

**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（校交）1601**

**学 号： U201614345**

**姓 名： 刘文长**

**指导教师： 魏巍**

**报告日期： 2017年 11月 15 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

## 问题描述

实现基于顺序存储结构的线性表及以此为基础的多表管理和文件读写功能，按如下要求设计数据的逻辑结构及基本运算。

1.采用顺序表作为线性表的物理结构，实现§1.2的基本运算。其中ElemType为数据元素的类型名（本次实验中为int类型），其它有关类型和常量的定义和引用详见文献[1]的p10。

2．构造一个具有菜单的功能演示系统。系统可选择实现线性表的文件形式保存。 具体说明见1.1.3。

4.（附加）演示系统可选择实现多个线性表管理。设有专门切换线性表的切换功能以实现多表操作。

## 1.1.1 实验目的

（1）加深对线性表的概念、基本运算的理解

（2）熟练掌握线性表的逻辑结构与物理结构的关系

（3）物理结构采用顺序表,熟练掌握线性表的基本运算的实现。

## 1.1.2 线性表抽象数据类型ADT

数据对象:D = { ai|ai ∈ ElemSet, i = 1,2, ……,n, n≥0 }

数据关系:R1 = { <ai-1,ai>|ai-1,ai∈D, i = 2,……,n }

15个功能：

⑴初始化表：函数名称是InitaList(L)；初始条件是线性表L不存在已存在；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已存在且非空，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）读取文件中的线性表：函数名称是LoadFromFile(L)，初始条件是线性表L已经创建；操作结果是将文件中的数据写入当前线性表L。

（14）将当前线性表导出到文件：函数名称是ExportToFile(L)，初始条件是线性表线性表L已经创建，操作结果是将L中的数据写入文件中保存。

（15）多表操作，实现线性表的切换：函数名称是Switch\_List()，无初始条件，操作结果是从当前线性表切换到其它线性表进行操作。

1.1.3 演示系统与数据文件

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，对失败操作返回ERROR，并给出适当的操作提示显示，具体操作提示见数据测试模块。

演示系统可选择实现线性表的文件形式保存。其中，采用txt文件进行文件的读取和保存，文件的格式为一个“数字”+一个空格。文件读写方式采用fopen\_s函数（vs中认为fopen不安全）。

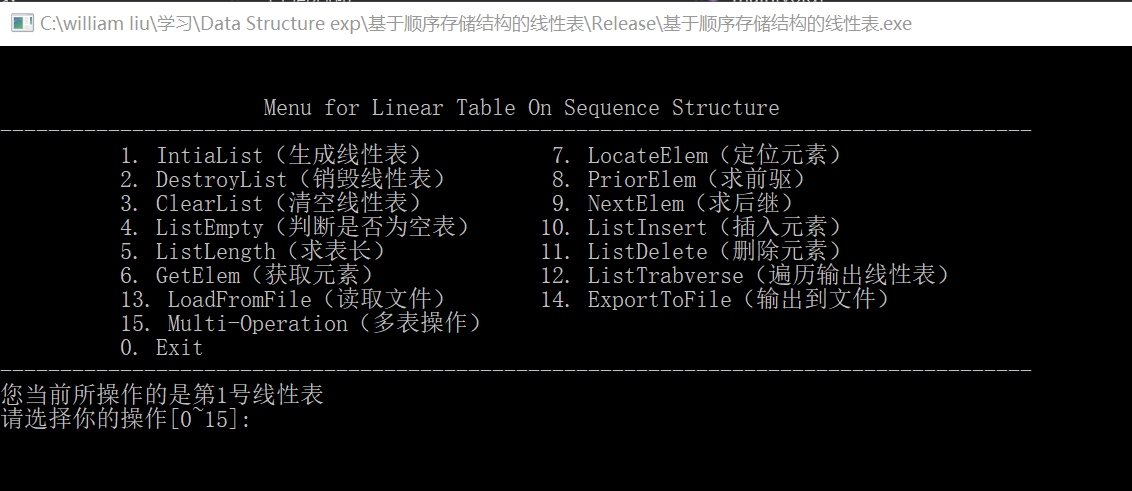


图1-1 演示系统菜单界面展示

## 系统设计

1.2.1 线性表的数据结构设计

typedef struct { //顺序表（顺序结构）的定义

ElemType \* elem; //指向线性表元素的线性存储区

int length;//表中元素的个数

int listsize;//表的最大存储空间

}SqList;

多表结构用数组SqList List[]来实现

本实验中ElemType定义为int

1.2.2 演示系统设计

演示系统由用户操作界面与功能调用部分组成。

用户操作界面输出可选的线性表操作项、操作的结果及错误信息，基于c语言的标准输入输出库(stdio.h)及控制台的cls命令实现。

功能调用部分将用户输入的有关信息传递给线性数据结构的操作函数进行调用，并对函数的返回值进行处理判断输出相应的提示信息。



图1-2 演示系统结构图

1.2.3 文件存储结构设计

考虑到简洁高效与直观实用的原则，按如下方式设计数据文件存储格式。

文件格式:txt，便于直接查看。

存放方式：以“data”+“空格”的方式，顺序存放数据值的ASCILL码

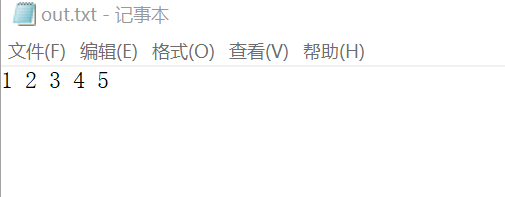


图1-3 文件存储格式

1.2.4 功能函数算法设计

1. InitiaList(&L)

**算法思想：**使用malloc函数分配LIST\_INIT\_SIZE个ElemType类型的存储空间并使L->elem指向这一段空间，若分配失败则返回ERROR，否则L->length=0,L->listsize=LIST\_INIT\_SIZE，返回OK。

**时空效率分析：**算法的时间复杂度为常量阶O(1)。

1. DestroyList(&L)

**算法思想：**首先判断全局变量：L.elem是否为空，若是则返回ERROR；否则使用free函数释放L->elem指向的存储空间，L->elem置空，返回OK。

**时空效率分析：**算法的时间复杂度为常量阶O(1)。

1. ClearList(&L)

**算法思想：**首先判断L->elem是否为空指针，若是则返回ERROR；将L.length设置为0，成功则返回OK，否则返回ERROR。

**时空效率分析：**算法的时间复杂度为常量阶O(1)。

1. ListEmplty(L)

**算法思想：**首先判断L->elem是否为空指针，若是则返回ERROR；然后判断若L.length=0，则返回TRUE，否则返回FALSE。

**时空效率分析：**算法的时间复杂度为常量阶O(1)。

1. ListLength(L)

**算法思想：**首先判断L->elem是否为空指针，若是则返回ERROR；否则返回L.length。

**时空效率分析：**算法的时空复杂度为常量阶O(1)。

1. GetElem(L, pos, &e)

**算法思想：**首先判断当L.elem是空指针时、表为空表时、当输入的位置pos小于等于零或大于表的长度+1时返回ERROR；然后将L.elem中第pos-1位的元素送入e所指的存储单元中，返回OK。

**时空效率分析：**算法的时间复杂度为常量阶O(1)。

1. LocateElem(L, e)

**算法思想：**首先判断当L.elem是空指针、L为空表时返回ERROR；然后对L.elem中的元素从第一个起开始查找，当找到第一个与e与给定元素相等时返回其序号，若遍历完所有元素仍未找到则返回ERROR。

**时空效率分析：**算法中有一层循环，循环执行的次数L->length不定，假设其值为n（表中有n个元素），且第一个相同的元素在表每个位置的概率相等（均为1/n）则算法的时间复杂度

O(n)==，为线性阶O(n)。

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

**算法思想：**首先判断当L.elem是空指针、表为空表时、cur是表头元素或cur不在表元素范围内时返回ERROR；然后对L.elem中的元素从第二个起开始查找，当找到第一个与cur相等的元素时将其前驱元素送入pre\_e所指的存储单元中，返回OK，若遍历完所有元素仍未找到则返回ERROR。

**时空效率分析：**算法中有一层循环，循环执行的次数L->length不定，假设其值为n（表中有n个元素），且第一个与e相等的元素在表每个位置的概率相等（均为1/(n-1)）则算法的时间复杂度

O(n)==，为线性阶O(n)。

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

**算法思想：**首先判断当L.elem是空指针、表为空表时、cur是表尾元素或cur不在表元素范围内时返回ERROR；然后对L.elem中的元素从第一个起开始查找，当找到第一个与cur相等的元素时将其后继元素送入next\_e所指的存储单元中，返回OK，若遍历完所有元素仍未找到则返回ERROR。

**时空效率分析：**算法中有一层循环，循环执行的次数L->length不定，假设其值为n（表中有n个元素），且第一个与e相等的元素在表每个位置的概率相等（均为1/(n-1)）则算法的时间复杂度

O(n)==，为线性阶O(n)。

1. ListInsert(&L, pos, e)

**算法思想：**流程图见后面的图1-4插入删除算法：将pos后的元素后移一位，再在指定位置插入元素，最后将表长+1**。**

**时空效率分析：**插入算法的时间耗费主要在移动元素上，循环执行的次数L->length-pos+1不定，假设表中有n个元素，且在每个位置插入的可能性相等(1/n+1)，则算法的时间复杂度

O(n)==，为线性阶O(n)。

1. ListDelete(&L, pos, &e)

**算法思想：**流程图见后面的图1-4插入删除算法：与插入算法类似，删除算法需要将位置pos以后的元素向前移一位，将第pos位的元素送到e存储单元中并使表长减一。

**时空效率分析：**假设表中有n个元素，且在每个位置删除的可能性相等(1/n)，则算法的时间复杂度

O(n)==，为线性阶O(n)。

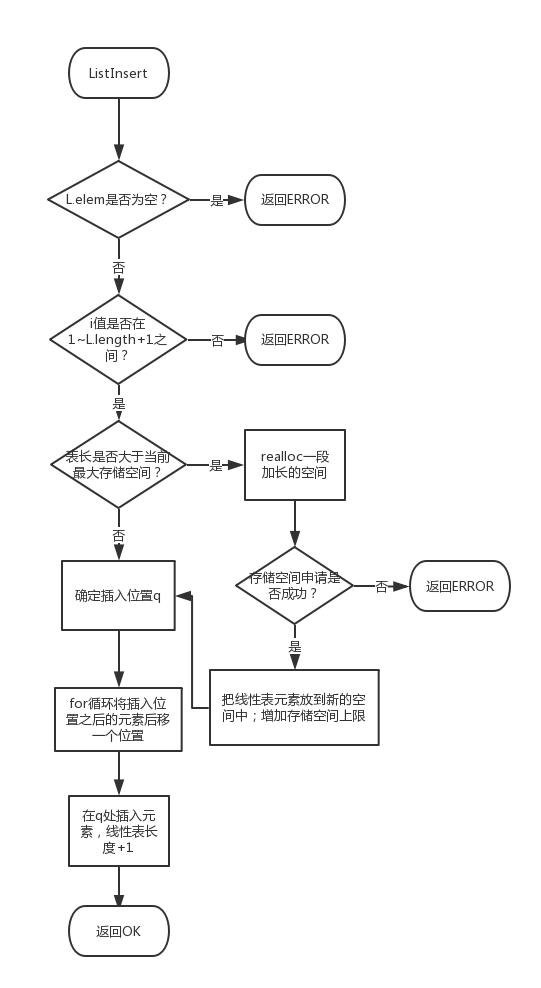
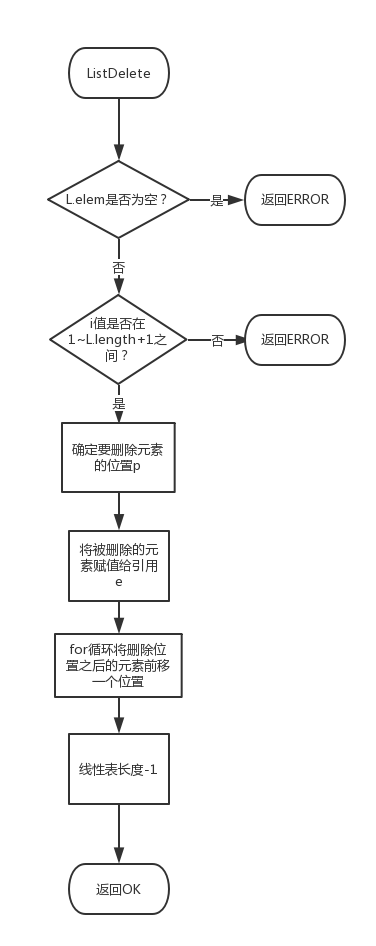


图1-4 插入删除算法

1. ListTrabverse(L)

**算法思想：**首先判断L->elem是否为空指针，若是则返回ERROR；否则从一号元素起对表中元素循环遍历输出，完成后返回OK。

**时空效率分析：** O(n)=。

1. 线性表的保存算法

**算法流程框图见下页图1-5。**

**时空效率分析：**假设表中有n个元素，则算法时空复杂度为O(n)线性阶。

1. 线性表的读取算法

**算法流程框图见下页图1-5。**

**时空效率分析：**假设导入的文件中有n个元素，由于是在表的末尾位置插入不需要移动元素所以算法的时间复杂度为线性阶O(n)。

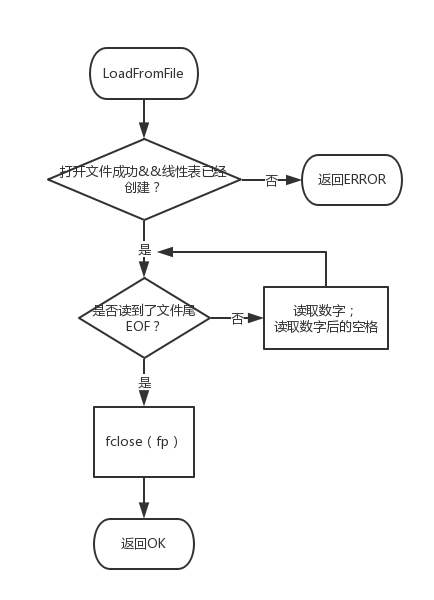


图1-5 文件保存、读取算法

## 1.3 系统实现

1.3.1 实验环境

实验代码使用Windows10和visual studio 2017 community编写与编译，解决方案名:基于顺序存储结构的线性表。

1.3.2 演示系统测试

**测试方案及预期结果：**包括给出正常数据及不合理数据测试系统响应

单表操作的已创建线性表测试默认已经读入out.txt中的线性表1 2 3 4 5 进行测试

测试方案表格如下：

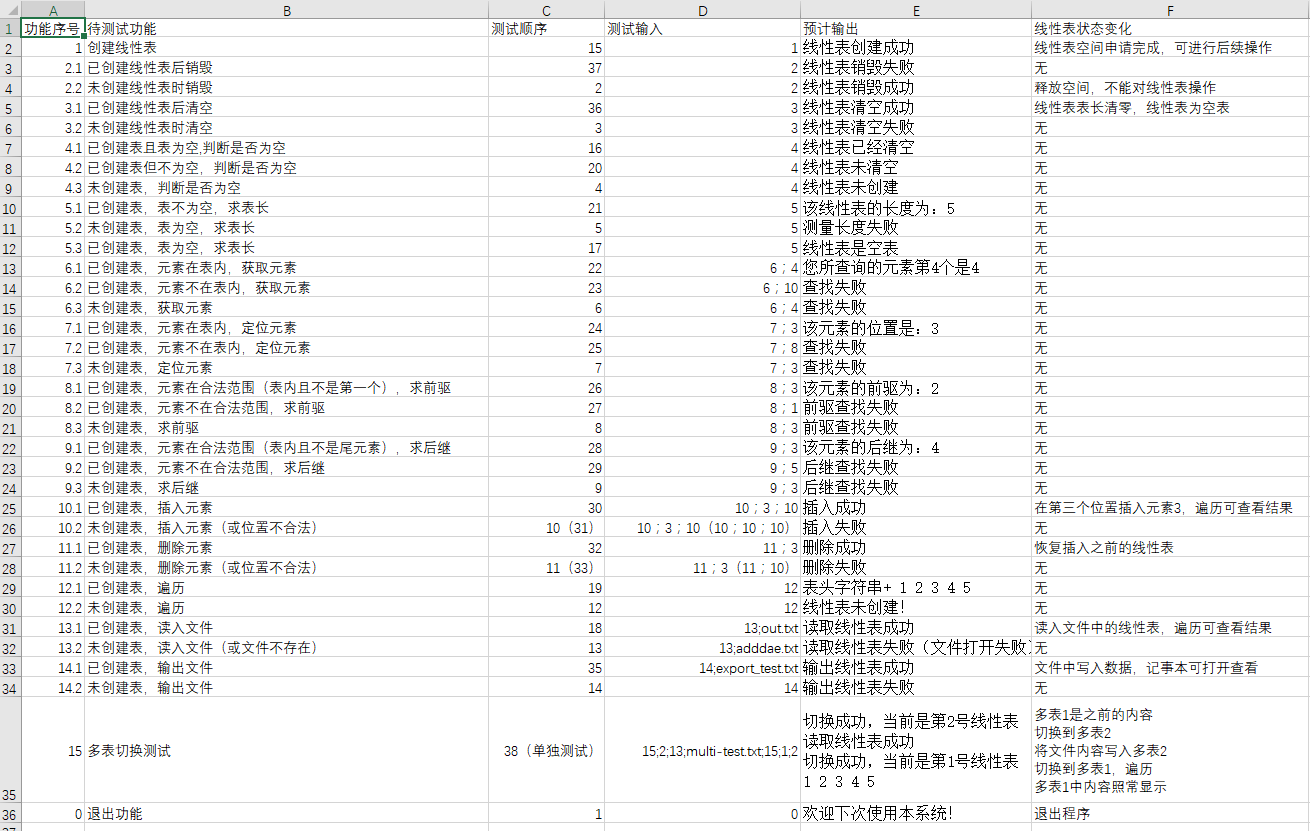


表1-1 测试方案表格

**实际结果截图：**

1. 退出功能：输入0，输出：欢迎下次使用本系统！ 并正常退出程序

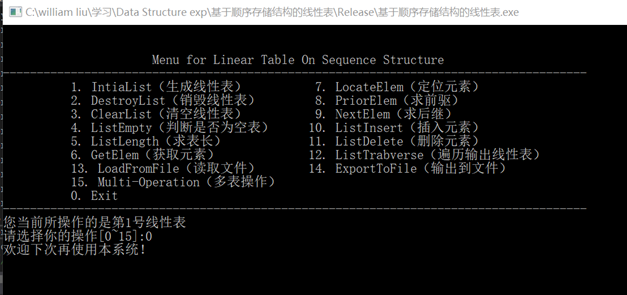


图1-6 退出功能截图

1. 创建线性表功能：输入1，输出：线性表创建成功 新创建的线性表为空表（若之前有数据，重新创建会覆盖之前数据）

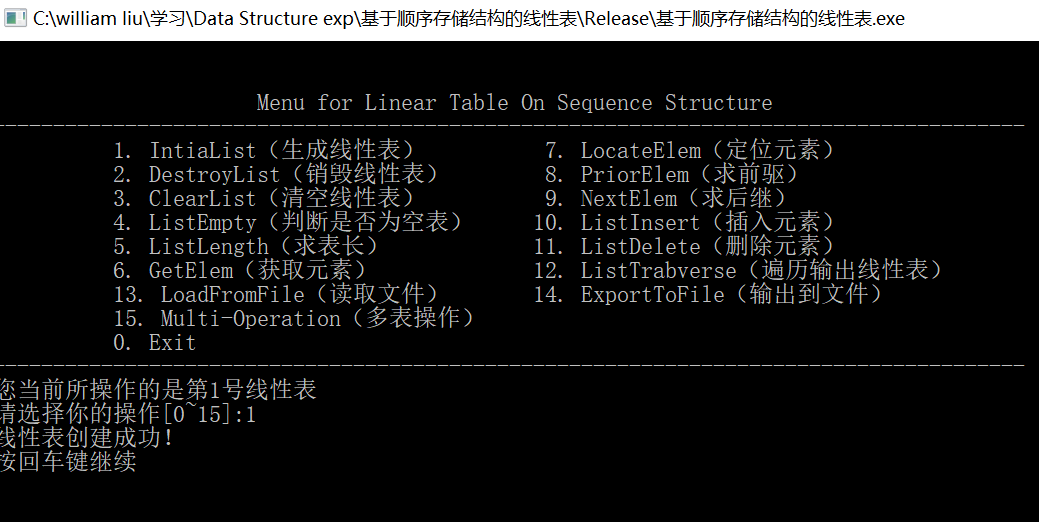


图1-7 测试1截图

2、销毁线性表功能：

（1）在创建过线性表时，输入2，输出：线性表销毁成功

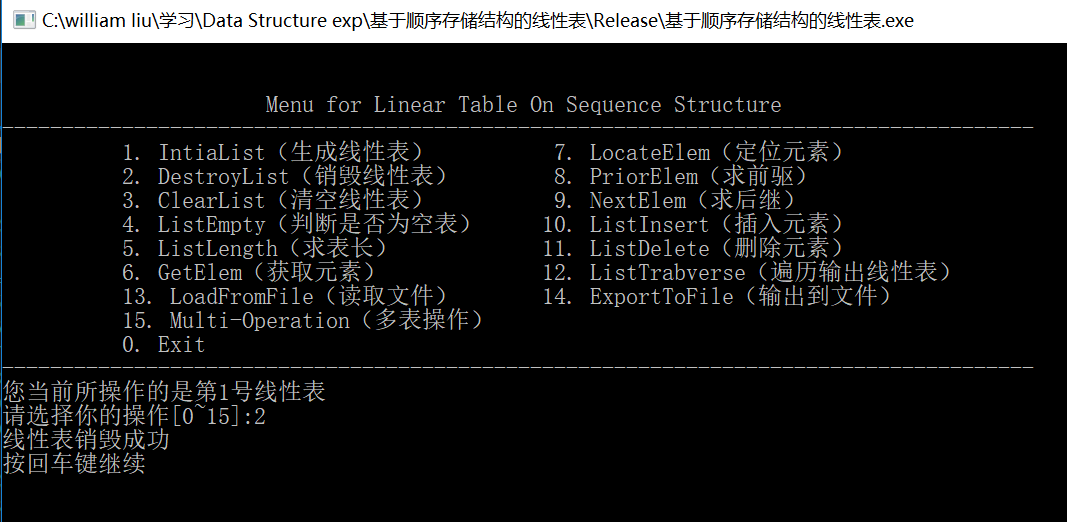


图1-8 测试2（1）截图

（2）在未创建过线性表时，输入2，输出：线性表销毁失败



图1-9 测试2（2）截图

3、清空线性表功能：

（1）已创建线性表，输入3，输出：线性表清空成功

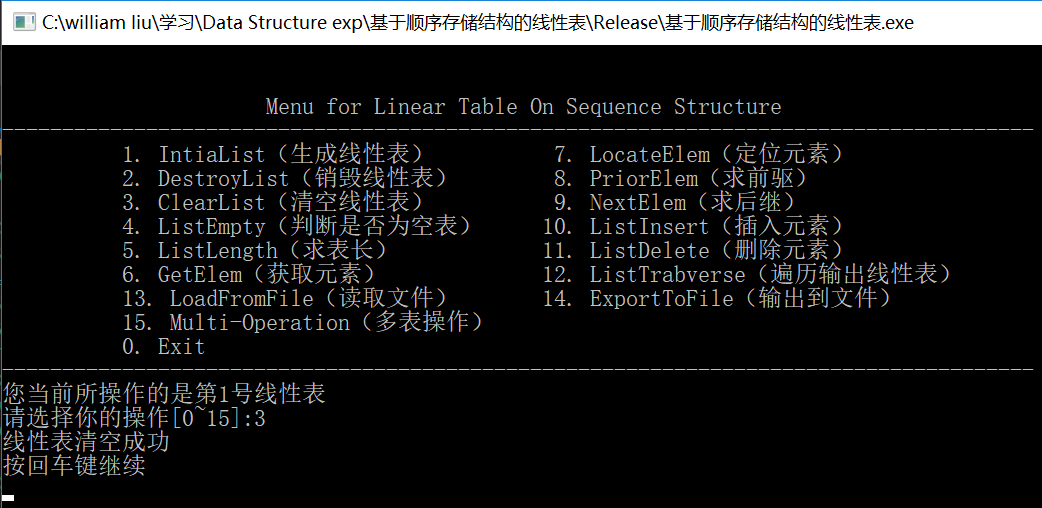


图1-10 测试3（1）截图

（2）未创建线性表，输入3，输出：线性表清空失败

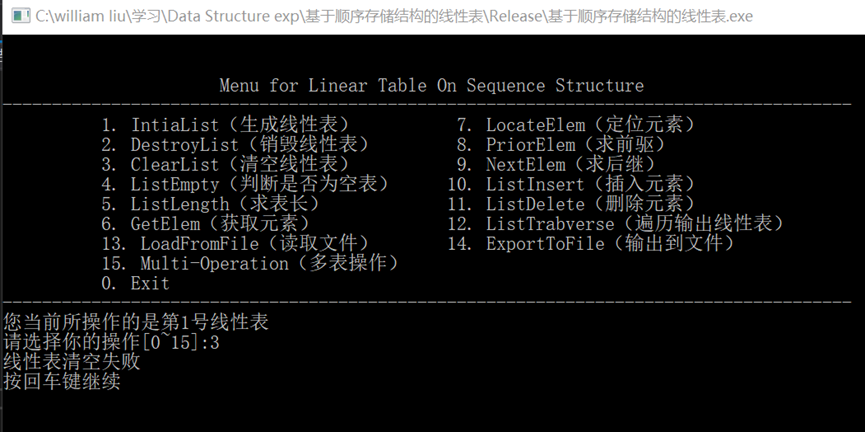


图1-11 测试3（2）截图

4、判断线性表是否为空功能：

（1）已创建线性表但为空：在3情况下，输入4，输出：线性表已经清空

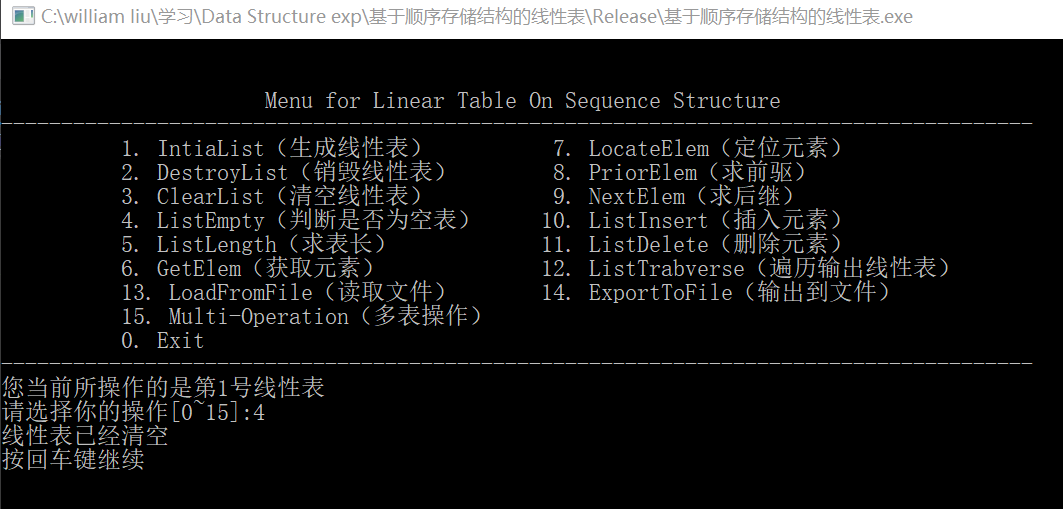


图1-12 测试4（1）截图

（2）已创建线性表但不为空：输入4，输出：线性表未清空

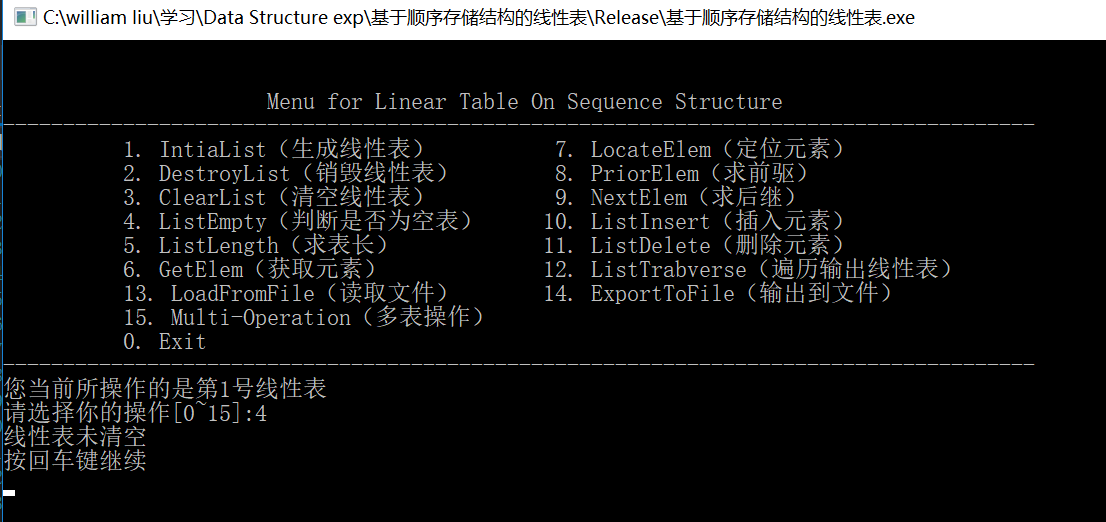


图1-13 测试4（2）截图

（3）未创建线性表：输入4，线性表未创建

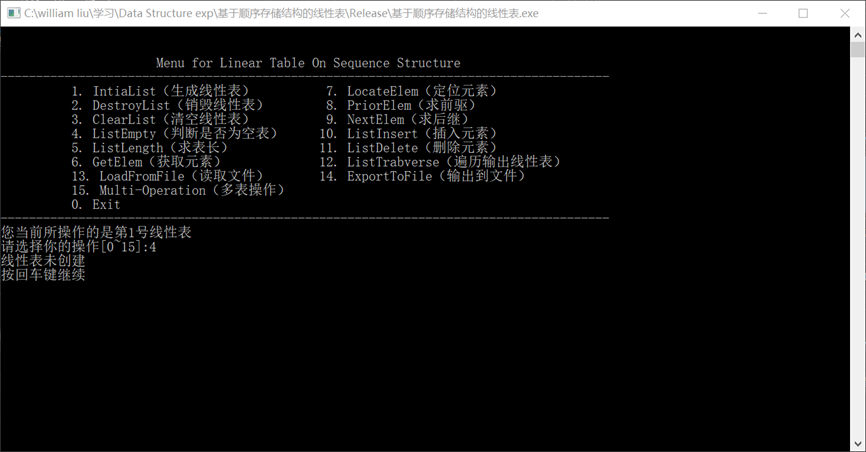


图1-14 测试4（3）截图

5、求表长功能：

（1）已创建线性表，线性表不为空，输入5，输出：该线性表的长度为：5

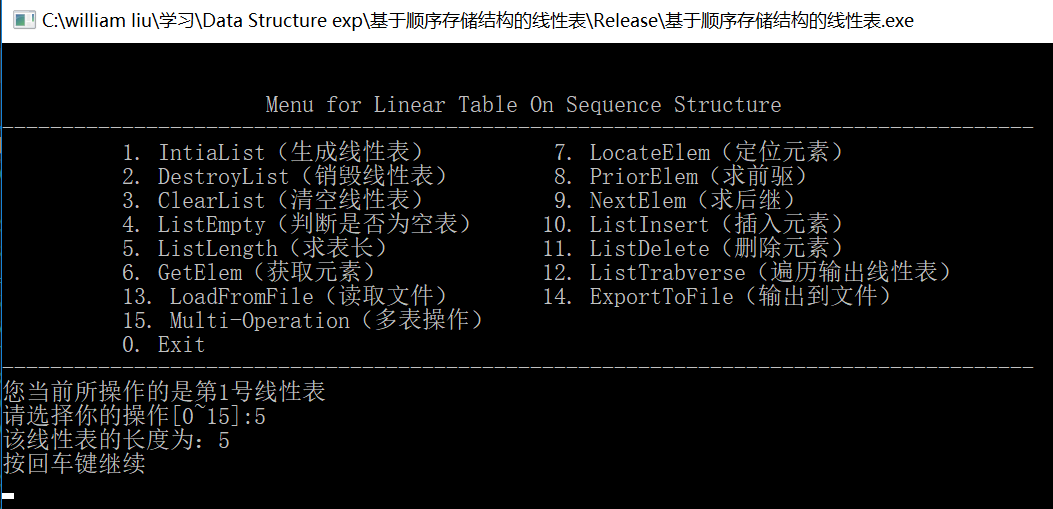


图1-15 测试5（1）截图

（2）未创建线性表，输入5，输出：测量长度失败

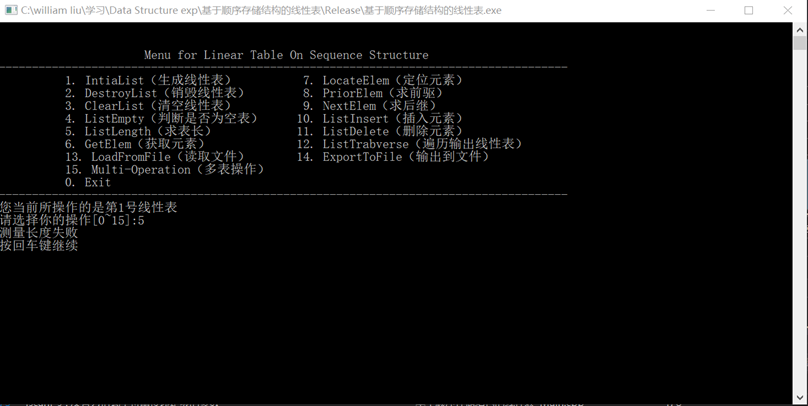


图1-16 测试5（2）截图

（3）已创建线性表，线性表为空，在3情况下，输入5，输出：线性表是空表

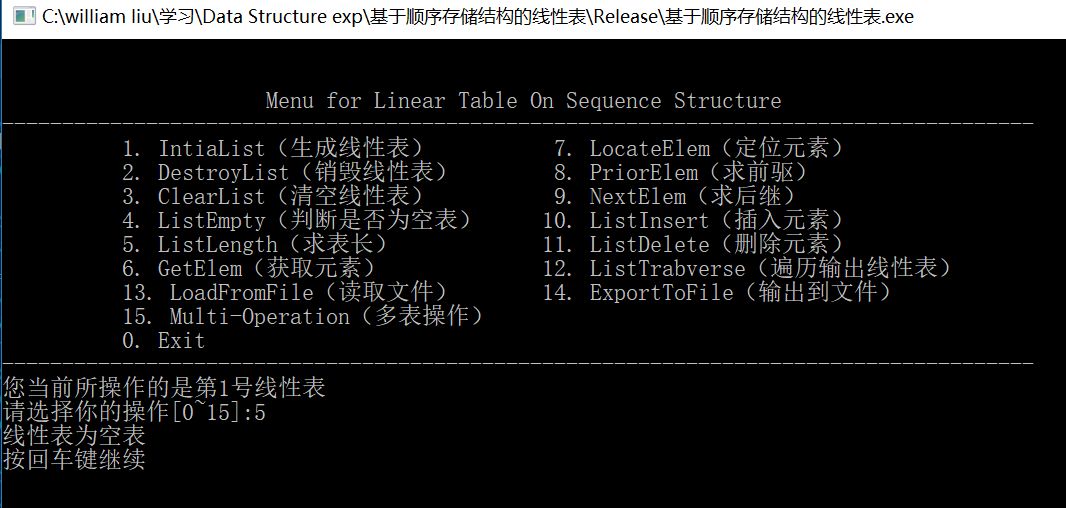


图1-17 测试5（3）截图

6、获取元素：

（1）已经创建线性表，获取元素在线性表中：输入6，元素序号4，输出：您所查询的元素第4个是4

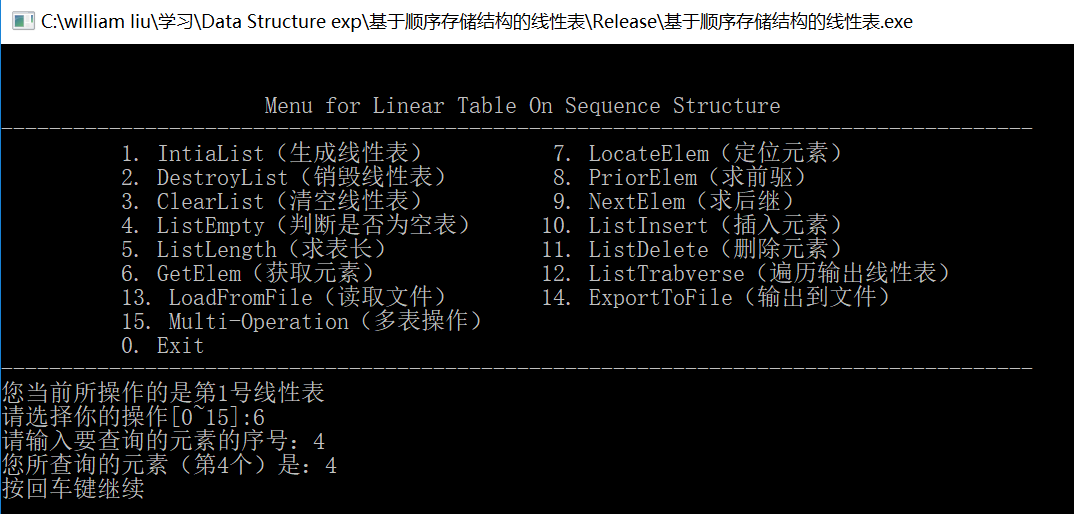


图1-18 测试6（1）截图

（2）已经创建线性表，获取元素不在线性表中：输入6，元素序号10，输出：查找失败

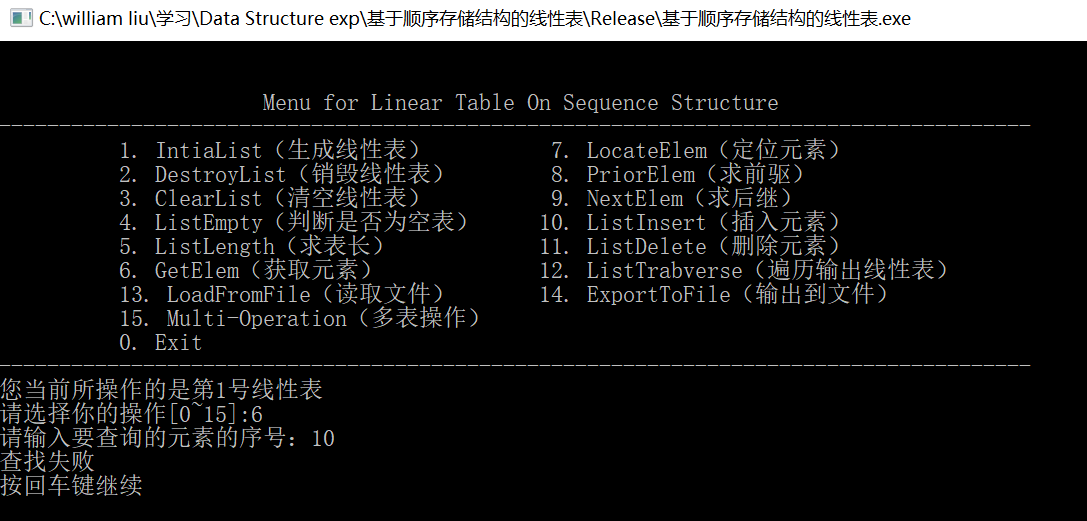


图1-19 测试6（2）截图

（3）未创建线性表：输入6，元素序号4，输出：查找失败

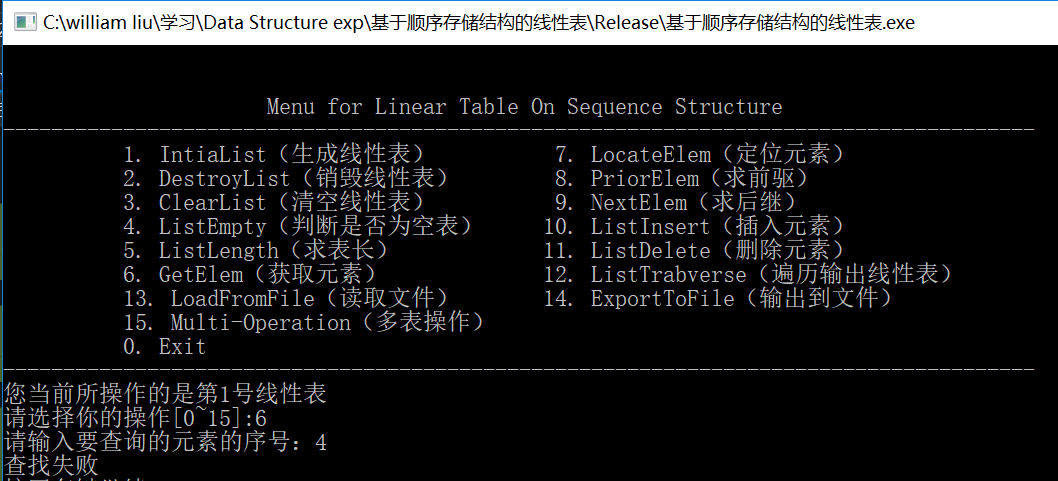


图1-20 测试6（3）截图

7、定位元素：

（1）已经创建线性表，元素在线性表中：输入7、3，输出：该元素的位置是：3

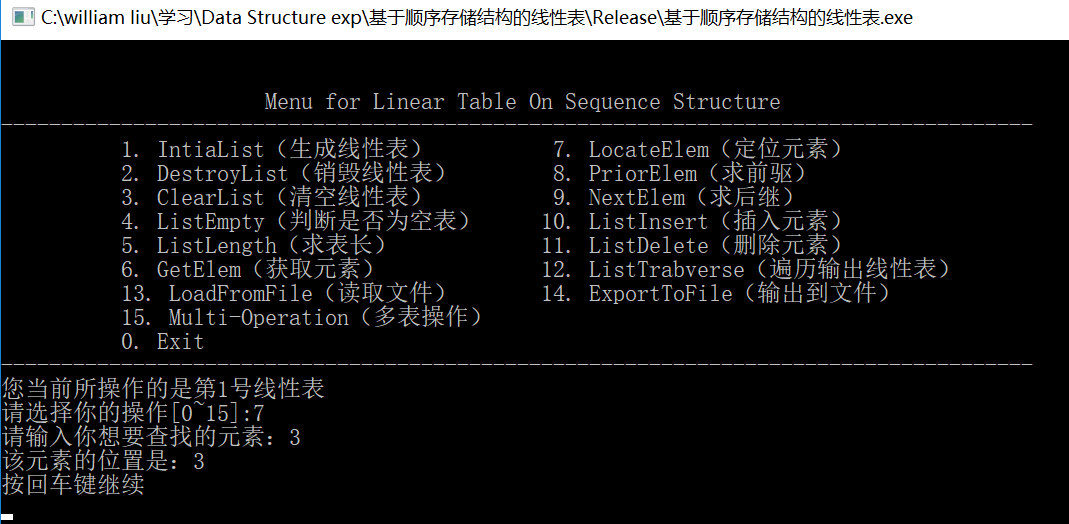


图1-21 测试7（1）截图

（2）已经创建线性表，元素不在线性表中：输入7、8，输出：查找失败

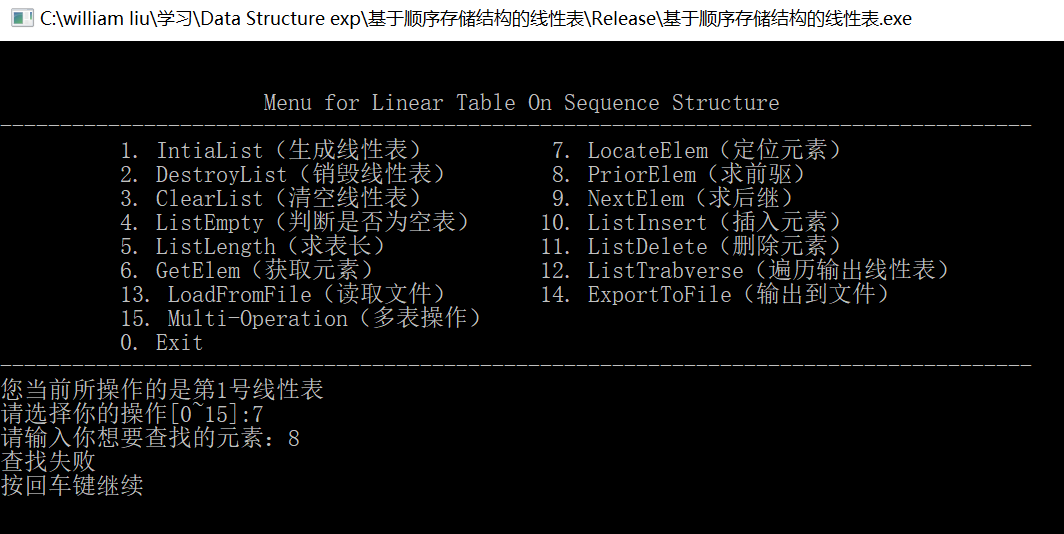


图1-22 测试7（2）截图

（3）未创建线性表：输入7、3，输出：查找失败

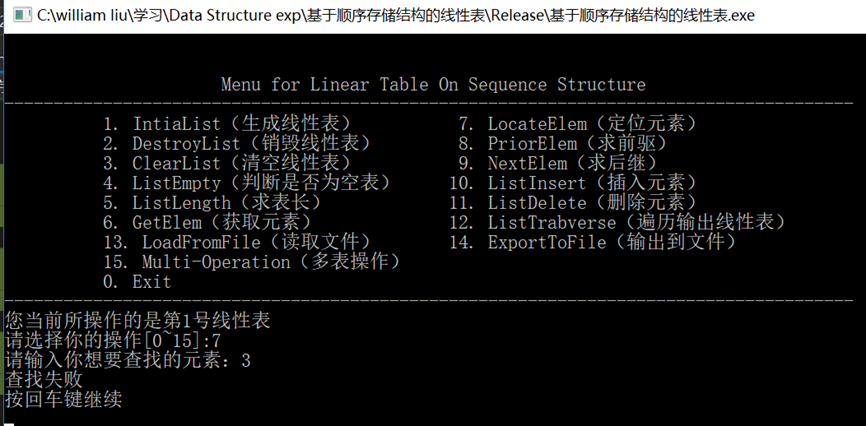


图1-23 测试7（3）截图

8、求前驱：

（1）已经创建线性表，元素在线性表中：输入8、3，输出：该元素的前驱为：2

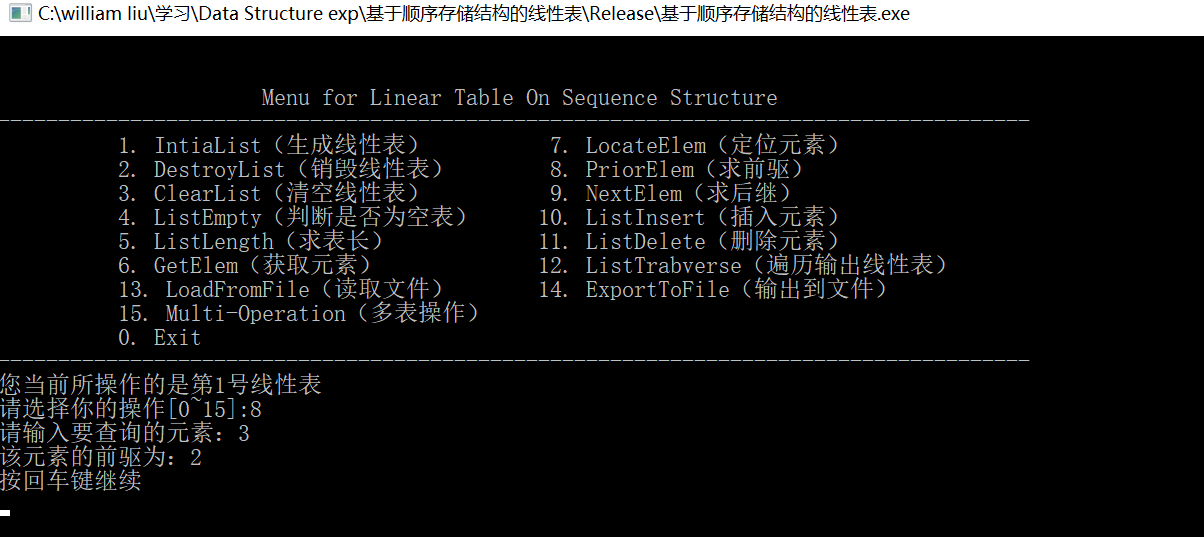


图1-24 测试8（1）截图

（2）已经创建线性表，元素不在线性表中（或是第一个元素）：输入8、1，输出：前驱查找失败



图1-25 测试8（2）截图

（3）未创建线性表：输入8、3，输出：前驱查找失败

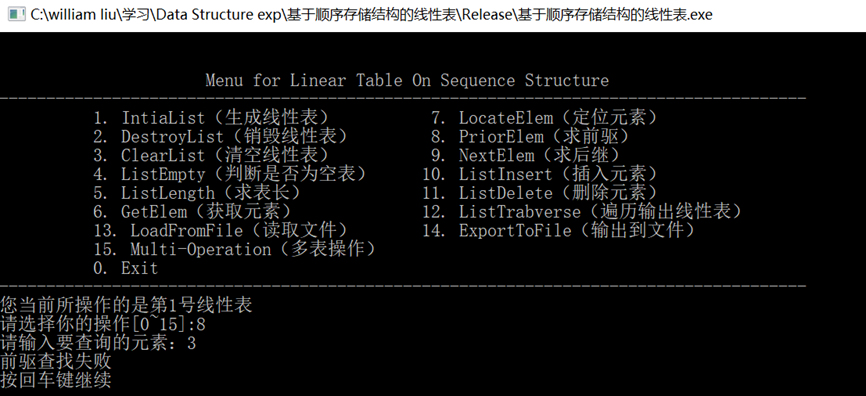


图1-26 测试8（3）截图

9、求后继：

（1）已经创建线性表，元素在线性表中：输入9、3，该元素的后继为：4



图1-27 测试9（1）截图

（2）已经创建线性表，元素不在线性表中（或是最后一个元素）：输入9、5，输出：后继查找失败

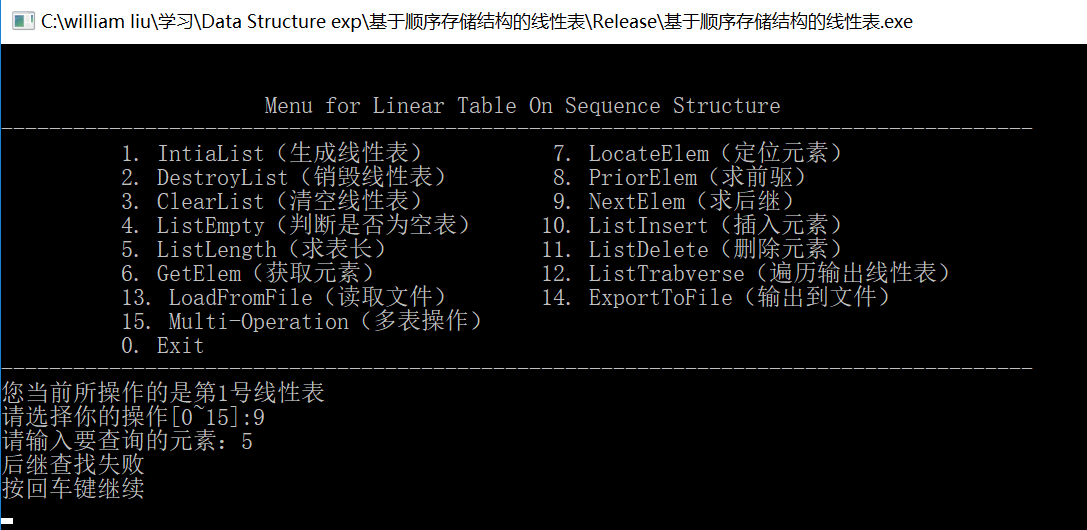


图1-28 测试9（2）截图

（3）未创建线性表：输入9、3，输出：后继查找失败

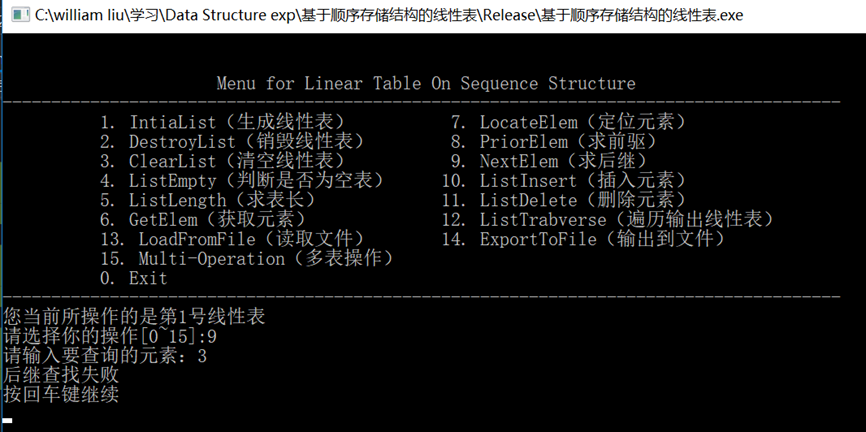
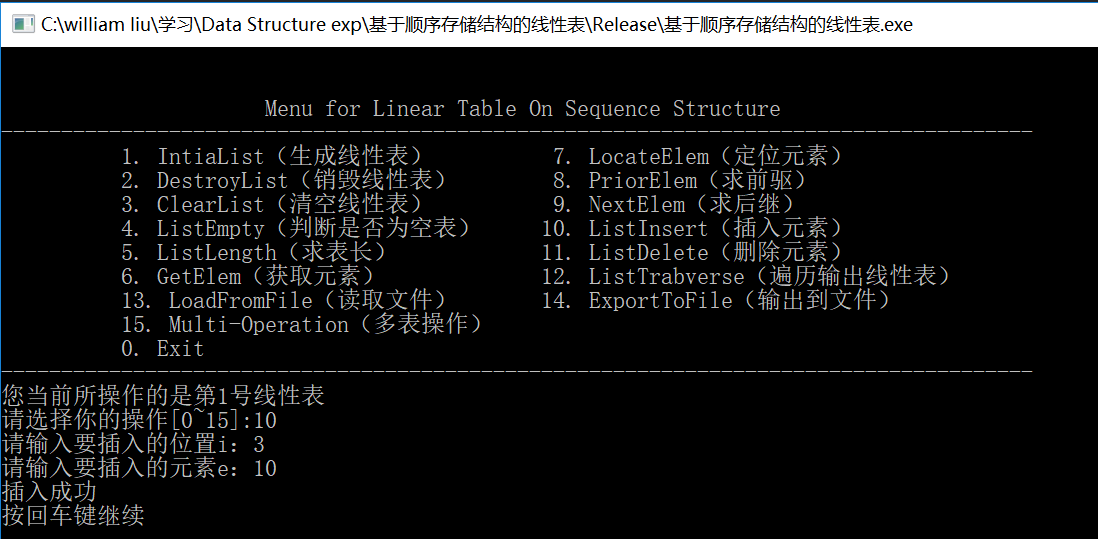


图1-29 测试9（3）截图

10、插入元素：

（1）已经创建线性表：输入10、3、10，输出： 再输入12查看结果，输出：1 2 10 3 4 5



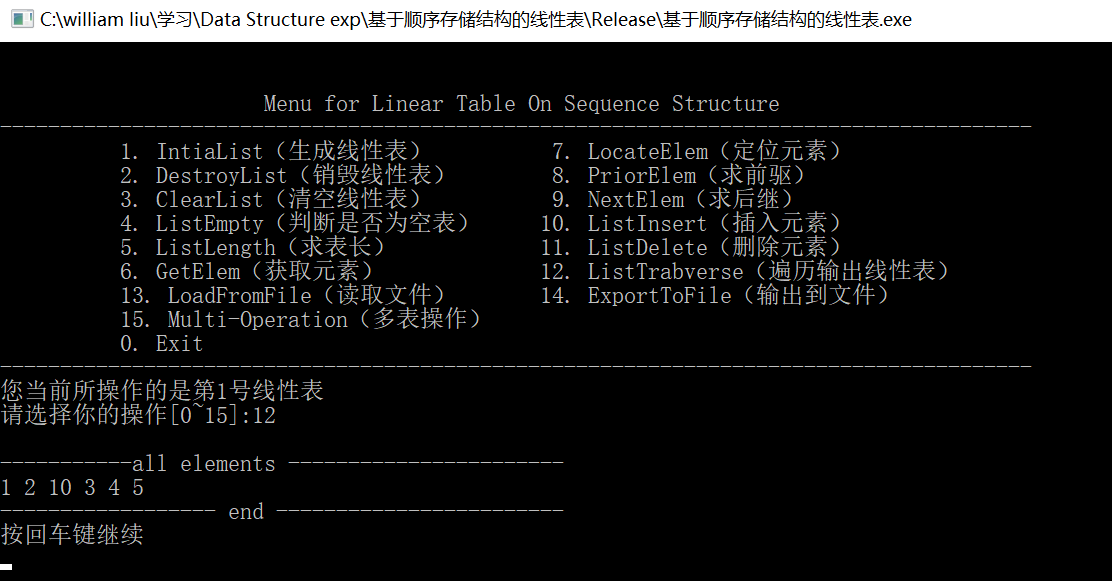


图1-30 测试10（1）截图

（2）未创建线性表：输入10、3、10 输出：插入失败



图1-31 测试10（2）截图

11、删除元素：

（1）已经创建线性表：在10的基础上，输入：11、3，输出：删除成功 所删除的元素是10 再输入12查看结果，输出：1 2 3 4 5

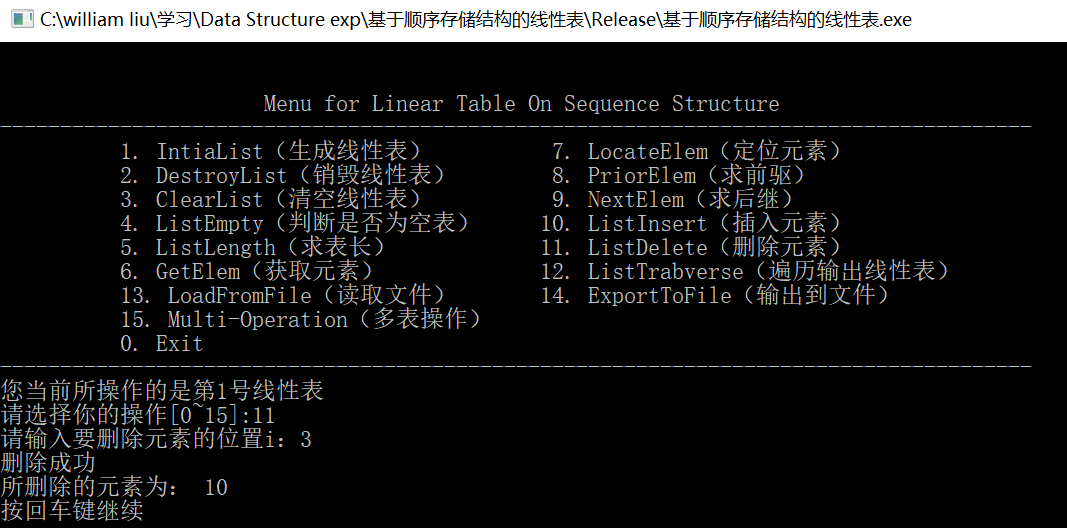




图1-32 测试11（1）截图

（2）未创建线性表：输入11、3，输出：删除失败

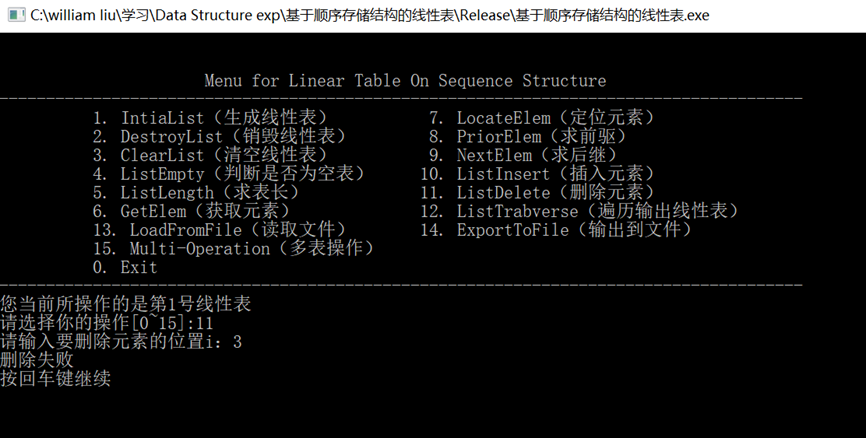


图1-33 测试11（2）截图

12、遍历输出线性表：

（1）已经创建线性表：输入12，输出：“—all elements--”和“--end--”的字符串标识，之间是线性表的内容：1 2 3 4 5

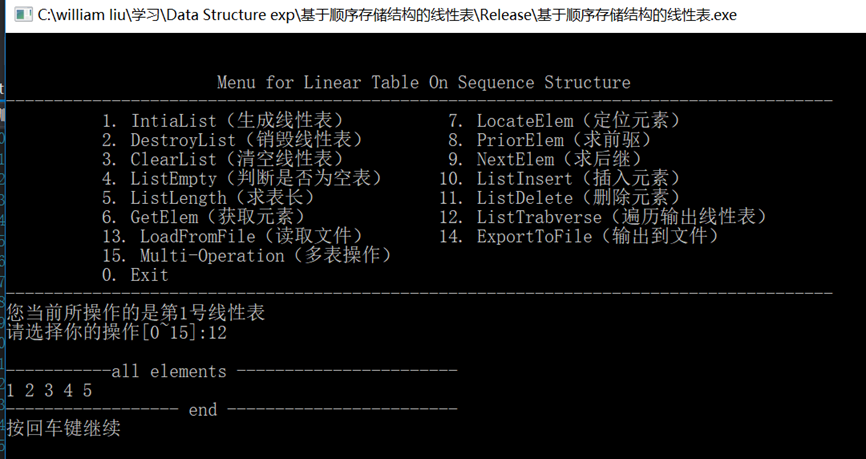


图1-34 测试12（1）截图

（2）未创建线性表：输入12，输出：线性表未创建！

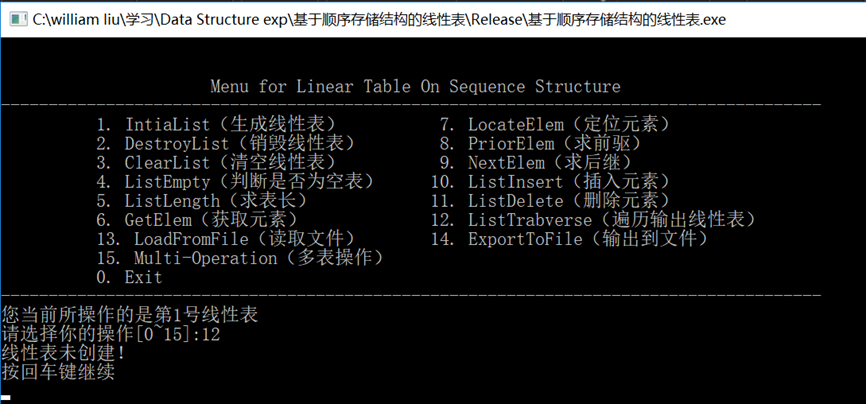


图1-35 测试12（2）截图

13、读取文件：

（1）已经创建线性表：输入13，out.txt，输出：读取线性表成功 再使用12遍历便可验证此结果

如果找不到指定文件，则报错

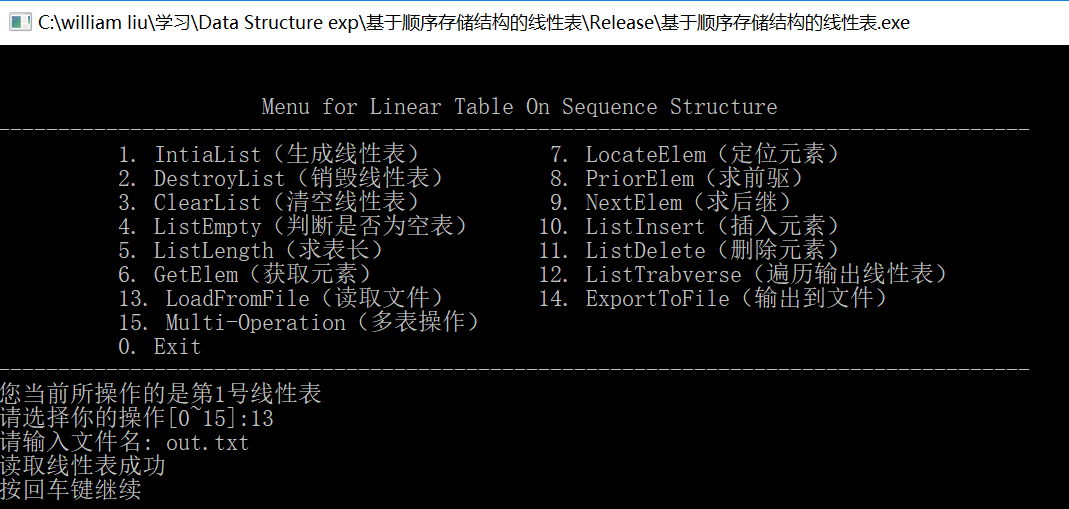




图1-36 测试13（1）截图

（2）未创建线性表：输入13，输出：读取线性表失败

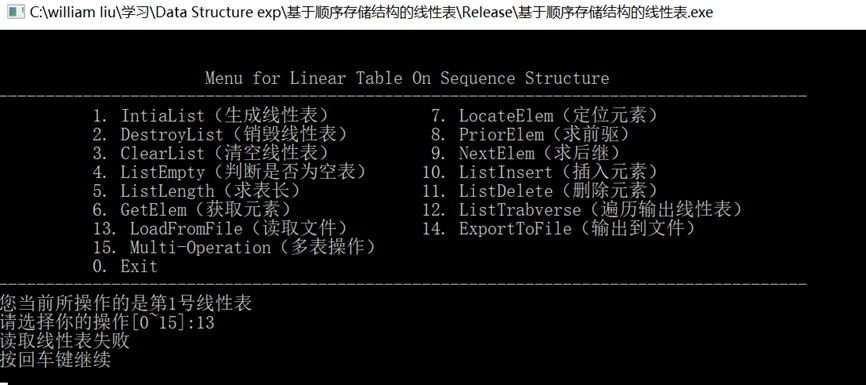
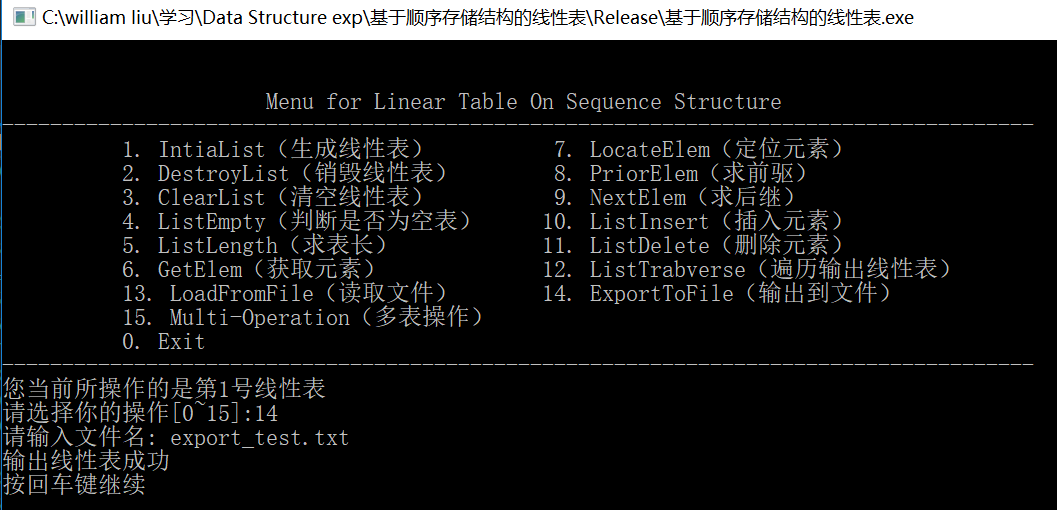


图1-37 测试13（2）截图

14、输出到文件：

（1）已经创建线性表：输入14，export\_test.txt，输出：输出线性表成功 再用记事本打开txt即可查看结果



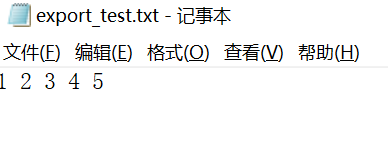


图1-38 测试14（1）截图

（2）未创建线性表：输入14，输出：输出线性表失败



图1-39 测试14（2）截图

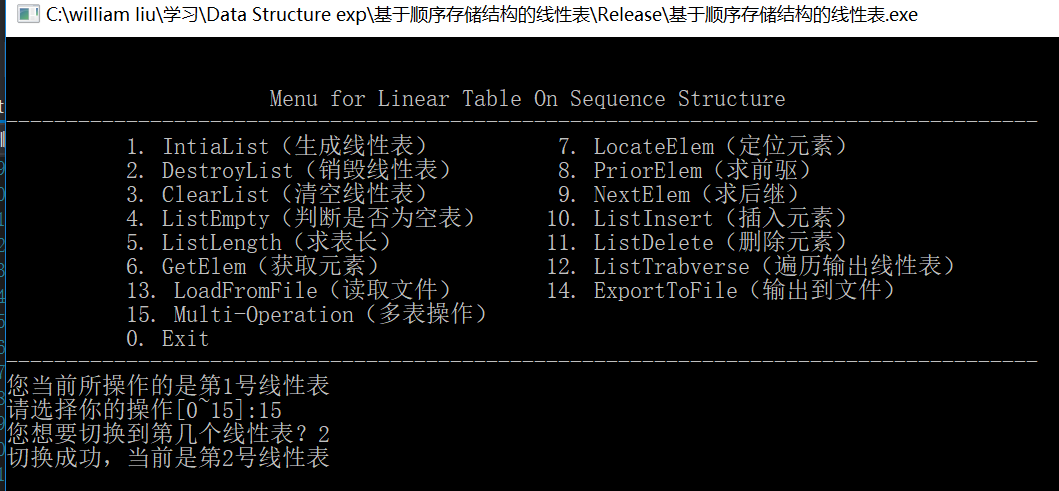
15、多表切换：

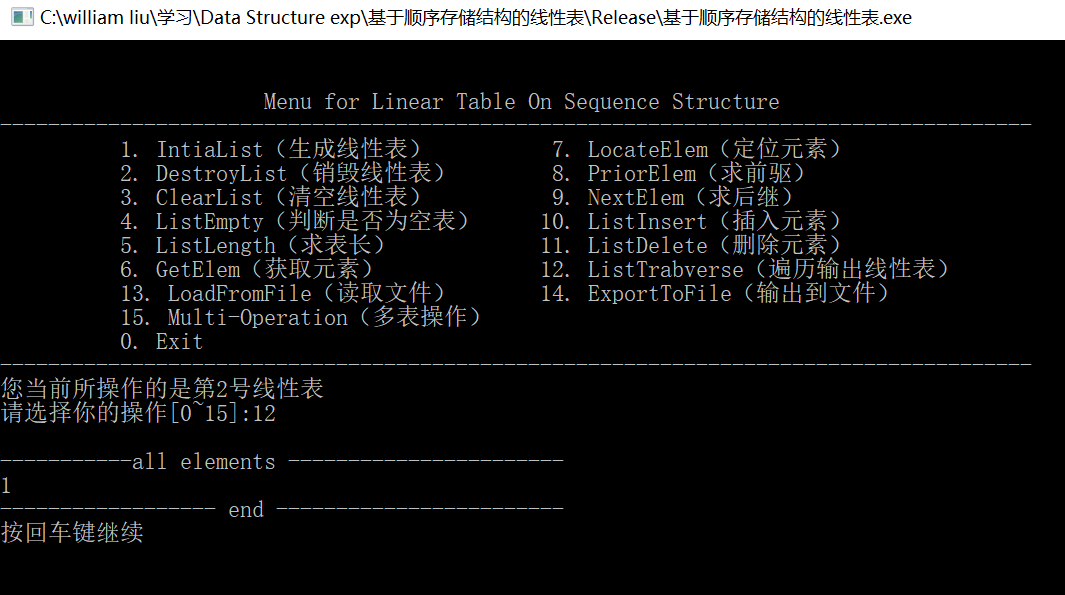
输入：15、2 输出：切换成功，当前是第2号线性表 并且继续后，演示菜单的提示信息会更新

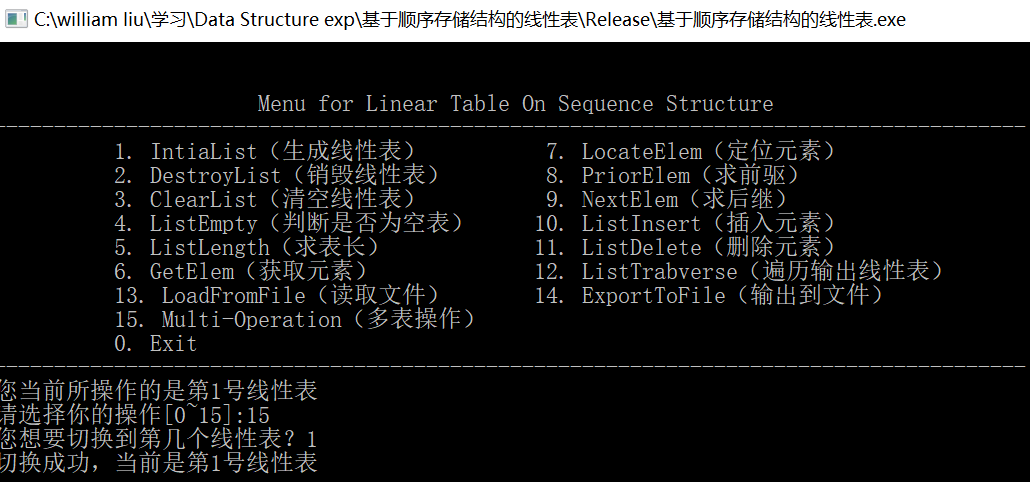
使用1，13 读入multi-test.txt中的线性表，并切换回表1，遍历表1，查看表1内容是否依然保存

输入13，multi-test.txt，15,1,12

输出：读取线性表成功，切换成功，当前是第1号线性表，1 2 3 4 5







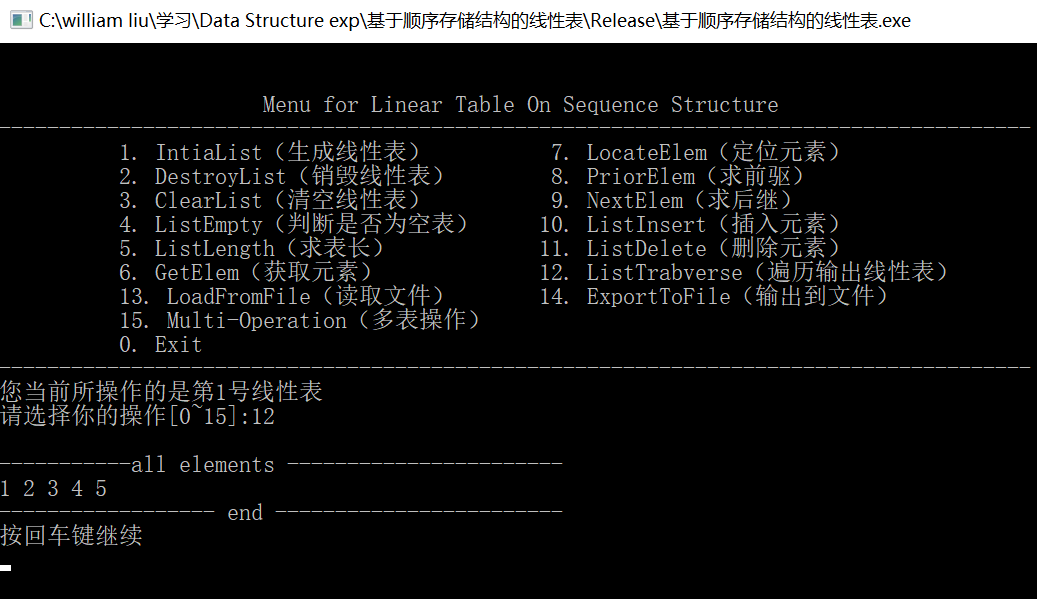


图1-40 测试15 截图

## 1.4 实验小结

本次实验主要考察了创建顺序线性表，逻辑结构较比简单，只有插入和删除会比较麻烦一些，时间复杂度也相对较高，菜单框架和模板都按照给好的制作，并添加了文件读取输出和多表操作功能。

通过自己写代码，回顾了c语言的文件读写操作，加深了对于顺序存储结构的线性表的理解，同时也熟悉了与之前c语言所学不同的c++引入的“&”引用类型，拥有和指针类似的功能，但是更安全也更方便，同时vs对于一些不安全操作给出了新的方案，比如scanf变成scanf\_s等。附加项多表操作则是考察了举一反三和知识迁移的能力，上课并没有讲过，但是却可以通过学习的知识和自己的不断尝试完成该功能，很好地锻炼了代码的能力。

自己仍有做的不足之处，比如判断线性表是否创建的时候使用了全局变量，没有能够将所有的操作都封存在功能函数中，模块化略显不足。

# 

# 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

1.declaration.h

#pragma once

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

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

#define TRUE 1

#define FALSE -1

#define OK 1

#define ERROR -2

//#define INFEASTABLE -1

//#define OVERFLOW -2

#define MaxLength 100

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=NULL;

int length;

int listsize;

}SqList;

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

status IntiaList(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);

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

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); //简化过

status LoadFromFile(FILE\*fp,SqList &L);

status ExportToFile(FILE\*fp,SqList L);

int Switch\_List();

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

2.Main.cpp

#include "declaration.h"//声明&定义头文件

//bool ListCreated[MaxLength];//判断线性表是否存在

FILE \*fp;

char filename[30];

int index = 1;

void main(void) {

SqList L;

SqList List[MaxLength];

int op = 1;

ElemType elem,cur;

int pos=0,result;

//Main Menu Entrance

while (op) {

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

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

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

printf(" 1. IntiaList（生成线性表） 7. LocateElem（定位元素）\n");

printf(" 2. DestroyList（销毁线性表） 8. PriorElem（求前驱）\n");

printf(" 3. ClearList（清空线性表） 9. NextElem（求后继） \n");

printf(" 4. ListEmpty（判断是否为空表） 10. ListInsert（插入元素）\n");

printf(" 5. ListLength（求表长） 11. ListDelete（删除元素）\n");

printf(" 6. GetElem（获取元素） 12. ListTrabverse（遍历输出线性表）\n");

printf(" 13. LoadFromFile（读取文件） 14. ExportToFile（输出到文件）\n");

printf(" 15. Multi-Operation（多表操作）\n");

printf(" 0. Exit\n");

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

printf("您当前所操作的是第%d号线性表\n",index);

L = List[index];

printf("请选择你的操作[0~15]:");

scanf\_s("%d", &op);

//根据用户输入调用相应函数，执行相应功能

switch (op) {

case 1:

if (IntiaList(L) == OK)

{

printf("线性表创建成功！\n");

List[index] = L;

//ListCreated[index] = true;

}

else printf("线性表创建失败！\n");

printf("按回车键继续\n");

getchar(); getchar();

break;

case 2:

if (DestroyList(&L) == OK)

{

printf("线性表销毁成功\n");

List[index] = L;

//ListCreated[index] = false;

}

else

{

printf("线性表销毁失败\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 3:

if (ClearList(L) == OK)

{

printf("线性表清空成功\n");

List[index] = L;

}

else

{

printf("线性表清空失败\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 4:

if (ListEmpty(L) == TRUE)

{

printf("线性表已经清空\n");

}

else if(ListEmpty(L)== FALSE)

{

printf("线性表未清空\n");

}

else if (ListEmpty(L) == ERROR)

{

printf("线性表未创建\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 5:

result = ListLength(L);

if (result == -2)

{

printf("测量长度失败\n");

}

else if(result == 0)

{

printf("线性表为空表\n");

}

else if (result != 0)

{

printf("该线性表的长度为：%d\n", result);

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 6:

printf("请输入要查询的元素的序号：");

scanf\_s("%d", &pos);

if (GetElem(L, pos, elem) == OK)

{

printf("您所查询的元素（第%d个）是：%d\n", pos,elem);

}

else

{

printf("查找失败\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 7:

printf("请输入你想要查找的元素：");

scanf\_s("%d", &elem);

result = LocateElem(L, elem);

if (result != 0 && result!=ERROR)

{

printf("该元素的位置是：%d\n",result);

}

else

{

printf("查找失败\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 8:

printf("请输入要查询的元素：");

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

if (PriorElem(L, cur, elem) == OK)

{

printf("该元素的前驱为：%d\n", elem);

}

else

{

printf("前驱查找失败\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 9:

printf("请输入要查询的元素：");

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

if (NextElem(L, cur, elem) == OK)

{

printf("该元素的后继为：%d\n", elem);

}

else

{

printf("后继查找失败\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 10:

printf("请输入要插入的位置i：");

scanf\_s("%d", &pos);

printf("请输入要插入的元素e：");

scanf\_s("%d", &elem);

if (ListInsert(L, pos, elem) == OK)

{

printf("插入成功\n");

List[index] = L;

}

else

{

printf("插入失败\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 11:

printf("请输入要删除元素的位置i：");

scanf\_s("%d", &pos);

if (ListDelete(L, pos, elem) == OK)

{

printf("删除成功\n");

printf("所删除的元素为： %d\n",elem);

List[index] = L;

}

else

{

printf("删除失败\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 12:

if (ListTrabverse(L)!=OK) {

printf("线性表未创建！\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 13:

printf("请输入文件名: ");

scanf\_s("%s", filename, 30);

if (fopen\_s(&fp, filename, "r+") != 0)

{

printf("文件打开失败\n ");

}

else if (LoadFromFile(fp,L) == OK)

{

printf("读取线性表成功\n");

List[index] = L;

}

else

{

printf("读取线性表失败\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 14:

printf("请输入文件名: ");

scanf\_s("%s", filename, 30);

if (fopen\_s(&fp, filename, "w+") != 0)

{

printf("打开文件失败\n");

}

else if (ExportToFile(fp,L) == OK)

{

printf("输出线性表成功\n");

}

else

{

printf("输出线性表失败\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 15:

if (Switch\_List() == OK)

{

printf("切换成功，当前是第%d号线性表\n",index);

}

else

{

printf("切换失败\n");

}

printf("按回车键继续\n");

getchar(); getchar();

break;

case 0:

printf("欢迎下次再使用本系统！\n");

getchar(); getchar();

break;

}//end of switch

}//end of while

}//end of main()

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

/\*\*

\* 函数名称：InitiaList

\* 函数参数：线性表L的地址

\* 函数功能：构造一个空的线性表

\* 返回值：成功构造返回OK，否则返回ERROR

\*\*/

status IntiaList(SqList & L) {

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

if (!L.elem) return ERROR;//存储分配失败

L.length = 0;//空表长度为0

L.listsize = LIST\_INIT\_SIZE;//初始存储容量

return OK;

}

/\*\*

\* 函数名称：DestroyList

\* 函数参数：线性表L的地址

\* 函数功能：删除线性表

\* 返回值：成功销毁返回OK，否则返回ERROR

\*\*/

status DestroyList(SqList \* L)

{

//判断线性表是否已创建

if (L->elem!=NULL)

{

//释放空间

free(L->elem);

L->elem = NULL;

return OK;

}

else

{

return ERROR;

}

}

/\*\*

\* 函数名称：ClearList

\* 函数参数：线性表L的地址

\* 函数功能：重置线性表

\* 返回值：成功置空返回OK，否则返回ERROR

\*\*/

status ClearList(SqList &L)

{

if (L.elem!=NULL)

{

L.length = 0;//清空：将长度置0

return OK;

}

else

{

return ERROR;

}

}

/\*\*

\* 函数名称：ListEmpty

\* 函数参数：线性表L

\* 函数功能：判断线性表是否为空

\* 返回值：若L为空表则返回TRUE，否则返回FALSE

\*\*/

status ListEmpty(SqList L)

{

if (L.elem!=NULL)

{

if (L.length == 0)

{

return TRUE;

}

else

{

return FALSE;

}

}

else

{

return ERROR;

}

}

/\*\*

\* 函数名称：ListLength

\* 函数参数：线性表L

\* 函数功能：计算线性表L中数据元素个数；

\* 返回值：成功返回L中数据元素个数，失败返回ERROR

\*\*/

int ListLength(SqList L)

{

if (L.elem == NULL)

{

return ERROR;

}

return L.length;

}

/\*\*

\* 函数名称：GetElem

\* 函数参数：线性表L，e为第i个数据的地址；

\* 函数功能：查找并显示L中第i个元素的值；

\* 返回值：成功则返回第i个元素的值，否则返回ERROR

\*\*/

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

{

if (L.elem!=NULL)

{

if (i <= L.length && i > 0)//判断第i个元素是否存在

{

e = \*(L.elem + i - 1);//第1个位置对应elem，第i个即从elem向后移动i-1个位置

return OK;

}

else

{

return ERROR;

}

}

else

{

return ERROR;

}

}

/\*\*

\* 函数名称：LocateElem

\* 函数参数：线性表L，相对比的元素值；

\* 函数功能：查找L中与e相同数据所在的位序；位序从1开始

\* 返回值：成功则返回第一个与e相同的数据的位序，不存在则返回0，其余情况返回ERROR

\*\*/

int LocateElem(SqList L, ElemType e)//简化过

{

if (L.elem!=NULL)

{

for (int j = 1; j <= L.length; j++)//循环遍历线性表

{

if (\*L.elem != e)//查找值为e的元素

{

L.elem++;//当前不等，则比较下一个

}

else

{

return j;//找到了，返回其下标

}

}

return 0;

}

else

{

return ERROR;

}

}

/\*\*

\* 函数名称：PriorElem

\* 函数参数：线性表L，查找的数据cur\_e,前驱pre\_e；

\* 函数功能：查找L中与cur\_e相同的第一个数据，并返回其前驱

\* 返回值：成功则返回与cur\_e相同的第一个数据，并返回其第一个前驱；若无前驱以及其他情况则返回ERROR

\*\*/

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

{

if (L.elem!=NULL)

{

for (int j = 1; j <= L.length - 1; j++)//遍历线性表

{

if (L.elem[j] == cur)//找到目标元素

{

pre\_e = L.elem[j - 1];//获取目标元素下标-1的元素

return OK;

}

}

}

return ERROR;

}

/\*\*

\* 函数名称：NextElem

\* 函数参数：线性表L，查找的数据cur\_e,后驱next\_e；

\* 函数功能：查找L中与cur\_e相同的第一个数据，并返回其后驱

\* 返回值：成功则返回与cur\_e相同的第一个数据，并返回其后驱；若无后驱以及其他情况则返回ERROR

\*\*/

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

{

if (L.elem!=NULL)

{

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

{

if (L.elem[j] == cur)

{

next\_e = L.elem[j + 1];//获取后继

return OK;

}

}

}

return ERROR;

}

/\*\*

\* 函数名称：ListInsert

\* 函数参数：线性表L的地址，插入的位置i，插入的数据元素e。

\* 函数功能：在L的第i个位置之前插入新的数据元素e；若已插满，则按照分配增量，分配更大的空间。

\* 返回值：成功插入返回OK，否则返回ERROR

\*\*/

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

{

if (L.elem!=NULL)

{

//在顺序线性表L中第i个位置之前插入新的元素e，i的合法值为1~ListLength+1

if (i<1 || i>L.length + 1) return ERROR; //i值不合法

if (L.length >= L.listsize)

{

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

if (!newbase) return ERROR;//存储分配失败

L.elem = newbase;//新基址

L.listsize += LISTINCREMENT;//增加存储容量

}

int \*q = &(L.elem[i - 1]);//q为插入位置

//遍历，将插入位置后面的元素后移一位

for (int \*p = &(L.elem[L.length - 1]); p >= q; --p)

{

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

}

\*q = e;

++L.length;//线性表长度+1

return OK;

}

else

{

return ERROR;

}

}

/\*\*

\* 函数名称：ListDelete

\* 函数参数：线性表L的地址，删除的位序i，指针e的地址。

\* 函数功能：删除L的第i个数据元素，用e返回其值。

\* 返回值：成功删除返回OK，否则返回ERROR

\*\*/

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

{

if (L.elem!=NULL)

{

//在顺序线性表L中删除第i个元素，并用e返回其值

//i的合法值为1~ListLength

if ((i < 1 || (i > L.length))) return ERROR;//i值不合法

int \*p = &(L.elem[i - 1]);//p为被删除元素的值赋给e

e = \*p;//被删除的元素赋给e

int \*q = L.elem + L.length - 1;//表尾元素的位置

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

{

\*(p - 1) = \*p;//被删除元素后元素左移

}

--L.length;//表长减1

return OK;

}

else

{

return ERROR;

}

}

/\*\*

\* 函数名称：ListTrabverse

\* 函数参数：线性表L

\* 函数功能：依次显示线性表中的每个元素。

\* 返回值：成功遍历返回线性表的长度，否则返回ERROR

\*\*/

status ListTrabverse(SqList L) {

if (L.elem!=NULL) {

int i;

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

//循环遍历输出

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

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

return OK;

}

else

{

return ERROR;

}

}

/\*\*

\* 函数名称：LoadFromFile

\* 函数参数：文件指针，线性表L

\* 函数功能：数据加载，为线性表输入元素。

\* 返回值：加载成功返回OK，加载失败返回E

\*\*/

status LoadFromFile(FILE\*fp,SqList &L)

{

if (L.elem!=NULL && fp!=NULL)

{

L.length = 0;

char vacant;//用于读取空格

while (fscanf\_s(fp, "%d", &L.elem[L.length])!=EOF)

{

fscanf\_s(fp, "%c", &vacant);//读取数字后的空格

L.length++;

}

fclose(fp);

return OK;

}

else

{

return ERROR;

}

}

/\*\*

\* 函数名称：ExportToFile

\* 函数参数：文件指针，线性表L

\* 函数功能：将L中的数据保存到fp指针指向的文件中

\* 返回值：OK

\*\*/

status ExportToFile(FILE\*fp,SqList L)

{

if (L.elem!=NULL && fp!=NULL)

{

//循环遍历线性表，并将线性表内容放入文件中

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

{

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

}

fclose(fp);

return OK;

}

else

{

return ERROR;

}

}

//多表操作函数，切换线性表

status Switch\_List()

{

int num;

printf("您想要切换到第几个线性表？");

scanf\_s("%d", &num);

if (num > MaxLength-1||num < 1)//若超过最大值的范围，则返回ERROR

{

return ERROR;

}

else

{

index = num;

return OK;

}

}

# 