

**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201806**

**学 号： U201813676**

**姓 名： 刘 汉 鹏**

**指导教师： 许贵平**

**报告日期： 2020年 10月 16 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](#_Toc53772003)

[1.1 问题描述 2](#_Toc53772004)

[1.1.1 线性表的基本运算定义 2](#_Toc53772005)

[1.2 系统设计 3](#_Toc53772006)

[1.3 系统实现 7](#_Toc53772007)

[**1.4 系统测试** 7](#_Toc53772008)

[1.5 实验小结 7](#_Toc53772009)

[参考文献 8](#_Toc53772010)

[附录A 基于顺序存储结构线性表实现的源程序 9](#_Toc53772011)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

采用顺序表作为线性表的物理结构，实现线性表的基本运算，具体运算描述见**1.1.1**。其中ElemType为数据元素的类型名，具体含义为int。

要求构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

## 1.1.1 线性表的基本运算定义

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)；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()。

(13)线性表读写文件:函数名称是SaveList(L,FileName[]),初始条件是线性表L已经存在，FileName已知；操作结果是将L中的元素依次写到FileName文件中。

(14)增加一个新线性表:函数名称是AddList(Lists,ListName[]),初始条件是多线性表Lists已经存在；操作结果是Lists新增一个线性表，名字是ListName.

(15)移出一个线性表:函数名称是RemoveList(Lists,ListName[]),初始条件是多线性表Lists已经存在，ListName已知；操作结果是将ListName从多线性表中删除。

(16)查找线性表:函数名称是Locate(Lists,ListName[]),初始化条件是多线性表Lists已经存在，ListName已知；操作结果是返回ListName的位置。

## 1.2 系统设计

（1）ListLength(L)：图1-1

输入：线性表的表名L

输出：线性表的长度length

算法处理步骤：1）输入表名

2）(a)判断表是否存在，若不存在，返回INFEASIBLE；

（b）若表存在，对表进行遍历，当表的元素存在时，length++；

3）返回表长

算法复杂度：时间复杂度：O（n）

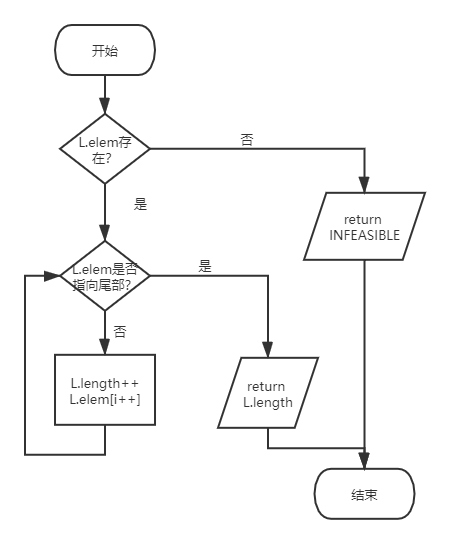
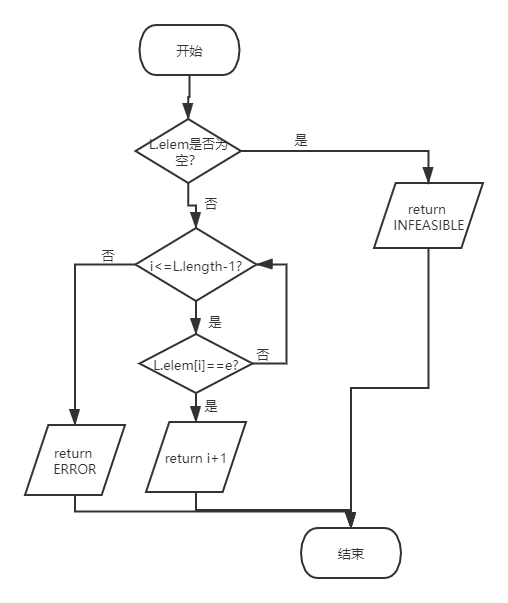
 

图1-1 ListLength(L) 图1-2 LocateElem(L,e)

（2）LocateElem(L,e)：图1-2

输入：线性表的表名L，想要确定位置的元素e

输出：元素e的位置

算法处理步骤：1)输入表名L，元素e

2）a)线性表L是否存在，若不存在，则返回INFEASIBLE；

b)若存在，从表头开始，对表进行遍历，判断L.elem[i]与元素e是否相等，若相等，则返回元素e的位置i+1；若不等，则将i+1,直到i指向表尾。

3）返回元素e的位置i+1.

算法复杂度：时间复杂度O（n）

（3）ListInsert(L,i,e):图1-3

输入：线性表的表名L，插入的位置i，插入的元素e

输出：插入元素后的表L

算法处理步骤：1）输入线性表的表名L，插入的位置i，插入的元素e；

2）判断线性表L是否不存在，若不存在，返回INFEASIBLE；

3）判断插入的位置i是否在1～length+1之间，若不是，则返回ERROR；

4）判断表L是否为满表，若是，则将表容一位；

5）从表尾开始将每个元素向后移动一位，直到第i个元素，将第i个元素的值赋为e，表长加1，返回OK。

算法复杂度：时间复杂度：O（n）

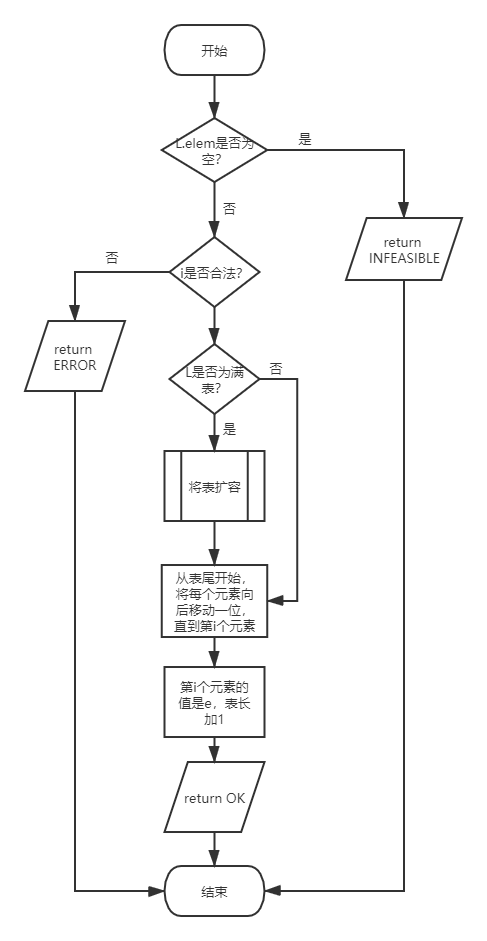
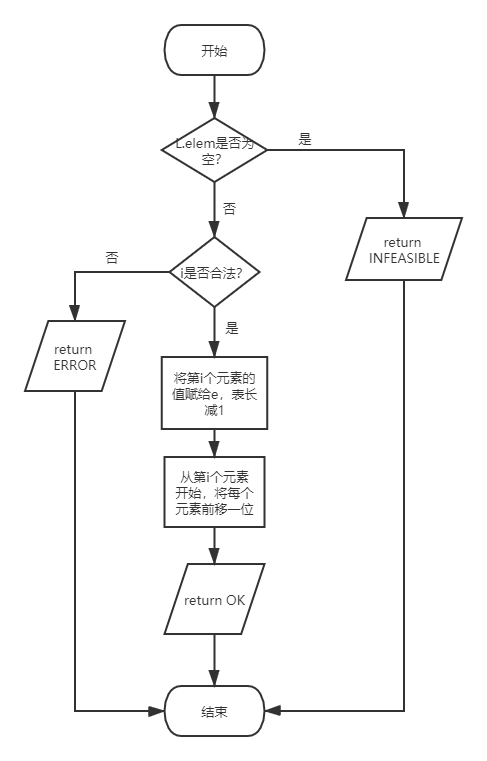
 

图1-3 ListInsert(L,i,e) 图1-4 ListDelete（L,i,e）

（4）ListDelete（L,i,e）:图1-4

输入：线性表的表名L，删除的位置i

输出：删除的元素e

算法处理步骤：1）输入线性表的表名L，删除的位置i

2）判断线性表L是否不存在，若不存在，返回INFEASIBLE；

3)判断删除的位置i是否在1～length之间，若不是，则返回ERROR；

4）将第i个元素的值赋给e，表长减1，从第i+1个元素开始，将每个元素向前移动一位，返回OK。

算法复杂：时间复杂度：O(n)

(5)AddList(Lists,ListName[]):图1-5

输入：要增加的表名ListName

输出：增加一个线性表后的总表

算法处理步骤：1)读取表的表名ListName

2)初始化一个表名为ListName的线性表，总表的长度加1

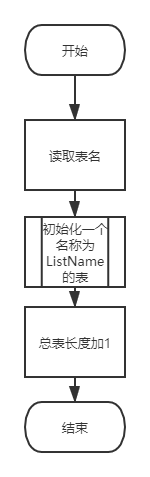
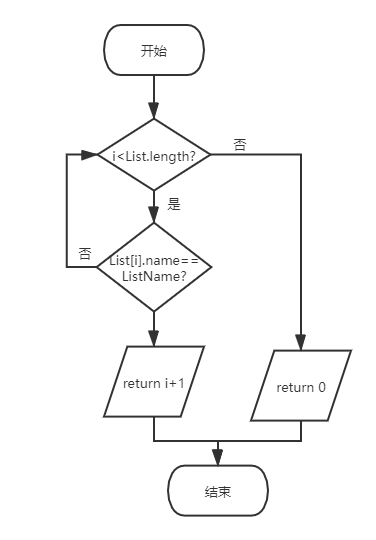
 

图1-5 AddList(Lists,ListName[]) 图1-6 LocateList(Lists,ListName[])

(6)LocateList(Lists,ListName[]):图1-6

输入：要确定位置的表名ListName

输出：线性表在总表的位置i

算法处理步骤：从表头开始对总表进行遍历，判断线性表的名称是否为ListName,若是则返回i+1,否则，返回 0

将封装好的子函数写入头文件中，然后利用swtich{case n:… break;}的结构，将程序进行可视化操作。（如图1-7）

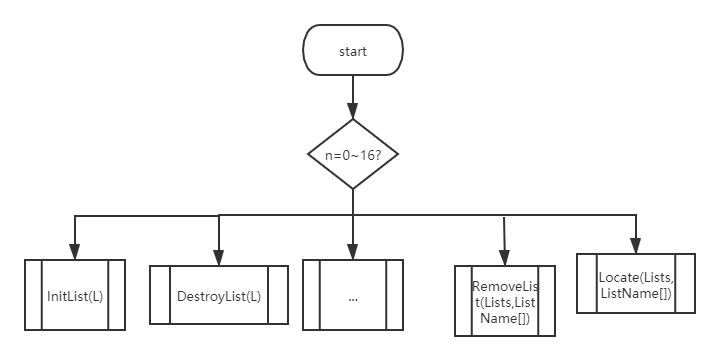


图1-7线性表操作演示流程示意图

## 1.3 系统实现

程序实现采用的集成环境是Visual Studio 2019,使用了C语言标准头文件stdio.h，主要采用了顺序编程结构。

**1.4 系统测试**

系统测试挑选了几个主要的程序进行了测试，测试结果如表一。经过在线下对所有函数的测试，所有函数对正确的输入有正确的输出，对错误的输入也有相应的报错提醒。

表1-1线性表测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 程序名称 | 程序输入 | 理论结果 | 运行结果 |
| ListInsert | 7,1 | 表头的元素是7 |  |
| ListInsert | 6,6 | 在表尾插入了6 |  |
| ListInsert | 9,3 | 第三个元素是9 |  |
| ListDetle | 3 | 返回的元素是9 |  |
| AddList | 湖北 | 多线性表存在一个名为湖北的线性表 |  |
| LocateList | 湖北 | 返回的位置为1 |  |
| RemoveList | 湖北 | 线性表存在，删除成功 |  |

## 1.5 实验小结

这次实验主要学习了如何对线性表进行一些基本的操作，如：初始化、插入、删除和查询等；以及进一步学会了利用多线性表对线性表进行集中管理，对线性表这种结构体有了更深的理解和认识；此次试验中没有单独编写头文件，希望能够吸取教训在接下来实验中有所改进。

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

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

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

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

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define INFEASIBLE -1

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;

typedef struct { //线性表的管理表定义

struct {

char name[30];

SqList L;

} elem[10];

int length;

int listsize;

}LISTS;

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

status InitList(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 LocateElem(SqList L,ElemType e); //简化过

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); //简化过

status InitLists(LISTS& Lists);

status workLists(LISTS Lists, char ListName[], SqList& L);

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

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

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

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

int main(void) {

LISTS Lists;

int e;

int n;

int i;

SqList L; int op = 1;

while (op) {

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

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

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

printf(" 1. InitList 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. ListTraverse\n");

printf(" 13. InitLists 14. ADDLists\n");

printf(" 15. RemoveList 16. LocateList\n");

printf(" 17. workLists\n");

printf(" 0. Exit\n");

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

printf("请选择你的操作[0~17]:");

scanf("%d", &op);

switch (op) {

case 1:

//printf("\n----IntiList功能待实现！\n");

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

else printf("线性表创建失败！\n");

getchar(); getchar();

break;

case 2:

//printf("\n----DestroyList功能待实现！\n");

if (DestroyList(L) == OK) printf("线性表销毁成功\n");

else printf("线性表销毁失败\n");

getchar(); getchar();

break;

case 3:

//printf("\n----ClearList功能待实现！\n");

if (ClearList(L) == OK) printf("clear ok\n");

else printf("clear error\n");

getchar(); getchar();

break;

case 4:

//printf("\n----ListEmpty功能待实现！\n");

if (ListEmpty(L) == OK) printf("ListEmpty OK\n");

else printf("ListEmpty error\n");

getchar(); getchar();

break;

case 5:

//printf("\n----ListLength功能待实现！\n");

{

if (( ListLength(L)) != FALSE) printf("ListLength=%d\n",L.length);

else printf("ListLength error\n");

getchar(); getchar();

break;

}

case 6:

{

//printf("\n----GetElem功能待实现！\n");

printf("请输入元素位置\n");

scanf("%d", &n);

if (GetElem(L, n, e) == OK) printf("第%d号元素是:%d\n", n, e);

else printf("GetElem error\n");

getchar(); getchar();

break;

}

case 7:

{

//printf("\n----LocateElem功能待实现！\n");

printf("输入你需要查找的元素\n");

scanf("%d", &e);

if ((i = LocateElem(L, e) )!= FALSE) printf("元素%d的位置为%d\n", e, i);

else printf("Locate error\n");

getchar(); getchar();

break;

}

case 8:

{

//printf("\n----PriorElem功能待实现！\n");

printf("输入你需要确定前驱的元素\n");

scanf("%d", &e);

int pre;

if (PriorElem(L, e, pre) != FALSE) printf("%d 的前驱是：%d\n", e, pre);

else printf("PriorElem error\n");

getchar(); getchar();

break;

}

case 9:

{

//printf("\n----NextElem功能待实现！\n");

printf("输入你要确定后继的元素\n");

scanf("%d", &e);

int next;

if (NextElem(L, e, next) != FALSE) printf("%d 的后继是：%d\n", e, next);

else printf("NextElem error\n");

getchar(); getchar();

break;

}

case 10:

{

//printf("\n----ListInsert功能待实现！\n");

printf("输入你要插入的元素\n");

scanf("%d", &e);

printf("输入你要插入的位置\n");

scanf("%d", &n);

if (ListInsert(L, n, e) != FALSE) printf("插入成功\n");

else printf("插入失败\n");

getchar(); getchar();

break;

}

case 11:

{

//printf("\n----ListDelete功能待实现！\n");

printf("输入你要删除的位置\n");

scanf("%d", &i);

if (ListDelete(L, i, e) != FALSE) printf("删除第%d号位置的元素是：%d\n", i, e);

else printf("删除失败\n");

getchar(); getchar();

break;

}

case 12:

{

//printf("\n----ListTraverse功能待实现！\n");

if (!ListTraverse(L)) printf("线性表是空表！\n");

getchar(); getchar();

break;

}

case 13:

{

if (InitLists(Lists) == OK) printf("Lists ok \n");

else printf("Lists error\n");

getchar(); getchar();

break;

}

case 14:

{

printf("输入你要插入的表的名字：");

char name[10] = { 0 };

scanf("%s", name);

if (AddList(Lists, name) != FALSE) printf("插入成功\n");

else printf("插入失败\n");

getchar(); getchar();

break;

}

case 15:

{

printf("输入你要删除的表的名字:");

char dname[10] = { 0 };

scanf("%s", dname);

if (RemoveList(Lists, dname) != FALSE) printf("delet ok \n");

else printf("delet erro\n");

getchar(); getchar();

break;

}

case 16:

{

printf("输入你要确定位置的表名:");

char qname[10] = { 0 };

scanf("%s", qname);

int q;

if ((q = LocateList(Lists, qname)) != FALSE) printf("位置是%d\n", q);

else printf("locate erro\n");

getchar(); getchar();

break;

}

case 17:

{

printf("输入你要操作的表的名字:");

char wname[10] = { 0 };

scanf("%s", wname);

if (workLists(Lists, wname, L) != FALSE) printf("work start\n");

else printf("work error\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 InitList(SqList& L) {

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

if (!L.elem) exit(OVERFLOW);

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

return OK;

}

/\*

status ListTraverse(SqList L) {

int i;

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

for (i = 0; i < L.length; i++) printf("%d ", L.elem[i]);

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

return L.length;

}

\*/

status DestroyList(SqList& L)

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

{

// 请在这里补充代码，完成本关任务

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

if (L.elem)

{

L.length = 0;

L.listsize = 0;

free(L.elem);

L.elem = NULL;

return OK;

}

else return INFEASIBLE;

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

}

status ClearList(SqList& L)

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

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

else

{

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

{

L.elem[i] = 0;

}

L.length = 0;

return OK;

}

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

}

status ListEmpty(SqList L)

// 如果线性表L存在，判断线性表L是否为空，空就返回TRUE，否则返回FALSE；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

else if (L.length == 0)

{

return TRUE;

}

else return FALSE;

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

}

status ListLength(SqList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

else

{

int i = 0;

while (L.elem[i] != 0)

{

i++;

L.length++;

}

return L.length;

}

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

}

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

// 如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK；如果i不合法，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

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

{

return ERROR;

}

else

{

e = L.elem[i - 1];

return OK;

}

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

}

status LocateElem(SqList L, ElemType e)

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

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

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

{

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

return i + 1;

}

return ERROR;

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

}

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

// 如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返回OK；如果没有前驱，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

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

{

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

{

if (i == 0)

return ERROR;

else

{

pre = L.elem[i - 1];

return OK;

}

}

}

return ERROR;

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

}

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

// 如果线性表L存在，获取线性表L元素e的后继，保存在next中，返回OK；如果没有后继，返回ERROR；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

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

{

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

{

if (i == L.length - 1)

return ERROR;

else

{

next = L.elem[i + 1];

return OK;

}

}

}

return ERROR;

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

}

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

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

{

// 请在这里补充代码，完成本关任务

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

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

int j;

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

return ERROR;

if (L.length >= L.listsize)

{

int\* newbase;

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

if (newbase == NULL) return ERROR;

L.elem = newbase;

L.listsize += LISTINCREMENT;

}

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

{

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

}

L.elem[i - 1] = e;

L.length++;

return OK;

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

}

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

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

{

// 请在这里补充代码，完成本关任务

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

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

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

int\* p = &(L.elem[i - 1]);

e = \*p;

int\* q = L.elem + L.length - 1;

for (++p; p <= q; ++p)

\*(p - 1) = \*p;

--L.length;

return OK;

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

}

status ListTraverse(SqList L)

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

{

// 请在这里补充代码，完成本关任务

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

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

if (L.length != 0)

{

for (int i = 0; i <= L.length - 2; i++)

{

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

}

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

}

return OK;

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

}

status SaveList(SqList L, char FileName[])

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

{

// 请在这里补充代码，完成本关任务

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

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

FILE\* pp;

pp = fopen(FileName, "wb");

if (pp == NULL) return ERROR;

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

{

fwrite(&L.elem[i], sizeof(int), 1, pp);

}

fclose(pp);

return OK;

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

}

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

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

{

// 请在这里补充代码，完成本关任务

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

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

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

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

FILE\* fr;

fr = fopen(FileName, "rb");

if (fr == NULL) return ERROR;

while (fread(&L.elem[L.length], sizeof(int), 1, fr))

{

L.length++;

}

fclose(fr);

return OK;

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

}

status InitLists(LISTS& Lists)

{

Lists.length = 0;

Lists.listsize = LISTINCREMENT;

return OK;

}

status workLists(LISTS Lists,char ListName[],SqList&L)

{

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

{

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

{

L = Lists.elem[i].L;

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

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

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

return OK;

}

}

return FALSE;

}

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

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

{

// 请在这里补充代码，完成本关任务

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

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

int\* newbase;

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

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

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

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

Lists.length++;

return OK;

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

}

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

// Lists中删除一个名称为ListName的线性表

{

// 请在这里补充代码，完成本关任务

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

int i = 0;

bool coms;

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

{

for (int j = 0; j <= 4; j++)

{

if (Lists.elem[i].name[j] == ListName[j])

coms = TRUE;

else

{

coms = FALSE;

break;

}

}

if (coms == TRUE)

break;

}

if (coms == FALSE)

return ERROR;

else

{

for (int k = i; k <= Lists.length - 2; k++)

{

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

}

Lists.length--;

return OK;

}

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

}

int LocateList(LISTS Lists, char ListName[])

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

{

// 请在这里补充代码，完成本关任务

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

int i = 0;

bool coms=FALSE;

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

{

for (int j = 0; j <= 4; j++)

{

if (Lists.elem[i].name[j] == ListName[j])

coms = TRUE;

else

{

coms = FALSE;

break;

}

}

if (coms == TRUE)

break;

}

if (coms == FALSE)

return 0;

else

return i + 1;

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

}