

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术20203**

**学 号： U202015359**

**姓 名： 张庙松**

**指导教师： 陈奇**

**报告日期： 2021年 6月 30 日**

**计算机科学与技术学院**

目 录

[1 基于顺序存储结构的线性表实现 2](#_Toc458159879)

[1.1 问题描述 2](#_Toc458159880)

[1.2 系统设计 2](#_Toc458159882)

[1.3 系统实现 5](#_Toc458159883)

[1.4 系统测试 11](#_1.3_系统实现)

[1.5 实验小结 17](#_Toc458159884)

[2 基于链式存储结构的线性表实现 18](#_Toc458159885)

[2.1 问题描述 18](#_Toc458159886)

[2.2 系统设计 18](#_Toc458159887)

[2.3 系统实现 20](#_Toc458159888)

[2.4 系统测试 31](#_2.4_系统测试)

[2.5 实验小结 37](#_Toc458159889)

[3 基于二叉链表的二叉树实现 38](#_Toc458159890)

[3.1 问题描述 38](#_Toc458159891)

[3.2 系统设计 39](#_Toc458159892)

[3.3 系统实现 43](#_Toc458159893)

[3.4 实验小结 73](#_Toc458159894)

[4 基于二叉链表的二叉树实现 74](#_Toc458159895)

[4.1 问题描述 74](#_Toc458159896)

[4.2 系统设计 74](#_Toc458159897)

[4.3 系统实现 78](#_4.3_系统实现)

[4.4 实验小结 110](#_Toc458159899)

[参考文献 111](#_Toc458159900)

[附录A 基于顺序存储结构线性表实现的源程序 112](#_Toc458159901)

[附录B 基于链式存储结构线性表实现的源程序 132](#_Toc458159902)

[附录C 基于二叉链表二叉树实现的源程序 148](#_Toc458159903)

[附录D 基于邻接表图实现的源程序 174](#_Toc458159904)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

线性表是最常用且是最简单的一种数据结构。形如：A1、A2、A3….An这样含有有限的数据序列，我们就称之为线性表。为了加深我们对线性表的理解与运用，在这次实验中，构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示。定义了线性表的初始化表、销毁线性表、清空线性表、判定空表、求表长和获得元素等基本操作对应的函数，并给出适当的操作提示显示，可以选择以文件的形式进行存储和加载，也可以将线性表存入相应的文件中，从文件中获得线性表，并且构造了一个具有菜单演示功能的系统，可以实现多个线性表管理。

根据本次实验要求，本次实验应该以顺序表的物理结构为基础，实现基础的12种运算。

根据本次实验要求，本次实验要实现良好的功能展示程序，在主函数中完成函数调用所需的参数值的准备和实现函数运行结果的展示，并且提供适当的操作提示信息，演示系统应当实现输入验证和异常结果的反应，便于对于异常的输入进行提示，并能够让用户重新输入。

根据本次实验要求，本次实验应该要能够进行文件的指定路径的保存和读取并且还要能够实现多线性表的管理，多线性表的管理应该实现新建等功能。

## 1.2 系统设计

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

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;//抽象数据元素

int length;//表长

int listsize;//存储容量

}SqList;

具体运算功能定义如下。

⑴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。

⑵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。

⑶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。

⑷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。

⑸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。

⑹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。

⑺查找元素：函数名称是LocateElem(L,e)；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。

⑻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。

⑼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。

⑽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≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

⑿遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。

以下为附增函数：

存储表：函数名称是SaveList(L,FileName),其中FileName为数组，在函数中写好，为存储的目的文件，操作结果是对于线性表中已有的元素进行存储。

读取文件：函数名称是LoadList(L,FileName)，其中FileName为数组，在函数中写好，为读取的目的文件，操作结果是对于文件中的元素，依次写入到空的线性表中。

多线性表管理的物理结构如下：

typedef struct{ //线性表的管理表定义

struct { char name[30];

SqList L;

} elem[10];

int length;

int listsize;

}LISTS;

初始化线性表集合：函数名称是InitLists(Lists)，建立一个线性表集合，使得集合中的元素可以实现基础的12种操作。

新增集合中的线性表：函数名称AddList(Lists,ListName)，对于线性表集合中，增加一个名为ListName的线性表。

查找集合中的线性表：函数名称LocateList(Lists，ListName)，其基本操作是在线性表集合中，查找名为ListName的线性表，并且返回该线性表在线性表集合中的位置。

遍历线性表集合：函数名称ListsTraverse(Lists)，其基本操作是遍历线性表集合中的元素，即依次遍历各个线性表，并依次输出。

将集合中的线性表载入：函数名称为EditList(Lists,L)，其基本操作是将线性表集合中的指定线性表载入到初始面板的线性表中，方便对于该线性表进行各种基础操作。

删除集合中线性表：函数名称RemoveList(Lists,ListName)，其基本操作为移除线性表集合中名称为ListName的线性表。

将线性表存储到集合中：函数名称为SaveEdit(Lists,L)，其基本操作为将原本在初始面板中的线性表存储到原线性表集合中，对于原线性表集合完成修改。

## 1.3 系统实现

1. 本系统通过命令式菜单来调用各函数功能，系统开始运行时会输出菜单界面，依照菜单中给出的各功能的指令输入命令，以实现相关功能的使用。

系统的具体实现方式为：

首先根据线性表的基本操作实现完成基本函数构建，然后通过对于多种情况的考虑完善线性表的各个子函数。

在主函数内部完成函数调用所需实参值的准备和函数执行结果的显示。利用op存储当前命令，并且采用switch-case语句，依据输入的op进行选择，调用函数实现相应功能。

每个子函数的实现方法如下：

⑴初始化表InitList(L)：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。首先利用malloc函数在内存中寻找一片大小为LIST\_INIT\_SIZE\*sizeof(ElemType)的存储空间，将其首地址赋给线性表L中的数组首地址，后将L.length置0，L.listsize置为LIST\_INIT\_SIZE，之后返回OK。当L已存在时，便不可再次初始化，此时需返回INFEASIBLE

⑵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。将顺序表L中数组的首地址进行释放，调用free函数，并将该数组的首地址置为空指针NULL，L.length置为0，L.listsize置为0，如果线性表不存在，那么应该先初始化线性表。

⑶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。首先将当前线性表摧毁(执行(2))，之后再次将线性表L初始化即可。

⑷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。判断线性表L.length的大小，如果为0则返回TRUE,否则返回FALSE。

⑸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。当线性表不存在时，函数返回INFEASIBLE；否则返回线性表长度即可。

⑹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。当线性表不存在时，函数返回INFEASIBLE；当线性表存在，但是i < 1或者i > L.length时，选取的位置非法，故返回ERROR；当线性表存在且i >= 1, i <= L.length时(即选取位置合法时)，将L.elem[i]中数据赋值到e，返回OK即可。

⑺查找元素：函数名称是LocateElem(L,e)；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。当线性表不存在时，函数返回INFEASIBLE；当线性表存在时，扫描线性表中所有元素，并在扫描过程中将每一个已在线性表中的元素与e比较，如果相等则返回当前元素在数组中的序号+1，如果扫描过程中没有任何一个元素与e相等，查找失败，则返回ERROR。该程序实现的相应流程图如下图1-1所示



图1-1查找元素的相应流程图

⑻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。当线性表不存在时，函数返回INFEASIBLE; 当线性表存在时，扫描线性表，且扫描过程中比较当前位置元素L.elem[i]是否与cur\_e相等，如果相同，则继续判断i是否为0，如果为0，则返回ERROR，表明此元素无前驱；如果i不为0，则将L.elem[i – 1]的值赋给pre\_e，并返回OK；此外，若线性表扫描过程中不存在i使得L.elem[i]等于cur\_e，则函数返回ERROR，表明不存在此元素。该程序实现的相应流程图如图1-2所示



图1-2获得前驱元素的程序流程图

⑼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。当线性表不存在时，函数返回INFEASIBLE; 当线性表存在时，扫描线性表，且扫描过程中比较当前位置元素L.elem[i]是否与cur\_e相等，如果相同，则继续判断i是否为L.length-1，如果为L.length-1，表明这已经是线性表最后一个元素，无后继，函数返回ERROR；如果i不为L.length-1，则将L.elem[i+1]的值赋给next\_e，并返回OK；此外，若线性表扫描过程中不存在i使得L.elem[i]等于cur\_e，则函数返回ERROR，表明不存在此元素，当然在线性表中也就不存在它的后继。该程序流程图与获得前驱流程图相近。

⑽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≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。当线性表不存在时，函数返回INFEASIBLE；当线性表存在但是i<1或者i> L.length+1时，插入位置非法，函数返回ERROR；当线性表存在且i >= 1, i <= L.length + 1时，首先判断L.length + 1是否大于L.listsize，如果是，则应利用realloc函数对该线性表的数组重新分配大小为LIST\_INIT\_SIZE\*sizeof(ElemType)的存储空间，之后将原数组中地址位于[L.elem+i-1,L.elem+L.length–1]中的元素全部向地址增大的方向移动一位，之后将e的值赋给L.elem[i–1]，函数返回OK即可；如果L.length+1<=L.listsize，只需要将原数组中地址位于[L.elem+i,Lelem+L.length–1]中的元素全部向地址增大的方向移动一位，之后将e的值赋给L.elem[i–1]，函数返回OK即可。该程序实现的流程图如图1-3所示



图1-3插入元素的程序流程图

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。当线性表不存在时，函数返回INFEASIBLE；当线性表存在但是i < 1或者i > L.length时，删除位置非法，函数返回ERROR；当线性表存在且i >= 1, i <= L.length时，将数组中地址位于[ L.elem + i, L.elem + L.length – 1 ]的所有元素向地址减小的方向移动一位，后将原线性表长度减1即可。该程序实现的流程图与插入元素的流程图相近

⑿遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。当线性表不存在时，函数返回INFEASIBLE；当线性表存在且长度为0时，输出“空”，函数返回OK；当线性表存在且长度不为0时，利用循环结构将线性表扫描一遍，并输出线性表中每一个元素的值，函数返回OK。

线性表文件存取的子函数：

(13)线性表数据存入文件SaveList(L, SaveFile):利用文件指针以”w+”的形式打开(文件不存在时自动创建)文件名为SaveFile的文件，如果此步打开文件失败则函数返回ERROR，否则就将线性表L扫描一遍，并将每一个元素输入到目标文件中，且每输入一个元素后就立即输入一个空格，否则会导致存入文件中的数据连在一起难以取出。输入完毕后利用fclose函数关闭该目标文件。最后返回OK；

(14)读取文件中数据存入线性表LoadList(L, LoadFile)：利用文件指针以”r”的形式打开文件名为LoadFile的文件，如果此步打开文件失败则函数返回ERROR，否则就将文件中每一个数据利用fscanf函数读入线性表，读取完毕后利用fclose函数关闭该目标文件。最后返回OK；

线性表集合操作：

(15)线性表集合初始化InitLists(Lists):首先判断原线性表集合中第一个线性表是否存在，若存在则表明线性表集合中可能已存在数据，不可再次进行初始化，返回INFEASIBLE;否则就将线性表集合长度置为0，对第一个线性表，利用之前已定义的InitList(L)函数将其初始化后，返回OK即可。

(16)向集合中新增线性表AddList(Lists, ListName)：首先判断，Lists当前长度是否已经是最大容量，如果是，则返回OVERFLOW，如果不是，则先利用strcpy函数将ListName拷贝至Lists.elem[Lists.length].name，再利用已定义的函数InitList(L)和InitInput(L)初始化Lists.elem[Lists.length].L，完成后将线性表集合的长度加一，再返回OK即可。

(17)查找线性表LocateList(Lists, ListName)：扫描当前线性表集合，过程中利用strcmp函数判断当前线性表名称是否与ListName相同，若有相同则返回当前线性表的逻辑位置;若扫描结束仍未发现名为ListName的线性表，则返回0，表明查无此表。

(18)遍历集合ListsTraverse(Lists)：首先判断线性表集合是否存在，不存在则返回INFEASIBLE，若存在再判断长度是否为0，若为0则返回0，否则扫描线性表集合，并输出每一个线性表的名称和其中的数据，返回OK。

(19) 将集合中的线性表载入：函数名称为EditList(Lists,L)，其基本操作是将线性表集合中的指定线性表载入到初始面板的线性表中，方便对于该线性表进行各种基础操作。输入线性表名称，声明变量k对于线性表集合进行调用k=LocateList(Lists, ListName)，对于k进行判断线性表集合中是否有该线性表，如果有，则将初始面板中的线性表改为集合中指定的线性表。

(20)删除集合中名为ListName的线性表RemoveList(Lists, ListName):首先，扫描当前线性表集合，并在扫描过程中利用strcmp函数判断每一个线性表的名称是否与ListName相同，若相同则将之后的每一个线性表向前移动一位后，将集合长度减一，返回OK即可；如果扫描结束后仍未找到某个名为ListName的线性表，返回0即可。

(21) 将线性表存储到集合中：函数名称为SaveEdit(Lists,L)，其基本操作为将原本在初始面板中的线性表存储到原线性表集合中，对于原线性表集合完成修改。输入线性表名称，声明变量k对于线性表集合进行调用k=LocateList(Lists, ListName)，对于k进行判断线性表集合中是否有该线性表，如果有，则将线性表集合中的指定线性表修改为初始面板中的线性表。

**1.4 系统测试**

根据本次实验要求，系统测试包括如下内容：

（1）测试准备

（2）测试计划

（3）测试结果

（4）测试分析

1.测试准备：

在本次测试中，主要测试线性表集合操作的准确性以及完备性，对于集合中的线性表要进行初始化，输入，等多种函数的实现，并且，集合中的线性表还能够进行存储，并且能够读取到集合中的任意线性表中。

2.测试计划：

在本次测试中，由于线性表物理结构的简单性，所以直接从键盘中输入线性表的内容，通过测试不同的线性表，在该程序中，为实现简洁性，所以只用到了3个线性表，但是在实际运用中可以达到所需要的要求。

3.测试结果：

此次系统测试不包含线性表的载入与保存的输入选择，对于线性表集合中的进行基本操作时，应该进行载入，在修改完成后，应该保存到线性表集合中。

|  |  |  |  |
| --- | --- | --- | --- |
| 线性表集合主要功能测试 | 参与的线性表序号 | 输入的参数 | 预计的输出结果 |
| 初始化线性表集合 | 无 | 15 | 初始化成功！ |
| 新增一个名称为1，内容为123的线性表 | 1 | 16 1 1 1 2 3 0 | 线性表"1"存储成功！ |
| 新增一个名称为2，内容为456的线性表 | 2 | 16 2 1 4 5 6 0 | 线性表"2"存储成功！ |
| 展示线性表2 | 2 | 12 | 当前线性表中数据如下:  4 5 6 |
| 对于线性表2，向其中第二个位置插入数据2 | 2 | 10 2 2 | 插入成功！  当前线性表中数据如下:  4 2 5 6 |
| 遍历线性表集合 | 1、2 | 18 | 当前线性表集合中共包含以下2个线性表：  1  1 2 3  2  4 2 5 6 |
| 存储线性表2到指定文件中 | 2 | 13 | 已将数据存入 D:\数据结构实验\文件.dat |
| 将文件中的数据读取到线性表1中 | 1 | 3 14 | 数据已从 D:\数据结构实验\文件.dat 中读出 |
| 再次遍历线性表集合 | 1、2 | 18 | 当前线性表集合中共包含以下2个线性表：  1  4 2 5 6  2  4 2 5 6 |
| 删除线性表2 | 2 | 20 | 线性表2删除成功！ |
| 再次遍历线性表集合 | 1 | 18 | 当前线性表集合中共包含以下1个线性表：  1  4 2 5 6 |

1. 对于线性表进行初始化，程序运行结果如图1-4所示

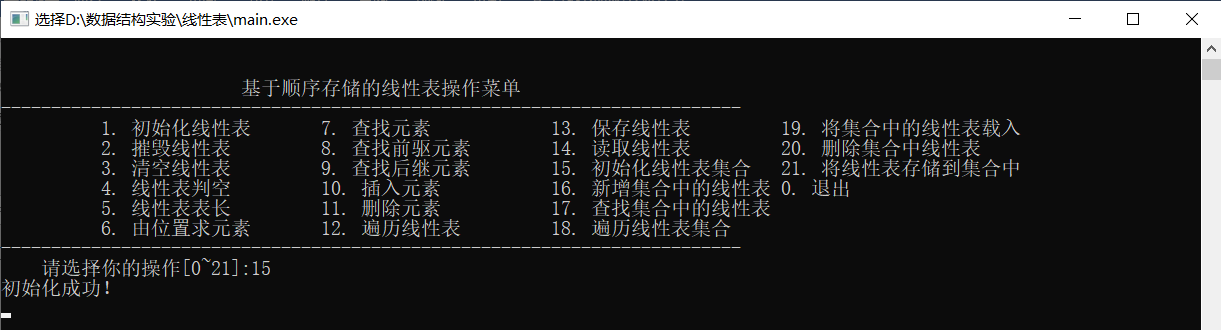


图1-4初始化线性表的运行结果

1. 新增一个名称为1，内容为123的线性表，程序运行结果如图1-5所示

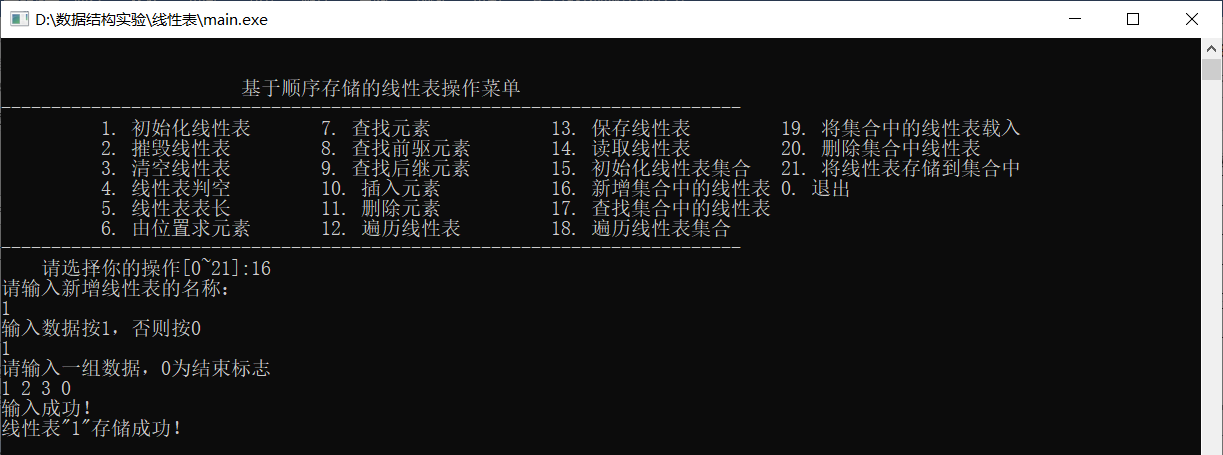


图1-5新增线性表的运行结果

1. 新增一个名称为2，内容为456的线性表，程序运行结果如图1-6所示

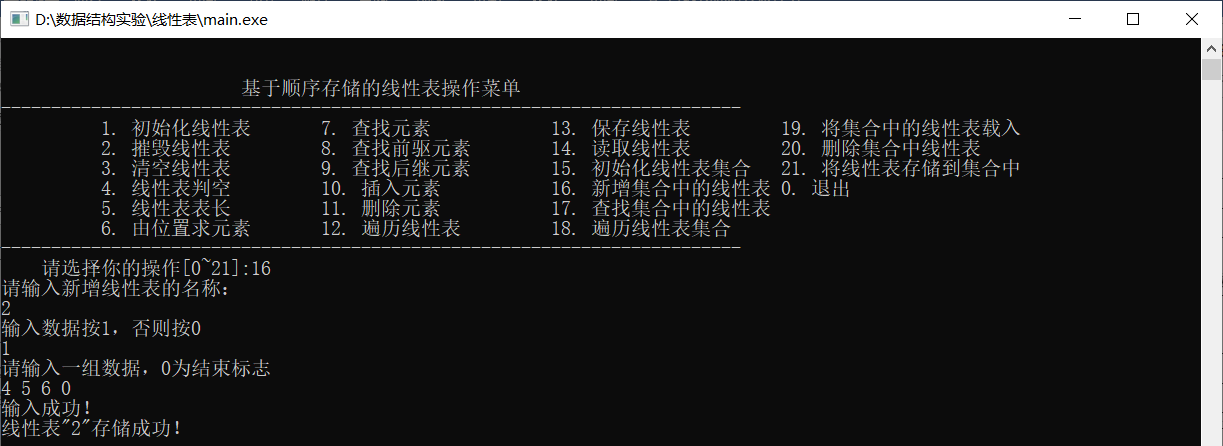


图1-6新增线性表运行结果

1. 展示线性表2，程序运行结果如图1-7所示

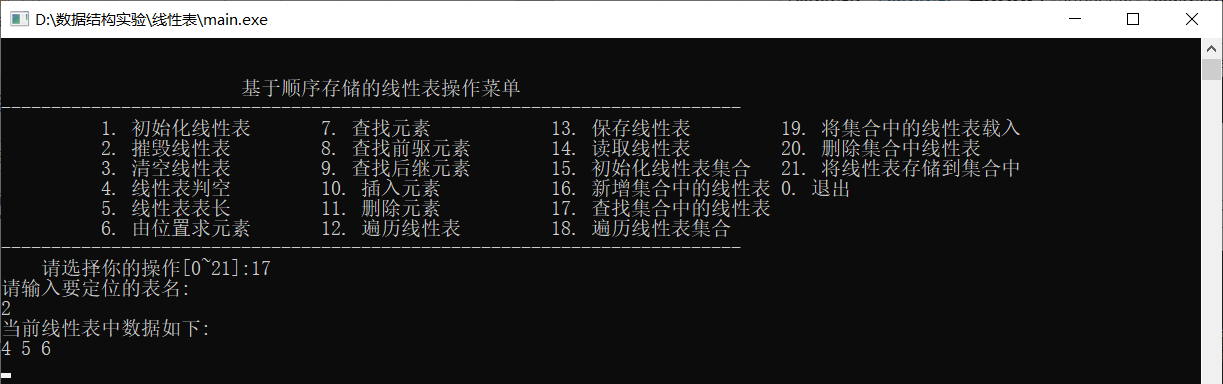


图1-7展示线性表的运行结果

1. 对于线性表2，向其中第二个位置插入数据2，程序运行结果如图1-8所示

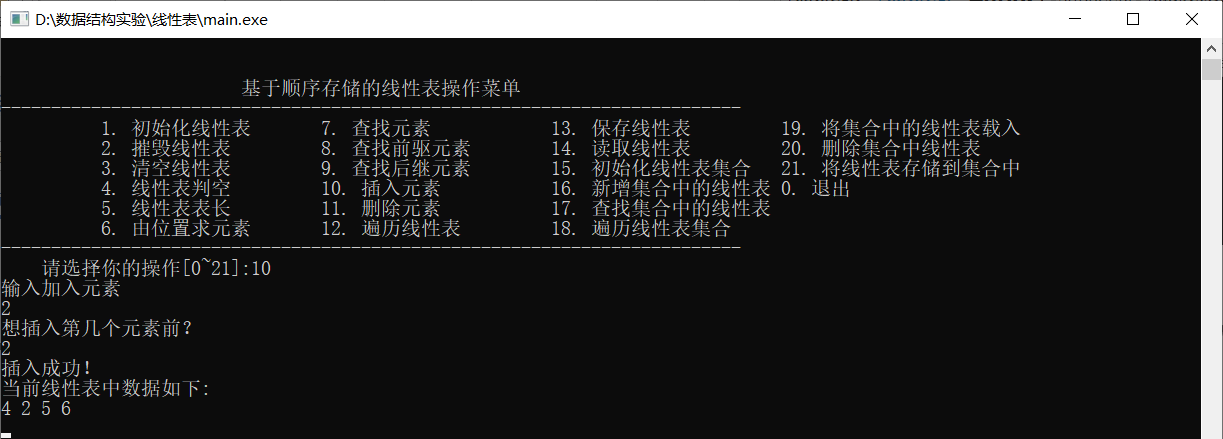


图1-8线性表元素插入的运行结果

1. 遍历线性表集合，运行结果如图1-9所示

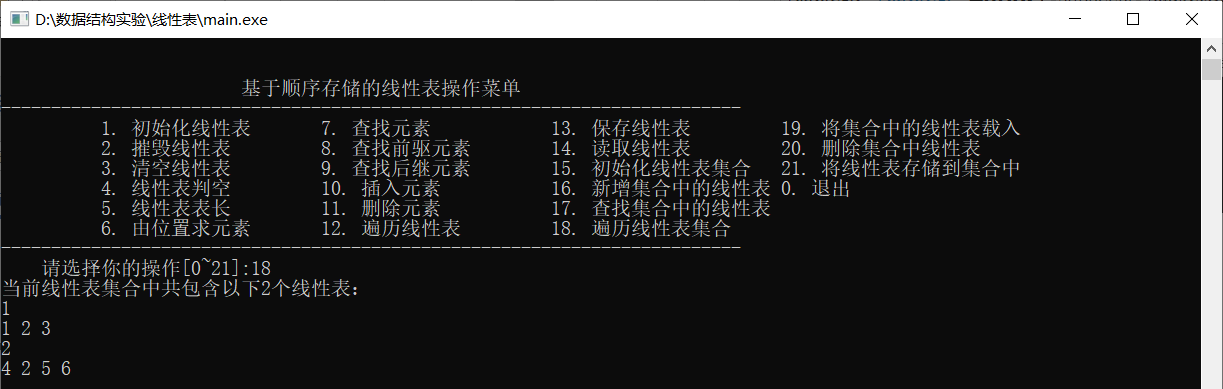


图1-9遍历线性表集合的运行结果

1. 存储线性表2到指定文件中，运行结果如图 1-10所示

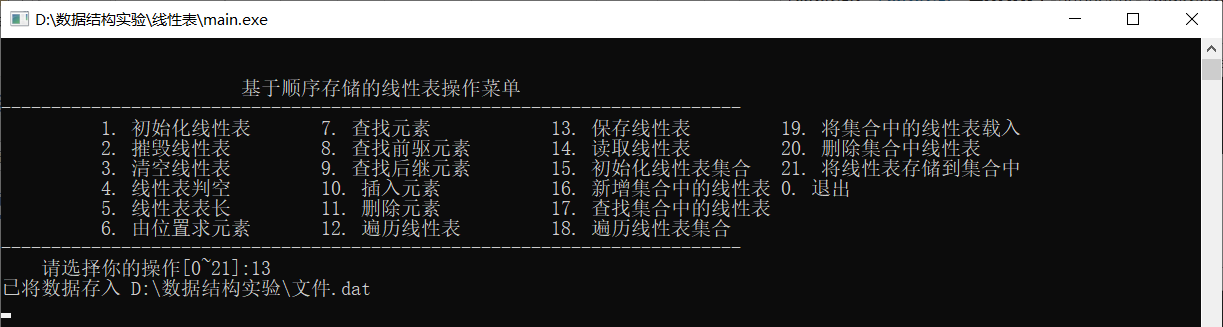


图1-10存储线性表的运行结果

1. 将文件中的数据读取到线性表1中，运行结果如图1-11所示

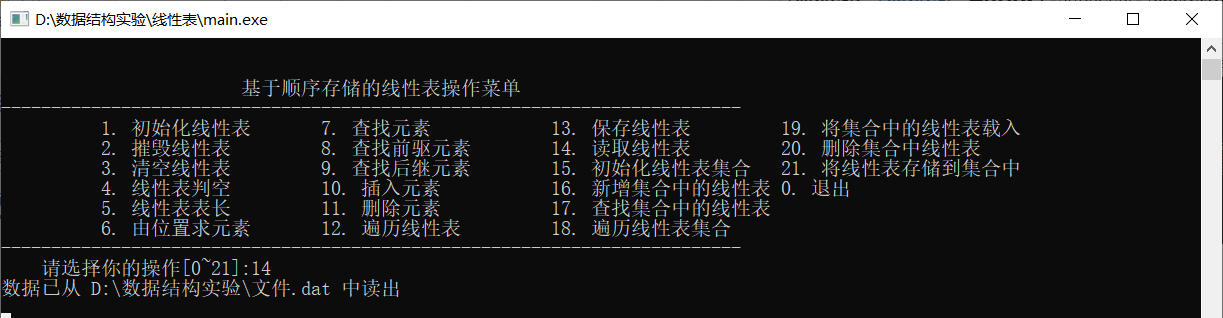


图1-11文件读取到线性表的运行结果

1. 再次遍历线性表集合，运行结果如图1-12所示

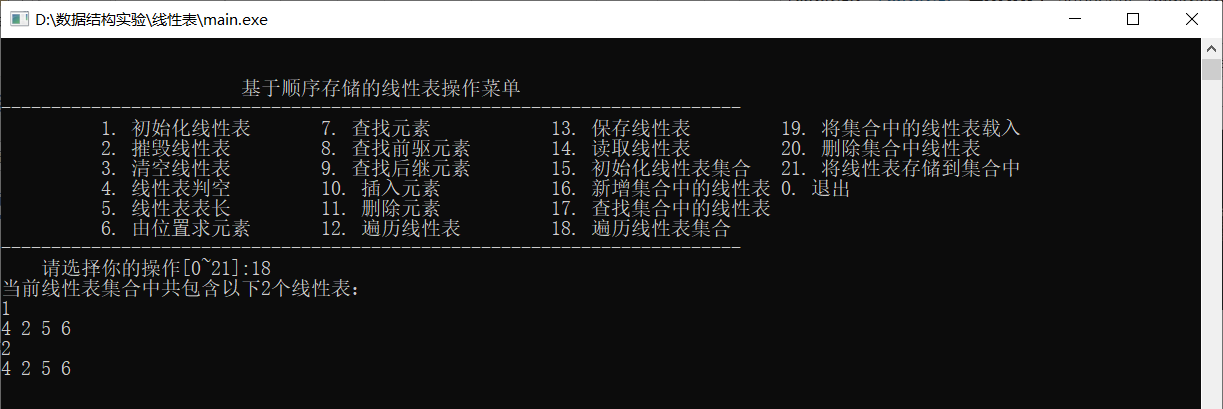


图1-12再次遍历线性表集合的运行结果

1. 删除线性表2，运行结果如图1-13所示

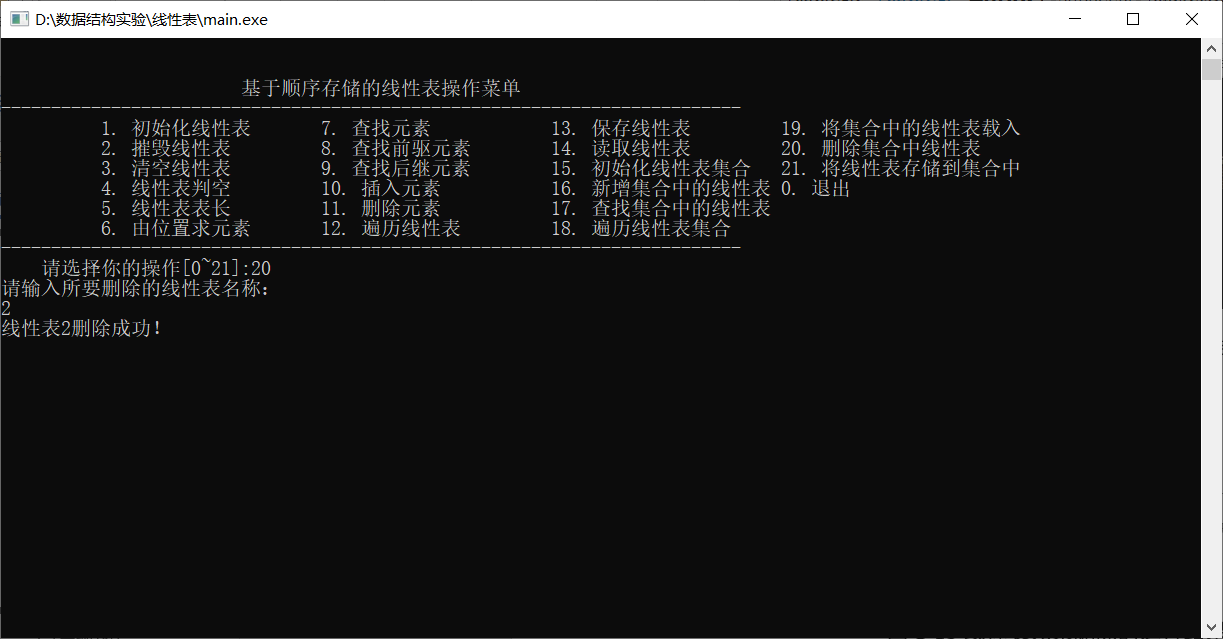


图1-13删除线性表2的运行结果

1. 再次遍历线性表集合，运行结果如图1-14所示

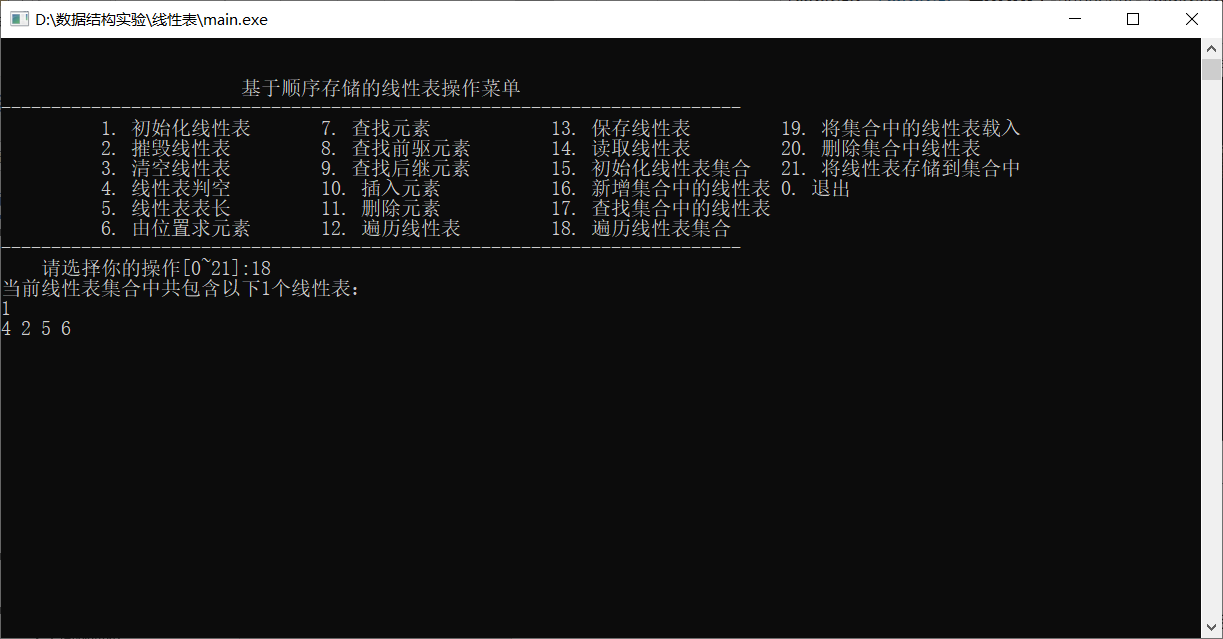


图1-14再次遍历线性表集合的运行结果

4.测试分析：

本次实验测试的结果都较为符合预期，但是，由于一开始程序设计时的思路不对，导致线性表集合的中的线性表需要手动的进行保存，如果没有保存会导致错误，以及在集合中的线性表实际上属于二级菜单，而线性表中的基本操作是在一级菜单中，所以会有一定的逻辑复杂性。

## 1.5 实验小结

通过本次实验，加深了我对于线性表的理解和运用，在实验过程中，平台中的对于线性表的多重操作都需要进行线性表的判断，判断线性表是否存在，以及多次进行边界判断，体现了算法的健壮性，这是之前的课程中所没有的。对于线性表的文件保存操作未能完成，寻求了老师和同学的帮助后才能够记起。在完成了初次的实验检查后，发现自己的系统并没能按照要求，因此对于原本的界面进行修改。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

线性表是由n（n≥0）个数据元素（结点）a[0]，a[1]，a[2],…，a[n-1]组成的有限序列。本实验的目的是封装一个基于顺序存储结构的线性表，提供线性表基本的、常用的12种运算和操作。要求中还有关于文件存储和读取的细节需要实现，使得线性表可以实现内外存交换，便于数据读写。同时，需要实现一个演示系统实现简单的演示，以此作为可用性检查的工具。

根据本次实验要求，本次实验应该以顺序表的物理结构为基础，实现基础的12种运算。

根据本次实验要求，本次实验要实现良好的功能展示程序，在主函数中完成函数调用所需的参数值的准备和实现函数运行结果的展示，并且提供适当的操作提示信息，演示系统应当实现输入验证和异常结果的反应，便于对于异常的输入进行提示，并能够让用户重新输入。

根据本次实验要求，本次实验应该要能够进行文件的指定路径的保存和读取并且还要能够实现多线性表的管理，多线性表的管理应该实现新建等功能。

## 2.2 系统设计

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

typedef struct LNode{ //单链表（链式结构）结点的定义

ElemType data;

struct LNode \*next;

}LNode,\*LinkList;

具体运算功能定义如下。

⑴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。

⑵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。

⑶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。

⑷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。

⑸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。

⑹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。

⑺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。

⑻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。

⑼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。

⑽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≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

⑿遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。

以下为附增函数：

存储表：函数名称是SaveList(L,FileName),其中FileName为数组，在函数中写好，为存储的目的文件，操作结果是对于线性表中已有的元素进行存储。

读取文件：函数名称是LoadList(L,FileName)，其中FileName为数组，在函数中写好，为读取的目的文件，操作结果是对于文件中的元素，依次写入到空的线性表中。

多线性表管理的物理结构如下：

typedef struct { //线性表的管理表定义

struct {

char name[30];

LinkList L;

} elem[10];

int length;

int listsize;

}LISTS;

初始化线性表集合：函数名称是InitLists(Lists)，建立一个线性表集合，使得集合中的元素可以实现基础的12种操作。

新增集合中的线性表：函数名称AddList(Lists,ListName)，对于线性表集合中，增加一个名为ListName的线性表。

查找集合中的线性表：函数名称LocateList(Lists，ListName)，其基本操作是在线性表集合中，查找名为ListName的线性表，并且返回该线性表在线性表集合中的位置。

遍历线性表集合：函数名称ListsTraverse(Lists)，其基本操作是遍历线性表集合中的元素，即依次遍历各个线性表，并依次输出。

将集合中的线性表载入：函数名称为EditList(Lists,L)，其基本操作是将线性表集合中的指定线性表载入到初始面板的线性表中，方便对于该线性表进行各种基础操作。

删除集合中线性表：函数名称RemoveList(Lists,ListName)，其基本操作为移除线性表集合中名称为ListName的线性表。

将线性表存储到集合中：函数名称为SaveEdit(Lists,L)，其基本操作为将原本在初始面板中的线性表存储到原线性表集合中，对于原线性表集合完成修改。

## 2.3 系统实现

本系统通过命令式菜单来调用各函数功能，系统开始运行时会输出菜单界面，依照菜单中给出的各功能的指令输入命令，以实现相关功能的使用。

系统的具体实现方式为：

首先根据线性表的基本操作实现完成基本函数构建，然后通过对于多种情况的考虑完善线性表的各个子函数。

在主函数内部完成函数调用所需实参值的准备和函数执行结果的显示。利用op存储当前命令，并且采用switch-case语句，依据输入的op进行选择，调用函数实现相应功能。

每个子函数的实现方法如下：

⑴初始化表InitList(L)：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。首先利用malloc函数在存储中寻找到一个链表节点的空间大小，并将该节点的后继置为空指针，之后返回OK。当L已存在时，便不可再次初始化，此时需返回INFEASIBLE

⑵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。先声明一个暂时指针，使得该指针指向头指针，当指针的后继存在的时候，对于指针后继进行历遍，并且依次释放链表节点的空间，如果线性表不存在，那么应该先初始化线性表。

⑶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，依次释放后续节点的空间。

⑷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。判断线性表L.length的大小，如果为0则返回TRUE,否则返回FALSE。

⑸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。当线性表不存在时，函数返回INFEASIBLE；否则返回线性表长度即可。

⑹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。当线性表不存在时，函数返回INFEASIBLE；当线性表存在，但是i<1或者i>线性表表长时，选取的位置非法，故返回ERROR；当线性表存在且i>= 1, i<= 线性表表长时(即选取位置合法时)，将线性表相应数据赋值到e，返回OK即可。

⑺查找元素：函数名称是LocateElem(L,e)；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。当线性表不存在时，函数返回INFEASIBLE；当线性表存在时，扫描线性表中所有元素，并在扫描过程中将每一个已在线性表中的元素与e比较，如果相等则返回当前元素在数组中的序号+1，如果扫描过程中没有任何一个元素与e相等，查找失败，则返回ERROR。该程序实现的相应流程图如图2-1所示



图2-1查找元素的相应流程图

⑻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。当线性表不存在时，函数返回INFEASIBLE; 当线性表存在时，扫描线性表，且扫描过程中比较当前位置元素L.elem[i]是否与cur\_e相等，如果相同，则继续判断i是否为0，如果为0，则返回ERROR，表明此元素无前驱；如果i不为0，则将L.elem[i – 1]的值赋给pre\_e，并返回OK；此外，若线性表扫描过程中不存在i使得L.elem[i]等于cur\_e，则函数返回ERROR，表明不存在此元素。该程序实现的相应流程图如图1-2所示



图2-2获得前驱的相应流程图

⑼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。当线性表不存在时，函数返回INFEASIBLE; 当线性表存在时，扫描线性表，且扫描过程中比较当前位置元素L.elem[i]是否与cur\_e相等，如果相同，则继续判断i是否为L.length-1，如果为L.length-1，表明这已经是线性表最后一个元素，无后继，函数返回ERROR；如果i不为L.length-1，则将L.elem[i+1]的值赋给next\_e，并返回OK；此外，若线性表扫描过程中不存在i使得L.elem[i]等于cur\_e，则函数返回ERROR，表明不存在此元素，当然在线性表中也就不存在它的后继。该程序流程图与获得前驱流程图相近。

⑽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≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。当线性表不存在时，函数返回INFEASIBLE；当线性表存在但是i<1或者i> L.length+1时，插入位置非法，函数返回ERROR；当线性表存在且i >= 1, i <= L.length + 1时，首先判断L.length + 1是否大于L.listsize，如果是，则应利用realloc函数对该线性表的数组重新分配大小为LIST\_INIT\_SIZE\*sizeof(ElemType)的存储空间，之后将原数组中地址位于[L.elem+i-1,L.elem+L.length–1]中的元素全部向地址增大的方向移动一位，之后将e的值赋给L.elem[i–1]，函数返回OK即可；如果L.length+1<=L.listsize，只需要将原数组中地址位于[L.elem+i,Lelem+L.length–1]中的元素全部向地址增大的方向移动一位，之后将e的值赋给L.elem[i–1]，函数返回OK即可。该程序实现的流程图如图2-3所示



图2-3插入元素的相应流程图

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。当线性表不存在时，函数返回INFEASIBLE；当线性表存在但是i < 1或者i > L.length时，删除位置非法，函数返回ERROR；当线性表存在且i >= 1, i <= L.length时，将数组中地址位于[ L.elem + i, L.elem + L.length – 1 ]的所有元素向地址减小的方向移动一位，后将原线性表长度减1即可。该程序实现的流程图与插入元素的流程图相近

⑿遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。当线性表不存在时，函数返回INFEASIBLE；当线性表存在且长度为0时，输出“空”，函数返回OK；当线性表存在且长度不为0时，利用循环结构将线性表扫描一遍，并输出线性表中每一个元素的值，函数返回OK。

线性表文件存取的子函数：

(13)线性表数据存入文件SaveList(L, SaveFile):利用文件指针以”w+”的形式打开(文件不存在时自动创建)文件名为SaveFile的文件，如果此步打开文件失败则函数返回ERROR，否则就将线性表L扫描一遍，并将每一个元素输入到目标文件中，且每输入一个元素后就立即输入一个空格，否则会导致存入文件中的数据连在一起难以取出。输入完毕后利用fclose函数关闭该目标文件。最后返回OK；存入文件的相应流程图如图2-4所示



图2-4存入文件的相应流程图

(14)读取文件中数据存入线性表LoadList(L, LoadFile)：利用文件指针以”r”的形式打开文件名为LoadFile的文件，如果此步打开文件失败则函数返回ERROR，否则就将文件中每一个数据利用fscanf函数读入线性表，读取完毕后利用fclose函数关闭该目标文件。最后返回OK；

线性表集合操作：

(15)向集合中新增线性表AddList(Lists, ListName)：首先判断，Lists当前长度是否已经是最大容量，如果是，则返回OVERFLOW，如果不是，则先利用strcpy函数将ListName拷贝至Lists.elem[Lists.length].name，再利用已定义的函数InitList(L)和InitInput(L)初始化Lists.elem[Lists.length].L，完成后将线性表集合的长度加一，再返回OK即可。

(16) 删除集合中名为ListName的线性表RemoveList(Lists, ListName):首先，扫描当前线性表集合，并在扫描过程中利用strcmp函数判断每一个线性表的名称是否与ListName相同，若相同则将之后的每一个线性表向前移动一位后，将集合长度减一，返回OK即可；如果扫描结束后仍未找到某个名为ListName的线性表，返回0即可。

(17)定位集合中的线性表，调用LocateList(Lists, ListName)函数，判断集合中是否有该名称的线性表，没有则输出没有，如果有就输出该线性表在线性表集合中的位置。

(18)遍历集合ListsTraverse(Lists)：首先判断线性表集合是否存在，不存在则返回INFEASIBLE，若存在再判断长度是否为0，若为0则返回0，否则扫描线性表集合，并输出每一个线性表的名称和其中的数据，返回OK。

(19) 将集合中的线性表载入：函数名称为EditList(Lists,L)，其基本操作是将线性表集合中的指定线性表载入到初始面板的线性表中，方便对于该线性表进行各种基础操作。输入线性表名称，声明变量k对于线性表集合进行调用k=LocateList(Lists, ListName)，对于k进行判断线性表集合中是否有该线性表，如果有，则将初始面板中的线性表改为集合中指定的线性表。

(20) 定位线性表LocateList(Lists, ListName)：扫描当前线性表集合，过程中利用strcmp函数判断当前线性表名称是否与ListName相同，若有相同则返回当前线性表的逻辑位置;若扫描结束仍未发现名为ListName的线性表，则返回0，表明查无此表。

(21) 将线性表存储到集合中：函数名称为SaveEdit(Lists,L)，其基本操作为将原本在初始面板中的线性表存储到原线性表集合中，对于原线性表集合完成修改。输入线性表名称，声明变量k对于线性表集合进行调用k=LocateList(Lists, ListName)，对于k进行判断线性表集合中是否有该线性表，如果有，则将线性表集合中的指定线性表修改为初始面板中的线性表。

## 2.4 系统测试

根据本次实验要求，系统测试包括如下内容：

（1）测试准备

（2）测试计划

（3）测试结果

（4）测试分析

1.测试准备：

在本次测试中，主要测试线性表集合操作的准确性以及完备性，对于集合中的线性表要进行初始化，输入，等多种函数的实现，并且，集合中的线性表还能够进行存储，并且能够读取到集合中的任意线性表中。

2.测试计划：

在本次测试中，由于线性表物理结构的简单性，所以直接从键盘中输入线性表的内容，通过测试不同的线性表，在该程序中，为实现简洁性，所以只用到了3个线性表，但是在实际运用中可以达到所需要的要求。

3.测试结果：

此次系统测试不包含线性表的载入与保存的输入选择，对于线性表集合中的进行基本操作时，应该进行载入，在修改完成后，应该保存到线性表集合中。

|  |  |  |  |
| --- | --- | --- | --- |
| 线性表集合主要功能测试 | 参与的线性表序号 | 输入的参数 | 预计的输出结果 |
| 新增一个名称为1，内容为123的线性表 | 1 | 15 1 1 2 3 0 | 线性表"1"存储成功！ |
| 新增一个名称为2，内容为456的线性表 | 2 | 15 2 4 5 6 0 | 线性表"2"存储成功！ |
| 展示线性表2 | 2 | 20 2 | 当前线性表中数据如下:  4 5 6 |
| 对于线性表2，向其中第二个位置插入数据2 | 2 | 10 2 2 | 插入成功！  当前线性表中数据如下:  4 2 5 6 |
| 遍历线性表集合 | 1、2 | 18 | 当前线性表集合中共包含以下2个线性表：  1  1 2 3  2  4 2 5 6 |
| 存储线性表2到指定文件中 | 2 | 13 D:\\a | 保存到文件成功 |
| 将文件中的数据读取到线性表1中 | 1 | 2 14 D:\\a | 从文件读取成功 |
| 再次遍历线性表集合 | 1、2 | 18 | 当前线性表集合中共包含以下2个线性表：  1  4 2 5 6  2  4 2 5 6 |
| 删除线性表2 | 2 | 20 | 删除成功！ |
| 再次遍历线性表集合 | 1 | 18 | 当前线性表集合中共包含以下1个线性表：  1  4 2 5 6 |

1. 新增一个名称为1，内容为123的线性表，程序运行结果如图2-5所示

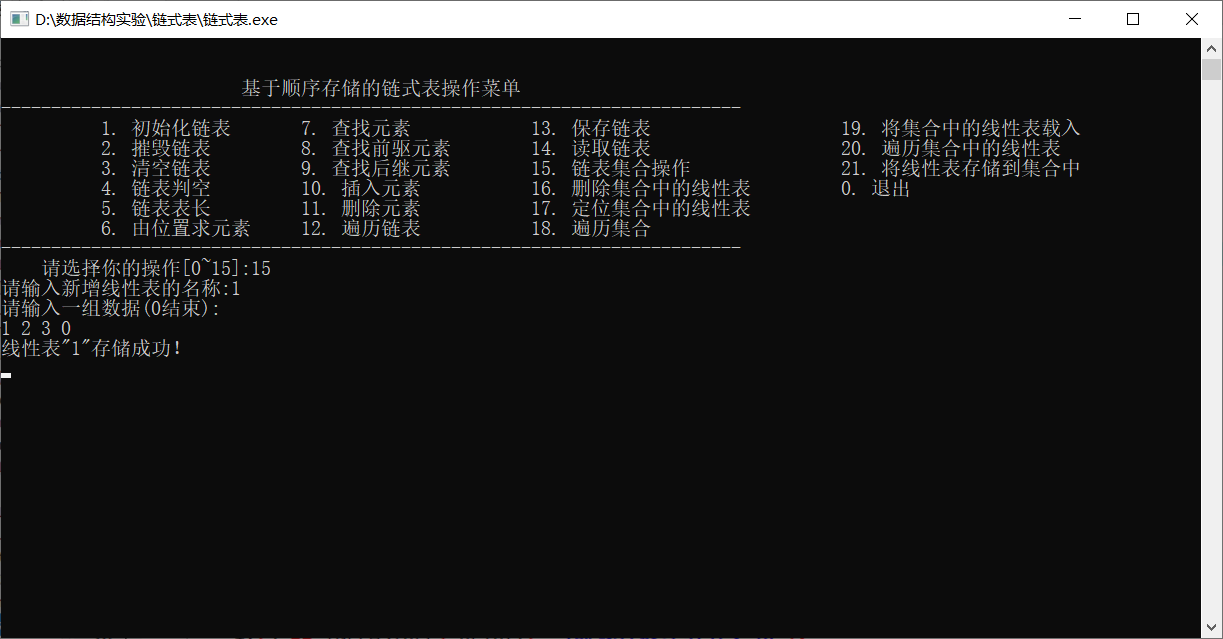


图2-5新增线性表的运行结果

1. 新增一个名称为2，内容为456的线性表，程序运行结果如图2-6所示

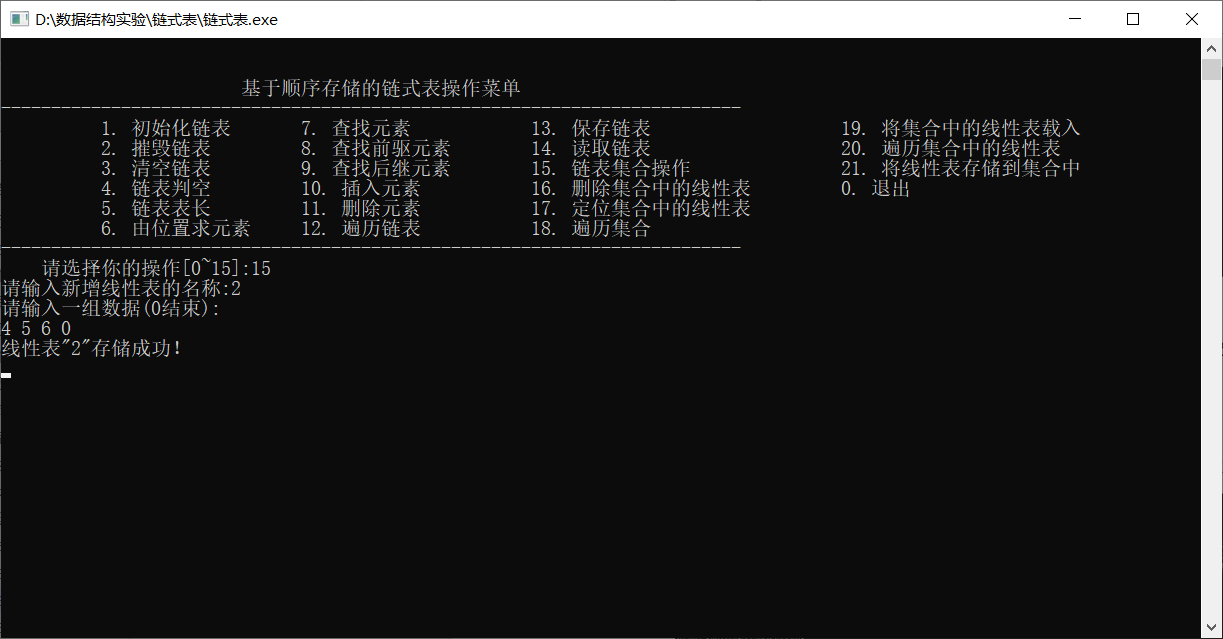


图2-6新增线性表运行结果

1. 展示线性表2，程序运行结果如图2-7所示

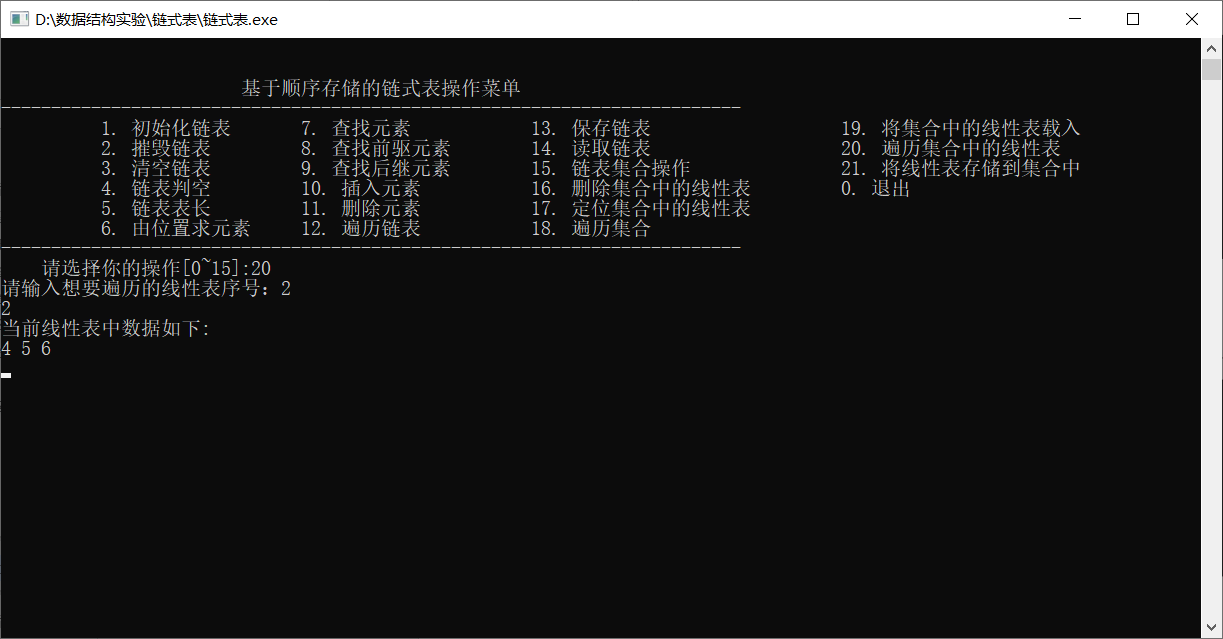


图2-7展示线性表的运行结果

1. 对于线性表2，向其中第二个位置插入数据2，程序运行结果如图2-8所示

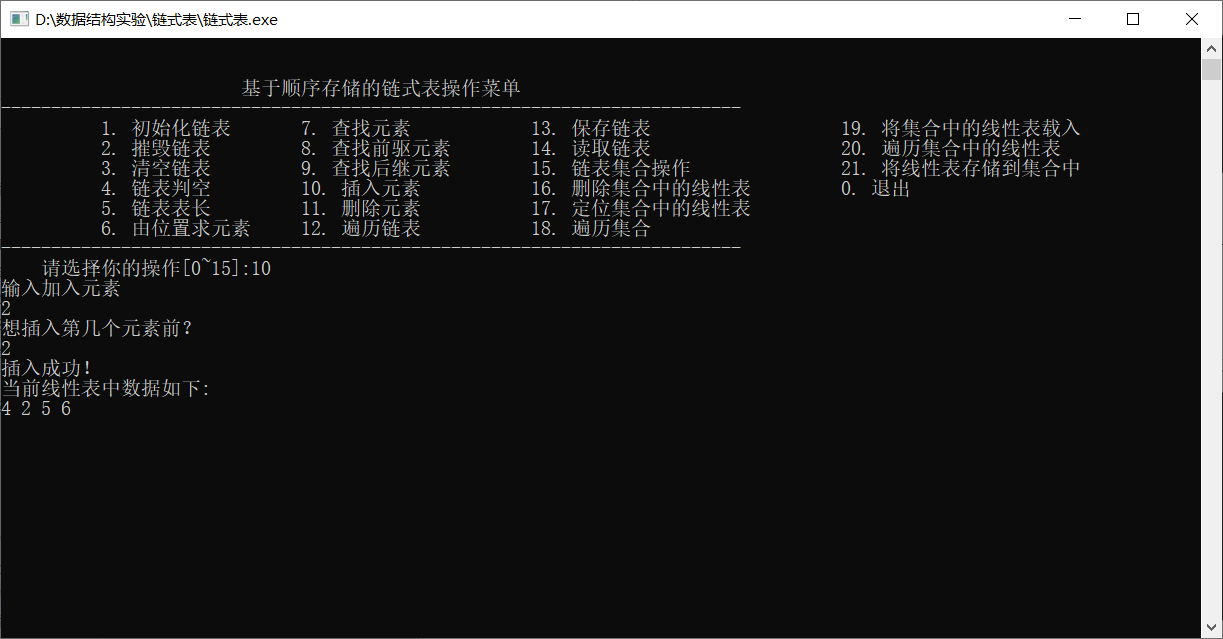


图2-8线性表元素插入的运行结果

1. 遍历线性表集合，运行结果如图2-9所示

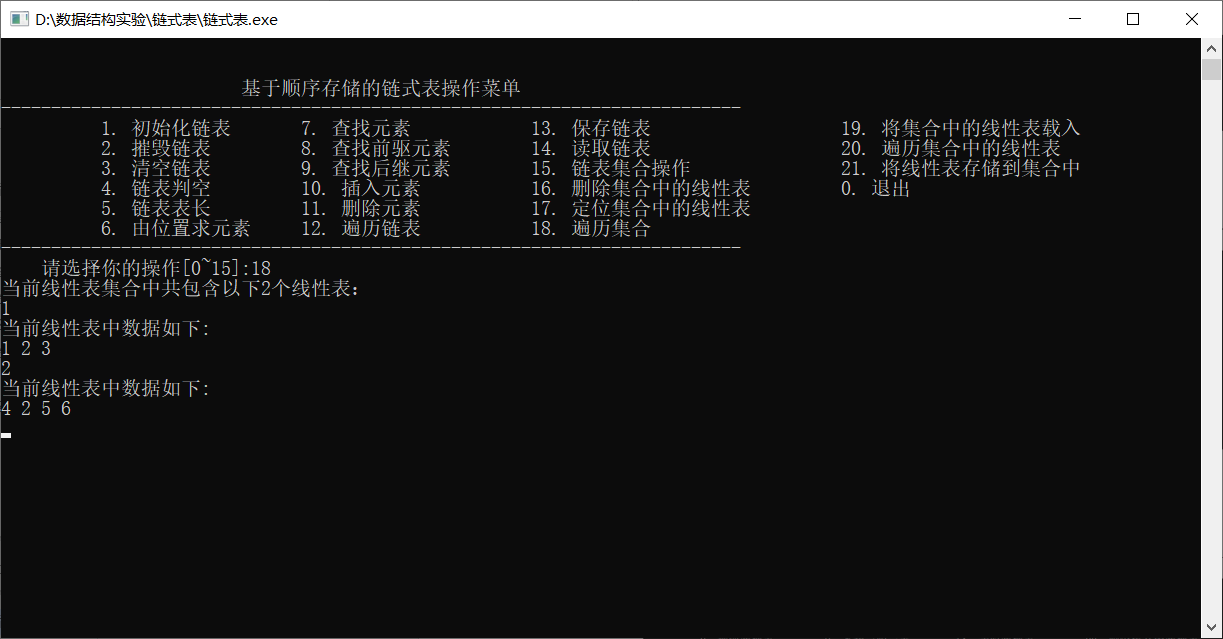


图2-9遍历线性表集合的运行结果

1. 存储线性表2到指定文件中，运行结果如图 2-10所示

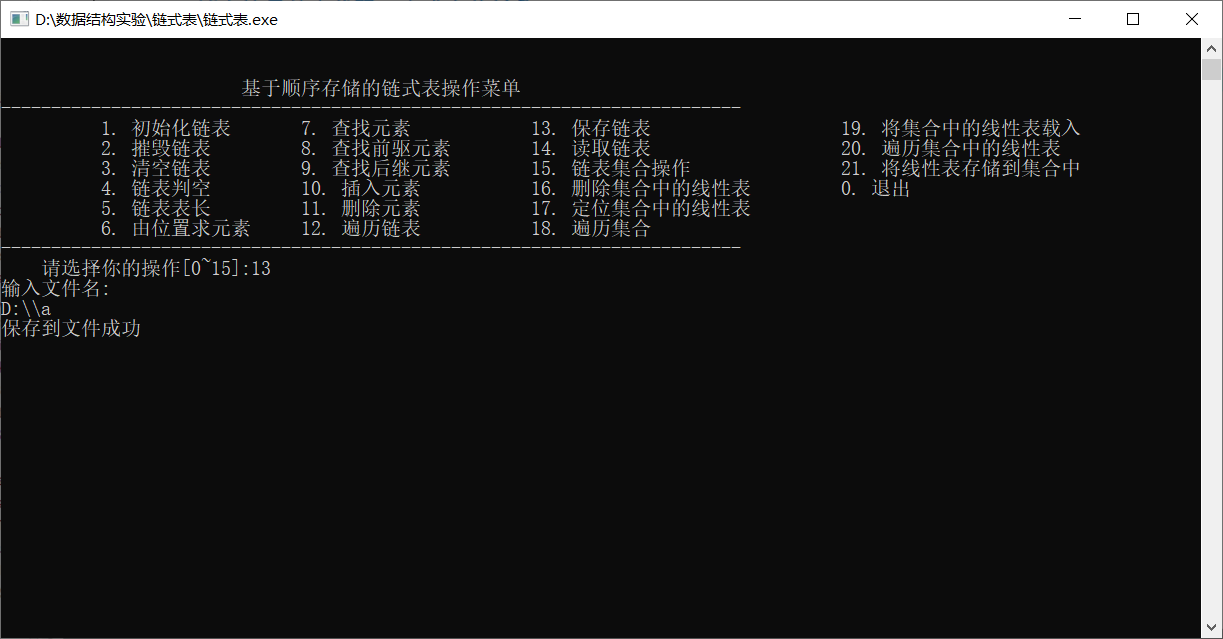


图2-10存储线性表的运行结果

1. 将文件中的数据读取到线性表1中，运行结果如图2-11所示

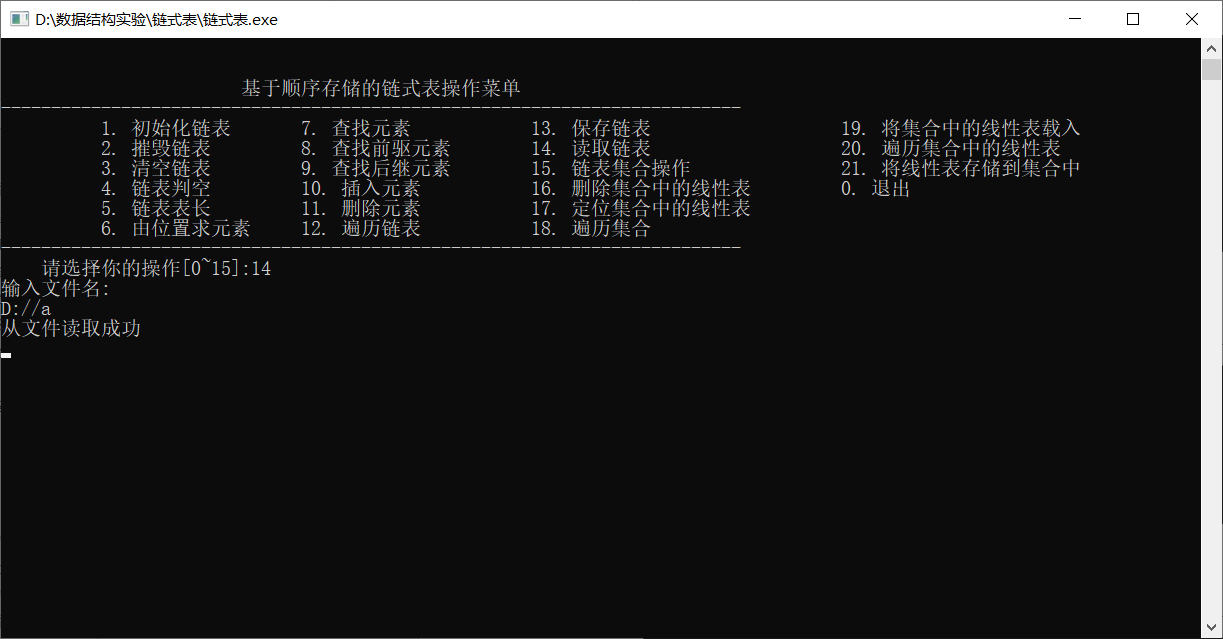


图2-11文件读取到线性表的运行结果

1. 再次遍历线性表集合，运行结果如图2-12所示

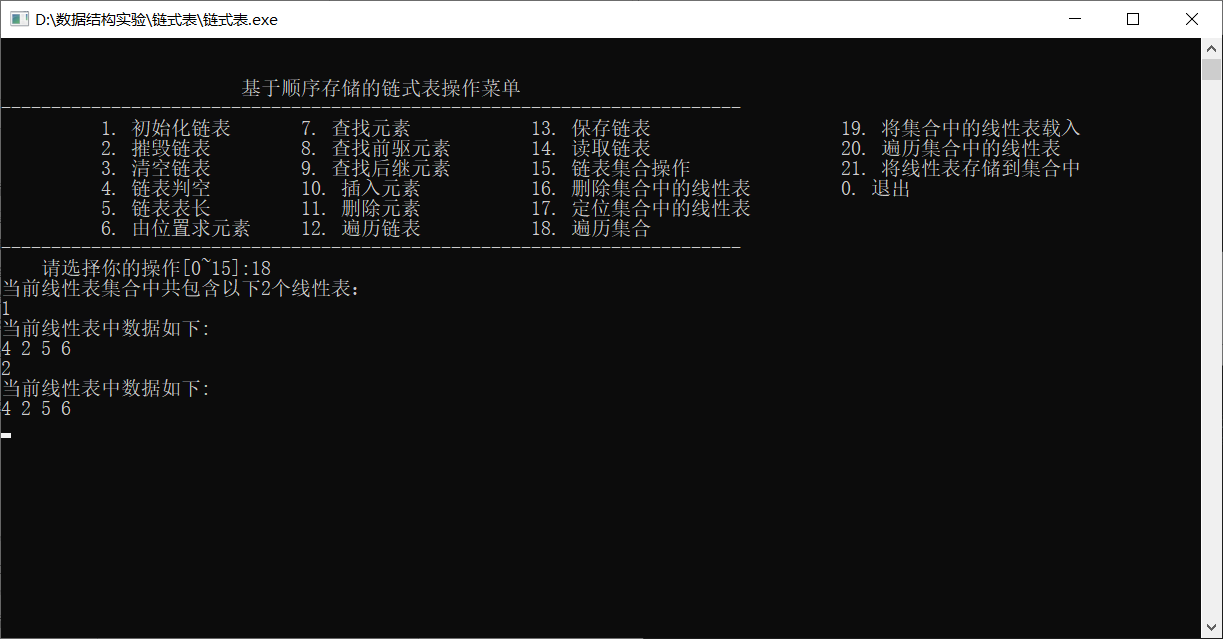


图1-12再次遍历线性表集合的运行结果

1. 删除线性表2，运行结果如图2-13所示

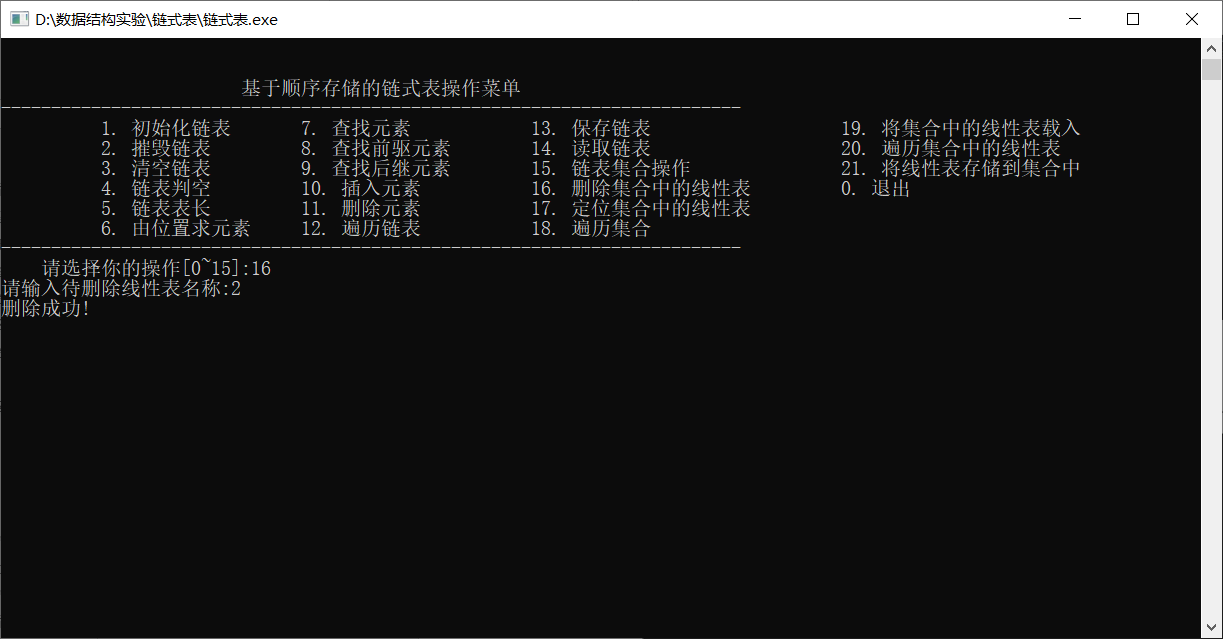


图2-13删除线性表2的运行结果

1. 再次遍历线性表集合，运行结果如图2-14所示

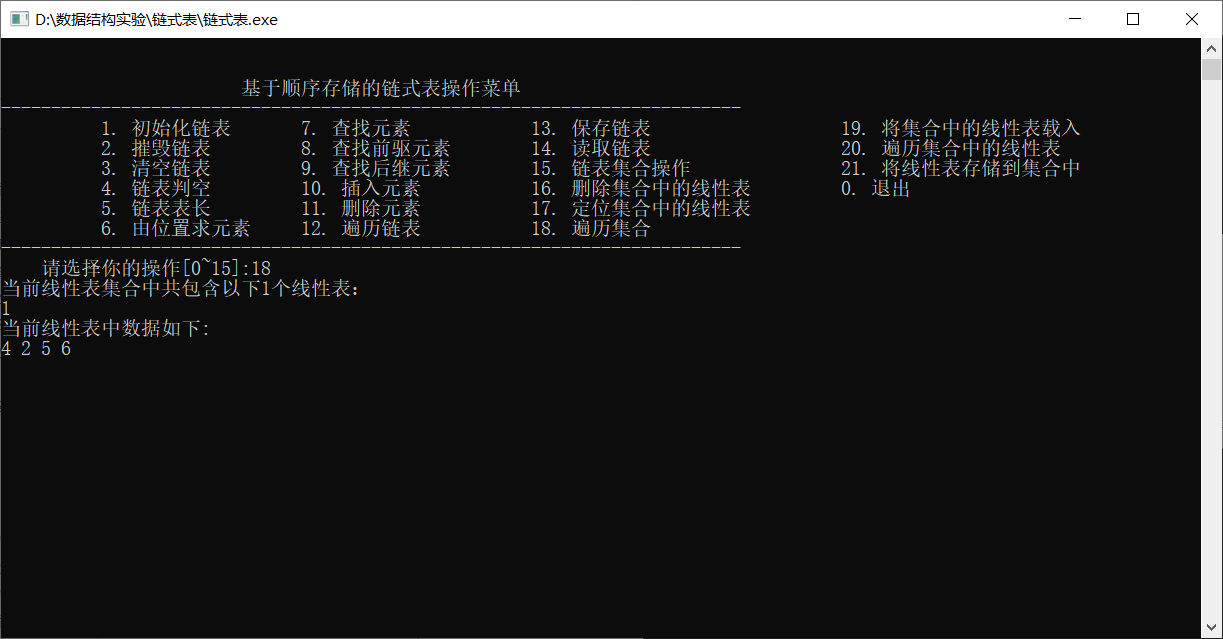


图2-14再次遍历线性表集合的运行结果

4.测试分析：

本次实验测试的结果都较为符合预期，但是，由于一开始程序设计时的思路不对，导致线性表集合的中的线性表需要手动的进行保存，如果没有保存会导致错误，以及在集合中的线性表实际上属于二级菜单，而线性表中的基本操作是在一级菜单中，所以会有一定的逻辑复杂性。

## 2.5 实验小结

本次实验的大致模板仍然与上一次实验相同，由于第一次实验和第二次饰演的系统同时检查，因此，本次实验仍然有很多要修改的地方，由于链表的操作不熟悉以及链表本身所具有的一定难度，本次实验的完成并不顺利，在进行检查时，对于代码进行多次修改，但是，由于之前的思路不正确导致的原系统无法使用，在当前的系统操作中，存在有一定的逻辑复杂性。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

二叉树是树形结构的一个重要类型。许多实际问题抽象出来的数据结构往往是二叉树形式，即使是一般的树也能简单地转换为二叉树，而且二叉树的存储结构及其算法都较为简单，因此二叉树显得特别重要。二叉树特点是每个结点最多只能有两棵子树，且有左右之分 。

二叉树是n个有限元素的集合，该集合或者为空、或者由一个称为根的元素及两个不相交的、被分别称为左子树和右子树的二叉树组成，是有序树。当集合为空时，称该二叉树为空二叉树。在二叉树中，一个元素也称作一个结点 。

问题1：二叉树的实现

根据本次实验要求，通过以二叉树结构的最小完备性和常用性相结合的基本原则，以二叉树的物理结构为基础，通过函数形式定义二叉树的初始化，输入，销毁，清空，判空，定位节点，节点赋值，查找兄弟节点，插入节点，删除节点，先序遍历，中序遍历，后序遍历等12中基础运算。

问题二：构造具有菜单的功能演示系统

根据本次实验要求，应该构造能够良好体现二叉树功能的演示程序，至少应当显示提供二叉树操作的主菜单，其中，应该在主函数中完成函数调用所需的实参值的准备和实现函数执行结果的显示，并提供适当的操作提示信息，演示系统应当实现输入验证和异常响应，对于用户的非法输入给与提示，并能够允许重新输入，对于程序的运行错误也能实现异常信息的输入，演示系统应当有合理的层次

问题三：实现二叉树的文件形式的保存

根据本次实验要求，实验应当实现二叉树的文件形式的保存，包括

（1）设计文件的记录格式，使得文件的保存与读取均能实现高效率。

（2）设计二叉树的保存，使得保存后重读取的二叉树能够实现与原二叉树相同的格式。

## 3.2 系统设计

根据本次实验要求，系统设计包括以下的内容：

1. 设计结构类型，实现二叉树的物理结构。
2. 设计具有菜单的演示系统的框架。
3. 设计森林格式，使得可以进行多个二叉树管理
4. 设计文件格式，使得保存后重读取的文件格式保持一致性。

设计一：设计结构类型，实现二叉树的物理结构

二叉树的物理结构如下：

typedef struct {

KeyType key;

char others[20];

} TElemType; //二叉树结点类型定义

typedef struct BiTNode{ //二叉链表结点的定义

TElemType data;

struct BiTNode \*lchild,\*rchild;

} BiTNode, \*BiTree;

利用两个结构类型，实现二叉树的节点以及链表的物理结构，能够使得对于确定二叉树的访问准确。

设计二：演示系统框架设计

本实验采用多级菜单设计，在第一层的菜单为基础的二叉树修改操作，包括有12种基本操作和二叉树的存档读档，第二级菜单为森林操作，可以实现初始化，种树，删除树，清空树，查找树，修改树，遍历森林，摧毁森林。通过修改树这一操作，完成第二级菜单向第一级菜单的过渡。

设计三：设计森林格式，使得可以进行多个二叉树管理

森林的物理结构如下：

typedef struct { //森林中每棵树的定义

BiTree T;

char name[30];

} Tree, \*pTree;

typedef struct { //森林定义(线性表结构存储)

pTree \*tree;

int length;

int size;

} FOREST, \*FST;

利用两个结构类型，实现对于森林，以及森林中的树的操作。

设计四：使得保存后重读取的文件格式保持一致性

对于文件的保存，采取和输入一样的格式，利用结构：

typedef struct {

int pos;

TElemType data;

} DEF;

typedef struct {

KeyType key;

char others[20];

} TElemType; //二叉树结点类型定义

使得读入文件的数据结构类型与输入的数据结构类型一致

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

⑴创建二叉树：函数名称是CreateBiTree(T,definition)；初始条件是definition 给出二叉树T的定义，如带空子树的二叉树前序遍历序列、或前序+中序、或后序+中序；操作结果是按definition构造二叉树T。

注：①要求T中各结点关键字具有唯一性。后面各操作的实现，也都要满足一棵二叉树中关键字的唯一性，不再赘述；②CreateBiTree中根据definition生成T，不应在CreateBiTree中输入二叉树的定义。

⑵销毁二叉树：函数名称是DestroyBiTree(T)；初始条件是二叉树T已存在；操作结果是销毁二叉树T。

⑶清空二叉树：函数名称是ClearBiTree (T)；初始条件是二叉树T存在；操作结果是将二叉树T清空。

⑷判定空二叉树：函数名称是BiTreeEmpty(T)；初始条件是二叉树T存在；操作结果是若T为空二叉树则返回TRUE，否则返回FALSE。

⑸求二叉树深度：函数名称是BiTreeDepth(T)；初始条件是二叉树T存在；操作结果是返回T的深度。

⑹查找结点：函数名称是LocateNode(T,e)；初始条件是二叉树T已存在，e是和T中结点关键字类型相同的给定值；操作结果是返回查找到的结点指针，如无关键字为e的结点，返回NULL。

⑺结点赋值：函数名称是Assign(T,e,value)；初始条件是二叉树T已存在，e是和T中结点关键字类型相同的给定值；操作结果是关键字为e的结点赋值为value。

⑻获得兄弟结点：函数名称是GetSibling(T,e)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值；操作结果是返回关键字为e的结点的（左或右）兄弟结点指针。若关键字为e的结点无兄弟，则返回NULL。

⑼插入结点：函数名称是InsertNode(T,e,LR,c)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值，LR为0或1，c是待插入结点；操作结果是根据LR为0或者1，插入结点c到T中，作为关键字为e的结点的左或右孩子结点，结点e的原有左子树或右子树则为结点c的右子树。

特殊情况，c插入作为根结点？可以考虑LR为-1时，作为根结点插入，原根结点作为c的右子树。

⑽删除结点：函数名称是DeleteNode(T,e)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值。操作结果是删除T中关键字为e的结点；同时，如果关键字为e的结点度为0，删除即可；如关键字为e的结点度为1，用关键字为e的结点孩子代替被删除的e位置；如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为e的左子树中最右结点的右子树。

⑾前序遍历：函数名称是PreOrderTraverse(T,Visit())；初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）；操作结果：先序遍历，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⑿中序遍历：函数名称是InOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）；操作结果是中序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒀后序遍历：函数名称是PostOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）；操作结果是后序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒁按层遍历：函数名称是LevelOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是层序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

以下为附增函数：

存储二叉树：函数名称是SaveBiTree(T,fp,i)：初始条件是二叉树存在且文件能够打开，i表示对于写入二叉树中节点所在的位置，操作结果是将T中的数据按照满二叉树的次序写入到文件中，操作结束后，返回OK。

读取二叉树：函数名称是LoadBiTree(T,FileName)：初始条件是二叉树存在，打开名为FileName的文件，将文件中的数据依次读取到二叉树中，操作结束后，返回OK。

以下为森林操作：

森林集成操作，函数名称是gather\_operate(fst)：函数的作用结果是进入二级森林操作系统。

初始化森林：函数名称是InitForest(fst)：函数的作用结果是对于森林fst分配空间，并将森林的长度改为0，成功返回OK

向森林中种树：函数名称是AddBiTree(fst, pos, BiTreeName)：函数的作用结果是向森林中的pos位种入名为BiTreeName的树，成功返回OK。

删除森林中的树：函数名称是RemoveBiTree(fst, BiTreeName)：函数的作用结果是删除森林中名为BiTreeName的树，成功返回OK。

查找森林中的树：函数名称是LocateBiTree(fst, BiTreeName)：函数的作用结果是定位在森林中名为BiTreeName的树，成功返回该树在森林中的位置。

遍历森林中的树：函数名称是ForestTraverse(fst)：函数的作用结果是遍历森林，其中会显示遍历的方式p，按照指定的方式遍历完成森林。

## 3.3 系统实现

本系统通过命令式菜单来调用各函数功能，系统开始运行时会输出菜单界面，依照菜单中给出的各功能的指令输入命令，以实现相关功能的使用。

系统的具体实现方式为：

首先根据二叉树的基本操作实现完成基本函数构建，然后通过对于多种情况的考虑完善各个子函数。

在主函数内部完成函数调用所需实参值的准备和函数执行结果的显示。利用op存储当前命令，并且采用switch-case语句，依据输入的op进行选择，调用函数实现相应功能。

每个子函数的实现方法如下：

1.初始化二叉树：InitBiTree(BiTree &T)，对于二叉树T，只是声明了这一指针，还没有分配空间，所以，在该函数中完成对于空间的分配，T的左右子树置为空指针。成功返回OK。

2.创建二叉树：CreateBiTree(BiTree &T,DEF definition[])，首先在main函数中，提示按照满二叉树形式输入，对于输入的数据按照序位号，键值，内容依次存入到definition中，传入definition的地址，使得能够对其进行访问，采用静态变量声明一个指针数组p，对于序位号存在的二叉树，向p中的该位置分配空间，并完成内容的输入，当序位号不是1，即不是空节点时，按照满二叉树的性质，满二叉树的双亲的序号为其孩子序号的一半，依次将对应位置的二叉树左右孩子进行指向。最后，将T改为p[1]的地址并且返回OK。该函数对应的流程图如图3-1所示：



图3-1创建二叉树函数的流程图

3.清空二叉树：ClearBiTree(BiTree &T)，本处采用递归的形式，当递归到的二叉树节点为空的时候进行返回，否则，依次递归二叉树的左右子树，当该节点的左右子树的节点都为空时，释放二叉树的空间，并将其改为空指针，最后返回。

4.求二叉树的深度ClearBiTree(BiTree &T)，此处也采用递归的形式，对于不为空的节点，持续递归，只有到空节点时才返回0，这样依次左右节点的深度加一，再比较左右的深度，返回值为最大的那个，即为二叉树的深度。

5.查找结点LocateNode(BiTree T,KeyType e)，此处采用递归形式，对于键值e，向二叉树中进行递归，利用中间节点p（该节点一开始置为空），当p不为空值时，表明已经找到了节点，直接返回，当T为空时，表明已经到了树的底部也直接返回，当e的值与对应的节点的键值一致时，将该节点的指针赋给p，并返回，持续左右递归，这样，返回的p要么是空指针，即没有找到节点，要么不为空，找到了节点。该函数对应的流程图如图3-2所示：



图3-2查找结点函数的流程图

6.实现节点赋值Assign(BiTree &T,KeyType e,TElemType value)，此处调用了LocateNode函数，对于要赋值的节点进行定位，利用指针q存储定位到的节点的指针，利用指针p来存储修改的键值，当p=q时，说明该二叉树中已存在相应键值的节点，当q不为空指针，且q为空指针时，对于相应的键值和内容进行修改。完成后返回OK。

7.二叉树判空BiTreeEmpty(BiTree T)当二叉树的左右子树都不为空指针时返回TRUE，否则返回ERROR

8.获得兄弟节点GetSibling(BiTree T,KeyType e)此处采用递归，利用中间指针rec，当该节点的指针为空指针时，返回rec，当该节点的左子树不为空且左子树的键值与查找的键值相同的时候，将该节点的右指针赋给rec并返回，如果该节点的右子树与查找的键值相同的时候，将该节点的左指针赋给rec并返回，递归进行，这样返回的指针rec要么为空，即没有找到相应的兄弟节点，要么持续返回了相应的兄弟节点。该函数对应的流程图如图3-3所示：



图3-3获得兄弟节点函数的流程图

9.插入节点InsertNode(BiTree &T,KeyType e,int LR,TElemType c)此处调用了定位节点的函数LocateNode，用aim来存储查找到的节点的指针，并利用repeat来判断节点的键值是否重复，当定位的节点不为空且键值不重复是，建立新节点NewNode，将相应的数据写入到新结点中，根据LR的值进行判断， LR为0时插入为相应的键值节点的左孩子，LR为1的时候插入为相应的键值节点的右孩子，当LR为-1时该节点作为根，为根时新节点的右孩子为原来的根节点，左孩子置为空指针，并将根节点T修改为NewNode。该函数对应的流程图如图3-4所示：



图3-4插入节点函数的流程图

10.定位双亲节点ParentNode(BiTree T, KeyType e, BiTree &p)该函数为实现删除节点的必要函数，函数的参数p用来存放对应键值的双亲节点的指针，当对应的节点的左孩子或者右孩子的键值与相应的键值相同的时候，将p的值修改为相应节点的指针，当相应的键值为根节点时，p即为根节点的指针。当返回OK表示删除结点为根节点，返回LEFT表示e为p的左孩子，返回RIGHT表示e为p的右孩子。

11定位最右节点RightMostNode(BiTree T)返回二叉树最右节点的指针

12.删除节点DeleteNode(BiTree &T,KeyType e)调用了5、10和11中的函数利用p节点来存放相应节点的双亲指针的值，定位到节点，如果该节点没有左子，那么，该节点改变为该该节点的右子树，并释放该节点的空间，如果该节点没有右子树，那么，该节点改变为该节点的左子树，并释放该节点的空间，如果该节点的左右子树都存在，那么根据要求，用根节点的左孩子代替被删除的位置，该节点的右子树作为该节点的左子树中最右结点的右子树。该函数对应的流程图如图3-5所示 ：



图3-5删除节点函数的流程图

13.先序遍历PreOrderTraverse(BiTree T,void (\*visit)(BiTree))该遍历为非递归结构，采用栈来实现，声明一个指针数组，来存放指向二叉树节点指针，再声明一个中间指针p来进行遍历，利用top，表明置空栈，当二叉树的节点不为空时，进入循环，向指针数组中写入该节点的指针，top++，当栈非空时，即top！=0时，清空栈并访问该节点，按照先序遍历的要求，需要将该节点的右子树存入到栈中，top++，随后存入左子树，top++，然后继续循环，直到左子树完成遍历后再开始遍历右子树，该函数对应的流程图如图3-6所示：  


图3-6前序遍历函数的流程图

14.中序遍历InOrderTraverse(BiTree T,void (\*visit)(BiTree))该遍历采用了递归遍历，利用InOrderTraverse(T->lchild,visit);先遍历访问该二叉树的左子树，之后访问根节点，最后再访问右子树。

15.后续遍历PostOrderTraverse(BiTree T,void (\*visit)(BiTree))该遍历采用递归，利用PostOrderTraverse(T->lchild, visit);先遍历访问二叉树的左子树，再遍历访问二叉树的右子树，最后访问根节点。

16.层序遍历LevelOrderTraverse(BiTree T,void (\*visit)(BiTree))，该遍历利用一个指针数组trees[200]来存放二叉节点的指针，先对指针数组置空，利用变量l，r来控制，其中l为该层遍历的节点的个数，r表示在该层总共拥有的节点首先将数组中的0位置为根节点，利用l，r来控制循环，先访问其中作为双亲的节点，当该节点有左子树时，向数组中对应位置存储，在下一层遍历中，先进行访问，当有右子树时，向后面存储，后一步访问，该函数对应的流程图如图3-7所示：



图3-7层序遍历函数的流程图

17.存储二叉树SaveBiTree(BiTree T,FILE \*fp,int i)，该函数使用递归的方式，文件的打开与关闭都在main函数中完成，对于不为空的节点，写入该节点所对应的满二叉树中的序号，键值，内容，利用先序遍历的方式进行读入，如果遇到空指针，直接返回。

18.读取二叉树LoadBiTree(BiTree &T, char FileName[])打开文件后，向DEF结构定义的difinition数组中依次写入，该数组写入后，可以直接调用CreateBiTree(T,definition);函数创建二叉树，关闭文件，返回OK。

以下为森林操作：

本次实验采用多级菜单的形式，在进入到森林操作时，自动将第一级菜单中的二叉树替换为森林中的树，并利用flag来判断是否为森林中的树flag=1，是表示为森林中的树，在main函数中输入0表示将第一级菜单中的数据写入到森林菜单中，在森林菜单中，通过修改函数建立起两级菜单之间的联系。森林的各个操作函数如下：

1.初始化森林操作InitForest(FST& fst)，向森林的指针分配相应的空间，向森林中的树分配相应的空间，森林的长度置为0，森林的大小即为森林中最多容纳的树的多少。

2.向森林中加入树AddBiTree(const FST fst, int i, char BiTreeName[])，向森林的指定位置加入相应名字的树，由于为指针数组，所以还要申请空间为对应树的大小，调用InitBiTree(fst->tree[i - 1]->T);完成对于森林中的树的初始化，森林的长度+1，返回OK。

3.删除森林中的树RemoveBiTree(const FST fst, char BiTreeName[])，查找对应名字的树，调用ClearBiTree(fst->tree[i]->T);清空森林中的树，并释放掉指向该树的对应指针数组中的指针的空间，对于森林中的后续树，依次前移，并将树的长度-1.返回OK。

4.定位森林中的树LocateBiTree(const FST fst, char BiTreeName[])遍历森林中的树的名字，并于对应的名字进行比对，发现相同时，返回该树的位置，没有发现时，返回ERROR。

5.遍历森林ForestTraverse(const FST fst)利用指针p来暂时存放相应的遍历，向用户提出选择，从先序遍历，中序遍历，后序遍历，按层遍历中进行选择，并对于相应的选择，使得p为该种遍历方式的指针，调用四种遍历的函数，完成遍历。

6.摧毁森林DestroyForest(FST &fst)对于森林中的树，依次释放掉树的空间，并将森林指针指向空指针。

## 3.4 系统测试

根据本次实验要求，系统测试包括如下内容：

（1）测试准备

（2）测试计划

（3）测试结果

（4）测试分析

1.测试准备：

在本次测试中，主要测试二叉树森林集合操作的准确性以及完备性，对于森林中的二叉树要进行初始化，输入，等多种函数的实现，并且，森林中的二叉树还能够进行存储，并且能够读取到集合中的任意二叉树中。

2.测试计划：

在本次测试中，初始化时调用了文件中的二叉树1 a 2 b 3 c 4 d 5 e 0 0 0，文件中的内容根据在本次实验中的输入格式要求进行了格式上的更改，该系统较于之前的系统进行了菜单层面上的优化，将森林操作载入到二级菜单中，并且能够自动实现保存到森林集合中，此次测试包含有多重二叉树。

3.测试结果：

此次系统测试二叉树的步骤展示在以下表格中：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 二叉树菜单的系统测试 | 输入的参数 | 预期的输出结果 |
| 1 | 初始化二叉树 | 1 | 初始化成功！ |
| 2 | 初始化输入二叉树为1,a 2,b 3,c 4,d 5,e | 2  1 1 a 2 2 b 3 3 c 4 4 d 5 5 e 0 0 0 | 输入成功！ |
| 3 | 先序遍历二叉树 | 11 | 前序遍历当前树中结点信息如下：  1,a 2,b 4,d 5,e 3,c |
| 4 | 查找二叉树中节点键值为5的节点 | 6  5 | 当前结点键值为:5, 数据为:e |
| 5 | 修改键值为5的节点的内容修改后为5 z | 7  5  5  z | 更改成功！ |
| 6 | 按层遍历二叉树 | 14 | 按层遍历当前树中结点信息如下：  1,a 2,b 3,c 4,d 5,z |
| 7 | 插入根节点为7,k | 9  1  -1  7  k | 插入节点成功！ |
| 8 | 插入节点4的左节点8,n | 9  4  0  8  n | 插入节点成功！ |
| 9 | 获得节点5的兄弟节点 | 8  5 | 这个结点的兄弟结点的键值为：4, 数据为：d |
| 10 | 获得节点7的兄弟节点 | 8  7 | 这个结点没有兄弟 |
| 11 | 判断该二叉树的深度 | 5 | 当前树的深度为5 |
| 12 | 中序遍历该二叉树 | 12 | 中序遍历当前树中结点信息如下：  7,k 8,n 4,d 2,b 5,z 1,a 3,c |
| 13 | 将该二叉树存入到文件当中去 | 16  0 | 已将数据存入 D:\数据结构实验\文件.dat |

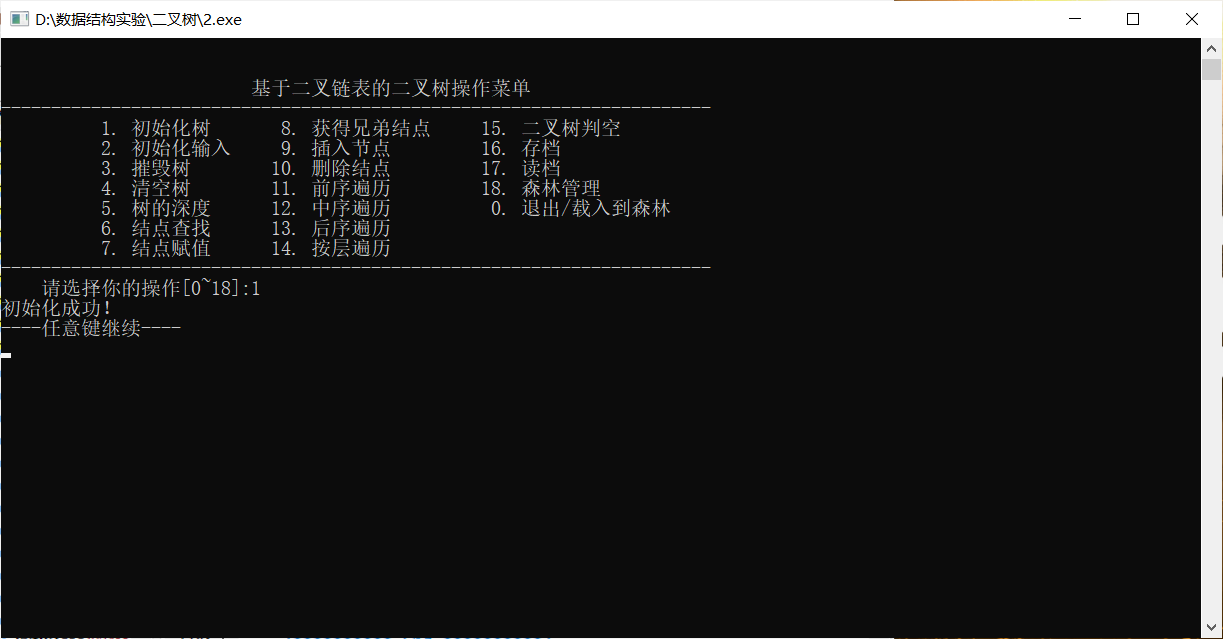


图3-8测试序号一的程序运行结果

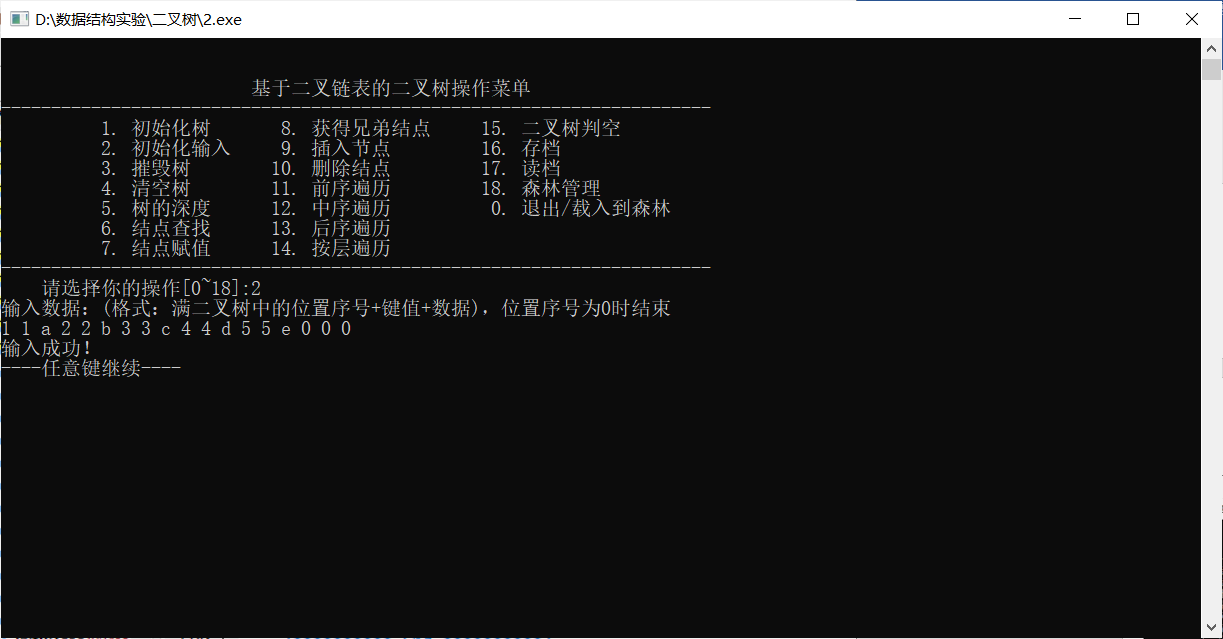


图3-9测试序号二的程序运行结果

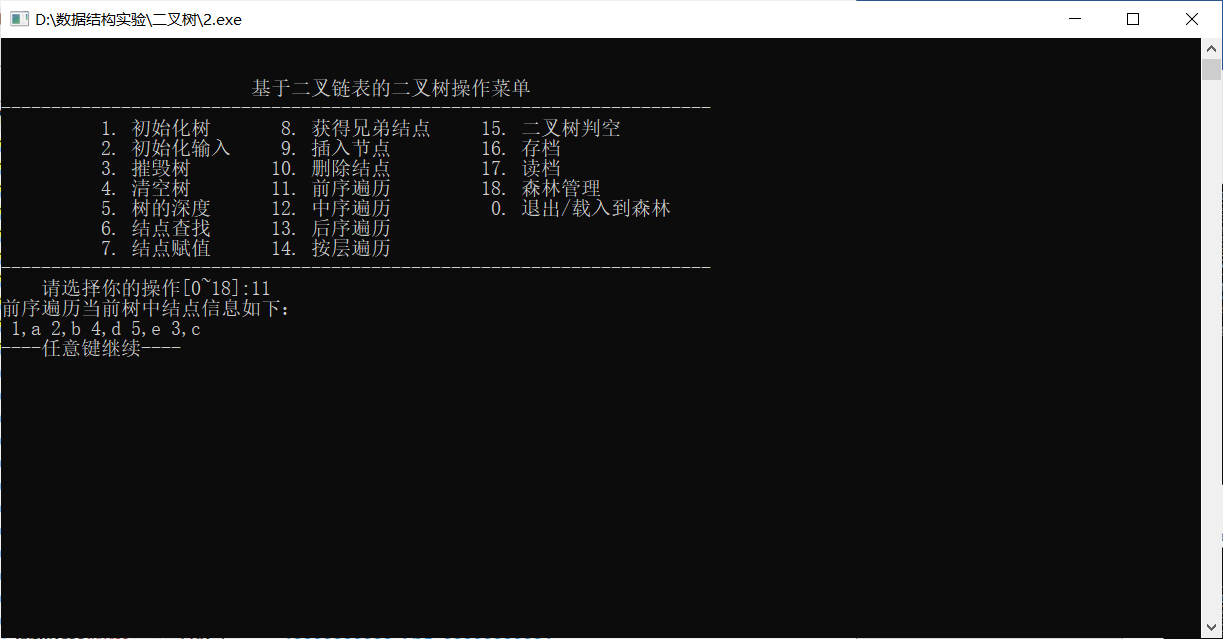


图3-10测试序号三的程序运行结果

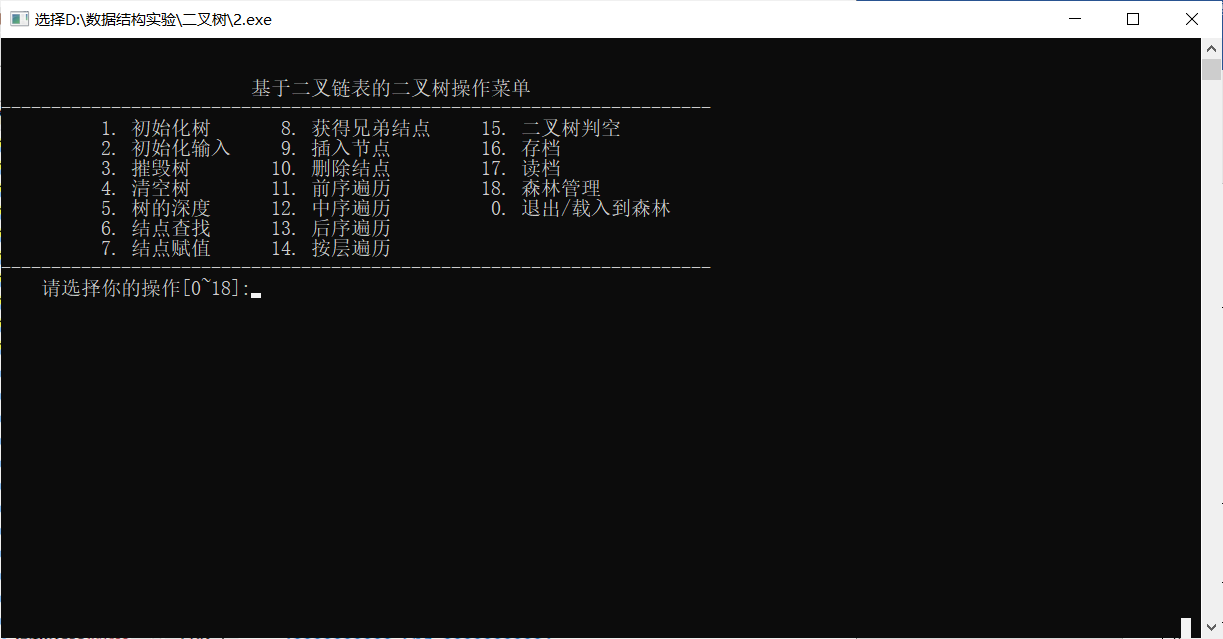


图3-11测试序号四的程序运行结果

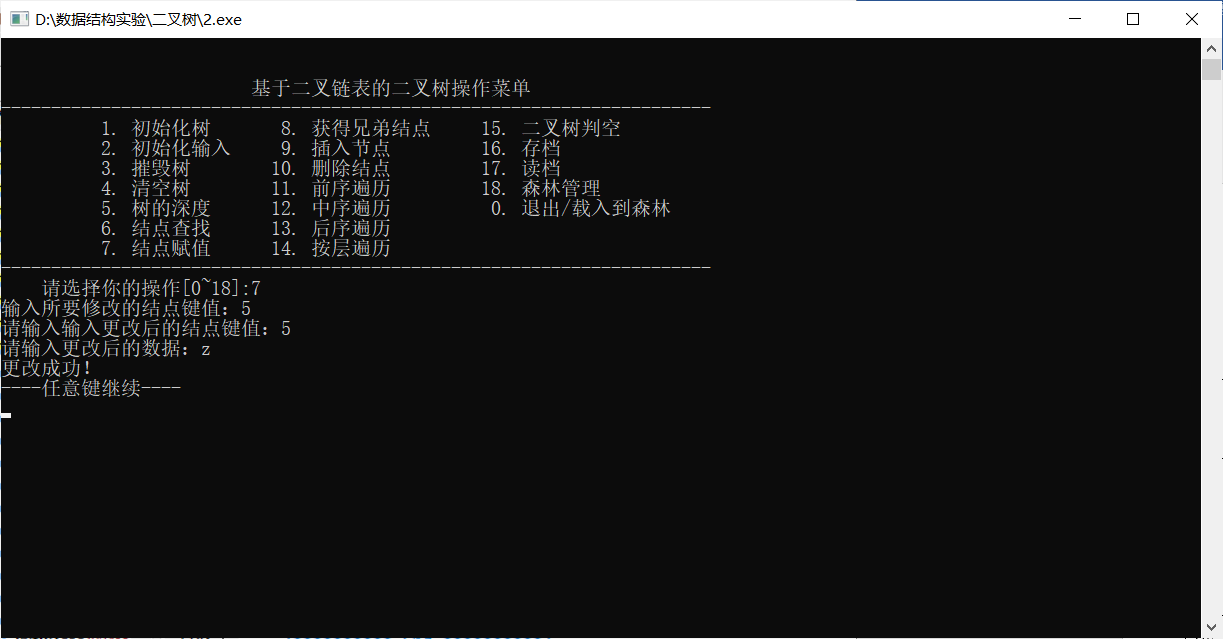


图3-12测试序号五的程序运行结果

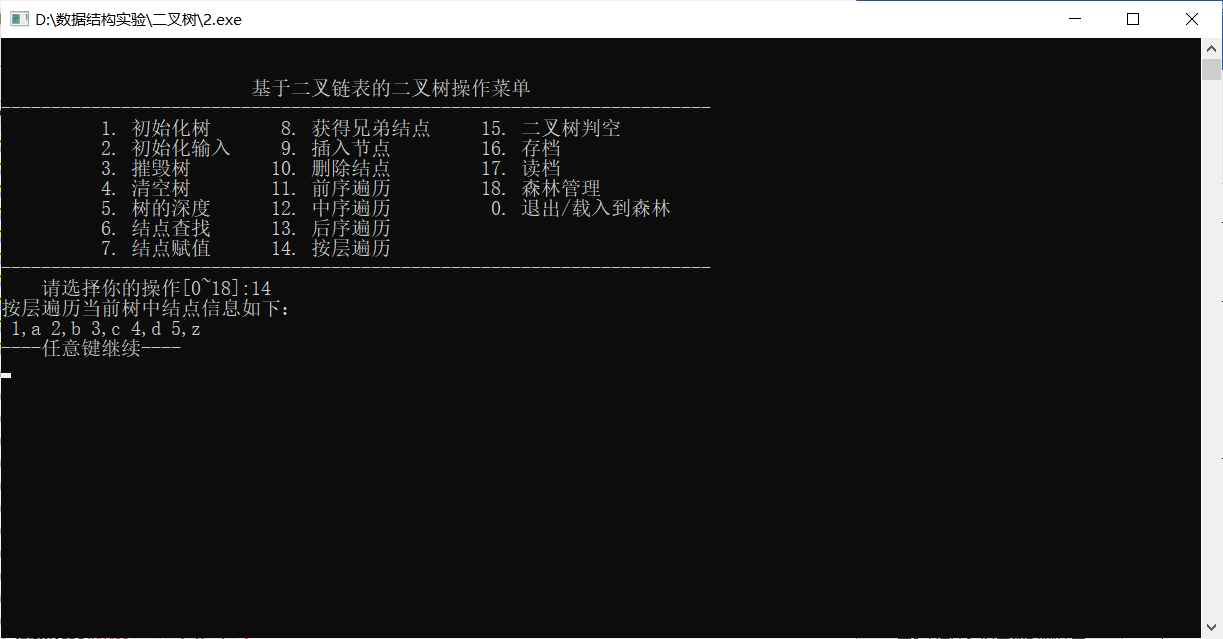


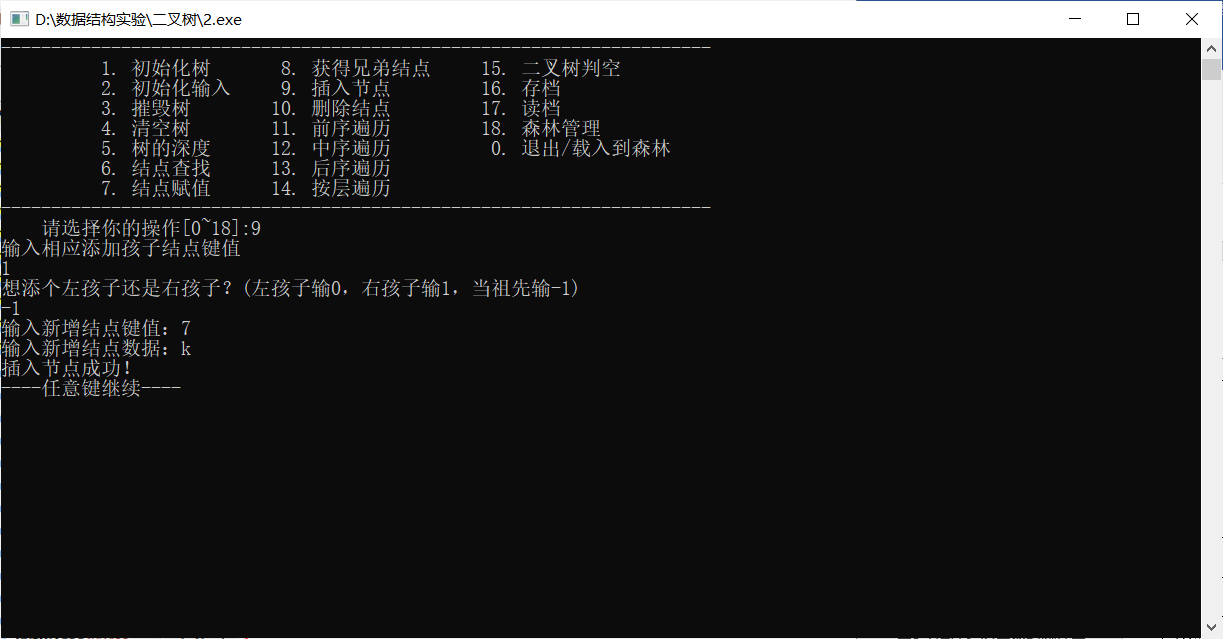
图3-13测试序号六的程序运行结果 

图3-14测试序号七的程序运行结果

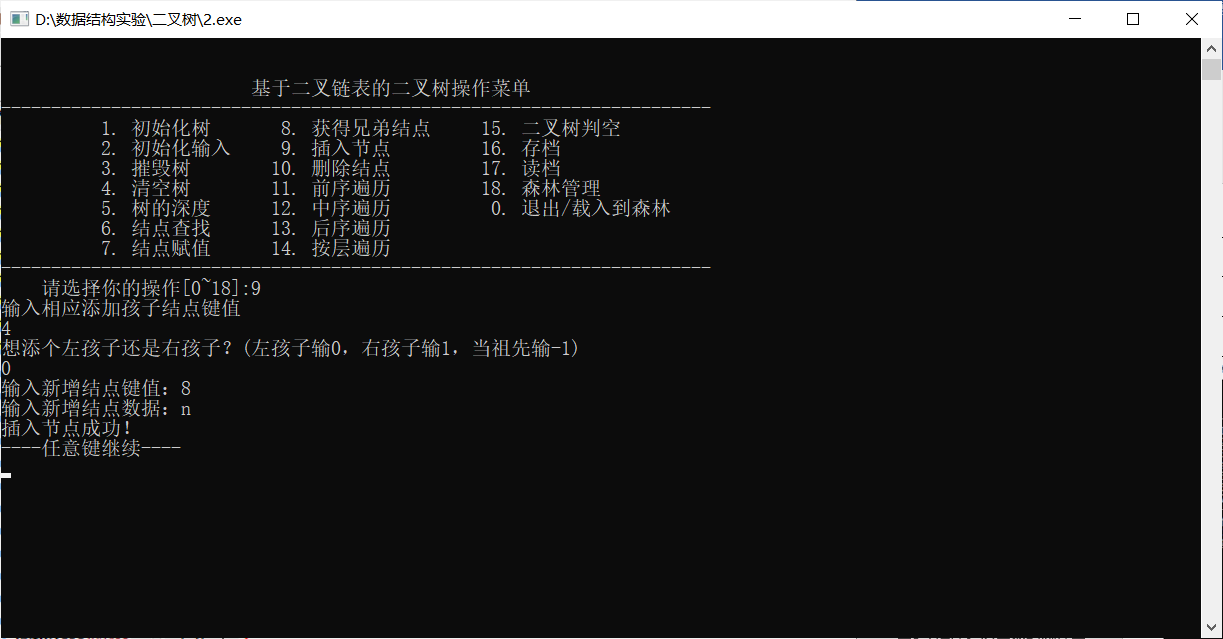


图3-15测试序号八的程序运行结果

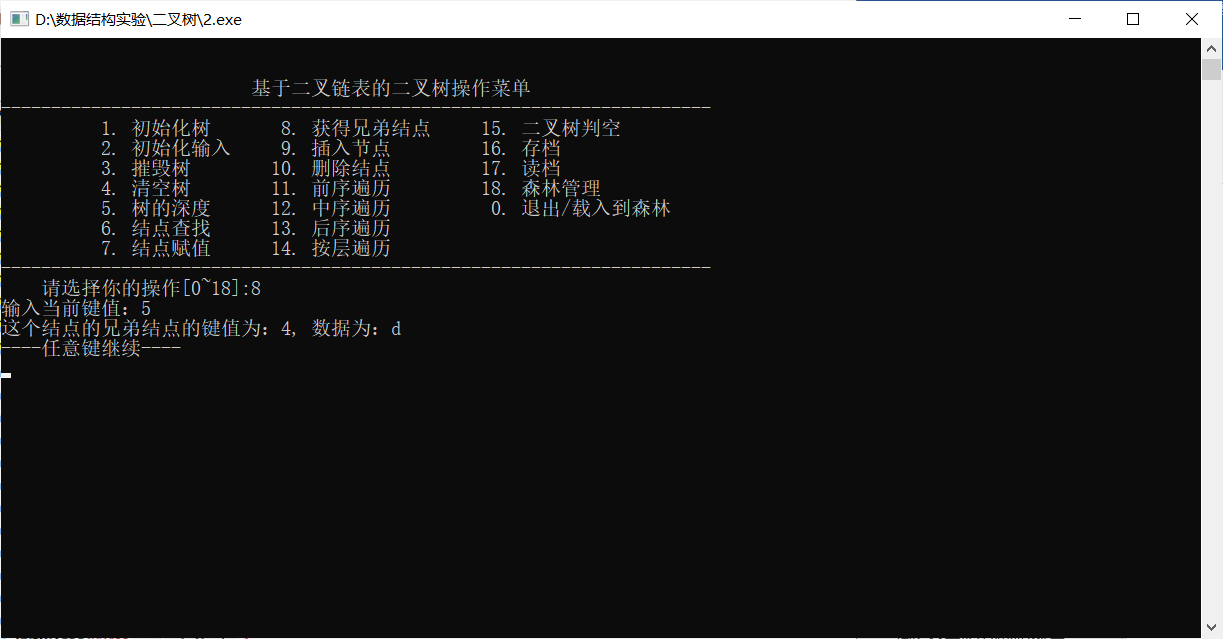


图3-16测试序号九的程序运行结果

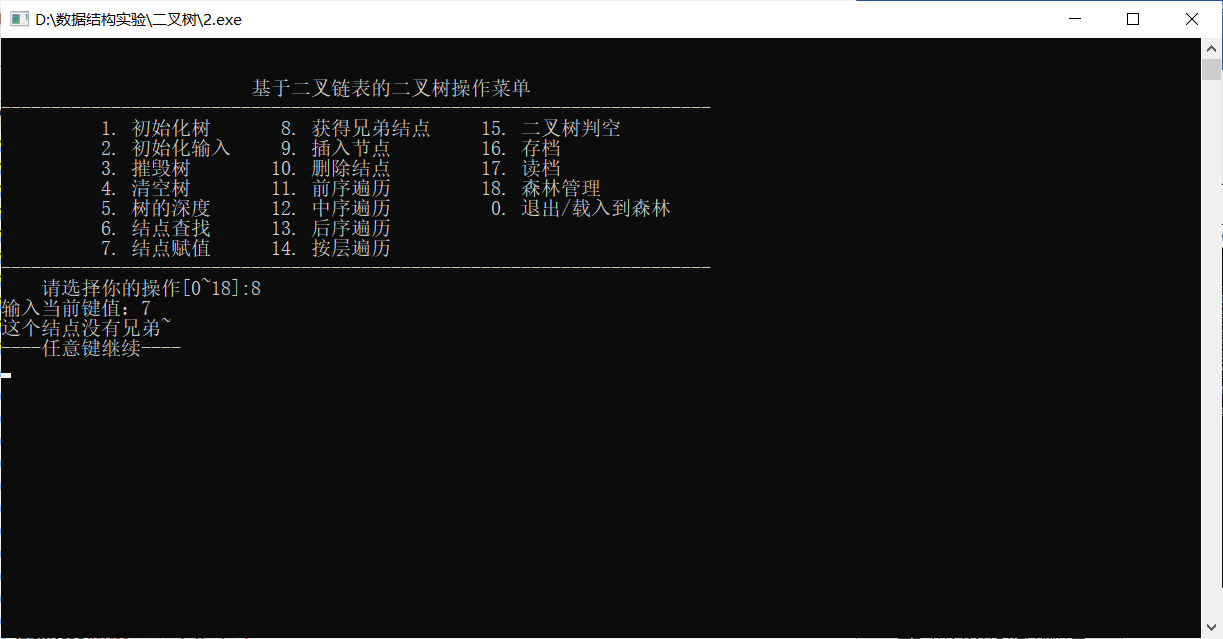


图3-17测试序号十的程序运行结果

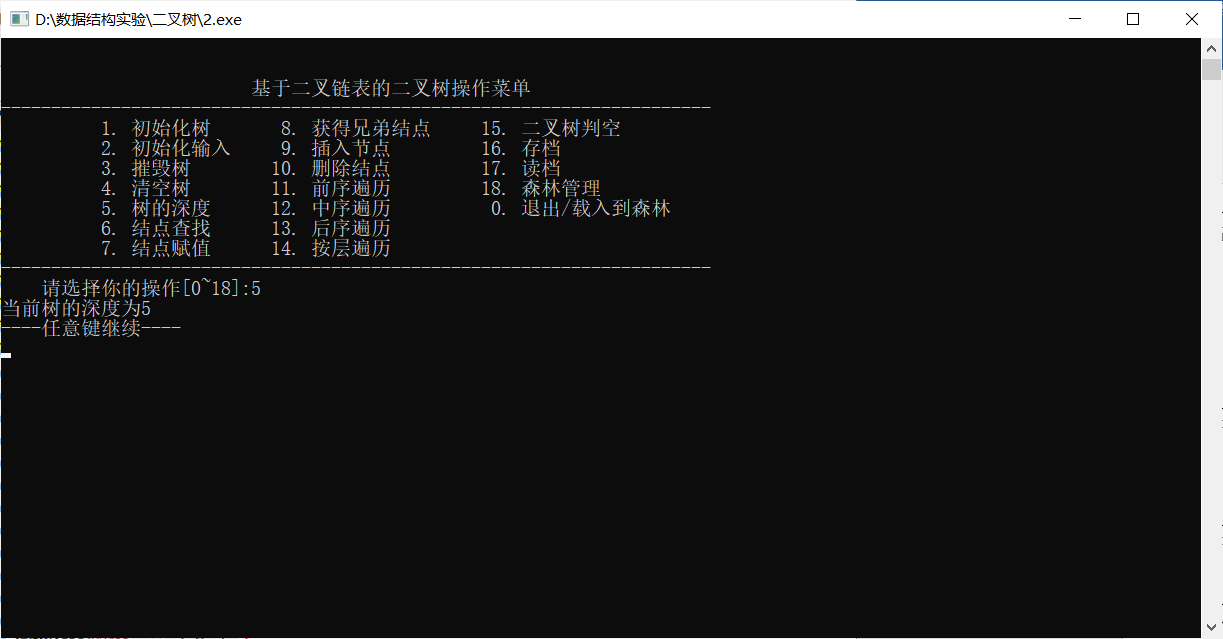


图3-18测试序号十一的程序运行结果

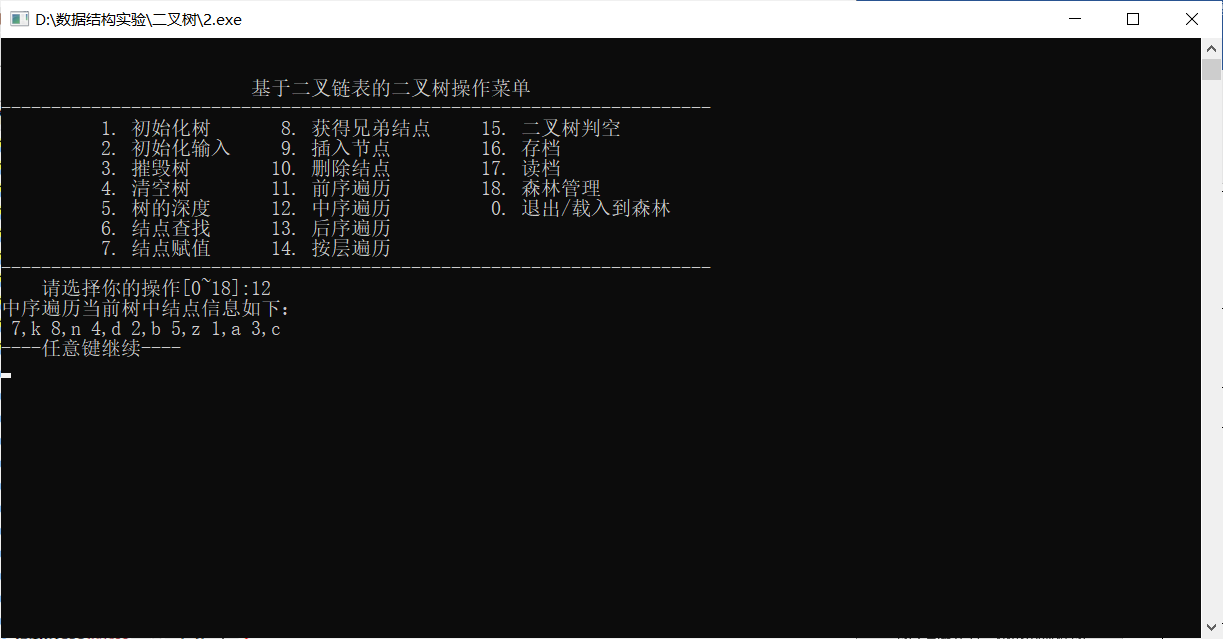


图3-19测试序号十二的程序运行结果

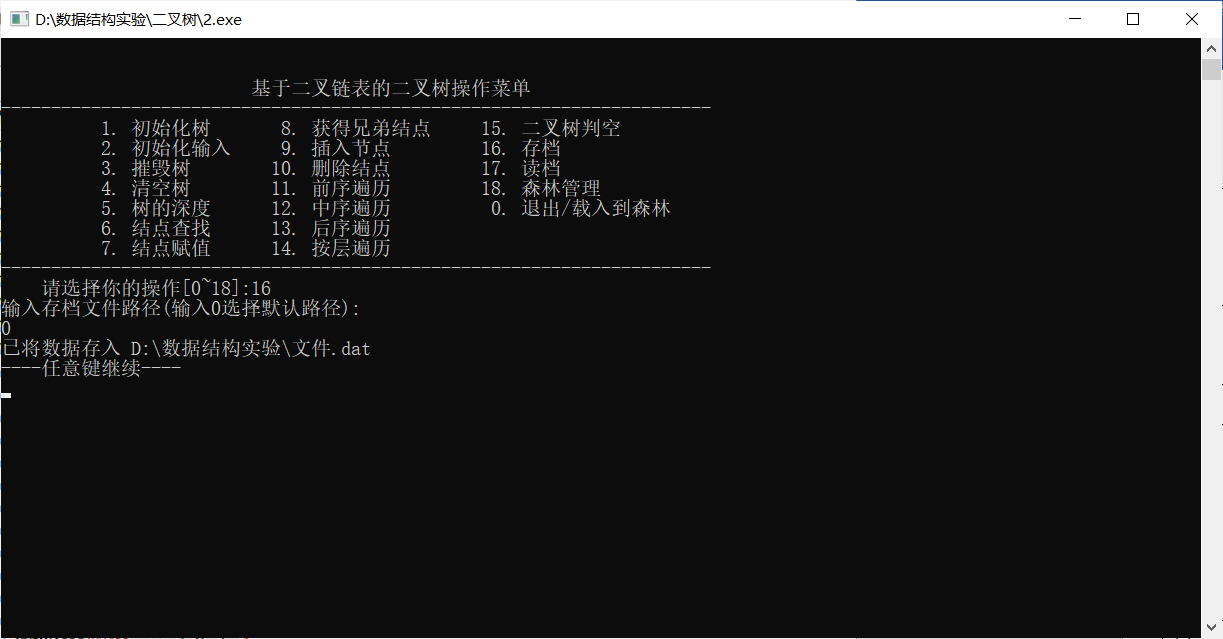


图3-20测试序号十三的程序运行结果

以下是森林的测试步骤展示：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 森林菜单的系统测试 | 输入的参数 | 预期结果 |
| 14 | 初始化森林 | 18  1 | 初始化成功！ |
| 15 | 向森林中添加名称为“1”的树 | 2  1  1 | 树"1"成功种进森林了！ |
| 16 | 向森林中添加名称为“2”的树 | 2  2  2 | 树"2"成功种进森林了！ |
| 17 | 修改森林中树1，输入数据1,1 2,2 3,3 4,4 | 6 1 0 2 1 1 1 2 2 2 3 3 3 4 4 4 0 0 0 | 输入成功！ |
| 18 | 将该二叉树的数据存入到文件中 | 16 0 | 已将数据存入 D:\数据结构实验\文件.dat |
| 19 | 载入到森林 | 0 | 原主界面的树中数据已载入！ |
| 20 | 将文件中的数据读取到树2中 | 18  6  2  0  17  0 | 数据已从 D:\数据结构实验\文件.dat 中读出 |
| 21 | 载入完成后按层遍历森林 | 0  18  4 | 第1棵树：  1:  1,1 2,2 3,3 4,4  第2棵树：  2:  1,1 2,2 3,3 4,4 |
| 22 | 清空森林中树1 | 5  1 | 清空成功 |
| 23 | 先序遍历森林中的树 | 7  1 | 第1棵树：  1:  空  第2棵树：  2:  1,1 2,2 4,4 3,3 |
| 24 | 删除森林中名为1的树 | 3  1 | 树"1"成功被砍没了！ |
| 25 | 后序遍历森林 | 7 | 当前树中数据如下所示：  第1棵树：  2:  4,4 2,2 3,3 1,1 |

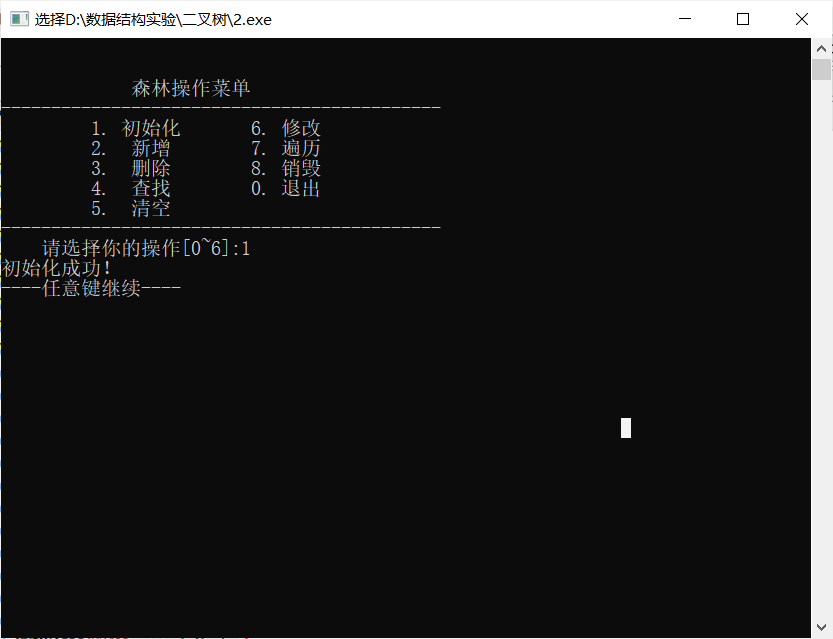


图3-21测试序号十四的程序运行结果

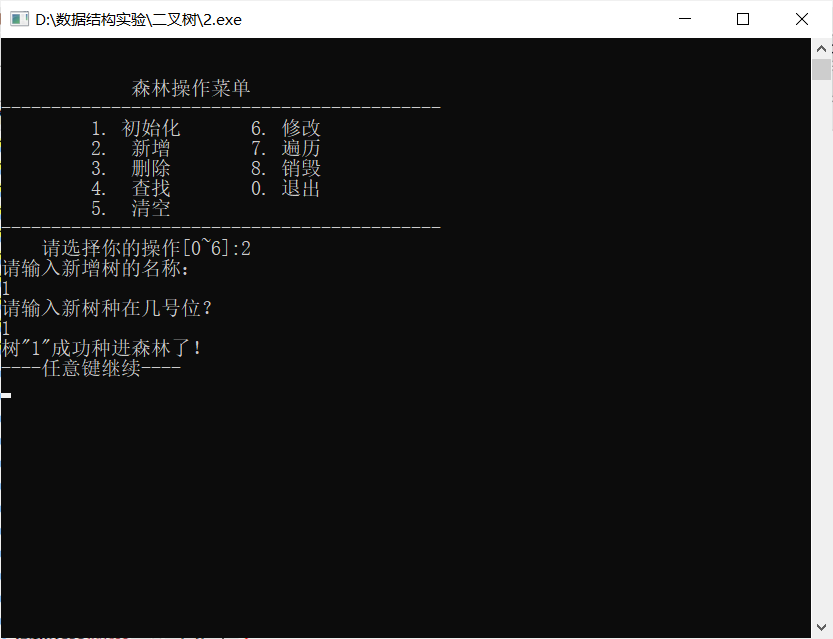


图3-22测试序号十五大的程序运行结果

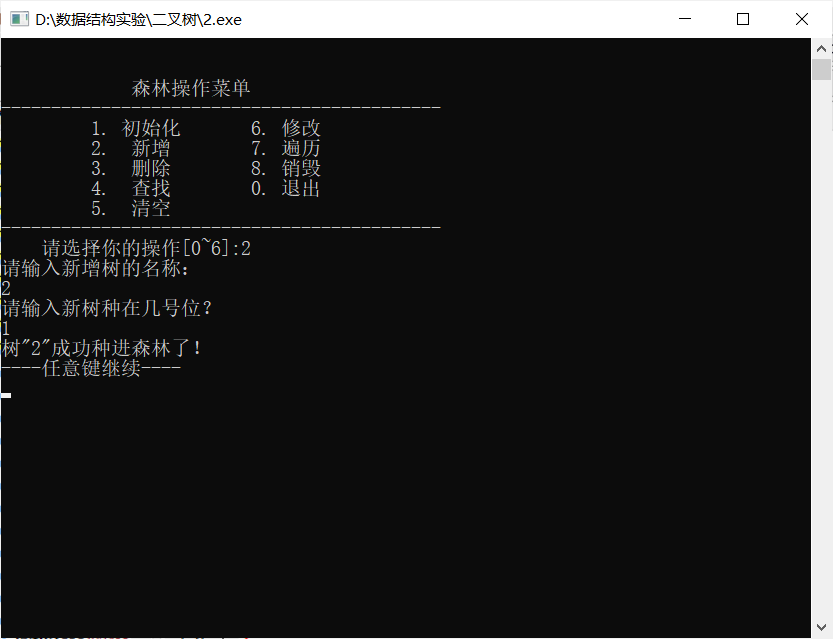


图3-23测试序号十六的程序运行结果

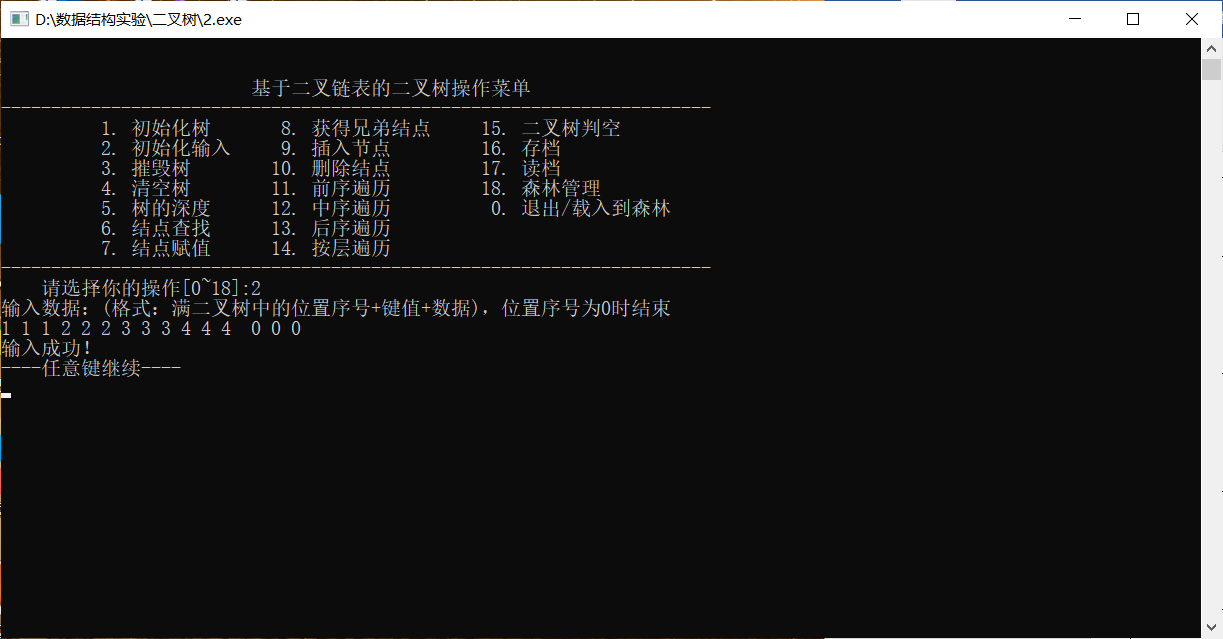


图3-24测试序号十七的程序运行结果

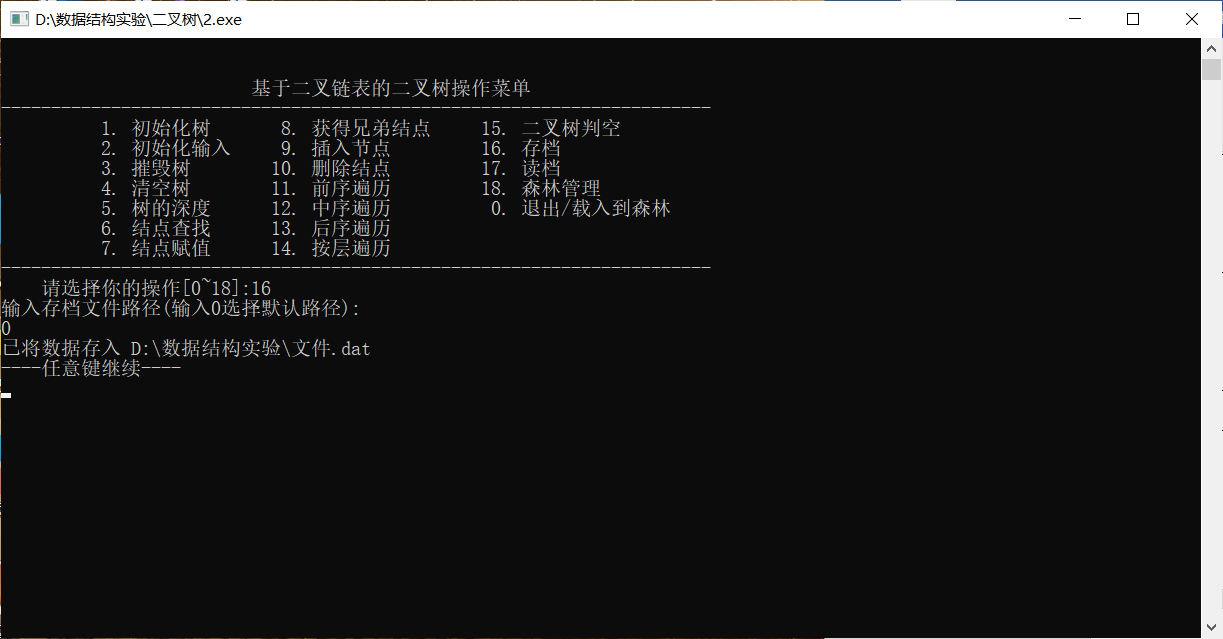


图3-25测试序号十八的程序运行结果

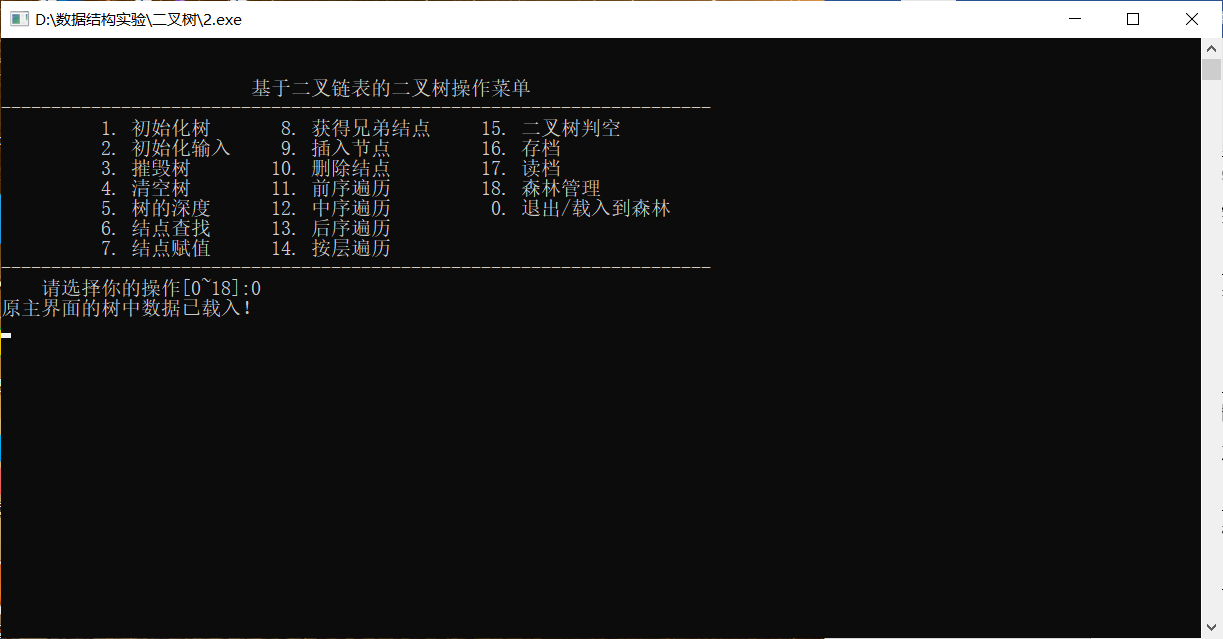


图3-26测试序号十九的程序运行结果

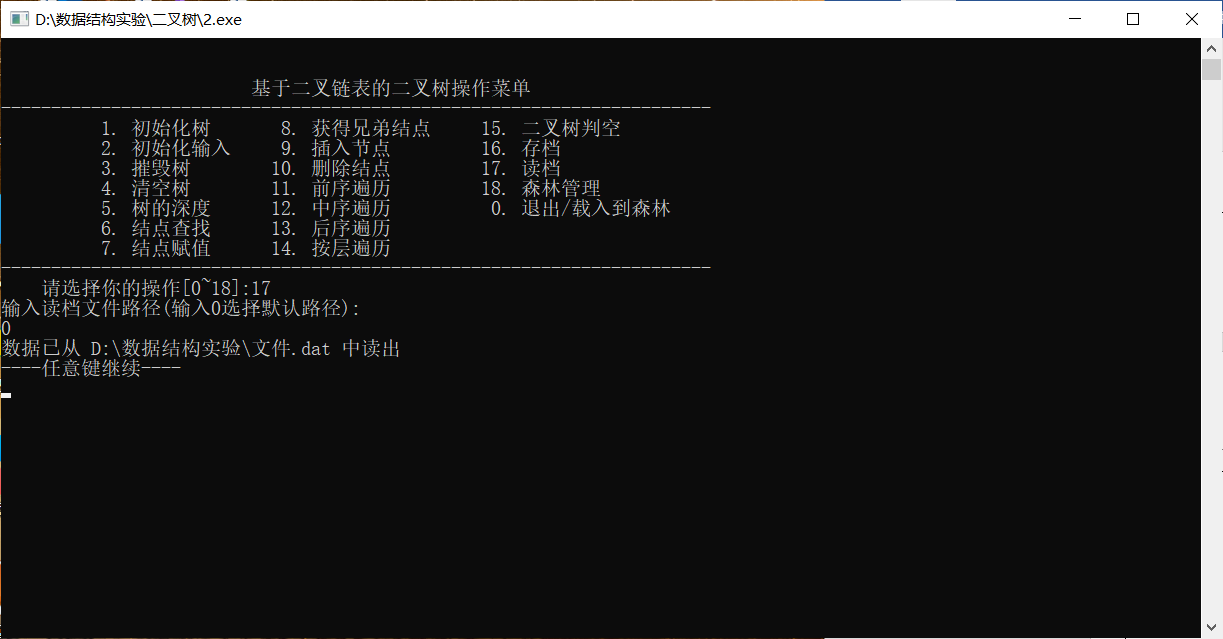


图3-27测试序号二十的程序运行结果

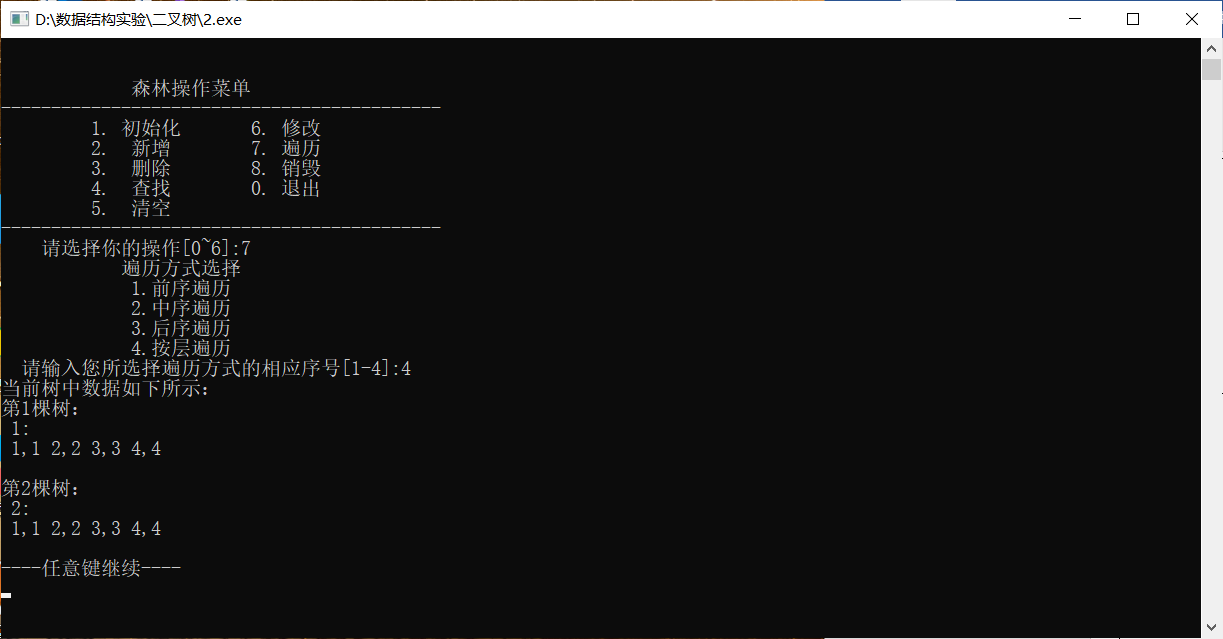


图3-28测试序号二十一的程序运行结果

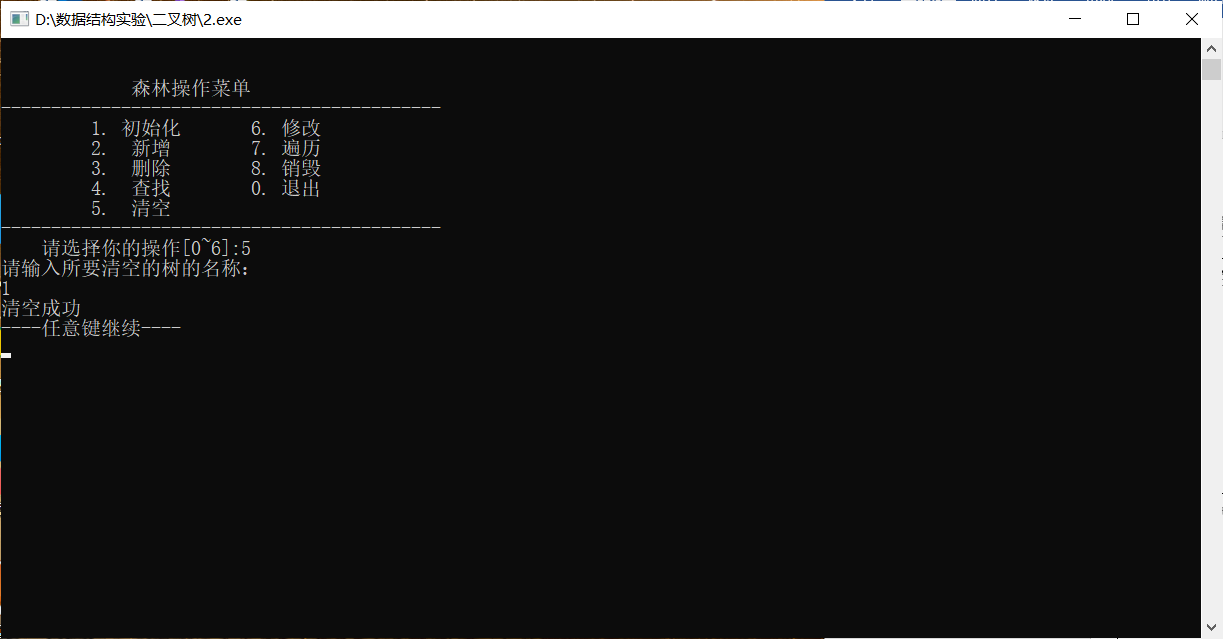


图3-29测试序号二十二的程序运行结果

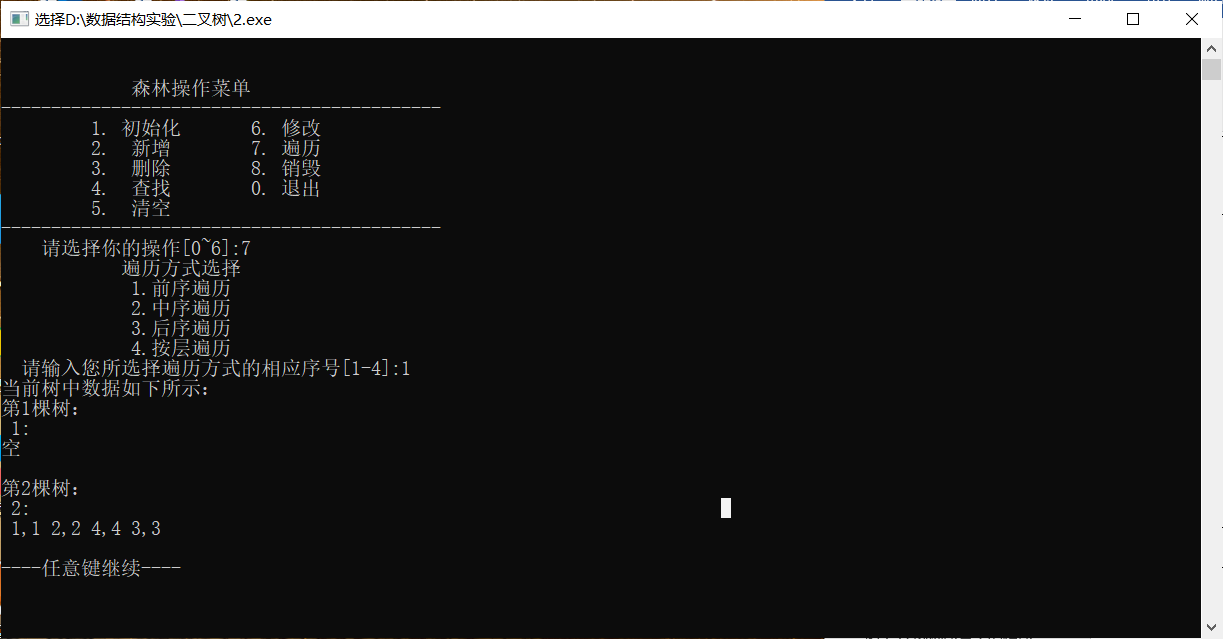


图3-30测试序号二十三的程序运行结果

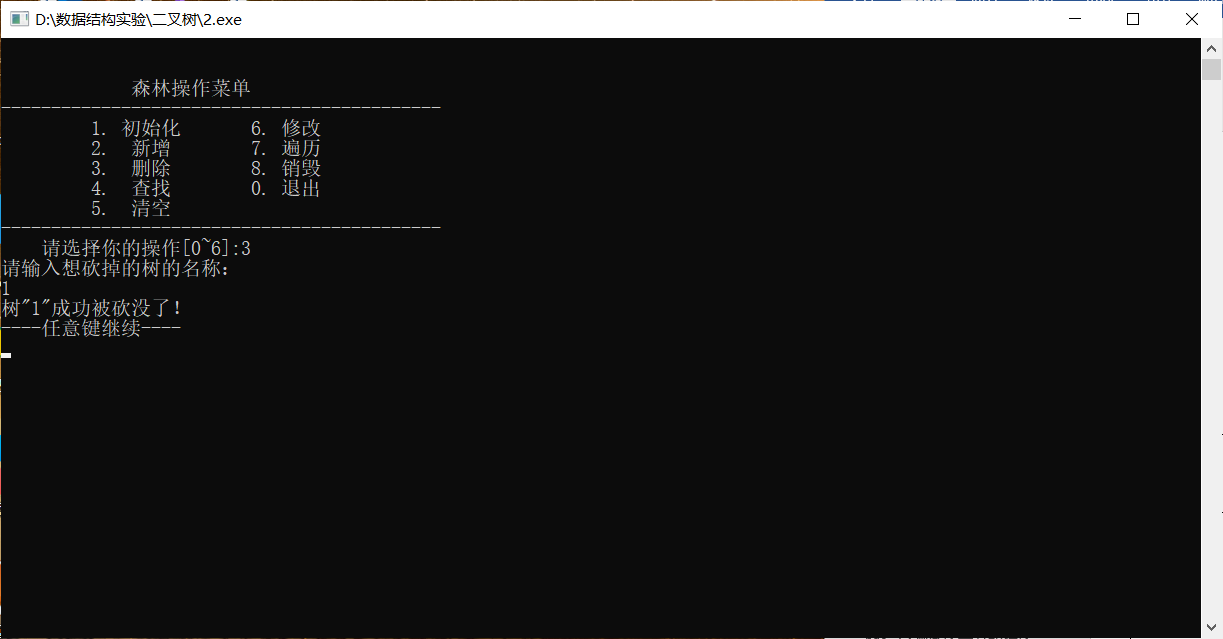


图3-31测试序号二十四的程序运行结果

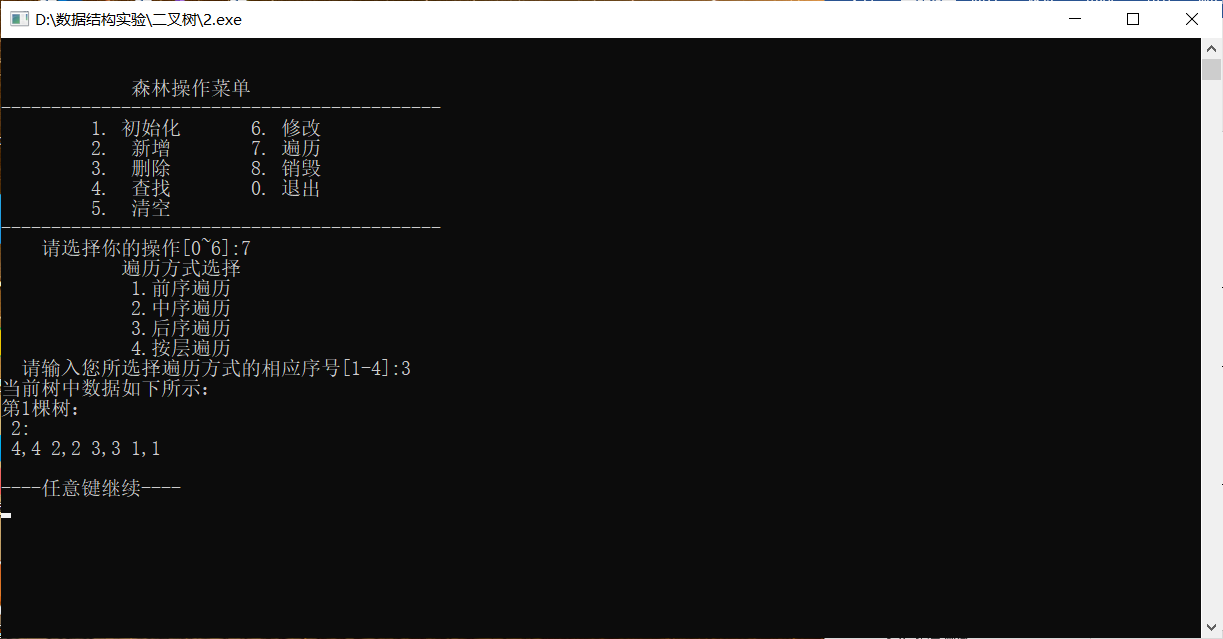


图3-32测试序号二十五的程序运行结果

4.测试分析：

相较于之前的系统操作，本次系统操作采用二级菜单的形式，逻辑上大为简化，但是，二级菜单的存在，表明一级菜单中的部分操作为二级菜单中的操作所覆盖，并且，由于二级菜单的简洁性，对于二叉树的单个操作又放入到一级菜单中，对于单个树的操作还需要调用一级菜单，但是可以自动保存，使得系统的逻辑性有一定的简化。

## 3.5 实验小结

通过本次实验，主要操作是对基于二叉链表的二叉树这一存储结构来进行。在学习过程中，我有以下几点收获：

灵活运用合适的数据结构：不同于前两次的实验，从操作上来讲，这次实验中所用到的存储结构相较于前两次的线性存储结构明显要复杂一些，但此次实验也能看出其它数据结构的影子：非递归中序遍历(访问受限的线性表)、森林(利用线性表存储二叉树根结点地址)……而这，也让我感受到了在实际中对各种数据结构的灵活使用是多么重要。

对递归这一方法更多的认识：此次实验中，不少函数用递归可以非常简单且快速的完成，但是对于递归的不熟悉以及诸多弊端的畏惧使得之前的练习中几乎没有采取递归的方法，递归其实就是一个函数里不断地产生“代码块”。

对静态变量的了解：由于此次实验，由于对于递归的全面运用，使用了很多全局变量，许多使用了静态变量的递归函数，没有手动将其还原成初始状态，导致非法访问、长度错误等等一系列问题。

# 4 基于邻接表的图实现

## 4.1 问题描述

图(Graph)是由非空的顶点集合和一个描述顶点之间关系――边（或者弧）的集合组成，其形式化定义为：G＝（V，E）其中，G 表示一个图，V 是图G 中顶点的集合，(vi,vj)表示顶点vi 和顶点vj 之间有一条直接连线，即偶对(vi,vj)表示一条边。在图形结构中，节点之间的关系是可以任意的，图中任意两个数据元素都可能相关。

问题一：图的实现

根据本次实验要求，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。

问题二：构造具有菜单的功能演示系统

根据本次实验要求，应该构造能够良好体现图的功能的演示程序，至少应当显示提供图操作的主菜单，其中，应该在主函数中完成函数调用所需的实参值的准备和实现函数执行结果的显示，并提供适当的操作提示信息，演示系统应当实现输入验证和异常响应，对于用户的非法输入给与提示，并能够允许重新输入，对于程序的运行错误也能实现异常信息的输入，演示系统应当有合理的层次

问题三：实现图的文件形式的保存

根据本次实验要求，实验应当实现图的文件形式的保存，包括

（1）设计文件的记录格式，使得文件的保存与读取均能实现高效率。

（2）设计图的保存，使得保存后重读取的图能够实现与原图相同的格式。

## 4.2 系统设计

根据本次实验要求，系统设计包括以下的内容：

1. 设计结构类型，实现图的物理结构。
2. 设计具有菜单的演示系统的框架。
3. 设计多重图格式，使得可以进行多个图管理
4. 设计文件格式，使得保存后重读取的文件格式保持一致性。

设计一：设计结构类型，实现图的物理结构

图的物理结构如下：

typedef struct {

KeyType key;

char others[20];

} VertexType; //顶点类型定义

typedef struct ArcNode { //表结点类型定义

int adjvex; //顶点位置编号

struct ArcNode \*nextarc; //下一个表结点指针

} ArcNode;

typedef struct VNode{ //头结点及其数组类型定义

VertexType data; //顶点信息

ArcNode \*firstarc; //指向第一条弧

} VNode,AdjList[MAX\_VERTEX\_NUM];

typedef struct { //邻接表的类型定义

AdjList vertices; //头结点数组

int vexnum,arcnum; //顶点数、弧数（度数）

GraphKind kind; //图的类型

} ALGraph;

利用四个结构类型，使用邻接表对于图进行存储，使得图的存储能够实现对于不同情况下的各种访问。

设计二：演示系统框架设计

本实验采用多级菜单设计，在第一层的菜单为基础的图修改操作，包括有12种基本操作和图的存档读档，第二级菜单为多重图操作，可以实现初始化，增加图，删除图，清空图，查找图，修改图，遍历图，摧毁多重图。通过修改图这一操作，完成第二级菜单向第一级菜单的过渡。

设计三：设计多重图格式，使得可以进行多个图管理

多重图的物理结构如下：

typedef struct {

char name[30];//图的名字

ALGraph G;

}Graph, \*G;

typedef struct { //多图定义

G \*elem;

int length;//多重图的长度

int size;//多重图的大小

}ALGraphs;

利用两个结构类型，实现对于多重图，以及其中的图的操作。

设计四：使得保存后重读取的文件格式保持一致性

对于文件的保存，由于最开始的输入中，图中对应顶点的弧所对应到邻接表中采用的是首插法，所以在保存后的文件中，不能直接采用对应格式的输入，所以，对于要保存的文件，直接进行对应元素的保存，使得弧的顺序也没有发生改变，通过换行符，进行不同的节点的区分，在读取的时候，也按照存储的方式读，使得读入文件的数据结构类型与输入的数据结构类型一致。

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。具体运算功能定义和说明如下。具体运算功能定义如下。

（1）创建图：函数名称是CreateCraph(G,V,VR)；初始条件是V是图的顶点集，VR是图的关系集；操作结果是按V和VR的定义构造图G。

注：①要求图G中顶点关键字具有唯一性。后面各操作的实现，也都要满足一个图中关键字的唯一性，不再赘述；② V和VR对应的是图的逻辑定义形式，比如V为顶点序列，VR为关键字对的序列。不能将邻接矩阵等物理结构来代替V和VR。

⑵销毁图：函数名称是DestroyGraph(G)；初始条件图G已存在；操作结果是销毁图G。

(3)查找顶点：函数名称是LocateVex(G,u)；初始条件是图G存在，u是和G中顶点关键字类型相同的给定值；操作结果是若u在图G中存在，返回关键字为u的顶点位置序号（简称位序），否则返回其它表示“不存在”的信息。

⑷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；操作结果是对关键字为u的顶点赋值value。

⑸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；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。

⑹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。

⑺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。（在这里也保持顶点关键字的唯一性）

⑻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；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。

⑼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；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>。

⑽删除弧：函数名称是DeleteArc(G,v,w)；初始条件是图G存在，v、w是和G中顶点关键字类型相同的给定值；操作结果是在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>。

⑾深度优先搜索遍历：函数名称是DFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

⑿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；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。

以下为附增函数：

保存图：函数名称是SaveGraph(\*G, SaveFile);初始条件是图存在，操作结果是将图存入到对应名字的文件中。

读取图：函数名称是LoadGraph(\*G, LoadFile)；初始条件是图中没有信息，操作结果是对于文件中的数据，将文件中的数据读取到图中。

以下为多重图操作：

（1）初始化多图：函数名称是InitALGraphs(GS)，初始条件是多重图没有被初始化，操作结果是对于多重图表中丰沛空间，多重图表的长度置为0，大小置为20，成功后返回OK

（2）向多重图表中新增图：函数名称是AddGraph(GS, i, GraphName)初始条件是多重图已经被初始化，操作结果是向多重图表的i位置上新增一个对应名字的图。成功返回OK。

（3）删除多重图中的对应图：函数名称为RemoveGraph(GS, GraphName)，初始条件是多重图已被初始化，操作结果是删除多重图中对应名字的图，成功返回OK。

（4）定位多重图中的对应图：函数名称是LocateGraph(GS, GraphName)，初始条件是多重图已被初始化，操作结果是返回对应的图在多重图中的位置。

（5）遍历多重图：函数名称是ALGraphsTraverse(GS)，初始条件是多重图已被初始化，操作结果是，按照对应的输入，进行深度遍历或者广度遍历，依次输出多重图中的图中的数据。

（6）清空多重图：函数名称是ClearALGraphs(GS)，初始条件是当前多重图已被初始化，操作结果是，清空多重图中数据，返还存储空间，将多重图的长度重新置为0，多重图的第一位指向一个空的地址。

## 4.3 系统实现

本系统通过命令式菜单来调用各函数功能，系统开始运行时会输出菜单界面，依照菜单中给出的各功能的指令输入命令，以实现相关功能的使用。

系统的具体实现方式为：

首先根据图的基本操作实现完成基本函数构建，然后通过对于多种情况的考虑完善各个子函数。

在主函数内部完成函数调用所需实参值的准备和函数执行结果的显示。利用op存储当前命令，并且采用switch-case语句，依据输入的op进行选择，调用函数实现相应功能。

每个子函数的实现方法如下：

1.输入创建图：该函数分为CreateGraph(ALGraph &G)与CreateCraph(ALGraph &G,VertexType V[],KeyType VR[][2])两个，对于第一个函数，提供输入的格式提示（"按照\"键值 名称 键值 名称 ... -1 nil 连接键值 连接键值 ... -1 -1\"输入\n），并且，按照对应的格式存储，使得存储的格式与线上平台中的一致，这样就可以直接调用线上平台的函数，即第二个函数，实现对于图的创建，利用数组V来实现对于图中的对应节点的键值和内容的存储，利用二维数组VR实现对于图中对应弧的头尾节点的存储，在图的创建中，根据数组V中的数据，依次写入到图中的相应节点，并将节点的头连接节点置为空，随着每一个节点的读入，图中额顶点数目依次+1，然后实现对于弧的头尾指针的创建，对于VR中的数据，分别用head和tail来存放相对应的一条弧的头尾根据首插法，重新定义一个节点指针，该指针置为节点的头指针，节点中的值置为对应连接弧的节点，同时，所有一个节点的度都存放在count中，图的度即为count/2.该函数实现的流程图如图4-1所示：



图4-1创建无向图的函数流程图

2.摧毁无向图：该函数为DestroyGraph(ALGraph& G)依次遍历图中的各个节点，对于每一个节点，释放掉邻接链表中的数据，邻接头指针置为空，将图中的节点数和弧数都置为0.

3.定位图中的节点：该函数为LocateVex(ALGraph G, KeyType u)依次遍历图中的各个节点，当找到节点的键值与u的值相同的时候，返回键值所在的位置i，否则返回-1表示没有查找到。

4.修改图中的对应节点：该函数为PutVex(ALGraph &G,KeyType u,VertexType value)根据输入的u和value调用LocateVex(ALGraph G, KeyType u)函数，只有当没有定位到value.key的节点，且u在图中有对应的节点的时候，将对应的节点的数据改变为value中的数据。

5.查找图中顶点的第一邻接顶点：该函数为FirstAdjVex(ALGraph G,KeyType u)根据输入的u定位到图中的对应节点，并返回该节点的第一邻接点，如果查找失败，返回ERROR。

6.查找图中节点对应邻接节点的下一节点：该函数为NextAdjVex(ALGraph G,KeyType v,KeyType w)根据输入的v定位图中的顶点，如果在图中没有找到，返回-1，查找该顶点的邻接节点键值为w的下一节点的键值，成功返回该下一节点的键值。没有找到也返回-1.

7.像图中插入节点：该函数为InsertVex(ALGraph &G,VertexType v)向图中插入键值为v的顶点，首先要定位图中键值为v的顶点，如果存在则返回ERROR否则，说明图中没有键值为v的顶点，判断图中的顶点个数是否超过了最大值，如果超过，返回ERROR，没有超过，则图的顶点数代表了图中的顶点个数，也可以通过该顶点数在对应的位置上新增一个节点，该节点的邻接头节点为空，最后图中的顶点数+1返回OK。

8.删除图中的节点以及相关的弧：该函数为DeleteVex(ALGraph &G,KeyType v)首先要定位到图中的键值为v的节点，当图中的节点总数不为1且对应的节点在图中，进行更改，首先是释放连接到该节点中邻接表的空间，利用指针p依次遍历图中所有节点的邻接链表，如果存在该节点为图中其他节点的邻接头节点，那么就将节点置为p的后继节点，并释放p的空间，随后利用p指针直接释放该节点的邻接链表，依次删除图中的弧的总数，然后调整节点顶点的位置，在该节点后的节点依次前移，图的总顶点数-1，调整弧的位置，并返回OK。该程序实现的流程图如图4-2所示：



图4-2删除图中节点的流程图

9.向图中插入弧：该函数为InsertArc(ALGraph &G,KeyType v,KeyType w)首先，当图中总的节点数小于2时以及在图中没有定位到相应的两个节点，不满足插入弧的条件，返回ERROR，当节点都存在于图中时，但是仍然要判断是否图中已经存在该弧，存在的话就返回失败，当一切都满足插入弧的条件的时候，对于弧进行插入，还是利用首插法，新创建一个邻接指针，使得指针的内容指向对应的节点，指针的后继置为原头邻接点，原头邻接点该置为指针，图中的弧数+1，返回OK。该函数对应的流程图如图4-3所示：



图4-3向图中插入弧的流程图

10.在图中删除弧：该函数为DeleteArc(ALGraph &G,KeyType v,KeyType w) 首先，当图中总的节点数小于2时以及在图中没有定位到相应的两个节点，不满足删除弧的条件，返回ERROR，当节点都存在于图中时，但是仍然要判断是否图中已经存在该弧，不存在的话就返回失败，当图中存在对应的弧的时候，利用指针p进行定位，将p的前置节点的指针指向p的后置节点，并释放p的位置。成功返回OK。该函数对应的流程图如图4-4所示：



图4-4在图中删除弧的流程图

11.深度优先搜索遍历：该函数为DFSTraverse(ALGraph G, void (\*visit)(VertexType))首先，利用state数组来存放图中对应节点的状态，当对应节点的值为0时，表示还没有进行访问，当对应节点的值为1时，表示已经被访问过，调用辅助函数DFS，来进行递归遍历。辅助函数为：DFS(ALGraph G,int i ,void (\*visit)(VertexType))访问每一次进入的节点，并将状态函数state置为1，表示已经访问过了。根据深度优先遍历的要求，访问该节点的邻接点，当邻接点的状态函数为0时，进行递归访问，如果已经访问过，就访问该邻接点的后继节点，成功，返回OK。该函数对应的流程图如图4-5所示：



图4-5深度优先遍历的流程图

12.广度优先搜索遍历：该函数为BFSTraverse(ALGraph &G,void (\*visit)(VertexType))利用队列来辅助完成广度优先搜索遍历，利用state数组存放状态，队列的head和tail都置为0遍历图中的节点，当state数组表示没有被访问时，节点的第一邻接点入栈，tail++访问该节点，状态数组置为1，当head！=tail时，表示该队列不为空，队列的头节点出队，front+1，当队列的头节点不为空的时候，且该节点没有访问过的的时候，访问该节点，状态数组置为1，并且当邻接表的第一邻接点的后继节点不为空的时候，即该节点仍然还有弧，将弧节点入队，直到后继节点为空，到最后访问完成，将队列空间释放，返回OK。该函数实现的流程图如图4-6所示：



图4-6广度优先搜索遍历的流程图

13.保存图：该函数为SaveGraph(ALGraph G, char FileName[])打开相应名字的文件，遍历访问图中的各个节点，将节点的内容以及节点的邻接链表的内容放在一行，记录完成一个节点后，载入空格，方便后续的读取，遍历访问完成后，关闭文件，返回OK。

14.读取图：该函数为LoadGraph(ALGraph &G, char FileName[])打开相应的文件，利用ch变量来表明区别，图的节点数和弧数都置为0，读取文件中的数据，一开始读取的是节点，所以图的节点数+1，ch读取后续的字符，ch不为换行时，表示为该节点还有邻接节点，邻接节点进行赋值，弧数加一，如果不为换行，表示没有邻接节点，所以邻接头节点置为空，最后，将图中的节点数-1，弧数为统计的弧数的一般，关闭文件，返回OK。

以下为多重图操作

本次实验采用多级菜单的形式，在进入到多重图操作时，自动将第一级菜单中的图替换为多重图中的图，并利用flag来判断是否为多重图中的图flag=1，是表示为多重图中的图，在main函数中输入0表示将第一级菜单中的数据写入到多重图菜单中，在多重图菜单中，通过修改函数建立起两级菜单之间的联系。多重图的各个操作函数如下：

1.初始化多图：该函数为InitALGraphs(ALGraphs& GS)利用顺序表的物理结构，对于多重表进行空间分配，多重表的长度为0，多重表的大小置为20，返回OK。

2.向多重图中加入图：该函数为AddGraph(ALGraphs& GS, int i, char GraphName[])，当多重图初始化后，向多重图中对应的位置加入图，对于图进行空间分配，并询问是否要进行初始化输入，将对应位置置为该图，多重图的长度+1，返回OK。

3.移除多重图中的图：该函数为RemoveGraph(ALGraphs& GS, char GraphName[])遍历多重图，寻找名称符合的图，调用DestroyGraph函数，释放图中相应节点的空间，多重图移位，多重图的长度-1，返回OK。

4.遍历多重图，该函数为ALGraphsTraverse(ALGraphs GS)对于多重图，进行遍历方式的选择，分为深度优先遍历和广度优先遍历，依次调用遍历函数，

5.清空多重图，该函数为ClearALGraphs(ALGraphs& GS)调用DestroyGraph函数，将多重图中的每个节点依次摧毁，多重图的长度置为0，返回OK。

## 4.4 系统测试

根据本次实验要求，系统测试包括如下内容：

（1）测试准备

（2）测试计划

（3）测试结果

（4）测试分析

1.测试准备：

在本次测试中，主要测试多重图集合操作的准确性以及完备性，对于多重图中的图要进行初始化，输入，等多种函数的实现，并且，多重图中的图还能够进行存储，并且能够读取到集合中的任意图中。

2.测试计划：

在本次测试中，初始化时调用了文件中的图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文件中的内容根据在本次实验中的输入格式要求进行了格式上的更改，该系统进行了菜单层面上的优化，将多重图载入到二级菜单中，并且能够自动实现保存到多重图集合中。

3.测试结果：

此次系统测试采用到的文件“1”内容为：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

本次系统测试的步骤按照如下的表格进行：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 图菜单的测试 | 输入的参数 | 输出结果 |
| 1 | 创建图 | 1 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 | 创建成功！ |
| 2 | 查找键值为5的顶点 | 3 5 | 键值为5的结点在这个图里是第1个点, 其中的数据是：key=5, others=线性表 |
| 3 | 修改键值为6的顶点改为键值为9内容为有向图 | 4 6 9 有向图 | 更改成功！ |
| 4 | 获得键值为5的结点的第一邻接点 | 5 5 | 键值为5的点的第一邻接点是第3号点, 其中的数据是：key=7, others=二叉树 |
| 5 | 获得键值为6的下一邻接点 | 6 6 | 该图中没有与之对应的点 |
| 6 | 获得键值为5的点的键值为9的下一邻接点 | 6 5 9 | 无下一邻接点！ |
| 7 | 获得键值为5的点的键值为7的下一邻接点 | 6 5 7 | 键值为5的点相对于键值为7的点的下一邻接点为第4号点，其数据为：key=9, others=有向图 |
| 8 | 向图中添加键值为6，内容为图的结点 | 7 6 图 | 添加成功！ |
| 9 | 在键值为5和6的结点间添加弧 | 9 5 6 | 添加成功！ |
| 10 | 删除键值为9的结点 | 8 9 | 删除成功！ |
| 11 | 删除键值为5和7结点之间的弧 | 5 7 | 删除成功！ |
| 12 | 获得键值为5的结点的第一邻接点 | 5 5 | 键值为5的点的第一邻接点是第4号点, 其中的数据是：key=6, others=图 |
| 13 | 获得键值为5的点中键值为6的点的下一邻接点 | 6 5 6 | 无下一邻接点！ |
| 14 | 进行深度优先遍历 | 11 | 深度优先遍历时，当前图中所有点的信息如下：  5 线性表 6 图 8 集合 7 二叉树 |
| 15 | 进行广度优先遍历 | 12 | 广度优先遍历时，当前图中所有点的信息如下：  5 线性表 6 图 8 集合 7 二叉树 |
| 16 | 将图中中的数据存档 | 13 0 | 已将数据存入 D:\数据结构实验\文件.dat |
| 17 | 销毁图 | 2 | 销毁成功！ |

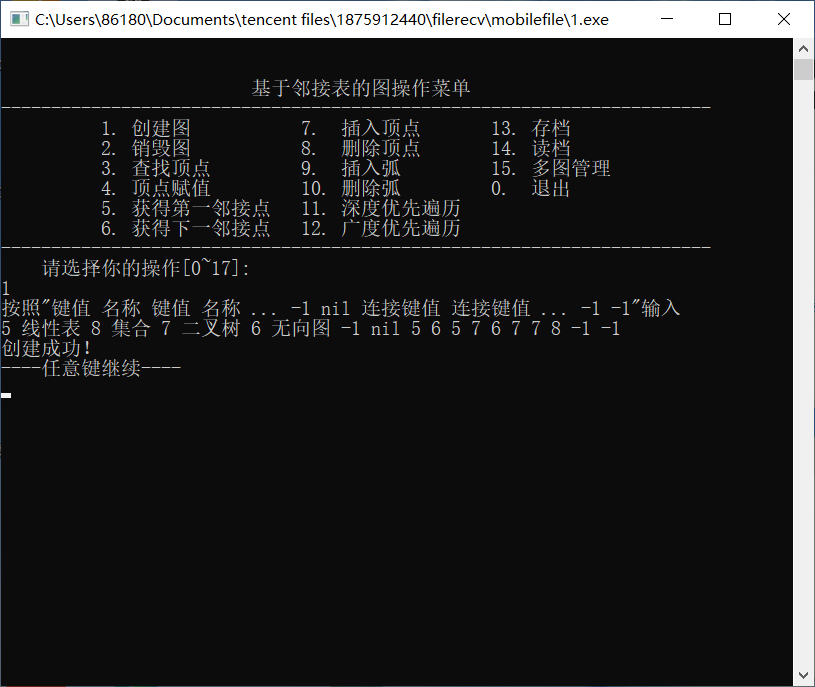


图4-7测试序号一的程序运行结果

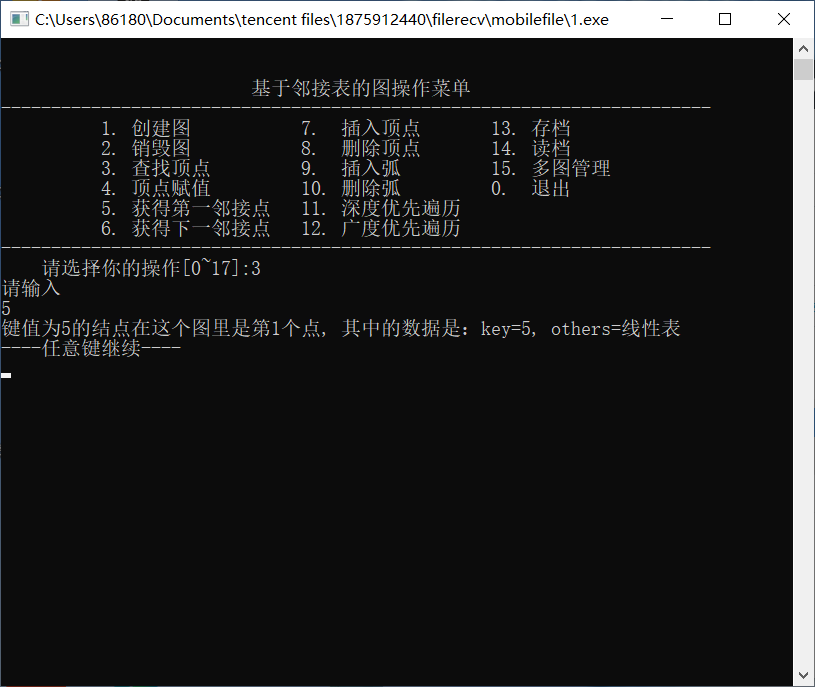


图4-8测试序号二的程序运行结果

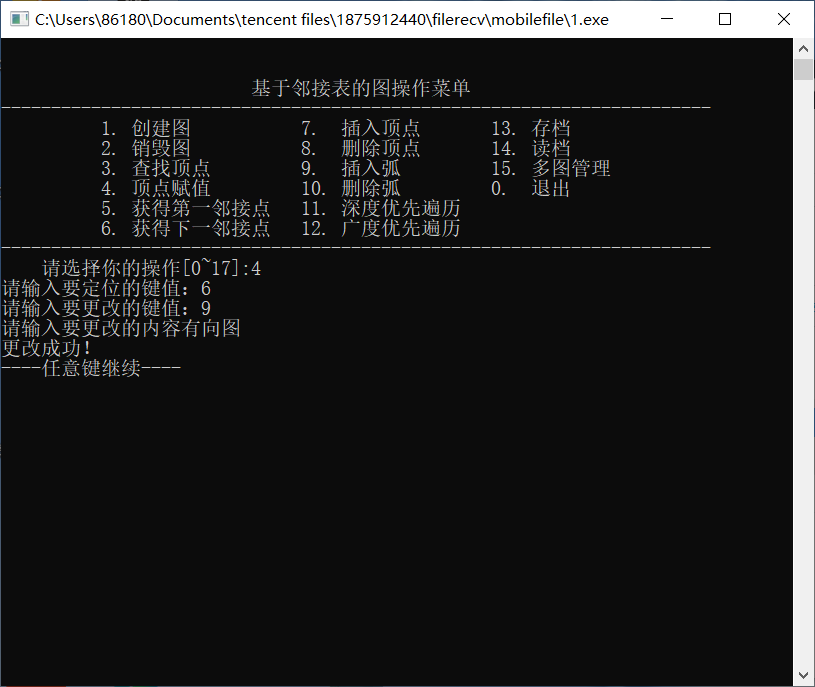


图4-9测试序号三的程序运行结果

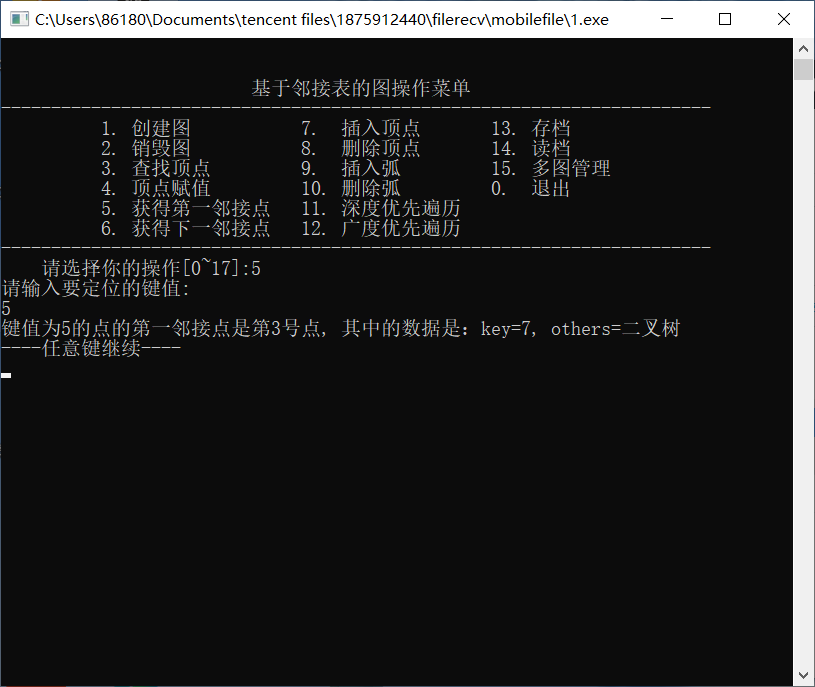


图4-10测试序号四的程序运行结果

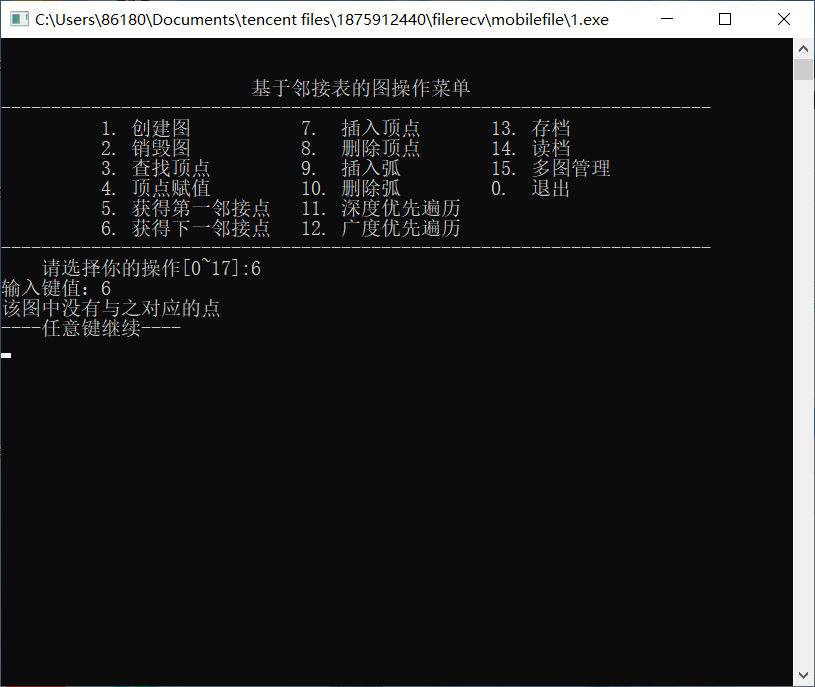


图4-11测试序号五的程序运行结果

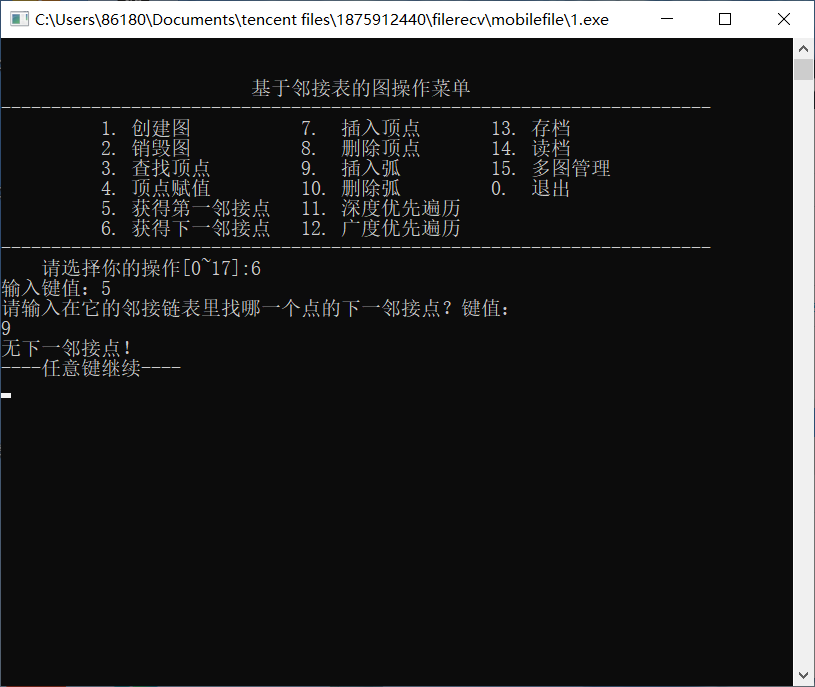


图4-12测试序号六的程序运行结果

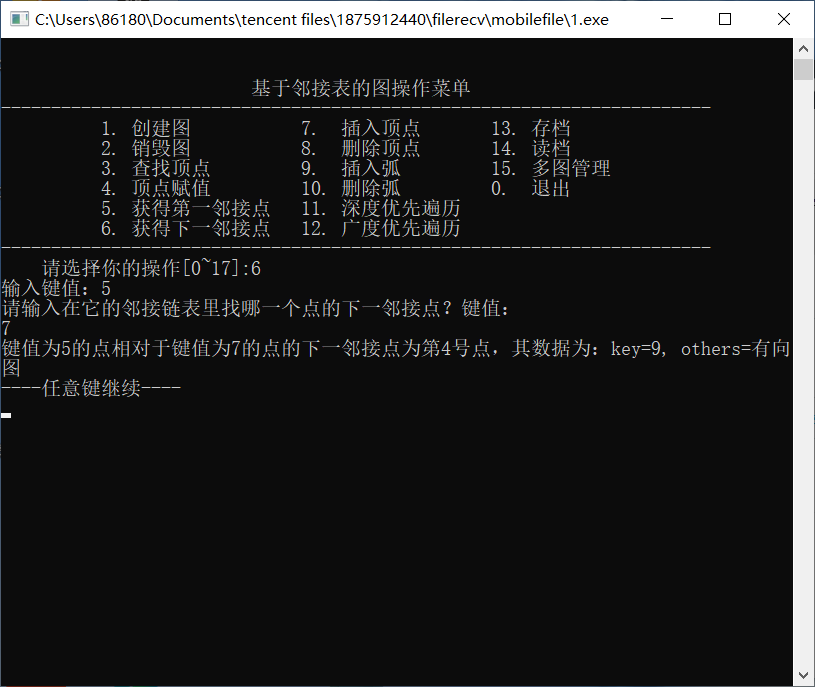


图4-13测试序号七的程序运行结果

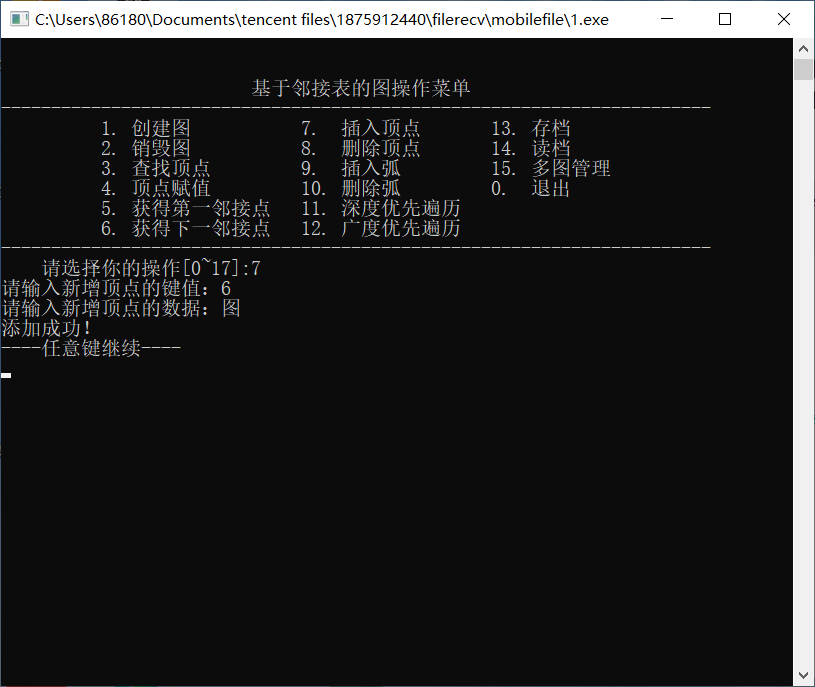


图4-14测试序号八的程序运行结果

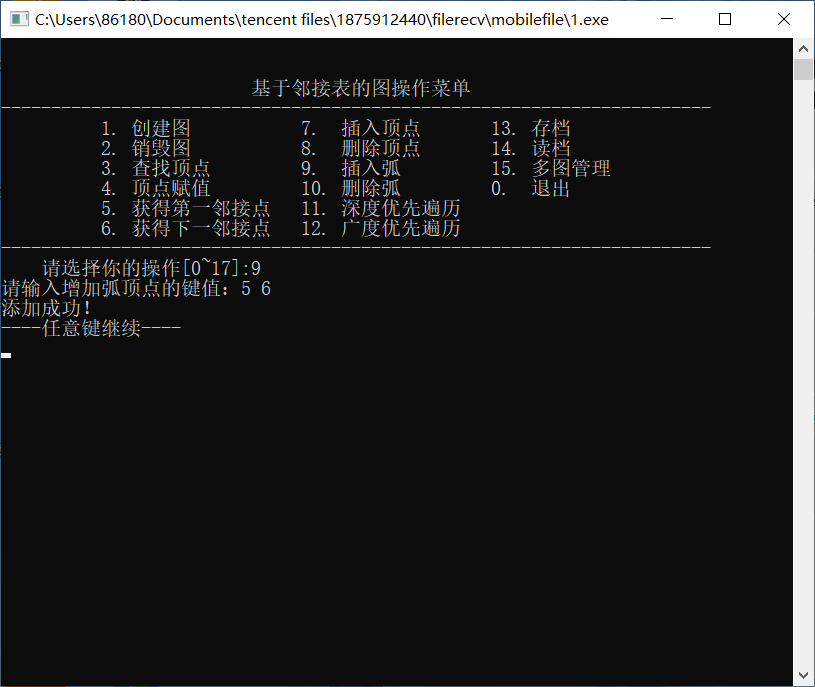


图4-15测试序号九的程序运行结果

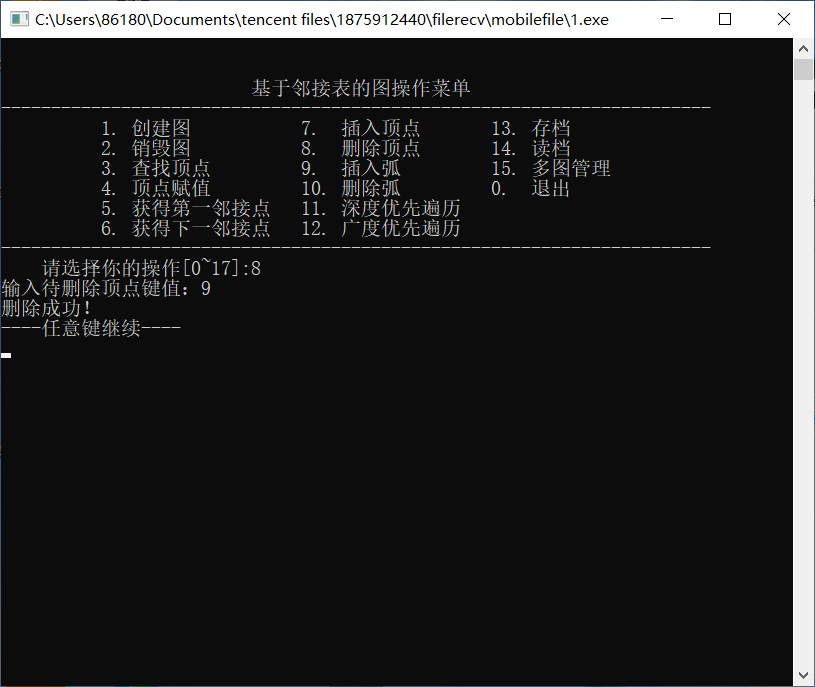


图4-16测试序号十的程序运行结果

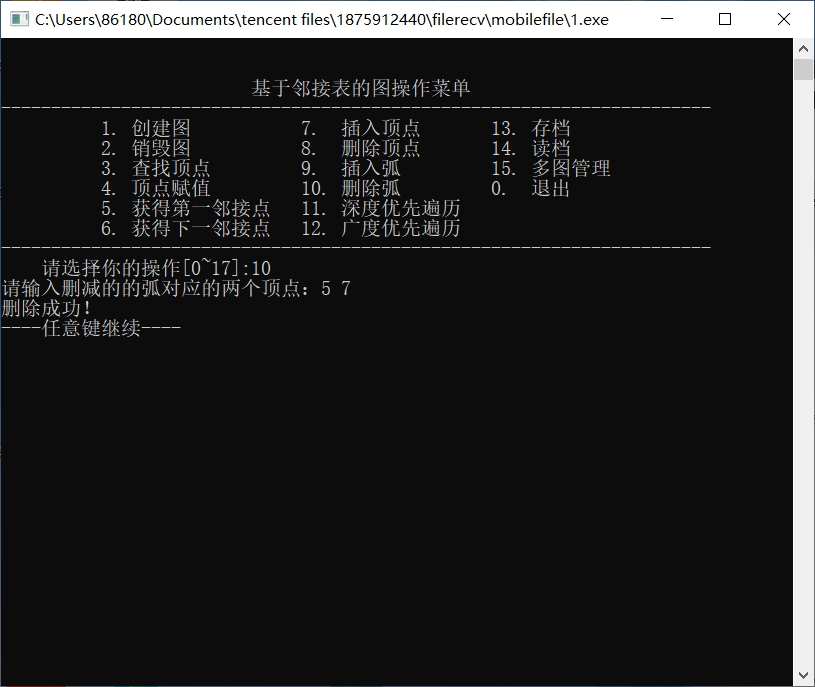


图4-17测试序号十一的程序运行结果

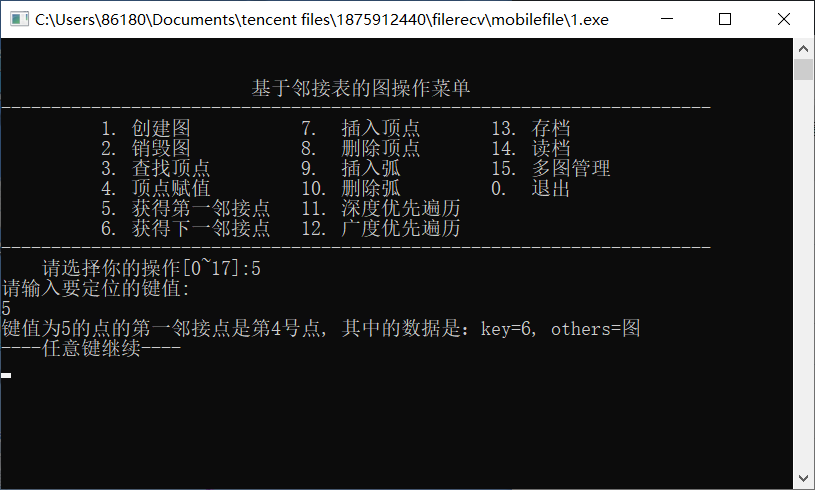


图4-18测试序号十二的程序运行结果

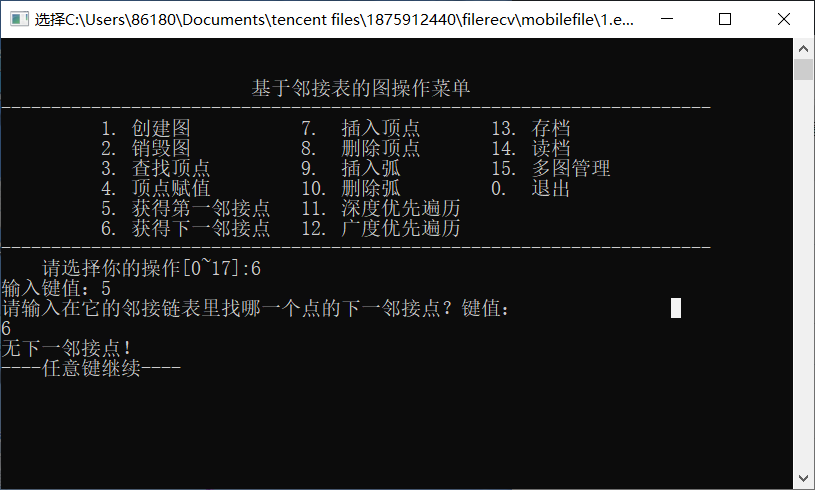


图4-19测试序号十三的程序运行结果

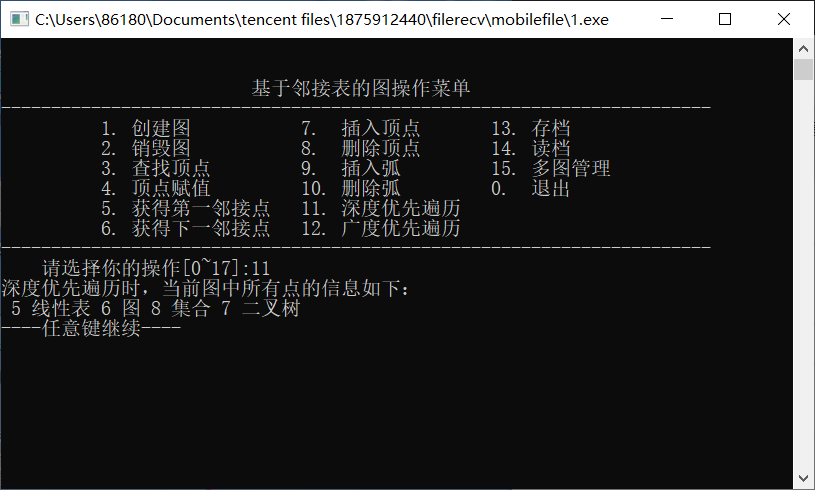


图4-20测试序号十四的程序运行结果

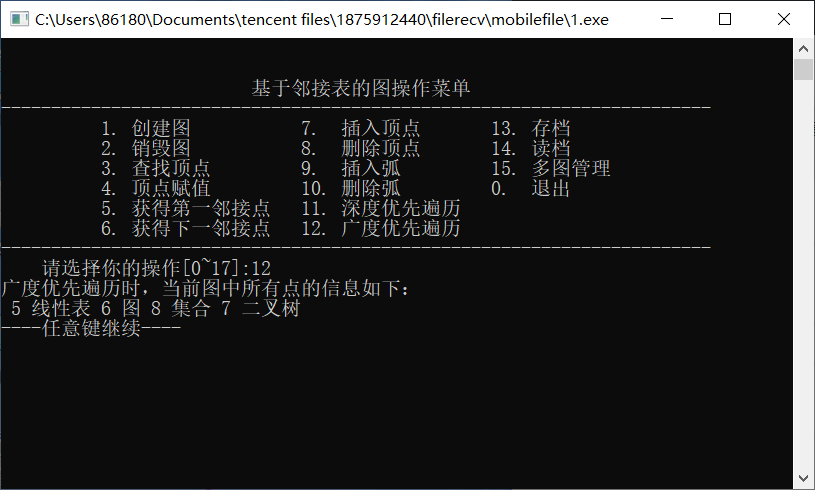


图4-21测试序号十五的程序运行结果

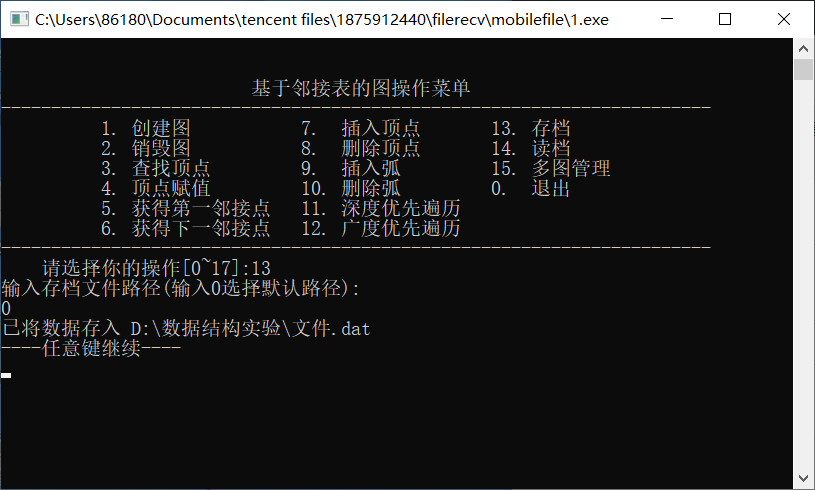


图4-22测试序号十六的程序运行结果

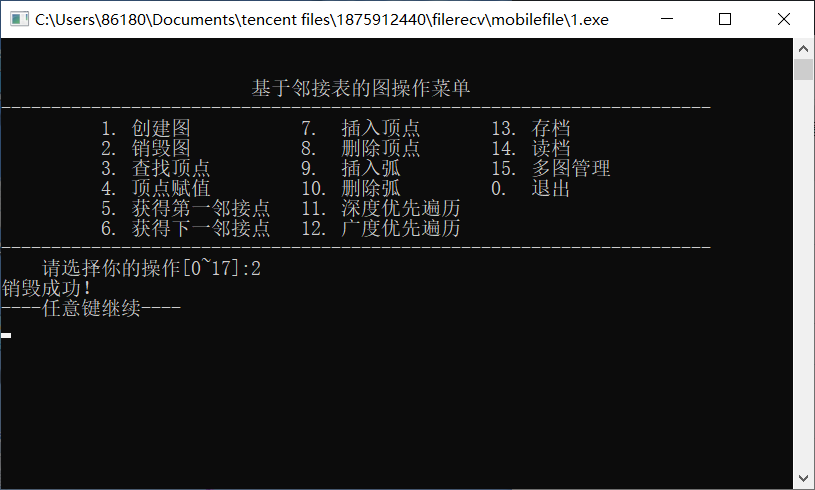


图4-23测试序号十七的程序运行结果

以下为多重图操作的系统测试的操作步骤对应的表格：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 多重图的系统操作 | 参数输入 | 程序输出 |
| 18 | 初始化多重图 | 15 1 | 初始化成功！ |
| 19 | 向多重图中新增名称为1的图 | 2 1 1 0 | 图"1"存储成功！ |
| 20 | 向多重图中的第二位新增名称为2的图 | 2 2 2 0 | 图"2"存储成功！ |
| 21 | 对于多重图中的名称为1的图进行修改，读取文件“1”中的数据 | 5 1 14 1 | 数据已从 1 中读出 |
| 22 | 深度优先遍历图 | 11 | 深度优先遍历时，当前图中所有点的信息如下：  5 线性表 7 二叉树 8 集合 6 无向图 |
| 23 | 在图中新增结点键值为9，内容为有向图的结点 | 7 9 有向图 | 添加成功！ |
| 24 | 新增结点5 9之间的弧 | 9 5 9 | 添加成功！ |
| 25 | 广度优先遍历图 | 12 | 广度优先遍历时，当前图中所有点的信息如下：  5 线性表 9 有向图 7 二叉树 6 无向图 8 集合 |
| 26 | 将图中的数据存到文件“2”中 | 13 2 | 已将数据存入 2 |
| 27 | 将图载入到多重图中 | 0 | 原主界面的图中数据已载入！ |
| 28 | 深度优先遍历多重图中的数据 | 6 1 | 当前集合中数据如下所示：  第1个图：  1:  5 线性表 9 有向图 7 二叉树 8 集合 6 无向图  第2个图：  2: |
| 29 | 修改第二个图，将文件“2”中的数据读取到该图 | 5 2 14 2 | 数据已从 2 中读出 |
| 30 | 将第二个图载入 | 0 | 原主界面的图中数据已载入！ |
| 31 | 删除第一个图 | 3 1 | 图"1"删除成功！ |
| 32 | 广度优先遍历多重图 | 6 2 | 当前集合中数据如下所示：  第1个图：  2:  5 线性表 9 有向图 7 二叉树 6 无向图 8 集合 |
| 33 | 清空集合 | 7 | 清空成功! |

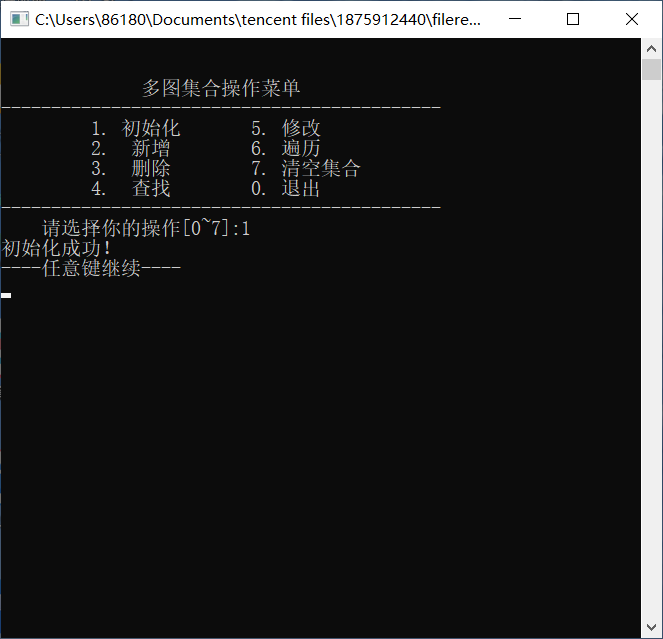


图4-24测试序号十八的程序运行结果

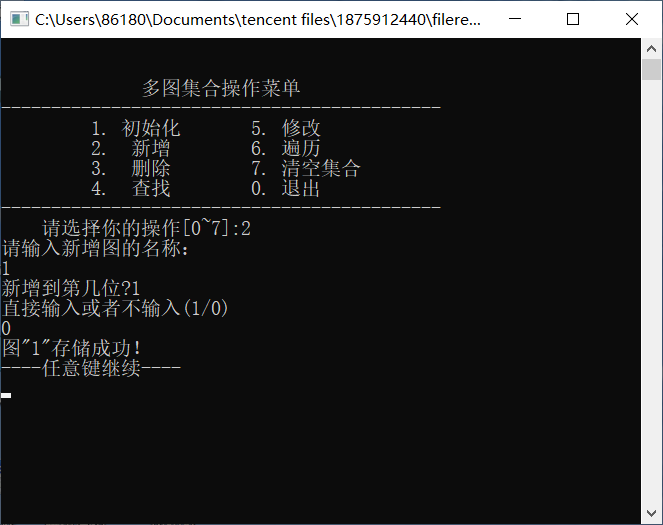


图4-25测试序号十九的程序运行结果

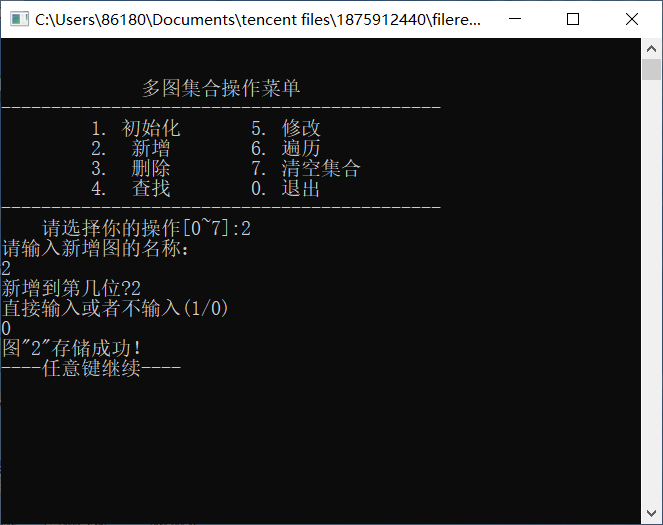


图4-26测试序号二十的程序运行结果



图4-27测试序号二十一的程序运行结果

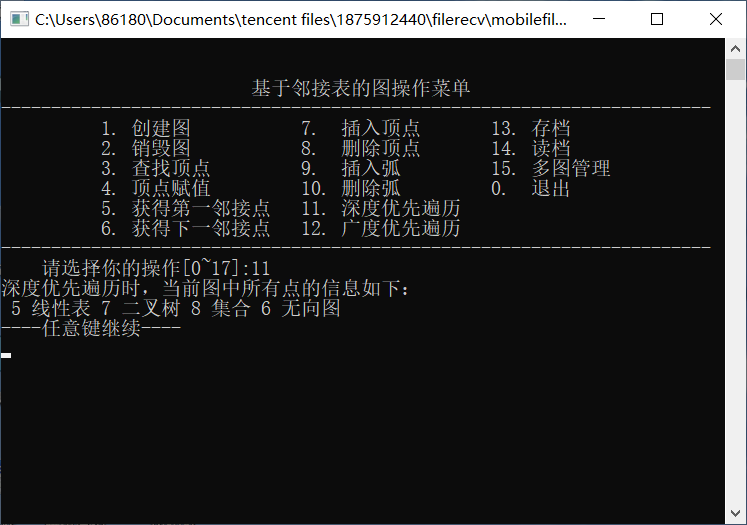


图4-28测试序号二十二的程序运行结果

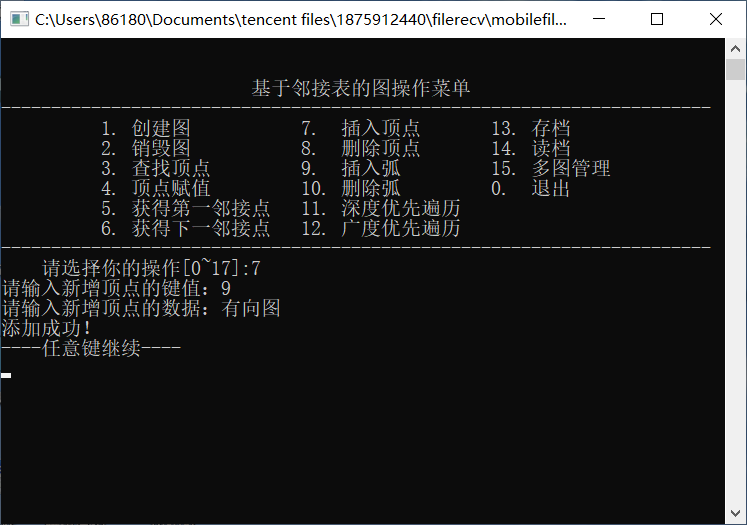


图4-29测试序号二十三的程序运行结果

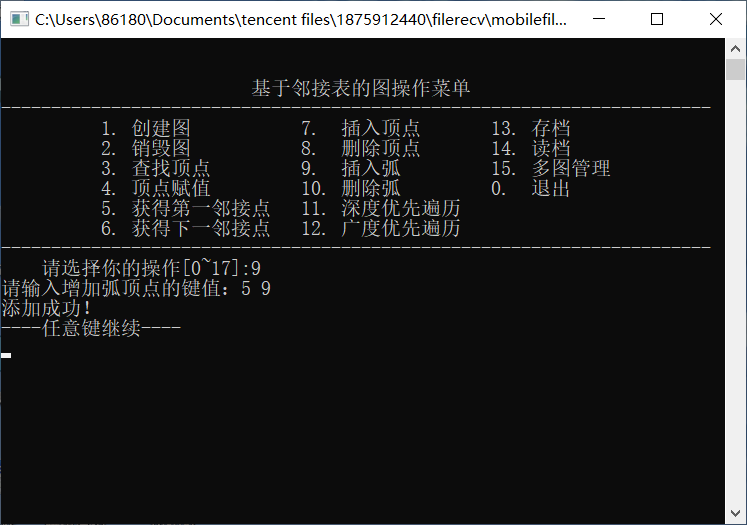


图4-30测试序号二十四的程序运行结果



图4-31测试序号二十五的程序运行结果



图4-32测试序号二十六的程序运行结果

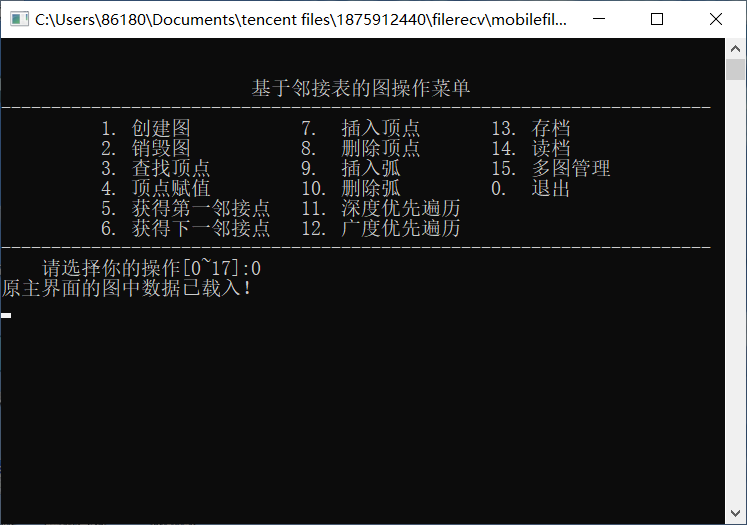


图4-33测试序号二十七的程序运行结果



图4-34测试序号二十八的程序运行结果

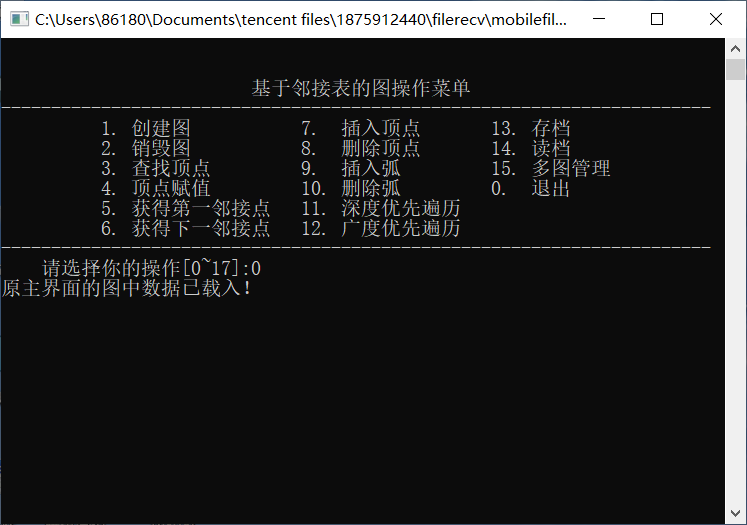


图4-35测试序号二十九的程序运行结果



图4-36测试序号三十的程序运行结果

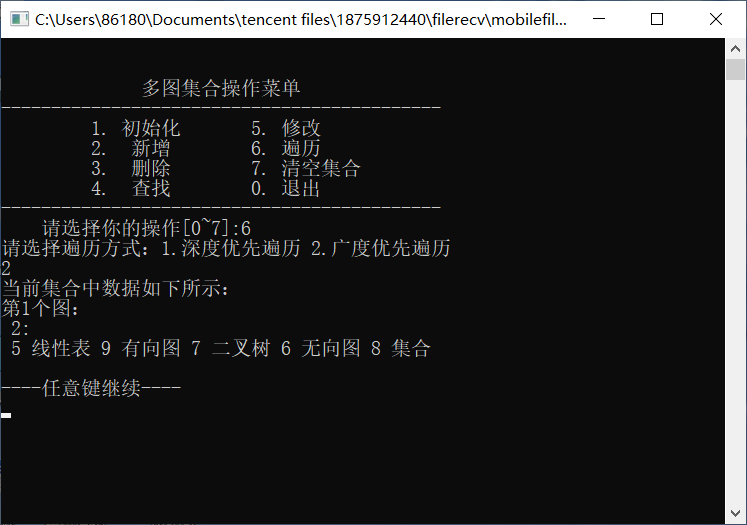


图4-37测试序号三十一的程序运行结果



图4-38测试序号三十二的程序运行结果

4.测试分析：

本次实验的多重图操作能够按照实验要求进行准确更改和输出，但是实验中的图形并不能够直接的表示出来显得不够直观，往往只能根据邻接表的内容进行手动的绘制，但是受限于试验人员的水平并不能实现这一操作。程序本身能够实现既定的多维操作，但仍然有局限性。

## 4.5 实验小结

本次实验过程较为曲折，在平台上的第一关，创建图时，就没有理解题目中的含义，以至于前前后后拖了好久问了许多人才勉强将第一关的函数写出，随后最大的障碍是删除节点，程序在一开始时并没有考虑健壮性，没有删除节点所对应的弧，后来的程序中才逐渐了解到邻接表对于无向图的意义，在邻接表中，顶点是作为弧的头结点存放在线性表中的；而与顶点相连的弧则以链表的形式串在顶点所在的头结点上。利用这种储存形式，很容易求得顶点和边的各种信息。

深度优先遍历运用到了递归的思想。首先访问结点v，接着以同样的方式访问结点v的邻接结点。从本质上而言，深度优先遍历类似于树的先序遍历。

广度优先遍历用到了队列作为辅助。先入先出式的遍历，从一点为中心，层层铺设开来。从本质上而言，广度优先遍历类似于树的层序遍历。

程序固有其不足之处，但是最不足的是，我没有能够将无向图具体的表现出来，无向图一直以抽象的形式存放在文件中，对于无向图的表示，应该能够调用画图软件进行绘出，这样后续的诸多操作也就会显得直观。

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

[2] [Larry Nyhoff](http://www.calvin.edu/~nyhl/index.html). [ADTs, Data Structures, and Problem Solving with C++.](http://vig.prenhall.com/catalog/academic/product/0,1144,0131409093,00.html)Second Edition, [Calvin College](http://cs.calvin.edu/), 2005

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define FAIL -3//预定义常量

typedef int status;//函数状态结果标志

typedef int ElemType; //数据元素类型定义

#define LIST\_INIT\_SIZE 100//线性表存储结构的最大分配量

#define LISTINCREMENT 10//线性表的分配增量

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;//抽象数据元素

int length;//表长

int listsize;//存储容量

}SqList;

typedef struct{ //线性表的管理表定义

struct { char name[30];

SqList L;

} elem[10];

int length;

int listsize;

}LISTS;

status InitList(SqList& L); //线性表初始化

status InitInput(SqList &L); //线性表输入数据

status DestroyList(SqList& L); //销毁线性表

status ClearList(SqList&L); //清空线性表

status ListEmpty(SqList L); //线性表判空 return status

int ListLength(SqList L); //求表长 return int

status GetElem(SqList L,int i,ElemType& e); //获取指定位置 i 处元素 e

status LocateElem(SqList L,ElemType e); //由元素内容e 查找元素位置return status

status PriorElem(SqList L,ElemType cur,ElemType &pre\_e);//由当前元素内容cur 获取其前驱元素pre\_e

status NextElem(SqList L,ElemType cur,ElemType &next\_e);//由当前元素内容cur 获取其后继元素next\_e

status ListInsert(SqList&L,int i,ElemType e); //向线性表指定位置 i 前插入元素e

status ListDelete(SqList&L,int i,ElemType& e); //删除位置 i 处元素存入 e 中

status ListTrabverse(SqList L); //遍历当前线性表

status SaveList(SqList L,char FileName[]); //当前线性表存档

status LoadList(SqList &L,char FileName[]); //当前线性表读档

status InitLists(LISTS &Lists); //线性表集合初始化

status AddList(LISTS &Lists, char ListName[]); //向集合中添加名为 ListsName 的线性表

status RemoveList(LISTS &Lists,char ListName[]);//删除集合中名为 ListsName 的线性表

int LocateList(LISTS Lists,char ListName[]); //在当前集合中查找名为 ListsName 的线性表

status ListsTraverse(LISTS Lists); //遍历当前集合

status SaveLists(LISTS Lists, char \*FileName); //线性表集合存档

status LoadLists(LISTS &Lists, char \*FileName); //线性表集合读档 （有问题）

status ChangeList(SqList &L); //修改集合中指定线性表

status EditList(LISTS Lists,SqList &L);

status SaveEdit(LISTS& Lists, SqList L) ;

char\* SaveFile = "D:\\数据结构实验\\文件.dat";

char\* LoadFile = "D:\\数据结构实验\\文件.dat";

char\* SaveFile1 = "D:\\数据结构实验\\文件.dat";

char\* LoadFile1 = "D:\\数据结构实验\\文件.dat";

/\*--------------------------------------------\*/

int main(){

SqList L;

int op=1; int flag = 0;

int j;

L.elem = NULL;

LISTS Lists;

Lists.elem[0].L.elem = NULL;

SqList temp ;

SqList \*L\_ope;

temp.elem = NULL;

while(op){

system("cls"); printf("\n\n");

printf(" 基于顺序存储的线性表操作菜单\n");

printf("--------------------------------------------------------------------------\n");

printf(" 1. 初始化线性表 7. 查找元素 13. 保存线性表 19. 将集合中的线性表载入\n");

printf(" 2. 摧毁线性表 8. 查找前驱元素 14. 读取线性表 20. 删除集合中线性表\n");

printf(" 3. 清空线性表 9. 查找后继元素 15. 初始化线性表集合 21. 将线性表存储到集合中\n");

printf(" 4. 线性表判空 10. 插入元素 16. 新增集合中的线性表 0. 退出\n");

printf(" 5. 线性表表长 11. 删除元素 17. 查找集合中的线性表\n");

printf(" 6. 由位置求元素 12. 遍历线性表 18. 遍历线性表集合\n");

printf("--------------------------------------------------------------------------\n");

if(op)

printf(" 请选择你的操作[0~21]:");

scanf("%d",&op);

switch(op){

case 1:

if(InitList(L)==OK)

{

printf("线性表创建成功！\n");

}

else printf("线性表创建失败！\n");

getchar();getchar();

break;

case 2:

j = DestroyList(L);

if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else printf("销毁成功！\n");

getchar();getchar();

break;

case 3:

j = ClearList(L);

if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else printf("清空成功！\n");

getchar();getchar();

break;

case 4:

j = ListEmpty(L);

if(j == INFEASIBLE) printf("当前线性表不存在\n");

else if(j) printf("当前线性表为空\n");

else printf("当前线性表不为空\n");

getchar();getchar();

break;

case 5:j = ListLength(L);

if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else printf("当前线性表长度为%d\n",j);

getchar();getchar();

break;

case 6:

if(L.elem==NULL)

{

printf("当前线性表不存在！\n");

getchar();getchar();

break;

}

int i, e;

printf("查找第几位元素？\n");

scanf("%d",&i);

j = GetElem(L, i, e);

if(j == ERROR) printf("查找位置不合法！\n");

else printf("当前线性表中第%d个元素为%d\n", i, e);

getchar();getchar();

break;

case 7:

printf("输入所需查找的元素内容\n");

scanf("%d", &e);

j = LocateElem(L, e);

if(j == ERROR) printf("表中不存在元素 %d\n",e);

else if(j == INFEASIBLE) printf("当前线性表为不存在！\n");

else printf("元素%d位于线性表中第%d个位置\n", e, j);

getchar();getchar();

break;

case 8:

int prior;

printf("输入当前元素\n");

scanf("%d", &e);

j = PriorElem(L, e, prior);

if(j == ERROR) printf("此元素无前驱！\n");

else if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else printf("当前元素前驱为%d\n", prior);

getchar();getchar();

break;

case 9:

int next;

printf("输入当前元素\n");

scanf("%d", &e);

j = NextElem(L, e, next);

if(j == ERROR) printf("此元素无后继！\n");

else if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else printf("当前元素后继为%d\n", next);

getchar();getchar();

break;

case 10:

printf("输入加入元素\n");

scanf("%d", &e);

printf("想插入第几个元素前？\n");

scanf("%d", &i);

j = ListInsert(L, i, e);

if(j == ERROR) printf("插入位置非法！\n");

else if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else

{

printf("插入成功！\n");

if(!ListTrabverse(L))

printf("当前线性表不存在！\n");

}

getchar();getchar();

break;

case 11:

printf("删除第几个元素？\n");

scanf("%d",&i);

j = ListDelete(L, i, e);

if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else if(j == ERROR) printf("删除位置非法！\n");

else

{

printf("删除成功！\n");

if(!ListTrabverse(L))

printf("当前线性表不存在！\n");

}

getchar();getchar();

break;

case 12:

if(ListTrabverse(L) == INFEASIBLE) printf("当前线性表不存在！\n");

getchar();getchar();

break;

case 13:

if(SaveList(L, SaveFile) == INFEASIBLE) printf("当前线性表不存在！\n");

else printf("已将数据存入 %s\n",SaveFile);

getchar();getchar();

break;

case 14:

if((j = LoadList(L, LoadFile)) == ERROR) printf("当前线性表中已有数据！\n");

else if(j == OK) printf("数据已从 %s 中读出\n",LoadFile);

else if(j == INFEASIBLE) printf("当前线性表不存在！请先初始化！\n");

else if(j == FAIL) printf("存档文件建立失败！\n");

getchar();getchar();

break;

case 15:

if(InitLists(Lists) == INFEASIBLE) printf("当前线性表集合已被初始化！\n");

else printf("初始化成功！\n");

getchar();getchar();

break;

case 16:

if(Lists.elem[0].L.elem == NULL)

{

printf("当前线性表集合不存在，请先初始化！\n");

getchar();getchar();

break;

}

char ListName[30];

printf("请输入新增线性表的名称：\n");

scanf("%s",ListName);

if(AddList(Lists, ListName) == OVERFLOW) printf("线性表集合已满，添加失败！\n");

else printf("线性表\"%s\"存储成功！\n", ListName);

getchar();getchar();

break;

case 20:

if(Lists.elem[0].L.elem == NULL)

{

printf("当前线性表集合不存在，请先初始化！\n");

getchar();getchar();

break;

}

printf("请输入所要删除的线性表名称：\n");

scanf("%s",&ListName);

if(RemoveList(Lists, ListName))

{

printf("线性表%s删除成功！\n",ListName);

}

else

{

printf("查无此表！\n");

}

getchar();getchar();

break;

case 17:

if(Lists.elem[0].L.elem == NULL)

{

printf("当前线性表集合不存在，请先初始化！\n");

getchar();getchar();

break;

}

printf("请输入要定位的表名:\n");

scanf("%s",ListName);

j = LocateList(Lists, ListName);

if(j == 0)

{

printf("查无此表！\n");

getchar();getchar();

break;

}

else

{

ListTrabverse(Lists.elem[j - 1].L);

}

getchar();getchar();

break;

case 18:

if((j = ListsTraverse(Lists)) == 0) printf("当前线性表集合为空！\n");

else if(j == INFEASIBLE) printf("当前线性表集合不存在！\n");

getchar();getchar();

break;

case 19:

if(Lists.elem[0].L.elem == NULL)

{

printf("当前线性表集合不存在，请先初始化！\n");

getchar();getchar();

break;

}

EditList(Lists,L);

getchar();getchar();

break;

case 21:

if(Lists.elem[0].L.elem == NULL)

{

printf("当前线性表集合不存在，请先初始化！\n");

getchar();getchar();

break;

}

SaveEdit(Lists,L);

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

return 0;

}//end of main()

status SaveEdit(LISTS& Lists, SqList L)

{

//更新管理系统中的线性表

char ListName[30];

printf("请输入载入的线性表名称");

scanf("%s",ListName);

int k=LocateList(Lists,ListName);

if(!k)

{

printf("未找到该线性表");

return ERROR;

}

if(L.elem==NULL){printf("当前线性表不存在\n");return ERROR;}

Lists.elem[k - 1].L=L;

printf("%s更新成功！", ListName);

return OK;

}

status EditList(LISTS Lists,SqList &L)

{

char ListName[30];

printf("请输入载入的线性表名称");

scanf("%s",ListName);

int k=LocateList(Lists,ListName);

if(!k)

{

printf("未找到该线性表");

return ERROR;

}

InitList(L);

L=(Lists.elem[k-1].L);

printf("线性表载入成功");

return OK;

}

status InitList(SqList& L)

// 线性表L不存在，构造一个空的线性表，返回OK，否则返回INFEASIBLE。

{

if(!L.elem)

{

L.elem = (ElemType\*)malloc(LIST\_INIT\_SIZE \* sizeof(ElemType));

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

return OK;

}

else

{

return INFEASIBLE;

}

}

status InitInput(SqList &L)

{

int ord, temp;

printf("输入数据按1，否则按0\n");

scanf("%d",&ord);

if(ord == 0)

{

return 0;

}

else

{

printf("请输入一组数据，0为结束标志\n");

scanf("%d",&temp);

while(temp != 0)

{

if(L.length > L.listsize)

{

L.elem = (ElemType \*)realloc(L.elem, (L.listsize + LISTINCREMENT) \* sizeof(ElemType));

L.listsize += LISTINCREMENT;

}

ListInsert(L, L.length + 1, temp);

scanf("%d",&temp);

}

printf("输入成功！\n");

return OK;

}

}

status DestroyList(SqList& L)

// 如果线性表L存在，销毁线性表L，释放数据元素的空间，返回OK，否则返回INFEASIBLE。

{

if(!L.elem)

{

return INFEASIBLE;

}

else

{

free(L.elem);

L.length = 0;

L.listsize = 0;

L.elem = NULL;

return OK;

}

}

status ClearList(SqList& L)

// 如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回INFEASIBLE。

{

if(!L.elem)

{

return INFEASIBLE;

}

else

{

free(L.elem);

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

L.elem = (ElemType\*)malloc(LIST\_INIT\_SIZE \* sizeof(ElemType));

return OK;

}

}

status ListEmpty(SqList L)

// 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。

{

if(!L.elem)

{

return INFEASIBLE;

}

else

{

if(L.length == 0)

{

return TRUE;

}

else

{

return FALSE;

}

}

}

int ListLength(SqList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

if(!L.elem)

{

return INFEASIBLE;

}

else

{

return L.length;

}

}

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

// 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。

{

if(!L.elem)

{

return INFEASIBLE;

}

else

{

if(i > L.length || i < 1)

{

return ERROR;

}

else

{

e = L.elem[i - 1];

return OK;

}

}

}

status LocateElem(SqList L,ElemType e)

// 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OK；如果e不存在，返回ERROR；当线性表L不存在时，返回INFEASIBLE。

{

if(!L.elem)

{

return INFEASIBLE;

}

else

{

for(int i = 0; i < L.length; i++)

{

if(L.elem[i] == e)

{

return i + 1;

}

}

return ERROR;

}

}

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

// 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。

{

if(!L.elem)

{

return INFEASIBLE;

}

else

{

for(int i = 0; i < L.length; i++)

{

if(L.elem[i] == e)

{

if(i)

{

pre = L.elem[i - 1];

return OK;

}

else

{

return ERROR;

}

}

}

return ERROR;

}

}

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

// 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。

{

if(!L.elem)

{

return INFEASIBLE;

}

else

{

for(int i = 0; i < L.length; i++)

{

if(L.elem[i] == e)

{

if(i == L.length - 1)

{

return ERROR;

}

else

{

next = L.elem[i + 1];

return OK;

}

}

}

return ERROR;

}

}

status ListInsert(SqList &L,int i,ElemType e)

// 如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回OK；当插入位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

if(!L.elem)

{

return INFEASIBLE;

}

else

{

if(i > L.length + 1 || i < 1)

{

return ERROR;

}

else

{

if(L.listsize < L.length + 1)

{

L.elem = (ElemType\*)realloc(L.elem, (L.listsize + LISTINCREMENT) \* sizeof(ElemType));

if(L.elem)

{

L.listsize += LISTINCREMENT;

}

else

{

return OVERFLOW;

}

}

for(int j = L.length; j >= i; j--)

{

L.elem[j] = L.elem[j - 1];

}

L.elem[i - 1] = e;

L.length++;

return OK;

}

}

}

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

// 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

if(!L.elem)

{

return INFEASIBLE;

}

else

{

if(i < 1 || i > L.length)

{

return ERROR;

}

else

{

e = L.elem[i - 1];

for(int j = i; j < L.length; j++)

{

L.elem[j - 1] = L.elem[j];

}

L.length--;

return OK;

}

}

}

status ListTrabverse(SqList L)

// 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，返回OK；如果线性表L不存在，返回INFEASIBLE。

{

if(!L.elem)

{

return INFEASIBLE;

}

else

{

printf("当前线性表中数据如下:\n");

for(int i = 0; i < L.length; i++)

{

printf("%d",L.elem[i]);

if(i != L.length - 1)

{

printf(" ");

}

else

{

printf("\n");

}

}

return OK;

}

}

status SaveList(SqList L,char filename[])

// 如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

{

FILE \*fp;

if(L.elem==NULL)

{

printf("未初始化线性表！\n");

return -1;

}

if ((fp=fopen(filename,"wb"))==NULL)

{

printf("File open error\n ");

return -1;

}

fwrite(L.elem,sizeof(ElemType),L.length,fp);

//这里是1次性写入，对于其它物理结构，可通过遍历，逐个访问数据元素

//并写入到文件中。也可以先写入表长，再写入全部元素，这样读入会更方便

fclose(fp);

return OK;

}

status LoadList(SqList &L,char filename[])

// 如果线性表L为非空，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

{

FILE \*fp;

if(L.elem==NULL)

{

InitList(L);

}

if(L.length)

{

printf("线性表不为空！\n");

return -1;

}

if ((fp=fopen(filename,"rb"))==NULL)

{

printf("File open error\n ");

return -1;

}

while(fread(&L.elem[L.length],sizeof(ElemType),1,fp))

L.length++;

//这里从文件中逐个读取数据元素恢复顺序表，对于不同的物理结构，可通过读//取的数据元素恢复内存中的物理结构。

fclose(fp);

return OK;

}

status InitLists(LISTS &Lists)

//线性表集合初始化

{

if(Lists.elem[0].L.elem != NULL)

{

return INFEASIBLE;

}

else

{

Lists.length = 0;

Lists.listsize = 10;

InitList(Lists.elem[0].L);

return OK;

}

}

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

// 在Lists中增加一个名称为ListName的空线性表后输入线性表数据

{

if(Lists.length == Lists.listsize) //集合溢出判断

{

return OVERFLOW;

}

Lists.elem[Lists.length].L.elem = (ElemType\*)malloc(LIST\_INIT\_SIZE \* sizeof(ElemType));

Lists.elem[Lists.length].L.length = 0;

Lists.elem[Lists.length].L.listsize = LIST\_INIT\_SIZE;

strcpy(Lists.elem[Lists.length].name ,ListName);

InitInput(Lists.elem[Lists.length].L);

Lists.length++;

return OK;

}

status RemoveList(LISTS &Lists,char ListName[])

// Lists中删除一个名称为ListName的线性表

{

for(int i = 0; i < Lists.length; i++)

{

if(!strcmp(ListName,Lists.elem[i].name))

{

for(; i < Lists.length - 1; i++)

{

Lists.elem[i] = Lists.elem[i + 1];

}

Lists.length--;

return OK;

}

}

return 0;

}

int LocateList(LISTS Lists,char ListName[])

// 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回0

{

for(int i = 0; i < Lists.length; i++)

{

if(!strcmp(ListName, Lists.elem[i].name))

{

return i + 1;

}

}

return 0;

}

status ListsTraverse(LISTS Lists)

//遍历线性表集合，如果集合为空返回0，否则将集合中线性表的全部数据输出，并返回OK

{

if(Lists.elem[0].L.elem == NULL)

{

return INFEASIBLE;

}

if(Lists.length == 0)

{

return 0;

}

printf("当前线性表集合中共包含以下%d个线性表：\n", Lists.length);

for(int i = 0; i < Lists.length; i++)

{

printf("%s\n",Lists.elem[i].name);

if(Lists.elem[i].L.length == 0)

{

printf("空\n");

}

for(int j = 0; j < Lists.elem[i].L.length; j++)

{

printf("%d",Lists.elem[i].L.elem[j]);

if(j != Lists.elem[i].L.length - 1)

{

printf(" ");

}

else

{

printf("\n");

}

}

}

return OK;

}

status SaveLists(LISTS Lists, char \*FileName)

//将集合中数据存入文件 FileName 中，若线性表集合为空或不存在时，返回INFEASIBLE,否则返回OK

{

if(Lists.elem == NULL || Lists.length == 0)

{

return INFEASIBLE;

}

FILE \*fp = fopen(FileName,"w+");

char ch = '\0';

for(int i = 0; i < Lists.length; i++)

{

fprintf(fp, "%s", Lists.elem[i].name);

fprintf(fp,"%c",'\n');

for(int j = 0; j < Lists.elem[i].L.length; j++)

{

fprintf(fp, "%d", Lists.elem[i].L.elem[j]);

if(j == Lists.elem[i].L.length - 1)

{

fprintf(fp, "%c", '\n');

}

else

{

fprintf(fp, "%c", ' ');

}

}

}

fclose(fp);

return OK;

}

# 附录B 基于链式存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef int ElemType;

typedef struct LNode{ //单链表（链式结构）结点的定义

ElemType data;

struct LNode \*next;

}LNode,\*LinkList;

typedef struct { //线性表的管理表定义

struct {

char name[30];

LinkList L;

} elem[10];

int length;

int listsize;

}LISTS;

/\*-------------线性表操作---------------\*/

status InitList(LinkList &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。

status DestroyList(LinkList &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。

status ClearList(LinkList &L);// 如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回INFEASIBLE。

status ListEmpty(LinkList L);// 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。

status ListLength(LinkList L);// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

status GetElem(LinkList L,int i,ElemType &e);// 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。

status LocateElem(LinkList L,ElemType e);// 如果线性表L存在，查找元素e在线性表L中的位置序号；如果e不存在，返回ERROR；当线性表L不存在时，返回INFEASIBLE。

status PriorElem(LinkList L,ElemType e,ElemType &pre);// 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。

status NextElem(LinkList L,ElemType e,ElemType &next);// 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。

status ListInsert(LinkList &L,int i,ElemType e);// 如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回OK；当插入位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

status ListDelete(LinkList &L,int i,ElemType &e);// 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

status ListTraverse(LinkList L);// 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，返回OK；如果线性表L不存在，返回INFEASIBLE。

status SaveList(LinkList L,char FileName[]);// 如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

status LoadList(LinkList &L,char FileName[]);// 如果线性表L不存在，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

/\*-------------线性表操作---------------\*/

/\*-------------线性表集合操作---------------\*/

status InitInput(LinkList L);

status AddList(LISTS& Lists, char ListName[]);// 只需要在Lists中增加一个名称为ListName的空线性表，线性表数据由后台测试程序插入。

status RemoveList(LISTS& Lists, char ListName[]);// Lists中删除一个名称为ListName的线性表

status LocateList(LISTS Lists, char ListName[]);// 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回0

status SaveEdit(LISTS& Lists, LinkList L, char ListName[]);//更新管理系统中的线性表

status EditList(LISTS Lists, LinkList& L, char ListName[]);

status ListGet(LISTS Ls,int i);//遍历指定线性表

status TraverseLists(LISTS Lists);//遍历所有线性表

/\*-------------线性表集合操作---------------\*/

int main(){

int op=1;

LinkList L;//链式线性表的定义L

L=NULL;

int e,i,j;

char filename[100];

char Listname[30];

LISTS Ls;

Ls.length=0,Ls.listsize=10;

while(op){

system("cls"); printf("\n\n");

printf(" 基于顺序存储的链式表操作菜单\n");

printf("--------------------------------------------------------------------------\n");

printf(" 1. 初始化链表 7. 查找元素 13. 保存链表 19. 将集合中的线性表载入\n");

printf(" 2. 摧毁链表 8. 查找前驱元素 14. 读取链表 20. 遍历集合中的线性表\n");

printf(" 3. 清空链表 9. 查找后继元素 15. 链表集合操作 21. 将线性表存储到集合中\n");

printf(" 4. 链表判空 10. 插入元素 16. 删除集合中的线性表 0. 退出\n");

printf(" 5. 链表表长 11. 删除元素 17. 定位集合中的线性表\n");

printf(" 6. 由位置求元素 12. 遍历链表 18. 遍历集合\n");

printf("--------------------------------------------------------------------------\n");

printf(" 请选择你的操作[0~15]:");

scanf("%d",&op);

switch(op){

case 1:

if((j=InitList(L))==OK) printf("线性表创建成功！\n");

else if(j==-1)printf("线性表已创建！\n");

else printf("线性表创建失败！\n");

getchar();getchar();

break;

case 2:

j = DestroyList(L);

if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else printf("销毁成功！\n");

getchar();getchar();

break;

case 3:

j=ClearList(L);

if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else printf("清除成功！\n");

getchar();getchar();

break;

case 4:

j=ListEmpty(L);

if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else if(j) printf("当前线性表为空\n");

else printf("当前线性表不为空\n");

getchar();getchar();

break;

case 5:

j=ListLength(L);

if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else printf("当前线性表长度为%d\n",j);

getchar();getchar();

break;

case 6:

printf("查找第几位元素？\n");

scanf("%d",&i);

j = GetElem(L,i,e);

if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else if(j == ERROR) printf("查找位置不合法！\n");

else printf("当前线性表中第%d个元素为%d\n", i, e);

getchar();getchar();

break;

case 7:

printf("请输入要查找的元素：\n");

scanf("%d",&e);

j=LocateElem(L,e);

if(j==-1) printf("当前线性表不存在！\n");

else if(j==0) printf("未找到该元素！\n");

else printf("元素%d位于线性表中第%d个位置\n", e, j);

getchar();getchar();

break;

case 8:

int cur,pre;

printf("输入当前元素\n");

scanf("%d", &cur);

j=PriorElem(L,cur,pre);

if(j==-1) printf("当前线性表不存在！\n");

else if(j==0) printf("该元素无前驱！\n");

else printf("元素%d的前驱是%d\n",cur,pre);

getchar();getchar();

break;

case 9:

int next;

printf("输入当前元素\n");

scanf("%d", &cur);

j=NextElem(L,cur,next);

if(j==-1) printf("当前线性表不存在！\n");

else if(j==0) printf("该元素无后继！\n");

else printf("元素%d的后继是%d\n",cur,next);

getchar();getchar();

break;

case 10:

printf("输入加入元素\n");

scanf("%d", &e);

printf("想插入第几个元素前？\n");

scanf("%d", &i);

j = ListInsert(L, i, e);

if(j == ERROR) printf("插入位置非法！\n");

else if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else

{

printf("插入成功！\n");

ListTraverse(L);

}

getchar();getchar();

break;

case 11:

printf("输入想删除的元素位置\n");

scanf("%d", &i);

j = ListDelete(L, i, e);

if(j == ERROR) printf("删除位置非法！\n");

else if(j == INFEASIBLE) printf("当前线性表不存在！\n");

else

{

printf("%d已删除成功！\n",e);

ListTraverse(L);

}

getchar();getchar();

break;

case 12:

if(ListTraverse(L)==-1) printf("当前线性表不存在！\n");

getchar();getchar();

break;

case 13:

printf("输入文件名:\n");

scanf("%s",filename);

if((j=SaveList(L,filename))!=-1)printf("保存到文件成功\n");

getchar();getchar();

break;

case 14:

printf("输入文件名:\n");

scanf("%s",filename);

if((j=LoadList(L,filename))!=-1)printf("从文件读取成功 \n");

getchar();getchar();

break;

case 15:

//新建管理系统中线性表

printf("请输入新增线性表的名称:");

scanf("%s", Listname);

if (AddList(Ls, Listname) == ERROR) {

int choice;

printf("线性表名称已存在,是否继续?(1:是,0:否)\n");

scanf("%d", &choice);

if (choice) {

RemoveList(Ls, Listname);

AddList(Ls, Listname);

printf("线性表\"%s\"存储成功！\n", Listname);

}

}

else printf("线性表\"%s\"存储成功！\n", Listname);

getchar(); getchar();

break;

case 16:

//删除管理系统中线性表

printf("请输入待删除线性表名称:");

scanf("%s", Listname);

if (RemoveList(Ls, Listname) == FALSE) printf("该线性表不存在\n");

else printf("删除成功!\n");

getchar(); getchar();

break;

case 17:

//查询管理系统中线性表

printf("请输入要定位的表名:\n");

scanf("%s", Listname);

if ((j = LocateList(Ls, Listname)) == 0) printf("该线性表不存在\n");

else printf("该线性表的次序为%d\n", j);

getchar(); getchar();

break;

case 18:

if(TraverseLists(Ls)==-1)printf("线性表集合为空\n");

getchar();getchar();

break;

case 19:

//编辑管理系统中线性表

printf("请输入待编辑线性表名称:");

scanf("%s", Listname);

EditList(Ls, L, Listname);

getchar(); getchar();

break;

case 20:

int i;

printf("请输入想要遍历的线性表序号：");

scanf("%d",&i);

if(ListGet(Ls,i)==-1)printf("序号不正确\n");

getchar();getchar();

break;

case 21:

//更新管理系统线性表

printf("请输入待更新的线性表名称:");

scanf("%s", Listname);

SaveEdit(Ls, L, Listname);

getchar(); getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

status InitList(LinkList &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。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L!=NULL)return INFEASIBLE;

L=(LinkList)malloc(sizeof(LNode));

L->next=NULL;

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status InitInput(LinkList L)

// 线性表不存在，返回INFEASIBLE；如果线性表为空表，输入数据；如果线性表不为空，返回ERROR

{

{

ElemType temp = 0;

printf("请输入一组数据(0结束):\n");

scanf("%d",&temp);

while(temp)

{

L->next = (LinkList)malloc(sizeof(LNode));

L = L->next;

L->data = temp;

scanf("%d",&temp);

}

L->next = NULL;

}

}

status DestroyList(LinkList &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。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL)return INFEASIBLE;

LinkList q=L,p;

while(p) //把链表中除头结点外的所有结点释放

{

p=q->next;

free(q);

q=p;

}

L=NULL;

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ClearList(LinkList &L)

// 如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL)return INFEASIBLE;

LinkList p=L->next,q;

while(p)

{

q=p->next;

free(p);

p=q;

}

L->next=NULL;

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListEmpty(LinkList L)

// 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。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL)return INFEASIBLE;

if(L->next==NULL)return TRUE;

else return FALSE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

int ListLength(LinkList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL)return INFEASIBLE;

int x=0;

L=L->next;

while(L)

{

x++;

L=L->next;

}

return x;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

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

// 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。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL)return INFEASIBLE;

if(i<1)return ERROR;

while(i>0)

{

L=L->next;

i--;

if(L==NULL)return ERROR;

}

e=L->data;

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status LocateElem(LinkList L,ElemType e)

// 如果线性表L存在，查找元素e在线性表L中的位置序号；如果e不存在，返回ERROR；当线性表L不存在时，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL)return INFEASIBLE;

int i=0;

while(L->next)

{

L=L->next;i++;

if(L->data==e)return i;

}

return ERROR;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

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

// 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。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL)return INFEASIBLE;

L=L->next;

if(L==NULL)return ERROR;

LinkList p=L;

L=L->next;

while(L)

{

if(L->data==e){pre=p->data;return OK;}

p=L;L=L->next;

}

return ERROR;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

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

// 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。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL)return INFEASIBLE;

L=L->next;

if(L==NULL)return ERROR;

while(L->next)

{

if(L->data==e){next=L->next->data;return OK;}

L=L->next;

}

return ERROR;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListInsert(LinkList &L,int i,ElemType e)

// 如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回OK；当插入位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL)return INFEASIBLE;

LinkList p=L;int j=1;

while(p&&j<i){p=p->next;j++;}

if(!p||j>i)return ERROR;

LinkList s=(LinkList)malloc(sizeof(LNode));

s->data=e;s->next=p->next;p->next=s;

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

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

// 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL)return INFEASIBLE;

LinkList p=L;int j=1;

while(p->next&&j<i){p=p->next;j++;}

if(!p->next||j>i)return ERROR;

LinkList q=p->next;p->next=q->next;

e=q->data;

free(q);

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListTraverse(LinkList L)

// 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，返回OK；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L==NULL)return INFEASIBLE;

printf("当前线性表中数据如下:\n");

while(L->next)

{

L=L->next;

printf("%d",L->data);if(L->next)printf(" ");

}

printf("\n");

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status SaveList(LinkList L,char FileName[])

// 如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 1 \*\*\*\*\*\*\*\*\*/

if(L==NULL){printf("未初始化线性表\n");return INFEASIBLE;}

FILE \*fp;

fp=fopen(FileName,"w");

if(fp==NULL){printf("文件打开失败\n");return INFEASIBLE;}

while(L->next)

{

L=L->next;

fprintf(fp,"%d ",L->data);

}

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 1 \*\*\*\*\*\*\*\*\*\*/

}

status LoadList(LinkList &L,char FileName[])

// 如果线性表L不存在，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 2 \*\*\*\*\*\*\*\*\*/

LinkList p;

if(L!=NULL){printf("线性表已存在\n");return INFEASIBLE;}

FILE \*fp;

L=(LinkList)malloc(sizeof(LNode));

p=L;

fp=fopen(FileName,"r");

if(fp==NULL){printf("文件打开失败\n");return INFEASIBLE;}

int e;

while(fscanf(fp,"%d ",&e)!=EOF)

{

p->next=(LinkList)malloc(sizeof(LNode));

p=p->next;

p->data=e;

}

p->next=NULL;

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 2 \*\*\*\*\*\*\*\*\*\*/

}

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

// 只需要在Lists中增加一个名称为ListName的空线性表，线性表数据由后台测试程序插入。

{

int k = LocateList(Lists, ListName);

if (k)

return ERROR;//该名称已存在

strcpy(Lists.elem[Lists.length].name, ListName);

Lists.elem[Lists.length].L = NULL;

InitList(Lists.elem[Lists.length].L);

InitInput(Lists.elem[Lists.length].L);

Lists.length++;

return OK;

}

status RemoveList(LISTS& Lists, char ListName[])

// Lists中删除一个名称为ListName的线性表

{

int i;

for (i = 0; i < Lists.length; i++) {

if (!strcmp(Lists.elem[i].name, ListName)) break;

}

if (i == Lists.length) return FALSE;//未找到

DestroyList(Lists.elem[i].L);

for (; i < Lists.length - 1; i++) {//移动管理系统中的元素

Lists.elem[i] = Lists.elem[i + 1];

}

Lists.length--;//管理系统中线性表个数减一

return OK;

}

status LocateList(LISTS Lists, char ListName[])

// 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回0

{

int i;

for (i = 0; i < Lists.length; i++) {

if (!strcmp(Lists.elem[i].name, ListName)) break;

}

if (i == Lists.length) return 0;//未找到

return i + 1;//逻辑序号为下标+1

}

status SaveEdit(LISTS& Lists, LinkList L, char ListName[])

{

//更新管理系统中的线性表

int k = LocateList(Lists, ListName);

if (!k) {

printf("未找到文件名为%s的线性表\n", ListName);

return ERROR;

}

if(L==NULL){printf("当前线性表不存在\n");return ERROR;}

ClearList(Lists.elem[k - 1].L);

LinkList temp, p, q;

p = L->next;

q = Lists.elem[k-1].L;

while (p)

{

temp = (LinkList)malloc(sizeof(LNode));

temp->data = p->data;

q->next = temp;

q = q->next;

p = p->next;

}

q->next = NULL;

printf("%s更新成功！", ListName);

return OK;

}

status EditList(LISTS Lists, LinkList& L, char ListName[]) {

//将管理表中名称为ListName的线性表加载到L

if (!ListEmpty(L)) {//当前编辑区线性表非空

int choice;

printf("现在线性表非空,继续操作可能会丢失数据，是否继续？(1:是,0:否):\n");

scanf("%d", &choice);

if (!choice) return ERROR;

else DestroyList(L);

}

int k = LocateList(Lists, ListName);

if (!k) {

printf("\n未找到文件名为%s的线性表\n", ListName);

return ERROR;

}

InitList(L);

LinkList q = L;

LinkList p = Lists.elem[k - 1].L->next;

LinkList temp;

while (p)

{

temp = (LinkList)malloc(sizeof(LNode));

temp->data = p->data;

q->next = temp;

q = q->next;

p = p->next;

}

q->next = NULL;

printf("\n现在可对%s进行编辑,记得使用保存编辑\n", ListName);

return OK;

}

status ListGet(LISTS Ls,int i)

{

if(i>Ls.length||i<1)return -1;

printf("%s\n",Ls.elem[i-1].name);

ListTraverse(Ls.elem[i-1].L);

}

status TraverseLists(LISTS Ls)

{

if(Ls.length==0)return -1;

else

{

printf("当前线性表集合中共包含以下%d个线性表：\n", Ls.length);

for(int i=0;i<Ls.length;i++)

{

printf("%s\n",Ls.elem[i].name);

ListTraverse(Ls.elem[i].L);

}

}

return OK;

}

# 附录C 基于二叉链表二叉树实现的源程序

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

#include "ctype.h"

#include <conio.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LEFT 4

#define RIGHT 6

#define max(a,b) ((a>b)?a:b)

//用于按层遍历中definition扩容

#define INIT\_DEF\_SIZE 64

#define INCREMENT 16

//用于森林扩容

#define INIT\_SIZE 100

typedef int status;

typedef int KeyType;

typedef struct {

KeyType key;

char others[20];

} TElemType; //二叉树结点类型定义

typedef struct BiTNode{ //二叉链表结点的定义

TElemType data;

struct BiTNode \*lchild,\*rchild;

} BiTNode, \*BiTree;

typedef struct {

int pos;

TElemType data;

} DEF;

typedef struct { //二叉树的结点definition，包含pos以及pos对应的结点地址，仅用于按层遍历

int pos;

BiTNode \*node;

} def;

typedef struct { //森林中每棵树的定义

BiTree T;

char name[30];

} Tree, \*pTree;

typedef struct { //森林定义(线性表结构存储)

pTree \*tree;

int length;

int size;

} FOREST, \*FST;

status InitBiTree(BiTree &T);

status CreateBiTree(BiTree &T,DEF definition[]);

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

/\*根据先序遍历序列definition创建二叉树 T，definition中如有重复则返回ERROR，否则返回OK\*/

int BiTreeEmpty(BiTree T);//二叉树判空

status ClearBiTree(BiTree &T); //将二叉树设置成空，并删除所有结点，释放结点空间

bool IsRepeat(TElemType definition[]); //判断先序序列definition中是否有重复key值，用于创建树

int BiTreeDepth(BiTree T); //求二叉树T的深度

BiTNode\* LocateNode(BiTree T,KeyType e); //查找结点

status Assign(BiTree &T,KeyType e,TElemType value); //实现结点赋值。

BiTNode\* GetSibling(BiTree T,KeyType e); //实现获得兄弟结点

status InsertNode(BiTree &T,KeyType e,int LR,TElemType c); //实现插入节点，在e的LR(左右)孩子处插入c

status ParentNode(BiTree T, KeyType e, BiTree &p);

/\*寻找key值为e元素的双亲结点，由参数p带出函数，返回OK表示删除结点为根节点，

返回LEFT表示e为p的左孩子，返回RIGHT表示e为p的右孩子，返回ERROR表示没有此结点\*/

BiTNode\* RightMostNode(BiTree T); //寻找 T 最右端结点，用于删除

status DeleteNode(BiTree &T,KeyType e); //删除结点。

status PreOrderTraverse(BiTree T,void (\*visit)(BiTree)); //先序遍历二叉树T(递归)

status InOrderTraverse(BiTree T,void (\*visit)(BiTree)); //先序遍历二叉树T(非递归)

status PostOrderTraverse(BiTree T,void (\*visit)(BiTree)); //后序遍历二叉树T(递归)

def\* CreateDef(BiTree T, int &length); //按照结点位置构造二叉树的definition

void DefOrder(def \*definition, int length); //按结点位置对definition排序

status LevelOrderTraverse(BiTree T,void (\*visit)(BiTree)); //按层遍历二叉树T

//status SaveBiTree(BiTree T, FILE \*fp);

status SaveBiTree(BiTree T,FILE \*fp,int i);

status LoadBiTree(BiTree &T, char FileName[]); //读档

status TreeLikeTraverse(BiTree T, void (\*visit)(BiTree)); //树形遍历，输出树的图形表示

void visit(BiTree T);

BiTree\* gather\_operate(FST& fst);

//森林操作

status InitForest(FST& fst)

//初始化森林(以线性表形式定义)

{

if(fst != NULL) return INFEASIBLE;

fst = (FST)malloc(sizeof(FOREST));

fst->tree = (pTree\*)malloc(INIT\_SIZE \* sizeof(pTree));

fst->length = 0;

fst->size = INIT\_SIZE;

return OK;

}

status AddBiTree(const FST fst, int i, char BiTreeName[])

//向森林中第i位种入名为BiTreeName的树

{

if(fst == NULL) return INFEASIBLE;

if(i > fst->length + 1 || i < 1) return ERROR;

if(fst->length >= fst->size) //扩容

{

fst->size += INCREMENT;

fst->tree = (pTree\*)realloc(fst->tree, fst->size \* sizeof(pTree));

}

for(int j = fst->length; j >= i; j--) //移位置

{

fst->tree[j] = fst->tree[j - 1];

}

fst->tree[i - 1] = (pTree)malloc(sizeof(Tree));

strcpy(fst->tree[i - 1]->name, BiTreeName); //树命名 && 树初始化

fst->tree[i - 1]->T = NULL;

InitBiTree(fst->tree[i - 1]->T);

fst->length++;

return OK;

}

status RemoveBiTree(const FST fst, char BiTreeName[])

//砍去森林中名为BiTreeName的树

{

if(fst == NULL) return INFEASIBLE;

for(int i = 0; i < fst->length; i++)

{

if(!strcmp(BiTreeName, fst->tree[i]->name))

{

ClearBiTree(fst->tree[i]->T);

free(fst->tree[i]);

for(int a = i + 1; a < fst->length; a++)

fst->tree[a - 1] = fst->tree[a];

fst->length--;

return OK;

}

}

return ERROR;

}

int LocateBiTree(const FST fst, char BiTreeName[])

//查找当前森林中名为BiTreeName的树

{

if(fst == NULL) return INFEASIBLE;

for(int i = 0; i < fst->length; i++)

{

if(!strcmp(BiTreeName, fst->tree[i]->name)) return i + 1;

}

return ERROR;

}

status ForestTraverse(const FST fst)

//遍历森林

{

if(fst == NULL) return INFEASIBLE;

if(fst->length == 0) return ERROR;

int op = 1;

status (\*p)(BiTree, void (\*)(BiTree)) = NULL;

while(op)

{

printf(" 遍历方式选择\n");

printf(" 1.前序遍历\n");

printf(" 2.中序遍历\n");

printf(" 3.后序遍历\n");

printf(" 4.按层遍历\n");

printf(" 请输入您所选择遍历方式的相应序号[1-4]:");

scanf("%d", &op);

switch(op)

{

case 1: p = PreOrderTraverse; break;

case 2: p = InOrderTraverse; break;

case 3: p = PostOrderTraverse; break;

case 4: p = LevelOrderTraverse; break;

default: printf("输错\n");

op = 6; \_getch(); break;

}

if(op != 6) op = 0;

}

printf("当前树中数据如下所示：\n");

for(int i = 0; i < fst->length; i++)

{

printf("第%d棵树：\n %s:\n", i + 1, fst->tree[i]->name);

p(fst->tree[i]->T->lchild, visit);

printf("\n\n");

}

return OK;

}

status ClearForest(const FST fst)

{

if(fst == NULL) return INFEASIBLE;

for(int i = 0; i < fst->length; i++)

{

ClearBiTree(fst->tree[i]->T); //ClearBiTree函数：参数为T时为销毁，参数为T->lchild时清空

free(fst->tree[i]);

fst->tree[i] = NULL;

}

fst->length = 0;

return OK;

}

status DestroyForest(FST &fst)

{

if(fst == NULL) return INFEASIBLE;

ClearForest(fst);

fst = NULL;

return OK;

}

//森林操作

//bool IsRepeat(TElemType definition[])

////判断先序序列definition中是否有重复key值，用于创建树

//{

// int a = 0, b = 0;

// for(int i = 0; (a = definition[i].key) != -1; i++)

// {

// for(int j = i + 1; (b = (definition[j].key)) != -1; j++)

// {

// if(a == b && a != 0)

// {

// return true;

// }

// }

// }

// return false;

//}

status InitBiTree(BiTree &T)

{

if(T != NULL) return INFEASIBLE;

T = (BiTree)malloc(sizeof(BiTNode));

T->lchild = NULL;

T->rchild = NULL;

return OK;

}

status CreateBiTree(BiTree &T,DEF definition[])

{

int i=0,j;

static BiTNode \*p[100];

while (j=definition[i].pos)

{

p[j]=(BiTNode \*)malloc(sizeof(BiTNode));

p[j]->data=definition[i].data;

p[j]->lchild=NULL;

p[j]->rchild=NULL;

if (j!=1)

if (j%2) p[j/2]->rchild=p[j];

else p[j/2]->lchild=p[j];

i++;

}

T=p[1];

return OK;

}

//

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

////根据带空枝的二叉树先根遍历序列definition构造一棵二叉树，将根节点指针赋值给T并返回OK，

////如果有相同的关键字，返回ERROR。

//{

// static int i = 0, flag = 0;//i为递归变量，遍历definition；flag控制IsRepeat函数只被调用一次；over处理二叉树结束标志-1

// if(flag == 0 && (flag = 1) && IsRepeat(definition)) //判断是否有重复，并由于IsRepeat只需运行一次，通过flag使其成为一次性。

// {

// return ERROR;

// }

// T = (BiTree)malloc(sizeof(BiTNode)); //先对当前结点分配空间

// //printf("In CreateBiTree, definition[%d].key = %d\n", i, definition[i].key);

// if(definition[i].key == 0) //对于 0,null，直接回溯，将分配的空间free，T置NULL，definition后移

// {

// free(T);

// T = NULL;

// i++;

// if(definition[i].key == -1)

// {

// i = 0; //函数结束，全局变量应恢复原状

// flag = 0;

// }

// return OK;

// }

// else //数据进树

// {

// T->data.key = definition[i].key;

// strcpy(T->data.others,definition[i].others);

// }

// i++; //definition后移

// CreateBiTree(T->lchild, definition);

// CreateBiTree(T->rchild, definition);

//}

status ClearBiTree(BiTree &T)

//将二叉树设置成空，并删除所有结点，释放结点空间

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(T == NULL)

{

return OK;

}

ClearBiTree(T->lchild);

ClearBiTree(T->rchild);

if(T->lchild == NULL && T->rchild == NULL)

{

free(T);

T = NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

int BiTreeDepth(BiTree T)

//求二叉树T的深度

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(T == NULL)

{

return 0;

}

int LeftLen = 1 + BiTreeDepth(T->lchild);

int RightLen = 1 + BiTreeDepth(T->rchild);

return max(LeftLen, RightLen);

}

BiTNode\* LocateNode(BiTree T,KeyType e)

//查找结点

{

static BiTNode \*p = NULL;

if(p != NULL && p->data.key != e) p = NULL;

//在别处多次调用(非递归)时，防止上一次使用遗留下来的p未被置NULL而导致之后使用故障

if(p != NULL) return p;

//p不为NULL时，表明已找到目标节点，直接回溯即可

if(T == NULL) return p;

//当前T为空时，直接回溯即可

if(e == T->data.key)//当找到目标节点，将目标节点地址传至p，返回即可

{

p = T;

return p;

}

LocateNode(T->lchild,e);

LocateNode(T->rchild,e);

}

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

//实现结点赋值。

{

BiTNode \*p = LocateNode(T, value.key);

//p记录树中key值为value.key的结点地址用于保证赋值后的树中无key值重复项

BiTNode \*q = LocateNode(T, e);

//q记录结点编号为e的结点地址以查重以及赋值

if(!q || (p && q && p != q)) //所有错误状况

//q为空，代表树中无编号为e结点，不可赋值；p!=q且p,q均不为空，说明赋值后会有key值重复

{

return ERROR;

}

else

{

q->data.key = value.key;

strcpy(q->data.others, value.others);

return OK;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

int BiTreeEmpty(BiTree T)

{

if(T->lchild==NULL&&T->rchild==NULL)

return TRUE;

else return ERROR;

}

BiTNode\* GetSibling(BiTree T,KeyType e)

//实现获得兄弟结点

{

BiTNode \*rec = NULL; //记录返回值

if(T == NULL) //当前树为空时，返回空值，表明找不到，直接回溯

{

return NULL;

}

if(T->lchild != NULL && T->lchild->data.key == e) return T->rchild; //当前结点左儿子为所找结点时，返回当前结点右儿子

else if(T->rchild != NULL && T->rchild->data.key == e) return T->lchild;//当前结点左儿子为所找结点时，返回当前结点右儿子

else //不满足以上两种，继续深入搜索

{

if(rec = GetSibling(T->lchild, e)) return rec; //如果左边找到了，返回其返回值

else if(rec = GetSibling(T->rchild, e)) return rec;//如果右边找到了，返回其返回值

else return NULL; //都没有，返回NULL

}

}

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

//实现插入节点，在e的LR(左右)孩子处插入c

{

BiTNode \*aim = LocateNode(T, e); //查找e所在结点

BiTNode \*repeat = LocateNode(T, c.key); //判断即将插入的结点key值在原树中是否已存在

if(aim == NULL || repeat != NULL) return ERROR; //当未找到 key=e 的结点或者将插入的结点key值在原树中已存在

else

{

BiTNode \*NewNode = (BiTNode\*)malloc(sizeof(BiTNode));

NewNode->data.key = c.key;

strcpy(NewNode->data.others, c.others); //新建结点，存储新结点值

if(LR == 0) //左插

{

NewNode->rchild = aim->lchild;

NewNode->lchild = NULL;

aim->lchild = NewNode;

return OK;

}

else if(LR == 1) //右插

{

NewNode->rchild = aim->rchild;

NewNode->lchild = NULL;

aim->rchild = NewNode;

return OK;

}

else if(LR == -1) //头插

{

NewNode->rchild = T;

NewNode->lchild = NULL;

T = NewNode;

return OK;

}

else return ERROR;

}

}

status ParentNode(BiTree T, KeyType e, BiTree &p)

/\*寻找key值为e元素的双亲结点，由参数p带出函数，返回OK表示删除结点为根节点，

返回LEFT表示e为p的左孩子，返回RIGHT表示e为p的右孩子，返回ERROR表示没有此结点\*/

{

status state = ERROR;

if(T == NULL) return ERROR;

if(T->data.key == e) {p = T; return OK;}

else if(T->lchild && T->lchild->data.key == e) {p = T; return LEFT;}

else if(T->rchild && T->rchild->data.key == e) {p = T; return RIGHT;}

state = ParentNode(T->lchild, e, p);

if(state) return state;

return ParentNode(T->rchild, e, p);

}

BiTNode\* RightMostNode(BiTree T)

//寻找 T 最右端结点

{

while(T->rchild != NULL) T = T->rchild;

return T;

}

status DeleteNode(BiTree &T,KeyType e)

//删除结点。

{

BiTNode \*p = NULL, \*del = NULL;

status state = ParentNode(T, e, p);

if(state == OK) //根节点为所要删除的结点

{

if(p->lchild == NULL)//根结点没有左孩子

{

T = T->rchild;

free(p);

return OK;

}

else if(p->rchild == NULL)//根结点没有右孩子

{

T = T->lchild;

free(p);

return OK;

}

else //根结点度为二

{

T = T->lchild;

RightMostNode(T)->rchild = p->rchild;//根节点左孩子的最右置为根节点右孩子

free(p);

return OK;

}

}

else if(state == LEFT) //待删除结点为p的左孩子时

{

del = p->lchild; //记录待删除结点

if(del->lchild != NULL)//如果要删除的结点有左孩子

{

RightMostNode(del->lchild)->rchild = del->rchild; //要删除的结点的右孩子移至要删除的结点的左孩子的最右结点的右子树处

p->lchild = del->lchild;

}

else p->lchild = del->rchild;

free(del);

return OK;

}

else if(state == RIGHT) //待删除结点为p的右孩子时

{

del = p->rchild; //记录待删除结点

if(del->lchild != NULL) //待删除结点有左孩子

{

RightMostNode(del->lchild)->rchild = del->rchild;//要删除的结点的右孩子移至要删除的结点的左孩子的最右结点的右子树处

p->rchild = del->lchild;

}

else p->rchild = del->rchild;//待删除结点没有左孩子

free(del);

return OK;

}

else return ERROR; //未查找到待删除结点

}

status PreOrderTraverse(BiTree T,void (\*visit)(BiTree))

//先序遍历二叉树T(非递归)

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(T == NULL) {

printf("空");

return OK;

}

BiTree st[20],p;//用栈实现

int top=0;//置空栈；

if(T){

st[top++]=T;

while(top){

p=st[--top];

visit(p);

if(p->rchild)

st[top++]=p->rchild;

if(p->lchild)

st[top++]=p->lchild;

}

}

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status InOrderTraverse(BiTree T,void (\*visit)(BiTree))

//中序遍历二叉树T（递归实现）

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (T)

{

InOrderTraverse(T->lchild,visit);

visit(T);

InOrderTraverse(T->rchild,visit);

return OK;

}

else return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status PostOrderTraverse(BiTree T,void (\*visit)(BiTree))

//后序遍历二叉树T

{

if(T == NULL) return OK;

PostOrderTraverse(T->lchild, visit);

PostOrderTraverse(T->rchild, visit);

visit(T);

return OK;

}

status LevelOrderTraverse(BiTree T,void (\*visit)(BiTree))

//按层遍历二叉树T

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

static int l=0,r=1;

static BiTree trees[200];//建立数组来持续遍历

if(l||r!=1||trees)

{

l=0;

r=1;

trees[200]=NULL;

}

trees[0]=T;//从根节点开始，不用考虑右子树

for(l=0;l<r;l++)

{

visit(trees[l]);

if(trees[l]->lchild)//所遍历的该层该节点有左子树，在下一层前面遍历

{

trees[r++]=trees[l]->lchild;

}

if(trees[l]->rchild)//有右子树

{

trees[r++]=trees[l]->rchild;

}

}

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status SaveBiTree(BiTree T,FILE \*fp,int i)

{//以满二叉树的形式进行存储

if(T==NULL) return OK;

fprintf(fp,"%d %d %s ",i,T->data.key,T->data.others);

SaveBiTree(T->lchild,fp,i\*2);

SaveBiTree(T->rchild,fp,i\*2+1);

}

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

{

FILE \*fp = fopen(FileName, "r");

if(fp == NULL) return ERROR;

int i = 0;

int number = 0;

DEF definition[100];

while(fscanf(fp,"%d",&definition[i].pos)!=EOF&&fscanf(fp,"%d",&definition[i].data.key)!=EOF&&fscanf(fp,"%s",&definition[i].data.others)!=EOF)

{

i++;

}

CreateBiTree(T,definition);

fclose(fp);

return OK;

}

void visit(BiTree T)

{

printf(" %d,%s",T->data.key,T->data.others);

}

BiTree\* gather\_operate(FST& fst)

//森林操作

{

int op = 1; int j = 0;

char BiTreeName[30];

while(op)

{

system("cls"); printf("\n\n");

printf(" 森林操作菜单\n");

printf("--------------------------------------------\n");

printf(" 1. 初始化 6. 修改\n");

printf(" 2. 新增 7. 遍历\n");

printf(" 3. 删除 8. 销毁\n");

printf(" 4. 查找 0. 退出\n");

printf(" 5. 清空\n");

printf("--------------------------------------------\n");

printf(" 请选择你的操作[0~6]:");

scanf("%d", &op);

switch(op)

{

case 1:

if(InitForest(fst) == INFEASIBLE) printf("当前森林已被初始化！\n");

else printf("初始化成功！\n");

printf("----任意键继续----\n");

\_getch();

break;

case 2:

if(fst == NULL) printf("当前森林不存在，请先初始化！\n");

else

{

char BiTreeName[30];

int pos = 0;

printf("请输入新增树的名称：\n");

scanf("%s",BiTreeName);

printf("请输入新树种在几号位？\n");

scanf("%d", &pos);

if(AddBiTree(fst, pos, BiTreeName) == ERROR) printf("种树位置不合法！第%d位之前仍有空位！\n");

else printf("树\"%s\"成功种进森林了！\n", BiTreeName);

}

printf("----任意键继续----\n");

\_getch();

break;

case 3:

if(fst == NULL) printf("当前森林不存在，请先初始化！\n");

else if(fst->length == 0) printf("当前森林里没有树！\n");

else

{

printf("请输入想砍掉的树的名称：\n");

scanf("%s",BiTreeName);

if(RemoveBiTree(fst, BiTreeName)) printf("树\"%s\"成功被砍没了！\n",BiTreeName);

else printf("查无此树！\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 4:

if(fst == NULL) printf("当前森林不存在，请先初始化！\n");

else

{

printf("请输入所要查找的树的名称：\n");

scanf("%s",BiTreeName);

if(j = LocateBiTree(fst, BiTreeName)) printf("树%s位于森林中第%d位\n", BiTreeName, j);

else printf("查无此树！\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 5:

// if(ClearForest(fst) == INFEASIBLE) printf("当前森林不存在，请先初始化！\n");

// else printf("清空成功！森林空了！\n");

if(fst == NULL) printf("当前森林不存在，请先初始化！\n");

else

{

printf("请输入所要清空的树的名称：\n");

scanf("%s",BiTreeName);

if(j = LocateBiTree(fst, BiTreeName))

ClearBiTree(fst->tree[j-1]->T->lchild);

printf("清空成功\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 6:

if(fst == NULL)

{

printf("当前森林不存在，请先初始化！\n");

printf("----任意键继续----\n");

\_getch();

break;

}

else

{

printf("请输入所需更改的树的名称:\n");

scanf("%s",BiTreeName);

j = LocateBiTree(fst, BiTreeName);

if(j == 0)

{

printf("查无此树！\n");

printf("----任意键继续----\n");

\_getch();

break;

}

else

{

int ord = 0;

printf("是否要修改树的名字?(是.1 / 否.0)：");

scanf("%d", &ord);

if(ord)

{

printf("输入新名字：");

scanf("%s", fst->tree[j - 1]->name);

printf("修改成功！\n");

printf("继续修改数据吗？(是.1 / 否.0):");

scanf("%d", &ord);

if(!ord) break;

}

return &(fst->tree[j - 1]->T);

}

}

case 7:

if((j = ForestTraverse(fst)) == 0) printf("当前森林一棵树也没有！\n");

else if(j == INFEASIBLE) printf("当前森林不存在！\n");

printf("----任意键继续----\n");

\_getch();

break;

case 8:

if(DestroyForest(fst) == INFEASIBLE) printf("啧..你还是不记得初始化！\n");

else printf("销毁成功！森林没了！\n");

printf("----任意键继续----\n");

\_getch();

break;

case 0:

break;

default:

printf("指令错误！\n");

printf("----任意键继续----\n");

\_getch();

break;

}

}

return NULL;

}

int main(){

BiTree T = NULL; //树

FST fst = NULL; //森林

int op = 1; //操作指令

int flag = 0; //森林操作完成标志

DEF definition[100]; //输入树中的数据载体

int e = 0; //键值载体

BiTNode \*node = NULL; //结点载体

BiTree \*T\_ope = NULL, temp = NULL;//森林操作树的载体

TElemType value; //结点数据载体

int LR; //插入节点：定左右

while(op){

system("cls"); printf("\n\n");

printf(" 基于二叉链表的二叉树操作菜单\n");

printf("-----------------------------------------------------------------------\n");

printf(" 1. 初始化树 8. 获得兄弟结点 15. 二叉树判空\n");

printf(" 2. 初始化输入 9. 插入节点 16. 存档\n");

printf(" 3. 摧毁树 10. 删除结点 17. 读档\n");

printf(" 4. 清空树 11. 前序遍历 18. 森林管理\n");

printf(" 5. 树的深度 12. 中序遍历 0. 退出/载入到森林\n");

printf(" 6. 结点查找 13. 后序遍历\n");

printf(" 7. 结点赋值 14. 按层遍历\n");

printf("-----------------------------------------------------------------------\n");

if(op)

printf(" 请选择你的操作[0~18]:");

scanf("%d",&op);

switch(op){

case 1:

if(InitBiTree(T) == OK) printf("初始化成功！\n");

else printf("当前的树里已经有点东西了~\n");

printf("----任意键继续----\n");

\_getch();

break;

case 2:

if(T == NULL) printf("请先初始化！\n");

else if(T->lchild != NULL) printf("当前的树非空\n");

else{

int i=0;

printf("输入数据：(格式：满二叉树中的位置序号+键值+数据)，位置序号为0时结束\n");

do {

scanf("%d%d%s",&definition[i].pos,&definition[i].data.key,definition[i].data.others);

} while (definition[i++].pos);

CreateBiTree(T->lchild,definition);

printf("输入成功！\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 3:

if(T == NULL) printf("请先初始化！\n");

else{

ClearBiTree(T);

printf("销毁成功！\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 4:

if(T == NULL) printf("请先初始化！\n");

else if(flag==0){

ClearBiTree(T->lchild);

printf("清空成功！\n");

}

else if(flag)

printf("在当前页面不支持树的删除功能，请在森林中完成\n");

printf("----任意键继续----\n");

\_getch();

break;

case 5:

if(T == NULL) printf("树不存在\n");

else printf("当前树的深度为%d\n", BiTreeDepth(T->lchild));

printf("----任意键继续----\n");

\_getch();

break;

case 6:

if(T == NULL) printf("请先初始化！\n");

else{

printf("输入键值：");

scanf("%d", &e);

node = LocateNode(T->lchild, e);

if(node == NULL) printf("当前树里面没有key值为%d的结点\n", e);

else printf("当前结点键值为:%d, 数据为:%s\n", node->data.key, node->data.others);

}

printf("----任意键继续----\n");

\_getch();

break;

case 7:

if(T == NULL) printf("当前树不存在！\n");

else{

printf("输入所要修改的结点键值：");

scanf("%d", &e);

node = LocateNode(T->lchild, e);

if(node == NULL) printf("当前树里面没有key值为%d的结点\n", e);

else

{

printf("请输入输入更改后的结点键值：");

scanf("%d", &value.key);

printf("请输入更改后的数据：");

scanf("%s", value.others);

if(Assign(T->lchild, e, value) == ERROR)

printf("更改键值出错！因为原树中已存在其他结点的键值为%d！\n", value.key);

else printf("更改成功！\n");

}

}

printf("----任意键继续----\n");

\_getch();

break;

case 8:

if(T == NULL) printf("请先初始化！\n");

else

{

printf("输入当前键值：");

scanf("%d", &e);

node = LocateNode(T->lchild, e);

if(node == NULL) printf("当前树里面没有key值为%d的结点\n", e);

else

{

node = GetSibling(T->lchild, e);

if(node == NULL) printf("这个结点没有兄弟~\n");

else printf("这个结点的兄弟结点的键值为：%d, 数据为：%s\n", node->data.key, node->data.others);

}

}

printf("----任意键继续----\n");

\_getch();

break;

case 9:

if(T == NULL) printf("请先初始化！\n");

else

{

printf("输入相应添加孩子结点键值\n");

scanf("%d", &e);

node = LocateNode(T->lchild, e);

if(node == NULL) printf("当前树里面没有key值为%d的结点\n", e);

else{

printf("想添个左孩子还是右孩子？(左孩子输0，右孩子输1，当祖先输-1)\n");

scanf("%d", &LR);

printf("输入新增结点键值：");

scanf("%d", &value.key);

printf("输入新增结点数据：");

scanf("%s", value.others);

if(InsertNode(T->lchild, e, LR, value) == ERROR) printf("新增键值出错！因为原树中已存在其他结点的键值为%d！\n", value.key);

else printf("插入节点成功！\n");

}

}

printf("----任意键继续----\n");

\_getch();

break;

case 10:

if(T == NULL) printf("请先初始化！\n");

else{

printf("输入待删除结点的键值：");

scanf("%d", &e);

if(DeleteNode(T, e) == ERROR) printf("当前树里面没有key值为%d的结点\n", e);

else printf("删除成功！\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 11:

if(T == NULL) printf("先请先初始化！\n");

else

{

printf("前序遍历当前树中结点信息如下：\n");

PreOrderTraverse(T->lchild, visit);

printf("\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 12:

if(T == NULL) printf("请先初始化！\n");

else

{

printf("中序遍历当前树中结点信息如下：\n");

InOrderTraverse(T->lchild, visit);

printf("\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 13:

if(T == NULL) printf("请先初始化！\n");

else

{

printf("后序遍历当前树中结点信息如下：\n");

PostOrderTraverse(T->lchild, visit);

printf("\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 14:

if(T == NULL) printf("请先初始化！\n");

else

{

printf("按层遍历当前树中结点信息如下：\n");

LevelOrderTraverse(T->lchild, visit);

printf("\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 15:

if(T == NULL) printf("树不存在\n");

else

{

BiTreeEmpty(T);

printf("清空成功");

printf("\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 16:

if(T == NULL) printf("您还未初始化！\n");

else{

char SaveFile[40];

printf("输入存档文件路径(输入0选择默认路径):\n");

scanf("%s",SaveFile);

if(!strcmp(SaveFile,"0")) strcpy(SaveFile, "D:\\数据结构实验\\文件.dat");

FILE \*fp = fopen(SaveFile, "w");

SaveBiTree(T->lchild,fp,1);

fclose(fp);

printf("已将数据存入 %s\n",SaveFile);

}

printf("----任意键继续----\n");

\_getch();

break;

case 17:

if(T == NULL) printf("树不存在树不存在树不存在！\n");

else if(T->lchild != NULL) printf("当前这颗树非空！\n");

else

{

char LoadFile[40];

printf("输入读档文件路径(输入0选择默认路径):\n");

scanf("%s",LoadFile);

if(!strcmp(LoadFile,"0")) strcpy(LoadFile, "D:\\数据结构实验\\文件.dat");

if(LoadBiTree(T->lchild, LoadFile) == ERROR) printf("存档文件%s无法打开！\n", LoadFile);

else printf("数据已从 %s 中读出\n",LoadFile);

}

printf("----任意键继续----\n");

\_getch();

break;

case 18:

if(flag == 1) //修改状态，不可再次选择线性表集合操作

{

printf("当前有未结束的线性表修改，请先结束前一修改！\n");

printf("----任意键继续----\n");

\_getch();

break;

}

T\_ope = gather\_operate(fst);//采用指向指针的指针，方便进行读取

if(T\_ope != NULL)//只要选择了集合操作中的修改，就将主界面线性表置换成集合中所选线性表

{

temp = T; //temp暂存主界面线性表数据

T = \*T\_ope; //主界面后台线性表置换为集合内所选线性表

flag = 1; //系统状态置为集合操作态

}

break;

case 0:

break;

printf("----任意键继续----\n");

\_getch();

break;

}//end of switch

if(flag && op == 0) //flag判断当前是否在线性表集合操作中，此时退出返回主界面

{

T = temp; //返还原始主界面线性表数据

op = 1; //保持不退出while

flag = 0; //状态置为初始界面

printf("原主界面的树中数据已载入！\n");

\_getch();

}

}//end of while

printf("欢迎下次再使用本系统！\n");

printf("----任意键结束----\n");

\_getch();

return 0;

}//end of main()

# 附录D 基于邻接表图实现的源程序

#include "stdio.h"

#include "stdlib.h"

#include<string.h>

#include "conio.h"

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 20

typedef int status;

typedef int KeyType;

typedef enum {DG,DN,UDG,UDN} GraphKind;

typedef struct {

KeyType key;

char others[20];

} VertexType; //顶点类型定义

typedef struct ArcNode { //表结点类型定义

int adjvex; //顶点位置编号

struct ArcNode \*nextarc; //下一个表结点指针

} ArcNode;

typedef struct VNode{ //头结点及其数组类型定义

VertexType data; //顶点信息

ArcNode \*firstarc; //指向第一条弧

} VNode,AdjList[MAX\_VERTEX\_NUM];

typedef struct { //邻接表的类型定义

AdjList vertices; //头结点数组

int vexnum,arcnum; //顶点数、弧数（度数）

GraphKind kind; //图的类型

} ALGraph;

typedef struct {

char name[30];//图的名字

ALGraph G;

}Graph, \*G;

typedef struct { //多图定义

G \*elem;

int length;//多重图的长度

int size;//多重图的大小

}ALGraphs;

int IsRepeat(ArcNode\* arc, int num);//判断是否有重复的弧

int IsRepeat(ALGraph G, KeyType key);// 判断是否有重复的键值

status CreateGraph(ALGraph &G);//新建一个图

int LocateVex(ALGraph G, KeyType u);// 定位图中的键值

status CreateCraph(ALGraph& G, VertexType V[], KeyType VR[][2]);//按照一定的规则进行图的创建

status DestroyGraph(ALGraph& G);//摧毁一个图

status PutVex(ALGraph &G,KeyType u,VertexType value);//改变图中的键值

int FirstAdjVex(ALGraph G,KeyType u);//定位对应键值的第一邻接点

int NextAdjVex(ALGraph G,KeyType v,KeyType w);//定位对应键值的邻接点的下一邻接点

status InsertVex(ALGraph &G,VertexType v);//插入顶点

status DeleteVex(ALGraph &G,KeyType v);//删除节点

status InsertArc(ALGraph &G,KeyType v,KeyType w);//插入弧

status DeleteArc(ALGraph &G,KeyType v,KeyType w);//删除弧

status DFSTraverse(ALGraph G, void (\*visit)(VertexType));//广度遍历

status DFS(ALGraph G,int i ,void (\*visit)(VertexType));//广度遍历辅助函数

status BFSTraverse(ALGraph &G,void (\*visit)(VertexType));//深度遍历

status visitNode(ALGraph G, ArcNode\*\* Q, int& head, int& tail, int\* rec, void (\*visit)(VertexType));//深度遍历辅助函数

void visit(VertexType v);//访问函数

status SaveGraph(ALGraph G, char FileName[]);//保存图

status LoadGraph(ALGraph &G, char FileName[]);//读取图

//以下为多重图操作

status InitALGraphs(ALGraphs& GS); //初始化多图(以顺序线性表形式定义)

status AddGraph(ALGraphs& GS, int i, char GraphName[]);//新增一个图

status RemoveGraph(ALGraphs& GS, char GraphName[]);//移除一个图

status ALGraphsTraverse(ALGraphs GS);//多重图的遍历

status ClearALGraphs(ALGraphs& GS);//清空多重图中的内容

status DestroyALGraphs(ALGraphs& GS);//摧毁多重图

ALGraph\* gather\_operate(ALGraphs& GS);//多重图集合操作

int main()

{

ALGraph \*G = (ALGraph\*)malloc(sizeof(ALGraph)); //图

G->vexnum = 0;

G->arcnum = 0;

int op = 1; //操作指令

int j = 0, k = 0; //函数执行状态

KeyType key = 0, key1 = 0; //键值载体

VertexType value;

ALGraphs GS; //多图

GS.elem = NULL;

int flag = 0; //多图操作完成标志

ALGraph\* G\_ope = NULL, \*temp = NULL; //多图操作图的载体

while (op) {

system("cls"); printf("\n\n");

printf(" 基于邻接表的图操作菜单\n");

printf("-----------------------------------------------------------------------\n");

printf(" 1. 创建图 7. 插入顶点 13. 存档\n");

printf(" 2. 销毁图 8. 删除顶点 14. 读档\n");

printf(" 3. 查找顶点 9. 插入弧 15. 多图管理\n");

printf(" 4. 顶点赋值 10. 删除弧 0. 退出 \n");

printf(" 5. 获得第一邻接点 11. 深度优先遍历\n");

printf(" 6. 获得下一邻接点 12. 广度优先遍历\n");

printf("-----------------------------------------------------------------------\n");

if (op)

printf(" 请选择你的操作[0~17]:");

scanf("%d", &op);

switch(op) {

case 1:

if((j = CreateGraph(\*G)) == OK) printf("创建成功！\n");

else if(j == INFEASIBLE) printf("当前的图里已经有点东西了~\n");

else if(j == ERROR) printf("输入数据有误，键值重复，创建失败！\n");

printf("----任意键继续----\n");

\_getch();

break;

case 2:

if (G->vexnum == 0) printf("空图\n");

else

{

DestroyGraph(\*G);

printf("销毁成功！\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 3:

if (G->vexnum == 0) printf("空图\n");

else

{

printf("请输入\n");

scanf("%d", &key);

if((j = LocateVex(\*G, key)) == -1) printf("没找到键值是%d的点\n", key);

else printf("键值为%d的结点在这个图里是第%d个点, 其中的数据是：key=%d, others=%s\n", key, j + 1, G->vertices[j].data.key, G->vertices[j].data.others);

}

printf("----任意键继续----\n");

\_getch();

break;

case 4:

if (G->vexnum == 0) printf("空图\n");

else {

printf("请输入要定位的键值：");

scanf("%d", &key);

if (LocateVex(\*G, key) == -1)

{

printf("没有键值为%d的顶点\n", key);

printf("----任意键继续----\n");

\_getch();

break;

}

else

{

printf("请输入要更改的键值：");

scanf("%d", &value.key);

if (LocateVex(\*G, value.key) != -1 && value.key != key)

{

printf("没有找到该键值的点\n");

printf("----任意键继续----\n");

\_getch();

break;

}

else {

printf("请输入要更改的内容");

scanf("%s", value.others);

}

}

PutVex(\*G, key, value);

printf("更改成功！\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 5:

if (G->vexnum == 0) printf("空图\n");

else

{

printf("请输入要定位的键值:\n");

scanf("%d", &key);

if ((j = FirstAdjVex(\*G, key)) == -1) printf("没有找到\n");

else printf("键值为%d的点的第一邻接点是第%d号点, 其中的数据是：key=%d, others=%s\n", key, j + 1, G->vertices[j].data.key, G->vertices[j].data.others);

}

printf("----任意键继续----\n");

\_getch();

break;

case 6:

if (G->vexnum == 0) printf("空图\n");

else {

printf("输入键值：");

scanf("%d", &key);

if ((j = LocateVex(\*G, key)) != -1)

{

printf("请输入在它的邻接链表里找哪一个点的下一邻接点？键值：\n");

scanf("%d", &key1);

if (IsRepeat(G->vertices[j].firstarc, LocateVex(\*G, key1)))

{

if ((j = NextAdjVex(\*G, key, key1)) == -1) printf("无下一邻接点！\n");

else printf("键值为%d的点相对于键值为%d的点的下一邻接点为第%d号点，其数据为：key=%d, others=%s\n", key, key1, j + 1, G->vertices[j].data.key, G->vertices[j].data.others);

}

else printf("键值为%d的点与键值为%d的点不邻接！\n",key, key1);

}

else

{

printf("该图中没有与之对应的点\n");

}

}

printf("----任意键继续----\n");

\_getch();

break;

case 7:

if (G->vexnum == MAX\_VERTEX\_NUM) printf("超过图的上线\n");

else {

printf("请输入新增顶点的键值：");

scanf("%d", &value.key);

if (IsRepeat(\*G, value.key)) printf("重复的键值\n");

else

{

printf("请输入新增顶点的数据：");

scanf("%s", value.others);

InsertVex(\*G, value);

printf("添加成功！\n");

}

}

printf("----任意键继续----\n");

\_getch();

break;

case 8:

if (G->vexnum == 0) printf("空图\n");

else

{

printf("输入待删除顶点键值：");

scanf("%d", &key);

j = LocateVex(\*G, key);

if (j == -1) printf("当前图里面没有key值为%d的顶点\n", key);

else

{

DeleteVex(\*G, key);

printf("删除成功！\n");

}

}

printf("----任意键继续----\n");

\_getch();

break;

case 9:

if (G->vexnum < 2) printf("不符合增减条件\n");

else

{

printf("请输入增加弧顶点的键值：");

scanf("%d%d", &key, &key1);

j = InsertArc(\*G, key, key1);

if(j == INFEASIBLE) printf("这两点之间已经有弧了!\n");

else if (j == ERROR)

{

if ((k = LocateVex(\*G, key)) == -1)

{

if (LocateVex(\*G, key1) == -1)

printf("没有找到对应的点\n");

else

printf("键值为%d的点不在这个图里\n", key);

}

else if (k != -1) printf("键值为%d的点不在这个图里\n", key1);

}

else printf("添加成功！\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 10:

if (G->vexnum < 2) printf("不满足增减条件\n");

else

{

printf("请输入删减的的弧对应的两个顶点：");

scanf("%d%d", &key, &key1);

j = DeleteArc(\*G, key, key1);

if(j == INFEASIBLE) printf("这两个点不邻接！\n");

else if (j == ERROR)

{

if ((k = LocateVex(\*G, key)) == -1)

{

if (LocateVex(\*G, key1) == -1)

printf("没有找到对应的点\n");

else

printf("键值为%d的点不在这个图里\n", key);

}

else printf("键值为%d的点不在这个图里\n", key1);

}

else printf("删除成功！\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 11:

if (G->vexnum == 0) printf("空图\n");

else

{

printf("深度优先遍历时，当前图中所有点的信息如下：\n");

DFSTraverse(\*G, visit);

printf("\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 12:

if (G->vexnum == 0) printf("空图\n");

else

{

printf("广度优先遍历时，当前图中所有点的信息如下：\n");

BFSTraverse(\*G, visit);

printf("\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 13:

if (G->vexnum == 0) printf("空图\n");

else {

char SaveFile[40];

printf("输入存档文件路径(输入0选择默认路径):\n");

scanf("%s", SaveFile);

if (!strcmp(SaveFile, "0")) strcpy(SaveFile, "D:\\数据结构实验\\文件.dat");

SaveGraph(\*G, SaveFile);

printf("已将数据存入 %s\n", SaveFile);

}

printf("----任意键继续----\n");

\_getch();

break;

case 14:

if(G->vexnum != 0) printf("当前图中已有其它结点!\n");

else

{

char LoadFile[40];

printf("输入读档文件路径(输入0选择默认路径):\n");

scanf("%s", LoadFile);

if (!strcmp(LoadFile, "0")) strcpy(LoadFile, "D:\\数据结构实验\\文件.dat");

if ((j = LoadGraph(\*G, LoadFile)) == ERROR) printf("存档文件%s无法打开！\n", LoadFile);

else if(j == INFEASIBLE) printf("存档文件中点的个数超出容量!读档失败!\n");

else printf("数据已从 %s 中读出\n", LoadFile);

// printf("当前图中有%d个点, %d条边\n", G->vexnum, G->arcnum);

}

printf("----任意键继续----\n");

\_getch();

break;

case 15:

if (flag == 1) //修改状态，不可再次选择线性表集合操作

{

printf("当前有未结束的修改，请先结束前一修改！\n");

printf("----任意键继续----\n");

\_getch();

break;

}

G\_ope = gather\_operate(GS);

if (G\_ope != NULL)//只要选择了集合操作中的修改，就将主界面的图置换成集合中所选图

{

temp = G; //temp暂存主界面线性表数据

G = G\_ope; //主界面后台线性表置换为集合内所选图

flag = 1; //系统状态置为集合操作态

}

break;

case 0:

break;

}//end of switch

if (flag && op == 0) //flag判断当前是否在线性表集合操作中，此时退出返回主界面

{

G = temp; //返还原始主界面线性表数据

op = 1; //保持不退出while

flag = 0; //状态置为初始界面

printf("原主界面的图中数据已载入！\n");

\_getch();

}

}//end of while

printf("欢迎下次再使用本系统！\n");

printf("----任意键结束----\n");

\_getch();

return 0;

}

int IsRepeat(ArcNode\* arc, int num)

//在当前结点及其后续链表中是否存在num

{

while (arc != NULL)

{

if (arc->adjvex == num)

return 1;

arc = arc->nextarc;

}

return 0;

}

int IsRepeat(ALGraph G, KeyType key)

//图G中查找键值与key相同的点

{

for (int i = 0; i < G.vexnum; i++)

{

if (key == G.vertices[i].data.key)

return 1;

}

return 0;

}

status CreateGraph(ALGraph &G)

//输入创建图

{

if (G.vexnum != 0) return INFEASIBLE;

VertexType V[30];

KeyType VR[100][2];

int i = 0;

G.arcnum = G.vexnum = 0;

printf("按照\"键值 名称 键值 名称 ... -1 nil 连接键值 连接键值 ... -1 -1\"输入\n");

do {

scanf("%d%s", &V[i].key, V[i].others);

} while (V[i++].key != -1);

i = 0;

do {

scanf("%d%d", &VR[i][0], &VR[i][1]);

} while (VR[i++][0] != -1);

if (CreateCraph(G, V, VR) == ERROR) return ERROR;

else return OK;

}

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

/\*根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR

如果有相同的关键字，返回ERROR。此题允许通过增加其它函数辅助实现本关任务\*/

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i=0;

for(i=0;V[i].key!=-1;i++)//先判断其中的头节点数据的个数是否超

{

if(i>=MAX\_VERTEX\_NUM)

return ERROR;

}

for(i=0;V[i].key!=-1;i++)//是否有键值重复

{

int j=0;

for(j=i+1;V[j].key!=-1;j++)

{

if(V[i].key== V[j].key)

return ERROR;

}

}

i=0;

if(V[i].key==-1)//是否从一开始就没有输入

return ERROR;

G.vexnum=0;

for(i=0; V[i].key!=-1; i++)

{

G.vertices[i].data.key = V[i].key;

strcpy(G.vertices[i].data.others, V[i].others);

G.vertices[i].firstarc = NULL;

G.vexnum++;

}

//指针域链接

int count = 0; //记录图的度degree，每链接一次count++，最终count为degree的两倍

G.arcnum = 0;//度数置为0

for ( i = 0; VR[i][0] != -1; i++)

{

int head = LocateVex(G, VR[i][0]); //定位到要插入的键值

int tail = LocateVex(G, VR[i][1]);//定位到要插入的键值

if (head == -1|| tail == -1) //VR中包含V中不存在的点，错误

return ERROR;

if ( tail != head)

{

count++;

ArcNode\* in = (ArcNode\*)malloc(sizeof(ArcNode));

in->nextarc = G.vertices[head].firstarc;//利用首插法

G.vertices[head].firstarc = in;

in->adjvex = tail;

}

if (tail != head)

{

count++;

ArcNode\* back = (ArcNode\*)malloc(sizeof(ArcNode));

back->nextarc = G.vertices[tail].firstarc;

G.vertices[tail].firstarc = back;

back->adjvex = head;

}

}

G.arcnum = count / 2;

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status DestroyGraph(ALGraph& G)

/\*销毁无向图G\*/

{

for (int i = 0; i < G.vexnum; i++)

{

//释放每个结点后的链表

ArcNode\* p = G.vertices[i].firstarc, \* del = NULL;

while (p != NULL)

{

del = p;

p = p->nextarc;

free(del);

del = NULL;

}

}

G.vexnum = 0;

G.arcnum = 0;

return OK;

}

int LocateVex(ALGraph G, KeyType u)

//根据u在图G中查找顶点，查找成功返回位序，否则返回-1；

{

for (int i = 0; i < G.vexnum; i++)

{

if (G.vertices[i].data.key == u)

return i;

}

return -1;

}

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

//根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK；

//如果查找失败或关键字不唯一，返回ERROR

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i,j;

i=LocateVex(G,u);

j=LocateVex(G,value.key);

if(j!=-1||i==-1)

return ERROR;

else {

G.vertices[i].data=value;

return OK;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

int FirstAdjVex(ALGraph G,KeyType u)

//根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，否则返回-1；

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i;

i=LocateVex(G,u);

if(i==-1)

{

return ERROR;

}

else

{

return G.vertices[i].firstarc->adjvex;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

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

//根据u在图G中查找顶点，查找成功返回顶点v的邻接顶点相对于w的下一邻接顶点的位序，查找失败返回-1；

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i;

i=LocateVex(G,v);

if(i==-1)

{

return -1;

}

else

{

ArcNode\* p;

p=G.vertices[i].firstarc ;

while(p){

int pos = p->adjvex;

p = p->nextarc;

if (G.vertices[pos].data.key == w)

break;

}

if (p == NULL) return -1; //没找到 w 或者 w没有下一个

else return p->adjvex;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status InsertVex(ALGraph &G,VertexType v)

//在图G中插入顶点v，成功返回OK,否则返回ERROR

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

int i;

i=LocateVex(G,v.key);

if(i!=-1)

{

return ERROR;

}

else{

if(G.vexnum == MAX\_VERTEX\_NUM )

return ERROR;

G.vertices[G.vexnum].data.key = v.key;

strcpy(G.vertices[G.vexnum].data.others, v.others);

G.vertices[G.vexnum].firstarc = NULL;

G.vexnum++;

return OK;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status DeleteVex(ALGraph &G,KeyType v)

//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

{

int i;

i=LocateVex(G,v);

if(G.vexnum==1||i==-1)return ERROR;//若删除后为空图 或者没有找到该节点

for(int j=0;j<G.vexnum;j++)//将该顶点前的顶点中含有该顶点的弧空间释放

{

ArcNode \*p=G.vertices[j].firstarc,\*q;

if(!p)continue;

if(p->adjvex==i)

{

G.vertices[j].firstarc=p->nextarc;

free(p);

continue;

}

while(p->nextarc)

{

if(p->nextarc->adjvex==i)

{

q=p->nextarc;

p->nextarc=q->nextarc;

free(q);

break;

}

p=p->nextarc;

}

}

ArcNode \*p=G.vertices[i].firstarc,\*q;

while(p)//将该顶点上的弧空间释放

{

q=p;

p=p->nextarc;

free(q);

G.arcnum--;

}

G.vertices[i].firstarc=NULL;

G.vexnum--;//无向图中顶点数-1

//无向图中弧数量减少sum

for(int j=i;j<G.vexnum;j++)//顶点移位

{

G.vertices[j]=G.vertices[j+1];

}

for(int j=0;j<G.vexnum;j++)//调整弧的顶点位序

{

p=G.vertices[j].firstarc;

while(p)

{

if(p->adjvex>i)

p->adjvex--;

p=p->nextarc;

}

}

return OK;

return ERROR;//未找到该顶点

}

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

//在图G中增加弧<v,w>，成功返回OK,否则返回ERROR

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

//顶点数小于2的图直接

if (G.vexnum < 2) return ERROR;

int i,j;

i=LocateVex(G,v);

j=LocateVex(G,w);

if(i==-1||j==-1)//如果两个节点都没有，返回错误

return ERROR;

else//当节点都存在时

{

ArcNode\* p = G.vertices[i].firstarc;

while (p != NULL)

{

if (p->adjvex == j)

return INFEASIBLE;

p = p->nextarc;

}

//v后加 w

ArcNode\* newj = (ArcNode\*)malloc(sizeof(ArcNode));

newj->adjvex =j;

newj->nextarc = G.vertices[i].firstarc;

G.vertices[i].firstarc = newj;

//w后加 v

ArcNode\* newi = (ArcNode\*)malloc(sizeof(ArcNode));

newi->adjvex = i;

newi->nextarc = G.vertices[j].firstarc;

G.vertices[j].firstarc = newi;

G.arcnum++;

return OK;

}

}

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

//在图G中删除弧<v,w>，成功返回OK,否则返回ERROR

{ // 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (G.vexnum < 2) return ERROR;

int i,j;

i=LocateVex(G,v);

j=LocateVex(G,w);

if(i==-1||j==-1)//如果两个节点都没有，返回错误

return ERROR;

ArcNode\* p = G.vertices[i].firstarc;

ArcNode\* q = G.vertices[j].firstarc;

ArcNode\* temp;

int flag = 0;

while (p != NULL)

{

if (p->adjvex == j)

flag = 1;

p = p->nextarc;

}

if (!flag) return ERROR;

p = G.vertices[i].firstarc;

if(p->adjvex==j)

{

G.vertices[i].firstarc=p->nextarc;

free(p);

}

while(p->nextarc)

{

if(p->nextarc->adjvex==j)

{

temp=p->nextarc;

p->nextarc=temp->nextarc;

free(temp);

break;

}

p=p->nextarc;

}

if(q->adjvex==i)

{

G.vertices[j].firstarc=q->nextarc;

free(q);

}

while(q->nextarc)

{

if(q->nextarc->adjvex==i)

{

temp=q->nextarc;

q->nextarc=temp->nextarc;

free(temp);

break;

}

q=q->nextarc;

}

G.arcnum--;

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

int state[20];//记录状态，确保只遍历一次

status DFS(ALGraph G,int i ,void (\*visit)(VertexType))

{

int j;

//1.处理起始点

visit(G.vertices[i].data);//1.输出起始结点

state[i]=1;//2.将已访问的结点标志成1

//2.由起始点开始，对后续结点进行操作

ArcNode\* p=G.vertices[i].firstarc;

while(p)

{

j=p->adjvex;//访问其邻接点

if(state[j]==0)//当满足有边且未被访问过时，递归调用去查找该邻接点

{

DFS(G,j,visit);

}

p=p->nextarc;

}

return OK;

}

void visit(VertexType v)

{

printf(" %d %s",v.key,v.others);

}

status DFSTraverse(ALGraph G, void (\*visit)(VertexType)){

//对图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次

int i=0;

for(i=0;i<G.vexnum;i++)

{

state[i]=0;

}

for(i=0;i<G.vexnum;i++)

{

if(!state[i])//当出现非联通分支的时候，重新遍历

DFS(G,i,visit);

}

return OK;

}

status BFSTraverse(ALGraph &G,void (\*visit)(VertexType))

//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

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

ArcNode\*\* Q = (ArcNode\*\*)calloc(G.vexnum, sizeof(ArcNode\*));

int state[20]={0} ;

int head = 0, tail = 0;

for (int i = 0; i < G.vexnum; i++)

{

if (!state[i])

{

Q[tail++] = G.vertices[i].firstarc;

visit(G.vertices[i].data);

state[i] = 1;

while (head != tail)

{

//访问当前结点并出队

ArcNode\* p = Q[head++];

if(p == NULL) return OK;

if (!state[p->adjvex])

{

visit(G.vertices[p->adjvex].data);

state[p->adjvex] = 1;

}

//进队

while (p != NULL)

{

if (!state[p->adjvex])

{

Q[tail++] = p;

}

p = p->nextarc;

}

}

}

}

free(Q);

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status SaveGraph(ALGraph G, char FileName[])

//将图的数据写入到文件FileName中

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 1 \*\*\*\*\*\*\*\*\*/

FILE\* fp = fopen(FileName, "w+");

for (int i = 0; i < G.vexnum; i++)

{

fprintf(fp, "%d %s", G.vertices[i].data.key, G.vertices[i].data.others);

ArcNode\* p = G.vertices[i].firstarc;

while (p != NULL)

{

fprintf(fp, " %d", p->adjvex);

p = p->nextarc;

}

fprintf(fp, "\n");

}

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 1 \*\*\*\*\*\*\*\*\*\*/

}

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

//读入文件FileName的图数据，创建图的邻接表

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 2 \*\*\*\*\*\*\*\*\*/

FILE\* fp = fopen(FileName, "r");

if(fp == NULL) return ERROR;

int ch;

G.vexnum = G.arcnum = 0;

for (int i = 0; ch != EOF; i++)

{

//读取部分

fscanf(fp, "%d %s", &G.vertices[i].data.key, G.vertices[i].data.others);

G.vexnum++;

ch = fgetc(fp);

if (ch != '\n')

{

G.vertices[i].firstarc = (ArcNode\*)malloc(sizeof(ArcNode));

ArcNode\* p = G.vertices[i].firstarc;

//读一个空格，读一个数据，读到换行直接退，此行的链表构建完成

while ((ch != '\n') && (ch != EOF) && fscanf(fp, "%d", &ch) != EOF)

{

p->adjvex = ch;

ch = fgetc(fp);

if (ch == '\n' || ch == EOF)

p->nextarc = NULL;

else

{

p->nextarc = (ArcNode\*)malloc(sizeof(ArcNode));

p = p->nextarc;

p->nextarc = NULL;

}

G.arcnum++;

}

p = NULL;

}

else G.vertices[i].firstarc = NULL;

}

--G.vexnum;

G.arcnum /= 2;

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 2 \*\*\*\*\*\*\*\*\*\*/

}

status InitALGraphs(ALGraphs& GS)

//初始化多图(以顺序线性表形式定义)

{

if (GS.elem != NULL) return INFEASIBLE;

GS.elem = (G\*)malloc(20\*sizeof(Graph));

GS.length = 0;

GS.size = 20;

return OK;

}

status AddGraph(ALGraphs& GS, int i, char GraphName[])

//第i位加入名为GraphName的图

{

if (GS.elem == NULL) return INFEASIBLE;

if (i > GS.length + 1 || i < 1) return ERROR;

for(int i = 0; i < GS.length; i++)

{

if(!strcmp(GraphName, GS.elem[i]->name))

return ERROR;

}

//建立新图

G p = (Graph\*)malloc(sizeof(Graph));

int j = 0; int ord;

strcpy(p->name, GraphName); //图命名 && 图初始化

p->G.arcnum = p->G.vexnum = 0;

printf("直接输入或者不输入(1/0)\n");

scanf("\n%c", &ord);

if(ord == 1)

{

j = CreateGraph(p->G);

if(j == ERROR) printf("输入数据有误，部分顶点键值重复，新增失败！\n");

}

if (GS.length >= GS.size) //扩容

{

GS.size += 10;

GS.elem = (G\*)realloc(GS.elem, GS.size \* sizeof(G));

}

for (int j = GS.length; j >= i; j--) //移位置

{

GS.elem[j] = GS.elem[j - 1];

}

GS.elem[i - 1] = p;

GS.length++;

return OK;

}

status RemoveGraph(ALGraphs& GS, char GraphName[])

//删去名为GraphName的图

{

if (GS.elem == NULL) return INFEASIBLE;

for (int i = 0; i < GS.length; i++)

{

if (!strcmp(GraphName, GS.elem[i]->name))

{

DestroyGraph(GS.elem[i]->G);

free(GS.elem[i]);

for (int a = i + 1; a < GS.length; a++)

GS.elem[a - 1] = GS.elem[a];

GS.length--;

return OK;

}

}

return ERROR;

}

int LocateGraph(ALGraphs GS, char GraphName[])

//查找当前森林中名为GraphName的树

{

if (GS.elem == NULL) return INFEASIBLE;

for (int i = 0; i < GS.length; i++)

{

if (!strcmp(GraphName, GS.elem[i]->name)) return i + 1;

}

return ERROR;

}

status ALGraphsTraverse(ALGraphs GS)

//遍历多图

{

if (GS.elem == NULL) return INFEASIBLE;

if (GS.length == 0) return ERROR;

int op = 0;

if(op==0)

{

printf("请选择遍历方式：1.深度优先遍历 2.广度优先遍历\n");

scanf("%d", &op);

}

if(op==1)

{

printf("当前集合中数据如下所示：\n");

for (int i = 0; i < GS.length; i++)

{

printf("第%d个图：\n %s:\n", i + 1, GS.elem[i]->name);

DFSTraverse(GS.elem[i]->G, visit);

printf("\n\n");

}

}

else{printf("当前集合中数据如下所示：\n");

for (int i = 0; i < GS.length; i++)

{

printf("第%d个图：\n %s:\n", i + 1, GS.elem[i]->name);

BFSTraverse(GS.elem[i]->G, visit);

printf("\n\n");

}

}

return OK;

}

status ClearALGraphs(ALGraphs& GS)

{

if (GS.elem == NULL) return INFEASIBLE;

for (int i = 0; i < GS.length; i++)

{

DestroyGraph(GS.elem[i]->G);

free(GS.elem[i]);

GS.elem[i] = NULL;

}

GS.length = 0;

return OK;

}

status DestroyALGraphs(ALGraphs& GS)

{

if (GS.elem == NULL) return INFEASIBLE;

ClearALGraphs(GS);

GS.size = 0;

free(GS.elem);

GS.elem = NULL;

return OK;

}

ALGraph\* gather\_operate(ALGraphs& GS)

{

int op = 1; int j = 0; int i = 0;

char GraphName[30];

while (op)

{

system("cls"); printf("\n\n");

printf(" 多图集合操作菜单\n");

printf("--------------------------------------------\n");

printf(" 1. 初始化 5. 修改\n");

printf(" 2. 新增 6. 遍历\n");

printf(" 3. 删除 7. 清空集合\n");

printf(" 4. 查找 0. 退出\n");

printf("--------------------------------------------\n");

printf(" 请选择你的操作[0~7]:");

scanf("%d", &op);

switch (op)

{

case 1:

if (InitALGraphs(GS) == INFEASIBLE) printf("当前图集合已被初始化！\n");

else printf("初始化成功！\n");

printf("----任意键继续----\n");

\_getch();

break;

case 2:

if (GS.elem == NULL)

{

printf("当前图集合不存在，请先初始化！\n");

printf("----任意键继续----\n");

\_getch();

break;

}

printf("请输入新增图的名称：\n");

scanf("%s", GraphName);

printf("新增到第几位?");

scanf("%d", &i);

if ((j = AddGraph(GS, i, GraphName)) == ERROR) printf("插入位置非法，添加失败！\n");

else printf("图\"%s\"存储成功！\n", GraphName);

printf("----任意键继续----\n");

\_getch();

break;

case 3:

if (GS.elem == NULL)

{

printf("当前图集合不存在，请先初始化！\n");

printf("----任意键继续----\n");

\_getch();

break;

}

printf("请输入所要删除的图名称：\n");

scanf("%s", GraphName);

if (RemoveGraph(GS, GraphName) == OK)

{

printf("图\"%s\"删除成功！\n", GraphName);

}

else

{

printf("查无此图！\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 4:

if (GS.elem == NULL)

{

printf("当前图集合不存在，请先初始化！\n");

printf("----任意键继续----\n");

\_getch();

break;

}

printf("请输入所要查找的图名称：\n");

scanf("%s", &GraphName);

if (j = LocateGraph(GS, GraphName))

{

printf("图\"%s\"位于集合中第%d位\n", GraphName, j);

}

else

{

printf("无图！\n");

}

printf("----任意键继续----\n");

\_getch();

break;

case 5:

if (GS.elem == NULL)

{

printf("当前图集合不存在，请先初始化！\n");

printf("----任意键继续----\n");

\_getch();

break;

}

printf("请输入所需更改的图的名称:\n");

scanf("%s", GraphName);

j = LocateGraph(GS, GraphName);

if (j == 0)

{

printf("查无此图！\n");

printf("----任意键继续----\n");

\_getch();

break;

}

else

{

return &(GS.elem[j - 1]->G);

}

case 6:

if ((j = ALGraphsTraverse(GS)) == ERROR) printf("当前图集合为空！\n");

else if (j == INFEASIBLE) printf("当前图集合不存在！\n");

printf("----任意键继续----\n");

\_getch();

break;

case 7:

if(ClearALGraphs(GS) == INFEASIBLE) printf("当前集合尚未被初始化!\n");

else printf("清空成功!\n");

printf("----任意键继续----\n");

\_getch();

break;

case 0:

break;

}

}

return NULL;

}

/\*多图操作\*/