# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

采用顺序表的物理结构，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。演示系统实现多线性表管理，可进行多表操作。

设计线性表文件保存和加载操作合理模式，即将生成的线性表存入到文件中，也可以从文件中获取线性表进行操作；并设计文件数据记录格式，以高效保存线性表数据逻辑结构的完整结构以及信息。

## 1.1.1 线性表的基本概念与逻辑结构与基本运算

线性表是最常用且最简单的一种数据结构，即n个数据元素的有限序列。线性表中元素的个数n定义为线性表的长度，n=0时成为空表。在非空表中的每个数据元素都有一个确定的位置。线性表的存储结构分为线性存储和链式存储。本次实验需要实现的是基于线性存储的线性表。

线性表的数据逻辑结构定义如下:

ADT List｛

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｝

｝

## 1.1.2 线性表的基本运算

依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的12种基本运算，具体运算功能定义如下：

1. 初始化表InitList(L)；

初始条件：线性表L不存在。

操作结果：构造一个空的线性表。

1. 销毁表DestroyList(L)；

初始条件：线性表L已存在。

操作结果：销毁线性表L。

1. 清空表ClearList(L)；

初始条件：线性表L已存在。

操作结果：将L重置为空表。

1. 判定空表ListEmpty(L)；

初始条件：线性表L已存在。

操作结果：若L为空表则返回TRUE，否则返回FALSE。

1. 求表长ListLength(L)；

初始条件：线性表L已存在。

操作结果：返回L中数据元素的个数。

1. 获得元素GetElem(L,i,e)；

初始条件：线性表L已存在，且1≤i≤L.Length。

操作结果：用e返回L中第i个数据元素的值。

1. 查找元素LocateElem(L,e)；

初始条件：线性表L已存在。

操作结果：返回线性表L中第1个与e相同的数据元素的位序；若这样的数

据元素不存在，则返回值为0。

1. 获得前驱PriorElem(L,cur\_e,pre\_e)；

初始条件：线性表L已存在。

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。

1. 获得后继NextElem(L,cur\_e,next\_e)；

初始条件：线性表L已存在。

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。

1. 插入元素ListInsert(L,i,e)；

初始条件：线性表L已存在，且1≤i≤L.Length+1；

操作结果：在L第i个位置之前插入新的数据元素e。

1. 删除元素ListDelete(L,i,e)；

初始条件：线性表L已存在，线性表非空，且1≤i≤L.Length；

操作结果：删除L第i个数据元素，用e返回其值。

1. 遍历表ListTraverse(L)；

初始条件：线性表L已存在；

操作结果：依次对L的每个数据元素进行输出。

## 1.2 系统设计

## 1.2.1 数据存储结构与形式

线性表的数据物理结构如下：

typedef struct{

ElemType \* elem; //定义元素类型指针，存储空间基址

int length; //线性表的长度

int listsize; //当前分配的存储容量

char name[20]; //线性表的名称

}SqList;

PS.若要实现同时对多个线性表管理，只需要定义一个线性表的结构数组，通过线性表的名称属性对线性表进行操作。

## 1.2.2 演示系统设计

通过WHILE循环与菜单界面，用户通过选择菜单中的选项实现交互，使用OP变量获取用户选择选项值（OP初始化值为1，以便第一次能进入循环）。

进入循环后系统首先显示功能菜单，然后用户输入选择0-14，其中1-12分别代表线性表的一个基本运算，13与14选项分别是线性表的文件保存，以及显示全部线性表信息（**本选项用于方便查看在存在多个线性表的情况下，各个线性表的长度与名称信息**），在主函数中通过SWITCH语句对应到相应的函数功能，执行完该功能后BREAK跳出SWITCH语句，继续执行while循环，直至用户输入0退出当前演示系统。

在进行线性表的操作时，除了1和14选项（1选项是进行线性表的生成和初始化，14选项用于查看所有线性表信息），系统会要求输入需要处理的线性表的名称（本系统多线性表操作以线性表名称作为查找特征），若线性表不存在表会返回菜单界面重新操作（演示系统结构如图1-1所示）。

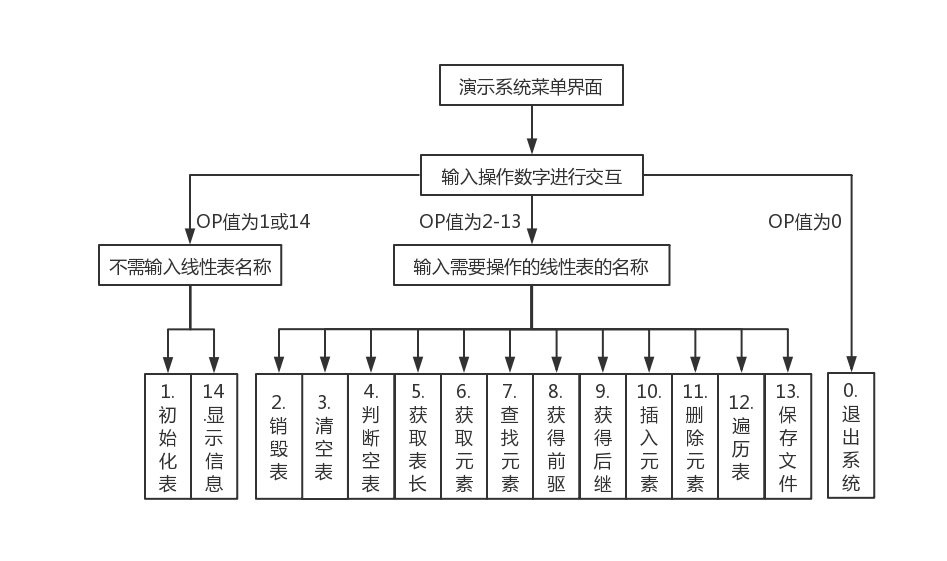


图1-1 演示系统菜单操作示意图

## 1.3 系统实现

**1.3.1 编程环境与运行环境描述**

编程环境：采用CodeBlocks 16.01编程软件编写。

运行环境：微软Windows 10系统。

**1.3.2 头文件及预定义常量说明**

1.头文件

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

2.预定义常量

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

3.类型表达式

typedef int status;

typedef int ElemType;

**1.3.3 算法设计与实现**

（1）初始化表status InitList(SqList &L)；

算法实现：使用malloc函数分配连续内存空间，将首地址返回赋值给L.elem，将L.length初始化为0，L.listsize初始化为LIST\_INIT\_SIZE。

1. 销毁表status DestroyList(SqList &L)；

算法实现：使用free函数释放掉以L.elem为首地址的连续内存空间，并将L.length，L.listsize重新赋值为0，将L.name清空。

1. 清空表status ClearList(SqList &L)；

算法实现：若线性表不存在返回ERROR，若线性表存在，则将线性表的长度L.length的值置为0。

1. 判定空表status ListEmpty(SqList L)；

算法实现：若线性表不存在，返回-1，若线性表存在则读取线性表L.length，若其值为0则返回FALSE，否则返回TRUE。

1. 求表长int ListLength(SqList L)；

算法实现：若线性表不存在，返回-1，若线性表存在则直接返回L.length即为所求线性表的表长。

1. 获得元素status GetElem(SqList L,int i,ElemType &e)；

算法实现：若1≤i≤L.Length则通过随机访问数组的方式获取元素，将L.elem[i-1]的值赋值给e；否则返回ERROR。

1. 查找元素status LocateElem(SqList L,ElemType e)；

算法实现：通过循环遍历线性表，将每一个元素与给定值比较看是否相等，如果相等就返回该元素的次序，否则返回0。

1. 获得前驱status PriorElem(SqList L,ElemType cur,ElemType &pre\_e)；

算法实现：若cur的值和L.elem[0]相同，则返回-1；若遍历线性表时cur与L.elem[i]相等，则将L.elem[i-1]赋值给pre\_e；若无元素符合，则ERROR（获得前驱函数流程图，如图1-2所示）。

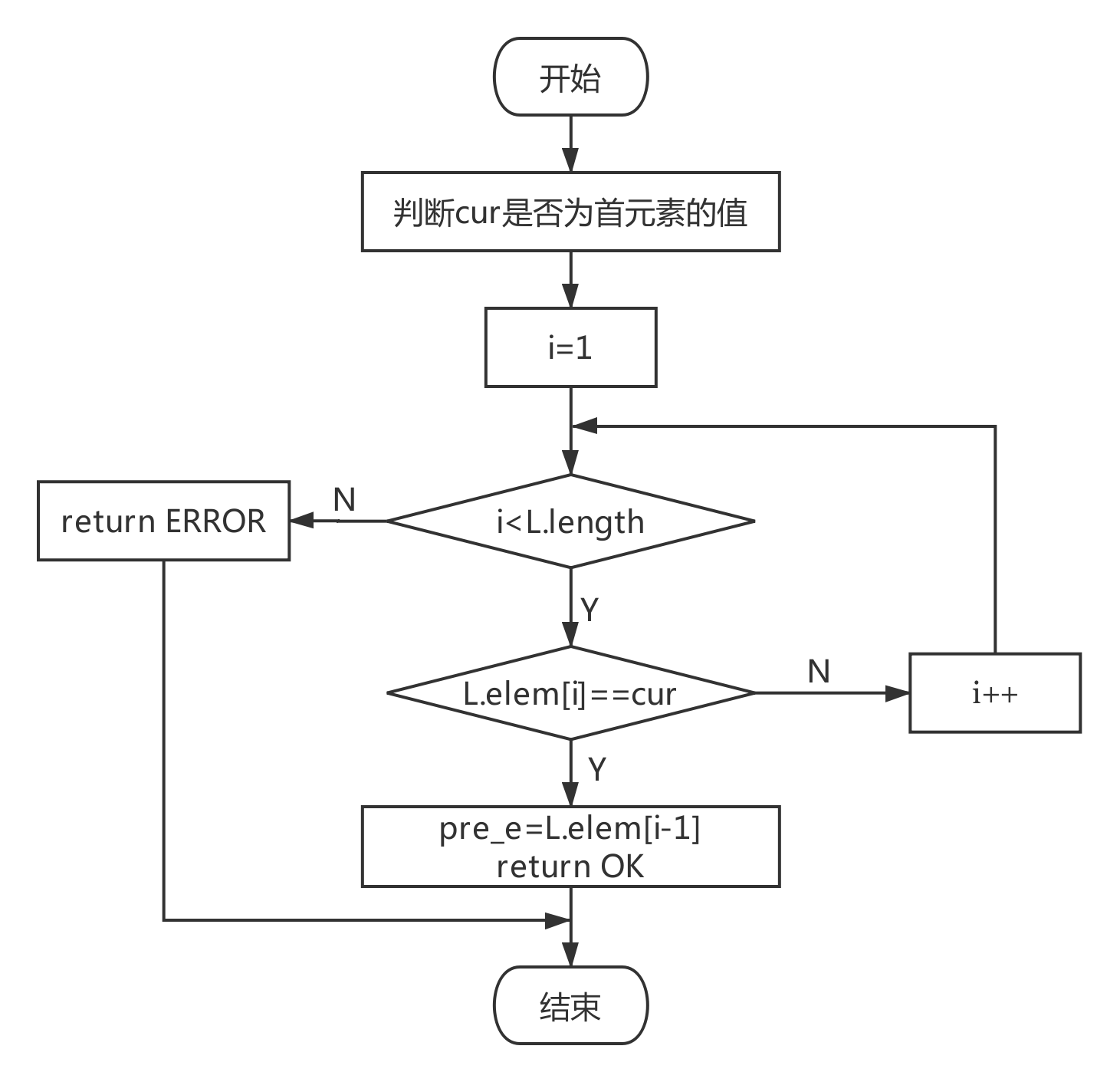


图1-2 获得前驱函数流程图

1. 获得后继status NextElem(SqList L,ElemType cur,ElemType &next\_e)；

算法实现：通过循环遍历线性表，查找前L.length-1个元素是否符合，若符合则将其后继赋值给next\_e，然后判断是否为最后一个元素，若是的话返回-1，否则返回ERROR。

1. 插入元素status ListInsert(SqList &L,int i,ElemType e)；

算法实现：首先判断插入位置的合法性，若1≤i≤L.Length+1则继续，否则返回ERROR；然后判断当前存储空间是否已满，若满了则用realloc函数重新分配空间，插入元素时，从最后一个元素开始向前遍历，每个元素以此往后移一个单元直到插入点，最后插入e，表长L.length数值增加1（插入元素函数流程图，如图1-3所示）。

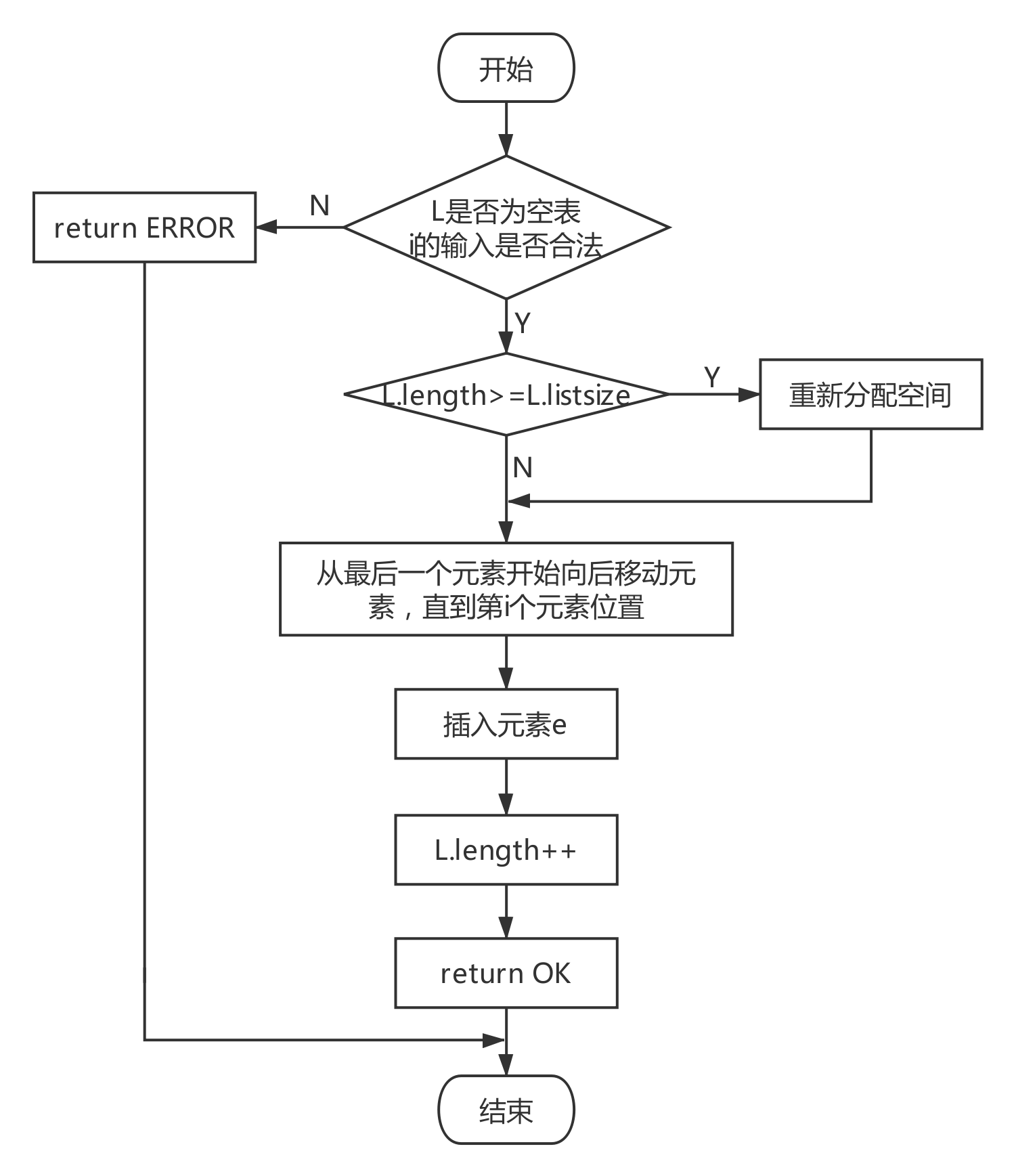


图1-3 插入元素函数流程图

1. 删除元素status ListDelete(SqList &L,int i,ElemType &e)；

算法实现：首先判断位序的合法性，并判断线性表是否为空表，即若1≤i≤L.Length且L.length不为0则继续，否则返回ERROR；在之后直接将删除元素位置后一个元素直到最后一个元素以此从前往后向前移动一个单元。

1. 遍历表status ListTrabverse(SqList L)；

算法实现：采用循环遍历线性表，输出线性表中的每一个元素。

1. 读取文件status OpenFile(SqList &L)；

算法实现：使用fopen函数打开指定文件，然后使用fscanf函数读取文件的数据，并将数据赋值给线性表，同时线性表的长度依次增加；若空间不足则用realloc函数重新分配空间，最后关闭文件。

1. 键盘赋值status SetValue(SqList &L)；

算法实现：在键盘上输入数据，数据集以Ctrl+Z为结束标志，然后使用scanf函数从缓冲区中读取数据，赋值给线性表，同时线性表的长度依次增加；若空间不足则用realloc函数重新分配空间。

1. 文件保存status SaveFile(SqList &L)；

算法实现：首先，选择读写文件的方式，选择1选项不删除原有的线性表，在原有数据之后添加数据，或者选择2选项删除原有线性表数据或生成新文件；然后使用fopen函数创建或打开文件，然后用fputs函数和fprintf函数输出数据，最后关闭文件（文件保存函数流程图，如图1-4所示）。

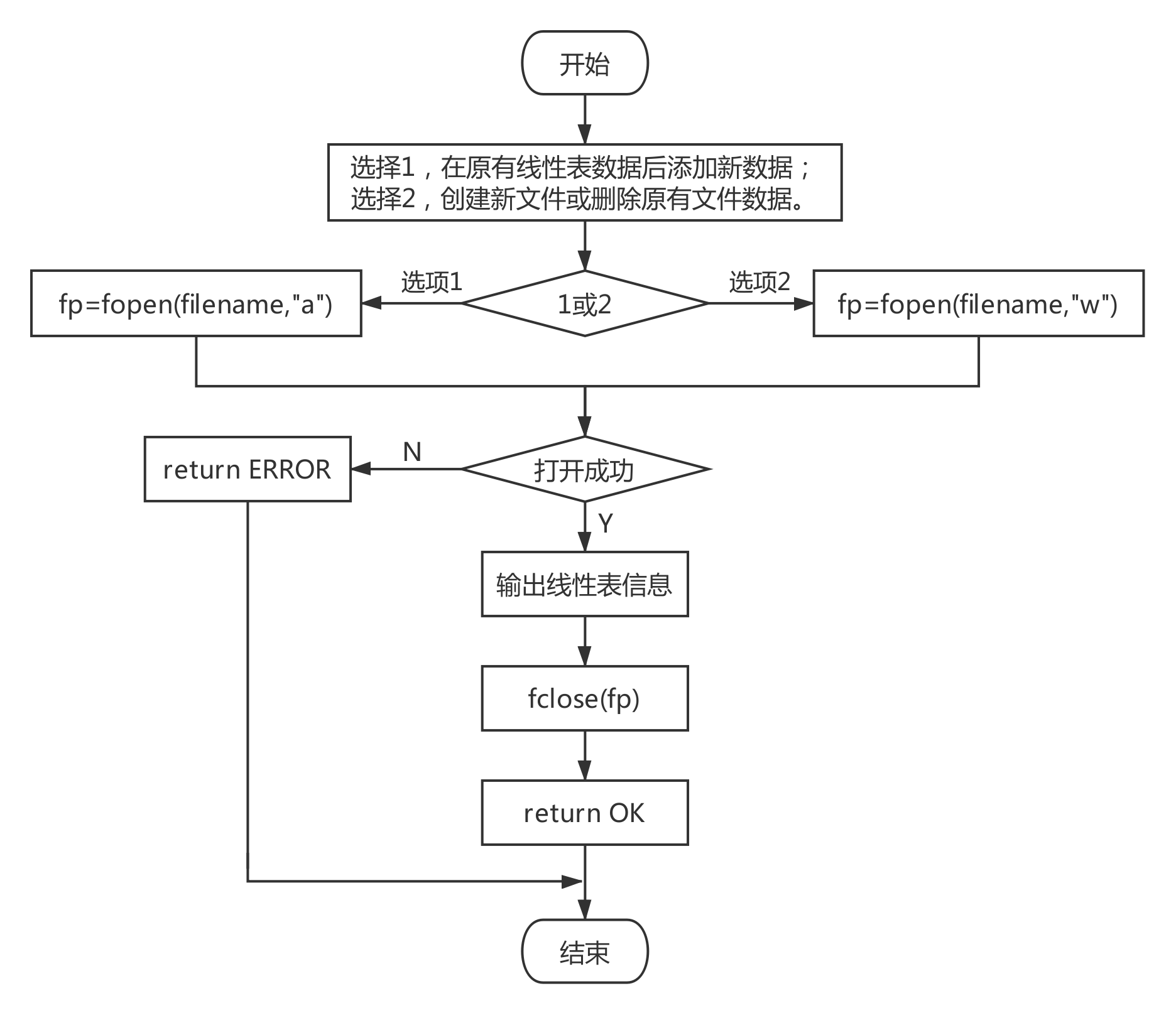


图1-4 文件保存函数流程图

**1.3.4 算法复杂度分析**

通过算法的分析与计算，确定了线性表的12个基本操作函数与自定义的三个文件处理与赋值函数的时间复杂度与空间复杂度（算法时间与空间复杂度表，如表1-1所示）。

表1-1 算法时间与空间复杂度表

|  |  |  |
| --- | --- | --- |
| 函数名称 | 时间复杂度T(n) | 空间复杂度S(n) |
| InitialList | O(1) | O(1) |
| DestroyList | O(1) | O(1) |
| ClearList | O(1) | O(1) |
| ListEmpty | O(1) | O(1) |
| ListLength | O(1) | O(1) |
| GetElem | O(n) | O(1) |
| LocateElem | O(n) | O(1) |
| PriorElem | O(N) | O(1) |
| NextElem | O(N) | O(1) |
| ListInsert | O(n) | O(1) |
| ListDelete | O(n) | O(1) |
| ListTrabverse | O(n) | O(1) |
| OpenFile | O(n) | O(1) |
| SetValue | O(n) | O(1) |
| SaveFile | O(n) | O(1) |

**1.4 系统测试**

1. 当程序开始运行时，会立刻进入菜单演示界面（菜单界面如图1-5所示）。

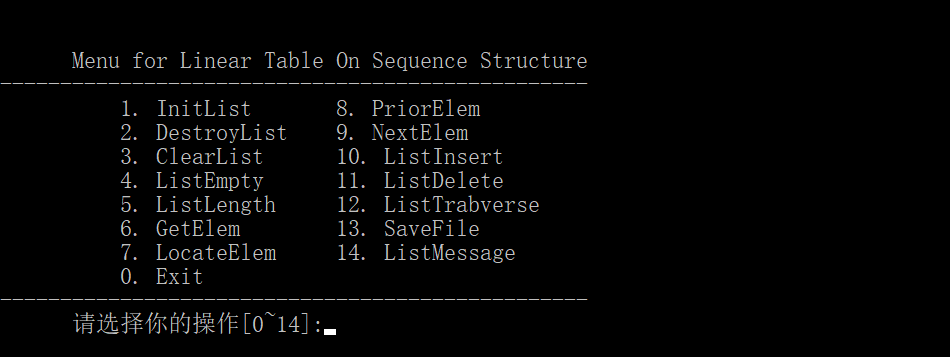


图1-5 演示系统菜单界面图

1. 按照菜单的提示输入需要完成的操作的序号，程序就会进入相对应的操作过程；当系统中没有线性表存在时，进行操作2-13，系统会做出提示显示“当前无线性表存在，请先创建线性表”（操作结果图如图1-6所示）。

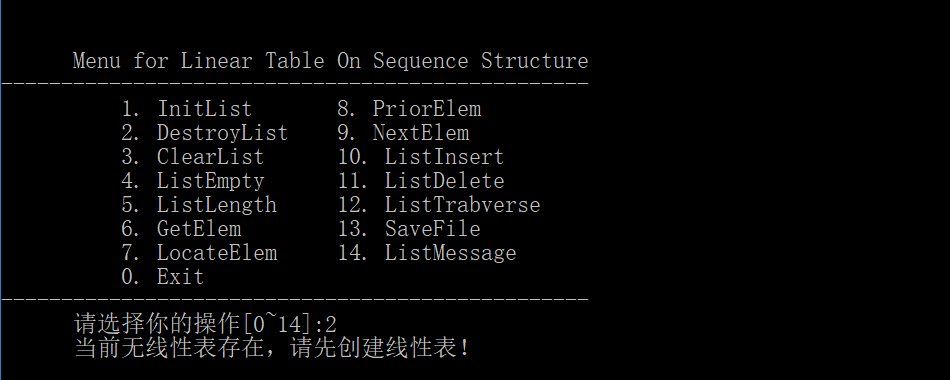


图1-6 系统无线性表存在时进行2-13操作结果图

1. 当系统中存在线性表时，可以进行所有操作，在进行2-13操作时，需要输入线性表的名称确定需要操作的线性表（操作界面图如图1-7所示）。

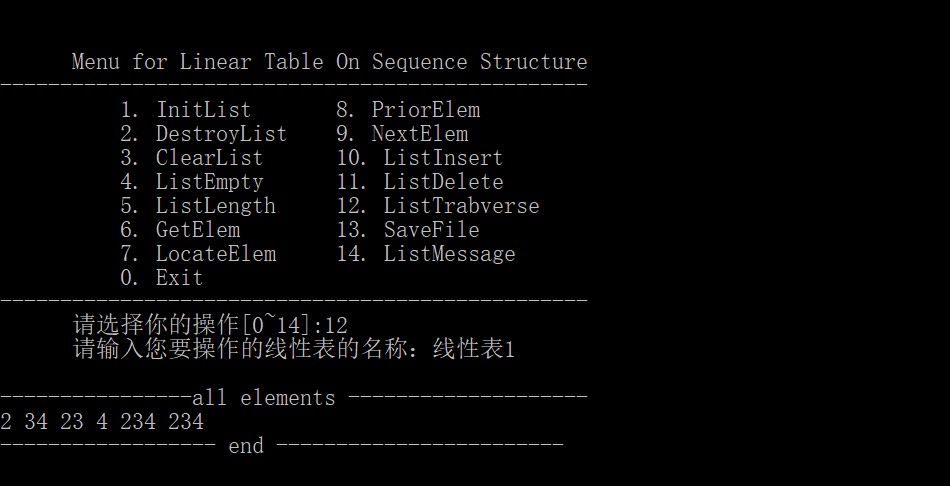


图1-7 输入线性表名称进行操作界面图

1. 当需要进行多表操作时，若需要查看所有线性表的信息，则可使用选项14操作，显示线性表信息（操作结果图如图1-8所示）。

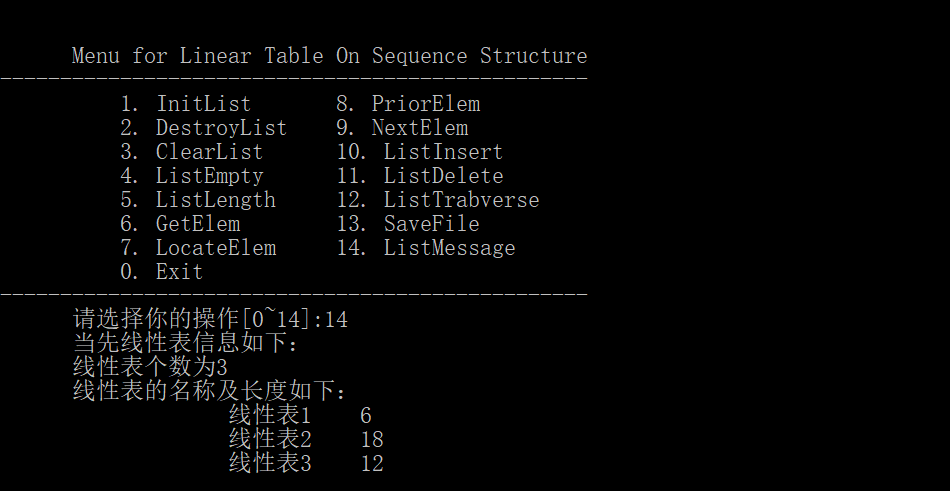


图1-8 显示多线性表信息操作结果图

1. 系统测试计划

使用预先保存的测试样例进行测试，本次测试挑选DestroyList、LocateElem、PriorElem、ListInsert、SaveFile共5个函数进行测试。本次使用的测试样例为多表测试样例，样例中包含4个线性表，其信息如下：

List1（表中包含5,12,0,44,94,137,47,3,9）；

List2（表中包含2,5,7,9,13,15,23）；

List3（空表）；

List4（表中包含4,23,5,33,7,23,5,7,77）；

其中，List1、List2与List3为键盘赋值（键盘赋值操作图如图1-9所示），List4为文件赋值（文件赋值操作图如图1-10所示）。

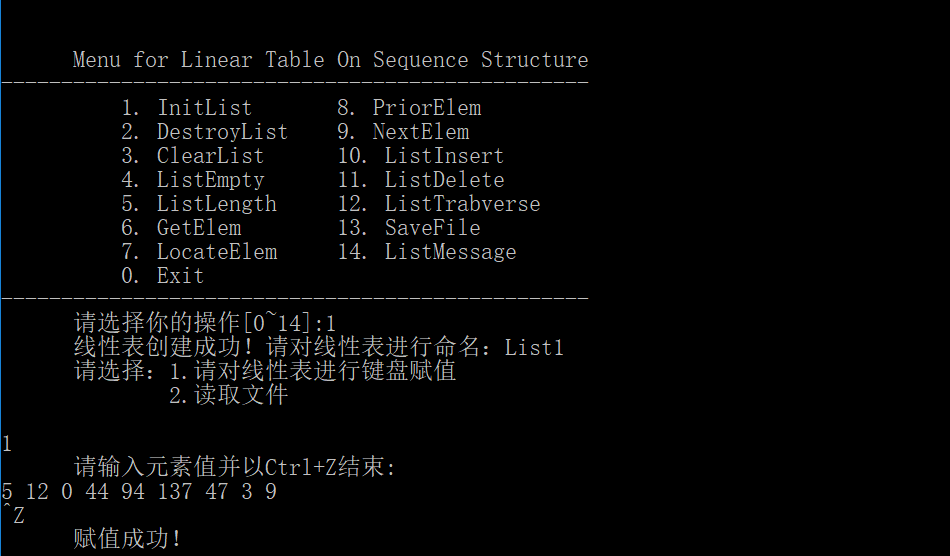


图1-9 线性表通过键盘赋值结果图

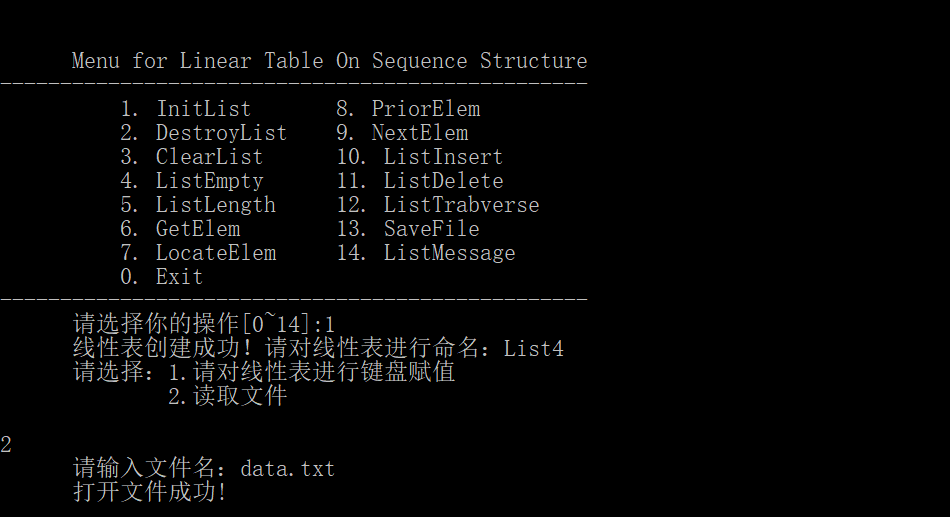
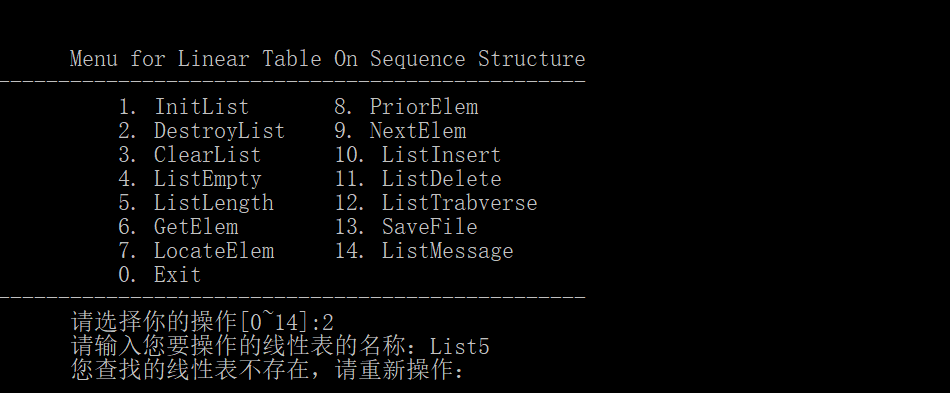


图1-10 线性表通过文件赋值结果图

1. DestroyList函数测试：

表1-2 DestroyList函数测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试线性表 | 程序输入 | 理论输出 | 实际输出 |
| 无 | 1. 主界面输入2进入操作； 2. 按提示输入“List5”作为需要销毁的线性表，此线性表不存在。 | 线性表不存在 | （失败结果图如图1-11所示） |
| List2 | 1. 主界面输入2进入操作； 2. 按提示输入“List2”作为需要销毁的线性表。 | 线性表销毁成功！ | List2线性表删除成功（结果图如图1-12所示） |

图1-11 线性表不存在时显示销毁失败结果图

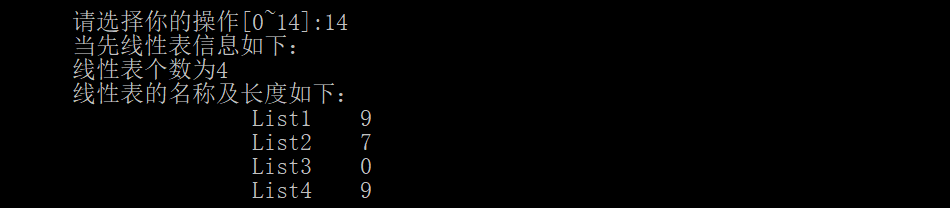
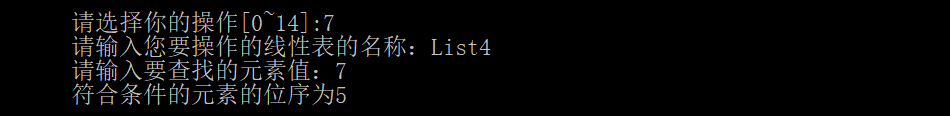
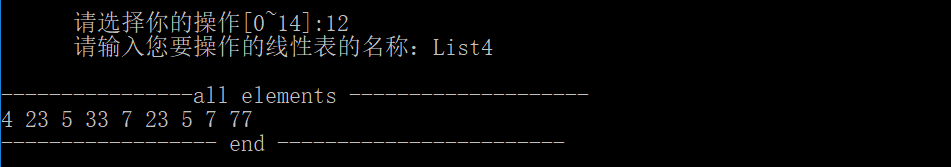


图1-12 线性表销毁前与销毁成功结果对比图

1. LocateElem函数测试：

表1-3 LocateElem函数测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试线性表 | 程序输入 | 理论输出 | 实际输出 |
| List4 | 1. 主界面输入7进入操作； 2. 按提示输入“List4”，按照提示输入“7”作为需要查找的元素。 | 符合条件的元素的位序为5 | （输出结果图如图1-13所示） |
| List4 | 1. 主界面输入7进入操作； 2. 按提示输入“List4”，按照提示输入“6”作为需要查找的元素。 | 没有符合条件的元素存在 | （输出结果图如图1-13所示） |



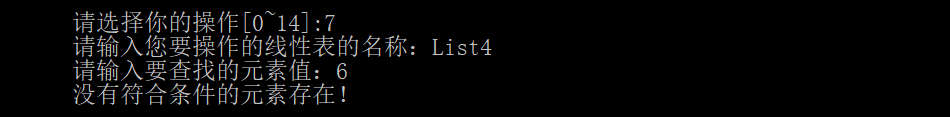
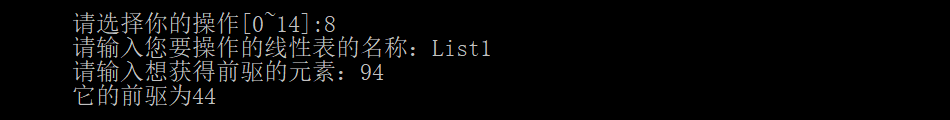
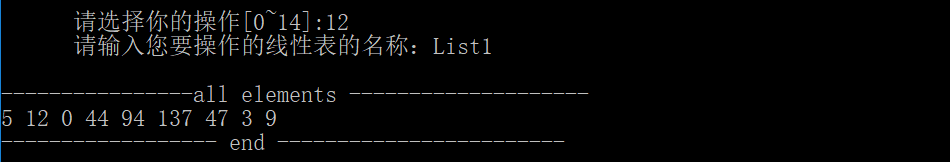


图1-13 查找元素函数测试结果图

1. PriorElem函数测试：

表1-4 PriorElem函数测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试线性表 | 程序输入 | 理论输出 | 实际输出 |
| List1 | 1. 主界面输入8进入操作； 2. 按提示输入“List1”，按照提示输入“94”作为需要查找前驱的元素。 | 它的前驱为44 | （输出结果图如图1-14所示） |
| List1 | 1. 主界面输入7进入操作； 2. 按提示输入“List1”，按照提示输入“5”作为需要查找前驱的元素。 | 该元素不存在前驱 | （输出结果图如图1-14所示） |



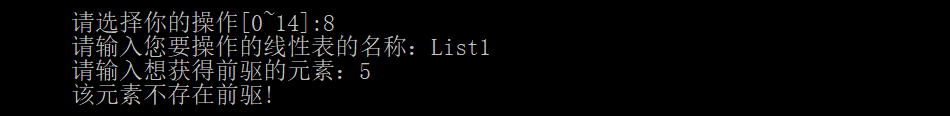


图1-14 查找元素前驱函数测试结果图

1. ListInsert函数测试：

表1-5 ListInsert函数测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试线性表 | 程序输入 | 理论输出 | 实际输出 |
| List1 | 1. 主界面输入10进入操作； 2. 按提示输入“List1”，输入“10”作为插入位点，“15”为插入元素。 | 操作成功 | （输出结果图如图1-15所示） |
| List1 | 1. 主界面输入10进入操作； 2. 按提示输入“List1”，输入“5”作为插入位点，“19”为插入元素。 | 操作成功 | （输出结果图如图1-15所示） |

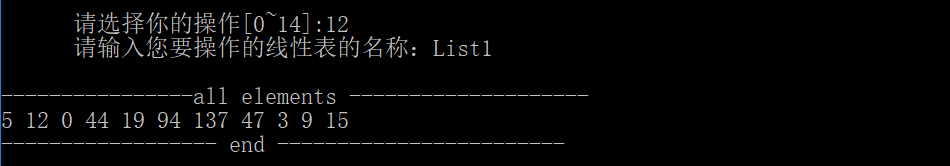
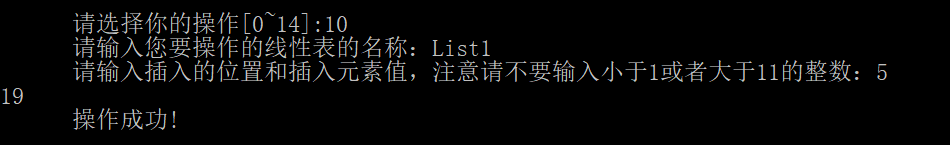
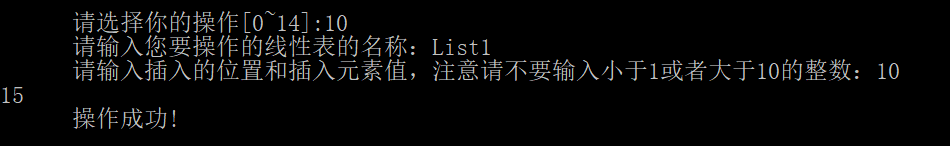
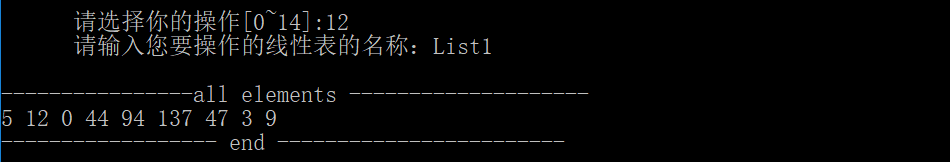
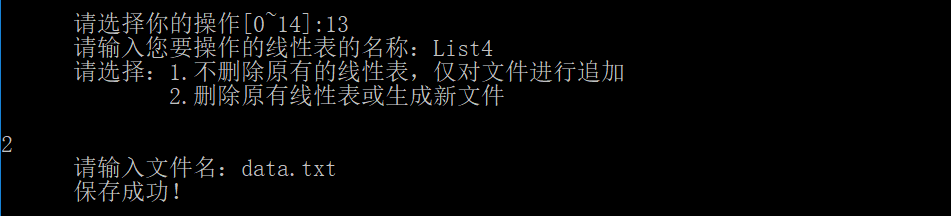


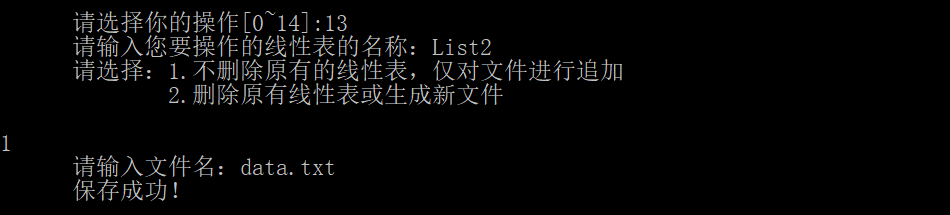
图1-15 插入元素函数结果图

1. SaveFile函数测试：

表1-6 SaveFile函数测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试线性表 | 程序输入 | 理论输出 | 实际输出 |
| List4 | 1. 主界面输入13进入操作； 2. 按提示输入“List4”，按照提示输入“2”创建新文件进行文件保存。 3. 输入文件名“data.txt” | 保存成功 | （输出结果图如图1-16所示） |
| List2 | 1. 主界面输入13进入操作； 2. 按提示输入“List4”，按照提示输入“1”在旧文件后添加数据。 3. 输入文件名“data.txt” | 保存成功 | （输出结果图如图1-16所示） |





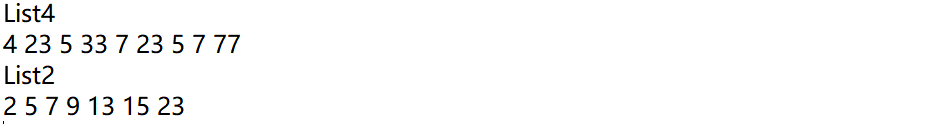


图1-16 文件保存函数测试结果图

## 1.5 实验小结

在本次实验中，我使用的语言为C语言，但是为了传递数据的方便，特意学习和使用了引用传递（C++语言中较为常用）作为传递参数的手段。通过代码的设计和编写完成了线性表的顺序存储。在编写过程中，我充分地了解到了线性表的顺序物理存储结构的编写过程和实现原理，并且熟练掌握了线性表的基本操作。

在设计算法和编写代码的过程，我遇到了一些问题，但是在同学的帮助和自身的努力下，最终我克服了问题，顺利地完成了本次实验的代码实现。其中，最困扰我的问题是关于文件的处理和多表操作，关于文件处理和多表操作，具体实现我在算法设计里有详细描述，这里就不赘述了，不过关于多表操作，为了方便查看多表信息和进行多表管理，我特意设计了14选项用于显示多表信息，以便于操作，这个功能设计是我比较满意的。除此之外，C语言课程结课较早，导致我的代码编写有些生疏，不过在几天的编写过程中，我已经逐渐找回了编代码的感觉，代码编写也变得流畅不少。

通过本次实验，我发现了我自身的一些不足，以及对C语言本身某些操作的生疏，除此之外，单独学习了引用传递的使用，加强了对“&”引用符号的理解。