

**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202011班**

**学 号： U202015084**

**姓 名： 张文浩**

**指导教师： 许贵平**

**报告日期： 2021年 6月 17 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 3](#_Toc458159879)

[1.1 问题描述 3](#_Toc458159880)

[1.1.1 实验目的 3](#_Toc458159880)

[1.1.2 实验要求 3](#_Toc458159880)

[1.2 系统设计 3](#_Toc458159882)

[1.2.1 整体系统结构设计 3](#_Toc458159880)

[1.2.2 数据结构设计 4](#_Toc458159880)

[1.2.3 有关类型和常量定义 6](#_Toc458159880)

[1.3 系统实现 7](#_Toc458159883)

[1.4 系统测试 1](#_Toc458159884)4

[1.5 实验小结 2](#_Toc458159884)4

[2 基于顺序链式结构的线性表实现 2](#_Toc458159879)5

[2.1 问题描述 2](#_Toc458159880)5

[2.1.1 实验目的 2](#_Toc458159880)5

[2.1.2 实验要求 2](#_Toc458159880)5

[2.2 系统设计 2](#_Toc458159882)5

[2.2.1 整体系统结构设计 2](#_Toc458159880)5

[2.2.2 数据结构设计 2](#_Toc458159880)7

[2.2.3 有关类型和常量定义 2](#_Toc458159880)8

[2.3 系统实现 2](#_Toc458159883)9

[2.4 系统测试 3](#_Toc458159884)7

2.5 [实验小结 4](#_Toc458159884)6

[3 基于二叉链表的二叉树实现](#_Toc458159879) 47

[3.1 问题描述](#_Toc458159880) 47

[3.1.1 实验目的](#_Toc458159880) 47

[3.1.2 实验要求](#_Toc458159880) 47

3.[2.2 系统设计](#_Toc458159882) 47

[3.2.1 整体系统结构设计](#_Toc458159880) 47

[3.2.2 数据结构设计](#_Toc458159880) 49

[3.2.3 有关类型和常量定义](#_Toc458159880) 51

[3.3 系统实现](#_Toc458159883) 51

[3.4 系统测试](#_Toc458159884) 66

3.5 [实验小结](#_Toc458159884) 73

[4 基于邻接表的图实现](#_Toc458159879) 74

[4.1 问题描述](#_Toc458159880) 74

4[.1.1 实验目的](#_Toc458159880) 74

[4.1.2 实验要求](#_Toc458159880) 74

4.[2.2 系统设计](#_Toc458159882) 74

[4.2.1 整体系统结构设计](#_Toc458159880) 74

[4.2.2 数据结构设计](#_Toc458159880) 76

[4.2.3 有关类型和常量定义](#_Toc458159880) 78

[4.3 系统实现](#_Toc458159883) 78

[4.4 系统测试](#_Toc458159884) 87

4.5 [实验小结](#_Toc458159884) 93

[参考文献](#_Toc458159900) 94

[附录A 基于顺序存储结构线性表实现的源程序](#_Toc458159901) 95

[附录B 基于链式存储结构线性表实现的源程序](#_Toc458159901) 112

[附录C 基于二叉链表的二叉树实现的源程序](#_Toc458159900) 131

[附录D 基于邻接表的图实现的源程序](#_Toc458159900) 152

**1 基于顺序储存结构的线性表实现**

**1.1 问题描述**

构造顺序表，呈现一个简易菜单的功能演示系统，该演示系统可选择实现多个线性表管理以及单线性表的操作。

* + 1. **实验目的**

(1)加深对线性表的概念、基本运算的理解；

(2)熟练掌握线性表的逻辑结构与物理结构的关系；

(3)物理结构采用顺序表,熟练掌握线性表的基本运算的实现。

**1.1.2 实验要求**

需要在主程序中完成函数调用以及所需实参值和函数执行结果的输出。定义线性表及线性表集合的初始化、销毁、清空、判空、求表长和获得元素等函数，并给出适当的操作提示，并且可选择以文件的形式进行存储和加载。

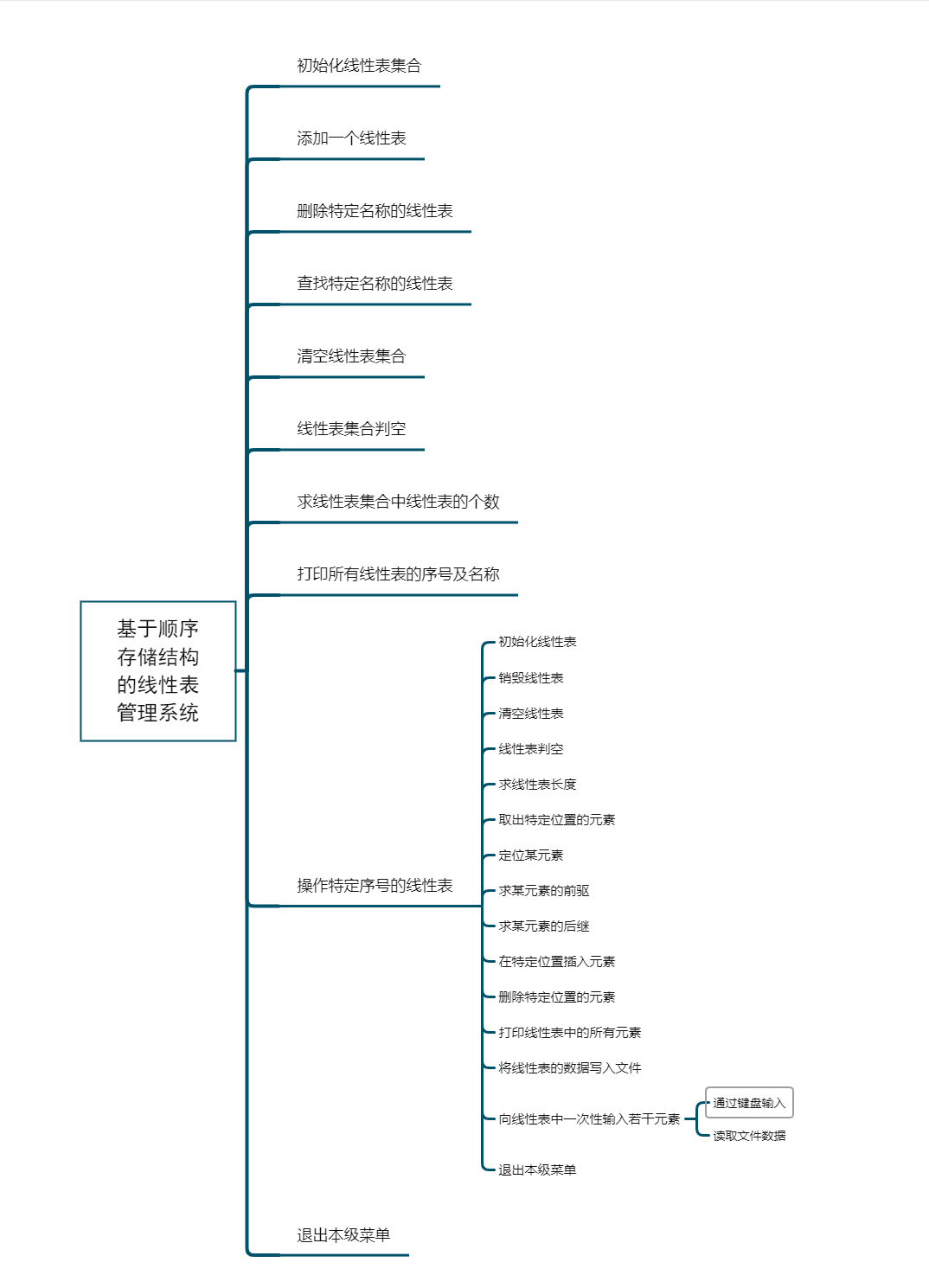
**1.2 系统设计**

**1.2.1 整体系统结构设计**

本演示系统可以实现通过操作菜单对多线性表进行管理，并对单个线性表进行操作，也支持通过文件的写入与读取。

多线性表管理菜单（线性表集合菜单/一级菜单）可供选择的操作有：初始化线性表集合、添加一个线性表、删除特定名称的线性表、查找特定名称的线性表、清空线性表集合、线性表集合判空、求线性表集合中线性表的个数、打印所有线性表的序号及名称、操作特定序号的线性表。

多线性表管理菜单中的“操作特定序号的线性表”可以调出单线性表操作菜单（二级菜单），可供选择的操作有：初始化线性表、销毁线性表、清空线性表、线性表判空、求线性表长度、取出特定位置的元素、定位某元素、求某元素的前驱、求某元素的后继、在特定位置插入元素、删除特定位置的元素、打印线性表中的所有元素、将线性表的数据写入文件、通过键盘输入或者读取文件数据向线性表中一次性输入若干元素。整体系统结构设计示意图如下：

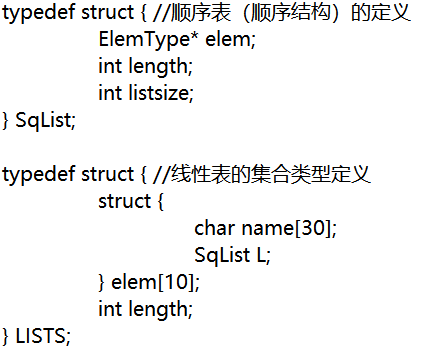
图1-1 整体系统结构设计

**1.2.2 数据结构设计**

本演示系统设计了两个物理结构，分别是线性表和线性表集合，两者均以结构形式定义。

线性表结构中包括数据数组指针、线性表长度、当前线性表最大长度。线性表集合结构中包括线性表数组（每个数组元素都包括线性表以及其名称）和线性表集合长度。

结构定义如下：

图1-2 物理结构设计

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，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()。

**1.2.3 有关常量和类型定义**

数据元素类型的定义：

typedef int status;

typedef int ElemType;

有关常量的定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

**1.3 系统实现**

本演示系统在Windows环境下，使用Visual Studio 2019完成。演示系统涉及的部分主要函数如下，其中（1）-（4）为对线性表集合的操作，（5）-（12）为对单线性表的操作。

(1)向线性表集合中添加一个空线性表（这里的实现与头歌的通关代码不同，这里将对添加的线性表的内存分配，长度和最大长度的初始化均移入对单线性表的操作中进行）

函数名称：status AddList(LISTS \*Lists,char ListName[])

输入：线性表集合指针以及需要添加的线性表的名称

输出：函数执行状态

思想：在Lists中加入一个名称为ListName的线性表，成功则返回OK

操作：

1. 将ListName数组拷贝到线性表集合中线性表数组的最后一个元素之后

的元素的name成员变量中

1. 将新添加的线性表的elem指针置空
2. 线性表集合长度加一
3. 结束，返回OK

时间复杂度：O（1）

空间复杂度：O（1）

(2)删除线性表集合中指定名称的线性表

函数名称：status RemoveList(LISTS \*Lists,char ListName[])

输入：线性表集合指针以及需要删除的线性表名称

输出：函数执行状态

思想：删除指定名称的线性表，删除成功则返回OK，否则返回ERROR

操作：

1. 定义i=0，当i小于线性表集合长度时，执行下列循环：

a.比较第i个线性表的名称和所给名称

b.当两者相同时，将第i个线性表的名称指针的首元素置为’\0’,并使elem指针分配的空间用free函数收回，并使elem指针指向NULL，将线性表数组中该线性表后面的线性表依次前移，线性表集合的长度减一，返回OK

② 若循环结束后函数没有返回值，说明集合中没有该名称的线性表，返回ERROR

时间复杂度：O（n）(设线性表集合长度为n)

空间复杂度：O（1）

(3)查找指定名称的线性表在集合中的位置

函数名称：status LocateList(LISTS Lists, char ListName[])

输入：线性表集合以及需要查找的线性表名称

输出：该线性表的序号或者函数执行状态  
思想：在集合中寻找指定名称的线性表，成功则返回其逻辑序号，否

则返回ERROR

操作：

① 定义i=0，当i小于线性表集合长度时，执行下列循环：

a.比较第i个线性表的名称与所给名称

b.如果相同，返回i+1，否则继续循环

② 如果循环执行完后函数没有返回值，说明集合中没有该名称的线性表，

返回ERROR

时间复杂度：O（n） （设线性表长度为n，此处为最坏情况）

空间复杂度：O（1）

(4)清空线性表集合

函数名称：status ClearLists(LISTS \*Lists)

输入：线性表集合指针

输出：函数执行状态

思想：将线性表集合的长度变成0，返回OK

操作：将Lists的length赋值为0，返回OK

时间复杂度：O（1）

空间复杂度：O（1）

(5)销毁线性表

函数名称：status DestroyList(SqList \*L)

输入：线性表指针

输出：函数执行状态

思想：如果线性表已经初始化，销毁线性表，释放数据元素的空间，返回

OK，否则返回INFEASIBLE

操作：如果线性表未初始化（elem指向NULL），返回INFEASIBLE，否则

释放elem的空间，将elem指向NULL，返回OK

时间复杂度：O(1)

空间复杂度：O（1）

(6)清空线性表

函数名称：status ClearList（SqList \*L）

输入：线性表指针

输出：函数执行状态

思想：如果线性表已经初始化，将其长度赋值为0，以此来达到删除所有元

素的目的

操作：如果线性表已经初始化，将其长度赋值为0，返回OK，否则返回INFEAS

IBLE

时间复杂度：O（1）

空间复杂度：O（1）

(7)获取线性表特定位置的元素

函数名称：status GetElem（SqList L,int i,ElemType \*e）

输入：线性表，需要获取的元素的位置，存放该元素的变量指针

输出：函数执行状态

思想：如果线性表已经初始化，获取线性表的第i个元素，保存在e中，返

OK，如果i不合理，返回ERROR；如果线性表未初始化，返回INFEASIBLE

操作：如果线性表已经初始化，将线性表的第i个元素，保存在e中，并返

OK，如果i不合理（i<1或者i>线性表长度），返回ERROR；如果线性表未初始化（elem指向NULL），返回INFEASIBLE

时间复杂度：O（1）

空间复杂度：O（1）

(8)确定某元素在线性表中的位置

函数名称：status LocateElem(Sqlist L,ElemType e)

输入：线性表，某元素

输出：元素位置或者函数执行状态

思想：若线性表已经初始化，查找元素位置并返回序号，查找不成功时返回相应的函数执行状态

操作：

①如果线性表未初始化，返回INFEASIBLE

②线性表已经初始化，定义i=0，执行下列循环：

当第i个元素不等于e且i小于线性表长度时，i自增。

③如果循环结束后i等于线性表长度，说明线性表中没有该元素，返回ERROR，否则返回i+1

时间复杂度：O（n）

空间复杂度：O（1）

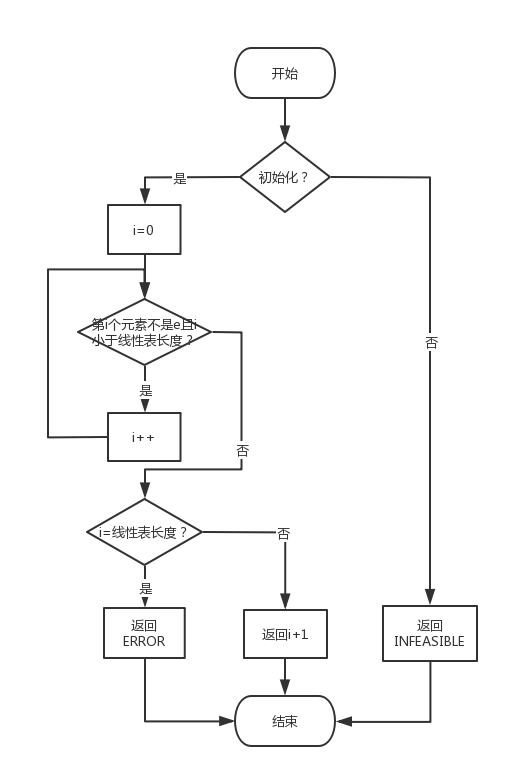


图1-3 LocateElem流程图

(9)在某位置插入某元素

函数名称：status ListInsert（SqList \*L，int i,ElemType e）

输入：线性表指针，插入位置，插入元素

输出：函数执行状态

思想：若线性表已经初始化，且插入位置正确，则插入元素，返回OK，否则返回相应的函数执行状态

操作：

①如果线性表未初始化，返回INFEASIBLE

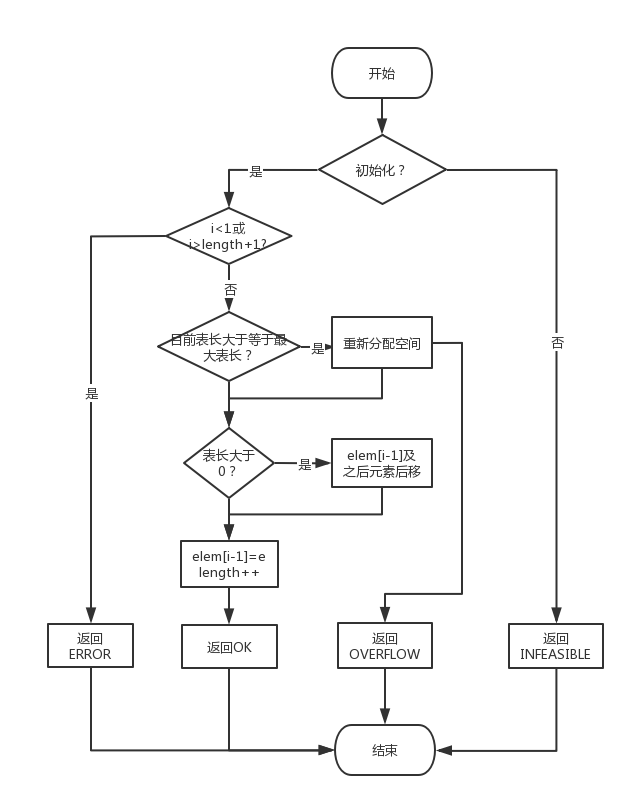
②线性表已经初始化：

a. 如果插入位置不正确，返回ERROR

b. 如果线性表长度大于等于最大长度，使用realloc扩容；如果长度大于0，将插入位置及之后的元素后移一位，将插入元素放在i位置，长度自增，返回OK

时间复杂度：O（n）

空间复杂度：O（1）

图1-4 ListInsert流程图

(10)删除某位置元素

函数名称：status ListDelete（SqList \*L，int i，ElemType \*e）

输入：线性表指针，删除位置，储存删除元素的变量指针

输出：函数执行状态

思想：若线性表已经初始化且删除位置正确，则删除对应位置元素，返回OK，否则返回相应的函数执行状态

操作：

①如果线性表未初始化，返回INFEASIBLE

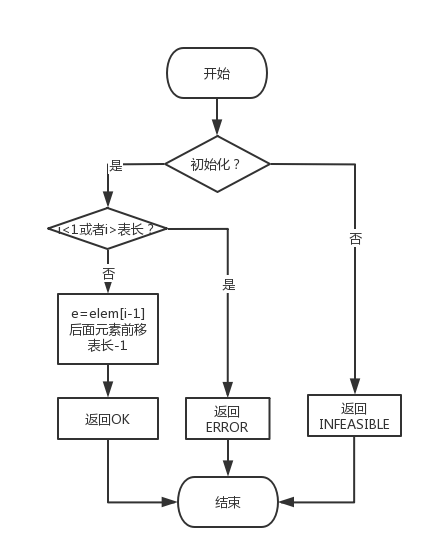
②线性表未初始化：

a.若删除位置不正确，返回ERROR

b.否则用e保存该位置元素，后面元素前移，长度减一，返回OK

时间复杂度：O（n）

空间复杂度：O（1）

图1-5 ListDelete流程图

(11)将线性表的元素写入某文件

函数名称：status SaveList（SqList L，char FileName[]）

输入：线性表，文件名称

输出：函数执行状态

思路：如果线性表已经初始化，将元素写入文件，否则返回对应的函数执行状态

操作：

①如果线性表未初始化，返回INFEASIBLE

②线性表已经初始化，定义文件指针并指向打开文件

a.若打开失败，返回ERROR

b.打开成功，将元素依次写入，并关闭文件指针，返回OK

时间复杂度：O（n）

空间复杂度：O（1）

(12)读取某文件中的数据到一空线性表

函数名称：status LoadList（SqList \*L，char FileName[]）

输入：线性表指针，文件名称

输出：函数执行状态

思路：如果线性表已经初始化，将文件中的数据读入线性表，否则返回相应的函数执行状态

操作：

①如果线性表未初始化，返回INFEASIBLE

②线性表已经初始化，定义文件指针并指向打开文件

a.若打开失败，返回ERROR

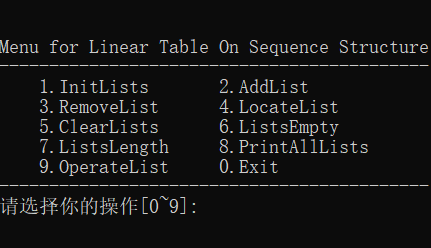
b.打开成功，将元素依次读取，并关闭文件指针，返回OK

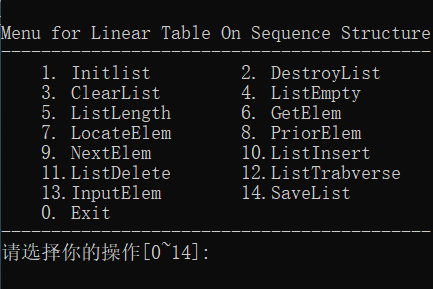
时间复杂度：O（n）

空间复杂度：O（1）

**1.4 系统测试**

程序采用较为简单的界面，一级菜单和二级菜单分别如图1-6，1-7所示（在一级菜单中可通过操作9进入二级菜单）。本次系统测试选取了针对线性表集合操作的AddList、RemoveList、LocateList函数以及针对单线性表操作的ClearList、ListEmpty、GetElem、LocateElem、PriorElem、NextElem、ListInsert、ListDelete、ListTrabverse、SaveList、LoadList函数。

图1-6 一级菜单示意图****

****图1-7 二级菜单示意图

下面是函数测试：

(1)添加线性表的测试：

测试用例及结果如表1-1所示

表1-1添加线性表测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1(集合中已有aaa线性表） | 名称：aaa | 名称有误！请重新输入！ | e2273fdeb0c4ef39b65407f57ddbc26  3cf809dffb8ee4bda6c55e8735bfd48 |
| 用例2（空集合） | 名称：bbb | 添加成功！ | 80548dbdfa1aac183a3c0d1ce7349ddb91b4f295e533bf0c9b5dc50a35a6a3  b91b4f295e533bf0c9b5dc50a35a6a3 |

综合上述测试，添加线性表功能对于输出名称重复时以及未重复时都可以正确处理，所以添加线性表功能符合实验要求。

(2)删除线性表的测试：

测试用例及结果如表1-2所示

表1-2删除线性表测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1(集合中已有aaa线性表） | 名称：aaa | 删除成功！ | e7400a5ee765f6be5cbf7a018ef9c3f  2a70f9f216a3839783fff1d3621f097 |
| 用例2(集合中已有aaa线性表） | 名称：bbb | 线性表集合中没有该线性表！ | 63f02528cd1c27e1d4aa1768d7c8be1  24f994e3e54d61d44e7ded242507c69 |
| 用例3(空集合） | 名称：aaa | 线性表集合中没有该线性表！ | 9e63d5f90a613e761fe4454215aa84f  2a70f9f216a3839783fff1d3621f097 |

综合上述测试，删除线性表功能对于线性表集合中有或无对应名称的线性表时都可以正确处理，所以删除线性表功能符合实验要求。

(3)定位线性表的测试

测试用例及结果如表1-3所示

表1-3定位线性表测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空集合） | 名称：aaa | 线性表集合中没有该线性表！ | 50d5fce15f1ecbc42e75bdfd7ef35dd |
| 用例2(集合中已有aaa，bbb线性表） | 名称：bbb | 该线性表位置是2 | 49a762d8f9592ba37fafb7260e3b291  e9238b583a521de4bc43743eeee2469 |
| 用例3(集合中已有aaa，bbb线性表） | 名称：ccc | 线性表集合中没有该线性表！ | e9238b583a521de4bc43743eeee2469  7e6dc5eab9601ee857085efa8dda46e |

综合上述测试，定位线性表功能对于线性表集合中有或无对应名称的线性表时都可以正确处理，所以定位线性表功能符合实验要求

(4)清空线性表的测试

测试用例及结果如表1-4所示

表1-4清空线性表测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空线性表） | 空线性表 | 线性表为空！ | 5dfdf50830d1dce4589affa96a727f1 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | a96036fbf39f2c73bced693a4fd4338 |
| 用例3(s={1,2,3}) | s={1,2,3} | 清空成功！ | f6dd5028a03fa18cef0758a96f272693a42da9dfe0f0b033ee531b4596cc3b |

综合上述测试，清空线性表功能对于线性表是否初始化、是否是空表都可以正确处理，所以清空线性表功能符合实验要求

(5)线性表判空的测试

测试用例及结果如表1-5所示

表1-5线性表判空测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空线性表） | 空线性表 | 线性表为空！ | ff6ecc4535c7cb5d6068c70fc83fe47 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | ecce4443db698ac5036bcb947f011f7 |
| 用例3(s={1,2,3}) | s={1,2,3} | 线性表不为空！ | a033e69c9839d964a9137ed3f23bcd2 |

综合上述测试，线性表判空功能对于线性表是否初始化、是否是空表都可以正确处理，所以线性表判空功能符合实验要求

(6)获取元素的测试

测试用例及结果如表1-6所示

表1-6获取元素测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 2 | 第2号元素为2！ | 91b8c80c73d5d2cb27bd75fc1f97f86 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | 991262e4842dab102c5417f8e59ffe0 |
| 用例3(s={1,2,3}) | 4 | 输入位置有误！ | 83179bb797611c19e13bfe08f3af694 |

综合上述测试，获取元素功能对于线性表是否初始化、输入位置是否正确都可以正确处理，所以获取元素功能符合实验要求

(7)查找元素的测试

测试用例及结果如表1-7所示

表1-7查找元素测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 4 | 线性表中没有该元素！ | 7acd6e3b0f605d4231561db5b48d00b |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | e673c204d186154ef57630479bdd1a8 |
| 用例3(s={1,2,3}) | 3 | 该元素在线性表中位置为3！ | e5be10005492f1fc61142584e647f4b |

综合上述测试，查找元素功能对于线性表是否初始化、输入元素是否在线性表中都可以正确处理，所以查找元素功能符合实验要求

(8)获取某元素的前驱的测试

测试用例及结果如表1-8所示

表1-8获取某元素的前驱测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 1 | 该元素没有前驱！ | 5c815b2b9c6ca42fb677803720cb5f7 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | 9940ff794eb3e12a0e736253b148671 |
| 用例3(s={1,2,3}) | 3 | 该元素的前驱为2！ | 0a5e8e420be8e69adeb1edb56334d0c |
| 用例4(s={1,2,3}) | 6 | 该元素没有前驱！ | 02f8f4f3d2e7d7e7c4d29bcf5f74dc8 |

综合上述测试，获取某元素前驱功能对于线性表是否初始化、输入不同位置的元素都可以正确处理，所以获取某元素前驱功能符合实验要求

(8)获取某元素的后继的测试

测试用例及结果如表1-9所示

表1-9获取某元素的后继测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 1 | 该元素的后继为2！ | 5983f1a01e18148f831c35e434a2f88 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | 2f910e717222859604f70b52b3afc51 |
| 用例3(s={1,2,3}) | 3 | 该元素没有后继！ | 7491a6fcfa50174884a18a12c9c86e7 |
| 用例4(s={1,2,3}) | 6 | 该元素没有后继！ | 6402b46b9630438482be41b41354cc8 |

综合上述测试，获取某元素后继功能对于线性表是否初始化、输入不同位置的元素都可以正确处理，所以获取某元素后继功能符合实验要求

(10)插入元素的测试

测试用例及结果如表1-10所示

表1-10插入元素测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 位置：1  元素：4 | 插入成功！ | bb37fed00fbe948cdbebaaa563897b3  26449750b5e4e3a72a41825f60b8c97 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | d014ab5d54d241600535026dc8224c2 |
| 用例3(s={1,2,3}) | 位置：5 | 插入位置有误！ | 4d6002677040756bc5ca7c853b14cc1 |

综合上述测试，插入元素功能对于线性表是否初始化、插入位置是否正确都可以正确处理，所以插入元素功能符合实验要求

(11)删除元素的测试

测试用例及结果如表1-11所示

表1-11删除元素测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 位置：1 | 删除成功，删除的元素是1！ | 5dbd51ba42c5287c1b80dcd7ee084b8 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | e51745de22038d521e201360112def6 |
| 用例3(s={1,2,3}) | 位置：5 | 删除位置有误！ | f7b3f631ba627fdc77d76792980a366 |

综合上述测试，删除元素功能对于线性表是否初始化、删除位置是否正确都可以正确处理，所以删除元素功能符合实验要求

(12)打印元素的测试

测试用例及结果如表1-12所示

表1-12打印元素测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 非空线性表 | 元素依次为：  1 2 3 | 4093ee88eeb49bc07eae72c2f3e9988 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | 36a5d4bed6a1a28c81998263a768072 |

综合上述测试，打印元素功能对于线性表是否初始化都可以正确处理，所以打印元素功能符合实验要求

(13)写入文件的测试

测试用例及结果如表1-13所示

表1-13写入文件测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | D:\code\test.txt | 写入成功 | caa090527fcb13b4d34429a4ef9b345 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | f1700b02c922b06a702b26309515442 |

综合上述测试，写入文件功能对于线性表是否初始化都可以正确处理，所以写入文件功能符合实验要求

(14)读取文件的测试

测试用例及结果如表1-14所示

表1-14读取文件测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空表） | D:\code\test.txt | 读入成功 | 927708d1d37a15a9c086e146607cbf3 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | b1722ee0f1fd212aed72ce36f60bcfc |
| 用例3 | 11 | 读入失败 | c7b65a10d8aa870fc352212da64aa26 |
| 用例4（非空表） | D:\code\test.txt | 线性表中已经有元素 | e028bdef757774dde00b52c8767c6a2 |

综合上述测试，读入文件功能对于线性表是否初始化，文件名是否正确，线性表中是否有元素都可以正确处理，所以读入文件功能符合实验要求

总结：综合上述所有函数的测试，测试结果都未发现问题，符合本次实验的要求，也较完整较正确地完成了题目要求。

**1.5 实验小结**

本次实验框架在提供的资料里面已经写好，只需要我们自己将所有的函数补充完整，这样不仅有针对性地练习了线性表的相关操作，也训练了我们的代码集成能力。

同时，在调试自己代码的时候，也出现了很多的问题：

①从前在使用释放内存的free函数之后，就不再处理原来的指针，本次实验必须要将释放内存后的指针指向NULL，这样也让指针的使用更加安全。

②我们写的函数里面，往往有多个返回值。第一次集成代码的时候，我将函数都放到了if，else if后的括号里面，这样就导致，当if判断完之后，相当于函数执行了一次，但当进入else if之后，就会再执行一次，这样执行多次，就会出现错误。最后改进方法是在进入分支语句之前，用变量来储存函数返回值，之后就只需要判断该变量的值就可以了。

③对文件的处理不太熟悉，导致刚开始无法将数据从文件中读取出来，最后又学习了一下相关函数，解决了相关问题。

④有一些储存执行状态的变量（如flag，k等）在执行一次之后没能恢复到原来的值，导致如果进行第二次操作的时候就会出现错误。

通过这次实验让我意识到自己还有很多地方需要改进，比如不够细心、不够严谨、考虑问题不够全面等，我会更加努力地改进与提高。

最后很感谢老师与助教对我的问题的及时解答，帮助我比较顺利地完成了这次实验。

**2 基于链式储存结构的线性表实现**

**2.1 问题描述**

构造链表，呈现一个简易菜单的功能演示系统，该演示系统可选择实现多个线性表管理以及单线性表的操作。

**2.1.1 实验目的**

(1)加深对线性表的概念、基本运算的理解；

(2)熟练掌握线性表的逻辑结构与物理结构的关系；

(3)物理结构采用链表,熟练掌握线性表的基本运算的实现。

**2.1.2 实验要求**

需要在主程序中完成函数调用以及所需实参值和函数执行结果的输出。定义线性表及线性表集合的初始化、销毁、清空、判空、求表长和获得元素等函数，并给出适当的操作提示，并且可选择以文件的形式进行存储和加载。

**2.2 系统设计**

**2.2.1 整体系统结构设计**

本演示系统可以实现通过操作菜单对多线性表进行管理，并对单个线性表进行操作，也支持通过文件的写入与读取。

多线性表管理菜单（线性表集合菜单/一级菜单）可供选择的操作有：初始化线性表集合、添加一个线性表、删除特定名称的线性表、查找特定名称的线性表、清空线性表集合、线性表集合判空、求线性表集合中线性表的个数、打印所有线性表的序号及名称、操作特定序号的线性表。

多线性表管理菜单中的“操作特定序号的线性表”可以调出单线性表操作菜单（二级菜单），可供选择的操作有：初始化线性表、销毁线性表、清空线性表、线性表判空、求线性表长度、取出特定位置的元素、定位某元素、求某元素的前驱、求某元素的后继、在特定位置插入元素、删除特定位置的元素、打印线性表中的所有元素、将线性表的数据写入文件、通过键盘输入或者读取文件数据向线性表中一次性输入若干元素。整体系统结构设计示意图如下：

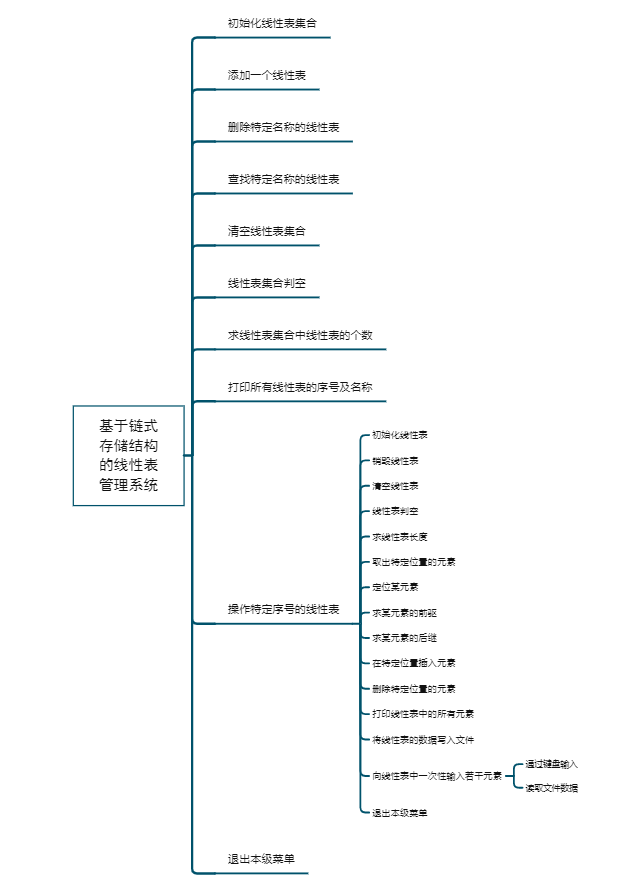
****

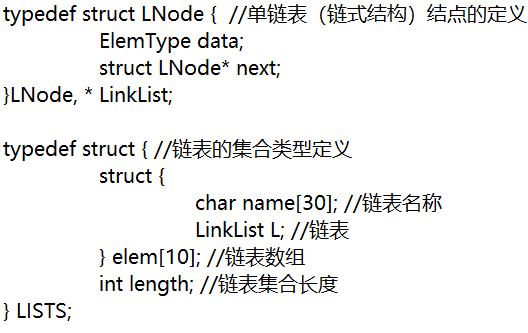
图2-1 整体系统结构设计

**2.2.2 数据结构设计**

本演示系统设计了两个物理结构，分别是线性表和线性表集合，两者均以结构形式定义。

线性表结构中包括数据与指向下一个结点的指针。线性表集合结构中包括线性表数组（每个数组元素都包括线性表以及其名称）和线性表集合长度。

结构定义如下：

图2-2 物理结构设计

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，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()。

**2.2.3 有关常量和类型定义**

数据元素类型的定义：

typedef int status;

typedef int ElemType;

有关常量的定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

**2.3 系统实现**

本演示系统在Windows环境下，使用Visual Studio 2019完成。演示系统涉及的部分主要函数如下，其中（1）-（4）为对线性表集合的操作，（5）-（12）为对单线性表的操作。

(1)向线性表集合中添加一个空线性表

函数名称：status AddList(LISTS \*Lists,char ListName[])

输入：线性表集合指针以及需要添加的线性表的名称

输出：函数执行状态

思想：在Lists中加入一个名称为ListName的线性表，成功则返回OK

操作：

将ListName数组拷贝到线性表集合中线性表数组的最后一个元素之后

的元素的name成员变量中；将新添加的线性表置空；线性表集合长度加一；

结束，返回OK。

时间复杂度：O（1）

空间复杂度：O（1）

(2)删除线性表集合中指定名称的线性表

函数名称：status RemoveList(LISTS \*Lists,char ListName[])

输入：线性表集合指针以及需要删除的线性表名称

输出：函数执行状态

思想：删除指定名称的线性表，删除成功则返回OK，否则返回ERROR

操作：

定义i=0，当i小于线性表集合长度时，执行下列循环：

a.比较第i个线性表的名称和所给名称

b.当两者相同时，将第i个线性表的名称指针的首元素置为’\0’,并使链表指针分配的空间用free函数收回，并使链表指针指向NULL，将线性表数组中该线性表后面的线性表依次前移，线性表集合的长度减一，返回OK

若循环结束后函数没有返回值，说明集合中没有该名称的线性表，返回ERROR

时间复杂度：O（n）(设线性表集合长度为n)

空间复杂度：O（1）

(3)查找指定名称的线性表在集合中的位置

函数名称：status LocateList(LISTS Lists, char ListName[])

输入：线性表集合以及需要查找的线性表名称

输出：该线性表的序号或者函数执行状态  
思想：在集合中寻找指定名称的线性表，成功则返回其逻辑序号，否

则返回ERROR

操作：

① 定义i=0，当i小于线性表集合长度时，执行下列循环：

a.比较第i个线性表的名称与所给名称

b.如果相同，返回i+1，否则继续循环

② 如果循环执行完后函数没有返回值，说明集合中没有该名称的线性表，

返回ERROR

时间复杂度：O（n） （设线性表长度为n，此处为最坏情况）

空间复杂度：O（1）

(4)清空线性表集合

函数名称：status ClearLists(LISTS \*Lists)

输入：线性表集合指针

输出：函数执行状态

思想：将线性表集合的长度变成0，返回OK

操作：将Lists的length赋值为0，返回OK

时间复杂度：O（1）

空间复杂度：O（1）

(5)销毁线性表

函数名称：status DestroyList(LinkList \*L)

输入：线性表指针

输出：函数执行状态

思想：如果线性表已经初始化，销毁线性表，释放每一个结点分配的空间，

返回OK，否则返回INFEASIBLE

操作：如果线性表未初始化（L指向NULL），返回INFEASIBLE，定义两个

指针q,p分别指向头结点和首结点。之后循环，当p不为空时，让q指向p的下一个结点并释放p，把q赋值给p。循环结束后释放头结点L，将L指向NULL，返回OK。

时间复杂度：O（n）

空间复杂度：O（1）

(6)清空线性表

函数名称：status ClearList（LinkList \*L）

输入：线性表指针

输出：函数执行状态

思想：如果线性表已经初始化，释放除头结点之外的所有结点，以此来达到

删除所有元素的目的。

操作：如果线性表未初始化（L指向NULL），返回INFEASIBLE，定义两个

指针q,p分别指向头结点和首结点。之后循环，当p不为空时，让q指向p的下一个结点并释放p，把q赋值给p。循环结束后将首结点指向NULL，返回OK。

时间复杂度：O（n）

空间复杂度：O（1）

(7)获取线性表特定位置的元素

函数名称：status GetElem（LinkList L,int i,ElemType \*e）

输入：线性表，需要获取的元素的位置，存放该元素的变量指针

输出：函数执行状态

思想：如果线性表已经初始化，获取线性表的第i个元素，保存在e中，返

OK，如果i不合理，返回ERROR；如果线性表未初始化，返回INFEASIBLE

操作：如果线性表已经初始化，定义一指针指向首结点，当未到达所需位置时且p不指向NULL时，将指针后移，循环以上操作。循环结束后将元素保存在e中，并返回OK，如果i不合理（i<1或者i>线性表长度），返回ERROR；如果线性表未初始化（L指向NULL），返回INFEASIBLE

时间复杂度：O（n）

空间复杂度：O（1）

(8)确定某元素在线性表中的位置

函数名称：status LocateElem(LinkList L,ElemType e)

输入：线性表，某元素

输出：元素位置或者函数执行状态

思想：若线性表已经初始化，查找元素位置并返回序号，查找不成功时返回相应的函数执行状态

操作：

①如果线性表未初始化，返回INFEASIBLE

②线性表已经初始化，定义i=1和指针p指向首结点，执行下列循环：

当p不指向NULL且p的数据域不为e时，i自增。

③如果循环结束后p指向NULL，说明线性表中没有该元素，返回ERROR，否则返回i

时间复杂度：O（n）

空间复杂度：O（1）

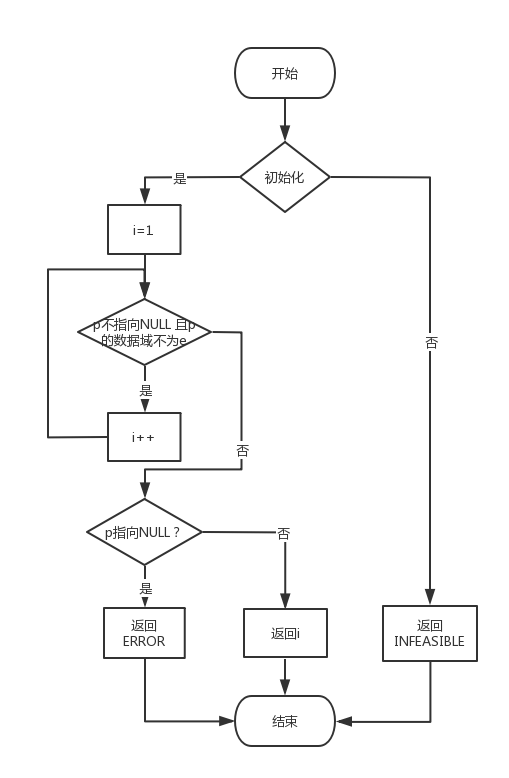


图2-3 LocateElem流程图

(9)在某位置插入某元素

函数名称：status ListInsert（LinkList \*L，int i,ElemType e）

输入：线性表指针，插入位置，插入元素

输出：函数执行状态

思想：若线性表已经初始化，且插入位置正确，则插入元素，返回OK，否则返回相应的函数执行状态

操作：

①如果线性表未初始化，返回INFEASIBLE

②线性表已经初始化：

a. 如果插入位置不正确（i < 1 或 i > 线性表长度加一)，返回ERROR

b. 插入位置正确时，定义k=0，指针p指向首结点，指针q指向头结点，当k小于i-1且p不为空时，执行循环：将p赋值给q，将p的next赋值给p，让k++。循环结束后创建新结点s，将e赋值给s的数据域，将s赋值给q的next，将p赋值给s的next，返回OK。

时间复杂度：O（n）

空间复杂度：O（1）

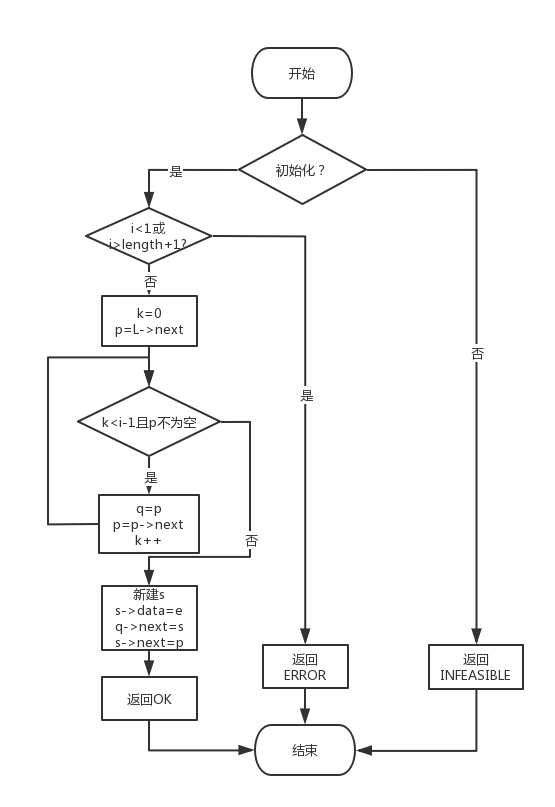


图2-4 ListInsert流程图

(10)删除某位置元素

函数名称：status ListDelete（LinkList \*L，int i，ElemType \*e）

输入：线性表指针，删除位置，储存删除元素的变量指针

输出：函数执行状态

思想：若线性表已经初始化且删除位置正确，则删除对应位置元素，返OK，

否则返回相应的函数执行状态

操作：

①如果线性表未初始化，返回INFEASIBLE

②线性表未初始化：

a. 如果插入位置不正确（i < 1 或 i > 线性表长度)，返回ERROR

b. 插入位置正确时，定义k=0，指针p指向首结点，指针q指向头结点，当k小于i-1且p不为空时，执行循环：将p赋值给q，将p的next赋值给p，让k++。循环结束后将p的data赋值给e，将p的next赋值给q的next，释放p,返回OK。

时间复杂度：O（n）

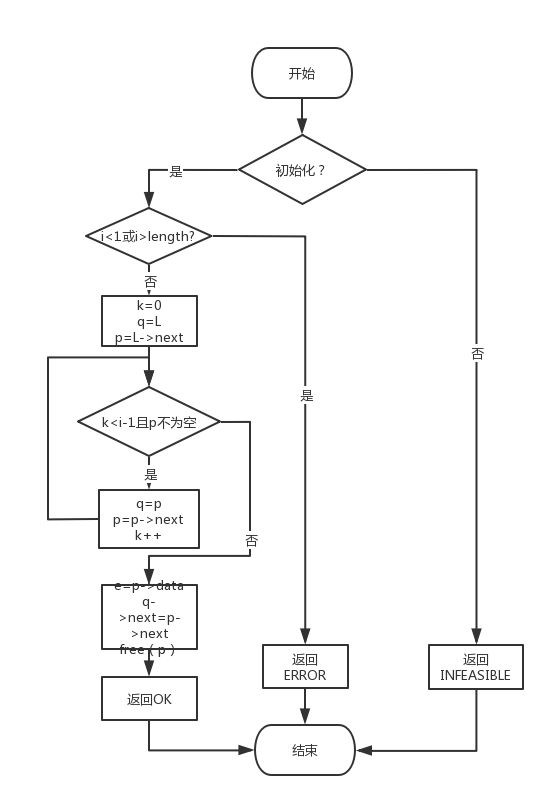
空间复杂度：O（1）

图2-5 ListDelete流程图

(11)将线性表的元素写入某文件

函数名称：status SaveList（LinkList L，char FileName[]）

输入：线性表，文件名称

输出：函数执行状态

思路：如果线性表已经初始化，将元素写入文件，否则返回对应的函数执行状态

操作：

①如果线性表未初始化，返回INFEASIBLE

②线性表已经初始化，定义文件指针并指向打开文件

a.若打开失败，返回ERROR

b.打开成功，将文件依次写入，并关闭文件指针，返回OK

时间复杂度：O（n）

空间复杂度：O（1）

(12)读取某文件中的数据到一空线性表

函数名称：status LoadList（LinkList \*L，char FileName[]）

输入：线性表指针，文件名称

输出：函数执行状态

思路：如果线性表已经初始化，将文件中的数据读入线性表，否则返回相应的函数执行状态

操作：

①如果线性表未初始化，返回INFEASIBLE

②线性表已经初始化，定义文件指针并指向打开文件

a.若打开失败，返回ERROR

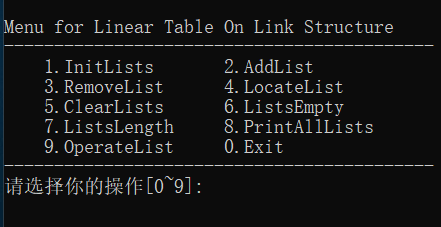
b.打开成功，将元素依次读取，并关闭文件指针，返回OK

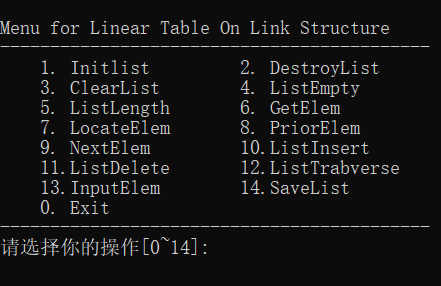
时间复杂度：O（n）

空间复杂度：O（1）

**2.4 系统测试**

程序采用较为简单的界面，一级菜单和二级菜单分别如图2-6，2-7所示（在一级菜单中可通过操作9进入二级菜单）。本次系统测试选取了针对线性表集合操作的AddList、RemoveList、LocateList函数以及针对单线性表操作的ClearList、ListEmpty、GetElem、LocateElem、PriorElem、NextElem、ListInsert、ListDelete、ListTrabverse、SaveList、LoadList函数。

图2-6 一级菜单示意图

图2-7 二级菜单示意图

下面是函数测试：

(1)添加线性表的测试：

测试用例及结果如表2-1所示

表2-1添加线性表测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1(集合中已有aaa线性表） | 名称：aaa | 名称有误！请重新输入！ | e2273fdeb0c4ef39b65407f57ddbc26  3cf809dffb8ee4bda6c55e8735bfd48 |
| 用例2（空集合） | 名称：bbb | 添加成功！ | 80548dbdfa1aac183a3c0d1ce7349ddb91b4f295e533bf0c9b5dc50a35a6a3  b91b4f295e533bf0c9b5dc50a35a6a3 |

综合上述测试，添加线性表功能对于输出名称重复时以及未重复时都可以正确处理，所以添加线性表功能符合实验要求。

(2)删除线性表的测试：

测试用例及结果如表2-2所示

表2-2删除线性表测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1(集合中已有aaa线性表） | 名称：aaa | 删除成功！ | e7400a5ee765f6be5cbf7a018ef9c3f  2a70f9f216a3839783fff1d3621f097 |
| 用例2(集合中已有aaa线性表） | 名称：bbb | 线性表集合中没有该线性表！ | 63f02528cd1c27e1d4aa1768d7c8be1  24f994e3e54d61d44e7ded242507c69 |
| 用例3(空集合） | 名称：aaa | 线性表集合中没有该线性表！ | 9e63d5f90a613e761fe4454215aa84f  2a70f9f216a3839783fff1d3621f097 |

综合上述测试，删除线性表功能对于线性表集合中有或无对应名称的线性表时都可以正确处理，所以删除线性表功能符合实验要求。

(3)定位线性表的测试

测试用例及结果如表2-3所示

表2-3定位线性表测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空集合） | 名称：aaa | 线性表集合中没有该线性表！ | 50d5fce15f1ecbc42e75bdfd7ef35dd |
| 用例2(集合中已有aaa，bbb线性表） | 名称：bbb | 该线性表位置是2 | 49a762d8f9592ba37fafb7260e3b291  e9238b583a521de4bc43743eeee2469 |
| 用例3(集合中已有aaa，bbb线性表） | 名称：ccc | 线性表集合中没有该线性表！ | e9238b583a521de4bc43743eeee2469  7e6dc5eab9601ee857085efa8dda46e |

综合上述测试，定位线性表功能对于线性表集合中有或无对应名称的线性表时都可以正确处理，所以定位线性表功能符合实验要求

(4)清空线性表的测试

测试用例及结果如表2-4所示

表2-4清空线性表测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空线性表） | 空线性表 | 线性表为空！ | 5dfdf50830d1dce4589affa96a727f1 |
| 用例2(s={1,2,3}) | s={1,2,3} | 清空成功！ | f6dd5028a03fa18cef0758a96f272693a42da9dfe0f0b033ee531b4596cc3b |
| 用例3(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | a96036fbf39f2c73bced693a4fd4338 |

综合上述测试，清空线性表功能对于线性表是否初始化、是否是空表都可以正确处理，所以清空线性表功能符合实验要求

(5)线性表判空的测试

测试用例及结果如表2-5所示

表2-5线性表判空测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空线性表） | 空线性表 | 线性表为空！ | ff6ecc4535c7cb5d6068c70fc83fe47 |
| 用例2(s={1,2,3}) | s={1,2,3} | 线性表不为空！ | a033e69c9839d964a9137ed3f23bcd2 |
| 用例3(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | ecce4443db698ac5036bcb947f011f7 |

综合上述测试，线性表判空功能对于线性表是否初始化、是否是空表都可以正确处理，所以线性表判空功能符合实验要求

(6)获取元素的测试

测试用例及结果如表2-6所示

表2-6获取元素测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 2 | 第2号元素为2！ | 91b8c80c73d5d2cb27bd75fc1f97f86 |
| 用例2(s={1,2,3}) | 4 | 输入位置有误！ | 83179bb797611c19e13bfe08f3af694 |
| 用例3(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | 991262e4842dab102c5417f8e59ffe0 |

综合上述测试，获取元素功能对于线性表是否初始化、输入位置是否正确都可以正确处理，所以获取元素功能符合实验要求

(7)查找元素的测试

测试用例及结果如表2-7所示

表2-7查找元素测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 4 | 线性表中没有该元素！ | 7acd6e3b0f605d4231561db5b48d00b |
| 用例2(s={1,2,3}) | 3 | 该元素在线性表中位置为3！ | e5be10005492f1fc61142584e647f4b |
| 用例3(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | e673c204d186154ef57630479bdd1a8 |

综合上述测试，查找元素功能对于线性表是否初始化、输入元素是否在线性表中都可以正确处理，所以查找元素功能符合实验要求

(8)获取某元素的前驱的测试

测试用例及结果如表2-8所示

表2-8获取某元素的前驱测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 1 | 该元素没有前驱！ | 5c815b2b9c6ca42fb677803720cb5f7 |
| 用例2(s={1,2,3}) | 3 | 该元素的前驱为2！ | 0a5e8e420be8e69adeb1edb56334d0c |
| 用例3(s={1,2,3}) | 6 | 该元素没有前驱！ | 02f8f4f3d2e7d7e7c4d29bcf5f74dc8 |
| 用例4(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | 9940ff794eb3e12a0e736253b148671 |

综合上述测试，获取某元素前驱功能对于线性表是否初始化、输入不同位置的元素都可以正确处理，所以获取某元素前驱功能符合实验要求

(8)获取某元素的后继的测试

测试用例及结果如表2-9所示

表2-9获取某元素的后继测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 1 | 该元素的后继为2！ | 5983f1a01e18148f831c35e434a2f88 |
| 用例2(s={1,2,3}) | 3 | 该元素没有后继！ | 7491a6fcfa50174884a18a12c9c86e7 |
| 用例3(s={1,2,3}) | 6 | 该元素没有后继！ | 6402b46b9630438482be41b41354cc8 |
| 用例4(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | 2f910e717222859604f70b52b3afc51 |

综合上述测试，获取某元素后继功能对于线性表是否初始化、输入不同位置的元素都可以正确处理，所以获取某元素后继功能符合实验要求

(10)插入元素的测试

测试用例及结果如表2-10所示

表2-10插入元素测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 位置：1  元素：4 | 插入成功！ | bb37fed00fbe948cdbebaaa563897b3  26449750b5e4e3a72a41825f60b8c97 |
| 用例2(s={1,2,3}) | 位置：5 | 插入位置有误！ | 4d6002677040756bc5ca7c853b14cc1 |
| 用例3(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | d014ab5d54d241600535026dc8224c2 |

综合上述测试，插入元素功能对于线性表是否初始化、插入位置是否正确都可以正确处理，所以插入元素功能符合实验要求

(11)删除元素的测试

测试用例及结果如表2-11所示

表2-11删除元素测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 位置：1 | 删除成功，删除的元素是1！ | 5dbd51ba42c5287c1b80dcd7ee084b8 |
| 用例2(s={1,2,3}) | 位置：5 | 删除位置有误！ | f7b3f631ba627fdc77d76792980a366 |
| 用例3(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | e51745de22038d521e201360112def6 |

综合上述测试，删除元素功能对于线性表是否初始化、删除位置是否正确都可以正确处理，所以删除元素功能符合实验要求

(12)打印元素的测试

测试用例及结果如表2-12所示

表2-12打印元素测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | 非空线性表 | 元素依次为：  1 2 3 | 4093ee88eeb49bc07eae72c2f3e9988 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | 36a5d4bed6a1a28c81998263a768072 |

综合上述测试，打印元素功能对于线性表是否初始化都可以正确处理，所以打印元素功能符合实验要求

(13)写入文件的测试

测试用例及结果如表2-13所示

表2-13写入文件测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（s={1,2,3}） | D:\code\test.txt | 写入成功 | caa090527fcb13b4d34429a4ef9b345 |
| 用例2(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | f1700b02c922b06a702b26309515442 |

综合上述测试，写入文件功能对于线性表是否初始化都可以正确处理，所以写入文件功能符合实验要求

(14)读取文件的测试

测试用例及结果如表2-14所示

表2-14读取文件测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空表） | D:\code\test.txt | 读入成功 | 927708d1d37a15a9c086e146607cbf3 |
| 用例2 | 11 | 读入失败 | c7b65a10d8aa870fc352212da64aa26 |
| 用例3（非空表） | D:\code\test.txt | 线性表中已经有元素 | e028bdef757774dde00b52c8767c6a2 |
| 用例4(未初始化的线性表） | 未初始化的线性表 | 线性表未初始化！ | b1722ee0f1fd212aed72ce36f60bcfc |

综合上述测试，读入文件功能对于线性表是否初始化，文件名是否正确，线性表中是否有元素都可以正确处理，所以读入文件功能符合实验要求

总结：综合上述所有函数的测试，测试结果都未发现问题，符合本次实验的要求，也较完整较正确地完成了题目要求。

**2.5 实验小结**

本次实验框架在提供的资料里面已经写好，只需要我们自己将所有的函数补充完整，这样不仅有针对性地练习了线性表的相关操作，也训练了我们的代码集成能力。

由于这次的实验与上次的实验相差不大，故这次的实验也比较顺利。

但写代码的时候仍然出现了一些问题：

①编写文件读取函数时，由于循环的问题，在最后实际上多建立了一个空结点，这样在遍历生成的线性表的时候最后总会多出一个0，解决方案是，另添加了一个指向结点的指针r，使这个指针总是指向在循环中新生成的结点。最后在循环结束后，指针p指向的是多出来的空结点，指针r指向的是它的前一个结点，这时只需将p释放掉，并让r->next指向NULL即可。

②有些函数传入的参数是LinkList \*L，由于LinkList定义的是指针类型，故传入的是二级指针，如果要对单链表进行操作，需要先解引用（\*L），再进行单链表的操作。

最后很感谢老师与助教对我的问题的及时解答，帮助我比较顺利地完成了这次实验。

**3 基于二叉链表的二叉树实现**

**3.1 问题描述**

构造二叉树，呈现一个简易菜单的功能演示系统，该演示系统可选择实现多个二叉树管理以及单二叉树的操作。

**3.1.1 实验目的**

(1)加深对二叉树的概念、基本运算的理解；

(2)熟练掌握二叉树的逻辑结构与物理结构的关系；

(3)物理结构采用二叉链表,熟练掌握二叉树的基本运算的实现。

**3.1.2 实验要求**

需要在主程序中完成函数调用以及所需实参值和函数执行结果的输出。定义二叉树的创建、清空、销毁、求深度、定位结点、结点赋值、获得兄弟结点、插入结点、删除结点、遍历等函数及集合的初始化、销毁、清空、判空、求表长和获得元素等函数，并给出适当的操作提示，并且可选择以文件的形式进行存储和加载。

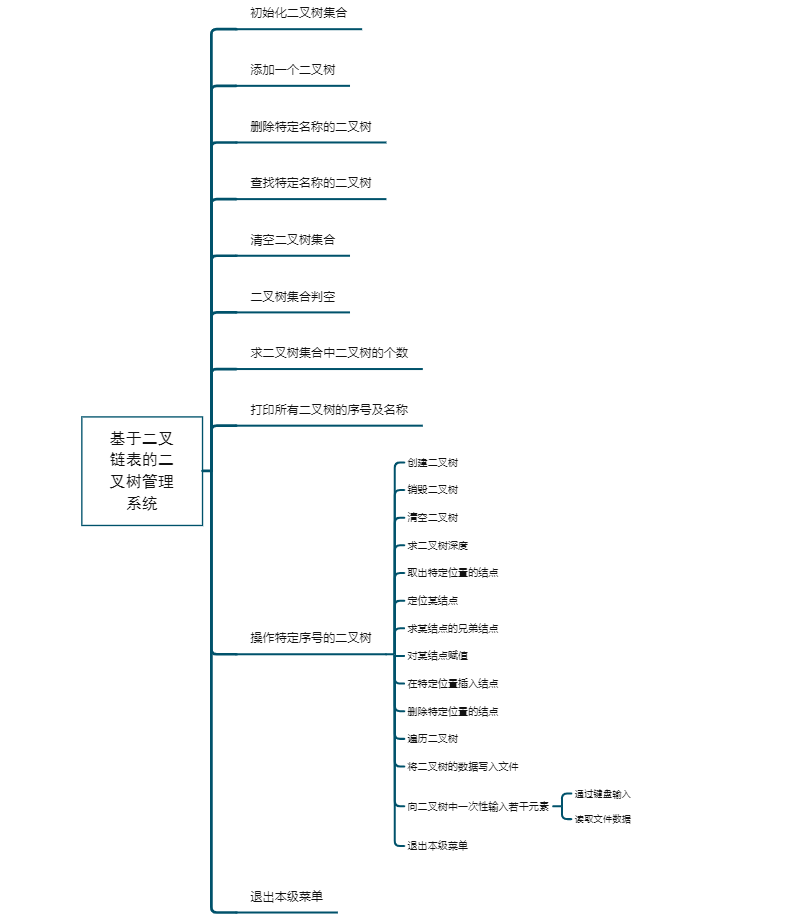
**3.2 系统设计**

**3.2.1 整体系统结构设计**

本演示系统可以实现通过操作菜单对多二叉树进行管理，并对单个二叉树进行操作，也支持通过文件的写入与读取。

多二叉树管理菜单（二叉树集合菜单/一级菜单）可供选择的操作有：初始化二叉树集合、添加一个二叉树、删除特定名称的二叉树、查找特定名称的二叉树、清空二叉树集合、二叉树集合判空、求二叉树集合中二叉树的个数、打印所有二叉树的序号及名称、操作特定序号的二叉树。

多二叉树管理菜单中的“操作特定序号的二叉树”可以调出单二叉树操作菜单（二级菜单），可供选择的操作有：创建二叉树、销毁二叉树、清空二叉树、求二叉树深度、定位某结点、结点赋值、求某结点的兄弟结点、在特定位置插入结点、删除特定位置的结点、遍历二叉树、将二叉树的数据写入文件、通过键盘输入或者读取文件数据向二叉树中一次性输入若干元素。整体系统结构设计示意图如下：

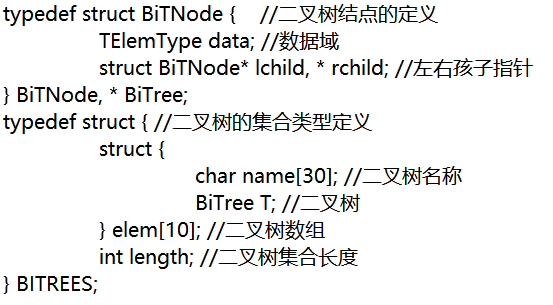
图3-1 整体系统结构设计

**3.2.2 数据结构设计**

本演示系统设计了两个物理结构，分别是二叉树和二叉树集合，两者均以结构形式定义。

二叉树结构中包括数据与指向左孩子和右孩子的指针。二叉树集合结构中包括二叉树数组（每个数组元素都包括二叉树以及其名称）和二叉树集合长度。

结构定义如下：

图3-2 数据结构设计

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。

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。

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。

**3.2.3 有关常量和类型定义**

数据元素类型的定义：

typedef int status;

typedef int KeyType;

typedef struct {

KeyType key;

char others[20];

} TElemType; //二叉树数据类型定义

有关常量的定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

**3.3 系统实现**

本演示系统在Windows环境下，使用Visual Studio 2019完成。演示系统涉及的部分主要函数如下，其中（1）-（4）为对二叉树集合的操作，（5）-（15）为对单二叉树的操作。

(1)向二叉树集合中添加一个空二叉树

函数名称：status AddBiTree(BITREES\* BiTrees, char BiTreeName[])

输入：二叉树集合指针以及需要添加的二叉树的名称

输出：函数执行状态

思想：在Lists中加入一个名称为ListName的二叉树，成功则返回OK

操作：

将ListName数组拷贝到二叉树集合中二叉树数组的最后一个元素之后

的元素的name成员变量中；将新添加的二叉树置空；二叉树集合长度加一；

结束，返回OK。

时间复杂度：O（1）

空间复杂度：O（1）

(2)删除二叉树集合中指定名称的二叉树

函数名称：status RemoveBiTree(BITREES\* BiTrees, char BiTreeName[])

输入：二叉树集合指针以及需要删除的二叉树名称

输出：函数执行状态

思想：删除指定名称的二叉树，删除成功则返回OK，否则返回ERROR

操作：

定义i=0，当i小于二叉树集合长度时，执行下列循环：

a.比较第i个二叉树的名称和所给名称

b.当两者相同时，将第i个二叉树的名称指针的首元素置为’\0’,并使二叉树指针分配的空间用free函数收回，并使二叉树指针指向NULL，将二叉树数组中该二叉树后面的二叉树依次前移，二叉树集合的长度减一，返回OK

若循环结束后函数没有返回值，说明集合中没有该名称的二叉树表，返回ERROR

时间复杂度：O（n）(设二叉树集合长度为n)

空间复杂度：O（1）

(3)查找指定名称的二叉树在集合中的位置

函数名称：status LocateBiTree(BITREES BiTrees, char BiTreeName[])

输入：二叉树集合以及需要查找的二叉树名称

输出：该二叉树的序号或者函数执行状态  
思想：在集合中寻找指定名称的二叉树，成功则返回其逻辑序号，否

则返回ERROR

操作：

① 定义i=0，当i小于二叉树集合长度时，执行下列循环：

a.比较第i个二叉树的名称与所给名称

b.如果相同，返回i+1，否则继续循环

② 如果循环执行完后函数没有返回值，说明集合中没有该名称的二叉树，

返回ERROR

时间复杂度：O（n） （设二叉树长度为n，此处为最坏情况）

空间复杂度：O（1）

(4)清空二叉树集合

函数名称：status ClearBiTrees(BITREES\* BiTrees)

输入：二叉树集合指针

输出：函数执行状态

思想：将二叉树集合的长度变成0，返回OK

操作：将Lists的length赋值为0，返回OK

时间复杂度：O（1）

空间复杂度：O（1）

(5)创建二叉树

函数名称：status CreateBiTree(BiTree& T, TElemType definition[])

输入：二叉树、数据数组

输出：函数执行状态

思想：采用递归思想，根据数据数组中数据的数据按先序创建二叉树

操作：

1. 用一指针p指向数据数组

1.判断数据数组中是否存在关键字重复的情况，若重复，则返回ERROR

2.若此时\*p的关键字为-1，说明创建完成，返回-1

3.如果\*p的关键字为0，说明此处为空，把T置空；如果不为0，将\*p赋值给T的数据域，p后移，再判断关键字是否为-1，是则返回OK，不是则创建左子树，结束后再判断是否为-1，不是则p后移，再判断p关键字是否为-1，不是则创建右子树。

时间复杂度：O（n）

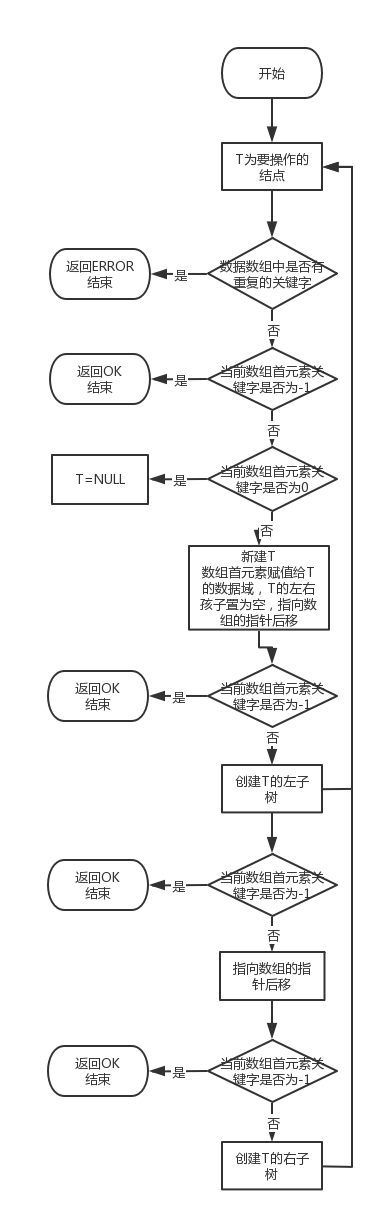
空间复杂度：O（1）

图 3-3 创建二叉树流程图

(6)清空二叉树

函数名称：status ClearBiTree(BiTree& T)

输入：二叉树

输出：函数执行状态

思想：采用递归思想，依次清除每一个结点

操作：如果T为空，返回OK，否则删除其左孩子，完成后删除右孩子，完成后free（T），把T置空，返回OK

时间复杂度：O（n）

空间复杂度：O（1）

(7)求二叉树深度

函数名称：int BiTreeDepth(BiTree T)

输入：二叉树

输出：树的深度

思想：采用递归思想，对于每个结点，求左右子树的深度，返回最大值加一

操作：如果T为空，返回0，否则依次求左子树深度和右子树深度并储存在变量中，返回左右子树深度最大值加一之后的数字

时间复杂度：O（n）

空间复杂度：O（n）

(8)结点赋值

函数名称：status Assign(BiTree& T, KeyType e, TElemType value)

输入：二叉树、待赋值的结点的关键字、赋值数据

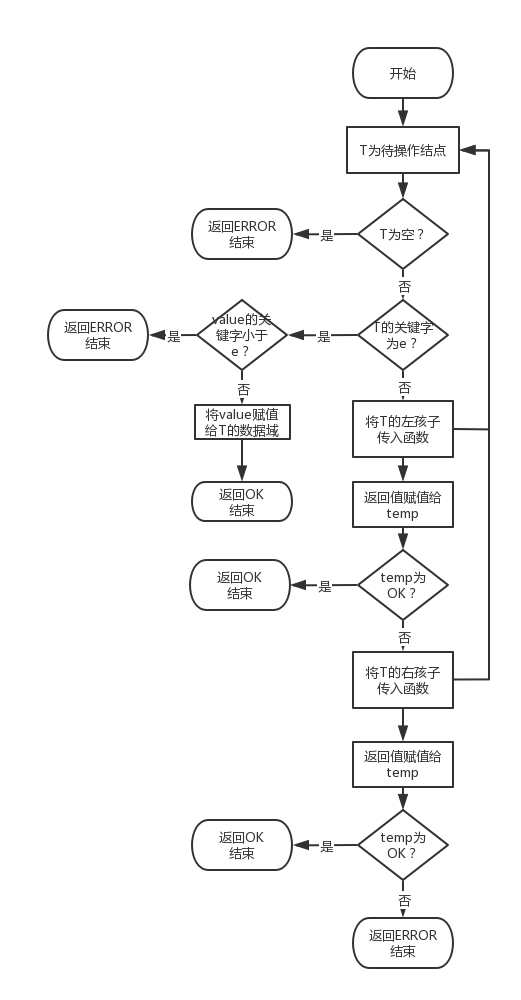
输出：函数执行状态

思想：采用递归思想，遍历寻找关键字为e的结点，若已找到但value.key<e，则会造成关键字重复，不合题意，否则进行赋值；若没有找到，就依次在结点的左右子树寻找。

操作：如果T为空，返回ERROR。若T不为空，且T->data.key==e,如果value.key<e,返回ERROR，否则T->data=value，返回OK；若T->data.key!=e，将该结点左子树传入函数的返回值赋值给temp,如果temp==OK，返回OK，如果等于ERROR，将该结点右子树传入函数的返回值赋值给temp，如果temp==OK，返回OK，否则返回ERROR。

时间复杂度：O（n）

空间复杂度：O（n）

图 3-4 结点赋值流程图

(9)获得兄弟结点

函数名称：BiTNode\* GetSibling(BiTree T, KeyType e)

输入：二叉树，需要获取兄弟结点的结点的关键字

输出：结点指针

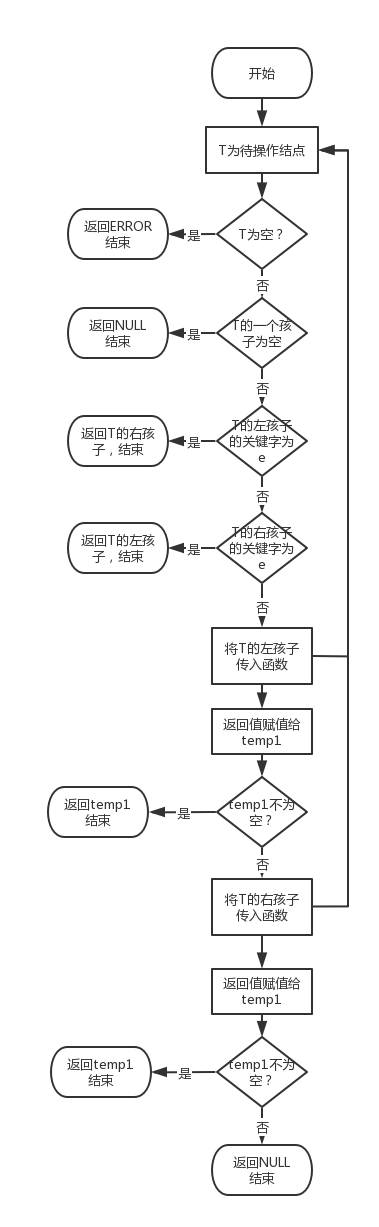
思想：采用递归思想，找到关键字等于e的结点的父结点，返回该父结点的另一个孩子。

操作：如果T为空，返回NULL。如果T有一个孩子为空，就返回NULL；如果T的左孩子的关键字为e，就返回T的右孩子；如果T的右孩子的关键字为e，就返回e的左孩子；否则将该结点的左孩子传入函数的返回值赋值给temp1，如果temp1不为空，就返回temp1，否则将该结点的右孩子传入函数的返回值赋值给temp1，如果temp1不为空，返回temp1，否则返回NULL

时间复杂度：O（n）

空间复杂度：O（n）

流程图见下页

图 3-5 获得兄弟结点流程图

(10)插入结点

函数名称：status InsertNode(BiTree& T,KeyType e,int LR, TElemType c)

输入：二叉树，插入位置的结点的关键字，插入方式，插入数据

输出：函数执行状态

思想：采用递归思想，找到关键字为e的结点，根据传入的数据进行相应方式的插入。

操作：如果T为空，返回ERROR；如果LR=-1且树中存在关键字为e的结点（用其他函数判断），新建结点，将原根结点作为新建结点的右孩子，返回OK；否则如果c.key<=e,说明关键字重复，返回ERROR，如果LR=0且T->data.key=e,新建结点，将新结点作为T的左孩子插入，返回OK，如果LR=1且T->data.key=e,新建结点，将新结点作为T的右孩子插入，返回OK；如果T->data.key!=e，将该结点左子树传入函数的返回值赋值给temp,如果temp==OK，返回OK，如果等于ERROR，将该结点右子树传入函数的返回值赋值给temp，如果temp==OK，返回OK，否则返回ERROR。

时间复杂度：O(n)

空间复杂度：O(n)

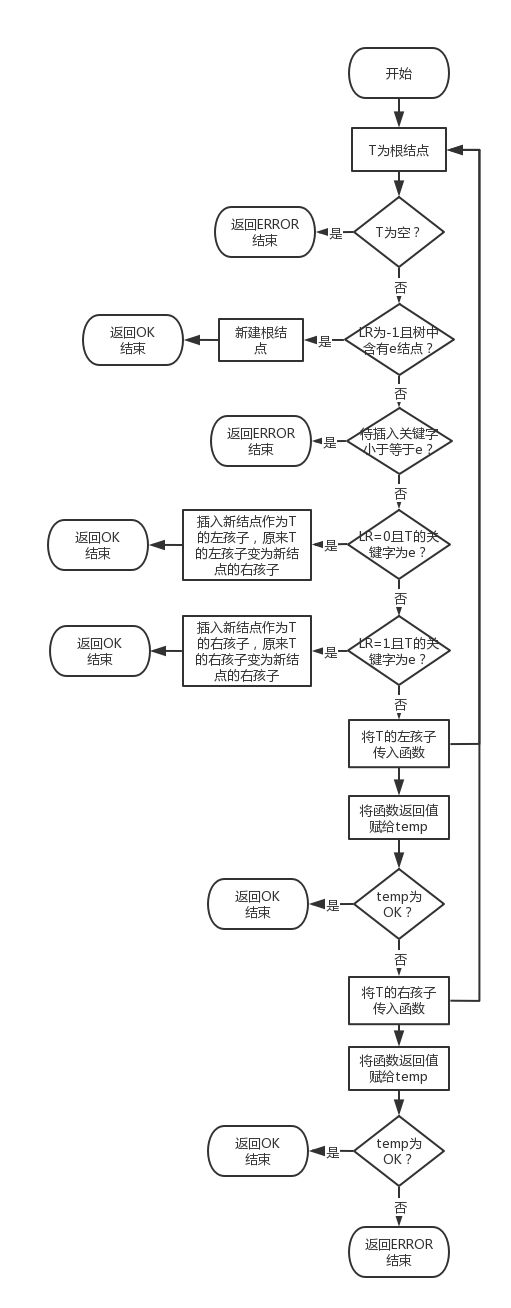


图 3-6 插入结点流程图

(11)删除结点

函数名称：status DeleteNode(BiTree& T, KeyType e)

输入：二叉树，要删除的结点的关键字

输出：函数执行状态

思想：如果根结点为要删除的结点，删除即可，否则找到关键字为e的结点的父结点（用其他函数完成），根据要删除的结点的度，进行不同的删除操作。

操作：如果关键字为e的结点为根结点，直接对根结点进行操作，否则找到关键字为e的结点的父结点，如果没有该结点，返回ERROR，否则对该父结点的关键字为e的孩子进行操作，操作为：如果关键字为e的结点度为0，删除即可；如关键字为e的结点度为1，用关键字为e的结点孩子代替被删除的e位置；如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为e的左子树中最右结点的右子树。 成功删除结点后返回OK，否则返回ERROR。

时间复杂度：O(1)或O(depth）或O(n)或O（n+depth）

空间复杂度：O（1）

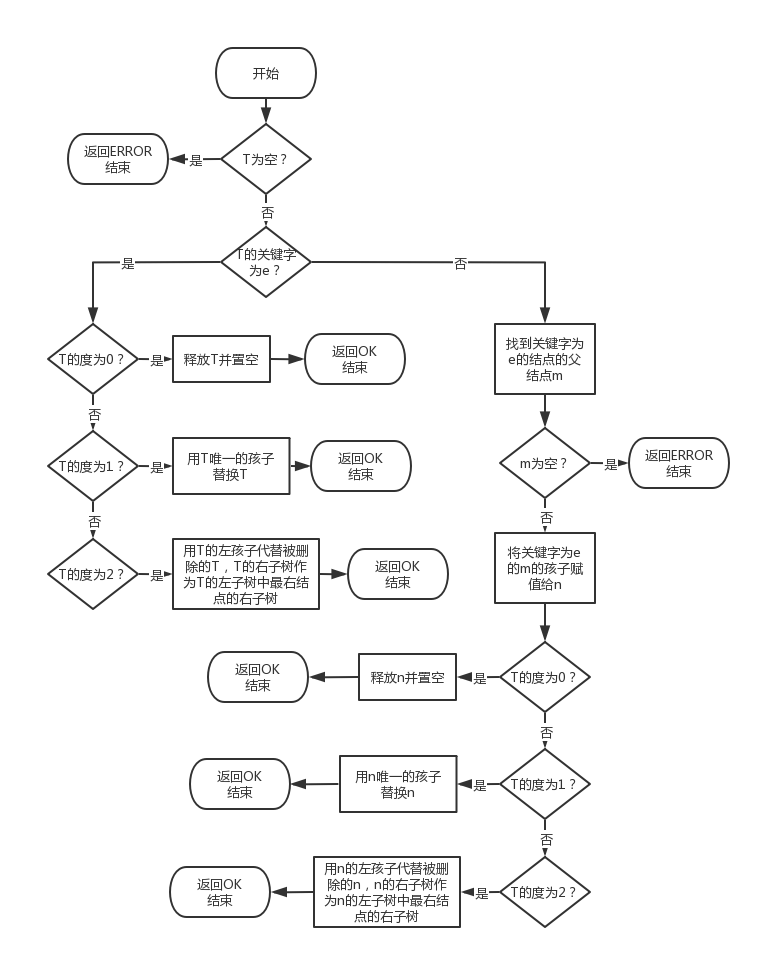


图 3-7 删除结点流程图

(12)先序遍历

函数名称：status PreOrderTraverse(BiTree T, void (\*visit)(BiTree))

输入：二叉树，操作函数指针

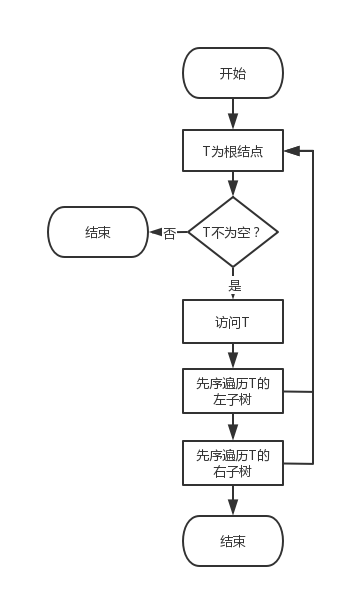
输出：函数执行状态

思想：采用递归思想，依次访问根结点，左子树，右子树

操作：如果T不为空，将T传入操作函数，然后访问其左子树，完成后访问其右子树，完成后返回OK

时间复杂度：O（n）

空间复杂度：O（1）

图 3-8 先序遍历流程图

(13)中序遍历

函数名称：status InOrderTraverse(BiTree T, void (\*visit)(BiTree))

输入：二叉树，操作函数指针

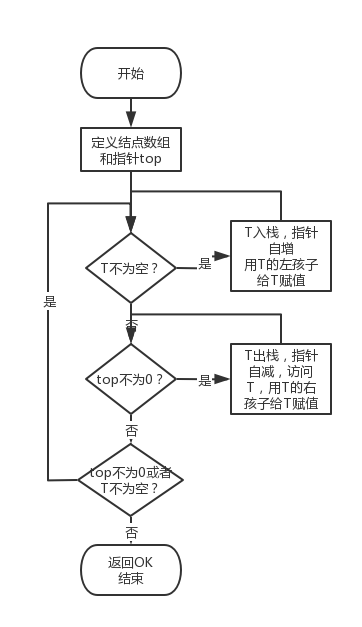
输出：函数执行状态

思想：设置一个栈存放所经过的根结点的信息；第一次访问到根结点时并不访问，而是入栈；中序遍历左子树，左子树遍历结束后，第一次遇到根结点，就将根结点退栈并访问，然后中序遍历它的右子树；当需要退栈时，如果栈为空就结束。

操作：定义一结点指针栈并置空。当T不为空时，T入栈并让T指向其左子树，循环结束后，如果栈非空，就弹出栈顶的根结点T并访问，之后让T指向其右子树，当栈非空或者T不为空时，继续进行以上循环。循环结束后返回OK

时间复杂度：O（n）

空间复杂度：O（1）

图 3-9 中序遍历流程图

(14)后序遍历

函数名称：status PostOrderTraverse(BiTree T, void (\*visit)(BiTree))

输入：二叉树，操作函数指针

输出：函数执行状态

思想：采用递归思想，依次访问根结点，左子树，右子树

操作：如果T不为空，访问其左子树，完成后访问其右子树，完成后将T传入操作函数，返回OK

时间复杂度：O（n）

空间复杂度：O（1）

(15)层序遍历

函数名称：status LevelOrderTraverse(BiTree T, void (\*visit)(BiTree))

输入：二叉树，操作函数指针

输出：函数执行状态

思想：设置一个结点数组，利用快慢指针，快指针遍历所有结点并存入数组，慢指针用于输出当前结点

操作：设置一个结点数组pt，设置快指针in和慢指针out,初始化为0，将根结点赋值给pt[in++],当in>out时执行下列循环：当pt[out]不为空时，访问pt[out],将pt[out]的左孩子赋值给pt[in++],将pt[out]的右孩子赋值给pt[in++]；之后out++,执行下一次循环。循环结束后返回OK

时间复杂度：O（n）

空间复杂度：O（1）

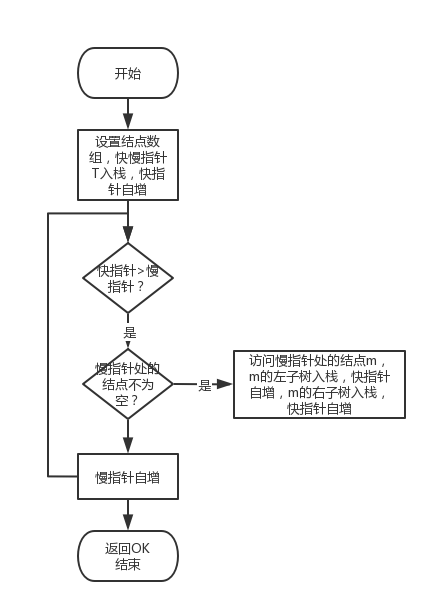
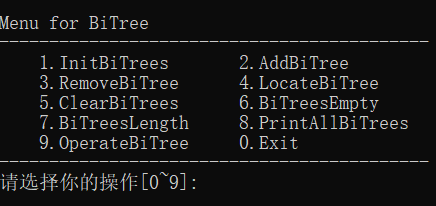
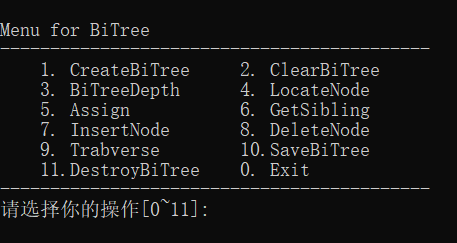


图 3-10 层序遍历流程图

**3.4 系统测试**

程序采用较为简单的界面，一级菜单和二级菜单分别如图3-11，3-12所示（在一级菜单中可通过操作9进入二级菜单）。本次系统测试选取了针对二叉树集合操作的AddBiTree、RemoveBiTree、LocateBiTree函数以及针对单二叉树操作的LocateNode、Assign、GetSibling、InsertNode、DeleteNode、Trabverse函数。

图 3-11 一级菜单示意图

图 3-12 二级菜单示意图

下面是函数测试：

函数测试中使用的非空单二叉树T的先序遍历数据为

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

(1)添加二叉树的测试：

测试用例及结果如表3-1所示

表3-1添加二叉树测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1(集合中已有11二叉树） | 名称：11 | 名称有误！请重新输入！ | 6%`2PKJ@PG3K((S@CR791RJ |
| 用例2（空集合） | 名称：11 | 添加成功！ | 2`NCO%{EFD0W33HRSEZLPBHb91b4f295e533bf0c9b5dc50a35a6a3 |

综合上述测试，添加二叉树功能对于输出名称重复时以及未重复时都可以正确处理，所以添加二叉树功能符合实验要求。

(2)删除二叉树的测试：

测试用例及结果如表3-2所示

表3-2删除二叉树测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1(集合中已有11二叉树） | 名称：11 | 删除成功！ | 9KE3OGTCB14V_T}B8Z05Z[2 |
| 用例2(集合中已有22二叉树） | 名称：11 | 二叉树集合中没有该二叉树！ | _{%9@_7N3{Z@SVPY)8N9I[4 |
| 用例3(空集合） | 名称：11 | 二叉树集合中没有该二叉树！ | _{%9@_7N3{Z@SVPY)8N9I[4 |

综合上述测试，删除二叉树功能对于二叉树集合中有或无对应名称的二叉树时都可以正确处理，所以删除二叉树功能符合实验要求。

(3)定位二叉树的测试

测试用例及结果如表3-3所示

表3-3定位二叉树测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空集合） | 名称：11 | 二叉树集合中没有该二叉树！ | WIVY9OC236T[HLPB8]}HWQ2 |
| 用例2(集合中已有11，22二叉树） | 名称：22 | 该二叉树位置是2 | _A_09(88}0(ID0{3_MLHO]V |
| 用例3(集合中已有11，22二叉树） | 名称：33 | 二叉树集合中没有该二叉树！ | QKCYA8`{HLZF$6O_O0A7JAC |

综合上述测试，定位二叉树功能对于二叉树集合中有或无对应名称的二叉树时都可以正确处理，所以定位二叉树功能符合实验要求.

1. 定位结点的测试

测试用例及结果如表3-4所示

表3-4定位结点测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空二叉树） | 关键字：1 | 二叉树中没有该元素 | BU%L`)G(1W~B`0$564`SQNO |
| 用例2(二叉树T） | 关键字：6 | 二叉树中没有该元素 | ZWQ2U3D481E@JUX@)2CUG0S |
| 用例3(二叉树T） | 关键字：3 | 该结点数据为  3 c | PA7)TI`{`T$JR(()[0NQI1R |

综合上述测试，定位结点功能对于空二叉树，二叉树中有无该元素时都能正确处理，所以定位结点功能符合实验要求。

1. 结点赋值的测试

测试用例及结果如表3-5所示

表3-5结点赋值测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空二叉树） | 关键字：1  数据：2 1 | 赋值失败 | 4G)I3TK%Z@}$VDFI35%HJ$8 |
| 用例2(二叉树T） | 关键字：9  数据：1 1 | 赋值失败 | [BOMSDHEA08%[BY6S`0S]88 |
| 用例3(二叉树T） | 关键字：4  数据：6 l | 赋值成功 | PC9W}@F)S8IW4$V_`L9ZTYB |
| 用例4(二叉树T） | 关键字：2  数据：1 l | 赋值失败 | HRHCGU(7VDZ}_W9[WH_5@60 |

综合上述测试，结点赋值功能对于空二叉树，二叉树中有无该元素，新关键字是否重复时都能正确处理，所以结点赋值功能符合实验要求。

1. 获得兄弟结点的测试

测试用例及结果如表3-6所示

表3-6获得兄弟结点测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空二叉树） | 关键字：1 | 获取失败 | $F2U{{{6@Q{%P6I6(%](RTC |
| 用例2(二叉树T） | 关键字：6 | 获取失败 | NMT%TRD]5FLF7DOJ%$JY6(G |
| 用例3(二叉树T） | 关键字：1 | 获取失败 | 8W8B]5U5~Y3NK]W86N{N$}0 |
| 用例4(二叉树T） | 关键字：2 | 兄弟结点的数据为：3 c | N9`V)1PRYUU%OCUEQS7%V)2 |

综合上述测试，获取兄弟结点功能对于空二叉树，二叉树中有无该元素，该结点是否有兄弟结点时都能正确处理，所以获取兄弟结点功能符合实验要求。

1. 插入结点的测试

测试用例及结果如表3-7所示

表3-7插入结点测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空二叉树） | 1 1 1 1 | 插入失败 | (VY~_K9A02T}2}$C{P_P@]4 |
| 用例2(二叉树T） | 2 -1 2 w | 插入成功 | V~KN2QB(GMR45_3FJ_HIE{G |
| 用例3(二叉树T） | 3 1 9 e | 插入成功 | 3]QB`XT(PA00CH]~85E0])8 |
| 用例4(二叉树T） | 6 1 2 v | 插入失败 | UZS1_D$[}X_A{XL]QVVE~DH |
| 用例5（二叉树T） | 4 0 8 m | 插入成功 | 4PARU5T1[RR_6C[B]2LGHWQ |

综合上述测试，插入结点功能对于空二叉树，二叉树中有无该元素，输入不同插入方式时都能正确处理，所以获取插入结点功能符合实验要求。

1. 删除结点的测试

测试用例及结果如表3-8所示

表3-8删除结点测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空二叉树） | 关键字：1 | 删除失败 | [S2}GFX71E@[@L{)NNZCZB8 |
| 用例2(二叉树T） | 关键字：9 | 删除失败 | )6R3)2S@1YXL`UYJ@0`U3`9 |
| 用例3(二叉树T） | 关键字：4 | 删除成功 | D_8S5[M_4G4C[S`~QYOD1C0 |
| 用例4（只有一个结点1） | 关键字：1 | 删除成功 | 7J~YS$GR27Q1D0P4_AWNCQJ |

综合上述测试，删除结点功能对于空二叉树，二叉树中有无该元素，二叉树中只有根结点时都能正确处理，所以删除结点功能符合实验要求。

1. 遍历二叉树的测试

测试用例及结果如表3-9所示

表3-9遍历二叉树测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空二叉树） | 无 | 二叉树中没有元素！ | UALX{{EVOX3G6E@E[9TEVIR |
| 用例2(二叉树T） | 无 | 输出四种遍历的结果 | 9}}W5}PJ`J9L~XUD~LHFCQP |

综合上述测试，遍历二叉树功能对于空二叉树，非空二叉树都能正确处理，所以遍历二叉树功能符合实验要求。

总结：综合上述所有函数的测试，测试结果都未发现问题，符合本次实验的要求，也较完整较正确地完成了题目要求。

**3.5 实验小结**

本次实验与前两次有很大的不同，而且也有一定的难度。但让我更加了解了递归函数的写法。

同时自己也存在很多问题。

在写创建二叉树的函数时，由于不清楚如何在递归函数中传入数组，导致出现了很多问题。最后使用了一个指向数组的全局指针，通过改变指针的指向来实现数组的依次访问。同时，自己在写删除结点的函数时，没有考虑到只有根结点的情况，导致在验收时出现了问题。而且自己很多次也没有考虑到指针判空的问题，导致调用结点的左孩子或者右孩子但指向空时出现了问题。而且我的文件操作也不太熟练，在同学的指导之下完成了本次实验的文件操作。

最后感谢老师和同学的指导，让我顺利完成本次实验。

**4 基于邻接表的图实现**

**4.1 问题描述**

构造无向图，呈现一个简易菜单的功能演示系统，该演示系统可选择实现多个无向图管理以及单无向图的操作。

**4.1.1 实验目的**

(1)加深对无向图的概念、基本运算的理解；

(2)熟练掌握无向图的逻辑结构与物理结构的关系；

(3)物理结构采用邻接表,熟练掌握无向图的基本运算的实现。

**4.1.2 实验要求**

需要在主程序中完成函数调用以及所需实参值和函数执行结果的输出。定义无向图的创建、销毁、定位顶点、顶点赋值、获得第一邻接结点、获得第二邻接顶点、插入顶点、删除顶点、插入弧、删除弧、深度优先搜索、广度优先搜索等函数及集合的初始化、销毁、清空、判空、求表长和获得元素等函数，并给出适当的操作提示，并且可选择以文件的形式进行存储和加载。

**4.2 系统设计**

**4.2.1 整体系统结构设计**

本演示系统可以实现通过操作菜单对多无向图进行管理，并对单个无向图进行操作，也支持通过文件的写入与读取。

多无向图管理菜单（无向图集合菜单/一级菜单）可供选择的操作有：初始化无向图集合、添加一个无向图、删除特定名称的无向图、查找特定名称的无向图、清空无向图集合、无向图集合判空、求无向图集合中无向图的个数、打印所有无向图的序号及名称、操作特定序号的无向图。

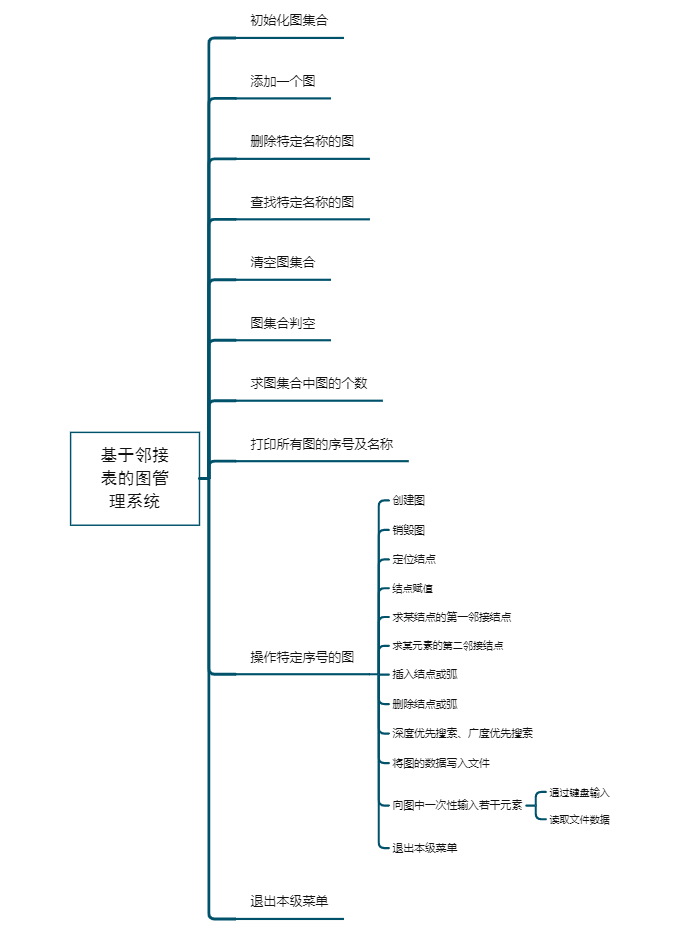
多无向图管理菜单中的“操作特定序号的无向图”可以调出单无向图操作菜单（二级菜单），可供选择的操作有：创建无向图、销毁无向图、定位某顶点、顶点赋值、求某顶点的第一、第二邻接顶点、插入顶点或弧、删除顶点或弧、深度优先搜索、广度优先搜索、将无向图的数据写入文件、通过键盘输入或者读取文件数据向无向图中一次性输入若干元素。整体系统结构设计示意图如下：

图4-1 整体系统结构设计

**4.2.2 数据结构设计**

本演示系统设计了五个物理结构，分别是顶点、表结点、头结点、邻接表和图集合，均以结构形式定义。

顶点结构中包括关键字与字符串；表结点结构中包括顶点位置编号和下一个表结点指针；头结点结构中包括顶点数据和指向第一条弧的表结点指针；邻接表结构中包括头结点数组、顶点数、弧数、图的类型；图集合结构中包括图数组（每个数组元素都包括图以及其名称）和图集合长度。

结构定义如下：

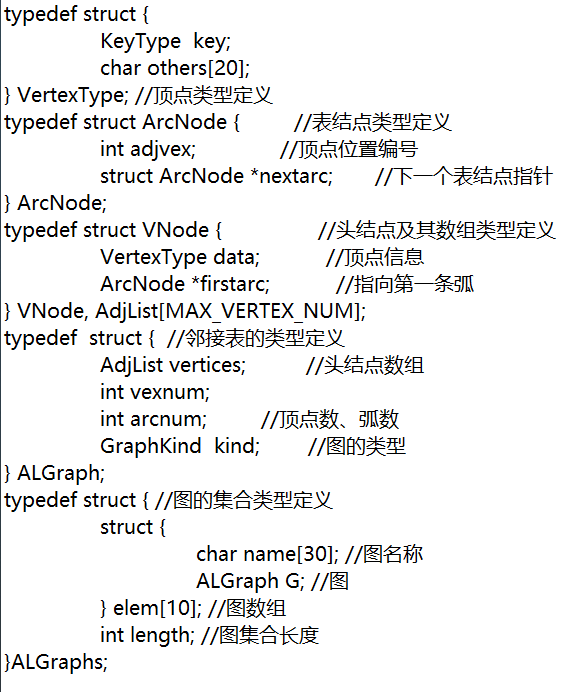


图4-2 数据结构设计

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广深度优先搜索遍历：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。

**4.2.3 有关常量和类型定义**

数据元素类型的定义：

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

有关常量的定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 20

**4.3 系统实现**

本演示系统在Windows环境下，使用Visual Studio 2019完成。演示系统涉及的部分主要函数如下，其中（1）-（4）为对图集合的操作，（5）-（15）为对单无向图的操作。

(1)向无向图集合中添加一个空无向图

函数名称：status AddALGraph(ALGraphs\* ALGraphs, char ALGraphName[])

输入：无向图集合指针以及需要添加的无向图的名称

输出：函数执行状态

思想：在Lists中加入一个名称为ListName无向图，成功则返回OK

操作：

将ListName数组拷贝到无向图集合中无向图数组的最后一个元素之后

的元素的name成员变量中；无向图集合长度加一；结束，返回OK。

时间复杂度：O（1）

空间复杂度：O（1）

(2)删除无向图集合中指定名称的无向图

函数名称：status RemoveALGraph(ALGraphs\* ALGraphs, char ALGraphName[])

输入：无向图集合指针以及需要删除的无向图名称

输出：函数执行状态

思想：删除指定名称的无向图，删除成功则返回OK，否则返回ERROR

操作：

定义i=0，当i小于无向图集合长度时，执行下列循环：

a.比较第i个无向图的名称和所给名称

b.当两者相同时，将第i个无向图的名称指针的首元素置为’\0’，将无向图数组中该无向图后面的无向图依次前移，无向图集合的长度减一，返回OK

若循环结束后函数没有返回值，说明集合中没有该名称的无向图，返回ERROR

时间复杂度：O（n）(设无向图集合长度为n)

空间复杂度：O（1）

(3)查找指定名称的无向图在集合中的位置

函数名称：status LocateALGraph(ALGraphs ALGraphs, char

ALGraphName[])

输入：无向图集合以及需要查找的无向图名称

输出：该无向图的序号或者函数执行状态  
思想：在集合中寻找指定名称的无向图，成功则返回其逻辑序号，否

则返回ERROR

操作：

① 定义i=0，当i小于无向图集合长度时，执行下列循环：

a.比较第i个无向图的名称与所给名称

b.如果相同，返回i+1，否则继续循环

② 如果循环执行完后函数没有返回值，说明集合中没有该名称的无向图，

返回ERROR

时间复杂度：O（n） （设无向图长度为n，此处为最坏情况）

空间复杂度：O（1）

(4)清空无向图集合

函数名称：status ClearALGraphs(ALGraphs\* ALGraphs)

输入：无向图集合指针

输出：函数执行状态

思想：将无向图集合的长度变成0，返回OK

操作：将Lists的length赋值为0，返回OK

时间复杂度：O（1）

空间复杂度：O（1）

(5)创建无向图

函数名称：status CreateGraph(ALGraph &G, VertexType V[], KeyType VR[][2])

输入：无向图、顶点数据数组、弧数据数组

输出：函数执行状态

操作：

首先输入顶点数据，当输入的关键字不为-1且没有重复时，执行循环，将数据放入顶点数组并将对应顶点的首弧指针置空，顶点数增加。之后输入表结点（弧）数据，当输入不为-1时，在每条边涉及的两个顶点对应的链表上头插对应结点，并保存对应节点的位序，弧数增加，返回OK，其他情况表明输入数据有误，返回ERROR。

时间复杂度：O（n）

空间复杂度：O（1）

(6)顶点赋值

函数名称：status PutVex(ALGraph &G,KeyType u,VertexType value)

输入：图、目标顶点的关键字、待赋值的数据

输出：函数执行状态

思想：当输入的目标关键字和数据关键字符合要求时，遍历找到顶点并赋值。

操作：当输入的顶点不存在或者待赋值数据的关键字重复时，返回ERROR，否则遍历顶点数组，找到关键字等于u的结点并赋值，返回OK。

时间复杂度：O（n）

空间复杂度：O（1）

(7)获得第一邻接顶点

函数名称：int FirstAdjVex(ALGraph G,KeyType u)

输入：图，需要获取第一邻接顶点的结点的关键字

输出：第一邻接顶点位序

思想：遍历顶点数组，找到对应顶点后输出第一邻接顶点。

操作：遍历顶点数组，如果当前结点的关键字等于u，如果该顶点的首弧为NULL，返回-1，否则返回首弧指向顶点的位置；如果循环结束，说明数组中没有关键字为u的顶点，返回-1.

时间复杂度：O（n）

空间复杂度：O（1）

(8)获得第二邻接结点

函数名称：int NextAdjVex(ALGraph G,KeyType v,KeyType w)

输入：图、目标顶点的关键字，相对顶点的关键字

输出：第二邻接顶点位序

思想：遍历找到目标顶点，遍历对应链表找到关键字为w的结点。返回其下一个顶点的位序。

操作：遍历顶点数组，找到关键字为w的顶点的位序，并同时判断图中是否有关键字为v的顶点，没有则返回-1。否则遍历关键字为v的顶点的表结点链表，当当前顶点的位序等于w的位序时，如果它的下一个表结点为NULL，返回-1，否则返回下一个表结点的位序，返回OK。

时间复杂度：O(n)

空间复杂度：O(1)

（9）插入顶点

函数名称：status InsertVex(ALGraph &G,VertexType v)

输入：图、插入顶点的数据

输出：函数执行状态

思想：当插入数据的关键字符合要求时，放入顶点数组。

操作：当插入数据的关键字重复或者顶点数组已满时，返回ERROR；否则将

v的key赋值给顶点数组第一个空结点的关键字，将v的others拷贝给空结点的字符串，顶点的首弧置为NULL，图的顶点数加一，返回OK。

时间复杂度：O（n）

空间复杂度：O（1）

(10)删除顶点

函数名称：status DeleteVex(ALGraph &G,KeyType v)

输入：图，要删除的顶点的关键字

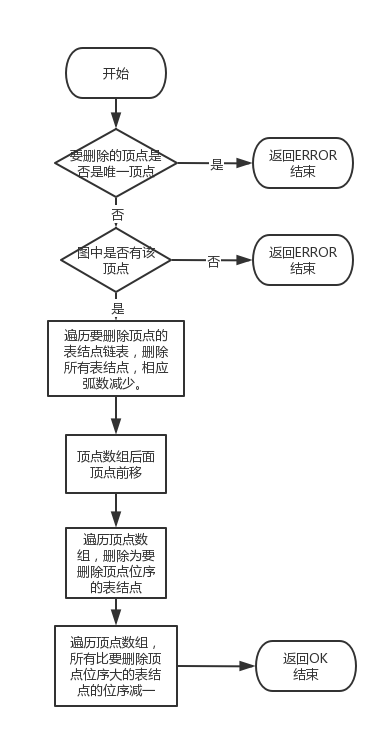
输出：函数执行状态

思想：删除关键字为v的结点和与之相关的边。

操作：如果要删除的结点是图中唯一一个顶点，返回ERROR；遍历顶点数组，如果没有关键字为v的顶点，返回ERROR；对每个顶点后面的表结点链表，清空即可（简单的链表操作，不做赘述）在清空过程中，每删除一个结点，弧数减一，然后顶点数组中让删除的顶点的后面的顶点前移；顶点数减一，之后再次遍历顶点数组，将表结点位置数据等于被删除顶点的位序的结点删除，将表结点位置数据大于被删除顶点的位序的结点的位置数据减一，返回OK。

时间复杂度：O(m\*n)

空间复杂度：O（1）

图 4-3 删除顶点流程图

(11)插入弧

函数名称：status InsertArc(ALGraph &G,KeyType v,KeyType w)

输入：图、新添加弧的两个顶点的关键字

输出：函数执行状态

思想：当输入的关键字符合要求时，在两个顶点后头插对应含对应顶点位序的结点。

操作：遍历顶点数组，记下关键字为v，w的顶点的位序i，j；如果数组中没有顶点v或者w，返回ERROR；遍历对应顶点的表结点链表，如果已有要插入的弧，返回ERROR；否则在两个顶点后头插对应顶点。返回OK。

时间复杂度：O（n）

空间复杂度：O（1）

（12）删除弧

函数名称：status DeleteArc(ALGraph &G,KeyType v,KeyType w)

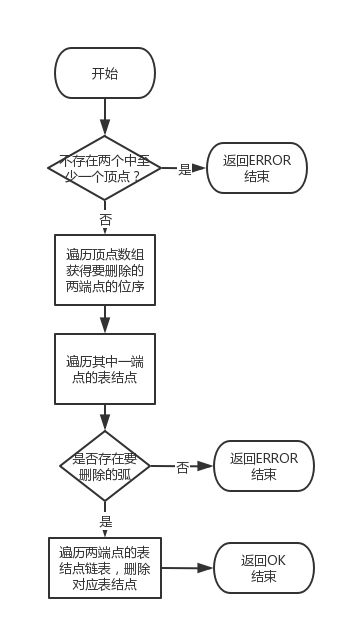
输入：图、需要删除弧的两个顶点v，w

输出：函数执行状态

思想：当输入关键字符合要求时，遍历目标顶点链表，删除对应表结点。

操作：遍历顶点数组，记下关键字为v，w的顶点的位序i，j；如果数组中没有顶点v或者w，返回ERROR；遍历对应顶点的表结点链表，如果没有要删除的弧，返回ERROR；否则遍历两个顶点的表结点链表，删除对应的表结点，弧数减一，返回OK。  
 时间复杂度：O（m\*n）

空间复杂度：O（1）

图4-4 删除弧流程图

（13）深度优先搜索

函数名称：status DFSTraverse(ALGraph &G,void (\*visit)(VertexType))

void DFS(ALGraph &G,int v)

输入：图、访问函数指针、操作顶点

输出：函数执行状态

思想：从选定顶点开始，任意选取其一邻居节点，沿一条路径直到无路可走，就回退到上一个顶点，寻找另一条可行路径，直到全部顶点均被访问一次之后结束。

操作：设置一个访问标记数组，设置循环遍历顶点数组，从第一个顶点开始，如果第一个顶点未被访问，就对其进行深度优先搜索，并标记，设置循环遍历其邻接结点，如果未被访问，就对其进行深度优先搜索。所有循环都结束之后返回OK。

时间复杂度：O（n+m）

空间复杂度：O（d）

1. 广度优先搜索

函数名称：status BFSTraverse(ALGraph &G,void (\*visit)(VertexType))

输入：图、访问指针函数

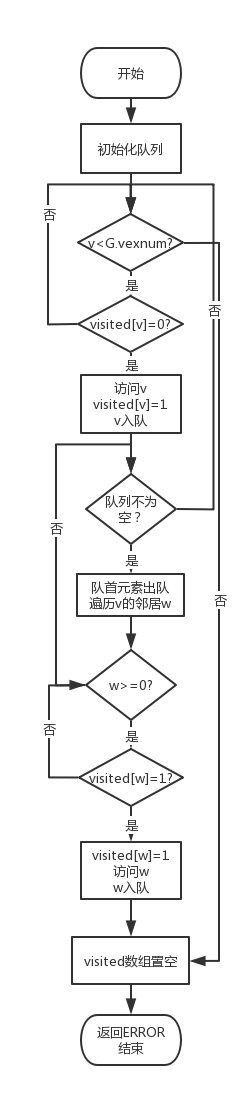
思想：从选定顶点开始，遍历其邻接结点，对于每一个邻接结点，再次遍历

它的邻接结点，直到所有结点都被访问完毕。

操作：设置一个队列和标记数组。循环遍历顶点数组，如果当前结点未被访问，就访问该结点并标记，同时该结点入队，当队列不为空时，队首结点出队，依次遍历该结点的邻接结点，如果邻接结点未被访问，就访问邻接结点并标记，之后将节点入栈。直到循环结束，返回OK。

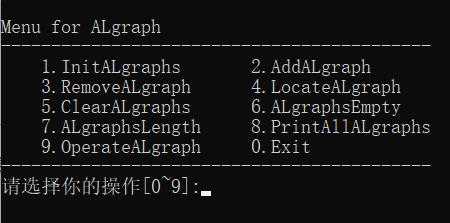
时间复杂度：O（n+m）

空间复杂度：O（1）

图 4-5 BFS流程图

**4.4 系统测试**

程序采用较为简单的界面，一级菜单和二级菜单分别如图4-6，4-7所示（在一级菜单中可通过操作9进入二级菜单）。本次系统测试选取了针对无向图集合操作的AddALGraph、RemoveALGraph、LocateALGraph函数以及针对单无向图操作的PutVex、NextAdjVex、InsertVex、DeleteArc、DFSTraverse、BFSTraverse函数。

图 4-6 一级菜单示意图

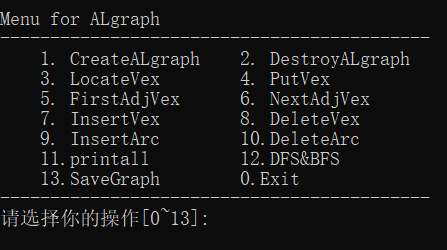


图 4-7 二级菜单示意图

下面是函数测试：

函数测试中使用的非空图G的数据为

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

(1)添加无向图的测试：

测试用例及结果如表4-1所示

表4-1添加无向图测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1(集合中已有11无向图） | 名称：11 | 名称重复！请重新输入！ | _3T1RQ%8YESZ$GTNG3FCQEB |
| 用例2（空集合） | 名称：11 | 添加成功！ | b91b4f295e533bf0c9b5dc50a35a6a3IWRMO{EOU7J1`MAGQ0W0B12 |

综合上述测试，添加无向图功能对于输出名称重复时以及未重复时都可以正确处理，所以添加无向图功能符合实验要求。

(2)删除无向图的测试：

测试用例及结果如表4-2所示

表4-2删除无向图测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1(集合中已有11无向图） | 名称：11 | 删除成功！ | 1R1EFD_UC%U5L6MSJ~OLSRM |
| 用例2(集合中已有22无向图） | 名称：11 | 图集合中没有该图！ | H]1IUO3YY)[S{4A4VQUW%LM |
| 用例3(空集合） | 名称：11 | 图集合中没有该图！ | [K`I2ZYIB`GU4P3[IPD`(@Q |

综合上述测试，删除无向图功能对于无向图集合中有或无对应名称的无向图时都可以正确处理，所以删除无向图功能符合实验要求。

(3)定位无向图的测试

测试用例及结果如表4-3所示

表4-3定位无向图测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空集合） | 名称：11 | 无向图集合中没有该无向图！ | [K`I2ZYIB`GU4P3[IPD`(@Q |
| 用例2(集合中已有11，22无向图） | 名称：22 | 该无向图位置是2 | B)W}9P@F4%2[Y[DC1I[3[OJ |
| 用例3(集合中已有11，22无向图） | 名称：33 | 无向图集合中没有该无向图！ | {8OSO%OH67{)MO`NGAML{[Q |

综合上述测试，定位无向图功能对于无向图集合中有或无对应名称的无向图时都可以正确处理，所以定位无向图功能符合实验要求.

（4）顶点赋值的测试

测试用例及结果如表4-4所示

表4-4结点赋值测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空图） | 无 | 图为空 | @7L[XLGL29F0T7WRE}]YDE2 |
| 用例2(图G） | 关键字：5  数据：1 a | 赋值成功 | ~BU8$BD5B4_J7J9JUVP5T2S |
| 用例3(图G） | 关键字：5  数据：5 a | 赋值成功 | ]~KF0KAG4MU8)_Z~%(5$%HI |
| 用例4(图G） | 关键字：5  数据：6 a | 赋值失败 | 564@YR23(_$1GD@XEPVZINE |
| 用例4(图G） | 关键字：9  数据：1 1 | 赋值失败 | }B]GS7ZU%K_LN}Y78DYA0MN |

综合上述测试，顶点赋值功能对于空图，图中有无该元素，新关键字是否重复时都能正确处理，所以结点赋值功能符合实验要求。

（5）获得第二邻接结点的测试

测试用例及结果如表4-5所示

表4-5获得第二邻接结点测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空图） | 无 | 图为空 | ]R0_%`Q2G5KSK]6Y1WLZ%FV |
| 用例2(图G） | 关键字：5 7 | 该结点位序为3  数据为 6 无向图 | TR5B]SFLTYCGXGYLYFNP{CI |
| 用例3(图G） | 关键字：5 6 | 获取失败 | 4HQHNP%ILNO6}M3(~~T4J[H |
| 用例4(图G） | 关键字：9 5 | 获取失败 | TZY{_[UW%5G{FQU1MN8{FGL |

综合上述测试，获取兄弟结点功能对于空图，图中有无该元素，该结点是否有第二邻接结点时都能正确处理，所以获取第二邻接结点功能符合实验要求。

（6）插入顶点的测试

测试用例及结果如表4-6所示

表4-6插入顶点测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空图） | 无 | 图为空 | 3%2HWG0DF1}_06QZ9)56E[2 |
| 用例4(图G） | 5 a | 插入失败 | 2HC[C7J[8_4F3@~D%9{L)}W |
| 用例5（图G） | 1 a | 插入成功 | GYBM{M{T7L(V[K@3A8EPL`R |

综合上述测试，插入顶点功能对于空图，图中有无该元素时都能正确处理，所以获取插入顶点功能符合实验要求。

（7）删除弧的测试

测试用例及结果如表4-7所示

表4-7删除弧测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空图） | 无 | 图为空 | GXLMGZR0R97`Z3W(EO_T7G0 |
| 用例2(图G+顶点1 a） | 1 5 | 删除失败 | QP5K}%ST1CQZHGKQD4WL}IN |
| 用例3(图G） | 5 8 | 删除失败 | HLJWK5DBFJI[2D{(@0BO1II |
| 用例4（图G） | 5 6 | 删除成功 | TQMWD[GDLLHL)S2$QR6G{4M |

综合上述测试，删除结点功能对于空图，图中有无该弧时都能正确处理，所以删除弧功能符合实验要求。

（8）DFS&BFS的测试

测试用例及结果如表4-8所示

表4-8DFS&BFS测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例1（空图） | 无 | 图为空 | 9HRU{)6SE{G%]FXXR6@_FYN |
| 用例2(图G） | 无 | DFS&BFS的结果 | 2F9}44U7`ZUW7)D1S2KW5OL |

综合上述测试，DFS&BFS功能对于空图，非空图都能正确处理，所以DFS&BFS功能符合实验要求。

总结：综合上述所有函数的测试，测试结果都未发现问题，符合本次实验的要求，也较完整较正确地完成了题目要求。

**4.5 实验小结**

本次实验较为顺利，没有出现太大的错误。

同时自己也存在很多问题。

由于本次实验的邻接表中也涉及到了链表，但链表的删除结点等功能不是太熟练，导致经常出现指针乱用的错误。涉及到删除结点和弧等的函数里面，总是忘记把图中弧的个数减少，导致头歌好几次提交不通过。还有很多函数的一些特殊情况没有考虑周全，导致验收的时候没有通过特殊样例。

这次实验让我再次认识到了算法健壮性的重要，以及写程序时一定要细心和考虑周全，这样才能完美通过。

最后感谢老师和同学的指导，让我顺利完成本次实验。

**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**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef struct { //线性表（顺序结构）的定义

ElemType\* elem; //元素指针

int length; //线性表长度

int listsize; //线性表最大长度

} SqList;

typedef struct { //线性表的集合类型定义

struct {

char name[30]; //线性表名称

SqList L; //线性表

} elem[10]; //线性表数组

int length; //线性表集合长度

} LISTS;

LISTS Lists; //线性表集合的定义Lists

/\*\*\*\*\*\*\*\*\*\*\*函数声明\*\*\*\*\*\*\*\*\*\*\*\*/

status InitList(SqList\* L);

status DestroyList(SqList\* L);

status ClearList(SqList\* L);

status ListEmpty(SqList L);

status ListLength(SqList L);

status GetElem(SqList L, int i, ElemType\* e);

status LocateElem(SqList L, ElemType e);

status PriorElem(SqList L, ElemType e, ElemType\* pre);

status NextElem(SqList L, ElemType e, ElemType\* next);

status ListInsert(SqList\* L, int i, ElemType e);

status ListDelete(SqList\* L, int i, ElemType\* e);

status ListTrabverse(SqList L);

status SaveList(SqList L, char FileName[]);

status LoadList(SqList\* L, char FileName[]);

status AddList(LISTS\* Lists, char ListName[]);

status RemoveList(LISTS\* Lists, char ListName[]);

status LocateList(LISTS Lists, char ListName[]);

status ClearLists(LISTS\* Lists);

status ListsEmpty(LISTS Lists);

status ListsLength(LISTS Lists);

status PrintAllLists(LISTS Lists);

status Initlists(LISTS\* Lists);

/\*\*\*\*\*\*\*\*\*\*\*主程序\*\*\*\*\*\*\*\*\*\*\*\*/

int main() {

SqList L;

LISTS Lists;

char ListName1[100],ListName2[100],ListName3[100],name[100],name2[100];

int op1 = 1, op2 = 1, n, k =1 ,choice,i,j,flag=0,flag2=0,e;

int ltmp1,ltmp2,ltmp3;

int num1,num;

int tmp1,tmp2,tmp3,tmp4,tmp5,tmp6,tmp7,tmp8,tmp9,tmp0,temp0,temp00,elem1,elem2,elem3, elem4,elem5,pre,next;

int loc ,deloc;

while (op1) {

system("cls");

printf("\n\n");

printf("Menu for Linear Table On Sequence Structure \n");

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

printf(" 1.InitLists 2.AddList\n");

printf(" 3.RemoveList 4.LocateList\n");

printf(" 5.ClearLists 6.ListsEmpty\n");

printf(" 7.ListsLength 8.PrintAllLists\n");

printf(" 9.OperateList 0.Exit\n");

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

printf("请选择你的操作[0~9]:");

scanf("%d", &op1);

switch (op1) {

case 0:

break;

case 1:

Initlists(&Lists);

printf("初始化成功！\n");

getchar();

getchar();

break;

case 2:

printf("请输入需要添加的线性表的名称：\n");

do

{

flag=0;

scanf("%s", ListName1);

for(j=0;j<Lists.length;j++)

if(!strcmp(ListName1,Lists.elem[j].name)) //查找集合中是否早已又该该名称的线性表

{

flag=1;

printf("名称重复！请重新输入！\n");

break;

}

}while(flag==1);

AddList(&Lists, ListName1);

printf("添加成功！\n");

getchar();

getchar();

break;

case 3:

printf("请输入需要删除的线性表的名称：\n");

scanf("%s", ListName2);

ltmp1 = RemoveList(&Lists, ListName2);

if (ltmp1 == ERROR)

printf("线性表集合中没有该线性表！\n");

else

printf("删除成功！\n");

getchar();

getchar();

break;

case 4:

printf("请输入需要定位的线性表的名称：\n");

scanf("%s", ListName3);

ltmp2 = LocateList(Lists, ListName3);

if (ltmp2 == ERROR)

printf("线性表集合中没有该线性表！\n");

else

printf("该线性表的位置为%d\n", LocateList(Lists, ListName3));

getchar();

getchar();

break;

case 5:

ClearLists(&Lists);

printf("清空成功！\n");

getchar();

getchar();

break;

case 6:

ltmp3 = ListsEmpty(Lists);

if (ltmp3 == TRUE)

printf("该线性表集合为空！\n");

else

printf("该线性表集合不为空！\n");

getchar();

getchar();

break;

case 7:

printf("该线性表集合长度为%d\n", ListsLength(Lists));

getchar();

getchar();

break;

case 8:

if (!PrintAllLists(Lists))

printf("该线性表集合中没有元素！\n");

getchar();

getchar();

break;

case 9:

flag2=0;

printf("请输入需要操作的线性表的序号：\n");

scanf("%d", &num1);

num = num1 - 1;

while (num1 < 1 || num1 > Lists.length) {

printf("输入序号有误！\n");

scanf("%d", &num1);

}

while (op2) {

if(flag2==1) //线性表已经被销毁，就退回一级菜单

{

op2=1;

break;

}

system("cls");

printf("\n\n");

printf("Menu for Linear Table On Sequence Structure \n");

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

printf(" 1. Initlist 2. DestroyList\n");

printf(" 3. ClearList 4. ListEmpty\n");

printf(" 5. ListLength 6. GetElem \n");

printf(" 7. LocateElem 8. PriorElem \n");

printf(" 9. NextElem 10.ListInsert \n");

printf(" 11.ListDelete 12.ListTrabverse\n");

printf(" 13.InputElem 14.SaveList\n");

printf(" 0. Exit\n");

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

printf("请选择你的操作[0~14]:");

scanf("%d", &op2);

switch (op2) {

case 0:

printf("按回车退出！\n");

break;

case 1:

if (InitList(&Lists.elem[num].L) == INFEASIBLE)

printf("线性表已经初始化！\n");

else

printf("初始化成功！\n");

getchar();

getchar();

break;

case 2:

flag2=0; //flag2作用：代表销毁状态

if (DestroyList(&Lists.elem[num].L) == OK)

{

RemoveList(&Lists, Lists.elem[num].name);

printf("线性表销毁成功！\n");

flag2=1;

}

else

printf("线性表销毁失败！\n");

getchar();

getchar();

break;

case 3:

tmp0=ClearList(&Lists.elem[num].L);

if (tmp0 == OK)

printf("线性表清空成功！\n");

else if(tmp0==ERROR)

printf("线性表为空！\n");

else printf("线性表未初始化！\n");

getchar();

getchar();

break;

case 4:

tmp1 = ListEmpty(Lists.elem[num].L);

if (tmp1 == INFEASIBLE)

printf("线性表未初始化！\n");

else if (tmp1 == TRUE)

printf("线性表为空！\n");

else

printf("线性表不为空！\n");

getchar();

getchar();

break;

case 5:

tmp2 = ListLength(Lists.elem[num].L);

if (tmp2 == INFEASIBLE)

printf("线性表未初始化！\n");

else

printf("线性表的长度为%d\n", ListLength(Lists.elem[num].L));

getchar();

getchar();

break;

case 6:

printf("请输入需要获取的元素的位置：\n");

scanf("%d", &i);

tmp3 = GetElem(Lists.elem[num].L, i, &e);

if (tmp3 == INFEASIBLE)

printf("线性表未初始化！\n");

else if (tmp3 == ERROR)

printf("输入位置有误！\n");

else

printf("第%d号元素为%d。\n", i, e);

getchar();

getchar();

break;

case 7:

printf("请输入需要查找的元素：\n");

scanf("%d", &elem1);

tmp4 = LocateElem(Lists.elem[num].L, elem1);

if (tmp4 == INFEASIBLE)

printf("线性表未初始化！\n");

else if (tmp4 == ERROR)

printf("线性表中没有该元素！\n");

else

printf("该元素在线性表中的位置为%d \n", tmp4);

getchar();

getchar();

break;

case 8:

printf("请输入需要获取哪个元素的前驱：\n");

scanf("%d", &elem2);

tmp5 = PriorElem(Lists.elem[num].L, elem2, &pre);

if (tmp5 == INFEASIBLE)

printf("线性表未初始化！\n");

else if (tmp5 == ERROR)

printf("该元素没有前驱！\n");

else

printf("该元素的前驱为%d\n", pre);

getchar();

getchar();

break;

case 9:

printf("请输入需要获取哪个元素的后继：\n");

scanf("%d", &elem3);

tmp6 = NextElem(Lists.elem[num].L, elem3, &next);

if (tmp6 == INFEASIBLE)

printf("线性表未初始化！\n");

else if (tmp6 == ERROR)

printf("该元素没有后继！\n");

else

printf("该元素的后继为%d", next);

getchar();

getchar();

break;

case 10:

printf("请输入插入元素的位置以及需要插入的元素：\n");

scanf("%d%d", &loc, &elem4);

tmp7 = ListInsert(&Lists.elem[num].L, loc, elem4);

if (tmp7 == INFEASIBLE)

printf("线性表未初始化！\n");

else if (tmp7 == ERROR)

printf("插入位置有误！\n");

else

printf("插入成功\n");

getchar();

getchar();

break;

case 11:

printf("请输入需要删除的元素的位置：\n");

scanf("%d", &deloc);

tmp8 = ListDelete(&Lists.elem[num].L, deloc, &elem5);

if (tmp8 == INFEASIBLE)

printf("线性表未初始化！\n");

else if (tmp8 == ERROR)

printf("删除位置有误！\n");

else

printf("删除成功，删除的元素为%d\n", elem5);

getchar();

getchar();

break;

case 12:

tmp9 = ListTrabverse(Lists.elem[num].L);

if (tmp9 == ERROR)

printf("线性表是空表！\n");

else if (tmp9 == INFEASIBLE)

printf("线性表未初始化！\n");

getchar();

getchar();

break;

case 13:

k=1;

printf("请选择通过键盘对线性表中元素赋值（1）或者通过读入文件赋值（2）：\n");

scanf("%d", &choice);

if (choice == 1)

{

printf("请输入需要输入的元素个数：\n");

scanf("%d", &n);

printf("请输入数据：\n");

while (k <= n) {

int temp;

scanf("%d", &temp);

ListInsert(&Lists.elem[num].L, k, temp); //相当于一次 一次地插入元素

k++;

}

printf("输入成功\n");

}

else

{

printf("请输入文件名：\n");

scanf("%s", name);

temp0 = LoadList(&Lists.elem[num].L, name);

if (temp0 == INFEASIBLE)

printf("线性表未初始化！\n");

else if (temp0 == ERROR) printf("读入失败！\n");

else if(temp0==OK)

printf("读入成功！\n");

}

getchar();

getchar();

break;

case 14:

printf("请输入储存数据的文件名:\n");

scanf("%s", name2);

temp00=SaveList(Lists.elem[num].L, name2);

if (temp00 == INFEASIBLE)

printf("线性表未初始化！\n");

else if (temp00 == ERROR) printf("写入失败！\n");

else

printf("写入成功！\n");

getchar();

getchar();

break;

}

}

op2 = 1;

if(flag2==0)

{

getchar();

getchar();

break;

}

}

}

return 0;

}

status InitList(SqList\* L)

// 线性表L未初始化，构造一个空的线性表，返回OK，否则返回INFEASIBLE。

{

if (L->elem == NULL) {

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

L->length = 0;

L->listsize = LIST\_INIT\_SIZE;

return OK;

}

else

return INFEASIBLE;

}

status DestroyList(SqList\* L)

// 如果线性表L已初始化，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 != NULL) {

free(L->elem);

L->elem = NULL;

L->length = 0;

L->listsize = 0;

return OK;

}

else

return INFEASIBLE;

}

status ClearList(SqList\* L)

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

{

if (L->elem == NULL)

return INFEASIBLE;

else if(L->length==0)

return ERROR;

else {

L->length = 0;

return OK;

}

}

status ListEmpty(SqList L)

// 如果线性表L已初始化，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 == NULL)

return INFEASIBLE;

else {

if (L.length == 0)

return TRUE;

else

return FALSE;

}

}

status ListLength(SqList L)

// 如果线性表L已初始化，返回线性表L的长度，否则返回INFEASIBLE。

{

if (L.elem == NULL)

return INFEASIBLE;

else

return L.length;

}

status GetElem(SqList L, int i, ElemType\* e)

// 如果线性表L已初始化，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；否则返回INFEASIBLE。

{

if (L.elem == NULL)

return INFEASIBLE;

else {

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

return ERROR;

else {

\*e = L.elem[i - 1];

return OK;

}

}

}

status LocateElem(SqList L, ElemType e)

// 如果线性表L已初始化，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；否则返回INFEASIBLE。

{

if (L.elem == NULL)

return INFEASIBLE;

else {

int i = 0;

while (L.elem[i] != e && i < L.length)

i++;

if (i == L.length)

return ERROR;

else

return i + 1;

}

}

status PriorElem(SqList L, ElemType e, ElemType\* pre)

// 如果线性表L已初始化，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；否则返回INFEASIBLE。

{

if (L.elem == NULL)

return INFEASIBLE;

else {

int i = 0;

while (L.elem[i] != e && i < L.length)

i++;

if (i == L.length || i == 0)

return ERROR;

else {

\*pre = L.elem[i - 1];

return OK;

}

}

}

status NextElem(SqList L, ElemType e, ElemType\* next)

// 如果线性表L已初始化，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；否则返回INFEASIBLE。

{

if (L.elem == NULL)

return INFEASIBLE;

else {

int i = 0;

while (L.elem[i] != e && i < L.length)

i++;

if (i == L.length || i == L.length - 1)

return ERROR;

else {

\*next = L.elem[i + 1];

return OK;

}

}

}

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

// 如果线性表L已初始化，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；否则返回INFEASIBLE。

{

if (L->elem == NULL)

return INFEASIBLE;

else {

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

return ERROR;

else {

if (L->length >= L->listsize) {

ElemType\* newbase = (ElemType\*)realloc(L->elem, (L->length + LISTINCREMENT) \* sizeof(ElemType));

if (newbase == NULL)

return OVERFLOW;

L->elem = newbase;

L->listsize += LISTINCREMENT;

}

int j;

if (L->length > 0) {

for (j = L->length - 1; j >= i - 1; j--)

L->elem[j + 1] = L->elem[j];

}

L->elem[i - 1] = e;

L->length++;

return OK;

}

}

}

status ListDelete(SqList\* L, int i, ElemType\* e)

// 如果线性表L已初始化，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；否则返回INFEASIBLE。

{

if (L->elem == NULL)

return INFEASIBLE;

else {

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

return ERROR;

else {

\*e = L->elem[i - 1];

for (int k = i - 1; k < L->length; k++)

L->elem[k] = L->elem[k + 1];

L->length--;

return OK;

}

}

}

status ListTrabverse(SqList L)

// 如果线性表L已初始化，依次显示线性表中的元素，每个元素间空一格，返回OK；否则返回INFEASIBLE。

{

if (L.elem == NULL)

return INFEASIBLE;

else if (L.length == 0) return ERROR;

else {

int i;

printf("元素依次为：\n");

for (i = 0; i < L.length - 1; i++)

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

if (L.length > 0)

printf("%d", L.elem[L.length - 1]);

return OK;

}

}

status SaveList(SqList L, char FileName[])

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

{

if (L.elem == NULL)

return INFEASIBLE;

else {

FILE\* fp;

if ((fp = fopen(FileName, "w")) == NULL)

return ERROR;

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

fwrite(&L.elem[i], sizeof(ElemType), 1, fp); //用循环写入

fclose(fp);

return OK;

}

}

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

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

if (L->elem==NULL)

return INFEASIBLE;

else {

if(L->length!=0)

{

printf("线性表中已经有元素！\n");

return -2;

}

FILE\* fp;

if ((fp = fopen(FileName, "r")) == NULL)

return ERROR;

while (fread(&L->elem[L->length], sizeof(ElemType), 1, fp)) //用循环读取

L->length++;

fclose(fp);

return OK;

}

}

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。

{

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

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

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, k;

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

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

continue;

else {

Lists->elem[i].name[0] = '\0';

free(Lists->elem[i].L.elem);

Lists->elem[i].L.elem = NULL;

Lists->elem[i].L.length = 0;

Lists->elem[i].L.listsize = 0;

for (k = i; k < Lists->length; k++)

Lists->elem[k] = Lists->elem[k + 1];

Lists->length--;

return OK;

}

}

return ERROR;

}

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))

continue;

else

return i + 1;

}

return ERROR;

}

status ClearLists(LISTS\* Lists) {

//清空线性表集合

Lists->length = 0;

return OK;

}

status ListsEmpty(LISTS Lists) {

//判断线性表集合是否为空

if (Lists.length == 0)

return TRUE;

else

return FALSE;

}

status ListsLength(LISTS Lists) {

//求线性表集合的长度

return Lists.length;

}

status PrintAllLists(LISTS Lists) {

//打印线性表集合中的所有元素

if (Lists.length == 0) return ERROR;

else {

printf("所有线性表依次为:\n");

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

printf("%d\t%s\n", i + 1, Lists.elem[i].name);

}

}

return OK;

}

status Initlists(LISTS\* Lists) {

//初始化线性表集合

Lists->length = 0;

return OK;

}

**附录B 基于链式存储结构线性表实现的源程序**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

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

ElemType data;

struct LNode\* next;

}LNode, \* LinkList;

typedef struct { //链表的集合类型定义

struct {

char name[30]; //链表名称

LinkList L; //链表

} elem[10]; //链表数组

int length; //链表集合长度

} LISTS;

LISTS Lists; //链表集合的定义Lists

/\*\*\*\*\*\*\*\*\*\*\*函数声明\*\*\*\*\*\*\*\*\*\*\*\*/

status InitList(LinkList\* L);

status DestroyList(LinkList\* L);

status ClearList(LinkList\* L);

status ListEmpty(LinkList L);

status ListLength(LinkList L);

status GetElem(LinkList L, int i, ElemType\* e);

status LocateElem(LinkList L, ElemType e);

status PriorElem(LinkList L, ElemType e, ElemType\* pre);

status NextElem(LinkList L, ElemType e, ElemType\* next);

status ListInsert(LinkList\* L, int i, ElemType e);

status ListDelete(LinkList\* L, int i, ElemType\* e);

status ListTrabverse(LinkList L);

status SaveList(LinkList L, char FileName[]);

status LoadList(LinkList\* L, char FileName[]);

status AddList(LISTS\* Lists, char ListName[]);

status RemoveList(LISTS\* Lists, char ListName[]);

status LocateList(LISTS Lists, char ListName[]);

status ClearLists(LISTS\* Lists);

status ListsEmpty(LISTS Lists);

status ListsLength(LISTS Lists);

status PrintAllLists(LISTS Lists);

status Initlists(LISTS\* Lists);

/\*\*\*\*\*\*\*\*\*\*\*主程序\*\*\*\*\*\*\*\*\*\*\*\*/

int main() {

LISTS Lists;

char ListName1[100], ListName2[100], ListName3[100], name[100], name2[100];

int op1 = 1, op2 = 1, n, k = 1, choice, i, j, flag = 0, flag2 = 0, e;

int ltmp1, ltmp2, ltmp3;

int num1, num;

int tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7, tmp8, tmp9, tmp0, temp0, temp00, elem1, elem2, elem3, elem4, elem5, pre, next;

int loc, deloc;

while (op1) {

system("cls");

printf("\n\n");

printf("Menu for Linear Table On Link Structure \n");

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

printf(" 1.InitLists 2.AddList\n");

printf(" 3.RemoveList 4.LocateList\n");

printf(" 5.ClearLists 6.ListsEmpty\n");

printf(" 7.ListsLength 8.PrintAllLists\n");

printf(" 9.OperateList 0.Exit\n");

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

printf("请选择你的操作[0~9]:");

scanf("%d", &op1);

switch (op1) {

case 0:

break;

case 1:

Initlists(&Lists);

printf("初始化成功！\n");

getchar();

getchar();

break;

case 2:

printf("请输入需要添加的链表的名称：\n");

do

{

flag = 0;

scanf("%s", ListName1);

for (j = 0; j < Lists.length; j++)

if (!strcmp(ListName1, Lists.elem[j].name)) //查找集合中是否早已又该该名称的链表

{

flag = 1;

printf("名称重复！请重新输入！\n");

break;

}

} while (flag == 1);

AddList(&Lists, ListName1);

printf("添加成功！\n");

getchar();

getchar();

break;

case 3:

printf("请输入需要删除的链表的名称：\n");

scanf("%s", ListName2);

ltmp1 = RemoveList(&Lists, ListName2);

if (ltmp1 == ERROR)

printf("链表集合中没有该链表！\n");

else

printf("删除成功！\n");

getchar();

getchar();

break;

case 4:

printf("请输入需要定位的链表的名称：\n");

scanf("%s", ListName3);

ltmp2 = LocateList(Lists, ListName3);

if (ltmp2 == ERROR)

printf("链表集合中没有该链表！\n");

else

printf("该链表的位置为%d\n", LocateList(Lists, ListName3));

getchar();

getchar();

break;

case 5:

ClearLists(&Lists);

printf("清空成功！\n");

getchar();

getchar();

break;

case 6:

ltmp3 = ListsEmpty(Lists);

if (ltmp3 == TRUE)

printf("该链表集合为空！\n");

else

printf("该链表集合不为空！\n");

getchar();

getchar();

break;

case 7:

printf("该链表集合长度为%d\n", ListsLength(Lists));

getchar();

getchar();

break;

case 8:

if (!PrintAllLists(Lists))

printf("该链表集合中没有元素！\n");

getchar();

getchar();

break;

case 9:

flag2 = 0;

printf("请输入需要操作的链表的序号：\n");

scanf("%d", &num1);

num = num1 - 1;

while (num1 < 1 || num1 > Lists.length) {

printf("输入序号有误！\n");

scanf("%d", &num1);

}

while (op2) {

if (flag2 == 1)

{

op2=1;

break;

} //链表已经被销毁，就退回一级菜单

system("cls");

printf("\n\n");

printf("Menu for Linear Table On Link Structure \n");

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

printf(" 1. Initlist 2. DestroyList\n");

printf(" 3. ClearList 4. ListEmpty\n");

printf(" 5. ListLength 6. GetElem \n");

printf(" 7. LocateElem 8. PriorElem \n");

printf(" 9. NextElem 10.ListInsert \n");

printf(" 11.ListDelete 12.ListTrabverse\n");

printf(" 13.InputElem 14.SaveList\n");

printf(" 0. Exit\n");

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

printf("请选择你的操作[0~14]:");

scanf("%d", &op2);

switch (op2) {

case 0:

printf("按回车退出！\n");

break;

case 1:

if (InitList(&Lists.elem[num].L) == INFEASIBLE)

printf("链表已经初始化！\n");

else

printf("初始化成功！\n");

getchar();

getchar();

break;

case 2:

flag2 = 0; //flag2作用：代表销毁状态

if (DestroyList(&Lists.elem[num].L) == OK)

{

RemoveList(&Lists, Lists.elem[num].name);

printf("链表销毁成功！\n");

flag2 = 1;

}

else

printf("链表销毁失败！\n");

getchar();

getchar();

break;

case 3:

tmp0 = ClearList(&Lists.elem[num].L);

if (tmp0 == OK)

printf("链表清空成功！\n");

else if (tmp0 == ERROR)

printf("链表为空！\n");

else printf("链表未初始化！\n");

getchar();

getchar();

break;

case 4:

tmp1 = ListEmpty(Lists.elem[num].L);

if (tmp1 == INFEASIBLE)

printf("链表未初始化！\n");

else if (tmp1 == TRUE)

printf("链表为空！\n");

else

printf("链表不为空！\n");

getchar();

getchar();

break;

case 5:

tmp2 = ListLength(Lists.elem[num].L);

if (tmp2 == INFEASIBLE)

printf("链表未初始化！\n");

else

printf("链表的长度为%d\n", ListLength(Lists.elem[num].L));

getchar();

getchar();

break;

case 6:

printf("请输入需要获取的元素的位置：\n");

scanf("%d", &i);

tmp3 = GetElem(Lists.elem[num].L, i, &e);

if (tmp3 == INFEASIBLE)

printf("链表未初始化！\n");

else if (tmp3 == ERROR)

printf("输入位置有误！\n");

else

printf("第%d号元素为%d。\n", i, e);

getchar();

getchar();

break;

case 7:

printf("请输入需要查找的元素：\n");

scanf("%d", &elem1);

tmp4 = LocateElem(Lists.elem[num].L, elem1);

if (tmp4 == INFEASIBLE)

printf("链表未初始化！\n");

else if (tmp4 == ERROR)

printf("链表中没有该元素！\n");

else

printf("该元素在链表中的位置为%d \n", tmp4);

getchar();

getchar();

break;

case 8:

printf("请输入需要获取哪个元素的前驱：\n");

scanf("%d", &elem2);

tmp5 = PriorElem(Lists.elem[num].L, elem2, &pre);

if (tmp5 == INFEASIBLE)

printf("链表未初始化！\n");

else if (tmp5 == ERROR)

printf("该元素没有前驱！\n");

else

printf("该元素的前驱为%d\n", pre);

getchar();

getchar();

break;

case 9:

printf("请输入需要获取哪个元素的后继：\n");

scanf("%d", &elem3);

tmp6 = NextElem(Lists.elem[num].L, elem3, &next);

if (tmp6 == INFEASIBLE)

printf("链表未初始化！\n");

else if (tmp6 == ERROR)

printf("该元素没有后继！\n");

else

printf("该元素的后继为%d", next);

getchar();

getchar();

break;

case 10:

printf("请输入插入元素的位置以及需要插入的元素：\n");

scanf("%d%d", &loc, &elem4);

tmp7 = ListInsert(&Lists.elem[num].L, loc, elem4);

if (tmp7 == INFEASIBLE)

printf("链表未初始化！\n");

else if (tmp7 == ERROR)

printf("插入位置有误！\n");

else

printf("插入成功\n");

getchar();

getchar();

break;

case 11:

printf("请输入需要删除的元素的位置：\n");

scanf("%d", &deloc);

tmp8 = ListDelete(&Lists.elem[num].L, deloc, &elem5);

if (tmp8 == INFEASIBLE)

printf("链表未初始化！\n");

else if (tmp8 == ERROR)

printf("删除位置有误！\n");

else

printf("删除成功，删除的元素为%d\n", elem5);

getchar();

getchar();

break;

case 12:

tmp9 = ListTrabverse(Lists.elem[num].L);

if (tmp9 == ERROR)

printf("链表是空表！\n");

else if (tmp9 == INFEASIBLE)

printf("链表未初始化！\n");

getchar();

getchar();

break;

case 13:

k = 1;

printf("请选择通过键盘对链表中元素赋值（1）或者通过读入文件赋值（2）：\n");

scanf("%d", &choice);

if (choice == 1)

{

printf("请输入需要输入的元素个数：\n");

scanf("%d", &n);

printf("请输入数据：\n");

while (k <= n) {

int temp;

scanf("%d", &temp);

ListInsert(&Lists.elem[num].L, k, temp); //相当于一次一次地插入元素

k++;

}

printf("输入成功\n");

}

else

{

printf("请输入文件名：\n");

scanf("%s", name);

temp0 = LoadList(&Lists.elem[num].L, name);

if (temp0 == INFEASIBLE)

printf("链表未初始化！\n");

else if (temp0 == ERROR) printf("读入失败！\n");

else if (temp0 == OK)

printf("读入成功！\n");

}

getchar();

getchar();

break;

case 14:

printf("请输入储存数据的文件名:\n");

scanf("%s", name2);

temp00 = SaveList(Lists.elem[num].L, name2);

if (temp00 == INFEASIBLE)

printf("链表未初始化！\n");

else if (temp00 == ERROR) printf("写入失败！\n");

else

printf("写入成功！\n");

getchar();

getchar();

break;

}

}

op2 = 1;

if(flag2==0)

{

getchar();

getchar();

break;

}

}

}

return 0;

}

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;

else

{

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

(\*L)->next = NULL;

return OK;

}

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

}

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;

else {

LNode\* p, \* q;

p = (\*L)->next;

q = (\*L);

while (p)

{

q = p->next;

free(p);

p = q;

}

free((\*L));

(\*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;

else {

LNode\* p, \* q;

p = (\*L)->next;

q = (\*L);

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;

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

else return FALSE;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;

else

{

int len = 0;

LNode\* p = L->next;

while (p)

{

len++;

p = p->next;

}

return len;

}

/\*\*\*\*\*\*\*\*\*\* 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;

else

{

int len = 0;

LNode\* p = L->next;

while (p)

{

len++;

p = p->next;

}

if (i<1 || i>len)

return ERROR;

else

{

int k = 0;

p = L->next;

while (k < i - 1 && p)

{

p = p->next;

k++;

}

\*e = p->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;

else {

LNode\* p = L->next;

int i = 1;

while (p && p->data != e)

{

p = p->next;

i++;

}

if (!p) return ERROR;

else return i;

}

/\*\*\*\*\*\*\*\*\*\* 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;

else {

LNode\* p = L->next, \* q = L;

while (p && p->data != e)

{

q = p;

p = p->next;

}

if (p == L->next || !p) return ERROR;

else

{

\*pre = q->data;

return OK;

}

}

/\*\*\*\*\*\*\*\*\*\* 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;

else {

LNode\* p = L->next;

while (p && p->data != e)

p = p->next;

if (p == NULL)

return ERROR;

if (p->next == NULL)

return ERROR;

else

{

\*next = p->next->data;

return OK;

}

}

/\*\*\*\*\*\*\*\*\*\* 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;

else {

int len = 0;

LNode\* p = (\*L)->next, \* q = (\*L);

while (p)

{

len++;

p = p->next;

}

if (i < 1 || (i > len + 1))

return ERROR;

else {

int k = 0;

p = (\*L)->next;

while (k < i - 1 && p)

{

q = p;

p = p->next;

k++;

}

LNode\* s = (LNode\*)malloc(sizeof(LNode));

s->data = e;

q->next = s;

s->next = p;

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;

else

{

int len = 0;

LNode\* p = (\*L)->next, \* q = (\*L);

while (p)

{

len++;

p = p->next;

}

if (i<1 || i>len)

return ERROR;

else

{

int k = 0;

p = (\*L)->next;

while (k < i - 1 && p)

{

q = p;

p = p->next;

k++;

}

\*e = p->data;

q->next = p->next;

free(p);

return OK;

}

}

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

}

status ListTrabverse(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;

else if(L->next==NULL) return ERROR;

else

{

LNode\* p = L->next;

printf("元素依次为：\n");

while (p)

{

printf("%d ", p->data);

p = p->next;

}

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)

return INFEASIBLE;

else {

FILE\* fp;

if ((fp = fopen(FileName, "w")) == NULL)

return ERROR;

LNode\* p = L->next;

while (p)

{

fwrite(&p->data, sizeof(ElemType), 1, fp);

p = p->next;

}//用循环写入

fclose(fp);

return OK;

}

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

}

status LoadList(LinkList\* L, char FileName[])

// 如果线性表L已经初始化而且为空，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if ((\*L) == NULL)

return INFEASIBLE;

else {

if((\*L)->next!=NULL)

printf("链表中已经有元素！\n");

else

{

FILE\* fp;

if ((fp = fopen(FileName, "r")) == NULL)

return ERROR;

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

LNode\* p = (LNode\*)malloc(sizeof(LNode));

LNode\* q=NULL, \* r=NULL;

(\*L)->next = p;

while (fread(&p->data, sizeof(ElemType), 1, fp))

{

q = (LNode\*)malloc(sizeof(LNode));

q->next = NULL;

r = p;

p->next = q;

p = q;

}

free(p);

r->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。

{

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

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

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, k;

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

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

continue;

else {

Lists->elem[i].name[0] = '\0';

free(Lists->elem[i].L);

Lists->elem[i].L = NULL;

for (k = i; k < Lists->length; k++)

Lists->elem[k] = Lists->elem[k + 1];

Lists->length--;

return OK;

}

}

return ERROR;

}

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))

continue;

else

return i + 1;

}

return ERROR;

}

status ClearLists(LISTS\* Lists) {

//清空线性表集合

Lists->length = 0;

return OK;

}

status ListsEmpty(LISTS Lists) {

//判断线性表集合是否为空

if (Lists.length == 0)

return TRUE;

else

return FALSE;

}

status ListsLength(LISTS Lists) {

//求线性表集合的长度

return Lists.length;

}

status PrintAllLists(LISTS Lists) {

//打印线性表集合中的所有元素

if (Lists.length == 0) return ERROR;

else {

printf("所有线性表依次为:\n");

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

printf("%d\t%s\n", i + 1, Lists.elem[i].name);

}

}

return OK;

}

status Initlists(LISTS\* Lists) {

//初始化线性表集合

Lists->length = 0;

return OK;

}

# 附录C 基于二叉链表的二叉树实现的源程序

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int KeyType;

typedef struct {

KeyType key;

char others[20];

} TElemType; //二叉树数据类型定义

typedef struct BiTNode { //二叉树结点的定义

TElemType data; //数据域

struct BiTNode\* lchild, \* rchild; //左右孩子指针

} BiTNode, \* BiTree;

typedef struct { //二叉树的集合类型定义

struct {

char name[30]; //二叉树名称

BiTree T; //二叉树

} elem[10]; //二叉树数组

int length; //二叉树集合长度

} BITREES;

BiTree temp1;

status judge(TElemType definition[]);

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

status ClearBiTree(BiTree& T);

int BiTreeDepth(BiTree T);

BiTNode\* LocateNode(BiTree T, KeyType e);

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

BiTNode\* GetSibling(BiTree T, KeyType e);

status Traverse(BiTree T, KeyType e);

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

BiTNode\* GetFather(BiTree T, KeyType e);

BiTNode\* Getlast(BiTree T);

status DeleteNode(BiTree& T, KeyType e);

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

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

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

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

status SaveBiTree(BiTree T, char FileName[]);

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

status AddBiTree(BITREES\* BiTrees, char BiTreeName[]);

status RemoveBiTree(BITREES\* BiTrees, char BiTreeName[]);

status LocateBiTree(BITREES BiTrees, char BiTreeName[]);

status ClearBiTrees(BITREES\* BiTrees);

status BiTreesEmpty(BITREES BiTrees);

status BiTreesLength(BITREES BiTrees);

status PrintAllBiTrees(BITREES BiTrees);

status InitBiTrees(BITREES\* BiTrees);

void visit(BiTree T);

TElemType\* p;

/\*\*\*\*\*\*\*\*\*\*\*主程序\*\*\*\*\*\*\*\*\*\*\*\*/

int main() {

BITREES BiTrees;

char BiTreeName1[100], BiTreeName2[100], BiTreeName3[100], name[100], name2[100];

int op1 = 1, op2 = 1, n, k = 1, choice, i, j, flag = 0, flag2 = 0,flag3=0, e, LR;

int ltmp1, ltmp2, ltmp3;

int num1, num;

int tmp1, tmp2, tmp3, tmp4, tmp0, temp0, temp00, elem2;

TElemType\* definition, value, c;

BiTree temp;

while (op1) {

system("cls");

printf("\n\n");

printf("Menu for BiTree \n");

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

printf(" 1.InitBiTrees 2.AddBiTree\n");

printf(" 3.RemoveBiTree 4.LocateBiTree\n");

printf(" 5.ClearBiTrees 6.BiTreesEmpty\n");

printf(" 7.BiTreesLength 8.PrintAllBiTrees\n");

printf(" 9.OperateBiTree 0.Exit\n");

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

printf("请选择你的操作[0~9]:");

scanf("%d", &op1);

switch (op1) {

case 0:

break;

case 1:

InitBiTrees(&BiTrees);

printf("初始化成功！\n");

getchar();

getchar();

break;

case 2:

printf("请输入需要添加的二叉树的名称：\n");

do

{

flag = 0;

scanf("%s", BiTreeName1);

for (j = 0; j < BiTrees.length; j++)

if (!strcmp(BiTreeName1, BiTrees.elem[j].name)) //查找集合中是否早已又该该名称的二叉树

{

flag = 1;

printf("名称重复！请重新输入！\n");

break;

}

} while (flag == 1);

AddBiTree(&BiTrees, BiTreeName1);

printf("添加成功！\n");

getchar();

getchar();

break;

case 3:

printf("请输入需要删除的二叉树的名称：\n");

scanf("%s", BiTreeName2);

ltmp1 = RemoveBiTree(&BiTrees, BiTreeName2);

if (ltmp1 == ERROR)

printf("二叉树集合中没有该二叉树！\n");

else

printf("删除成功！\n");

getchar();

getchar();

break;

case 4:

printf("请输入需要定位的二叉树的名称：\n");

scanf("%s", BiTreeName3);

ltmp2 = LocateBiTree(BiTrees, BiTreeName3);

if (ltmp2 == ERROR)

printf("二叉树集合中没有该二叉树！\n");

else

printf("该二叉树的位置为%d\n", LocateBiTree(BiTrees, BiTreeName3));

getchar();

getchar();

break;

case 5:

ClearBiTrees(&BiTrees);

printf("清空成功！\n");

getchar();

getchar();

break;

case 6:

ltmp3 = BiTreesEmpty(BiTrees);

if (ltmp3 == TRUE)

printf("该二叉树集合为空！\n");

else

printf("该二叉树集合不为空！\n");

getchar();

getchar();

break;

case 7:

printf("该二叉树集合长度为%d\n", BiTreesLength(BiTrees));

getchar();

getchar();

break;

case 8:

if (!PrintAllBiTrees(BiTrees))

printf("该二叉树集合中没有元素！\n");

getchar();

getchar();

break;

case 9:

flag3 = 0;

printf("请输入需要操作的二叉树的序号：\n");

scanf("%d", &num1);

num = num1 - 1;

while (num1 < 1 || num1 > BiTrees.length) {

printf("输入序号有误！\n");

scanf("%d", &num1);

}

while (op2) {

if (flag3 == 1)

{

op2=1;

break;

} //链表已经被销毁，就退回一级菜单

system("cls");

printf("\n\n");

printf("Menu for BiTree \n");

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

printf(" 1. CreateBiTree 2. ClearBiTree\n");

printf(" 3. BiTreeDepth 4. LocateNode\n");

printf(" 5. Assign 6. GetSibling \n");

printf(" 7. InsertNode 8. DeleteNode \n");

printf(" 9. Trabverse 10.SaveBiTree \n");

printf(" 11.DestroyBiTree 0. Exit\n");

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

printf("请选择你的操作[0~11]:");

scanf("%d", &op2);

switch (op2) {

case 0:

printf("按回车退出！\n");

break;

case 1:

k = 1;

printf("请选择通过键盘对二叉树中元素赋值（1）或者通过读入文件赋值（2）：\n");

scanf("%d", &choice);

if (choice == 1)

{

printf("请输入需要输入的元素个数：\n");

scanf("%d", &n);

definition = (TElemType\*)malloc(sizeof(TElemType) \* n);

p=definition;

printf("请输入数据(输入结点数值为-1结束）：\n");

for (i = 0; i < n+1; i++)

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

CreateBiTree(BiTrees.elem[num].T, definition);

printf("输入成功\n");

}

else

{

printf("请输入文件名：\n");

scanf("%s", name);

temp0 = LoadBiTree(BiTrees.elem[num].T, name);

if (temp0 == ERROR) printf("读入失败！\n");

else

printf("读入成功！\n");

}

getchar();

getchar();

break;

case 2:

tmp0 = ClearBiTree(BiTrees.elem[num].T);

if (tmp0 == OK)

printf("二叉树清空成功！\n");

getchar();

getchar();

break;

case 3:

tmp0 = BiTreeDepth(BiTrees.elem[num].T);

printf("二叉树深度为%d\n", tmp0);

getchar();

getchar();

break;

case 4:

printf("请输入需要查找的元素的数值：\n");

scanf("%d", &tmp1);

temp = LocateNode(BiTrees.elem[num].T, tmp1);

if (temp) printf("二叉树中存在该元素\n");

else printf("二叉树中没有该元素\n");

getchar();

getchar();

break;

case 5:

printf("请输入需要赋值的结点的数值：\n");

scanf("%d", &flag2);

printf("请输入新的数值和串：\n");

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

tmp2 = Assign(BiTrees.elem[num].T, flag2, value);

if (tmp2 == ERROR) printf("赋值失败\n");

else printf("赋值成功\n");

getchar();

getchar();

break;

case 6:

printf("请输入需要获取其兄弟结点的结点数值：\n");

scanf("%d", &tmp3);

temp = GetSibling(BiTrees.elem[num].T, tmp3);

if (temp == NULL) printf("获取失败\n");

else

printf("兄弟结点的数据为：%d\t%s", temp->data.key, temp->data.others);

getchar();

getchar();

break;

case 7:

printf("请输入插入位置的数据、方式、待插入结点的数值和串:\n");

scanf("%d%d%d%s", &tmp4, &LR, &c.key, c.others);

if (InsertNode(BiTrees.elem[num].T, tmp4, LR, c) == ERROR) printf("插入失败\n");

else printf("插入成功\n");

getchar();

getchar();

break;

case 8:

printf("请输入需要删除的结点的数值：\n");

scanf("%d", &elem2);

if (DeleteNode(BiTrees.elem[num].T, elem2) == ERROR)

printf("删除失败\n");

else

printf("删除成功\n");

getchar();

getchar();

break;

case 9:

if(BiTrees.elem[num].T==NULL)

printf("树中没有元素！\n");

else

{

printf("前序：\n");

PreOrderTraverse(BiTrees.elem[num].T, &visit);

printf("中序：\n");

InOrderTraverse(BiTrees.elem[num].T, &visit);

printf("后序：\n");

PostOrderTraverse(BiTrees.elem[num].T, &visit);

printf("层序：\n");

LevelOrderTraverse(BiTrees.elem[num].T, &visit);

}

getchar();

getchar();

break;

case 10:

printf("请输入储存数据的文件名:\n");

scanf("%s", name2);

temp00 = SaveBiTree(BiTrees.elem[num].T, name2);

if (temp00 == INFEASIBLE)

printf("链表未初始化！\n");

else if (temp00 == ERROR) printf("写入失败！\n");

else

printf("写入成功！\n");

getchar();

getchar();

break;

case 11:

flag3 = 0; //flag3作用：代表销毁状态

ClearBiTree(BiTrees.elem[num].T);

RemoveBiTree(&BiTrees, BiTrees.elem[num].name);

printf("链表销毁成功！\n");

flag3 = 1;

getchar();

getchar();

break;

}

}

op2=1;

if(flag3==0)

{

getchar();

getchar();

break;

}

}

}

return 0;

}

status judge(TElemType definition[])

{

int i = 0, j = 0, judge[100] = { 0 };

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

judge[definition[i].key]++;

for (j = 1; j <= i; j++)

if (judge[j] > 1) return ERROR;

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。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 \*\*\*\*\*\*\*\*\*/

if (judge(definition) == ERROR) return ERROR;

if ((\*p).key == -1) return OK;

else

{

if ((\*p).key == 0)

T = NULL;

else {

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

T->lchild = NULL;

T->rchild = NULL;

T->data = \*p;

p++;

if ((\*p).key == -1) return OK;

CreateBiTree(T->lchild, p);

if ((\*p).key == -1) return OK;

p++;

if ((\*p).key == -1) return OK;

CreateBiTree(T->rchild, p);

}

}

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

}

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;

else

{

ClearBiTree(T->rchild);

ClearBiTree(T->lchild);

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;

else

{

int dept = 0;

int ldept = BiTreeDepth(T->lchild);

int rdept = BiTreeDepth(T->rchild);

dept = ldept >= rdept ? (ldept + 1) : (rdept + 1);

return dept;

}

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

}

BiTNode\* LocateNode(BiTree T, KeyType e)

//查找结点

{

// 请在这里补充代码，完成本关任务

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

if (T == NULL) return NULL;

else

{

if (T->data.key == e) return T;

else {

temp1=LocateNode(T->lchild, e);

if(temp1) return temp1;

else

{

temp1=LocateNode(T->rchild, e);

if(temp1) return temp1;

else return NULL;

}

}

}

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

}

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

//实现结点赋值。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 temp;

if (T == NULL) return ERROR;

else

{

if (T->data.key == e)

{

if (e > value.key)

return ERROR;

T->data = value;

return OK;

}

else

{

temp=Assign(T->lchild, e, value);

if(temp) return OK;

else

{

temp=Assign(T->rchild, e, value);

if(temp) return OK;

else return ERROR;

}

}

}

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

}

BiTNode\* GetSibling(BiTree T, KeyType e)

//实现获得兄弟结点

{

// 请在这里补充代码，完成本关任务

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

if (T == NULL) return NULL;

else

{

if (!T->lchild || !T->rchild)

return NULL;

else if (T->lchild->data.key == e)

return T->rchild;

else if (T->rchild->data.key == e)

return T->lchild;

else

{

temp1=GetSibling(T->lchild, e);

if(temp1) return temp1;

else{

temp1=GetSibling(T->rchild, e);

if(temp1) return temp1;

else return NULL;

}

}

}

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

}

status Traverse(BiTree T, KeyType e)

{

int temp;

if (T)

{

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

temp=Traverse(T->lchild, e);

if(temp) return OK;

else{

temp=Traverse(T->rchild, e);

if(temp) return OK;

}

}

}

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

{

int temp;

if (T == NULL) return ERROR;

if (LR == -1 && Traverse(T, e))

{

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

m->data = c;

m->rchild = T;

m->lchild = NULL;

T = m;

return OK;

}

else

{

if (c.key <= e) return ERROR;

else if (LR == 0 && T->data.key == e)

{

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

m->data = c;

m->rchild = T->lchild;

m->lchild = NULL;

T->lchild = m;

return OK;

}

else if (LR == 1 && T->data.key == e)

{

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

m->data = c;

m->rchild = T->rchild;

m->lchild = NULL;

T->rchild = m;

return OK;

}

else

{

temp=InsertNode(T->lchild, e, LR, c);

if(temp) return OK;

else{

temp=InsertNode(T->rchild, e, LR, c);

if(temp) return OK;

else return ERROR;

}

}

}

}

BiTNode\* GetFather(BiTree T, KeyType e)

//实现获得父结点

{

// 请在这里补充代码，完成本关任务

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

BiTree temp;

if (T == NULL) return NULL;

else

{

if ((T->lchild && T->lchild->data.key == e) || (T->rchild && T->rchild->data.key == e))

return T;

else

{

temp=GetFather(T->lchild, e);

if(temp) return temp;

else{

temp=GetFather(T->rchild, e);

if(temp) return temp;

else return NULL;

}

}

}

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

}

BiTNode\* Getlast(BiTree T)

{

// 请在这里补充代码，完成本关任务

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

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

return T;

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

}

status DeleteNode(BiTree& T, KeyType e)

{

if (T == NULL) return ERROR;

if (T->data.key == e)

{

BiTree m = T;

if (T->lchild && !T->rchild)

{

T = T->lchild;

free(m);

return OK;

}

if (T->rchild && !T->lchild)

{

T = T->rchild;

free(m);

return OK;

}

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

{

BiTree n = Getlast(T->lchild);

n->rchild = T->rchild;

T = T->lchild;

free(m);

return OK;

}

if(!T->lchild&&!T->rchild)

{

free(T);

T=NULL;

return OK;

}

}

BiTree m = GetFather(T, e);

if (m == NULL) return ERROR;

else

{

if (m->lchild&&m->lchild->data.key == e)

{

BiTree n = m->lchild;

if (!n->lchild && !n->rchild)

{

free(n);

n = NULL;

m->lchild = NULL;

return OK;

}

else if (n->lchild && !n->rchild)

{

m->lchild = n->lchild;

free(n);

n = NULL;

return OK;

}

else if (n->rchild && !n->lchild)

{

m->lchild = n->rchild;

free(n);

n = NULL;

return OK;

}

else

{

m->lchild = n->lchild;

BiTree p = Getlast(n->lchild);

p->rchild = n->rchild;

free(n);

n = NULL;

return OK;

}

}

if (m->rchild&&m->rchild->data.key == e)

{

BiTree n = m->rchild;

if (!n->lchild && !n->rchild)

{

free(n);

n = NULL;

m->rchild = NULL;

return OK;

}

else if (n->lchild && !n->rchild)

{

m->rchild = n->lchild;

free(n);

n = NULL;

return OK;

}

else if (n->rchild && !n->lchild)

{

m->rchild = n->rchild;

free(n);

n = NULL;

return OK;

}

else

{

m->rchild = n->lchild;

BiTree p = Getlast(n->lchild);

p->rchild = n->rchild;

free(n);

n = NULL;

return OK;

}

}

}

}

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

//先序遍历二叉树T

{

if (T)

{

visit(T);

PreOrderTraverse(T->lchild, visit);

PreOrderTraverse(T->rchild, visit);

return OK;

}

}

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

//中序遍历二叉树T

{

BiTree pt[100];

int top = 0;

do

{

while (T)

{

pt[top++] = T;

T = T->lchild;

}

if (top)

{

T = pt[--top];

visit(T);

T = T->rchild;

}

} while (top || T);

return OK;

}

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

//后序遍历二叉树T

{

if (T)

{

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

{

BiTree pt[100];

int in = 0, out = 0;

pt[in++] = T;

while (in > out)

{

if (pt[out])

{

visit(pt[out]);

pt[in++] = pt[out]->lchild;

pt[in++] = pt[out]->rchild;

}

out++;

}

return OK;

}

status SaveBiTree(BiTree T, char FileName[])

//将二叉树的结点数据写入到文件FileName中

{

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

if (T == NULL) return INFEASIBLE;

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

if (f == NULL) return ERROR;

BiTree BFS[100];

int num[100];

int top = 0;

int bottom = 0;

int top1 = 0;

int bottom1 = 0;

BFS[++top] = T;

num[++top1] = 1;

while (bottom != top)

{

int n = num[++bottom1];

BiTree temp = BFS[++bottom];

fprintf(f, "%d %d %s ", n, bottom, temp->data.others);

if (temp->lchild)

{

num[++top1] = n \* 2;

BFS[++top] = temp->lchild;

}

if (temp->rchild)

{

num[++top1] = n \* 2 + 1;

BFS[++top] = temp->rchild;

}

}

fprintf(f, "0 0 null");

fclose(f);

return OK;

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

}

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

{

BiTree pt[100];

int n, key;

char str[100];

FILE\* fp;

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

return ERROR;

fscanf(fp, "%d%d%s", &n, &key, str);

while (n)

{

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

p->lchild = NULL;

p->rchild = NULL;

p->data.key = key;

strcpy(p->data.others, str);

pt[n] = p;

if (n == 1) T = p;

else

{

if (!(n % 2)) pt[n / 2]->lchild = p;

else pt[n / 2]->rchild = p;

}

fscanf(fp, "%d%d%s", &n, &key, str);

}

fclose(fp);

return OK;

}

status AddBiTree(BITREES\* BiTrees, char BiTreeName[])

// 只需要在BiTrees中增加一个名称为BiTreeName的空二叉树，二叉树数据由后台测试程序插入。

{

strcpy(BiTrees->elem[BiTrees->length].name, BiTreeName);

BiTrees->elem[BiTrees->length].T = NULL;

BiTrees->length++;

return OK;

}

status RemoveBiTree(BITREES\* BiTrees, char BiTreeName[])

// BiTrees中删除一个名称为BiTreeName的二叉树

{

int i, k;

for (i = 0; i < BiTrees->length; i++) {

if (strcmp(BiTrees->elem[i].name, BiTreeName))

continue;

else {

BiTrees->elem[i].name[0] = '\0';

free(BiTrees->elem[i].T);

BiTrees->elem[i].T = NULL;

for (k = i; k < BiTrees->length; k++)

BiTrees->elem[k] = BiTrees->elem[k + 1];

BiTrees->length--;

return OK;

}

}

return ERROR;

}

status LocateBiTree(BITREES BiTrees, char BiTreeName[])

// 在BiTrees中查找一个名称为BiTreeName的二叉树，成功返回逻辑序号，否则返回0

{

int i;

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

if (strcmp(BiTrees.elem[i].name, BiTreeName))

continue;

else

return i + 1;

}

return ERROR;

}

status ClearBiTrees(BITREES\* BiTrees) {

//清空二叉树集合

BiTrees->length = 0;

return OK;

}

status BiTreesEmpty(BITREES BiTrees) {

//判断二叉树集合是否为空

if (BiTrees.length == 0)

return TRUE;

else

return FALSE;

}

status BiTreesLength(BITREES BiTrees) {

//求二叉树集合的长度

return BiTrees.length;

}

status PrintAllBiTrees(BITREES BiTrees) {

//打印二叉树集合中的所有元素

if (BiTrees.length == 0) return ERROR;

else {

printf("所有二叉树依次为:\n");

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

printf("%d\t%s\n", i + 1, BiTrees.elem[i].name);

}

}

return OK;

}

status InitBiTrees(BITREES\* BiTrees) {

//初始化二叉树集合

BiTrees->length = 0;

return OK;

}

void visit(BiTree T)

{

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

}

**4 基于邻接表的图实现源程序**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 20

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;

int arcnum; //顶点数、弧数

GraphKind kind; //图的类型

} ALGraph;

typedef struct { //图的集合类型定义

struct {

char name[30]; //图名称

ALGraph G; //图

} elem[10]; //图数组

int length; //图集合长度

}ALGraphs;

typedef struct{

int queue[MAX\_VERTEX\_NUM];

int front;

int tail;

}Queue;

int visited[MAX\_VERTEX\_NUM]={0};

void (\*Visit)(VertexType);

void initqueue(Queue &Q);

int queueempty(Queue Q);

void inqueue(Queue &Q,int v);

int dequeue(Queue &Q,int e);

status Judge(int a[], int n);

status CreateGraph(ALGraph &G, VertexType V[], KeyType VR[][2]);

status DestroyGraph(ALGraph &G);

void print(ALGraph G);

int LocateVex(ALGraph G,KeyType u);

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

int FirstAdjVex(ALGraph G,KeyType u);

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

status InsertVex(ALGraph &G,VertexType v);

status DeleteVex(ALGraph &G,KeyType v);

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

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

void visit(VertexType v);

void DFS(ALGraph &G,int v);

status DFSTraverse(ALGraph &G,void (\*visit)(VertexType));

status BFSTraverse(ALGraph &G,void (\*visit)(VertexType));

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

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

status AddALGraph(ALGraphs\* ALGraphs, char ALGraphName[]);

status RemoveALGraph(ALGraphs\* ALGraphs, char ALGraphName[]);

status LocateALGraph(ALGraphs ALGraphs, char ALGraphName[]);

status ClearALGraphs(ALGraphs\* ALGraphs);

status ALGraphsEmpty(ALGraphs ALGraphs);

status ALGraphsLength(ALGraphs ALGraphs);

status PrintAllALGraphs(ALGraphs ALGraphs);

status InitALGraphs(ALGraphs\* ALGraphs);

void printall(ALGraph G);

int main() {

ALGraphs ALGraphs;

VertexType value,c,a[100];

KeyType b[100][2];

char ALGraphName1[100], ALGraphName2[100], ALGraphName3[100], name[100], name2[100];

int op1 = 1, op2 = 1, n, k = 1, choice, i, j, flag = 0, flag2 = 0,flag3=0, e, LR;

int ltmp1, ltmp2, ltmp3;

int num1, num;

int tmp1, tmp2, tmp3, tmp4, tmp0, temp0, temp00, elem2,temp;

int v,w;

while (op1) {

system("cls");

printf("\n\n");

printf("Menu for ALgraph \n");

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

printf(" 1.InitALgraphs 2.AddALgraph\n");

printf(" 3.RemoveALgraph 4.LocateALgraph\n");

printf(" 5.ClearALgraphs 6.ALgraphsEmpty\n");

printf(" 7.ALgraphsLength 8.PrintAllALgraphs\n");

printf(" 9.OperateALgraph 0.Exit\n");

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

printf("请选择你的操作[0~9]:");

scanf("%d", &op1);

switch (op1) {

case 0:

break;

case 1:

InitALGraphs(&ALGraphs);

printf("初始化成功！\n");

getchar();

getchar();

break;

case 2:

printf("请输入需要添加的图的名称：\n");

do

{

flag = 0;

scanf("%s", ALGraphName1);

for (j = 0; j < ALGraphs.length; j++)

if (!strcmp(ALGraphName1, ALGraphs.elem[j].name)) //查找集合中是否早已又该该名称的图

{

flag = 1;

printf("名称重复！请重新输入！\n");

break;

}

} while (flag == 1);

AddALGraph(&ALGraphs, ALGraphName1);

printf("添加成功！\n");

getchar();

getchar();

break;

case 3:

printf("请输入需要删除的图的名称：\n");

scanf("%s", ALGraphName2);

ltmp1 = RemoveALGraph(&ALGraphs, ALGraphName2);

if (ltmp1 == ERROR)

printf("图集合中没有该图！\n");

else

printf("删除成功！\n");

getchar();

getchar();

break;

case 4:

printf("请输入需要定位的图的名称：\n");

scanf("%s", ALGraphName3);

ltmp2 = LocateALGraph(ALGraphs, ALGraphName3);

if (ltmp2 == ERROR)

printf("图集合中没有该图！\n");

else

printf("该图的位置为%d\n", LocateALGraph(ALGraphs, ALGraphName3));

getchar();

getchar();

break;

case 5:

ClearALGraphs(&ALGraphs);

printf("清空成功！\n");

getchar();

getchar();

break;

case 6:

ltmp3 = ALGraphsEmpty(ALGraphs);

if (ltmp3 == TRUE)

printf("该图集合为空！\n");

else

printf("该图集合不为空！\n");

getchar();

getchar();

break;

case 7:

printf("该图集合长度为%d\n", ALGraphsLength(ALGraphs));

getchar();

getchar();

break;

case 8:

if (!PrintAllALGraphs(ALGraphs))

printf("该图集合中没有元素！\n");

getchar();

getchar();

break;

case 9:

flag3 = 0;

printf("请输入需要操作的图的序号：\n");

scanf("%d", &num1);

num = num1 - 1;

while (num1 < 1 || num1 > ALGraphs.length) {

printf("输入序号有误！\n");

scanf("%d", &num1);

}

while (op2) {

if (flag3 == 1)

{

op2=1;

break;

} //图已经被销毁，就退回一级菜单

system("cls");

printf("\n\n");

printf("Menu for ALgraph \n");

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

printf(" 1. CreateALgraph 2. DestroyALgraph\n");

printf(" 3. LocateVex 4. PutVex\n");

printf(" 5. FirstAdjVex 6. NextAdjVex \n");

printf(" 7. InsertVex 8. DeleteVex \n");

printf(" 9. InsertArc 10.DeleteArc \n");

printf(" 11.printall 12.DFS&BFS\n");

printf(" 13.SaveGraph 0.Exit \n");

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

printf("请选择你的操作[0~13]:");

scanf("%d", &op2);

switch (op2) {

case 0:

printf("按回车退出！\n");

break;

case 1:

k = 1;

printf("请选择通过键盘对图中元素赋值（1）或者通过读入文件赋值（2）：\n");

scanf("%d", &choice);

if (choice == 1)

{

printf("请输入数据(输入结点数值为-1结束）：\n");

for (i = 0; i < n+1; i++)

CreateGraph(ALGraphs.elem[num].G,a,b);

printf("输入成功\n");

}

else

{

printf("请输入文件名：\n");

scanf("%s", name);

temp0 = LoadGraph(ALGraphs.elem[num].G, name);

if (temp0 == ERROR) printf("读入失败！\n");

else

printf("读入成功！\n");

}

getchar();

getchar();

break;

case 2:

flag3=0;

tmp0 = DestroyGraph(ALGraphs.elem[num].G);

RemoveALGraph(&ALGraphs,ALGraphs.elem[num].name);

if (tmp0 == OK)

printf("图销毁成功！\n");

flag3=1;

getchar();

getchar();

break;

case 3:

if(ALGraphs.elem[num].G.vexnum==0)

{

printf("图为空！\n");

getchar();

getchar();

break;

}

printf("请输入需要查找的元素的数值：\n");

scanf("%d", &tmp1);

temp = LocateVex(ALGraphs.elem[num].G, tmp1);

if (temp!=-1)

printf("该结点位序为%d,该结点数据为 %d %s\n",temp,ALGraphs.elem[num].G.vertices[temp].data.key,ALGraphs.elem[num].G.vertices[temp].data.others);

else printf("图中没有该元素\n");

getchar();

getchar();

break;

case 4:

if(ALGraphs.elem[num].G.vexnum==0)

{

printf("图为空！\n");

getchar();

getchar();

break;

}

printf("请输入需要赋值的结点的数值：\n");

scanf("%d", &flag2);

printf("请输入新的数值和串：\n");

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

tmp2 = PutVex(ALGraphs.elem[num].G, flag2, value);

if (tmp2 == ERROR) printf("赋值失败\n");

else printf("赋值成功\n");

getchar();

getchar();

break;

case 5:

if(ALGraphs.elem[num].G.vexnum==0)

{

printf("图为空！\n");

getchar();

getchar();

break;

}

printf("请输入需要获取其第一邻接结点的结点数值：\n");

scanf("%d", &tmp3);

temp =FirstAdjVex(ALGraphs.elem[num].G, tmp3);

if (temp==-1) printf("获取失败\n");

else

printf("该结点位序为%d,该结点数据为 %d %s\n",temp,ALGraphs.elem[num].G.vertices[temp].data.key,ALGraphs.elem[num].G.vertices[temp].data.others);

getchar();

getchar();

break;

case 6:

if(ALGraphs.elem[num].G.vexnum==0)

{

printf("图为空！\n");

getchar();

getchar();

break;

}

printf("请输入查找顶点和相对顶点：\n");

scanf("%d%d",&v,&w);

temp0=NextAdjVex(ALGraphs.elem[num].G,v,w);

if (temp0==-1) printf("获取失败\n");

else

printf("该结点位序为%d,该结点数据为 %d %s\n",temp0,ALGraphs.elem[num].G.vertices[temp0].data.key,ALGraphs.elem[num].G.vertices[temp0].data.others);

getchar();

getchar();

break;

case 7:

if(ALGraphs.elem[num].G.vexnum==0)

{

printf("图为空！\n");

getchar();

getchar();

break;

}

printf("请输入待插入结点的数值和串:\n");

scanf("%d%s",&c.key, c.others);

if (InsertVex(ALGraphs.elem[num].G,c) == ERROR) printf("插入失败\n");

else printf("插入成功\n");

getchar();

getchar();

break;

case 8:

if(ALGraphs.elem[num].G.vexnum==0)

{

printf("图为空！\n");

getchar();

getchar();

break;

}

printf("请输入需要删除的结点的数值：\n");

scanf("%d", &elem2);

if (DeleteVex(ALGraphs.elem[num].G, elem2) == ERROR)

printf("删除失败\n");

else

printf("删除成功\n");

getchar();

getchar();

break;

case 9:

if(ALGraphs.elem[num].G.vexnum==0)

{

printf("图为空！\n");

getchar();

getchar();

break;

}

printf("请输入需要增加的弧的两个顶点：\n");

scanf("%d%d",&v,&w);

temp=InsertArc(ALGraphs.elem[num].G,v,w);

if(temp==ERROR) printf("插入失败\n");

else printf("插入成功\n");

getchar();

getchar();

break;

case 10:

if(ALGraphs.elem[num].G.vexnum==0)

{

printf("图为空！\n");

getchar();

getchar();

break;

}

printf("请输入需要删除的弧的两个顶点：\n");

scanf("%d%d",&v,&w);

temp=DeleteArc(ALGraphs.elem[num].G,v,w);

if(temp==ERROR) printf("删除失败\n");

else printf("删除成功\n");

getchar();

getchar();

break;

case 11:

if(ALGraphs.elem[num].G.vexnum==0)

{

printf("图为空！\n");

getchar();

getchar();

break;

}

printf("所有数据为：\n");

printall(ALGraphs.elem[num].G);

getchar();

getchar();

break;

case 12:

if(ALGraphs.elem[num].G.vexnum==0)

{

printf("图为空！\n");

getchar();

getchar();

break;

}

printf("深度优先搜索：\n");

DFSTraverse(ALGraphs.elem[num].G,visit);

printf("广度优先搜索：\n");

BFSTraverse(ALGraphs.elem[num].G,visit);

getchar();

getchar();

break;

case 13:

if(ALGraphs.elem[num].G.vexnum==0)

{

printf("图为空！\n");

getchar();

getchar();

break;

}

printf("请输入储存数据的文件名:\n");

scanf("%s", name2);

temp00 = SaveGraph(ALGraphs.elem[num].G, name2);

if (temp00 == ERROR) printf("写入失败！\n");

else

printf("写入成功！\n");

getchar();

getchar();

break;

}

}

op2=1;

if(flag3==0)

{

getchar();

getchar();

break;

}

}

}

return 0;

}

void initqueue(Queue &Q)

{

Q.front=0;

Q.tail=-1;

memset(Q.queue,0,sizeof(Q.queue));

}

int queueempty(Queue Q)

{

if(Q.tail==-1)

return 0;

return 1;

}

void inqueue(Queue &Q,int v)

{

Q.queue[++Q.tail]=v;

}

int dequeue(Queue &Q,int e)

{

e=Q.queue[Q.front];

for(int i=0;i<Q.tail;i++)

{

Q.queue[i]=Q.queue[i+1];

}

Q.tail--;

return e;

}

status Judge(int a[], int n) {

int judge[100] = { 0 };

for (int i = 0; i < n; i++)

judge[a[i]]++;

for (int j = 0; j < n; j++)

if (judge[j] > 1)

return ERROR;

return OK;

}

status CreateGraph(ALGraph &G, VertexType V[], KeyType VR[][2]) {

int key[100], k = 0, cnt = 0, m = 0,l=0, i, j;

ArcNode \*p,\*q;

G.kind = UDG;

G.arcnum = 0;

G.vexnum = 0;

scanf("%d%s", &V[l].key, V[l].others);

if(V[l].key==-1) return ERROR;

while (V[l].key != -1) {

key[k] = V[l].key;

if (Judge(key, k) == ERROR)

return ERROR;

k++;

G.vertices[cnt].data.key = V[l].key;

strcpy(G.vertices[cnt].data.others, V[l].others);

G.vertices[cnt].firstarc = NULL;

cnt++;

G.vexnum++;

l++;

scanf("%d%s", &V[l].key, V[l].others);

}

if(G.vexnum>20) return ERROR;

scanf("%d%d", &VR[m][0], &VR[m][1]);

while (VR[m][0] != -1) {

p = (ArcNode \*)malloc(sizeof(ArcNode));

q = (ArcNode \*)malloc(sizeof(ArcNode));

for (i = 0; i < G.vexnum; i++)

if (G.vertices[i].data.key == VR[m][1])

break;

if(i==G.vexnum) return ERROR;

for (j = 0; j < G.vexnum; j++)

if (G.vertices[j].data.key == VR[m][0])

break;

if(j==G.vexnum) return ERROR;

p->adjvex = i;

q->adjvex = j;

p->nextarc = G.vertices[j].firstarc;

q->nextarc = G.vertices[i].firstarc;

G.vertices[j].firstarc = p;

G.vertices[i].firstarc = q;

G.arcnum++;

m++;

scanf("%d%d", &VR[m][0], &VR[m][1]);

}

return OK;

}

status DestroyGraph(ALGraph &G)

/\*销毁无向图G,删除G的全部顶点和边\*/

{

ArcNode \*p,\*q;

for(int i=0;i<G.vexnum;i++)

{

if(G.vertices[i].firstarc==NULL)

continue;

p=G.vertices[i].firstarc->nextarc;

q=G.vertices[i].firstarc;

while(p){

q=p->nextarc;

free(p);

p=q;

}

p=G.vertices[i].firstarc;

free(p);

G.vertices[i].firstarc=NULL;

}

G.arcnum=0;

G.vexnum=0;

return OK;

}

void print(ALGraph G)

{

for(int i=0;i<G.vexnum;i++)

{

ArcNode \*p=G.vertices[i].firstarc;

printf("%d %s ",G.vertices[i].data.key,G.vertices[i].data.others);

while(p)

{

printf("%d ",p->adjvex);

p=p->nextarc;

}

printf("\n");

}

}

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；

{

// 请在这里补充代码，完成本关任务

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

for(int i=0;i<G.vexnum;i++)

if(G.vertices[i].data.key==u)

return i;

return -1;

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

}

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 flag[20]={0};

flag[value.key]++;

for(int i=0;i<G.vexnum;i++)

flag[G.vertices[i].data.key]++;

if(flag[u]==0||flag[value.key]==2&&value.key!=u)

return ERROR;

for(int i=0;i<G.vexnum;i++)

if(G.vertices[i].data.key==u)

{

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 \*\*\*\*\*\*\*\*\*/

for(int i=0;i<G.vexnum;i++)

{

if(G.vertices[i].data.key==u)

{

if(G.vertices[i].firstarc==NULL)

return -1;

return G.vertices[i].firstarc->adjvex;

}

}

return -1;

/\*\*\*\*\*\*\*\*\*\* 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 flag[20]={0},k=0;

for(int i=0;i<G.vexnum;i++)

{

flag[G.vertices[i].data.key]++;

if(G.vertices[i].data.key==w)

k=i;

}

if(flag[v]==0)

return -1;

for(int i=0;i<G.vexnum;i++)

{

if(G.vertices[i].data.key==v)

{

ArcNode \*p=G.vertices[i].firstarc;

while(p)

{

if(p->adjvex==k)

{

if(p->nextarc==NULL) return -1;

else return p->nextarc->adjvex;

}

p=p->nextarc;

}

return -1;

}

}

/\*\*\*\*\*\*\*\*\*\* 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 flag[20]={0};

flag[v.key]++;

for(int i=0;i<G.vexnum;i++)

flag[G.vertices[i].data.key]++;

if(flag[v.key]==2||G.vexnum+1>20)

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

{

// 请在这里补充代码，完成本关任务

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

if(G.vexnum==1&&v==G.vertices[0].data.key) return ERROR;

int flag[20]={0},i;

for(int i=0;i<G.vexnum;i++)

flag[G.vertices[i].data.key]++;

if(flag[v]==0)

return ERROR;

ArcNode \*p,\*q;

for(i=0;i<G.vexnum;i++) //删除结点

{

if(G.vertices[i].data.key==v)

{

if(G.vertices[i].firstarc==NULL)

break;

p=G.vertices[i].firstarc->nextarc;

q=G.vertices[i].firstarc;

while(p){

q=p->nextarc;

free(p);

p=q;

G.arcnum--;

}

p=G.vertices[i].firstarc;

free(p);

G.arcnum--;

G.vertices[i].firstarc=NULL;

break;

}

}

for(int j=i+1;j<G.vexnum;j++) //后面节点前移

{

G.vertices[j-1].data.key=G.vertices[j].data.key;

strcpy(G.vertices[j-1].data.others,G.vertices[j].data.others);

G.vertices[j-1].firstarc=G.vertices[j].firstarc;

}

G.vexnum--;

for(int k=0;k<G.vexnum;k++)

{

if(G.vertices[k].firstarc==NULL) continue;

ArcNode \*p=G.vertices[k].firstarc,\*r=G.vertices[k].firstarc,\*q=p->nextarc;

if(p->adjvex==i)

{

G.vertices[k].firstarc=q;

free(p);

continue;

}

p=G.vertices[k].firstarc->nextarc;

while(p)

{

if(p->adjvex==i)

{

r->nextarc=p->nextarc;

free(p);

break;

}

p=p->nextarc;

r=r->nextarc;

}

}

for(int l=0;l<G.vexnum;l++)

{

ArcNode \*p=G.vertices[l].firstarc;

while(p)

{

if(p->adjvex>i) p->adjvex--;

p=p->nextarc;

}

}

return OK;

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

}

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 \*\*\*\*\*\*\*\*\*/

int flag[20]={0},i,j;

for(int i=0;i<G.vexnum;i++)

flag[G.vertices[i].data.key]++;

if(flag[v]==0||flag[w]==0)

return ERROR;

for (i = 0; i < G.vexnum; i++)

if (G.vertices[i].data.key == v)

break;

for (j = 0; j < G.vexnum; j++)

if (G.vertices[j].data.key == w)

break;

ArcNode \*p=G.vertices[i].firstarc;

while(p)

{

if(p->adjvex==j) return ERROR;

p=p->nextarc;

}

p=(ArcNode \*)malloc(sizeof(ArcNode));

p->adjvex=j;

p->nextarc=G.vertices[i].firstarc;

G.vertices[i].firstarc=p;

ArcNode \*q=(ArcNode \*)malloc(sizeof(ArcNode));

q->adjvex=i;

q->nextarc=G.vertices[j].firstarc;

G.vertices[j].firstarc=q;

return OK;

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

}

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 \*\*\*\*\*\*\*\*\*/

int flag[20]={0},i,j,tag=0;

for(int i=0;i<G.vexnum;i++)

flag[G.vertices[i].data.key]++;

if(flag[v]==0||flag[w]==0)

return ERROR;

for (i = 0; i < G.vexnum; i++)

if (G.vertices[i].data.key == v)

break;

for (j = 0; j < G.vexnum; j++)

if (G.vertices[j].data.key == w)

break;

ArcNode \*m=G.vertices[i].firstarc;

while(m)

{

if(m->adjvex==j)

{

tag=1;

break;

}

m=m->nextarc;

}

if(tag==0) return ERROR;

ArcNode \*p=G.vertices[i].firstarc,\*r=G.vertices[i].firstarc,\*q=p->nextarc;

if(p->adjvex==j)

{

G.vertices[i].firstarc=q;

free(p);

p=NULL;

}

else

{

p=G.vertices[i].firstarc->nextarc;

while(p)

{

if(p->adjvex==j)

{

r->nextarc=p->nextarc;

free(p);

p=NULL;

break;

}

p=p->nextarc;

r=r->nextarc;

}

}

p=G.vertices[j].firstarc,r=G.vertices[j].firstarc,q=p->nextarc;

if(p->adjvex==i)

{

G.vertices[j].firstarc=q;

free(p);

p=NULL;

}

else

{

p=G.vertices[j].firstarc->nextarc;

while(p)

{

if(p->adjvex==i)

{

r->nextarc=p->nextarc;

free(p);

p=NULL;

break;

}

p=p->nextarc;

r=r->nextarc;

}

}

G.arcnum--;

return OK;

}

void visit(VertexType v)

{

printf("%d %s\n",v.key,v.others);

}

void DFS(ALGraph &G,int v)

{

visited[v]=1;

Visit(G.vertices[v].data);

for(int w=FirstAdjVex(G,G.vertices[v].data.key);w>=0;w=NextAdjVex(G,G.vertices[v].data.key,G.vertices[w].data.key))

{

if(!visited[w])

DFS(G,w);

}

}

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

{

// 请在这里补充代码，完成本关任务

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

Visit=visit;

for(int v=0;v<G.vexnum;v++)

{

if(!visited[v])

DFS(G,v);

}

memset(visited,0,sizeof(visited));

return OK;

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

}

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 \*\*\*\*\*\*\*\*\*/

Visit=visit;

Queue Q;

initqueue(Q);

for(int v=0;v<G.vexnum;v++)

{

if(!visited[v])

{

visited[v]=1;

Visit(G.vertices[v].data);

inqueue(Q,v);

while(queueempty(Q))

{

int e;

e=dequeue(Q,e);

for(int w=FirstAdjVex(G,G.vertices[e].data.key);w>=0;w=NextAdjVex(G,G.vertices[e].data.key,G.vertices[w].data.key))

{

if(!visited[w])

{

visited[w]=1;

Visit(G.vertices[w].data);

inqueue(Q,w);

}

}

}

}

}

memset(visited,0,sizeof(visited));

return 0;

/\*\*\*\*\*\*\*\*\*\* 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;

if((fp=fopen(FileName,"w"))==NULL)

return ERROR;

for(int v=0;v<G.vexnum;v++)

{

fprintf(fp,"%d %s ",G.vertices[v].data.key,G.vertices[v].data.others);

}

fprintf(fp,"-1 nil ");

for(int v=0;v<G.vexnum;v++)

{

ArcNode \*p=G.vertices[v].firstarc;

while(p)

{

fprintf(fp,"%d %d ",G.vertices[v].data.key,G.vertices[p->adjvex].data.key);

p=p->nextarc;

}

}

fprintf(fp,"-1 -1 ");

fprintf(fp,"%d ",G.arcnum);

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 \*\*\*\*\*\*\*\*\*/

ArcNode \*p,\*q;

G.kind = UDG;

G.arcnum = 0;

G.vexnum = 0;

FILE \*fp;

if((fp=fopen(FileName,"r"))==NULL)

return ERROR;

int key=0;

char others[100];

int cnt=0;

fscanf(fp,"%d%s",&key,others);

while(key!=-1)

{

G.vertices[cnt].data.key=key;

strcpy(G.vertices[cnt].data.others,others);

G.vertices[cnt].firstarc = NULL;

cnt++;

G.vexnum++;

fscanf(fp,"%d%s",&key,others);

}

int m,n,i,j;

fscanf(fp,"%d%d",&m,&n);

while (m!= -1) {

p = (ArcNode \*)malloc(sizeof(ArcNode));

for (i = 0; i < G.vexnum; i++)

if (G.vertices[i].data.key == n)

break;

for (j = 0; j < G.vexnum; j++)

if (G.vertices[j].data.key == m)

break;

ArcNode \*p1=G.vertices[j].firstarc;

while(p1&&p1->nextarc)

p1=p1->nextarc;

p->adjvex = i;

p->nextarc = NULL;

if(!p1) G.vertices[j].firstarc=p;

else p1->nextarc = p;

fscanf(fp,"%d%d",&m,&n);

}

fscanf(fp,"%d",&G.arcnum);

fclose(fp);

return OK;

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

}

status AddALGraph(ALGraphs\* ALGraphs, char ALGraphName[])

// 只需要在ALGraphs中增加一个名称为ALGraphName的空图，图数据由后台测试程序插入。

{

ALGraphs->elem[ALGraphs->length].G.kind = UDG;

ALGraphs->elem[ALGraphs->length].G.arcnum = 0;

ALGraphs->elem[ALGraphs->length].G.vexnum = 0;

strcpy(ALGraphs->elem[ALGraphs->length].name, ALGraphName);

ALGraphs->length++;

return OK;

}

status RemoveALGraph(ALGraphs\* ALGraphs, char ALGraphName[])

// ALGraphs中删除一个名称为ALGraphName的图

{

int i, k;

for (i = 0; i < ALGraphs->length; i++) {

if (strcmp(ALGraphs->elem[i].name, ALGraphName))

continue;

else {

ALGraphs->elem[i].name[0] = '\0';

for (k = i; k < ALGraphs->length; k++)

ALGraphs->elem[k] = ALGraphs->elem[k + 1];

ALGraphs->length--;

return OK;

}

}

return ERROR;

}

status LocateALGraph(ALGraphs ALGraphs, char ALGraphName[])

// 在ALGraphs中查找一个名称为ALGraphName的图，成功返回逻辑序号，否则返回0

{

int i;

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

if (strcmp(ALGraphs.elem[i].name, ALGraphName))

continue;

else

return i + 1;

}

return ERROR;

}

status ClearALGraphs(ALGraphs\* ALGraphs) {

//清空图集合

ALGraphs->length = 0;

return OK;

}

status ALGraphsEmpty(ALGraphs ALGraphs) {

//判断图集合是否为空

if (ALGraphs.length == 0)

return TRUE;

else

return FALSE;

}

status ALGraphsLength(ALGraphs ALGraphs) {

//求图集合的长度

return ALGraphs.length;

}

status PrintAllALGraphs(ALGraphs ALGraphs) {

//打印图集合中的所有元素

if (ALGraphs.length == 0) return ERROR;

else {

printf("所有图依次为:\n");

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

printf("%d\t%s\n", i + 1, ALGraphs.elem[i].name);

}

}

return OK;

}

status InitALGraphs(ALGraphs\* ALGraphs) {

//初始化图集合

ALGraphs->length = 0;

return OK;

}

void printall(ALGraph G)

{

for(int i=0;i<G.vexnum;i++)

{

printf("%d %s ",G.vertices[i].data.key,G.vertices[i].data.others);

ArcNode \*p=G.vertices[i].firstarc;

while(p)

{

printf("%d ",p->adjvex);

p=p->nextarc;

}

printf("\n");

}

}