

**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201801**

**学 号： U201816030**

**姓 名： 车春池**

**指导教师： 周全**

**报告日期： 2019年 11月 7 日**

**计算机科学与技术学院**

目 录

[1 基于顺序存储结构的线性表实现 2](#_Toc458159879)

[1.1 问题描述 2](#_Toc458159880)

[1.2 系统设计 2](#_Toc458159882)

[1.3 系统实现 2](#_Toc458159883)

[1.4 实验小结 2](#_Toc458159884)

[2 基于链式存储结构的线性表实现 2](#_Toc458159885)

[2.1 问题描述 2](#_Toc458159886)

[2.2 系统设计 2](#_Toc458159887)

[2.3 系统实现 2](#_Toc458159888)

[2.4 实验小结 2](#_Toc458159889)

[3 基于二叉链表的二叉树实现 2](#_Toc458159890)

[3.1 问题描述 2](#_Toc458159891)

[3.2 系统设计 2](#_Toc458159892)

[3.3 系统实现 2](#_Toc458159893)

[3.4 实验小结 2](#_Toc458159894)

[4 基于二叉链表的二叉树实现 2](#_Toc458159895)

[4.1 问题描述 2](#_Toc458159896)

[4.2 系统设计 2](#_Toc458159897)

[4.3 系统实现 2](#_Toc458159898)

[4.4 实验小结 2](#_Toc458159899)

[参考文献 2](#_Toc458159900)

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

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

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

[附录D 基于邻接表图实现的源程序 2](#_Toc458159904)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

依据最小完备性和常用性相结合的原则，以函数形式定义线性表的初始化表、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。采用顺序表作为线性表的物理结构，实现线性表基本运算。其中ElemType为数据元素的类型名，具体含义可自行定义。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。演示系统实现线性表的文件形式保存。演示系统实现多个线性表管理。

## 1.1.1 线性表抽象数据类型

ADT Sqlist{

数据对象：D={ai|ai属于ElemSet,i=1,2,...n,n>=0}

数据关系：Rl={<ai-1,ai>|ai-1,ai属于D,i=1,2,...n}

基本操作：

InitList(&L)

初始条件：线性表L不存在

操作结果：构造一个空线性表

DestroyList（&L）

初始条件：线性表L存在

操作结果：摧毁线性表L

ClearList（&L）

初始条件：线性表存在

操作结果：将L重置为空表

ListEmpty（&L）

初始条件：线性表L存在

操作结果：L为空表返回TRUE，否则返回FALSE

ListLength（&L）

初始条件：线性表L存在

操作结果：返回L中数据元素的个数

GetElem（L,i,e）

初始条件：线性表L存在,1≤i≤ListLength(L)

操作结果：用e返回L中第i个数据元素的值

LocateElem（L,e,compare()）

初始条件：线性表存在

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。

PriorElem(L,cur\_e,pre\_e)

初始条件：线性表存在

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。

ListInsert(L,i,e)

初始条件：线性表存在

操作结果：在L的第i个位置之前插入新的数据元素e

ListDelete(L,i,e)

初始条件：线性表存在

操作结果：删除L的第i个数据元素，用e返回其值

ListTraverse(L,visit())

初始条件：线性表存在

操作结果：依次对L的每个数据元素调用函数visit()

ListSet(L)

初始条件：线性表存在

操作结果：对线性表进行排序

}ADT SqList

## 1.1.2 多线性表的管理

我定义了一个结构体指针数组，用来存储指向各个线性表的指针值，指针在数组中对应的下标就是相应线性表的标识（从0开始），在演示菜单中可以实现各个线性表的切换。

## 1.1.3 演示系统和文件存储的设计

演示系统借鉴了A中给出的框架，该框架将完成函数调用所需实参值的准备和函数执行结果的实现，并给出适当的操作提示显示，整体以命令行呈现。

此外设计了数据文件实时存储，文件存储的操作包括在演示系统上，用户可以自行选择是否将当前线性表存储，存储方式为将当前的线性表结构体和数据区域直接将内存区块写入文件中。文件读取操作也包括在演示系统里面，读取时用户可以选择当前目录中已经保存的文件加载，来还原该文件所存储的一个线性表的数据。

## 1.2 系统设计

## 1.2.1 数据物理结构

1.线性表的存储数据结构

线性表结构体定义如下：

//the main struct of linear list

typedef struct

{

//顺序表（顺序存储）的定义

ElemType\*elem;//数据元素

int length;//表当前的长度

int listsize; //表当前容量

} SqList,\*SqListP;

2.多线性表的存储数据结构

//ListNumber为管理的线性表个数

SqListP ListGroup[ListNumber];

3.宏定义

**#define TRUE 1**

**#define FALSE 0**

**#define OK 1**

**#define ERROR 0**

**#define INFEASTABLE -1**

**#define OVERFLOW -2**

**typedef int status;**

**typedef int ElemType;**

在本程序中，数据原子类型被定义为整型int

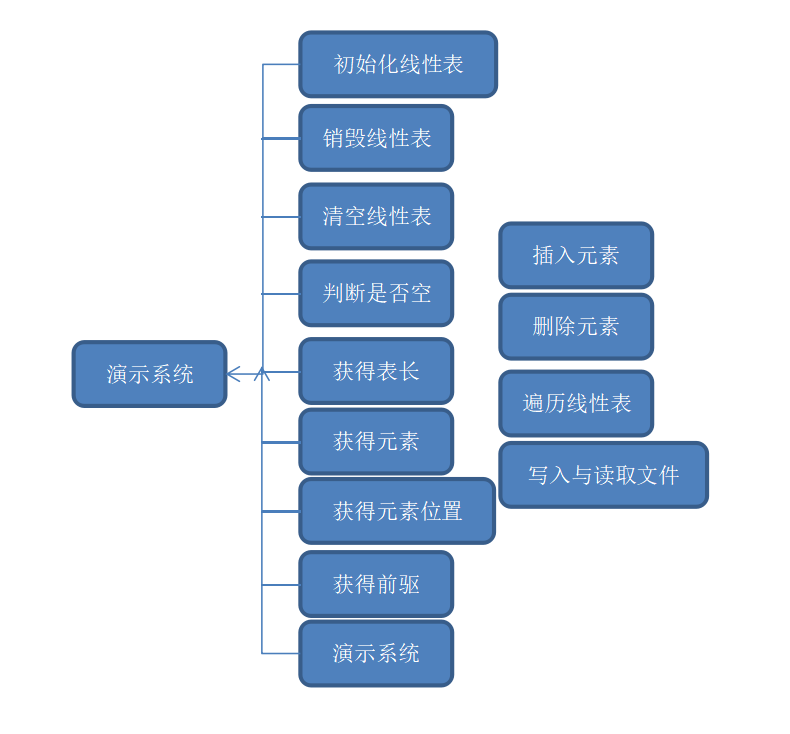
**1.2.2 演示系统**

演示系统包括用户操作界面和功能调用部分。

演示系统界面语言为英文，有的操作和提示语言为中文。

用户操作界面输出可选的线性表操作，用户输入数字选择要进行的操作。在用户选择操作后，系统提示用户输入参数，并且显示参数的信息。

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。



**1.2.3 线性表运算实现算法**

1. status InitList(SqListP L)

功能：初始化线性表

算法实现：为线性表L的elem区域申请空间，如果申请失败则返回ERROR，否则返回OK。

时空效率分析：算法的时间复杂度为O（1），空间上为L的elem区域申请空间，所以空间复杂度为O（1）。

1. status DestoryList(SqListP L)

功能：摧毁线性表

算法实现：如果L的elem区域不为空，则free掉elem区域，然后将整个L的表长置为0。

时空效率分析：算法的时间复杂度为O（1），空间复杂度为O（1）

1. status ClearList(SqListP L)

功能：清空线性表

算法实现：如果L的elem区域不为空，则将L的表长置为0

时空效率分析：算法的时间复杂度为O（1），空间复杂度为O（0）

1. status ListEmpty(SqListP L)

功能：判断线性表是否为空

算法实现：如果L的表长为0，则返回TRUE，否则返回FALSE

时空效率分析：时间复杂度为O（1），空间复杂度为O（1）

1. int ListLength(SqListP L)

功能：求线性表的长度

算法实现：直接返回线性表结构体中存储的L->length即可

时空效率分析：时间复杂度为O（1），空间复杂度为O（1）

1. status GetElem(SqListP L,int i,ElemType\* e)

功能：获得线性表中指定位置的数据

算法实现：首先判断指定位置是否合法，若不是则返回ERROR，否则把L->elem[i-1]的值赋给\*e，返回OK

时空效率分析：时间复杂度为O（1），空间复杂度为O（1）

1. int LocateElem(SqListP L,ElemType e,int(\*compare)(ElemType x,ElemType e))

功能：寻找指定元素在线性表中的位置

算法实现：遍历线性表的数据区域，如果当前数据元素与e符合compare（）关系，则，返回index+1，否则返回0

时空效率分析：时间复杂度为O（n），空间复杂度为O（1）

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

功能：获得指定元素之前的一个一个元素

算法实现：直接遍历寻找，然后把前一位位置的元素的地址赋值给pre\_e。如果找到就返回OK，否则返回ERROR。

时空效率分析：时间复杂度为O（n），空间复杂度为O（1）

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

功能：获得指定元素之后的一个元素

算法实现：直接遍历寻找，然后把后一位位置的元素的地址赋值给next\_e，如果找到就返回OK，否则返回ERROR

时空效率分析：时间复杂度为O（n），空间复杂度为O（1）

1. status ListInsert(SqListP L,int i,ElemType e)

功能：插入元素

算法实现：首先要判断插入的位置是否合法，如果不合法则返回ERROR，然后判断L->size和L->length是否相同，如果相同，则线性表已满，就再给线性表分配固定大小的内存空间，然后将指定位置之后的元素全部向后移动一个位置，将新的元素插入到指定位置，返回OK

时空效率分析：时间复杂度为O（n），空间复杂度为O（1）

1. status ListDelete(SqListP L,int i,ElemType\* e)

功能：删除元素

算法实现：首先要判断删除的位置是否合法，若不是则返回ERROR，然后将指定位置之后的元素全部向前移动一个位置，返回OK

时空效率分析：时间复杂度为O（n），空间复杂度为O（1）

1. status ListTrabverse(SqListP L,void(\*visit)(ElemType\*e))

功能：遍历线性表并将每个数据元素调用visit函数

算法实现：直接遍历并对每个数据元素调用visit（）函数即可

时空效率分析：时间复杂度O（n），空间复杂度O（1）

1. status ListSort(SqListP L)

功能：对线性表中的元素进行排序

算法实现：运用冒泡排序法对L中的元素进行排序

时空效率分析：时间复杂度O（n^2），空间复杂度O（1）

**1.2.4 多线性表管理实现算法**

由于多线性表管理采用结构体指针数组管理，只涉及到查找操作，而且查找操作直接利用数组下标，所以时间复杂度为O（1），空间复杂度为O（1）

**1.2.5 文件存储实现算法**

1. 写文件

算法实现：用户输入要保存的文件名，打开文件，根据线性表的L->size存入数据空间，将当前的线性表作为文件保存，数据保存完毕，关闭文件。

时空效率分析：时间复杂度为O（1），空间复杂度为O（n）

1. 读取文件

算法实现：用户输入要读取的文件名，打开文件，读取该文件的数据存入当前线性表，直到文件中的数据读入完成，关闭文件。

时空效率分析：时间复杂度为O（1），空间复杂度为（n）

## 1.3 系统实现

**1.3.1 实验环境**

实验环境为Windows10，编译器版本为TDM-GCC 4.7.1，代码采用开源的编译器Dev-C++编写。由指定的MakeFile来完成编译。

文件说明：

\*LinearList.h：线性表库头文件

\*test.cpp:线性表及演示系统实现

\*LinearList\_1.dev:DEV文件

\*LinearList\_1.exe:可执行文件

\*LinearList\_1.layout:LAYOUT文件

\*Makefile.win:自动化编译命令

**1.3.2 代码亮点**

所有代码书写借鉴Google C/C++标准代码规范，函数库在必要的地方加上了注释，符合生产规范。错误检查和提示全面，根据不同的错误信息会有不同的提示信息。

**1.3.3 操作演示**

**界面演示：**

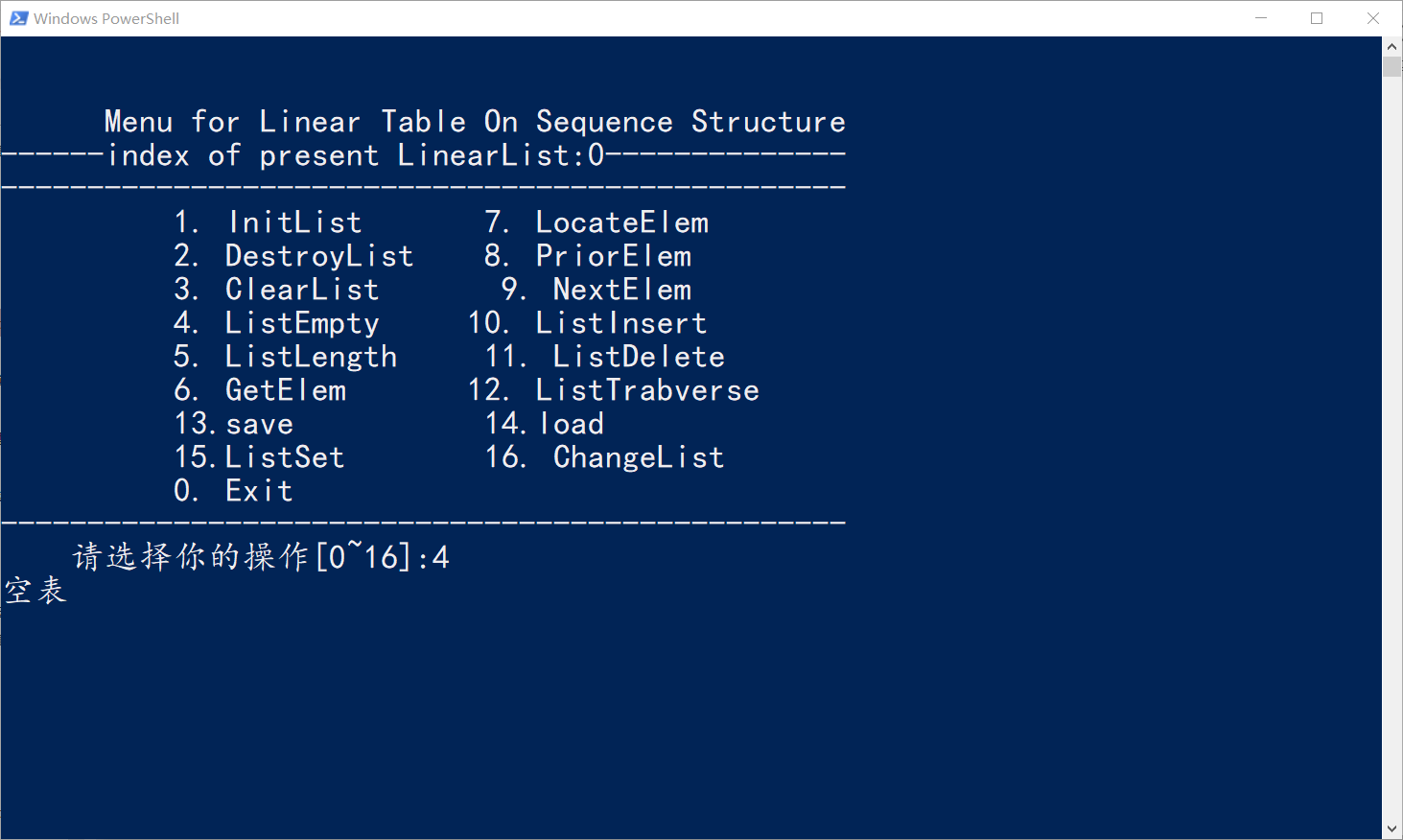
****

图1-2演示系统界面

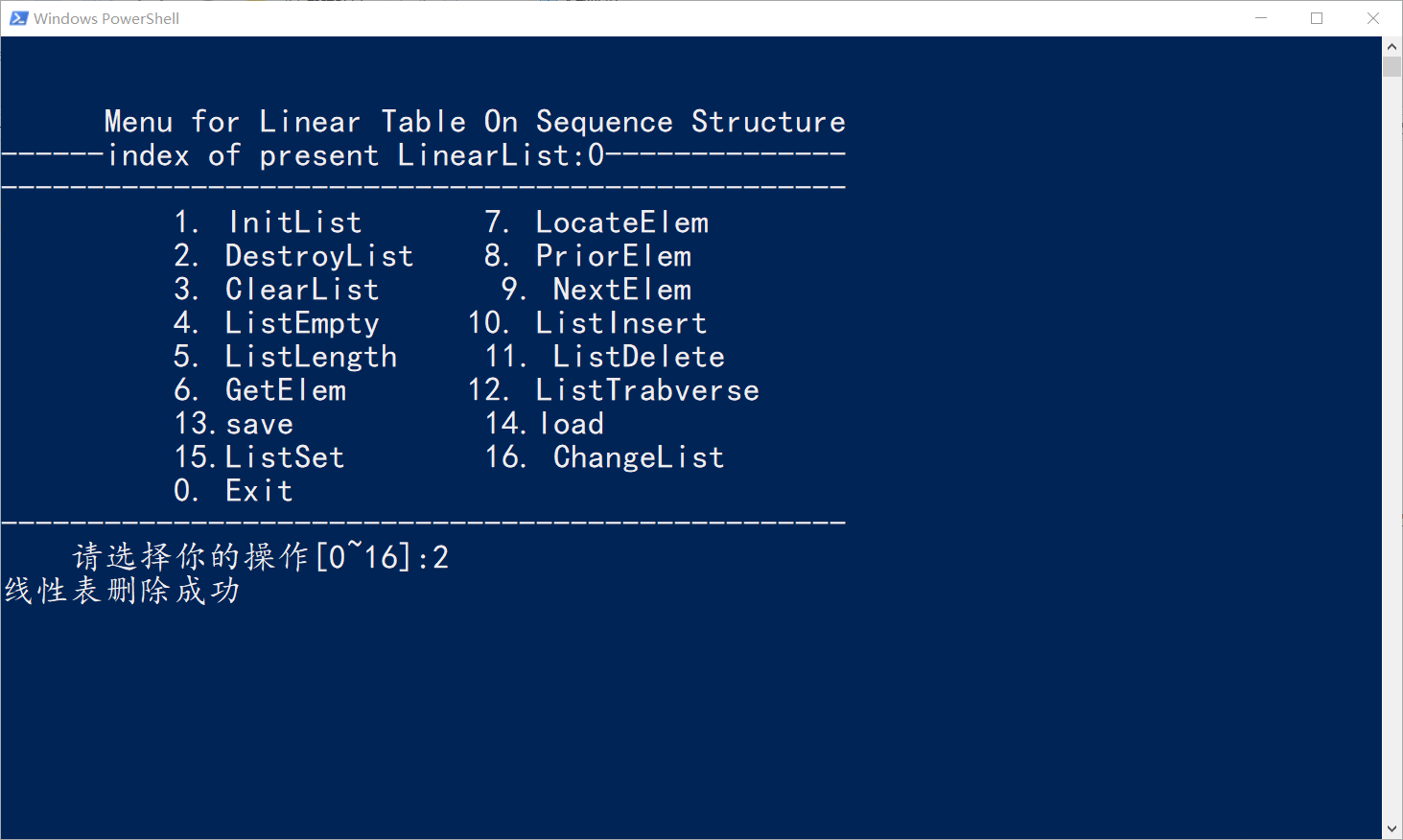
1. **首先创建一个线性表**

****

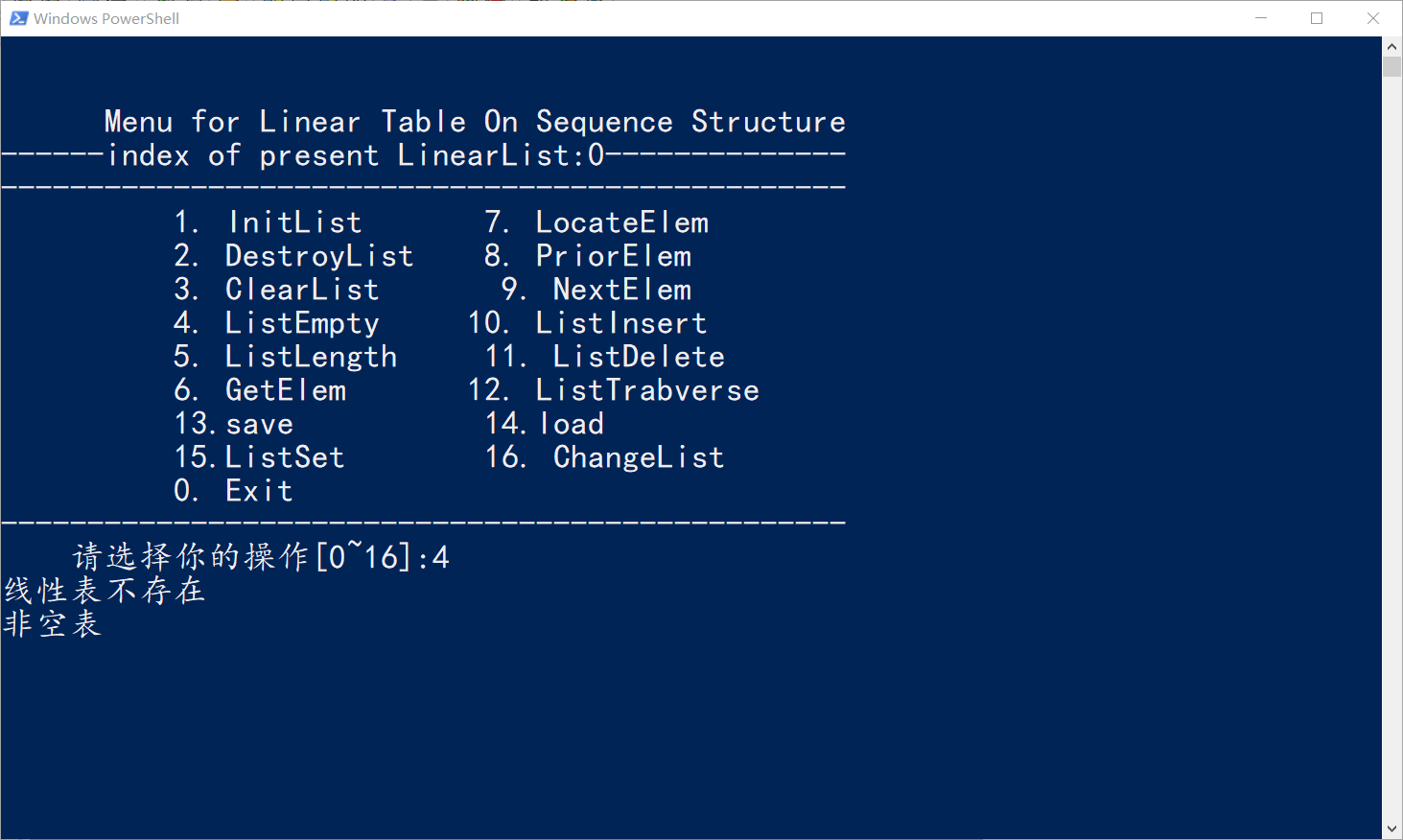
1. **然后用ListEmpty查看，显示这个新创建的表是空表**

****

1. **然后删除这个表**

****

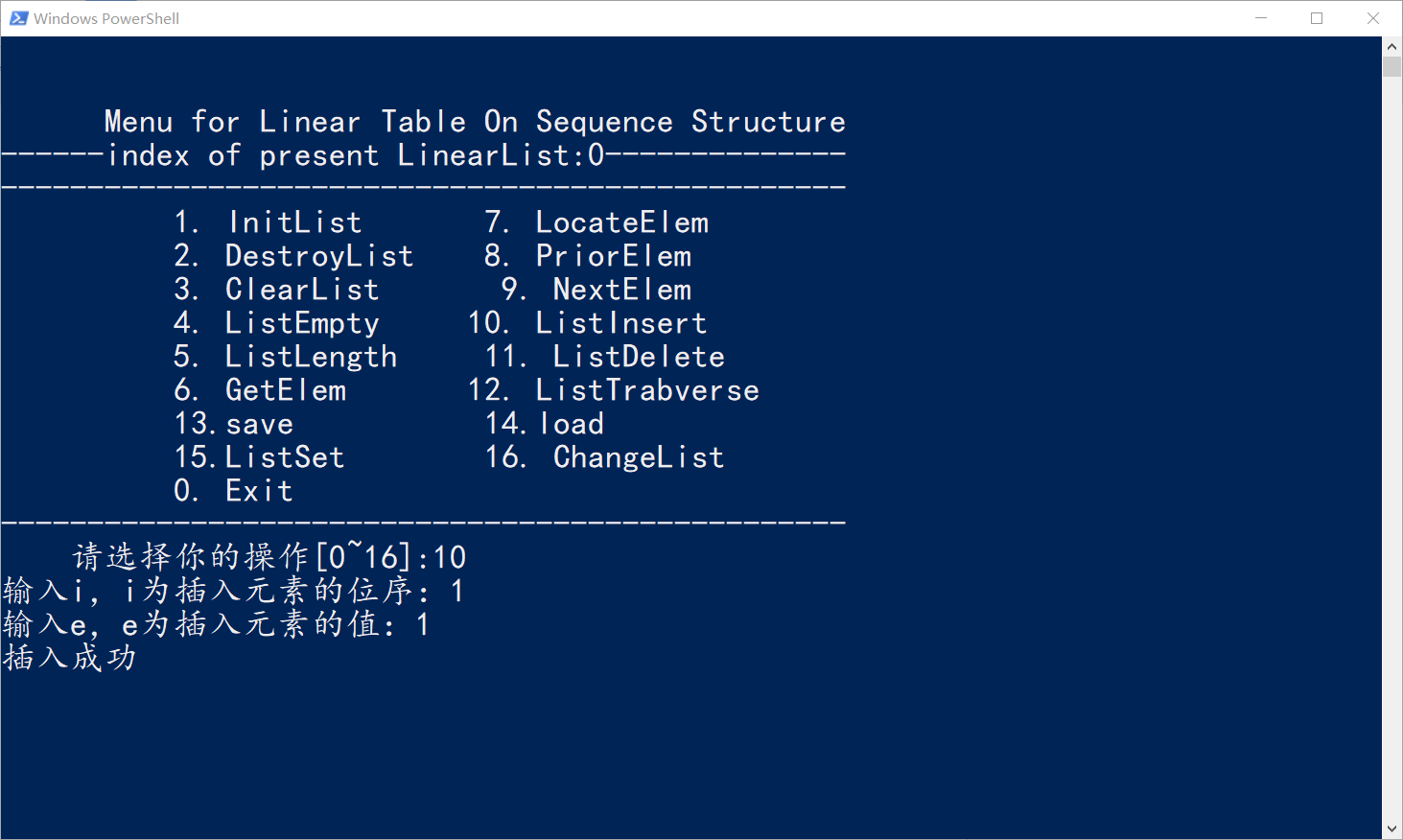
1. **再用ListEmpty查看，显示线性表不存在**

****

1. **重新创建一个线性表**

****

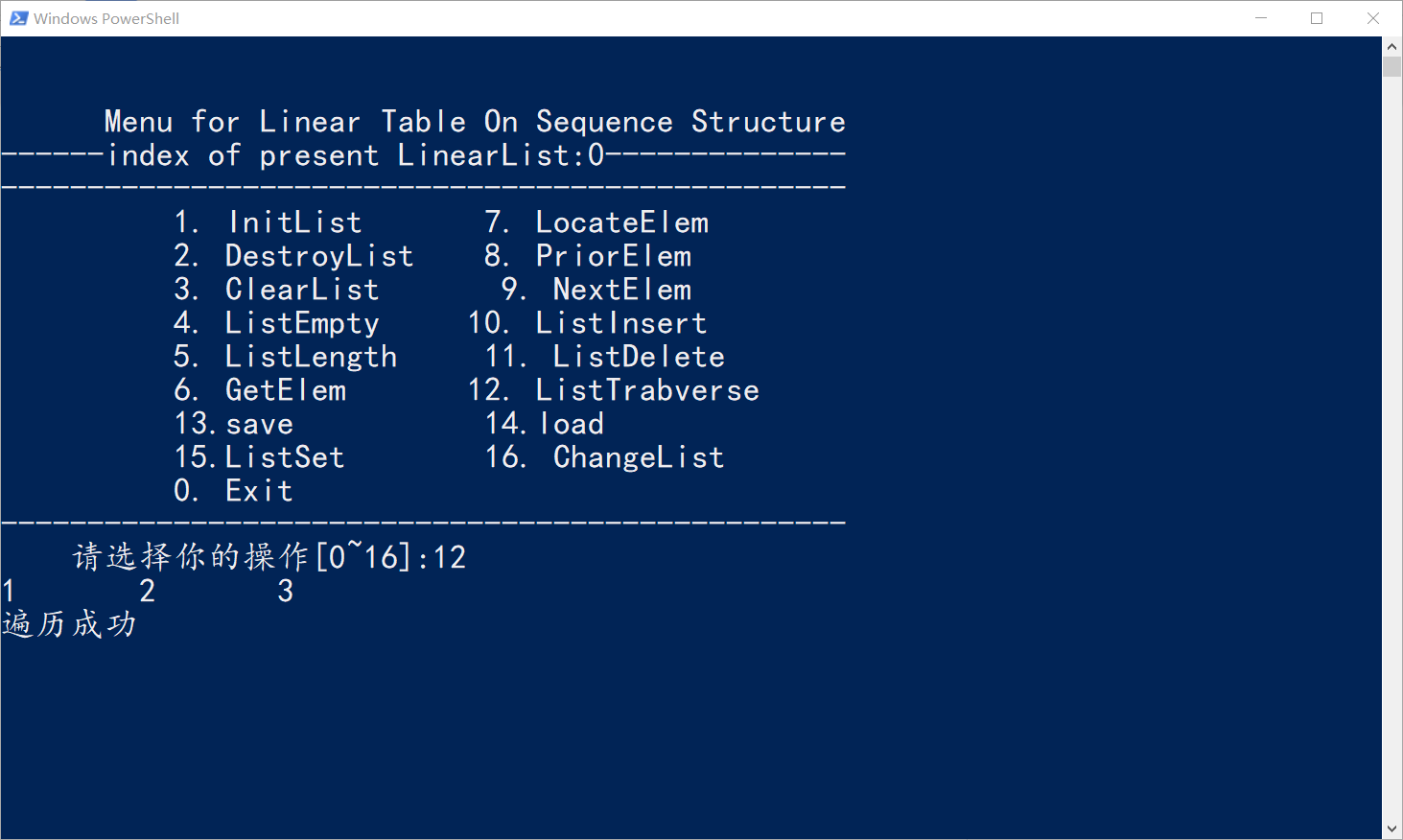
1. **依次按顺序插入三个元素1,2,3**

****

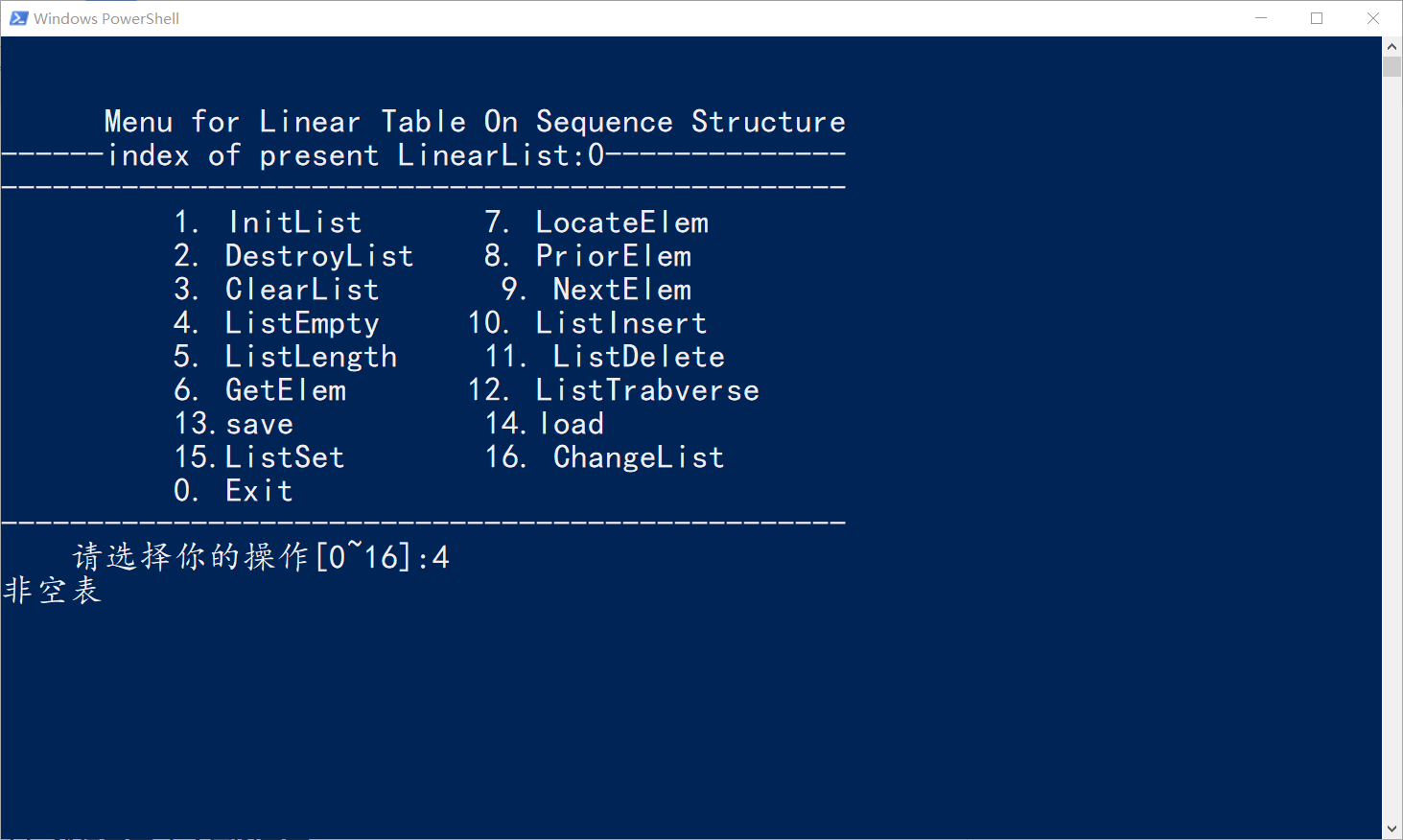
****

****

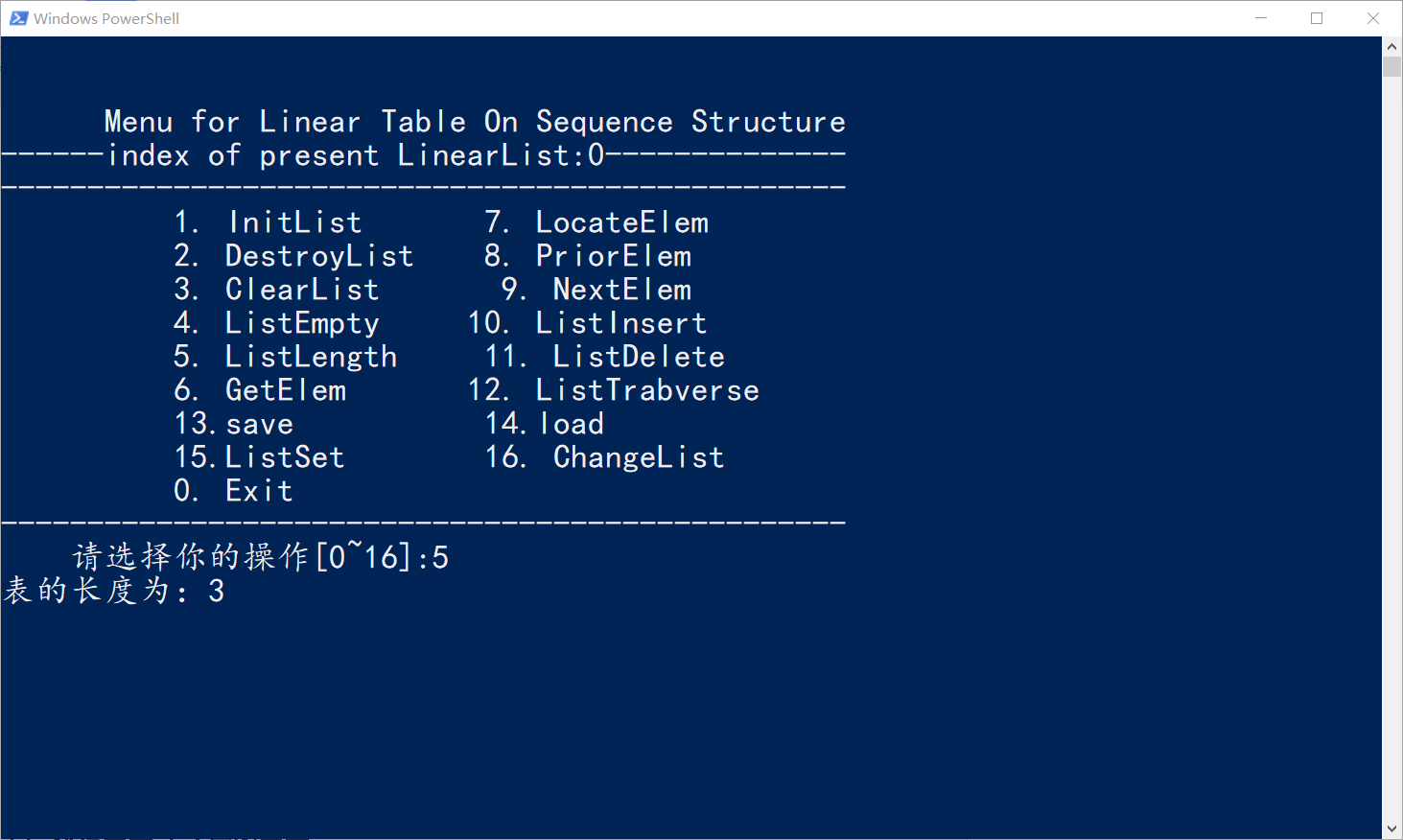
1. **打印这个表（操作12中的visit（x）函数是将x打印出来）**

****

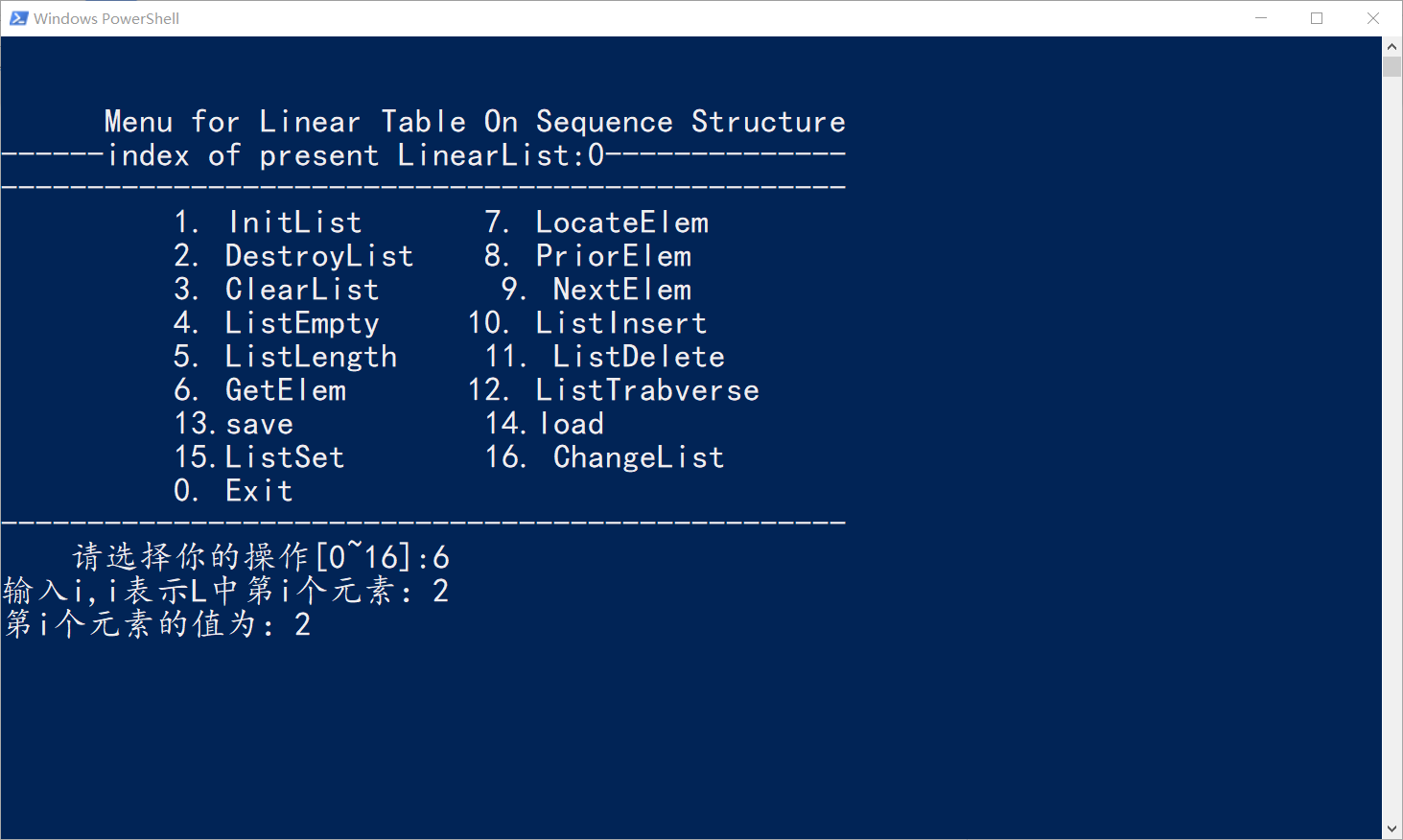
1. **用ListEmpty查看，显示这是个非空表**

****

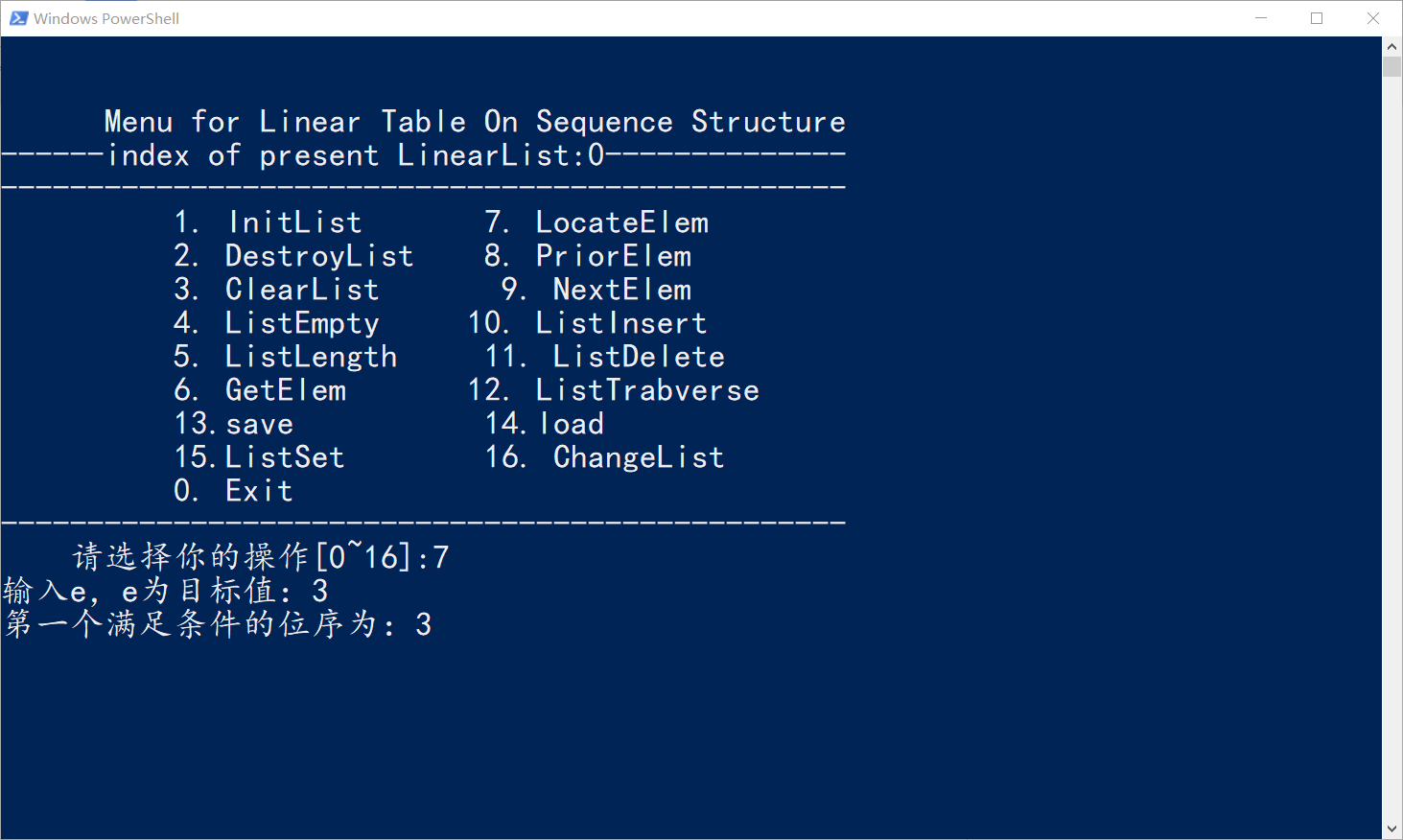
1. **查看这个表的长度，显示是3**

****

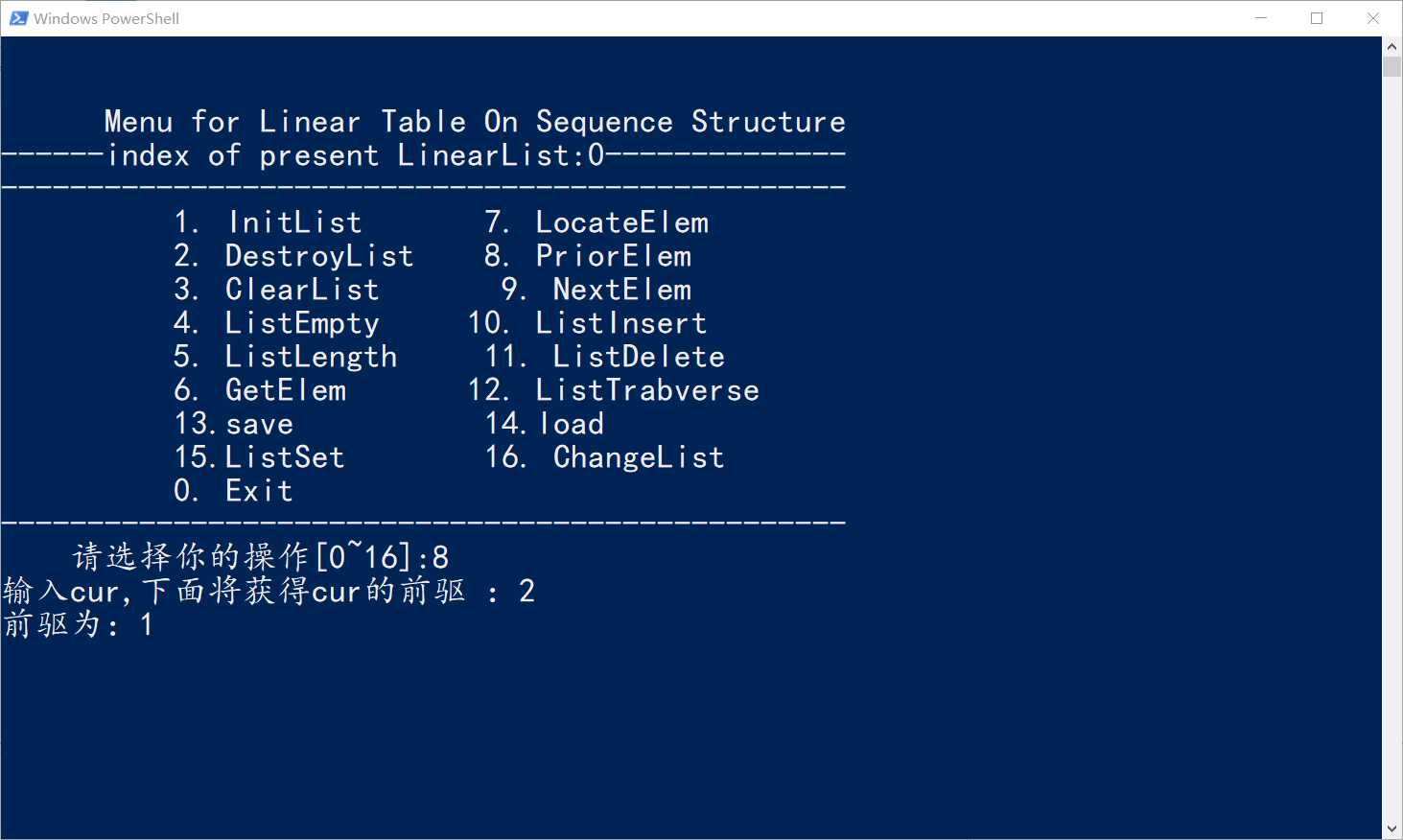
1. **获得这个表中第2个数据元素，显示是2**

****

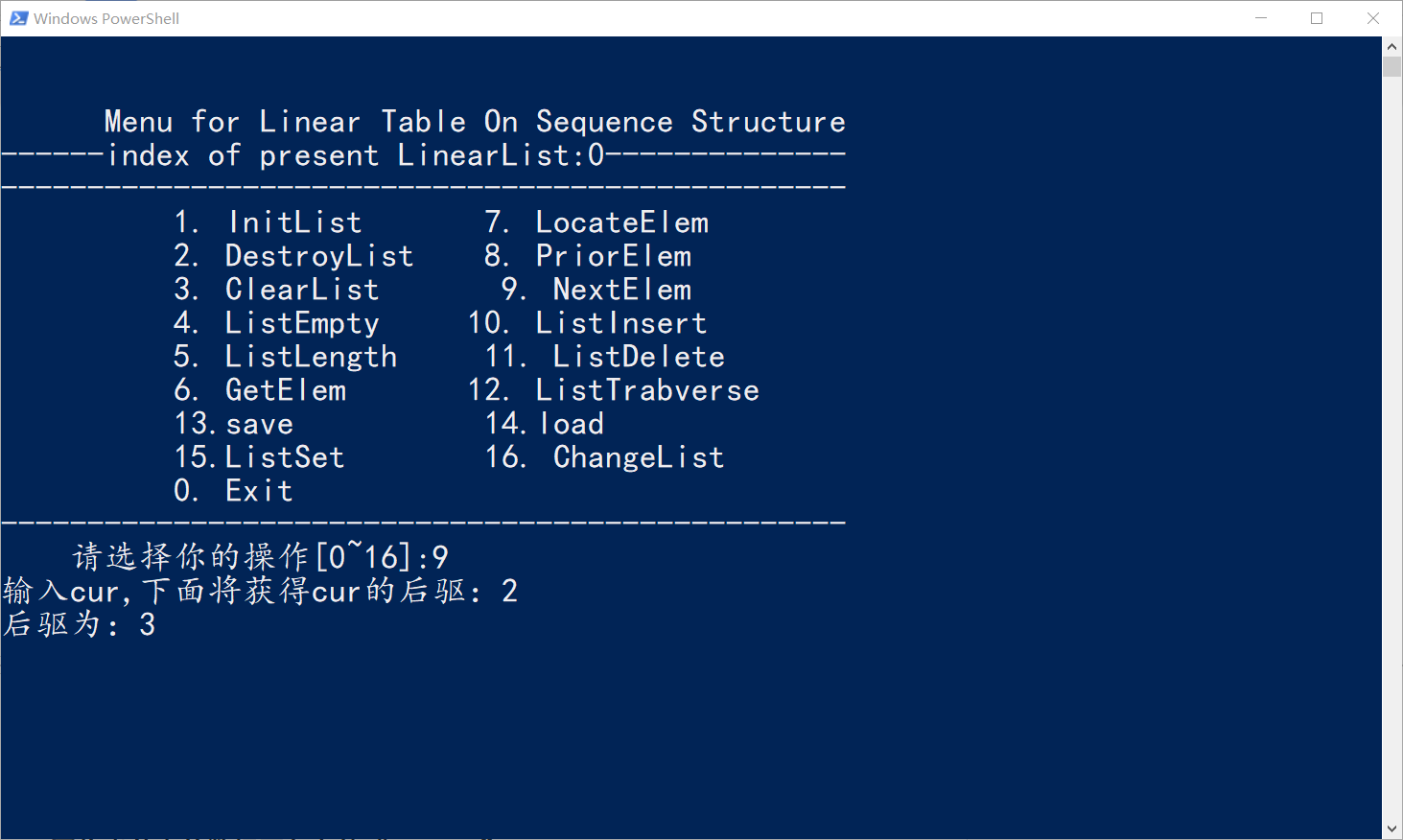
1. **获得这个表中第一个与3相等的数据元素（操作7的compare（x，y）函数是判断x与y是否相等）**

****

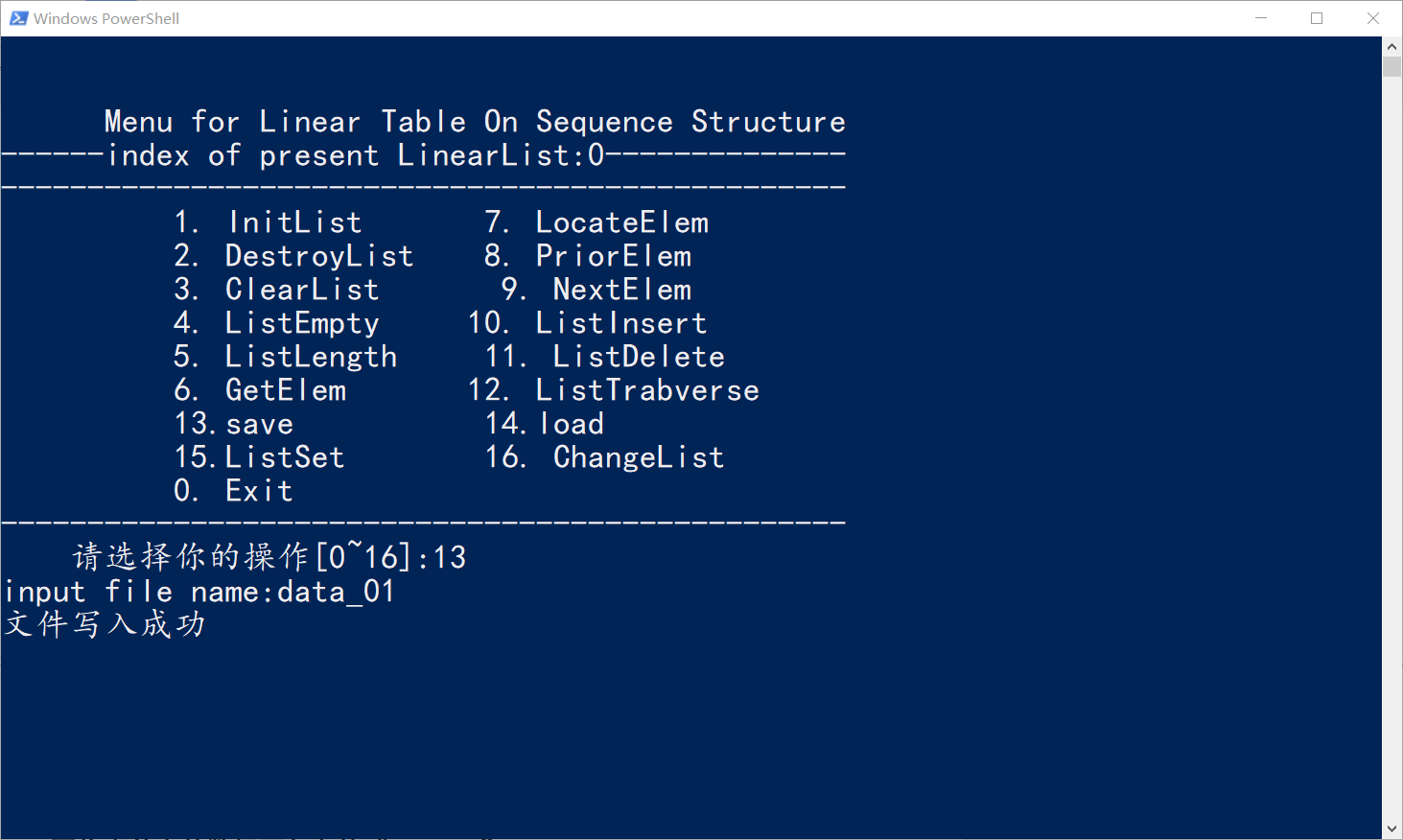
1. **获得元素2之前的元素**

****

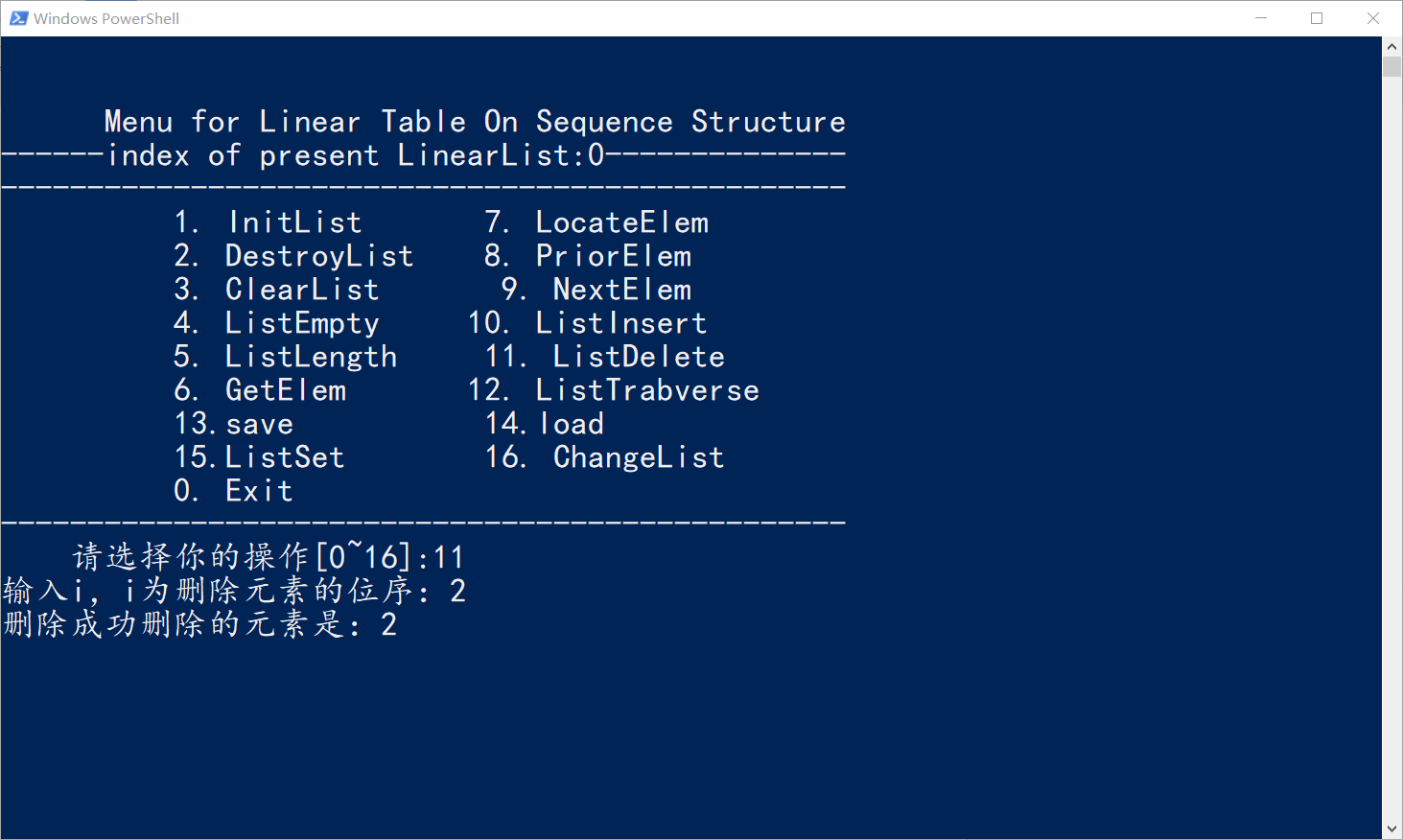
1. **获得元素2之后的元素**

****

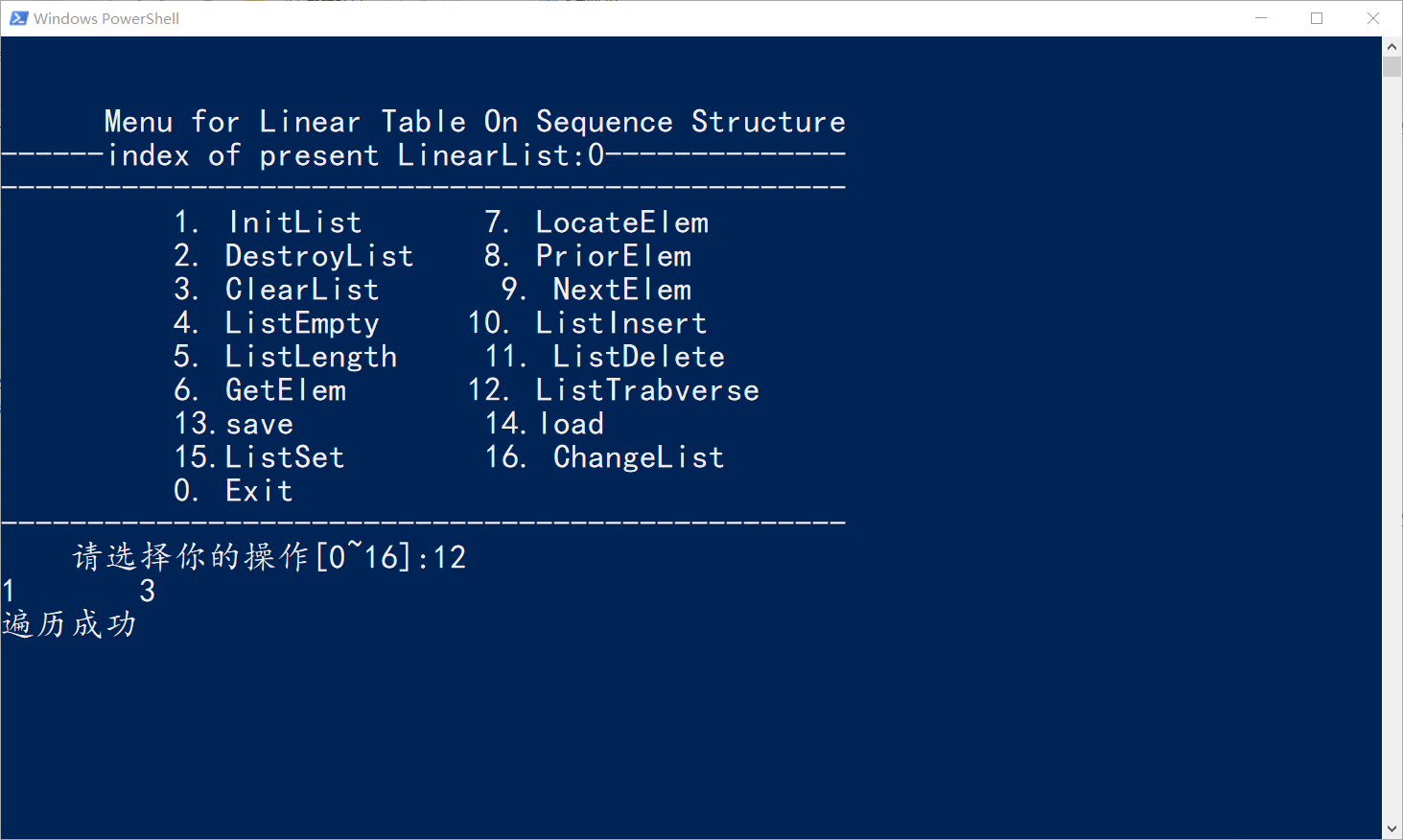
1. **将当前线性表的数据写入文件“data\_01”**

****

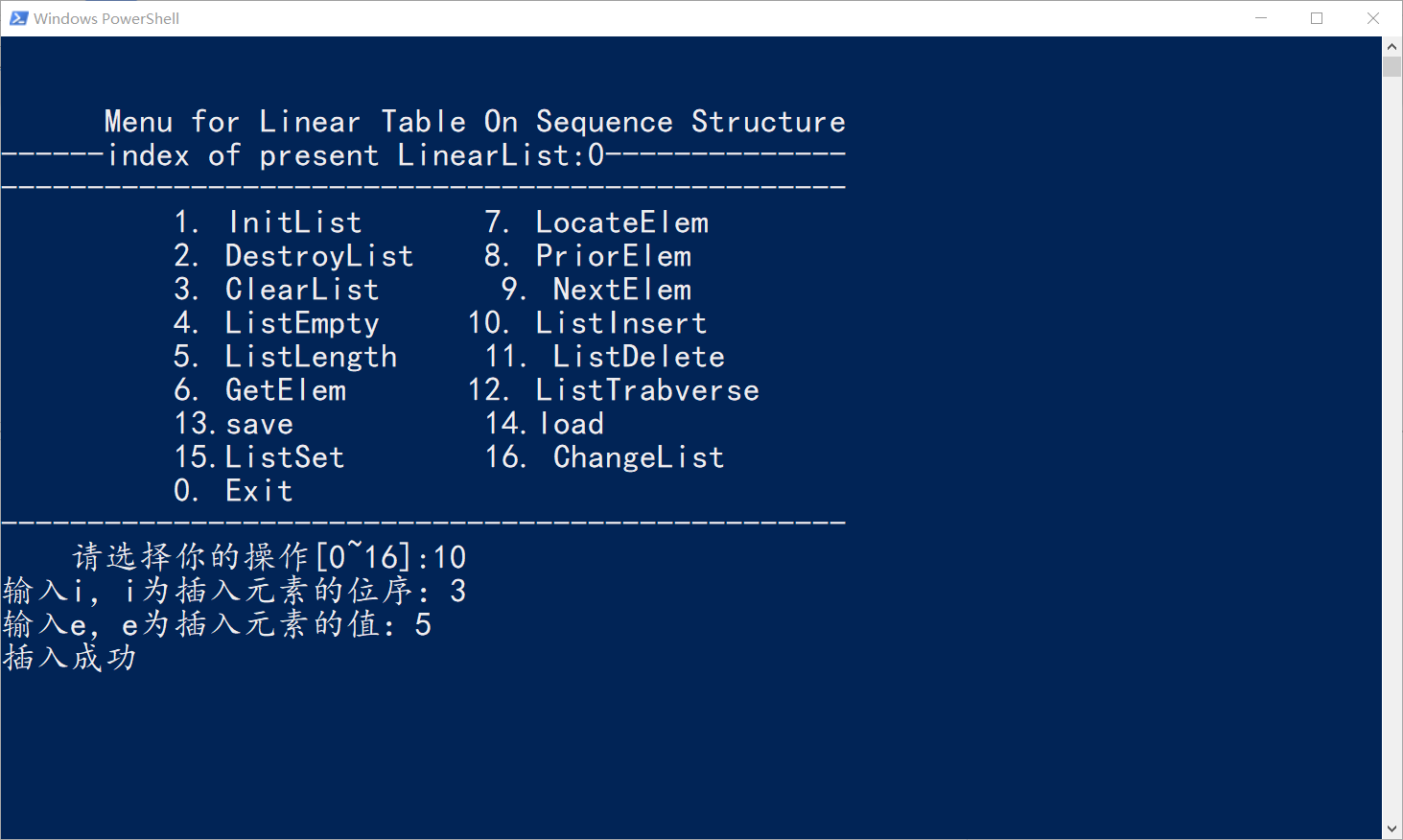
1. **删除当前表的第2个元素**

****

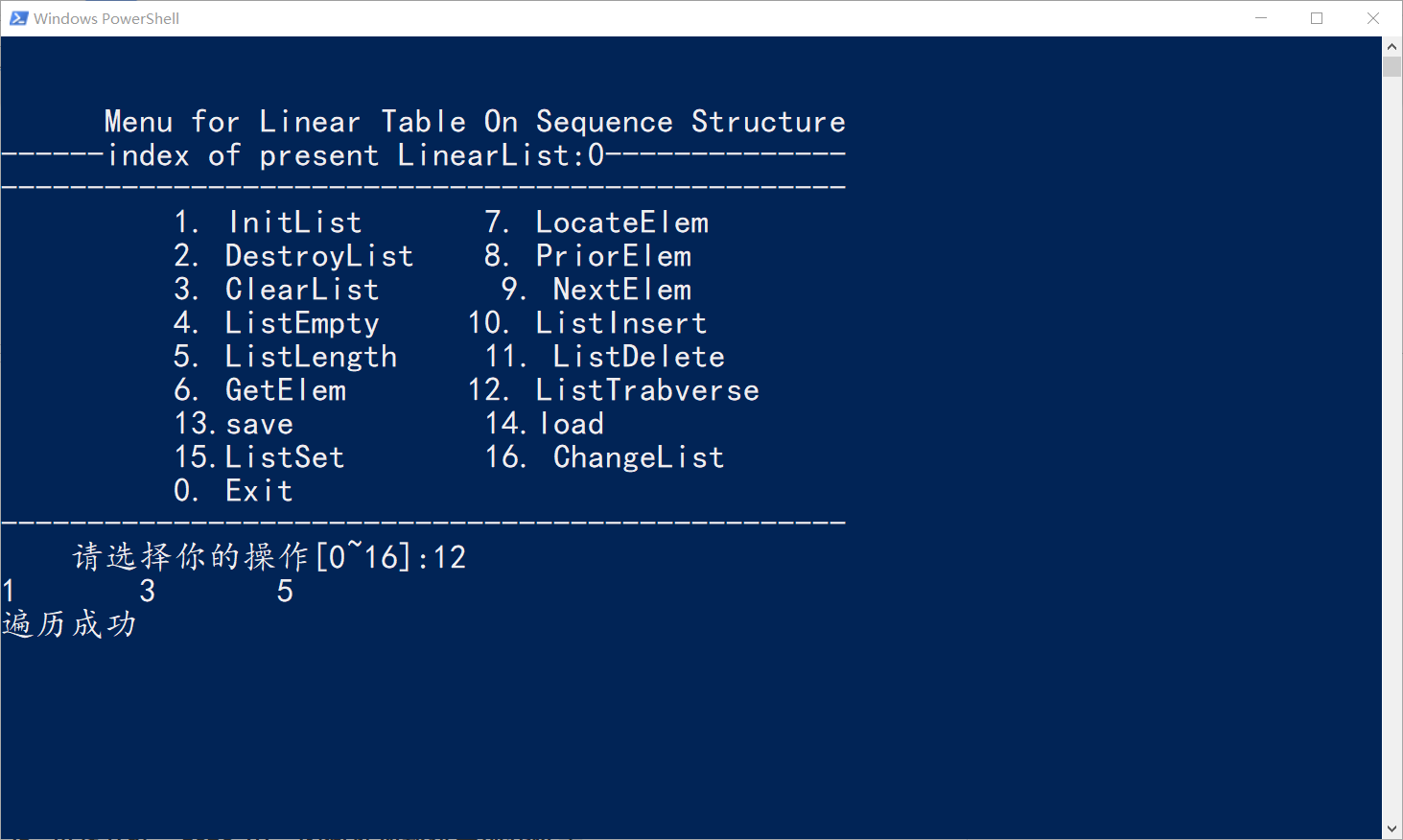
1. **打印这个表，发现第2个元素成功被删除**

****

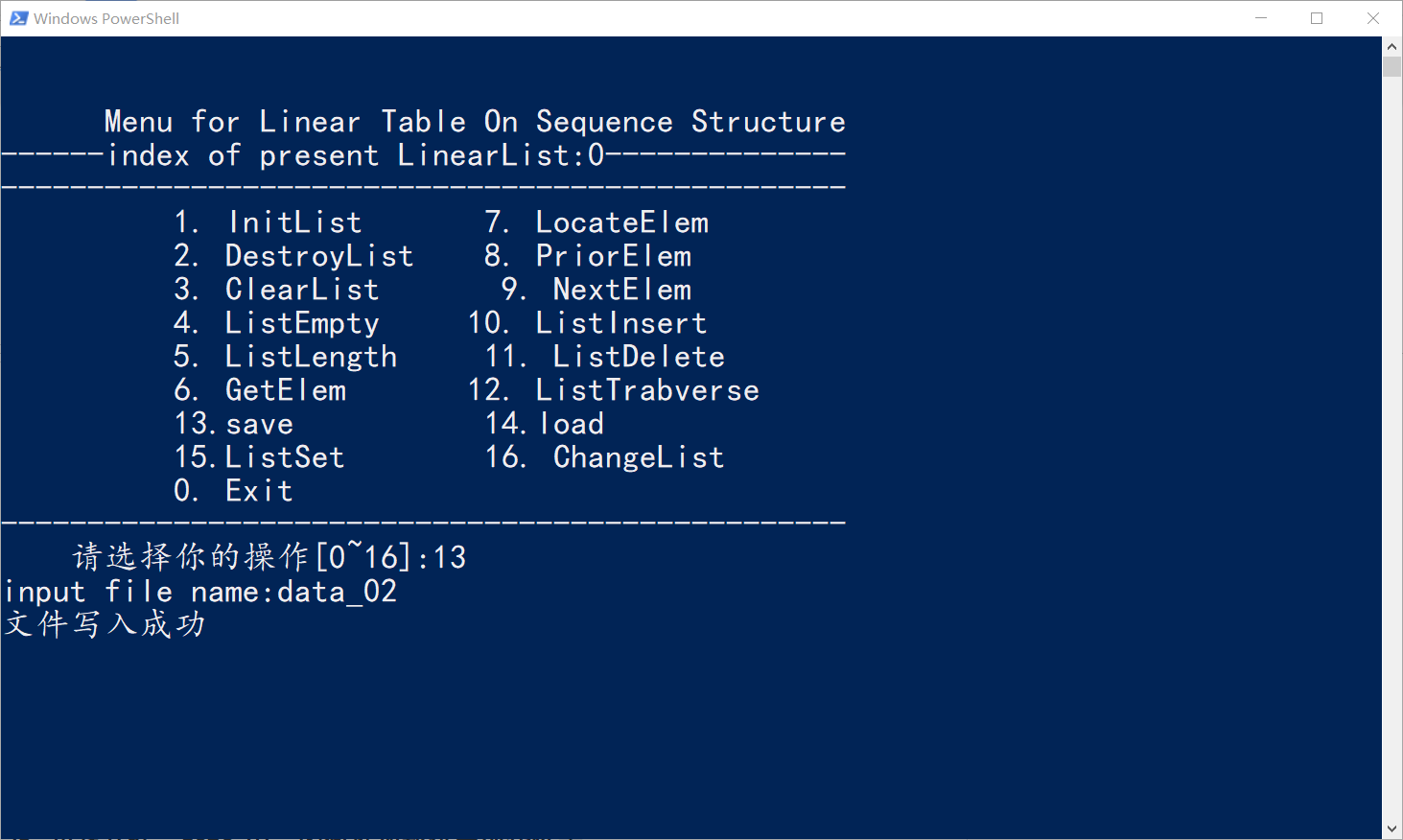
1. **在表的第3个位置插入值5**

****

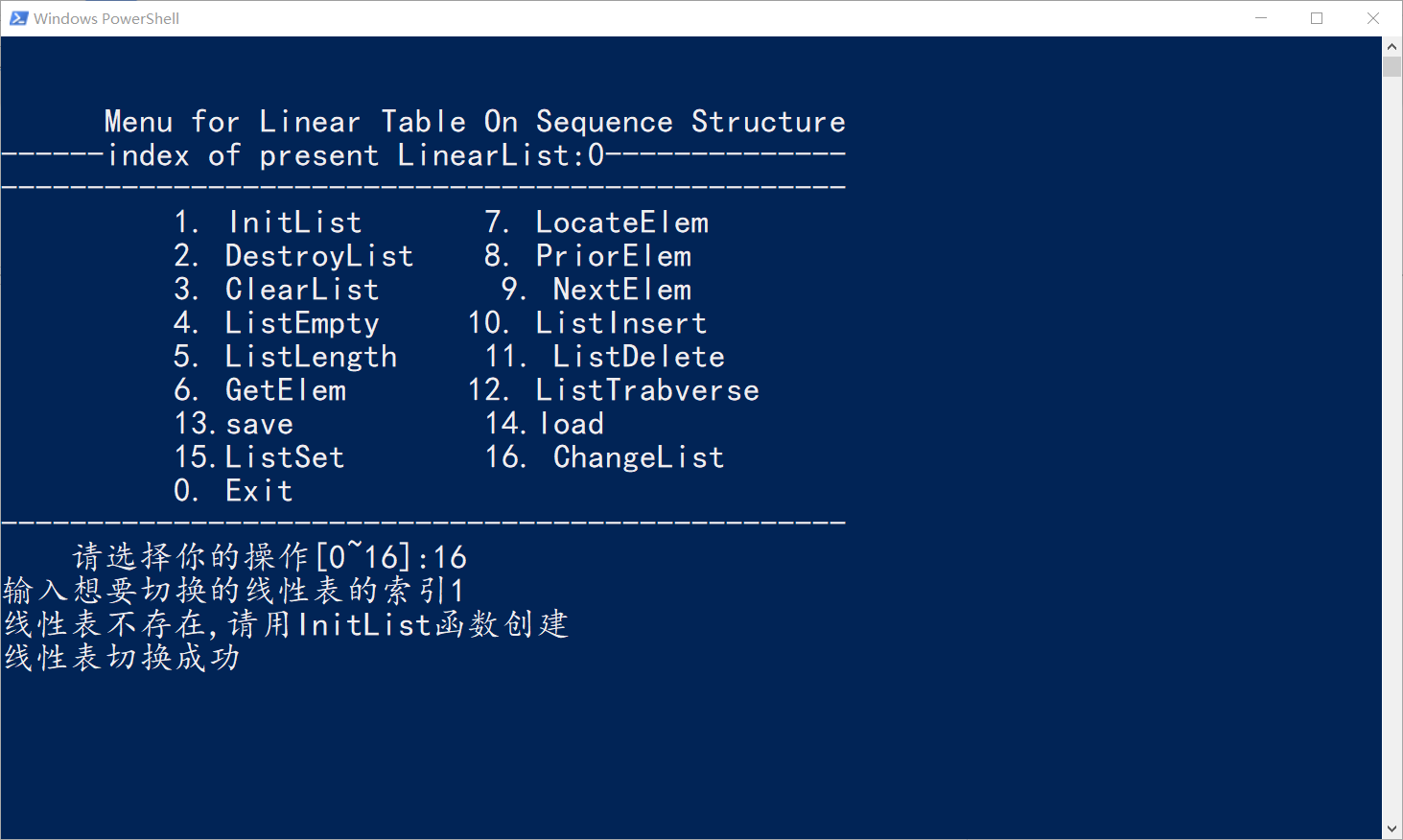
1. **打印这个表，发现插入数据成功**

****

1. **再将当前表的数据写入文件“data\_02”**

****

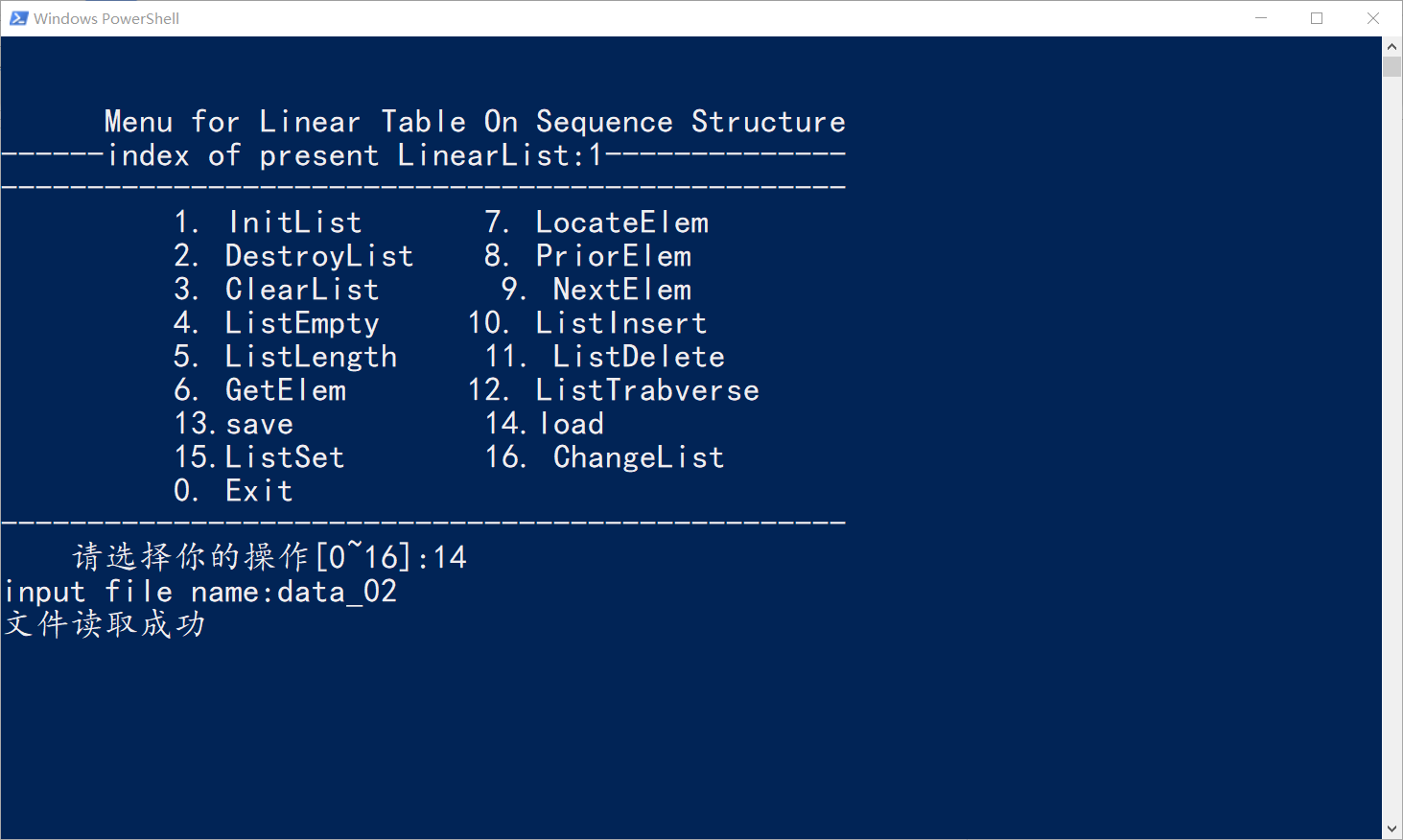
1. **切换线性表，切换索引为1**

****

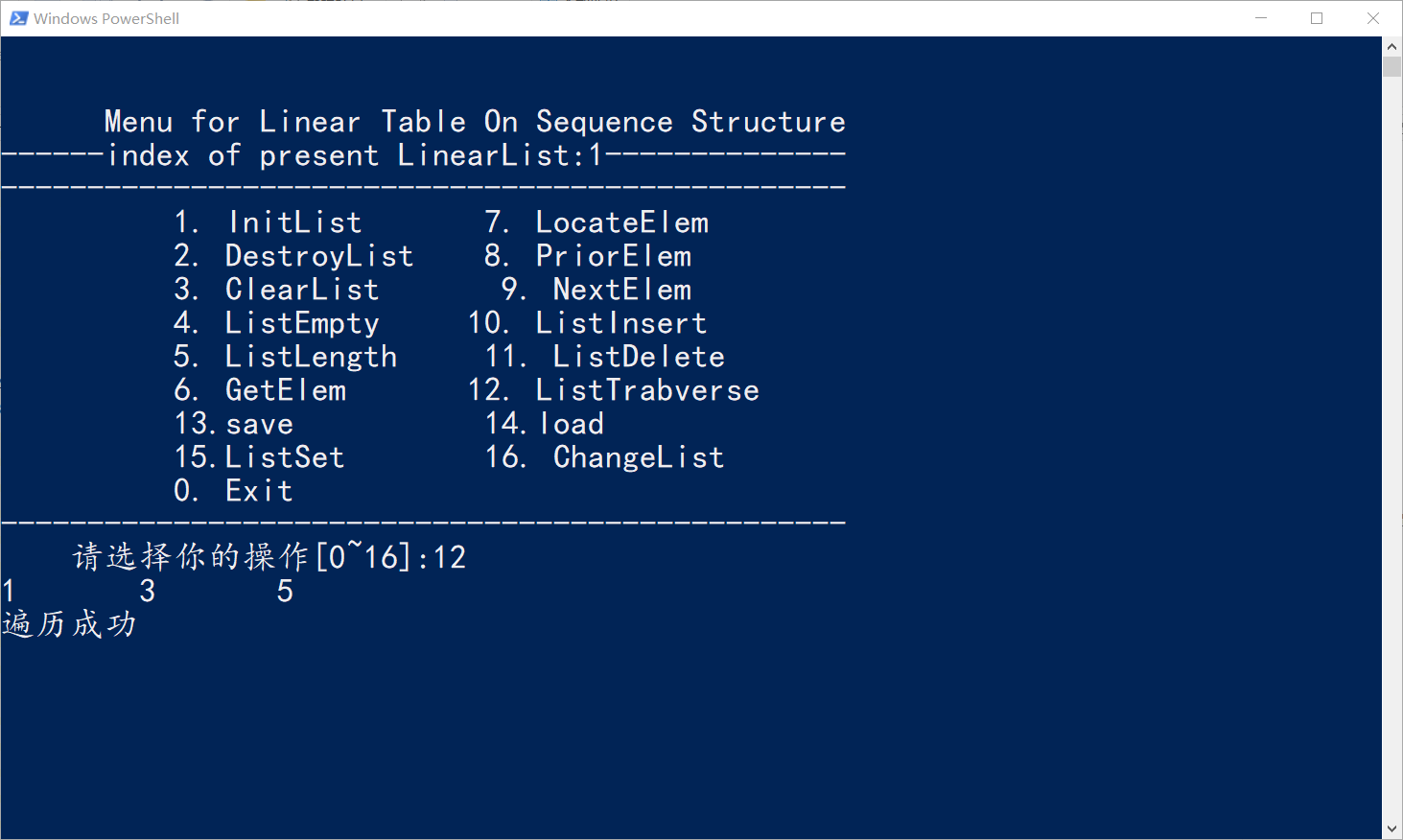
1. **创建一个新表**

****

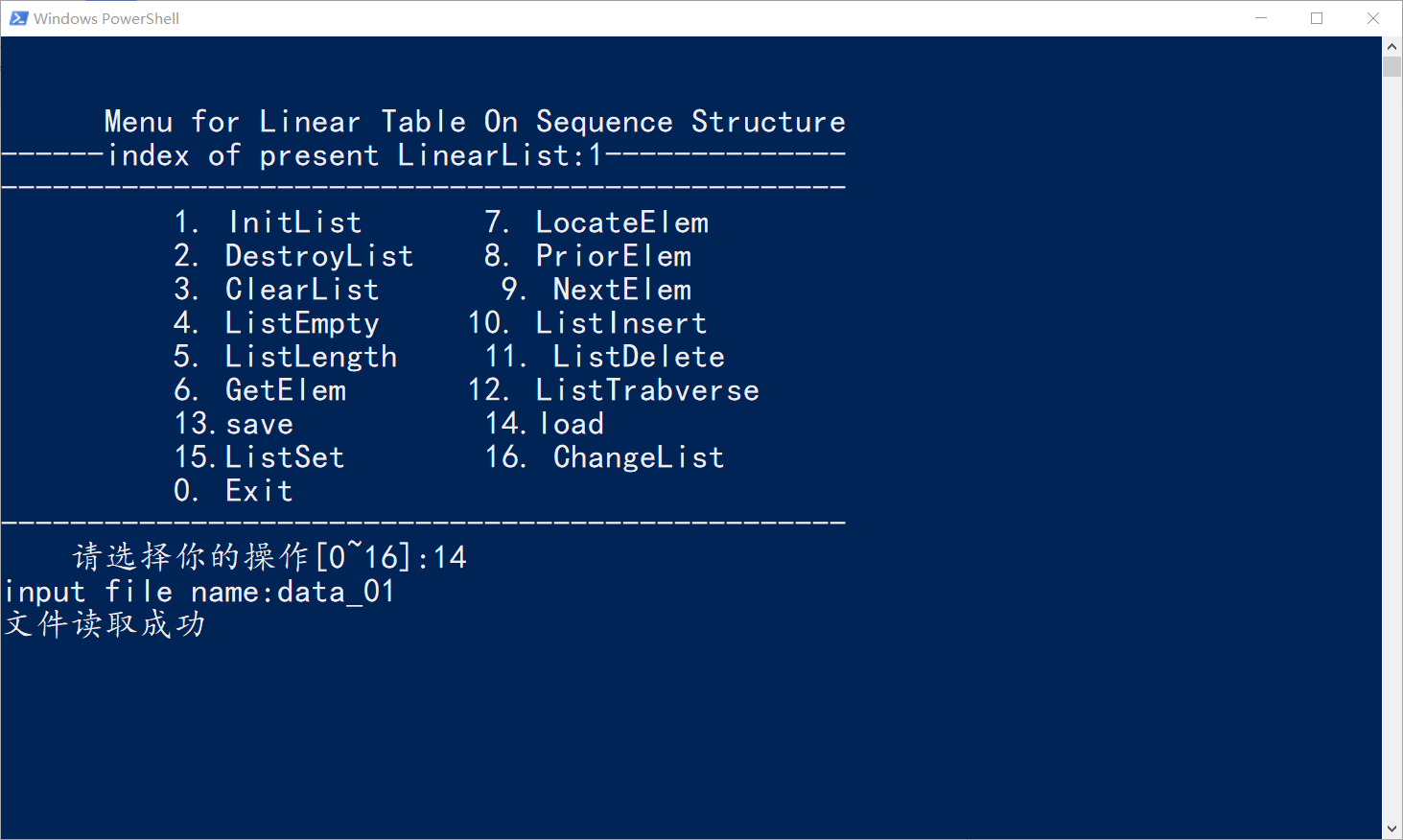
1. **将文件“data\_02”的数据加载进当前线性表**

****

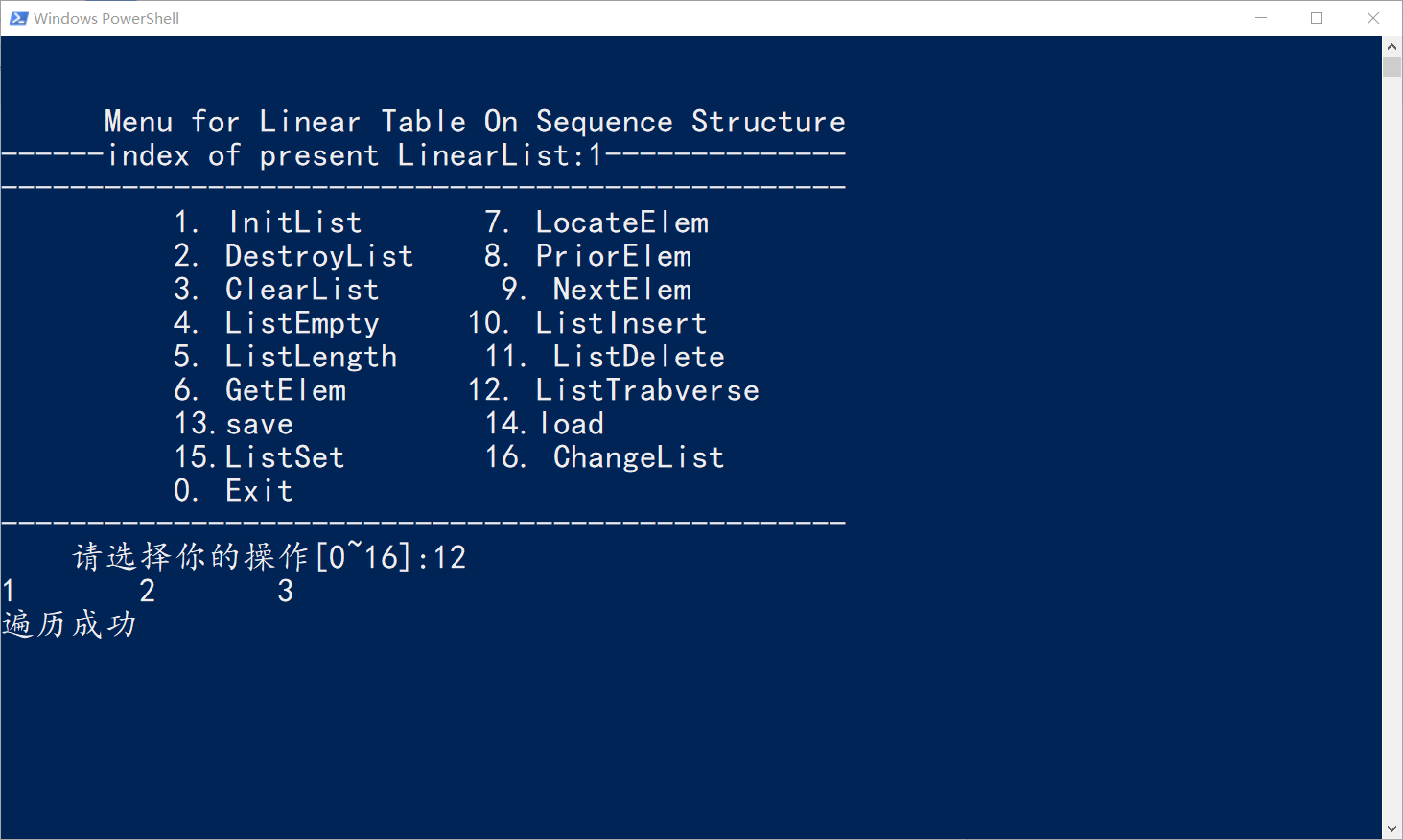
1. **打印这个表，发现数据被成功加载**

****

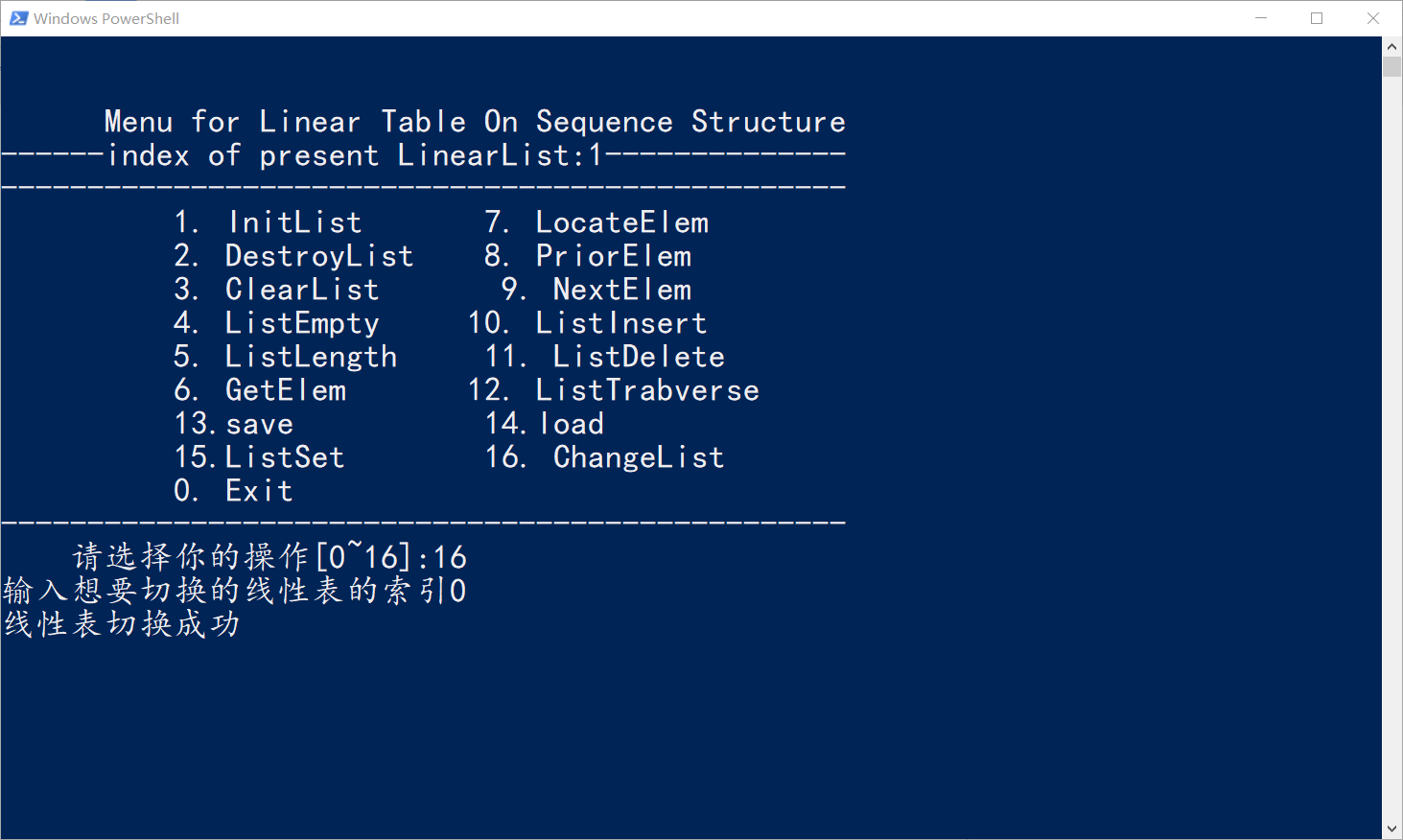
1. **再将文件“data\_01”的数据加载进当前线性表**

****

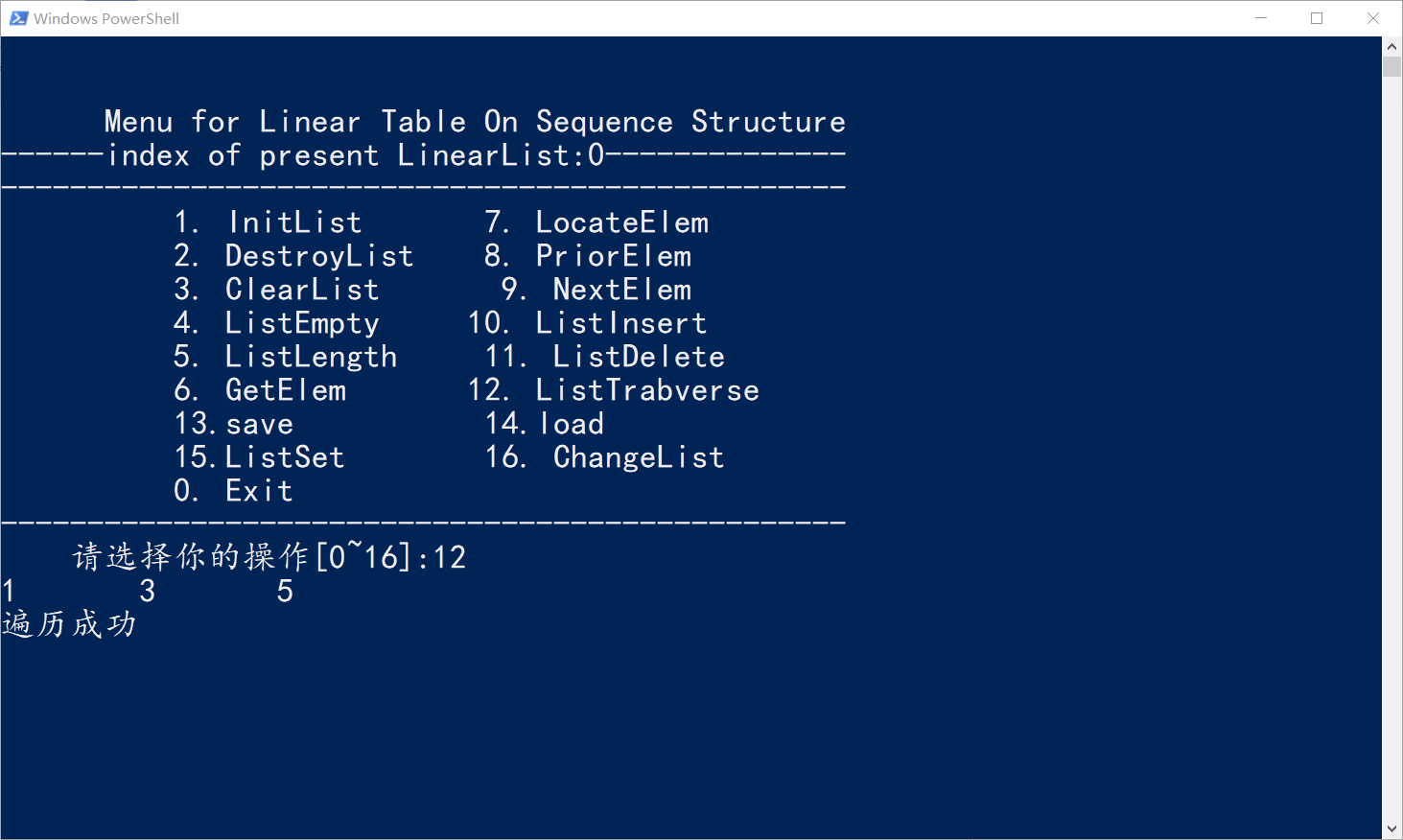
1. **打印这个表，发现数据也被成功加载**

****

1. **切换回索引为0的线性表**

****

1. **打印当前表，显示的数据与data\_02中的数据相同，说明各线性表之间的操作相互独立**

****

1. **正常退出系统**

****

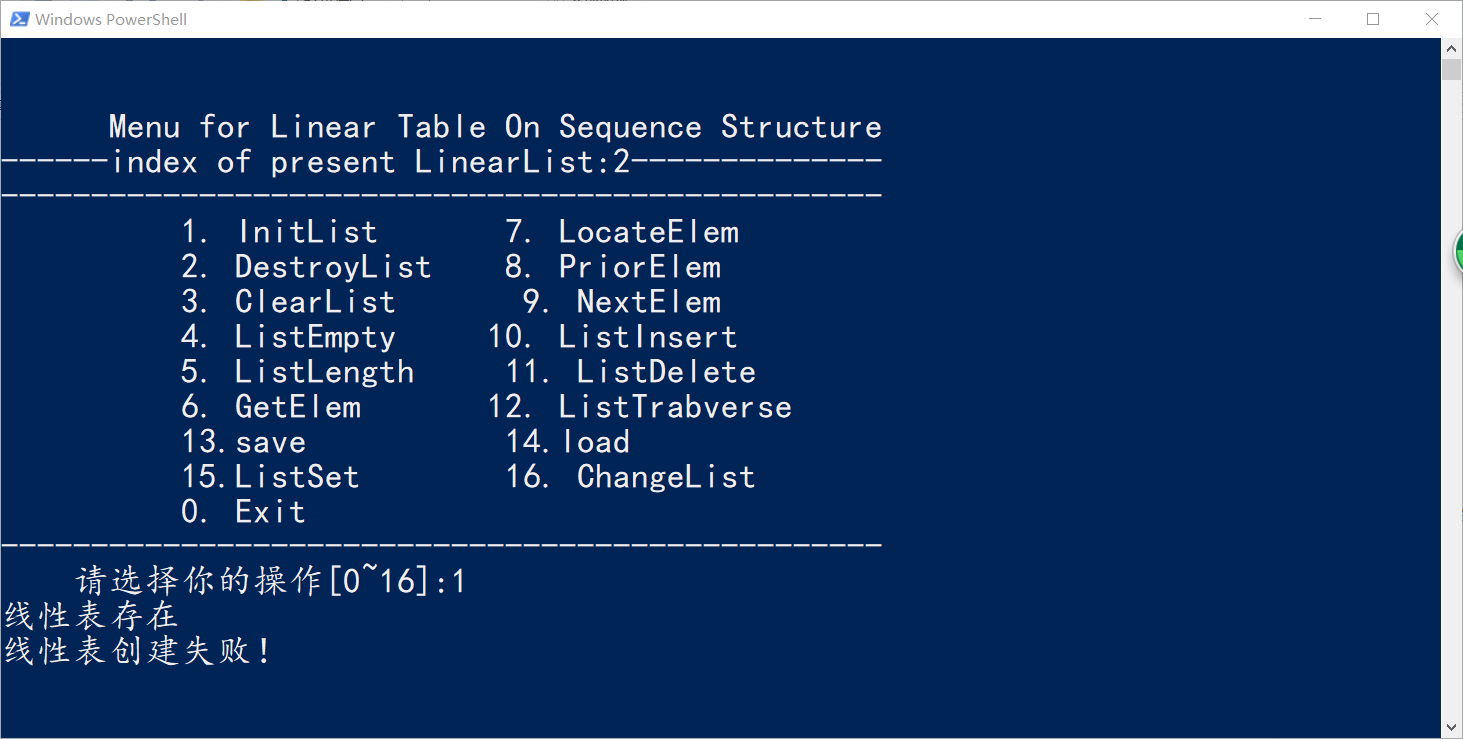
通过以上的演示证明，我们的线性表的功能是完整实现了的，可以演示要求的十二种线性表上的运算，并且做到了多线性表管理，实现了文件存储功能，不存在内存泄漏，完整的达到了实验要求。

**1.3.3 系统测试**

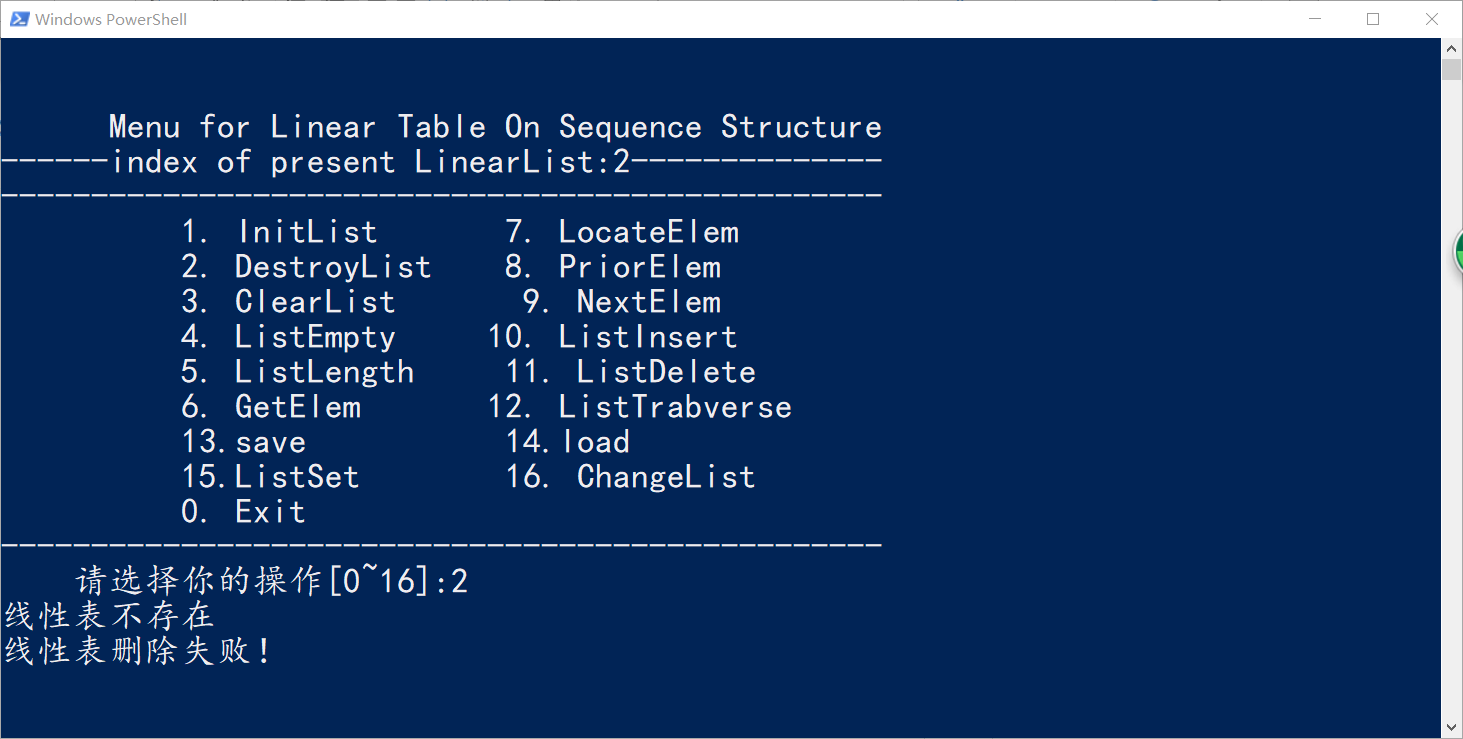
在上面操作的基础上，我们再创建两个线性表，索引分别为2和3，索引为2的线性表中的数据元素是1,4,2,7,5；索引为3的线性表中的数据元素是12,4,10,3,6；分别写入文件“data\_03”和“data\_04”

下面以这两个线性表基础，进行系统测试。

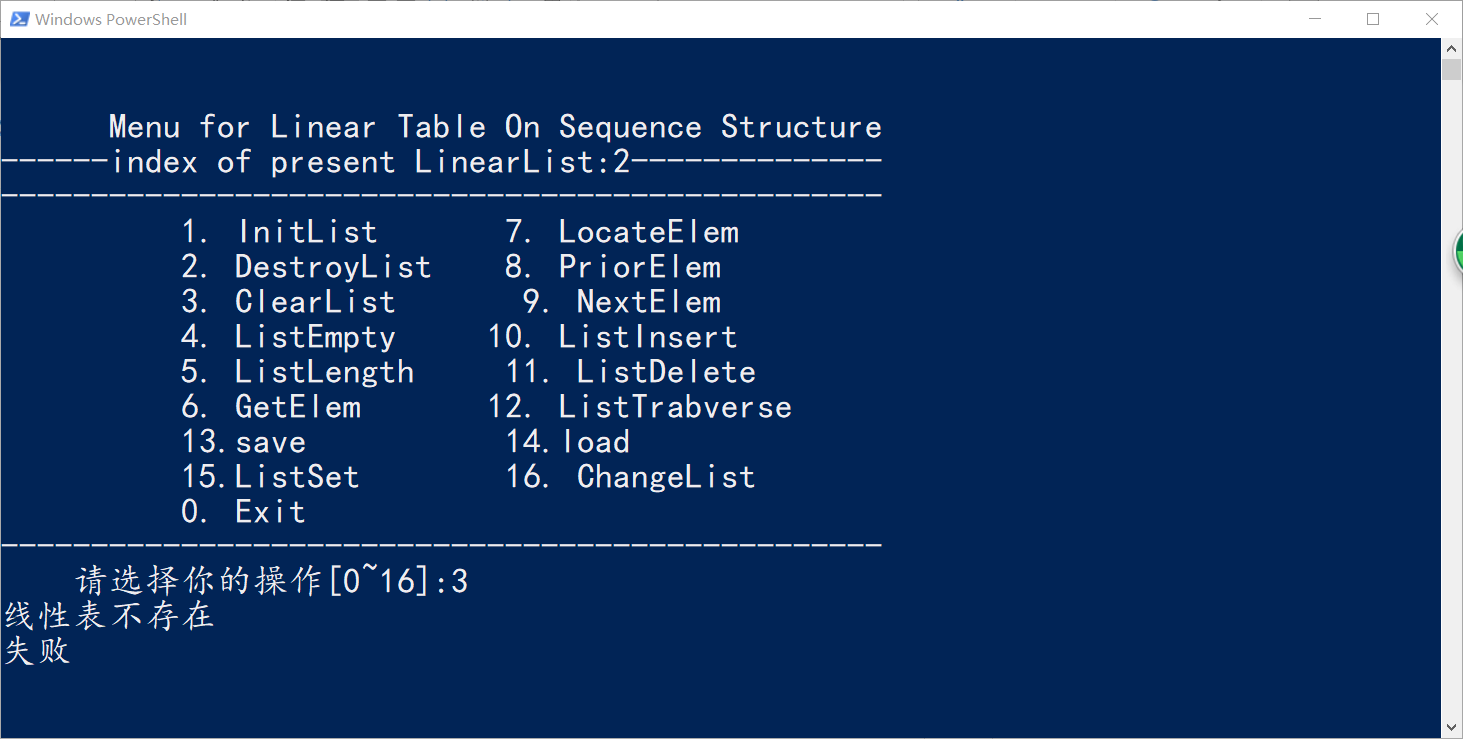
1. **初始化已存在的线性表**

****

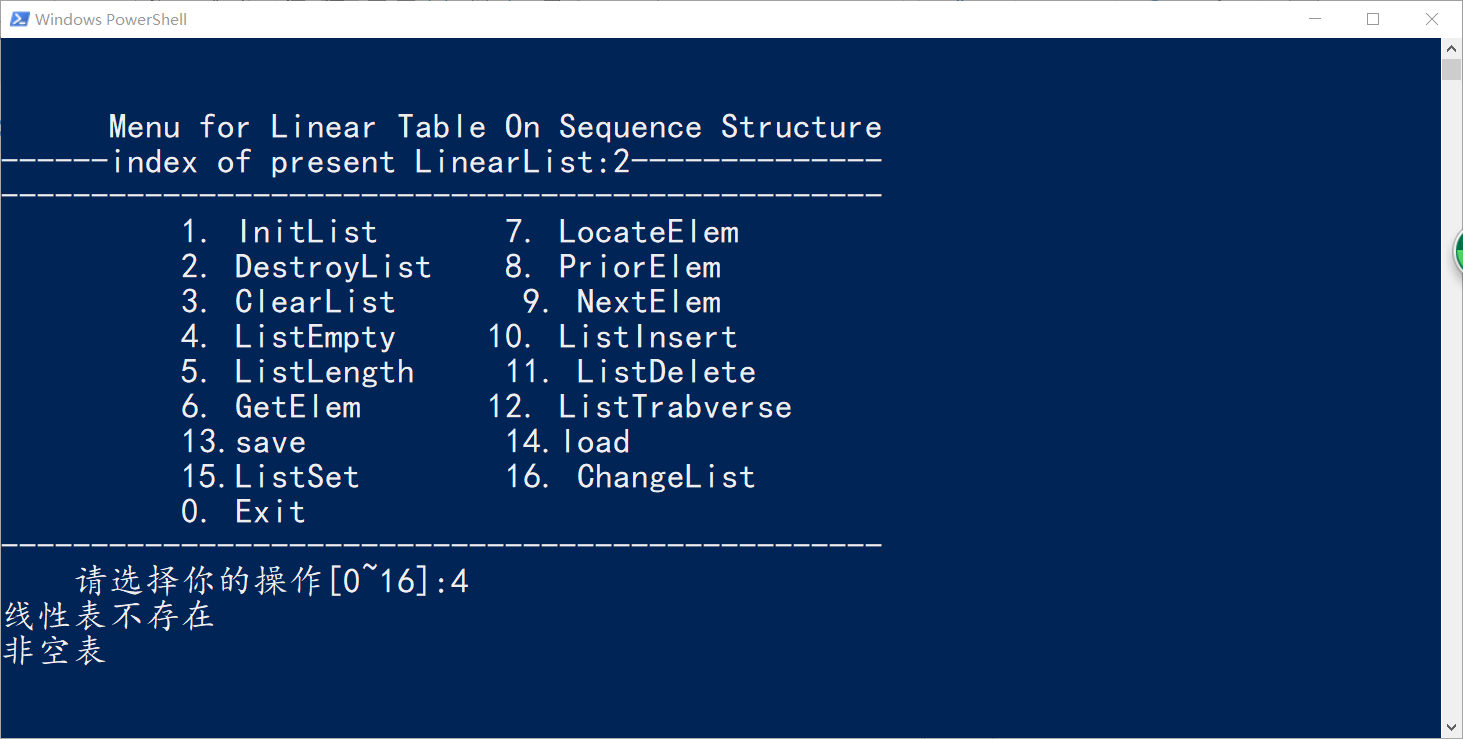
1. **删除不存在的线性表**

****

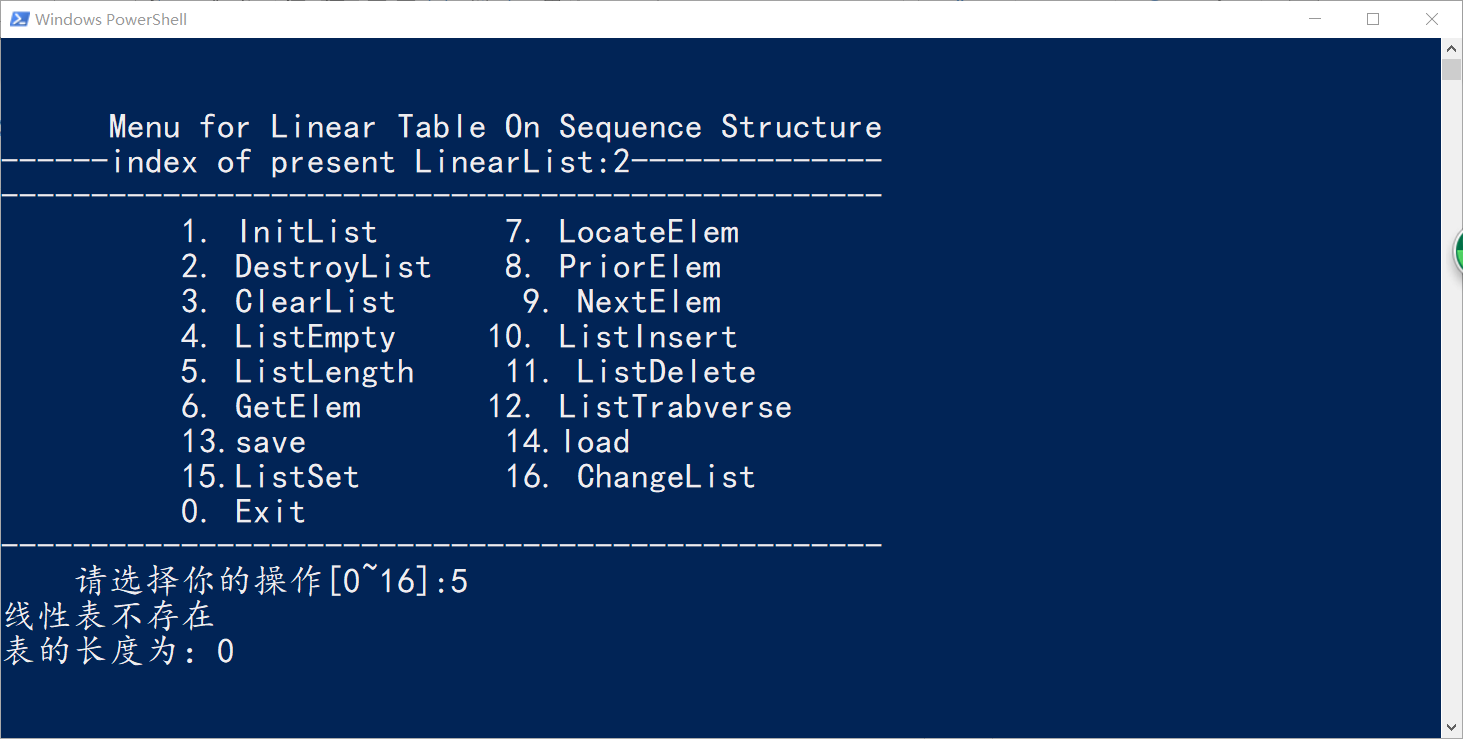
1. **清空不存在的线性表**

****

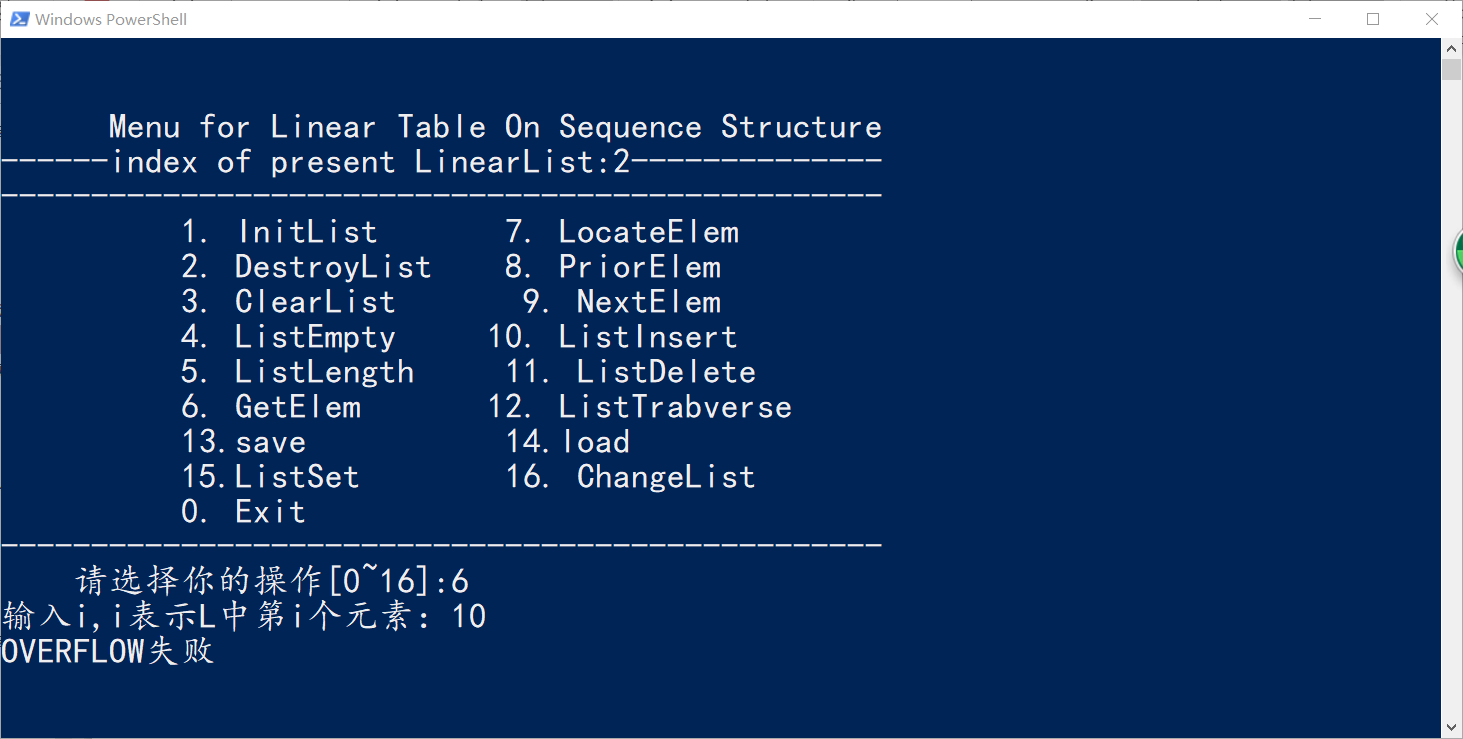
1. **判断不存在的线性表是否为空**

****

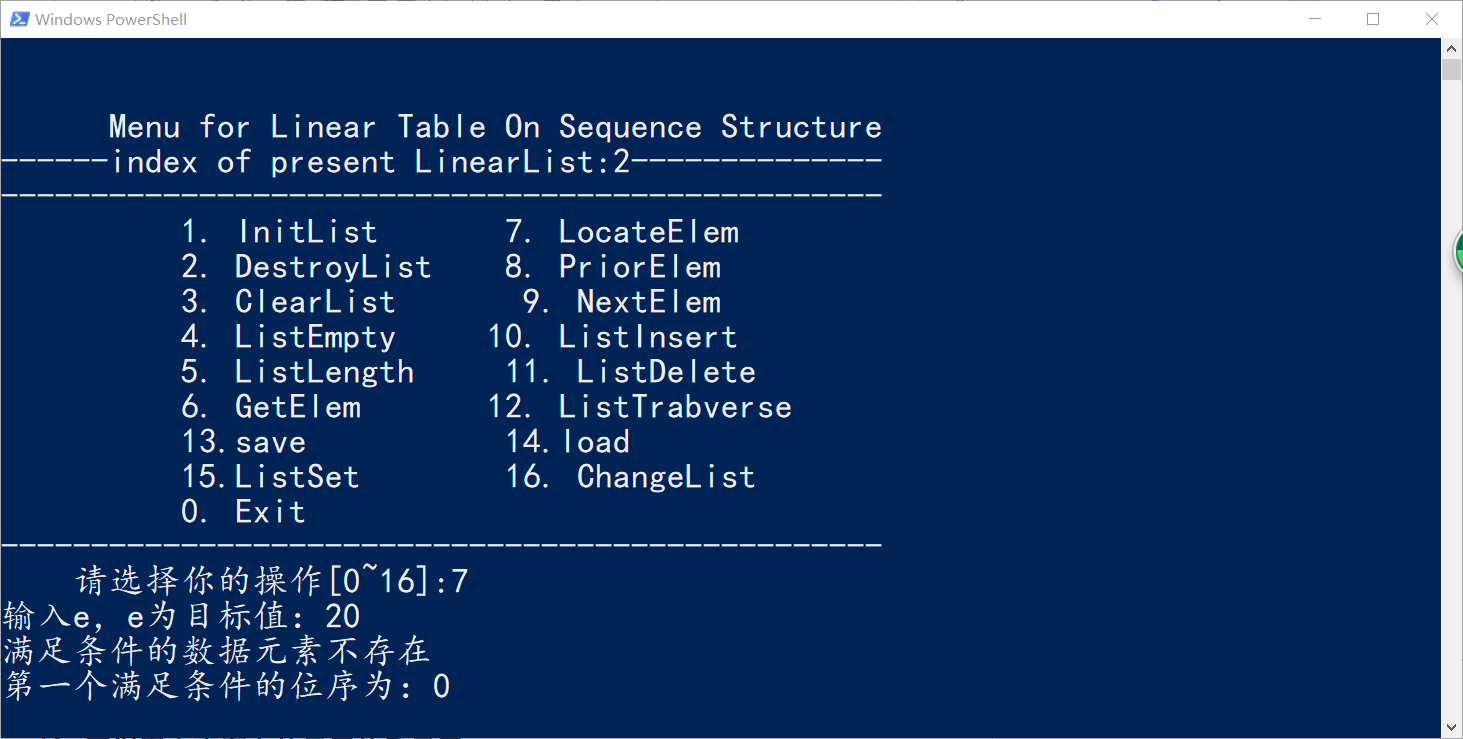
1. **获得不存在的线性表的表长**

****

1. **获得表中不存在的元素**

****

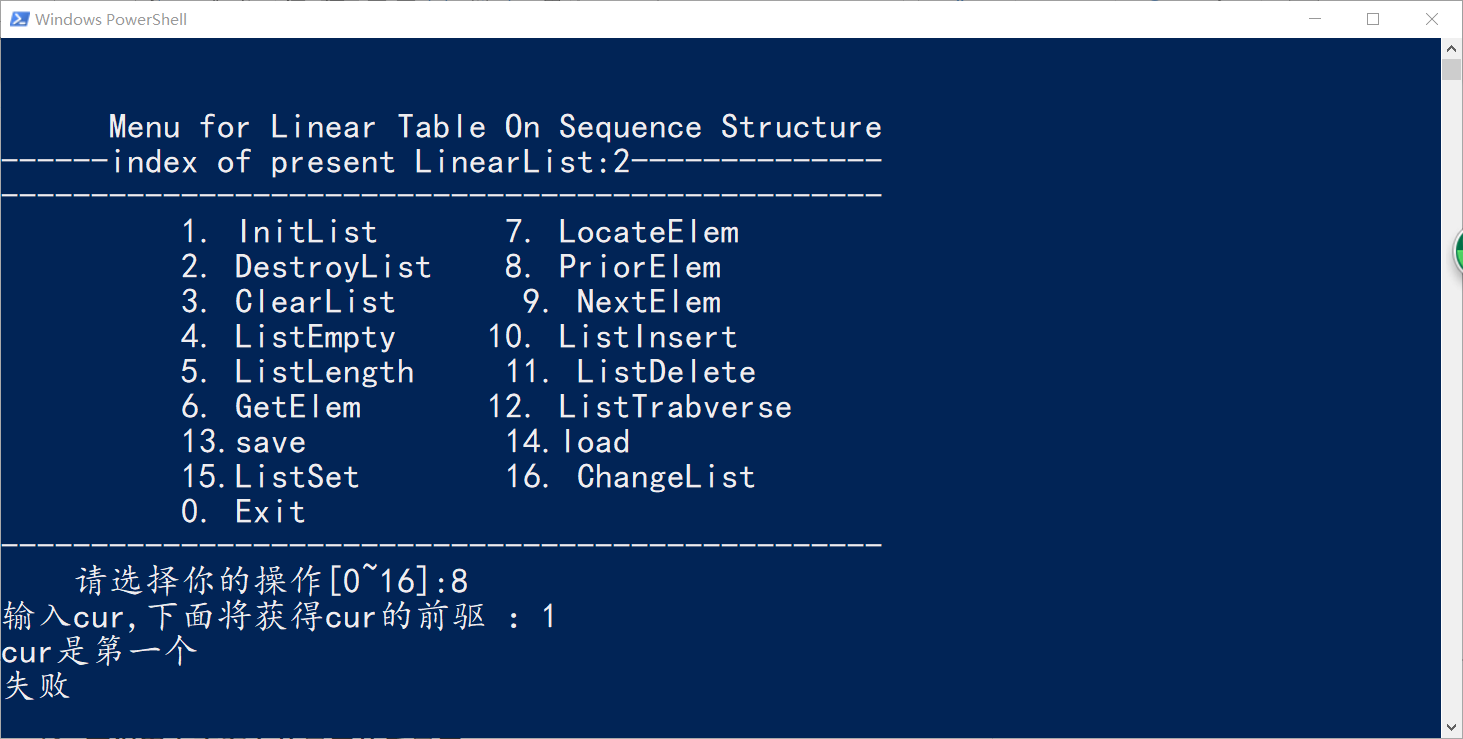
1. **确定表中不存在的元素的位置**

****

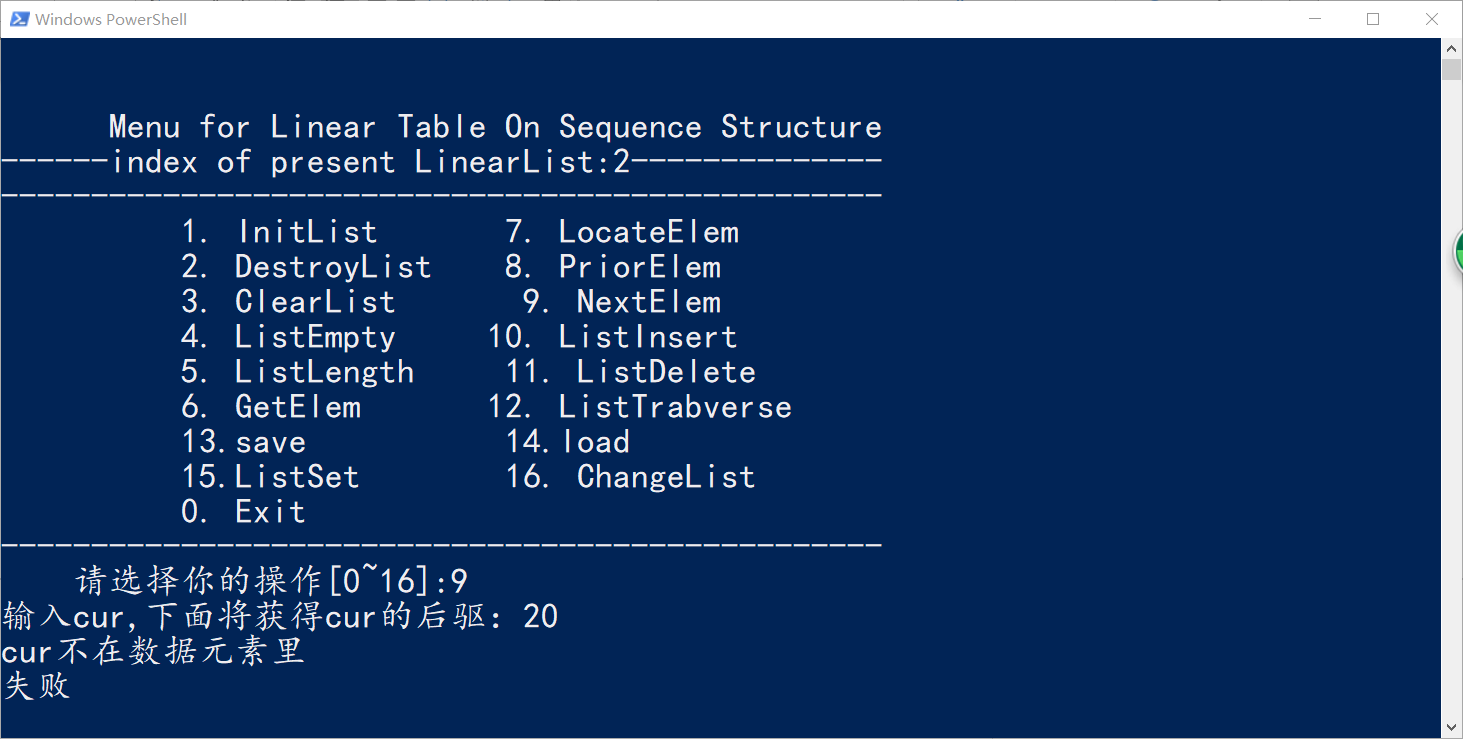
1. **获得表中不存在的元素的前元素**

****

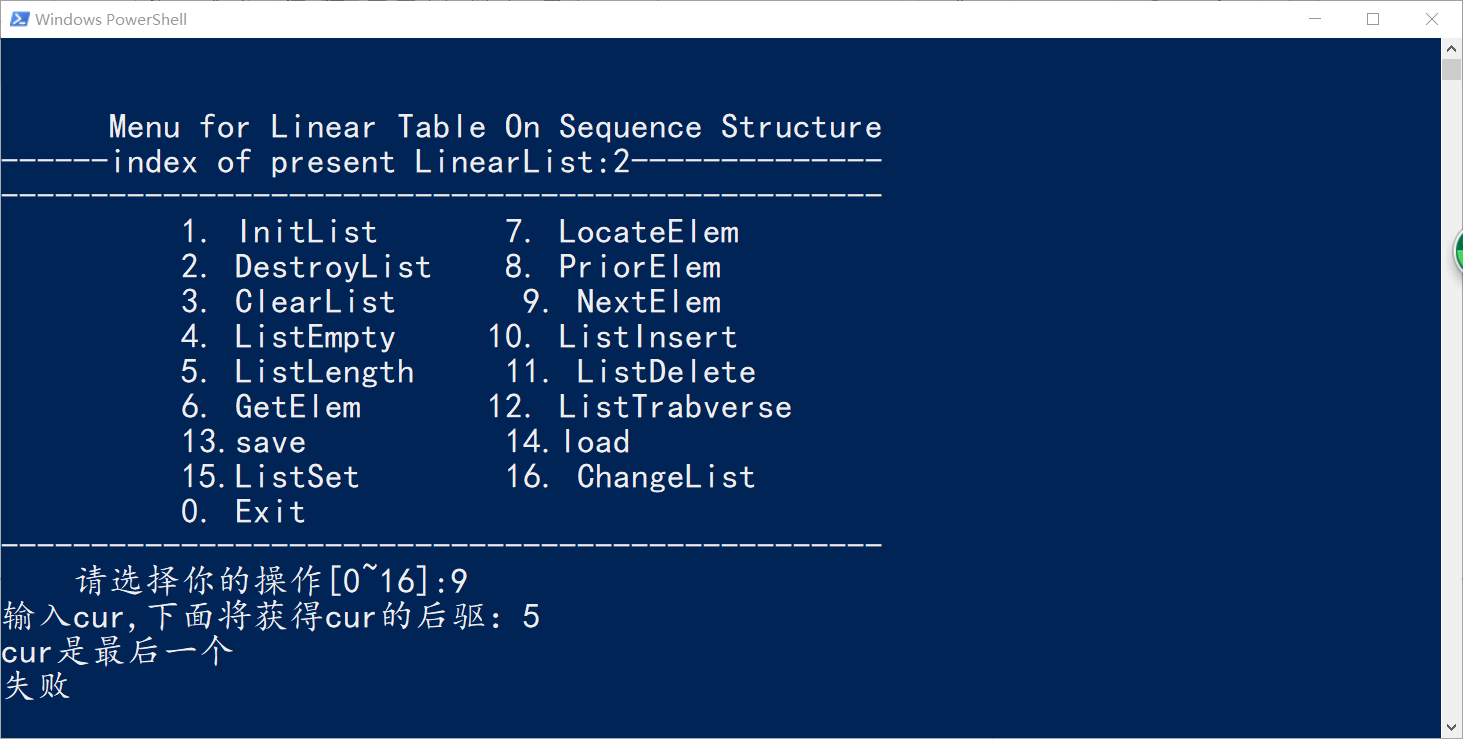
1. **获得表中第一个元素的前元素**

****

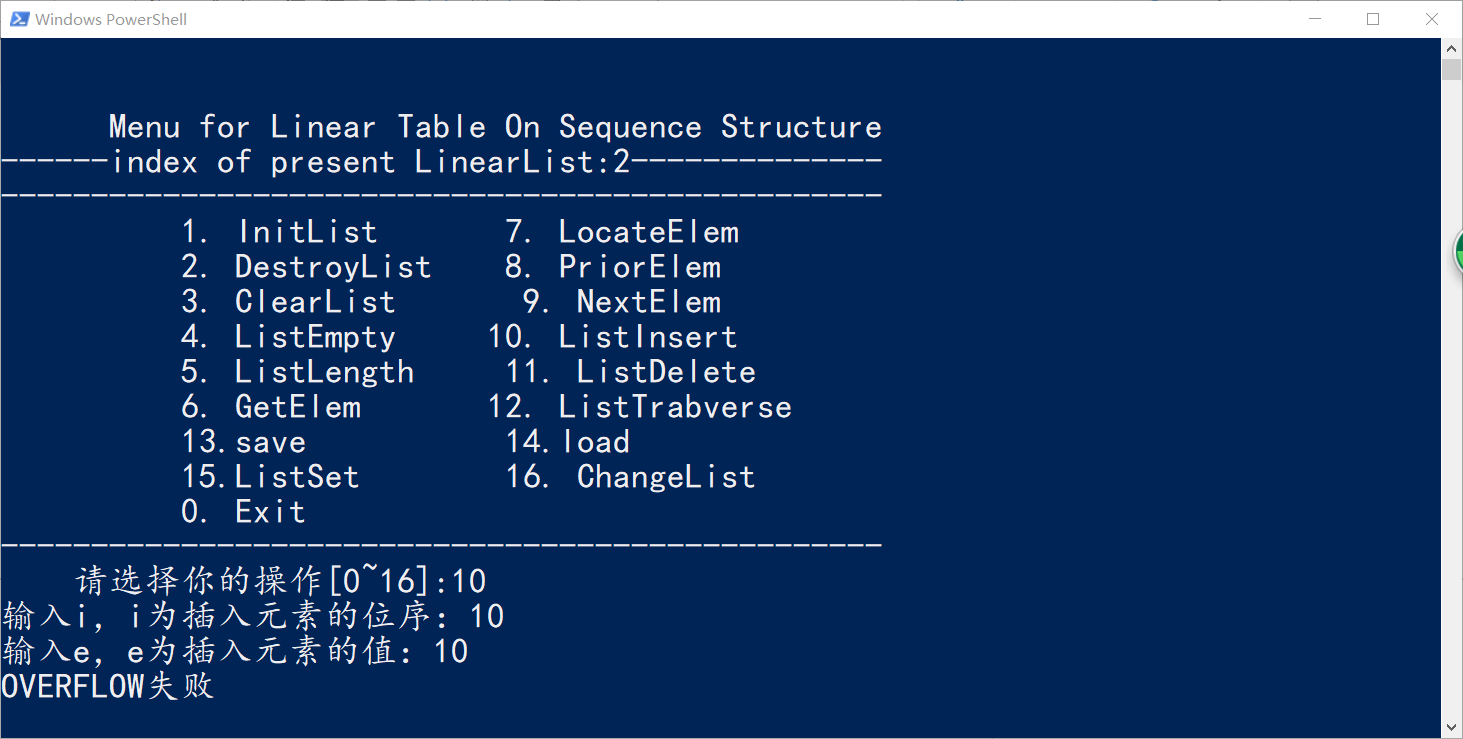
1. **获得表中不存在的元素的后元素**

****

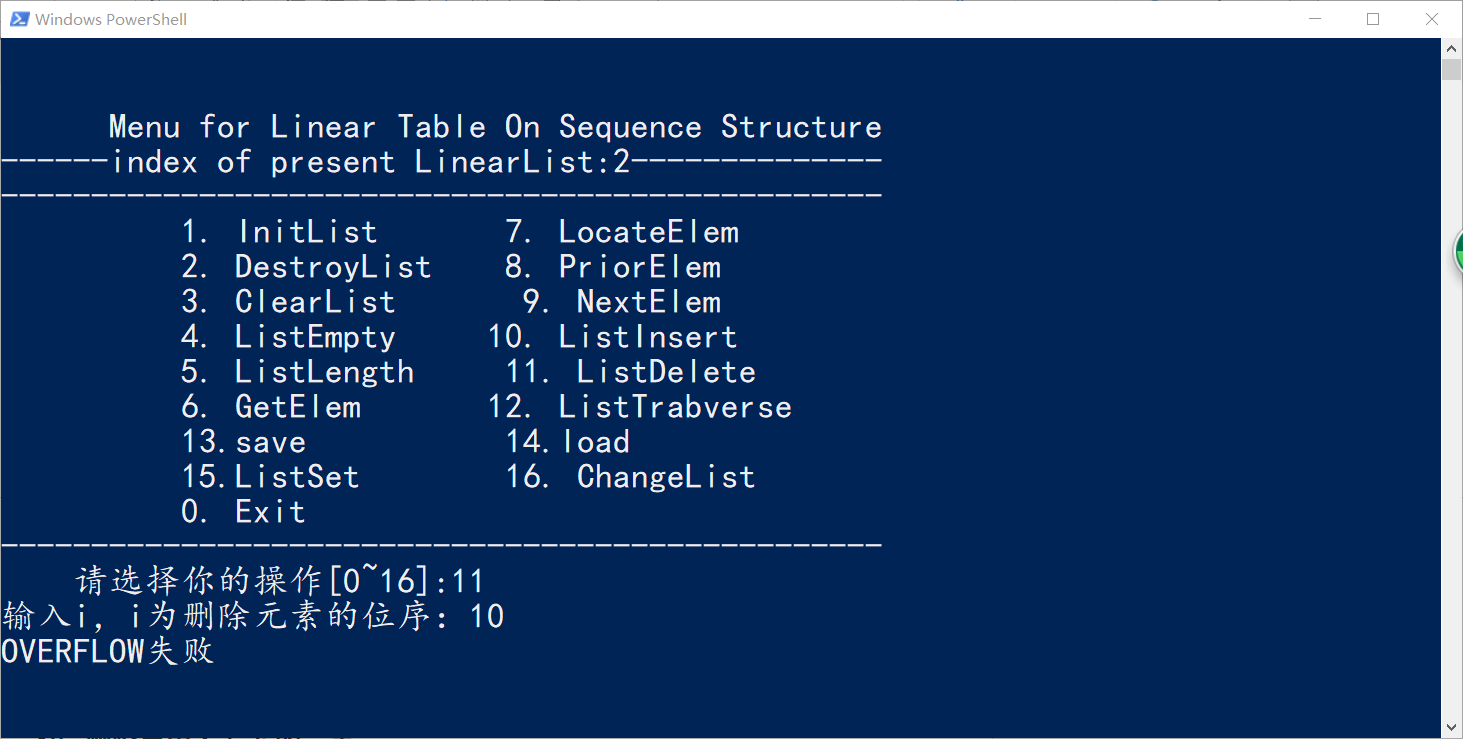
1. **获得表中最后一个元素的后元素**

****

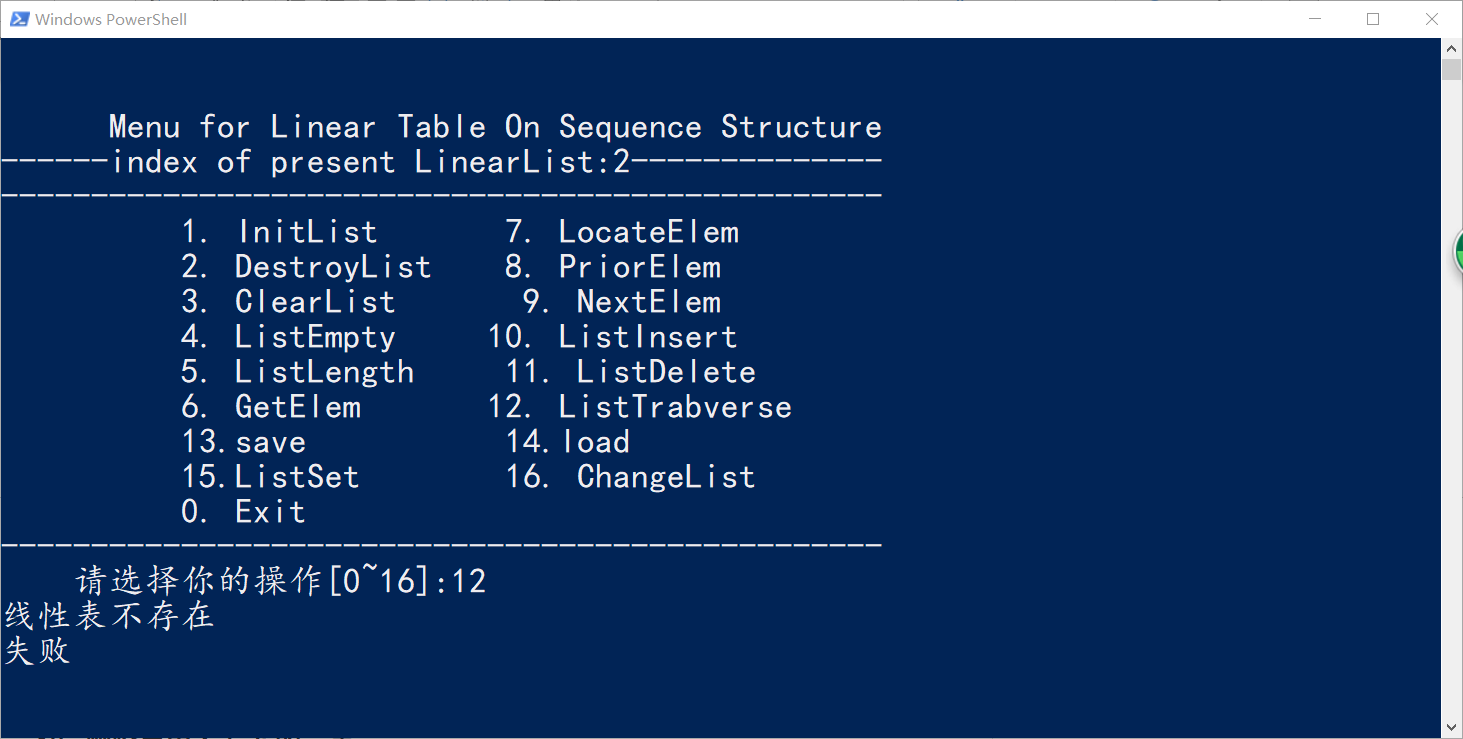
1. **插入元素超过线性表的最大大小**

****

1. **删除表中不存在的元素**

****

1. **打印不存在的线性表**

****

## 1.4 实验小结

这次实验是数据结构实验课程的第一次实验，难度不算特别大，但还是要注意一些细节，比如必要的注释，错误输入提醒，系统健壮性的实现等等，个人感觉这次实验还有许多做得不足之处，希望后面几次实验能将这些不足改进，达到提高我的编程能力的目的。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

实现基于链式存储结构的线性表，并实现线性表的基本运算。

构造一个具有菜单的功能演示系统。

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。

演示系统实现了线性表的文件形式存储。

演示系统可以实现多线性表管理。

整个系统的设计模式如下：

* + 1. **线性表抽象数据类型**

依据最小完备性和常用性相结合的原则，设计了线性表的数据对象和数据关系，并定义了线性表的初始化表，销毁表，清空表，判定空表，求表长和和获得元素等12种基本运算，具体数据和运算功能定义如下。

ADT Sqlist{

数据对象：D={ai|ai属于ElemSet,i=1,2,...n,n>=0}

数据关系：Rl={<ai-1,ai>|ai-1,ai属于D,i=1,2,...n}

基本操作：

InitList(&L)

初始条件：线性表L不存在

操作结果：构造一个空线性表

DestroyList（&L）

初始条件：线性表L存在

操作结果：摧毁线性表L

ClearList（&L）

初始条件：线性表存在

操作结果：将L重置为空表

ListEmpty（&L）

初始条件：线性表L存在

操作结果：L为空表返回TRUE，否则返回FALSE

ListLength（&L）

初始条件：线性表L存在

操作结果：返回L中数据元素的个数

GetElem（L,i,e）

初始条件：线性表L存在,1≤i≤ListLength(L)

操作结果：用e返回L中第i个数据元素的值

LocateElem（L,e,compare()）

初始条件：线性表存在

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。

PriorElem(L,cur\_e,pre\_e)

初始条件：线性表存在

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。

ListInsert(L,i,e)

初始条件：线性表存在

操作结果：在L的第i个位置之前插入新的数据元素e

ListDelete(L,i,e)

初始条件：线性表存在

操作结果：删除L的第i个数据元素，用e返回其值

ListTraverse(L,visit())

初始条件：线性表存在

操作结果：依次对L的每个数据元素调用函数visit()

ListSet(L)

初始条件：线性表存在

操作结果：对线性表进行排序

}ADT SqList

* + 1. **多线性表的管理**

同实验一一样我定义了一个结构体指针数组，用来存储指向各个线性表的指针值，指针在数组中对应的下标就是相应线性表的标识（从0开始），在演示菜单中可以实现各个线性表的切换。

* + 1. **演示系统和文件存储的设计**

演示系统借鉴了附录A中给出的框架，该框架将完成函数调用所需实参值的准备和函数执行结果的实现，并给出适当的操作提示显示，整体以命令行呈现。

此外设计了数据文件实时存储，文件存储的操作包括在演示系统上，用户可以自行选择是否将当前线性表存储，存储方式为将当前的线性表结构体和数据区域直接将内存区块写入文件中。文件读取操作也包括在演示系统里面，读取时用户可以选择当前目录中已经保存的文件加载，来还原该文件所存储的一个线性表的数据。

## 系统设计

* + 1. **数据物理结构**

线性表的存储数据结构

线性表结构体定义如下：

**typedef struct LNode//define of ADT**

**{**

**ElemType data;//data area**

**struct LNode \*next;//pointer area**

**}LNode,\*LinkList;**

宏定义：

**#define TRUE 1**

**#define FALSE 0**

**#define OK 1**

**#define ERROR 0**

**#define INFEASTABLE -1**

**#define OVERFLOW -2**

**typedef int status;**

**typedef int ElemType;**

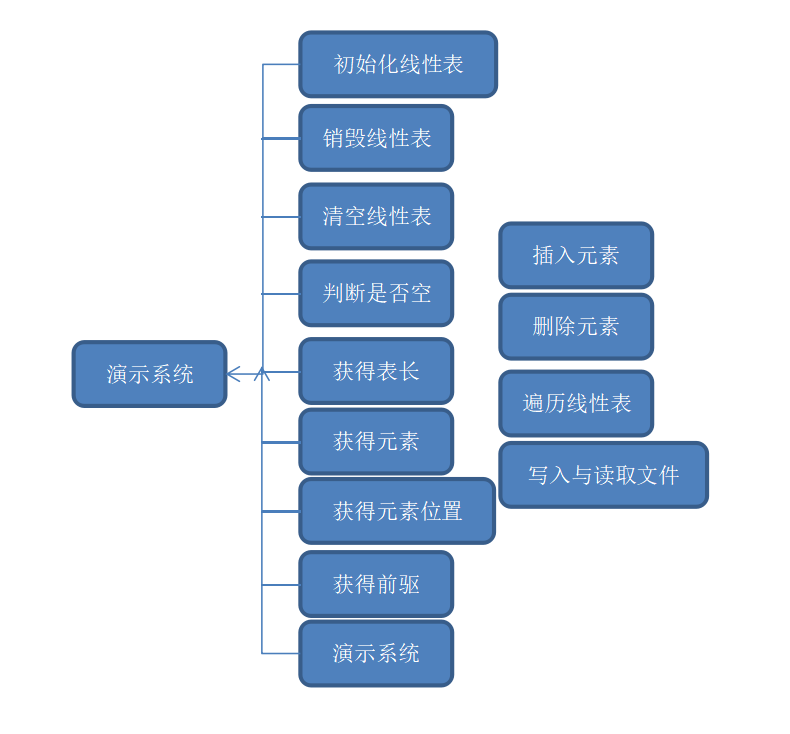
* + 1. **演示系统**

演示系统包括用户操作界面和功能调用部分。

演示系统界面语言为英文，所有操作和提示语言均为英文。

用户操作界面输出可选的线性表操作，用户输入数字选择要进行的操作。在用户选择操作后，功能调用部分会显示函数的名称，参数，返回值和作用，系统提示用户输入参数。

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。



* + 1. **线性表运算实现算法**

1. status InitList(LinkList&L);

功能：初始化线性表

算法实现：如果L不为空，则打印L已经存在，返回ERROR；否则为L申请内存空间，并将节点数据初始化，返回OK

时空效率分析：算法的时间复杂度为O（1），空间上为L指向的节点申请空间，所以空间复杂度为O（1）。

1. status DestroyList(LinkList&L);

功能：摧毁线性表

算法实现：如果L为空，则打印错误信息，返回ERROR；如果L不为空，则free整个链表的每个节点的存储空间，然后让L指向NULL，防止L称为野指针，返回OK。

时空效率分析：算法的时间复杂度为O（1），空间复杂度为O（1）

1. Status ClearList(LinkList&L);

功能：清空线性表

算法实现：如果L为空，则打印错误信息，返回ERROR；如果L不为空，则将L头结点后面的节点free掉，然后将表长置为零

时空效率分析：算法的时间复杂度为O（1），空间复杂度为O（0）

1. Status ListEmpty(LinkList&L);

功能：判断线性表是否为空

算法实现：如果L为空，则打印错误信息，返回ERROR；如果L不存在，打印错误信息，返回ERROR，否则判断表长是否为0，为0返回TRUE，部位0返回FALSE。

时空效率分析：时间复杂度为O（1），空间复杂度为O（1）

1. Int ListLength(LinkList&L);

功能：求线性表的长度

算法实现：如果L为空，则打印错误信息，返回ERROR；直接返回线性表头结点的L->data（表示表长）即可

时空效率分析：时间复杂度为O（1），空间复杂度为O（1）

1. Status GetElem(LinkList&L,int i,ElemType&e);

功能：获得线性表中指定位置的数据

算法实现：如果L为空，则打印错误信息，返回ERROR；首先判断指定位置是否合法，若不是则返回ERROR，否则遍历链表，把链表中第i个节点的data值赋给e，返回OK

时空效率分析：时间复杂度为O（1），空间复杂度为O（1）

1. Int LocateElem(LinkList&L,ElemType e,int(\*compare)(ElemType x,ElemType e));

功能：寻找指定元素在线性表中的位置

算法实现：如果L为空，则打印错误信息，返回ERROR；遍历链表，直到找到与e符合关系compare的值，返回给节点的位序，返回OK，如果没找到，则打印错误信息，返回ERROR

时空效率分析：时间复杂度为O（n），空间复杂度为O（1）

1. Status PriorElem(LinkList&L,ElemType cur,ElemType&pre\_e);

功能：获得指定元素之前的一个一个元素

算法实现：如果L为空，则打印错误信息，返回ERROR；直接遍历寻找，然后把前一位位置的节点中的data成员的地址赋值给pre\_e。如果找到就返回OK，否则返回ERROR。

时空效率分析：时间复杂度为O（n），空间复杂度为O（1）

1. Status NextrElem(LinkList&L,ElemType cur,ElemType&next\_e);

功能：获得指定元素之后的一个元素

算法实现：如果L为空，则打印错误信息，返回ERROR；直接遍历寻找，然后把后一位位置的节点中的data成员的地址赋值给next\_e，如果找到就返回OK，否则返回ERROR

时空效率分析：时间复杂度为O（n），空间复杂度为O（1）

1. Status ListInsert(LinkList&L,int i,ElemType e);

功能：插入元素

算法实现：如果L为空，则打印错误信息，返回ERROR；首先要判断插入的位置是否合法，如果不合法则返回ERROR，然后找到要插入位置的前一个节点，在插入位置新建一个节点，将该节点data成员的值赋值为e，返回OK

时空效率分析：时间复杂度为O（n），空间复杂度为O（1）

1. Status ListDelete(LinkList&L,int i,EemType&e);

功能：删除元素

算法实现：如果L为空，则打印错误信息，返回ERROR；首先要判断删除的位置是否合法，若不是则返回ERROR，然后将指定位置的节点free掉，free掉之后恢复链表的链接，返回OK

时空效率分析：时间复杂度为O（n），空间复杂度为O（1）

1. Status ListTraverse(LinkList&L,void(\*visit)(ElemType e));

功能：遍历线性表并将每个数据元素调用visit函数

算法实现：如果L为空，则打印错误信息，返回ERROR；直接遍历链表对每个节点的data成员调用visit函数即可

时空效率分析：时间复杂度O（n），空间复杂度O（1）

1. Status ListSort(LinkList&L);

功能：对线性表中的元素进行排序

算法实现：如果L为空，则打印错误信息，返回ERROR；运用冒泡排序法对链表中的节点进行排序

时空效率分析：时间复杂度O（n^2），空间复杂度O（1）

1. ListMerger(LinkList&La,LinkList&Lb,LinkList&Lc);

功能：合并两个线性表

算法实现：如果La或Lb或Lc为空，则打印错误信息，返回ERROR；先为Lc头节点创建长度为len(La)+len(Lb)的链表，并将La和Lb中每个节点的值复制到Lc中，然后对Lc进行排序，排序成功，返回OK，否则返回ERROR

时空效率分析：时间复杂度O（n^2），空间复杂度：O（n）

* + 1. **多线性表管理实现算法**

由于多线性表管理采用结构体指针数组管理，只涉及到查找操作，而且查找操作直接利用数组下标，所以时间复杂度为O（1），空间复杂度为O（1）

* + 1. **文件存储实现算法**

1. 写文件

算法实现：用户输入要保存的文件名，打开文件，根据线性表的L->size存入数据空间，将当前的线性表作为文件保存，数据保存完毕，关闭文件。

时空效率分析：时间复杂度为O（1），空间复杂度为O（n）

1. 读取文件

算法实现：用户输入要读取的文件名，打开文件，读取该文件的数据存入当前线性表，直到文件中的数据读入完成，关闭文件。

时空效率分析：时间复杂度为O（1），空间复杂度为（n）

## 系统实现

* + 1. **实验环境**

实验环境为Windows10，编译器版本为TDM-GCC 4.7.1，代码采用开源的编译器VS2017编写。由指定的MakeFile来完成编译。

文件说明：

\*.vs：VS系统文件

\*LinearList\_2.cpp:线性表及演示系统实现

\*Debug:调试文件夹

\*LinearList\_2:源码文件夹

\*LinearList\_2.sln：VS Solution文件

源码文件里面关键文件说明：

\*LinearList\_2.h:链表头文件

\*LinearList\_2.cpp:测试代码文件

\*data\_01,data\_02:存好的线性表文件

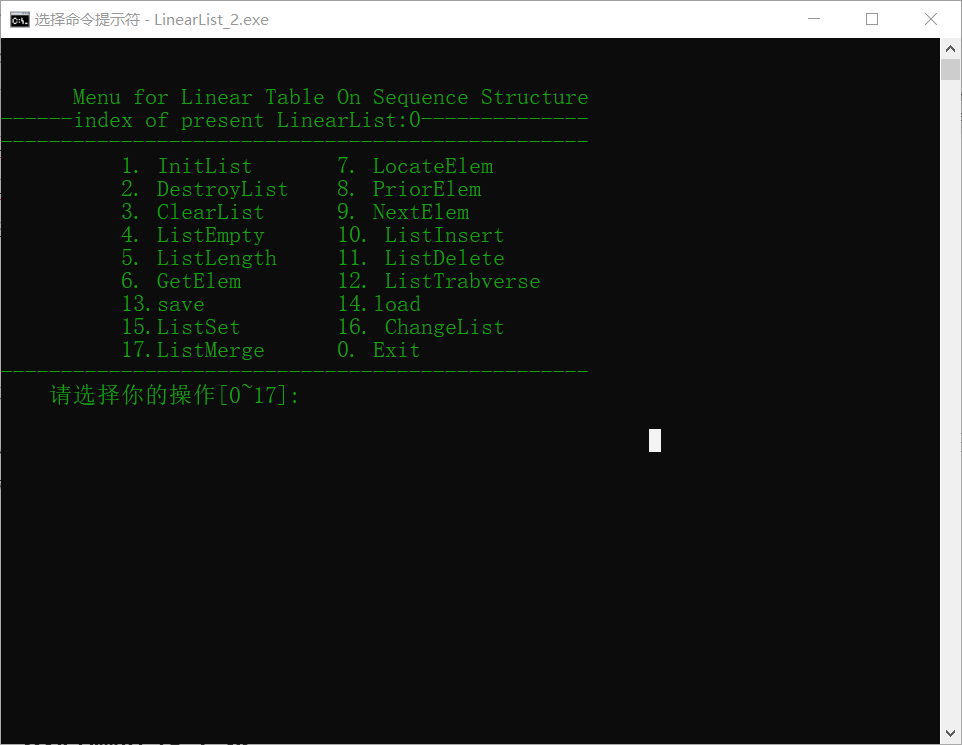
* + 1. **代码亮点**

所有代码采用Google C/C++标准代码规范，函数库所有注释采用英文，符合生产规范。错误检查和提示全面，根据不同的错误会有不同的提示信息。由于在Windows环境下编程，所有的API接口命名采用标准短横线命名模式。

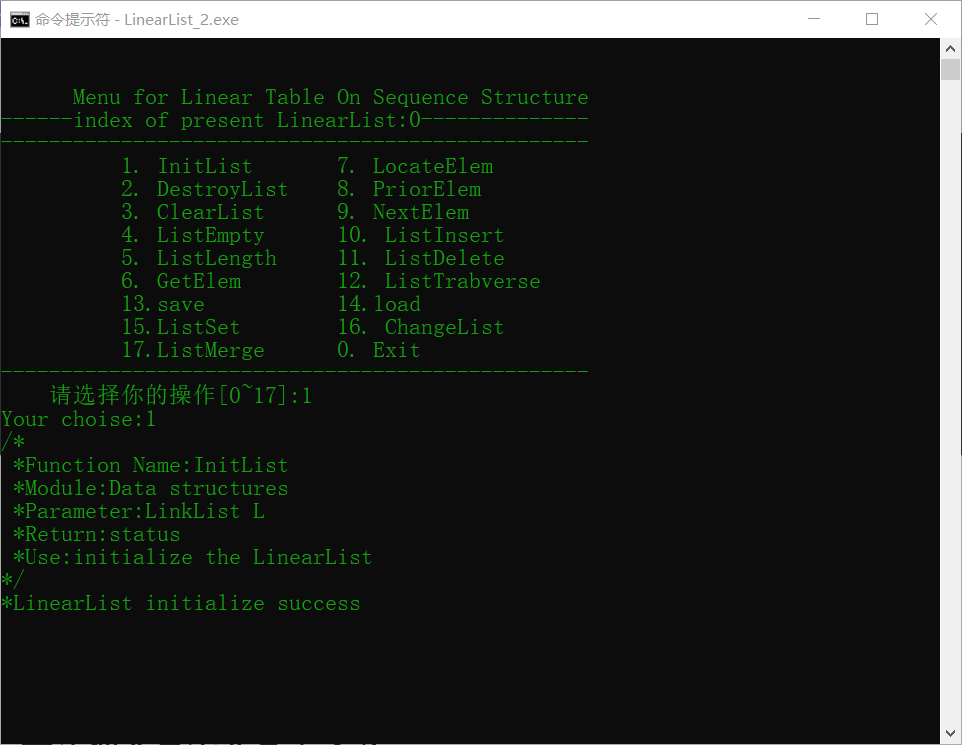
用户每次输入操作都会整齐打印出函数的名称，参数，返回值和作用，而且是用英文，专有名词使用准确不会产生歧义，整个演示系统整洁美观，对用户而言具有操作友好性。

* + 1. **操作演示**

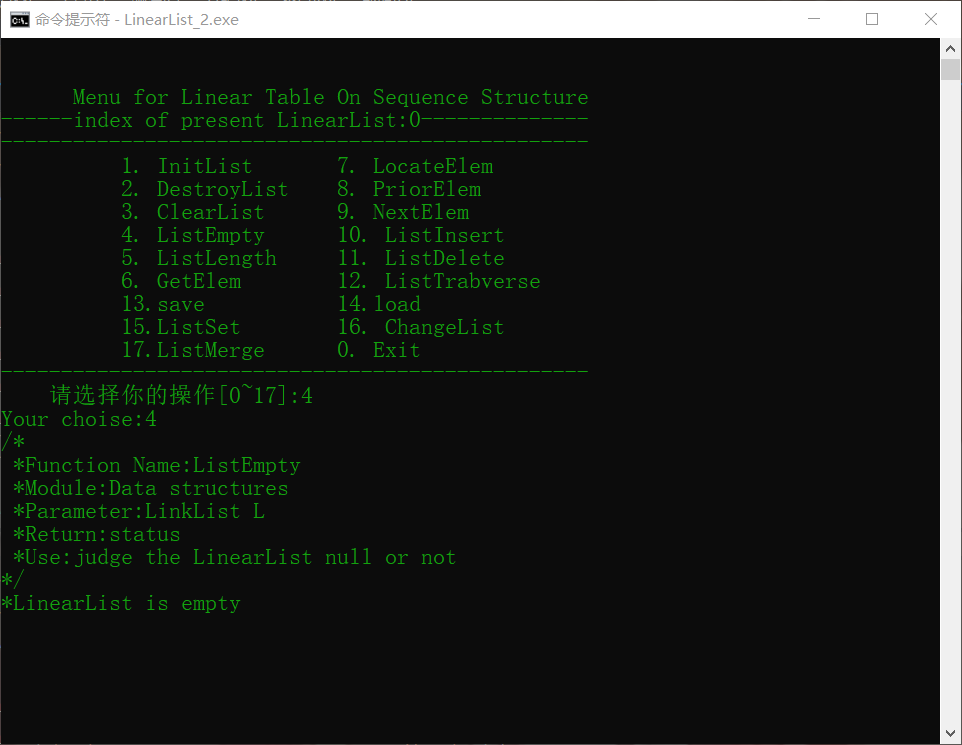
**界面展示：**

****

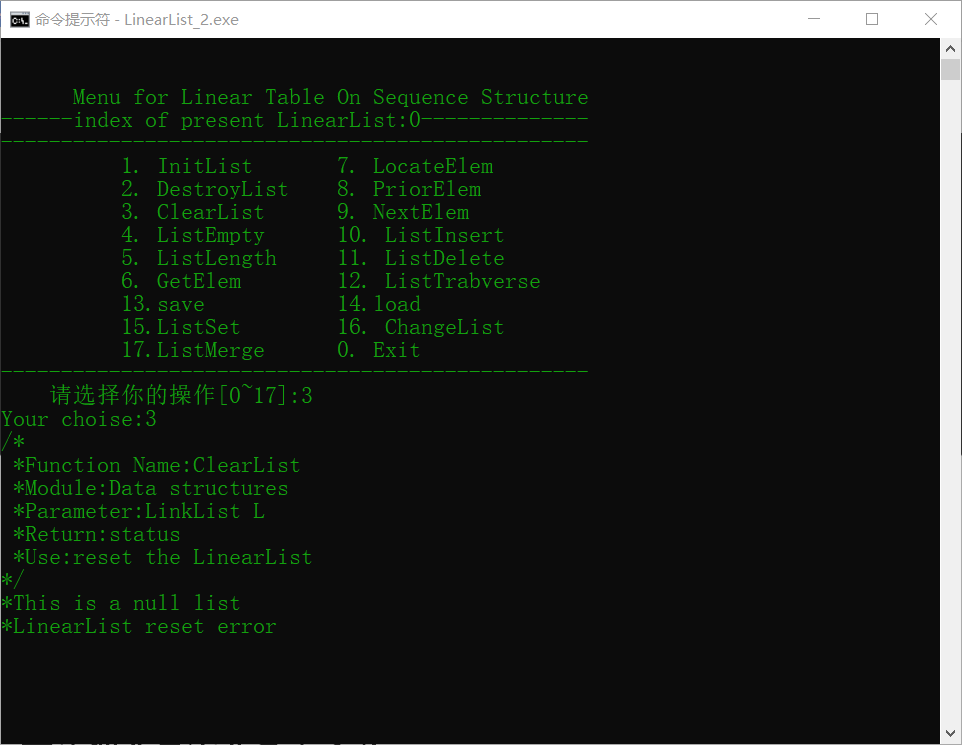
1. **首先创建一个线性表**

****

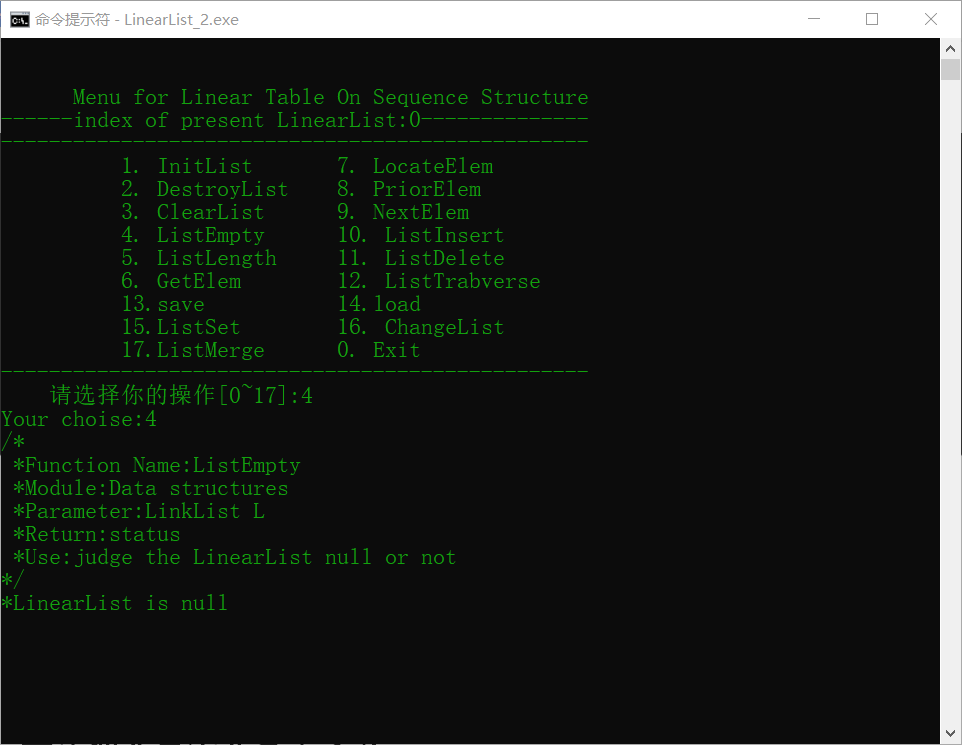
1. **然后用ListEmpty查看，显示这个新建的表是空表**

****

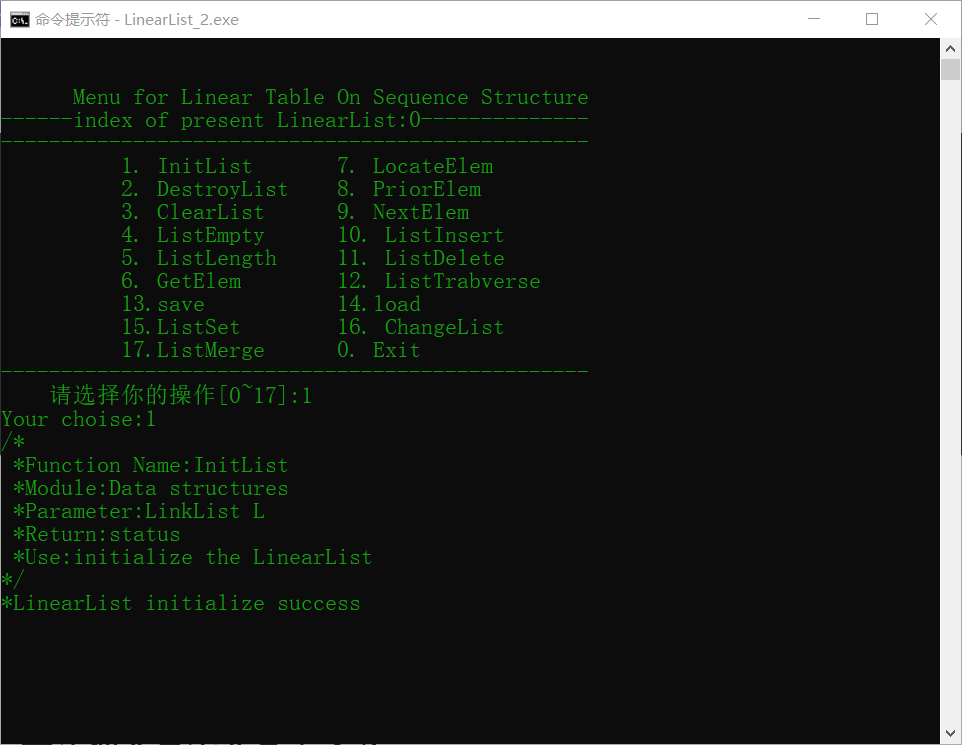
1. **然后删除这个表**

****

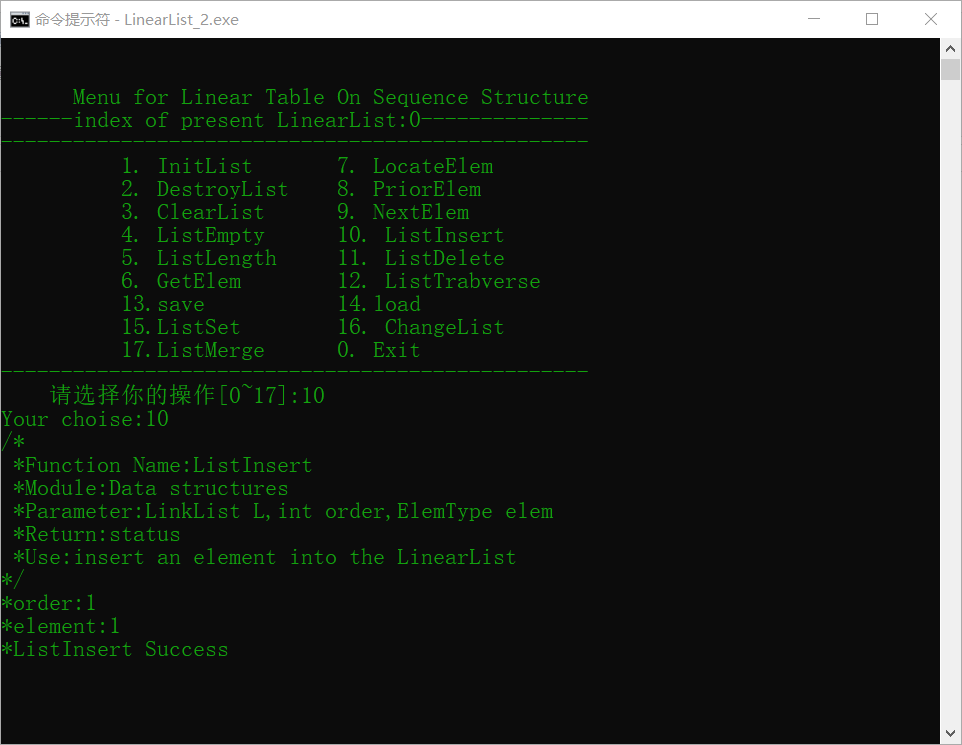
1. **再用ListEmpty查看，显示线性表不存在**

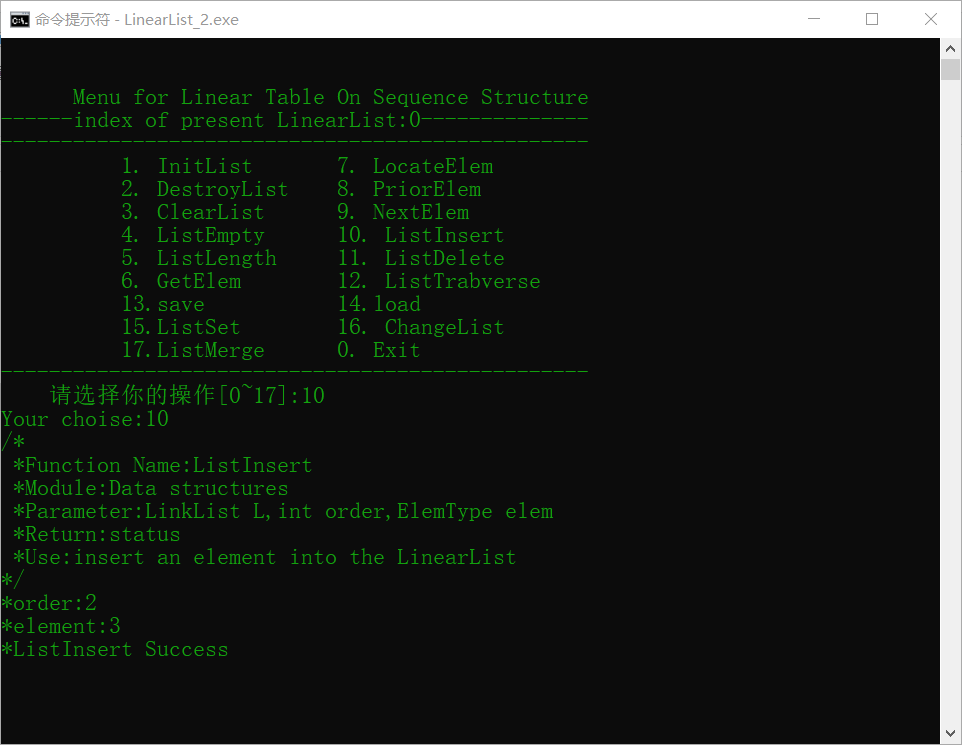
****

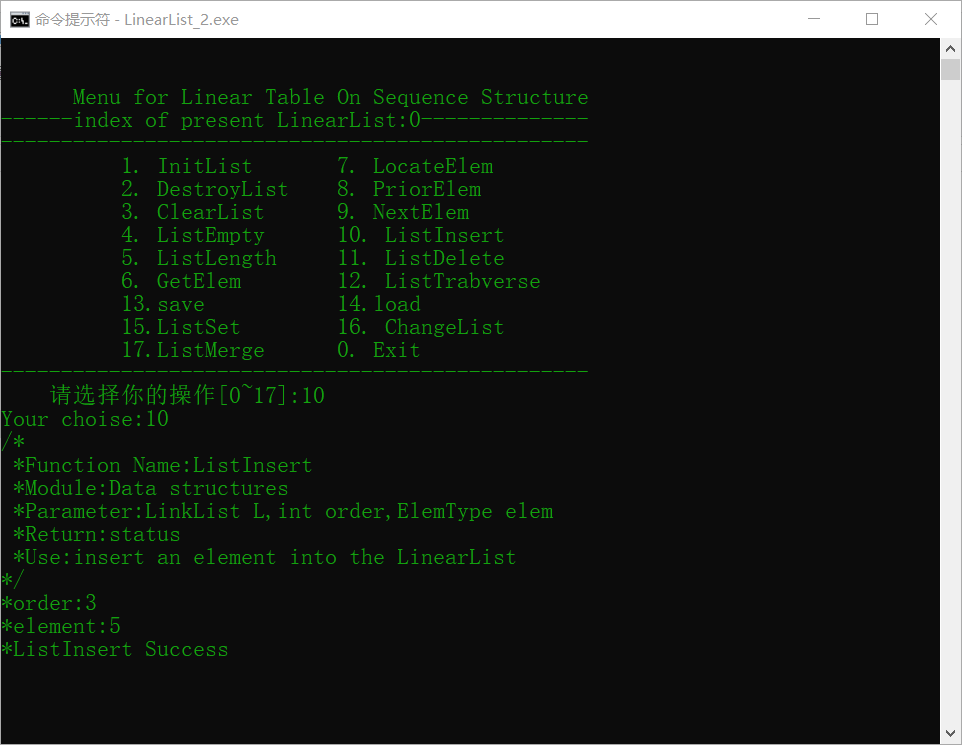
1. **重新创建一个线性表**

****

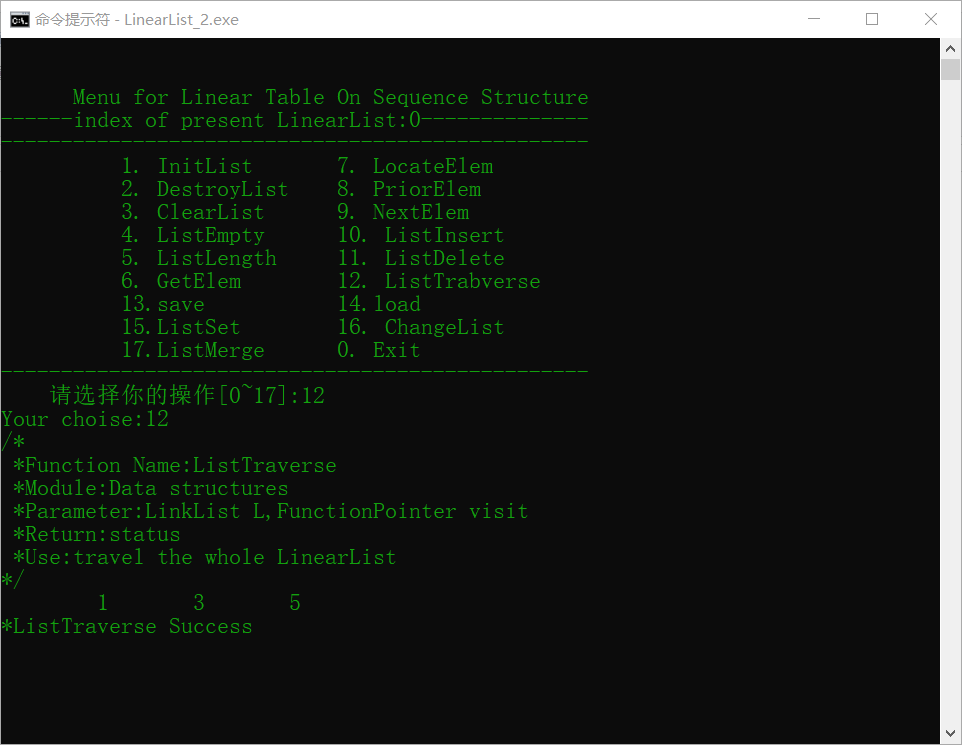
1. **一次插入三个元素1,3,5**

****

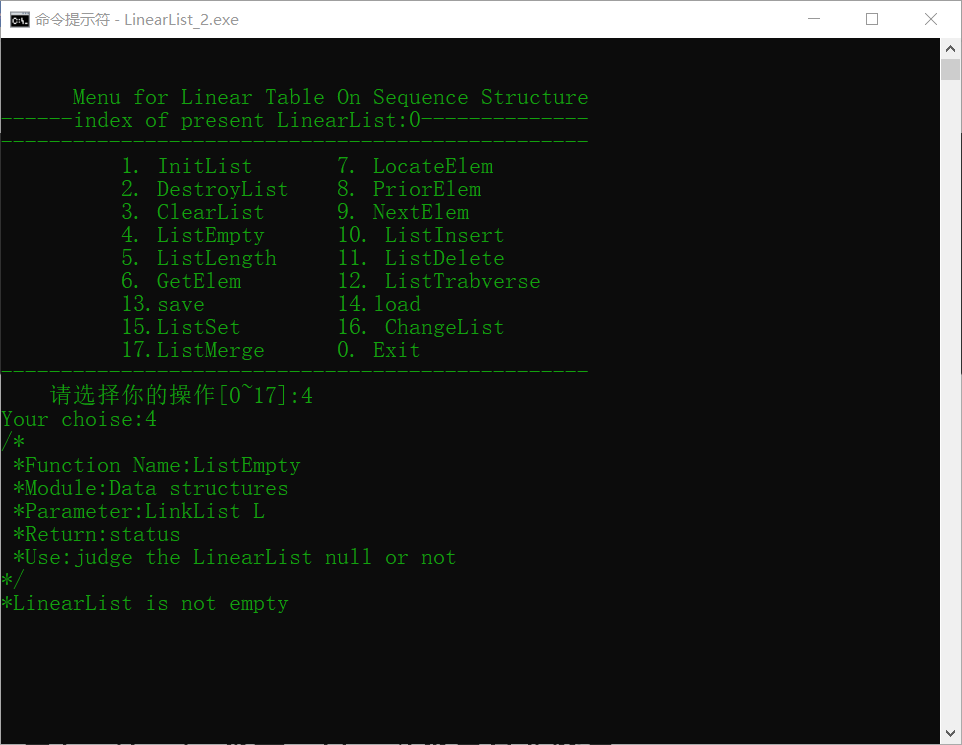
****

****

1. **打印这个表（操作12中的visit函数是将元素的值打印）**

****

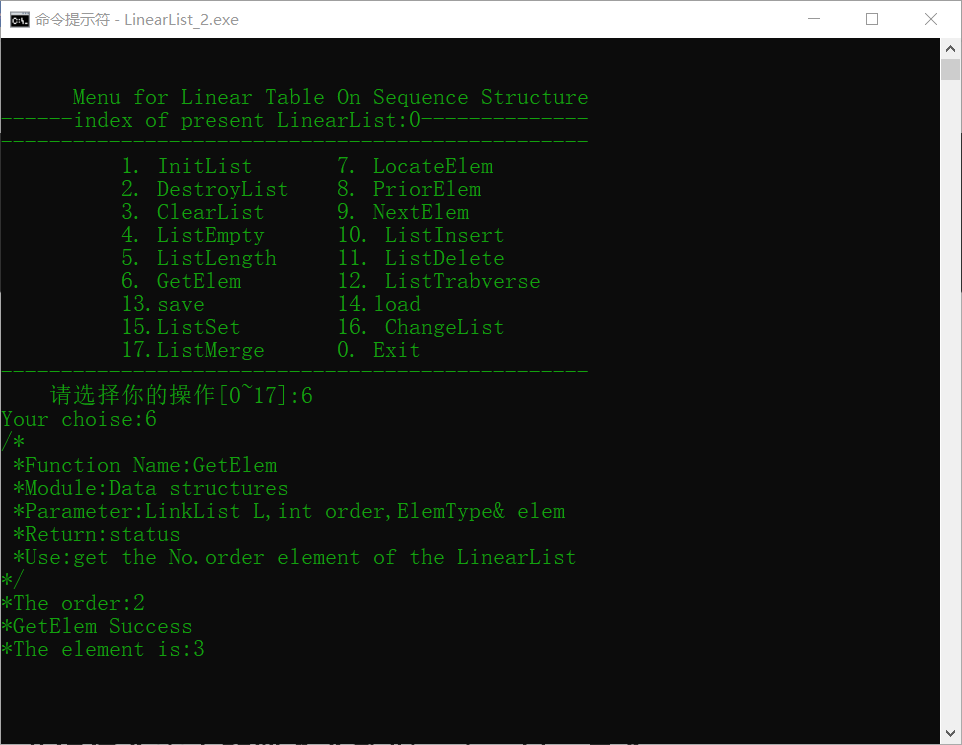
1. **用ListEmpty查看，显示这个表是非空表**

****

1. **查看这个表的长度，显示是3**

****

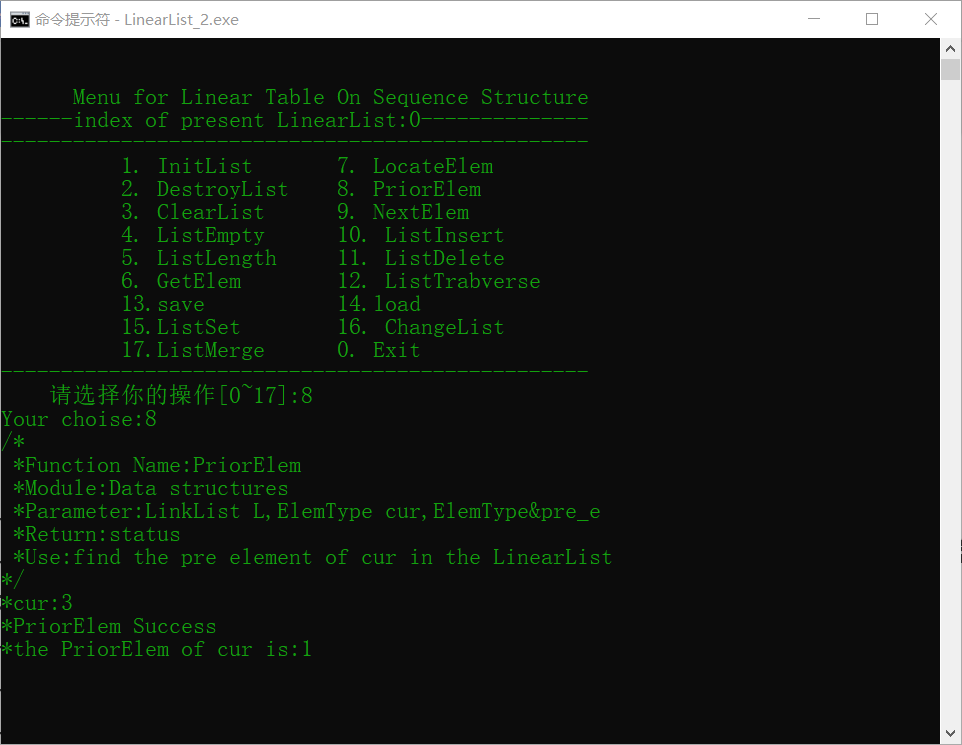
1. **获得这个表中的第2个数据元素，显示是3**

****

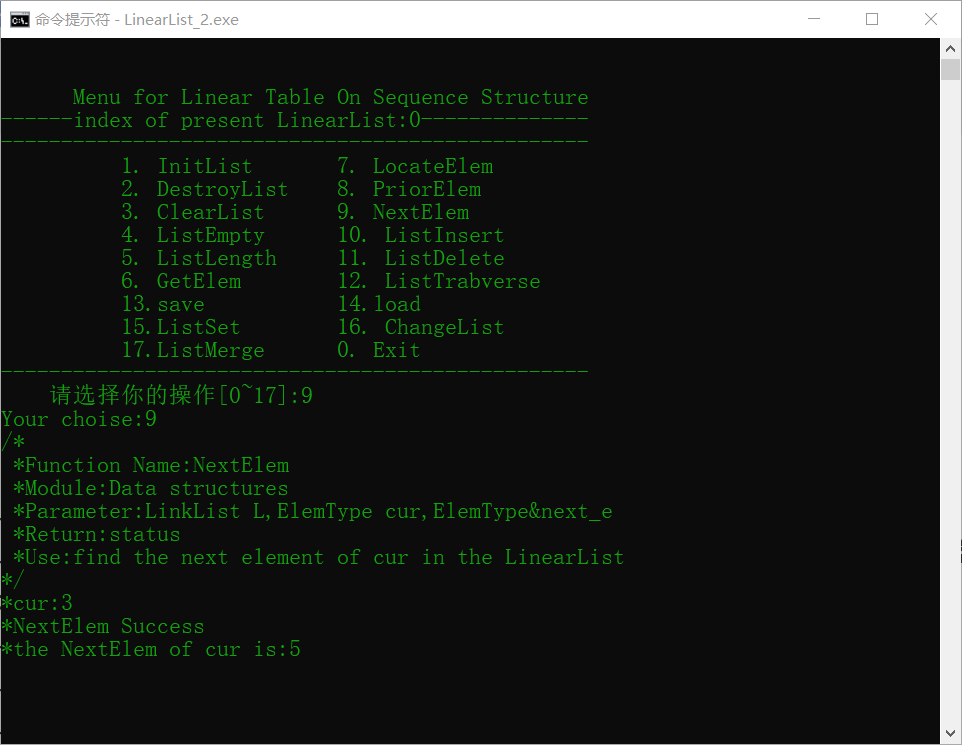
1. **获得这个表中第一个与3相等的数据元素（操作7的compare函数是判断值是否相等）**

****

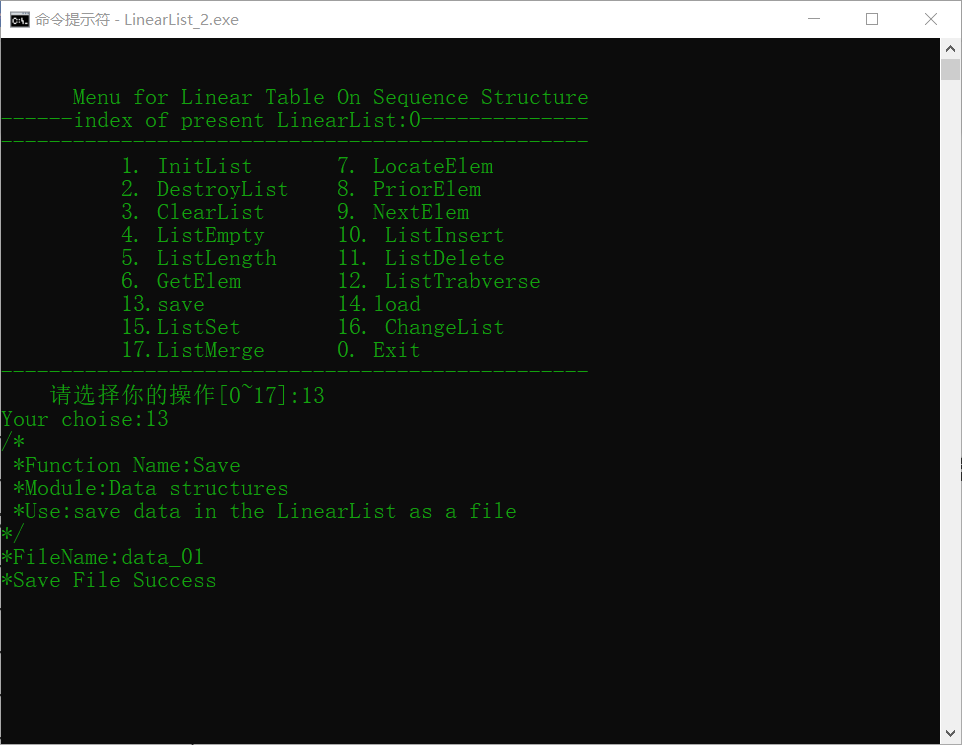
1. **获得元素3之前的元素**

****

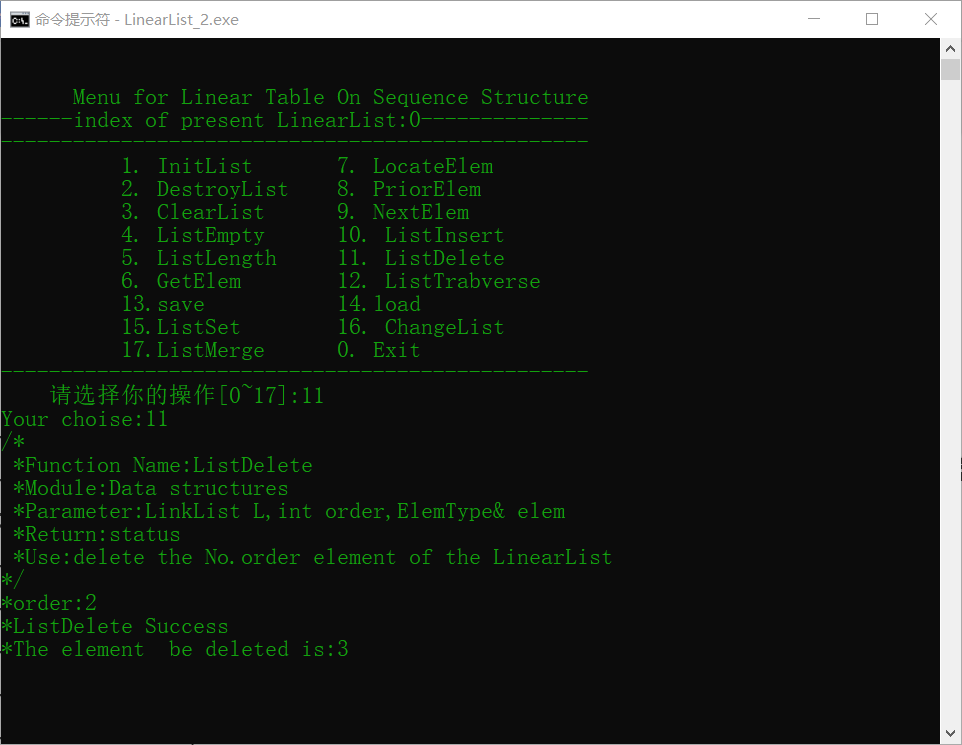
1. **获得元素3之后的元素**

****

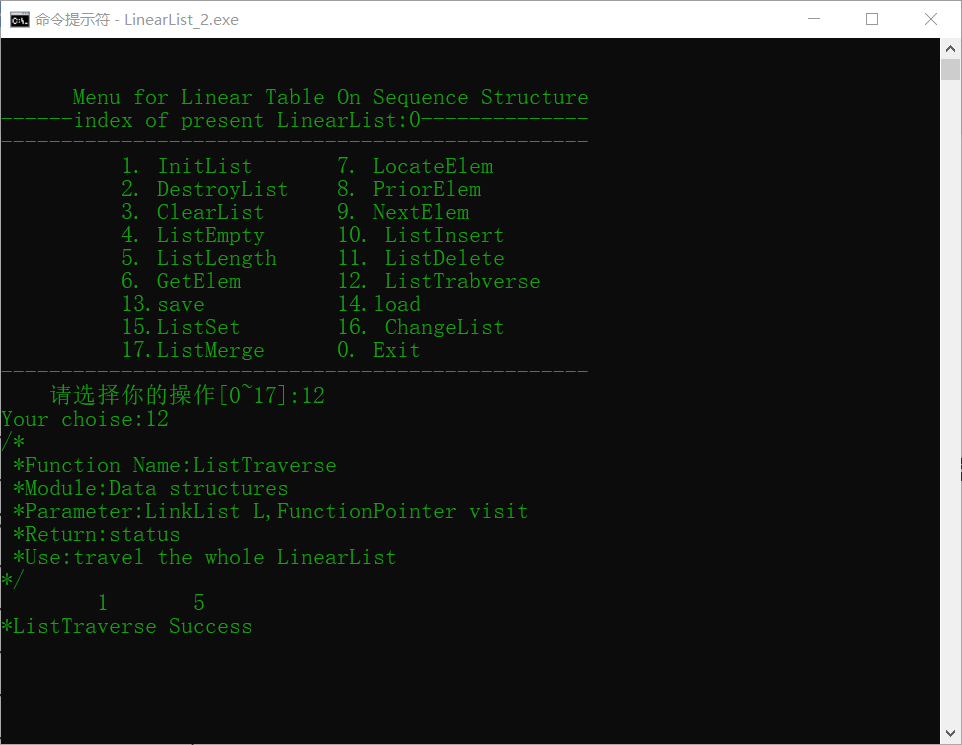
1. **将当前线性表的数据写入“data\_01”**

****

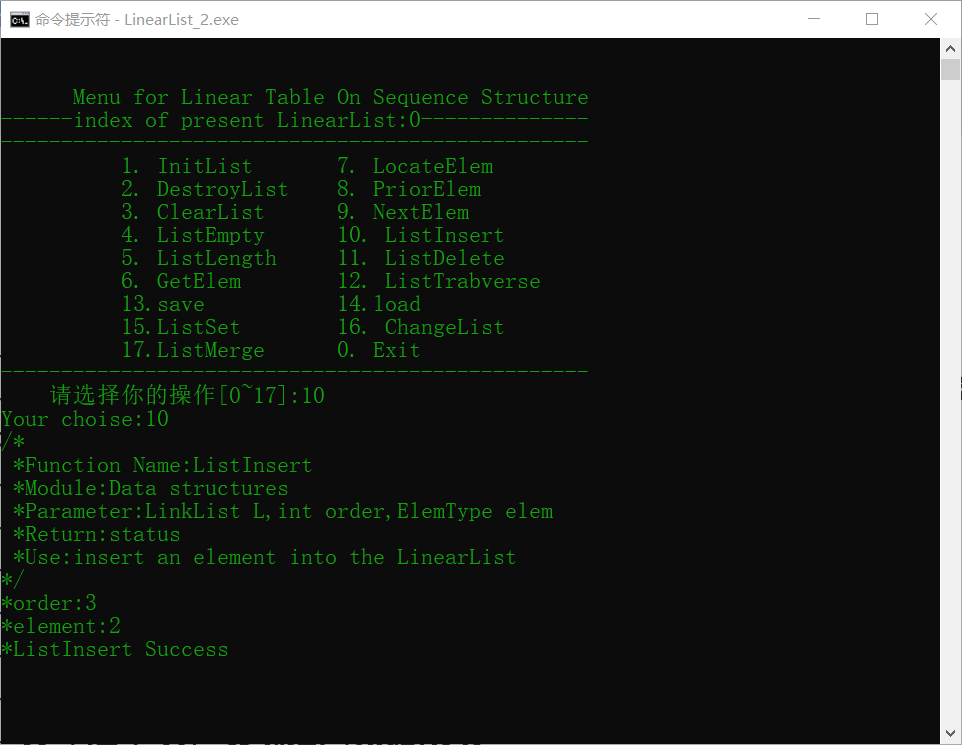
1. **删除当前表的第2个元素**

****

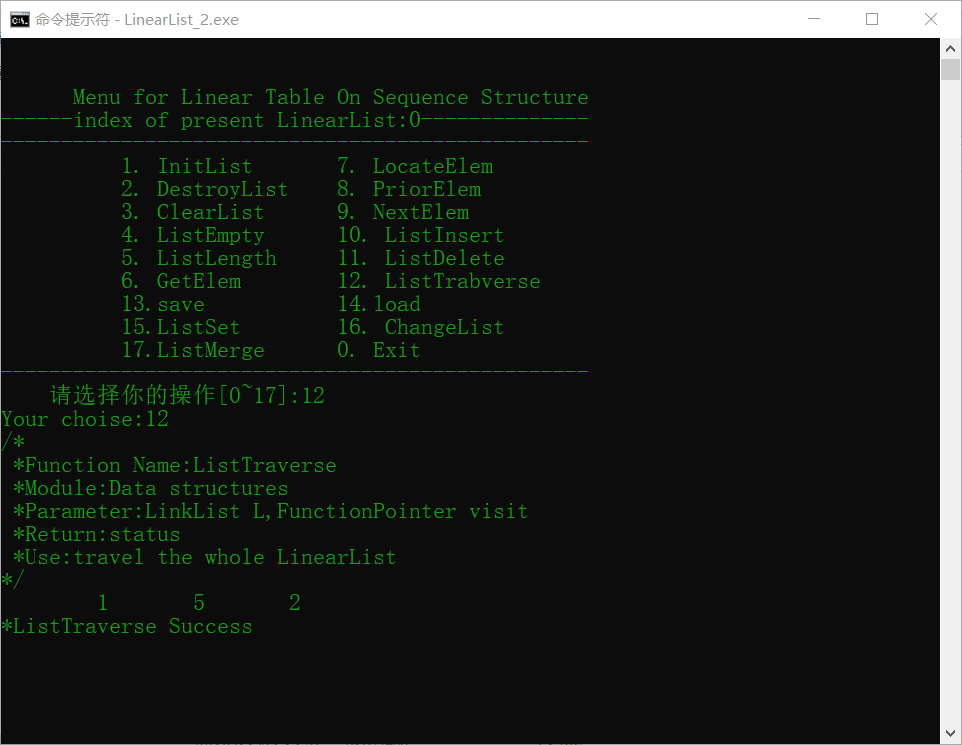
1. **打印这个表，发现第二个元素成功删除**

****

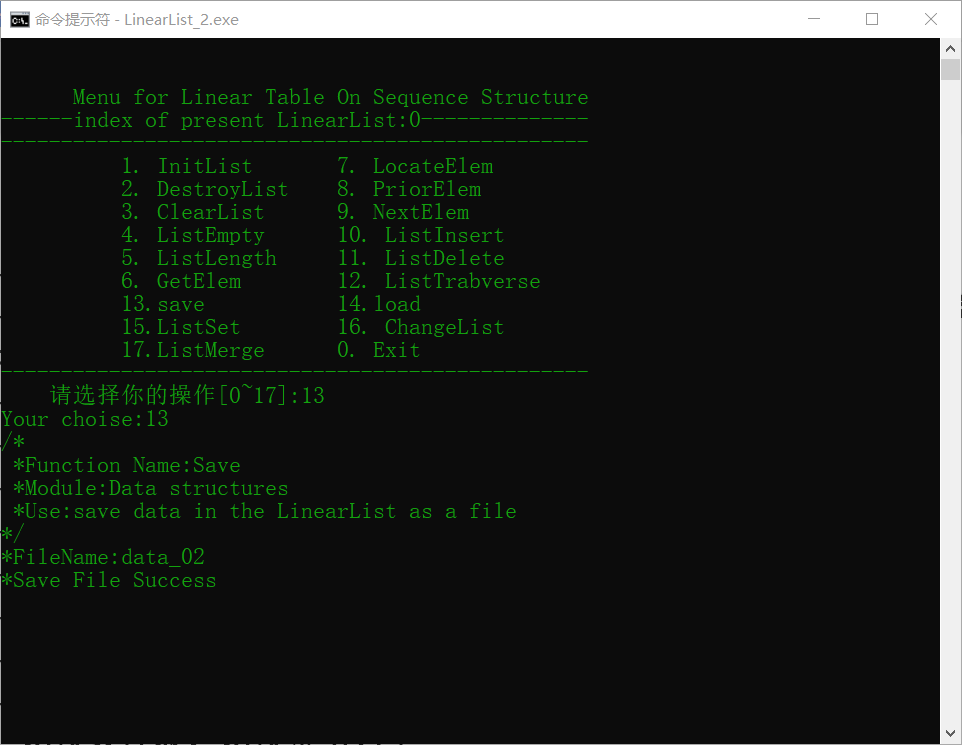
1. **在表的第三个位置插入2**

****

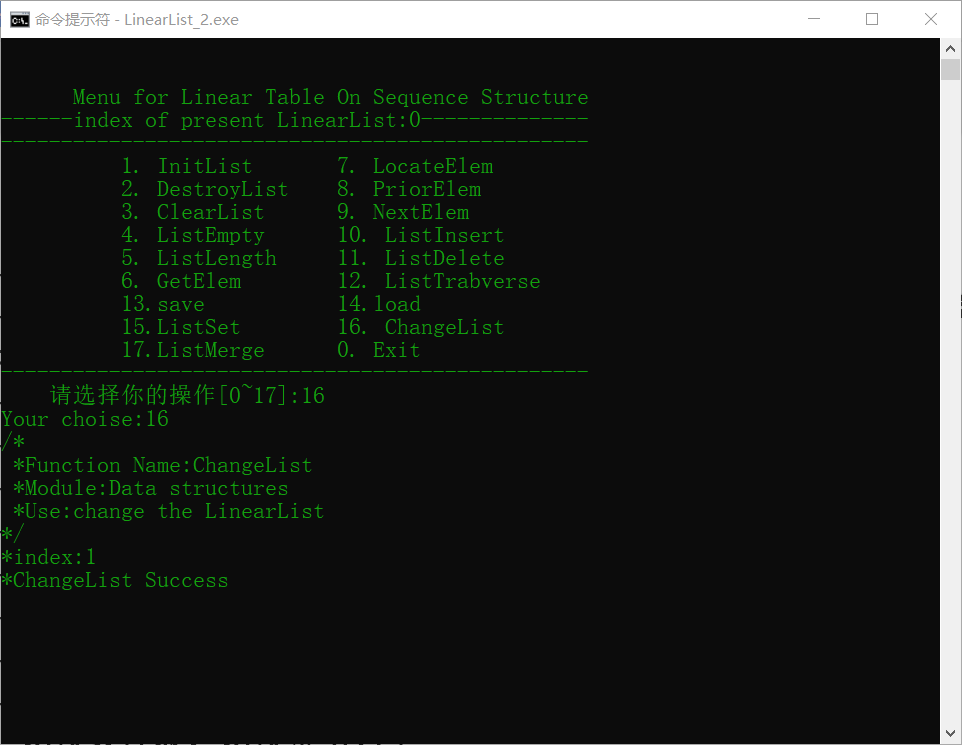
1. **打印这个表，发现插入数据成功**

****

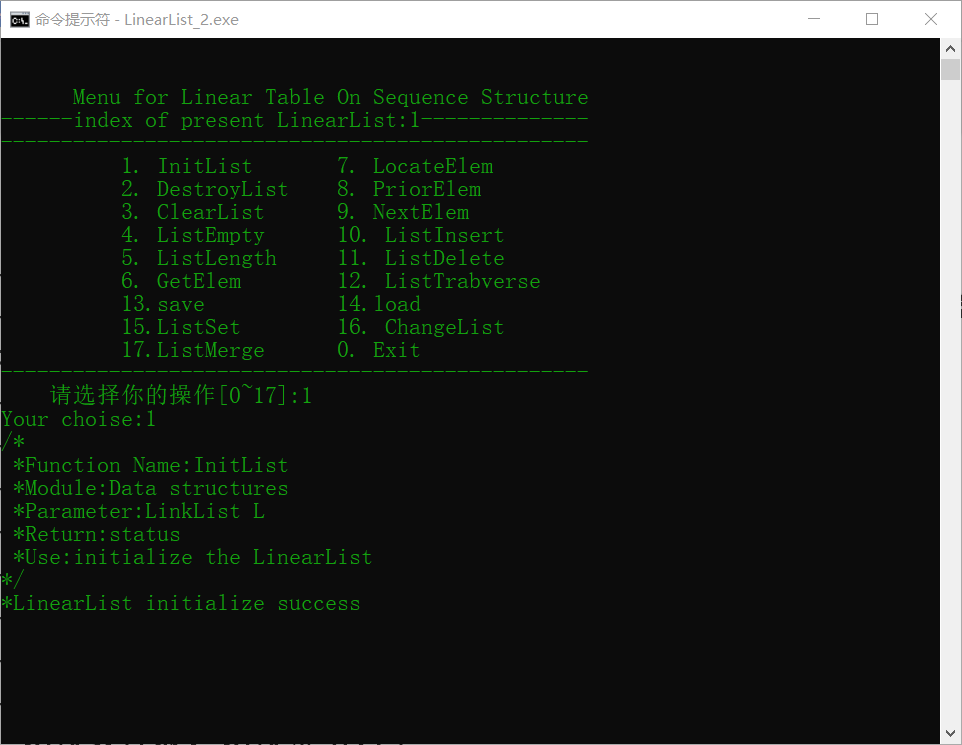
1. **再将当前表的数据写入文件“data\_02”**

****

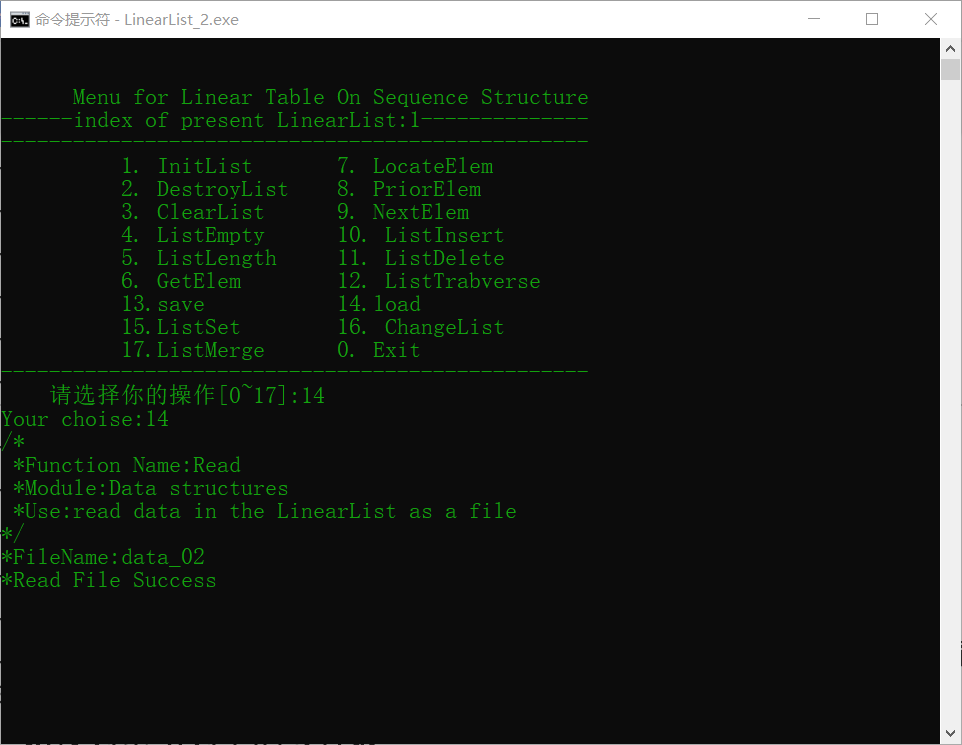
1. **切换线性表，切换索引为1**

****

1. **创建一个新表**

****

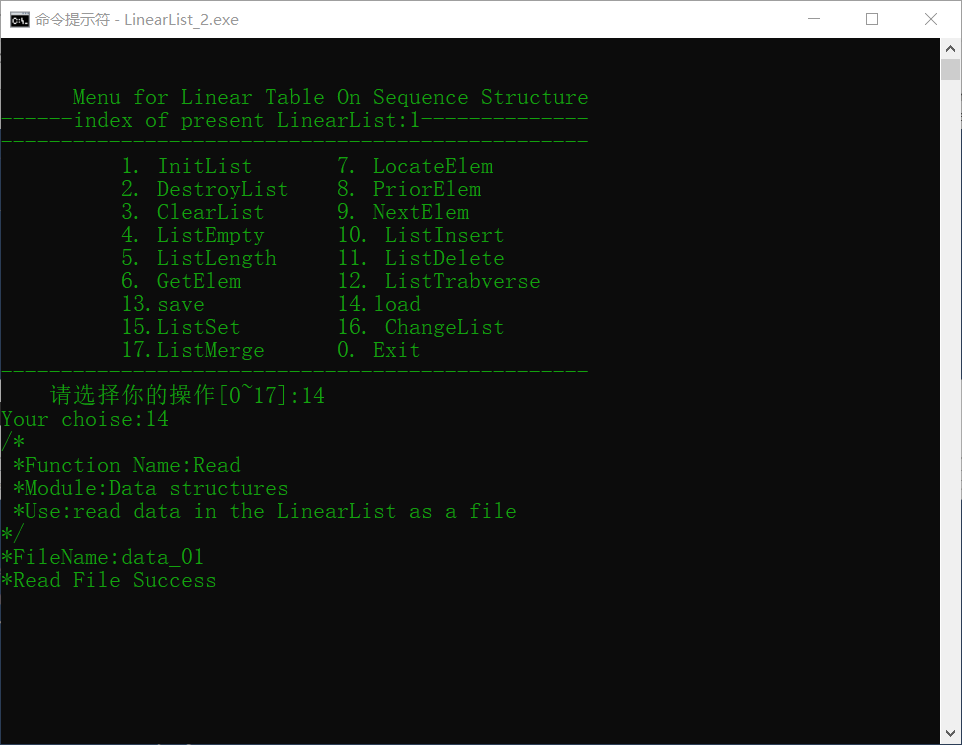
1. **将文件“data\_02”的数据加载进当前线性表**

****

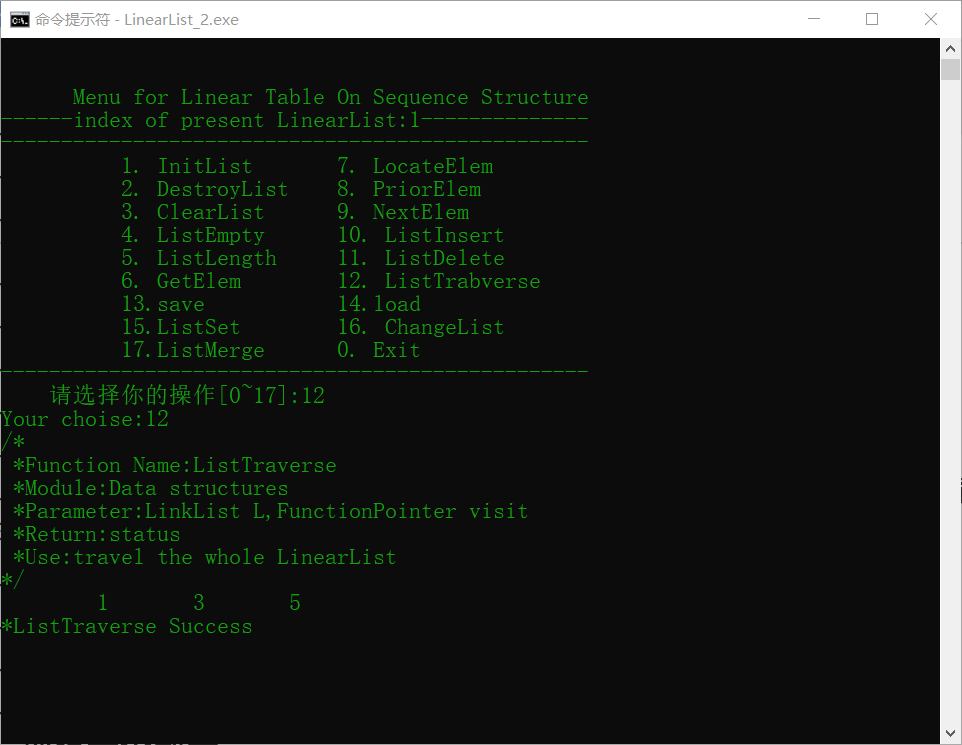
1. **打印这个表，发现数据被成功加载**

****

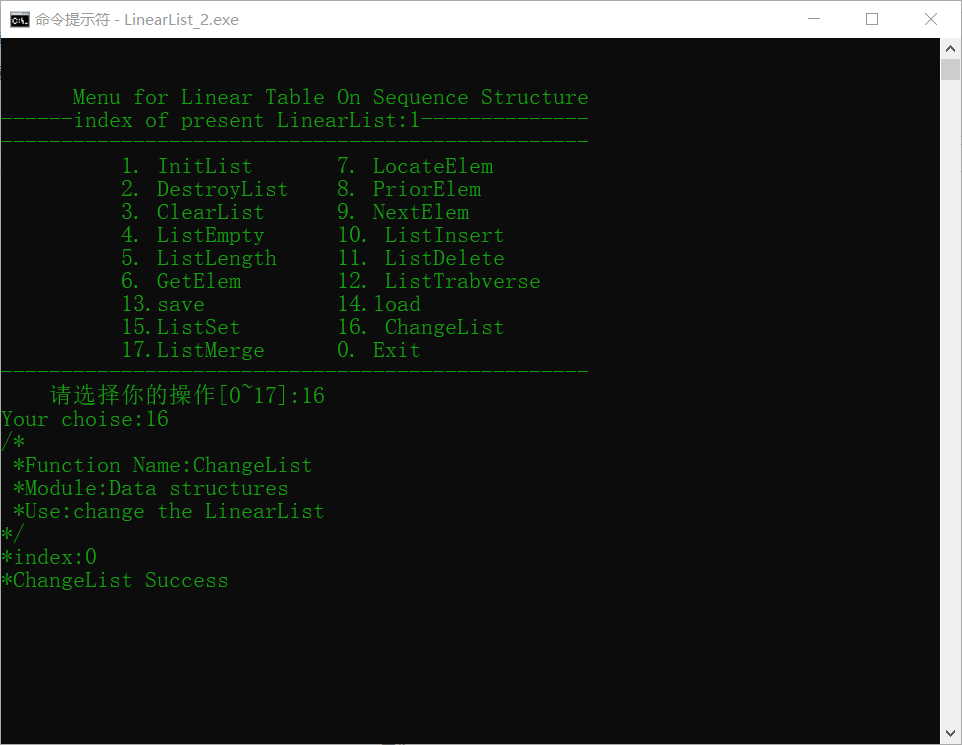
1. **再将文件“data\_01”的数据加载进当前线性表**

****

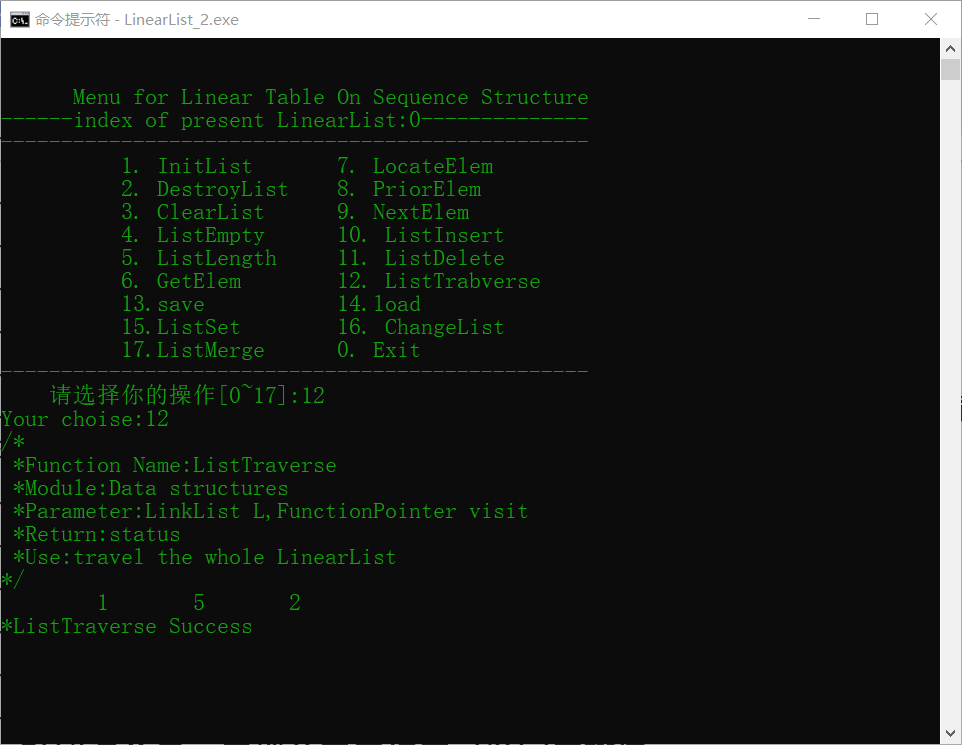
1. **打印这个表，发现数据也被成功加载**

****

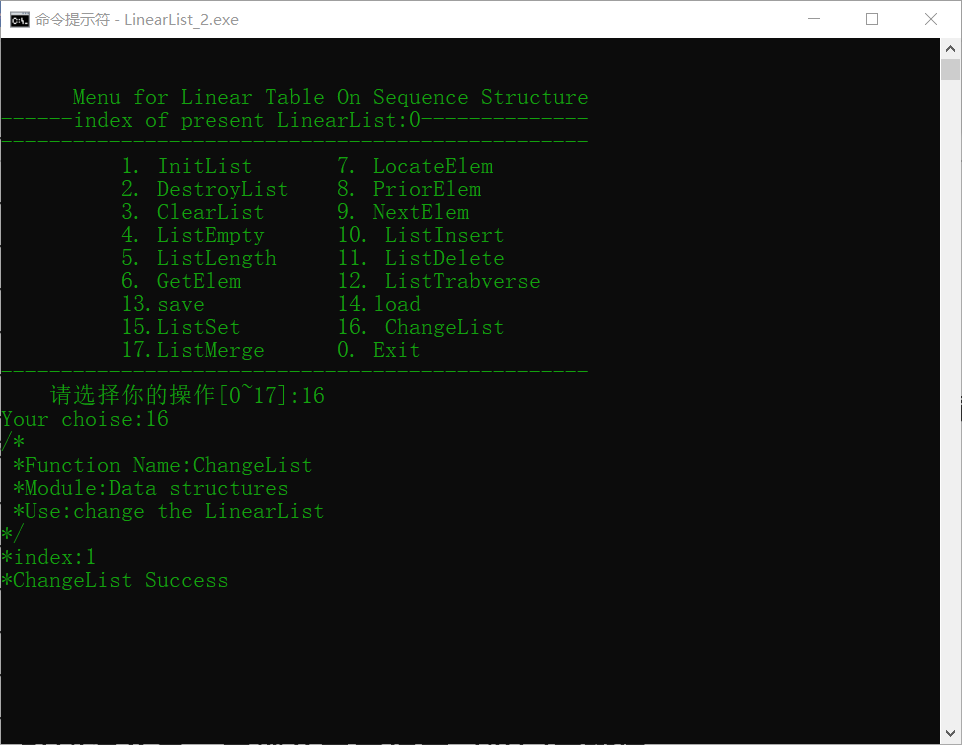
1. **切换回索引为0的线性表**

****

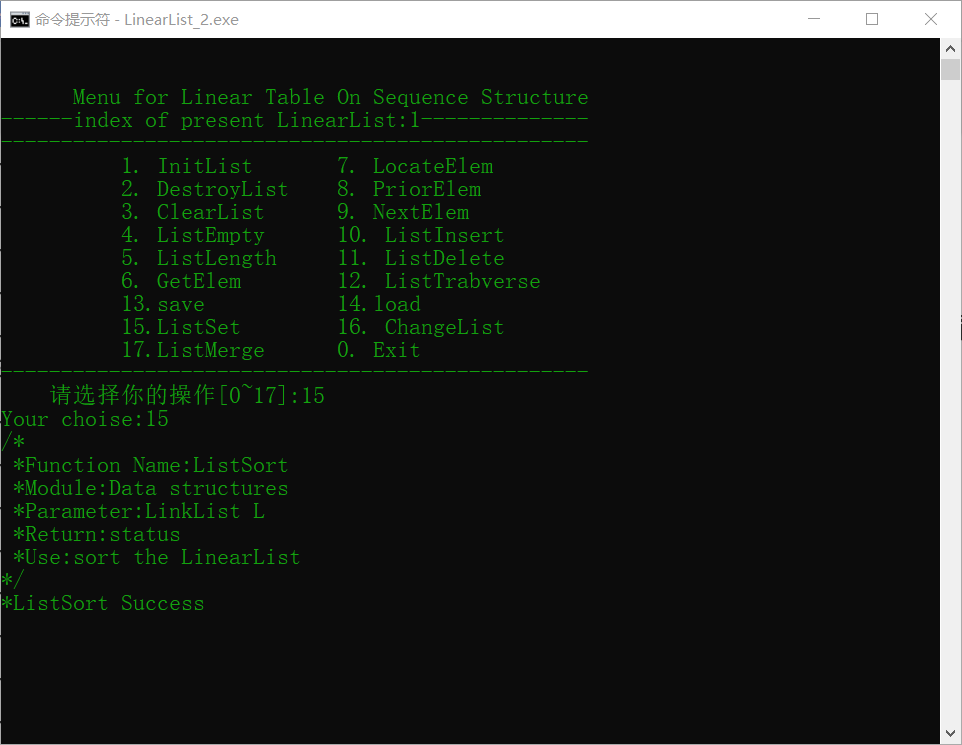
1. **打印当前表，显示的数据与data\_02中的数据相同，说明各线性表之间的操作相互独立**

****

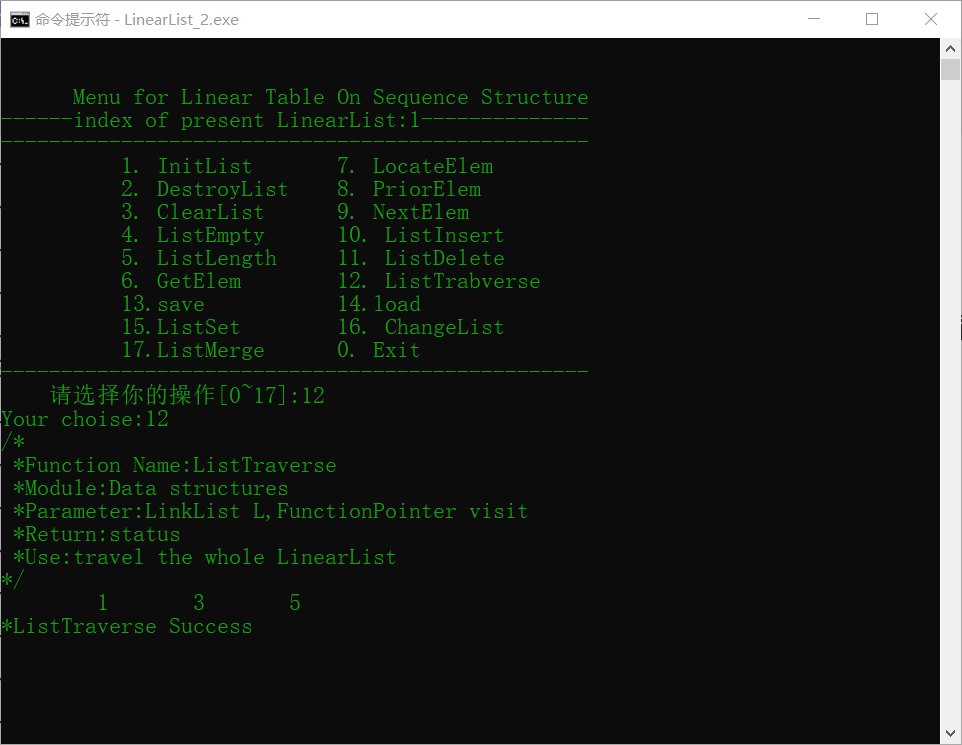
1. **切换为索引为1的线性表**

****

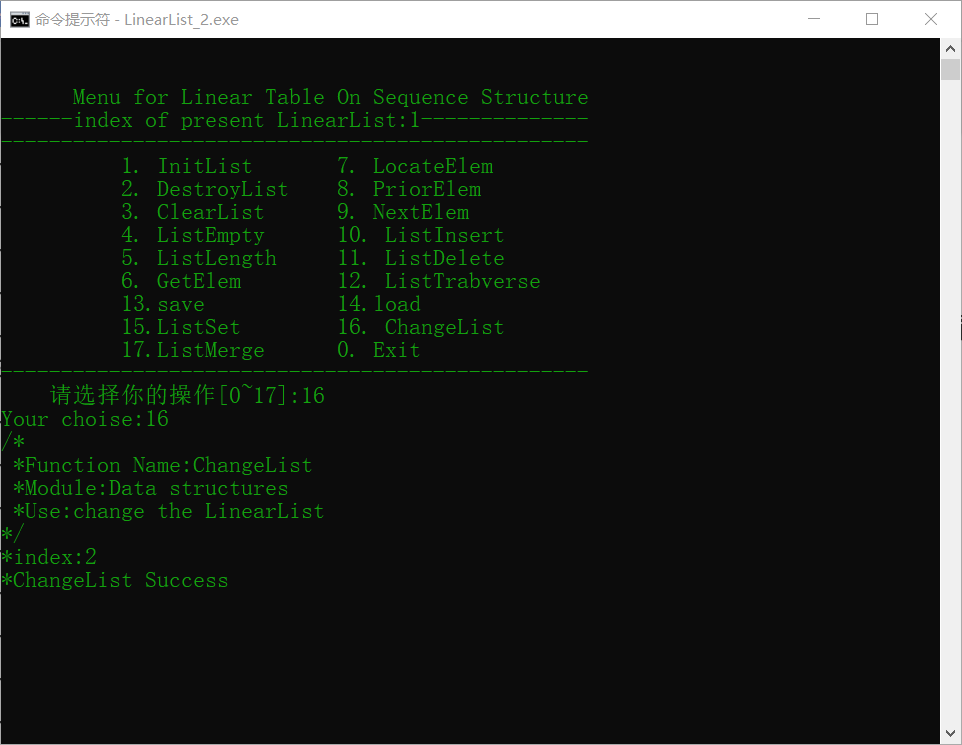
1. **将当前线性表进行排序**

****

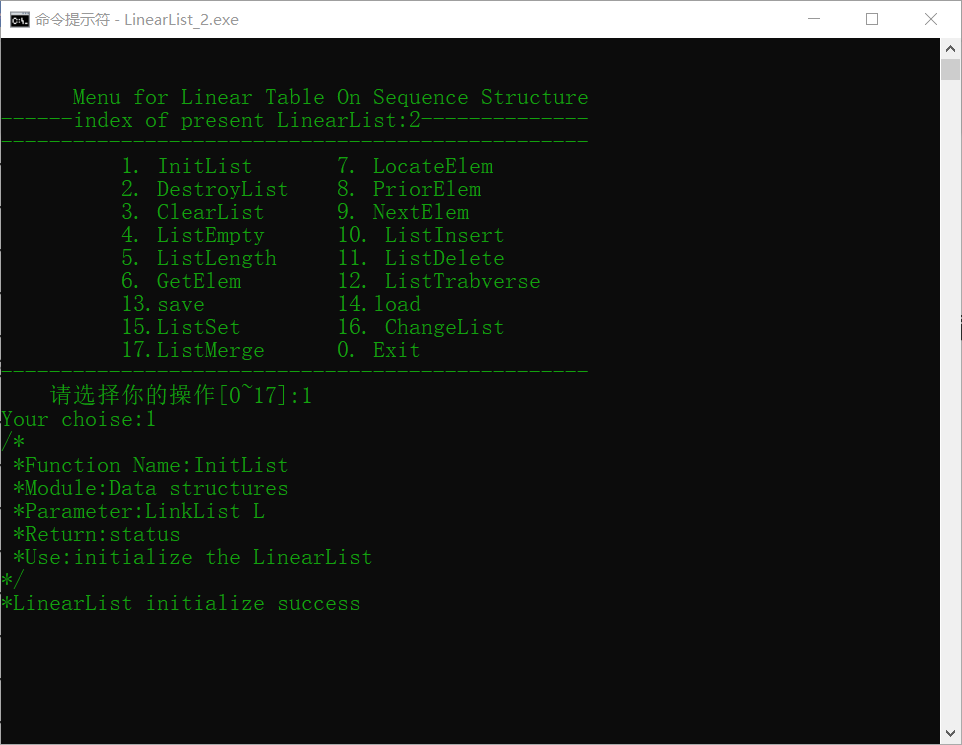
1. **打印这个表，发现排序成功**

****

1. **切换为索引为2的线性表**

****

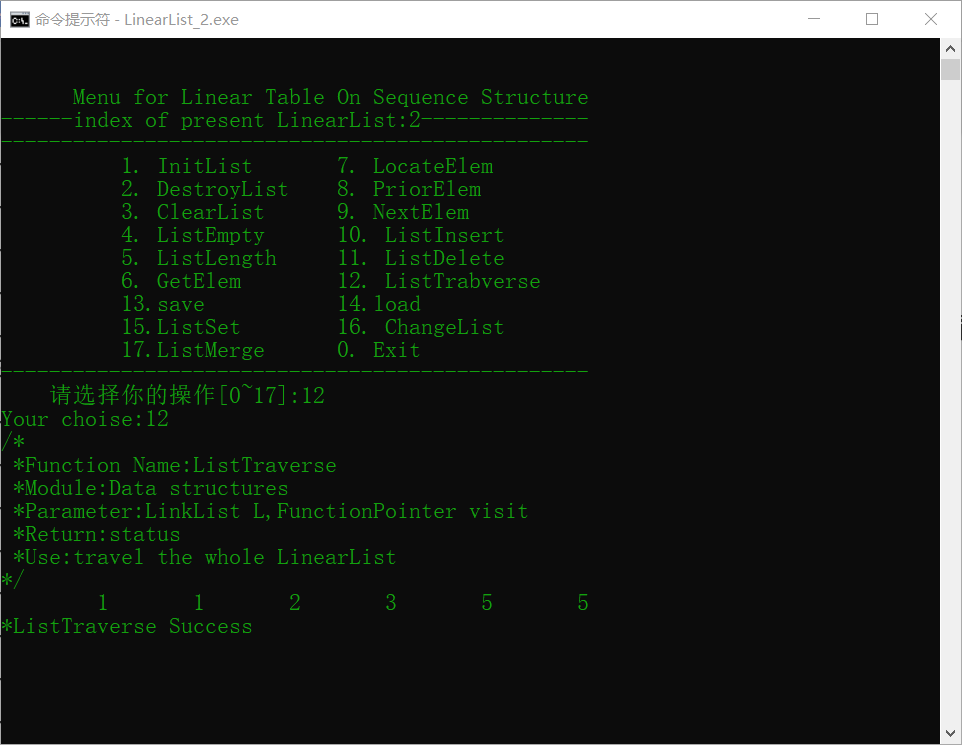
1. **创建一个新表**

****

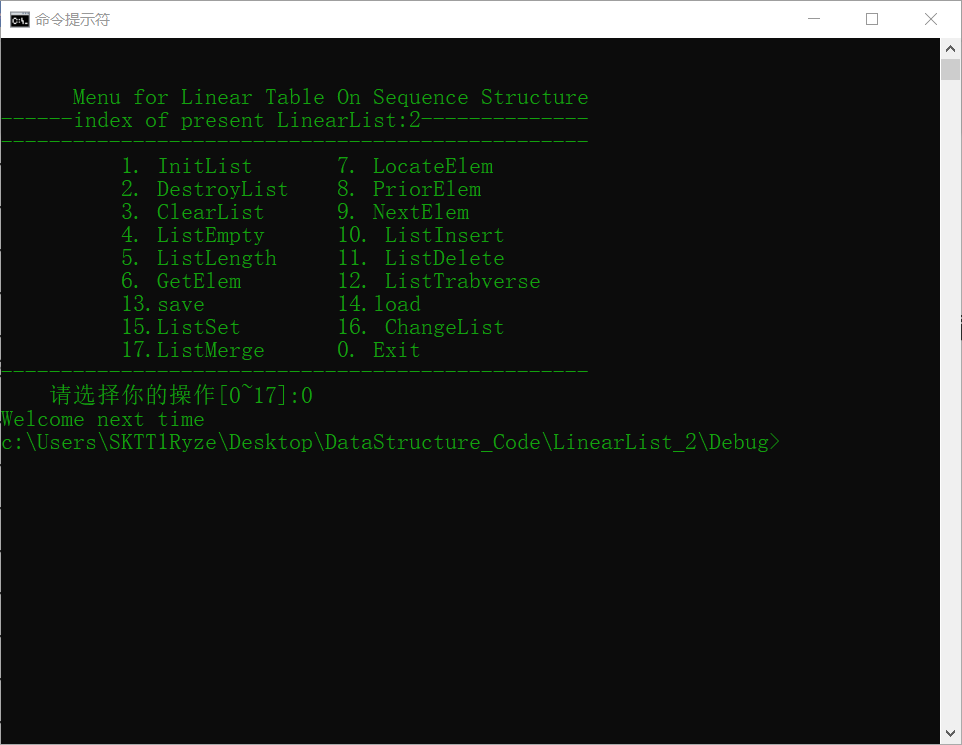
1. **将线性表0和线性表1合并为线性表2**

****

1. **打印这个表，发现合并成功并已经完成排序**

****

1. **正常退出系统**

****

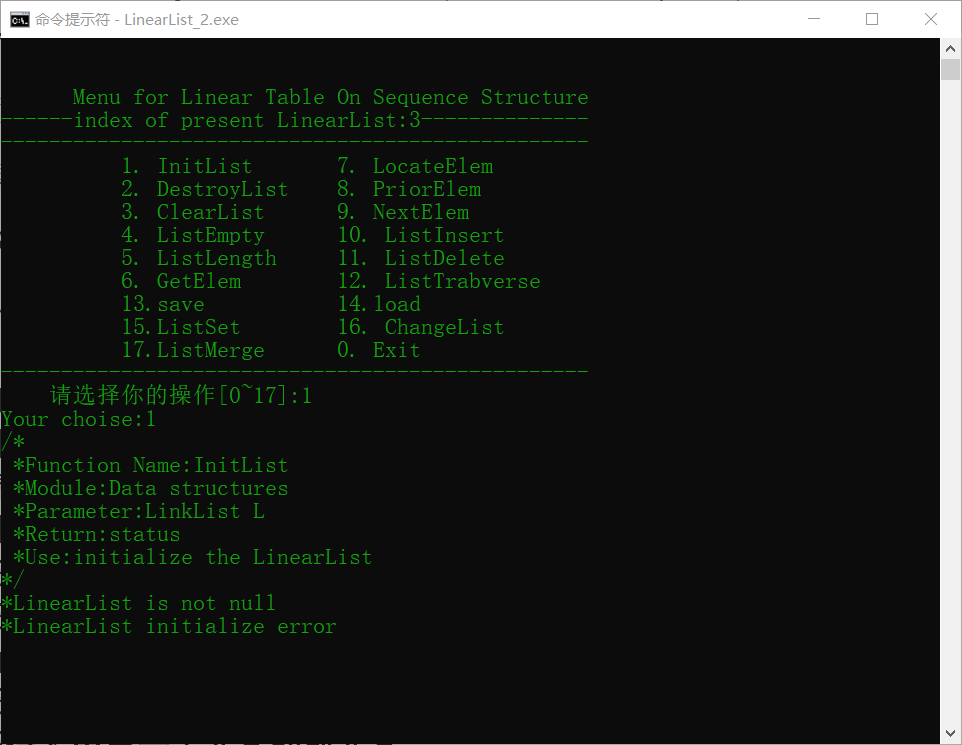
通过以上的演示证明，我们的线性表的功能是完整实现了的，可以演示要求的十二种线性表上的运算，并且做到了多线性表管理，实现了文件存储功能，还有线性表排序和线性表合并功能，不存在内存泄漏，完整的达到了实验要求。

## 系统测试

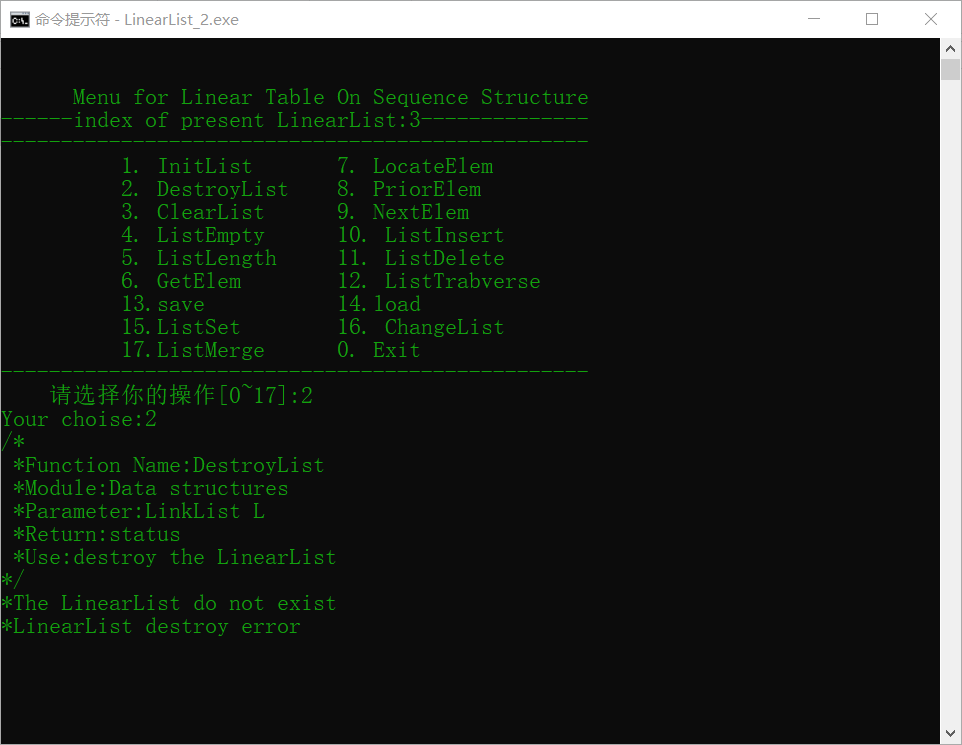
在上面操作的基础上，我们再创建两个线性表，索引分别为3和4，索引为3的线性表中的数据元素是1,4,2,7,5；索引为4的线性表中的数据元素是12,4,10,3,6；分别写入文件“data\_03”和“data\_04”

下面以这两个线性表基础，进行系统测试。

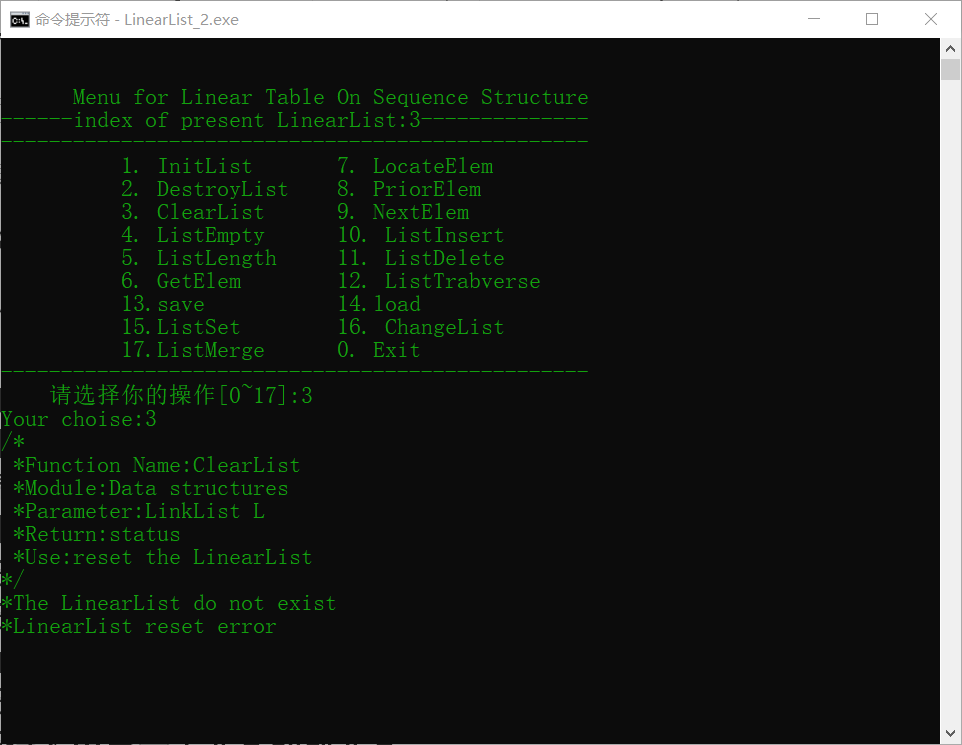
1. **初始化已存在的线性表**

****

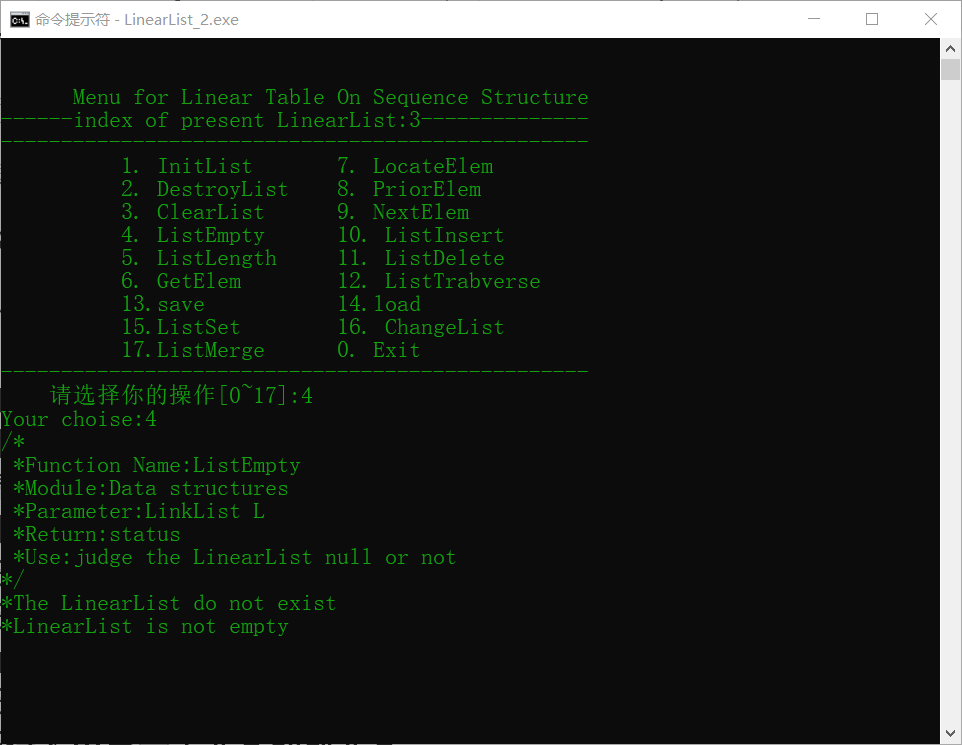
1. **删除不存在的线性表**

****

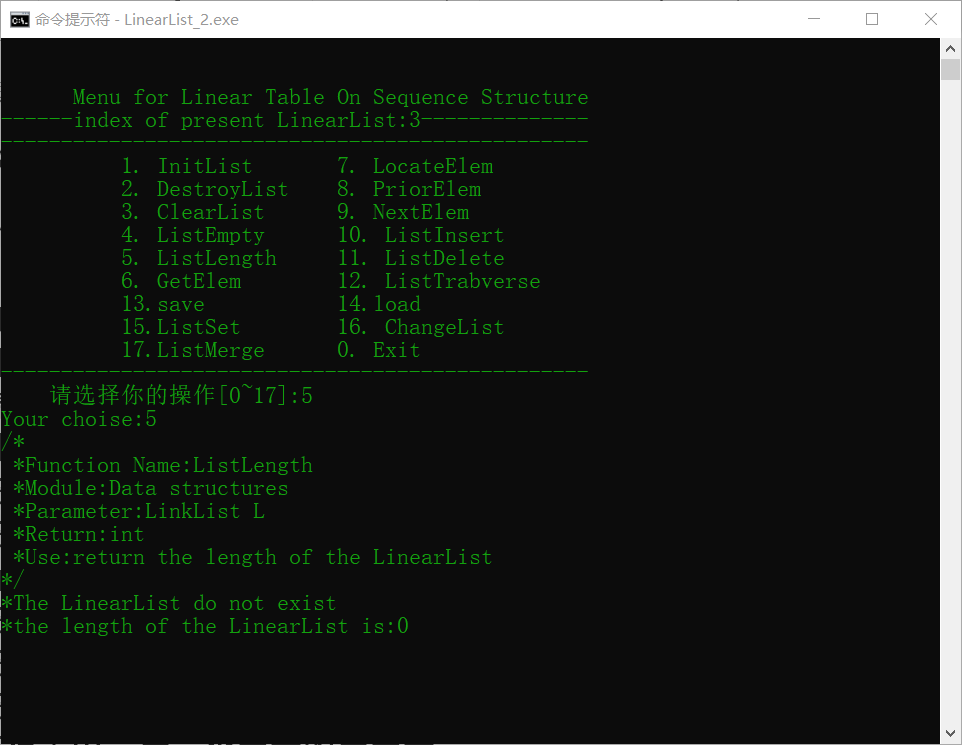
1. **清空不存在的线性表**

****

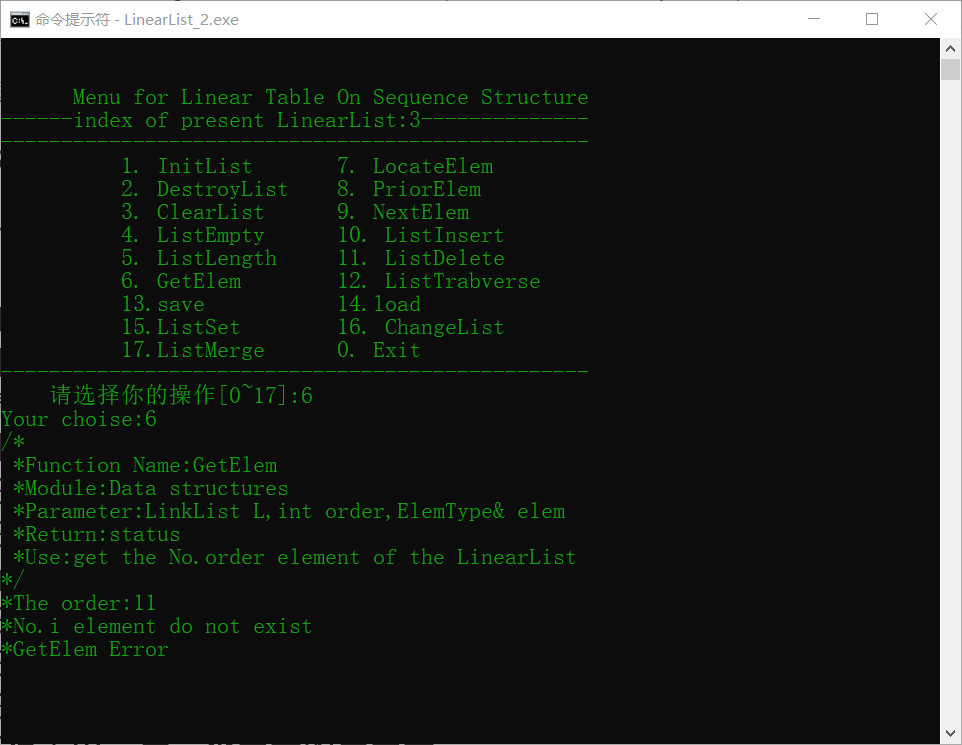
1. **判断不存在的线性表是否为空**

****

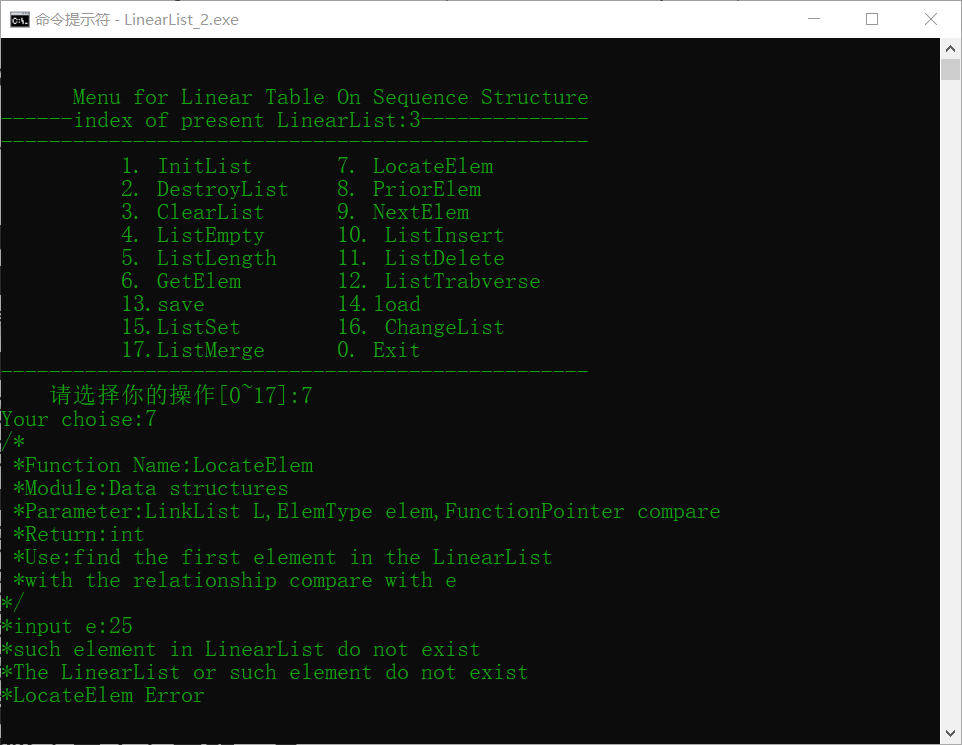
1. **获得不存在的线性表的表长**

****

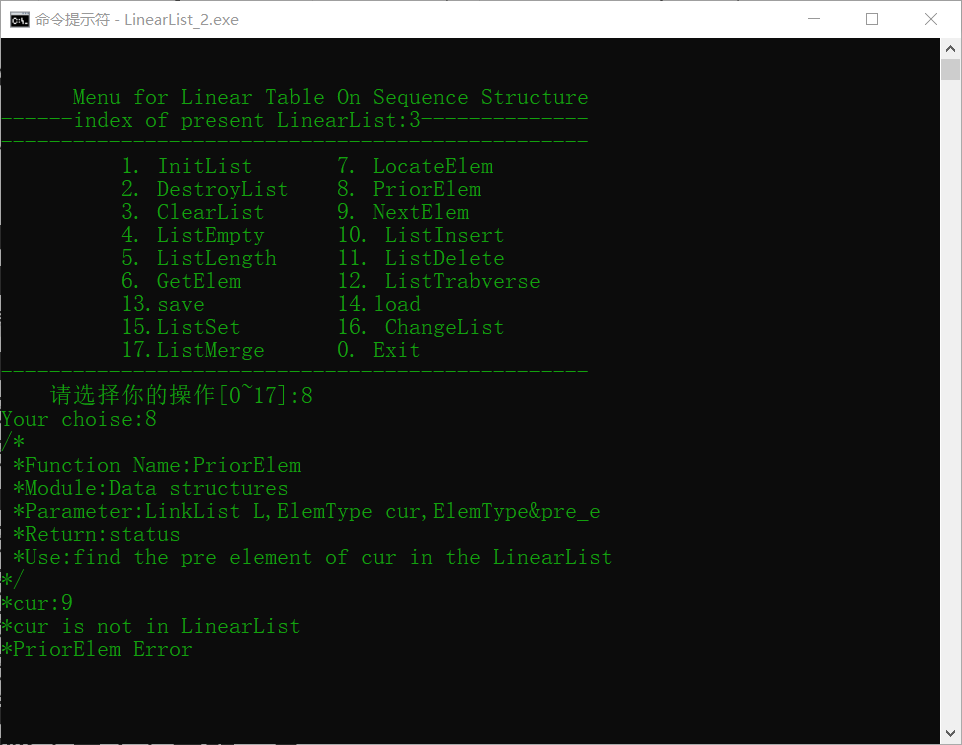
1. **获得表中不存在的元素**

****

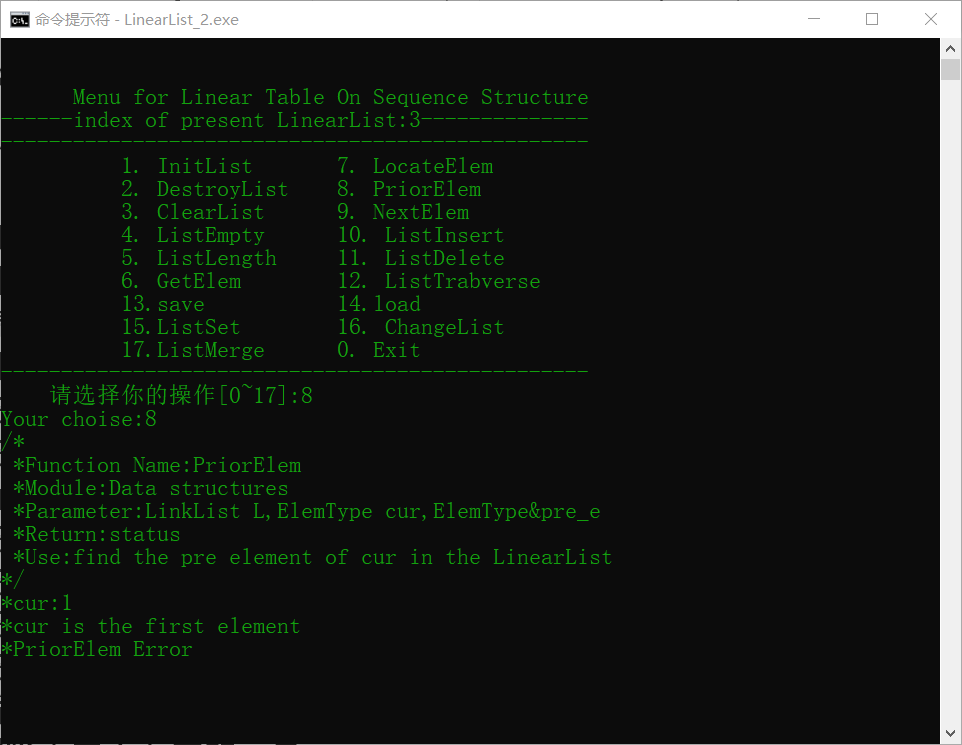
1. **确定表中不存在的元素的位置**

****

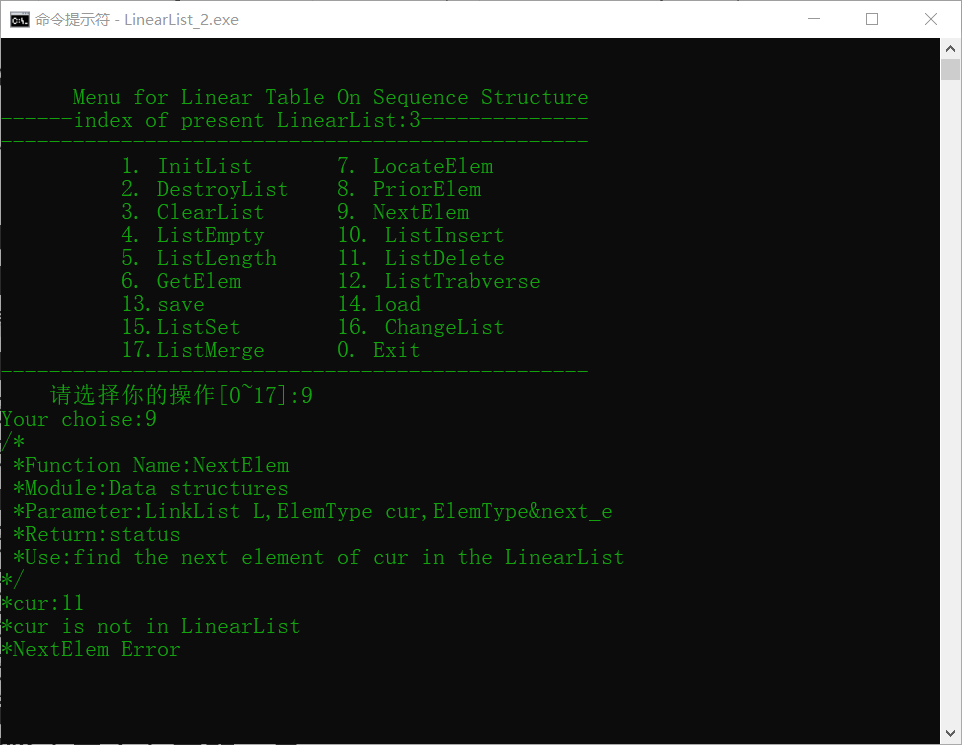
1. **获得表中不存在的元素的前元素**

****

1. **获得表中第一个元素的前元素**

****

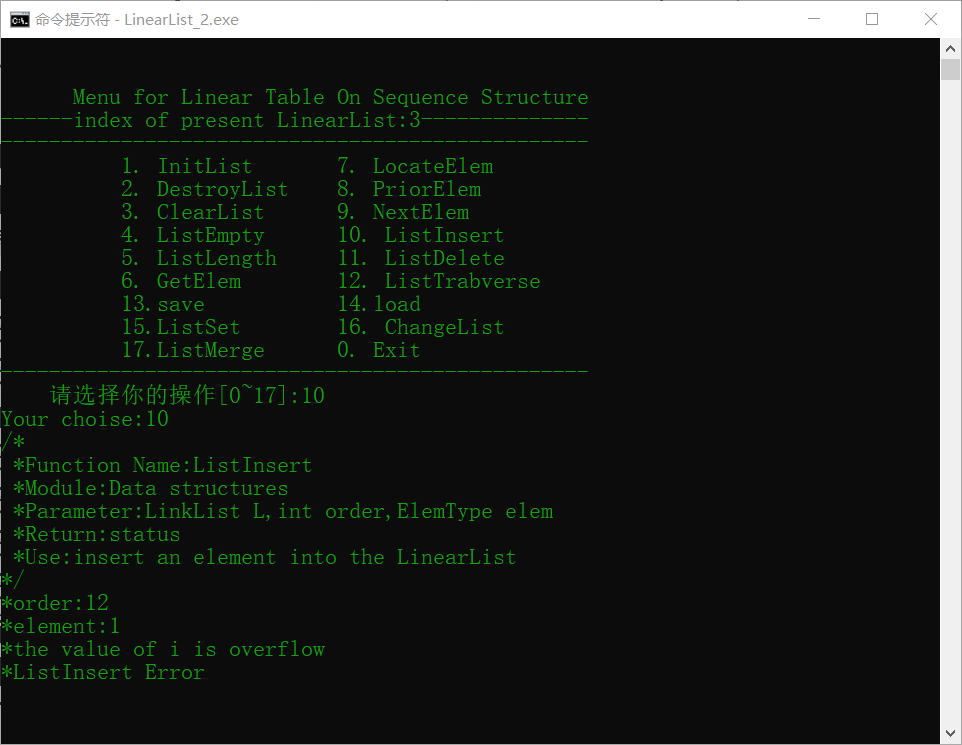
1. **获得表中不存在的元素的后元素**

****

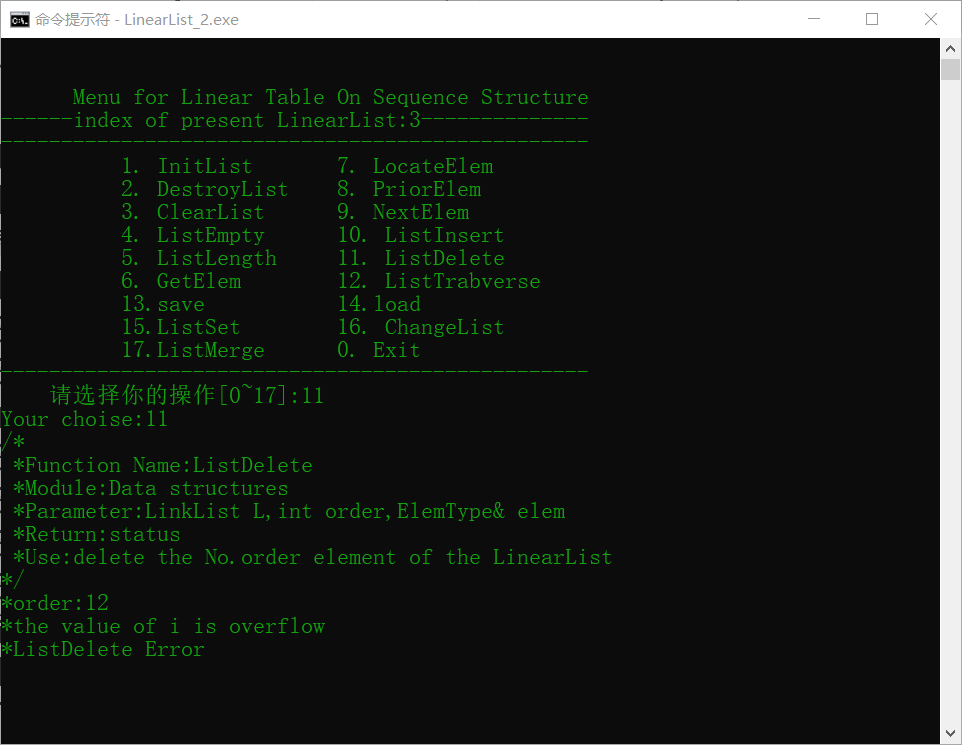
1. **获得表中最后一个元素的后元素**

****

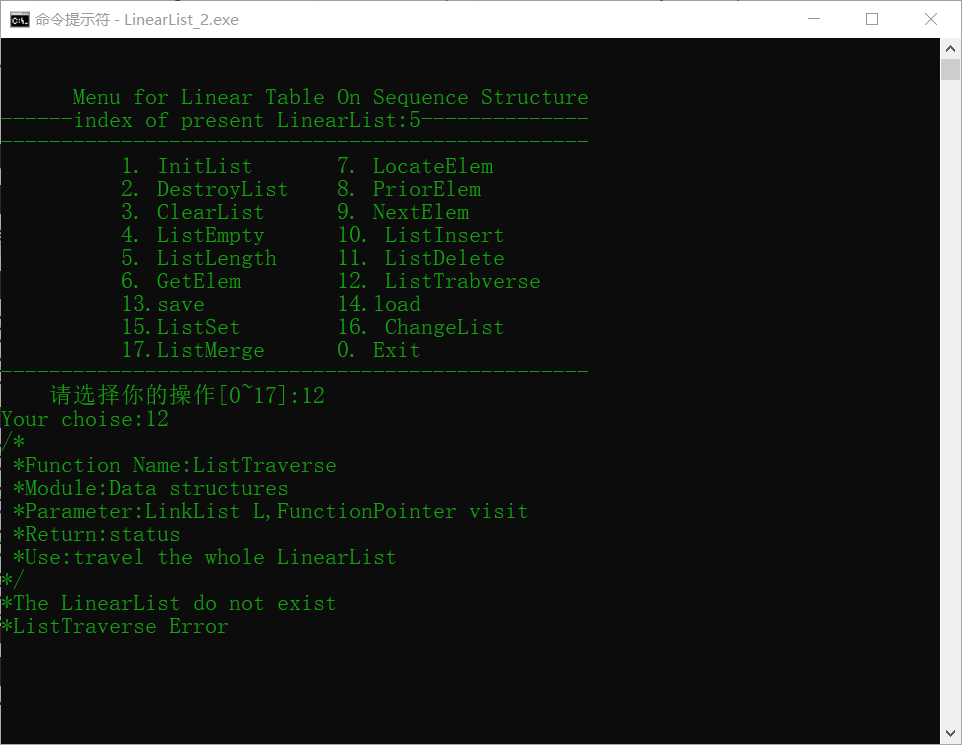
1. **插入元素不合法**

****

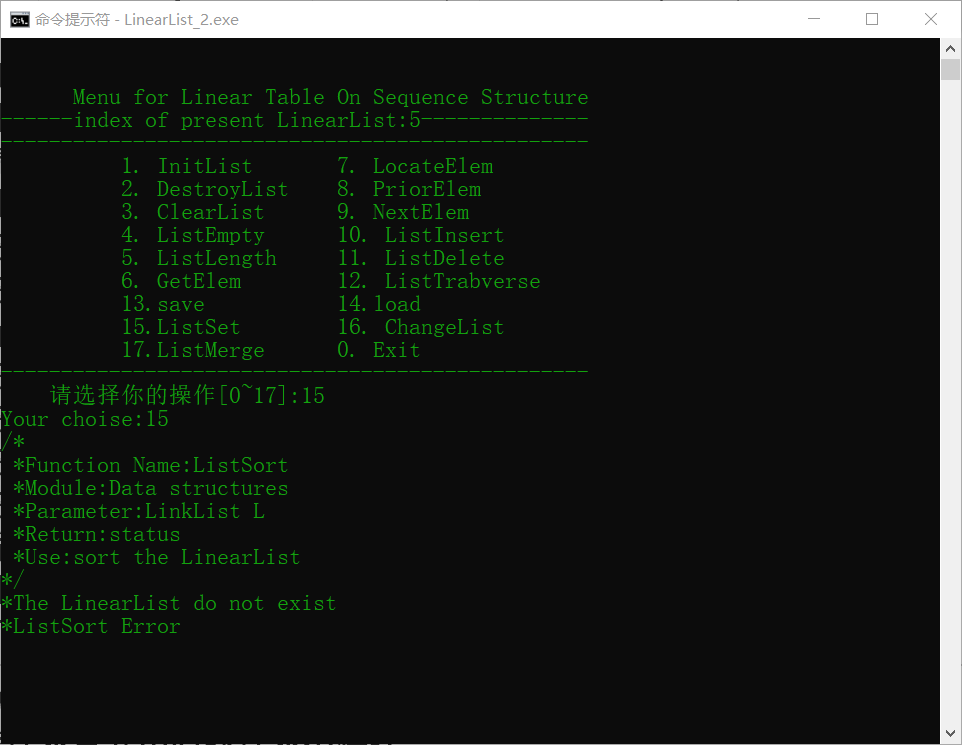
1. **删除表中不存在的元素**

****

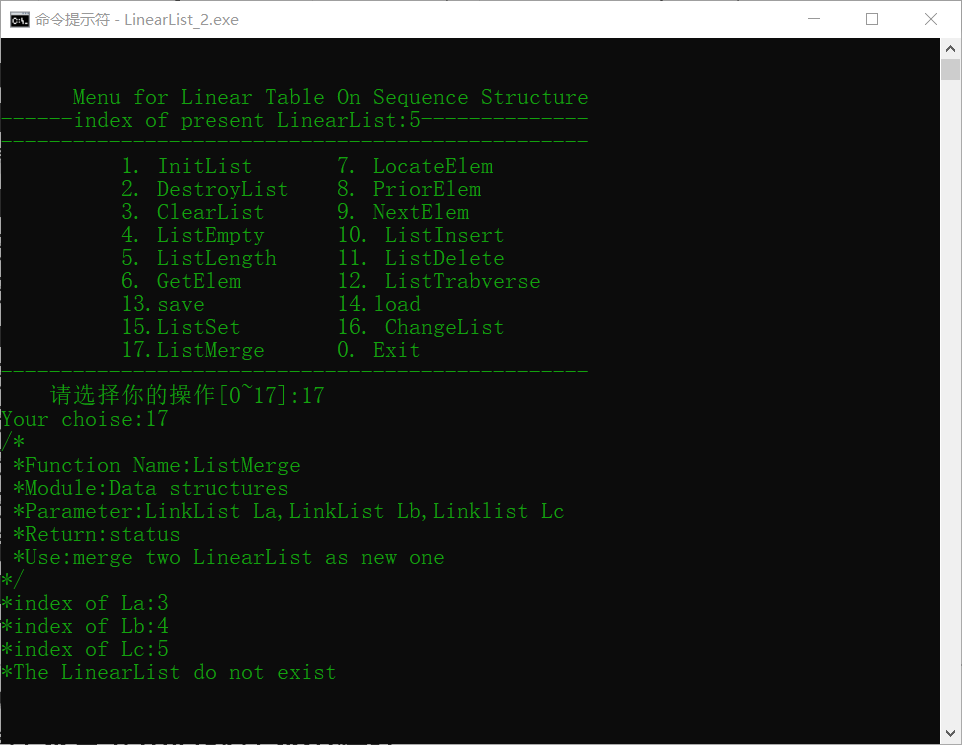
1. **打印不存在的线性表**

****

1. **为不存在的线性表排序**

****

1. **线性表合并中有线性表不存在**

****

## 实验小结

这次的实验内容是基于上次实验的改进，改为链式存储结构，虽然难度上比第一次实验有所提升，但是由于链式存储结构在日常开发中很常用，队列，树我大多都用链式存储实现，所以我对这次的实验内容还是很熟悉的。

本系统完整地实现了课程要求的全部功能，实现了多线性表管理和文件存储功能，系统健壮性良好，可以输出各种情况下的错误提示，并且在整个实验过程中避免了内存泄漏，甚至还加上附加功能排序和合并线性表，较好的达到了预期效果。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

## 3.2 系统设计

## 3.3 系统实现

## 3.4 系统测试

## 3.5 实验小结

# 4 基于邻接表的图实现

## 4.1 问题描述

## 4.2 系统设计

## 4.3 系统实现

## 4.4 系统测试

## 4.5 实验小结

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

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

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

status ListInsert(SqListP L,int i,ElemType e)

{

//在顺序线性表L中第i个位置之前插入新的元素e

if(L->elem==NULL)//判断L是否存在

{

printf("线性表不存在\n");

return ERROR;

}

//i的合法值为1<=i<=L->length+1

if(i<1||i>L->length+1)//i值不合法

{

printf("OVERFLOW");

return ERROR;

}

if(L->length>=L->listsize)

{

//当前存储空间已经满，增加分配

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

if(!newbase)exit(OVERFLOW);//分配内存失败

L->elem=newbase;//新基址

L->listsize+=LISTINCREMENT;//增加存储容量

}

ElemType\* q=&(L->elem[i-1]);//q为插入位置

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

\*(p+1)=\*p;//插入位置及之后的元素右移

\*q=e;//插入e

++L->length;//表长增加1

return OK;

}//ListInsert

# status ListTrabverse(SqListP L,void(\*visit)(ElemType\*e))

# {

# //依次对L的每个数据元素调用函数visit()

# if(L->elem==NULL)//判断L是否存在

# {

# printf("线性表不存在\n");

# return ERROR;

# }

# int i=0;//从表头开始遍历

# while(i<(L->length))

# {

# visit(&(L->elem[i]));//对每个数据元素调用visit

# i++;

# }

# printf("\n");

# return OK;

# }

//写文件

printf("input file name:");

scanf("%s",writefilename);

if((fp=fopen(writefilename,"wb"))==NULL)

{

printf("File open erroe\n ");

getchar();

getchar();

break;

}

fwrite(L->elem,sizeof(ElemType),L->length,fp);

fclose(fp);

Printf("文件写入成功");

//读文件

printf("input file name:");

scanf("%s",readfilename);

L->length=0;

if((fp=fopen(readfilename,"rb"))==NULL)

{

printf("File open error\n ");

getchar();

getchar();

break;

}

while(fread(&L->elem[L->length],sizeof(ElemType),1,fp))

L->length++;

fclose(fp);

printf("文件读取成功");

# 附录B 基于链式存储结构线性表实现的源程序

# 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