

**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202008**

**学 号： U202015533**

**姓 名： 徐瑞达**

**指导教师： 袁凌**

**报告日期： 2021年 5月6日**

**计算机科学与技术学院**

目 录

[1 基于顺序存储结构的线性表实现 2](#_Toc458159879)

[1.1 问题描述 2](#_Toc458159880)

[1.2 系统设计 2](#_Toc458159882)

[1.3 系统实现 2](#_Toc458159883)

[1.4 实验小结 2](#_Toc458159884)

[2 基于链式存储结构的线性表实现 2](#_Toc458159885)

[2.1 问题描述 2](#_Toc458159886)

[2.2 系统设计 2](#_Toc458159887)

[2.3 系统实现 2](#_Toc458159888)

[2.4 实验小结 2](#_Toc458159889)

[3 基于二叉链表的二叉树实现 2](#_Toc458159890)

[3.1 问题描述 2](#_Toc458159891)

[3.2 系统设计 2](#_Toc458159892)

[3.3 系统实现 2](#_Toc458159893)

[3.4 实验小结 2](#_Toc458159894)

[4 基于二叉链表的二叉树实现 2](#_Toc458159895)

[4.1 问题描述 2](#_Toc458159896)

[4.2 系统设计 2](#_Toc458159897)

[4.3 系统实现 2](#_Toc458159898)

[4.4 实验小结 2](#_Toc458159899)

[参考文献 2](#_Toc458159900)

[附录A 基于顺序存储结构线性表实现的源程序 2](#_Toc458159901)

[附录B 基于链式存储结构线性表实现的源程序 2](#_Toc458159902)

[附录C 基于二叉链表二叉树实现的源程序 2](#_Toc458159903)

[附录D 基于邻接表图实现的源程序 2](#_Toc458159904)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

该实验要解决的基本问题是实现线性表的各个基本功能，如初始化表、销毁表、清空表、判定空表、求表长和获得元素等等。

其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示。可选择以文件的形式进行存储和加载，也即将生成的线性表存入到相应的文件中，也可以从文件中获取线性表进行操作。

## 1.1.1需实现的基本运算

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()。

⒀保存线性表：函数名称是SaveList(L,FileName)，初始条件是线性表L已存在；操作结果是将线性表元素按顺序保存至文件名为FileName的文件中。

⒁加载线性表：函数名称是LoadList(L,FileName)，初始条件是线性表L不存在；操作结果是将文件名为FileName的文件加载到线性表中。

⒂在管理表中插入一个线性表：函数名称是AddList(Lists,ListName)，无初始条件；操作结果是将文件名为ListName的线性表添加到管理表中。

⒃在管理表中删除一个线性表：函数名称是RemoveList(Lists, ListName)，初始条件是多线性表非空；操作结果是将文件名为ListName的线性表删除。

⒄在管理表中查找一个线性表：函数名称是LocateList(Lists, ListName)，初始条件是多线性表非空；操作结果是查找文件名为ListName的线性表并可对其进行操作。

## 1.2 系统设计

本次实验使用的数据结构定义为：

/\*-------- Macro Definition ---------\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 10

#define LISTINCREMENT 10

/\*-------- Declare the type ---------\*/

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

typedef struct{ //线性表的管理表定义

struct{

char name[30];

SqList L;

}elem[10];

int length;

int listsize;

}LISTS;

将菜单演示和用户选择写入到while循环中，用OP获取用户的选择，OP初始化为1,以便第一次能进入循环。进入循环后系统首先显示功能菜单，然后用户输入选择0-14,其中1-17分别代表对线性表或线性表的管理表的一个基本运算，在主函数中通过switch语句对应到相应的函数功能，执行完该功能后break跳出switch语句，继续执行while循环，直至用户输入0退出当前演示系统。 (演示系统结构如图1-1所示)

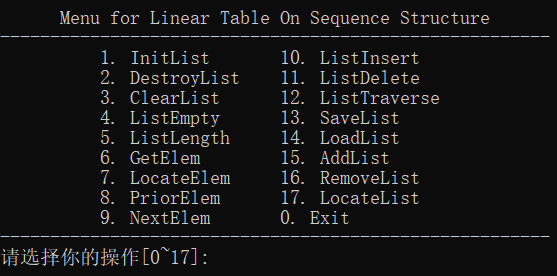


图1-1 演示系统结构图

## 1.3 系统实现

线性表运算算法思想与设计如下：

1．初始化线性表思想：将线性表初始化过程写成函数，其中传入函数的参数是主函数中定义的结构型变量L的引用。在函数中，首先判断L是否创建，如果未创建则使用malloc函数分配LIST\_INIT\_SIZE大小的连续内存空间，如果分配成功则将newbase赋值给L.elem，由于线性表的长度为0，将 L.length初始化为0，L.listsize初始化为LIST\_INIT\_SIZE，即完成了线性表的初始化。经分析，算法的时间复杂度为O(1)。

2．销毁线性表思想：将销毁线性表的过程写成函数，其中传入函数的参数是主函数中定义的结构性变量L的引用。在函数中，首先使用free函数释放掉以L.elem为首地址的连续内存空间，再将 L.length，L.listsize重新赋值为0。经分析，该算法的时间复杂度为O(1)。

3．清空线性表的思想：将清空表的过程写成函数，其中将主函数中定义的 结构性变量L的引用作为函数参数。在函数中，由于清空操作并不释放内存空间，故只需将线性表的长度置为0即可。经分析，该算法的时间复杂度为O(1)。

4．求线性表表长的思想：将求表长过程写成函数，其中主函数中定义的结 构性变量L的引用作为函数的参数，在函数中，直接返回L.length即为所求线性表的表长。经分析，该算法的时间复杂度为O(1)。

5．获得元素的算法思想：将获得线性表元素写成函数，其中函数的参数是结构型变量L以及数据元素的序号i,由于采取的是线性存储结构，故直接通过访问数组的方式即L.elem[i-1]来获取元素，当然，在这之前需要判断合法性。经分析，该算法的时间复杂度为O(1)。

6. 查找元素的算法思想：将查找线性表特定值的数据元素写成函数，其中函数的参数是主函数中定义的结构类型变量L以及查找的数据元素的值，通过循环对线性表中的每一个元素与给定值比较是否相等，如果相等就返回该元素的位置。经分析，该算法的时间复杂度为O(n)。

7．获得直接前驱算法思想：将获得直接前驱元素的过程写成函数，函数的参数是结构体类型变量以及特定数据元素的值，接受前驱的变量。在函数中，首先判断线性表是否为空，若不为空则调用获得元素的函数判断该线性表中特定数据元素的位序，如果位置为1，则返回ERROR, 否则将其前一个元素即L.elem[i-2]赋给接受前驱的变量。经分析，该算法的时间复杂度为O(n)。

8. 获得后继算法思想：将获得直接后继元素的过程写成函数，函数的参数是结构体类型变量以及特定数据元素的值，接受后继的变量。在函数中，首先判断线性表是否为空，若不为空则调用获得元素的函数判断该线性表中特定数据元素的位序。如果位置为L.length，则返回ERROR, 否则将其后一个元素即L.elem[i]赋给接受后继的变量。经分析，该算法的时间复杂度为O(n)。

9．插入元素算法思想：将插入函数写成函数，函数的参数是结构型变量的引用，插入元素的值以及插入位置。在函数中，首先判断插入位置的合法性， 即是否在线性表中合适的位置，其次还要判断当前存储空间是否已满，如果存储空间已满则要用realloc函数重新分配空间，插入元素时要先从L.length开始将元素依次后移，然后将待插入元素插入到空位中。经分析，该算法的时间复杂度为O(n)。

10．删除元素算法思想：将删除线性表中元素写成函数，函数的参数是结构类型变量的引用，待删除元素的位置，接受待删除元素的变量。在函数中，首先判断位序的合法性，然后直接将删除元素位置后一个元素直到最后一个元素从前往后向前移动一个单元。经分析，该算法的时间复杂度为O(n)。

11．遍历线性表算法思想：将遍历线性表写成一个函数，函数的参数是结构类型变量，直接用一个循环来对线性表中的每一个元素进行操作（此处进行输出操作）。经分析，该算法的时间复杂度为 O(n)。

12．保存线性表到文件中的算法思想：函数的参数是结构体变量L和待保存文件的路径。在函数中，创建一个指向待保存文件的文件指针，通过循环依次将线性表中的数据写入文件中。经分析，该算法的算法复杂度是O(n)。

13．加载文件到线性表中的算法：函数的参数是结构体变量L的引用和待加载文件的路径。在函数中，创建一个指向待加载文件的文件指针，并要为线性表L分配空间，通过循环依次将文件中的数据读入线性表中，如果存储已满，还要增加分配空间。经分析，该算法的算法复杂度是O(n)。

14．插入一个线性表算法思想：函数的参数是管理表的引用和待插入线性表的名称。在函数中，先在已存在的管理表中的末尾处开辟新的线性表的存储空间，并将管理表长度增加1。经分析，该算法的时间复杂度是O(1)。

15.删除一个线性表算法思想：函数的参数是管理表的引用和待删除线性表的名称。在函数中，通过循环比较线性表名称和待删除线性表的名称，如果没有找到则返回ERROR，否则就先摧毁待删除线性表，然后自该位置起从前往后将下一个线性表复制到前一个线性表中。经分析，该算法复杂度是O（n2）。

16.定位一个线性表算法思想：函数参数是管理表Lists和待查找的线性表名称。在函数中，通过循环比较待查找线性表名称和管理表中线性表的名称，如果匹配成功则返回该线性表位序，否则返回ERROR。经分析，该算法复杂度是O(n)。

**1.4 系统测试**

说明：为达到测试目的，各个函数的测试是使用的educoder上的main函数分别在演示系统外测试的，其程序和源代码保存在各函数实现文件夹中，最后的全局测试即对功能演示系统的连续测试。

1. InitList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 1 | INFEASIBLE |  |
| 0 | OK |  |

2. DestroyList的测试（此处使用功能演示系统进行演示）

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 在已销毁或者未创建线性表的前提下进行销毁 | 线性表未创建，无需销毁 |  |
| 在已创建线性表的前提下进行销毁 | 线性表销毁成功 |  |

3. ClearList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 | OK |  |

4. ListEmpty的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 2 | TRUE |  |
| 1 | FALSE |  |
| 0 | INFEASIBLE |  |

5. ListLength的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 0 | 5 |  |
| 2  2 4 6 8 10 12 14 16 18 20 0 | 10 |  |

6. GetElem的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1  13 24 55 22 88 72 54 0  5 | OK  88 |  |
| 1  11 22 33 44 55 0  9 | ERROR |  |

7. LocateElem的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 6 0 4 | 4 |  |
| 1 12 34 56 78 90 0 5 | ERROR |  |

8. PriorElem的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 0 5 | OK 4 |  |
| 1 1 2 3 4 5 0 1 | ERROR |  |
| 1 1 2 3 4 5 0 6 | ERROR |  |

9. NextElem的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 0 5 | ERROR |  |
| 1 1 2 3 4 5 0 1 | OK 2 |  |
| 1 1 2 3 4 5 0 6 | ERROR |  |

10. ListInsert的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 0 6 7 | OK 1 2 3 4 5 6 |  |
| 1 1 2 3 4 5 0 0 6 | ERROR 1 2 3 4 5 |  |
| 1 1 2 3 4 5 0 7 6 | ERROR 1 2 3 4 5 |  |

11. ListDelete的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 0 3 | OK  3  1 2 4 5 |  |
| 1 1 2 3 4 5 0 0 | ERROR 1 2 3 4 5 |  |
| 1 1 2 3 4 5 0 7 | ERROR 1 2 3 4 5 |  |

12. ListTraverse的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 0 | 1 2 3 4 5 |  |

13. SaveList和LoadList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 0 | 5  1 2 3 4 5 |  |

14. AddList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 3  Xiaohong 1 2 3 0  Xiaogang 0  Xiaoming 0 | Xiaohong 1 2 3  Xiaogang  Xiaoming |  |
| 2  Henan 0  Beijing 1 2 3 0 | Henan  Beijing 1 2 3 |  |

15. RemoveList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 3  Xiaohong 1 2 3 0  Xiaogang 0  Xiaoming 0  Xiaowang | 删除失败 |  |
| 2  Henan 0  Beijing 1 2 3 0  Henan | Beijing 1 2 3 |  |

16. LocateList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 3  Xiaohong 1 2 3 0  Xiaogang 0  Xiaoming 0  Xiaogang | Xiaogang |  |
| 2  Henan 0  Beijing 1 2 3 0  Hubei | 查找失败 |  |

17.全局测试

|  |  |  |
| --- | --- | --- |
| 测试顺序 | 调用函数说明 | 实际输出 |
| 2 | DestroyList销毁线性表 |  |
| 3 | ClearList清空线性表 |  |
| 5 | ListLength求表长 |  |
| 1 | InitList创建线性表 |  |
| 1 | 再次创建会报错 |  |
| 4 | ListEmpty判空，还未插入元素线性表为空 |  |
| 5 | 此时表长为0 |  |
| 10 | ListInsert插入元素 |  |
| 10 |  |  |
| 10 |  |  |
| 12 | ListTraverse遍历线性表 |  |
| 4 | 此时不为空 |  |
| 5 | 此时长度为2 |  |
| 6 | GetElem获取某位置的元素 |  |
| 6 |  |  |
| 6 |  |  |
| 7 | LocateElem查找某元素 |  |
| 7 |  |  |
| 8 | PriorElem获取直接前驱 |  |
| 8 |  |  |
| 8 |  |  |
| 9 | NextElem获取直接后继 |  |
| 9 |  |  |
| 9 |  |  |
| 11 | ListDelete删除元素 |  |
| 11 |  |  |
| 11 |  |  |
| 10 | 在第一个元素后依次插入5 7 9 11 23等5个元素，过程截图略 | |
| 12 | 遍历当前线性表 |  |
| 13 | SaveList保存文件 |  |
| 2 | 销毁线性表 |  |
| 14 | 加载文件 |  |
| 12 | 遍历当前线性表 |  |
| 15 | AddList添加线性表 |  |
| 16 | RemoveList移除线性表 |  |
| 16 |  |  |
| 17 | LocateList查找线性表 |  |
| 17 |  |  |

## 1.5 实验小结

在这次实验中，我掌握了如何构建功能演示系统，学会了部分system指令的使用。

在实验中，我遇到了如下几个错误：

(1)在功能演示系统中，调用 插入元素函数时将参数写反，导致出错，由于自己在测试时使用的数据中位置和元素一样未发现错误，后来在助教检查时方测试出来。这也给我提了醒，以后在测试时要尽量使测试数据随机。

1. 在添加case语句时，由于忘记添加break而导致了两个函数的测试过程相连接，经过同学提醒才意识到这一点。
2. 在代码健壮性方面，未能及时添加文件是否打开成功，文件指针的释放等代码，尽管在测试中可能影响不大，但在实际应用中可能是隐患。同时，对于判断分配空间是否成功这一点也是在后来优化代码时添加的。
3. 在代码优化部分，在对多线性表的管理程序的测试时发现每次输入数据过程繁琐，因此添加了从文件读入的选择，大大节省了测试时间。在以后的实验中，也要注意使用文件输入来进行测试。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

该实验要解决的基本问题是实现单链表的各个基本功能，如初始化表、销毁表、清空表、判定空表、求表长和获得元素等等。

其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示。可选择以文件的形式进行存储和加载，也即将生成的单链表存入到相应的文件中，也可以从文件中获单链表进行操作。

**2.1.1需实现的基本运算**

依据最小完备性和常用性相结合的原则，以函数形式定义了单链表的初始化表、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)；初始条件是单链表L不存在；操作结果是构造一个空的单链表。

⑵销毁表：函数名称是DestroyList(L)；初始条件是单链表L已存在；操作结果是销毁单链表L。

⑶清空表：函数名称是ClearList(L)；初始条件是单链表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)；初始条件是单链表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)；初始条件是单链表已存在；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)；初始条件是单链表已存在，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())；初始条件是单链表已存在；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)；初始条件是单链表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)；初始条件是单链表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)；初始条件是单链表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)；初始条件是单链表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())，初始条件是单链表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()。

⒀保存单链表：函数名称是SaveList(L,FileName)，初始条件是单链表L已存在；操作结果是将单链表元素按顺序保存至文件名为FileName的文件中。

⒁加载单链表：函数名称是LoadList(L,FileName)，初始条件是单链表L不存在；操作结果是将文件名为FileName的文件加载到单链表中。

⒂在管理表中插入一个单链表：函数名称是AddList(Lists,ListName)，无初始条件；操作结果是将文件名为ListName的单链表添加到管理表中。

⒃在管理表中删除一个单链表：函数名称是RemoveList(Lists, ListName)，初始条件是多单链表非空；操作结果是将文件名为ListName的单链表删除。

⒄在管理表中查找一个单链表：函数名称是LocateList(Lists, ListName)，初始条件是多单链表非空；操作结果是查找文件名为ListName的单链表并可对其进行操作。

## 2.2 系统设计

本次实验使用的数据结构定义为：

/\*-------- Macro Definition ---------\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 10

#define LISTINCREMENT 10

//#define LL Lists.elem[num-1].L

/\*-------- Declare the type ---------\*/

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{ //单链表的管理表定义

struct { char name[30];

LinkList L;

}elem[10];

int length;

}LISTS;

将菜单演示和用户选择写入到while循环中，用OP获取用户的选择，OP初始化为1,以便第一次能进入循环。进入循环后系统首先显示功能菜单，然后用户输入选择0-14,其中1-17分别代表对单链表或单链表的管理表的一个基本运算，在主函数中通过switch语句对应到相应的函数功能，执行完该功能后break跳出switch语句，继续执行while循环，直至用户输入0退出当前演示系统。 (演示系统结构如图2-1所示)

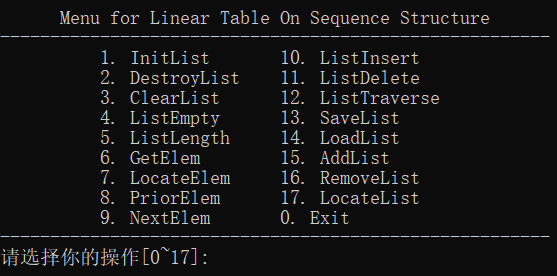


图2-1 演示系统结构图

## 2.3 系统实现

单链表运算算法思想与设计如下：

1．初始化单链表思想：将单链表初始化过程写成函数，其中传入函数的参数是主函数中定义的结构型变量L的引用。在函数中，首先判断L是否创建，如果未创建则使用malloc函数分配一个节点的内存空间，并把L的next指针域置为空。经分析，算法的时间复杂度为O(1)。

2．销毁单链表思想：将销毁单链表的过程写成函数，其中传入函数的参数是主函数中定义的结构性变量L的引用。在函数中，使用循环依次将每个节点的内存空间释放，最后将头指针置为空。经分析，该算法的时间复杂度为O(n)。

3．清空单链表的思想：将清空单链表的过程写成函数，其中将主函数中定义的结构性变量L的引用作为函数参数。在函数中，使用循环从头结点开始依次释放内存，最后将L的指针域置为空。经分析，该算法的时间复杂度为O(n)。

4．求单链表表长的思想：将求表长过程写成函数，其中主函数中定义的结 构性变量L的引用作为函数的参数，在函数中，通过循环和计数变量记录节点数，最后返回计数结果即为表长。经分析，该算法的时间复杂度为O(n)。

5．获得元素的算法思想：将获得单链表元素写成函数，其中函数的参数是结构型变量L以及数据元素的序号i,由于采取的是链式存储结构，故通过循环依次访问节点并比较数据域的方式来获取元素，当然，如果位置不合法应返回ERROR。经分析，该算法的时间复杂度为O(n)。

6. 查找元素的算法思想：将查找单链表特定值的数据元素写成函数，其中函数的参数是主函数中定义的结构类型变量L以及查找的数据元素的值，通过循环对单链表中的每一个节点的数据域与给定值比较是否相等，如果相等就返回该元素的位置。经分析，该算法的时间复杂度为O(n)。

7．获得直接前驱算法思想：将获得直接前驱元素的过程写成函数，函数的参数是结构体类型变量以及特定数据元素的值，接受前驱的变量。在函数中，首先判断单链表是否为空，若不为空则通过循环遍历元素比较，如果头结点元素即为所找元素则表明无直接前驱，如果头结点不是则用另一个指针指向当前遍历结点的前一个节点，当找到元素时将前一个节点的值赋给pre，若找不到则返回ERROR。经分析，该算法的时间复杂度为O(n)。

8. 获得直接后继算法思想：将获得直接后继元素的过程写成函数，函数的参数是结构体类型变量以及特定数据元素的值，接受后继的变量。在函数中，首先判断单链表是否为空，若不为空则通过循环遍历元素比较，当找到元素时将后一个节点的值赋给next，若该元素对应最后一个节点则表明无直接后继，若找不到元素则返回ERROR。经分析，该算法的时间复杂度为O(n)。

9．插入元素算法思想：将插入元素写成函数，函数的参数是结构型变量的引用，插入元素的值以及插入位置。在函数中，首先判断插入位置的合法性， 即是否在单链表中合适的位置，插入元素时如果单链表节点数为0，则直接令L指向新创建的待插入节点，若不是则先定位至插入位置处更改节点的指针域来实现插入。经分析，该算法的时间复杂度为O(n)。

10．删除元素算法思想：将删除单链表中元素写成函数，函数的参数是结构类型变量的引用，待删除元素的位置，接受待删除元素的变量。在函数中，先定位至待删除元素的位置，将其赋值给接受变量，然后通过改变指针域移除该节点，最后释放该节点的内存空间。经分析，该算法的时间复杂度为O(n)。

11．遍历单链表算法思想：将遍历单链表写成一个函数，函数的参数是结构类型变量，直接用一个循环来对单链表中的每一个元素进行操作（此处进行输出操作）。经分析，该算法的时间复杂度为 O(n)。

12．保存单链表到文件中的算法思想：函数的参数是结构体变量L和待保存文件的路径或名称。在函数中，创建一个指向待保存文件的文件指针，通过循环依次将单链表中的数据写入文件中。经分析，该算法的算法复杂度是O(n)。

13．加载文件到单链表中的算法：函数的参数是结构体变量L的引用和待加载文件的路径。在函数中，创建一个指向待加载文件的文件指针，并根据读写逐个分配单链表L中结点的内存空间，最后将尾结点的指针域置为空并释放文件指针。经分析，该算法的算法复杂度是O(n)。

14．插入一个单链表算法思想：函数的参数是管理表的引用和待插入单链表的名称。在函数中，先在已存在的管理表中的末尾处开辟新的单链表的存储空间，并将管理表长度增加1.经分析，该算法的时间复杂度是O(1)。

15.删除一个单链表算法思想：函数的参数是管理表的引用和待删除单链表的名称。在函数中，通过循环比较单链表名称和待删除单链表的名称，如果没有找到则返回ERROR，否则就先摧毁待删除单链表，然后自该位置起从前往后将下一个单链表复制到前一个单链表中。经分析，该算法复杂度是O（n2）。

16.定位一个单链表算法思想：函数参数是管理表Lists和待查找的单链表名称。在函数中，通过循环比较待查找单链表名称和管理表中单链表的名称，如果匹配成功则返回该单链表位序，否则返回ERROR。经分析，该算法复杂度是O(n)。

## 2.4 系统测试

说明：为达到测试目的，各个函数的测试是使用的educoder上的main函数分别在演示系统外测试的，其程序和源代码保存在各函数实现文件夹中，最后的全局测试即对功能演示系统的连续测试。

1. InitList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 1 | INFEASIBLE |  |
| 0 | OK |  |

2. DestroyList的测试（此处使用功能演示系统进行演示）

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 在已销毁或者未创建单链表的前提下进行销毁 | 单链表未创建，无需销毁 |  |
| 在已创建单链表的前提下进行销毁 | 单链表销毁成功 |  |

3. ClearList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 | OK |  |

4. ListEmpty的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 2 | TRUE |  |
| 1 | FALSE |  |
| 0 | INFEASIBLE |  |

5. ListLength的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 2  3 5 7 11 13 17 19 23 0 | 8 |  |

6. GetElem的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1  3 5 7 11 13 17 19 23 0  5 | OK  13 |  |
| 1  3 5 7 11 17 19 23 0  9 | ERROR |  |

7. LocateElem的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 2 5 8 3 6 9 0 3 | 4 |  |
| 1 3 5 15 34 35 0 51 | ERROR |  |

8. PriorElem的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 3 5 15 34 35 0 34 | OK 15 |  |
| 1 3 5 15 34 35 0 3 | ERROR |  |
| 1 3 5 15 34 35 0 31 | ERROR |  |

9. NextElem的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 3 5 15 34 35 0 35 | ERROR |  |
| 1 3 5 15 34 35 0 34 | OK 35 |  |
| 1 3 5 15 34 35 0 31 | ERROR |  |

10. ListInsert的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 0 6 7 | OK 1 2 3 4 5 6 |  |
| 1 1 2 3 4 5 0 0 6 | ERROR 1 2 3 4 5 |  |
| 1 1 2 3 4 5 0 7 6 | ERROR 1 2 3 4 5 |  |

11. ListDelete的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 0 3 | OK  3  1 2 4 5 |  |
| 1 1 2 3 4 5 0 0 | ERROR 1 2 3 4 5 |  |
| 1 1 2 3 4 5 0 7 | ERROR 1 2 3 4 5 |  |

12. ListTraverse的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 0 | 1 2 3 4 5 |  |

13. SaveList和LoadList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 0 | INFEASIBLE |  |
| 1 1 2 3 4 5 0 | 5  1 2 3 4 5 |  |

14. AddList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 3  Lihong 1 2 3 0  Ligang 0  Liming 0 | Lihong 1 2 3  Ligang  Liming |  |
| 2  WHU 0  HUST 100 200 300 0 | WHU  HUST 100 200 300 |  |

15. RemoveList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 3  Lihong 1 2 3 0  Ligang 0  Liming 0  Lixiao | 删除失败 |  |
| 2  WHU 0  HUST 100 200 300 0  WHU | HUST 100 200 300 |  |

16. LocateList的测试

|  |  |  |
| --- | --- | --- |
| 测试用例 | 预期输出 | 实际输出 |
| 3  Lihong 1 2 3 0  Ligang 0  Liming 0  Ligang | Ligang |  |
| 2  WHU 0  HUST 100 200 300 0  Peking | 查找失败 |  |

17.全局测试

|  |  |  |
| --- | --- | --- |
| 测试顺序 | 调用函数说明 | 实际输出 |
| 2 | DestroyList销毁单链表 |  |
| 3 | ClearList清空单链表 |  |
| 5 | ListLength求表长 |  |
| 1 | InitList创建单链表 |  |
| 1 | 再次创建会报错 |  |
| 4 | ListEmpty判空，还未插入元素单链表为空 |  |
| 5 | 此时表长为0 |  |
| 10 | ListInsert插入元素 |  |
| 10 |  |  |
| 10 |  |  |
| 10 |  |  |
| 12 | ListTraverse遍历单链表 |  |
| 4 | 此时不为空 |  |
| 5 | 此时长度为3 |  |
| 6 | GetElem获取某位置的元素 |  |
| 6 |  |  |
| 6 |  |  |
| 7 | LocateElem查找某元素 |  |
| 7 |  |  |
| 8 | PriorElem获取直接前驱 |  |
| 8 |  |  |
| 8 |  |  |
| 9 | NextElem获取直接后继 |  |
| 9 |  |  |
| 9 |  |  |
| 11 | ListDelete删除元素 |  |
| 11 |  |  |
| 11 |  |  |
| 10 | 依次调用插入函数使单链表节点的数据域依次为3 5 7 11 13 17 19 23等8个元素，过程截图略 | |
| 12 | 遍历当前单链表 |  |
| 13 | SaveList保存文件 |  |
| 2 | 销毁线性表 |  |
| 14 | 加载文件 |  |
| 12 | 遍历当前线性表 |  |
| 15 | AddList添加线性表 |  |
| 16 | RemoveList移除线性表 |  |
| 16 |  |  |
| 17 | LocateList查找线性表 |  |
| 17 |  |  |

## 2.5 实验小结

在实验中，我遇到了如下几个错误：

（1）在写各个函数的代码中，总是因为混淆L和L->next而导致程序总是出现千奇百怪的输出结果，后来重新看了一遍关于单链表的节点的有关知识才避免了各种错误，这告诉我，在处理一种数据结构之前先要了解其逻辑结构。

（2）在编写程序时，我有几次都忘记了要释放链表中不需要的节点或者文件指针。这与上次实验中我犯的错误有相似之处，我会吸取教训，不再给代码埋下这种隐患。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

## 3.2 系统设计

## 3.3 系统实现

## 3.4 系统测试

## 3.5 实验小结

# 4 基于邻接表的图实现

## 4.1 问题描述

## 4.2 系统设计

## 4.3 系统实现

## 4.4 系统测试

## 4.5 实验小结

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

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

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

/\*-------- Macro Definition ---------\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 10

#define LISTINCREMENT 10

/\*-------- Declare the type ---------\*/

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

typedef struct{ //线性表的管理表定义

struct{

char name[30];

SqList L;

}elem[10];

int length;

int listsize;

}LISTS;

SqList L;

/\*-----Function Declaration---------\*/

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 ListTraverse(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[]);

/\*----- The main function --------\*/

int main(void){

L.elem=NULL;

LISTS Lists;

int op=1,res=0;

while(op){

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

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

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

printf(" 1. InitList 10. ListInsert\n");

printf(" 2. DestroyList 11. ListDelete\n");

printf(" 3. ClearList 12. ListTraverse\n");

printf(" 4. ListEmpty 13. SaveList\n");

printf(" 5. ListLength 14. LoadList\n");

printf(" 6. GetElem 15. AddList\n");

printf(" 7. LocateElem 16. RemoveList\n");

printf(" 8. PriorElem 17. LocateList\n");

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

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

printf("请选择你的操作[0~17]:");

scanf("%d",&op);

switch(op){

case 1:

res=InitList(L);

if(res==OK) printf("线性表创建成功");

else if(res==INFEASIBLE) printf("线性表已创建");

else printf("线性表创建失败");

getchar();getchar();

break;

case 2:

res=DestroyList(L);

if(res==OK)

{

if(L.elem) printf("线性表销毁失败");

else printf("线性表销毁成功");

}

else if(res==INFEASIBLE) printf("线性表未创建，无需销毁");

else printf("线性表销毁失败");

getchar();getchar();

break;

case 3:

res=ClearList(L);

if (L.length||(!L.elem)) printf("线性表清除失败");

if (res==OK) printf("线性表清除成功");

if(res==INFEASIBLE) printf("线性表未创建");

getchar();getchar();

break;

case 4:

res=ListEmpty(L);

if (res==TRUE) printf("线性表为空表");

else if(res==FALSE)printf("线性表不为空表");

else printf("线性表未创建");

getchar();getchar();

break;

case 5:

res=ListLength(L);

if(res==L.length) printf("线性表的长度为%d",ListLength(L));

else printf("线性表未创建");

getchar();getchar();

break;

case 6:

int pos,e;

printf("请输入元素位置:");

scanf("%d",&pos);

res=GetElem(L,pos,e);

if(res==OK)printf("线性表中第%d个元素是%d",pos,e);

else if(res==ERROR) printf("输入位置不合法");

else printf("线性表未创建");

getchar();getchar();

break;

case 7:

int data;

printf("请输入元素:");

scanf("%d",&data);

res=LocateElem(L,data);

if(res!=ERROR) printf("线性表中元素%d的位置是%d",data,LocateElem(L,data));

else if(res==ERROR)printf("该元素不在线性表中");

else printf("线性表未创建");

getchar();getchar();

break;

case 8:

int data1,pre;

printf("请输入元素:");

scanf("%d",&data1);

res=PriorElem(L,data1,pre);

if(res==INFEASIBLE)printf("该元素无直接前驱");

else if(res!=ERROR)printf("线性表中元素%d的直接前驱元素是%d",data1,pre);

else printf("该元素不在线性表中");

getchar();getchar();

break;

case 9:

int data2,next;

printf("请输入元素:");

scanf("%d",&data2);

res=NextElem(L,data2,next);

if(res==INFEASIBLE) printf("该元素无直接后继");

else if(res!=ERROR)printf("线性表中元素%d的直接后继元素是%d\n",data2,next);

else printf("该元素不在线性表中");

getchar();getchar();

break;

case 10:

// for(int i=1;i<=30;i++) ListInsert(L,i,i);//此行代码用于快速添加元素以便进行测试

int data3,pos1;

printf("请输入插入的元素及插入的位置:");

scanf("%d%d",&data3,&pos1);

res=ListInsert(L,pos1,data3);

if(res&&res!=INFEASIBLE)printf("插入元素成功，当前线性表的长度为%d",ListLength(L));

else if(res==ERROR) printf("元素位置不合法");

else printf("线性表未创建");

getchar();getchar();

break;

case 11:

int data4,pos2;

printf("请输入删除的元素位置:");

scanf("%d",&pos2);

res=ListDelete(L,pos2,data4);

if(res&&res!=INFEASIBLE)printf("删除元素成功，被删除元素为%d，当前线性表的长度为%d",data4,ListLength(L));

else if(res==ERROR) printf("元素位置不合法");

else printf("线性表未创建");

getchar();getchar();

break;

case 12:

res=ListTraverse(L);

if(!res) printf("线性表是空表");

else if(res==INFEASIBLE) printf("线性表未创建");

getchar();getchar();

break;

case 13:

char FileName[30];

printf("请输入文件保存路径或文件名:");

scanf("%s",FileName);

res=SaveList(L,FileName);

if(res) printf("文件成功保存至%s\n",FileName);

getchar();getchar();

break;

case 14:

char FileName1[30];

printf("请输入待加载文件路径或文件名:");

scanf("%s",FileName1);

res=LoadList(L,FileName1);

if(res) printf("文件成功加载至当前线性表\n");

else printf("当前线性表未销毁，不能进行加载操作");

getchar();getchar();

break;

case 15:

int n,e1,f;

char name[30];

Lists.length=0;

printf("请选择从文件加载(0)或键盘输入(1)：");

scanf("%d",&f);

if(!f)

{

char FileName2[50];

printf("请输入待加载文件路径或文件名:");

scanf("%s",FileName2);

FILE \*fp=fopen(FileName2,"r");

if(fp==NULL) printf("未成功打开文件");

while(!feof(fp))

{

fscanf(fp,"%d",&n);

while(n--)

{

fscanf(fp,"%s",name);

AddList(Lists,name);

fscanf(fp,"%d",&e1);

while (e1)

{

ListInsert(Lists.elem[Lists.length-1].L,Lists.elem[Lists.length-1].L.length+1,e1);

fscanf(fp,"%d",&e1);

}

}

}

fclose(fp);

printf("插入线性表完毕\n当前管理表为:\n");

}

else

{

printf("请输入要添加的线性表个数:");

scanf("%d",&n);

printf("请依次输入线性表名称和线性表元素:");

while(n--)

{

scanf("%s",name);

AddList(Lists,name);

scanf("%d",&e1);

while (e1)

{

ListInsert(Lists.elem[Lists.length-1].L,Lists.elem[Lists.length-1].L.length+1,e1);

scanf("%d",&e1);

}

}

printf("插入线性表完毕\n当前管理表为:\n");

}

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

{

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

ListTraverse(Lists.elem[n].L);

putchar('\n');

}

getchar();getchar();

break;

case 16:

char check[30];

printf("请输入要删除的线性表名称:");

scanf("%s",check);

if (RemoveList(Lists,check)==OK)

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

{

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

ListTraverse(Lists.elem[n].L);

putchar('\n');

}

else printf("删除线性表失败");

getchar();getchar();

break;

case 17:

char check1[30];

printf("请输入要查找的线性表名称:");

scanf("%s",check1);

if(n=LocateList(Lists,check1))

{

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

ListTraverse(Lists.elem[n-1].L);

putchar('\n');

}

else printf("该线性表不在此管理表中");

getchar();getchar();

break;

case 0:

break;

// printf("请选择你的操作[0~17]:");

}//end of switch

}//end of while

printf("\nWelcome to use this system again next time!\n");

return 0;

//system("pause");

}//end of main()

/\*----- Implementation of functions --------\*/

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

函数功能:初始化线性表

函数参数:引用线性表L的地址

函数返回值:返回int类型，创建成功返回1

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

status InitList(SqList &L)

{

if(L.elem==NULL)

{

ElemType \*newbase=(ElemType \*)malloc(sizeof(ElemType)\*LIST\_INIT\_SIZE);

if(!newbase) exit(OVERFLOW);

L.elem=newbase;

L.length=0;

L.listsize=LIST\_INIT\_SIZE;

return OK;

}

else return INFEASIBLE;

}

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

函数功能:销毁线性表

函数参数:引用线性表L的地址

函数返回值:返回int类型，销毁成功返回1

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

status DestroyList(SqList &L)

{

if(L.elem==NULL) return INFEASIBLE;

else

{

free(L.elem);

L.elem=NULL;

L.length=0;

L.listsize=0;

return OK;

}

}

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

函数功能:清空线性表

函数参数:引用线性表L的地址

函数返回值:返回int类型，清空成功返回1

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

status ClearList(SqList &L)

{

if(L.elem!=NULL)

{

L.length=0;

return OK;

}

else return INFEASIBLE;

}

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

函数功能:线性表判空

函数参数:线性表L

函数返回值:返回int类型，线性表为空返回1

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

status ListEmpty(SqList L)

{

if(L.elem!=NULL)

{

if(L.length==0) return TRUE;

else return FALSE;

}

else return INFEASIBLE;

}

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

函数功能:求线性表表长

函数参数:线性表L

函数返回值:返回int类型的线性表长度

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

status ListLength(SqList L)

{

if(L.elem!=NULL) return L.length;

else return INFEASIBLE;

}

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

函数功能:得到线性表中的元素并将该元素赋值给e

函数参数:线性表L，元素的位置i，引用元素e的地址

函数返回值:返回int类型，获得成功返回1

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

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

{

if(L.elem==NULL) return INFEASIBLE;

else if(i<1||i>L.length) return ERROR;

else

{

e=L.elem[i-1];

return OK;

}

}

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

函数功能:查找元素

函数参数:线性表L，待查找元素e

函数返回值:返回int类型，查找成功返回元素位置，否则返回ERROR

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

status LocateElem(SqList L,ElemType e)

{

if(L.elem==NULL) return INFEASIBLE;

else

{

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

if(L.elem[i-1]==e) return i;

return ERROR;

}

}

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

函数功能:得到元素的直接前驱并赋值给pre

函数参数:线性表L，元素e，引用待返回元素的地址

函数返回值:返回int类型，得到前驱成功返回1

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

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

{

if(L.elem==NULL) return INFEASIBLE;

else

{

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

{//此处也可以直接调用LocateElem函数得到元素e的位置

if(L.elem[i-1]==e)

{

if(i==1) return INFEASIBLE;

else {pre=L.elem[i-2];return OK;}

}

}

return ERROR;

}

}

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

函数功能:得到元素的直接后继并赋值给next

函数参数:线性表L，元素e，引用待返回元素的地址

函数返回值:返回int类型，得到后继成功返回1

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

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

{

if(L.elem==NULL) return INFEASIBLE;

else

{

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

{

if(L.elem[i-1]==e)

{

if(i==L.length) return INFEASIBLE;

else {next=L.elem[i];return OK;}

}

}

return ERROR;

}

}

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

函数功能:插入元素在第i个元素前

函数参数:引用线性表L的地址，待插入元素元素e，待插入位置i

函数返回值:返回int类型，插入成功返回1

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

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

{

if(L.elem==NULL) return INFEASIBLE;

else

{

if(i>L.length+1||i<=0) return ERROR;

if(L.length==L.listsize)

{

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

if(!newbase) exit(OVERFLOW);

L.elem=newbase;

L.listsize+=LISTINCREMENT;

}

for(int j=L.length;j>=i;j--) L.elem[j]=L.elem[j-1];

L.elem[i-1]=e;

L.length++;

return OK;

}

}

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

函数功能:删除第i个元素并得到被删除元素的值

函数参数:引用线性表L的地址，引用返回元素e的地址，待删除位置i

函数返回值:返回int类型，插入成功返回1

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

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

{

if(L.elem==NULL) return INFEASIBLE;

else

{

if(i>=L.length+1||i<=0) return ERROR;

e=L.elem[i-1];

for(int j=i;j<=L.length;j++) L.elem[j-1]=L.elem[j];

L.length--;

return OK;

}

}

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

函数功能:遍历输出线性表中的元素

函数参数:线性表L的

函数返回值:返回int类型，遍历成功返回1

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

status ListTraverse(SqList L)

{

if(L.elem==NULL) return INFEASIBLE;

else

{

for(int i=1;i<=L.length;i++) printf("%d ",L.elem[i-1]);

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

}

return OK;

}

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

函数功能:保存线性表到指定文件中

函数参数:线性表L，待保存文件的路径FileName

函数返回值:返回int类型，保存成功返回1

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

status SaveList(SqList L,char FileName[])

{

if(L.elem==NULL) return INFEASIBLE;

else

{

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

if(fout==NULL) {printf("未成功打开文件");return ERROR;}

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

fprintf(fout,"%d ",L.elem[i-1]);

fclose(fout);

return OK;

}

}

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

函数功能:加载文件到线性表中

函数参数:线性表L，待加载文件的路径FileName

函数返回值:返回int类型，加载成功返回1

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

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

{

if(L.elem!=NULL) return INFEASIBLE;

else

{

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

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

if(fp==NULL) {printf("未成功打开文件");return ERROR;}

int i=1,flag=0;

L.length=0;

L.listsize=LIST\_INIT\_SIZE;

while(!feof(fp))

{

if(L.length==L.listsize)

{

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

if(!newbase) exit(OVERFLOW);

L.elem=newbase;

L.listsize+=LISTINCREMENT;

}

fscanf(fp,"%d",&L.elem[i-1]);

i++;

L.length++;

}

L.length--;

fclose(fp);

return OK;

}

}

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

函数功能:在线性表的管理表中插入一个新的线性表

函数参数:引用线性表的管理表Lists的地址，待插入线性表的名称FileName

函数返回值:返回int类型，插入成功返回1

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

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

{

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

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

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

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

Lists.length++;

return OK;

}

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

函数功能:在线性表的管理表中删除一个线性表

函数参数:引用线性表的管理表Lists的地址，待删除线性表的名称FileName

函数返回值:返回int类型，删除成功返回1

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

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

{

for(int j=1;j<=Lists.length;j++)

{

if(strcmp(ListName,Lists.elem[j-1].name)==0)

{

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

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

{

strcpy(Lists.elem[i-1].name,Lists.elem[i].name);

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

Lists.elem[i-1].L.length=Lists.elem[i].L.length;

Lists.elem[i-1].L.listsize=Lists.elem[i].L.listsize;

}

Lists.length--;

return OK;

}

}

return ERROR;

}

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

函数功能:在线性表的管理表中查找线性表

函数参数:线性表的管理表Lists，待查找线性表的名称FileName

函数返回值:返回int类型，查找成功返回该线性表的位置

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

int LocateList(LISTS Lists,char ListName[])

{

for(int i=1;i<=Lists.length;i++)

if(strcmp(ListName,Lists.elem[i-1].name)==0)

{

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

return i;

}

return ERROR;

}

# 附录B 基于链式存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

/\*-------- Macro Definition ---------\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 10

#define LISTINCREMENT 10

//#define LL Lists.elem[num-1].L

/\*-------- Declare the type ---------\*/

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{ //单链表的管理表定义

struct { char name[30];

LinkList L;

}elem[10];

int length;

}LISTS;

LinkList L;

/\*-----Function Declaration---------\*/

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 ListTraverse(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[]);

/\*----- The main function --------\*/

int main(void){

L=NULL;

LISTS Lists;

int op=1,res=0;

while(op){

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

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

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

printf(" 1. InitList 10. ListInsert\n");

printf(" 2. DestroyList 11. ListDelete\n");

printf(" 3. ClearList 12. ListTraverse\n");

printf(" 4. ListEmpty 13. SaveList\n");

printf(" 5. ListLength 14. LoadList\n");

printf(" 6. GetElem 15. AddList\n");

printf(" 7. LocateElem 16. RemoveList\n");

printf(" 8. PriorElem 17. LocateList\n");

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

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

printf("请选择你的操作[0~17]:");

scanf("%d",&op);

switch(op){

case 1:

res=InitList(L);

if(res==OK) printf("单链表创建成功");

else if(res==INFEASIBLE) printf("单链表已创建");

else printf("单链表创建失败");

getchar();getchar();

break;

case 2:

res=DestroyList(L);

if(res==OK)

{

if(L) printf("单链表销毁失败");

else printf("单链表销毁成功");

}

else if(res==INFEASIBLE) printf("单链表未创建，无需销毁");

else printf("单链表销毁失败");

getchar();getchar();

break;

case 3:

res=ClearList(L);

if(res==INFEASIBLE) printf("单链表未创建,");

if (!L) printf("单链表清除失败");

if (res==OK) printf("单链表清除成功");

getchar();getchar();

break;

case 4:

res=ListEmpty(L);

if (res==TRUE) printf("单链表为空表");

else if(res==FALSE)printf("单链表不为空表");

else printf("单链表未创建");

getchar();getchar();

break;

case 5:

res=ListLength(L);

if(res!=-1) printf("单链表的长度为%d",ListLength(L));

else printf("单链表未创建");

getchar();getchar();

break;

case 6:

int pos,e;

printf("请输入元素位置:");

scanf("%d",&pos);

res=GetElem(L,pos,e);

if(res==OK)printf("单链表中第%d个元素是%d",pos,e);

else if(res==ERROR) printf("输入位置不合法");

else printf("单链表未创建");

getchar();getchar();

break;

case 7:

int data;

printf("请输入元素:");

scanf("%d",&data);

res=LocateElem(L,data);

if(res!=ERROR) printf("单链表中元素%d的位置是%d",data,LocateElem(L,data));

else if(res==ERROR)printf("该元素不在单链表中");

else printf("单链表未创建");

getchar();getchar();

break;

case 8:

int data1,pre;

printf("请输入元素:");

scanf("%d",&data1);

res=PriorElem(L,data1,pre);

if(res==INFEASIBLE)printf("该元素无直接前驱");

else if(res!=ERROR)printf("单链表中元素%d的直接前驱元素是%d",data1,pre);

else printf("该元素不在单链表中");

getchar();getchar();

break;

case 9:

int data2,next;

printf("请输入元素:");

scanf("%d",&data2);

res=NextElem(L,data2,next);

if(res==INFEASIBLE) printf("该元素无直接后继");

else if(res!=ERROR)printf("单链表中元素%d的直接后继元素是%d\n",data2,next);

else printf("该元素不在单链表中");

getchar();getchar();

break;

case 10:

// for(int i=1;i<=30;i++) ListInsert(L,i,i);//此行代码用于快速添加元素以便进行测试

int data3,pos1;

printf("请输入插入的元素及插入的位置:");

scanf("%d%d",&data3,&pos1);

res=ListInsert(L,pos1,data3);

if(res&&res!=INFEASIBLE)printf("插入元素成功，当前单链表的长度为%d",ListLength(L));

else if(res==ERROR) printf("元素位置不合法");

else printf("单链表未创建");

getchar();getchar();

break;

case 11:

int data4,pos2;

printf("请输入删除的元素位置:");

scanf("%d",&pos2);

res=ListDelete(L,pos2,data4);

if(res&&res!=INFEASIBLE)printf("删除元素成功，被删除元素为%d，当前单链表的长度为%d",data4,ListLength(L));

else if(res==ERROR) printf("元素位置不合法");

else printf("单链表未创建");

getchar();getchar();

break;

case 12:

res=ListTraverse(L);

if(!res) printf("单链表是空表");

else if(res==INFEASIBLE) printf("单链表未创建");

getchar();getchar();

break;

case 13:

char FileName[30];

printf("请输入文件保存路径或文件名:");

scanf("%s",FileName);

res=SaveList(L,FileName);

if(res) printf("文件成功保存至%s\n",FileName);

getchar();getchar();

break;

case 14:

char FileName1[30];

printf("请输入待加载文件路径或文件名:");

scanf("%s",FileName1);

res=LoadList(L,FileName1);

if(res) printf("文件成功加载至当前单链表\n");

else printf("当前单链表未销毁，不能进行加载操作");

getchar();getchar();

break;

case 15:

int n,e1,f;

char name[30];

Lists.length=0;

printf("请选择从文件加载(0)或键盘输入(1)：");

scanf("%d",&f);

if(!f)

{

char FileName2[50];

printf("请输入待加载文件路径或文件名:");

scanf("%s",FileName2);

FILE \*fp=fopen(FileName2,"r");

if(fp==NULL) printf("未成功打开文件");

while(!feof(fp))

{

fscanf(fp,"%d",&n);

while(n--)

{

fscanf(fp,"%s",name);

AddList(Lists,name);

fscanf(fp,"%d",&e1);

while (e1)

{

ListInsert(Lists.elem[Lists.length-1].L,ListLength(Lists.elem[Lists.length-1].L)+1,e1);

fscanf(fp,"%d",&e1);

}

}

}

fclose(fp);

printf("插入单链表完毕\n当前管理表为:\n");

}

else

{

printf("请输入要添加的单链表个数:");

scanf("%d",&n);

printf("请依次输入单链表名称和单链表元素:");

while(n--)

{

scanf("%s",name);

AddList(Lists,name);

scanf("%d",&e1);

while(e1)

{

ListInsert(Lists.elem[Lists.length-1].L,ListLength(Lists.elem[Lists.length-1].L)+1,e1);

scanf("%d",&e1);

}

}

printf("插入单链表完毕\n当前管理表为:\n");

}

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

{

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

ListTraverse(Lists.elem[n].L);

putchar('\n');

}

getchar();getchar();

break;

case 16:

char check[30];

printf("请输入要删除的单链表名称:");

scanf("%s",check);

if (RemoveList(Lists,check)==OK)

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

{

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

ListTraverse(Lists.elem[n].L);

putchar('\n');

}

else printf("删除单链表失败");

getchar();getchar();

break;

case 17:

char check1[30];

printf("请输入要查找的单链表名称:");

scanf("%s",check1);

if(n=LocateList(Lists,check1))

{

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

ListTraverse(Lists.elem[n-1].L);

putchar('\n');

}

else printf("该单链表不在此管理表中");

getchar();getchar();

break;

case 0:

break;

// printf("请选择你的操作[0~17]:");

}//end of switch

}//end of while

printf("\nWelcome to use this system again next time!\n");

return 0;

//system("pause");

}//end of main()

/\*----- Implementation of functions --------\*/

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

函数功能:初始化单链表

函数参数:引用单链表L的地址

函数返回值:返回int类型，创建成功返回1

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

status InitList(LinkList &L)

{

if(L==NULL)

{

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

L->next=NULL;

return OK;

}

else return INFEASIBLE;

}

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

函数功能:销毁单链表

函数参数:引用单链表L的地址

函数返回值:返回int类型，销毁成功返回1

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

status DestroyList(LinkList &L)

{

if(L==NULL) return INFEASIBLE;

else

{

LinkList q=L->next;

while(q!=NULL){

free(L);

L=q;

q=L->next;

}

}

L=NULL;

return OK;

}

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

函数功能:清空单链表

函数参数:引用单链表L的地址

函数返回值:返回int类型，清空成功返回1

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

status ClearList(LinkList &L)

{

if(L!=NULL)

{

LinkList pre,p;

p = L->next;

while(p!=NULL) {

pre=p;

p=p->next;

free(pre);

}

L->next=NULL;

return OK;

}

else return INFEASIBLE;

}

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

函数功能:单链表判空

函数参数:单链表L

函数返回值:返回int类型，单链表为空返回1

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

status ListEmpty(LinkList L)

{

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

else if(L==NULL)return INFEASIBLE;

else return FALSE;

}

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

函数功能:求单链表表长

函数参数:单链表L

函数返回值:返回int类型的单链表长度

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

status ListLength(LinkList L)

{

if(L==NULL)return INFEASIBLE;

else

{

int i=0;

LinkList p=L->next;

while(p)

{

i++;

p=p->next;

}

return i;

}

}

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

函数功能:得到单链表中的元素并将该元素赋值给e

函数参数:单链表L，元素的位置i，引用元素e的地址

函数返回值:返回int类型，获得成功返回1

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

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

{

if(L==NULL)return INFEASIBLE;

else

{

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

LinkList p=L->next;

int j=1;

while(p&&j<i)

{

++j;

p=p->next;

}

if(p==NULL||j>i) return ERROR;

e=p->data;

return OK;

}

}

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

函数功能:查找元素

函数参数:单链表L，待查找元素e

函数返回值:返回int类型，查找成功返回元素位置，否则返回ERROR

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

status LocateElem(LinkList L,ElemType e)

{

if(L==NULL)return INFEASIBLE;

else

{

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

LinkList p=L->next;

int j=1;

while(p)

{

if(p->data==e) return j;

p=p->next;

j++;

}

return ERROR;

}

}

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

函数功能:得到元素的直接前驱并赋值给pre

函数参数:单链表L，元素e，引用待返回元素的地址

函数返回值:返回int类型，得到前驱成功返回1

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

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

{

if(L==NULL)return INFEASIBLE;

else

{

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

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

if(p->data==e)return INFEASIBLE;

while(p)

{

if(p->data==e)

{

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

pre=q->data;

return OK;

}

q=p;

p=p->next;

}

return ERROR;

}

}

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

函数功能:得到元素的直接后继并赋值给next

函数参数:单链表L，元素e，引用待返回元素的地址

函数返回值:返回int类型，得到后继成功返回1

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

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

{

if(L==NULL)return INFEASIBLE;

else

{

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

LinkList p=L->next;

while(p)

{

if(p->data==e)

{

if(p->next==NULL)return INFEASIBLE;

next=p->next->data;

return OK;

}

p=p->next;

}

return ERROR;

}

}

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

函数功能:插入元素在第i个元素前

函数参数:引用单链表L的地址，待插入元素元素e，待插入位置i

函数返回值:返回int类型，插入成功返回1

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

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

{

if(L==NULL)return INFEASIBLE;

else

{

if(i<=0)return ERROR;

LinkList p=L;

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

q->data=e;

int j=1;

if(p->next==NULL)

{

L->next=q;

q->next=NULL;

return OK;

}

while(p&&j<i)

{

p=p->next;

j++;

}

if(p==NULL)return ERROR;

q->next=p->next;

p->next=q;

return OK;

}

}

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

函数功能:删除第i个元素并得到被删除元素的值

函数参数:引用单链表L的地址，引用返回元素e的地址，待删除位置i

函数返回值:返回int类型，插入成功返回1

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

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

{

if(L==NULL)return INFEASIBLE;

else

{

if(i<=0||L->next==NULL)return ERROR;

LinkList p=L,q;

int j=1;

while(p&&j<i)

{

p=p->next;

if(p==NULL)return ERROR;

j++;

}

if(p->next==NULL)return ERROR;

q=p->next;

e=q->data;

p->next=q->next;

free(q);

return OK;

}

}

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

函数功能:遍历输出单链表中的元素

函数参数:单链表L的

函数返回值:返回int类型，遍历成功返回1

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

status ListTraverse(LinkList L)

{

if(L==NULL)return INFEASIBLE;

else

{

LinkList p=L->next;

while(p)

{

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

if(p->next!=NULL)printf(" ");

p=p->next;

}

return OK;

}

}

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

函数功能:保存单链表到指定文件中

函数参数:单链表L，待保存文件的路径FileName

函数返回值:返回int类型，保存成功返回1

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

status SaveList(LinkList L,char FileName[])

{

if(L==NULL)return INFEASIBLE;

else

{

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

LinkList q=L->next;

while(q)

{

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

if(q->next!=NULL)fprintf(fp," ");

q=q->next;

}

fclose(fp);

return OK;

}

}

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

函数功能:加载文件到单链表中

函数参数:单链表L，待加载文件的路径FileName

函数返回值:返回int类型，加载成功返回1

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

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

{

if(L!=NULL)return INFEASIBLE;

else

{

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

if(!fin)return ERROR;

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

LinkList q=L;

while(!feof(fin))

{

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

fscanf(fin,"%d",&p->data);

q->next=p;

q=q->next;

}

q->next=NULL;

fclose(fin);

return OK;

}

}

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

函数功能:在单链表的管理表中插入一个新的单链表

函数参数:引用单链表的管理表Lists的地址，待插入单链表的名称FileName

函数返回值:返回int类型，插入成功返回1

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

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

{

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

Lists.elem[Lists.length].L=(LNode\*)malloc(sizeof(LNode));

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

Lists.length++;

return OK;

}

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

函数功能:在单链表的管理表中删除一个单链表

函数参数:引用单链表的管理表Lists的地址，待删除单链表的名称FileName

函数返回值:返回int类型，删除成功返回1

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

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

{

for(int j=1;j<=Lists.length;j++)

{

if(strcmp(ListName,Lists.elem[j-1].name)==0)

{

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

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

{

strcpy(Lists.elem[i-1].name,Lists.elem[i].name);

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

}

Lists.length--;

return OK;

}

}

return ERROR;

}

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

函数功能:在单链表的管理表中查找单链表

函数参数:单链表的管理表Lists，待查找单链表的名称FileName

函数返回值:返回int类型，查找成功返回该单链表的位置

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

int LocateList(LISTS Lists,char ListName[])

{

for(int i=1;i<=Lists.length;i++)

if(strcmp(ListName,Lists.elem[i-1].name)==0)

{

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

return i;

}

return ERROR;

}

# 附录C 基于二叉链表二叉树实现的源程序

# 附录D 基于邻接表图实现的源程序