

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 CS2209

学 号 U202215643

姓 名 王国豪

指导教师 李剑军

报告日期 2023 年 6 月 1 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，编译代码等，避免由于不兼容导致的报错问题。

此外，操作演示系统时，要注意中文提示，避免因输入数据问题导致的程序报错

| | |
|---------|---------|
| 运行的操作系统 | Windows |
| 编译环境 | CLion |
| 代码的编码 | GBK |

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 | 13 |
| 1.5 实验小结 | 32 |
| 2 基于邻接表的图实现 | 34 |
| 2.1 问题描述 | 34 |
| 2.2 系统设计 | 34 |
| 2.3 系统实现 | 39 |
| 2.4 系统测试 | 46 |
| 2.5 实验小结 | 59 |
| 3 课程的收获和建议 | 60 |
| 3.1 基于顺序存储结构的线性表实现 | 60 |
| 3.2 基于链式存储结构的线性表实现 | 60 |
| 3.3 基于二叉链表的二叉树实现 | 61 |
| 3.4 基于邻接表的图实现 | 61 |
| 附录 A 基于顺序存储结构线性表实现的源程序 | 63 |
| 附录 B 基于链式存储结构线性表实现的源程序 | 107 |
| 附录 C 基于二叉链表二叉树实现的源程序 | 145 |
| 附录 D 基于邻接表图实现的源程序 | 194 |

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

```
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。

```
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  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
```


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 | (输入表长及数据后) 线性表创建成功 (并返回线性表已有数据) | 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

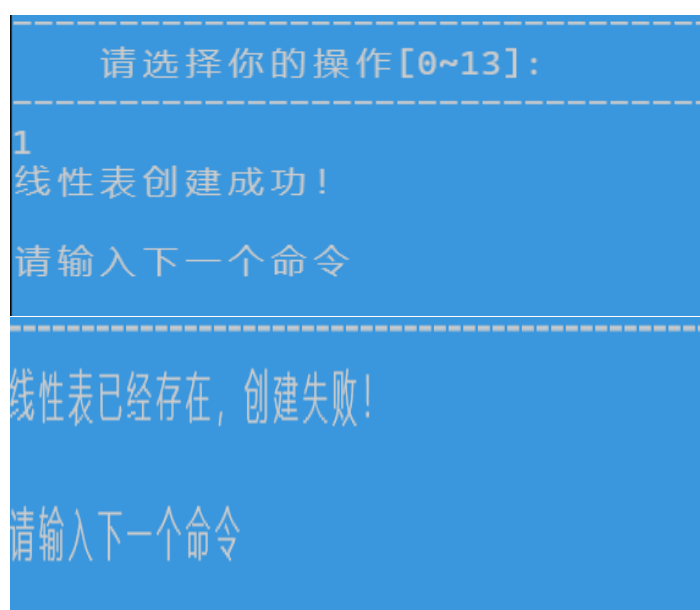


图 1-3 InitList 测试

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

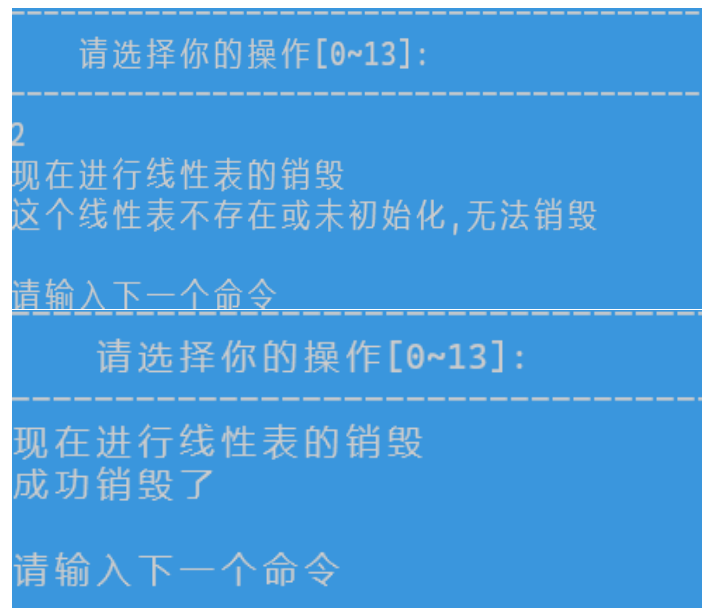


图 1-4 DestroyList 测试

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

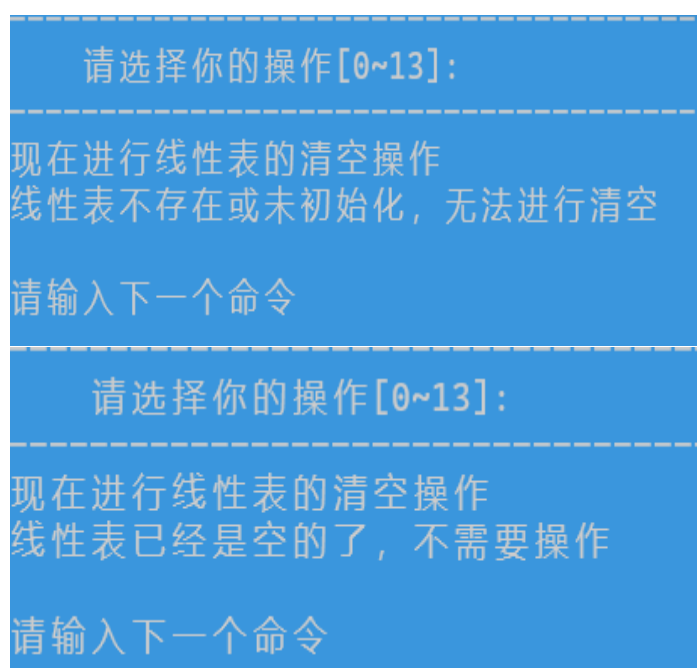


图 1-5 ClearList 测试

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

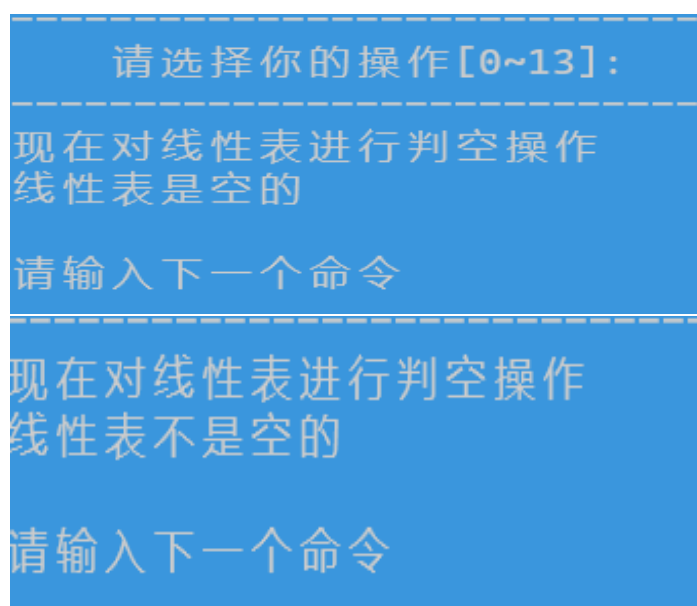


图 1-6 ListEmpty 测试

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

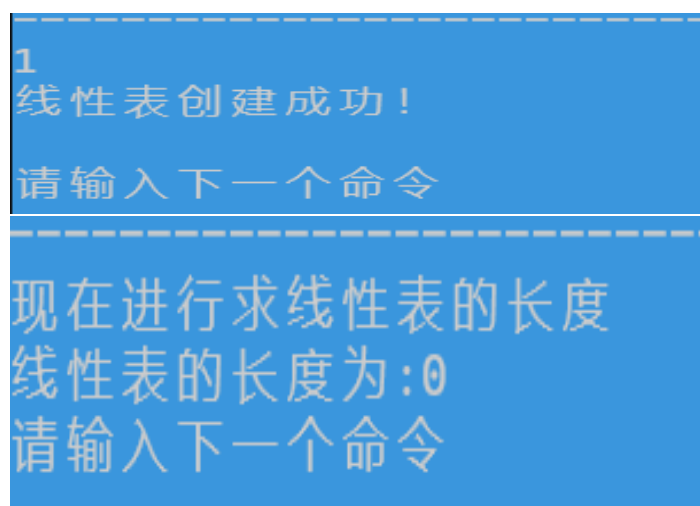


图 1-8 ListLength 测试 2

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

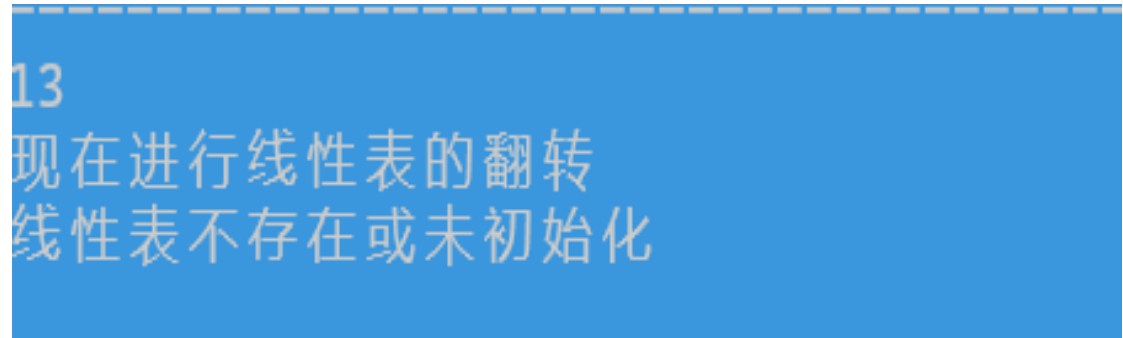
12
现在进行线性表的遍历
线性表不存在或未初始化，无法进行操作

图 1-27

1.4.13 void reverseList(LinkList L)

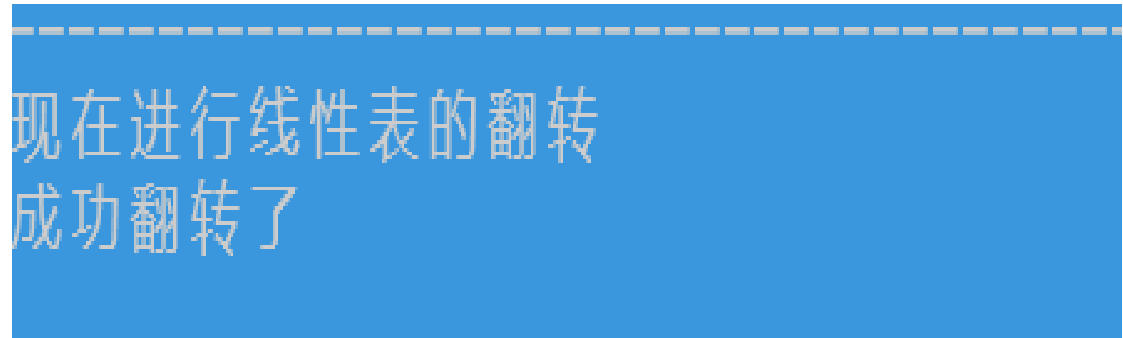
| 序号 | 进行此操作前的线性表状态 | 输入的函数参数 | 预计输出 | 操作后线性表的状态 |
|----|-------------------------------|---------|---------------------|-----------|
| 1 | 线性表未初始化 | 无 | 线性表不存在或未初始化，无法进行操作。 | 线性表不发生变化 |
| 2 | 线性表已经初始化 (初始化的数据为 9 3 -1 2 6) | 无 | 成功翻转 6 2 -1 3 9 | 线性表发生翻转。 |

表 1-13 线性表翻转元素



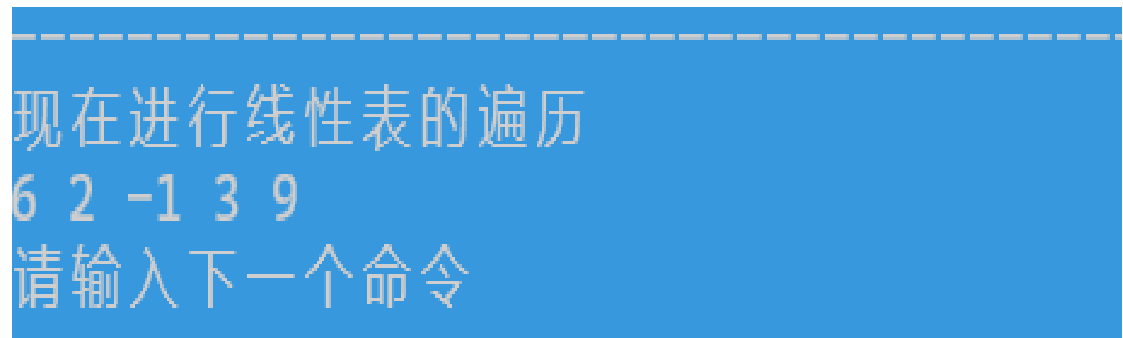
13
现在进行线性表的翻转
线性表不存在或未初始化

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

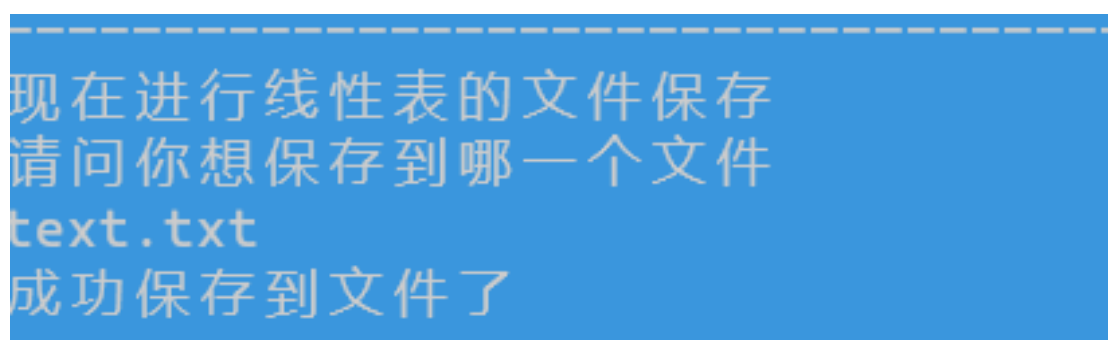


图 1-36

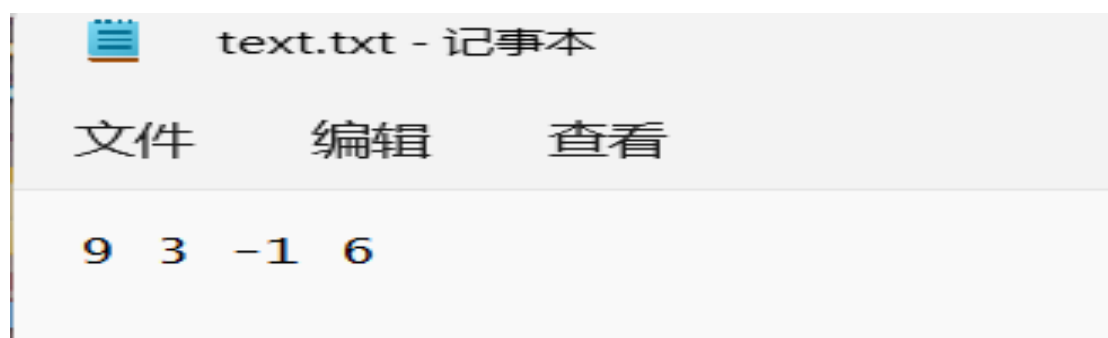


图 1-37

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

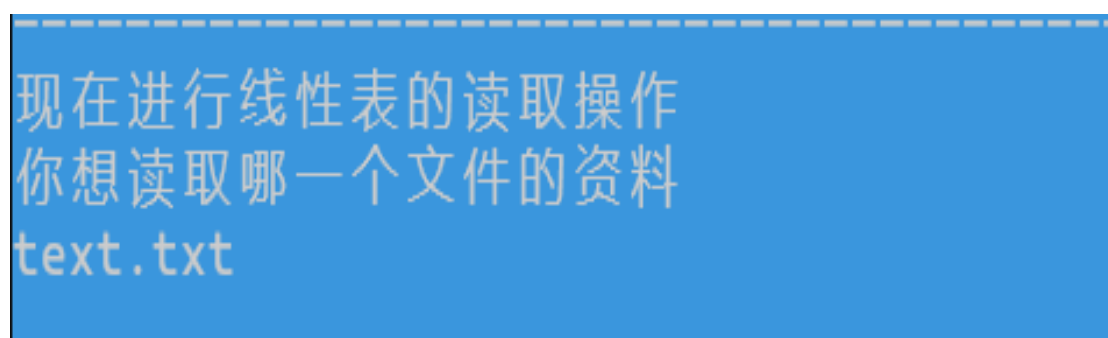


图 1-38

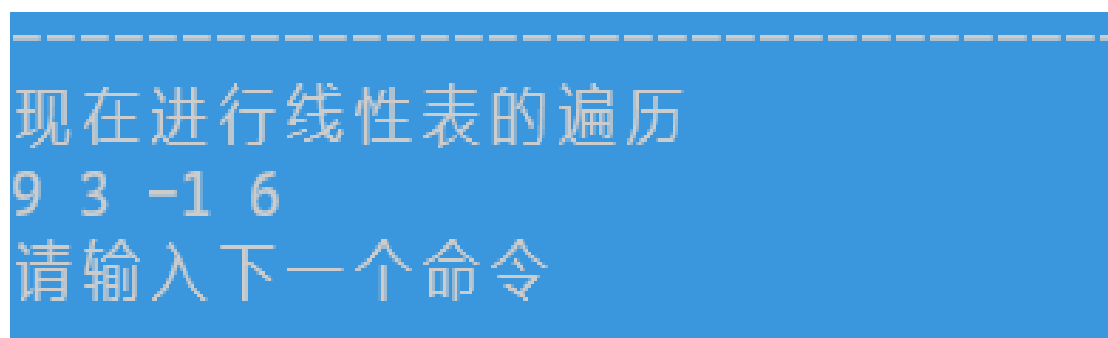


图 1-39

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，通过学习图的基本运算，我对图的概念、逻辑结构和物理结构有了更深入的理解，并且获得了实际操作图的能力。这些知识将对我在编程和数据处理领域中处理和管理数据结构起到积极的作用。

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。

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
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。

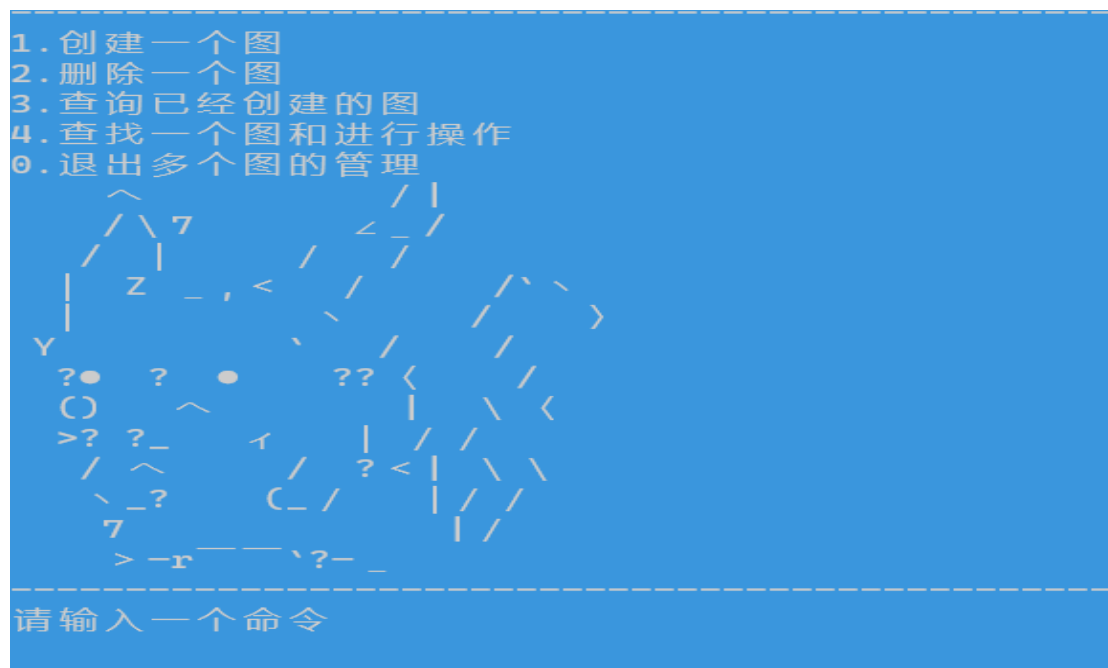


图 2-1 多个图管理菜单

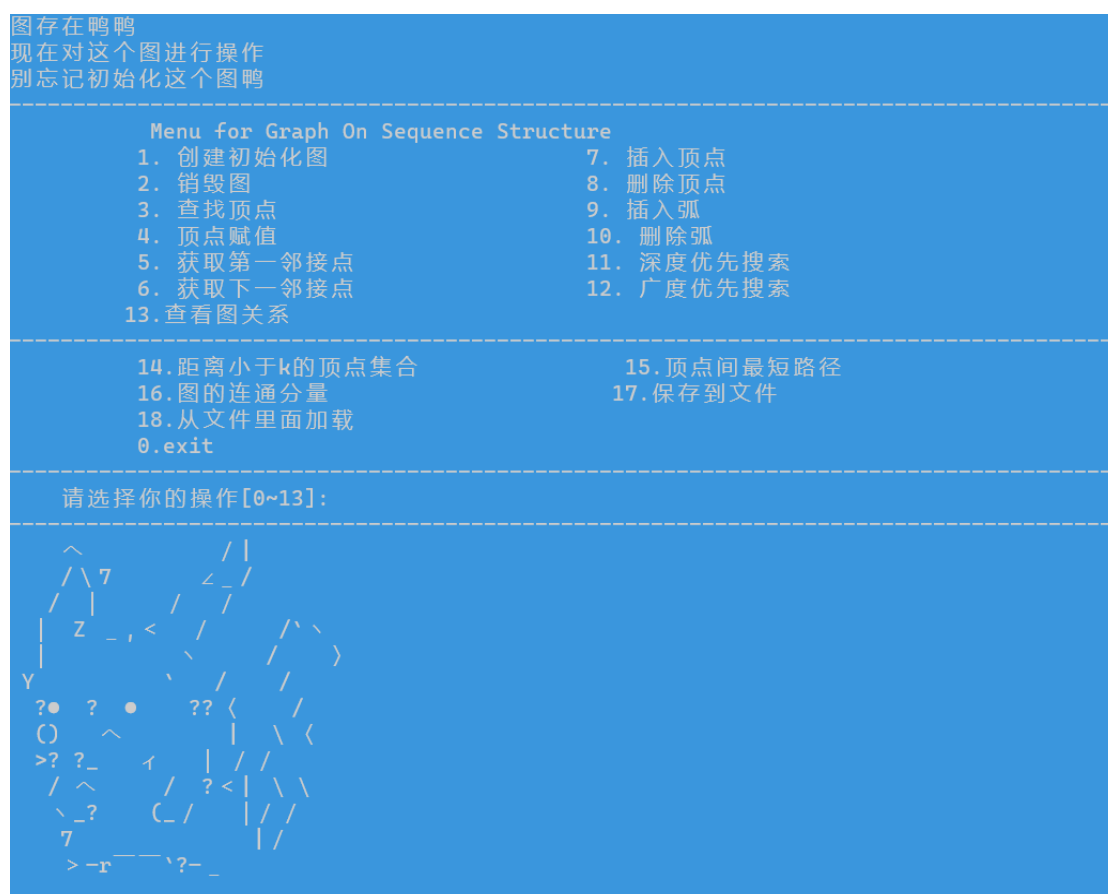


图 2-2 单个图管理菜单

2.3 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图进行判定，若无图，则返回 INFEASIBLE，在各函数具体设计思路中一般不再宿舍此条。

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

设计思路：

- 1) 首先判断图是否已经存在，如果已经存在则返回错误代码。
- 2) 使用标记数组和标记边的二维数组来辅助构建图。初始化标记数组和标记边数组。
- 3) 检查输入的顶点和关系是否符合要求，如果不符合则返回错误代码。
- 4) 遍历顶点数组，将节点信息存储到图的顶点数组中，并标记每个节点的位置。
- 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：(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

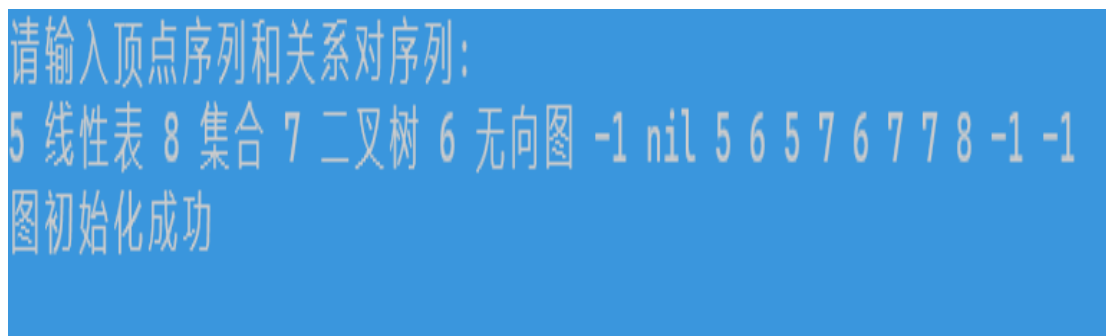


图 2-3 序号 1 中正常初始化

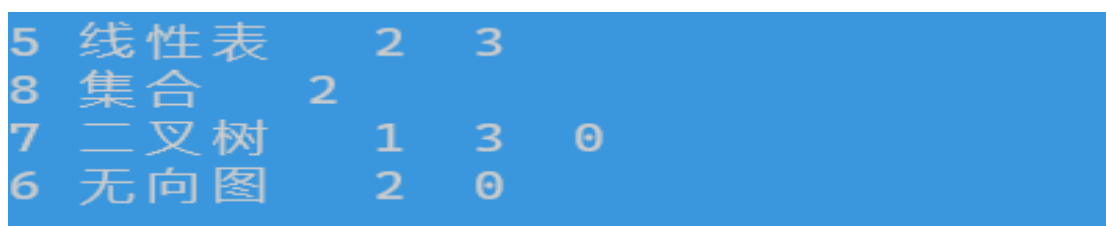


图 2-4 序号 1 中初始化后查看图中的关系图

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

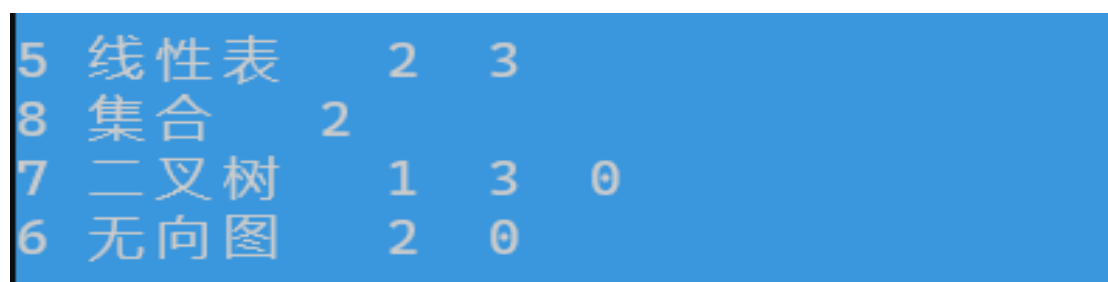


图 2-27 图中初始化的关系图

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着广泛应用。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（Vertices）和关系（Relationships）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。

掌握图的基本运算：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。

理解图的附加功能：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。

文件形式保存与加载：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。

多个图管理：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，我获得以下方面的收益：

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 A 基于顺序存储结构线性表实现的源程序

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

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

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

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

```
119         break;
120     case 4:
121         // 显示功能菜单
122         show();
123         // 判断线性表是否为空
124         ListEmpty(L);
125         getchar();
126         break;
127     case 5:
128         // 显示功能菜单
129         show();
130         // 获取线性表长度
131         int getdata ;
132         getdata = ListLength(L);
133         if( getdata !=INFEASIBLE )
134         {
135             printf ("线性表的长度是%d",getdata);
136         }
137         getchar();
138         break;
139     case 6:
140         // 显示功能菜单
141         show();
142         // 获取指定位置的元素
143         printf ("请输入你想获取第几个元素\n");
144         int i ;
145         scanf ("%d",&i);
146         int n ; int getdata1 ;
147         n = GetElem(L,i,getdata1);
148
149         if(n == OK)
```

```
150         {
151             printf("成功获取到第%d个元素的值: %d\n",i,
                    getdata1);
152         }
153         getchar();
154         break;
155     case 7:
156         // 显示功能菜单
157         show();
158         // 查找指定元素
159         int u ;
160         printf("请输入一个数值e\n");
161         int e ; scanf("%d",&e);
162         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");
163         int getorder ; scanf("%d",&getorder);
164         int q;
165         if( getorder == 1)
166         {
167             printf("你想要这个数据比e大多少\n");
168
169             scanf("%d",&q);
170             if(LocateElem(L,e+q,compare) > 0)
171             {
172                 printf("你要查找到数据的下标的是%d\n",
                        LocateElem(L,e+q,compare));
173             }
174         }
175         else {
176             printf("你想要这个数据比e小多少\n");
177
```

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



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

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

```
270     system("pause");
271     return 0;
272 }
273
274
275
276
277
278
279
280
281 // 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；
282 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。
283 {
284     // 请在这里补充代码，完成本关任务
285     /***** Begin *****/
286     if(L.elem)
287         return INFEASIBLE;//线性表已存在，返回线性表不存在的错误代码
288
289     L.elem = (ElemType *)malloc(sizeof(ElemType)*LIST_INIT_SIZE);//为线性表分配内存空间
290     L.length = 0; // 将线性表的长度设置为0
291     L.listsize = LIST_INIT_SIZE;//设置线性表的容量
292
293     return OK;//返回操作成功的代码
294
295     /***** End *****/
296 }
```

```
297
298 // 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；
299 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。
300 {
301 // 请在这里补充代码，完成本关任务
302 /***** Begin *****/
303     if(!L.elem)// 如果线性表不存在，返回线性表不存在的错误代码
304     {
305         printf("线性表不存在\n");
306         return INFEASIBLE;
307     }
308
309     free(L.elem); // 释放线性表中元素的内存空间
310     L.elem = NULL; // 将线性表指针置为空指针
311     L.length = 0; // 将线性表长度置为0
312     L.listsize = 0; // 将线性表容量置为0
313     printf("成功销毁线性表\n");
314     return OK; // 返回操作成功的代码
315
316 /***** End *****/
317 }
318 // 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；
319 status ClearList(SqList& L)
320 // 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。
321 {
322 // 判断线性表是否存在或是否为空
323     if(!L.length || !L.elem )
```

```
324     {
325         printf("笨蛋，线性表不存在或者没有元素\n");
326         return INFEASIBLE;
327
328     }
329
330     L.length = 0;    // 将线性表长度设为0，相当于清空了线性表
331     printf("成功删除线性表里面的元素啦\n");
332     return OK;
333 }
334
335 // 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；
336 status ListEmpty(SqList L)
337 // 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。
338 {
339     // 判断线性表是否存在
340     if(!L.elem)
341     {
342         printf("猪头，线性表不存在\n");
343         return INFEASIBLE;
344     }
345
346     // 判断线性表是否为空
347     if(L.length==0)
348     {
349         printf("线性表是空的\n");
350         return TRUE;
351     }
352     printf("线性表不是空的\n");
```

```
353     return FALSE;
354 }
355
356 // 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；
357 status ListLength(SqList L)
358 // 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。
359 {
360 // 判断线性表是否存在
361     if (L.elem==NULL)
362     {
363         printf ("线性表不存在\n");
364         return INFEASIBLE;
365     }
366
367 // 返回线性表长度
368     else {
369         return L.length;
370     }
371 }
372
373 // 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；
374 status GetElem(SqList L, int i, ElemType &e)
375 // 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。
376 {
377 // 判断线性表是否存在
378     if (!L.elem)
379     {
```

```
380     printf ("线性表不存在\n");
381     return INFEASIBLE;
382 }
383
384 // 判断线性表序号i的合法性
385 if(i<1 || i>L.length)
386 {
387     printf ("获取的元素i不合法\n");
388     return ERROR;
389 }
390
391 // 获取线性表第i个元素
392 e = L.elem[i-1];
393 {
394     return OK;
395 }
396 }
397
398 // 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；
399 int LocateElem(SqList L,ElemType e,int (*p)(int ,int ))
400 // 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
    。
401 {
402 // 判断线性表是否存在
403     if (!L.elem)
404     {
405         printf ("笨蛋，线性表不存在\n");
406         return INFEASIBLE;
```

```
407     }
408
409 // 在线性表中查找元素，若找到返回位置序号
410     for( int k=0;k<L.length;k++)
411     {
412         if(compare(L.elem[k],e))
413         {
414             return k+1;
415         }
416     }
417
418 // 没有找到符合的元素
419     printf ("抱歉，没找到符合的元素");
420     return ERROR;
421 }
422 // 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
423 status PriorElem(SqList L, ElemType cur_e, ElemType &pre_e) {
424     // 如果线性表L为空，返回INFEASIBLE
425     if (!L.elem) {
426         printf ("Error: 该线性表不存在.\n");
427         return INFEASIBLE;
428     }
429
430     // 如果要获取前驱的元素是第一个，返回ERROR
431     if (L.elem[0] == cur_e) {
432         printf ("Error: 要获取前驱的元素是第一个，不存在前驱.\n");
433         return ERROR;
434     }
435
```



```
436 // 查询要获取前驱的元素在表中的位置
437 for (int k = 1; k < L.length; k++) {
438     if (L.elem[k] == cur_e ) {
439         pre_e = L.elem[k-1];
440         return OK;
441     }
442 }
443
444 // 如果要获取前驱的元素不存在，返回ERROR
445 printf ("Error: 该元素不存在于该线性表中。\\n");
446 return ERROR;
447 }
448
449 // 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；
450 status NextElem(SqList L, ElemType cur_e, ElemType &next_e) {
451     // 如果线性表L为空，返回INFEASIBLE
452     if (!L.elem) {
453         printf ("Error: 该线性表不存在。\\n");
454         return INFEASIBLE;
455     }
456
457     // 如果要获取后继的元素是最后一个，返回ERROR
458     if (L.elem[L.length-1] == cur_e) {
459         printf ("Error: 要获取后继的元素是最后一个，不存在后继。\\n"
460             );
461         return ERROR;
462     }
463
464     // 查询要获取后继的元素在表中的位置
```

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

```
493         // 顺序表已满，需要重新分配空间
494         ElemType * newbase = (ElemType *)realloc(L.elem, sizeof(
            ElemType) * (L. listsize + LISTINCREMENT));
495         if (!newbase)
496             return ERROR;
497         L.elem = newbase;
498         L. listsize += LISTINCREMENT;
499     }
500
501     // 将位置k及其后面的元素后移一位
502     for (int p = L.length - 1; p >= k; p--) {
503         L.elem[p + 1] = L.elem[p];
504     }
505
506     // 插入元素e
507     L.length++;
508     L.elem[k] = e;
509     return OK;
510 }
511
512 // 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。
513 status ListDelete (SqList &L, int i, ElemType &e)
514 {
515     if (!L.elem) {
516         // 线性表为空
517         printf ("线性表为空\n");
518         return INFEASIBLE;
519     }
520
```

```
521     if (i < 1 || i > L.length) {
522         // 删除位置不正确
523         printf("删除位置不正确\n");
524         return ERROR;
525     }
526
527     e = L.elem[i - 1];
528     // 将位置i后面的元素前移一位
529     for (int k = i - 1; k < L.length - 1; k++) {
530         L.elem[k] = L.elem[k + 1];
531     }
532     L.length--;
533     return OK;
534 }
535
536 // 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。
537 status ListTraverse (SqList L, void (* visit )(int))
538 {
539     if (!L.elem) {
540         // 线性表为空
541         printf("线性表为空\n");
542         return INFEASIBLE;
543     }
544
545     for (int k = 0; k < L.length; k++) {
546         // 调用 visit 对每个元素进行操作
547         visit (L.elem[k]);
548         if (k != L.length - 1) {
549             putchar(' ');
550         }
```

```
551     }
552     return OK;
553 }
554
555 int compare(int a ,int b)
556 {
557     if(a == b)
558     {
559         return 1;
560     }
561     return 0;
562 }
563 // 遍历输出时候调用的visit函数
564 void visit ( int elem)
565 {
566     printf ("%d",elem);
567 }
568 // 求最大连续子数组
569 int MaxSubArray(SqList L)
570 {
571     if(!L.elem) // 如果线性表不存在
572     {
573         printf ("笨蛋，线性表不存在");
574         return INFEASIBLE;
575     }
576     int max ,current ; // 定义max表示最大值，current表示当前值
577     max = current = L.elem[0]; // 初始化max和current为线性表L的第一个元素
578     for( int k = 1;k<L.length;k++) // 遍历线性表L
579     {
580         if( current <= 0) // 如果当前值小于等于0
```

```
581     {
582         current = L.elem[k]; // 将当前值置为下一个元素的值
583     } else {
584         current += L.elem[k]; // 将当前值做累加操作
585     }
586     if(max < current) // 如果最大值小于当前值
587     {
588         max = current; // 更新最大值为当前值
589     }
590 }
591 return max; // 返回最终最大值
592 }
593 // 计算线性表L中和为u的子数组个数
594 int SubArrayNum(SqList L, int u)
595 {
596     if(!L.elem)
597     {
598         printf("笨蛋，线性表不存在");
599         return INFEASIBLE; // 返回线性表不存在的错误代码
600     }
601     int count = 0; // 子数组个数计数器
602     for(int k = 0; k < L.length; k++)
603     {
604         int sum = 0; // 子数组元素之和
605         for(int i = k; i < L.length; i++)
606         {
607             sum += L.elem[i]; // 累加元素值
608             if(sum == u)
609             {
610                 count ++; // 子数组和为u，计数器加一
611             }
612         }
613     }
614 }
```

```
612     }
613 }
614 return count ;
615 } // 冒泡排序实现顺序表排序
616 int sort (SqList &L)
617 {
618     for( int k = 0; k < L.length - 1; k++)//控制轮数
619     {
620         for( int i = 0; i < L.length - 1 - k; i++)//每轮比较相邻元素
621         {
622             if(L.elem[i] > L.elem[i+1])// 如果左边元素大于右边元素
623             {
624                 int tmp = L.elem[i]; // 交换两个元素的位置
625                 L.elem[i] = L.elem[i+1];
626                 L.elem[i+1] = tmp;
627             }
628         }
629     }
630 }
631
632 /* 输出程序的菜单 */
633 void show()
634 {
635     for( int k = 0; k <= 119 ;k++) // 打印分割线
636     {
637         putchar('—');
638     }
639     printf ("                Menu for Linear Table On Sequence Structure \n"
640             );
641     printf ("                1. InitList                7.
        LocateElem\n");
```

```

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

```



```

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

```

```

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

```

```
721
722     putchar('\n');
723
724 }
725
726 void menu()
727 {
728     printf("1.创建一个线性表\n");
729     printf("2.删除一个线性表\n");
730     printf("3.查找一个线性表和进行操作\n");
731     printf("0.退出线性表的管理\n");
732
733 }
734
735 // 函数名: savetofile
736 // 功能: 将线性表 L 中的元素保存到文件中
737 // 输入参数: 线性表 L
738 // 返回值: 操作成功返回 OK, 否则返回 INFEASIBLE
739 int savetofile (SqList L)
740 {
741     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
742     char arr [30];
743     scanf("%s",arr); // 获取用户输入的文件名
744
745     FILE *fp;
746     fp = fopen(arr, "w"); // 以写入方式打开文件
747     if(fp == NULL){ // 如果文件打开失败
748         printf("error"); // 输出错误信息
749         return INFEASIBLE; // 返回错误代码
750     }
751
```

```
752     int i ;
753     for( i =0; i< L.length; i++) // 遍历线性表中的元素
754     {
755         fprintf (fp, "%d ",L.elem[i]); // 将元素写入文件中
756     }
757
758     fclose (fp); // 关闭文件
759     return OK; // 返回操作成功代码
760 }
761
762 // 函数名: getfromfile
763 // 功能: 从文件中读取线性表 L 的元素
764 // 输入参数: 线性表 L 的地址
765 // 返回值: 操作成功返回 OK, 否则返回 INFEASIBLE
766 int getfromfile (SqList &L)
767 {
768     if (!L.elem) // 如果线性表不存在
769     {
770         printf ("笨蛋, 线性表不存在\n"); // 输出错误信息
771         return INFEASIBLE; // 返回错误代码
772     }
773
774     printf ("请输入你要读取的文件名:\n"); // 提示输入要读取的文件名
775     char name[30] = {'\0'};
776     scanf ("%s",name); // 获取用户输入的文件名
777
778     FILE *fp = fopen(name,"r"); // 以只读方式打开文件
779     if (fp == NULL){ // 如果文件打开失败
780         printf ("error"); // 输出错误信息
781         return INFEASIBLE; // 返回错误代码
782     }
```

```
783
784     int j;
785     while( fscanf(fp,"%d",&j) !=EOF) // 读取文件中的元素
786     {
787         L.elem[L.length++] = j; // 将元素添加到线性表中
788     }
789
790     fclose(fp); // 关闭文件
791     return OK; // 返回操作成功代码
792 }
793
794 // 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
795 // 如果已经存在名为ListName的线性表，则返回INFEASIBLE，否则返
    回OK
796 status AddList(LISTS &Lists,char ListName[])
797 {
798     // 遍历已有线性表，查找是否已经存在名为ListName的线性表
799     for( int i = 0; i<Lists.length ;i++)
800     {
801         if(strcmp(ListName,Lists.elem[i].name) == 0)
802         {
803             printf("这个名字的线性表已经存在了");
804             return INFEASIBLE;
805         }
806     }
807
808     // 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
809     Lists.length++; // 在线性表管理系统中增加一个线性表
810     int n = 0; // 初始化线性表的长度n为0，即线性表为空
811
```

```
812 // 将新线性表的名字设置为ListName
813     strcpy ( Lists .elem[ Lists . length -1].name ,ListName);
814
815 // 将新线性表的数据初始化为空，即没有元素
816     Lists .elem[ Lists . length -1].L.elem=NULL;
817
818 // 注释掉的 InitList 不需要调用，因为L.elem已设置为NULL
819 //  InitList ( Lists .elem[ Lists . length -1].L);
820
821 // 返回添加线性表操作执行成功
822     return OK;
823 }
824
825 status RemoveList(LISTS &Lists,char ListName[])
826 // Lists 中删除一个名称为ListName的线性表
827 {
828     // 请在这里补充代码，完成本关任务
829     /***** Begin *****/
830     // 遍历线性表
831     for( int k = 0;k<Lists . length ;k++)
832     {
833         // 找到名称为ListName的线性表
834         if(strcmp( Lists .elem[k].name,ListName)==0)
835         {
836             // 销毁线性表
837             DestroyList( Lists .elem[k].L);
838             // 将后面的线性表前移，覆盖当前位置
839             for( int i = k;i<Lists . length -1;i++)
840             {
841                 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
```

```
842         Lists.elem[i].L.elem = Lists.elem[i+1].L.elem;
843         Lists.elem[i].L.length = Lists.elem[i+1].L.length;
844         Lists.elem[i].L.listsize = Lists.elem[i+1].L.listsize;
845     }
846     Lists.length--; // 线性表数量减一
847     return OK;
848 }
849 }
850 // 没有找到目标线性表，返回错误
851 return ERROR;
852
853 /***** End *****/
854 }
855
856 int LocateList(LISTS Lists, char ListName[])
857 // 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
858 {
859     // 请在这里补充代码，完成本关任务
860     /***** Begin *****/
861     // 循环遍历线性表数组
862     for(int k = 0; k < Lists.length; k++)
863     {
864         // 如果找到了指定名称的线性表
865         if(strcmp(Lists.elem[k].name, ListName) == 0)
866         {
867             // 返回该线性表的逻辑序号（索引位置+1）
868             return k+1;
869         }
870     }
871     // 如果未找到指定名称的线性表，则返回-1
```

```
872     return -1;
873
874     /***** End *****/
875 }
876
877
878 void funtion ()
879 {
880     showplus(); // 显示菜单页面头部
881     SqList S ; // 用于操作的线性表，S代表单个线性表
882
883     int order ;
884     scanf("%d",&order); // 获取用户输入的命令
885     while(order) { // 如果用户输入了命令，即非0，则进入循环
886
887         system("cls"); // 清空控制台屏幕
888         showplus(); // 重新显示菜单页面头部
889
890         switch (order) { // 根据用户命令进行相应操作
891             case 1:
892                 // 创建线性表
893
894                 S.elem = (ElemType *) malloc(sizeof(ElemType) *
                        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
                        小为LIST_INIT_SIZE
897
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");
```



```
900         int number;
901         scanf("%d", &number);
902         char c;
903         c = getchar();
904         while (c != '\n') { // 当输入未结束时，循环获取数
                                据并添加到线性表中
905             S.elem[S.length++] = number;
906             scanf("%d", &number);
907             c = getchar();
908         }
909         S.elem[S.length++] = number; // 将最后一个数据添加
                                到线性表中
910         break;
911     case 2:
912         // 寻找最大子序列
913         int r;
914
915         r = MaxSubArray(S); // 调用MaxSubArray函数计算最
                                大子序列之和
916         if (r != -1) {
917             printf("最大连续子序列之和是%d\n", r);
918         }
919         // printf("请输入下一个命令\n");
920         break;
921     case 3:
922         // 寻找和为K的连续子序列
923         printf("这个函数来寻找一个和为K的连续子序列\n");
924         int k;
925         printf("请输入一个K值\n");
926         scanf("%d", &k);
927         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
928     printf ("线性表中和为k的连续子列数量为%d\n",
              SubArrayNum(S, k));
929     }
930     break;
931 case 4:
932     // 对线性表进行排序
933     sort (S);    // 调用sort函数对线性表进行排序
934     printf ("排完序后的线性表如下:\n");
935     ListTraverse (S, visit );    // 使用 ListTraverse函数遍历
    并输出线性表中的数据
936     putchar ('\n');
937     break;
938 case 5:
939     // 将线性表保存到文件中
940     printf ("现在进行线性表的文件形式保存\n");
941     int message;
942     message = savetofile (S);    // 调用 savetofile 函数将线
    性表保存到文件中
943     if (message == OK)
944     {
945         printf ("存储成功\n");
946         // printf ("请输入下一个命令\n");
947     }
948     break;
949 case 6:
950     // 从文件中读取线性表数据
951     printf ("现在进行文件的读取\n");
952     SqList p;
953     p.elem = NULL;
954     InitList (p);    // 用这个线性表来存储读取的数据
```

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

```
983         }
984         if (u == INFEASIBLE) {
985             system("pause");
986         }
987         break;
988         // 删除线性表
989     case 2:
990         printf("现在进行删除线性表\n");
991         printf("请输入你想创建的线性表的名字\n");
992         );
993         char name2[30];
994         scanf("%s", name2);
995         RemoveList(Lists, name2); // 删除线性表
996         break;
997         // 查找和操作线性表
998     case 3:
999         printf("现在进行线性表的查找和操作\n");
1000         printf("请输入你想查找和操作的线性表的");
1001         名字\n");
1002         char name3[30];
1003         scanf("%s", name3);
1004         int judge;
1005         judge = LocateList( Lists , name3); // 查找
1006         线性表
1007         if (judge == -1) {
1008             printf("不存在这个线性表\n");
1009             system("pause");
1010         } else {
1011             printf("线性表存在鸭鸭\n");
1012             printf("现在对这个线性表进行操作\n");
1013             ;
```

```
1010         int order;
1011         show(); // 显示线性表操作菜单
1012         scanf("%d", &order);
1013         while (order) {
1014             switch (order) {
1015                 // 初始化线性表
1016                 case 1:
1017                     show1(); // 显示操作提示
1018                     if ( InitList ( Lists .elem[
1019                         judge - 1].L) == OK)
1020                         printf ("线性表创建成功! \n");
1021                     else printf ("线性表已经
1022                         存在, 创建失败! \n");
1023                     getchar ();
1024                     break;
1025                     // 删除线性表
1026                 case 2:
1027                     show1();
1028                     DestroyList ( Lists .elem[
1029                         judge - 1].L); // 销毁
1030                     线性表
1031                     getchar ();
1032                     break;
1033                     // 清空线性表
1034                 case 3:
1035                     show1();
1036                     ClearList ( Lists .elem[judge
1037                         - 1].L); // 清空线性
1038                     表
1039                     getchar ();
```

```
1034         break;
1035         // 判断线性表是否为空
1036     case 4:
1037         show1();
1038         ListEmpty(Lists.elem[judge
1039                     - 1].L);
1040         getchar();
1041         break;
1042         // 获取线性表的长度
1043     case 5:
1044         show1();
1045         int getdata;
1046         getdata = ListLength(Lists
1047                               .elem[judge - 1].L);
1048         if (getdata !=
1049             INFEASIBLE) {
1050             printf("线性表的长度
1051                    是%d", getdata);
1052         }
1053         getchar();
1054         break;
1055         // 获取线性表中的元素
1056     case 6:
1057         show1();
1058         printf("请输入你想获取第
1059                几个元素\n");
1060         int i;
1061         scanf("%d", &i);
1062         int n;
1063         int getdata1;
1064         n = GetElem(Lists.elem[
```

```
judge - 1].L, i,
getdata1);
1060 if (n == OK) {
1061     printf ("成功获取到第
           %d个元素的值: %
           d\n", i, getdata1);
1062 }
1063 getchar();
1064 break;
1065 // 查找线性表中的元素
1066 case 7:
1067     show1();
1068     int u;
1069     printf ("请输入一个数值e\n
           ");
1070     int e;
1071     scanf ("%d", &e);
1072     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");
1073     int getorder;
1074     scanf ("%d", &getorder);
1075     int q;
1076     if (getorder == 1) {
1077         printf ("你想要这个数
           据比e大多少\n");
1078         scanf ("%d", &q);
1079         if (LocateElem(Lists.
           elem[order - 1].L,
           e + q, compare) >
```

```
1080         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))
            ;
1081     }
1082 } else {
1083     printf("你想要这个数
                据比e小多少\n");
1084     scanf("%d", &q);
1085     if (LocateElem(Lists.
                elem[judge - 1].L,
                e + q, compare) >
                0) {
1086         printf("你要查找
                到数据的下标
                的是%d\n",
                LocateElem(
                Lists.elem[
                judge - 1].L, e
                + q, compare))
                ;
1087     }
1088 }
1089 getchar();
1090 break;
1091 // 获取线性表中某个元素
```



```

                                的前驱
1092         case 8:
1093             show1();
1094             printf("请问你想获得那个
                                元素的前驱\n");
1095             int v;
1096             scanf("%d", &v);
1097             int pre_e;
1098             if (PriorElem( Lists .elem[
                                judge - 1].L, v, pre_e
                                ) == OK) {
1099                 printf("成功获得前
                                驱，是%d\n", pre_e
                                );
1100             }
1101             getchar();
1102             break;
1103             // 获取线性表中某个元素
                                的后驱
1104         case 9:
1105             show1();
1106             printf("请问你想获得哪个
                                元素的后驱\n");
1107             int p;
1108             scanf("%d", &p);
1109             int next_e;
1110             if (NextElem(Lists.elem[
                                judge - 1].L, p,
                                next_e) == OK) {
1111                 printf("成功获取后
                                驱，是%d\n",
```

```

next_e);
1112     }
1113     getchar();
1114     break;
1115     // 在线性表中插入元素
1116 case 10:
1117     show1();
1118     printf("请问你想在第几个
           位置之前插入元素\n");
1119     int r;
1120     scanf("%d", &r);
1121     printf("插入的元素的值为
           \n");
1122     int a;
1123     scanf("%d", &a);
1124     if ( ListInsert ( Lists .elem[
           judge - 1].L, r, a) ==
           OK) {
1125         printf("插入成功\n");
1126     }
1127     getchar();
1128     break;
1129     // 在线性表中删除元素
1130 case 11:
1131     show1();
1132     printf("请问你想删除第几
           个数据元素\n");
1133     int b;
1134     scanf("%d", &b);
1135     int ee;
1136     if ( ListDelete ( Lists .elem[
```

```
1137         judge - 1].L, b, ee)
1138         == OK) {
1139             printf ("删除的数据元
1140                     素是%d\n", ee);
1141         }
1142         getchar ();
1143         break;
1144         // 遍历线性表
1145     case 12:
1146         show1();
1147         if ( ListTraverse ( Lists .
1148             elem[judge - 1].L,
1149             visit )) {
1150             printf ("\n成功遍历\n"
1151                     );
1152         }
1153         getchar ();
1154         break;
1155     case 0:
1156         break;
1157     }
1158     scanf ("%d", &order);
1159 }
1160 }
1161     break;
1162 case 0:
1163     break;
1164 }
1165 system ("cls");
1166 menu(); // 显示菜单
1167 printf ("请输入下一个命令\n");
```

```

1162         scanf("%d", &a);
1163     }
1164     break;
1165     default :
1166         printf("输入的命令不正确, 请再次输入\n");
1167         break;
1168     }
1169     printf("请输入下一个命令\n");
1170     scanf("%d", &order);
1171 }
1172 }
1173
1174 void showplus()
1175 {
1176     for(int k = 0; k<= 119 ;k++)
1177     {
1178         putchar('-');
1179     } putchar('\n');
1180     printf("这是一些附加的功能, 和之前的线性表不兼容\n");
1181     printf("          1.初始化线性表\n");
1182     printf("          2.maxSubarray          3.
          subarrayNum\n");
1183     printf("          4. sortList          5.
          savetofile \n");
1184     printf("          6. getfromfile          7.
          managelist\n");
1185     printf("          0. Exit\n");
1186     for(int k = 0; k<= 119 ;k++)
1187     {
1188         putchar('-');
1189     } putchar('\n');

```

1190 }

附录 B 基于链式存储结构线性表实现的源程序

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

```

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

```

```
60 void SearchList(LISTS Lists); //展示已经创建的线性表
61 status compare(int a, int b); //判断位置函数时候调用的比较函数
62 void visit (int x); //遍历函数时候调用的输出函数
63 void reverseList (LinkList L); //翻转线性表
64 void RemoveNthFromEnd(LinkList L,int n); //删除倒数元素
65 void sortList (LinkList L); //排序
66 void savetofile (LinkList L,char name[]); //保存到文件
67 void getfromfile (LinkList L,char name[]); //读取文件
68 void fun01(); //封装的多个线性表的处理函数
69 void fun02(LinkList &L ); //封装的处理单个线性表的处理函数
70 void menu(); //管理多个线性表的菜单
71 void show_normal(); //单个线性表的菜单
72 void Menuofinsert(); //插入的菜单
73
74 /*----- main主函数 -----*/
75 int main()
76 {
77     system("color 37");
78     fun01(); //调用封装函数
79
80 }
81
82
83 /*----- 函数定义 -----*/
84 // (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；
85 status InitList (LinkList &L)
86 // 线性表L不存在，构造一个空的线性表，返回OK，否则返回
    INFEASIBLE。
87 {
88 // 如果线性表L已存在，则返回 INFEASIBLE
```



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

```
118         free(p);
119 // 将指针 p 指向 q, 继续循环
120         p = q;
121     }
122 // 返回 OK
123     return OK;
124 }
125
126 // (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;
127 status ClearList (LinkedList &L)
128 {
129     // 如果线性表L不存在, 返回INFEASIBLE
130     if (!L)
131     {
132         printf ("线性表不存在或未初始化, 无法进行清空\n");
133         return INFEASIBLE;
134     }
135     // 如果线性表L为空, 不需要操作
136     if (L->next == NULL)
137     {
138         printf ("线性表已经是空的了, 不需要操作\n");
139         return INFEASIBLE;
140     }
141     LinkedList p = L->next; // 指向第一个元素节点
142     while(p)
143     {
144         free(p); // 释放当前节点
145         p = p->next; // 指向下一个节点
146     }
147     L->next = NULL; // 将头节点指向NULL, 清空线性表
```

```
148     return OK;
149 }
150
151 // (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;
152 status ListEmpty(LinkList L)
153 {
154     // 如果线性表L不存在, 返回INFEASIBLE
155     if (!L)
156     {
157         printf ("线性表不存在或未初始化, 无法进行清空\n");
158         return INFEASIBLE;
159     }
160     // 如果够了L的第一个元素为空, 表明线性表为空, 返回TRUE
161     if (L->next == NULL)
162     {
163         printf ("线性表是空的\n");
164         return TRUE;
165     }
166     // 线性表不为空, 返回FALSE
167     printf ("线性表不是空的\n");
168     return FALSE;
169 }
170
171 // (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;
172 int ListLength(LinkList L)
173 // 如果线性表L存在, 返回线性表L的长度, 否则返回INFEASIBLE。
174 {
175     /***** Begin *****/
176     // 首先要检查线性表是否存在
```

```

177     if (!L) {
178         printf("线性表不存在或未初始化, 无法进行清空\n");
179         return INFEASIBLE;
180     }
181
182     L = L->next;    // 不要把头节点考虑
183     int number = 0;    // 用来 记录长度
184     while (L) {      // 遍历链表, 计算数据元素的个数
185         number++;
186         L = L->next;
187     }
188     return number;
189
190     /***** End *****/
191 }
192
193 // (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;
194 status GetElem(LinkList L, int i, ElemType &e)
195 // 获取线性表L中第i个元素, 将其存储在e中
196 {
197     if (!L) // 若L为空, 表示线性表不存在或未初始化
198     {
199         printf("线性表不存在或未初始化, 无法进行清空\n");
200         return INFEASIBLE; // 返回INFEASIBLE
201     }
202     int number = 0; // 记录当前遍历到的节点数, 从0开始
203     LinkList p = L; // 定义p指针, 指向L
204     p = L->next; // 跳过头节点, 从第一个存储数据的节点开始遍历
205     while(p)
206     {

```

```

207         number++; // 遍历到一个节点, number加1
208         if(number == i) // 找到第i个节点
209         {
210             e = p->data; // 将找到的节点的数据存储在e中
211             return OK; // 返回OK
212         }
213         p = p->next; // 指针p指向下一个节点
214     }
215     printf("i的值不合法, 无法操作\n");
216     return ERROR; // 遍历完整个线性表, 未找到第i个节点, 返回
        ERROR
217 }
218
219 // (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;
220 // 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;
221 status LocateElem(LinkList L,ElemType e,int (*vis)(int ,int ))
222 // 查找元素e在线性表L中的位置序号
223 // 当e存在时, 返回其在线性表中的位置序号
224 // 当e不存在时, 返回ERROR
225 // 当线性表L不存在时, 返回INFEASIBLE
226 {
227     // 当线性表L不存在时, 返回INFEASIBLE
228     if(!L)
229     {
230         printf("线性表不存在或未初始化, 无法进行查找\n");
231         return INFEASIBLE;
232     }
233
234     // 从头结点的下一个结点开始向后遍历

```

```
235     LinkedList p = L->next;
236
237     // 记录当前位置序号
238     int number = 0;
239
240     // 遍历链表，查找元素e
241     while(p)
242     {
243         number++;
244
245         if (vis(p->data,e) == 1) // 如果找到元素e，返回其位置序号
246         {
247             return number;
248         }
249         p = p->next;
250     }
251
252     // 如果遍历完整个线性表仍未找到元素e，返回ERROR
253     printf("没有所要查询的元素\n");
254     return ERROR;
255 }
256
257 // (8) 获得前驱：函数名称是PriorElem(L,cur_e,pre_e);
258 // 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；
259 status PriorElem(LinkedList L,ElemType e,ElemType &pre)
260 // 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。
261 {
```

```
262     if(!L)    // 如果链表L不存在，则返回INFEASIBLE，表示不可行
263     {
264         printf("线性表不存在或未初始化，无法进行清空\n");
265         return INFEASIBLE;
266     }
267     if(L->next == NULL) // 如果链表L为空，则返回ERROR，表示出错
268     {
269         printf("线性表里面没有元素\n");
270         return ERROR;
271     }
272     LinkList p = L->next;
273     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
274     {
275         printf("所要查找的元素是第一个元素，没有前驱\n");
276         return ERROR;
277     }
278     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
279     {
280         L = p->next;
281         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
282         {
283             pre = p->data;
284             return OK;
285         }
286         p = L;
287     }
288     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
```

```

289     return ERROR;
290 }
291
292 // (9) 获得后继: 函数名称是NextElem(L,cur_e,next_e);
293 // 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;
294 status NextElem(LinkList L,ElemType e,ElemType &next)
295 // 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。
296 {
297     if (!L) // 如果线性表L不存在
298     {
299         printf("线性表不存在或未初始化, 无法进行查询\n");
300         return INFEASIBLE;
301     }
302     if (L->next==NULL) // 如果线性表L是空表
303     {
304         printf("这个线性表是空的\n");
305         return ERROR;
306     }
307     LinkList p = L->next; // 设p为第一个结点, p做前驱结点
308     L = p->next; // 设L为p的后继结点, L做当前结点
309     while(p) // 如果p不为空
310     {
311         if( !L) // 如果L为空, 表示已经没有后继结点了
312         {
313             printf("查询不到后继结点\n");
314             return ERROR;
315         }

```



```

316         if(p->data == e)    // 如果p的数据等于给定数据e
317         {
318             next = L->data;    // 将L的数据赋给next
319             return OK;    // 返回操作成功
320         }
321         p = L ;    // 将p移到L的位置，作为新的前驱结点
322         L = p->next;    // 将L移到下一个结点位置，作为新的当前结点
323     }
324     return ERROR; // 如果循环结束时仍未查询到，返回操作失败
325 }
326
327 // (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；
328 status ListInsert (LinkList &L,int i,int num)
329 {
330     // 先进行特判，如果线性表不存在，返回 INFEASIBLE
331     if (!L)
332     {
333         printf ("线性表不存在或未初始化，无法进行插入\n");
334         return INFEASIBLE;
335     }
336     int e;
337
338     // 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
339     LinkList p = L, next = L->next;
340     int number = 1;    // 来记录当前位置
341
342     // 遍历链表，找到要插入的位置
343     printf ("请输入元素：\n");
344     while(next)

```

```
345     {
346         if(number == i)
347         {
348             // 当找到插入位置时，使用一个循环插入 num 个元素
349             while (num)
350             {
351                 // 创建新的节点，获取用户输入的数据
352                 LinkList insert = (LinkList)malloc(sizeof(LNode));
353                 scanf("%d",&e);
354                 insert->data = e;
355
356                 // 修改链表指针指向，完成插入操作
357                 p->next = insert ;
358                 insert->next = next;
359                 p = insert ;
360                 num--;
361             }
362
363             return OK;
364         }
365
366         // 继续向下遍历
367         number++;
368         p = next;
369         next = p->next;
370     }
371
372     // 如果插入位置为最后一个位置，则在链表尾部插入
373     if( number == i)
374     {
375         LinkList insert ;
```

```
376
377 // 使用循环将 num 个数据插入到尾部
378 while (num) {
379     scanf("%d",&e);
380     insert = (LinkList)malloc( sizeof(LNode));
381     insert->data = e;
382     insert->next = NULL;
383     p->next = insert ;
384     p = insert ;
385     num--;
386 }
387 return OK;
388 }
389
390 // 如果插入位置不正确，返回错误
391 printf("插入的位置不对\n");
392 return ERROR;
393 }
394
395 // (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；
396 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
397 {
398     if(!L) // 如果线性表L不存在，返回INFEASIBLE
399     {
400         printf("线性表不存在或未初始化，无法进行清空\n");
401         return INFEASIBLE;
402     }
403     int number = 0; // 用于记数，记录当前扫描到的元素的位置
```

```
404
405     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
406
407     while(next) // 遍历线性表，直到到达表尾
408     {
409         number++; // 计数器加1，记录当前扫描到的元素的位置
410
411         if(number == i) // 如果找到第i个元素
412         {
413             e = next->data; // 将该元素的值保存在e中
414             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
415             free(next); // 释放内存
416             return OK; // 返回执行成功
417         }
418         pre = next; // 当前元素保存到前一个元素变量pre中
419         next = pre->next; // 后一个元素保存到当前元素变量next中，
            实现遍历
420     }
421     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
422     return ERROR; // 返回执行失败
423 }
424
425 // (12) 遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表
        L已存在；
426 // 操作结果是依次对L的每个数据元素调用函数visit()。
427 status ListTraverse (LinkList L,void (*vi)(int ))
428 // 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。
429 // 如果线性表 L 不存在，返回 INFEASIBLE，否则返回 OK。
```

```
430 {
431     if(!L) // 如果线性表 L 不存在
432     {
433         printf("线性表不存在或未初始化，无法进行操作\n");
434         return INFEASIBLE; // 返回 INFEASIBLE
435     }
436     LinkList p = L->next; // 从 L 中第一个元素开始遍历
437     while(p) // 只要当前节点不是尾节点
438     {
439         vi(p->data); // 对当前节点的元素使用函数指针 vi 进行处理
440         p = p->next; // 指向下一个节点
441         if(p) // 如果当前不是最后一个节点
442         {
443             putchar(' '); // 输出一个空格，与下一个元素分隔开来
444         }
445     }
446     return OK; // 遍历结束，返回 OK
447 }
448
449 // 在 Lists 中增加一个名称为 ListName 的空线性表
450 // Lists：线性表集合，包含多个线性表
451 // ListName: 待添加的线性表名称
452 // 返回值：操作状态，成功为OK，否则为INFEASIBLE
453 status AddList(LISTS &Lists, char ListName[])
454 {
455     // 循环查找是否已经存在同名线性表
456     for(int i = 0; i < Lists.length; i++)
457     {
458         if(strcmp(ListName, Lists.elem[i].name) == 0) // 判断线性表名
459             称是否相同
460     }
```

```
460         printf("这个名字的线性表已经存在了"); // 输出提示信息
461         return INFEASIBLE; // 返回INFEASIBLE表示操作失败
462     }
463 }
464 // 未找到同名线性表，可以继续添加
465 Lists.length++; // 线性表集合长度+1
466 int n = 0;
467 // 将新线性表的名称和数据初始化
468 strcpy ( Lists.elem[ Lists.length-1].name, ListName); // 将线性表名称赋值
469 Lists.elem[ Lists.length-1].L = NULL; // 将存储数据的指针初始化为NULL
470 return OK; // 返回OK表示操作成功
471 }
472
473 status RemoveList(LISTS &Lists, char ListName[])
474 // Lists 中删除一个名称为ListName的线性表
475 {
476     // 遍历线性表数组找到需要删除的线性表
477     for( int k = 0; k < Lists.length; k++)
478     {
479         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
480         {
481             DestroyList( Lists.elem[k].L); // 先销毁这个线性表本身的空间
482             // 指针和名称逐一向前移动
483             for( int i = k; i < Lists.length-1; i++)
484             {
485                 strcpy ( Lists.elem[i].name, Lists.elem[i+1].name); // 逐一将后面的线性表的名称复制到前面
```

```
486             Lists.elem[i].L=Lists.elem[i+1].L;    // 将后面线性
               表的指针复制到前面
487         }
488         // 线性表数组的长度减 1
489         Lists.length--;
490         return OK; // 删除成功
491     }
492 }
493 return ERROR; // 没有找到要删除的线性表，删除失败
494 }
495
496 int LocateList(LISTS Lists,char ListName[])
497 // 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
498 {
499     // 开始遍历线性表
500     for(int k = 0 ;k< Lists.length;k++)
501     {
502         // 比较线性表名称是否匹配
503         if(strcmp( Lists.elem[k].name,ListName)==0)
504         {
505             return k+1; // 返回序号（序号从 1 开始）
506         }
507     }
508     return -1; // 查找失败，返回 -1
509 }
510
511 void SearchList(LISTS Lists) // 这个函数负责展示已经创建的线性表
512 {
513     int i =0;
514     printf("已经存在的线性表有： \n");
```

```
515     for( ;i<Lists.length;i++)
516     {
517         printf("序号 %d) 线性表的名称:%s\n",(i+1),Lists.elem[i].name
518             );
519     }
520 }
521
522 // 翻转线性表
523 void reverseList (LinkedList L)
524 {
525     // 判断线性表是否存在或未初始化
526     if (!L)
527     {
528         printf("线性表不存在或未初始化\n");
529         return ;
530     }
531     // 获取线性表的长度
532     int len = ListLength(L);
533     // 申请一个长度为 len 的数组
534     int *arr =(int *) malloc(sizeof(int)*len);
535     // 遍历链表，存储链表中的元素到数组中
536     LinkedList p1 = L->next;
537     for( int k=0;k<len;k++)
538     {
539         arr[k] = p1->data;
540         p1 = p1->next;
541     }
542     // 从尾到头遍历链表，将数组中的元素依次存储到链表中
543     LinkedList p2 = L->next;
544     for( int i =len;i>0;i--)
```



```
545     {
546         p2->data = arr[i-1];
547         p2 = p2->next;
548     }
549     printf("成功翻转了\n");
550     return ;
551 }
552
553 void RemoveNthFromEnd(LinkList 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
554 {
555     if(!L) // 如果单链表为空或未初始化
556     {
557         printf("线性表不存在或着未初始化\n"); // 打印错误信息
558         return ; // 退出函数
559     }
560
561     int len = ListLength(L); // 获取单链表的长度
562     int e; // 用来存储被删除节点的数据元素
563     int feedback; // 存储ListDelete函数的返回值，即删除操作的结果
564
565     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
566
567     if(feedback == OK) // 如果删除成功
568     {
569         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
570     }
571 }
572 // 排序
```

```
573 void sortList (LinkedList L)
574 {
575     //判断线性表是否存在或者未初始化
576     if (!L)
577     {
578         printf ("线性表不存在或者未初始化\n");
579         return ;
580     }
581     //获取线性表长度
582     int len = ListLength(L);
583     //申请动态内存，用于存放线性表数据
584     int *arr =(int *) malloc( sizeof( int)*len);
585     //遍历线性表，将数据存放到arr数组中
586     LinkedList p1 = L->next;
587     for( int k=0;k<len;k++)
588     {
589         arr[k] = p1->data;
590         p1 = p1->next;
591     }
592     //对arr数组进行冒泡排序
593     for( int k=0;k<len-1;k++)
594     {
595         for( int i=0;i<len-1-k;i++)
596         {
597             if( arr[i] > arr[i+1])
598             {
599                 int tmp = arr[i]; //中间变量tmp
600                 arr[i] = arr[i+1];
601                 arr[i+1] = tmp;
602             }
603         }
```

```
604     }
605     // 将排序后的数据写回线性表中
606     LinkList p2 = L->next;
607     for( int k =0;k<len;k++)
608     {
609         p2->data = arr[k];
610         p2 = p2->next;
611     }
612 }
613
614 // 从线性表L中将数据保存到文件name中
615 void savetofile (LinkList L,char name[])
616 {
617     if(!L) // 如果L不存在或未初始化，无法进行操作
618     {
619         printf("线性表不存在或未初始化\n");
620         return ;
621     }
622     FILE *fp = fopen(name,"w"); // 打开文件，以写的方式
623     if(fp == NULL) // 如果无法找到文件，报错
624     {
625         printf("打开文件失败\n");
626         return ;
627     }
628     LinkList current = L->next; // 指向第一个节点
629     while ( current != NULL) // 循环遍历线性表中的每个节点
630     {
631         fprintf (fp, "%d ", current->data); // 将节点的数据写入文件
        中
632         current = current->next;
633     }
```

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

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

```
689         {
690             printf("创建成功啦\n"); // 输出提示信息
691         }
692         if(u == INFEASIBLE) //如果插入失败
693         {
694             system("pause"); // 暂停程序运行
695         }
696         break;
697
698         // 如果a等于2, 执行以下代码
699     case 2:
700         printf("现在进行删除线性表\n"); //输出提示信息
701         printf("请输入你想创建的线性表的名字\n"); //输出提示信息
702         char name2[30]; // 定义一个名字为name2且长度最大为
703             30的字符数组
704         scanf("%s", name2); // 读取字符串
705         RemoveList(Lists, name2); // 从线性表数组中删除指定
706             的线性表
707         break;
708
709         // 如果a等于3, 执行以下代码
710     case 3:
711         printf("现在进行查询创建了哪些线性表\n"); //输出提示信息
712         SearchList(Lists); // 查询线性表数组中所有线性表的
713             名字并输出
714         break;
715
716         // 如果a等于4, 执行以下代码
717     case 4:
```

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

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



```
771         {
772             printf ("成功销毁了\n");
773         }
774         else if (feedback == INFEASIBLE)
775         {
776             printf ("这个线性表不存在或未初始化,无法销毁\n");
777             ;
778         }
779         break;
780     case 3:
781         // 清空线性表
782         printf ("现在进行线性表的清空操作\n");
783         feedback = ClearList (L);
784         if (feedback == OK)
785         {
786             printf ("成功清空线性表\n");
787         }
788         break;
789     case 4:
790         // 判断线性表是否为空
791         printf ("现在对线性表进行判空操作\n");
792         ListEmpty(L);
793         break;
794     case 5:
795         // 求线性表的长度
796         printf ("现在进行求线性表的长度\n");
797         feedback = ListLength(L);
798         if (feedback != INFEASIBLE)
799         {
800             printf ("线性表的长度为:%d",feedback);
801         }
```

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

```
831         scanf("%d",&cur_e);
832         feedback = PriorElem(L,cur_e,pre_e);
833         if(feedback == OK)
834         {
835             printf("你所要查找的前驱是: %d\n",pre_e);
836         }
837         break;
838     case 9:
839         // 查找指定元素的后驱
840         printf("现在进行查找后驱的操作\n");
841         int cur,next_e;    //接收元素，并存储后继
842         printf("请输入你想查找哪个元素的后驱\n");
843         scanf("%d",&cur);
844         feedback = NextElem(L,cur,next_e);
845         if(feedback == OK)
846         {
847             printf("你所要查找的元素的后驱是: %d\n",next_e);
848         }
849         break;
850     case 10:
851         // 在指定位置插入元素
852         printf("现在进行插入元素的操作\n");
853         printf("请问你想在第几个位置插入元素\n");
854         int position;    //接收元素的位置
855         scanf("%d",&position);
856         printf("请问你想插入元素的个数\n");
857         int number;
858         scanf("%d",&number);
859         feedback = ListInsert (L, position ,number);
860         if(feedback == OK)
861         {
```

```
862         printf("插入成功\n");
863     }
864     break;
865 case 11:
866     // 删除指定位置的元素
867     printf("现在进行删除元素的操作\n");
868     printf("请问你想删除第几个位置的元素\n");
869     int position01 ;
870     int e1;
871     scanf("%d",&position01);
872     feedback = ListDelete (L,position01 ,e1);
873     if(feedback == OK)
874     {
875         printf("删除成功，删除的元素是: %d\n",e1);
876     }
877     break;
878 case 12:
879     // 遍历线性表
880     printf("现在进行线性表的遍历\n");
881     ListTraverse (L, visit );
882     break;
883 case 13:
884     // 翻转线性表
885     printf("现在进行线性表的翻转\n");
886     reverseList (L);
887     break;
888 case 14:
889     // 删除倒数第n个元素
890     printf("现在进行删除链表倒数元素的操作\n");
891     printf("你想删除链表倒数第几个节点\n");
892     int positon2;
```

```
893         scanf("%d",&positon2);
894         RemoveNthFromEnd(L,positon2);
895         break;
896     case 15:
897         // 排序线性表
898         printf("现在进行链表的排序\n");
899         sortList (L);
900         printf("搞定力\n");
901         break;
902     case 16:
903         // 将线性表保存到文件
904         printf("现在进行线性表的文件保存\n");
905         printf("请问你想保存到哪一个文件\n");
906         char name1[30];
907         scanf("%s",name1);
908         savetofile (L,name1);
909         break;
910     case 17:
911         // 从文件中读取线性表
912         printf("现在进行线性表的读取操作\n");
913         printf("你想读取哪一个文件的资料\n");
914         char name2[30];
915         scanf("%s",name2);
916         getfromfile (L,name2);
917         break;
918     default :
919         // 输入错误命令序号
920         printf("命令输入有问题\n");
921 }
922 putchar('\n');
923 printf("请输入下一个命令\n");
```

```
924     scanf("%d",&order);
925     system("cls"); // 清空命令行窗口
926     if (order != 0)
927     {
928         show_normal(); // 显示命令列表
929     }
930     else {
931         menu(); // 返回主菜单
932     }
933
934 }
935 }
936
937 void show_normal() // 单个线性表的菜单
938 {
939     // 输出横线
940     for(int k = 0; k<= 119 ;k++)
941     {
942         putchar('-');
943     }
944     putchar('\n');
945
946     // 输出菜单标题
947     printf("          Menu for Linear Table On Sequence Structure \n"
948
949 );
948
949     // 输出菜单选项
950     printf("          1. InitList          7.
951             LocateElem\n");
951     printf("          2. DestroyList       8.
952             PriorElem\n");
```

| | | | |
|-----|-------------------------------|-----------------|-------|
| 952 | printf (" | 3. ClearList | 9. |
| | NextElem \n"); | | |
| 953 | printf (" | 4. ListEmpty | 10. |
| | ListInsert \n"); | | |
| 954 | printf (" | 5. ListLength | 11. |
| | ListDelete \n"); | | |
| 955 | printf (" | 6. GetElem | 12. |
| | ListTraverse \n"); | | |
| 956 | | | |
| 957 | // 输出横线 | | |
| 958 | for(int k = 0; k<= 119 ;k++) | | |
| 959 | { | | |
| 960 | putchar('-'); | | |
| 961 | } | | |
| 962 | | | |
| 963 | putchar('\n'); | | |
| 964 | | | |
| 965 | // 输出额外的菜单选项 | | |
| 966 | printf (" | 13. reverseList | 14. |
| | RemoveNthFromEnd\n"); | | |
| 967 | printf (" | 15. sortList | 16. |
| | saveFile \n"); | | |
| 968 | printf (" | 17. getFile | \n"); |
| 969 | printf (" | 0. exit | \n"); |
| 970 | | | |
| 971 | // 输出横线 | | |
| 972 | for(int k = 0; k<= 119 ;k++) | | |
| 973 | { | | |
| 974 | putchar('-'); | | |
| 975 | } | | |
| 976 | | | |

```

977     putchar('\n');
978
979     // 输出小猫咪
980     printf("      ^      /\n");
981     printf("    /\ 7      □_ ^n");
982     printf("  / |      / /\n");
983     printf(" | Z_,<  /  /'F\n");
984     printf(" |      F  /  > \n");
985     printf(" Y      '  /  ^n");
986     printf(" ?● ? ●  ?? <  ^n");
987     printf(" () ^      | \ <n");
988     printf(" >??_ 彳  | / /\n");
989     printf("  / ^      / ?<| \ \n");
990     printf("  F_?  ( /  | / /\n");
991     printf("    7      | /\n");
992     printf("    >—r - - '?— _\n");
993
994     putchar('\n');
995
996     // 输出横线
997     for(int k = 0; k<= 119 ;k++)
998     {
999         putchar('—');
1000     }
1001
1002     putchar('\n');
1003
1004     // 输出提示文字
1005     printf("    请选择你的操作[0~13:]");
1006
1007     putchar('\n');

```



```
1008
1009 // 输出横线
1010 for(int k = 0; k<= 119 ;k++)
1011 {
1012     putchar('-');
1013 }
1014
1015 putchar('\n');
1016 }
1017
1018 void menu()
1019 {
1020     for(int k = 0; k<= 119 ;k++) // 打印分隔线，共120个 '-'
1021     {
1022         putchar('-');
1023     }
1024     putchar('\n'); // 打印换行符
1025
1026     printf("1.创建一个线性表\n"); // 打印操作1的描述
1027     printf("2.删除一个线性表\n"); // 打印操作2的描述
1028     printf("3.查询已经创建的线性表\n"); // 打印操作3的描述
1029     printf("4.查找一个线性表和进行操作\n"); // 打印操作4的描述
1030     printf("0.退出线性表的管理\n"); // 打印操作0的描述
1031
1032     // 打印一只猫的 ASCII Art，增加菜单的趣味性
1033     printf("      ^                /\n");
1034     printf("      /\ 7          □ _ ^\n");
1035     printf("      /  |          /  /\n");
1036     printf("      |  Z _,<    /    /'F'\n");
1037     printf("      |          F    /    > \n");
1038     printf("      Y          '  /    /\n");
```

```

1039     printf ("  ?●  ?  ●   ?? <   ^n");
1040     printf ("  ()  ^         |  \ <n");
1041     printf ("  >? ?_  ı   |  /  /n");
1042     printf ("  / ^         /  ?<|  \  \n");
1043     printf ("  ☒_?   ( /   |  /  /n");
1044     printf ("  7               |  /n");
1045     printf ("  >—r - - ‘?— _n");
1046
1047     for( int k = 0; k<= 119 ;k++) // 其他同上，打印分隔线
1048     {
1049         putchar('—');
1050     }
1051     putchar('\n'); // 最后再打印一行空行，方便视觉上的分离
1052 }
1053
1054 // 下方是一个插入操作的菜单，类似于前面的 menu()函数
1055 void Menuofinsert()
1056 {
1057     for( int k = 0; k<= 119 ;k++)
1058     {
1059         putchar('—');
1060     }
1061     putchar('\n');
1062
1063     printf ("1.插入多个元素\n");
1064     printf ("2.定点插入元素\n");
1065
1066     printf ("  ^               / |n");
1067     printf ("  /\ 7           □ _ ^n");
1068     printf ("  /  |           /   /n");
1069     printf ("  |  Z _,<   /       /‘☒n");

```

```

1070     printf (" |      F      /      > \n");
1071     printf (" Y      ' /      ^\n");
1072     printf (" ?● ? ●      ?? <      ^\n");
1073     printf (" () ^      | \ <\n");
1074     printf (" >??_ 彳      | / / \n");
1075     printf (" / ^      / ?<| \ \ \n");
1076     printf (" F_?      ( /      | / / \n");
1077     printf (" 7      | / \n");
1078     printf (" >—r - - '?— _ \n");
1079
1080     for( int k = 0; k<= 119 ;k++)
1081     {
1082         putchar('—');
1083     }
1084     putchar('\n');
1085 }
1086
1087 int compare(int a, int b)
1088 {
1089     if(a == b)
1090     {
1091         return 1;
1092     }
1093     return 0;
1094 }
1095
1096 void visit ( int x)
1097 {
1098     printf ("%d",x);
1099 }

```

附录 C 基于二叉链表二叉树实现的源程序

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

```

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

```

60 status PreOrderTraverse(BiTree T, void(* visit )(BiTree)); // 前序遍历
61 status InOrderTraverse(BiTree T, void(* visit )(BiTree)); // 中序遍历
62 status PostOrderTraverse(BiTree T, void(* visit )(BiTree)); // 后续遍历
63 status LevelOrderTraverse(BiTree T, void(* visit )(BiTree)); // 层序遍
    历
64 status SaveBiTree(BiTree T, char FileName[]); // 保存到文件
65 status LoadBiTree(BiTree &T, char FileName[]); // 从文件中加载
66 int MaxPathSum(BiTree T); // 最大路径和
67 BiTree LowestCommonAncestor(BiTree T, int e1, int e2); // 最近公共祖
    先
68 BiTree InvertTree (BiTree T); // 翻转二叉树
69 void menufirst (); // 管理多个图的菜单
70 void menu(); // 管理单个图的菜单
71 void fun01(); // 管理多个图的封装函数
72 void fun02(BiTree & T ); // 管理单个图的封装函数
73
74 /*----- main主函数 -----*/
75 int main()
76 {
77     system("color 37"); // 设置颜色
78     fun01();
79 }
80
81
82
83
84 /*----- 函数定义 -----*/
85
86 status CreateBiTree(BiTree &T, TElemType definition [])
87 {
88     /*
    
```

```
89      * 根据带空枝的二叉树先根遍历序列definition构造一棵二叉树，
90      * 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。
91      */
92
93      static int i = 0; // 静态变量，记录当前已经处理到的序列下标
94      int j = 0, k = 0; // 循环计数器
95
96      // 第一次调用时，检查数据是否合法
97      if (i == 0) {
98          // 依次检查每个节点关键字是否合法
99          for (j = 0; ( definition + j)->key != -1; j++) {
100              for (k = j + 1; ( definition + k)->key != -1; k++) {
101                  // 如果有两个关键字相同，且不为0，返回错误
102                  if (( definition + j)->key == ( definition + k)->key
                      && (definition + j)->key != 0)
103                      return ERROR;
104              }
105          }
106      }
107
108      // 递归出口：序列遍历结束，T为空值，返回OK
109      if (( definition + i)->key == -1) {
110          T = NULL;
111          i = 0;
112          return OK;
113      }
114
115      // 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
116      if (( definition + i)->key == 0) {
```

```
117         i++;
118         return OK;
119     }
120
121     // 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
122     T = (BiTree)malloc( sizeof(BiTNode));
123     T->lchild = NULL;
124     T->rchild = NULL;
125     T->data = *( definition + i);
126     i++;
127     CreateBiTree(T->lchild, definition );
128     CreateBiTree(T->rchild, definition );
129     return OK;
130 }
131
132 status DestroyBiTree(BiTree &T)
133 { // 将二叉树设置成空，并删除所有结点，释放结点空间
134
135     if (T != NULL) // 如果这个二叉树不为空
136     {
137         if (T->lchild) // 如果左子树不为空
138             DestroyBiTree(T->lchild); // 递归地销毁左子树（因为左子
            树也是一棵二叉树）
139
140         if (T->rchild) // 如果右子树不为空
141             DestroyBiTree(T->rchild); // 递归地销毁右子树（因为右子
            树也是一棵二叉树）
142
143         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）
144     }
```



```
145         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
146     }
147
148     return OK; // 返回操作成功
149 }
150
151 // 初始条件是二叉树 T 存在；操作结果是将二叉树 T 清空
152 status ClearBiTree(BiTree &T)
153 {
154     // 如果二叉树 T 不为空，则需要清空它；否则直接返回 OK
155     if (T != NULL)
156     {
157         // 如果 T 的左子树不为空，则递归清空左子树
158         if (T->lchild)
159             DestroyBiTree(T->lchild);
160
161         // 如果 T 的右子树不为空，则递归清空右子树
162         if (T->rchild)
163             DestroyBiTree(T->rchild);
164
165         // 释放 T 的内存空间，并将 T 的指针设为 NULL
166         free(T); // 使用递归依次释放左子树、右子树、根节点指针
167         T = NULL;
168     }
169
170     // 返回函数执行结果
171     return OK;
172 }
173
174 status BiTreeEmpty(BiTree &T)
```

```
175 {
176     // 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
177     if (T != NULL)
178         return FALSE;
179     else
180         return TRUE;
181 }
182
183 int BiTreeDepth(BiTree T)
184 {
185     // 求二叉树T的深度
186
187     int depth = 0; // 定义变量 depth 并初始化为 0
188
189     if (T != NULL)
190     {
191         int lchilddepth , rchilddepth ;
192         // 递归求解左子树的深度
193         lchilddepth = BiTreeDepth(T->lchild);
194         // 递归求解右子树的深度
195         rchilddepth = BiTreeDepth(T->rchild);
196
197         // 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
198         if ( lchilddepth >= rchilddepth )
199             depth = lchilddepth + 1;
200         else
201             depth = rchilddepth + 1;
202     }
203 }
```

```

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

```

```
232     {
233         printf("二叉树不存在\n");
234         return ERROR;
235     }
236
237     int flag = 0; // 用于标记是否找到了目标结点
238     BiTree st[100], p; // 定义一个数组作为栈，一个指针用于遍历二叉
        树
239     int top = 0; // 栈顶指针
240     st[top++] = T; // 将二叉树根节点入栈
241
242     // 循环遍历二叉树
243     while (top != 0)
244     {
245         p = st[--top]; // 弹出栈顶元素
246
247         // 判断插入的结点关键字是否和二叉树中的其他结点重复
248         if (p->data.key == value.key && e != value.key)
249         {
250             printf("关键字重复\n");
251             return ERROR;
252         }
253
254         // 找到了目标结点
255         if (p->data.key == e)
256         {
257             p->data = value; // 将目标结点的数据修改为新的数据
258             flag = 1; // 标记为已找到
259         }
260
261         // 遍历左右子树
```

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

```

292         return OK; // 插入成功
293     }
294     if (T == NULL)
295         return ERROR; // 如果待插入的二叉树为空，则返回错误
296     BiTree q = (BiTree)malloc( sizeof (BiTNode)); // 动态分配内存并定
        义为二叉树结点
297     q->lchild = q->rchild = NULL; // 左右子树均为空
298     q->data = c; // 新结点的数据域为待插入数据c
299     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
300     {
301         printf ("关键字重复\n");
302         return ERROR;
303     }
304     BiTree p = LocateNode(T, e); // 定位待插入结点的父结点
305     if (!p) // 如果找不到父结点，则返回错误
306         return ERROR;
307     else {
308         if (LR) { // 如果要插入的结点为父节点的右孩子
309             q->rchild = p->rchild; // 将父节点的右子树挂在新结点的
                右子树上
310             p->rchild = q; // 将新结点挂在父节点的右子树上
311             return OK; // 插入成功
312         }
313         if (!LR) { // 如果要插入的结点为父节点的左孩子
314             q->rchild = p->lchild; // 将父节点的左子树挂在新结点的
                右子树上
315             p->lchild = q; // 将新结点挂在父节点的左子树上
316             return OK; // 插入成功
317         }
318     }

```

```
319 }
320
321 // 找到二叉树中最右边的结点
322 BiTree findrightTNode(BiTree T)
323 {
324     // 创建一个栈来保存已遍历的结点
325     BiTree stack[1000], p = NULL;
326     // 定义栈的顶部指针为0
327     int top = 0;
328     // 如果二叉树不为空
329     if (T != NULL)
330     {
331         // 把二叉树的根节点放入栈中
332         stack[top++] = T;
333         // 循环，直到栈为空
334         while (top)
335         {
336             // 取出栈顶元素，并把该元素赋给p
337             p = stack[--top];
338             // 将p的右子树放入栈中
339             if (p->rchild != NULL)
340                 stack[top++] = p->rchild;
341             // 将p的左子树放入栈中
342             if (p->lchild != NULL)
343                 stack[top++] = p->lchild;
344         }
345     }
346     // 返回最后一个遍历到的结点
347     return p;
348 }
349
```

```
350 // 删除结点
351 status DeleteNode(BiTree &T, KeyType e)
352 {
353     if (T == NULL) //如果T是空树，就无法删除，返回错误代码
354         return INFEASIBLE;
355
356     BiTree stack[1000], p, lp, TNode; //定义栈，以及三个指针
357     int top = 0; // 栈顶
358
359     if (T != NULL) //如果T不为空树
360     {
361         if (T->data.key == e) // 如果T的根结点就是要删除的结点
362         {
363             lp = T; // 记下要删除的结点
364             if (T->lchild == NULL && T->rchild == NULL) //如果要删除的结点没有左右子树
365             {
366                 free(lp); // 直接释放内存
367                 T = NULL; //将根结点设为NULL
368                 return OK; //返回成功操作
369             }
370             else if (T->lchild != NULL && T->rchild == NULL) //如果要删除的结点只有左子树
371             {
372                 T = T->lchild; // 将根结点的左子树变成新的根结点
373                 free(lp); // 释放内存
374                 return OK; //返回成功操作
375             }
376             else if (lp->lchild == NULL && lp->rchild != NULL) //如果要删除的结点只有右子树
377             {
```



```

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

```

```

404         return OK; //返回成功操作
405     }
406     else if (lp->lchild != NULL && lp->rchild ==
407             NULL) //如果要删除的结点只有左子树
408     {
409         p->rchild = lp->lchild;
410         free(lp);
411         return OK; //返回成功操作
412     }
413     else if (lp->lchild == NULL && lp->rchild !=
414             NULL) //如果要删除的结点只有右子树
415     {
416         p->rchild = lp->rchild;
417         free(lp);
418         return OK; //返回成功操作
419     }
420     else // 如果要删除的结点既有左子树又有右子树
421     {
422         TNode = findrightTNode(lp->lchild); // 找到要
423         // 删除的结点的左子树的最右结点
424         TNode->rchild = lp->rchild; // 将要删除的结点
425         // 的右子树挂在左子树的最右结点下
426         p->rchild = lp->lchild; // 将左子树作为父结点
427         // 的右子树
428         free(lp);
429         return OK; //返回成功操作
430     }
431 }
432 stack[top++] = p->rchild; // 右子树入栈
433 }
434 if (p->lchild != NULL) //如果栈顶元素有左子树

```

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

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

```
484     st[top++] = T;
485 // 当栈不为空时，循环遍历
486     while (top != 0)
487     {
488 // 指针p指向栈顶元素
489         p = st[--top];
490 // 对p进行访问
491         visit (p);
492 // 如果p右子树不为空，右子树先入栈
493         if (p->rchild != NULL)
494             st[top++] = p->rchild;
495 // 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
496         if (p->lchild != NULL)
497             st[top++] = p->lchild;
498     }
499 // 遍历完成，返回OK
500     return OK;
501 }
502
503 // 中序遍历二叉树T
504 status InOrderTraverse(BiTree T, void(* visit )(BiTree))
505 {
506 // 如果T为空，则直接返回
507     if (T == NULL)
508         return OK;
509 // 如果T非空，则进行遍历操作
510     if (T != NULL)
511     {
512 // 对T的左子树进行遍历
513         if (InOrderTraverse(T->lchild, visit ))
514             {
```

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

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

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



```

601         if (st[p]->rchild == NULL)
602             fprintf (fp, "%d null ", 0); // 如果右子树为空, 写入
                null
603         else {
604             st[p + 1] = st[p]->rchild; // 否则将右子树结点入栈
605             p++; // 指针后移
606             mark[p] = 0; // 新结点状态初始化
607         }
608     }
609     else if (mark[p] == 2) { // 第三次访问该结点
610         mark[p] = 0; // 状态置为未访问
611         st[p] = NULL; // 将该结点出栈
612         p--; // 指针前移
613     }
614 }
615 // 写入结束标志符
616 fprintf (fp, "%d null", -1);
617 fclose (fp); // 关闭文件指针
618 return OK;
619 }
620
621 // 该函数用于读取文件中的数据, 创建一颗二叉树
622 status LoadBiTree(BiTree &T, char FileName[])
623 {
624     // 如果传入的二叉树已经存在, 则无法进行操作
625     if (T != NULL)
626     {
627         printf ("二叉树已经初始化, 无法操作\n");
628         return INFEASIBLE;
629     }
630

```

```
631 //尝试打开文件
632 FILE *fp = fopen(FileName, "r");
633 if (fp == NULL)
634     return ERROR;//文件指针打不开，返回错误
635 TElemType definitionfile [100]; //定义结构体类型数组，用于存储读
    取出来的值
636 BiTree st [100]; //定义指向节点的指针数组
637 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
638
639 //读入文件中的数据，存储到definitionfile[]数组中
640 int t = -1;
641 do {
642     t++;
643     fscanf(fp, "%d%s", &definitionfile[t].key, definitionfile[t].
        others);
644 } while (definitionfile[t].key != -1);
645
646 //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
647 if (definitionfile[0].key != -1) {
648     T = st[0] = (BiTNode*)malloc(sizeof(BiTNode)), mark[0] = 0;
649     st[0]->data = definitionfile[t++];
650 }
651 else
652     return INFEASIBLE;//返回错误
653
654 t = 0;
655 //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
656 while (definitionfile[t].key != -1) {
657     //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
```

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

```

        Mark数组位置+1
685         st[p + 1] = st[p]->rchild;
686         p++;
687         st[p]->data = definitionfile [t];
688         mark[p] = 0;
689     }
690     t++;
691 }
692     // 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
693     else if (mark[p] == 2)
694     {
695         mark[p] = 0; // 重置Mark数组位置上的值为0
696         st[p] = NULL; // 将该位置的指针置NULL
697         p--; // 标记位置指针退回上一级节点
698     }
699 }
700
701     fclose(fp); // 关闭文件
702     return OK;
703 }
704
705 int MaxPathSum(BiTree T)
706 { // 初始条件是二叉树T存在；操作结果是返回根节点到叶子节点的最大
        路径和；
707
708     // 如果当前结点是叶子结点，则直接返回该结点的键值
709     if (T->lchild == NULL && T->rchild == NULL)
710         return T->data.key;
711
712     // 如果左子树为空，则仅考虑右子树节点的路径和
```

```

713     else if (T->lchild == NULL && T->rchild != NULL)
714         return MaxPathSum(T->rchild) + T->data.key;
715
716         // 如果右子树为空，则仅考虑左子树节点的路径和
717     else if (T->lchild != NULL && T->rchild == NULL)
718         return MaxPathSum(T->lchild) + T->data.key;
719
720     // 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
721     int leftmax = 0, rightmax = 0;
722     leftmax = MaxPathSum(T->lchild);
723     rightmax = MaxPathSum(T->rchild);
724     if (leftmax > rightmax)
725         return leftmax + T->data.key;
726     else
727         return rightmax + T->data.key;
728 }
729
730 // 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
731 BiTree LowestCommonAncestor(BiTree T, int e1, int e2)
732 {
733     // 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
734     BiTree p1 = LocateNode(T,e1);
735     BiTree p2 = LocateNode(T,e2);
736     // 设置一个标志变量flag，用于标记是否判断过结点是否存在
737     static int flag = 0;
738     if (flag == 0) // 如果flag是0，说明还没有判断过结点是否存在
739     {
740         flag = 1;
741         // 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，

```

```
742 // 则输出错误并返回NULL
743     if( p1 == NULL || p2 == NULL)
744     {
745         printf("输入的关键字错误\n");
746         return NULL;
747     }
748 }
749
750 // 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
751     if (T == NULL || T->data.key == e1 || T->data.key == e2)
752         return T;
753 // 递归查找左子树
754     BiTree left = LowestCommonAncestor(T->lchild, e1, e2);
755 // 递归查找右子树
756     BiTree right = LowestCommonAncestor(T->rchild, e1, e2);
757 // 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
758     if ( left == NULL)
759         return right ;
760 // 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
761     if ( right == NULL)
762         return left ;
763 // 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
764 // T结点就是e1和e2的公共祖先!
765     return T;
766 }
767
768 // 函数名称: BiTree InvertTree(BiTree T)
769 // 函数功能: 将二叉树T翻转, 使其所有节点的左右节点互换
```

```
770 //参数说明：二叉树T
771 //返回值：翻转后的二叉树T
772
773 BiTree InvertTree (BiTree T)
774 {
775     //如果二叉树为空，则直接返回
776     if (T == NULL)
777         return NULL;
778
779     //递归处理左子树，返回左子树翻转后的结果
780     BiTree left = InvertTree (T->lchild);
781
782     //递归处理右子树，返回右子树翻转后的结果
783     BiTree right = InvertTree (T->rchild);
784
785     //交换左右节点
786     T->lchild = right ;
787     T->rchild = left ;
788
789     //返回翻转后的结果
790     return T;
791 }
792
793 void fun01()
794 {
795     menufirst(); // 输出主菜单，提供可选的操作命令
796     int a ; // 命令编号/选择
797     printf ("请输入一个命令\n");
798     scanf ("%d",&a);
799
800     while (a) // 当输入非0时，继续进行操作
```

```
801     {
802         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
803
804         int feedback; // 操作返回值
805
806         switch (a) { // 根据命令编号进行相应的操作
807             case 1: // 创建一个新的二叉树
808                 printf ("现在进行创建一个新的二叉树\n");
809                 printf ("请输入你想创建的二叉树的名字\n");
810                 char name1[30]; // 用于存储输入的二叉树名字
811                 scanf ("%s",name1);
812                 int i , flag ; flag = 0; // 标记位，用于判断是否已存
                                在同名二叉树
813
814                 // 要进行名字的判断，遍历数组中的所有二叉树名字
815                 for( i =0;i<Lists . length ;i++)
816                 {
817                     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
818                     {
819                         printf ("该二叉树已经存在，创建失败\n");
820                         flag = 1;
821                     }
822                 }
823
824                 if( flag == 0) // 如果不存在同名二叉树，则可以创建
825                 {
826                     // 将新的二叉树名字加入到数组Lists中，并将Lists
                                的长度加1
827                     strcpy ( Lists . elem[ Lists . length ] . name,name1);
```



```
828         Lists . length++;
829         printf ("创建成功\n");
830     }
831     break;
832
833     case 2: // 删除二叉树
834         int flag2 ; // 标记位，用于记录要删除的二叉树在数
                        组中的位置
835         printf ("现在进行删除二叉树的操作\n");
836         printf ("请输入你想删除的二叉树的名字\n");
837         char name2[30]; // 用于存储目标二叉树名字
838         scanf ("%s",name2);
839         flag2 = -1; // flag2用于标记要删除的二叉树在Lists数
                        组中的位置
840
841         // 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
842         for( i =0;i<Lists . length ;i++)
843         {
844             if(strcmp(name2,Lists . elem[i ]. name) == 0)
845             {
846                 flag2 = i;
847             }
848         }
849
850         if( flag2 == -1) // 如果不存在目标二叉树，则无法进
                        行删除操作
851         {
852             printf ("该二叉树不存在，无法删除\n");
853         }
854         else {
```

```
855         feedback = DestroyBiTree( Lists .elem[ flag2 ]. T); //
           调用DestroyBiTree函数销毁指定位置处的二叉树
856
857         if( feedback == OK) // 如果操作成功
858         {
859             // 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
860             int k;
861             for( k = 0 ;k < Lists . length-1 ;k++)
862             {
863                 Lists .elem[k] = Lists .elem[k+1];
864             }
865             Lists . length--;
866             printf ( "删除成功\n");
867         }
868     }
869     break;
870
871     case 3: // 查询创建了哪些二叉树
872         printf ( "现在进行查询创建了哪些二叉树\n");
873         printf ( "所有的二叉树如下:\n");
874
875         // 遍历数组中的所有二叉树名字，输出每个二叉树的名
           称
876         for( i = 0; i<Lists . length ;i++)
877         {
878             printf ( "%d)  %s\n",i+1, Lists .elem[i ]. name);
879         }
880         break;
881
882     case 4: // 对二叉树进行操作
```

```
883     printf("现在进行二叉树的查找和操作\n");
884     printf("请输入你想查找和操作的二叉树的名字\n");
885     char name3[30]; // 用于存储目标二叉树名字
886     scanf("%s",name3);
887     int flag3 ;flag3 = -1; // flag3用于标记要操作的二叉
        树在Lists数组中的位置
888
889     //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
890     for( i =0 ; i<Lists.length ;i++)
891     {
892         if(strcmp( Lists .elem[i ].name,name3) == 0)
893         {
894             flag3 = i;
895         }
896     }
897
898     if( flag3 ==-1) // 如果不存在目标二叉树，则无法进行
        操作
899     {
900         printf("不存在这个二叉树\n");
901         system("pause");
902     }
903     else {
904         //调用fun02函数对特定位置处的二叉树进行操作
905         fun02( Lists .elem[flag3 ].T);
906     }
907     break;
908
909     default :
910         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

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

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

```
        others );
971    } while ( definition [ i++].key != -1);
972    if (CreateBiTree(T, definition ) == OK)
973    {
974        printf ("二叉树创建成功\n");
975    }
976    else
977        printf ("二叉树创建失败! \n");
978    break;
979    case 2:
980        /**
981         * 调用DestroyBiTree函数进行二叉树的销毁
982         * 如果销毁成功，输出相应的提示语句
983         * 如果销毁失败，输出相应的提示语句
984         */
985        printf ("现在进行二叉树的销毁\n");
986        if (DestroyBiTree(T) == OK)
987            printf ("二叉树销毁成功!\n");
988        else
989            printf ("二叉树销毁失败!\n");
990        break;
991    case 3:
992        /**
993         * 调用ClearBiTree函数进行二叉树的清空
994         * 如果清空成功，输出相应的提示语句
995         * 如果清空失败，输出相应的提示语句
996         */
997        printf ("现在进行二叉树的清空\n");
998        if (ClearBiTree(T) == OK)
999            printf ("二叉树清空成功!\n");
1000        else
```

```
1001         printf ("二叉树清空失败!\n");
1002     break;
1003 case 4:
1004     /**
1005      * 调用BiTreeEmpty函数判断二叉树是否为空
1006      * 如果为空，输出相应的提示语句
1007      * 如果不为空，输出相应的提示语句
1008      */
1009     printf ("现在进行二叉树的判空操作\n");
1010     if (BiTreeEmpty(T) == TRUE)
1011         printf ("二叉树是空树!\n");
1012     else if (BiTreeEmpty(T) == FALSE)
1013         printf ("二叉树不是空树!\n");
1014     else
1015         printf ("二叉树不存在!\n");
1016     break;
1017 case 5:
1018     /**
1019      * 调用BiTreeDepth函数求二叉树的深度
1020      * 如果求解成功，输出相应的提示语句
1021      * 如果求解失败，输出相应的提示语句
1022      */
1023     printf ("现在求二叉树的深度\n");
1024     int j5; // 接收二叉树的深度函数的返回值
1025     if (T == NULL) {
1026         printf ("二叉树不存在!\n");
1027         break;
1028     }
1029     j5 = BiTreeDepth(T);
1030     if(j5 == -1)
1031     {
```

```
1032         printf ("操作失败\n");
1033     }
1034     else {
1035         printf ("二叉树的深度为%d!\n", j5);
1036     }
1037     break;
1038 case 6:
1039     /**
1040      * 输入需要查找的结点关键字
1041      * 调用LocateNode函数进行查找
1042      * 如果查找成功，输出相应的提示语句及查找结果
1043      * 如果查找失败，输出相应的提示语句
1044      */
1045     // 定义需要查找的结点关键字
1046     int e6;
1047     // 定义二叉树结点指针
1048     BiTree p6;
1049     // 判断二叉树是否存在
1050     if (T == NULL) {
1051         printf ("二叉树不存在!\n");
1052         break;
1053     }
1054     // 提示用户输入需要查找的结点关键字
1055     printf ("请输入你要查找的结点关键字: \n");
1056     // 读入用户输入的需要查找的结点关键字
1057     scanf ("%d", &e6);
1058     // 调用LocateNode函数进行二叉树的查找操作，返回查
        找结果到p6中
1059     p6 = LocateNode(T, e6);
1060     // 判断是否查找到目标结点
1061     if (p6 == NULL)
```



```
1062         printf("查找失败!\n");
1063     else
1064         // 输出查找结果
1065         printf("查找成功!其值为: %s\n",p6->data.others);
1066     break;
1067 case 7:
1068     int e7, j7; // 定义变量
1069     KeyType k7; // 定义关键字类型
1070     TElemType value7; // 定义结点内容类型
1071     if (T == NULL) { // 如果二叉树为空, 输出提示语句
1072         printf("二叉树不存在!\n");
1073         break;
1074     }
1075     printf("请输入你要赋值的结点的关键字: \n"); // 提示
        输入
1076     scanf("%d", &k7); // 获取关键字
1077     printf("请输入你要赋值的内容: \n"); // 提示输入结点
        内容
1078     scanf("%d %s", &value7.key, &value7.others); // 获取
        结点内容
1079     j7 = Assign(T,k7,value7); // 调用赋值函数, 返回状态
        值
1080     if (j7 == ERROR) // 如果赋值失败, 输出相应的提示语
        句
1081         printf("赋值失败!\n");
1082     else if (j7 == INFEASIBLE) // 如果二叉树不存在, 输
        出相应的提示语句
1083         printf("二叉树不存在!\n");
1084     else // 如果赋值成功, 输出相应的提示语句
1085         printf("赋值成功!\n");
1086     break;
```

```
1087
1088         case 8:
1089             int e8; // 定义变量
1090             BiTree p8; // 定义二叉树指针
1091             if (T == NULL) { // 如果二叉树为空，输出相应的提示
                语句
1092                 printf("二叉树不存在!\n");
1093                 break;
1094             }
1095             printf("请输入你要获得的兄弟结点的关键字: \n"); //
                提示输入
1096             scanf("%d", &e8); // 获取关键字
1097             p8 = GetSibling(T, e8); // 调用获取兄弟结点函数，返
                回兄弟结点指针
1098             if (p8 == NULL) // 如果获取失败，输出相应的提示语
                句
1099                 printf("获取失败!\n");
1100             else // 如果获取成功，输出相应的提示语句和兄弟结
                点内容
1101                 printf("获取兄弟结点成功!其值为: %d %s\n", p8->
                    data.key, p8->data.others);
1102             break;
1103             case 9: // 插入结点
1104                 int e9, j9, LR; // e9、j9、LR为变量，用于存储用户
                    输入的值
1105                 TElemType value9; // value9为结构体类型，用于存储用
                    户输入的内容
1106                 if (T == NULL) { // 如果二叉树不存在，则输出提示信
                    息并结束
1107                     printf("二叉树不存在!\n");
1108                     getchar(); getchar();
```

```
1109         break;
1110     }
1111     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
1112     scanf("%d %d", &e9, &LR); // 从输入流中获取用户输入
1113
1114     printf("请输入待插入的内容 (格式: l a) : \n"); // 提示
        用户输入待插入内容
1115     scanf("%d%s", &value9.key, value9.others); // 从输入流
        中获取用户输入
1116     j9 = InsertNode(T, e9, LR, value9); // 调用InsertNode
        函数插入结点
1117     if (j9 == ERROR) // 如果插入失败, 输出提示信息
        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
        printf("插入结点成功!\n");
1119
1120
1121
1122     break; // 结束case 9
1123
1124     case 10: // 删除结点
1125         int e10, j10; // e10、j10为变量, 用于存储用户输入的
            值
1126         if (T == NULL) { // 如果二叉树不存在, 则输出提示信
            息并结束
1127             printf("二叉树不存在!\n");
1128             break;
1129         }
1130         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
1131         scanf("%d", &e10); // 从输入流中获取用户输入
1132         j10 = DeleteNode(T, e10); // 调用DeleteNode函数删除
```

```

        结点
1133         if (j10 == ERROR) // 如果删除失败，输出提示信息
1134             printf ("删除结点失败!\n");
1135         else if (j10 == OK) // 如果删除成功，输出提示信息
1136             printf ("删除结点成功!\n");
1137
1138         break; // 结束case 10
1139
1140     case 11: // 先序遍历
1141         int j11; // j11为变量，用于存储先序遍历函数的返回值
1142         if (T == NULL) { // 如果二叉树不存在，则输出提示信息并结束
1143             printf ("二叉树不存在!\n");
1144             break;
1145         }
1146         j11 = PreOrderTraverse(T, visit ); // 调用
            PreOrderTraverse函数完成先序遍历
1147         if (j11 == OK) // 如果遍历成功，输出提示信息
1148             printf ("\n完成先序遍历!\n");
1149         else { // 如果遍历失败，输出提示信息
1150             printf ("遍历失败\n");
1151         }
1152
1153         break; // 结束case 11
1154
1155     case 12:
1156         int j12; // 声明一个整型变量j12，用于存储
            InOrderTraverse 函数的返回值
1157         if (T == NULL) { // 如果二叉树 T 不存在
1158             printf ("二叉树不存在!\n"); // 输出提示信息

```

```
1159         break; // 跳出 switch-case 循环
1160     }
1161     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
1162     if (j12 == OK) // 如果遍历成功
1163         printf ( "\n完成中序遍历!\n" ); // 输出提示信息
1164     else
1165         printf ( "\n遍历失败!\n" ); // 输出提示信息
1166
1167     break; // 跳出 switch-case 循环
1168 case 13:
1169     int j13; // 声明一个整型变量 j13, 用于存储
        PostOrderTraverse 函数的返回值
1170     if (T == NULL) { // 如果二叉树 T 不存在
1171         printf ( "二叉树不存在!\n" ); // 输出提示信息
1172
1173         break; // 跳出 switch-case 循环
1174     }
1175     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
1176     if (j13 == OK) // 如果遍历成功
1177         printf ( "\n完成后序遍历!\n" ); // 输出提示信息
1178     else
1179         printf ( "\n遍历失败!\n" ); // 输出提示信息
1180
1181     break; // 跳出 switch-case 循环
1182 case 14:
1183     int j14; // 声明一个整型变量 j14, 用于存储
        LevelOrderTraverse 函数的返回值
1184     if (T == NULL) { // 如果二叉树 T 不存在
1185         printf ( "二叉树不存在!\n" ); // 输出提示信息
```

```
1186
1187         break; // 跳出 switch-case 循环
1188     }
1189     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
1190     if (j14 == OK) // 如果遍历成功
1191         printf ( "\n完成按层遍历!\n" ); // 输出提示信息
1192     else
1193         printf ( "\n遍历失败!\n" ); // 输出提示信息
1194
1195     break; // 跳出 switch-case 循环
1196 case 15:
1197     int j15; // 声明一个整型变量 j15，用于存储 MaxPathSum 函数的返回值
1198     if (T == NULL) { // 如果二叉树 T 不存在
1199         printf ( "二叉树不存在!\n" ); // 输出提示信息
1200         getchar (); getchar (); // 暂停程序执行，等待用户输入
1201
1202         break; // 跳出 switch-case 循环
1203     }
1204     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
1205     printf ( "二叉树最大路径和为: %d!\n", j15 ); // 输出计算结果
1206
1207     break; // 跳出 switch-case 循环
1208
1209 case 16:
1210     if (T == NULL) {
```

```
1211         printf ("二叉树不存在!\n");
1212
1213         break;
1214     }
1215     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
1216     BiTree T16; // 查找到的公共祖先节点
1217     printf ("请输入你要搜索公共祖先的两个结点的关键
                  字: \n");
1218     scanf ("%d %d", &i16, &j16);
1219     T16 = LowestCommonAncestor(T, i16, j16);
1220     if (T16 == NULL)
1221     {
1222         printf ("查找失败!\n");
1223     }
1224
1225     else
1226     {
1227         printf ("查找成功! \n");
1228         printf ("公共祖先的关键字为: %d,其内容为%s\n",
                  T16->data.key, T16->data.others);
1229     }
1230
1231     break;
1232
1233 case 17:
1234     if (T == NULL) {
1235         printf ("二叉树不存在!\n");
1236         getchar ();
1237         break;
1238     }
```

```
1239         InvertTree(T);
1240         printf("已成功翻转二叉树!\n");
1241
1242         break;
1243     case 18:
1244         int j18;
1245         char FileName18[30]; // 保存到的文件名
1246         printf("请输入要写入的文件名:\n");
1247         scanf("%s", FileName18);
1248         j18 = SaveBiTree(T, FileName18);
1249         if (j18 == INFEASIBLE)
1250             printf("二叉树不存在!\n");
1251         else
1252             printf("成功将二叉树写入文件名为: %s的文件中!\n", FileName18);
1253
1254         break;
1255     case 19:
1256         int j19;
1257         char FileName19[30]; // 待读取数据的文件名
1258         printf("请输入要读取的文件名:\n");
1259         scanf("%s", FileName19);
1260         j19 = LoadBiTree(T, FileName19);
1261         if (j19 == INFEASIBLE)
1262             printf("二叉树存在!无法覆盖!\n");
1263         else if (j19 == ERROR) {
1264             printf("读取文件失败!\n");
1265         }
1266         else {
1267             printf("成功将%s文件中的数据读入到二叉树中!\n",
                    FileName19);
```



```
1268         }
1269
1270         break;
1271
1272
1273         case 0:
1274             break;
1275     }
1276     putchar('\n');
1277     printf("请输入下一个命令\n");
1278     scanf("%d",&op);
1279     system("cls");
1280     if(op != 0) // 如果退出就加载第一个菜单
1281     {
1282         menu();
1283     }
1284     else {
1285         menufirst();
1286     }
1287
1288
1289 }
1290 }
1291
1292 void menufirst()
1293 {
1294     for(int k = 0; k <= 119 ;k++)
1295     {
1296         putchar('-');
1297     } putchar('\n');
1298     printf("1.创建一个二叉树\n");
```

```

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

```

华中科技大学课程实验报告

```

1328     printf ("**      1. 创建初始化二叉树   11.前序遍历          **\n");
1329     printf ("**      2. 销毁二叉树          12. 中序遍历          **\n");
1330     printf ("**      3. 清空二叉树          13. 后序遍历          **\n");
1331     printf ("**      4. 二叉树判空          14. 层序遍历          **\n");
1332     printf ("**      5. 求二叉树的深度      15. 最大路径和          **\n");
1333     printf ("**      6. 查找结点            16. 最近公共祖先        **\n");
1334     printf ("**      7. 结点赋值            17. 翻转二叉树          **\n");
1335     printf ("**      8. 获得兄弟结点            18. 二叉树的文件保存
1336     **\n");
1336     printf ("**      9. 插入结点            19. 二叉树的文件加载    **\n");
1337     printf ("**      10. 删除结点           0. Exit              **\n");
1338     printf ("
1339     ");
1340
1341     printf ("      请选择你的操作[0~19]:\n");
1342
1343     // 获取指定结点e的兄弟结点
1344     BiTNode* GetSibling(BiTree T, KeyType e)
1345     {
1346         if (T == NULL) //若二叉树为空，则返回空指针

```

```
1347     {
1348         printf("二叉树不存在\n");
1349         return NULL;
1350     }
1351
1352     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
1353     int top = 0; // 定义一个栈顶指针，初始值为0
1354     st[top++] = T; // 将整棵树压入栈中
1355
1356     while (top != 0) { // 当栈不为空时，进行以下操作
1357         p = st[--top]; // 取出栈顶元素，同时栈顶指针减1
1358         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
1359             return p->lchild; // 则返回p的左子结点
1360         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
1361             return p->rchild; // 则返回p的右子结点
1362         // 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
1363         if (p->rchild->rchild != NULL && p->rchild->lchild != NULL)
1364             st[top++] = p->rchild;
1365         if (p->lchild->rchild != NULL && p->lchild->lchild != NULL)
1366             st[top++] = p->lchild;
1367     }
1368     return NULL; // 若未找到兄弟结点，则返回空指针
1369 }
```

附录 D 基于邻接表图实现的源程序

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

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

```

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

```

```
88  /*----- main主函数 -----*/
89  int main()
90  {
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     // 如果图不存在，返回“不可行”的错误信息
251     if(G.vexnum == 0)
252     {
253         printf("该图不存在或未初始化\n");
254         return INFEASIBLE;
255     }
256     int i = 0;
257     // 循环遍历所有的顶点
258     while (i < G.vexnum)
259     {
260         // 如果找到关键字值为u的顶点，返回它的位序
261         if(G.vertices[i].data.key == u)
262         {
263             return i;
264         }
265         i++;
```

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

```
296     }
297     i++;
298 }
299 // 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
300 if(num != 1)
301 {
302     printf("查找失败,无法操作\n");
303     return ERROR;
304 }
305 // 将找到的符合条件的顶点的值修改成指定的 value 值
306 G.vertices [ flag ]. data = value;
307 // 操作成功，返回 OK
308 return OK;
309 }
310 // 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；
311 // 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；
312 int FirstAdjVex(ALGraph G, KeyType u)
313 {
314     // 如果图不存在，返回错误信息 INFEASIBLE
315     if(G.vexnum == 0) // 图不存在
316     {
317         printf("该图不存在或未初始化\n");
318         return INFEASIBLE;
319     }
320     // 在图 G 中寻找给定关键字对应的顶点
321     int i=0; // 用来计数
322     while (i<G.vexnum)
323     {
```

```

324         if(G.vertices[i].data.key == u)
325         {
326             // 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
327             return G.vertices[i].firstarc->adjvex;
328         }
329         i++;
330     }
331     // 如果未找到给定关键字对应的顶点，返回信息“不存在”，即-1
332     return -1;
333 }
334
335 // 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
336 // 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；
337 // 参数说明：G为有向图，v是源节点，w是目标节点
338 int NextAdjVex(ALGraph G, KeyType v, KeyType w)
339 {
340     // 如果图不存在，则返回“不存在”的信息
341     if (G.vexnum == 0)
342     {
343         printf("该图不存在或未初始化\n");
344         return INFEASIBLE;
345     }
346     int i = 0;
347     int flagv = -1, flagw = -1; // 用来记录v和w对应的下标
348     // 找到v和w在G.vertices数组中的下标
349     while (i < G.vexnum)
350     {

```



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

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

```
411 // 在G.vertices 数组的最后一个位置插入新结点，更新G.vexnum
412     G.vertices [G.vexnum].data = v;
413     G.vertices [G.vexnum].firstarc = NULL;
414     G.vexnum++;
415     return OK;
416 }
417
418 // 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；
419 // 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
420 status DeleteVex(ALGraph &G, KeyType v)
421 // 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
422 {
423     // 请在这里补充代码，完成本关任务
424     // 若图不存在或未初始化，则返回不可行状态
425     if(G.vexnum == 0)
426     {
427         printf("该图不存在或未初始化\n");
428         return INFEASIBLE;
429     }
430     // 若图中只有一个顶点，则无法删除，返回错误状态
431     if(G.vexnum == 1)
432     {
433         printf("图中只有一个顶点，不能删除\n");
434         return ERROR;
435     }
436     int i = 0; // 标记下标
437     // 寻找要删除的顶点
438     while (i < G.vexnum)
439     {
```

```

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

```

```
469     ArcNode * train = NULL; // 记录操作
470     ArcNode * p = NULL;
471     int k = 0;
472     while (k < G.vexnum)
473     {
474         train = G.vertices[k].firstarc;
475         p = train;
476         while (train != NULL)
477         {
478             // 找到与要删除的顶点有关的弧进行删除
479             if (location == train->adjvex)
480             {
481                 if (train == p)
482                 {
483                     G.vertices[k].firstarc = train->nextarc;
484                     train = train->nextarc;
485                     free(p);
486                     p = NULL;
487                     continue;
488                 }
489                 p->nextarc = train->nextarc;
490                 p = train;
491                 train = p->nextarc;
492                 free(p);
493                 p = NULL;
494                 continue;
495             }
496             // 将所有大于要删除位置的顶点位置减一
497             if (train->adjvex > location)
498             {
499
```

```

500         train->adjvex--;
501     }
502     p = train ;
503     train = p->nextarc;
504
505 }
506 k++;
507 }
508 // 删除成功，返回操作成功状态
509     return OK;
510 }
511
512 // 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；
513 // 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>;
514 status InsertArc (ALGraph &G,KeyType v,KeyType w)
515 // 在图G中增加弧<v,w>，成功返回OK,否则返回ERROR
516 {
517     if(G.vexnum == 0) // 如果图不存在
518     {
519         printf ("该图不存在或未初始化\n");
520         return INFEASIBLE;
521     }
522     if(v == w) // 如果插入的是重边
523     {
524         printf ("插入的是重边\n");
525         return ERROR;
526     }
527     int flagv ==-1, flagw ==-1;
528

```

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

```

560         return ERROR;
561     }
562     pv = pv->nextarc;
563 }
564
565 // 分别创建结构体 newv 和 neww, 构建新边
566 ArcNode *newv = (ArcNode *) malloc(sizeof(ArcNode));
567 newv->adjvex = flagv; // 邻接点下标为 w
568 newv->nextarc = NULL; // 下一条边为空
569 if(G.vertices [ flagv ]. firstarc != NULL) // 如果 v 有边
570 {
571     newv->nextarc = G.vertices[ flagv ]. firstarc ; // 新边指向 v 的第
        一条边
572 }
573 G.vertices [ flagv ]. firstarc = newv; // 更新头指针, 即 v 的第一条边
        为新边
574
575 // 和上面的操作类似
576 ArcNode *neww = (ArcNode *) malloc(sizeof(ArcNode));
577 neww->adjvex = flagv; // 邻接点下标为 v
578 neww->nextarc = NULL; // 下一条边为空
579 if(G.vertices [ flagw ]. firstarc != NULL) // 如果 w 有边
580 {
581     neww->nextarc = G.vertices[ flagw ]. firstarc ; // 新边指向 w 的第
        一条边
582 }
583 G.vertices [ flagw ]. firstarc = neww; // 更新头指针, 即 w 的第一条
        边为新边
584
585 G.arcnum++; // 边数加 1
586 return OK; // 插入成功

```



```
587 }
588
589 status DeleteArc(ALGraph &G,KeyType v,KeyType w)
590 //在图G中删除弧<v,w>, 成功返回OK,否则返回ERROR
591 {
592     if(G.vexnum == 0) //图不存在
593     {
594         printf("该图不存在或未初始化\n");
595         return INFEASIBLE;
596     }
597     if(v == w) //如果v和w相等,说明删除环,返回错误
598     {
599         printf("你输入的是环\n");
600         return ERROR;
601     }
602     int i = 0;
603     int flagv = -1; //用来记录下标
604     int flagw = -1;
605     int sign = 0; //用来标记是否有边
606
607     //查找边<v,w>对应的顶点下标
608     while (i<G.vexnum)
609     {
610         if(G.vertices[i].data.key == v)
611         {
612             flagv = i ;
613         }
614         if(G.vertices[i].data.key == w)
615         {
616             flagw = i ;
617         }
618     }
```

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

```

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

```

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

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

```

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

```

```

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

```
797     {
798         if(flag12[i] == 0)
799             { //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
800                 flag12[i]=1;
801                 BFS(G,visit,i);
802             }
803     }
804     return OK;
805
806
807
808     /***** End *****/
809 }
810
811 // visit 函数
812
813 void visit (VertexType p)
814 {
815     printf ("%d %s",p.key,p.others );
816 }
817
818 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
819 {
820     k--; //由于是从起始点算起的距离，所以距离上限k需要减1
821     if(G.vexnum == 0) //如果图不存在，返回NULL
822     {
823         printf ("该图不存在或未初始化\n");
824         return NULL;
825     }
```



```
826     int record[100] = {0}; //记录访问过的结点,初始化为0 (表示未
      访问过)
827     int i = 0;
828     int flag = -1; // flag变量初始化为-1 (表示没找到起始结点)
829
830     //遍历图的顶点,找到起始结点v,记录其位置到flag变量中
831     for( ; i < G.vexnum; i++)
832     {
833         if(G.vertices[i].data.key == v)
834         {
835             flag = i;
836             break;
837         }
838     }
839     record[flag] = 1; //标记起始结点v已经被访问过
840
841     if(flag == -1) //如果未找到起始点,返回NULL
842     {
843         printf("找不到结点\n");
844         return NULL;
845     }
846     static int srr[100]; //静态数组用于存储距离小于k的顶点集合
847     int num = 0; //num变量用于记录已经存储了多少个顶点
848     srr[num++] = flag; //将起始点v加入到顶点集合中
849     //下面进行查找
850     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
851     memset(Que,0,sizeof (Que)); //初始化队列为0 (表示未被访问过)
852     int head = 0, tail = 0; //队列的头和尾指针
853     Que[head][0] = flag; //起始结点v作为队列的第一个元素
854     Que[head][1] = 0; //起始结点v的距离为0
```

```

855
856 // 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
857 while (head <= tail && Que[head][1] != (k+1))
858 {
859
860     ArcNode * p = G.vertices[Que[head][0]].firstarc; // 获取队头
      元素的邻接链表
861     while (p)
862     {
863         if(record[p->adjvex] == 0) // 如果邻接结点未被访问过
864         {
865             if(Que[head][1] <= (k-1))
866             {
867                 srr[num++] = p->adjvex; // 将邻接结点加入到顶点
      集合中
868             }
869
870             tail++; // 队列尾指针加1
871             Que[tail][0] = p->adjvex; // 将邻接结点加入到队列中
872             Que[tail][1] = Que[head][1] + 1; // 计算邻接结点距离
873             record[p->adjvex]++; // 标记邻接结点已经被访问过
874         }
875         p = p->nextarc; // 遍历下一个邻接结点
876     }
877     head++; // 处理完队头结点，队头指针加1
878 }
879 srr[num] = -1; // 将数组以-1结尾，以便在函数外部访问到数组长
      度
880
881 return srr; // 返回存储顶点集合的数组指针

```

```

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

```

```

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

```

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

```
974         // 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
975         if (flag16[i] == 0) {
976             count++;
977             dfs(G, i);
978         }
979     }
980
981     // 返回连通分量的计数器count
982     return count;
983 }
984
985 status SaveGraph(ALGraph G, char FileName[]) // 保存图的数据到文件
986 {
987     if(G.vexnum == 0) // 如果图是空的，直接返回错误
988     {
989
990         printf("图是空的\n");
991
992         return -1;
993
994     }
995     FILE * fp = fopen(FileName, "w"); // 打开文件，只可写入
996     if(fp == NULL)
997     {
998         return ERROR; // 如果无法打开文件，返回错误
999     }
1000
1001     // 先写入结点数 和 边数
1002     fprintf (fp, "%d %d\n", G.vexnum, G.arcnum); // 写入顶点数和边数
1003     // 再写入顶点
```

```

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

```

```

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

```



```

面
1056         p = newnode; // 令p指向新结点
1057     }
1058     newnode = (ArcNode * ) malloc( sizeof(ArcNode)); // 新建
        一个结点
1059     fscanf( fp, "%d ", &newnode->adjvex); // 读取新结点的邻接点
        编号
1060     newnode->nextarc = NULL; // 将新结点的下一条边设为
        NULL
1061 }
1062
1063 }
1064 fclose( fp); // 关闭文件
1065 return OK; // 返回成功
1066 }
1067
1068 void menu()
1069 {
1070     for( int k = 0; k<= 119 ;k++)
1071     {
1072         putchar( '-' );
1073     } putchar( '\n' );
1074     printf( "1. 创建一个图\n" );
1075     printf( "2. 删除一个图\n" );
1076     printf( "3. 查询已经创建的图\n" );
1077     printf( "4. 查找一个图和进行操作\n" );
1078     printf( "0. 退出多个图的管理\n" );
1079
1080     printf( "      ^                / \n" );
1081     printf( "      /\ 7          □ _ \n" );
1082     printf( "      / |          /   \n" );

```

```

1083     printf ("      | Z _,< / /'F\n");
1084     printf ("      |      F / > \n");
1085     printf (" Y      ' / ^n");
1086     printf (" ?● ? ● ?? < ^n");
1087     printf (" () ^ | \ <n");
1088     printf (" >? ?_ ı | / / \n");
1089     printf (" / ^ / ?<| \ \n");
1090     printf (" F_? ( / | / / \n");
1091     printf (" 7 | / \n");
1092     printf (" >—r - - '— _ \n");
1093     for( int k = 0; k<= 119 ;k++)
1094     {
1095         putchar('—');
1096     } putchar('\n');
1097 }
1098
1099 void menu2()
1100 {
1101     for( int k = 0; k<= 119 ;k++)
1102     {
1103         putchar('—');
1104     }
1105     putchar('\n');
1106     printf ("          Menu for Graph On Sequence Structure \n");
1107     //     printf
1108     ("-----\n");
1109     ;
1108     printf ("          1. 创建初始化图          7. 插入
          顶点\n");
1109     printf ("          2. 销毁图          8. 删除
          顶点\n");

```

| | | | |
|------|-------------------------------|-------------------|--------|
| 1110 | printf (" | 3. 查找顶点 | 9. 插入 |
| | 弧 \n"); | | |
| 1111 | printf (" | 4. 顶点赋值 | 10. 删除 |
| | 弧\n"); | | |
| 1112 | printf (" | 5. 获取第一邻接点 | 11. 深度 |
| | 优先搜索\n"); | | |
| 1113 | printf (" | 6. 获取下一邻接点 | 12. 广度 |
| | 优先搜索\n"); | | |
| 1114 | printf (" | 13.查看图关系\n"); | |
| 1115 | for(int k = 0; k<= 119 ;k++) | | |
| 1116 | { | | |
| 1117 | putchar ('-'); | | |
| 1118 | } putchar ('\n'); | | |
| 1119 | printf (" | 14.距离小于k的顶点集合 | 15.顶 |
| | 点间最短路径\n"); | | |
| 1120 | printf (" | 16.图的连通分量 | 17.保 |
| | 存到文件\n"); | | |
| 1121 | printf (" | 18.从文件里面加载 | \n"); |
| 1122 | printf (" | 0. exit | \n"); |
| 1123 | for(int k = 0; k<= 119 ;k++) | | |
| 1124 | { | | |
| 1125 | putchar ('-'); | | |
| 1126 | } putchar ('\n'); | | |
| 1127 | // printf ("一些附加的功能"); | | |
| 1128 | | | |
| 1129 | printf (" | 请选择你的操作[0~13:]"); | |
| 1130 | putchar ('\n'); | | |
| 1131 | for(int k = 0; k<= 119 ;k++) | | |
| 1132 | { | | |
| 1133 | putchar ('-'); | | |
| 1134 | } | | |

```

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

```

```

1165     switch (a) { // 根据不同命令进行不同的操作
1166         case 1: // 如果命令为1
1167             printf ("现在进行创建一个新的图\n");
1168             printf ("请输入你想创建的图的名字\n");
1169             char name1[30]; // 定义字符串变量name1，长度为30
1170             scanf ("%s",name1); // 输入图的名字，保存在name1中
1171             int i , flag ; flag = 0; // 定义整型变量i和flag，flag 初
                始化为0
1172             // 要进行名字的判断
1173             for( i =0;i<graphs.length;i++) // 遍历所有已经存在的
                图
1174             {
1175                 if(strcmp(name1,graphs.elem[i].name) == 0) // 如
                    果名字已经被使用，则提示创建失败
1176                 {
1177                     printf ("该图已经存在，创建失败\n");
1178                     flag = 1; // 将flag的值设为1，表示创建失败
1179                 }
1180             }
1181             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
1182             {
1183                 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
1184                 graphs.length++; // 将已经存在的图的数量加1
1185                 printf ("创建成功\n");
1186             }
1187             break;
1188         case 2: // 如果命令为2
1189             int flag2 ; // 定义整型变量flag2

```

```
1190     printf("现在进行删除图的操作\n");
1191     printf("请输入你想删除的图的名字\n");
1192     char name2[30]; // 定义字符串变量name2，长度为30
1193     scanf("%s",name2); // 输入要删除的图的名字，保存在
                          name2中
1194
1195     flag2 = -1; // flag2 初始化为-1
1196
1197     // 要进行名字的判断
1198     for( i =0;i<graphs.length;i++) // 遍历所有已经存在的
                          图
1199     {
1200         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
1201         {
1202             flag2 = i;
1203         }
1204     }
1205
1206     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
1207     {
1208         printf("该图不存在，无法删除\n");
1209     }
1210     else{ // 如果找到了要删除的图的名字
1211         if(1)
1212         {
1213             int k;
1214             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
1215         {
1216             graphs.elem[k] = graphs.elem[k+1];
1217         }
1218         graphs.length--; // 已存在图的数量减1
1219         printf("删除成功\n");
1220     }
1221 }
1222 break;
1223 case 3: // 如果命令为3
1224     printf("现在进行查询创建了哪些图\n");
1225     printf("所有的图如下:\n");
1226     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
1227     {
1228         printf("%d)  %s\n", i+1, graphs.elem[i].name);
1229     }
1230     break;
1231 case 4: // 如果命令为4
1232     printf("现在进行图的查找和操作\n");
1233     printf("请输入你想查找和操作的图的名字\n");
1234     char name3[30]; // 定义字符串变量name3，长度为30
1235     scanf("%s", name3); // 输入要查找和操作的图的名字，
                                保存在name3中
1236
1237     int flag3 ; flag3 = -1; // 定义整型变量flag3，初始化为
                                -1
1238     for( i = 0 ; i < graphs.length ; i++) // 遍历已存在的图的
                                数组
1239     {
1240         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
1241         {
1242             flag3 = i;
1243         }
1244     }
1245
1246     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
1247     {
1248         printf ("不存在这个图\n");
1249         system("pause"); // 暂停程序的执行，等待用户按下
            任意键
1250     }
1251     else { // 如果找到了要查找和操作的图的名字
1252         fun02(graphs.elem[ flag3 ].G); // 调用fun02函数对该
            图进行操作
1253     }
1254     break;
1255
1256     default : // 如果命令无法识别，则提示输入错误，并重新显
        示菜单
1257         printf ("输入的命令错误，请再次输入");
1258     }
1259     printf ("请输入下一个命令\n");
1260     scanf ("%d",&a); // 提示输入下一个命令，保存在a中
1261     system("cls"); // 清空控制台的输出，准备显示菜单
1262     menu(); // 再次显示菜单
1263 }
1264 }
1265

```



```

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

```

```

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

```

                                data.key,G.vertices[feedback].data.others);
1325         } else {
1326             printf("所要查找的顶点不存在\n");
1327         }
1328         break;
1329     case 4:
1330         printf("现在进行顶点赋值的操作\n");
1331         printf("请输入你想对哪一个关键字进行操作\n");
1332         int key4;    // 存储关键字
1333         scanf("%d",&key4);
1334         printf("请输入你想改变的关键字和名称\n");
1335         VertexType value;
1336         scanf("%d %s",&value.key,value.others);
1337         feedback = PutVex(G,key4,value);
1338         if(feedback == OK)
1339         {
1340             printf("操作成功\n");
1341         }
1342         break;
1343     case 5:
1344         printf("现在进行获取第一邻接点的操作\n");
1345         printf("输入你想操作的关键字\n");
1346         int key5;    // 存储关键字
1347         scanf("%d",&key5);
1348         feedback = FirstAdjVex(G,key5);
1349         if(feedback != -1)
1350         {
1351             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);
1352         } else {

```

```

1353         printf ("操作失败\n");
1354     }
1355     break;
1356 case 6:
1357     printf ("现在进行获取下一邻接点的操作\n");
1358     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");
1359     int v,w;    // 来存储顶点的值
1360     scanf ("%d %d",&v,&w);
1361     feedback = NextAdjVex(G,v,w);
1362     if (feedback != -1)
1363     {
1364         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);
1365     } else {
1366         printf ("操作失败\n");
1367     }
1368     break;
1369 case 7:
1370     printf ("现在进行插入顶点的操作\n");
1371     printf ("输入你想插入的顶点的关键字和名称\n");
1372     VertexType v7;    // 存储插入的顶点信息
1373     scanf ("%d %s",&v7.key,v7.others);
1374     feedback = InsertVex (G,v7);
1375     if (feedback == OK)
1376     {
1377         printf ("插入成功\n");
1378     } else {
1379         printf ("插入失败\n");
1380     }

```

```
1381         break;
1382     case 8:
1383         printf("现在进行删除顶点的操作\n");
1384         printf("请输入你想删除的顶点的关键字\n");
1385         int key8;    // 存储关键字
1386         scanf("%d",&key8);
1387         feedback = DeleteVex(G,key8);
1388         if(feedback == OK)
1389         {
1390             printf("操作成功\n");
1391         }
1392         else {
1393             printf("操作失败\n");
1394         }
1395         break;
1396     case 9:
1397         printf("现在进行插入弧的操作\n");
1398         int v9,w9;    // 存储边的两个顶点
1399         printf("输入你想插入的弧\n");
1400         scanf("%d %d",&v9,&w9);
1401         feedback = InsertArc (G,v9,w9);
1402         if(feedback == OK)
1403         {
1404             printf("操作成功\n");
1405         }
1406         else {
1407             printf("操作失败\n");
1408         }
1409         break;
1410     case 10:
1411         printf("现在进行删除弧的操作\n");
```

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

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

```
1473     }
1474     else {
1475         printf ("距离小于%d 的顶点的集合是: \n",k);
1476         while ((*p) != -1)
1477         {
1478             printf ("%d %s\n",G.vertices[*p].data.key,G.
                vertices [*p].data.others);
1479
1480             p++;
1481         }
1482         printf ("操作成功\n");
1483     }
1484     break;
1485 case 15:
1486     printf ("现在进行顶点间的最短路程的操作\n");
1487     printf ("请输入顶点v和顶点w的关键字\n");
1488     int v15,w15;    // 存储两个关键字
1489     scanf ("%d %d",&v15,&w15);
1490     feedback = ShortestPathLength (G,v15,w15);
1491     if (feedback != -1)
1492     {
1493         printf ("最短路径为 %d \n",feedback);
1494
1495     } else {
1496         printf ("操作失败\n");
1497     }
1498     break;
1499 case 16:
1500     printf ("现在进行图的连通分量的计算\n");
1501     feedback = ConnectedComponentsNums(G);
1502     if (feedback != 0)
```



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

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