

**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201701**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期： 2017年 11月 22 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)

[1.2 系统设计 3](#_Toc458159882)

[1.3 系统实现 17](#_Toc458159883)

[1.4 实验小结 24](#_Toc458159884)

[2 基于链式存储结构的线性表实现 24](#_Toc458159885)

[2.1 问题描述 25](#_Toc458159886)

[2.2 系统设计 25](#_Toc458159887)

[2.3 系统实现 38](#_Toc458159888)

[2.4 实验小结 45](#_Toc458159889)

[3 基于二叉链表的二叉树实现 47](#_Toc458159890)

[3.1 问题描述 47](#_Toc458159891)

[3.2 系统设计 47](#_Toc458159892)

[3.3 系统实现 47](#_Toc458159893)

[3.4 实验小结 47](#_Toc458159894)

[4 基于二叉链表的二叉树实现 48](#_Toc458159895)

[4.1 问题描述 48](#_Toc458159896)

[4.2 系统设计 48](#_Toc458159897)

[4.3 系统实现 48](#_Toc458159898)

[4.4 实验小结 48](#_Toc458159899)

[参考文献 49](#_Toc458159900)

[附录A 基于顺序存储结构线性表实现的源程序 52](#_Toc458159901)

[附录B 基于链式存储结构线性表实现的源程序 67](#_Toc458159902)

[附录C 基于二叉链表二叉树实现的源程序 85](#_Toc458159903)

[附录D 基于邻接表图实现的源程序 87](#_Toc458159904)

# 1 基于顺序存储结构的线性表实现

## 1.1问题描述

在本次实验中，我以顺序表作为线性表的物理结构，使用C语言实现了动态分配顺序表的基本运算，演示过程中仅对一个顺序表进行操作，并将数据元素抽象成为一个整型变量,具体应用背景下可修改数据元素类型。程序源代码可在VS2015编译环境下编译。

具体到程序的实现，我的程序实现了线性表的12个基本运算：初始化表，销毁表，清空表，判定表空，求表长，获得元素，查找元素，获得前驱，获得后继，插入元素，删除元素，遍历表。2个附加功能：保存数据和加载数据。以及1个简单的菜单框架进行演示。在实现过程中，我为一些函数增加了特殊功能：（1）查找操作：运用了函数指针，通过compare()函数进行查找，这样在面对不同应用背景下的线性表查找操作只需更改compare()函数即可继续实现该背景下的查找。（2）遍历操作，使用visit()函数进行遍历，可以只修改visit()函数即可实现不同背景下的遍历内容。（3）存储操作，使用二进制的方式进行存储，存储效率高，速度快。（4）在调用功能函数前已对线性表的初始条件进行检验，确保程序不会异常退出。

## 1.2 系统设计

1.2.1 系统总体设计

用户打开程序后会看到如下界面：

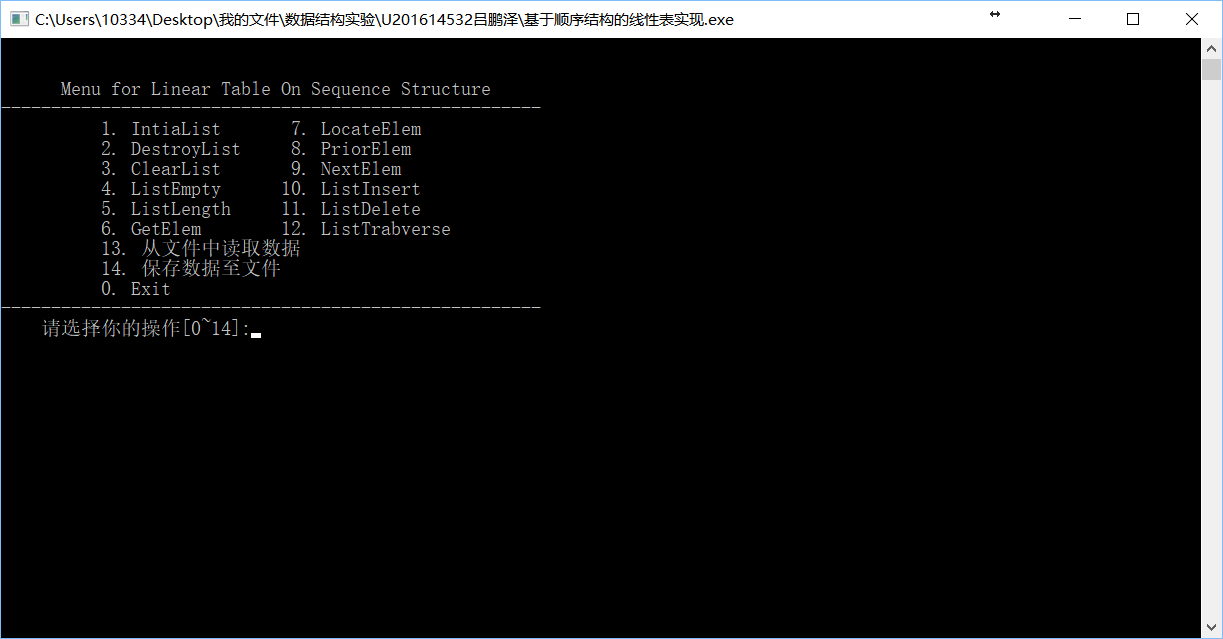


图1-1系统载入界面

用户通过输入在[1,14]区间的整数选择相应的功能，输入0退出程序。演示的过程可抽象为如图1-2所示的流程图。

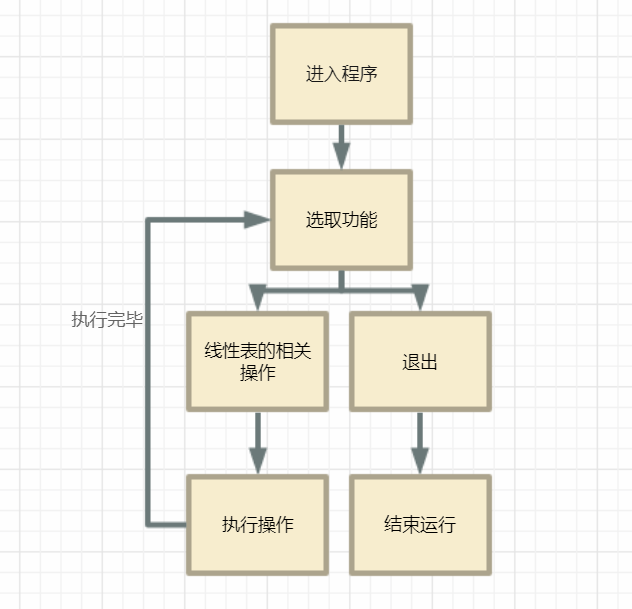


图1-2系统总体结构

1.2.2线性表物理结构

如图1-3，表示的是顺序表的物理结构。表的信息存储在顺序表结构L中，elem指示线性表的基地址，length指示线性表的当前长度，Listsize指示表所能存储的最大的数据元素的个数。

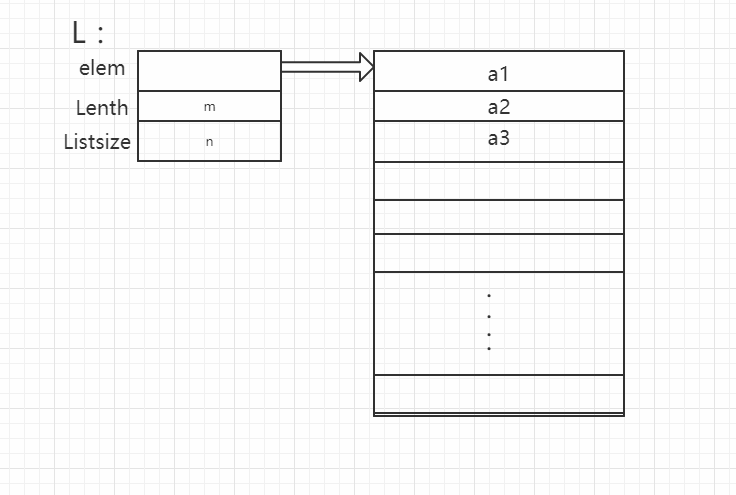


图1-3 顺序表物理结构

1.2.3相关常量的类型与定义

1.函数返回状态定义：

函数运行成功返回TRUE，失败返回FALSE，正常执行完毕返回OK，异常结束返回ERROR，动态分配空间不足返回OVERFLOW。在我的程序中用C语言描述如下所示：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

2.相关常量

FILENAME表示保存顺序表信息的文件名称，status表示函数运行状态，Elemtype为数据元素类型，LIST\_INIT\_SIZE为初始表大小，LISTINCREMENT为表增容大小。C语言描述如下所示：

#define FILENAME "data"

typedef int status;

typedef int ElemType;

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

3.顺序表结构定义

表的信息存储在顺序表结构L中，elem指示线性表的基地址，length指示线性表的当前长度，Listsize指示表所能存储的最大的数据元素的个数。C语言描述如下所示：

typedef struct{

ElemType \* elem;

int length;

int listsize;

}SqList;

1.2.4算法设计

* 1. InitiaList(&L)

算法思想：1.申请存储数据的空间；2.置表长为0。

操作结果：构造一个空的线性表

时间复杂度：O(1) （说明：以下时间复杂度均指平均时间复杂度）

流程图：

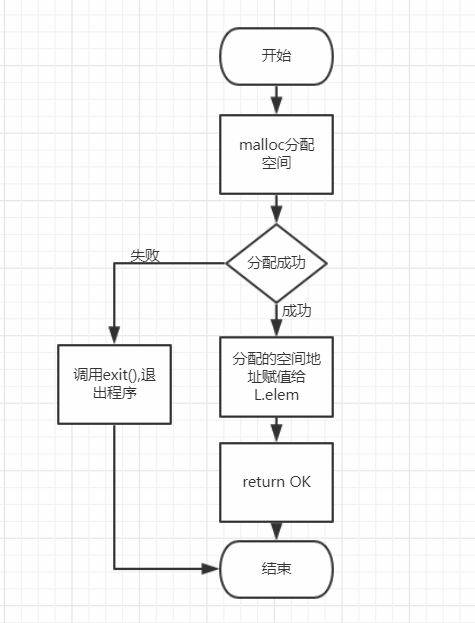


图1-4 InitiaList()流程图

* 1. DestroyList(&L)

算法思想：1.释放线性表的存储空间；2.置表指针为空

操作结果：销毁线性表L

时间复杂度：O(1)

流程图：

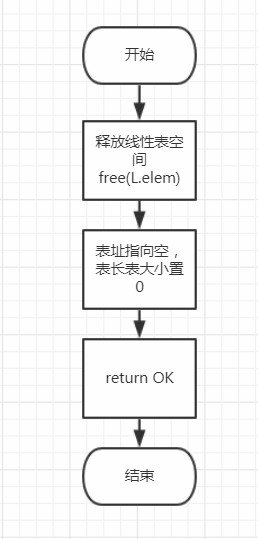


图1-5 DestroyList()流程图

* 1. ClearList(&L)

算法思想：置表长为0

操作结果：线性表L置空

时间复杂度：O(1)

流程图：

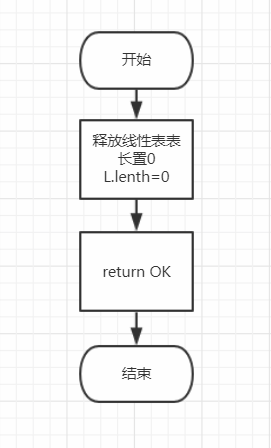


图1-6 ClearList()流程图

* 1. ListEmpty(L)

算法思想：若表长为0返回”TRUE”，否则返回”FALSE”。

操作结果：L为空返回TRUE,否则返回FALSE

时间复杂度：O(1)

流程图：

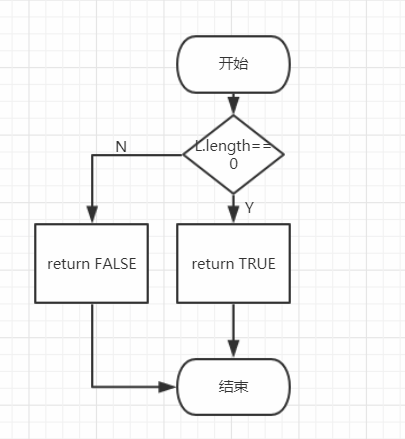


图1-7 ListEmpty()流程图

* 1. ListLength(L)

算法思想：返回表长

操作结果：返回线性表中元素的个数

时间复杂度：O(1)

流程图：

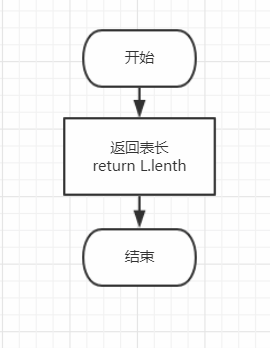


图1-8 ListLength(L)流程图

* 1. GetElem(L,i,&e)

算法思想：1.寻址公式定位第i个元素。2.将第i个元素赋值给e

操作结果：用e返回L中第i个数据元素的值

时间复杂度：O(1)

流程图：

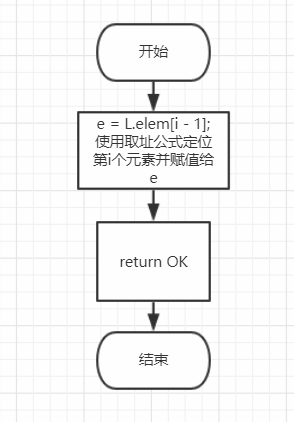


图1-9 GetElem()流程图

* 1. LocateElem(L,e,compare())

算法思想：1.用compare()函数查找e;2.找到返回该元素的位序,否则返回0。

操作结果：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。

时间复杂度：O(n)

流程图：

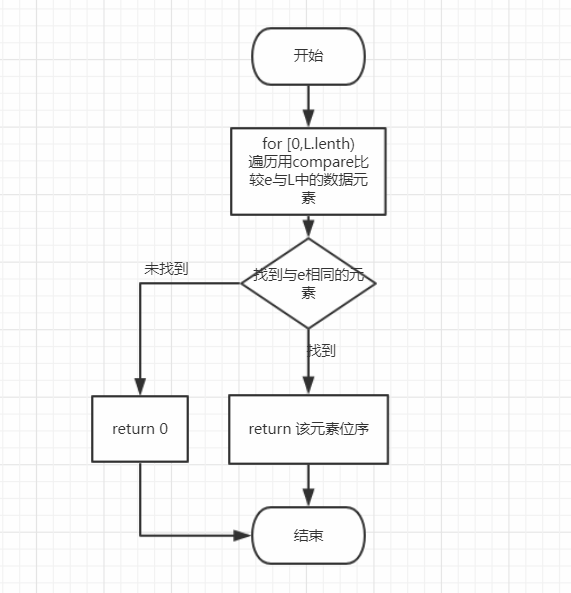


图1-10 LocateElem()流程图

* 1. PriorElem(L,cur\_e,&pre\_e)

算法思想：1.查找cur\_e获得其序号order; 2.若order>1,将order-1单元的元素值赋值给pre\_e,否则返回FALSE。

操作结果：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

时间复杂度：O(n)

流程图：

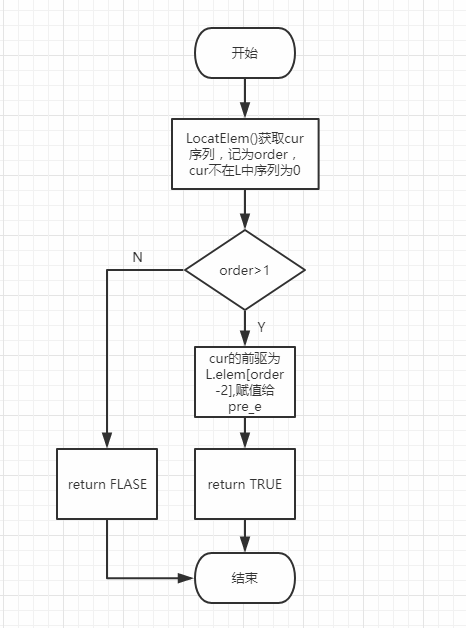


图1-11 PriorElem()流程图

* 1. NextElem(L,cur\_e,&next\_e)

算法思想：1.查找cur\_e获得其序号order; 2.若order<表长,将order+1单元的元素值赋值next\_e,否则返回FALSE。

操作结果：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

时间复杂度：O(n)

流程图：

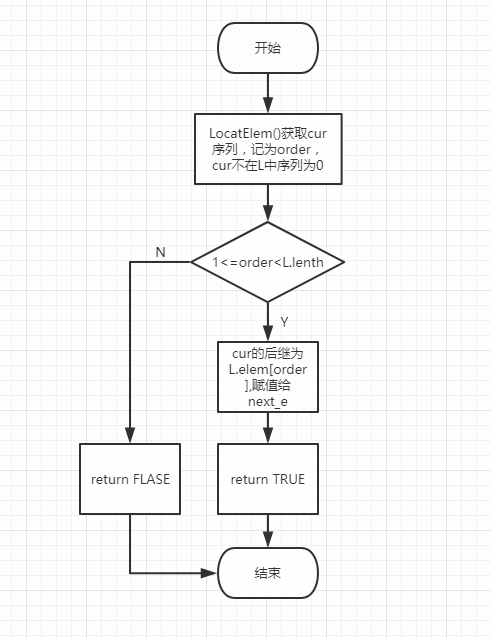


图1-12 NextElem()流程图

* 1. ListInsert(&L,i,e)

算法思想：1.判断空间是否已满，若满则增配空间，并修改listsize2.将序号为i-L.lenth的元素依次后移一位3.位置i插入e4.表长+1

操作结果：在L的第i个位置之前插入新的数据元素e。

时间复杂度：O(n)

流程图：

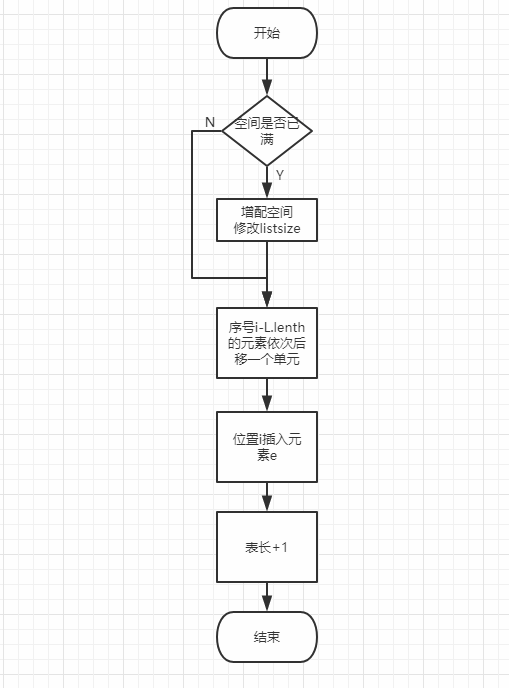


图1-13 ListInsert()流程图

* 1. ListDelete(&L,i,&e)

算法思想：1.i单元的值赋值给e；2.序号i+1-L.lenth的元素依次前移一个单元；3.表长-1

操作结果：删除L的第i个数据元素，用e返回其值。

时间复杂度：O(n)

流程图：

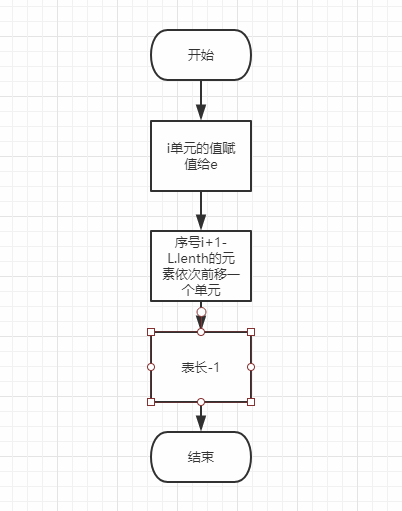


图1-14 ListDelete()流程图

* 1. ListTraverse(L,visit())

算法思想：使用visit()函数依次访问1-L.lenth数据元素

操作结果：对L的每个数据元素用函数visit()访问

时间复杂度：O(n)

流程图：

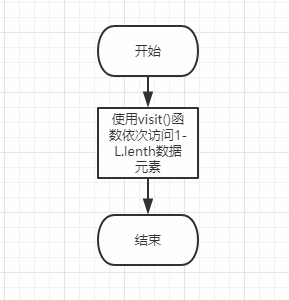


图1-15 ListTraverse()流程图

## 1.3系统实现

顺序表初始值为1，2，3，4，5，6，7，8共8个数据元素

1.初始界面

用户打开程序后可以看到如下界面，输入数字选择相应功能函数，输入0退出程序。

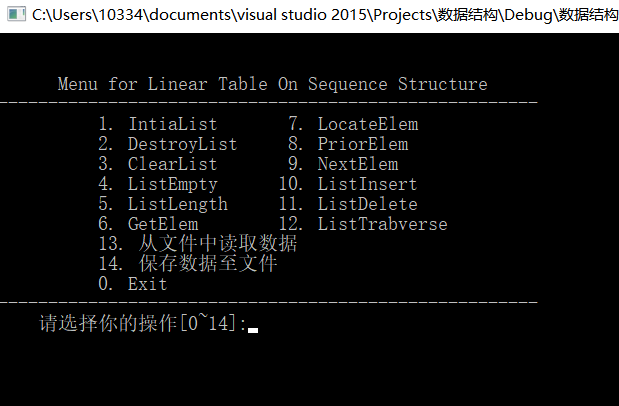
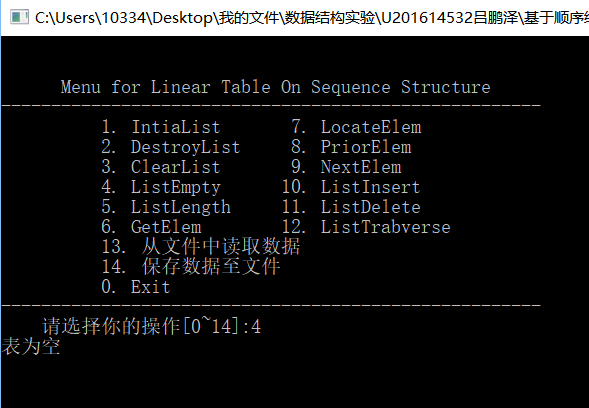


图1-16 系统载入界面

2.创建表

如图a，在表未创建前无法对表进行操作，如图b，c，在表创建后才可以对表进行操作。

|  |  |
| --- | --- |
| a | b |

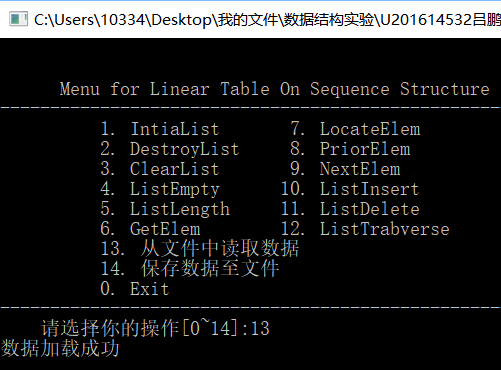


c

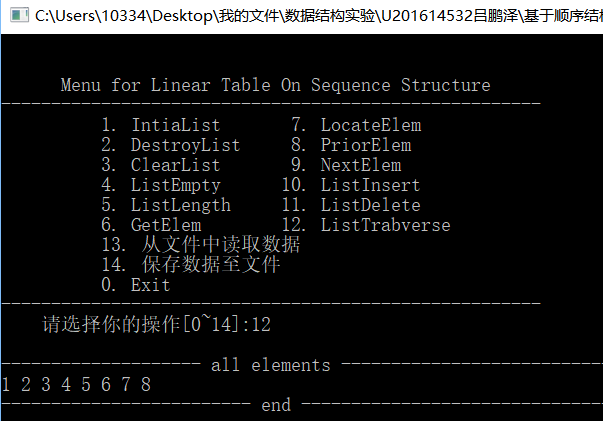
图1-16 创建表操作演示

3.从文件中加载数据及遍历操作的演示

创建表后表中不含数据元素，如图a，遍历表的结果为空。如图b，从文件中加载数据后再次遍历，结果如图c。



a b

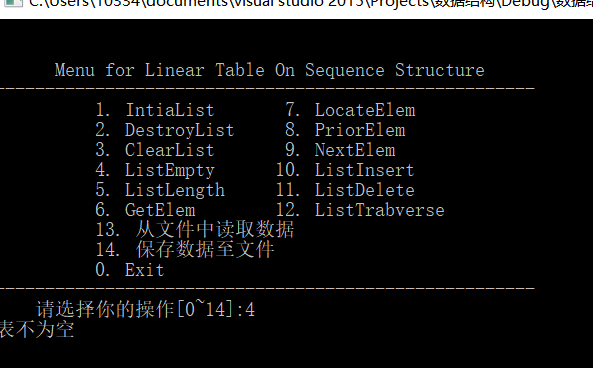
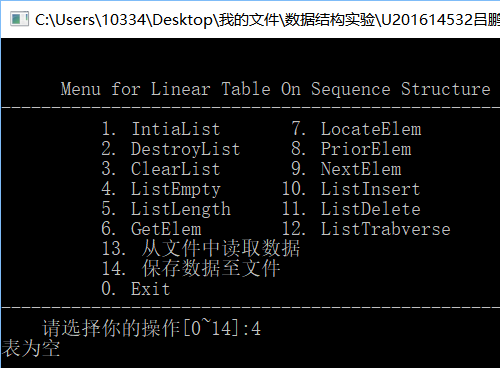


c

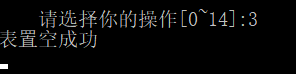
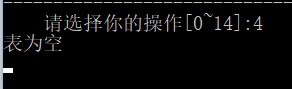
图1-17 加载数据及遍历操作的演示

4.判表空及置空表

如图a，初始化表后的操作结果，此时表为空。如图b，从文件中加载数据后再次判空表，此时表不为空。如图c、d，置空表后再次判表空。



a b

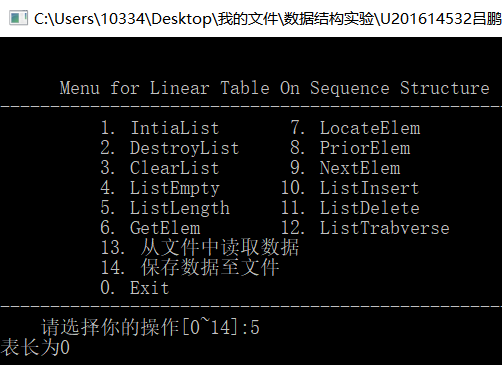
 

c d

图1-18 判空表

5.求表长

如图a，表为空是求表长的结果。如图b，从文件中读取数据后求表长的结果

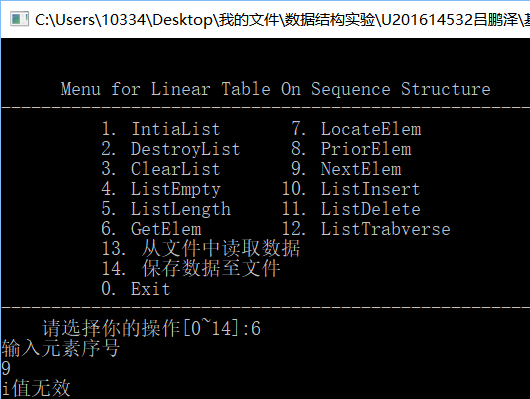
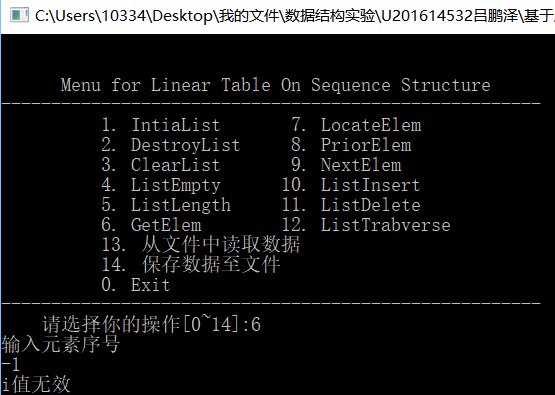


a b

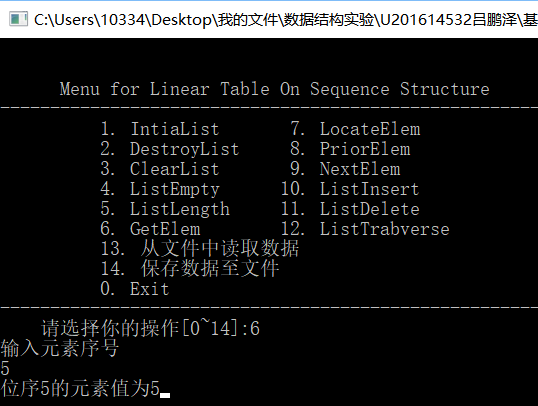
图1-19 求表长

6.获取元素

如图a，b，当元素序号的值小于1或大于表长时，程序会提示出错。如图c，只有元素序号大于等于1，小于等于表长时才能正确获取元素值。



a b

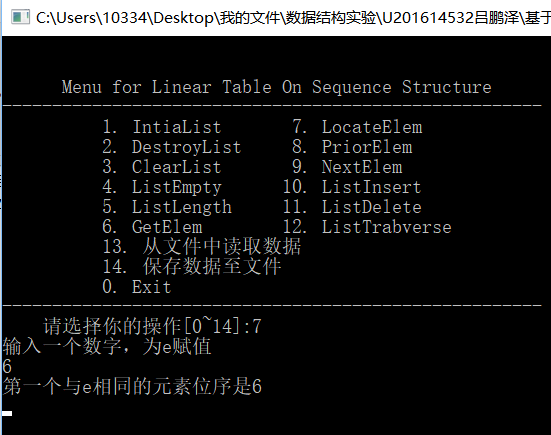
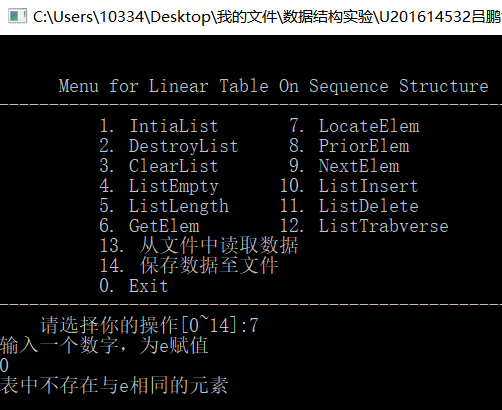


c

图1-20 获取元素

7.查找元素

如图a，为元素不在表中的查找结果。如图b，为元素在表中的查找结果。

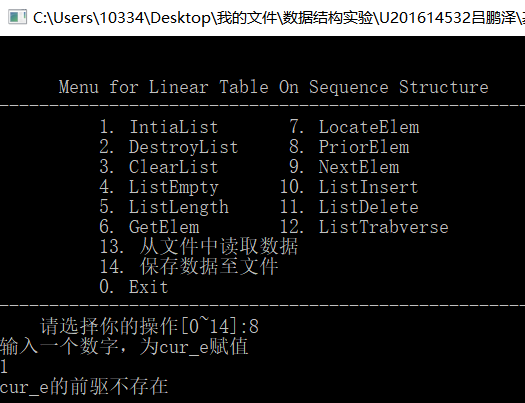
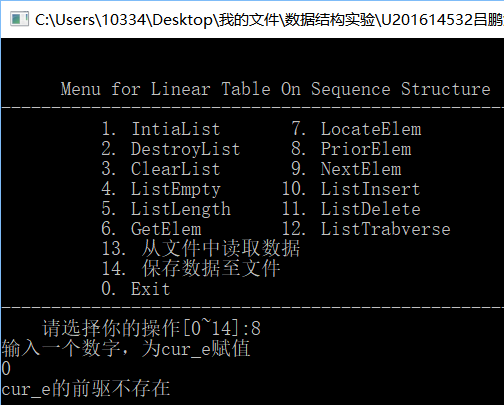


a b

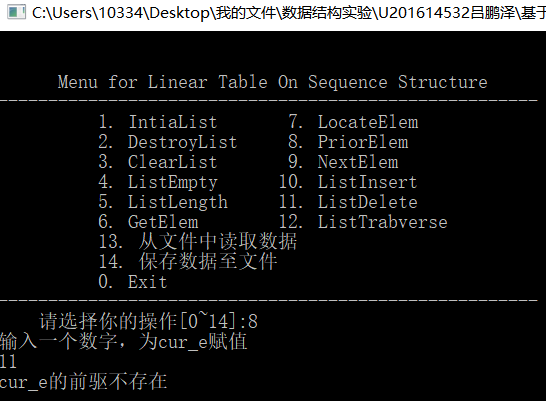
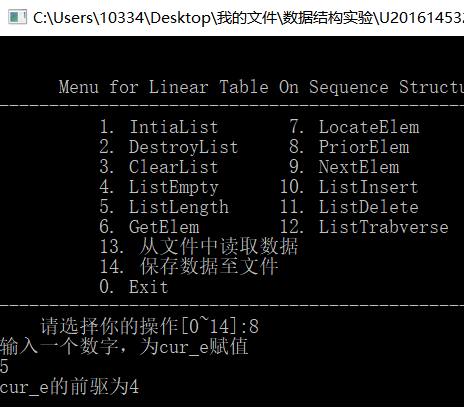
图1-21 查找元素

8.获取前驱

如图a、d，为元素cur不在表中的结果。如图b，为元素在表头的结果。如图c，为元素在表中的结果。



a b

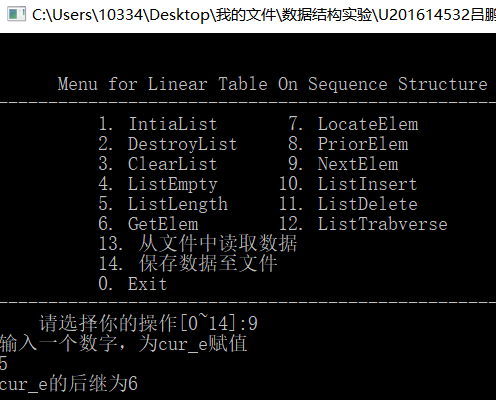
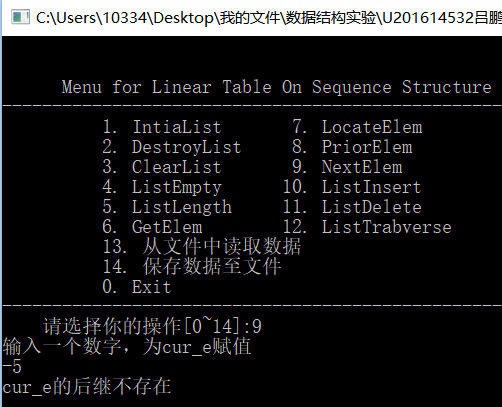


c d

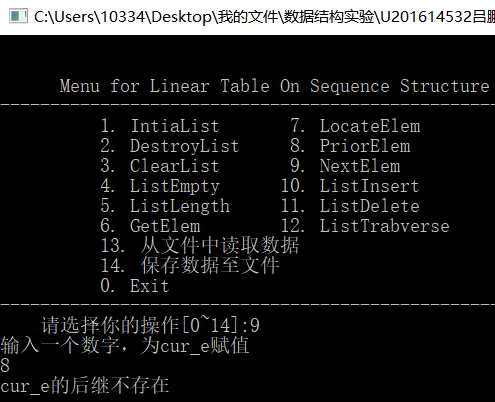
图1-22 获取前驱

9.获取后继：

如图a，为元素不在表中的结果。如图b，为元素在表中的结果。如图c，为元素在表尾的结果。



a b



c

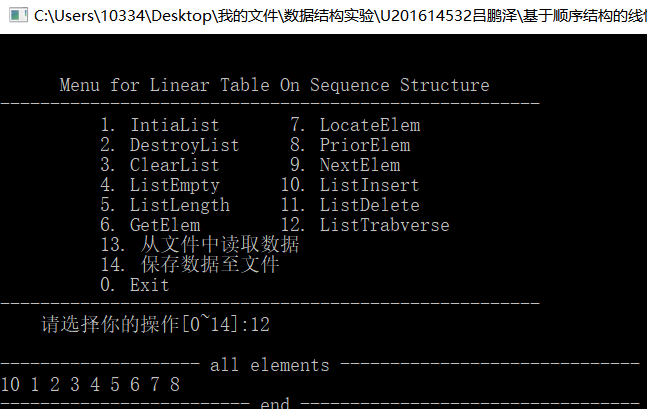
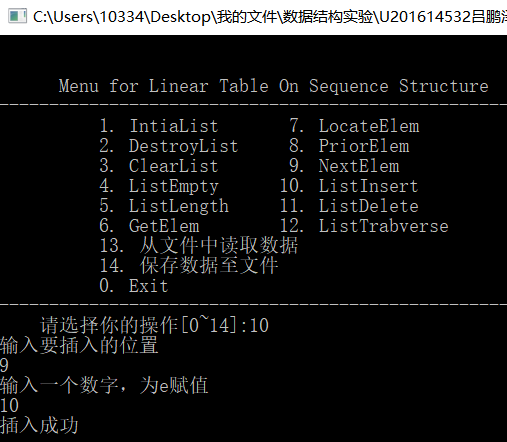
图1-23 获取后继

10.插入元素及插入后的结果

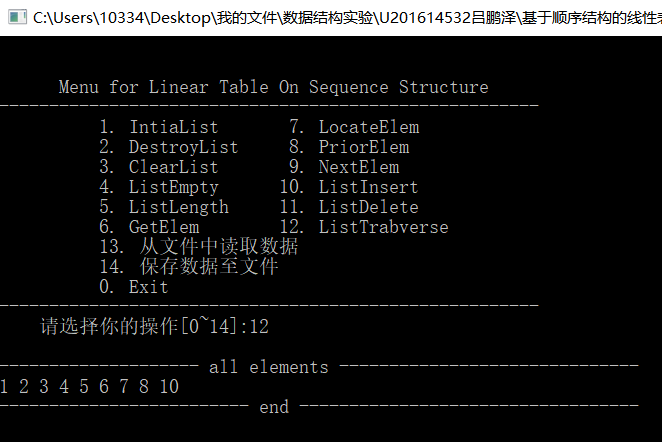
如图a，为在表外插入的结果。如图b、c，在表头插入结果及插入后的遍历。如d、e，在表尾插入结果及插入后的遍历结果。



a b

c d

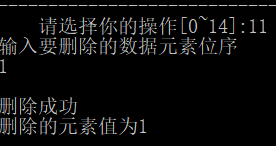
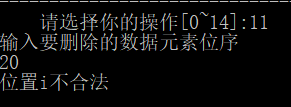


e

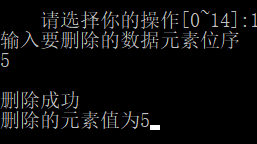
图1-24 插入元素

11.删除元素及删除后的结果：

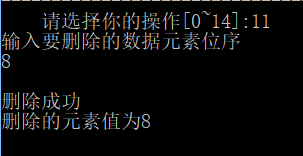
如图a，删除表外的元素会提示位置不合法。如图b、c，删除表头元素及删除后的遍历结果。如图d、e，删除表中元素及遍历后的结果。如图f、g，删除表尾元素及遍历后的结果。



a b



c d

e f



g

13.销毁表

销毁表后其余功能无法被调用。

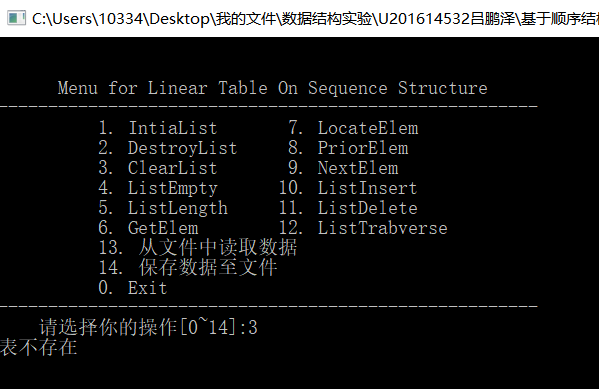


图1-26 销毁表

## 1.4 实验小结

本次数据结构实验让我加深了对线性表的理解，并让我复习了malloc()的用法，且重新掌握了realloc()的用法，以及形参&修饰符的运用。在课堂上的理论学习过程中，我一开始觉得线性表挺简单的，但是通过这次实际操作我发现从理论到时间还是有许多工作要做的，比如在调用函数时初始条件的判断、线性表空间的增配的都是十分容易出错的地方，在实验过程中，虽然没有遇到大的问题，但小问题遇到了不少，比如一开始在清空表时我把他同销毁表搞混了，后来仔细阅读书上的要求后才更正；另外，在写case是忘记加break了，也出现了一些小问题。在这次实验中，我强化了自己的编程能力。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

在本次实验中，我将以链表作为线性表的物理结构，实验中我采用了带头结点的动态链表，使用C语言实现了该链表的线性表基本运算，演示过程中对两个动态链表进行操作，因此用户在调用功能前需要先确认对哪一个链表进行操作。此外，实验过程中我将数据元素抽象成为一个整型变量，在具体的应用背景下可修改数据元素类型来满足具体需求。程序源代码可在VS2015编译环境下编译通过。

具体到程序的实现，我的程序实现了线性表的12个基本运算：初始化表，销毁表，清空表，判定表空，求表长，获得元素，查找元素，获得前驱，获得后继，插入元素，删除元素，遍历表。3个附加功能：多链表操作、保存数据、加载数据。并使用1个简单的菜单框架进行演示。在实现过程中，我为一些函数增加了特殊功能：（1）查找操作：运用了函数指针，通过compare()函数进行比较，这样在面对不同应用背景下的线性表查找操作只需更改compare()函数即可继续实现该背景下的比较。（2）遍历操作，使用visit()函数进行遍历，可以只修改visit()函数即可实现不同应用背景下的遍历。（3）存储操作，使用二进制的方式进行存储，存储效率高，速度快。（4）在调用功能函数前已对线性表的初始条件进行检验，确保程序不会异常退出。（5）实现了对多链表进行操作。

## 2.2 系统设计

2.2.1 系统总体设计：

用户打开程序后会看到如下界面：

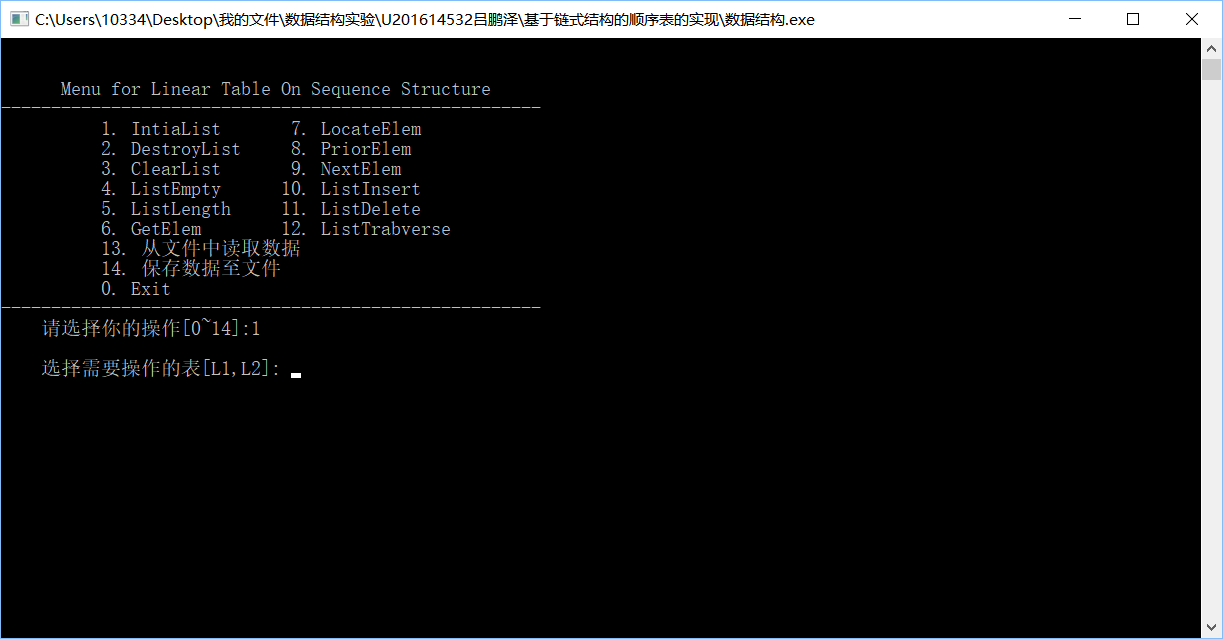


图2-1系统载入界面

用户通过输入在[1,14]区间的整数选择相应的功能，输入0退出程序，输入回车确认后需要选择操作的链表，输入L1或L2，。演示的过程可抽象为如图2-2所示的流程图。

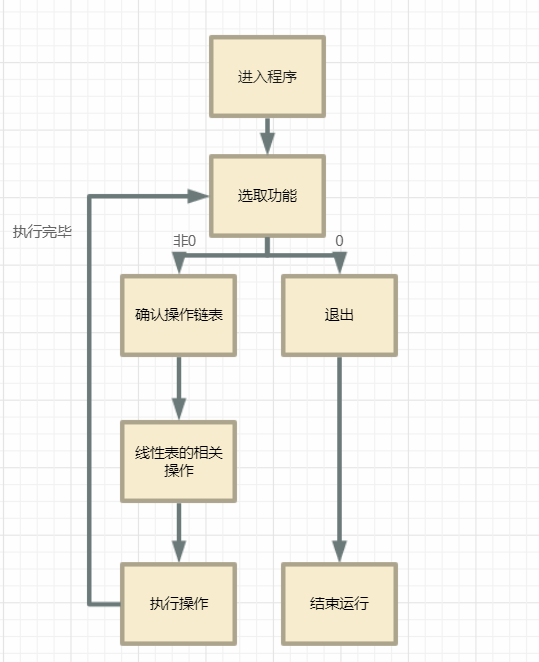
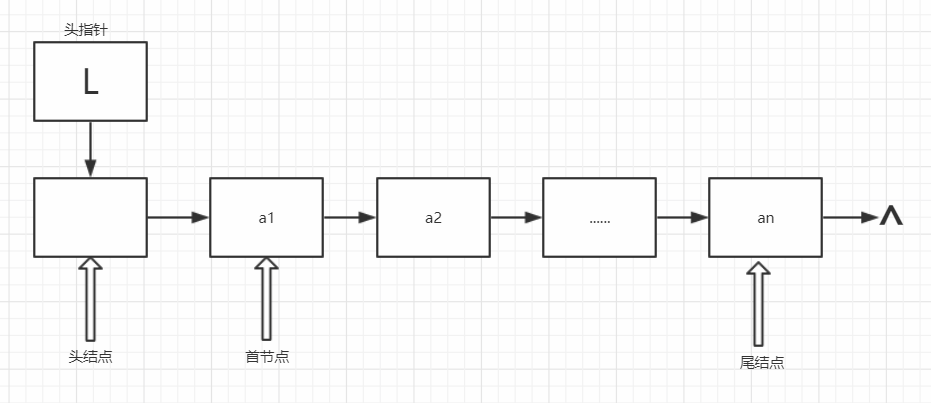


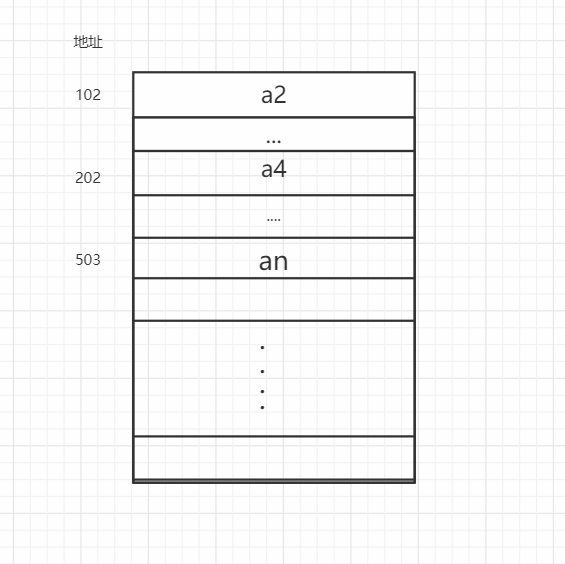
图2-2系统总体结构

2.2.2链表的物理结构

如图2-3-a，表示的是动态链表的逻辑结构，有一个头指针L指向头结点，每一个结点表示一个数据元素，每个结点包含数据域和指针域，尾节点指向空。如图2-3-b，表示的是动态链表在计算机中的存储结构，各个结点随机分布在内存中，并通过指针相连接。



a 动态链表逻辑结构



b 动态链表存储结构

图2-3 动态链表结构

2.2.3相关常量的类型与定义

1.函数返回状态定义：

函数运行成功返回TRUE，失败返回FALSE，正常执行完毕返回OK，异常结束返回ERROR，动态分配空间不足返回OVERFLOW。在我的程序中用C语言描述如下所示：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

2.相关常量

FILENAME\_L表示保存动态链表信息的文件名称，status表示函数执行状态。C语言描述如下所示：

#define FILENAME\_L1 "dataL1"

#define FILENAME\_L2 "dataL2"

typedef int status;

3.顺序表结构定义

SqList表示头指针，指向头结点，Elemtype为数据元素类型，包含数据域和指针域，在本实验中数据域抽象为一个整数，指针域仅包含指向下一个节点的指针。C语言描述如下所示：

typedef ElemType\* SqList;//头指针

typedef struct elemtype

{

int date;

elemtype \* next;

}ElemType; //数据元素类型定义

1.2.4算法设计

* 1. InitiaList(&L)

算法思想：1.头指针声明一个存储结点；2.存储结点的指针域置空

操作结果：构造一个空的线性表

时间复杂度：O(1) （说明：以下时间复杂度均指平均时间复杂度）

流程图：

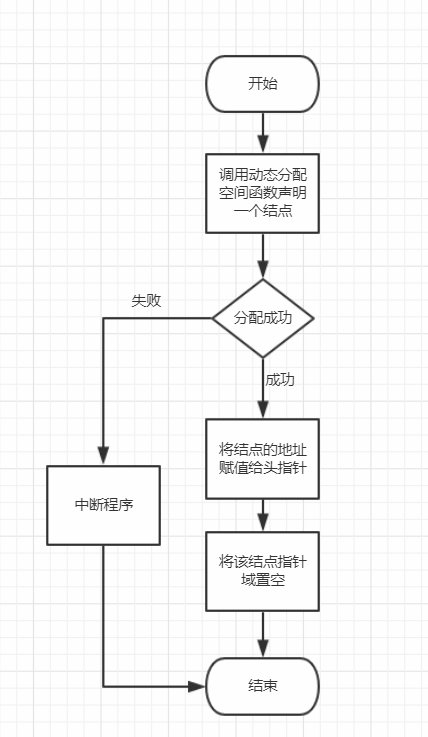


图2-4 InitiaList()流程图

* 1. DestroyList(&L)

算法思想：1.释放包括头结点在内的所有存储结点；2.置头指针为空

操作结果：销毁线性表L

时间复杂度：O(n)

流程图：

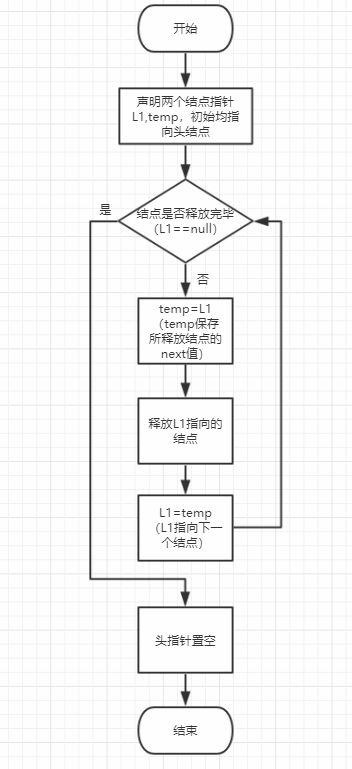


图2-5 DestroyList()流程图

* 1. ClearList(&L)

算法思想：1.释放除头结点之外的所有存储结点；2.头结点指针域置空

操作结果：线性表L置空

时间复杂度：O(n)

流程图：

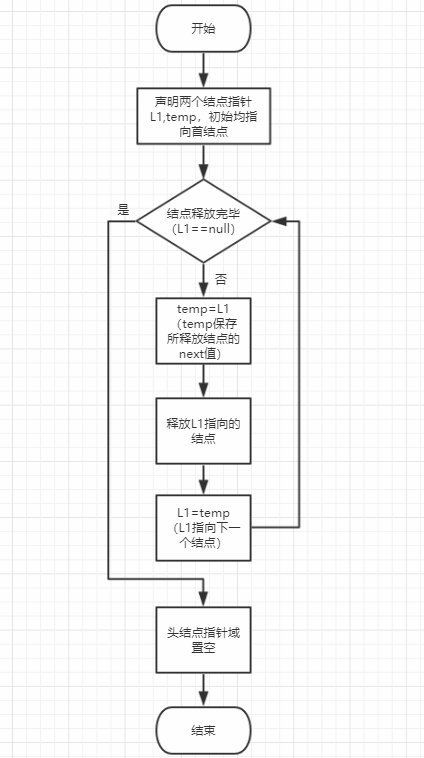


图2-6 ClearList()流程图

* 1. ListEmpty(L)

算法思想：若表长为0返回”TRUE”，否则返回”FALSE”。

操作结果：L为空返回TRUE,否则返回FALSE

时间复杂度：O(1)

流程图：

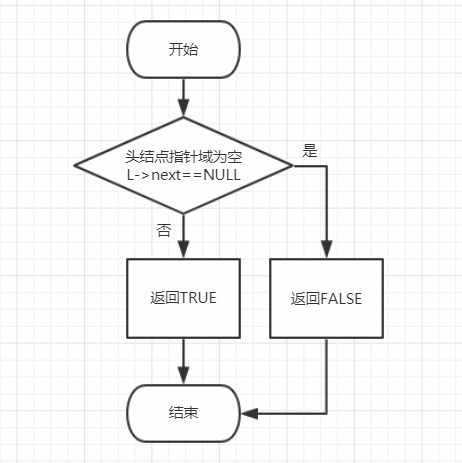


图2-7 ListEmpty()流程图

* 1. ListLength(L)

算法思想：1.指针p指向头结点；2.当p非空时移动p，统计移动次数count；3.返回count

操作结果：返回线性表中元素的个数

时间复杂度：O(n)

流程图：

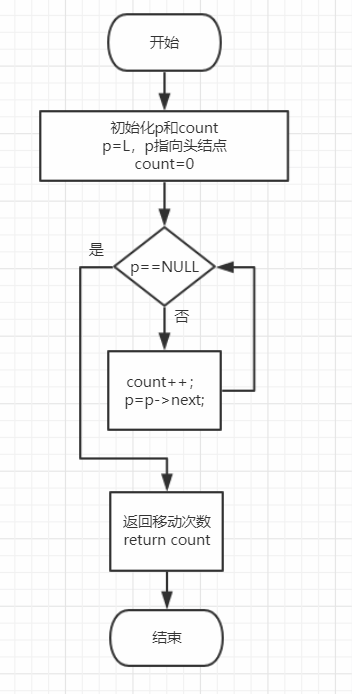


图2-8 ListLength(L)流程图

* 1. GetElem(L,i,&e)

算法思想：1.遍历链表定位第i个元素；2.将第i个元素赋值给e

操作结果：用e返回L中第i个数据元素的值

时间复杂度：O(n)

流程图：

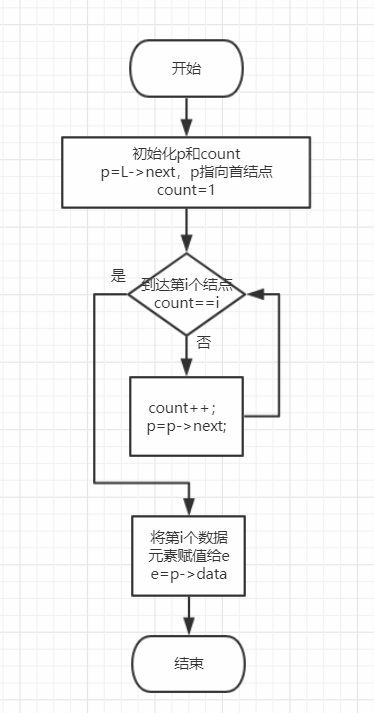


图2-9 GetElem()流程图

* 1. LocateElem(L,e,compare())

算法思想：1.遍历表中所有结点，分别与e进行比较，并记录结点位置；2.若找到与e满足compare关系的结点，返回该节点位置；3.未找到返回0；

操作结果：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。

时间复杂度：O(n)

流程图：

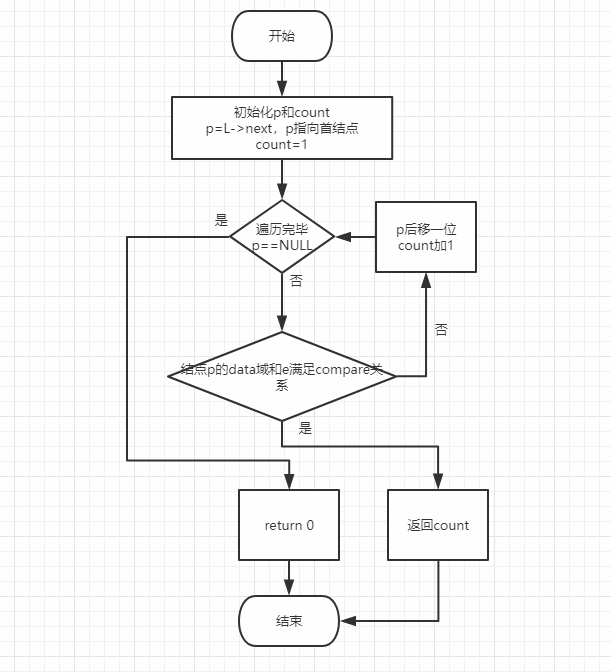


图2-10 LocateElem()流程图

* 1. PriorElem(L,cur\_e,&pre\_e)

算法思想：1.在表中查找cur\_e; 2.若找到并且cur\_e不是第一个元素，将cur\_e前驱的元素值赋值pre\_e,否则返回FALSE。

操作结果：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

时间复杂度：O(n)

流程图：

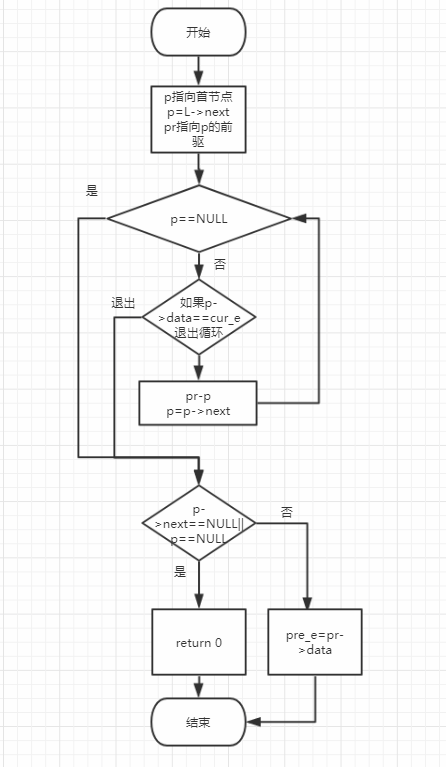


图2-11 PriorElem()流程图

* 1. NextElem(L,cur\_e,&next\_e)

算法思想：1.在表中查找cur\_e; 2.若找到并且cur\_e不是最后一个元素，将cur\_e下一个单元的元素值赋值next\_e,否则返回FALSE。

操作结果：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

时间复杂度：O(n)

流程图：

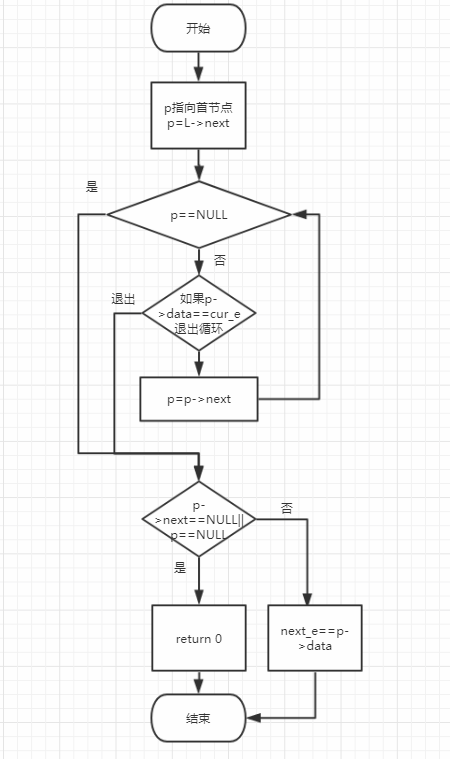


图2-12 NextElem()流程图

* 1. ListInsert(&L,i,e)

算法思想：1. 将指针p向后移动指向第i存储结点的前驱结点；2. 指针q指向申请的新结点，数据元素e赋值 到该结点的数据域；3.修改p，q指针。

操作结果：在L的第i个位置之前插入新的数据元素e。

时间复杂度：O(n)

流程图：

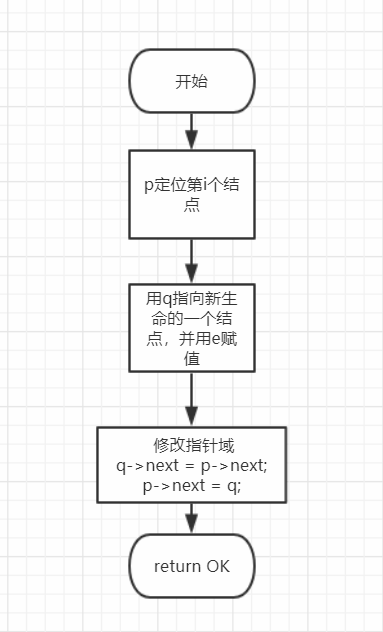


图2-13 ListInsert()流程图

* 1. ListDelete(&L,i,&e)

算法思想：1. 将指针p向后移动指向第i存储结点之直接前驱结点； 指针q指向第i存储结点，该结点的数据赋值到变量e ；2.修改指针；3.释放结点；

操作结果：删除L的第i个数据元素，用e返回其值。

时间复杂度：O(n)

流程图：

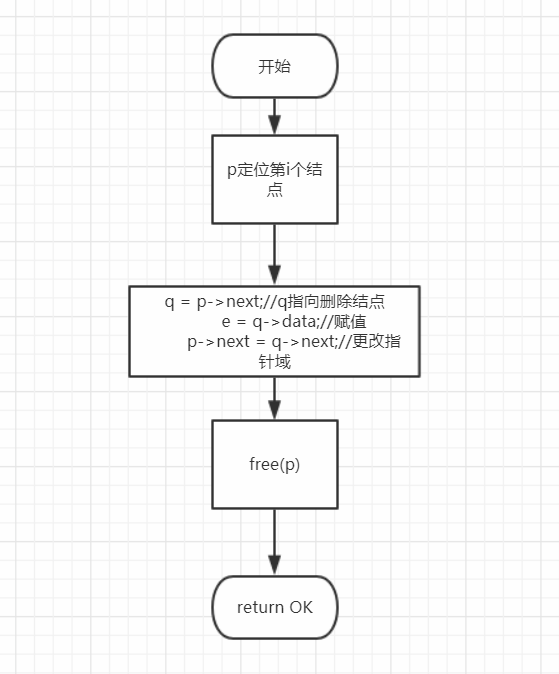


图2-14 ListDelete()流程图

* 1. ListTraverse(L,visit())

算法思想：使用visit()函数依次访问所有数据元素

操作结果：对L的每个数据元素用函数visit()访问

时间复杂度：O(n)

流程图：

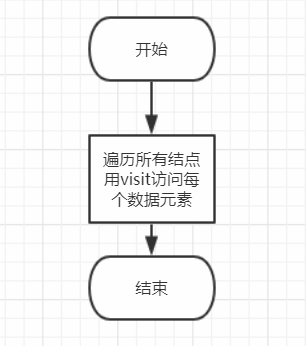


图2-15 ListTraverse()流程图

* 1. Savedata(L,filename)

算法思想：使用二进制存储方式保存所有结点的值

操作结果：将链表L的数据保存到filename文件中

时间复杂度：O(n)

流程图：

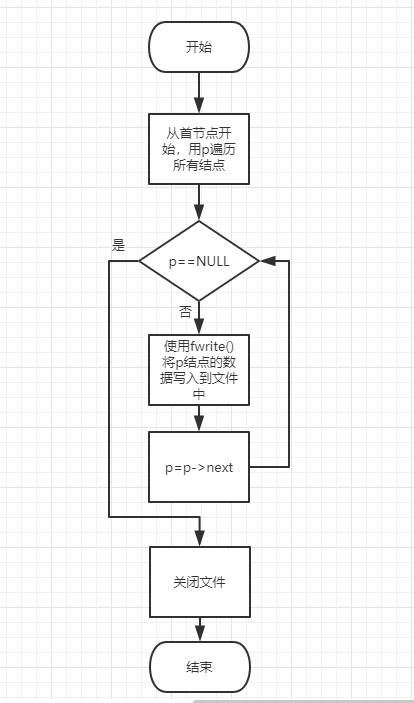


图2-16 Savedata()流程图

* 1. Loaddata(L,filename)

算法思想：使用二进制读取方式加载所有结点的值，动态创建链表

操作结果：将filename文件中的数据加载到动态链表中

时间复杂度：O(n)

流程图：

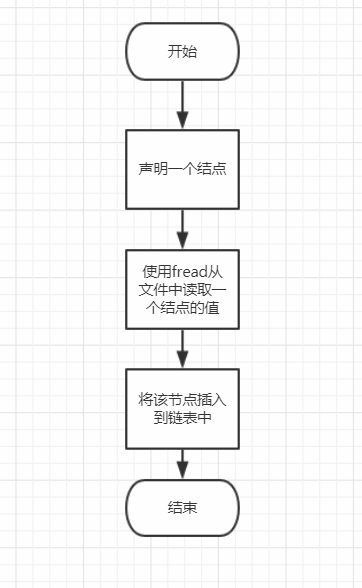


图2-16 Loaddata()流程图

* 1. visit(e)

算法思想：输出数据元素e的值

操作结果：输出数据元素e的值

时间复杂度：O(1)

流程图：

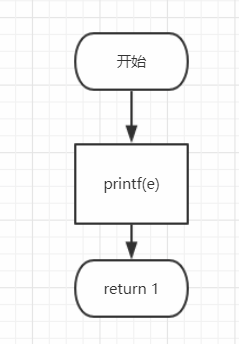


图2-17 visit()流程图

## 2.3 系统实现

L1的数据文件值为1-8，L2的数据文件值为1-5,为便于表现多链表操作，函数演示交替调用L1、L2。

1.初始界面

用户打开程序后可以看到如下界面，输入数字选择相应功能函数，输入0退出程序。

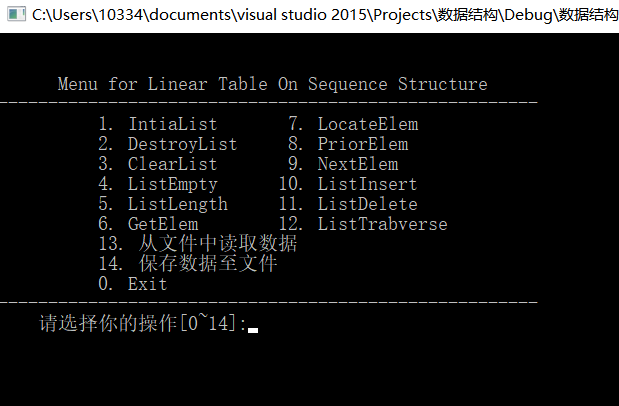


图2-18 系统载入界面

2.创建表

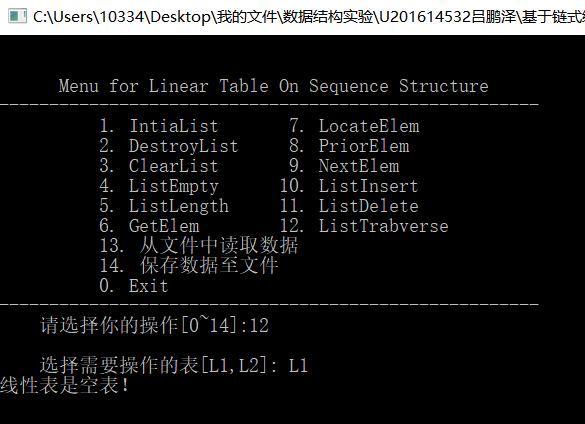
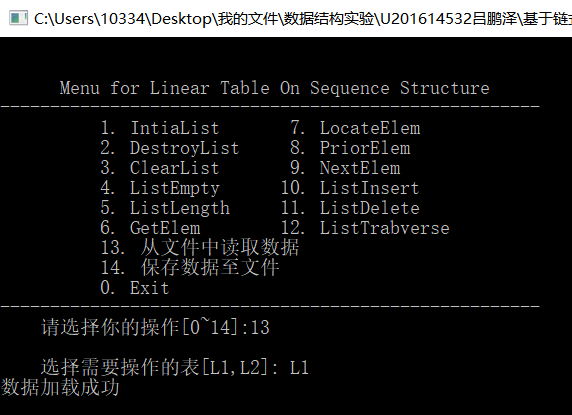
如图a，在表L1未创建前无法对表进行操作，如图b，表创建成功。

|  |  |
| --- | --- |
| a | b |

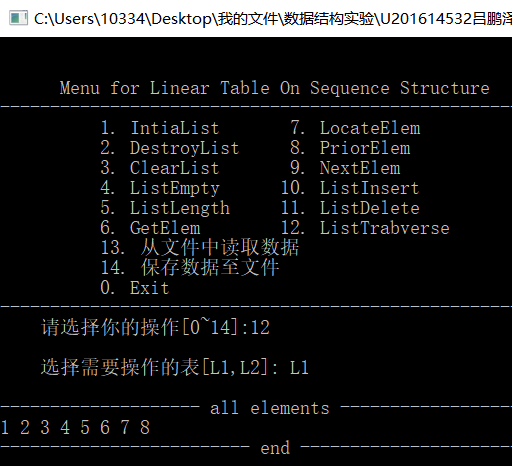
图2-19 创建表操作演示

3.从文件中加载数据及遍历操作的演示

创建表L1后表中不含数据元素，如图a，遍历表的结果为空。如图b，从文件中加载数据后再次遍历，结果如图c。

a b

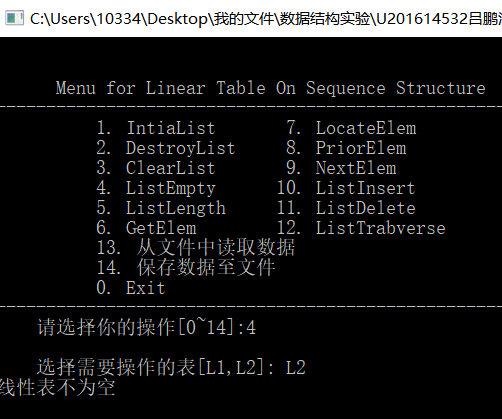
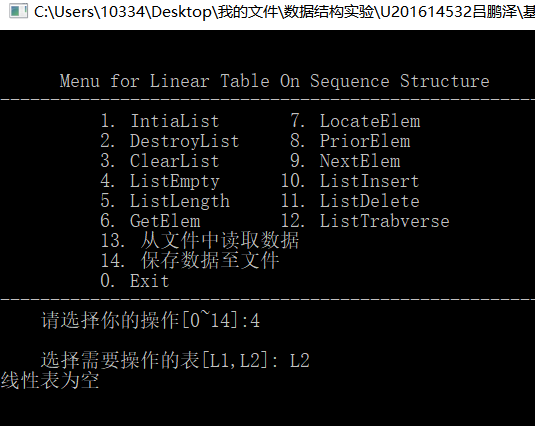


c

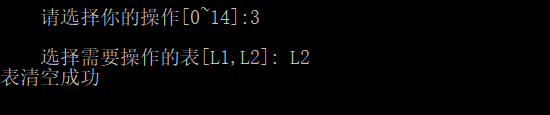
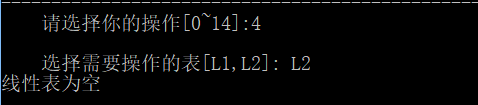
图2-20 加载数据及遍历操作的演示

4.判表空及置空表

如图a，初始化表L2后的操作结果，此时表为空。如图b，从文件中加载数据后再次判空表，此时表不为空。如图c、d，置空表后再次判表空。



a b

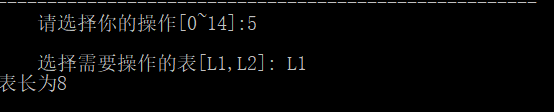
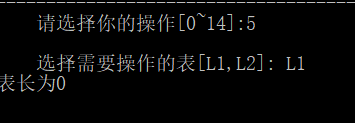
 

c d

图2-21 判空表

5.求表长

如图a，表为空是求表长的结果。如图b，从文件中读取数据后求L1表长的结果

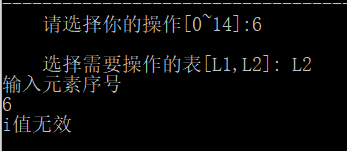
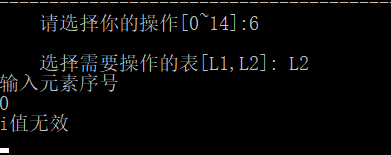


a b

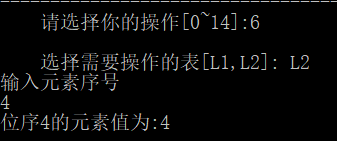
图2-22 求表长

6.获取元素

如图a，b，当元素序号的值小于1或大于L2表长时，程序会提示出错。如图c，只有元素序号大于等于1，小于等于表长时才能正确获取元素值。



a b

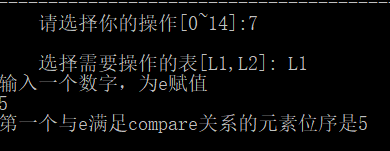
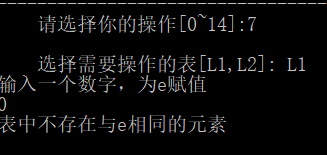


c

图2-23 获取元素

7.查找元素

如图a，为元素不在表L1中的查找结果。如图b，为元素在表中的查找结果。

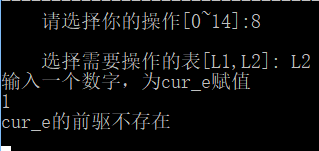
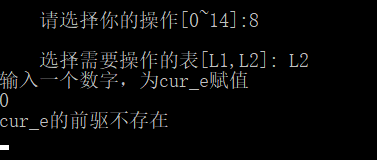


a b

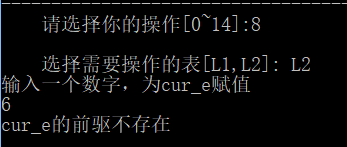
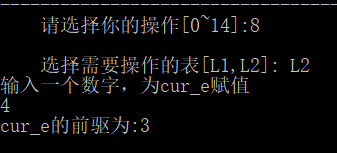
图2-24 查找元素

8.获取前驱

如图a、d，为元素cur不在表L2中的结果。如图b，为元素在表头的结果。如图c，为元素在表中的结果。



a b

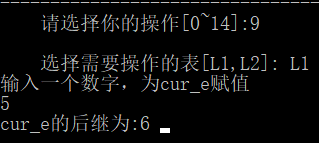
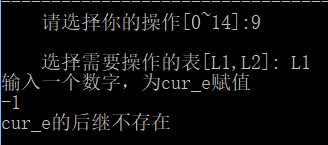


c d

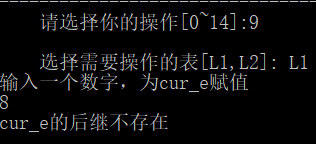
图2-25 获取前驱

9.获取后继：

如图a，为元素不在表L1中的结果。如图b，为元素在表中的结果。如图c，为元素在表尾的结果。



a b

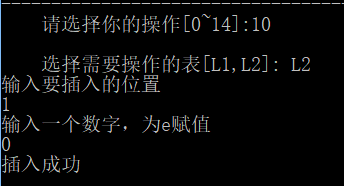
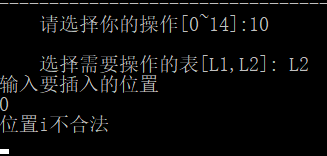


c

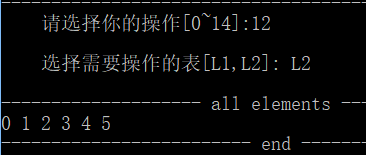
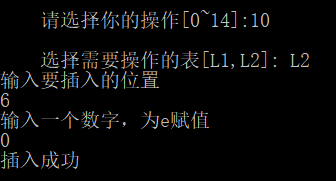
图2-26 获取后继

10.插入元素及插入后的结果

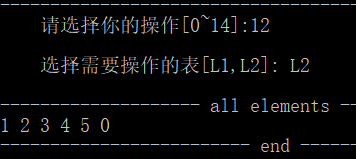
如图a，为在表L2外插入的结果。如图b、c，在表头插入结果及插入后的遍历。如d、e，在表尾插入结果及插入后的遍历结果。



a b

c d

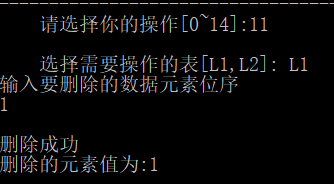
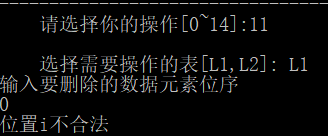


e

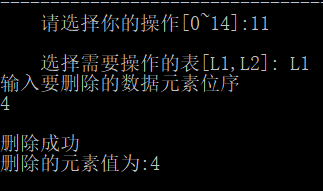
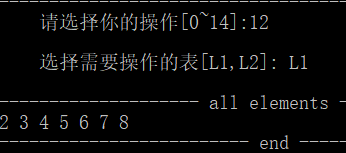
图2-27 插入元素

11.删除元素及删除后的结果：

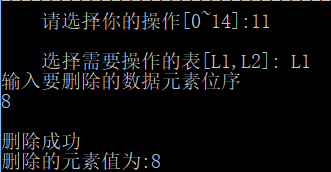
如图a，删除表L1外的元素会提示位置不合法。如图b、c，删除表头元素及删除后的遍历结果。如图d、e，删除表中元素及遍历后的结果。如图f、g，删除表尾元素及遍历后的结果。



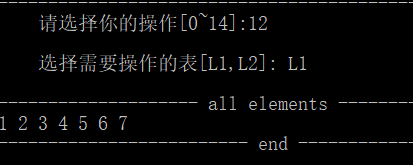
a b



c d

e f



g

图2-28 删除元素

13.销毁表

销毁表后其余功能无法被调用。

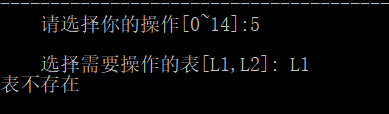
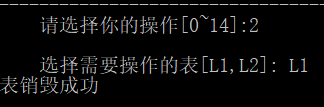


图2-29 销毁表

## 2.4 实验小结

在这次实验中，我实现了最基本的带头结点的动态链表。虽然C语言已经学过链表了，但通过理论课的学习我发现我对链表的掌握还很不够，尤其是通过这次的上机实验，我发现了自己动手能力不足，理论课听懂了不代表自己会写，这是我平时动手少的原因。另外，在实现的过程中，我看到了函数编程的好处，在编写链表的基本运算时，我出现了一些问题，由于是函数编程，我只修改一个函数就可以修正错误，而不用扫描整个程序代码。更重要的是，通过这两次实验，我感受到了自己程序调试水平的提高，之前好久都找不到的错误经过这两次的上机训练可以很快找出。

# 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 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 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

指导教师评定意见

一、对实验报告的评语

|  |
| --- |
|  |

二、对实验报告评分

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 程序内容  (36.8分) | 程序规范  (9.2分) | 报告内容  (36.8分) | 报告规范  (9.2分) | 考勤  （8分） | 逾期扣分 | 合 计  (100分) |
| 得分 |  |  |  |  |  |  |  |

# 附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <memory.h>

/\*--------------------page 10 on textbook ----------------------------------\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define FILENAME "data"//数据保存的文件名

typedef int status;

typedef int ElemType; //数据元素类型定义

/\*--------------------page 22 on textbook ----------------------------------\*/

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

/\*--------------------page 19 on textbook ----------------------------------\*/

status InitiaList(SqList & L);

status DestroyList(SqList & L);

status ClearList(SqList &L);

status ListEmpty(SqList L);

int ListLength(SqList L);

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

status LocatElem(SqList L, ElemType e, status(\*compare)(ElemType x, ElemType y));

status PriorElem(SqList L,ElemType cur,ElemType & pre\_e);

status NextElem(SqList L,ElemType cur,ElemType & next\_e);

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

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

status ListTrabverse(SqList L, status(\*visit)(ElemType e));

status compare(ElemType x, ElemType y);

status Loaddata(SqList &L);//加载数据

status Savedata(SqList L);

status visit(ElemType e);

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

void main(void){

SqList L;

L.elem = NULL;

int i;//插入位置

int op = 1;

int flat = 1;//表不存在

while (op) {

system("cls");

printf("\n\n");

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

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

printf(" 1. IntiaList 7. LocateElem\n");

printf(" 2. DestroyList 8. PriorElem\n");

printf(" 3. ClearList 9. NextElem \n");

printf(" 4. ListEmpty 10. ListInsert\n");

printf(" 5. ListLength 11. ListDelete\n");

printf(" 6. GetElem 12. ListTrabverse\n");

printf(" 13. 从文件中读取数据\n");

printf(" 14. 保存数据至文件\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~14]:");

scanf("%d", &op);

getchar();

switch (op) {

case 1:

if (!L.elem&&flat)//若线性表不存在，则创建

{

if (InitiaList(L) == OK) {

printf("线性表创建成功！\n");

flat = 0;//表已创建

}

else printf("线性表创建失败！\n");

}

else printf("线性表已存在\n");

getchar(); getchar();

break;

case 2:

if (L.elem)//若表存在

{

DestroyList(L);

printf("表销毁成功\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 3:

if (L.elem)//若表存在

{

ClearList(L);

printf("表置空成功\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 4:

if (L.elem)//若表存在

{

if (ListEmpty(L) == 1)

printf("表为空\n");

else printf("表不为空\n");

}

else printf("表不存在\n");

getchar();

break;

case 5:

if (L.elem)//若表存在

{

printf("表长为%d\n", ListLength(L));

}

else printf("表不存在\n");

getchar();

break;

case 6:

ElemType e;

if (L.elem)//表存在

{

printf("输入元素序号\n");

scanf(" %d", &i);

getchar();

if (i >= 1 && i <= ListLength(L))//若i值有效

{

GetElem(L, i, e);//获取位序为i的元素

printf("位序%d的元素值为%d", i, e);//输出获取的元素值

}

else printf("i值无效\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 7:

if (L.elem)

{

ElemType e;

printf("输入一个数字，为e赋值\n");

scanf("%d", &e);

getchar();

//输出第一个与e满足compare关系的元素位序，不存在位序为0

int order = LocatElem(L, e, compare);

if (order == 0)

printf("表中不存在与e相同的元素\n");

else printf("第一个与e相同的元素位序是%d\n", order);

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 8:

if (L.elem)//若表存在

{

ElemType cur\_e, pre\_e;

printf("输入一个数字，为cur\_e赋值\n");

scanf("%d", &cur\_e);

getchar();

if (PriorElem(L, cur\_e, pre\_e) == 1)//如果cur\_e存在前驱

printf("cur\_e的前驱为%d\n", pre\_e);

else printf("cur\_e的前驱不存在\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 9:

if (L.elem)//若表存在

{

ElemType cur\_e, next\_e;

printf("输入一个数字，为cur\_e赋值\n");

scanf("%d", &cur\_e);

getchar();

if (NextElem(L, cur\_e, next\_e) == 1)//如果cur\_e存在后继

printf("cur\_e的后继为%d\n", next\_e);

else printf("cur\_e的后继不存在\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 10:

if (L.elem)//若表存在

{

printf("输入要插入的位置\n");

scanf("%d", &i);

getchar();

if (i >= 1 && i <= ListLength(L) + 1)//i值合法

{

ElemType e;//插入的元素

printf("输入一个数字，为e赋值\n");

scanf("%d", &e);

getchar();

if (ListInsert(L, i, e))

printf("插入成功\n");

else printf("插入失败\n");

}

else printf("位置i不合法\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 11:

if (L.elem)//表存在

{

printf("输入要删除的数据元素位序\n");

scanf("%d", &i);

getchar();

if (i >= 1 && i <= ListLength(L))

{

ElemType e;//删除的元素存放在e中

getchar();

if (ListDelete(L, i, e))

{

printf("删除成功\n");

printf("删除的元素值为%d", e);

}

else printf("删除失败\n");

}

else printf("位置i不合法\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 12:

if (L.elem)//若表存在

{

if (!ListTrabverse(L, visit)) printf("线性表是空表！\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 13:

if (L.elem)//若表存在

{

if (Loaddata(L))

printf("数据加载成功\n");

else printf("数据加载失败\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 14:

if (L.elem)//若表存在

{

if (Savedata(L))

printf("数据保存成功\n");

else printf("数据保存失败\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

/\*--------------------page 23 on textbook ----------------------------------\*/

status InitiaList(SqList & L){

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

if(!L.elem) exit(OVERFLOW);//如果内存不足创建失败，则异常退出

L.length=0;

L.listsize=LIST\_INIT\_SIZE;

return OK;//创建成功

}

status DestroyList(SqList & L) {

free(L.elem);//释放空间

L.elem = NULL;//表址指向空

L.length = L.listsize = 0;//表厂及表大小置0

return TRUE;//销毁成功

}

status ClearList(SqList & L) {

L.length = 0;//置表长为0

return TRUE;

}

status ListEmpty(SqList L) {

if (L.length == 0)

return TRUE;//如果L为空表(即表长==0)，返回TRUE;

else return FALSE;//否则返回FALSE;

}

status ListLength(SqList L) {

return L.length;//返回元素个数

}

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

e = L.elem[i - 1];//使用取址公式定位第i个元素并赋值给e

return OK;

}

status LocatElem(SqList L, ElemType e, status(\*compare)(ElemType x, ElemType y))

{

int i;

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

{

if (compare(L.elem[i], e) == 0)//compare函数比较两个参数，相同返回0

return i + 1;//返回第一个与e满足关系compare的数据元素的位序

}

return FALSE;//不存在返回0

}

status PriorElem(SqList L, ElemType cur, ElemType & pre\_e) {

int order = LocatElem(L, cur, compare);//查找cur元素获取其序号（cur不在表中order为0）

if (order > 1)//如果cur在L中且不为第一个元素

{

pre\_e = L.elem[order - 2];//cur的前驱值赋给&pre\_e

return TRUE;

}

return FALSE;

}

status NextElem(SqList L, ElemType cur, ElemType & next\_e) {

int order = LocatElem(L, cur, compare);//查找cur元素获取其序号

if (order >= 1 && order<L.length)//如果cur在L中且不为最后一个元素

{

next\_e = L.elem[order];//cur的后继值赋给&next\_e

return TRUE;

}

return FALSE;

}

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

if (L.length + 1 >= L.listsize)//空间已满，增配空间

{

ElemType \* newbase;

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

if (newbase == NULL)

return FALSE;//分配失败

L.elem = newbase;//新基址

L.listsize += LISTINCREMENT;//增加容量

}

//i之后的元素依次后移

ElemType \*in\_i = &(L.elem[i - 1]);//in\_i指向将要插入的位置

ElemType \*cur;

for (cur = &(L.elem[L.length - 1]); cur >= in\_i; cur--)

\*(cur + 1) = \*cur;

\*in\_i = e;//i处插入e

L.length++;//表长加1

return TRUE;

}

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

{

e = L.elem[i - 1];//赋值e

ElemType \*tail = &L.elem[L.length - 1];//tail指向表尾的位置

ElemType \*cur;

for (cur = &(L.elem[i - 1]); cur < tail; cur++)

\*cur = \*(cur + 1);//i之后的元素依次前移

L.length--;//表长减1

return TRUE;

}

status ListTrabverse(SqList L, status(\*visit)(ElemType e)) {

int i;

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

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

{

if (!visit(L.elem[i])) return ERROR;//如果visit失败，则退出

}

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

return L.length;//遍历完毕，返回遍历的元素个数

}

status compare(ElemType x, ElemType y)

{/\*通过内存比较判断x和y元素,相等返回0，x<y返回0-，x>y返回0+\*/

return memcmp(&x, &y, sizeof(ElemType));

}

status Loaddata(SqList &L)

{

FILE \*fp;

L.length = 0;

if ((fp = fopen(FILENAME, "r")) == NULL)//文件打开失败

{

printf("File open error\n ");

return FALSE;

}

while (fread(&L.elem[L.length], sizeof(ElemType), 1, fp))//每次读取一个数据元素，顺序写入打顺序表中

{

if (L.length + 1 >= L.listsize)//空间已满，增配空间

{

ElemType \* newbase;

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

if (newbase == NULL)

return FALSE;//分配失败

L.elem = newbase;//新基址

L.listsize += LISTINCREMENT;//增加容量

}

L.length++;//准备写入下一个元素

}

fclose(fp);

return TRUE;//写入完毕

}

status Savedata(SqList L)

{

FILE \*fp;

if ((fp = fopen(FILENAME, "w")) == NULL)//打开文件失败

{

printf("File open erroe\n ");

return FALSE;

}

fwrite(L.elem, sizeof(ElemType), L.length, fp);//从地址L.elem开始，往后sizeof(ElemType)\*L.length个字节的数据一次性写入到文件中

fclose(fp);//关闭文件

return TRUE;

}

status visit(ElemType e)

{

printf("%d ", e);

return TRUE;

}

# 附录B 基于链式存储结构线性表实现的源程序

/\*含头结点的链式存储结构的线性表实现\*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include<string.h>

/\*--------------------page 10 on textbook ----------------------------------\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define FILENAME\_L1 "dataL1"

#define FILENAME\_L2 "dataL2"

typedef int status;

typedef int Elemtype;

typedef struct lnode

{

Elemtype data;

lnode \* next;

}Lnode; //数据元素类型定义

typedef Lnode\* SqList;//头指针

status IntiaList(SqList \*L);//L为指向头指针的指针

status DestroyList(SqList \* L);

status ClearList(SqList \*L);

status ListEmpty(SqList L);

int ListLength(SqList L);

status GetElem(SqList L, int i, Elemtype & e);

status LocateElem(SqList L, Elemtype e, status(\*compare)(Elemtype \*x, Elemtype \*y));

status PriorElem(SqList L, Elemtype cur, Elemtype & pre\_e);

status NextElem(SqList L, Elemtype cur, Elemtype & next\_e);

status ListInsert(SqList L, int i, Elemtype e);

status ListDelete(SqList L, int i, Elemtype & e);

status ListTrabverse(SqList L, status(\*visit)(Elemtype e));

status Loaddata(SqList \*L, char \*filename);//加载数据

status Savedata(SqList L, char \*filename);//保存数据

status visit(Elemtype e);

status compare(Elemtype \*x, Elemtype \*y);

SqList \* list\_verify(SqList \*L1, SqList \*L2);

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

void main(void) {

SqList L1, L2;

SqList \*L;

L1 = L2 = NULL;

int op = 1;

int i;

while (op) {

system("cls");

printf("\n\n");

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

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

printf(" 1. IntiaList 7. LocateElem\n");

printf(" 2. DestroyList 8. PriorElem\n");

printf(" 3. ClearList 9. NextElem \n");

printf(" 4. ListEmpty 10. ListInsert\n");

printf(" 5. ListLength 11. ListDelete\n");

printf(" 6. GetElem 12. ListTrabverse\n");

printf(" 13. 从文件中读取数据\n");

printf(" 14. 保存数据至文件\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~14]:");

scanf("%d", &op);

getchar();

L = list\_verify(&L1, &L2);//确认需要操作的表

switch (op) {

case 1:

if (\*L == NULL)//若线性表不存在，则创建

{

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

else printf("线性表创建失败！\n");

//printf("%p", (\*L)->next);

}

else printf("线性表已存在\n");

getchar(); getchar();

break;

case 2:

if (\*L)//若表存在

{

DestroyList(L);

printf("表销毁成功\n");

//printf("%p", \*L);

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 3:

if (\*L)//若表存在

{

ClearList(L);

printf("表清空成功\n");

//printf("%p", \*L);

//printf("%p", (\*L)->next);

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 4:

if (\*L)//若表存在

{

if (ListEmpty(\*L))

printf("线性表为空\n");

else printf("线性表不为空\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 5:

if (\*L)//若表存在

{

int len;

len = ListLength(\*L);

printf("表长为%d\n", len);

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 6:

if (\*L)//若表存在

{

Elemtype e;

printf("输入元素序号\n");

scanf(" %d", &i);

getchar();

if (i >= 1 && i <= ListLength(\*L))//若i值有效

{

GetElem(\*L, i, e);//获取位序为i的元素

printf("位序%d的元素值为:", i); visit(e);//输出获取的元素值

}

else printf("i值无效\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 7:

if (\*L)//若表存在

{

Elemtype e;

printf("输入一个数字，为e赋值\n");

scanf("%d", &e);

getchar();

//输出第一个与e满足compare关系的元素位序，不存在位序为0

int order = LocateElem(\*L, e, compare);

if (order == 0)

printf("表中不存在与e相同的元素\n");

else printf("第一个与e满足compare关系的元素位序是%d\n", order);

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 8:

if (\*L)//若表存在

{

Elemtype cur\_e, pre\_e;

printf("输入一个数字，为cur\_e赋值\n");

scanf("%d", &cur\_e);

getchar();

if (PriorElem(\*L, cur\_e, pre\_e) == 1)//如果cur\_e存在前驱

{

printf("cur\_e的前驱为:"); visit(pre\_e);

}

else printf("cur\_e的前驱不存在\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 9:

if (\*L)//若表存在

{

Elemtype cur\_e, next\_e;

printf("输入一个数字，为cur\_e赋值\n");

scanf("%d", &cur\_e);

getchar();

if (NextElem(\*L, cur\_e, next\_e) == 1)//如果cur\_e存在后继

{

printf("cur\_e的后继为:"); visit(next\_e);

}

else printf("cur\_e的后继不存在\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 10:

if (\*L)//若表存在

{

printf("输入要插入的位置\n");

scanf("%d", &i);

getchar();

if (i >= 1 && i <= ListLength(\*L) + 1)//i值合法

{

Elemtype e;//插入的元素

printf("输入一个数字，为e赋值\n");

scanf("%d", &e);

getchar();

if (ListInsert(\*L, i, e))

printf("插入成功\n");

else printf("插入失败\n");

}

else printf("位置i不合法\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 11:

if (\*L)//若表存在

{

printf("输入要删除的数据元素位序\n");

scanf("%d", &i);

getchar();

if (i >= 1 && i <= ListLength(\*L))//若i值合法

{

Elemtype e;//保存删除的结点数据

if (ListDelete(\*L, i, e))

{

printf("删除成功\n");

printf("删除的元素值为:"); visit(e);

}

else printf("删除失败\n");

}

else printf("位置i不合法\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 12:

if (\*L)//若表存在

{

if (!ListTrabverse(\*L, visit)) printf("线性表是空表！\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 13:

if (\*L == NULL || (\*L)->next == NULL)//若表不存在或表为空，则加载数据

{

int status;

if (\*L == L1)//选择L1

status = Loaddata(L, FILENAME\_L1);

else if (\*L == L2)//选择L2

status = Loaddata(L, FILENAME\_L2);

if (status)

printf("数据加载成功\n");

else printf("数据加载失败\n");

}

else printf("表已存在，请销毁或清空后加载数据\n");

getchar(); getchar();

break;

case 14:

if (\*L)//若表存在

{

int status;

if (\*L == L1)//选择L1

status = Savedata(\*L, FILENAME\_L1);

else if (\*L == L2)//选择L2

status = Savedata(\*L, FILENAME\_L2);

if (status)

printf("数据保存成功\n");

else printf("数据保存失败\n");

}

else printf("表不存在\n");

getchar(); getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

/\*--------------------page 23 on textbook ----------------------------------\*/

status IntiaList(SqList \* L) {

\*L = (Lnode \*)malloc(sizeof(Lnode));//头指针申请一个存储结点

if (\*L)//如果分配空间成功

(\*L)->next = NULL;//头结点指针域置空

else exit(OVERFLOW);//分配失败

return OK;

}

status DestroyList(SqList \* L)

{

Lnode \* L1;

Lnode \* temp;//存储上一个释放节点的next值

L1 = \*L;//L1用来遍历链表

for (L1 = \*L; L1 != NULL; L1 = temp)

{

temp = L1->next;//保存next值

free(L1);//依次释放线性表的内存空间

}

\*L = NULL;//头指针置空

return OK;

}

status ClearList(SqList \*L)

{

Lnode \* L1;

Lnode \* temp;//存储上一个释放节点的next值

L1 = \*L;//L1用来遍历链表

for (L1 = (\*L)->next; L1 != NULL; L1 = temp)//释放头结点之后的结点

{

temp = L1->next;//保存next值

free(L1);//依次释放线性表的内存空间

}

(\*L)->next = NULL;//头结点指针域置空

return OK;

}

status ListEmpty(SqList L)

{

if (L->next == NULL)//若头结点的指针域为空

return TRUE;

else return FALSE;

}

int ListLength(SqList L)

{

Lnode \*p;

int count;

for (p = L->next, count = 0; p != NULL; p = p->next)

count++;//p每后移一次，表长加1

return count;

}

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

{

Lnode \*p;

int count;

for (p = L->next, count = 1; count != i; p = p->next, count++);//定位第i个数据元素

e = p->data;//赋值

return OK;

}

status LocateElem(SqList L, Elemtype e, status(\*compare)(Elemtype \*x, Elemtype \*y))

{

Lnode \*p;

int count;

for (p = L->next, count = 1; p != NULL; p = p->next, count++)//从首节点开始比较

if (compare(&(p->data), &e) == 0)

return count;

return FALSE;//遍历完毕，未找到

}

status PriorElem(SqList L, Elemtype cur, Elemtype & pre\_e)

{

Lnode \*p;

Lnode \*pr;//p的前驱

p = L->next;//p指向首节点

if (p == NULL || p->data == cur)//如果链表为空或cur是首节点，不存在前驱

return FALSE;

pr = p;//pr指向首节点

p = p->next;//p指向第二结点

while (p != NULL)

{

if (p->data == cur)//如果cur在表中

{

pre\_e = pr->data;//赋值

return TRUE;

}

pr = p;

p = p->next;

}

return FALSE;//表中未找到cur，cur无前驱

}

status NextElem(SqList L, Elemtype cur, Elemtype & next\_e)

{

Lnode \*p;//p

p = L->next;//p指向首节点

if (p == NULL)//如果链表为空

return FALSE;

while (p != NULL)

{

if (p->data == cur)//如果cur在表中

break;

p = p->next;

}

if (!p || !(p->next))//cur不在链表中或cur为最后一个结点

return FALSE;

else {

next\_e = (p->next)->data;//赋值

return TRUE;

}

}

status ListInsert(SqList L, int i, Elemtype e)

{

int count;//记录当前位置

Lnode \*p;//使用p遍历链表查找位置i

Lnode \*q;

for (p = L, count = 0; count < i - 1; p = p->next, count++);//定位，执行完毕后p指向位置i的前驱

q = (Lnode \*)malloc(sizeof(Lnode));//为插入的结点声明空间

q->data = e;//将e的数据域赋值给q的数据域

q->next = p->next;//插入结点

p->next = q;

return OK;

}

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

{

Lnode \*p, \*q;

int count;

for (p = L, count = 0; count < i - 1; p = p->next, count++);//定位，执行完毕后p指向位置i的前驱

q = p->next;//q指向删除结点

e = q->data;//赋值

p->next = q->next;//更改指针域

free(q);//删除结点

return OK;

}

status ListTrabverse(SqList L, status(\*visit)(Elemtype e)) {

if (L->next == NULL)

return 0;

Lnode \* p;

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

for (p = L->next; p != NULL; p = p->next)//使用visit访问首结点及其之后的结点

visit(p->data);

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

return 1;

}

status Loaddata(SqList \*L, char \*filename)

{

FILE \*fp;

if ((fp = fopen(filename, "r")) == NULL)//文件打开失败

{

printf("File open error\n ");

return FALSE;

}

if (\*L == NULL)//若表不存在，则创建表及头结点

\*L = (Lnode \*)malloc(sizeof(Lnode));

(\*L)->next = NULL;//头结点指针域置空

Lnode newEle;

Lnode \*p, \*q;

q = \*L;//q指向将要插入位置的前驱结点

while (fread(&newEle, sizeof(Lnode), 1, fp))//每次读取一个数据元素

{

p = (Lnode \*)malloc(sizeof(Lnode));//动态声明一个结点，作为新结点插入到链表中

\*p = newEle;//为新节点赋值

//插入结点

p->next = q->next;

q->next = p;

q = p;//q后移一个结点，进行下一次的插入

}

fclose(fp);

return TRUE;//写入完毕

}

status Savedata(SqList L, char \*filename)

{

FILE \*fp;

if ((fp = fopen(filename, "w")) == NULL)//打开文件失败

{

printf("File open error\n ");

return FALSE;

}

Lnode \*p;

p = L->next;//p指向首结点

while (p)

{

fwrite(p, sizeof(Lnode), 1, fp);//以二进制读写一次写入一个结点的数据

p = p->next;

}

fclose(fp);//关闭文件

return TRUE;

}

status visit(Elemtype e)

{

printf("%d ", e);

return TRUE;

}

status compare(Elemtype \*x, Elemtype \*y)

{

return (\*x) - (\*y);//相同返回0

}

SqList \* list\_verify(SqList \*L1, SqList \*L2)

{

char Listname[9];

do {

printf("\n 选择需要操作的表[L1,L2]: ");

scanf("%s", Listname);

} while (strcmp(Listname, "L1") && strcmp(Listname, "L2"));//只能在L1，L2中选择

SqList \*plist;

if (strcmp(Listname, "L1") == 0)

plist = L1;

else plist = L2;//plist为用户所选的表

return plist;

}

# 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