示显示，可选择以文件的形式进行存储和加载，即将生成的线性表存入到相应的文件中，也可以从文件中获取线性表进行操作。
- 4) 还有一些附加功能，实现多个线性表的管理，最大连续子数组和，和为 K 的子数组，顺序表排序等，以及由多个线性表切换到单个线性表的管理。

1.2 系统设计

链表作为线性结构，是数据结构中最常用、最基本的结构类型，本实验采用链表的线性存储方式，并实现了链表各类基本功能如：增添、删除、遍历等，并在此基础之上附加翻转、倒序删除、排序。多线性表操作等功能，更好地方便使用者对链表进行操作，且附带了语言文字提示，降低了程序的使用难度。

1.2.1 头文件和预定义的声明

我们定义使用 TRUE 和 FALSE 的返回值分别为 1 和 0，表示为返回真假；OK 和 ERROR 表示系统运行是否正常，INFEASIBLE 表示函数运行错误，OVERFLOW 表示线性表内存不足；

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4 // 定义布尔类型TRUE和FALSE
5 #define TRUE 1
6 #define FALSE 0
```

```
7
8 // 定义函数返回值类型
9 #define OK 1
10 #define ERROR 0
11 #define INFEASIBLE -1
12 #define OVERFLOW -2
13
14 // 初始链表的最大长度
15 #define LIST_INIT_SIZE 100
16 // 每次新增的长度
17 #define LISTINCREMENT 10
```

1.2.2 链式存储的线性表的定义

typedef 了 int 基本数据类型为 ElemType，便于后续程序维护以及功能增加等，减少代码的修改量且增加代码的可读性，而且 typedef 结点的结构体变量和指针，减少代码的书写量，也增大了代码的可读性，是程序整体美观。LNode 为节点具体内存内容，包括一个数据域和指针域，指针域指向下一个节点，LinkList 表示指向节点的指针，大多时候作为头节点的指针表示而 LISTS 表示链表的集合

```
1 // 定义数据元素类型
2 typedef int ElemType;
3 typedef int status ;
4
5 // 定义单链表（链式结构）结点的结构体
6 typedef struct LNode{
7     ElemType data; // 结点的数据元素
8     struct LNode *next; // 指向下一个结点的指针
9 }LNode, *LinkList;
10
11 // 定义链表集合的结构体
```

```
12 typedef struct {
13     struct {
14         char name[30]; // 集合的名称，最多可以有 30 个字符
15         LinkList L; // 指向链表头结点的指针
16     }elem[30]; // 集合中最多包含 30 个链表
17     int length; // 集合中包含的链表数目
18 }LISTS;
```

1.2.3 函数总览

1. 初始化表：函数名称是 InitList(L)；初始条件是线性表 L 不存在；操作结果是构造一个空的线性表；
2. 销毁表：函数名称是 DestroyList(L)；初始条件是线性表 L 已存在；操作结果是销毁线性表 L；
3. 清空表：函数名称是 ClearList(L)；初始条件是线性表 L 已存在；操作结果是将 L 重置为空表；
4. 判定空表：函数名称是 ListEmpty(L)；初始条件是线性表 L 已存在；操作结果是若 L 为空表则返回 TRUE, 否则返回 FALSE；
5. 求表长：函数名称是 ListLength(L)；初始条件是线性表已存在；操作结果是返回 L 中数据元素的个数；
6. 获得元素：函数名称是 GetElem(L,i,e)；初始条件是线性表已存在, $1 \leq i \leq \text{ListLength}(L)$ ；操作结果是用 e 返回 L 中第 i 个数据元素的值；
7. 查找元素：函数名称是 LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回 L 中第 1 个与 e 满足关系 compare () 关系的数据元素的位序，若这样的数据元素不存在，则返回值为 0；
8. 获得前驱：函数名称是 PriorElem，初始条件是线性表 L 已存在；操作结果是若是 L 的数据元素，且不是第一个，则用返回它的前驱，否则操作失败，无定义；
9. 获得后继：函数名称是 NextElem，初始条件是线性表 L 已存在；操作结果是若是 L 的数据元素，且不是最后一个，则用返回它的后继，否则操作失败，无定义；

10. 插入元素: 函数名称是 ListInsert, 初始条件是线性表 L 已存在, $1 \leq i \leq \text{ListLength}(L)+1$; 操作结果是在 L 的第 i 个位置之前插入新的数据元素 e。
11. 删除元素: 函数名称是 ListDelete, 初始条件是线性表 L 已存在且非空, $1 \leq i \leq \text{ListLength}(L)$; 操作结果: 删除 L 的第 i 个数据元素, 用 e 返回的值;
12. 遍历表: 函数名称是 ListTraverse, 初始条件是线性表 L 已存在; 操作结果是依次对 L 的每个数据元素调用函数 visit()。

附加功能

- 1) 链表翻转: 函数名称是 reverseList(L), 初始条件是线性表 L 已存在; 操作结果是将 L 翻转;
- 2) 删除链表的倒数第 n 个结点: 函数名称是 RemoveNthFromEnd(L,n); 初始条件是线性表 L 已存在且非空, 操作结果是该链表中倒数第 n 个节点;
- 3) 链表排序: 函数名称是 sortList(L), 初始条件是线性表 L 已存在; 操作结果是将 L 由小到大排序;
- 4) 实现线性表的文件形式保存: 其中, □ 需要设计文件数据记录格式, 以高效保存线性表数据逻辑结构 (D,R) 的完整信息; □ 需要设计线性表文件保存和加载操作合理模式。
- 5) 实现多个线性表管理: 设计相应的数据结构管理多个线性表的查找、添加、移除等功能。

```
1  status  InitList (LinkList &L); // 新建
2  status  DestroyList (LinkList &L); // 销毁
3  status  ClearList (LinkList &L); // 清空
4  status  ListEmpty(LinkList L); // 判空
5  status  ListLength (LinkList L); // 求长度
6  status  GetElem(LinkList L, int i, ElemType &e); // 获取元素
7  status  LocateElem(LinkList L, ElemType e, int (*vis)(int ,int )); // 判
    断位置
8  status  PriorElem(LinkList L, ElemType e, ElemType &pre); // 前驱
9  status  NextElem(LinkList L, ElemType e, ElemType &next); // 后继
10 status  ListInsert (LinkList &L, int i, int num); // 插入
11 status  ListDelete (LinkList &L, int i, ElemType &e); // 删除
```

```
12 status ListTraverse ( LinkList L,void (*vi)( int )); //遍历
13 status AddList(LISTS &Lists,char ListName[]);
14 status RemoveList(LISTS &Lists,char ListName[]);
15 status LocateList (LISTS Lists, char ListName[]);
16 void SearchList (LISTS Lists); //展示已经创建的线性表
17 status compare(int a, int b); //判断位置函数时候调用的比较函数
18 void visit ( int x); //遍历函数时候调用的输出函数
19 void reverseList (LinkList L); //翻转线性表
20 void RemoveNthFromEnd(LinkList L,int n); //删除倒数元素
21 void sortList (LinkList L); //排序
22 void savetofile (LinkList L,char name[]); //保存到文件
23 void getfromfile (LinkList L,char name[]); //读取文件
24 void fun01(); //封装的多个线性表的处理函数
25 void fun02(LinkList &L ); //封装的处理单个线性表的处理函数
26 void menu(); //管理多个线性表的菜单
27 void show_normal(); //单个线性表的菜单
28 void Menuofinsert(); //插入的菜单
```

1.2.4 菜单实现

菜单采用简单移动的 UI，且增加了动漫特效，增加趣味。下图为线性表管理和单个线性表管理的实现

1.3 系统实现

以下主要说明各个主要函数的实现思想，函数和系统实现的源代码放在附录中。注：本实验所有函数在实现功能之前会先对是否已有线性表进行判定，若无线性表，则返回 INFEASIBLE，在各函数具体设计思路中一般不再叙述此条。

1.3.1 status InitList(LinkList &L)

设计思路：该函数接受一个链表 L 作为参数并返回一个状态码。如果链表 L 已经存在，函数返回 INFEASIBLE。否则，它为一个新节点分配内存，作为链表

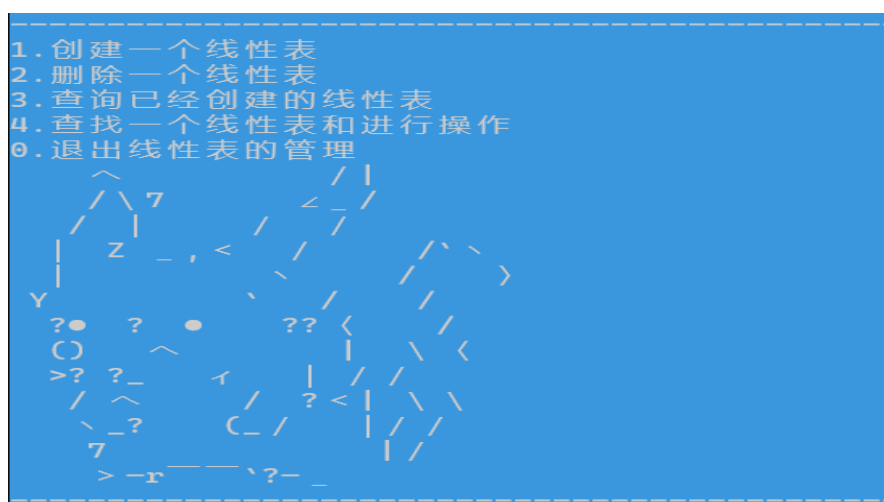


图 1-1 多个线性表管理菜单

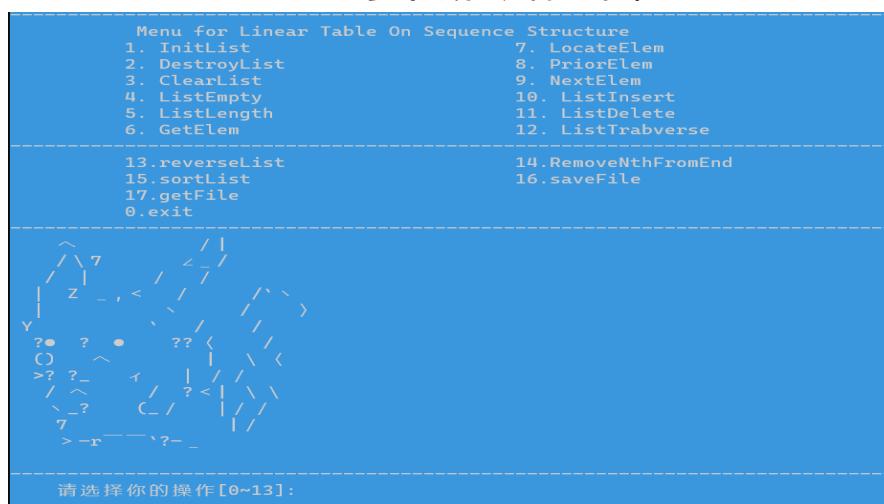


图 1-2 单个线性表管理菜单

的头节点，并将其 next 指针设置为 NULL。然后函数返回 OK。

时间复杂度：O(1)

空间复杂度：O(1)

1.3.2 status DestroyList(LinkList &L)

设计思路：这段代码是一个销毁链表的函数。该函数接受一个链表‘L’作为参数并返回一个状态码。如果链表‘L’不存在，函数返回‘INFEASIBLE’。否则，它定义两个指针‘p’和‘q’，其中‘p’指向当前节点，‘q’指向下一个节点。当当前节点不为空时，循环继续。将‘q’指向当前节点的下一个节点，释放当前节点的空间，并将指针‘p’指向‘q’，继续循环。最后函数返回‘OK’。

时间复杂度为：O(n)

空间复杂度为: $O(1)$

1.3.3 status ClearList(LinkList &L)

设计思路: 该函数接受一个链表 L 作为参数并返回一个状态码。如果链表 L 不存在, 函数返回 INFEASIBLE。如果链表 L 为空, 也不需要操作, 函数返回 INFEASIBLE。否则, 它定义一个指针 p 指向第一个元素节点。当 p 不为空时, 循环继续。释放当前节点的空间, 并将指针 p 指向下一个节点, 继续循环。最后将头节点的指针域置为空, 清空线性表, 并返回 OK。

时间复杂度为: $O(n)$

空间复杂度为: $O(1)$

1.3.4 status ListEmpty(LinkList L)

设计思路: 该函数接受一个链表 L 作为参数并返回一个状态码。如果链表 L 不存在, 函数返回 INFEASIBLE。如果链表 L 的第一个元素为空, 表明线性表为空, 函数返回 TRUE。否则, 线性表不为空, 函数返回 FALSE。

时间复杂度: $O(1)$

空间复杂度: $O(1)$

1.3.5 int ListLength(LinkList L)

设计思路: 该函数接受一个链表 L 作为参数并返回一个整数。如果链表 L 不存在, 函数返回 INFEASIBLE。否则, 它定义一个变量 number 来记录链表的长度, 并遍历链表, 每遍历到一个节点就将 number 加一。最后返回 number。

时间复杂度为: $O(n)$

空间复杂度为: $O(1)$

1.3.6 status GetElem(LinkList L, int i, ElemType &e)

设计思路: 这段代码是一个获取链表中第 i 个元素的函数。该函数接受一个链表 L、一个整数 i 和一个元素类型的引用 e 作为参数并返回一个状态码。如果链表 L 不存在, 函数返回 INFEASIBLE。否则, 它定义一个变量 number 来记录当前遍历到的节点数, 并定义一个指针 p 指向链表 L。从第一个存储数据的节点开始遍历链表, 每遍历到一个节点就将 number 加一。如果找到第 i 个节点, 将

该节点的数据存储在 e 中并返回 OK。如果遍历完整个链表都未找到第 i 个节点，返回 ERROR。

时间复杂度为 $O(n)$

空间复杂度为 $O(1)$

1.3.7 status LocateElem(LinkList L, ElemType e, int (*vis)(int, int))

设计思路：

- 首先，判断线性表 L 是否存在或已初始化，若不存在则返回错误 (INFEASIBLE)。
- 初始化指针 p 指向链表的第一个数据节点。
- 使用 `number` 变量记录当前位置序号，初始为 0。
- 通过遍历链表的方式查找元素 e ，直到找到与 e 满足关系 `compare()` 的数据元素或遍历完整个链表。
- 通过遍历链表的方式查找元素 e ，直到找到与 e 满足关系 `compare()` 的数据元素或遍历完整个链表。
- 若找到元素 e ，返回其位置序号 `number`；若遍历完整个链表仍未找到元素 e ，返回错误 (ERROR)。

时间复杂度为： $O(n)$

空间复杂度为： $O(1)$

1.3.8 status PriorElem(LinkList L, ElemType e, ElemType &pre)

设计思路：

- 首先，检查线性表 L 是否存在或已初始化，如果不存在则返回错误 (INFEASIBLE)。
- 检查链表 L 是否为空，如果是空链表，则返回错误 (ERROR)。
- 初始化指针 p 指向链表的第一个数据节点。
- 如果要查找的元素 e 是第一个元素，则不存在前驱，返回错误 (ERROR)。
- 从第二个元素开始遍历整个链表，查找元素 e 。
- 如果找到元素 e ，将其前驱保存在 `pre` 变量中，返回成功 (OK)。
- 如果整个链表中都没有找到元素 e ，则返回错误 (ERROR)。

时间复杂度为: $O(1)$

空间复杂度为: $O(1)$

1.3.9 status NextElem(LinkList L,ElemType e,ElemType &next)

设计思路:

- 首先,检查线性表 L 是否存在或已初始化。如果不存在,则返回状态码 INFEASIBLE。
- 检查线性表 L 是否为空。如果是空表,则返回状态码 ERROR。
- 初始化指针 p,指向第一个节点,作为前驱节点。
- 将 L 指向 p 的下一个节点,作为当前节点。
- 进入循环,遍历链表:
 - 如果当前节点 L 为空,表示没有后继节点,返回状态码 ERROR。
 - 如果前驱节点 p 的数据等于给定数据 e,将当前节点 L 的数据赋给 next,并返回状态码 OK。
 - 将前驱节点 p 移动到当前节点 L 的位置,成为新的前驱节点。
 - 将当前节点 L 移动到下一个节点的位置,成为新的当前节点。
- 如果循环结束仍未找到后继节点,则返回状态码 ERROR。

时间复杂度为: $O(n)$

空间复杂度为: $O(1)$

1.3.10 status ListInsert(LinkList &L,int i,int num)

设计思路:

- 首先,检查线性表 L 是否存在或已被初始化。如果不存在,则返回状态码 INFEASIBLE。
- 初始化两个指针 p 和 next,分别指向当前节点和下一个节点。
- 使用变量"number"记录当前位置,并遍历链表以找到插入位置 i。
- 如果找到插入位置 i,则进入循环以插入 num 个元素:
 - 创建一个新节点,并获取用户输入的数据。
 - 修改链表的指针完成插入操作。

- 如果插入位置是最后一个位置，则在链表末尾插入元素。
- 如果插入位置不正确，则返回状态码 ERROR。
- 如果操作成功，返回状态码 OK。

时间复杂度为：O(1)

空间复杂度为：O(num)

1.3.11 status ListDelete(LinkList &L,int i,ElemType &e)

设计思路：

- 首先，检查线性表 L 是否存在或已被初始化。如果不存在，则返回状态码 INFEASIBLE。
- 初始化两个指针 pre 和 next，分别指向当前节点的前一个节点和下一个节点。
- 使用变量“number”记录当前位置，并遍历链表以找到要删除的位置 i。
- 如果找到第 i 个元素：
 - 将该元素的值保存在变量 e 中。
 - 将当前元素的前一个节点 pre 的指针指向当前元素的后一个节点 next，实现删除操作。
 - 释放当前节点的内存空间。
 - 返回执行成功的状态码 OK。
- 如果遍历到表尾仍未找到第 i 个元素，则输出提示信息。
- 返回执行失败的状态码 ERROR。

时间复杂度为：O(n)

空间复杂度为：O(1)

1.3.12 status ListTraverse(LinkList L,void (*vi)(int))

设计思路：

- 首先，检查线性表 L 是否存在或已被初始化。如果不存在，则返回状态码 INFEASIBLE。
- 从线性表 L 的第一个元素开始遍历，使用指针 p 指向当前节点。

- 只要当前节点不是尾节点，就执行以下操作：
 - 对当前节点的元素使用函数指针 vi 进行处理。
 - 将指针 p 移动到下一个节点。
 - 如果当前不是最后一个节点，输出一个空格与下一个元素分隔开。
- 遍历结束后，返回执行成功的状态码 OK。

时间复杂度为： $O(n)$

空间复杂度为： $O(1)$

1.3.13 void reverseList(LinkList L)

设计思路：

- 首先，检查线性表 L 是否存在或已被初始化。如果不存在，则返回状态码 INFEASIBLE。
- 获取线性表的长度。
- 申请一个长度为 len 的数组。
- 遍历链表，将链表中的元素存储到数组中。
- 从尾到头遍历链表，将数组中的元素依次存储到链表中。
- 输出翻转成功的提示信息。

时间复杂度为： $O(n)$

空间复杂度为： $O(n)$

1.3.14 void RemoveNthFromEnd(LinkList L,int n)

设计思路：

- 首先，检查线性表 L 是否存在或已被初始化。如果不存在，则返回状态码 INFEASIBLE。
- 获取单链表的长度 len 。
- 调用 `ListDelete(L, len-n+1, e)` 函数删除第 $(len-n+1)$ 个节点，并将被删除节点的值存入 e 中。
- 如果删除成功，则打印成功信息及被删除节点的值。

时间复杂度为： $O(n)$

空间复杂度为： $O(1)$

1.3.15 void sortList(LinkList L)

设计思路：

- 首先，检查线性表 L 是否存在或已被初始化。如果不存在，则返回状态码 INFEASIBLE。
- 获取单链表的长度 len 。
- 使用动态内存申请了一个数组 arr ，用于存放单链表的数据。
- 遍历单链表，将数据存放到数组 arr 中。
- 对数组 arr 进行冒泡排序，以升序排列数组元素。
- 将排序后的数据写回单链表中。

时间复杂度为: $O(n^2)$

空间复杂度为: $O(n)$

1.3.16 void savetofile(LinkList L,char name[])

设计思路：

- 首先，检查线性表 L 是否存在或已被初始化。如果不存在，则返回状态码 INFEASIBLE。
- 打开文件 $name$ ，以写入的方式。
- 如果无法找到文件，报错并返回。
- 将指针 $current$ 指向链表的第一个节点。
- 循环遍历链表中的每个节点：
 - 将节点的数据写入文件。
 - 将指针 $current$ 移动到下一个节点。
- 关闭文件。
- 提示操作成功。

时间复杂度为: $O(n)$

空间复杂度为: $O(1)$

1.3.17 void getfromfile(LinkList L,char name[])

设计思路：

- 首先，检查线性表 L 的下一个节点是否为空，如果不为空，则表示线性表中已经有数据，无法进行操作。
- 打开文件 $name$ ，以读取的方式。
- 如果无法找到文件，报错并返回。
- 初始化指针 p 用于遍历线性表。
- 动态分配内存，创建一个新的节点 $insert$ ，用于存放从文件中读取的数据。
- 循环读取文件中的数据：
 - 将 $insert$ 节点插入到线性表中。
 - 将指针 p 移动到刚插入的节点。
 - 将 p 的下一个节点设为 $NULL$ 。
 - 动态分配内存，为下一个节点创建新的 $insert$ 节点。
- 关闭文件。

时间复杂度为： $O(n)$

空间复杂度为： $O(n)$

1.4 系统测试

以下主要说明针对各个函数正常和异常的测试用例及测试结果，并以图表的形式展示。

1.4.1 status InitList(LinkList &L)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表未初始化	无	线性表创建成功	系统为线性表分配了连续的存储空间，表刚创立时长度为 0，存储容量为预定义值。输入表长及数据后，表长更改为对应值。
2	线性表已经初始化	无	线性表创建失败	不发生变化

表 1-1 初始线性表

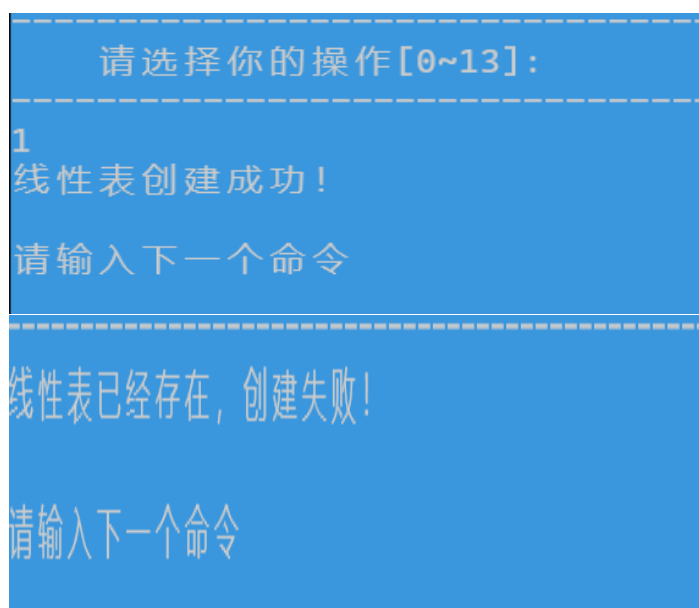


图 1-3 InitList 测试

1.4.2 status DestroyList(LinkList &L)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表未初始化	无	这个线性表不存在或未初始化, 无法销毁	不发生变化 (线性表未初始化)。
2	线性表已经初始化	无	线性表销毁成功	线性表被销毁, 释放了数据元素的空间, 表长度和存储容量都为 0。

表 1-2 初始线性表

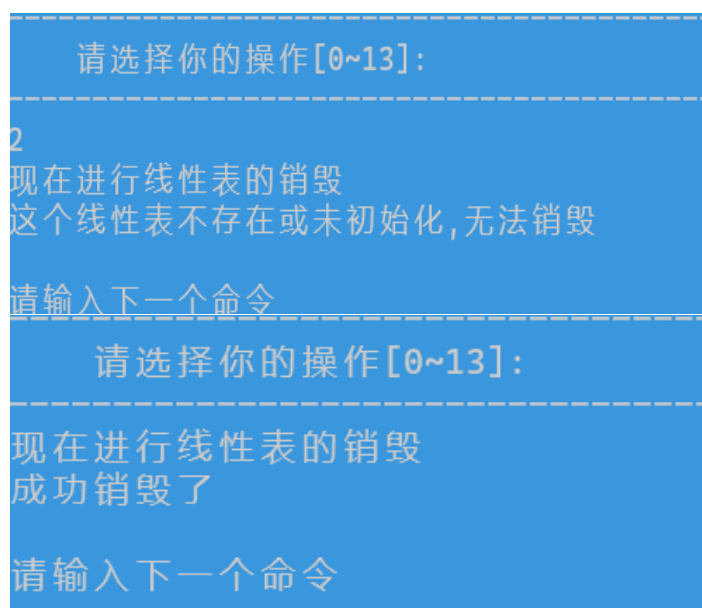


图 1-4 DestroyList 测试

1.4.3 status ClearList(LinkList &L)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表未初始化	无	这个线性表不存在或未初始化, 无法销毁	不发生变化 (线性表未初始化)。
2	线性表已经初始化	无	线性表清空成功	线性表存储空间被重置, length 值与 listsize 值均变为 0。

表 1-3 初始线性表

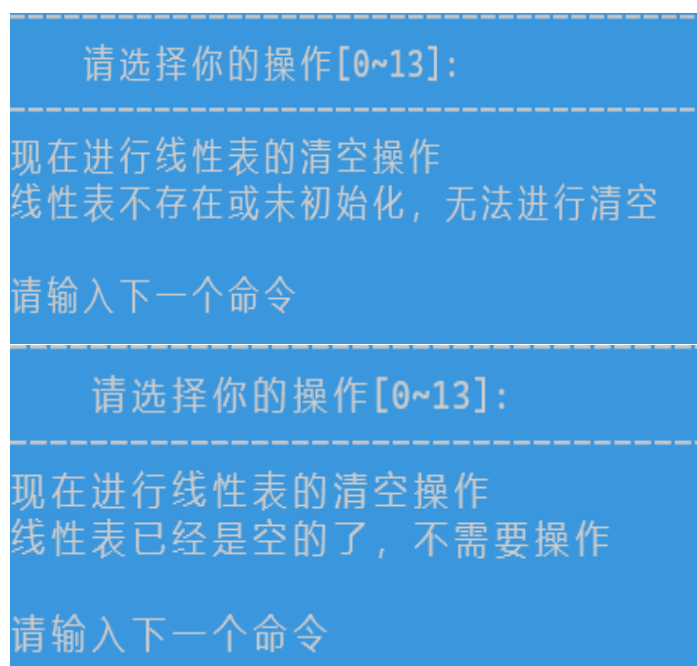


图 1-5 ClearList 测试

1.4.4 status ListEmpty(LinkList L)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表初始化(未赋值)	无	这个线性表是空的	不发生变化（线性表未初始化为空）。
2	线性表已经初始化	无	线性表不是空的	不发生变化（线性表不为空）。

表 1-4 线性表判空

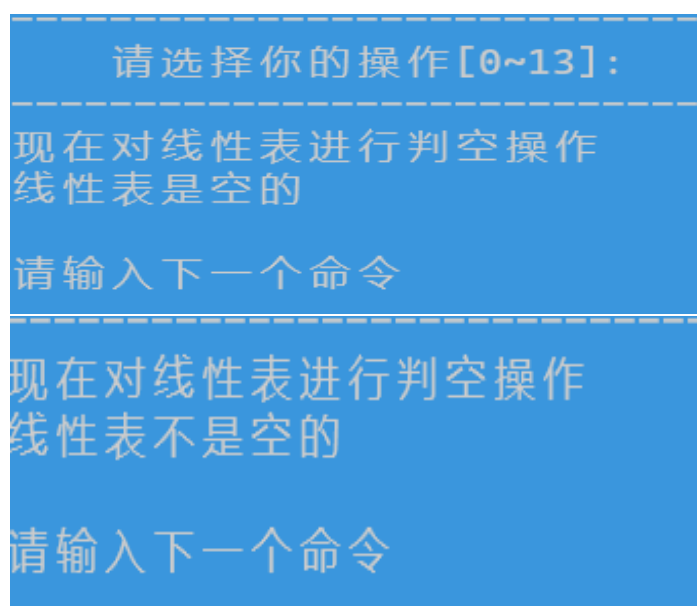


图 1-6 ListEmpty 测试

1.4.5 int ListLength(LinkList L)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表未初始化	无	这个线性表是空的	不发生变化（线性表未初始化为空）。
2	线性表已经初始化(初始化的数据为-1 6 2 -3 5)	无	线性表的长度为5	不发生变化。
3	线性表已经初始化(未赋值)	无	线性表的长度为0	不发生变化

表 1-5 线性表求长度

```

现在进行插入元素的操作
请问你想在第几个位置插入元素
1
请问你想插入元素的个数
5
请输入元素：
-1 6 2 -3 5
插入成功

-----

现在进行求线性表的长度
线性表的长度为:5
请输入下一个命令
    
```

图 1-7 ListLength 测试 1

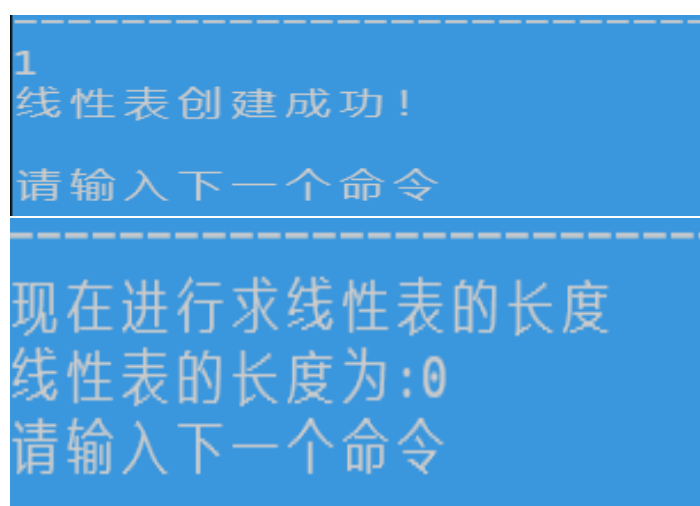


图 1-8 ListLength 测试 2

1.4.6 status GetElem(LinkList L, int i, ElemType &e)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表已经初始化 (初始化的数据为 1 2 3 4)	i = 2	成功获取, 第 2 个元素的值为: 2	不发生变化。
2	线性表已经初始化 (初始化的数据为 1 2 3 4)	i = 3	成功获取, 第 3 个元素的值为: 3	不发生变化。
3	线性表已经初始化 (初始化的数据为 1 2 3 4)	i = 5	i 的值不合法, 无法操作	不发生变化

表 1-6 线性表获取元素

请问你想插入元素的个数

4

请输入元素：

1 2 3 4

插入成功

图 1-9

现在进行元素获取操作

请输入你想获取第几个元素的值

2

成功获取，第2个元素的值为：2

请输入下一个命令

图 1-10

现在进行元素获取操作

请输入你想获取第几个元素的值

3

成功获取，第3个元素的值为：3

请输入下一个命令

图 1-11

现在进行元素获取操作

请输入你想获取第几个元素的值

5

i的值不合法，无法操作

图 1-12

1.4.7 status LocateElem(LinkList L,ElemType e,int (*vis)(int ,int))

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表已经初始化(初始化的数据为 1 2 3 4)	e = 1 和一个 compare 比较函数	成功获取, 第 2 个元素的值为: 2	不发生变化。
2	线性表已经初始化(初始化的数据为 1 2 3 4)	e = 3 和一个 compare 比较函数	成功获取, 第 3 个元素的值为: 3	不发生变化。
3	线性表已经初始化(初始化的数据为 1 2 3 4)	e = 5 和一个 compare 比较函数	没有所要查询的元素	不发生变化

表 1-7 线性表查找元素

现在进行查找元素的操作
 请输入你想查找的元素
 1
 所要查找的元素是第1个

图 1-13

现在进行查找元素的操作
 请输入你想查找的元素
 3
 所要查找的元素是第3个

图 1-14

现在进行查找元素的操作
请输入你想查找的元素
5
没有所要查询的元素

图 1-15

1.4.8 status PriorElem(LinkList L,ElemType e,ElemType &pre)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表已经初始化(初始化的数据为 1 2 3 4)	e = 1	多要查找的元素是第一个元素，没有前驱	不发生变化。
2	线性表已经初始化(初始化的数据为 1 2 3 4)	e = 3	你所要查找的前驱是：2	不发生变化。
3	线性表已经初始化(初始化的数据为 1 2 3 4)	e = 6	没有所要查询的元素不存在线性表里面，无法操作	不发生变化

表 1-8 线性表查找前驱元素

现在进行查找前驱的操作
请输入你想查找哪个元素的前驱
1
所要查找的元素是第一个元素，没有前驱

图 1-16

现在进行查找前驱的操作
请输入你想查找哪个元素的前驱
3
你所要查找的前驱是： 2

图 1-17

现在进行查找前驱的操作
请输入你想查找哪个元素的前驱
6
所要查找的元素不存在线性表里面,无法操作

图 1-18

1.4.9 status NextElem(LinkList L,ElemType e,ElemType &next)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表已经初始化(初始化的数据为 1 2 3 4)	e = 1	你所要查找的元素的后继是： 2	不发生变化。
2	线性表已经初始化(初始化的数据为 1 2 3 4)	e = 4	查询不到后继结点	不发生变化。
3	线性表已经初始化(初始化的数据为 1 2 3 4)	e = 5	查询不到后继结点	不发生变化

表 1-9 线性表查找后继元素

现在进行查找后驱的操作
请输入你想查找哪个元素的后驱
1
你所要查找的元素的后续是： 2

图 1-19

现在进行查找后驱的操作
请输入你想查找哪个元素的后驱
4
查询不到后继结点

图 1-20

现在进行查找后驱的操作
请输入你想查找哪个元素的后驱
5
查询不到后继结点

图 1-21

1.4.10 status ListInsert(LinkList &L,int i,int num)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表已经初始化 (初始化的数据为 6 1 4 7)	i = 1, num = 2 (-1 -2)	插入成功	线性表变为 (-1 -2 6 1 4 7)。
2	线性表已经初始化 (初始化的数据为 -1 -2 6 1 4 7)	i = -1 ,num = 1	插入的位置不对，无法操作	不发生变化。

表 1-10 线性表插入元素

```
-----
现在进行插入元素的操作
请问你想在第几个位置插入元素
1
请问你想插入元素的个数
2
请输入元素：
-1 -2
插入成功
```

图 1-22

```
-----
现在进行插入元素的操作
请问你想在第几个位置插入元素
-1
请问你想插入元素的个数
1
请输入元素：
插入的位置不对
```

图 1-23

1.4.11 status ListDelete(LinkList &L,int i,ElemType &e)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表已经初始化(初始化的数据为-1 -2 6 1 4 7)	i = 2	删除成功，删除的元素是 -2	线性表变为 (-1 6 1 4 7)。
2	线性表已经初始化(初始化的数据为-1 6 1 4 7)	i = -1	想要删除的位置有问题	不发生变化。

表 1-11 线性表删除元素

现在进行删除元素的操作
请问你想删除第几个位置的元素
2
删除成功，删除的元素是： -2

图 1-24

现在进行删除元素的操作
请问你想删除第几个位置的元素
-1
想要删除的位置存在问题

图 1-25

1.4.12 status ListTraverse(LinkList L,void (*vi)(int))

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表未初始化	无	线性表不存在或未初始化，无法进行操作。	线性表不发生变化
2	线性表已经初始化(初始化的数据为 -1 6 1 4 7)	无	成功遍历 -1 6 1 4 7	线性表不发生变化。

表 1-12 线性表遍历元素

现在进行线性表的遍历
-1 6 1 4 7
请输入下一个命令

图 1-26

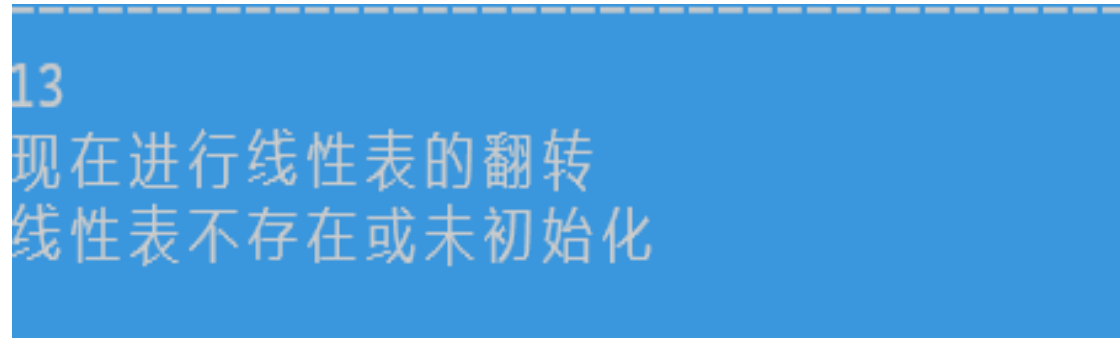
12
现在进行线性表的遍历
线性表不存在或未初始化，无法进行操作

图 1-27

1.4.13 void reverseList(LinkList L)

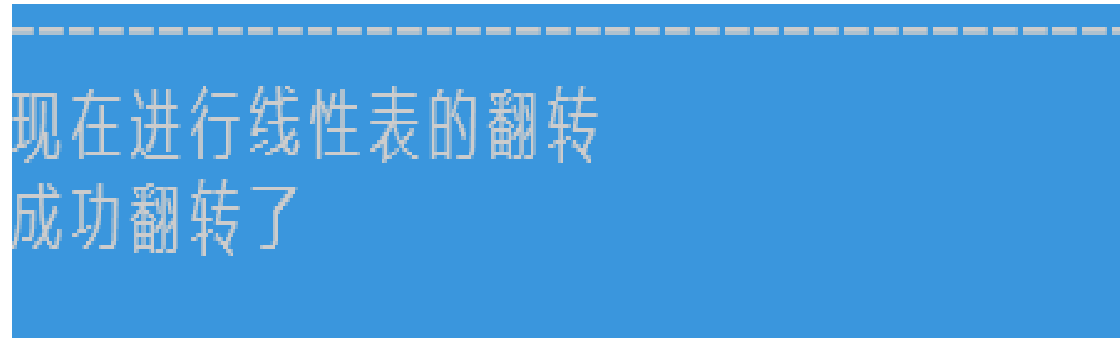
序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表未初始化	无	线性表不存在或未初始化，无法进行操作。	线性表不发生变化
2	线性表已经初始化 (初始化的数据为 9 3 -1 2 6)	无	成功翻转 6 2 -1 3 9	线性表发生翻转。

表 1-13 线性表翻转元素



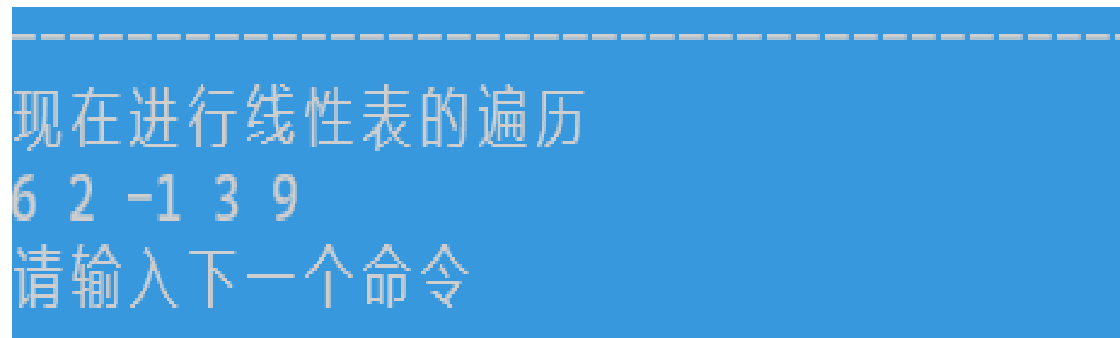
13
现在进行线性表的翻转
线性表不存在或未初始化

图 1-28



现在进行线性表的翻转
成功翻转了

图 1-29



现在进行线性表的遍历
6 2 -1 3 9
请输入下一个命令

图 1-30

1.4.14 void RemoveNthFromEnd(LinkList L,int n)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表未初始化	无	线性表不存在或未初始化，无法进行操作。	线性表不发生变化
2	线性表已经初始化(初始化的数据为 9 3 -1 2 6)	n = 2	成功删除，删除的元素是 2	线性表变为 9 3 -1 6
3	线性表已经初始化(初始化的数据为 9 3 -1 2 6)	n = -1	想要删除的位置有问题	线性表不发生变化

表 1-14 线性表删除倒数结点

```

现在进行删除链表倒数元素的操作
你想删除链表倒数第几个节点
2
成功删除，删除的元素是 :2
    
```

图 1-31

```

现在进行删除链表倒数元素的操作
你想删除链表倒数第几个节点
-1
想要删除的位置存在问题
    
```

图 1-32

1.4.15 void sortList(LinkList L)

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表未初始化	无	线性表不存在或未初始化，无法进行操作。	线性表不发生变化
2	线性表已经初始化(初始化的数据为 9 3 -1 6)	无	成功排序，排序是 -1 3 6 9	线性表变为 -1 3 6 9

表 1-15 线性表排序



图 1-33



图 1-34



图 1-35

1.4.16 void savetofile(LinkList L,char name[])

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表已经初始化(初始化的数据为 9 3 -1 6)	一个文件的 名字 name	成功保存到一个 名字为 name 的 文件	线性表在将数据全部导出到文件之后被销毁, 当从外部文件导入数据时, 线性表又再次被创建并分配空间

表 1-16 线性表的文件保存

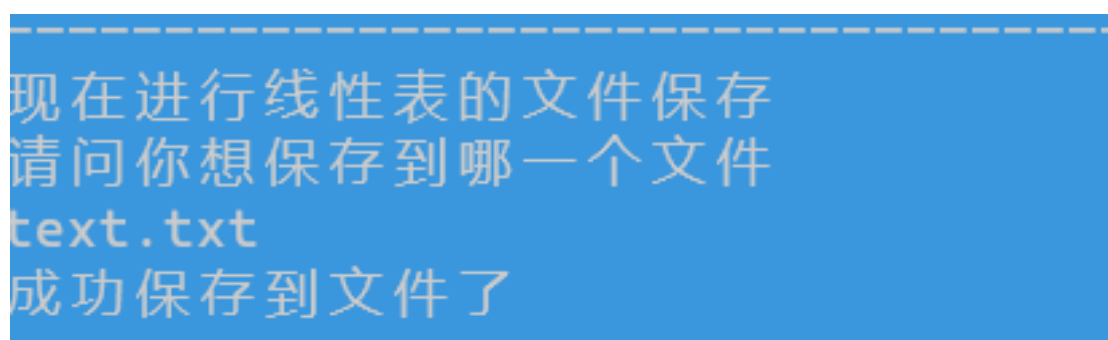


图 1-36

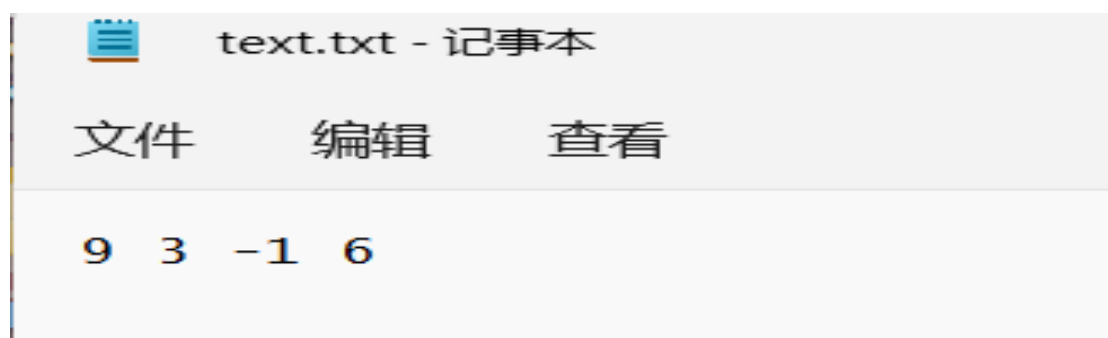


图 1-37

1.4.17 void getfromfile(LinkList L,char name[])

序号	进行此操作前的线性表状态	输入的函数参数	预计输出	操作后线性表的状态
1	线性表未初始化	一个文件的 名字 name	成功读取到一个 名字为 name 的 文件，将内容存 进线性表	线性表里面存储 了文件里面的数 据

表 1-17 线性表的文件读取

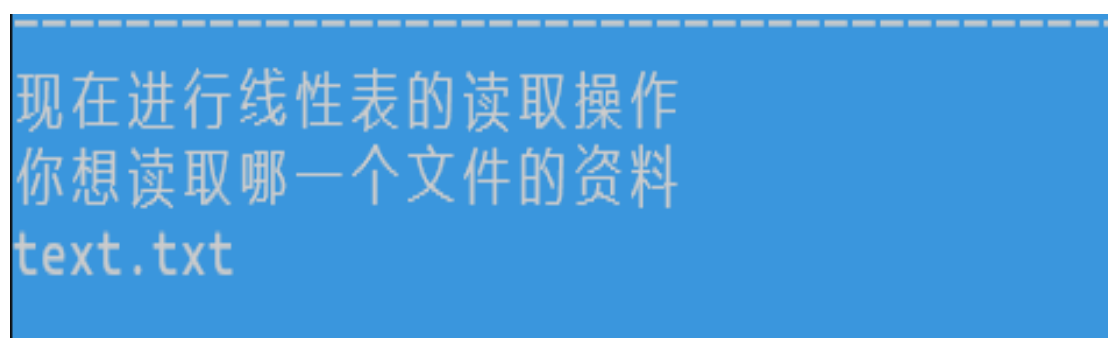


图 1-38

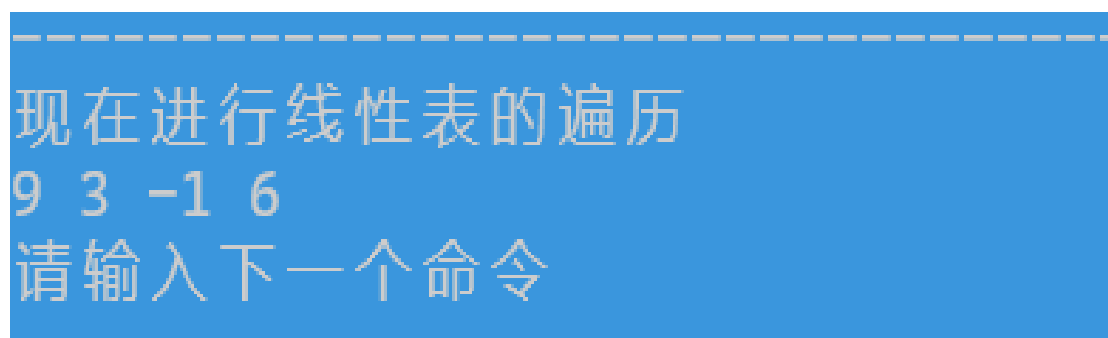


图 1-39

1.5 实验小结

学习完这些线性表的基本运算的定义和实现方法，我获得了以下收获和感想：

- 1) 深入理解线性表的概念和基本运算：通过学习这些基本运算的定义，我对线性表的含义和操作有了更深入的理解。我明白了线性表是由一系列数据

元素组成，并且可以进行初始化、插入、删除、查找等基本操作。

- 2) 理解逻辑结构与物理结构的关系：线性表的逻辑结构是指其抽象的数学模型，而物理结构是指实际存储线性表的方式。通过学习链式存储结构的实现，我了解到如何通过节点之间的指针连接来表示线性表的物理存储结构，将逻辑结构转化为物理结构。
- 3) 熟练掌握线性表的基本运算的实现：通过具体的函数定义和操作结果的说明，我学会了如何实现线性表的初始化、销毁、清空、判定空表、求表长、获得元素、查找元素、获得前驱、获得后继、插入元素、删除元素和遍历表等基本运算。这些操作对于对线性表进行数据处理和操作非常重要。
- 4) 最小完备性和常用性的原则：这些基本运算的定义是基于最小完备性和常用性的原则，意味着它们提供了足够的功能以满足大多数线性表操作的需求。这使得我们能够熟练地使用这些操作，处理和管理线性表的数据。

总的来说，通过进行基于链表的线性表的实验，我清楚的学会了链表的相关知识，并且基于此做了一个简单的演示系统，对链表的实际应用和物理结构有了深刻了解，并且掌握了基础的链表操作技能，学会了基本的系统撰写本领。这些知识将会在我在数据处理领域产生良好反应

2 基于邻接表的图实现

2.1 问题描述

- 1) 构造一个具有菜单功能的演示系统，演示采用邻接表的物理结构构建的图的程序代码，。
- 2) 在主程序中完成函数调用所需参值的准备和函数执行结果的显示。
- 3) 依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等 12 种基本运算。并给出适当的操作提示显示，可选择以文件的形式进行存储和加载，即将生成的图存入到相应的文件中，也可以从文件中获取图进行操作。
- 4) 还有一些附加功能，实现多个图的管理，距离小于 k 的顶点集合，顶点间最短路径和长度，图的连通分量等，以及由多个图切换到单个图的管理。
- 5) 加深对图的概念、基本运算的理解；
- 6) 熟练掌握图的逻辑结构与物理结构的关系；
- 7) 以邻接表作为物理结构，熟练掌握图基本运算的实现。

2.2 系统设计

基于最小完备性和常用性原则，以函数形式定义了创建图、销毁图、查找顶点、顶点赋值等 12 种基本运算。此外还提供了距离小于 k 的顶点集合、顶点间最短路径和长度、图的连通分量以及图的文件形式保存等附加功能。这些函数操作能够对图进行各种操作和查询，实现图的构建、修改、搜索和保存等功能。

2.2.1 头文件和预定义的声明

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include "string.h"
4
5 // 定义布尔类型TRUE和FALSE
6 #define TRUE 1
```

```
7  #define FALSE 0
8
9  // 定义函数返回值类型
10 #define OK 1
11 #define ERROR 0
12 #define INFEASIBLE -1
13 #define OVERFLOW -2
14 #define MAX_VERTEX_NUM 20
15
16 // 定义数据元素类型
17 typedef int ElemType;
18 typedef int status ;
19 typedef int KeyType;
20 typedef enum {DG,DN,UDG,UDN} GraphKind;
```

2.2.2 基于邻接表存储的图的定义

```
1
2 // 定义顶点类型，包含关键字和其他信息
3 typedef struct {
4     KeyType key; // 关键字
5     char others [20]; // 其他信息
6 } VertexType;
7
8 // 定义邻接表结点类型
9 typedef struct ArcNode {
10     int adjvex; // 顶点在顶点数组中的下标
11     struct ArcNode *nextarc; // 指向下一个结点的指针
12 } ArcNode;
13
14 // 定义头结点类型和数组类型（头结点和边表构成一条链表）
```

```
15 typedef struct VNode{
16     VertexType data; // 顶点信息
17     ArcNode *firstarc; // 指向第一条弧的指针
18 } VNode,AdjList[MAX_VERTEX_NUM];
19
20 // 定义邻接表类型, 包含头结点数组、顶点数、弧数和图的类型
21 typedef struct {
22     AdjList vertices; // 头结点数组
23     int vexnum,arcnum; // 顶点数和弧数
24     GraphKind kind; // 图的类型 (有向图、无向图等)
25 } ALGraph;
26
27 // 定义图集合类型, 包含一个结构体数组, 每个结构体包含图的名称和
    邻接表
28 typedef struct {
29     struct {
30         char name[30]="0"; // 图的名称
31         ALGraph G; // 对应的邻接表
32     }elem[30]; // 图的个数
33     int length; // 图集合中图的数量
34 }Graphs;
35
36 Graphs graphs; // 图的集合的定义
```

2.2.3 函数总览

1. 创建图: 函数名称是 CreateCraph(G,V,VR); 初始条件是 V 是图的顶点集, VR 是图的关系集; 操作结果是按 V 和 VR 的定义构造图 G;
2. 销毁图: 函数名称是 DestroyGraph(G); 初始条件图 G 已存在; 操作结果是销毁图 G;
3. 查找顶点: 函数名称是 LocateVex(G,u); 初始条件是图 G 存在, u 是和 G 中

- 顶点关键字类型相同的给定值；操作结果是若 u 在图 G 中存在，返回关键字为 u 的顶点位置序号（简称位序），否则返回其它表示“不存在”的信息；
4. 顶点赋值：函数名称是 $\text{PutVex}(G, u, \text{value})$ ；初始条件是图 G 存在， u 是和 G 中顶点关键字类型相同的给定值；操作结果是对关键字为 u 的顶点赋值 value ；
 5. 获得第一邻接点：函数名称是 $\text{FirstAdjVex}(G, u)$ ；初始条件是图 G 存在， u 是 G 中顶点的位序；操作结果是返回 u 对应顶点的第一个邻接顶点位序，如果 u 的顶点没有邻接顶点，否则返回其它表示“不存在”的信息；
 6. 获得下一邻接点：函数名称是 $\text{NextAdjVex}(G, v, w)$ ；初始条件是图 G 存在， v 和 w 是 G 中两个顶点的位序， v 对应 G 的一个顶点， w 对应 v 的邻接顶点；操作结果是返回 v 的（相对于 w ）下一个邻接顶点的位序，如果 w 是最后一个邻接顶点，返回其它表示“不存在”的信息；
 7. 插入顶点：函数名称是 $\text{InsertVex}(G, v)$ ；初始条件是图 G 存在， v 和 G 中的顶点具有相同特征；操作结果是在图 G 中增加新顶点 v 。（在这里也保持顶点关键字的唯一性）；
 8. 删除顶点：函数名称是 $\text{DeleteVex}(G, v)$ ；初始条件是图 G 存在， v 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中删除关键字 v 对应的顶点以及相关的弧；
 9. 插入弧：函数名称是 $\text{InsertArc}(G, v, w)$ ；初始条件是图 G 存在， v 、 w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中增加弧 $\langle v, w \rangle$ ，如果图 G 是无向图，还需要增加 $\langle w, v \rangle$ ；
 10. 删除弧：函数名称是 $\text{DeleteArc}(G, v, w)$ ；初始条件是图 G 存在， v 、 w 是和 G 中顶点关键字类型相同的给定值；操作结果是在图 G 中删除弧 $\langle v, w \rangle$ ，如果图 G 是无向图，还需要删除 $\langle w, v \rangle$ ；
 11. 深度优先搜索遍历：函数名称是 $\text{DFS Traverse}(G, \text{visit}())$ ；初始条件是图 G 存在；操作结果是图 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；
 12. 广度优先搜索遍历：函数名称是 $\text{BFS Traverse}(G, \text{visit}())$ ；初始条件是图 G 存在；操作结果是图 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。

附加功能

1. 距离小于 k 的顶点集合：函数名称是 `VerticesSetLessThanK(G,v,k)`，初始条件是图 G 存在；操作结果是返回与顶点 v 距离小于 k 的顶点集合；
2. 顶点间最短路径和长度：函数名称是 `ShortestPathLength(G,v,w)`；初始条件是图 G 存在；操作结果是返回顶点 v 与顶点 w 的最短路径的长度；
3. 图的连通分量：函数名称是 `ConnectedComponentsNums(G)`，初始条件是图 G 存在；操作结果是返回图 G 的所有连通分量的个数；
4. 实现图的文件形式保存：其中，□ 需要设计文件数据记录格式，以高效保存图的数据逻辑结构 (D,R) 的完整信息；□ 需要设计图文件保存和加载操作合理模式。附录 B 提供了文件存取的方法；
5. 实现多个图管理：设计相应的数据结构管理多个图的查找、添加、移除等功能。

```
1
2  status  isrepeat (VertexType V[]); //判断是否有重复结点
3  status  CreateCraph(ALGraph &G,VertexType V[],KeyType VR[][2]); //创
   建
4  status  DestroyGraph(ALGraph &G); //销毁
5  status  LocateVex(ALGraph G, KeyType u); //查找
6  status  PutVex(ALGraph &G, KeyType u, VertexType value); //顶点赋值
7  status  FirstAdjVex (ALGraph G, KeyType u); //获得第一邻接点
8  status  NextAdjVex(ALGraph G, KeyType v, KeyType w); //获得下一邻接
   点
9  status  InsertVex (ALGraph &G,VertexType v); //插入顶点
10 status  DeleteVex(ALGraph &G, KeyType v); //删除顶点
11 status  InsertArc (ALGraph &G,KeyType v,KeyType w); //插入弧
12 status  DeleteArc(ALGraph &G,KeyType v,KeyType w); //删除弧
13 void dfs (ALGraph G , void (* visit )(VertexType), int nownode);
14 status  DFSTraverse(ALGraph G,void (*visit)(VertexType)); // dfs遍历
15 void BFS(ALGraph G,void (* visit )(VertexType), int i);
16 status  BFSTraverse(ALGraph G,void (*visit)(VertexType)); // bfs遍历
17 void visit (VertexType p); //遍历的时候调用的输出函数
18 int * VerticesSetLessThanK(ALGraph G,int v, int k); // 顶点小于k的顶
```

点集合

```

19 int ShortestPathLength (ALGraph G,int v,int w); // 顶点间的最短路径
20 int ConnectedComponentsNums(ALGraph G); // 图的分量
21 status SaveGraph(ALGraph G, char FileName[]); // 图的文件保存
22 status LoadGraph(ALGraph &G, char FileName[]); // 图的文件读取
23 void menu(); // 多个图管理的菜单
24 void menu2(); // 单个图管理的菜单
25 void fun01(); // 多个图管理的封装函数
26 void fun02(ALGraph &G); // 单个图管理的封装函数
    
```

2.2.4 菜单实现

菜单采用简单移动的 UI，且增加了动漫特效，增加趣味。

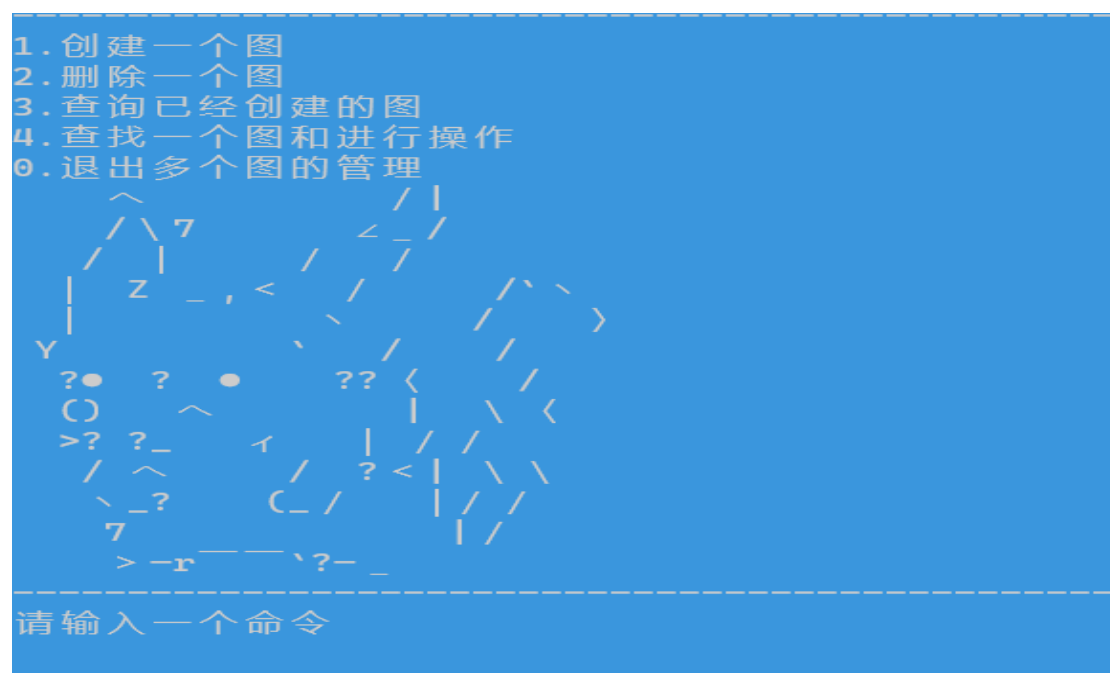


图 2-1 多个图管理菜单

- 5) 遍历关系数组，创建边。检查是否存在自环和重复边，如果存在则返回错误代码。同时检查边连接的节点是否已经出现过，如果没有出现过则返回错误代码。
- 6) 使用头插法插入边，构建邻接链表。
- 7) 再次遍历关系数组，创建另一条方向的边，并使用头插法插入到邻接链表中。
- 8) 返回成功状态码。

时间复杂度: $O(n+m)$

空间复杂度: $O(n)$

2.3.2 status DestroyGraph(ALGraph &G)

设计思路：该函数接受一个无向图 G 作为参数，并返回一个状态码。如果图 G 不存在（即顶点数量为 0），函数返回 INFEASIBLE。否则，函数循环遍历图 G 的每个顶点，对每个顶点循环遍历其邻接点，并删除对应的边。最后，将图 G 的顶点数量和边数量重置为 0，并返回 OK。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

2.3.3 int LocateVex(ALGraph G, KeyType u)

设计思路：该函数接受一个无向图 G 和一个关键字 u 作为参数，并返回一个整数值。如果图 G 不存在（即顶点数量为 0），函数打印错误信息并返回 INFEASIBLE。否则，函数循环遍历图 G 的每个顶点，查找关键字值为 u 的顶点。如果找到，返回其位序（即顶点在数组中的索引值），否则返回 -1。

时间复杂度: $O(V)$

空间复杂度: $O(1)$

2.3.4 status PutVex(ALGraph &G, KeyType u, VertexType value)

设计思路：该函数接受一个无向图 G 、一个关键字 u 和一个顶点值 $value$ 作为参数，并返回一个状态码。如果图 G 不存在（即顶点数量为 0），函数打印错误信息并返回 INFEASIBLE。函数通过循环遍历图 G 的每个顶点，查找关键字

值为 u 的顶点。如果关键字不唯一，即在图中存在多个值为 $value$ 的顶点且关键字不等于 u ，则函数打印错误信息并返回 **ERROR**。如果找到了符合条件的顶点，且只出现一次，则将其值修改为指定的 $value$ 。最后返回 **OK** 表示操作成功。

时间复杂度： $O(n)$

空间复杂度： $O(1)$

2.3.5 `int FirstAdjVex(ALGraph G, KeyType u)`

设计思路是：在图 G 中寻找给定关键字对应的顶点，如果找到了顶点，则返回该顶点对应的第一邻接顶点的位序。如果未找到给定关键字对应的顶点，返回信息“不存在”，即 -1 。

时间复杂度： $O(n)$

空间复杂度： $O(1)$

2.3.6 `int NextAdjVex(ALGraph G, KeyType v, KeyType w)`

设计思路是：在图 G 中寻找给定关键字对应的顶点，如果找到了顶点，则遍历该顶点的邻接链表，找到目标节点 w ，如果 w 不是最后一个邻接顶点，则返回其下一个邻接顶点的位序，否则返回“不存在”的信息。

时间复杂度： $O(n)$

空间复杂度： $O(1)$

2.3.7 `status InsertVex(ALGraph & G, VertexType v)`

设计思路是：在图 G 中插入一个新的顶点 v ，如果图不存在，则返回“不存在”的信息；如果图中顶点数量已达到最大限制，则返回 **ERROR**；查找图中是否已有 **KEY** 相同的结点，如果有，则返回 **ERROR**；在 $G.vertices$ 数组的最后一个位置插入新结点，更新 $G.vexnum$ 。

时间复杂度： $O(n)$

空间复杂度： $O(1)$

2.3.8 `status DeleteVex(ALGraph &G, KeyType v)`

设计思路：删除图 G 中关键字 v 对应的顶点以及相关的弧。首先，判断图 G 是否存在或已初始化。如果图中只有一个顶点，则无法删除，返回错误状态。

接着，寻找要删除的顶点。如果要删除的顶点不存在，则返回错误状态。如果要删除的顶点存在，则删除与这个顶点有关的弧。然后，将删除顶点之后的顶点位置全部向前移动一个位置，覆盖掉要删除的位置。最后，进行与这个顶点有关的弧的删除操作，以及将所有大于要删除位置的顶点位置减一。

时间复杂度为: $O(n^2)$

空间复杂度: $O(1)$

2.3.9 status InsertArc(ALGraph &G,KeyType v,KeyType w)

设计思路：在图 G 中增加弧 $\langle v,w \rangle$ 。首先，判断图 G 是否存在或已初始化。如果图不存在，则返回错误状态。如果插入的是重边，则返回错误状态。接着，寻找要插入的顶点。如果要插入的顶点不存在，则返回错误状态。如果要插入的顶点存在，则检查插入的是否为重复的边。如果是重复的边，则返回错误状态。分别创建结构体 `newv` 和 `neww`，构建新边。然后，将新边指向 v 的第一条边和 w 的第一条边，更新头指针，即 v 的第一条边为新边和 w 的第一条边为新边。最后，边数加 1，插入成功。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

2.3.10 status DeleteArc(ALGraph &G,KeyType v,KeyType w)

设计思路：删除图 G 中的弧 $\langle v,w \rangle$ 。函数首先检查图是否存在以及 v 和 w 是否相等。如果这两个条件中有任何一个为真，则函数返回错误。然后，函数在图中搜索顶点 v 和 w ，并检查它们之间是否存在弧。如果不存在弧，则函数返回错误。如果存在弧，则函数通过更新顶点 v 的邻接表来删除它。

时间复杂度: $O(n)$

空间复杂度: $O(1)$

2.3.11 status DFSTraverse(ALGraph G,void (*visit)(VertexType))

设计思路：

- 1) 使用一个标记数组 `flag11` 来记录每个节点是否被遍历过。
- 2) 首先，对图中的每个顶点进行标记初始化。

- 3) 对于每个未被遍历过的顶点，调用深度优先搜索函数 `dfs` 进行遍历。
- 4) 在 `dfs` 函数中，首先访问当前节点，然后将其标记为已遍历过。接着，遍历当前节点的所有邻接节点，如果邻接节点没有被遍历过，则递归调用 `dfs` 函数进行遍历。
- 5) 在遍历过程中，通过传入的 `visit` 函数对节点进行访问操作。

时间复杂度： $O(n)$

空间复杂度： $O(1)$

2.3.12 status BFSTraverse(ALGraph G,void (*visit)(VertexType))

设计思路：

- 1) 使用一个标记数组 `flag12` 来记录每个节点是否被遍历过。
- 2) 首先，对图中的每个顶点进行标记初始化。
- 3) 对于每个未被遍历过的顶点，调用 `BFS` 函数进行广度优先搜索遍历。
- 4) 在 `BFS` 函数中，使用一个队列 `Que` 来存放待遍历的顶点。初始时，将起始顶点 `i` 放入队列中。
- 5) 在每一次循环中，从队列的头部取出一个顶点，访问它，并将其邻接节点（未被访问过的）加入队列中。同时更新相应的标记数组 `flag12`。
- 6) 循环直到队列为空，即所有顶点都被访问完毕。

时间复杂度： $O(n)$

空间复杂度： $O(1)$

2.3.13 int * VerticesSetLessThanK(ALGraph G,int v,int k)

设计思路：

- 1) 首先，将给定的距离上限 `k` 减 1，因为是从起始点算起的距离。
- 2) 如果图 `G` 不存在，即顶点数为 0，则返回 `NULL`。
- 3) 创建一个记录数组 `record`，用于标记已访问过的节点，初始化为 0。
- 4) 遍历图的顶点，找到起始顶点 `v`，并记录其位置到 `flag` 变量中。
- 5) 如果未找到起始顶点，即 `flag` 仍为 -1，则返回 `NULL`。
- 6) 创建一个静态数组 `srr`，用于存储距离小于 `k` 的顶点集合。
- 7) 将起始顶点 `v` 加入顶点集合中，并将其标记为已访问过。

- 8) 创建一个二维数组 `Que` 作为队列, 用于存储待访问的节点及其距离。
- 9) 将起始顶点 `v` 作为队列的第一个元素, 距离为 0, 在队列非空且队列中第一个节点的距离不超过 `k` 的情况下, 进行队列的遍历, 遍历队头节点的邻接链表, 如果邻接节点未被访问过且距离不超过 `k-1`, 则将其加入顶点集合, 并加入队列中。
- 10) 更新队列的头尾指针和邻接节点的距离。
- 11) 直到队列为空或队列中第一个节点的距离超过 `k+1`。
- 12) 将顶点集合数组以 -1 结尾, 以便在函数外部访问到数组长度, 返回存储顶点集合的数组指针。

时间复杂度: $O(n)$

空间复杂度: $O(n)$

2.3.14 `int ShortestPathLength(ALGraph G, int v, int w)`

设计思路:

- 1) 首先, 检查图 `G` 是否存在, 若不存在则返回错误。
- 2) 创建一个记录数组 `record`, 用于标记每个节点是否被访问过, 初始化为 0, 创建一个二维数组 `arr`, 用作队列, 每个元素包含节点和距离的信息。
- 3) 初始化队列, 将队列头和尾指针 `head` 和 `tail` 设为 0, 遍历图的顶点, 找到起始顶点 `v` 和目标顶点 `w`, 记录它们的索引值到 `flag` 和 `flagw` 变量中, 如果未找到 `v` 或 `w` 节点, 即 `flag` 或 `flagw` 为 -1, 则返回错误。
- 4) 将起始顶点 `v` 加入队列, 并进入循环, 从队列中取出当前节点, 遍历其邻接边, 如果找到目标顶点 `w`, 返回距离, 如果邻接节点未被访问过, 将其加入队列, 并更新距离, 标记当前节点已被访问, 处理下一个节点, 继续循环直到队列为空。
- 5) 如果没有找到路径, 返回 -1。

时间复杂度: $O(n)$

空间复杂度: $O(n)$

2.3.15 `int ConnectedComponentsNums(ALGraph G)`

设计思路:

- 1) 定义一个全局数组 `flag16`，用于标记顶点是否被访问过。
- 2) 定义深度优先搜索函数 `dfs`，递归地遍历当前节点的邻接节点，并标记为已访问。
- 3) 定义连通分量计数函数 `ConnectedComponentsNums`，初始化计数器 `count` 为 0。
- 4) 遍历所有顶点，如果当前顶点未被访问，则调用 `dfs` 函数进行深度优先搜索，并将计数器 `count` 加 1。
- 5) 返回连通分量的计数器 `count`。

时间复杂度： $O(n)$

空间复杂度： $O(1)$

2.3.16 status SaveGraph(ALGraph G, char FileName[])

设计思路：

- 1) 检查图是否为空，如果为空则直接返回错误。
- 2) 打开指定文件，以只写模式打开。如果无法打开文件，返回错误。
- 3) 写入顶点数和边数到文件，遍历每个顶点，写入顶点的 `key` 和 `others` 到文件。
- 4) 遍历每个结点，从顶点的第一条边开始遍历，写入边的邻接点编号到文件，在每条边结束后写入-1。
- 5) 关闭文件，返回成功。

时间复杂度： $O(n)$

空间复杂度： $O(1)$

2.3.17 status LoadGraph(ALGraph &G, char FileName[])

- 1) 检查图是否为空，如果图不为空则无法读取，直接返回错误。
- 2) 打开指定文件，以只读模式打开。如果无法打开文件，返回错误。
- 3) 从文件中读取顶点数和边数。
- 4) 遍历每个顶点，从文件中读取顶点的 `key` 和 `others`，并将顶点的第一条边设为 `NULL`。遍历每个结点，从顶点的第一条边开始遍历，创建一个新的边结点，并从文件中读取邻接点编号。如果读取的邻接点编号不是-1，则将新结点添加到当前顶点的边链表中。

5) 关闭文件，返回成功。

时间复杂度: $O(n)$

空间复杂度: $O(n)$

2.4 系统测试

主要说明针对各个函数正常和异常的测试, 并结合表格和图片进行演示。

2.4.1 status CreateCraph(ALGraph &G, VertexType V[], KeyType VR[][2])

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图未初始化	一个新的图 G, 装有结点信息的一维数组 V 和装有边信息的 VR 二维数组	图初始化成功	G 中建立以邻接表为存储方式的图
2	图已经初始化	无	该图已经初始化, 不能再次初始化	不发生变化

表 2-1 初始化图

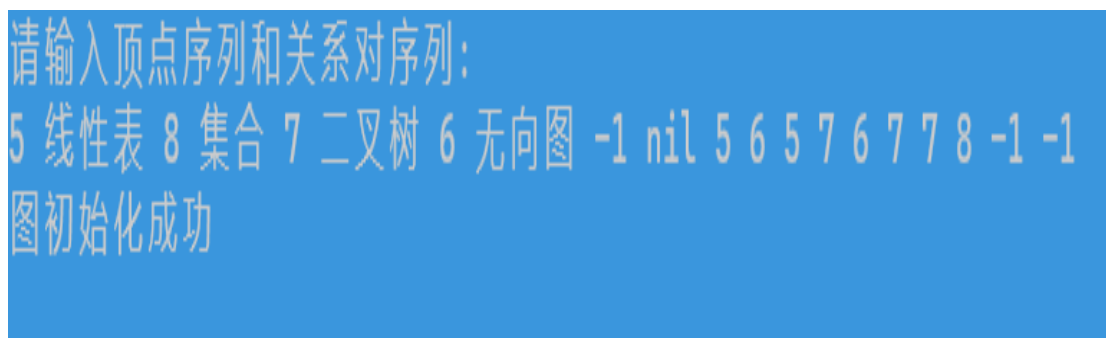


图 2-3 序号 1 中正常初始化

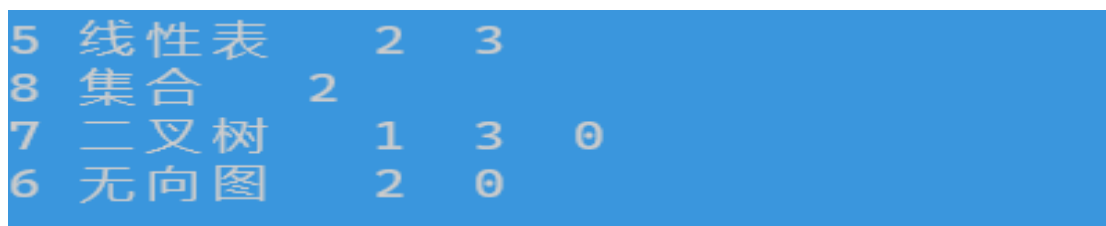


图 2-4 序号 1 中初始化后查看图中的关系图

请输入顶点序列和关系对序列:
5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1
该图已经初始化, 不能再次初始化

图 2-5 序号 2 中初始化失败

2.4.2 status DestroyGraph(ALGraph &G)

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图未初始化	无	图销毁失败	不发生变化
2	图已经初始化	无	该图销毁成功了	图中数据和空间被销毁, 变成一个未初始化的空图

表 2-2 销毁图

现在进行图的销毁操作
线性表未初始化或者不存在

图 2-6 序号 1 中销毁失败

现在进行图的销毁操作
图销毁成功了

图 2-7 序号 2 中销毁成功

2.4.3 int LocateVex(ALGraph G, KeyType u)

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图未初始化	无	图未初始化，查找失败	不发生变化
2	图已经初始化	u = 5 (查找关键字为 5 的结点)	所要查找的关键字为 5 的顶点的位置序号为 0，具体信息为 5	不发生变化
3	图已经初始化	u = 2 (查找关键字为 2 的结点)	所要查找的顶点不存在	不发生变化

表 2-3 图中查找顶点

现在进行查找顶点的操作
 请输入你想查找的顶点的关键字
 3
 该图不存在或未初始化

图 2-8 图未初始化

现在进行查找顶点的操作
 请输入你想查找的顶点的关键字
 5
 所要查找的关键字为 5 的顶点的位置序号为 0
 具体信息为 5 线性表

图 2-9 图初始化，查找顶点关键字为 5

现在进行查找顶点的操作
 请输入你想查找的顶点的关键字
 2
 所要查找的顶点不存在

图 2-10 图初始化，查找顶点关键字为 2

2.4.4 status PutVex(ALGraph &G, KeyType u, VertexType value)

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图未初始化	无	图未初始化，操作失败	不发生变化
2	图已经初始化 (图里存在关键字为 5 的顶点)	u = 5 (对关键字为 5 的结点进行操作), value(11 x)	操作成功	结点关键字为 5 的结点更改为 11 x
3	图已经初始化 (图里没有关键字为 5 的顶点)	u = 5 (对关键字为 5 的结点进行操作), value(2 链表)	查找失败，无法操作	不发生变化
4	图已经初始化 (图里存在关键字为 11 和 8 的顶点)	u = 11 (对关键字为 5 的结点进行操作), value(8 集合)	关键字不唯一，操作失败	不发生变化

表 2-4 图中顶点赋值

```
现在进行顶点赋值的操作
请输入你想对哪一个关键字进行操作
5
请输入你想改变的关键字和名称
11 x
该图不存在或未初始化
```

图 2-11

```
现在进行顶点赋值的操作
请输入你想对哪一个关键字进行操作
5
请输入你想改变的关键字和名称
11 x
操作成功
```

图 2-12

```
现在进行顶点赋值的操作
请输入你想对哪一个关键字进行操作
5
请输入你想改变的关键字和名称
2 链表
查找失败,无法操作
```

图 2-13

```
现在进行顶点赋值的操作
请输入你想对哪一个关键字进行操作
11
请输入你想改变的关键字和名称
8 集合
关键字不唯一,操作失败
```

图 2-14

2.4.5 int FirstAdjVex(ALGraph G, KeyType u)

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图未初始化	无	图未初始化，操作失败	不发生变化
2	图已经初始化(图里存在关键字为5的顶点)	u = 5 (查找关键字为5的结点的下一邻接点)	所要查找的关键字为5的顶点的位置序号为0，具体信息为5	不发生变化
3	图已经初始化(图里没有关键字为9的顶点)	u = 9 (查找关键字为9的结点的下一邻接点)	所要查找的顶点不存在	不发生变化

表 2-5 图中顶点赋值

现在进行获取第一邻接点的操作
 输入你想操作的关键字
 1
 该图不存在或未初始化
 操作失败

图 2-15 图未初始化

现在进行获取第一邻接点的操作
 输入你想操作的关键字
 5
 获取成功,第一邻接点的位序是2,具体信息为7 二叉树
 请输入下一个命令

图 2-16

现在进行获取第一邻接点的操作
输入你想操作的关键字
9
操作失败

图 2-17

2.4.6 int NextAdjVex(ALGraph G, KeyType v, KeyType w)

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图已经初始化, 见2-27	$v = 8, w = 7$	操作失败	不发生变化
2	图已经初始化, 见2-27	$v = 7, w = 8$	获取成功, 下一邻接点的位序是 3, 具体信息为 6 无向图	不发生变化

表 2-6 图中获取下一邻接点

5 线性表 2 3
8 集合 2
7 二叉树 1 3 0
6 无向图 2 0

图 2-18 图中初始化的关系图

现在进行获取下一邻接点的操作
请输入G中两个顶点的位序, v对应G的一个顶点, w对应v的邻接顶点
8 7
操作失败

图 2-19

```

现在进行获取下一邻接点的操作
请输入G中两个顶点的位序, v对应G的一个顶点, w对应v的邻接顶点
7 8
获取成功, 下一邻接点的位序是3, 具体信息为6 无向图
请输入下一个命令
    
```

图 2-20

2.4.7 status InsertVex(ALGraph & G, VertexType v)

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图未初始化	v (11 有向图)	图未初始化, 操作失败	不发生变化
2	图已经初始化, 见2-27	v (11 有向图)	插入成功	顶点集中插入 (11 有向图), 见2-23

表 2-7 图中插入顶点

```

5 线性表      2  3
8 集合        2
7 二叉树      1  3  0
6 无向图      2  0
    
```

图 2-21 图中初始化的关系图

```

现在进行插入顶点的操作
输入你想插入的顶点的关键字和名称
11 有向图
插入成功
    
```

图 2-22

```

5 线性表      2  3
8 集合        2
7 二叉树      1  3  0
6 无向图      2  0
11 有向图
    
```

图 2-23 插入顶点后的图关系

2.4.8 status DeleteVex(ALGraph &G, KeyType v)

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图未初始化	v = 6	图未初始化，操作失败	不发生变化
2	图已经初始化，见2-27	v = 6	操作成功	顶点集中删除关键字为 6 的顶点，以及与其连接的边，见2-26

表 2-8 图中删除顶点

```

5 线性表      2  3
8 集合        2
7 二叉树      1  3  0
6 无向图      2  0
    
```

图 2-24 图中初始化的关系图

```

现在进行删除顶点的操作
请输入你想删除的顶点的关键字
6
操作成功
    
```

图 2-25

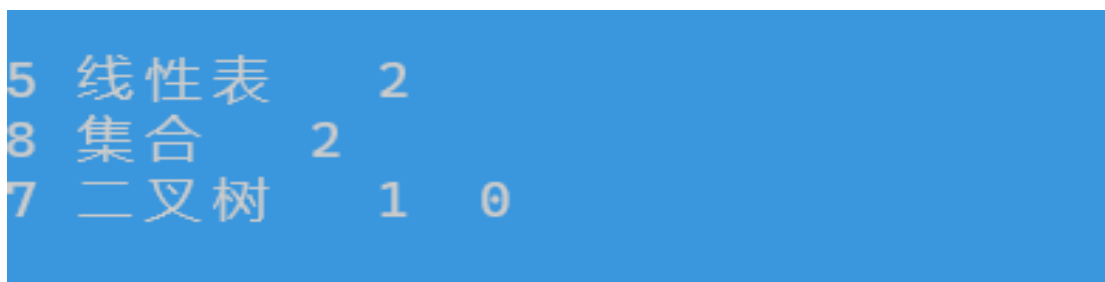


图 2-26

2.4.9 status InsertArc(ALGraph &G,KeyType v,KeyType w)

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图未初始化	$v = 1, w = 5$	图未初始化，操作失败	不发生变化
2	图已经初始化，见2-27	$v = 1, w = 5$	找不到要插入的顶点，操作失败	不发生变化
3	图已经初始化，见2-27	$v = 5, w = 8$	操作成功	插入边 5 8，见2-30
4	图已经初始化，见2-27	$v = 1, w = 5$	找不到要插入的顶点，操作失败	不发生变化
5	图已经初始化，见2-27	$v = 5, w = 6$	操作失败	不发生变化

表 2-9 图中插入弧

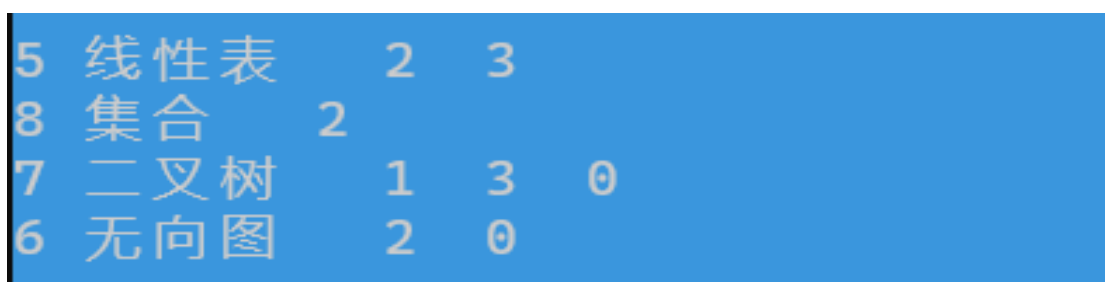


图 2-27 图中初始化的关系图

现在进行插入弧的操作
输入你想插入的弧
1 5
找不到要插入的顶点
操作失败

图 2-28

现在进行插入弧的操作
输入你想插入的弧
5 8
操作成功

图 2-29

5	线性表	1	2	3
8	集合	0	2	
7	二叉树	1	3	0
6	无向图	2	0	

图 2-30

现在进行插入弧的操作
输入你想插入的弧
1 5
找不到要插入的顶点
操作失败

图 2-31

现在进行插入弧的操作
输入你想插入的弧
5 6
操作失败

图 2-32

2.4.10 status DeleteArc(ALGraph &G,KeyType v,KeyType w)

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图未初始化	$v = 1, w = 5$	图未初始化，操作失败	不发生变化
2	图已经初始化，见2-27	$v = 5, w = 6$	操作成功	删除了边 5 和 6，见2-34

表 2-10 图中删除弧

现在进行删除弧的操作
输入你想删除的弧
5 6
操作成功

图 2-33

5 线性表 1 2
8 集合 0 2
7 二叉树 1 3 0
6 无向图 2

图 2-34

2.4.11 status DFSTraverse(ALGraph G,void (*visit)(VertexType))

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图未初始化	一个遍历的函数 visit	图未初始化，操作失败	不发生变化
2	图已经初始化，见2-27	一个遍历的函数 visit	操作成功	不发生变化

表 2-11 图中进行深度优先搜索



图 2-35

2.4.12 status BFSTraverse(ALGraph G,void (*visit)(VertexType))

序号	进行此操作前的图状态	输入的函数参数	预计输出	操作后图的状态
1	图未初始化	一个遍历的函数 visit	图未初始化，操作失败	不发生变化
2	图已经初始化，见2-27	一个遍历的函数 visit	操作成功	不发生变化

表 2-12 图中进行广度优先搜索



图 2-36

2.5 实验小结

学习完这些线性表的基本运算的定义和实现方法，我获得了以下收获和感想：

- 经过学习上述内容，我对图结构有了更深入的理解。通过对 12 种基本运算的学习，我明确了如何创建和销毁图、查找和插入顶点以及插入和删除边等基本操作。这些运算对于熟悉图的数据结构和操作非常有帮助。此外，还讲解了一些附加功能，如获得距离小于 k 的顶点集合、顶点间的最短路径和长度等，这些功能在实际操作中非常实用。
- 同时，也学习了如何实现图的文件保存和加载，这对于实际应用中的数据持久化非常重要。通过设计合适的数据记录格式，可以有效地保存图的逻辑结构，便于后续使用。此外，多图管理功能也让我了解到如何在一个程序中处理多个图结构，实现添加、移除和查找等功能。
- 总之，通过本次学习，我对图结构有了更全面的认识，学会了如何操作和管理图。在实际应用中，这些知识对于解决一些复杂问题具有很大帮助，比如设计地图导航系统、社交网络等。同时，在学习过程中，我也锻炼了自己的逻辑思维能力和编程水平。未来我会继续深入学习更多关于图的知识，以提升自己在这方面的专业能力。

3 课程的收获和建议

3.1 基于顺序存储结构的线性表实现

线性表是一种简单而重要的数据结构，它在计算机科学与程序设计中有广泛应用。了解线性表的概念及其基本运算有助于我们更好地处理和分析有关问题。

线性表有多种实现方式，顺序表是其中一种常见的实现方式，通过实际操作熟练掌握顺序表基本运算的实现，有助于理解和应用线性表这一数据结构。

通过实验，加深了对线性表的逻辑结构与物理结构之间关系的理解。这对我们今后学习其他数据结构，比如链表、树等，也是很有帮助的。

在实际编写代码过程中，关注边界条件的处理，例如索引越界、线性表空间不足等问题，要养成良好的编程习惯和思维。

将各种基本操作封装成函数，形成一个功能演示系统，有助于我们更好地组织代码，提高代码的可读性和复用性。

通过实现附加功能，进一步拓展了线性表的应用场景，让我们对线性表的应用有了更直观的了解。

总之，通过上述实验熟悉线性表及其基本操作，提高了自己的实际操作能力，为今后学习其他数据结构和算法打下了坚实的基础。同时，也培养了自己独立思考、分析问题的能力和良好的编程习惯。

3.2 基于链式存储结构的线性表实现

线性表是一种常见的数据结构，其特点是数据元素间存在一对一的线性关系，很多实际问题可以通过线性表进行有效地解决。通过本次实验，我对线性表的概念、基本运算以及逻辑结构和物理结构的关系有了更好的理解。

单链表作为线性表的一种物理结构，其特点是用一组任意的存储单元存放线性表的数据元素，每个存储单元包括数据域和指针域（存放存储该数据元素的后继元素的存储位置）。这样的存储结构使得单链表的插入和删除操作相对较快，而查找和访问操作较慢。

在实现线性表的基本运算时，我熟练掌握了单链表的相应操作。例如链表的初始化、插入元素、删除元素等操作。同时，我也学会了实现附加功能，如链表

翻转、删除链表的倒数第 n 个结点、链表排序等。

在本次实验中，我体会到了理论联系实际的重要性。仅仅了解线性表的概念和原理还远远不够，只有通过实际动手实现、编写代码，才能更好地理解和掌握线性表的相关知识。

通过构造具有菜单的功能演示系统，我学会了如何组织和设计程序，使之具有较好的交互性和易用性。在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，让我更好地体会到了程序设计的过程。

总之，本次实验让我对线性表有了更深入的理解和实践经验，为以后在程序设计和算法研究中遇到线性表相关问题做好了充分的准备。

3.3 基于二叉链表的二叉树实现

通过学习基于二叉链表的二叉树实现，我收获了很多关于二叉树这一数据结构的知识。首先，我更加深刻地理解了二叉树的基本概念、运算以及逻辑结构和物理结构的关系。其次，我掌握了各种基本运算的实现方法，如创建、销毁、清空、判断空二叉树、求深度等。此外，我还学习了如何插入和删除结点、遍历二叉树等操作。

在学习的过程中，前序、中序和后序遍历的算法让我印象深刻。尤其是非递归算法的实现，它让我意识到在实际编程过程中，很多时候需要分析问题的本质，找到解决问题的突破口，才能将一个复杂的问题化简并得以解决。

此外，附加功能的实现也让我拓展了知识面，例如最大路径和和最近公共祖先问题，让我了解到了二叉树在实际应用中的用途。翻转二叉树更是让我对如何操作二叉树数据结构有了更多的灵活性。文件形式保存和加载操作也让我尝试思考如何设计高效的保存和加载策略，将一个复杂的结构有效地存储和读取。

3.4 基于邻接表的图实现

理解图的基本概念：在这个实验中，我将学习图的基本概念，包括顶点和关系之间的连接关系。了解图的逻辑结构和物理结构之间的关系，以及如何使用邻接表来表示图的物理结构。

实验中定义了 12 种基本运算，涵盖了图的创建、销毁、查找、赋值、插入、删除以及遍历等操作。通过实际编写代码实现这些基本运算，掌握如何操作图的

数据结构，并且加深对这些操作的理解。

除了基本运算，实验还提供了一些附加功能，如计算距离小于 k 的顶点集合、计算最短路径长度、计算连通分量等。通过实现这些附加功能，进一步探索图的应用领域，例如路径搜索和图的分析。其中，我对图的一些实际应用还不是很了解，日后加以改进

在实验中，还包括了图的文件形式保存和加载的操作。将设计文件数据记录格式，以高效保存图的数据逻辑结构，并实现相应的图文件保存和加载操作模式。这锻炼我在处理文件操作和数据持久化方面的能力。

实验要求设计数据结构来管理多个图，包括创建、添加和移除图的功能。学习如何有效地管理多个图，并可以在不同的图之间自由切换和操作，提高了我对多图操作的能力。

通过进行这个实验，我获得对图的概念和基本运算有更深入的理解，掌握使用邻接表作为图的物理结构来实现基本运算，熟悉图的附加功能的实现，例如最短路径计算和连通分量分析，锻炼文件操作和数据持久化的能力，学会管理多个图并进行相关操作。

附录

附录 A 基于顺序存储结构线性表实现的源程序

```
1  /*—— 头文件的申明 ——*/
2  #include<stdio .h>
3  #include<stdlib .h>
4  #include<locale .h>
5  #include "string .h"
6  /*—— 预定义 ——*/
7  // 定义布尔类型TRUE和FALSE
8  #define TRUE 1
9  #define FALSE 0
10
11 // 定义函数返回值类型
12 #define OK 1
13 #define ERROR 0
14 #define INFEASIBLE -1
15 #define OVERFLOW -2
16 typedef int status ;
17
18 // 顺序表中数据元素的类型
19 typedef int ElemType;
20
21 // 定义顺序表的初始长度和每次扩展的长度
22 #define LIST_INIT_SIZE 100
23 #define LISTINCREMENT 10
24
25 // 定义顺序表类型
26 typedef struct {
27     ElemType * elem; // 存储数据元素的数组指针
```

```
28     int length; // 当前长度
29     int listsize; // 当前可容纳的最大长度
30 }SqList;
31
32 // 定义线性表集合中的每个线性表的类型
33 typedef struct {
34     char name[30]; // 线性表的名称
35     SqList L; // 线性表本身
36 }LIST_ELEM;
37
38 // 定义线性表集合类型
39 typedef struct {
40     LIST_ELEM elem[10]; // 存储线性表的数组
41     int length; // 当前集合长度
42 }LISTS;
43 LISTS Lists; // 声明线性表集合的变量名为Lists
44
45 /*----- 函数申明 -----*/
46
47 status InitList (SqList& L); // 新建
48 status DestroyList (SqList& L); // 销毁
49 status ClearList (SqList& L); // 清空
50 status ListEmpty(SqList L); // 判空
51 status ListLength(SqList L); // 求长度
52 status GetElem(SqList L, int i, ElemType &e); // 获取元素
53 status LocateElem(SqList L, ElemType e, int (*p)(int, int)); // 判断位置
54
55 status compare(int a, int b); // 判断位置函数中调用的compare函数
56 status PriorElem(SqList L, ElemType e, ElemType &pre); // 获得前驱
57 status NextElem(SqList L, ElemType e, ElemType &next); // 获得后继
58 status ListInsert (SqList &L, int i, ElemType e); // 插入元素
```

```
58 status ListDelete (SqList &L,int i,ElemType &e); // 删除元素
59 status ListTraverse (SqList L,void (* visit )( int )); // 遍历输出
60 void visit ( int elem); // 遍历输出时候调用的visit函数
61 status MaxSubArray(SqList L); // 最大连续子数组
62 status SubArrayNum(SqList L , int u); // 和为k的子数组个数
63 status sort (SqList &L); // 顺序表排序
64 void show(); // 单个线性表的菜单
65 void show1(); // 多个线性表的菜单
66 void menu(); // 多个线性表的入口菜单
67 int savetofile (SqList L); // 线性表保存到文件
68 int getfromfile (SqList &L); // 从文件中读取线性表
69 status AddList(LISTS &Lists,char ListName[]); // 在Lists中增加一个空
    线性表
70 status RemoveList(LISTS &Lists,char ListName[]); // Lists中删除一个
    线性表
71 status LocateList(LISTS Lists ,char ListName[]); // 在Lists中查找线性
    表
72 void funtion (); // 多个线性表管理的封装函数
73 void showplus(); // 附加功能的菜单
74
75
76
77
78
79 /*----- main主函数 -----*/
80 int main()
81 {
82     // 修改控制台输出颜色
83     system("color 0b");
84     // 显示功能菜单
85     show();
```

```
86 // 输入操作编号
87 int order;
88 // 声明一个线性表
89 SqList L;
90 scanf("%d",&order);
91 while(order)
92 {
93     // 清除屏幕上的内容
94     system("cls");
95     // 根据操作编号执行相应的功能
96     switch(order){
97         case 1:
98             // 显示功能菜单
99             show();
100             // 初始化线性表
101             if( InitList (L)==OK) printf("线性表创建成功! \n");
102             else printf("线性表已经存在, 创建失败! \n");
103             getchar();
104             break;
105         case 2:
106             // 显示功能菜单
107             show();
108             // 销毁线性表
109             DestroyList(L);
110             getchar();
111             break;
112         case 3:
113             // 显示功能菜单
114             show();
115             // 清空线性表
116             ClearList (L);
```



```
117         getchar();
118         break;
119     case 4:
120         // 显示功能菜单
121         show();
122         // 判断线性表是否为空
123         ListEmpty(L);
124         getchar();
125         break;
126     case 5:
127         // 显示功能菜单
128         show();
129         // 获取线性表长度
130         int getdata ;
131         getdata = ListLength(L);
132         if( getdata !=INFEASIBLE )
133         {
134             printf ("线性表的长度是%d",getdata);
135         }
136         getchar();
137         break;
138     case 6:
139         // 显示功能菜单
140         show();
141         // 获取指定位置的元素
142         printf ("请输入你想获取第几个元素\n");
143         int i ;
144         scanf ("%d",&i);
145         int n ; int getdata1 ;
146         n = GetElem(L,i,getdata1 );
147
```

```
148         if(n == OK)
149         {
150             printf("成功获取到第%d个元素的值: %d\n",i,
                    getdata1);
151         }
152         getchar();
153         break;
154     case 7:
155         // 显示功能菜单
156         show();
157         // 查找指定元素
158         int u ;
159         printf("请输入一个数值e\n");
160         int e ; scanf("%d",&e);
161         printf("你想在表里查找一个比e大还是小的数据，大请
                输入1，小请输入0\n");
162         int getorder ; scanf("%d",&getorder);
163         int q;
164         if( getorder == 1)
165         {
166             printf("你想要这个数据比e大多少\n");
167
168             scanf("%d",&q);
169             if(LocateElem(L,e+q,compare) > 0)
170             {
171                 printf("你要查找到数据的下标的是%d\n",
                        LocateElem(L,e+q,compare));
172             }
173         }
174         else {
175             printf("你想要这个数据比e小多少\n");
```

```
176
177         scanf("%d",&q);
178         if(LocateElem(L,e+q,compare) > 0)
179         {
180             printf("你要查找到数据的下标的是%d\n",
181                    LocateElem(L,e+q,compare));
182         }
183     }
184     getchar();
185     break;
186 case 8:
187     // 显示功能菜单
188     show();
189     // 获取指定元素的前驱
190     printf("请问你想获得那个元素的前驱\n");
191     int v ;
192     scanf("%d",&v);
193     int pre_e;
194     if(PriorElem(L,v,pre_e)==OK)
195     {
196         printf("成功获得前驱，是%d\n",pre_e);
197     }
198     getchar();
199     break;
200 case 9:
201     // 显示功能菜单
202     show();
203     // 获取指定元素的后继
204     printf("请问你想获得哪个元素的后驱\n");
205     int p ;
206     scanf("%d",&p);
```

```
206         int next_e;
207         if (NextElem(L,p,next_e) == OK)
208         {
209             printf ("成功获取后驱，是%d\n",next_e);
210         }
211         getchar ();
212         break;
213     case 10:
214         // 显示功能菜单
215         show();
216         // 在指定位置之前插入元素
217         printf ("请问你想在第几个位置之前插入元素\n");
218         int r ;
219         scanf ("%d",&r);
220         printf ("插入的元素的值为\n");
221         int a ;
222         scanf ("%d",&a);
223         if ( ListInsert (L,r,a) == OK)
224         {
225             printf ("插入成功\n");
226         }
227         getchar ();
228         break;
229     case 11:
230         // 显示功能菜单
231         show();
232         // 删除指定位置的元素
233         printf ("请问你想删除第几个数据元素\n");
234         int b ;
235         scanf ("%d",&b);
236         int ee;
```

```
237         if( ListDelete (L,b,ee) == OK)
238         {
239             printf ("删除的数据元素是%d\n",ee);
240         }
241         getchar ();
242         break;
243     case 12:
244         // 显示功能菜单
245         show();
246         // 遍历线性表
247         if( ListTraverse (L, visit ) )
248         {
249             printf ("\n成功遍历\n");
250         }
251         getchar ();
252         break;
253     case 13:
254         // 显示函数列表
255         funtion ();
256         // 清除屏幕上的内容
257         system("cls");
258         // 显示功能菜单
259         show();
260         break;
261     case 0:
262         break;
263 } //end of switch
264 // 再次输入操作编号
265 scanf("%d",&order);
266 } //end of while
267 // 退出程序
```

```
268     printf ("欢迎下次再使用本系统! \n");
269     system("pause");
270     return 0;
271 }
272
273
274
275
276
277
278
279
280 // 1. 初始化表: 函数名称是InitList(L); 初始条件是线性表L不存在; 操
    作结果是构造一个空的线性表;
281 status  InitList (SqList& L)//线性表L不存在, 构造一个空的线性表; 返
    回OK, 否则返回INFEASIBLE。
282 {
283     // 请在这里补充代码, 完成本关任务
284     /***** Begin *****/
285     if(L.elem)
286         return INFEASIBLE;//线性表已存在, 返回线性表不存在的错误
        代码
287
288     L.elem = (ElemType *)malloc(sizeof(ElemType)*LIST_INIT_SIZE);//
        为线性表分配内存空间
289     L.length = 0; // 将线性表的长度设置为0
290     L.listsize = LIST_INIT_SIZE;//设置线性表的容量
291
292     return OK;//返回操作成功的代码
293
294     /***** End *****/
```

```
295 }
296
297 // 2. 销毁表：函数名称是DestroyList(L)；初始条件是线性表L已存在；
    操作结果是销毁线性表L；
298 status DestroyList(SqList& L)//如果线性表L存在，销毁线性表L，释放
    数据元素的空间，返回OK，否则返回INFEASIBLE。
299 {
300 // 请在这里补充代码，完成本关任务
301 /***** Begin *****/
302     if(!L.elem)//如果线性表不存在，返回线性表不存在的错误代码
303     {
304         printf("线性表不存在\n");
305         return INFEASIBLE;
306     }
307
308     free(L.elem); // 释放线性表中元素的内存空间
309     L.elem = NULL; // 将线性表指针置为空指针
310     L.length = 0; // 将线性表长度置为0
311     L.listsize = 0; // 将线性表容量置为0
312     printf("成功销毁线性表\n");
313     return OK; // 返回操作成功的代码
314
315 /***** End *****/
316 }
317 // 3. 清空表：函数名称是ClearList(L)；初始条件是线性表L已存在；操
    作结果是将L重置为空表；
318 status ClearList(SqList& L)
319 // 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。
320 {
321 // 判断线性表是否存在或是否为空
```

```
322     if (!L.length || !L.elem )
323     {
324         printf ("笨蛋，线性表不存在或者没有元素\n");
325         return INFEASIBLE;
326
327     }
328
329     L.length = 0;    // 将线性表长度设为0，相当于清空了线性表
330     printf ("成功删除线性表里面的元素啦\n");
331     return OK;
332 }
333
334 // 4. 判定空表：函数名称是ListEmpty(L)；初始条件是线性表L已存在；
    操作结果是若L为空表则返回TRUE,否则返回FALSE；
335 status ListEmpty(SqList L)
336 // 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。
337 {
338     // 判断线性表是否存在
339     if (!L.elem)
340     {
341         printf ("猪头，线性表不存在\n");
342         return INFEASIBLE;
343     }
344
345     // 判断线性表是否为空
346     if (L.length==0)
347     {
348         printf ("线性表是空的\n");
349         return TRUE;
350     }
```



```
351     printf ("线性表不是空的\n");
352     return FALSE;
353 }
354
355 // 5.求表长：函数名称是ListLength(L)；初始条件是线性表已存在；操作结果是返回L中数据元素的个数；
356 status ListLength(SqList L)
357 // 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
358 {
359 // 判断线性表是否存在
360     if(L.elem==NULL)
361     {
362         printf ("线性表不存在\n");
363         return INFEASIBLE;
364     }
365
366 // 返回线性表长度
367     else {
368         return L.length;
369     }
370 }
371
372 // 6.获得元素：函数名称是GetElem(L,i,e)；初始条件是线性表已存在，
// 1≤i≤ListLength(L)；操作结果是用e返回L中第i个数据元素的值；
373 status GetElem(SqList L, int i, ElemType &e)
374 // 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。
375 {
376 // 判断线性表是否存在
377     if (!L.elem)
```

```
378     {
379         printf ("线性表不存在\n");
380         return INFEASIBLE;
381     }
382
383 // 判断线性表序号i的合法性
384 if(i<1 || i>L.length)
385     {
386         printf ("获取的元素i不合法\n");
387         return ERROR;
388     }
389
390 // 获取线性表第i个元素
391 e = L.elem[i-1];
392     {
393         return OK;
394     }
395 }
396
397 // 7.查找元素：函数名称是LocateElem(L,e,compare()); 初始条件是线性
    表已存在；操作结果是返回L中第1个与e满足关系compare（）关系
    的数据元素的位序，若这样的数据元素不存在，则返回值为0；
398 int LocateElem(SqList L,ElemType e,int (*p)(int ,int ))
399 // 如果线性表L存在，查找元素e在线性表L中的位置序号并返回该序
    号；如果e不存在，返回0；当线性表L不存在时，返回INFEASIBLE
    。
400 {
401 // 判断线性表是否存在
402     if(!L.elem)
403     {
404         printf ("笨蛋，线性表不存在\n");
```

```
405         return INFEASIBLE;
406     }
407
408 // 在线性表中查找元素，若找到返回位置序号
409     for( int k=0;k<L.length;k++)
410     {
411         if(compare(L.elem[k],e))
412         {
413             return k+1;
414         }
415     }
416
417 // 没有找到符合的元素
418     printf("抱歉，没有找到符合的元素");
419     return ERROR;
420 }
421 // 8. 获得前驱：函数名称是PriorElem(L,cur_e,pre_e); 初始条件是线性
    表L已存在；操作结果是若cur_e是L的数据元素，且不是第一个，则
    用pre_e返回它的前驱，否则操作失败，pre_e无定义
422 status PriorElem(SqList L, ElemType cur_e, ElemType &pre_e) {
423     // 如果线性表L为空，返回INFEASIBLE
424     if (!L.elem) {
425         printf("Error: 该线性表不存在.\n");
426         return INFEASIBLE;
427     }
428
429     // 如果要获取前驱的元素是第一个，返回ERROR
430     if (L.elem[0] == cur_e) {
431         printf("Error: 要获取前驱的元素是第一个，不存在前驱.\n");
432         return ERROR;
433     }
```

```
434
435 // 查询要获取前驱的元素在表中的位置
436 for (int k = 1; k < L.length; k++) {
437     if (L.elem[k] == cur_e ) {
438         pre_e = L.elem[k-1];
439         return OK;
440     }
441 }
442
443 // 如果要获取前驱的元素不存在, 返回ERROR
444 printf ("Error: 该元素不存在于该线性表中。\\n");
445 return ERROR;
446 }
447
448 // 9. 获得后继: 函数名称是NextElem(L,cur_e,next_e); 初始条件是线性
    表L已存在; 操作结果是若cur_e是L的数据元素, 且不是最后一个,
    则用next_e返回它的后继, 否则操作失败, next_e无定义;
449 status NextElem(SqList L, ElemType cur_e, ElemType &next_e) {
450     // 如果线性表L为空, 返回INFEASIBLE
451     if (!L.elem) {
452         printf ("Error: 该线性表不存在。\\n");
453         return INFEASIBLE;
454     }
455
456     // 如果要获取后继的元素是最后一个, 返回ERROR
457     if (L.elem[L.length-1] == cur_e) {
458         printf ("Error: 要获取后继的元素是最后一个, 不存在后继。\\n"
459             );
460         return ERROR;
461     }
```

```
462 // 查询要获取后继的元素在表中的位置
463 for (int k = 0; k < L.length-1; k++) {
464     if (L.elem[k] == cur_e) {
465         next_e = L.elem[k+1];
466         return OK;
467     }
468 }
469
470 // 如果要获取后继的元素不存在, 返回ERROR
471 printf("Error: 该元素不存在于该线性表中.\n");
472 return ERROR;
473 }
474
475 // 10.插入元素, 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。
476 status ListInsert (SqList &L, int i, ElemType e)
477 {
478     if (!L.elem) {
479         // 线性表为空
480         printf("线性表为空\n");
481         return INFEASIBLE;
482     }
483
484     if (i < 1 || i > L.length + 1) {
485         // 插入位置不正确
486         printf("插入位置不正确\n");
487         return ERROR;
488     }
489
490     int k = i - 1;
```

```
491     if (L.length == L.listsize ) {
492         // 顺序表已满，需要重新分配空间
493         ElemType * newbase = (ElemType *)realloc(L.elem, sizeof(
494             ElemType) * (L.listsize + LISTINCREMENT));
495         if (!newbase)
496             return ERROR;
497         L.elem = newbase;
498         L.listsize += LISTINCREMENT;
499     }
500     // 将位置k及其后面的元素后移一位
501     for (int p = L.length - 1; p >= k; p--) {
502         L.elem[p + 1] = L.elem[p];
503     }
504
505     // 插入元素e
506     L.length++;
507     L.elem[k] = e;
508     return OK;
509 }
510
511 // 11.删除元素，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。
512 status ListDelete (SqList &L, int i, ElemType &e)
513 {
514     if (!L.elem) {
515         // 线性表为空
516         printf("线性表为空\n");
517         return INFEASIBLE;
518     }
```

```
519
520     if (i < 1 || i > L.length) {
521         // 删除位置不正确
522         printf("删除位置不正确\n");
523         return ERROR;
524     }
525
526     e = L.elem[i - 1];
527     // 将位置i后面的元素前移一位
528     for (int k = i - 1; k < L.length - 1; k++) {
529         L.elem[k] = L.elem[k + 1];
530     }
531     L.length--;
532     return OK;
533 }
534
535 // 12.遍历表，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。
536 status ListTraverse (SqList L, void (* visit)(int))
537 {
538     if (!L.elem) {
539         // 线性表为空
540         printf("线性表为空\n");
541         return INFEASIBLE;
542     }
543
544     for (int k = 0; k < L.length; k++) {
545         // 调用 visit对每个元素进行操作
546         visit (L.elem[k]);
547         if (k != L.length - 1) {
548             putchar(' ');
```

```
549     }
550 }
551 return OK;
552 }
553
554 int compare(int a ,int b)
555 {
556     if(a == b)
557     {
558         return 1;
559     }
560     return 0;
561 }
562 // 遍历输出时候调用的visit函数
563 void visit ( int elem)
564 {
565     printf ("%d",elem);
566 }
567 // 求最大连续子数组
568 int MaxSubArray(SqList L)
569 {
570     if(!L.elem) // 如果线性表不存在
571     {
572         printf ("笨蛋，线性表不存在");
573         return INFEASIBLE;
574     }
575     int max ,current ; // 定义max表示最大值，current表示当前值
576     max = current = L.elem[0]; // 初始化max和current为线性表L的第一个元素
577     for( int k = 1;k<L.length;k++) // 遍历线性表L
578     {
```



```
579         if( current <= 0) //如果当前值小于等于0
580         {
581             current = L.elem[k]; //将当前值置为下一个元素的值
582         } else {
583             current += L.elem[k]; //将当前值做累加操作
584         }
585         if(max < current) //如果最大值小于当前值
586         {
587             max = current; //更新最大值为当前值
588         }
589     }
590     return max; //返回最终最大值
591 }
592 // 计算线性表L中和为u的子数组个数
593 int SubArrayNum(SqList L, int u)
594 {
595     if(!L.elem)
596     {
597         printf("笨蛋，线性表不存在");
598         return INFEASIBLE; //返回线性表不存在的错误代码
599     }
600     int count = 0; //子数组个数计数器
601     for( int k = 0; k < L.length; k++)
602     {
603         int sum = 0; //子数组元素之和
604         for( int i = k; i < L.length; i++)
605         {
606             sum += L.elem[i]; //累加元素值
607             if(sum == u)
608             {
609                 count ++; //子数组和为u，计数器加一
```

```
610         }
611     }
612 }
613     return count ;
614 } // 冒泡排序实现顺序表排序
615 int sort (SqList &L)
616 {
617     for (int k = 0; k < L.length - 1; k++) // 控制轮数
618     {
619         for (int i = 0; i < L.length - 1 - k; i++) // 每轮比较相邻元素
620         {
621             if (L.elem[i] > L.elem[i+1]) // 如果左边元素大于右边元素
622             {
623                 int tmp = L.elem[i]; // 交换两个元素的位置
624                 L.elem[i] = L.elem[i+1];
625                 L.elem[i+1] = tmp;
626             }
627         }
628     }
629 }
630
631 /* 输出程序的菜单 */
632 void show()
633 {
634     for (int k = 0; k <= 119 ; k++) // 打印分割线
635     {
636         putchar('—');
637     }
638     printf ("                Menu for Linear Table On Sequence Structure \n");
639     printf ("                1. InitList                7.
```

```

        LocateElem\n");
640    printf ("          2. DestroyList          8.
        PriorElem\n");
641    printf ("          3. ClearList          9.
        NextElem\n");
642    printf ("          4. ListEmpty          10.
        ListInsert\n");
643    printf ("          5. ListLength          11.
        ListDelete\n");
644    printf ("          6. GetElem          12.
        ListTraverse\n");
645    printf ("          13. plusfunction          \n")
        ;
646    printf ("          0. exit          \n");
647    for (int k = 0; k <= 119 ;k++) // 打印分割线
648    {
649        putchar('—');
650    }
651    putchar('\n');
652    // 下面是一个带有动态图像的输出，可以略过
653    printf (" 请选择你的操作[0~13]:");
654    putchar('\n');
655    for (int k = 0; k <= 119 ;k++)
656    {
657        putchar('—');
658    }
659
660    printf ("      ^          /\n");
661    printf ("      /\ 7          □ _ ^\n");
662    printf ("      /  |          /  /\n");
663    printf ("      |  Z _,<      /  /‘F\n");

```

```

664     printf(" |          [F]      /      > \n");
665     printf(" Y          ' /      ^\n");
666     printf(" ?● ? ●      ?? <      ^\n");
667     printf(" () ^          | \ <\n");
668     printf(" >? ?_   Й      | / / \n");
669     printf(" / ^      / ?<| \ \ \n");
670     printf(" [F]?      ( /      | / / \n");
671     printf(" 7          | / \n");
672     printf(" >—r - - '?— _ \n");
673
674     putchar('\n');
675 }
676
677 void show1()
678 {
679     // 打印菜单前的分隔符
680     printf("这是多个线性表管理中的菜单\n");
681     for(int k = 0; k<= 119 ;k++)
682     {
683         putchar('-');
684     }
685     // 打印菜单内容
686     printf("          Menu for Linear Table On Sequence Structure \n"
687           );
687     printf("          1. InitList          7.
688           LocateElem\n");
688     printf("          2. DestroyList       8.
689           PriorElem\n");
689     printf("          3. ClearList        9.
690           NextElem \n");
690     printf("          4. ListEmpty        10.

```

```

        ListInsert \n");
691     printf ("          5. ListLength          11.
        ListDelete \n");
692     printf ("          6. GetElem          12.
        ListTraverse \n");
693     printf ("          0. exit \n");
694     // 打印菜单后的分隔符
695     for(int k = 0; k<= 119 ;k++)
696     {
697         putchar(' ');
698     } putchar('\n');
699     // 打印额外的一些菜单内容
700     printf ("          请选择你的操作[0~1]:");
701     putchar('\n');
702     for(int k = 0; k<= 119 ;k++)
703     {
704         putchar(' ');
705     }
706     // 打印一段 ASCII 艺术
707     printf ("          ^          /\n");
708     printf ("          /\ 7          □ _\n");
709     printf ("          /  |          /  /\n");
710     printf ("          |  Z _,<  /  /["F"\n");
711     printf ("          |          ["F"  /  > \n");
712     printf ("          Y          '  /  /\n");
713     printf ("          ?● ? ●  ?? <  /\n");
714     printf ("          () ^          |  \ <\n");
715     printf ("          >? ?_  ı  |  /  /\n");
716     printf ("          / ^          /  ?<|  \ \n");
717     printf ("          ["F"?  ( /  |  /  /\n");
718     printf ("          7          |  /\n");

```

```
719     printf ("      >—r - - '?— _\n");
720
721     putchar ('\n');
722
723 }
724
725 void menu()
726 {
727     printf ("1.创建一个线性表\n");
728     printf ("2.删除一个线性表\n");
729     printf ("3.查找一个线性表和进行操作\n");
730     printf ("0.退出线性表的管理\n");
731
732 }
733
734 // 函数名: savetofile
735 // 功能: 将线性表 L 中的元素保存到文件中
736 // 输入参数: 线性表 L
737 // 返回值: 操作成功返回 OK, 否则返回 INFEASIBLE
738 int savetofile (SqList L)
739 {
740     printf ("请输入你想保存到的文件名\n"); // 提示输入要保存的文件名
741     char arr [30];
742     scanf ("%s", arr); // 获取用户输入的文件名
743
744     FILE *fp;
745     fp = fopen (arr, "w"); // 以写入方式打开文件
746     if (fp == NULL) { // 如果文件打开失败
747         printf ("error"); // 输出错误信息
748         return INFEASIBLE; // 返回错误代码
749     }
```

```
750
751     int i ;
752     for( i =0; i< L.length; i++) // 遍历线性表中的元素
753     {
754         fprintf (fp, "%d", L.elem[i]); // 将元素写入文件中
755     }
756
757     fclose (fp); // 关闭文件
758     return OK; // 返回操作成功代码
759 }
760
761 // 函数名: getfromfile
762 // 功能: 从文件中读取线性表 L 的元素
763 // 输入参数: 线性表 L 的地址
764 // 返回值: 操作成功返回 OK, 否则返回 INFEASIBLE
765 int getfromfile (SqList &L)
766 {
767     if (!L.elem) // 如果线性表不存在
768     {
769         printf ("笨蛋, 线性表不存在\n"); // 输出错误信息
770         return INFEASIBLE; // 返回错误代码
771     }
772
773     printf ("请输入你要读取的文件名:\n"); // 提示输入要读取的文件名
774     char name[30] = {'\0'};
775     scanf ("%s", name); // 获取用户输入的文件名
776
777     FILE *fp = fopen(name, "r"); // 以只读方式打开文件
778     if (fp == NULL) { // 如果文件打开失败
779         printf ("error"); // 输出错误信息
780         return INFEASIBLE; // 返回错误代码
```

```
781     }
782
783     int j;
784     while( fscanf(fp, "%d", &j) != EOF) // 读取文件中的元素
785     {
786         L.elem[L.length++] = j; // 将元素添加到线性表中
787     }
788
789     fclose(fp); // 关闭文件
790     return OK; // 返回操作成功代码
791 }
792
793 // 本函数的功能为在线性表管理系统Lists中增加一个名为ListName的空
    线性表
794 // 如果已经存在名为ListName的线性表，则返回INFEASIBLE，否则返
    回OK
795 status AddList(LISTS &Lists, char ListName[])
796 {
797     // 遍历已有线性表，查找是否已经存在名为ListName的线性表
798     for( int i = 0; i < Lists.length ; i++)
799     {
800         if( strcmp(ListName, Lists.elem[i].name) == 0)
801         {
802             printf("这个名字的线性表已经存在了");
803             return INFEASIBLE;
804         }
805     }
806
807     // 如果不存在名为ListName的线性表，则在Lists中添加一个新的线性表
808     Lists.length++; // 在线性表管理系统中增加一个线性表
809     int n = 0; // 初始化线性表的长度n为0，即线性表为空
```



```
810
811 // 将新线性表的名字设置为ListName
812     strcpy ( Lists .elem[ Lists . length -1].name ,ListName);
813
814 // 将新线性表的数据初始化为空，即没有元素
815     Lists .elem[ Lists . length -1].L.elem=NULL;
816
817 // 注释掉的 InitList不需要调用，因为L.elem已设置为NULL
818 //  InitList ( Lists .elem[ Lists . length -1].L);
819
820 // 返回添加线性表操作执行成功
821     return OK;
822 }
823
824 status  RemoveList(LISTS &Lists,char ListName[])
825 // Lists 中删除一个名称为ListName的线性表
826 {
827     // 请在这里补充代码，完成本关任务
828     /***** Begin *****/
829     // 遍历线性表
830     for( int k = 0;k<Lists . length ;k++)
831     {
832         // 找到名称为ListName的线性表
833         if(strcmp( Lists .elem[k].name,ListName)==0)
834         {
835             // 销毁线性表
836             DestroyList ( Lists .elem[k].L);
837             // 将后面的线性表前移，覆盖当前位置
838             for( int i = k;i<Lists . length -1;i++)
839             {
840                 strcpy ( Lists .elem[i ].name, Lists .elem[i +1].name); //
```

注意要名字和结构体一起移动，我感觉没有捷径

```
841         Lists.elem[i].L.elem = Lists.elem[i+1].L.elem;
842         Lists.elem[i].L.length = Lists.elem[i+1].L.length;
843         Lists.elem[i].L.listsize = Lists.elem[i+1].L.listsize;
844     }
845     Lists.length--; // 线性表数量减一
846     return OK;
847 }
848 }
849 // 没有找到目标线性表，返回错误
850 return ERROR;
851
852 /***** End *****/
853 }
854
855 int LocateList(LISTS Lists, char ListName[])
856 // 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-1
857 {
858     // 请在这里补充代码，完成本关任务
859     /***** Begin *****/
860     // 循环遍历线性表数组
861     for(int k = 0 ; k < Lists.length ; k++)
862     {
863         // 如果找到了指定名称的线性表
864         if(strcmp(Lists.elem[k].name, ListName) == 0)
865         {
866             // 返回该线性表的逻辑序号（索引位置+1）
867             return k+1;
868         }
869     }
```

```
870 // 如果未找到指定名称的线性表, 则返回-1
871 return -1;
872
873 /***** End *****/
874 }
875
876
877 void funtion ()
878 {
879     showplus(); // 显示菜单页面头部
880     SqList S ; // 用于操作的线性表, S代表单个线性表
881
882     int order ;
883     scanf("%d",&order); // 获取用户输入的命令
884     while(order) { // 如果用户输入了命令, 即非0, 则进入循环
885
886         system("cls"); // 清空控制台屏幕
887         showplus(); // 重新显示菜单页面头部
888
889         switch (order) { // 根据用户命令进行相应操作
890             case 1:
891                 // 创建线性表
892
893                 S.elem = (ElemType *) malloc(sizeof(ElemType) *
894                     LIST_INIT_SIZE); // 为线性表分配内存空间
895                 S.length = 0; // 初始化线性表长度为0
896                 S.listsize = LIST_INIT_SIZE; // 初始化线性表可用大
897                     小为LIST_INIT_SIZE
898
899                 printf("请输入一串数据, 数据之间用空格隔开,最后一
```

```
        个数据输入完直接回车\n");
899         int number;
900         scanf("%d", &number);
901         char c;
902         c = getchar();
903         while (c != '\n') { // 当输入未结束时, 循环获取数
            据并添加到线性表中
904             S.elem[S.length++] = number;
905             scanf("%d", &number);
906             c = getchar();
907         }
908         S.elem[S.length++] = number; // 将最后一个数据添加
            到线性表中
909         break;
910     case 2:
911         // 寻找最大子序列
912         int r;
913
914         r = MaxSubArray(S); // 调用MaxSubArray函数计算最
            大子序列之和
915         if (r != -1) {
916             printf("最大连续子序列之和是%d\n", r);
917         }
918         // printf("请输入下一个命令\n");
919         break;
920     case 3:
921         // 寻找和为K的连续子序列
922         printf("这个函数来寻找一个和为K的连续子序列\n");
923         int k;
924         printf("请输入一个K值\n");
925         scanf("%d", &k);
```

```
926         if (SubArrayNum(S, k) != -1) { // 调用SubArrayNum
           函数在线性表中查找和为K的连续子序列
927         printf ("线性表中和为k的连续子列数量为%d\n",
           SubArrayNum(S, k));
928     }
929     break;
930 case 4:
931     // 对线性表进行排序
932     sort (S); // 调用sort函数对线性表进行排序
933     printf ("排完序后的线性表如下:\n");
934     ListTraverse (S, visit ); // 使用 ListTraverse函数遍历
           并输出线性表中的数据
935     putchar ( '\n' );
936     break;
937 case 5:
938     // 将线性表保存到文件中
939     printf ("现在进行线性表的文件形式保存\n");
940     int message;
941     message = savetofile (S); // 调用 savetofile 函数将线
           性表保存到文件中
942     if(message == OK)
943     {
944         printf ("存储成功\n");
945         // printf ("请输入下一个命令\n");
946     }
947     break;
948 case 6:
949     // 从文件中读取线性表数据
950     printf ("现在进行文件的读取\n");
951     SqList p;
952     p.elem = NULL;
```

```
953         InitList (p);    // 用这个线性表来存储读取的数据
954         if( getfromfile (p) == OK)    // 调用 getfromfile 函数从文件
           中读取数据并保存到p中
955     {
956         printf ("读取成功\n读取的数据如下:\n");
957         ListTraverse (p, visit );    // 使用 ListTraverse 函数遍历并输出线性表中的数据
958     }
959     else {
960         printf ("读取失败");
961     }
962     // printf ("请输入下一个命令\n");
963     break;
964     // 进行多个线性表的管理
965     case 7:
966         printf ("现在进行多个线性表的管理\n");
967         menu(); // 显示菜单
968         int a;
969         scanf ("%d", &a);
970         while (a) {
971             switch (a) {
972                 // 创建线性表
973                 case 1:
974                     printf ("现在进行创建线性表\n");
975                     printf ("请输入你想创建的线性表的名字\n");
976                     char name1[30];
977                     scanf ("%s", name1);
978                     int u;
979                     u = AddList( Lists , name1); // 添加线性表
980                     if (u == OK) {
```

```
981         printf("创建成功啦\n");
982     }
983     if (u == INFEASIBLE) {
984         system("pause");
985     }
986     break;
987     // 删除线性表
988 case 2:
989     printf("现在进行删除线性表\n");
990     printf("请输入你想创建的线性表的名字\n"
991           );
992     char name2[30];
993     scanf("%s", name2);
994     RemoveList(Lists, name2); // 删除线性表
995     break;
996     // 查找和操作线性表
997 case 3:
998     printf("现在进行线性表的查找和操作\n");
999     printf("请输入你想查找和操作的线性表的
1000           名字\n");
1001     char name3[30];
1002     scanf("%s", name3);
1003     int judge;
1004     judge = LocateList( Lists , name3); // 查找
1005                                           线性表
1006     if (judge == -1) {
1007         printf("不存在这个线性表\n");
1008         system("pause");
1009     } else {
1010         printf("线性表存在鸭鸭\n");
1011         printf("现在对这个线性表进行操作\n")
```

```

;
1009     int order;
1010     show(); // 显示线性表操作菜单
1011     scanf("%d", &order);
1012     while (order) {
1013         switch (order) {
1014             // 初始化线性表
1015             case 1:
1016                 show1(); // 显示操作提示
1017                 if ( InitList ( Lists .elem[
1018                     judge - 1].L) == OK)
1019                     printf ("线性表创建成功! \n");
1020                 else printf ("线性表已经
1021                     存在, 创建失败! \n");
1022                 getchar ();
1023                 break;
1024                 // 删除线性表
1025             case 2:
1026                 show1();
1027                 DestroyList ( Lists .elem[
1028                     judge - 1].L); // 销毁
1029                 线性表
1030                 getchar ();
1031                 break;
1032                 // 清空线性表
1033             case 3:
1034                 show1();
1035                 ClearList ( Lists .elem[judge
1036                     - 1].L); // 清空线性
1037                 表

```



```
1032         getchar();
1033         break;
1034         // 判断线性表是否为空
1035     case 4:
1036         show1();
1037         ListEmpty(Lists.elem[judge
1038                     - 1].L);
1038         getchar();
1039         break;
1040         // 获取线性表的长度
1041     case 5:
1042         show1();
1043         int getdata;
1044         getdata = ListLength(Lists
1045                               .elem[judge - 1].L);
1045         if (getdata !=
1046             INFEASIBLE) {
1046             printf("线性表的长度
1047                    是%d", getdata);
1047         }
1048         getchar();
1049         break;
1050         // 获取线性表中的元素
1051     case 6:
1052         show1();
1053         printf("请输入你想获取第
1054                几个元素\n");
1054         int i;
1055         scanf("%d", &i);
1056         int n;
1057         int getdata1;
```

```
1058         n = GetElem(Lists.elem[
            judge - 1].L, i,
            getdata1);
1059         if (n == OK) {
1060             printf ("成功获取到第
                %d个元素的值: %
                d\n", i, getdata1);
1061         }
1062         getchar ();
1063         break;
1064         // 查找线性表中的元素
1065     case 7:
1066         show1();
1067         int u;
1068         printf ("请输入一个数值e\n
            ");
1069         int e;
1070         scanf ("%d", &e);
1071         printf ("你想在表里查找一个比e大还是小的数
            据, 大请输入1, 小请
            输入0\n");
1072         int getorder;
1073         scanf ("%d", &getorder);
1074         int q;
1075         if (getorder == 1) {
1076             printf ("你想要这个数
                据比e大多少\n");
1077             scanf ("%d", &q);
1078             if (LocateElem(Lists.
                elem[order - 1].L,
```

```
1079         e + q, compare) >
            0) {
            printf ("你要查找
                到数据的下标
                的是%d\n",
                LocateElem(
                Lists.elem[
                judge - 1].L, e
                + q, compare))
                ;
1080     }
1081 } else {
1082     printf ("你想要这个数
                据比e小多少\n");
1083     scanf ("%d", &q);
1084     if (LocateElem(Lists.
                elem[judge - 1].L,
                e + q, compare) >
                0) {
1085         printf ("你要查找
                到数据的下标
                的是%d\n",
                LocateElem(
                Lists.elem[
                judge - 1].L, e
                + q, compare))
                ;
1086     }
1087 }
1088 getchar ();
1089 break;
```

```
1090 // 获取线性表中某个元素
      的前驱
1091 case 8:
1092     show1();
1093     printf("请问你想获得那个
      元素的前驱\n");
1094     int v;
1095     scanf("%d", &v);
1096     int pre_e;
1097     if (PriorElem( Lists .elem[
      judge - 1].L, v, pre_e
      ) == OK) {
1098         printf("成功获得前
      驱，是%d\n", pre_e
      );
1099     }
1100     getchar();
1101     break;
1102 // 获取线性表中某个元素
      的后驱
1103 case 9:
1104     show1();
1105     printf("请问你想获得哪个
      元素的后驱\n");
1106     int p;
1107     scanf("%d", &p);
1108     int next_e;
1109     if (NextElem(Lists.elem[
      judge - 1].L, p,
      next_e) == OK) {
1110         printf("成功获取后
```

```

                                                                    驱，是%d\n",
                                                                    next_e);
1111     }
1112     getchar();
1113     break;
1114     // 在线性表中插入元素
1115 case 10:
1116     show1();
1117     printf("请问你想在第几个
                                                                    位置之前插入元素\n");
1118     int r;
1119     scanf("%d", &r);
1120     printf("插入的元素的值为
                                                                    \n");
1121     int a;
1122     scanf("%d", &a);
1123     if ( ListInsert ( Lists .elem[
                                                                    judge - 1].L, r, a) ==
                                                                    OK) {
1124         printf("插入成功\n");
1125     }
1126     getchar();
1127     break;
1128     // 在线性表中删除元素
1129 case 11:
1130     show1();
1131     printf("请问你想删除第几
                                                                    个数据元素\n");
1132     int b;
1133     scanf("%d", &b);
1134     int ee;
```

```
1135         if ( ListDelete ( Lists .elem[
                judge - 1].L, b, ee)
                == OK) {
1136             printf ("删除的数据元
                    素是%d\n", ee);
1137         }
1138         getchar ();
1139         break;
1140         // 遍历线性表
1141         case 12:
1142             show1();
1143             if ( ListTraverse ( Lists .
                elem[judge - 1].L,
                visit )) {
1144                 printf ("\n成功遍历\n"
                    );
1145             }
1146             getchar ();
1147             break;
1148         case 0:
1149             break;
1150     }
1151     scanf ("%d", &order);
1152 }
1153 }
1154 break;
1155 case 0:
1156     break;
1157 }
1158 system ("cls");
1159 menu(); // 显示菜单
```

```
1160         printf("请输入下一个命令\n");
1161         scanf("%d", &a);
1162     }
1163     break;
1164     default :
1165         printf("输入的命令不正确，请再次输入\n");
1166         break;
1167     }
1168     printf("请输入下一个命令\n");
1169     scanf("%d", &order);
1170 }
1171 }
1172
1173 void showplus()
1174 {
1175     for( int k = 0; k<= 119 ;k++)
1176     {
1177         putchar(' ');
1178     } putchar('\n');
1179     printf("这是一些附加的功能，和之前的线性表不兼容\n");
1180     printf("          1. 初始化线性表\n");
1181     printf("          2. maxSubarray          3.
          subarrayNum\n");
1182     printf("          4. sortList          5.
          savetofile \n");
1183     printf("          6. getfromfile          7.
          managelist\n");
1184     printf("          0. Exit\n");
1185     for( int k = 0; k<= 119 ;k++)
1186     {
1187         putchar(' ');
```

```
1188     } putchar('\n');
1189 }
```

附录 B 基于链式存储结构线性表实现的源程序

```
1  /*—— 头文件的申明 ——*/
2  #include<stdio.h>
3  #include<stdlib.h>
4  #include "string.h"
5
6  /*—— 预定义 ——*/
7  // 定义布尔类型TRUE和FALSE
8  #define TRUE 1
9  #define FALSE 0
10
11 // 定义函数返回值类型
12 #define OK 1
13 #define ERROR 0
14 #define INFEASIBLE -1
15 #define OVERFLOW -2
16
17 // 初始链表的最大长度
18 #define LIST_INIT_SIZE 100
19 // 每次新增的长度
20 #define LISTINCREMENT 10
21
22 // 定义数据元素类型
23 typedef int ElemType;
24 typedef int status ;
25
26 // 定义单链表（链式结构）结点的结构体
```



```

27 typedef struct LNode{
28     ElemType data; // 结点的数据元素
29     struct LNode *next; // 指向下一个结点的指针
30 }LNode, *LinkList;
31
32 // 定义链表集合的结构体
33 typedef struct {
34     struct {
35         char name[30]; // 集合的名称, 最多可以有 30 个字符
36         LinkList L; // 指向链表头结点的指针
37     }elem[30]; // 集合中最多包含 30 个链表
38     int length; // 集合中包含的链表数目
39 }LISTS;
40
41 LISTS Lists; // 链表集合实例化为Lists对象
42
43 /*—— 函数申明 ——*/
44 status InitList (LinkList &L); // 新建
45 status DestroyList(LinkList &L); // 销毁
46 status ClearList (LinkList &L); // 清空
47 status ListEmpty(LinkList L); // 判空
48 status ListLength(LinkList L); // 求长度
49 status GetElem(LinkList L, int i, ElemType &e); // 获取元素
50 status LocateElem(LinkList L, ElemType e, int (*vis)(int , int )); // 判
    断位置
51 status PriorElem(LinkList L, ElemType e, ElemType &pre); // 前驱
52 status NextElem(LinkList L, ElemType e, ElemType &next); // 后继
53 status ListInsert (LinkList &L, int i, int num); // 插入
54 status ListDelete (LinkList &L, int i, ElemType &e); // 删除
55 status ListTraverse (LinkList L, void (*vi)(int )); // 遍历
56 status AddList(LISTS &Lists, char ListName[]);

```

```
57 status RemoveList(LISTS &Lists,char ListName[]);
58 status LocateList(LISTS Lists,char ListName[]);
59 void SearchList(LISTS Lists); //展示已经创建的线性表
60 status compare(int a,int b); //判断位置函数时候调用的比较函数
61 void visit (int x); //遍历函数时候调用的输出函数
62 void reverseList (LinkList L); //翻转线性表
63 void RemoveNthFromEnd(LinkList L,int n); //删除倒数元素
64 void sortList (LinkList L); //排序
65 void savetofile (LinkList L,char name[]); //保存到文件
66 void getfromfile (LinkList L,char name[]); //读取文件
67 void fun01(); //封装的多个线性表的处理函数
68 void fun02(LinkList &L ); //封装的处理单个线性表的处理函数
69 void menu(); //管理多个线性表的菜单
70 void show_normal(); //单个线性表的菜单
71 void Menuofinsert(); //插入的菜单
72
73 /*----- main主函数 -----*/
74 int main()
75 {
76     system("color 37");
77     fun01(); //调用封装函数
78
79 }
80
81
82 /*----- 函数定义 -----*/
83 // (1) 初始化表: 函数名称是InitList(L); 初始条件是线性表L不存在;
// 操作结果是构造一个空的线性表;
84 status InitList (LinkList &L)
85 // 线性表L不存在, 构造一个空的线性表, 返回OK, 否则返回
// INFEASIBLE。
```

```
86 {
87 // 如果线性表L已存在，则返回 INFEASIBLE
88     if(L)
89         return INFEASIBLE;
90 // 分配一个新的节点作为线性表头结点
91     L = (LinkedList)malloc( sizeof(LNode));
92 // 将头结点的指针域置为空
93     L->next = NULL;
94 // 返回 OK
95     return OK;
96 }
97
98 // (2) 销毁表：函数名称是DestroyList(L)；初始条件是线性表L已存在；操作结果是销毁线性表L；
99 status DestroyList(LinkedList &L)
100 // 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。
101 {
102 // 如果线性表L不存在，则返回 INFEASIBLE
103     if(!L)
104     {
105 // printf ("这个线性表不存在或未初始化,无法销毁\n");
106         // printf ("线性表不存在或者未初始化\n");
107         return INFEASIBLE;
108     }
109 // 定义指针 p 指向当前结点，q 指向下一个结点
110     LinkedList p = L, q;
111 // 如果当前结点不为空，则继续循环
112     while(p)
113     {
114 // 将 q 指向当前结点的下一个结点
```

```
115         q = p->next;
116 // 释放当前结点的空间
117         free(p);
118 // 将指针 p 指向 q, 继续循环
119         p = q;
120     }
121 // 返回 OK
122     return OK;
123 }
124
125 // (3) 清空表: 函数名称是ClearList(L); 初始条件是线性表L已存在;
// 操作结果是将L重置为空表;
126 status ClearList (LinkedList &L)
127 {
128     // 如果线性表L不存在, 返回INFEASIBLE
129     if (!L)
130     {
131         printf ("线性表不存在或未初始化, 无法进行清空\n");
132         return INFEASIBLE;
133     }
134     // 如果线性表L为空, 不需要操作
135     if (L->next == NULL)
136     {
137         printf ("线性表已经是空的了, 不需要操作\n");
138         return INFEASIBLE;
139     }
140     LinkedList p = L->next; // 指向第一个元素节点
141     while(p)
142     {
143         free(p); // 释放当前节点
144         p = p->next; // 指向下一个节点
```

```
145     }
146     L->next = NULL;          // 将头节点指向NULL，清空线性表
147     return OK;
148 }
149
150 // (4) 判定空表：函数名称是ListEmpty(L)；初始条件是线性表L已存在；
    // 操作结果是若L为空表则返回TRUE,否则返回FALSE；
151 status ListEmpty(LinkList L)
152 {
153     // 如果线性表L不存在，返回INFEASIBLE
154     if (!L)
155     {
156         printf ("线性表不存在或未初始化，无法进行清空\n");
157         return INFEASIBLE;
158     }
159     // 如果够了L的第一个元素为空，表明线性表为空，返回TRUE
160     if (L->next == NULL)
161     {
162         printf ("线性表是空的\n");
163         return TRUE;
164     }
165     // 线性表不为空，返回FALSE
166     printf ("线性表不是空的\n");
167     return FALSE;
168 }
169
170 // (5) 求表长：函数名称是ListLength(L)；初始条件是线性表已存在；
    // 操作结果是返回L中数据元素的个数；
171 int ListLength(LinkList L)
172 // 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
173 {
```

```
174  /****** Begin *****/
175  // 首先要检查线性表是否存在
176  if (!L) {
177      printf ("线性表不存在或未初始化，无法进行清空\n");
178      return INFEASIBLE;
179  }
180
181  L = L->next;    // 不要把头节点考虑
182  int number = 0;  // 用来 记录长度
183  while (L) {      // 遍历链表，计算数据元素的个数
184      number++;
185      L = L->next;
186  }
187  return number;
188
189  /****** End *****/
190 }
191
192 // (6) 获得元素：函数名称是GetElem(L,i,e)；初始条件是线性表已存
    在， $1 \leq i \leq \text{ListLength}(L)$ ；操作结果是用e返回L中第i个数据元素的值；
193 status GetElem(LinkList L, int i, ElemType &e)
194 // 获取线性表L中第i个元素，将其存储在e中
195 {
196     if (!L) // 若L为空，表示线性表不存在或未初始化
197     {
198         printf ("线性表不存在或未初始化，无法进行清空\n");
199         return INFEASIBLE; // 返回INFEASIBLE
200     }
201     int number = 0; // 记录当前遍历到的节点数，从0开始
202     LinkList p = L; // 定义p指针，指向L
203     p = L->next; // 跳过头节点，从第一个存储数据的节点开始遍历
```

```

204     while(p)
205     {
206         number++; // 遍历到一个节点, number加1
207         if(number == i) // 找到第i个节点
208         {
209             e = p->data; // 将找到的节点的数据存储在e中
210             return OK; // 返回OK
211         }
212         p = p->next; // 指针p指向下一个节点
213     }
214     printf("i的值不合法, 无法操作\n");
215     return ERROR; // 遍历完整个线性表, 未找到第i个节点, 返回
        ERROR
216 }
217
218 // (7) 查找元素: 函数名称是LocateElem(L,e,compare()); 初始条件是线
        性表已存在;
219 // 操作结果是返回L中第1个与e满足关系compare () 关系的数据元素的
        位序, 若这样的数据元素不存在, 则返回值为0;
220 status LocateElem(LinkList L,ElemType e,int (*vis)(int ,int ))
221 // 查找元素e在线性表L中的位置序号
222 // 当e存在时, 返回其在线性表中的位置序号
223 // 当e不存在时, 返回ERROR
224 // 当线性表L不存在时, 返回INFEASIBLE
225 {
226     // 当线性表L不存在时, 返回INFEASIBLE
227     if(!L)
228     {
229         printf("线性表不存在或未初始化, 无法进行查找\n");
230         return INFEASIBLE;
231     }

```

```
232
233 // 从头结点的下一个结点开始向后遍历
234 LinkList p = L->next;
235
236 // 记录当前位置序号
237 int number = 0;
238
239 // 遍历链表，查找元素e
240 while(p)
241 {
242     number++;
243
244     if( vis(p->data,e) == 1) // 如果找到元素e，返回其位置序号
245     {
246         return number;
247     }
248     p = p->next;
249 }
250
251 // 如果遍历完整个线性表仍未找到元素e，返回ERROR
252 printf("没有所要查询的元素\n");
253 return ERROR;
254 }
255
256 // (8) 获得前驱：函数名称是PriorElem(L,cur_e,pre_e);
257 // 初始条件是线性表L已存在；操作结果是若cur_e是L的数据元素，且
    不是第一个，则用pre_e返回它的前驱，否则操作失败，pre_e无定
    义；
258 status PriorElem(LinkList L,ElemType e,ElemType &pre)
259 // 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。
260 {
261     if (!L)    // 如果链表L不存在，则返回INFEASIBLE，表示不可行
262     {
263         printf ("线性表不存在或未初始化，无法进行清空\n");
264         return INFEASIBLE;
265     }
266     if (L->next == NULL) // 如果链表L为空，则返回ERROR，表示出错
267     {
268         printf ("线性表里面没有元素\n");
269         return ERROR;
270     }
271     LinkList p = L->next;
272     if (p->data == e )    // 如果所要查找的元素e是第一个元素，不存在
                           前驱，返回ERROR
273     {
274         printf ("所要查找的元素是第一个元素，没有前驱\n");
275         return ERROR;
276     }
277     while(p->next)    // 从第二个元素开始往后遍历整个链表，找到要
                       查找的元素e
278     {
279         L = p->next;
280         if (L->data == e )    // 如果找到要查找的元素e，则将该元素的
                               前驱保存在pre中，返回OK
281         {
282             pre = p->data;
283             return OK;
284         }
285         p = L;
286     }
```

```
287     printf ("所要查找的元素不在线性表里面,无法操作\n"); // 如果整个
        链表中都没有找到要查找的元素e, 则返回ERROR, 表示出错
288     return ERROR;
289 }
290
291 // (9) 获得后继: 函数名称是NextElem(L,cur_e,next_e);
292 // 初始条件是线性表L已存在; 操作结果是若cur_e是L的数据元素, 且
        不是最后一个, 则用next_e返回它的后继, 否则操作失败, next_e无
        定义;
293 status NextElem(LinkList L,ElemType e,ElemType &next)
294 // 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。
295 {
296     if (!L) // 如果线性表L不存在
297     {
298         printf ("线性表不存在或未初始化, 无法进行查询\n");
299         return INFEASIBLE;
300     }
301     if (L->next==NULL) // 如果线性表L是空表
302     {
303         printf ("这个线性表是空的\n");
304         return ERROR;
305     }
306     LinkList p = L->next; // 设p为第一个结点, p做前驱结点
307     L = p ->next; // 设L为p的后继结点, L做当前结点
308     while(p) // 如果p不为空
309     {
310         if( !L) // 如果L为空, 表示已经没有后继结点了
311         {
312             printf ("查询不到后继结点\n");
```

```

313         return ERROR;
314     }
315     if(p->data == e)    // 如果p的数据等于给定数据e
316     {
317         next = L->data;    // 将L的数据赋给next
318         return OK;    // 返回操作成功
319     }
320     p = L ;    // 将p移到L的位置，作为新的前驱结点
321     L = p->next;    // 将L移到下一个结点位置，作为新的当前结点
322 }
323 return ERROR; // 如果循环结束时仍未查询到，返回操作失败
324 }
325
326 // (10) 插入元素：函数名称是ListInsert(L,i,e); 初始条件是线性表L已
    存在， $1 \leq i \leq \text{ListLength}(L)+1$ ；操作结果是在L的第i个位置之前插入新
    的数据元素e；
327 status ListInsert (LinkList &L,int i ,int num)
328 {
329     // 先进行特判，如果线性表不存在，返回 INFEASIBLE
330     if(!L)
331     {
332         printf ("线性表不存在或未初始化，无法进行插入\n");
333         return INFEASIBLE;
334     }
335     int e;
336
337     // 用两个指针 p 和 next 分别指向当前遍历到的节点和下一个节点
338     LinkList p = L, next = L->next;
339     int number = 1;    // 来记录当前位置
340
341     // 遍历链表，找到要插入的位置

```

```
342     printf ("请输入元素: \n");
343     while(next)
344     {
345         if(number == i)
346         {
347             // 当找到插入位置时, 使用一个循环插入 num 个元素
348             while (num)
349             {
350                 // 创建新的节点, 获取用户输入的数据
351                 LinkList insert = (LinkList)malloc( sizeof(LNode));
352                 scanf("%d",&e);
353                 insert->data = e;
354
355                 // 修改链表指针指向, 完成插入操作
356                 p->next = insert ;
357                 insert->next = next;
358                 p = insert ;
359                 num--;
360             }
361
362             return OK;
363         }
364
365         // 继续向下遍历
366         number++;
367         p = next;
368         next = p->next;
369     }
370
371     // 如果插入位置为最后一个位置, 则在链表尾部插入
372     if( number == i)
```

```

373     {
374         LinkList insert ;
375
376         // 使用循环将 num 个数据插入到尾部
377         while (num) {
378             scanf("%d",&e);
379             insert = (LinkList)malloc( sizeof(LNode));
380             insert->data = e;
381             insert->next = NULL;
382             p->next = insert ;
383             p = insert ;
384             num--;
385         }
386         return OK;
387     }
388
389     // 如果插入位置不正确，返回错误
390     printf("插入的位置不对\n");
391     return ERROR;
392 }
393
394 // (11) 删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已
    存在且非空， $1 \leq i \leq \text{ListLength}(L)$ ；操作结果：删除L的第i个数据元
    素，用e返回其值；
395 status ListDelete (LinkList &L,int i,ElemType &e) // 删除线性表L的第
    i个元素，并保存在e中，返回OK或ERROR或INFEASIBLE
396 {
397     if(!L) // 如果线性表L不存在，返回INFEASIBLE
398     {
399         printf("线性表不存在或未初始化，无法进行清空\n");
400         return INFEASIBLE;

```

```
401     }
402     int number = 0; // 用于记数，记录当前扫描到的元素的位置
403
404     LinkList pre = L, next = L->next; // pre用于记录当前扫描到的元
        素的前一个元素，next用于记录当前扫描到的元素
405
406     while(next) // 遍历线性表，直到到达表尾
407     {
408         number++; // 计数器加1，记录当前扫描到的元素的位置
409
410         if(number == i) // 如果找到第i个元素
411         {
412             e = next->data; // 将该元素的值保存在e中
413             pre->next = next->next; // 将当前元素的前一个元素的指
                针指向当前元素的后一个元素，实现删除操作
414             free(next); // 释放内存
415             return OK; // 返回执行成功
416         }
417         pre = next; // 当前元素保存到前一个元素变量pre中
418         next = pre->next; // 后一个元素保存到当前元素变量next中，
            实现遍历
419     }
420     printf("想要删除的位置存在问题\n"); // 如果遍历到表尾仍未找到
        第i个元素，则输出提示
421     return ERROR; // 返回执行失败
422 }
423
424 // (12) 遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表
        L已存在；
425 // 操作结果是依次对L的每个数据元素调用函数visit()。
426 status ListTraverse(LinkList L,void (*vi)(int ))
```

```
427 // 遍历线性表 L 中的元素，依次使用函数指针 vi 处理每个元素。
428 // 如果线性表 L 不存在，返回 INFEASIBLE，否则返回 OK。
429 {
430     if(!L) // 如果线性表 L 不存在
431     {
432         printf("线性表不存在或未初始化，无法进行操作\n");
433         return INFEASIBLE; // 返回 INFEASIBLE
434     }
435     LinkList p = L->next; // 从 L 中第一个元素开始遍历
436     while(p) // 只要当前节点不是尾节点
437     {
438         vi(p->data); // 对当前节点的元素使用函数指针 vi 进行处理
439         p = p->next; // 指向下一个节点
440         if(p) // 如果当前不是最后一个节点
441         {
442             putchar(' '); // 输出一个空格，与下一个元素分隔开来
443         }
444     }
445     return OK; // 遍历结束，返回 OK
446 }
447
448 // 在 Lists 中增加一个名称为 ListName 的空线性表
449 // Lists：线性表集合，包含多个线性表
450 // ListName: 待添加的线性表名称
451 // 返回值：操作状态，成功为 OK，否则为 INFEASIBLE
452 status AddList(LISTS &Lists, char ListName[])
453 {
454     // 循环查找是否已经存在同名线性表
455     for(int i = 0; i < Lists.length; i++)
456     {
457         if(strcmp(ListName, Lists.elem[i].name) == 0) // 判断线性表名
```

```
称是否相同
458     {
459         printf ("这个名字的线性表已经存在了"); // 输出提示信息
460         return INFEASIBLE; // 返回INFEASIBLE表示操作失败
461     }
462 }
463 // 未找到同名线性表，可以继续添加
464 Lists.length++; // 线性表集合长度+1
465 int n = 0;
466 // 将新线性表的名称和数据初始化
467 strcpy ( Lists.elem[ Lists.length-1].name, ListName); // 将线性表名
称赋值
468 Lists.elem[ Lists.length-1].L = NULL; // 将存储数据的指针初始化
为NULL
469 return OK; // 返回OK表示操作成功
470 }
471
472 status RemoveList(LISTS &Lists, char ListName[])
473 // Lists 中删除一个名称为ListName的线性表
474 {
475     // 遍历线性表数组找到需要删除的线性表
476     for ( int k = 0; k < Lists.length; k++)
477     {
478         if (strcmp( Lists.elem[k].name, ListName) == 0) // 逐一判断线性
表名称是否与要删除的名称相同
479         {
480             DestroyList ( Lists.elem[k].L); // 先销毁这个线性表本身
的空间
481             // 指针和名称逐一向前移动
482             for ( int i = k; i < Lists.length-1; i++)
483             {
```



```
484         strcpy ( Lists .elem[ i ], Lists .elem[ i +1].name); //
           逐一将后面的线性表的名称复制到前面
485         Lists .elem[ i ].L =Lists .elem[ i +1].L; // 将后面线性
           表的指针复制到前面
486     }
487     // 线性表数组的长度减 1
488     Lists .length--;
489     return OK; // 删除成功
490 }
491 }
492 return ERROR; // 没有找到要删除的线性表，删除失败
493 }
494
495 int LocateList (LISTS Lists ,char ListName[])
496 // 查找一个名称为ListName的线性表在Lists中的位置，成功返回逻辑序
   号，否则返回-1
497 {
498     // 开始遍历线性表
499     for( int k = 0 ;k< Lists .length;k++)
500     {
501         // 比较线性表名称是否匹配
502         if (strcmp( Lists .elem[k].name,ListName)==0)
503         {
504             return k+1; // 返回序号（序号从 1 开始）
505         }
506     }
507     return -1; // 查找失败，返回 -1
508 }
509
510 void SearchList (LISTS Lists) // 这个函数负责展示已经创建的线性表
511 {
```

```
512     int i =0;
513     printf("已经存在的线性表有: \n");
514     for( ;i<Lists.length;i++)
515     {
516         printf("序号 %d) 线性表的名称:%s\n",(i+1),Lists.elem[i].name
517             );
518     }
519 }
520
521 // 翻转线性表
522 void reverseList (LinkedList L)
523 {
524     // 判断线性表是否存在或未初始化
525     if (!L)
526     {
527         printf("线性表不存在或未初始化\n");
528         return ;
529     }
530     // 获取线性表的长度
531     int len = ListLength(L);
532     // 申请一个长度为 len 的数组
533     int *arr =(int *) malloc( sizeof( int ) *len);
534     // 遍历链表，存储链表中的元素到数组中
535     LinkedList p1 = L->next;
536     for( int k=0;k<len;k++)
537     {
538         arr[k] = p1->data;
539         p1 = p1->next;
540     }
541     // 从尾到头遍历链表，将数组中的元素依次存储到链表中
```

```
542     LinkedList p2 = L->next;
543     for( int i =len; i>0; i--)
544     {
545         p2->data = arr[i-1];
546         p2 = p2->next;
547     }
548     printf("成功翻转了\n");
549     return ;
550 }
551
552 void RemoveNthFromEnd(LinkedList L,int n) // RemoveNthFromEnd函数的
    定义，参数为一个单链表和要删除的节点位置
553 {
554     if(!L) // 如果单链表为空或未初始化
555     {
556         printf("线性表不存在或着未初始化\n"); // 打印错误信息
557         return ; // 退出函数
558     }
559
560     int len = ListLength(L); // 获取单链表的长度
561     int e; // 用来存储被删除节点的数据元素
562     int feedback; // 存储ListDelete函数的返回值，即删除操作的结果
563
564     feedback = ListDelete (L,len-n+1,e); // 调用ListDelete函数删除第(
        len-n+1)个节点，并将被删除节点的值存入e中
565
566     if(feedback == OK) // 如果删除成功
567     {
568         printf("成功删除，删除的元素是 :%d\n",e); // 打印成功信息及
            被删除节点的值
569     }
```

```
570 }
571 // 排序
572 void sortList (LinkedList L)
573 {
574     // 判断线性表是否存在或者未初始化
575     if (!L)
576     {
577         printf ("线性表不存在或者未初始化\n");
578         return ;
579     }
580     // 获取线性表长度
581     int len = ListLength(L);
582     // 申请动态内存，用于存放线性表数据
583     int *arr =(int *) malloc( sizeof( int)*len);
584     // 遍历线性表，将数据存放到arr数组中
585     LinkedList p1 = L->next;
586     for( int k=0;k<len;k++)
587     {
588         arr[k] = p1->data;
589         p1 = p1->next;
590     }
591     // 对arr数组进行冒泡排序
592     for( int k=0;k<len-1;k++)
593     {
594         for( int i=0;i<len-1-k;i++)
595         {
596             if( arr[i] > arr[i+1])
597             {
598                 int tmp = arr[i]; // 中间变量tmp
599                 arr[i] = arr[i+1];
600                 arr[i+1] = tmp;
```

```
601     }
602 }
603 }
604 // 将排序后的数据写回线性表中
605 LinkList p2 = L->next;
606 for( int k =0;k<len;k++)
607 {
608     p2->data = arr[k];
609     p2 = p2->next;
610 }
611 }
612
613 // 从线性表L中将数据保存到文件name中
614 void savetofile (LinkList L,char name[])
615 {
616     if(!L) // 如果L不存在或未初始化, 无法进行操作
617     {
618         printf("线性表不存在或未初始化\n");
619         return ;
620     }
621     FILE *fp = fopen(name,"w"); // 打开文件, 以写的方式
622     if(fp == NULL) // 如果无法找到文件, 报错
623     {
624         printf("打开文件失败\n");
625         return ;
626     }
627     LinkList current = L->next; // 指向第一个节点
628     while ( current != NULL) // 循环遍历线性表中的每个节点
629     {
630         fprintf (fp, "%d ", current->data); // 将节点的数据写入文件
        中
```

```
631     current = current->next;
632 }
633 fclose(fp); // 关闭文件
634 printf("成功保存到文件了\n"); // 提示信息，表明操作成功
635 return ;
636 }
637
638 // 从文件name中读取数据，保存到线性表L中
639 void getfromfile (LinkedList L,char name[])
640 {
641     if(L->next) // 如果L不是空的，说明已经有数据了，无法进行操作
642     {
643         printf("这不是一个空的线性表，读取数据会导致原来的数据被
        覆盖，无法操作\n");
644         return ;
645     }
646     FILE *fp = fopen(name,"r"); // 打开文件，以读的方式
647     if(fp == NULL) // 如果无法找到文件，报错
648     {
649         printf("打开文件失败\n");
650         return ;
651     }
652     LinkedList p = L; // 用于遍历线性表
653     LinkedList insert = (LinkedList) malloc(sizeof(LNode)); // 动态分配内存，用于存放从文件中读取的数据
654     while ( fscanf(fp,"%d",&insert->data) != EOF) // 循环读取文件中的数据
655     {
656         p->next = insert ; // 插入到线性表中
657         p = insert ; // 指向刚插入的节点
658         p->next = NULL; // 该节点的下一个节点为空
```

```
659         insert = (LinkedList) malloc(sizeof(LNode)); // 为下一个节点动态分配内存
660     }
661     fclose(fp); // 关闭文件
662     return ;
663 }
664
665 void fun01() // 定义一个函数
666 {
667     menu(); // 调用菜单
668     int a; // 定义一个整型变量
669     printf("请输入一个命令\n"); // 输出提示信息
670     scanf("%d", &a); // 读取一个整型变量
671
672     // 通过while循环来进行命令处理
673     while (a) // 当a不为0时执行循环体
674     {
675         fflush ( stdin ); // 清空输入流，防止上一次操作影响本次操作
676
677         switch (a) {
678             // 如果a等于1，执行以下代码
679             case 1:
680                 printf("现在进行创建线性表\n"); // 输出提示信息
681                 printf("请输入你想创建的线性表的名字\n"); // 输出提示信息
682                 char name1[30]; // 定义一个名字为name1且长度最大为30的字符数组
683                 scanf("%s", name1); // 读取字符串
684
685                 int u; // 定义一个整型变量
686                 u = AddList( Lists , name1); // 向一个线性表数组中插入
```

```
        一个新的空线性表
687         if(u == OK) //如果插入成功
688         {
689             printf("创建成功啦\n"); // 输出提示信息
690         }
691         if(u == INFEASIBLE) //如果插入失败
692         {
693             system("pause"); // 暂停程序运行
694         }
695         break;
696
697         // 如果a等于2, 执行以下代码
698     case 2:
699         printf("现在进行删除线性表\n"); //输出提示信息
700         printf("请输入你想创建的线性表的名字\n"); //输出提示
            信息
701         char name2[30]; // 定义一个名字为name2且长度最大为
            30的字符数组
702         scanf("%s", name2); // 读取字符串
703         RemoveList(Lists, name2); // 从线性表数组中删除指定
            的线性表
704         break;
705
706         // 如果a等于3, 执行以下代码
707     case 3:
708         printf("现在进行查询创建了哪些线性表\n"); //输出提示
            信息
709         SearchList( Lists ); // 查询线性表数组中所有线性表的
            名字并输出
710         break;
711
```



```
712         // 如果a等于4, 执行以下代码
713         case 4:
714             printf("现在进行线性表的查找和操作\n"); // 输出提示信息
715             printf("请输入你想查找和操作的线性表的名字\n"); // 输出提示信息
716             char name3[30]; // 定义一个名字为name3且长度最大为30的字符数组
717             scanf("%s", name3); // 读取字符串
718             int judge; // 定义一个整型变量
719             judge=LocateList( Lists , name3); // 查找线性表数组中指定名字的线性表, 并返回其下标
720
721             if(judge ==-1) // 如果查找失败
722             {
723                 printf("不存在这个线性表\n"); // 输出提示信息
724                 system("pause"); // 暂停程序运行
725             }
726             else { // 如果查找成功
727                 fun02( Lists .elem[judge-1].L); // 执行另一个函数
728             }
729             break;
730
731         // 如果a不等于1、2、3、4, 执行以下代码
732         default :
733             printf("输入的命令错误, 请再次输入"); // 输出提示信息
734             }
735
736         printf("请输入下一个命令\n"); // 输出提示信息
737         scanf("%d", &a); // 读取一个整型变量
```

```
738         system("cls"); // 清屏
739         menu(); // 再次调用另一个函数
740     }
741 }
742
743 void fun02(LinkList &L) // 这个函数进行每个线性表的详细功能实现
744 {
745     system("cls"); // 清空命令行窗口
746     printf("线性表存在鸭鸭\n");
747     printf("现在对这个线性表进行操作\n");
748     printf("别忘记初始化这个线性表鸭\n");
749
750     // 接下来会进行各种操作，要求用户输入命令
751     int order;
752     show_normal(); // 显示命令列表
753     scanf("%d",&order); // 读取用户输入的命令序号
754     while (order) // 如果用户输入的命令不是0（退出），就继续循环
755     {
756         fflush ( stdin ); // 清空输入缓冲区，防止影响下一次输入
757         int feedback; // 来接收函数返回值
758         switch (order) { // 根据命令序号进行相应的操作
759             case 1:
760                 // 创建线性表
761                 if ( InitList (L) == OK)
762                     printf("线性表创建成功! \n");
763                 else printf("线性表已经存在，创建失败! \n");
764                 break;
765             case 2:
766                 // 销毁线性表
767                 printf("现在进行线性表的销毁\n");
```

```
768         feedback = DestroyList(L);
769         if(feedback == OK)
770         {
771             printf("成功销毁了\n");
772         }
773         else if(feedback == INFEASIBLE)
774         {
775             printf("这个线性表不存在或未初始化,无法销毁\n")
776             ;
777         }
778         break;
779     case 3:
780         // 清空线性表
781         printf("现在进行线性表的清空操作\n");
782         feedback = ClearList(L);
783         if(feedback == OK)
784         {
785             printf("成功清空线性表\n");
786         }
787         break;
788     case 4:
789         // 判断线性表是否为空
790         printf("现在对线性表进行判空操作\n");
791         ListEmpty(L);
792         break;
793     case 5:
794         // 求线性表的长度
795         printf("现在进行求线性表的长度\n");
796         feedback = ListLength(L);
797         if(feedback != INFEASIBLE)
798         {
```

```
798         printf("线性表的长度为:%d",feedback);
799     }
800     break;
801 case 6:
802     // 获取线性表中指定位置的元素
803     printf("现在进行元素获取操作\n");
804     int e; int i;        // 用e来接收元素的值, i是所要获取
                        // 元素的位置
805     printf("请输入你想获取第几个元素的值\n");
806     scanf("%d",&i);
807     feedback = GetElem(L,i,e);
808     if(feedback == OK)
809     {
810         printf("成功获取, 第%d个元素的值为: %d",i,e);
811     }
812     break;
813 case 7:
814     // 查找线性表中指定值的元素
815     printf("现在进行查找元素的操作\n");
816     int ee; // 接收所要查找的元素
817     printf("请输入你想查找的元素\n");
818     scanf("%d",&ee);
819     feedback = LocateElem(L,ee,compare);
820     if(feedback > 0)
821     {
822         printf("所要查找的元素是第%d个\n",feedback);
823     }
824     break;
825 case 8:
826     // 查找指定元素的前驱
827     printf("现在进行查找前驱的操作\n");
```

```
828         int cur_e,pre_e;    //接收元素，并存储前驱
829         printf("请输入你想查找哪个元素的前驱\n");
830         scanf("%d",&cur_e);
831         feedback = PriorElem(L,cur_e,pre_e);
832         if(feedback == OK)
833         {
834             printf("你所要查找的前驱是: %d\n",pre_e);
835         }
836         break;
837     case 9:
838         // 查找指定元素的后驱
839         printf("现在进行查找后驱的操作\n");
840         int cur,next_e;    //接收元素，并存储后继
841         printf("请输入你想查找哪个元素的后驱\n");
842         scanf("%d",&cur);
843         feedback = NextElem(L,cur,next_e);
844         if(feedback == OK)
845         {
846             printf("你所要查找的元素的后驱是: %d\n",next_e);
847         }
848         break;
849     case 10:
850         // 在指定位置插入元素
851         printf("现在进行插入元素的操作\n");
852         printf("请问你想在第几个位置插入元素\n");
853         int position;    //接收元素的位置
854         scanf("%d",&position);
855         printf("请问你想插入元素的个数\n");
856         int number;
857         scanf("%d",&number);
858         feedback = ListInsert (L, position ,number);
```

```
859         if(feedback == OK)
860         {
861             printf("插入成功\n");
862         }
863         break;
864     case 11:
865         // 删除指定位置的元素
866         printf("现在进行删除元素的操作\n");
867         printf("请问你想删除第几个位置的元素\n");
868         int position01 ;
869         int e1;
870         scanf("%d",&position01);
871         feedback = ListDelete (L,position01 ,e1);
872         if(feedback == OK)
873         {
874             printf("删除成功，删除的元素是: %d\n",e1);
875         }
876         break;
877     case 12:
878         // 遍历线性表
879         printf("现在进行线性表的遍历\n");
880         ListTraverse (L, visit );
881         break;
882     case 13:
883         // 翻转线性表
884         printf("现在进行线性表的翻转\n");
885         reverseList (L);
886         break;
887     case 14:
888         // 删除倒数第n个元素
889         printf("现在进行删除链表倒数元素的操作\n");
```

```
890         printf ("你想删除链表倒数第几个节点\n");
891         int positon2;
892         scanf ("%d",&positon2);
893         RemoveNthFromEnd(L,positon2);
894         break;
895     case 15:
896         // 排序线性表
897         printf ("现在进行链表的排序\n");
898         sortList (L);
899         printf ("搞定力\n");
900         break;
901     case 16:
902         // 将线性表保存到文件
903         printf ("现在进行线性表的文件保存\n");
904         printf ("请问你想保存到哪一个文件\n");
905         char name1[30];
906         scanf ("%s",name1);
907         savetofile (L,name1);
908         break;
909     case 17:
910         // 从文件中读取线性表
911         printf ("现在进行线性表的读取操作\n");
912         printf ("你想读取哪一个文件的资料\n");
913         char name2[30];
914         scanf ("%s",name2);
915         getfromfile (L,name2);
916         break;
917     default :
918         // 输入错误命令序号
919         printf ("命令输入有问题\n");
920 }
```

```
921     putchar('\n');
922     printf("请输入下一个命令\n");
923     scanf("%d",&order);
924     system("cls"); // 清空命令行窗口
925     if (order != 0)
926     {
927         show_normal(); // 显示命令列表
928     }
929     else {
930         menu(); // 返回主菜单
931     }
932
933 }
934 }
935
936 void show_normal() // 单个线性表的菜单
937 {
938     // 输出横线
939     for (int k = 0; k <= 119 ;k++)
940     {
941         putchar('-');
942     }
943     putchar('\n');
944
945     // 输出菜单标题
946     printf("          Menu for Linear Table On Sequence Structure \n"
947
948
949     printf("          1. InitList          7.
          LocateElem\n");
```


950	printf ("	2. DestroyList	8.
	PriorElem\n");		
951	printf ("	3. ClearList	9.
	NextElem \n");		
952	printf ("	4. ListEmpty	10.
	ListInsert \n");		
953	printf ("	5. ListLength	11.
	ListDelete \n");		
954	printf ("	6. GetElem	12.
	ListTraverse \n");		
955			
956	// 输出横线		
957	for(int k = 0; k<= 119 ;k++)		
958	{		
959	putchar('-');		
960	}		
961			
962	putchar('\n');		
963			
964	// 输出额外的菜单选项		
965	printf ("	13. reverseList	14.
	RemoveNthFromEnd\n");		
966	printf ("	15. sortList	16.
	saveFile \n");		
967	printf ("	17. getFile	\n");
968	printf ("	0. exit	\n");
969			
970	// 输出横线		
971	for(int k = 0; k<= 119 ;k++)		
972	{		
973	putchar('-');		

```

974     }
975
976     putchar('\n');
977
978     // 输出小猫咪
979     printf("    ^           /\n");
980     printf("    /\ 7       □_ ^\n");
981     printf("    / |       / /\n");
982     printf("    | Z_,<    /    /['\n");
983     printf("    |           ['    /    > \n");
984     printf("    Y           '    /    ^\n");
985     printf("    ?● ? ●    ?? <    ^\n");
986     printf("    () ^       | \ \ <\n");
987     printf("    >??_ 彳    | / /\n");
988     printf("    / ^       / ?<| \ \ \n");
989     printf("    ['?    (/_    | / /\n");
990     printf("    7           | / \n");
991     printf("    >—r - - '?— _\n");
992
993     putchar('\n');
994
995     // 输出横线
996     for(int k = 0; k<= 119 ;k++)
997     {
998         putchar('-');
999     }
1000
1001     putchar('\n');
1002
1003     // 输出提示文字
1004     printf("    请选择你的操作[0~13]:");

```

```
1005
1006     putchar('\n');
1007
1008     // 输出横线
1009     for( int k = 0; k<= 119 ;k++)
1010     {
1011         putchar('-');
1012     }
1013
1014     putchar('\n');
1015 }
1016
1017 void menu()
1018 {
1019     for( int k = 0; k<= 119 ;k++) // 打印分隔线，共120个 '-'
1020     {
1021         putchar('-');
1022     }
1023     putchar('\n'); // 打印换行符
1024
1025     printf("1.创建一个线性表\n"); // 打印操作1的描述
1026     printf("2.删除一个线性表\n"); // 打印操作2的描述
1027     printf("3.查询已经创建的线性表\n"); // 打印操作3的描述
1028     printf("4.查找一个线性表和进行操作\n"); // 打印操作4的描述
1029     printf("0.退出线性表的管理\n"); // 打印操作0的描述
1030
1031     // 打印一只猫的 ASCII Art，增加菜单的趣味性
1032     printf("    ^                /\n");
1033     printf("    /\ 7          □ _ ^\n");
1034     printf("    /  |          /  /\n");
1035     printf("    |  Z _,<    /    /'F\n");
```

```

1036     printf(" |          [F]      /      > \n");
1037     printf(" Y          ' /      ^\n");
1038     printf(" ?● ? ●      ?? <      ^\n");
1039     printf(" () ^          | \ <\n");
1040     printf(" >??_   ι      | / / \n");
1041     printf(" / ^      / ?<| \ \ \n");
1042     printf(" [F]?      ( /      | / / \n");
1043     printf(" 7          | / \n");
1044     printf(" >—r - - '?—_ \n");
1045
1046     for( int k = 0; k<= 119 ;k++) // 其他同上，打印分隔线
1047     {
1048         putchar('—');
1049     }
1050     putchar('\n'); // 最后再打印一行空行，方便视觉上的分离
1051 }
1052
1053 // 下方是一个插入操作的菜单，类似于前面的 menu()函数
1054 void Menuofinsert()
1055 {
1056     for( int k = 0; k<= 119 ;k++)
1057     {
1058         putchar('—');
1059     }
1060     putchar('\n');
1061
1062     printf("1.插入多个元素\n");
1063     printf("2.定点插入元素\n");
1064
1065     printf(" ^          / \n");
1066     printf(" /\ 7      □ _ \n");

```

```

1067     printf ("  / |      / / \n");
1068     printf (" | Z _,<  /      /['F\n");
1069     printf (" |          [F      /      > \n");
1070     printf (" Y          ' /      ^\n");
1071     printf (" ?● ? ●      ?? <      ^\n");
1072     printf (" () ^      | \ <\n");
1073     printf (" >? ?_  ı      | / / \n");
1074     printf (" / ^      / ?<| \ \ \n");
1075     printf (" [F_?      ( /      | / / \n");
1076     printf (" 7          | / \n");
1077     printf (" >—r - - '?— _ \n");

```

```

1078
1079     for( int k = 0; k<= 119 ;k++)
1080     {
1081         putchar('—');
1082     }
1083     putchar('\n');
1084 }

```

```

1085
1086 int compare(int a, int b)
1087 {
1088     if(a == b)
1089     {
1090         return 1;
1091     }
1092     return 0;
1093 }

```

```

1094
1095 void visit ( int x)
1096 {
1097     printf ("%d",x);

```

1098 }

附录 C 基于二叉链表二叉树实现的源程序

```
1  /*—— 头文件的申明 ——*/
2  #include<stdio .h>
3  #include<stdlib .h>
4  #include "string .h"
5
6  /*—— 预定义 ——*/
7  // 定义布尔类型TRUE和FALSE
8  #define TRUE 1
9  #define FALSE 0
10
11 // 定义函数返回值类型
12 #define OK 1
13 #define ERROR 0
14 #define INFEASIBLE -1
15 #define OVERFLOW -2
16
17 // 定义数据元素类型
18 typedef int status ;
19 typedef int KeyType;
20
21 // 二叉树结点数据类型定义
22 typedef struct {
23     KeyType key; // 结点关键字
24     char others [20]; // 数据
25 } TElemType;
26
27 // 二叉链表结点的定义
```

```

28 typedef struct BiTNode {
29     TElemType data; // 结点数据
30     struct BiTNode *lchild, *rchild; // 左右子树指针
31 } BiTNode, *BiTree;
32
33 // 二叉树的集合类型定义
34 typedef struct {
35     // 元素的集合
36     struct {
37         char name[30]; // 标识元素的名称
38         BiTree T; // 二叉树
39     } elem[10]; // 最多存储 10 个元素
40     int length; // 元素个数
41 } LIST;
42
43 LIST Lists; // 二叉树集合的定义 Lists
44
45
46 /*----- 函数申明 -----*/
47 status CreateBiTree(BiTree &T, TElemType definition []); // 创建
48 status DestroyBiTree(BiTree &T); // 销毁
49 status ClearBiTree(BiTree &T); // 清空
50 status BiTreeEmpty(BiTree &T); // 判空
51 int BiTreeDepth(BiTree T); // 求深度
52 BiTNode* LocateNode(BiTree T, KeyType e); // 查找结点
53 status Assign(BiTree &T, KeyType e, TElemType value); // 结点赋值
54 BiTNode* GetSibling(BiTree T, KeyType e); // 获得兄弟结点
55 status InsertNode(BiTree &T, KeyType e, int LR, TElemType c); // 插入
    结点
56 BiTree findrightTNode(BiTree T); // 找到右子树
57 status DeleteNode(BiTree &T, KeyType e); // 删除结点
    
```

```

58 void visit (BiTree T); //遍历中调用的访问函数
59 status PreOrderTraverse(BiTree T, void(* visit )(BiTree)); // 前序遍历
60 status InOrderTraverse(BiTree T, void(* visit )(BiTree)); // 中序遍历
61 status PostOrderTraverse(BiTree T, void(* visit )(BiTree)); // 后续遍历
62 status LevelOrderTraverse(BiTree T, void(* visit )(BiTree)); // 层序遍
    历
63 status SaveBiTree(BiTree T, char FileName[]); // 保存到文件
64 status LoadBiTree(BiTree &T, char FileName[]); // 从文件中加载
65 int MaxPathSum(BiTree T); // 最大路径和
66 BiTree LowestCommonAncestor(BiTree T, int e1, int e2); // 最近公共祖
    先
67 BiTree InvertTree (BiTree T); // 翻转二叉树
68 void menufirst (); // 管理多个图的菜单
69 void menu(); // 管理单个图的菜单
70 void fun01(); // 管理多个图的封装函数
71 void fun02(BiTree & T ); // 管理单个图的封装函数
72
73 /*----- main主函数 -----*/
74 int main()
75 {
76     system("color 37"); // 设置颜色
77     fun01();
78 }
79
80
81
82
83 /*----- 函数定义 -----*/
84
85 status CreateBiTree(BiTree &T, TElemType definition [])
86 {

```



```
87  /*
88     * 根据带空枝的二叉树先根遍历序列definition构造一棵二叉树，
89     * 将根节点指针赋值给T并返回OK，如果有相同的关键字，返回
        ERROR。
90     */
91
92     static int i = 0; // 静态变量，记录当前已经处理到的序列下标
93     int j = 0, k = 0; // 循环计数器
94
95     // 第一次调用时，检查数据是否合法
96     if (i == 0) {
97         // 依次检查每个节点关键字是否合法
98         for (j = 0; (definition + j)->key != -1; j++) {
99             for (k = j + 1; (definition + k)->key != -1; k++) {
100                 // 如果有两个关键字相同，且不为0，返回错误
101                 if ((definition + j)->key == (definition + k)->key
102                     && (definition + j)->key != 0)
103                     return ERROR;
104             }
105         }
106
107         // 递归出口：序列遍历结束，T为空值，返回OK
108         if ((definition + i)->key == -1) {
109             T = NULL;
110             i = 0;
111             return OK;
112         }
113
114         // 如果当前节点为0，表示空结点，无需创建二叉树结点，i自增并
            返回OK
```

```
115     if (( definition + i)->key == 0) {
116         i++;
117         return OK;
118     }
119
120     // 创建二叉树结点，分别处理其左右子树，递归构建整棵二叉树
121     T = (BiTree)malloc( sizeof(BiTreeNode));
122     T->lchild = NULL;
123     T->rchild = NULL;
124     T->data = *( definition + i);
125     i++;
126     CreateBiTree(T->lchild, definition );
127     CreateBiTree(T->rchild, definition );
128     return OK;
129 }
130
131 status DestroyBiTree(BiTree &T)
132 { //将二叉树设置成空，并删除所有结点，释放结点空间
133
134     if (T != NULL) // 如果这个二叉树不为空
135     {
136         if (T->lchild) // 如果左子树不为空
137             DestroyBiTree(T->lchild); // 递归地销毁左子树（因为左子
            树也是一棵二叉树）
138
139         if (T->rchild) // 如果右子树不为空
140             DestroyBiTree(T->rchild); // 递归地销毁右子树（因为右子
            树也是一棵二叉树）
141
142         free(T); // 释放当前节点的空间（因为当前节点的左右子树已
            经被销毁了）
```

```
143
144     T = NULL; // 将当前节点的指针设置为NULL，表示这个节点已
        经被销毁了
145 }
146
147     return OK; // 返回操作成功
148 }
149
150 // 初始条件是二叉树 T 存在；操作结果是将二叉树 T 清空
151 status ClearBiTree(BiTree &T)
152 {
153     // 如果二叉树 T 不为空，则需要清空它；否则直接返回 OK
154     if (T != NULL)
155     {
156         // 如果 T 的左子树不为空，则递归清空左子树
157         if (T->lchild)
158             DestroyBiTree(T->lchild);
159
160         // 如果 T 的右子树不为空，则递归清空右子树
161         if (T->rchild)
162             DestroyBiTree(T->rchild);
163
164         // 释放 T 的内存空间，并将 T 的指针设为 NULL
165         free(T); // 使用递归依次释放左子树、右子树、根节点指针
166         T = NULL;
167     }
168
169     // 返回函数执行结果
170     return OK;
171 }
172
```

```
173 status BiTreeEmpty(BiTree &T)
174 {
175     // 初始条件是二叉树T存在；操作结果是若T为空二叉树则返回TRUE
        , 否则返回FALSE
176     if (T != NULL)
177         return FALSE;
178     else
179         return TRUE;
180 }
181
182 int BiTreeDepth(BiTree T)
183 {
184     // 求二叉树T的深度
185
186     int depth = 0; // 定义变量 depth 并初始化为 0
187
188     if (T != NULL)
189     {
190         int lchilddepth , rchilddepth ;
191         // 递归求解左子树的深度
192         lchilddepth = BiTreeDepth(T->lchild);
193         // 递归求解右子树的深度
194         rchilddepth = BiTreeDepth(T->rchild);
195
196         // 取左右子树深度较大值并将其加 1，即为当前节点所在子树的
            深度
197         if ( lchilddepth >= rchilddepth )
198             depth = lchilddepth + 1;
199         else
200             depth = rchilddepth + 1;
201     }
```

```
202
203     return depth;    // 返回当前节点所在子树的深度
204 }
205
206 BiTNode* LocateNode(BiTree T, KeyType e)
207 { // 查找结点
208     if (T == NULL) // 如果树为空，则返回空指针
209         return NULL;
210     BiTree st[100], p; // 定义一个栈 st 和当前遍历的结点 p
211     int top = 0; // top 表示栈顶指针
212     st[top++] = T; // 将根节点入栈
213     while (top != 0) // 当栈不为空时
214     {
215         p = st[--top]; // 取出栈顶元素
216         if (p->data.key == e) // 若当前结点的值等于 e，则返回当前结
            点
217             return p;
218         if (p->rchild != NULL) // 若当前结点的右子树不为空，则将右
            子树入栈
219             st[top++] = p->rchild;
220         if (p->lchild != NULL) // 若当前结点的左子树不为空，则将左
            子树入栈
221             st[top++] = p->lchild;
222     }
223     return NULL; // 如果未找到结点，则返回空指针
224 }
225
226 // 函数功能：给二叉树中某个结点赋值
227 status Assign(BiTree &T, KeyType e, TElemType value)
228 {
229     // 判断二叉树是否存在
```

```
230     if (T == NULL)
231     {
232         printf("二叉树不存在\n");
233         return ERROR;
234     }
235
236     int flag = 0; // 用于标记是否找到了目标结点
237     BiTree st[100], p; // 定义一个数组作为栈，一个指针用于遍历二叉
        树
238     int top = 0; // 栈顶指针
239     st[top++] = T; // 将二叉树根节点入栈
240
241     // 循环遍历二叉树
242     while (top != 0)
243     {
244         p = st[--top]; // 弹出栈顶元素
245
246         // 判断插入的结点关键字是否和二叉树中的其他结点重复
247         if (p->data.key == value.key && e != value.key)
248         {
249             printf("关键字重复\n");
250             return ERROR;
251         }
252
253         // 找到了目标结点
254         if (p->data.key == e)
255         {
256             p->data = value; // 将目标结点的数据修改为新的数据
257             flag = 1; // 标记为已找到
258         }
259
```

```

260         //遍历左右子树
261         if (p->rchild != NULL)
262             st[top++] = p->rchild; // 右子树入栈
263         if (p->lchild != NULL)
264             st[top++] = p->lchild; // 左子树入栈
265     }
266
267     // 判断是否成功修改了结点数据
268     if(flag)
269     {
270         return OK;
271     }
272     return ERROR;
273 }
274
275 status InsertNode(BiTree &T, KeyType e, int LR, TElemType c)
276 // 参数说明:
277 // T: 待插入结点的二叉树
278 // e: 待插入结点的父节点关键字
279 // LR: 待插入结点的左孩子还是右孩子, 当LR为-1时作为根结点插入
280 // c: 待插入结点
281 {
282     BiTree t; // 定义一个二叉树结点t
283     int top = 0; // 栈顶指针初始化为0, 这个变量好像没用到
284     if (LR == -1) // 如果待插入结点要插入为根结点
285     {
286         t = (BiTree)malloc(sizeof(BiTNode)); // 动态分配内存并定义
           为二叉树结点
287         t->rchild = T; // 将原有的二叉树T挂在新结点的右子树上
288         t->lchild = NULL; // 新结点的左子树为空
289         t->data = c; // 新结点的数据域为待插入数据c

```

```

290     T = t; // 原有的二叉树T被替换为新结点t
291     return OK; // 插入成功
292 }
293 if (T == NULL)
294     return ERROR; // 如果待插入的二叉树为空，则返回错误
295 BiTree q = (BiTree)malloc(sizeof(BiTNode)); // 动态分配内存并定
        义为二叉树结点
296 q->lchild = q->rchild = NULL; // 左右子树均为空
297 q->data = c; // 新结点的数据域为待插入数据c
298 if(LocateNode(T,c.key) != NULL) // 如果新结点的关键字已经存在
        于T中，则返回错误
299 {
300     printf("关键字重复\n");
301     return ERROR;
302 }
303 BiTree p = LocateNode(T, e); // 定位待插入结点的父结点
304 if (!p) // 如果找不到父结点，则返回错误
305     return ERROR;
306 else {
307     if (LR) { // 如果要插入的结点为父节点的右孩子
308         q->rchild = p->rchild; // 将父节点的右子树挂在新结点的
            右子树上
309         p->rchild = q; // 将新结点挂在父节点的右子树上
310         return OK; // 插入成功
311     }
312     if (!LR) { // 如果要插入的结点为父节点的左孩子
313         q->rchild = p->lchild; // 将父节点的左子树挂在新结点的
            右子树上
314         p->lchild = q; // 将新结点挂在父节点的左子树上
315         return OK; // 插入成功
316     }

```



```
317     }
318 }
319
320 // 找到二叉树中最右边的结点
321 BiTree findrightTNode(BiTree T)
322 {
323     // 创建一个栈来保存已遍历的结点
324     BiTree stack[1000], p = NULL;
325     // 定义栈的顶部指针为0
326     int top = 0;
327     // 如果二叉树不为空
328     if (T != NULL)
329     {
330         // 把二叉树的根节点放入栈中
331         stack[top++] = T;
332         // 循环，直到栈为空
333         while (top)
334         {
335             // 取出栈顶元素，并把该元素赋给p
336             p = stack[--top];
337             // 将p的右子树放入栈中
338             if (p->rchild != NULL)
339                 stack[top++] = p->rchild;
340             // 将p的左子树放入栈中
341             if (p->lchild != NULL)
342                 stack[top++] = p->lchild;
343         }
344     }
345     // 返回最后一个遍历到的结点
346     return p;
347 }
```

```
348
349 // 删除结点
350 status DeleteNode(BiTree &T, KeyType e)
351 {
352     if (T == NULL) //如果T是空树，就无法删除，返回错误代码
353         return INFEASIBLE;
354
355     BiTree stack[1000], p, lp, TNode; //定义栈，以及三个指针
356     int top = 0; // 栈顶
357
358     if (T != NULL) //如果T不为空树
359     {
360         if (T->data.key == e) // 如果T的根结点就是要删除的结点
361         {
362             lp = T; // 记下要删除的结点
363             if (T->lchild == NULL && T->rchild == NULL) //如果要删除的结点没有左右子树
364             {
365                 free(lp); // 直接释放内存
366                 T = NULL; //将根结点设为NULL
367                 return OK; //返回成功操作
368             }
369             else if (T->lchild != NULL && T->rchild == NULL) //如果要删除的结点只有左子树
370             {
371                 T = T->lchild; // 将根结点的左子树变成新的根结点
372                 free(lp); // 释放内存
373                 return OK; //返回成功操作
374             }
375             else if (lp->lchild == NULL && lp->rchild != NULL) //如果要删除的结点只有右子树
```

```
376         {
377             T = T->rchild; // 将根结点的右子树变成新的根结点
378             free(lp); // 释放内存
379             return OK; // 返回成功操作
380         }
381     else // 如果要删除的结点既有左子树又有右子树
382     {
383         TNode = findrightTNode(T->lchild); // 找到要删除的结
            点的左子树的最右结点
384         TNode->rchild = T->rchild; // 将要删除的结点的右子树
            挂在左子树的最右结点下
385         T = T->lchild; // 将左子树作为新的树
386         free(lp); // 释放内存
387         return OK; // 返回成功操作
388     }
389 }
390 stack[top++] = T; // 根结点入栈
391 while (top) // 如果栈还有元素
392 {
393     p = stack[--top]; // 栈顶元素出栈
394     if (p->rchild != NULL) // 如果栈顶元素有右子树
395     {
396         if (p->rchild->data.key == e) // 如果栈顶元素的右子树
            就是要删除的结点
397         {
398             lp = p->rchild; // 记下要删除的结点
399             if (lp->lchild == NULL && lp->rchild == NULL) //
                如果要删除的结点没有左右子树
400             {
401                 free(lp); // 直接释放内存
402                 p->rchild = NULL; // 将父结点的右子树设为
```

```

NULL
403         return OK; //返回成功操作
404     }
405     else if (lp->lchild != NULL && lp->rchild ==
406             NULL) //如果要删除的结点只有左子树
407     {
408         p->rchild = lp->lchild;
409         free(lp);
410         return OK; //返回成功操作
411     }
412     else if (lp->lchild == NULL && lp->rchild !=
413             NULL) //如果要删除的结点只有右子树
414     {
415         p->rchild = lp->rchild;
416         free(lp);
417         return OK; //返回成功操作
418     }
419     else //如果要删除的结点既有左子树又有右子树
420     {
421         TNode = findrightTNode(lp->lchild); //找到要
422         删除的结点的左子树的最右结点
423         TNode->rchild = lp->rchild; //将要删除的结点
424         的右子树挂在左子树的最右结点下
425         p->rchild = lp->lchild; //将左子树作为父结点
426         的右子树
427         free(lp);
428         return OK; //返回成功操作
429     }
430 }
431 stack[top++] = p->rchild; //右子树入栈
432 }
```

```
428         if (p->lchild != NULL) //如果栈顶元素有左子树
429         {
430             if (p->lchild->data.key == e) //如果栈顶元素的左子树
                就是要删除的结点
431             {
432                 lp = p->lchild; //记下要删除的结点
433                 if (lp->lchild == NULL && lp->rchild == NULL) //
                    如果要删除的结点没有左右子树
434                 {
435                     free(lp); //直接释放内存
436                     p->lchild = NULL; //将父结点的左子树设为
                        NULL
437                     return OK; //返回成功操作
438                 }
439                 else if (lp->lchild != NULL && lp->rchild ==
                    NULL) //如果要删除的结点只有左子树
440                 {
441                     p->lchild = lp->lchild;
442                     free(lp);
443                     return OK; //返回成功操作
444                 }
445                 else if (lp->lchild == NULL && lp->rchild !=
                    NULL) //如果要删除的结点只有右子树
446                 {
447                     p->lchild = lp->rchild;
448                     free(lp);
449                     return OK; //返回成功操作
450                 }
451                 else //如果要删除的结点既有左子树又有右子树
452                 {
453                     TNode = findrightTNode(lp->lchild); //找到要
```

```

                                删除的结点的左子树的最右结点
454         TNode->rchild = lp->rchild; // 将要删除的结点的
                                右子树挂在左子树的最右结点下
455         p->lchild = lp->lchild; // 将左子树作为父结点的
                                左子树
456         free(lp);
457         return OK; // 返回成功操作
458     }
459 }
460     stack[top++] = p->lchild; // 左子树入栈
461 }
462 }
463 }
464     return ERROR; // 返回错误操作
465 }
466
467 // 一个简单的输出函数
468 void visit (BiTree T)
469 {
470     printf (" %d,%s", T->data.key, T->data.others);
471 }
472
473 // 先序遍历二叉树T
474 status PreOrderTraverse(BiTree T, void(* visit )(BiTree))
475 {
476     // 如果T是空树, 直接返回
477     if (T == NULL)
478         return OK;
479     // 定义一个栈st, 同时定义栈的指针top和一个指针p
480     BiTree st[100], p;
481     int top = 0;

```

```
482 // 根节点先入栈
483     st[top++] = T;
484 // 当栈不为空时，循环遍历
485     while (top != 0)
486     {
487         // 指针p指向栈顶元素
488         p = st[--top];
489         // 对p进行访问
490         visit (p);
491         // 如果p右子树不为空，右子树先入栈
492         if (p->rchild != NULL)
493             st[top++] = p->rchild;
494         // 如果p左子树不为空，左子树后入栈，保证左子树可以先遍历
495         if (p->lchild != NULL)
496             st[top++] = p->lchild;
497     }
498 // 遍历完成，返回OK
499     return OK;
500 }
501
502 // 中序遍历二叉树T
503 status InOrderTraverse(BiTree T, void(* visit )(BiTree))
504 {
505     // 如果T为空，则直接返回
506     if (T == NULL)
507         return OK;
508     // 如果T非空，则进行遍历操作
509     if (T != NULL)
510     {
511         // 对T的左子树进行遍历
512         if (InOrderTraverse(T->lchild, visit ))
```

```
513         {
514     // 对T进行访问操作
515         visit (T);
516     // 对T的右子树进行遍历
517         if (InOrderTraverse(T->rchild, visit ))
518     // 如果遍历成功, 则返回OK, 表示遍历操作成功
519         return OK;
520     }
521     // 如果左子树或右子树遍历失败, 则返回ERROR, 表示遍历操作失败
522     return ERROR;
523 }
524 // 如果T为空, 则直接返回
525 else
526     return OK;
527 }
528
529 // 后序遍历二叉树T
530 status PostOrderTraverse(BiTree T, void(* visit )(BiTree))
531 {
532     // 如果二叉树为空, 则遍历结束
533     if (T == NULL)
534         return OK;
535     // 如果二叉树非空
536     if (T != NULL)
537     {
538     // 递归遍历左子树, 如果左子树遍历成功
539         if (PostOrderTraverse(T->lchild, visit )) {
540     // 递归遍历右子树, 如果右子树遍历成功
541         if (PostOrderTraverse(T->rchild, visit )) {
542     // 访问当前结点
543             visit (T);
```



```
544         return OK;
545     }
546 }
547 // 如果左子树或右子树遍历失败，则返回失败
548     return 0;
549 }
550 else
551     return OK;
552 }
553
554 // 层序遍历
555 status LevelOrderTraverse(BiTree T, void(* visit )(BiTree))
556 { // 按层遍历二叉树T
557     if (T == NULL) // 如果输入的二叉树为空树，直接返回
558         return OK;
559     BiTree st[200], p; // 定义一个数组，数组成员为BiTree类型，和一个结点p
560     int front = 0, rear = 0; // 定义两个变量，front代表队列开头，rear代表队列结尾
561     st[rear++] = T; // 将输入的二叉树的根节点放入数组的第一个结点
562     do // 使用循环实现遍历
563     {
564         p = st[front++]; // 将队列开头的元素赋值给变量p，并更新队列开头
565         visit(p); // 对当前结点进行遍历
566         if (p->lchild != NULL) // 如果当前结点有左孩子，将其放入队尾
567             st[rear++] = p->lchild;
568         if (p->rchild != NULL) // 如果当前结点有右孩子，将其放入队尾
569             st[rear++] = p->rchild;
```

```
570     } while (rear != front); // 队列仍有结点, 继续循环
571     return OK; // 遍历完成, 返回状态码
572 }
573
574 // 将二叉树的结点数据写入到文件FileName中
575 status SaveBiTree(BiTree T, char FileName[]) {
576     if (T == NULL) // 二叉树未创建, 则操作不能完成
577         return INFEASIBLE;
578     FILE *fp = fopen(FileName, "w"); // 以写入方式打开文件
579     if (fp == NULL)
580         return ERROR; // 文件指针打不开, 错误
581     BiTree st[100]; // 定义数组模拟栈
582     int mark[100], p = 0; // 记录每个结点的状态, p为栈顶指针
583     // 初始化栈顶指针
584     st[0] = T, mark[0] = 0;
585     while (p != -1) { // 栈非空则继续遍历
586         if (mark[p] == 0) { // 第一次访问该结点
587             // 将结点数据写入文件
588             fprintf(fp, "%d %s", st[p]->data.key, st[p]->data.others)
589                 ;
590             mark[p]++; // 将状态置为已访问左子树
591             if (st[p]->lchild == NULL)
592                 fprintf(fp, "%d null", 0); // 如果左子树为空, 写入
593                 null
594             else {
595                 st[p + 1] = st[p]->lchild; // 否则将左子树结点入栈
596                 p++; // 指针后移
597                 mark[p] = 0; // 新结点状态初始化
598             }
599         }
600         else if (mark[p] == 1) { // 第二次访问该结点
```

```

599         mark[p]++; // 状态置为已访问右子树
600         if (st[p]->rchild == NULL)
601             fprintf(fp, "%d null ", 0); // 如果右子树为空, 写入
                null
602         else {
603             st[p + 1] = st[p]->rchild; // 否则将右子树结点入栈
604             p++; // 指针后移
605             mark[p] = 0; // 新结点状态初始化
606         }
607     }
608     else if (mark[p] == 2) { // 第三次访问该结点
609         mark[p] = 0; // 状态置为未访问
610         st[p] = NULL; // 将该结点出栈
611         p--; // 指针前移
612     }
613 }
614 // 写入结束标志符
615 fprintf(fp, "%d null", -1);
616 fclose(fp); // 关闭文件指针
617 return OK;
618 }
619
620 // 该函数用于读取文件中的数据, 创建一颗二叉树
621 status LoadBiTree(BiTree &T, char FileName[])
622 {
623     // 如果传入的二叉树已经存在, 则无法进行操作
624     if (T != NULL)
625     {
626         printf("二叉树已经初始化, 无法操作\n");
627         return INFEASIBLE;
628     }

```

```
629
630 //尝试打开文件
631 FILE *fp = fopen(FileName, "r");
632 if (fp == NULL)
633     return ERROR;//文件指针打不开，返回错误
634 TElemType definitionfile [100]; //定义结构体类型数组，用于存储读
    取出来的值
635 BiTree st [100]; //定义指向节点的指针数组
636 int mark[100], p = 0; //定义一个标记数组，和一个标记位置的指针
    p，并初始化为0
637
638 //读入文件中的数据，存储到definitionfile[]数组中
639 int t = -1;
640 do {
641     t++;
642     fscanf(fp, "%d%s", & definitionfile [t].key, definitionfile [t].
        others);
643 } while ( definitionfile [t].key != -1);
644
645 //判断文件中的第一个结点是否为NULL，如果不是则创建根节点
646 if ( definitionfile [0].key != -1) {
647     T = st [0] = (BiTNode*)malloc(sizeof(BiTNode)), mark[0] = 0;
648     st[0]->data = definitionfile [t++];
649 }
650 else
651     return INFEASIBLE;//返回错误
652
653 t = 0;
654 //如果文件中的第一个结点不是NULL，则开始循环建立树形结构
655 while ( definitionfile [t].key != -1) {
656     //如果标记位置上的值为0，则说明需要在该节点的左子节点插
```

```

        入新节点
657         if (mark[p] == 0)
658         {
659             mark[p]++;
660             // 如果该节点的左子节点为NULL，则Mark数组位置+1，插
                入新节点
661             if ( definitionfile [t].key == 0)
662                 st[p]->lchild = NULL;
663             else
664             {
665                 st[p]->lchild = (BiTNode*)malloc(sizeof(BiTNode));
666                 // 将新节点指向的位置的值赋值为definitionfile[t]的值，
                    Mark数组位置+1
667                 st[p + 1] = st[p]->lchild ;
668                 p++;
669                 st[p]->data =  definitionfile [t];
670                 mark[p] = 0;
671             }
672             t++;
673         }
674         // 如果标记位置上的值为1，则说明需要在该节点的右子节
            点插入新节点
675         else if (mark[p] == 1)
676         {
677             mark[p]++;
678             // 如果该节点的右子节点为NULL，则Mark数组位置+1，插
                入新节点
679             if ( definitionfile [t].key == 0)
680                 st[p]->rchild = NULL;
681             else {
682                 st[p]->rchild = (BiTNode*)malloc(sizeof(BiTNode));

```

```

683         // 将新节点指向的位置的值赋值为definitionfile[t]的值,
           Mark数组位置+1
684         st[p + 1] = st[p]->rchild;
685         p++;
686         st[p]->data = definitionfile [t];
687         mark[p] = 0;
688     }
689     t++;
690 }
691 // 如果标记位置上的值为2, 则说明该节点的左右子节点都
           创建好了, 需要回退到上一级节点
692 else if (mark[p] == 2)
693 {
694     mark[p] = 0; // 重置Mark数组位置上的值为0
695     st[p] = NULL; // 将该位置的指针置NULL
696     p--; // 标记位置指针退回上一级节点
697 }
698 }
699
700 fclose (fp); // 关闭文件
701 return OK;
702 }
703
704 int MaxPathSum(BiTree T)
705 { // 初始条件是二叉树T存在; 操作结果是返回根节点到叶子结点的最大
           路径和;
706
707     // 如果当前结点是叶子结点, 则直接返回该结点的键值
708     if (T->lchild == NULL && T->rchild == NULL)
709         return T->data.key;
710

```

```
711         // 如果左子树为空，则仅考虑右子树节点的路径和
712         else if (T->lchild == NULL && T->rchild != NULL)
713             return MaxPathSum(T->rchild) + T->data.key;
714
715         // 如果右子树为空，则仅考虑左子树节点的路径和
716         else if (T->lchild != NULL && T->rchild == NULL)
717             return MaxPathSum(T->lchild) + T->data.key;
718
719         // 如果左右子树都非空，则计算左右子树的最大路径和，并将当前
           节点的键值加上左右子树的最大路径和中的较大值
720         int leftmax = 0, rightmax = 0;
721         leftmax = MaxPathSum(T->lchild);
722         rightmax = MaxPathSum(T->rchild);
723         if (leftmax > rightmax)
724             return leftmax + T->data.key;
725         else
726             return rightmax + T->data.key;
727     }
728
729     // 该函数的功能是：返回二叉树T中e1节点和e2节点的最近公共祖先
730     BiTree LowestCommonAncestor(BiTree T, int e1, int e2)
731     {
732         // 先找到节点p1和p2，分别代表e1和e2在二叉树中对应的结点
733         BiTree p1 = LocateNode(T, e1);
734         BiTree p2 = LocateNode(T, e2);
735         // 设置一个标志变量flag，用于标记是否判断过结点是否存在
736         static int flag = 0;
737         if (flag == 0) // 如果flag是0，说明还没有判断过结点是否存在
738         {
739             flag = 1;
740             // 如果e1或e2对应的结点不存在，或者它们中有一个对应的结点不存
```

在,

```
741 // 则输出错误并返回NULL
742     if( p1 == NULL || p2 == NULL)
743     {
744         printf ("输入的关键字错误\n");
745         return NULL;
746     }
747 }
748
749 // 如果二叉树为空, 或者T结点的关键字为e1或e2, 则返回T结点
750     if (T == NULL || T->data.key == e1 || T->data.key == e2)
751         return T;
752 // 递归查找左子树
753     BiTree left = LowestCommonAncestor(T->lchild, e1, e2);
754 // 递归查找右子树
755     BiTree right = LowestCommonAncestor(T->rchild, e1, e2);
756 // 如果left为空, 说明这两个节点在T结点的右子树上, 我们只需要返回
    右子树查找的结果即可
757     if (left == NULL)
758         return right;
759 // 如果right为空, 说明这两个节点在T结点的左子树上, 我们只需要返回
    左子树查找的结果即可
760     if (right == NULL)
761         return left;
762 // 如果left和right都不为空, 说明这两个节点一个在T的左子树上一个在
    T的右子树上
763 // T结点就是e1和e2的公共祖先!
764     return T;
765 }
766
767 // 函数名称: BiTree InvertTree(BiTree T)
```



```
768 // 函数功能：将二叉树T翻转，使其所有节点的左右节点互换
769 // 参数说明：二叉树T
770 // 返回值：翻转后的二叉树T
771
772 BiTree InvertTree (BiTree T)
773 {
774     // 如果二叉树为空，则直接返回
775     if (T == NULL)
776         return NULL;
777
778     // 递归处理左子树，返回左子树翻转后的结果
779     BiTree left = InvertTree (T->lchild);
780
781     // 递归处理右子树，返回右子树翻转后的结果
782     BiTree right = InvertTree (T->rchild);
783
784     // 交换左右节点
785     T->lchild = right ;
786     T->rchild = left ;
787
788     // 返回翻转后的结果
789     return T;
790 }
791
792 void fun01()
793 {
794     menufirst(); // 输出主菜单，提供可选的操作命令
795     int a ; // 命令编号/选择
796     printf("请输入一个命令\n");
797     scanf("%d",&a);
798 }
```

```
799 while (a) // 当输入非0时，继续进行操作
800 {
801     fflush ( stdin ); // 清空输入流，防止上一次操作结束后输入了数
                        据而影响本次操作
802
803     int feedback; // 操作返回值
804
805     switch (a) { // 根据命令编号进行相应的操作
806         case 1: // 创建一个新的二叉树
807             printf ("现在进行创建一个新的二叉树\n");
808             printf ("请输入你想创建的二叉树的名字\n");
809             char name1[30]; // 用于存储输入的二叉树名字
810             scanf ("%s",name1);
811             int i , flag ; flag = 0; // 标记位，用于判断是否已存
                        在同名二叉树
812
813             // 要进行名字的判断，遍历数组中的所有二叉树名字
814             for( i =0;i<Lists . length ;i++)
815             {
816                 if(strcmp(name1,Lists . elem[i ]. name) == 0) // 如果
                        名字已经存在，则无法创建这个二叉树
817                 {
818                     printf ("该二叉树已经存在，创建失败\n");
819                     flag = 1;
820                 }
821             }
822
823             if( flag == 0) // 如果不存在同名二叉树，则可以创建
824             {
825                 // 将新的二叉树名字加入到数组Lists中，并将Lists
                        的长度加1
```

```
826         strcpy ( Lists .elem[ Lists . length ]. name,name1);
827         Lists . length++;
828         printf ("创建成功\n");
829     }
830     break;
831
832     case 2: // 删除二叉树
833         int flag2 ; // 标记位，用于记录要删除的二叉树在数
            组中的位置
834         printf ("现在进行删除二叉树的操作\n");
835         printf ("请输入你想删除的二叉树的名字\n");
836         char name2[30]; // 用于存储目标二叉树名字
837         scanf ("%s",name2);
838         flag2 = -1; // flag2用于标记要删除的二叉树在Lists数
            组中的位置
839
840         //遍历数组中的所有二叉树名字，如果存在目标二叉
            树，则更新标记位
841         for( i =0;i<Lists . length ;i++)
842         {
843             if(strcmp(name2,Lists .elem[i ]. name) == 0)
844             {
845                 flag2 = i;
846             }
847         }
848
849         if( flag2 == -1) // 如果不存在目标二叉树，则无法进
            行删除操作
850         {
851             printf ("该二叉树不存在，无法删除\n");
852         }
```

```
853         else {
854             feedback = DestroyBiTree( Lists .elem[ flag2 ].T); //
                        调用DestroyBiTree函数销毁指定位置处的二叉树
855
856             if(feedback == OK) // 如果操作成功
857             {
858                 // 将Lists数组中指定位置之后的元素向前移动
                        一个位置，同时将Lists的长度减1
859                 int k;
860                 for( k = 0 ;k < Lists .length-1 ;k++)
861                 {
862                     Lists .elem[k] = Lists .elem[k+1];
863                 }
864                 Lists .length--;
865                 printf ("删除成功\n");
866             }
867         }
868         break;
869
870     case 3: // 查询创建了哪些二叉树
871         printf ("现在进行查询创建了哪些二叉树\n");
872         printf ("所有的二叉树如下:\n");
873
874         // 遍历数组中的所有二叉树名字，输出每个二叉树的名
                        称
875         for(i = 0; i<Lists .length ;i++)
876         {
877             printf ("%d)  %s\n",i+1, Lists .elem[i ].name);
878         }
879         break;
880
```

```
881         case 4: // 对二叉树进行操作
882             printf("现在进行二叉树的查找和操作\n");
883             printf("请输入你想查找和操作的二叉树的名字\n");
884             char name3[30]; // 用于存储目标二叉树名字
885             scanf("%s",name3);
886             int flag3 ; flag3 = -1; // flag3 用于标记要操作的二叉
                树在Lists数组中的位置
887
888             // 遍历数组中的所有二叉树名字，如果存在目标二叉
                树，则更新标记位
889             for( i =0 ; i<Lists.length ; i++)
890             {
891                 if(strcmp( Lists .elem[ i ]. name,name3) == 0)
892                 {
893                     flag3 = i;
894                 }
895             }
896
897             if( flag3 ==-1) // 如果不存在目标二叉树，则无法进行
                操作
898             {
899                 printf("不存在这个二叉树\n");
900                 system("pause");
901             }
902             else {
903                 // 调用fun02函数对特定位置处的二叉树进行操作
904                 fun02( Lists .elem[ flag3 ]. T);
905             }
906             break;
907
908         default :
```

```
909         printf("输入的命令错误, 请再次输入"); //如果输入的命令不在可选范围内, 则提示输入命令错误
910     }
911
912     printf("请输入下一个命令\n");
913     scanf("%d",&a);
914     system("cls"); //每次操作结束后, 清空屏幕并重新输出主菜单
915     menufirst();
916 }
917 }
918
919
920 /**
921  * 定义函数fun02, 传入参数BiTree &T
922  */
923 void fun02(BiTree &T)
924 {
925     /**
926      * 定义变量definition[100], 用于存储输入的二叉树的内容
927      * 定义变量op、i、next, 用于接收用户输入的命令
928      */
929     TElemType definition[100];
930     int op = 0, i = 0, next = 0;
931
932     /**
933      * 清空控制台输出
934      * 调用函数menu(), 输出主菜单
935      * 输出提示语句, 让用户输入命令
936      */
937     system("cls");
938     menu();
```

```
939     printf("请输入你的命令\n");
940     scanf("%d",&op);
941
942     /**
943      * 进入循环, 只要op不为0, 就执行代码块内的操作
944      */
945     while (op)
946     {
947         /**
948          * 使用switch语句, 根据不同的命令执行不同的操作
949          */
950         switch (op) {
951             case 1:
952                 i = 0;
953                 /**
954                  * 如果T已经存在, 输出相应的提示语句, 执行break跳出switch语句
955                  */
956                 if(T)
957                 {
958                     printf("二叉树已经初始化, 操作失败\n");
959                     break;
960                 }
961                 /**
962                  * 输入需要创建的二叉树的内容
963                  * 调用CreateBiTree函数进行创建
964                  * 如果创建成功, 输出相应的提示语句
965                  * 如果创建失败, 输出相应的提示语句
966                  */
967                 printf("请输入二叉树内容: \n");
968                 do {
```

```
969         scanf("%d%s", &definition[i].key, definition[i].  
          others);  
970     } while ( definition [i++].key != -1);  
971     if (CreateBiTree(T, definition ) == OK)  
972     {  
973         printf ("二叉树创建成功\n");  
974     }  
975     else  
976         printf ("二叉树创建失败! \n");  
977     break;  
978 case 2:  
979     /**  
980     * 调用DestroyBiTree函数进行二叉树的销毁  
981     * 如果销毁成功, 输出相应的提示语句  
982     * 如果销毁失败, 输出相应的提示语句  
983     */  
984     printf ("现在进行二叉树的销毁\n");  
985     if (DestroyBiTree(T) == OK)  
986         printf ("二叉树销毁成功!\n");  
987     else  
988         printf ("二叉树销毁失败!\n");  
989     break;  
990 case 3:  
991     /**  
992     * 调用ClearBiTree函数进行二叉树的清空  
993     * 如果清空成功, 输出相应的提示语句  
994     * 如果清空失败, 输出相应的提示语句  
995     */  
996     printf ("现在进行二叉树的清空\n");  
997     if (ClearBiTree(T) == OK)  
998         printf ("二叉树清空成功!\n");
```



```
999         else
1000             printf ("二叉树清空失败!\n");
1001         break;
1002     case 4:
1003         /**
1004          * 调用BiTreeEmpty函数判断二叉树是否为空
1005          * 如果为空，输出相应的提示语句
1006          * 如果不为空，输出相应的提示语句
1007          */
1008         printf ("现在进行二叉树的判空操作\n");
1009         if (BiTreeEmpty(T) == TRUE)
1010             printf ("二叉树是空树!\n");
1011         else if (BiTreeEmpty(T) == FALSE)
1012             printf ("二叉树不是空树!\n");
1013         else
1014             printf ("二叉树不存在!\n");
1015         break;
1016     case 5:
1017         /**
1018          * 调用BiTreeDepth函数求二叉树的深度
1019          * 如果求解成功，输出相应的提示语句
1020          * 如果求解失败，输出相应的提示语句
1021          */
1022         printf ("现在求二叉树的深度\n");
1023         int j5; // 接收二叉树的深度函数的返回值
1024         if (T == NULL) {
1025             printf ("二叉树不存在!\n");
1026             break;
1027         }
1028         j5 = BiTreeDepth(T);
1029         if(j5 == -1)
```

```
1030         {
1031             printf ("操作失败\n");
1032         }
1033     else {
1034         printf ("二叉树的深度为%d!\n", j5);
1035     }
1036     break;
1037 case 6:
1038     /**
1039      * 输入需要查找的结点关键字
1040      * 调用LocateNode函数进行查找
1041      * 如果查找成功，输出相应的提示语句及查找结果
1042      * 如果查找失败，输出相应的提示语句
1043      */
1044     // 定义需要查找的结点关键字
1045     int e6;
1046     // 定义二叉树结点指针
1047     BiTree p6;
1048     // 判断二叉树是否存在
1049     if (T == NULL) {
1050         printf ("二叉树不存在!\n");
1051         break;
1052     }
1053     // 提示用户输入需要查找的结点关键字
1054     printf ("请输入你要查找的结点关键字: \n");
1055     // 读入用户输入的需要查找的结点关键字
1056     scanf ("%d", &e6);
1057     // 调用LocateNode函数进行二叉树的查找操作，返回查
        找结果到p6中
1058     p6 = LocateNode(T, e6);
1059     // 判断是否查找到目标结点
```

```
1060         if (p6 == NULL)
1061             printf("查找失败!\n");
1062         else
1063             // 输出查找结果
1064             printf("查找成功!其值为: %s\n",p6->data.others);
1065         break;
1066     case 7:
1067         int e7, j7; // 定义变量
1068         KeyType k7; // 定义关键字类型
1069         TElemType value7; // 定义结点内容类型
1070         if (T == NULL) { // 如果二叉树为空, 输出提示语句
1071             printf("二叉树不存在!\n");
1072             break;
1073         }
1074         printf("请输入你要赋值的结点的关键字: \n"); // 提示
            输入
1075         scanf("%d", &k7); // 获取关键字
1076         printf("请输入你要赋值的内容: \n"); // 提示输入结点
            内容
1077         scanf("%d %s", &value7.key, &value7.others); // 获取
            结点内容
1078         j7 = Assign(T,k7,value7); // 调用赋值函数, 返回状态
            值
1079         if (j7 == ERROR) // 如果赋值失败, 输出相应的提示语
            句
1080             printf("赋值失败!\n");
1081         else if (j7 == INFEASIBLE) // 如果二叉树不存在, 输
            出相应的提示语句
1082             printf("二叉树不存在!\n");
1083         else // 如果赋值成功, 输出相应的提示语句
1084             printf("赋值成功!\n");
```

```
1085         break;
1086
1087     case 8:
1088         int e8; // 定义变量
1089         BiTree p8; // 定义二叉树指针
1090         if (T == NULL) { // 如果二叉树为空，输出相应的提示
            语句
1091             printf ("二叉树不存在!\n");
1092             break;
1093         }
1094         printf ("请输入你要获得的兄弟结点的关键字: \n"); //
            提示输入
1095         scanf ("%d", &e8); // 获取关键字
1096         p8 = GetSibling (T, e8); // 调用获取兄弟结点函数，返
            回兄弟结点指针
1097         if (p8 == NULL) // 如果获取失败，输出相应的提示语
            句
1098             printf ("获取失败!\n");
1099         else // 如果获取成功，输出相应的提示语句和兄弟结
            点内容
1100             printf ("获取兄弟结点成功!其值为: %d %s\n", p8->
                data.key, p8->data.others);
1101         break;
1102     case 9: // 插入结点
1103         int e9, j9, LR; // e9、j9、LR为变量，用于存储用户
            输入的值
1104         TElemType value9; // value9为结构体类型，用于存储用
            户输入的内容
1105         if (T == NULL) { // 如果二叉树不存在，则输出提示信
            息并结束
1106             printf ("二叉树不存在!\n");
```

```
1107         getchar(); getchar();
1108         break;
1109     }
1110     printf("请输入你要插入结点的关键字和LR: \n"); // 提示用户输入关键字和LR
1111     scanf("%d %d", &e9, &LR); // 从输入流中获取用户输入
1112
1113     printf("请输入待插入的内容 (格式: 1 a) : \n"); // 提示用户输入待插入内容
1114     scanf("%d%s", &value9.key, value9.others); // 从输入流中获取用户输入
1115     j9 = InsertNode(T, e9, LR, value9); // 调用InsertNode函数插入结点
1116     if (j9 == ERROR) // 如果插入失败, 输出提示信息
1117         printf("插入结点失败!\n");
1118     else if (j9 == OK) // 如果插入成功, 输出提示信息
1119         printf("插入结点成功!\n");
1120
1121     break; // 结束case 9
1122
1123     case 10: // 删除结点
1124         int e10, j10; // e10、j10为变量, 用于存储用户输入的值
1125         if (T == NULL) { // 如果二叉树不存在, 则输出提示信息并结束
1126             printf("二叉树不存在!\n");
1127             break;
1128         }
1129         printf("请输入你要删除结点的关键字: \n"); // 提示用户输入待删除结点的关键字
1130         scanf("%d", &e10); // 从输入流中获取用户输入
```

```
1131         j10 = DeleteNode(T, e10); // 调用DeleteNode函数删除
           结点
1132         if (j10 == ERROR) // 如果删除失败，输出提示信息
1133             printf ("删除结点失败!\n");
1134         else if (j10 == OK) // 如果删除成功，输出提示信息
1135             printf ("删除结点成功!\n");
1136
1137         break; // 结束case 10
1138
1139     case 11: // 先序遍历
1140         int j11; // j11为变量，用于存储先序遍历函数的返回
           值
1141         if (T == NULL) { // 如果二叉树不存在，则输出提示信
           息并结束
1142             printf ("二叉树不存在!\n");
1143             break;
1144         }
1145         j11 = PreOrderTraverse(T, visit ); // 调用
           PreOrderTraverse函数完成先序遍历
1146         if (j11 == OK) // 如果遍历成功，输出提示信息
1147             printf ("\n完成先序遍历!\n");
1148         else { // 如果遍历失败，输出提示信息
1149             printf ("遍历失败\n");
1150         }
1151
1152         break; // 结束case 11
1153
1154     case 12:
1155         int j12; // 声明一个整型变量 j12，用于存储
           InOrderTraverse 函数的返回值
1156         if (T == NULL) { // 如果二叉树 T 不存在
```

```
1157         printf("二叉树不存在!\n");//输出提示信息
1158         break; //跳出 switch-case 循环
1159     }
1160     j12 = InOrderTraverse(T, visit ); //执行中序遍历函数
        InOrderTraverse, 并将返回值存储到 j12 变量中
1161     if (j12 == OK)//如果遍历成功
1162         printf("\n完成中序遍历!\n");//输出提示信息
1163     else
1164         printf("\n遍历失败!\n");//输出提示信息
1165
1166     break; //跳出 switch-case 循环
1167 case 13:
1168     int j13; //声明一个整型变量 j13, 用于存储
        PostOrderTraverse 函数的返回值
1169     if (T == NULL) { //如果二叉树 T 不存在
1170         printf("二叉树不存在!\n");//输出提示信息
1171
1172         break; //跳出 switch-case 循环
1173     }
1174     j13 = PostOrderTraverse(T, visit ); //执行后序遍历函数
        PostOrderTraverse, 并将返回值存储到 j13 变量中
1175     if (j13 == OK)//如果遍历成功
1176         printf("\n完成后序遍历!\n");//输出提示信息
1177     else
1178         printf("\n遍历失败!\n");//输出提示信息
1179
1180     break; //跳出 switch-case 循环
1181 case 14:
1182     int j14; //声明一个整型变量 j14, 用于存储
        LevelOrderTraverse 函数的返回值
1183     if (T == NULL) { //如果二叉树 T 不存在
```

```
1184         printf("二叉树不存在!\n");//输出提示信息
1185
1186         break;//跳出 switch-case 循环
1187     }
1188     j14 = LevelOrderTraverse(T, visit );//执行按层遍历函数 LevelOrderTraverse, 并将返回值存储到 j14 变量中
1189     if (j14 == OK)//如果遍历成功
1190         printf("\n完成按层遍历!\n");//输出提示信息
1191     else
1192         printf("\n遍历失败!\n");//输出提示信息
1193
1194     break;//跳出 switch-case 循环
1195 case 15:
1196     int j15;//声明一个整型变量 j15, 用于存储 MaxPathSum 函数的返回值
1197     if (T == NULL) {//如果二叉树 T 不存在
1198         printf("二叉树不存在!\n");//输出提示信息
1199         getchar(); getchar();//暂停程序执行, 等待用户输入
1200
1201         break;//跳出 switch-case 循环
1202     }
1203     j15 = MaxPathSum(T);//执行计算二叉树最大路径和函数 MaxPathSum, 并将返回值存储到 j15 变量中
1204     printf("二叉树最大路径和为: %d!\n",j15);//输出计算结果
1205
1206     break;//跳出 switch-case 循环
1207
1208 case 16:
```



```
1209         if (T == NULL) {
1210             printf ("二叉树不存在!\n");
1211
1212             break;
1213         }
1214
1215
1216         int i16, j16; // 需要查找公共祖先的第一个节点的关键
                        // 字和第二个节点的关键字
1217         BiTree T16; // 查找到的公共祖先节点
1218         printf ("请输入你要搜索公共祖先的两个结点的关键
                        // 字: \n");
1219         scanf ("%d %d", &i16, &j16);
1220         T16 = LowestCommonAncestor(T, i16, j16);
1221         if (T16 == NULL)
1222         {
1223             printf ("查找失败!\n");
1224         }
1225
1226         else
1227         {
1228             printf ("查找成功! \n");
1229             printf ("公共祖先的关键字为: %d,其内容为%s\n",
                        // T16->data.key, T16->data.others);
1230         }
1231
1232
1233         break;
1234
1235     case 17:
1236         if (T == NULL) {
```

```
1237         printf ("二叉树不存在!\n");
1238         getchar ();
1239         break;
1240     }
1241     InvertTree (T);
1242     printf ("已成功翻转二叉树! \n");
1243
1244     break;
1245 case 18:
1246     int j18;
1247     char FileName18[30]; // 保存到的文件名
1248     printf ("请输入要写入的文件名: \n");
1249     scanf ("%s", FileName18);
1250     j18 = SaveBiTree(T, FileName18);
1251     if (j18 == INFEASIBLE)
1252         printf ("二叉树不存在!\n");
1253     else
1254         printf ("成功将二叉树写入文件名为: %s的文件中!\n", FileName18);
1255
1256     break;
1257 case 19:
1258     int j19;
1259     char FileName19[30]; // 待读取数据的文件名
1260     printf ("请输入要读取的文件名: \n");
1261     scanf ("%s", FileName19);
1262     j19 = LoadBiTree(T, FileName19);
1263     if (j19 == INFEASIBLE)
1264         printf ("二叉树存在!无法覆盖! \n");
1265     else if (j19 == ERROR) {
1266         printf ("读取文件失败! \n");
```

```
1267         }
1268         else {
1269             printf ("成功将%s文件中的数据读入到二叉树中!\n",
1270                     FileName19);
1271         }
1272         break;
1273
1274
1275         case 0:
1276             break;
1277     }
1278     putchar ('\n');
1279     printf ("请输入下一个命令\n");
1280     scanf ("%d",&op);
1281     system("cls");
1282     if (op != 0) // 如果退出就加载第一个菜单
1283     {
1284         menu();
1285     }
1286     else {
1287         menufirst ();
1288     }
1289
1290
1291 }
1292 }
1293
1294 void menufirst ()
1295 {
1296     for (int k = 0; k<= 119 ;k++)
```

```

1297     {
1298         putchar('-');
1299     } putchar('\n');
1300     printf("1.创建一个二叉树\n");
1301     printf("2.删除一个二叉树\n");
1302     printf("3.查询已经创建的二叉树\n");
1303     printf("4.查找一个二叉树和进行操作\n");
1304     printf("0.退出多个二叉树的管理\n");
1305
1306     printf("      ^             /\n");
1307     printf("      /\ 7         □ _\n");
1308     printf("      / |         /  /\n");
1309     printf("      | Z _,<    /    /'F\n");
1310     printf("      |          F    /    > \n");
1311     printf("      Y          '    /    \n");
1312     printf("      ?● ? ●    ?? <    \n");
1313     printf("      () ^        | \  <\n");
1314     printf("      >? ?_  彳    | /  /\n");
1315     printf("      / ^      /  ?<| \  \n");
1316     printf("      F_?      ( /    | /  /\n");
1317     printf("      7          | /  \n");
1318     printf("      >—r - - '?— _\n");
1319     for( int k = 0; k<= 119 ;k++)
1320     {
1321         putchar('-');
1322     } putchar('\n');
1323 }
1324
1325 void menu()
1326 {
1327

```

```

1328     printf ("          Menu for Binary Tree On Binary Linked List   \n");
1329     printf ("
          _____\
          n");
1330     printf ("**      1. 创建初始化二叉树    11.前序遍历
          **\n");
1331     printf ("**      2. 销毁二叉树          12. 中序遍历          **\
          n");
1332     printf ("**      3. 清空二叉树          13. 后序遍历          **\
          n");
1333     printf ("**      4. 二叉树判空          14. 层序遍历          **\
          n");
1334     printf ("**      5. 求二叉树的深度        15. 最大路径和
          **\n");
1335     printf ("**      6. 查找结点          16. 最近公共祖先          **\
          n");
1336     printf ("**      7. 结点赋值          17. 翻转二叉树          **\
          n");
1337     printf ("**      8. 获得兄弟结点          18. 二叉树的文件保存
          **\n");
1338     printf ("**      9. 插入结点          19. 二叉树的文件加载
          **\n");
1339     printf ("**      10. 删除结点          0. Exit          **\n"
          );
1340     printf ("
          _____\
          n");
1341     printf ("          请选择你的操作[0~19]:\n");
1342
1343 }
1344

```

```
1345 // 获取指定结点e的兄弟结点
1346 BiTNode* GetSibling(BiTree T, KeyType e)
1347 {
1348     if (T == NULL) //若二叉树为空，则返回空指针
1349     {
1350         printf("二叉树不存在\n");
1351         return NULL;
1352     }
1353
1354     BiTree st[100], p; // 定义一个存放结点指针的数组，同时声明一个
        指向树结构体的指针p
1355     int top = 0; // 定义一个栈顶指针，初始值为0
1356     st[top++] = T; // 将整棵树压入栈中
1357
1358     while (top != 0) { // 当栈不为空时，进行以下操作
1359         p = st[--top]; // 取出栈顶元素，同时栈顶指针减1
1360         if (p->rchild->data.key == e) // 如果p的右子结点为要查找兄弟
            结点的结点e
1361             return p->lchild; // 则返回p的左子结点
1362         if (p->lchild->data.key == e) // 如果p的左子结点为要查找兄弟
            结点的结点e
1363             return p->rchild; // 则返回p的右子结点
1364         // 如果p的右子树和左子树都不为空，则将它们分别压入栈中
1365         if (p->rchild->rchild != NULL && p->rchild->lchild != NULL)
1366             st[top++] = p->rchild;
1367         if (p->lchild->rchild != NULL && p->lchild->lchild != NULL)
1368             st[top++] = p->lchild;
1369     }
1370     return NULL; //若未找到兄弟结点，则返回空指针
1371 }
```

附录 D 基于邻接表图实现的源程序

```
1  /*—— 头文件的申明 ——*/
2  #include<stdio .h>
3  #include<stdlib .h>
4  #include "string .h"
5
6  /*—— 预定义 ——*/
7  // 定义布尔类型TRUE和FALSE
8  #define TRUE 1
9  #define FALSE 0
10
11 // 定义函数返回值类型
12 #define OK 1
13 #define ERROR 0
14 #define INFEASIBLE -1
15 #define OVERFLOW -2
16 #define MAX_VERTEX_NUM 20
17
18 // 定义数据元素类型
19 typedef int ElemType;
20 typedef int status ;
21 typedef int KeyType;
22 typedef enum {DG,DN,UDG,UDN} GraphKind;
23
24 // 定义顶点类型，包含关键字和其他信息
25 typedef struct {
26     KeyType key; // 关键字
27     char others [20]; // 其他信息
28 } VertexType;
29
```

```
30 //定义邻接表结点类型
31 typedef struct ArcNode {
32     int adjvex; // 顶点在顶点数组中的下标
33     struct ArcNode *nextarc; // 指向下一个结点的指针
34 } ArcNode;
35
36 //定义头结点类型和数组类型（头结点和边表构成一条链表）
37 typedef struct VNode{
38     VertexType data; // 顶点信息
39     ArcNode *firstarc; // 指向第一条弧的指针
40 } VNode,AdjList[MAX_VERTEX_NUM];
41
42 //定义邻接表类型，包含头结点数组、顶点数、弧数和图的类型
43 typedef struct {
44     AdjList vertices; // 头结点数组
45     int vexnum,arcnum; //顶点数和弧数
46     GraphKind kind; // 图的类型（有向图、无向图等）
47 } ALGraph;
48
49 //定义图集合类型，包含一个结构体数组，每个结构体包含图的名称和
    邻接表
50 typedef struct {
51     struct {
52         char name[30]="0"; // 图的名称
53         ALGraph G; //对应的邻接表
54     }elem[30]; // 图的个数
55     int length; // 图集合中图的数量
56 }Graphs;
57
58 Graphs graphs; // 图的集合的定义
59
```



```

60  /*----- 函数申明 -----*/
61  status  isrepeat (VertexType V[]); //判断是否有重复结点
62  status  CreateCraph(ALGraph &G,VertexType V[],KeyType VR[][2]); //创
    建
63  status  DestroyGraph(ALGraph &G); //销毁
64  status  LocateVex(ALGraph G, KeyType u); //查找
65  status  PutVex(ALGraph &G, KeyType u, VertexType value); //顶点赋值
66  status  FirstAdjVex (ALGraph G, KeyType u); //获得第一邻接点
67  status  NextAdjVex(ALGraph G, KeyType v, KeyType w); //获得下一邻接
    点
68  status  InsertVex (ALGraph &G,VertexType v); //插入顶点
69  status  DeleteVex(ALGraph &G, KeyType v); //删除顶点
70  status  InsertArc (ALGraph &G,KeyType v,KeyType w); //插入弧
71  status  DeleteArc(ALGraph &G,KeyType v,KeyType w); //删除弧
72  void  dfs (ALGraph G , void (* visit )(VertexType), int  nownode);
73  status  DFSTraverse(ALGraph G,void (*visit)(VertexType)); // dfs遍历
74  void  BFS(ALGraph G,void (* visit )(VertexType), int  i);
75  status  BFSTraverse(ALGraph G,void (*visit)(VertexType)); // bfs遍历
76  void  visit (VertexType p); //遍历的时候调用的输出函数
77  int  * VerticesSetLessThanK(ALGraph G,int  v, int  k); // 顶点小于k的顶
    点集合
78  int  ShortestPathLength (ALGraph G,int  v, int  w); // 顶点间的最短路径
79  int  ConnectedComponentsNums(ALGraph G); //图的分量
80  status  SaveGraph(ALGraph G, char FileName[]); // 图的文件保存
81  status  LoadGraph(ALGraph &G, char FileName[]); // 图的文件读取
82  void  menu(); // 多个图管理的菜单
83  void  menu2(); // 单个图管理的菜单
84  void  fun01(); // 多个图管理的封装函数
85  void  fun02(ALGraph &G); // 单个图管理的封装函数
86
87  /*----- main主函数 -----*/
    
```

```
88  int main()
89  {
90      system("color 37");
91      fun01(); //封装处理函数
92
93      return 0;
94  }
95
96
97
98
99  /*----- 函数的定义 -----*/
100
101  status  isrepeat (VertexType V[]) // 查找重复节点
102  {
103      int i=0;
104      int flag[1000]={0}; //设计标记数组
105      while (V[i].key != -1)
106      {
107          if( flag[V[i].key] != 0) // 如果有重复的结点，返回 1
108          {
109              return 1;
110          }
111          flag[V[i].key]++; // 标记关键字，以检测重复节点
112          i++;
113      }
114      return 0;
115  }
116
117  /*根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR*/
118  status  CreateCraph(ALGraph &G, VertexType V[], KeyType VR[][2])
```

```

119 {
120     if(G.vexnum != 0) // 如果图已经创建，不能再次初始化
121     {
122         printf("该图已经初始化，不能再次初始化\n");
123         return INFEASIBLE;
124     }
125     int i=0;
126     int flag[100000]; // 标记每个关键字出现的位置
127     int flagvr[500][500]={0}; // 标记每条边是否出现过，防止重复边
        和自环的出现
128
129     memset(flag,-1,sizeof(flag)); // 标记数组初始化为-1
130     if( isrepeat(V)==1 || V[0].key==-1 || (V[1].key ==-1 && VR
        [0][0]!=-1))
131     {
132         return ERROR; // 如果出现空图、自环、以及重复结点等，返
        回错误代码
133     }
134
135     while (V[i].key!= -1)
136     {
137         if(i >= MAX_VERTEX_NUM) // 如果超出节点的最大数量，返
        回错误代码
138         {
139             return ERROR;
140         }
141         G.vertices[i].data = V[i]; // 将节点信息存储到 vertices 数组
        中
142         G.vertices[i].firstarc = NULL; // 初始化节点的第一个邻接点
        为空
143         flag[V[i].key] = i; // 标记每个节点的位置
    
```

```

144         i++;
145     }
146
147     G.vexnum=i; // 存储节点数量
148
149     i=0;
150
151     while (VR[i][0]!= -1) // 创建边
152     {
153         flagvr[VR[i][0]][VR[i][1]]++; // 标记边是否出现过
154
155         // 如果出现环和重复的边，返回错误代码
156         if(VR[i][0]==VR[i][1] || (flagvr[VR[i][0]][VR[i][1]]+flagvr[
            VR[i][1]][VR[i][0]]) > 1)
157         {
158             return ERROR;
159         }
160
161         if(flag[VR[i][0]] == -1) // 如果边连接的节点没有出现过，
            返回错误代码
162         {
163             return ERROR;
164         }
165
166         // 插入结点，使用头插法，即插入到邻接链表的前面
167         ArcNode *last = G.vertices[flag[VR[i][1]]].firstarc ;
168         ArcNode *p = (ArcNode *) malloc(sizeof(ArcNode));
169         p->adjvex = VR[i][0];
170         p->nextarc = NULL;
171
172         if( last == NULL) // 如果是第一个邻接点，直接插入
    
```

```

173     {
174         G.vertices [ flag [VR[i ][1]]]. firstarc  = p;
175         i++; // 继续下一条边的操作
176
177     } else { // 如果不是第一个邻接点，使用头插法进行插入
178         p->nextarc = last ;
179         G.vertices [ flag [VR[i ][1]]]. firstarc  = p;
180         i++; // 继续下一条边的操作
181     }
182 }
183
184 i=0;
185
186 while(VR[i][1]!=-1) // 创建另一条方向的边
187 {
188     if( flag [VR[i ][1]] == -1)
189     {
190         return ERROR;
191     }
192
193     // 插入结点，使用头插法，即插入到邻接链表的前面
194     ArcNode *last = G.vertices [ flag [VR[i ][0]]]. firstarc ;
195     ArcNode *p =(ArcNode*) malloc(sizeof(ArcNode));
196     p->adjvex = flag [VR[i ][1]];
197     p->nextarc = NULL;
198
199     if( last == NULL) // 如果是第一个邻接点，直接插入
200     {
201         G.vertices [ flag [VR[i ][0]]]. firstarc  = p;
202         i++; // 继续下一条边的操作
203     }

```

```
204         else { // 如果不是第一个邻接点，使用头插法进行插入
205             p->nextarc = last ;
206             G.vertices [ flag [ VR[i ][0]]]. firstarc  = p;
207             i++; // 继续下一条边的操作
208         }
209     }
210
211     G.arcnum=i; // 存储边的数量
212     return OK;
213 }
214
215 status DestroyGraph(ALGraph &G)
216 /*销毁无向图G,删除G的全部顶点和边*/
217 {
218     // 如果图不存在，返回“不可行”的错误信息
219     if(G.vexnum == 0 )
220     {
221         return INFEASIBLE;
222     }
223     ArcNode *p=NULL;
224     ArcNode *sub=NULL;
225     int i=0;
226     // 循环遍历所有的顶点
227     while (i<G.vexnum)
228     {
229         sub = G.vertices [ i ]. firstarc ;
230         // 对每个顶点，循环遍历它的每个邻接点
231         while (sub)
232         {
233             p = sub;
234             sub = sub->nextarc;
```

```
235         // 删除该邻接点对应的边
236         free(p);
237         p = NULL;
238     }
239     i++;
240 }
241 // 重置计数器，表示图中没有顶点和边
242 G.vexnum = 0;
243 G.arcnum = 0;
244 return OK;
245 }
246
247 int LocateVex(ALGraph G, KeyType u)
248 // 根据u在图G中查找顶点，查找成功返回位序，否则返回-1；
249
250 {
251     // 如果图不存在，返回“不可行”的错误信息
252     if(G.vexnum == 0)
253     {
254         printf("该图不存在或未初始化\n");
255         return INFEASIBLE;
256     }
257     int i = 0;
258     // 循环遍历所有的顶点
259     while(i < G.vexnum)
260     {
261         // 如果找到关键字值为u的顶点，返回它的位序
262         if(G.vertices[i].data.key == u)
263         {
264             return i;
265         }
266     }
```

```
266         i++;
267     }
268     // 如果没找到关键字值为u的顶点，返回-1
269     return -1;
270 }
271
272
273 // 顶点赋值：函数名称是PutVex (G,u,value); 初始条件是图G存在，u是
    和G中顶点关键字类型相同的给定值；
274 // 操作结果是对关键字为u的顶点赋值value；
275 status PutVex(ALGraph &G, KeyType u, VertexType value)
276 {
277     // 如果图不存在，返回错误信息 INFEASIBLE
278     if(G.vexnum == 0)
279     {
280         printf("该图不存在或未初始化\n");
281         return INFEASIBLE;
282     }
283     int i=0; // 用来记录下标
284     int num=0; int flag=-1; // 计数和标记
285     while (i<G.vexnum)
286     {
287         // 如果关键字不唯一，返回错误信息 ERROR
288         if(value.key == G.vertices[i].data.key && value.key != u)
289         {
290             printf("关键字不唯一,操作失败\n");
291             return ERROR;
292         }
293         // 如果查找到了指定的顶点，记录其出现的次数和下标
294         if(G.vertices[i].data.key == u)
295         {
```



```
296         num++;           // 用来记录出现的次数
297         flag =i;          // 保存下标
298     }
299     i++;
300 }
301 // 如果未查找到指定的顶点或者查找到的次数不唯一，返回错误信息 ERROR
302 if(num != 1)
303 {
304     printf("查找失败,无法操作\n");
305     return ERROR;
306 }
307 // 将找到的符合条件的顶点的值修改成指定的 value 值
308 G.vertices [ flag ]. data = value;
309 // 操作成功，返回 OK
310 return OK;
311 }
312 // 获得第一邻接点：函数名称是FirstAdjVex(G, u)；初始条件是图G存在，u是G中顶点的位序；
313 // 操作结果是返回u对应顶点的第一个邻接顶点位序，如果u的顶点没有邻接顶点，否则返回其它表示“不存在”的信息；
314 int FirstAdjVex(ALGraph G, KeyType u)
315 {
316     // 如果图不存在，返回错误信息 INFEASIBLE
317     if(G.vexnum == 0) // 图不存在
318     {
319         printf("该图不存在或未初始化\n");
320         return INFEASIBLE;
321     }
322     // 在图 G 中寻找给定关键字对应的顶点
323     int i=0; // 用来计数
```

```

324     while (i < G.vexnum)
325     {
326         if (G.vertices[i].data.key == u)
327         {
328             // 如果找到了顶点，则返回该顶点对应的第一邻接顶点的
                位序
329             return G.vertices[i].firstarc ->adjvex;
330         }
331         i++;
332     }
333     // 如果未找到给定关键字对应的顶点，返回信息“不存在”，即 -1
334     return -1;
335 }
336
337 // 获得下一邻接点：函数名称是NextAdjVex(G, v, w)；初始条件是图G存
    在，v和w是G中两个顶点的位序，v对应G的一个顶点,w对应v的邻接
    顶点；操作结果
338 // 是返回v的（相对于w）下一个邻接顶点的位序，如果w是最后一个邻
    接顶点，返回其它表示“不存在”的信息；
339 // 参数说明：G为有向图，v是源节点，w是目标节点
340 int NextAdjVex(ALGraph G, KeyType v, KeyType w)
341 {
342     // 如果图不存在，则返回“不存在”的信息
343     if (G.vexnum == 0)
344     {
345         printf("该图不存在或未初始化\n");
346         return INFEASIBLE;
347     }
348     int i = 0;
349     int flagv = -1, flagw = -1; // 用来记录v和w对应的下标
350     // 找到v和w在G.vertices数组中的下标

```

```

351     while (i < G.vexnum)
352     {
353         if (G.vertices[i].data.key == v)
354         {
355             flagv = i;
356         }
357         if (G.vertices[i].data.key == w)
358         {
359             flagw = i;
360         }
361         i++;
362     }
363     // 若找不到v或w对应的结点，返回“不存在”的信息
364     if (flagv == -1 || flagw == -1)
365     {
366         printf("v或w对应的结点不存在\n");
367         return -1;
368     }
369     ArcNode* p = G.vertices[flagv].firstarc;
370     ArcNode* ptail = p->nextarc;
371     // 遍历源节点的邻接链表，找到目标节点w
372     while (ptail)
373     {
374         if (p->adjvex == flagw)
375         {
376             // 如果w不是最后一个邻接顶点，则返回其下一个邻接顶点的位序，否则返回“不存在”的信息
377             return ptail->adjvex;
378         }
379         p = ptail;
380         ptail = p->nextarc;

```

```
381     }
382     return -1;
383 }
384
385 //插入顶点：函数名称是InsertVex(G,v)；初始条件是图G存在，v和G中
        的顶点具有相同特征；操作结果是在图G中增加新顶点v。
386 // （在这里也保持顶点关键字的唯一性）
387 // 参数说明：G为有向图，v为要插入的结点
388 status InsertVex (ALGraph& G, VertexType v)
389 {
390     // 如果图不存在，则返回“不存在”的信息
391     if (G.vexnum == 0)
392     {
393         printf ("该图不存在或未初始化\n");
394         return INFEASIBLE;
395     }
396     int i = 0; //记录下标
397     // 如果图中顶点数量已达到最大限制，则返回ERROR
398     if (G.vexnum == MAX_VERTEX_NUM)
399     {
400         printf ("超出所能容纳的最大顶点管理空间\n");
401         return ERROR;
402     }
403     // 查找图中是否已有KEY相同的结点
404     while (i < G.vexnum)
405     {
406         if (G.vertices[i].data.key == v.key)
407         {
408             printf ("关键字不唯一\n");
409             return ERROR;
410         }
```

```
411         i++;
412     }
413     // 在G.vertices数组的最后一个位置插入新结点，更新G.vexnum
414     G.vertices[G.vexnum].data = v;
415     G.vertices[G.vexnum].firstarc = NULL;
416     G.vexnum++;
417     return OK;
418 }
419
420 // 删除顶点：函数名称是DeleteVex(G,v)；初始条件是图G存在，v是和G
    中顶点关键字类型相同的给定值；
421 // 操作结果是在图G中删除关键字v对应的顶点以及相关的弧
422 status DeleteVex(ALGraph &G, KeyType v)
423 // 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
424 {
425     // 请在这里补充代码，完成本关任务
426     // 若图不存在或未初始化，则返回不可行状态
427     if(G.vexnum == 0)
428     {
429         printf("该图不存在或未初始化\n");
430         return INFEASIBLE;
431     }
432     // 若图中只有一个顶点，则无法删除，返回错误状态
433     if(G.vexnum == 1)
434     {
435         printf("图中只有一个顶点，不能删除\n");
436         return ERROR;
437     }
438     int i = 0; // 标记下标
439     // 寻找要删除的顶点
```

```
440     while (i<G.vexnum)
441     {
442         if(G.vertices[i].data.key == v)
443         {
444             // 删除与这个顶点有关的弧
445             while (G.vertices[i].firstarc){
446                 G.arcnum--;
447                 ArcNode * p = G.vertices[i].firstarc ;
448                 G.vertices[i].firstarc = p->nextarc;
449                 free(p);
450                 p = NULL;
451             }
452             break;
453         }
454         i++;
455     }
456     int location = i; //记录位置
457     //若要删除的顶点不存在，则返回错误状态
458     if(i == G.vexnum)
459     {
460         printf("要删除的顶点不存在.无法操作\n");
461         return ERROR;
462     }
463     //将删除顶点之后的顶点位置全部向前移动一个位置，覆盖掉要删除的
    位置
464     while (i<G.vexnum-1)
465     {
466         G.vertices[i] = G.vertices[i+1];
467         i++;
468     }
469     G.vexnum--;
```

```
470 //下面还要进行与这个顶点有关的弧的删除操作，以及将所有大于要删除位置的顶点位置减一
    ArcNode * train = NULL; //记录操作
    ArcNode * p = NULL;
    int k = 0;
    while (k < G.vexnum)
    {
        train = G.vertices[k].firstarc;
        p = train;
        while (train != NULL)
        {
            //找到与要删除的顶点有关的弧进行删除
            if (location == train->adjvex)
            {
                if (train == p)
                {
                    G.vertices[k].firstarc = train->nextarc;
                    train = train->nextarc;
                    free(p);
                    p = NULL;
                    continue;
                }
                p->nextarc = train->nextarc;
                p = train;
                train = p->nextarc;
                free(p);
                p = NULL;
                continue;
            }
        }
        //将所有大于要删除位置的顶点位置减一
```

```

500         if( train ->adjvex > location )
501         {
502             train ->adjvex--;
503         }
504         p = train ;
505         train = p->nextarc;
506
507     }
508     k++;
509 }
510 // 删除成功，返回操作成功状态
511     return OK;
512 }
513
514 // 插入弧：函数名称是InsertArc(G,v,w); 初始条件是图G存在，v、w是
           和G中顶点关键字类型相同的给定值；
515 // 操作结果是在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,
           v>;
516 status  InsertArc (ALGraph &G,KeyType v,KeyType w)
517 // 在图G中增加弧<v,w>，成功返回OK,否则返回ERROR
518 {
519     if(G.vexnum == 0) // 如果图不存在
520     {
521         printf ("该图不存在或未初始化\n");
522         return INFEASIBLE;
523     }
524     if(v == w) // 如果插入的是重边
525     {
526         printf ("插入的是重边\n");
527         return ERROR;
528     }

```



```
529     int flagv ==-1, flagw ==-1;
530
531     //找到插入点 v 和 w 的下标
532     int i =0;
533     while ( i<G.vexnum)
534     {
535         if(G.vertices [ i ].data.key == v)
536         {
537             flagv =i;
538         }
539         if(G.vertices [ i ].data.key == w)
540         {
541             flagw =i;
542         }
543         i++;
544     }
545
546     //如果找不到插入点 v 或 w
547     if(flagv == -1 || flagw == -1)
548     {
549         printf ("找不到要插入的顶点\n");
550         return ERROR;
551     }
552
553     ArcNode *pv = NULL;
554     ArcNode *pw = NULL;
555
556     //检查插入的是否为重复的边
557     pv = G.vertices [ flagv ].firstarc ;
558     while (pv)
559     {
```

```

560         if(pv->adjvex == flagw) //找到了重复的边
561         {
562             return ERROR;
563         }
564         pv = pv->nextarc;
565     }
566
567     //分别创建结构体 newv 和 neww, 构建新边
568     ArcNode *newv = (ArcNode *) malloc(sizeof(ArcNode));
569     newv->adjvex = flagw; //邻接点下标为 w
570     newv->nextarc = NULL; //下一条边为空
571     if(G.vertices [ flagv ]. firstarc  != NULL) //如果 v 有边
572     {
573         newv->nextarc =G.vertices[ flagv ]. firstarc ; //新边指向 v 的第
                    一条边
574     }
575     G.vertices [ flagv ]. firstarc  = newv; //更新头指针, 即 v 的第一条边
                    为新边
576
577     //和上面的操作类似
578     ArcNode *neww = (ArcNode *) malloc(sizeof(ArcNode));
579     neww->adjvex = flagv; //邻接点下标为 v
580     neww->nextarc = NULL; //下一条边为空
581     if(G.vertices [ flagw ]. firstarc  != NULL) //如果 w 有边
582     {
583         neww->nextarc =G.vertices[ flagw ]. firstarc ; //新边指向 w 的第
                    一条边
584     }
585     G.vertices [ flagw ]. firstarc  = neww; //更新头指针, 即 w 的第一条
                    边为新边
586

```

```
587     G.arcnum++; //边数加 1
588     return OK; //插入成功
589 }
590
591 status DeleteArc(ALGraph &G,KeyType v,KeyType w)
592 // 在图G中删除弧<v,w>, 成功返回OK,否则返回ERROR
593 {
594     if(G.vexnum == 0) // 图不存在
595     {
596         printf ("该图不存在或未初始化\n");
597         return INFEASIBLE;
598     }
599     if(v == w) // 如果v和w相等,说明删除环,返回错误
600     {
601         printf ("你输入的是环\n");
602         return ERROR;
603     }
604     int i = 0;
605     int flagv = -1;    // 用来记录下标
606     int flagw = -1;
607     int sign = 0; // 用来标记是否有边
608
609     // 查找边<v,w>对应的顶点下标
610     while (i<G.vexnum)
611     {
612         if(G.vertices[i].data.key == v)
613         {
614             flagv = i ;
615         }
616         if(G.vertices[i].data.key == w)
617         {
```

```

618         flagw = i ;
619     }
620     i++;
621 }
622
623 if(flagv == -1 || flagw == -1) // 边<v,w>不存在
624 {
625     printf ("不存在这条边的顶点\n");
626     return ERROR;
627 }
628
629 // 遍历v顶点的出边
630 ArcNode * getv = NULL;
631 ArcNode * getw = NULL;
632 ArcNode * pre = NULL;
633
634 // 查找边<v,w>对应的出边，然后删除
635 getv = G.vertices [ flagv ]. firstarc ;
636 pre = G.vertices [ flagv ]. firstarc ;
637
638 while (getv)
639 {
640     if(getv->adjvex == flagw) // 如果找到边<v,w>
641     {
642         sign = 1; // 有边标记为1
643
644         if(getv == pre) // 如果边是第一条出边
645         {
646             G.vertices [ flagv ]. firstarc = getv->nextarc; // 直接将
647                 该边的下一条边作为第一条出边
648             free (getv); // 释放当前边

```

```

648         getv = NULL;
649         pre = NULL;
650         break;
651     }
652     else // 如果边不是第一条出边
653     {
654         pre->nextarc = getv->nextarc; // 将该边从前一条出边
        // 的nextarc中删掉，接上后一条边
655         pre = getv; // 更新前一条边的指针到当前边
656         free(getv); // 释放当前边
657         getv = NULL;
658         break;
659     }
660 }
661 pre = getv; // 前指针更新为当前边
662 getv = pre->nextarc; // 当前边更新为下一条出边
663 }
664
665 if(sign == 0) // 如果没有找到边，返回错误
666 {
667     printf("不存在这条边\n");
668     return ERROR;
669 }
670 getw = G.vertices[flagw].firstarc;
671 pre = G.vertices[flagw].firstarc;
672 // 如果图是无向图，还需要删除边<w,v>
673 while (getw)
674 {
675     if(getw->adjvex == flagv) // 如果找到边<w,v>
676     {
677         if(getw == pre) // 如果边是第一条出边

```

```

678         {
679             G.vertices[flagw].firstarc = getw->nextarc; //直
                接将该边的下一条边作为第一条出边
680             free(getw); //释放当前边
681             getw = NULL;
682             break;
683         }
684         else //如果边不是第一条出边
685         {
686             pre->nextarc = getw->nextarc; //将该边从前一条
                出边的nextarc中删掉，接上后一条边
687             pre = getw; //更新前一条边的指针到当前边
688             free(getw); //释放当前边
689             getw = NULL;
690             break;
691         }
692     }
693     pre = getw; //前指针更新为当前边
694     getw = pre->nextarc; //当前边更新为下一条出边
695 }
696
697
698 G.arcnum--; //边数减1
699 return OK;
700 }
701
702 // (11) 深度优先搜索遍历:
703 // 函数名称是DFSTraverse(G,visit());
704 // 初始条件是图G存在;
705 // 操作结果是图G进行深度优先搜索遍历,
706 // 依次对图中的每一个顶点使用函数visit访问一次,
    
```

```
707 // 且仅访问一次;
708
709 // 定义一个标记数组, 用于标记每个顶点是否已经被遍历过
710 int flag11 [100];
711
712 // 定义一个深度优先搜索函数, 并传入图G、visit函数和当前遍历的节点
713 void dfs(ALGraph G, void (* visit )(VertexType), int nownode)
714 {
715     // 首先访问当前节点
716     visit (G.vertices [nownode].data);
717     // 将当前节点标记为已遍历过
718     flag11 [nownode] = 1;
719
720     // 遍历当前节点的所有邻接节点
721     ArcNode * p = G.vertices [nownode]. firstarc ;
722     while (p)
723     {
724         // 如果邻接节点没有被遍历过, 则递归遍历它
725         if( flag11 [p->adjvex] == 0)
726         {
727             dfs(G, visit ,p->adjvex);
728         }
729         p = p->nextarc;
730     }
731 }
732
733 // 定义图的深度优先搜索遍历函数
734 status DFSTraverse(ALGraph G,void (*visit)(VertexType))
735 {
736     // 对图中每个顶点进行标记初始化
737     memset(flag11,0, sizeof( flag11 ));
```

```
738 //如果图不存在，返回INFEASIBLE（不可行）
739 if(G.vexnum == 0)
740 {
741     printf("该图不存在或未初始化\n");
742     return INFEASIBLE;
743 }
744 int i ;
745 //对每个未被遍历过的顶点进行深度优先搜索
746 for( i=0;i<G.vexnum ;i++)
747 {
748     if( flag11[i] == 0)
749     {
750         dfs(G, visit ,i);
751     }
752 }
753 return OK;
754 }
755
756
757 //（12）广度优先搜索遍历：函数名称是BFSTraverse(G,visit()); 初始条
    件是图G存在；
758 // 操作结果是图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。
759 int flag12[100] ;
760 void BFS(ALGraph G,void (* visit )(VertexType), int i)
761 {
762     int head = 0, tail = 0; //定义头指针head和尾指针tail
763     int Que[100]; //一个队列Que，用于存放待遍历的顶点。
764     Que[0] = i;
765     while (head<=tail)
766     {
```



```

767
768     visit (G.vertices [Que[head]]. data);
769     ArcNode * p = G.vertices [Que[head]]. firstarc ;
770     while (p)
771     {
772         if (flag12[p->adjvex] == 0)
773         {
774
775             tail ++;
776             Que[ tail ] = p->adjvex;
777             flag12[p->adjvex]++;
778
779         }
780         p = p->nextarc;
781     }
782     head++;
783 }
784 }
785
786 status BFSTraverse(ALGraph G,void (*visit)(VertexType))
787 // 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
788 {
789     // 请在这里补充代码，完成本关任务
790     /***** Begin *****/
791     memset(flag12,0,sizeof (flag12)); // 将flag12数组全部置为0
792     if (G.vexnum == 0) // 图不存在
793     {
794         printf ("该图不存在或未初始化\n");
795         return INFEASIBLE;
796     }

```

```

797     int i ;
798     for( i =0;i< G.vexnum ;i++)
799     {
800         if(flag12[i] == 0)
801         { // 遍历所有顶点，如果该顶点未被访问，则将其标记为已访问并调用BFS函数
802             flag12[i]=1;
803             BFS(G,visit,i);
804         }
805     }
806     return OK;
807
808
809
810     /***** End *****/
811 }
812
813 // visit 函数
814
815 void visit (VertexType p)
816 {
817     printf ("%d %s",p.key,p.others );
818 }
819
820 int * VerticesSetLessThanK(ALGraph G,int v,int k) // 函数定义，返回
            指针类型，输入参数包括图G、起始顶点v和距离上限k
821 {
822     k--; // 由于是从起始点算起的距离，所以距离上限k需要减1
823     if(G.vexnum == 0) // 如果图不存在，返回NULL
824     {
825         printf ("该图不存在或未初始化\n");

```

```
826         return NULL;
827     }
828     int record[100] = {0}; // 记录访问过的结点，初始化为0（表示未
        访问过）
829     int i = 0;
830     int flag = -1; // flag变量初始化为-1（表示没找到起始结点）
831
832     // 遍历图的顶点，找到起始结点v，记录其位置到flag变量中
833     for( ; i < G.vexnum ; i++)
834     {
835         if(G.vertices[i].data.key == v)
836         {
837             flag = i;
838             break;
839         }
840     }
841     record[flag] = 1; // 标记起始结点v已经被访问过
842
843     if(flag == -1) // 如果未找到起始点，返回NULL
844     {
845         printf("找不到结点\n");
846         return NULL;
847     }
848     static int srr[100]; // 静态数组用于存储距离小于k的顶点集合
849     int num = 0; // num变量用于记录已经存储了多少个顶点
850     srr[num++] = flag; // 将起始点v加入到顶点集合中
851     // 下面进行查找
852     int Que[100][2]; // 二维数组表示队列，用于存储待访问的结点及
        其距离
853     memset(Que, 0, sizeof(Que)); // 初始化队列为0（表示未被访问过）
854     int head = 0, tail = 0; // 队列的头和尾指针
```

```

855     Que[head][0] = flag; //起始结点v作为队列的第一个元素
856     Que[head][1] = 0; //起始结点v的距离为0
857
858     // 队列非空且队列中第一个结点距离不超过k的情况下，进行队列的
        遍历
859     while (head <= tail && Que[head][1] != (k+1))
860     {
861
862         ArcNode * p = G.vertices[Que[head][0]].firstarc; //获取队头
            元素的邻接链表
863         while (p)
864         {
865             if (record[p->adjvex] == 0) //如果邻接结点未被访问过
866             {
867                 if (Que[head][1] <= (k-1))
868                 {
869                     srr[num++] = p->adjvex; //将邻接结点加入到顶点
                        集合中
870                 }
871
872                 tail++; //队列尾指针加1
873                 Que[tail][0] = p->adjvex; //将邻接结点加入到队列中
874                 Que[tail][1] = Que[head][1] + 1; //计算邻接结点距离
875                 record[p->adjvex]++; //标记邻接结点已经被访问过
876             }
877             p = p->nextarc; //遍历下一个邻接结点
878         }
879         head++; //处理完队头结点，队头指针加1
880     }
881     srr[num] = -1; //将数组以-1结尾，以便在函数外部访问到数组长
        度
    
```

```

882
883     return srr; // 返回存储顶点集合的数组指针
884 }
885
886 int ShortestPathLength(ALGraph G, int v, int w)
887 {
888     if (G.vexnum == 0) // 图不存在
889     {
890         printf("该图不存在或未初始化\n");
891         return INFEASIBLE;
892     }
893     int head = 0, tail = 0; // 定义队列头和尾
894     int record[100] = {0}; // 记录每个节点是否被访问过
895     int arr[100][2]; // 定义存储节点和距离的队列
896     memset(arr, 0, sizeof(arr)); // 初始化队列
897     int i = 0;
898     int flag = -1; // 记录v节点的索引值
899     int flagw = -1; // 记录w节点的索引值
900     for (; i < G.vexnum; i++) // 遍历所有节点
901     {
902         if (G.vertices[i].data.key == v) // 找到v节点
903         {
904             flag = i;
905         }
906         if (G.vertices[i].data.key == w) // 找到w节点
907         {
908             flagw = i;
909         }
910     }
911     if (flag == -1 || flagw == -1) // 如果v或w节点不存在
912     {

```

```

913         printf("没有找到v对应的结点\n");
914         return INFEASIBLE;
915     }
916     arr[head][0] = flag; // 首个节点为v节点
917     while (head <= tail) // 当队列非空时循环
918     {
919         ArcNode *p = G.vertices[ arr[head][0]]. firstarc ; // 找到当前节
           点的第一条边
920
921         if (G.vertices[ arr[head][0]]. data.key == w) // 如果找到w节点
922         {
923             return arr[head][1]; // 返回距离
924         }
925         while (p) // 遍历当前节点的所有边
926         {
927             if (record[p->adjvex] == 0) // 如果该节点未被访问过
928             {
929                 tail++; // 队列尾部加入该节点
930                 arr[ tail ][0] = p->adjvex; // 存储节点
931                 arr[ tail ][1] = arr[head][1] + 1; // 存储距离
932                 record[ arr[head][0] ]++; // 标记该节点已被访问
933             }
934             p = p->nextarc; // 遍历下一条边
935         }
936         head++; // 处理下一个节点
937     }
938     return -1; // 如果没有找到路径, 返回-1
939 }
940
941 // 定义一个全局数组flag16用于标记顶点是否被访问过
942 int flag16[100] = {0};

```

```
943
944 // 定义深度优先搜索函数dfs，其中G为图，nownode为当前节点
945 void dfs(ALGraph G, int nownode) {
946     // 将当前节点标记为已访问
947     flag16[nownode] = 1;
948
949     // 遍历当前节点的邻接节点
950     ArcNode *p = G.vertices[nownode].firstarc ;
951     while (p) {
952         // 如果当前邻接节点未被访问，则递归调用dfs函数
953         if (flag16[p->adjvex] == 0) {
954             dfs(G, p->adjvex);
955         }
956         // 继续遍历下一个邻接节点
957         p = p->nextarc;
958     }
959 }
960
961 // 定义连通分量计数函数ConnectedComponentsNums，其中G为图
962 int ConnectedComponentsNums(ALGraph G) {
963     // 每次使用之前要将flag16数组清空
964     memset(flag16, 0, sizeof(flag16));
965
966     int i;
967     // 当图为空或未初始化时，返回0
968     if (G.vexnum == 0) {
969         printf("为初始化或者为空\n");
970         return 0;
971     }
972
973     int count = 0;
```

```
974 // 遍历所有顶点
975 for (i = 0; i < G.vexnum; i++) {
976     // 如果当前顶点未被访问，则递归调用dfs函数，并将计数器
        count加1
977     if (flag16[i] == 0) {
978         count++;
979         dfs(G, i);
980     }
981 }
982
983 // 返回连通分量的计数器count
984 return count;
985 }
986
987 status SaveGraph(ALGraph G, char FileName[]) // 保存图的数据到文件
988 {
989     if(G.vexnum == 0) // 如果图是空的，直接返回错误
990     {
991
992         printf("图是空的\n");
993
994         return -1;
995
996     }
997     FILE * fp = fopen(FileName, "w"); // 打开文件，只可写入
998     if(fp == NULL)
999     {
1000         return ERROR; // 如果无法打开文件，返回错误
1001     }
1002
1003     // 先写入 结点数 和 边数
```



```

1004     fprintf (fp, "%d %d\n", G.vexnum, G.arcnum); //写入顶点数和边数
1005     // 再写入顶点
1006     for (int k = 0; k < G.vexnum; k++) //遍历每一个顶点
1007     {
1008         fprintf (fp, "%d %s\n", G.vertices[k].data.key, G.vertices[k].data
            .others); //写入顶点的key和others
1009     }
1010     //下面输入每个结点对应的边
1011
1012     for (int i = 0; i < G.vexnum; i++) //遍历每一个结点
1013     {
1014         ArcNode * p = G.vertices[i].firstarc; //从顶点的第一条边开
            始遍历
1015         while (p)
1016         {
1017             fprintf (fp, "%d ", p->adjvex); //写入边的邻接点编号
1018             p = p->nextarc; //遍历下一条边
1019         }
1020         fprintf (fp, "-1\n"); //一条边结束后写入-1
1021     }
1022     fclose (fp); //关闭文件
1023     return OK; //返回成功
1024 }
1025
1026 status LoadGraph(ALGraph &G, char FileName[]) //从文件中读取图的数
    据
1027 {
1028     if (G.vexnum != 0) //如果图不为空, 则无法读取
1029     {
1030         printf ("这个图不是空的, 无法读取\n");
1031     }

```

```

1032 FILE *fp = fopen(FileName,"r"); //打开文件，只可读取
1033 if(fp == NULL)
1034 {
1035     return ERROR; //如果无法打开文件，返回错误
1036 }
1037 fscanf(fp, "%d %d\n", &G.vexnum, &G.arcnum); //读取顶点数和边数
1038 for(int i = 0; i < G.vexnum; i++) //遍历每一个顶点
1039 {
1040     fscanf(fp, "%d %s\n", &G.vertices[i].data.key, G.vertices[i].data
        .others); //读取顶点的key和others
1041     G.vertices[i].firstarc = NULL; //顶点的第一条边为NULL
1042 }
1043 for(int k = 0; k < G.vexnum; k++) //遍历每一个结点
1044 {
1045     ArcNode *p = G.vertices[k].firstarc; //从顶点的第一条边开始
        遍历
1046     ArcNode *newnode = (ArcNode *) malloc(sizeof(ArcNode)); //
        新建一个结点
1047     fscanf(fp, "%d", &newnode->adjvex); //读取新结点的邻接点编
        号
1048     newnode->nextarc = NULL; //将新结点的下一条边设为NULL
1049     while(newnode->adjvex != -1) //如果读取的邻接点编号不是-1
1050     {
1051         if(G.vertices[k].firstarc == NULL) //如果当前顶点的第一
            条边为NULL
1052         {
1053             G.vertices[k].firstarc = newnode; //将新结点设为该
                顶点的第一条边
1054             p = G.vertices[k].firstarc; //令p指向该顶点的第一
                条边
1055         }
    }

```

```

1056         else {
1057             p->nextarc = newnode; // 将新结点接到p指向的边的后
                面
1058             p = newnode; // 令p指向新结点
1059         }
1060         newnode = (ArcNode * ) malloc( sizeof( ArcNode)); // 新建
                一个结点
1061         fscanf( fp, "%d ", &newnode->adjvex); // 读取新结点的邻接点
                编号
1062         newnode->nextarc = NULL; // 将新结点的下一条边设为
                NULL
1063     }
1064
1065 }
1066 fclose( fp); // 关闭文件
1067 return OK; // 返回成功
1068 }
1069
1070 void menu()
1071 {
1072     for( int k = 0; k<= 119 ;k++)
1073     {
1074         putchar( '-' );
1075     }  putchar( '\n' );
1076     printf( "1.创建一个图\n");
1077     printf( "2.删除一个图\n");
1078     printf( "3.查询已经创建的图\n");
1079     printf( "4.查找一个图和进行操作\n");
1080     printf( "0.退出多个图的管理\n");
1081
1082     printf( "      ^      /\n");

```

```

1083     printf ("      /\ 7      □ _\n");
1084     printf ("      / |      / / \n");
1085     printf ("      | Z _,<      / /['F\n");
1086     printf ("      |      [F      / > \n");
1087     printf (" Y      ' /      \n");
1088     printf (" ?● ? ●      ?? <      \n");
1089     printf (" () ^      | \ <\n");
1090     printf (" >? ?_ 彳      | / / \n");
1091     printf ("      / ^      / ?<| \ \ \n");
1092     printf (" [F_?      ( /      | / / \n");
1093     printf ("      7      | / \n");
1094     printf ("      >—r - - '— _ \n");
1095     for( int k = 0; k<= 119 ;k++)
1096     {
1097         putchar('—');
1098     }   putchar('\n');
1099 }
1100
1101 void menu2()
1102 {
1103     for( int k = 0; k<= 119 ;k++)
1104     {
1105         putchar('—');
1106     }
1107     putchar('\n');
1108     printf ("          Menu for Graph On Sequence Structure \n");
1109     //      printf
1110     ("-----\n");
1111     ;
1112     printf ("          1. 创建初始化图          7. 插入
          顶点\n");

```

1111	printf ("	2. 销毁图	8. 删除
	顶点\n");		
1112	printf ("	3. 查找顶点	9. 插入
	弧\n");		
1113	printf ("	4. 顶点赋值	10. 删除
	弧\n");		
1114	printf ("	5. 获取第一邻接点	11. 深度
	优先搜索\n");		
1115	printf ("	6. 获取下一邻接点	12. 广度
	优先搜索\n");		
1116	printf ("	13.查看图关系\n");	
1117	for(int k = 0; k<= 119 ;k++)		
1118	{		
1119	putchar('-');		
1120	} putchar('\n');		
1121	printf ("	14.距离小于k的顶点集合	15.顶
	点间最短路径\n");		
1122	printf ("	16.图的连通分量	17.保
	存到文件\n");		
1123	printf ("	18.从文件里面加载	\n");
1124	printf ("	0. exit	\n");
1125	for(int k = 0; k<= 119 ;k++)		
1126	{		
1127	putchar('-');		
1128	} putchar('\n');		
1129	// printf ("一些附加的功能");		
1130			
1131	printf ("	请选择你的操作[0~13]:");	
1132	putchar('\n');		
1133	for(int k = 0; k<= 119 ;k++)		
1134	{		

```

1135     putchar(' ');
1136 }
1137 putchar('\n');
1138
1139 printf("    ^          /\n");
1140 printf("    /\ 7      □ _\n");
1141 printf("    / |      / /\n");
1142 printf("    | Z _,<    /    /['\n");
1143 printf("    |          ['    /    > \n");
1144 printf(" Y          '    /    \n");
1145 printf("  ?● ? ●    ?? <    \n");
1146 printf("  () ^      | \    \n");
1147 printf("  >? ?_ 彳    | / /\n");
1148 printf("    / ^      / ?<| \ \n");
1149 printf("    ['?    ( /    | / /\n");
1150 printf("    7          | /\n");
1151 printf("    >—r - - '?— _\n");
1152
1153 putchar('\n');
1154
1155 }
1156
1157 void fun01()          // 这个函数负责多线性表的管理
1158 {
1159     menu(); // 调用菜单函数
1160     int a ; // 定义整型变量a
1161     printf("请输入一个命令\n");
1162     scanf("%d",&a); // 输入命令，保存在a中
1163     while (a) // 如果a的值不为0，则循环执行代码块
1164     {
1165         fflush ( stdin ); // 清空输入流，防止上一次操作结束后影响本次

```

操作

```
1166     int feedback; // 定义整型变量feedback
1167     switch (a) { // 根据不同命令进行不同的操作
1168         case 1: // 如果命令为1
1169             printf("现在进行创建一个新的图\n");
1170             printf("请输入你想创建的图的名字\n");
1171             char name1[30]; // 定义字符串变量name1, 长度为30
1172             scanf("%s",name1); // 输入图的名字, 保存在name1中
1173             int i , flag ; flag = 0; // 定义整型变量i和flag, flag 初
                始化为0
1174             // 要进行名字的判断
1175             for( i =0;i<graphs.length;i++)// 遍历所有已经存在的
                图
1176             {
1177                 if(strcmp(name1,graphs.elem[i].name) == 0) // 如
                    果名字已经被使用, 则提示创建失败
1178                 {
1179                     printf("该图已经存在, 创建失败\n");
1180                     flag = 1; // 将flag的值设为1, 表示创建失败
1181                 }
1182             }
1183             if( flag == 0) // 如果flag的值为0, 表示没有相同的名字, 则创建新图并保存名字
1184             {
1185                 strcpy (graphs.elem[graphs.length].name,name1); //
                    使用strcpy函数将name1中的字符串复制到graphs
                    .elem[graphs.length].name中
1186                 graphs.length++; // 将已经存在的图的数量加1
1187                 printf("创建成功\n");
1188             }
1189             break;
```

```
1190         case 2: // 如果命令为2
1191             int flag2 ; // 定义整型变量flag2
1192             printf ("现在进行删除图的操作\n");
1193             printf ("请输入你想删除的图的名字\n");
1194             char name2[30]; // 定义字符串变量name2, 长度为30
1195             scanf ("%s",name2); // 输入要删除的图的名字, 保存在
                                   name2中
1196
1197             flag2 = -1; // flag2 初始化为-1
1198
1199             // 要进行名字的判断
1200             for( i =0;i<graphs.length;i++)// 遍历所有已经存在的
                                   图
1201             {
1202                 if(strcmp(name2,graphs.elem[i].name) == 0) // 如
                                   果找到了要删除的图的名字, 则将flag2设置为
                                   该图在已存在图数组中的下标
1203                 {
1204                     flag2 = i;
1205                 }
1206             }
1207
1208             if( flag2 == -1) // 如果flag2的值还是-1, 表示没有找到
                                   要删除的图的名字, 则无法删除
1209             {
1210                 printf ("该图不存在, 无法删除\n");
1211             }
1212             else{ // 如果找到了要删除的图的名字
1213                 if(1)
1214                 {
1215                     int k;
```



```

1216         for( k = flag2 ;k < graphs.length-1 ;k++) //将
           该图在已存在图数组中的下标之后的所有图
           向前移动一个位置
1217         {
1218             graphs.elem[k] = graphs.elem[k+1];
1219         }
1220         graphs.length--; // 已存在图的数量减1
1221         printf ("删除成功\n");
1222     }
1223 }
1224 break;
1225 case 3: // 如果命令为3
1226     printf ("现在进行查询创建了哪些图\n");
1227     printf ("所有的图如下:\n");
1228     for(i = 0; i<graphs.length ;i++) // 遍历已存在的图的
           数组，并按顺序输出每张图的名字
1229     {
1230         printf ("%d)  %s\n",i+1,graphs.elem[i ].name);
1231     }
1232     break;
1233 case 4: // 如果命令为4
1234     printf ("现在进行图的查找和操作\n");
1235     printf ("请输入你想查找和操作的图的名字\n");
1236     char name3[30]; // 定义字符串变量name3，长度为30
1237     scanf ("%s",name3); // 输入要查找和操作的图的名字，
           保存在name3中
1238
1239     int flag3 ;flag3 = -1; // 定义整型变量flag3，初始化为
           -1
1240     for( i =0 ; i<graphs.length ;i++) // 遍历已存在的图的
           数组
    
```

```
1241         {
1242             if(strcmp(graphs.elem[i].name,name3) == 0) // 如果
                找到了要查找和操作的图的名字，则将flag3设
                置为该图在已存在图数组中的下标
1243         {
1244             flag3 = i;
1245         }
1246     }
1247
1248     if(flag3 == -1) // 如果flag3还是-1，表示没有找到要查
                找和操作的图的名字，则提示不存在这个图
1249     {
1250         printf("不存在这个图\n");
1251         system("pause"); // 暂停程序的执行，等待用户按下
                任意键
1252     }
1253     else { // 如果找到了要查找和操作的图的名字
1254         fun02(graphs.elem[flag3].G); // 调用fun02函数对该
                图进行操作
1255     }
1256     break;
1257
1258     default: // 如果命令无法识别，则提示输入错误，并重新显
                示菜单
1259         printf("输入的命令错误，请再次输入");
1260     }
1261     printf("请输入下一个命令\n");
1262     scanf("%d",&a); // 提示输入下一个命令，保存在a中
1263     system("cls"); // 清空控制台的输出，准备显示菜单
1264     menu(); // 再次显示菜单
1265 }
```

```
1266 }
1267
1268
1269 void fun02(ALGraph &G)
1270 {
1271     system("cls");    // 清空屏幕
1272     printf ("图存在鸭鸭\n");
1273     printf ("现在对这个图进行操作\n");
1274     printf ("别忘记初始化这个图鸭\n");
1275     int order;    // 来接收命令
1276     menu2();    // 展示菜单
1277     scanf ("%d",&order);
1278     while (order)
1279     {
1280         fflush ( stdin ); // 这里的清空输入流是防止上一次操作结束后输
                               入了数据而影响本次操作
1281         int feedback;
1282         switch (order) {
1283             int feedback;
1284             case 1:
1285                 printf ("请输入顶点序列和关系对序列:\n");
1286
1287                 VertexType V[30];    // 装顶点集合
1288                 KeyType VR[100][2]; // 装边集合
1289                 int i,j; i =0;    // 用来计数
1290                 do {
1291                     scanf ("%d%s",&V[i].key,V[i].others);
1292                 } while(V[i++].key!=-1);
1293                 i=0;
1294                 do {
1295                     scanf ("%d%d",&VR[i][0],&VR[i][1]);
```

```
1296         } while(VR[i++][0]!=-1);
1297         feedback = CreateCraph(graphs.elem[graphs.length-1].G,
1298                                V,VR);
1299         if(feedback == OK)
1300         {
1301             printf("图初始化成功\n");
1302         }
1303         else {
1304             printf("初始化失败\n");
1305         }
1306         break;
1307     case 2:
1308         printf("现在进行图的销毁操作\n");
1309         feedback = DestroyGraph(G);
1310         if(feedback == OK)
1311         {
1312             printf("图销毁成功了\n");
1313         }
1314         else {
1315             printf("线性表未初始化或者不存在\n");
1316         }
1317         break;
1318     case 3:
1319         printf("现在进行查找顶点的操作\n");
1320         printf("请输入你想查找的顶点的关键字\n");
1321         int key ;    // 来存储关键字
1322         scanf("%d",&key);
1323         feedback = LocateVex(G,key);
1324         if(feedback != -1)
1325         {
1326             printf("所要查找的关键字为 %d 的顶点的位置序号
```

```

        为 %d\n",key,feedback);
1326         printf ("具体信息为%d %s\n",G.vertices[feedback].
            data.key,G.vertices[feedback].data.others);
1327     } else {
1328         printf ("所要查找的顶点不存在\n");
1329     }
1330     break;
1331 case 4:
1332     printf ("现在进行顶点赋值的操作\n");
1333     printf ("请输入你想对哪一个关键字进行操作\n");
1334     int key4;        // 存储关键字
1335     scanf ("%d",&key4);
1336     printf ("请输入你想改变的关键字和名称\n");
1337     VertexType value;
1338     scanf ("%d %s",&value.key,value.others);
1339     feedback = PutVex(G,key4,value);
1340     if (feedback == OK)
1341     {
1342         printf ("操作成功\n");
1343     }
1344     break;
1345 case 5:
1346     printf ("现在进行获取第一邻接点的操作\n");
1347     printf ("输入你想操作的关键字\n");
1348     int key5;        // 存储关键字
1349     scanf ("%d",&key5);
1350     feedback = FirstAdjVex(G,key5);
1351     if (feedback != -1)
1352     {
1353         printf ("获取成功,第一邻接点的位序是%d,具体信息
            为%d %s",feedback,G.vertices[feedback].data.key,

```

```

                                G.vertices[feedback].data.others);
1354         } else {
1355             printf("操作失败\n");
1356         }
1357         break;
1358     case 6:
1359         printf("现在进行获取下一邻接点的操作\n");
1360         printf("请输入G中两个顶点的位序，v对应G的一个顶
                                点,w对应v的邻接顶点\n");
1361         int v,w;    // 来存储顶点的值
1362         scanf("%d %d",&v,&w);
1363         feedback = NextAdjVex(G,v,w);
1364         if(feedback != -1)
1365         {
1366             printf("获取成功,下一邻接点的位序是%d,具体信息
                                为%d %s",feedback,G.vertices[feedback].data.key,
                                G.vertices[feedback].data.others);
1367         } else {
1368             printf("操作失败\n");
1369         }
1370         break;
1371     case 7:
1372         printf("现在进行插入顶点的操作\n");
1373         printf("输入你想插入的顶点的关键字和名称\n");
1374         VertexType v7;    // 存储插入的顶点信息
1375         scanf("%d %s",&v7.key,v7.others);
1376         feedback = InsertVex(G,v7);
1377         if(feedback == OK)
1378         {
1379             printf("插入成功\n");
1380         } else {

```

```
1381         printf ("插入失败\n");
1382     }
1383     break;
1384 case 8:
1385     printf ("现在进行删除顶点的操作\n");
1386     printf ("请输入你想删除的顶点的关键字\n");
1387     int key8;    // 存储关键字
1388     scanf ("%d",&key8);
1389     feedback = DeleteVex(G,key8);
1390     if(feedback == OK)
1391     {
1392         printf ("操作成功\n");
1393     }
1394     else {
1395         printf ("操作失败\n");
1396     }
1397     break;
1398 case 9:
1399     printf ("现在进行插入弧的操作\n");
1400     int v9,w9;    // 存储边的两个顶点
1401     printf ("输入你想插入的弧\n");
1402     scanf ("%d %d",&v9,&w9);
1403     feedback = InsertArc (G,v9,w9);
1404     if(feedback == OK)
1405     {
1406         printf ("操作成功\n");
1407     }
1408     else {
1409         printf ("操作失败\n");
1410     }
1411     break;
```

```
1412         case 10:
1413             printf("现在进行删除弧的操作\n");
1414             int v10,w10;      // 存储要删除的边的两个顶点
1415             printf("输入你想删除的弧\n");
1416             scanf("%d %d",&v10,&w10);
1417             feedback = DeleteArc(G,v10,w10);
1418             if(feedback == OK)
1419             {
1420                 printf("操作成功\n");
1421             } else {
1422                 printf("操作失败\n");
1423             }
1424             break;
1425         case 11:
1426             printf("现在进行深度优先搜索\n");
1427             feedback = DFSTraverse(G,visit);
1428             if(feedback == OK)
1429             {
1430                 printf("操作成功\n");
1431             }
1432             else {
1433                 printf("操作失败\n");
1434             }
1435             break;
1436         case 12:
1437             printf("现在进行广度优先搜索\n");
1438             feedback = BFSTraverse(G,visit);
1439             if(feedback == OK)
1440             {
1441                 printf("操作成功\n");
1442             }
```



```
1443         } else {
1444             printf ("操作失败\n");
1445
1446         }
1447         break;
1448     case 13:
1449         int u ; // 用来计数
1450         for(u = 0;u< G.vexnum ;u++)
1451         {
1452             printf ("%d %s ",G.vertices [u].data .key,G.vertices
1453                 [u].data .others );
1454             ArcNode * p = G.vertices [u]. firstarc ;
1455             while (p){
1456                 printf (" %d ",p->adjvex);
1457                 p = p->nextarc;
1458             }
1459             putchar ('\n');
1460         }
1461         break;
1462     case 14:
1463         printf ("现在进行查找小于k的顶点集合的操作\n");
1464         printf ("输入顶点的关键字\n");
1465         int key14; // 存储关键字
1466         scanf ("%d",&key14);
1467         printf ("输入距离k\n");
1468         int k;
1469         scanf ("%d",&k);
1470         int * p;
1471         p = VerticesSetLessThanK(G,key14,k);
1472         if(p == NULL)
```

```
1473         {
1474             printf ("操作失败\n");
1475         }
1476     else {
1477         printf ("距离小于%d 的顶点的集合是: \n",k);
1478         while ((*p) != -1)
1479         {
1480             printf ("%d %s\n",G.vertices[*p].data.key,G.
                vertices[*p].data.others);
1481
1482             p++;
1483         }
1484         printf ("操作成功\n");
1485     }
1486     break;
1487 case 15:
1488     printf ("现在进行顶点间的最短路程的操作\n");
1489     printf ("请输入顶点v和顶点w的关键字\n");
1490     int v15,w15;    // 存储两个关键字
1491     scanf ("%d %d",&v15,&w15);
1492     feedback = ShortestPathLength (G,v15,w15);
1493     if (feedback != -1)
1494     {
1495         printf ("最短路径为 %d \n",feedback);
1496
1497     } else {
1498         printf ("操作失败\n");
1499     }
1500     break;
1501 case 16:
1502     printf ("现在进行图的连通分量的计算\n");
```

```
1503         feedback = ConnectedComponentsNums(G);
1504         if(feedback != 0)
1505         {
1506             printf ("图的连通分量是%d\n",feedback);
1507
1508         }
1509
1510         break;
1511     case 17:
1512         printf ("现在进行文件的保存操作\n");
1513         printf ("请输入你想保存到哪一个文件\n");
1514         char name17[30]; // 存储要保存的文件名
1515         scanf ("%s",name17);
1516         feedback= SaveGraph(G,name17);
1517         if(feedback == OK)
1518         {
1519             printf ("保存成功\n");
1520         }
1521         break;
1522     case 18:
1523         printf ("现在进行文件的读取操作\n");
1524         printf ("你想读取哪一个文件的内容\n");
1525         char name18[30]; // 存储要读取的文件名
1526         scanf ("%s",name18);
1527         feedback = LoadGraph(G,name18);
1528         if(feedback == OK)
1529         {
1530             printf ("读取成功\n");
1531         }
1532         break;
1533     default :
```

```
1534         printf("命令输入有问题\n");
1535
1536
1537
1538     }
1539     putchar('\n');
1540     printf("请输入下一个命令\n");
1541     scanf("%d",&order);
1542     system("cls");
1543     if (order != 0)
1544     {
1545         menu2();
1546     }
1547     else {
1548         menu();
1549     }
1550
1551 }
1552 }
```