

**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201903**

**学 号： U201914995**

**姓 名： 罗虞阳**

**指导教师：许贵平、刘启灏、袁凌**

**报告日期： 2020年 10月 9 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

采用顺序表作为线性表的物理结构，并且呈现一个多级菜单的功能演示系统，该演示系统可以操作单线性表和实现多线性表的管理。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的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。演示系统可选择实现线性表的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存线性表数据逻辑结构(D,{R})的完整信息；②需要设计线性表文件保存和加载操作合理模式。

演示系统可选择实现多个线性表管理。

演示系统的源程序应按照代码规范增加注释和排版，目标程序务必是可以独立于IDE运行的EXE文件。

资料至少包括实验报告、实验源程序和实验目标程序。根据需要还可以增加测试用例文件等等。

## 1.1.1 需要实现的功能

需要实现的功能包括：

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()。

⒀线性表读写文件：函数名称是SaveList(SqList L, char FileName[])，初始条件是线性表L已存在；操作结果是将线性表中的每一个元素存入文件中。函数名称是LoadList(SqList &L,char FileName[]),初始条件是线性表L已存在，操作结果是将文件中的每一个元素存入线性表L中。

⒁多线性表的管理：增加一个新线性表：函数名称是AddList(LISTS &Lists,char ListName[])，初始条件是线性表集Lists已存在；操作结果是在已存在的线性表中新增加一个线性表。

⒂多线性表的管理：移除一个线性表：函数名称是RemoveList(LISTS &Lists,char ListName[])，初始条件是线性表集Lists已存在；操作结果是在已存在的线性表中删除一个名称为ListName的一个线性表。

⒃多线性表的管理：查找一个线性表：函数名称是LocateList(LISTS &Lists,char ListName[])，初始条件是线性表集Lists已存在；操作结果是在已存在的Lists中查找一个名称为ListName的线性表。

## 1.2 系统设计

本系统提供一个顺序储存的线性表。

菜单可供选择的操作有：初始化线性表，销毁线性表，清空线性表，判断空表，求表长，获取元素，查找元素，获得前驱，获得后继，插入元素，删除元素，遍历线性表，线性表读写文件，多线性表的管理：增加一个新的线性表，删除一个线性表，查找一个线性表。具体结构如下图： (如图1-1所示)

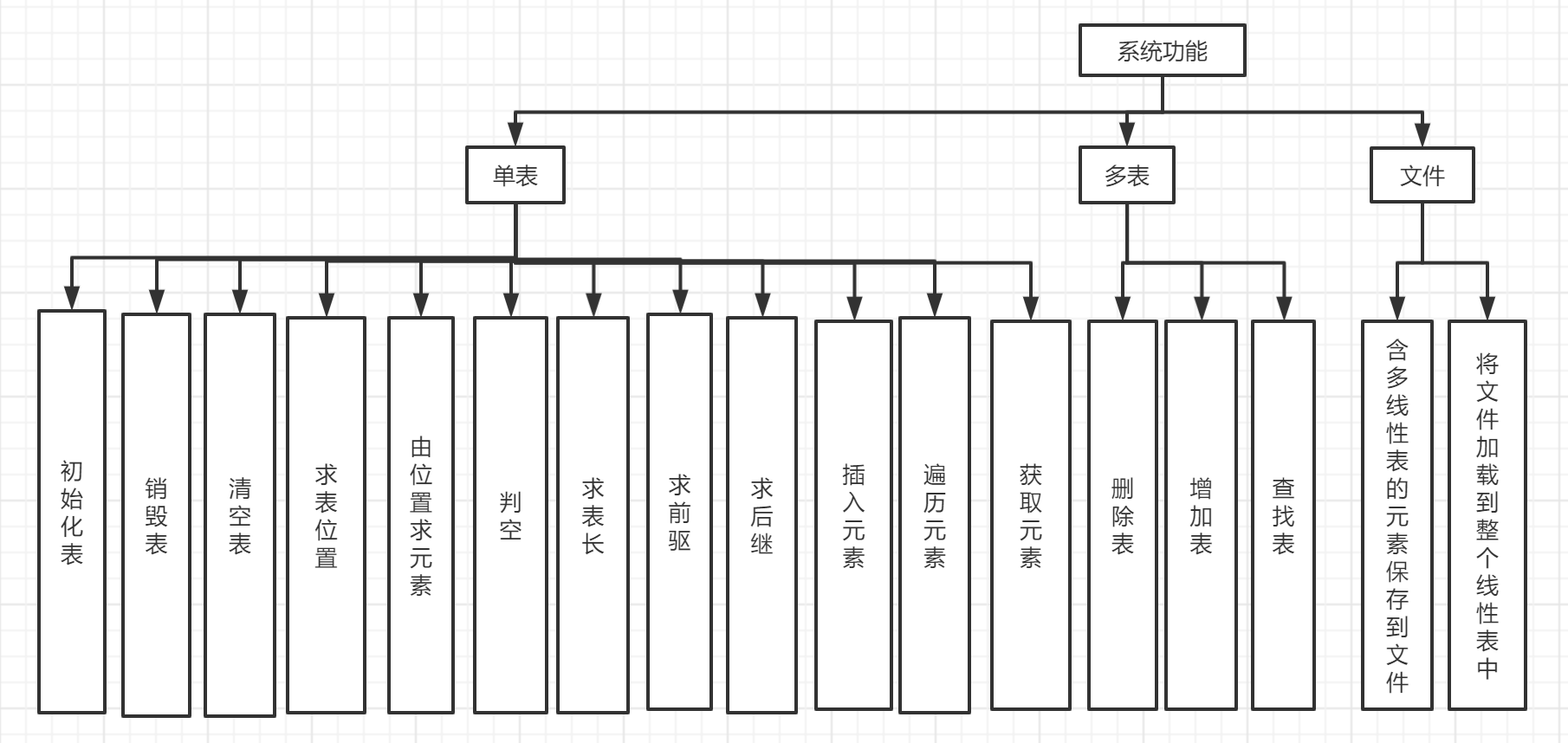


图1-1 系统总体设计结构

### 1.2.1 有关常量和类型定义

1. #define TRUE 1
2. #define FALSE 0
3. #define OK 1
4. #define ERROR 0
5. #define INFEASIBLE -1
6. #define OVERFLOW -2
7. typedef int status;
8. typedef int ElemType; //数据元素类型定义
9. #define LIST\_INIT\_SIZE 100
10. #define LISTINCREMENT 10
11. typedef int ElemType；
12. typedef struct{ //顺序表（顺序结构）的定义
13. ElemType \* elem;
14. int length;
15. int listsize;
16. }SqList;
17. typedef struct{ //线性表的集合类型定义
18. struct { char name[30];
19. SqList L;
20. } elem[10];
21. int length;
22. }LISTS;
23. LISTS Lists; //线性表集合的定义Lists

### 1.2.2 函数设计

（1）

函数名称：InitList（SqList &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；

算法思路：先分配储存空间后，将表长设置为0，再见线性表容量设为预定义的初始储存容量。

（2）

函数名称：DestroyList（SqList &L）；

初始条件：线性表L已存在；

操作结果：销毁线性表L；

算法思路：释放内存并且将个数据设置为初始值。

（3）

函数名称：ClearList（SqList &L）；

初始条件：线性表L已存在；

操作结果：将线性表L重置为空表；

算法思路：将表长设置为0。

（4）

函数名称：ListEmpty（SqList L）；

初始条件：线性表L存在；

操作结果：若L为空表则返回TRUE，否则返回FALSE；

算法思路：判断表长为0则为空表，否则不是空表。

（5）

函数名称：ListLength（SqList L）；

初始条件：线性表L存在；

操作结果：返回线性表L中元素的个数；

算法思路：返回线性表L中表长的值。

流程图：

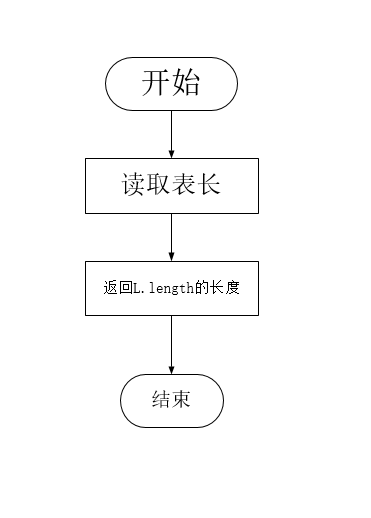


图1-2 ListLength（SqList L）流程图

（6）

函数名称：GetElem（SqList L,int I,ElemTyoe &e）；

初始条件：线性表L存在；

操作结果：返回线性表中第i个元素并将值赋给e；

算法思路：将线性表中第i个数据元素的值赋值给e。

流程图：

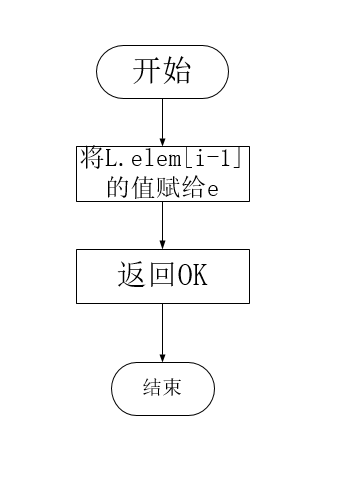


图1-3 GetElem（L,I,e）流程图

（7）

函数名称：LocateElem（L，e）

初始条件：线性表已存在

操作结果：返回线性表中第1个与e相等的数据元素的位置，若这样的数据元素不存在，则返回值为0；

算法思路：先遍历，将线性表中的数据元素依次与e进行比较，返回该元素的位序。

（8）

函数名称：PriorElem（L，e，pre）；

初始条件：线性表L已存在；

操作结果：若e是L的数据元素并且不是第一个数据元素，用pre返回它的前驱，否则操作失败。

算法思路：先遍历该线性表。若找到该结点，并且该结点有前驱元素，则将前驱元素赋值给pre。若未找到该结点，或者找到该结点但该结点不存在前驱元素，则返回FALSE。

流程图：如图所示：

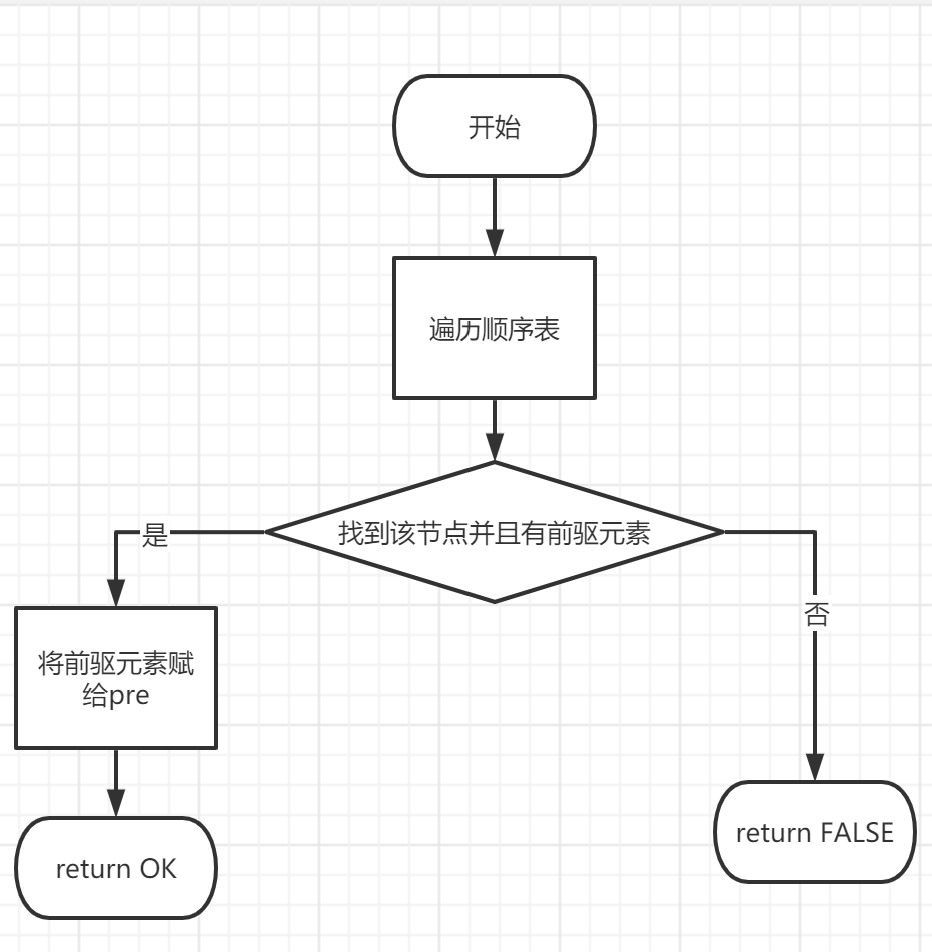


图1-4 PriorElem（L，e，pre）流程图

（9）

函数名称：NextElem（L，e，next）；

初始条件：线性表已存在；

操作结果：若e是L的数据元素，且不是最后一个，则用next返回它的后继，否则操作失败，next无定义。

算法思路：先遍历线性表，如果找到该结点并且该结点有后继元素，否则将后继结点元素赋值给next。若未找到该结点，后者找到该结点但该结点不存在后继元素，则返回FALSE。

流程图：

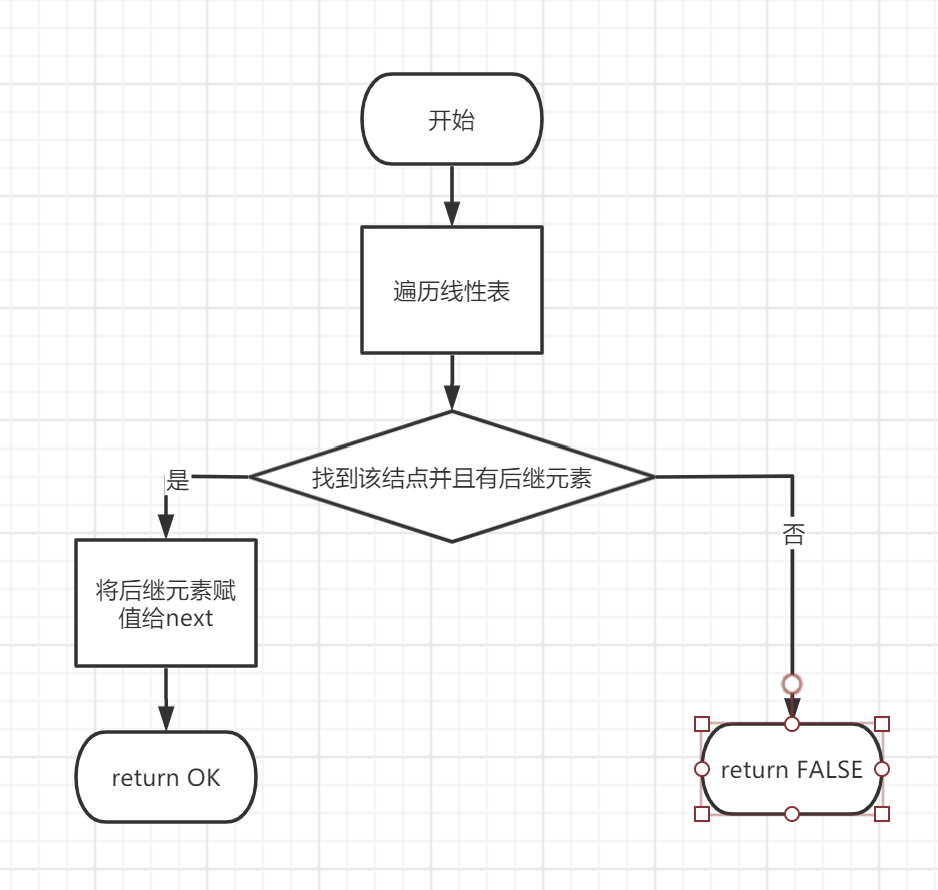


图1-5 NextElem（L，e，pre）流程图

（10）

函数名称：ListInsert（L，i，e）

初始条件：线性表L已存在且非空，1<=i<=ListLength(L)+1；

操作结果：在L的第i个位置之前插入新的数据元素e。

算法思路：先遍历顺序表，若线性表L已存在并且不为空，输入的i值不合法，则返回ERROR。若满足线性表L已存在且L非空，并且i的值合法，则在线性表的第i个位置之前插入新的数据元素e，返回OK。

流程图：



图1-6 ListInsert（L，i，e）流程图

（11）

函数名称：ListDelete（L，i,e）；

初始条件：线性表已存在且非空，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；

算法思路：先遍历顺序表，如果线性表L已存在且非空，并且输入的i值不合法，则返回ERROR。若满足线性表L已存在并且L非空，并且i的值合法，则删除线性表的第i个数据元素，并用e返回其值，返回OK。

流程图：

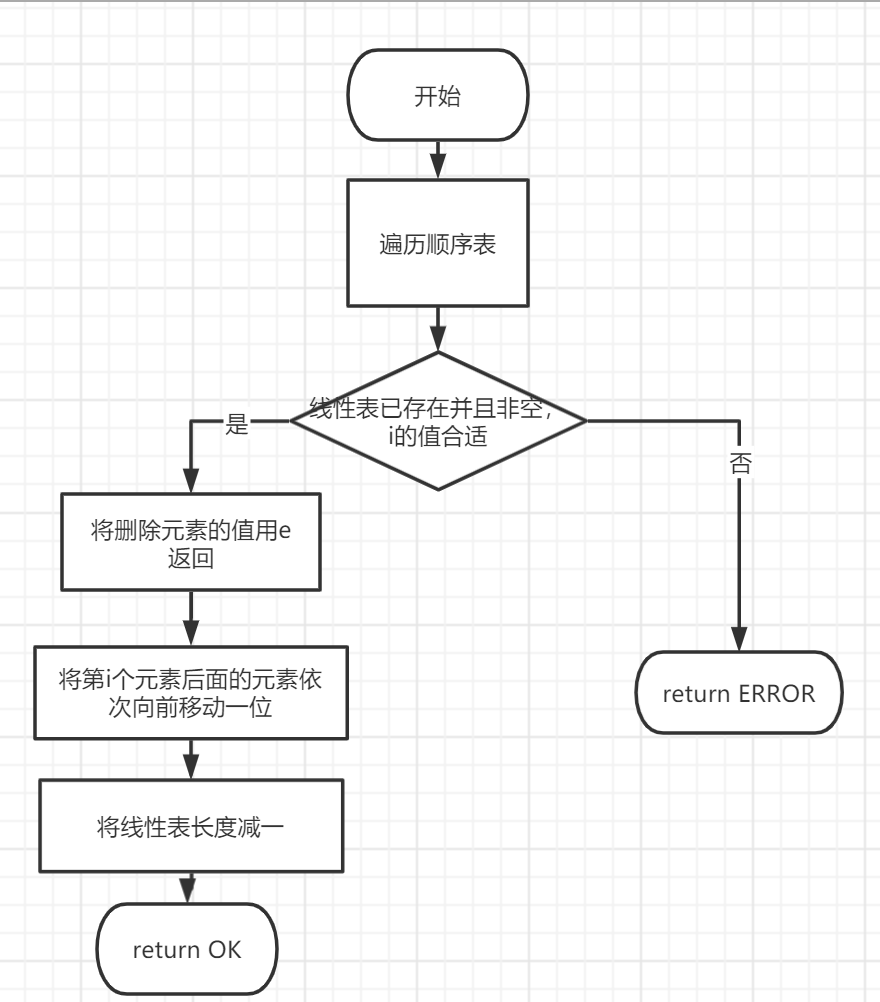


图1-7 ListDelete（L，i,e）流程图

（12）

函数名称：ListTraverse（L）；

初始条件：线性表存在；

操作结果：依次遍历L的每个数据元素并且输出；

算法思路：若线性表L存在，则遍历元素；否则返回ERROR。

流程图：

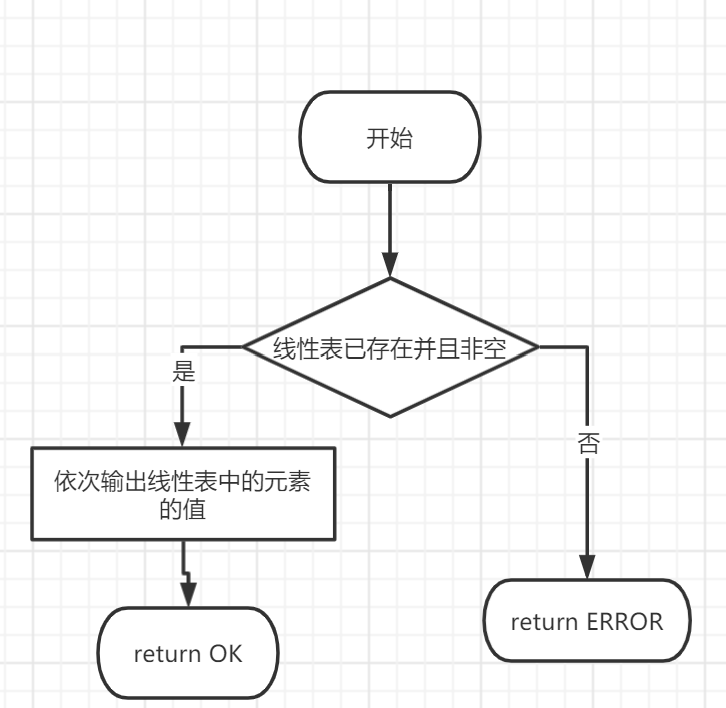


图1-8 ListTraverse（L）流程图

（13）

函数名称：SaveList（L，FileName）；

初始条件：线性表存在；

操作结果：将线性表L保存为文件形式；

算法思路：用fwrite保存为文件。

（14）

函数名称：LoadList（L，FileName）；

初始条件：线性表存在；

操作结果：将线性表L以文件形式加载读取；

算法思路：用fread将文件读取线性表。

流程图：

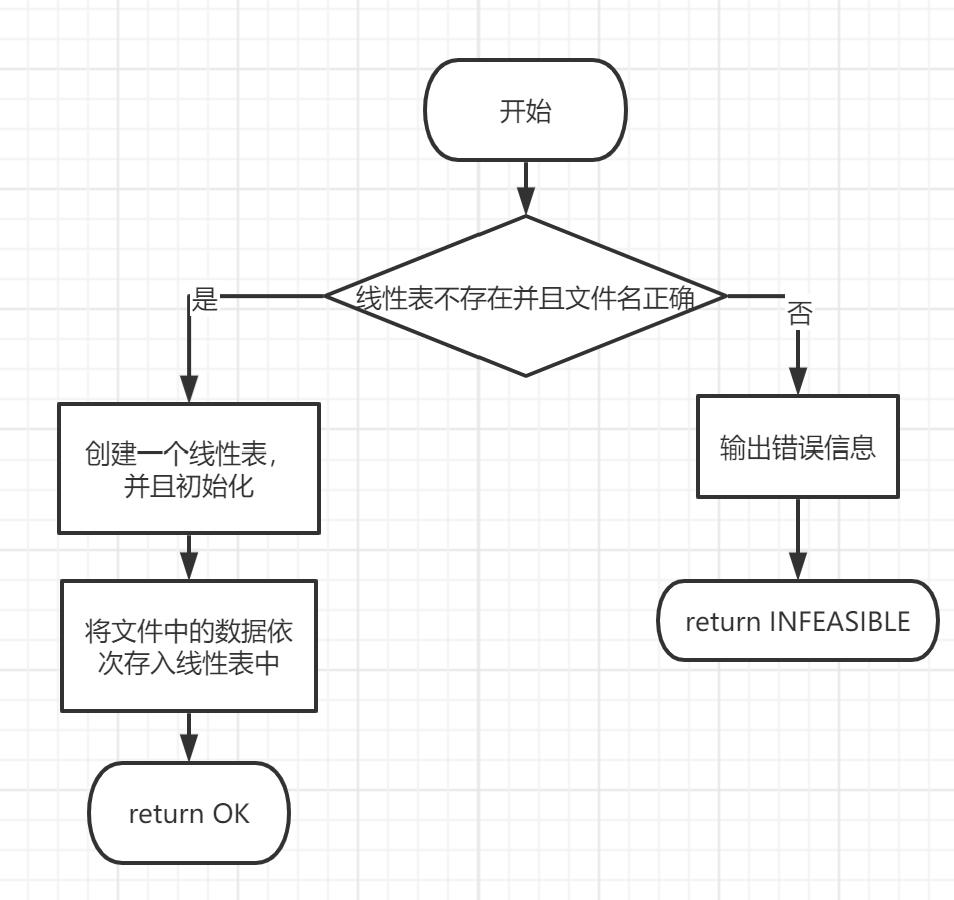


图1-9 LoadList（L，FileName）流程图

（15）

函数名称：AddList（Lists，ListName）；

初始条件：线性表集存在；

操作结果：在Lists中增加一个名称为ListName的空线性表；

算法思路：将ListName中的字符串复制到线性表的数据元素的名字数组里，然后初始化新的线性表。

（16）

函数名称：RemoveList（Lists，ListName）；

初始条件：线性表集存在；

操作结果：在Lists中删除一个名称为ListName的空线性表；

算法思路：遍历Lists中的线性表，将每个线性表的名字与ListName相比较，如果相同则将后面的线性表向前移动覆盖掉要删除的表。如果没找到则返回ERROR。

流程图：

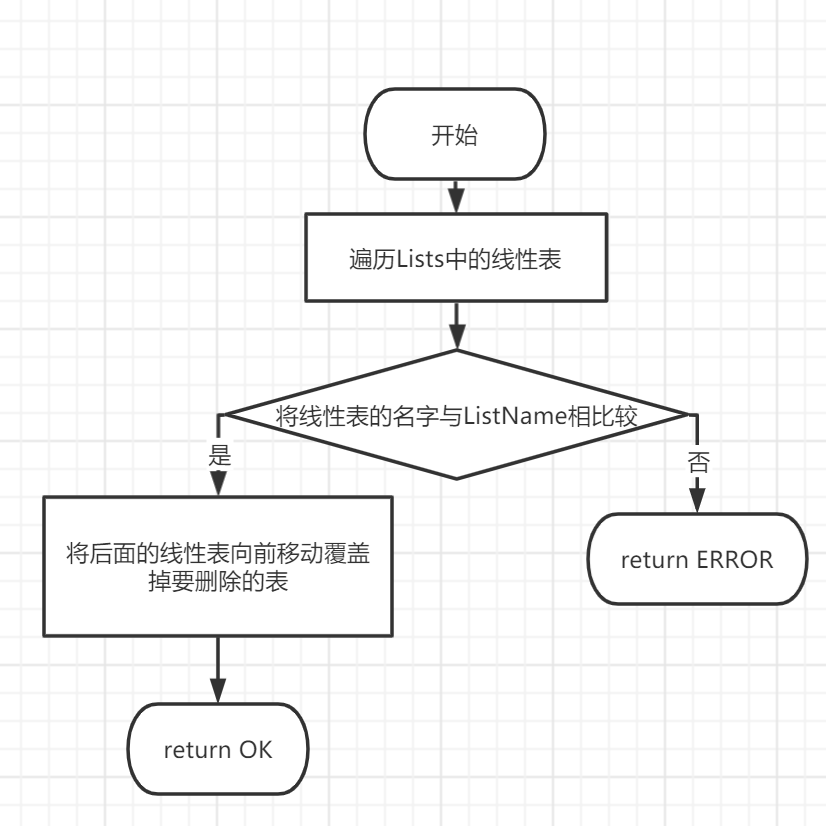


图1-10 RemoveList（Lists，ListName）流程图

（17）

函数名称：LocateList（Lists，ListName）；

初始条件：线性表集存在；

操作结果：在Lists中查找一个名称为ListName的线性表；

算法思路：遍历Lists中的线性表，如果第i个线性表的名字与ListName相同，则返回线性表在线性表集中的序号，否则返回ERROR。

流程图：

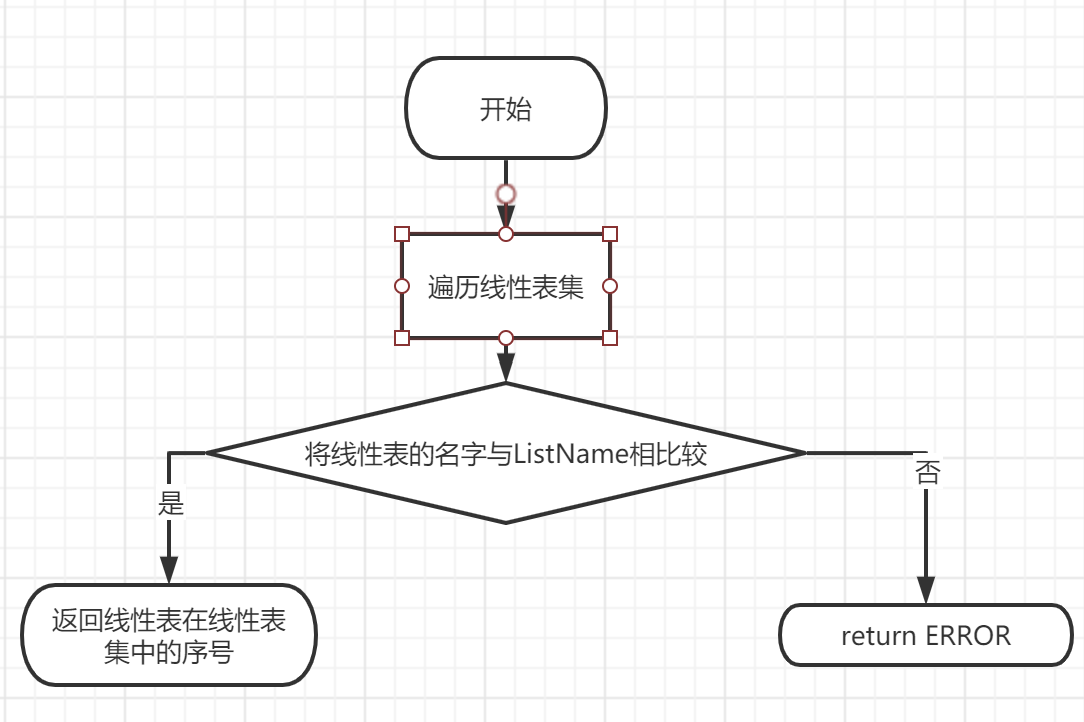


图1-11 LocateList（Lists，ListName）流程图

### 1.2.3 函数的时间和空间复杂度

InitList（SqList &L）:T(n)=O(1),S(n)=O(n)

DestroyList（SqList &L）: T(n)=O(1),S(n)=O(1)

ClearList（SqList &L）: T(n)=O(1),S(n)=O(n)

ListEmpty（SqList L）: T(n)=O(1),S(n)=O(n)

ListLength（SqList L）: T(n)=O(1),S(n)=O(n)

GetElem（SqList L,int I,ElemTyoe &e）: T(n)=O(1),S(n)=O(n)

LocateElem（L，e）: T(n)=O(n),S(n)=O(n)

PriorElem（L，e，pre）: T(n)=O(n),S(n)=O(n)

NextElem（L，e，next）: T(n)=O(n),S(n)=O(n)

ListInsert（L，i，e）: T(n)=O(n),S(n)=O(n)

ListDelete（L，i,e）: T(n)=O(n),S(n)=O(n)

ListTraverse（L）: T(n)=O(n),S(n)=O(n)

SaveList（L，FileName）: T(n)=O(n),S(n)=O(n)

LoadList（L，FileName）: T(n)=O(n),S(n)=O(n)

AddList（Lists，ListName）: T(n)=O(n),S(n)=O(n)

RemoveList（Lists，ListName）: T(n)=O(n),S(n)=O(n)

LocateList（Lists，ListName）: T(n)=O(n),S(n)=O(n)

## 1.3 系统实现

### 1.3.1 程序源代码

见《附录A 基于顺序存储结构线性表实现的源程序》

### 1.3.2 程序实现的环境，开发工具和开发语言

程序实现环境：Win10-x64

开发工具：Dev c++

开发语言：类C语言

### 1.3.3 头文件及预定义说明

头文件包括：预定义文件def.h和预定义函数文件defin.h。

预定义说明：

LIST\_INIT\_SIZE:线性表初始分配空间大小；

LISTINCREMENT:单空间不够时，线性表补充分配空间大小；

TRUE, FALSE, OK, ERROR, INFEASIBLE, OVERFLOW分别为状态产量，对应返回为真值，返回为假，程序运行正常，程序运行错误，程序运行不合法和程序溢出。

SqList为线性表结构定义类型，包括指向分配空间首地址的指针\*elem，表示线性表长度的length，和线性表最大尺度的listsize。

LISTS为多线性表结构定义类型，包括结构数组elem[10],长度length和最大尺寸listsize。

### 1.3.4 代码组织结构

本实验代码结构可分为三部分：主函数”顺序结构顺序表.cpp”,预定义常量“def.h”,和函数文件“defin.h”。预定义常量和函数文件不做赘述，具体可再前文中查找。

主函数主要采用了while循环和switch选择，通过用户输入来确定所要调用的函数。在switch下，还嵌套了一个while-switch循环用以对多线性表进行操作，从而实现多级菜单的功能。仍是通过读取用户输入判断要调用的函数。在嵌套结构下，可退回上一层，对选中的线性表进行操作。

**1.4 系统测试**

本程序采用简易操作界面，如图1-7所示，挑选ListLength，GetElem，PriorElem，LoadList，LocateList进行测试。



图2-1 程序初始界面

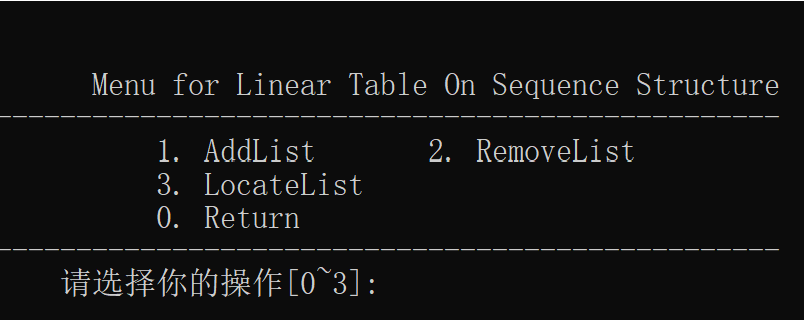


图2-2 多线性表界面

单线性表测试用例为：L{1，2，3，4，5}，空表，不存在线性表；多线性表测试用例为Lists{{‘jia’：1，2，3，4，5}，{‘ja’：1，3，5，7，9}，{‘j’}}；文件内容为{1，2，3，4，5}。

单线性表：

（1）ListLength测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| L | 界面选择5 | 线性表的长度为5 | 线性表的长度为5 |
| 空表 | 界面选择5 | 线性表的长度为0 | 线性表的长度为0 |
| 不存在线性表 | 界面选择5 | 线性表不存在 | 线性表不存在 |

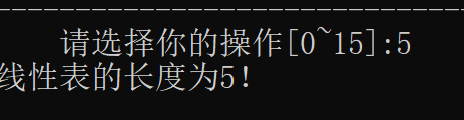


图2-3 ListLength测试1

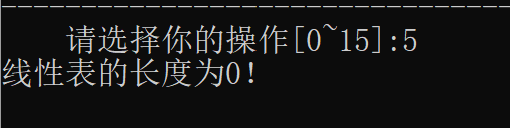


图2-4 ListLength测试2

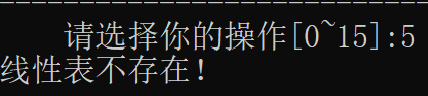


图2-5 ListLength测试3

（2）GetElem测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| L | 界面选择6，输入1， | 线性表的第1位是1 | 线性表的第1位是1 |
| 空表 | 界面选择6，输入1 | 输入的位置不合法，请重新输入 | 输入的位置不合法，请重新输入 |
| 不存在线性表 | 界面选择6，输入1 | 线性表不存在 | 线性表不存在 |

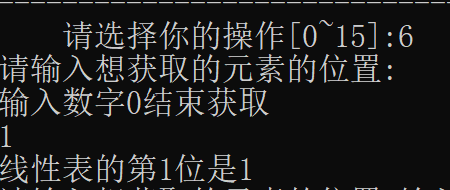


图2-6 GetElem测试1

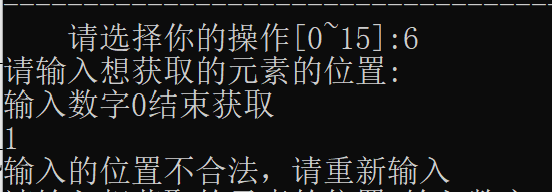


图2-7 GetElem测试2

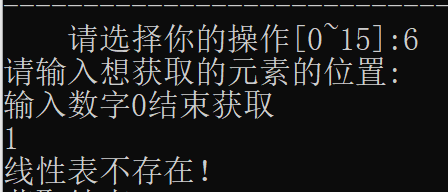


图2-8 GetElem测试3

（3）PriorElem测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| L | 界面选择8，输入4 | 元素4的前驱是3 | 元素4的前驱是3 |
| 空表 | 界面选择8，输入2 | 元素2没有前驱元素 | 元素2没有前驱元素 |
| 不存在线性表 | 界面选择8，输入2 | 线性表不存在 | 线性表不存在 |

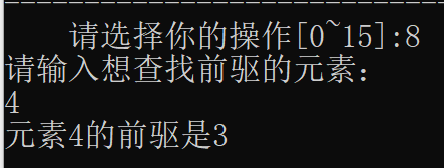


图2-9 PriorElem测试1

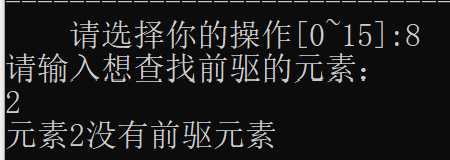


图2-10 PriorElem测试2

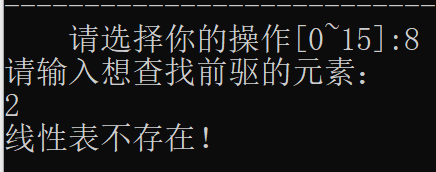


图2-11 PriorElem测试3

（4）LoadList测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| L | 界面选择14 | 读入失败，线性表已存在 | 读入失败，线性表已存在 |
| 空表 | 界面选择14，界面选择12 | 读入失败，线性表已存在 | 读入失败，线性表已存在 |
| 不存在线性表 | 界面选择14，姐买你选择12 | 1 2 3 4 5输出成功 | 1 2 3 4 5输出成功 |

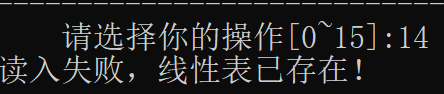


图2-12 LoadList测试1

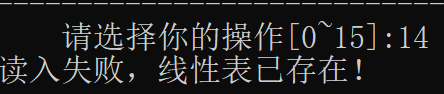
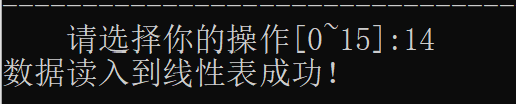


图2-13 LoadList测试2



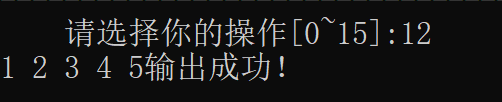


图2-14 LoadList测试3

多线性表：

（5）LocateList测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| Lists | 界面选择15，界面选择3，输入jia | 查找的线性表如下：jia 1 2 3 4 5 | 查找的线性表如下：jia 1 2 3 4 5 |
| 空表 | 界面选择15，界面选择3，输入j | 查找的线性表如下：j | 查找的线性表如下：j |
| 不存在线性表 | 界面选择15，界面选择3，输入abc | 查找失败 | 查找失败 |

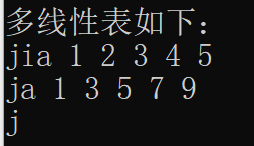


图2-15 LocateList测试集展示

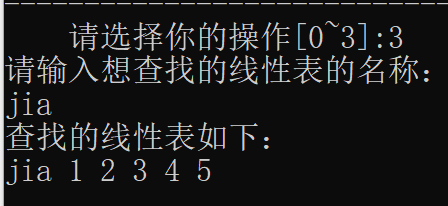


图2-16 LocateList测试1

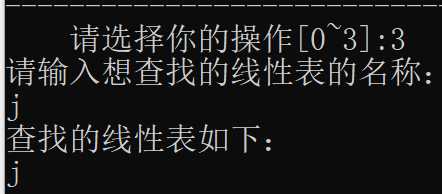


图2-17 LocateList测试2

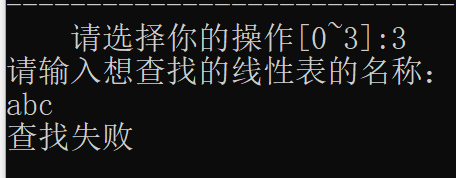


图2-18 LocateList测试3

## 1.5 实验小结

这次的实验主要是针对线性表的练习，因为老师已经在任务书中给出了线下集成系统的大体框架，我们只需要往里面填充函数以及增加一些细节，所以任务不是很重，也体会到了老师的良苦用心。

在这次的数据结构实验中，我大致掌握了如何实现单线性表的基本操作以及对多线性表的添加、删除、和查找操作。对线性表这一数据结构有了更深的了解和理解。

同时在编写程序时也遇到了一些问题，比如如何实现对多线性表中单个线性表进行操作。因为如果直接赋值给另一个结构，则修改过后不能返回给多线性表中的原表，后面想到先做判断，如果是从多线性表中进入的，则在执行完操作后再对原表进行修改。该有一种方式是用指针，但是前面的函数和程序都要进行改动，工作量会很大；还有就是如何实现多级菜单，以及找函数中的bug和解决。特别是如果涉及到函数调用函数，找起来就会比较麻烦。

这次的实验线上部分比较容易，线下组装会比较花费时间，还有就是对组装好的系统进行测试和修改，以匹配老师的要求，多级菜单，对多线性表内的表进行操作等。

这次的实验总的来说还比较顺利，加深了对线性表的掌握，同时也对数据结构这门课程有了更全面的认识。

# 

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

### 2.1.1 12个基本函数需求实现

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,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，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()。

### 2.1.2 演示系统与数据文件及多表管理

采用单链表作为线性表的物理结构，实现§2.2的基本运算。其中ElemType为数据元素的类型名，具体含义可自行定义，其它有关类型和常量的定义和引用详见文献[1]的p10。

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。

演示系统可选择实现线性表的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存线性表数据逻辑结构(D,{R})的完整信息；②需要设计线性表文件保存和加载操作合理模式。

演示系统可选择实现多个线性表管理。目标程序务必是可以独立于IDE运行的EXE文件。

## 2.2 系统设计

### 2.2.1 系统总体设计

本系统提供一个顺序储存的线性表。

菜单可供选择的操作有：初始化线性表，销毁线性表，清空线性表，判断空表，求表长，获取元素，查找元素，获得前驱，获得后继，插入元素，删除元素，遍历线性表，线性表读写文件，多线性表的管理：增加一个新的线性表，删除一个线性表，查找一个线性表，显示多线性表。具体结构如下图： (如图1-1所示)

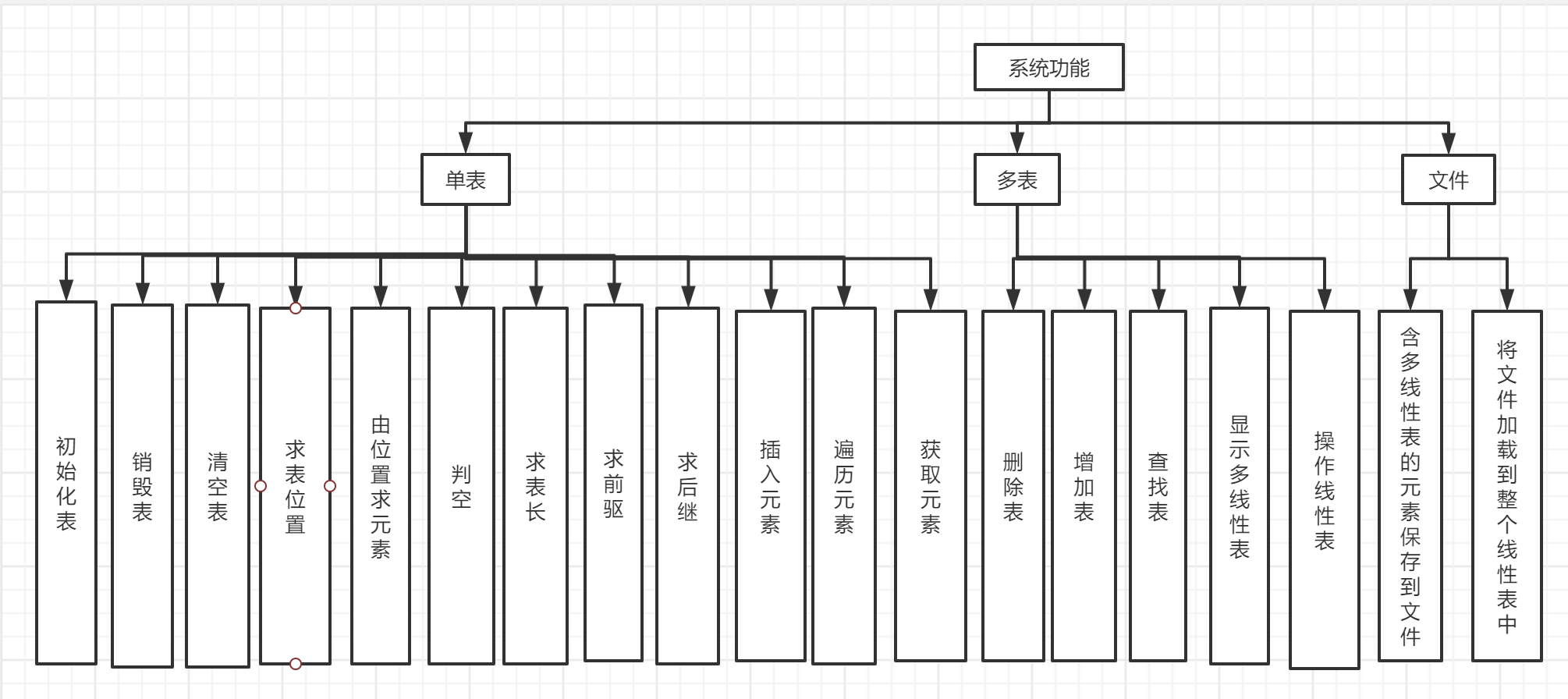


图1-1 系统总体设计

### 2.2.2 算法设计

（1）

函数名称：InitList(LinkList &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；

算法思路：先分配储存空间后，将表长设置为0，再见线性表容量设为预定义的初始储存容量；

时间和空间复杂度：T(n)=O(1),S(n)=O(1)

（2）

函数名称：DestroyList(LinkList &L)

初始条件：线性表L已存在；

操作结果：销毁线性表L；

算法思路：释放内存并且将个数据设置为初始值。

时间和空间复杂度：T(n)=O(1),S(n)=O(1)

（3）

函数名称：ClearList(LinkList &L)

初始条件：线性表L已存在；

操作结果：将线性表L重置为空表；

算法思路：将表长设置为0。

时间和空间复杂度：T(n)=O(1),S(n)=O(1)

（4）

函数名称：ListEmpty(LinkList L)

初始条件：线性表L存在；

操作结果：若L为空表则返回TRUE，否则返回FALSE；

算法思路：判断表长为0则为空表，否则不是空表。

时间和空间复杂度：T(n)=O(1),S(n)=O(1)

（5）

函数名称：ListLength(LinkList L)

初始条件：线性表L存在；

操作结果：返回线性表L中元素的个数；

算法思路：返回线性表L中表长的值。

时间和空间复杂度：T(n)=O(1),S(n)=O(1)

流程图：

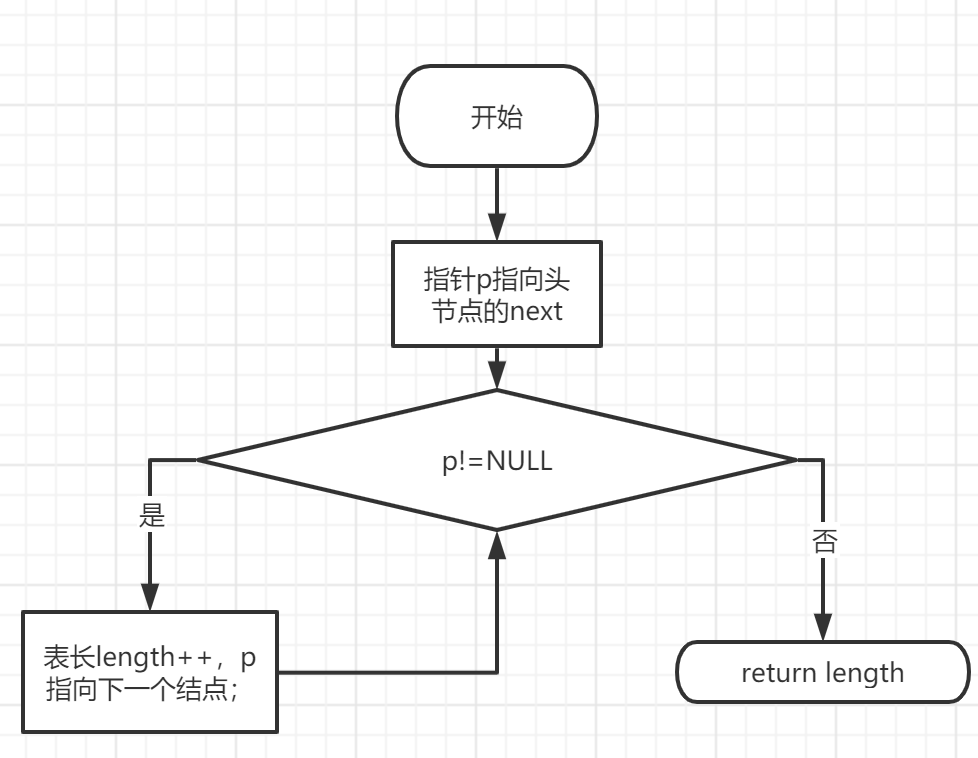


图1-2 ListLength(LinkList L)流程图

（6）

函数名称：GetElem(LinkList L,int i,ElemType &e)

初始条件：线性表L存在；

操作结果：返回线性表中第i个元素并将值赋给e；

算法思路：将线性表中第i个数据元素的值赋值给e。

时间和空间复杂度：T(n)=O(1),S(n)=O(1)

流程图：

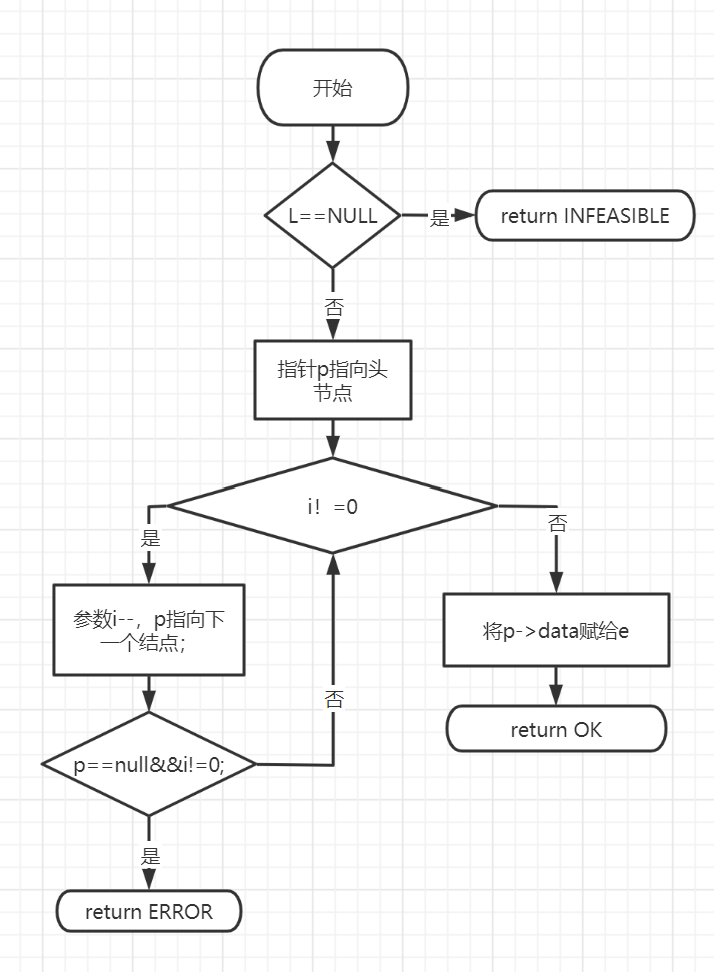


图1-3 GetElem（L,I,e）流程图

（7）

函数名称：LocateElem(LinkList L,ElemType e)

初始条件：线性表已存在

操作结果：返回线性表中第1个与e相等的数据元素的位置，若这样的数据元素不存在，则返回值为0；

算法思路：先遍历，将线性表中的数据元素依次与e进行比较，返回该元素的位序。

时间和空间复杂度：T(n)=O(n),S(n)=O(1)

（8）

函数名称：PriorElem(LinkList L,ElemType e,ElemType &pre)

初始条件：线性表L已存在；

操作结果：若e是L的数据元素并且不是第一个数据元素，用pre返回它的前驱，否则操作失败。

算法思路：先遍历该线性表。若找到该结点，并且该结点有前驱元素，则将前驱元素赋值给pre。若未找到该结点，或者找到该结点但该结点不存在前驱元素，则返回FALSE。

时间和空间复杂度：T(n)=O(n),S(n)=O(1)

流程图：如图所示：

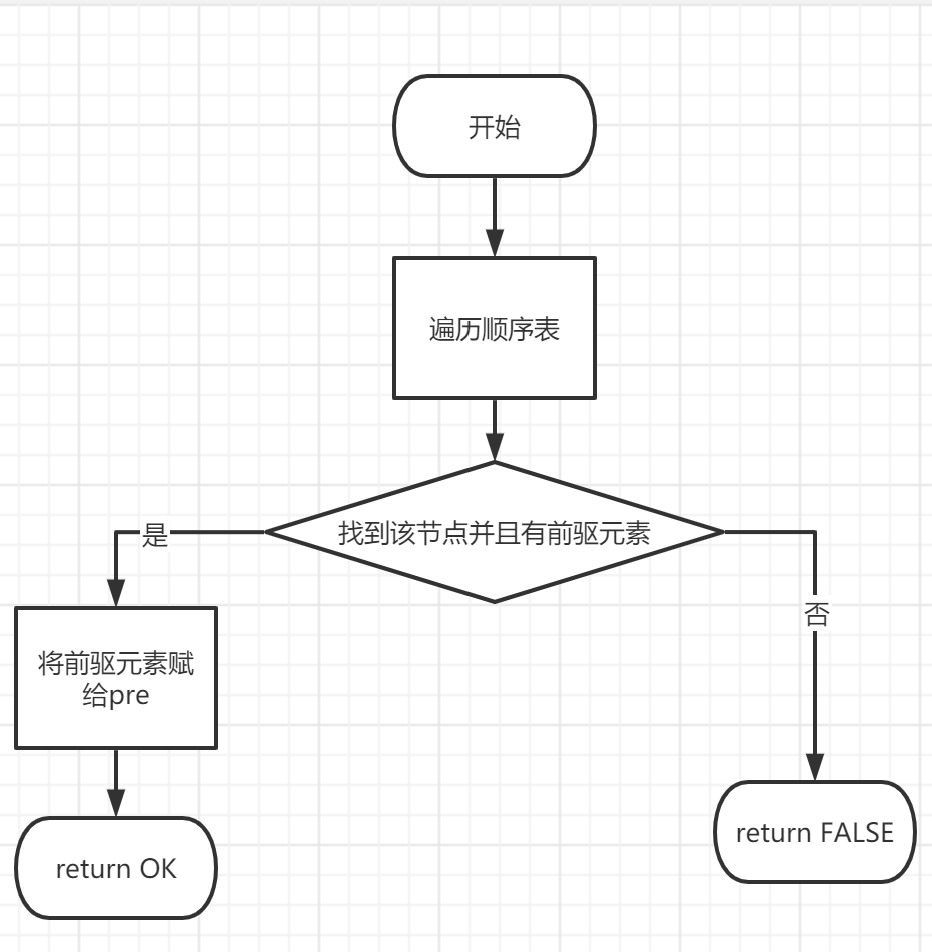


图1-4 PriorElem（L，e，pre）流程图

（9）

函数名称：NextElem(LinkList L,ElemType e,ElemType &next)

初始条件：线性表已存在；

操作结果：若e是L的数据元素，且不是最后一个，则用next返回它的后继，否则操作失败，next无定义。

算法思路：先遍历线性表，如果找到该结点并且该结点有后继元素，否则将后继结点元素赋值给next。若未找到该结点，后者找到该结点但该结点不存在后继元素，则返回FALSE。

时间和空间复杂度：T(n)=O(n),S(n)=O(1)

流程图：

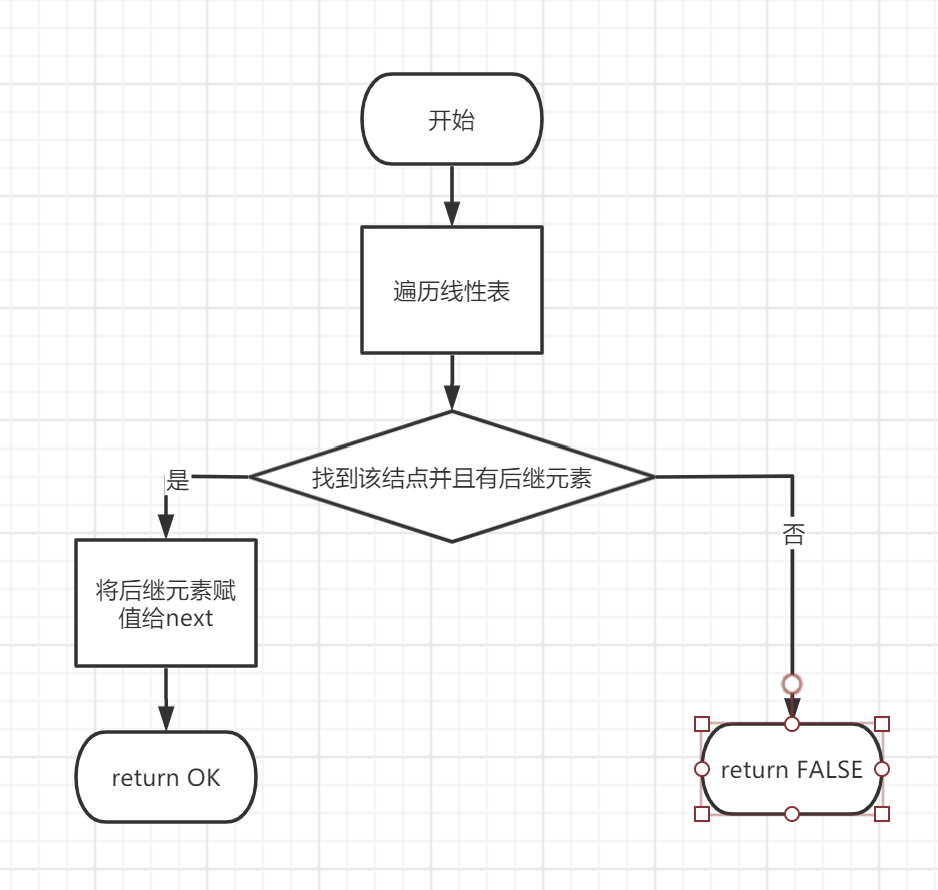


图1-5 NextElem（L，e，pre）流程图

（10）

函数名称：ListInsert(LinkList &L,int i,ElemType e)

初始条件：线性表L已存在且非空，1<=i<=ListLength(L)+1；

操作结果：在L的第i个位置之前插入新的数据元素e。

算法思路：先遍历顺序表，若线性表L已存在并且不为空，输入的i值不合法，则返回ERROR。若满足线性表L已存在且L非空，并且i的值合法，则在线性表的第i个位置之前插入新的数据元素e，返回OK。

时间和空间复杂度：T(n)=O(n),S(n)=O(n)

流程图：

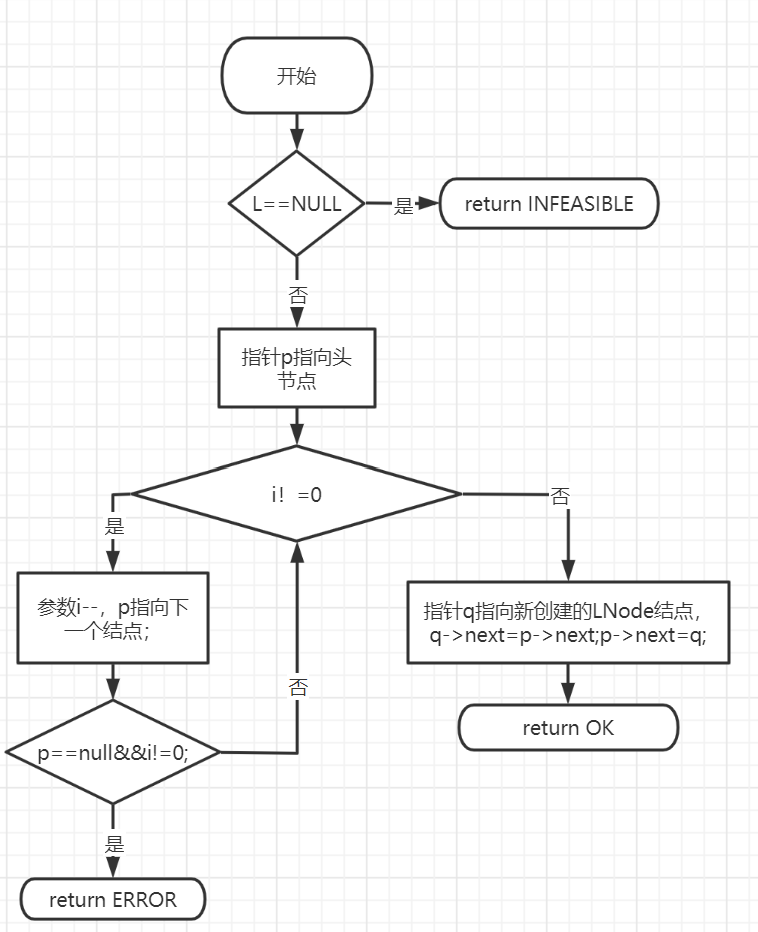


图1-6 ListInsert（L，i，e）流程图

（11）

函数名称：ListDelete(LinkList &L,int i,ElemType &e)

初始条件：线性表已存在且非空，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；

算法思路：先遍历顺序表，如果线性表L已存在且非空，并且输入的i值不合法，则返回ERROR。若满足线性表L已存在并且L非空，并且i的值合法，则删除线性表的第i个数据元素，并用e返回其值，返回OK。

时间和空间复杂度：T(n)=O(n),S(n)=O(1)

流程图：

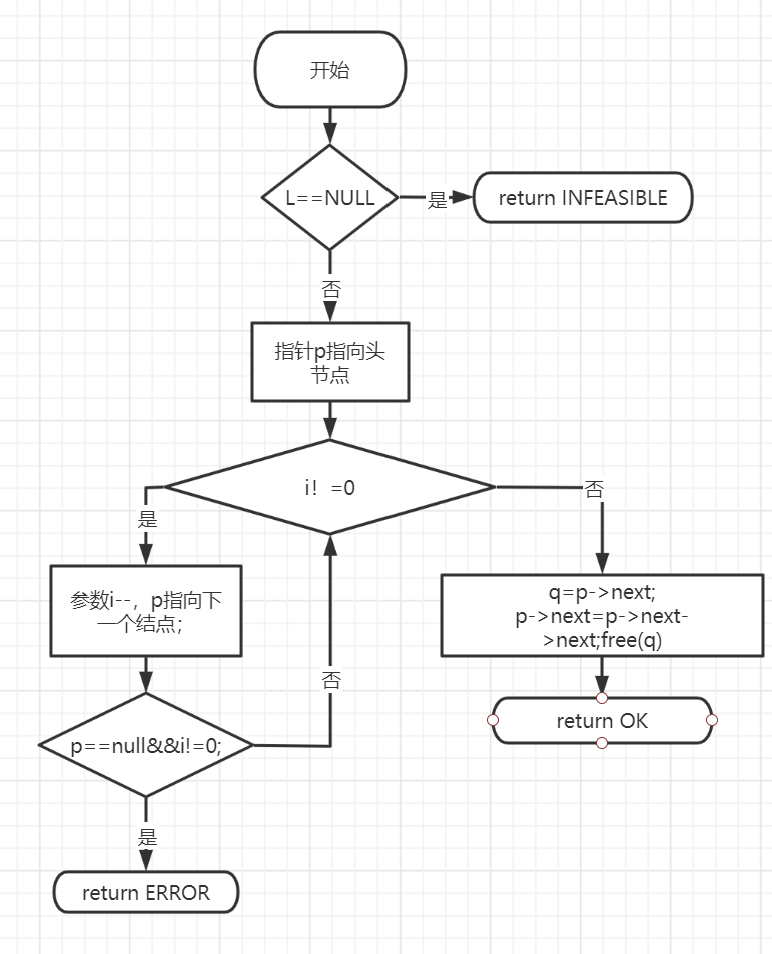


图1-7 ListDelete（L，i,e）流程图

（12）

函数名称：ListTrabverse(LinkList L)

初始条件：线性表存在；

操作结果：依次遍历L的每个数据元素并且输出；

算法思路：若线性表L存在，则遍历元素；否则返回ERROR。

时间和空间复杂度：T(n)=O(n),S(n)=O(1)

流程图：

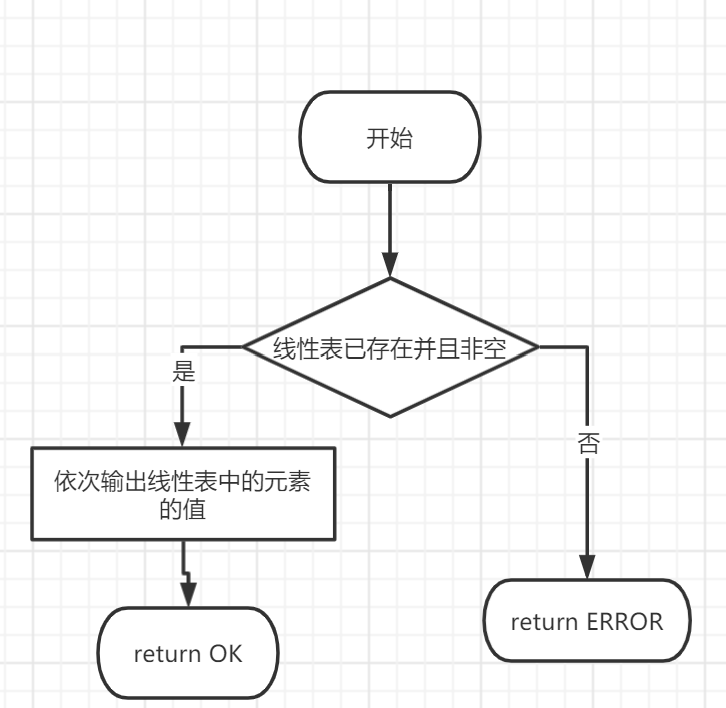


图1-8 ListTraverse（L）流程图

（13）

函数名称：SaveList(LinkList L,char FileName[])

初始条件：线性表存在；

操作结果：将线性表L保存为文件形式；

算法思路：用fwrite保存为文件。

时间和空间复杂度：T(n)=O(n),S(n)=O(1)

（14）

函数名称：LoadList(LinkList &L,char FileName[])

初始条件：线性表存在；

操作结果：将线性表L以文件形式加载读取；

算法思路：用fread将文件读取线性表。

时间和空间复杂度：T(n)=O(n),S(n)=O(n)

流程图：

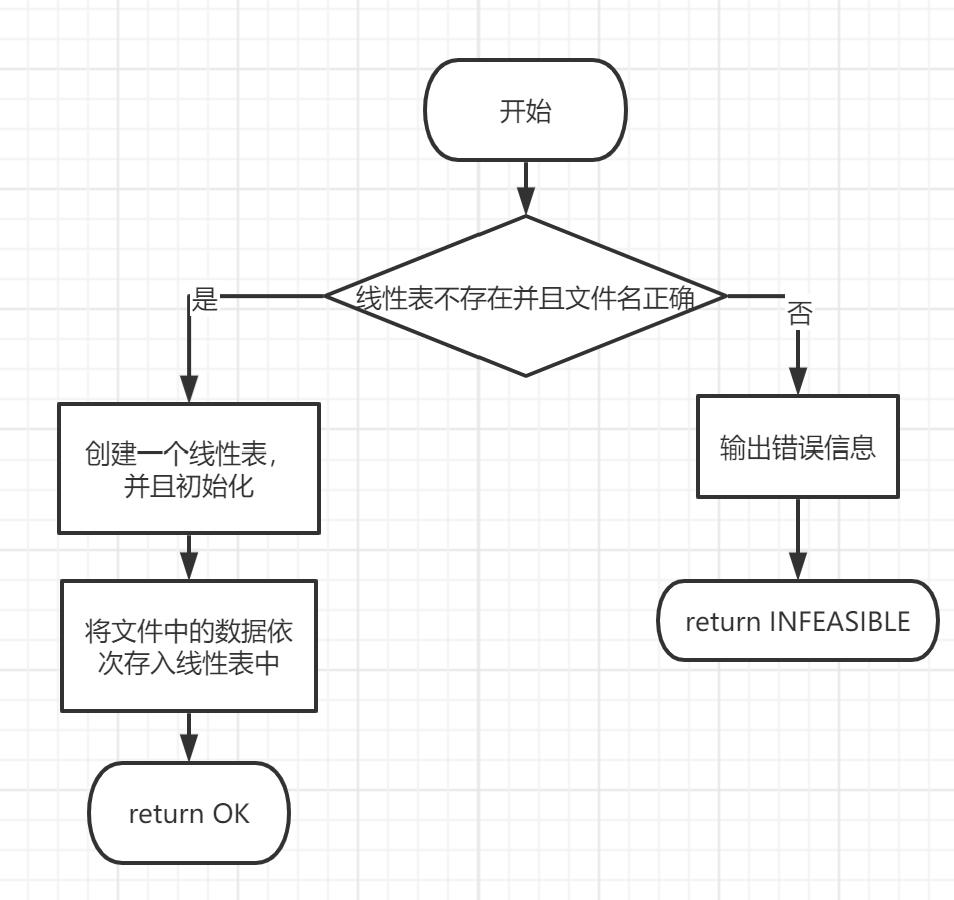


图1-9 LoadList（L，FileName）流程图

（15）

函数名称：AddList(LISTS &Lists,char ListName[])

初始条件：线性表集存在；

操作结果：在Lists中增加一个名称为ListName的空线性表；

算法思路：将ListName中的字符串复制到线性表的数据元素的名字数组里，然后初始化新的线性表。

时间和空间复杂度：T(n)=O(1),S(n)=O(1)

（16）

函数名称：RemoveList(LISTS &Lists,char ListName[])

初始条件：线性表集存在；

操作结果：在Lists中删除一个名称为ListName的空线性表；

算法思路：遍历Lists中的线性表，将每个线性表的名字与ListName相比较，如果相同则将后面的线性表向前移动覆盖掉要删除的表。如果没找到则返回ERROR。

时间和空间复杂度：T(n)=O(n),S(n)=O(1)

流程图：!!

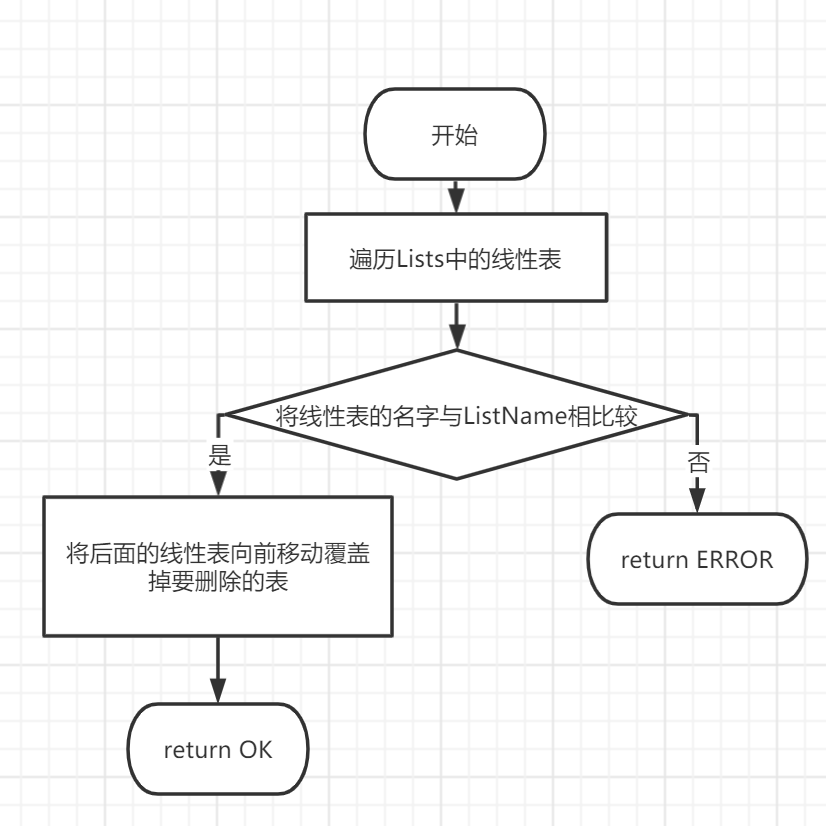


图1-10 RemoveList（Lists，ListName）流程图

（17）

函数名称：LocateList(LISTS Lists,char ListName[])

初始条件：线性表集存在；

操作结果：在Lists中查找一个名称为ListName的线性表；

算法思路：遍历Lists中的线性表，如果第i个线性表的名字与ListName相同，则返回线性表在线性表集中的序号，否则返回ERROR。

时间和空间复杂度：T(n)=O(n),S(n)=O(1)

流程图：

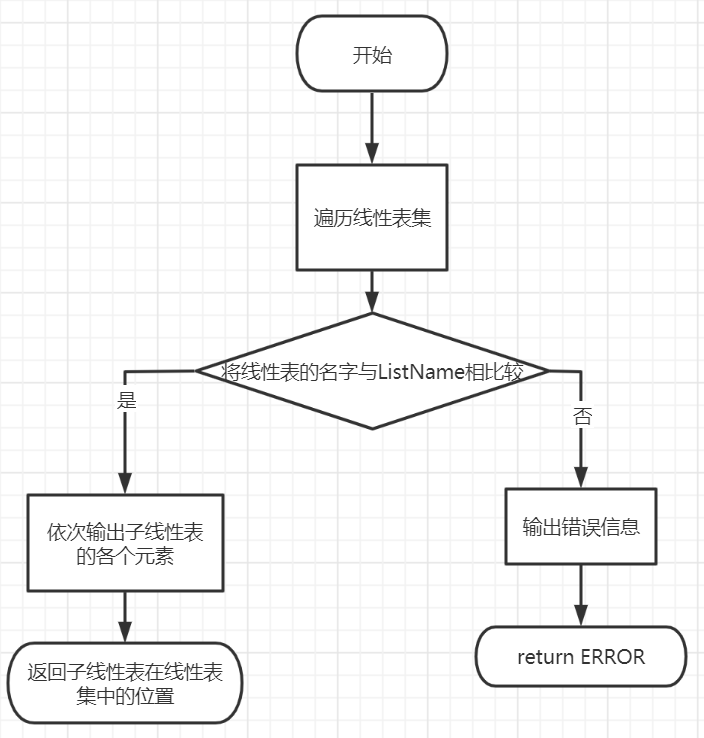


图1-11 LocateList（Lists，ListName）流程图

## 2.3 系统实现

### 2.3.1 程序实现的环境，开发工具与开发语言

程序实现环境：Win10-x64

开发工具：Dev c++

开发语言：C语言

### 2.3.2 头文件及预定义说明

头文件包括：

预定义文件def.h和预定义函数文件func.h。

预定义说明：

LIST\_INIT\_SIZE:线性表初始分配空间大小；

LISTINCREMENT:单空间不够时，线性表补充分配空间大小；

TRUE, FALSE, OK, ERROR, INFEASIBLE, OVERFLOW分别为状态产量，对应返回为真值，返回为假，程序运行正常，程序运行错误，程序运行不合法和程序溢出。

LNode为链式线性表结构定义类型，包括数据域data和指针域next， LinkList为指向LNode类型的指针。

LISTS为多线性表结构定义类型，包括结构数组elem[10],长度length和最大尺寸listsize。

### 2.3.3 代码组织结构

本实验代码结构可分为三部分：主函数”顺序结构顺序表.cpp”,预定义常量“def.h”,和函数文件“func.h”。预定义常量和函数文件不做赘述，具体可再前文中查找。

主函数主要采用了while循环和switch选择，通过用户输入来确定所要调用的函数。在switch下，还嵌套了一个while-switch循环用以对多线性表进行操作，从而实现多级菜单的功能。仍是通过读取用户输入判断要调用的函数。在嵌套结构下，可退回上一层，对选中的线性表进行操作。

## 2.4 系统测试

本程序采用简易操作界面，如图1-7所示，挑选ListLength，GetElem，PriorElem，LoadList，LocateList，AddList进行测试。



图2-1 程序初始界面

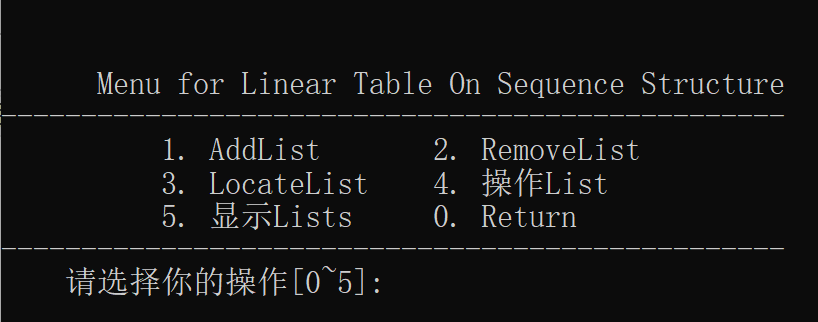


图2-2 多线性表界面

单线性表测试用例为：L{1，2，3，4，5}，空表，不存在线性表；多线性表测试用例为Lists{{‘jia’：1，2，3，4，5}，{‘ja’：1，3，5，7，9}，{‘j’}}；文件内容为{1，2，3，4，5}。

单线性表：

（1）ListLength测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| L | 界面选择5 | 线性表的长度为5 | 线性表的长度为5 |
| 空表 | 界面选择5 | 线性表的长度为0 | 线性表的长度为0 |
| 不存在线性表 | 界面选择5 | 线性表不存在 | 线性表不存在 |

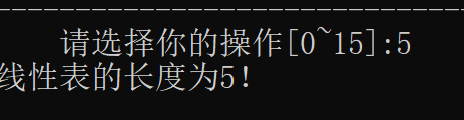


图2-3 ListLength测试1

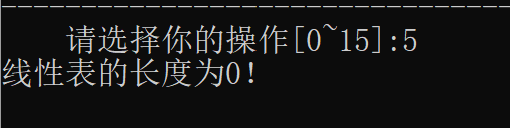


图2-4 ListLength测试2

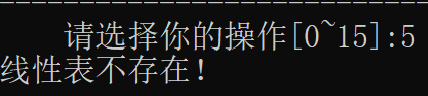


图2-5 ListLength测试3

（2）GetElem测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| L | 界面选择6，输入1， | 线性表的第1位是1 | 线性表的第1位是1 |
| 空表 | 界面选择6，输入1 | 输入的位置不合法，请重新输入 | 输入的位置不合法，请重新输入 |
| 不存在线性表 | 界面选择6，输入1 | 线性表不存在 | 线性表不存在 |

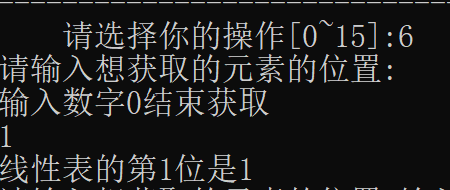


图2-6 GetElem测试1

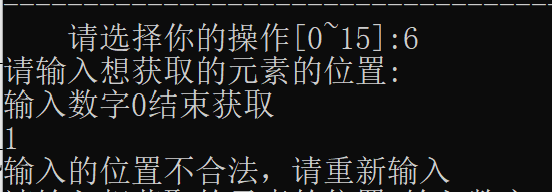


图2-7 GetElem测试2

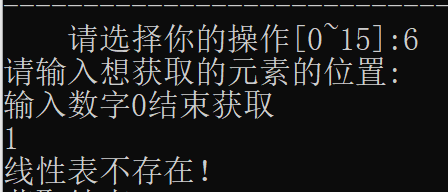


图2-8 GetElem测试3

（3）PriorElem测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| L | 界面选择8，输入4 | 元素4的前驱是3 | 元素4的前驱是3 |
| 空表 | 界面选择8，输入2 | 元素2没有前驱元素 | 元素2没有前驱元素 |
| 不存在线性表 | 界面选择8，输入2 | 线性表不存在 | 线性表不存在 |

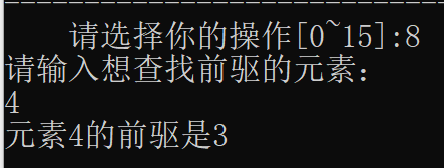


图2-9 PriorElem测试1

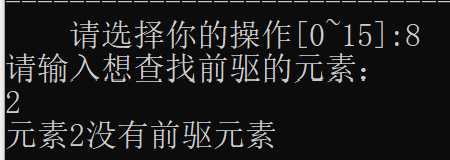


图2-10 PriorElem测试2

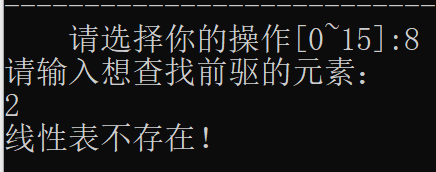


图2-11 PriorElem测试3

（4）LoadList测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| L | 界面选择14 | 读入失败，线性表已存在 | 读入失败，线性表已存在 |
| 空表 | 界面选择14 | 读入失败，线性表已存在 | 读入失败，线性表已存在 |
| 不存在线性表 | 界面选择14，界面选择12 | 输入14后输出：数据读入到线性表成功；输入12后输出：1 2 3 4 5输出成功 | 数据读入到线性表成功  1 2 3 4 5输出成功 |

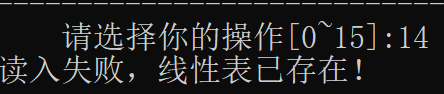


图2-12 LoadList测试1

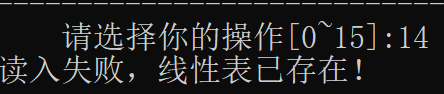
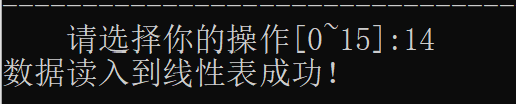


图2-13 LoadList测试2



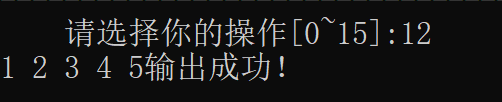


图2-14 LoadList测试3

多线性表：

（5）LocateList测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| Lists | 界面选择15，界面选择3，输入jia | 查找的线性表如下：jia 1 2 3 4 5 | 查找的线性表如下：jia 1 2 3 4 5 |
| 空表 | 界面选择15，界面选择3，输入j | 查找的线性表如下：j | 查找的线性表如下：j |
| 不存在线性表 | 界面选择15，界面选择3，输入abc | 查找失败 | 查找失败 |

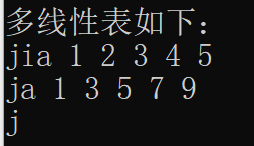


图2-15 LocateList测试集展示

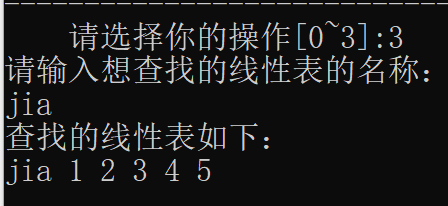


图2-16 LocateList测试1

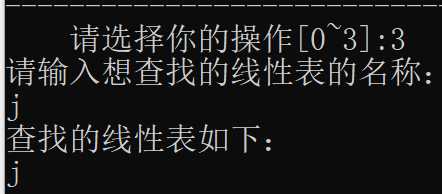


图2-17 LocateList测试2

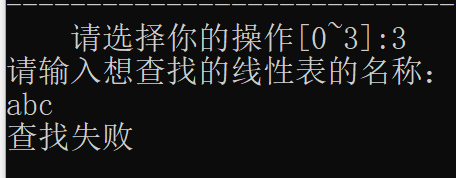


图2-18 LocateList测试3

(6)AddList测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| 空多线性表 | 界面选择15，界面选择1，输入1，输入jia,输入1 2 3 4 5 0； | 多线性表如下：jia 1 2 3 4 5 | 多线性表如下：jia 1 2 3 4 5 |
| 已有的多线性表，内容为{jia:1 2 3 4 5} | 界面选择15，界面选择1，输入1，输入ja,输入1 2 3 0； | 多线性表如下：jia 1 2 3 4 5  ja 1 2 3 | 多线性表如下：jia 1 2 3 4 5  Ja 1 2 3 |

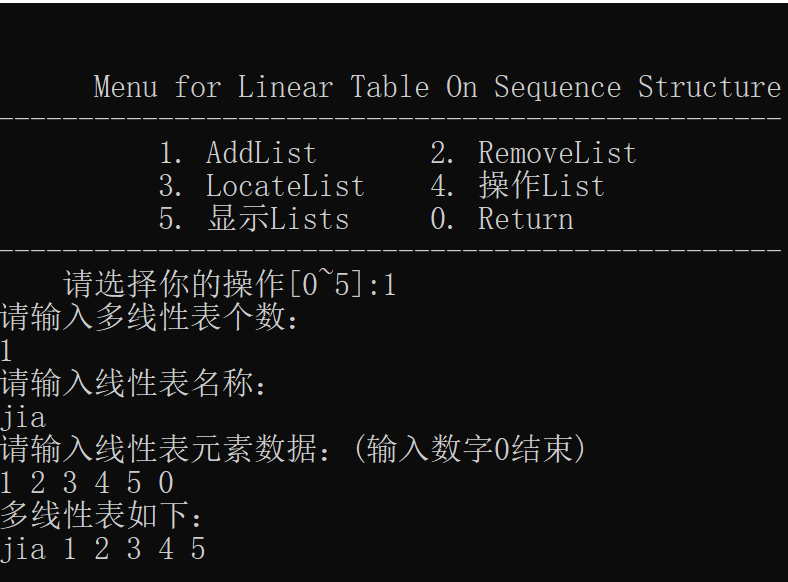


图2-19 AddList测试1

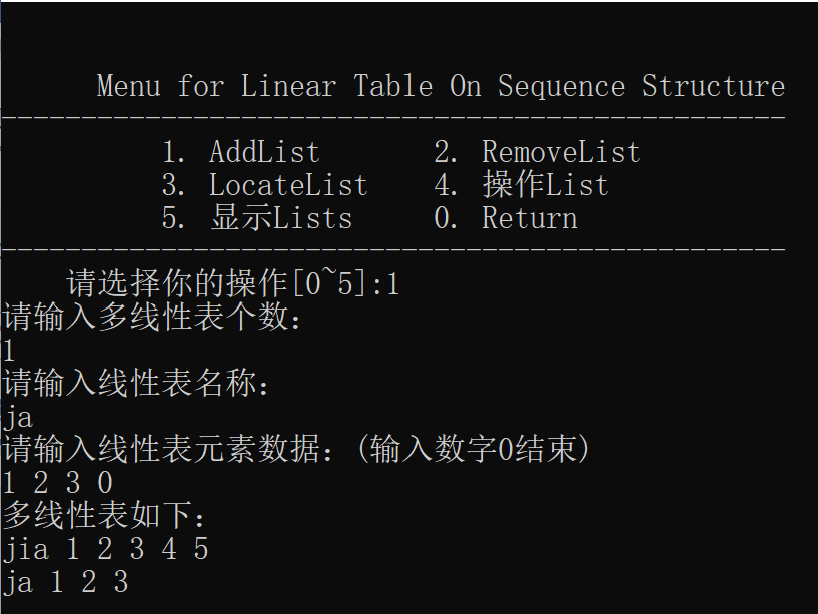


图2-20 AddList测试2

## 2.5 实验小结

经过这次的实验，我对线性表的顺序存储结构和链式存储结构有了更深的理解，同时也在实际的思考问题和写代码的过程中理解到了两种储存结构的异同，以及面对不同的问题如何选择更加适合的储存结构。相较之于顺序结构，链式结构在增加，删除，插入元素方面更加便利和快捷，但是在直接获取元素和求表长方面就不是那么有优势。而且顺序储存在表长变化较大时，难以确定合适的存储规模，所以运用链式线性表会更加便捷。所以做个总结的话就是，在线性表的长度变化较大或者需要频繁的进行插入删除操作的时，选择链表做存储结构会是一个更好的选择。

这次的实验相较之于第一次实验难度相同，在代码思路方面和第一次是近乎一致的，所以在完成线上内容时会快一些，因为已经有了之前的思考和思路。在组装系统的时候，利用第一次的结构可以很快地完成系统的组装，但是需要在部分细节的部分进行一些修改以适应链式的结构。

虽然完成了此次实验，但是对链表的操作和相关算法的学习还要加强，个人编程和解决问题的能力还需要加强。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

通过实验达到⑴加深对二叉树的概念、基本运算的理解；⑵熟练掌握二叉树的逻辑结构与物理结构的关系；⑶以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。

### 3.1.1 14个基本函数需求实现

依据最小完备性和常用性相结合的原则，以函数形式定义了二叉树的创建二叉树、销毁二叉树、清空二叉树、判定空二叉树和求二叉树深度等14种基本运算。具体运算功能定义和说明如下。

⑴创建二叉树：函数名称是CreateBiTree(T,definition)；初始条件是definition 给出二叉树T的定义，如带空子树的二叉树前序遍历序列、或前序+中序、或后序+中序；操作结果是按definition构造二叉树T。

⑵销毁二叉树：函数名称是DestroyBiTree(T)；初始条件是二叉树T已存在；操作结果是销毁二叉树T。

⑶清空二叉树：函数名称是ClearBiTree (T)；初始条件是二叉树T存在；操作结果是将二叉树T清空。

⑷判定空二叉树：函数名称是BiTreeEmpty(T)；初始条件是二叉树T存在；操作结果是若T为空二叉树则返回TRUE，否则返回FALSE。

⑸求二叉树深度：函数名称是BiTreeDepth(T)；初始条件是二叉树T存在；操作结果是返回T的深度。

⑹查找结点：函数名称是LocateNode(T,e)；初始条件是二叉树T已存在，e是和T中结点关键字类型相同的给定值；操作结果是返回查找到的结点指针，如无关键字为e的结点，返回NULL。

⑺结点赋值：函数名称是Assign(T,e,value)；初始条件是二叉树T已存在，e是和T中结点关键字类型相同的给定值；操作结果是关键字为e的结点赋值为value。

⑻获得兄弟结点：函数名称是GetSibling(T,e)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值；操作结果是返回关键字为e的结点的（左或右）兄弟结点指针。若关键字为e的结点无兄弟，则返回NULL。

⑼插入结点：函数名称是InsertNode(T,e,LR,c)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值，LR为0或1，c是待插入结点；操作结果是根据LR为0或者1，插入结点c到T中，作为关键字为e的结点的左或右孩子结点，结点e的原有左子树或右子树则为结点c的右子树。

特殊情况，c插入作为根结点？可以考虑LR为-1时，作为根结点插入，原根结点作为c的右子树。

⑽删除结点：函数名称是DeleteNode(T,e)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值。操作结果是删除T中关键字为e的结点；同时，如果关键字为e的结点度为0，删除即可；如关键字为e的结点度为1，用关键字为e的结点孩子代替被删除的e位置；如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为e的左子树中最右结点的右子树。

⑾前序遍历：函数名称是PreOrderTraverse(T,Visit())；初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）；操作结果：先序遍历，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

**注：**前序、中序和后序三种遍历算法，要求至少一个用非递归算法实现。

⑿中序遍历：函数名称是InOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）；操作结果是中序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒀后序遍历：函数名称是PostOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）；操作结果是后序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒁按层遍历：函数名称是LevelOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是层序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

### 3.1.2 演示系统与文件数据及多表管理

采用二叉链表表作为二叉树的物理结构。其中ElemType为数据元素的类型名，具体含义可自行定义，但要求二叉树结点类型为结构型，至少包含二个部分，一个是能唯一标识一个结点的关键字，另一个是其它部分。

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。

演示系统可选择实现二叉树的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存二叉树数据逻辑结构(D,{R})的完整信息；②需要设计二叉树文件保存和加载操作合理模式。

演示系统可选择实现多个二叉树管理。可采用线性表的方式管理多个二叉树，线性表中的每个数据元素为一个二叉树的基本属性，至少应包含有二叉树的名称。基于顺序表实现的二叉树管理。

## 3.2 系统设计

### 3.2.1 系统总体设计

该系统提供一个物理结构为二叉链表的二叉树，以及由线性表进行管理的多棵二叉树。

本简易演示系统可提供的操作：初始化二叉树、清空二叉树、创建二叉树、销毁二叉树、判定空二叉树、求二叉树深度、获得结点、插入子树、删除子树和遍历以及对森林进行操作。具体结构如图1-1所示。



图1-1 系统结构

### 3.2.2 算法设计

(1)

函数名称：InitTree(BiTree &T)

初始条件：二叉树T不存在

操作结果：构造空二叉树T根结点

算法思想实现：采用指针T指向二叉树的根结点，将T赋值为NULL完成二叉树的初始化。

复杂度：T(n)=O(1),S(n)=O(1)

(2)

函数名称：CreateBiTree(T)

初始条件：二叉树T已存在

操作结果：构造二叉树T

算法思想实现：传入根节点指针，使用递归算法，由用户自行输入每个根节点的信息。

复杂度：T(n)=O(n),S(n)=O(n)

(3)

函数名称：ClearBiTree (T)

初始条件：二叉树T存在

操作结果：将二叉树T清空

算法思想实现：采用递归算法，清除每个结点的左子树和右子树，然后最后将根节点设置为空。

复杂度：T(n)=O(n),S(n)=O(n)

(4)

函数名称：BiTreeDepth(T)

初始条件：二叉树T存在

操作结果：返回T的深度

算法思想实现：从根节点出发，树的深度等于左右子树深度较大值加1，若左右子树均为空，则该树深度为1；若根节点为空，则深度为0。

复杂度：T(n)=O(logn),S(n)=O(logn)

流程图：



(5)

函数名称：LocateNode(BiTree T,KeyType e)

初始条件：二叉树T存在

操作结果：返回与关键字相同结点的关键字和内容

算法思想实现：递归调用函数，通过递归先序遍历二叉树，即先比较根结点的key与给定是否一致，若一致，返回该结点的data，否则继续分别递归遍历其左子树和右子树。

复杂度：T(n)=O(n),S(n)=O(n)

流程图：



(6)

函数名称：Assign(BiTree &T,KeyType e,TElemType value)

初始条件：二叉树T已存在，e是T中的某个结点

操作结果：将结点e赋值为value

算法思想实现：通过LocateNode(T, e)函数根据输入的key获取到e结点，并对结点赋值。

复杂度：T(n)=O(n),S(n)=O(n)

(7)

函数名称：GetSibling(BiTree T,KeyType e)

初始条件：二叉树T已存在，e关键字

操作结果：获取关键字为e的结点的兄弟结点

算法思想实现：递归调用该函数，判断结点左子树和右子树是否满足关键字，如果左子树满足则返回右子树，反之返回左子树。

复杂度：T(n)=O(n),S(n)=O(n)

(8)

函数名称：InsertNode(BiTree &T,KeyType e,int LR,TElemType c)

初始条件：二叉树T存在，e为关键字，LR为0或1或-1，c是插入结点内容。

操作结果：根据LR为0或者1，插入c为T中与e关键字相同结点的左或右子树，若 LR为-1，则作为根节点插入，e关键字所示结点的原有左子树或右子树则为c的右子树

算法思想实现：e是和T中结点关键字类型相同的给定值，LR为0或1，c是待插入结点；根据LR为0或者1，插入结点c到T中，作为关键字为e的结点的左或右孩子结点，结点e的原有左子树或右子树则为结点c的右子树，返回OK。如果插入失败，返回ERROR。 特别地，当LR为-1时，作为根结点插入，原根结点作为c的右子树。

复杂度：T(n)=O(n),S(n)=O(n)

(9)

函数名称：DeleteNode(BiTree &T,KeyType e)

初始条件：二叉树T已存在，e是T中的某个结点

操作结果：将结点关键字为e的结点删除

算法思想实现：e是和T中结点关键字类型相同的给定值。删除T中关键字为e的结点；同时，如果关键字为e的结点度为0，删除即可；如关键字为e的结点度为1，用关键字为e的结点孩子代替被删除的e位置；如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为e的左子树中最右结点的右子树。

复杂度：T(n)=O(n),S(n)=O(n)

(10)

函数名称：PreOrderTraverse(T,Visit())

初始条件：二叉树T存在，Visit是对结点操作的应用函数

操作结果：先序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

算法思想实现：采用递归算法的思路，首先访问其根结点，对任一结点开始访问时都看作根结点，访问完根结点后若左孩子不为空，则按相同规则访问其左子树，访问完左子树后，再访问其右子树。

复杂度：T(n)=O(n),S(n)=O(n)

流程图：

(11)

函数名称：InOrderTraverse(T,Visit())

初始条件：二叉树T存在，Visit是对结点操作的应用函数

操作结果：中序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

算法思想实现：采用递归算法的思路，首先访问其根结点的左子树，对任一结点开始访问时都看作根结点，访问完后若左孩子为空，则访问根结点，访问完根结点后，再访问其右子树。

复杂度：T(n)=O(n),S(n)=O(n)

(12)

函数名称：PostOrderTraverse(T,Visit())

初始条件：二叉树T存在，Visit是对结点操作的应用函数

操作结果：后序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

算法思想实现：使用栈进行操作，要保证根结点在左孩子和右孩子访问之后才能访问，因此对于任一结点P，先将其入栈。如果P不存在左孩子和右孩子，则可以直接访问它；或者P存在左孩子或者右孩子，但是其左孩子和右孩子都已被访问过了，则同样可以直接访问该结点。若非上述两种情况，则将P的右孩子和左孩子依次入栈，这样就保证了每次取栈顶元素的时候，左孩子在右孩子前面被访问，左孩子和右孩子都在根结点前面被访问。

复杂度：T(n)=O(n),S(n)=O(logn)

(13)

函数名称：LevelOrderTraverse(T,Visit())

初始条件：二叉树T存在，Visit是对结点操作的应用函数

操作结果：层序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败

算法思想实现： 利用队列进行实现，结点入队，若队列非空，则出队，将非空的左和右孩子入队，循环，直到队列为空。

复杂度：T(n)=O(n),S(n)=O(n)

流程图：



## 3.3 系统实现

### 3.3.1 程序实现的环境，开发工具和开发语言

程序实现环境：Win10-x64

开发工具：Dev c++

开发语言：C语言

### 3.3.2 头文件及预定义说明

头文件包括预定义头文件def.h和函数声明头文件function.h。

在预定义头文件中：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAXQSIZE 100

typedef int status;

typedef int KeyType;

typedef struct {

KeyType key;

char others[20];

} TElemType; //二叉树结点类型定义

typedef struct BiTNode{ //二叉链表结点的定义

TElemType data;

struct BiTNode \*lchild,\*rchild;

} BiTNode, \*BiTree;

typedef struct{ //森林的管理表定义

struct {

char name[30];

BiTree L;

} elem[20];

int length;

int listsize;

}LISTS;

typedef struct {//创建二叉树输入类型结构

int pos;

TElemType data;

} Def;

typedef struct{//二叉树的位置和内容结构

int pos;

BiTree tree;

}Que;

typedef struct{//二叉树及位置队列结构

Que\* base;

int front;

int rear;

}SqQue;

typedef struct{//堆栈结构

BiTree\* base;

BiTree\* top;

int stacksize;

}SqStack;

typedef struct{//二叉树队列结构

BiTree\* base;

int front;

int rear;

}SqQueue;

### 3.3.3 代码组织结构

本实验代码结构可分为四个部分：主函数”main.cpp”,预定义常量“def.h”,和函数声明文件“function.h”还有函数实现文件“func.cpp”。预定义常量和函数文件不做赘述，具体可再前文中查找。

代码结构：主函数主要采用了while循环和switch选择，通过用户输入来确定所要调用的函数来对二叉树进行相应的操作。在switch下，还嵌套了一个while-switch循环用以对森林，即多二叉树进行操作，从而实现多级菜单的功能。仍是通过读取用户输入判断要调用的函数。在嵌套结构下，可退回上一层，对选中的二叉树进行操作。

头文件关系：主函数main.cpp调用了function.h中的功能函数以及def.h中的相关定义,func.cpp也调用了 def.h中的定义。

函数之间调用关系：Assign、InsertNode、DeleteNode函数内部调用了LocateNode函数，DeleteNode、PostOrderTraverse函数还调用了findtheright函数，用于寻找最右结点，以及栈相关函数，如initstack、gettop、push、pop函数，LevelOrderTraverse函数调用了队列相关函数，如enqueue、dequeue、queueempty函数，SaveBiTree函数内部调用了MyLevelOrderTraverse函数，MyLevelOrderTraverse函数是由LevelOrderTraverse函数改写而来，LoadBiTree函数调用了Create­­\_BiTree函数。

## 3.4 系统测试

本程序采用简易操作界面，如图2-1所示，挑选BiTreeDepth，LocateNode，PreOrderTraverse，LevelOrderTraverse，AddTree，LocateTree函数进行测试。

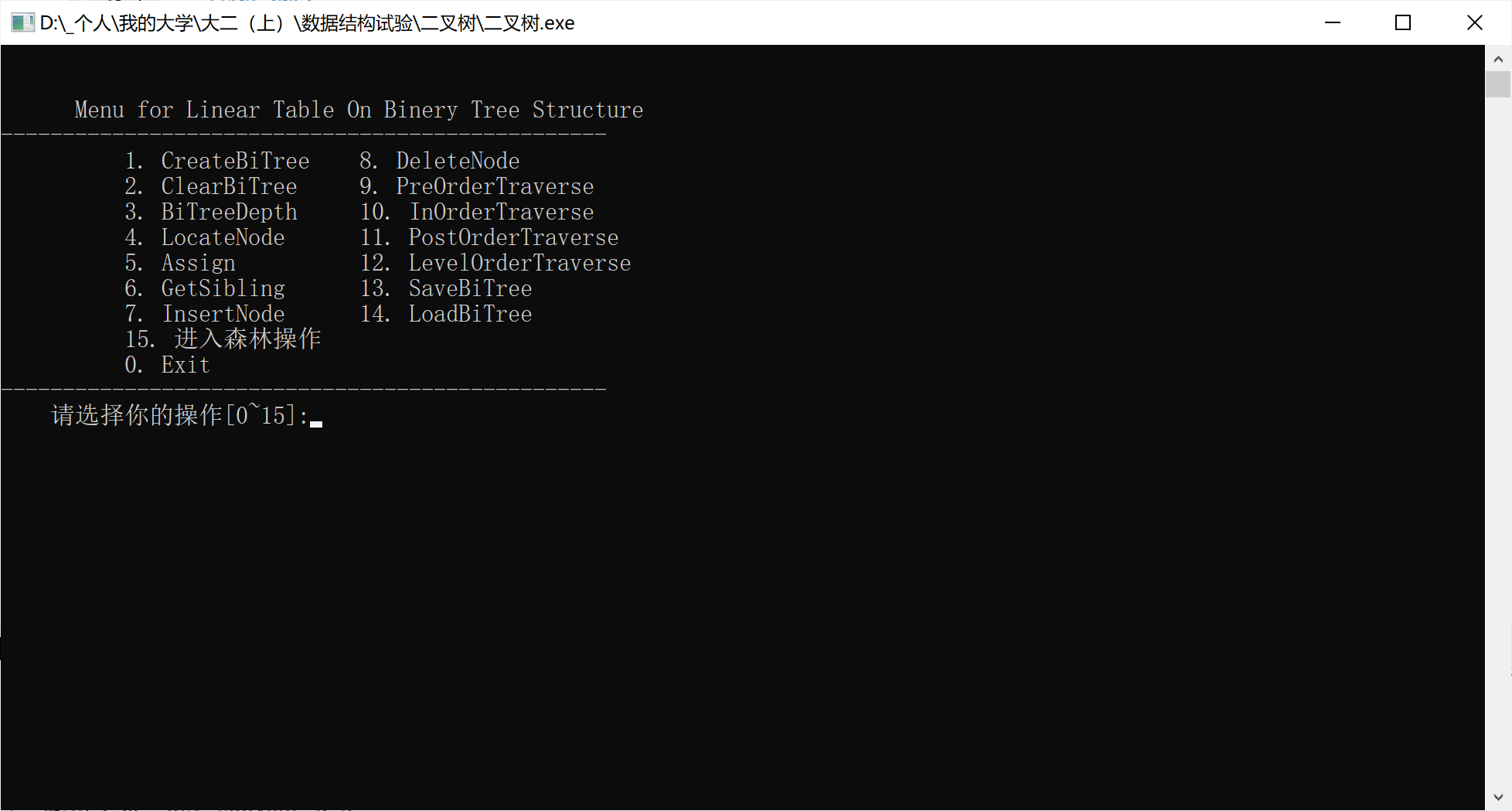


图2-1 程序初始界面

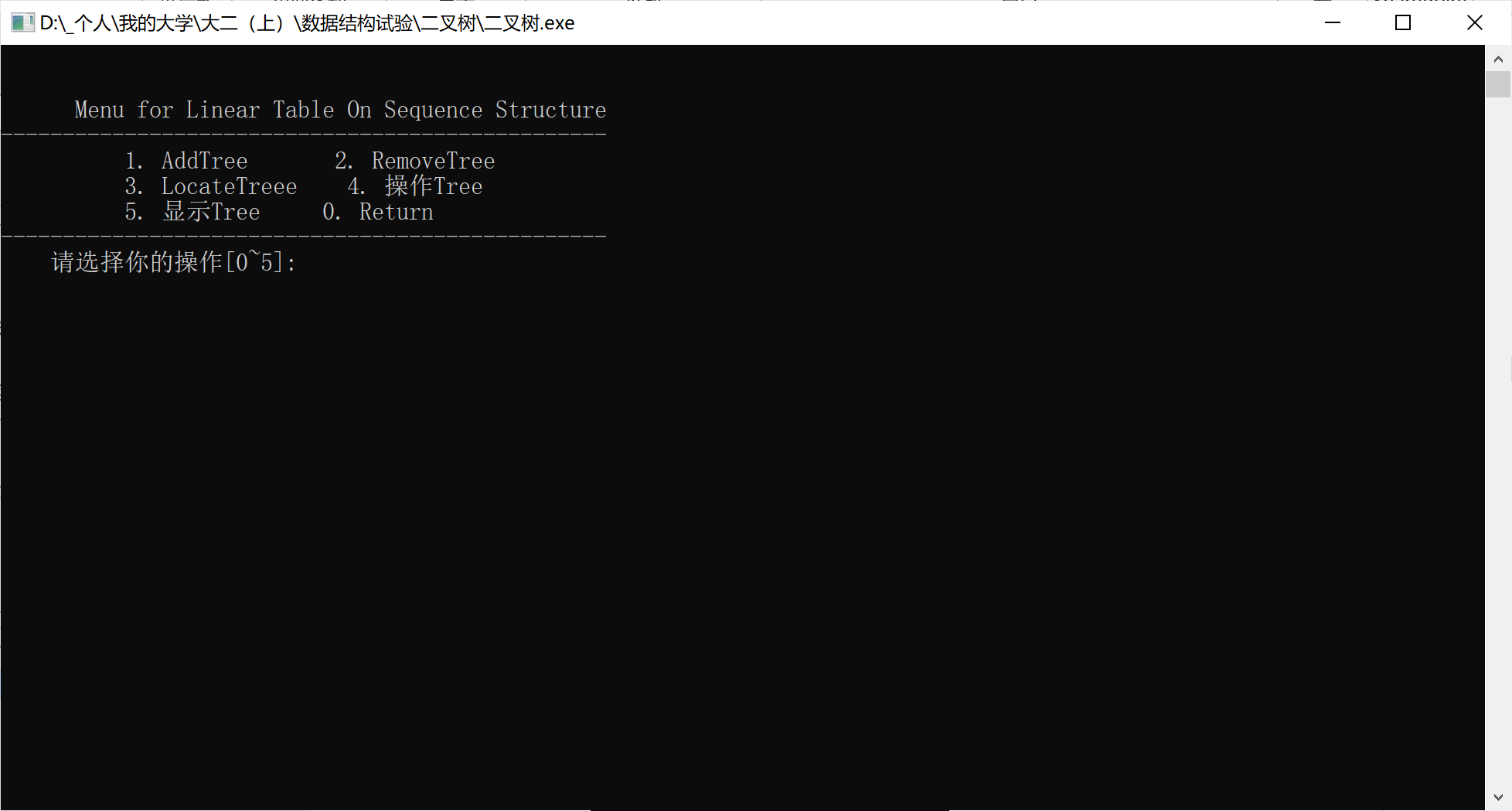


图2-2 多线性表界面



图2-3 二叉树图例



图2-4 多二叉树tree1图例



图2-5 多二叉树tree2图例

单二叉树测试用例：T(如图2-3所示)，空二叉树；多二叉树测试用例为{{‘tree1’：1 a,2 b,3 c}，{‘tree2’：1 g,2 b,3 c }}；文件内容为{1 1 g 2 2 b 3 3 c 6 4 d 7 5 e 0 0 NULL}。

单线性表：

（1）BiTreeDepth测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| T | 界面选择3 | 二叉树的深度为3 | 二叉树的深度为3 |
| 空二叉树 | 界面选择3 | 二叉树的深度为0 | 二叉树的深度为0 |

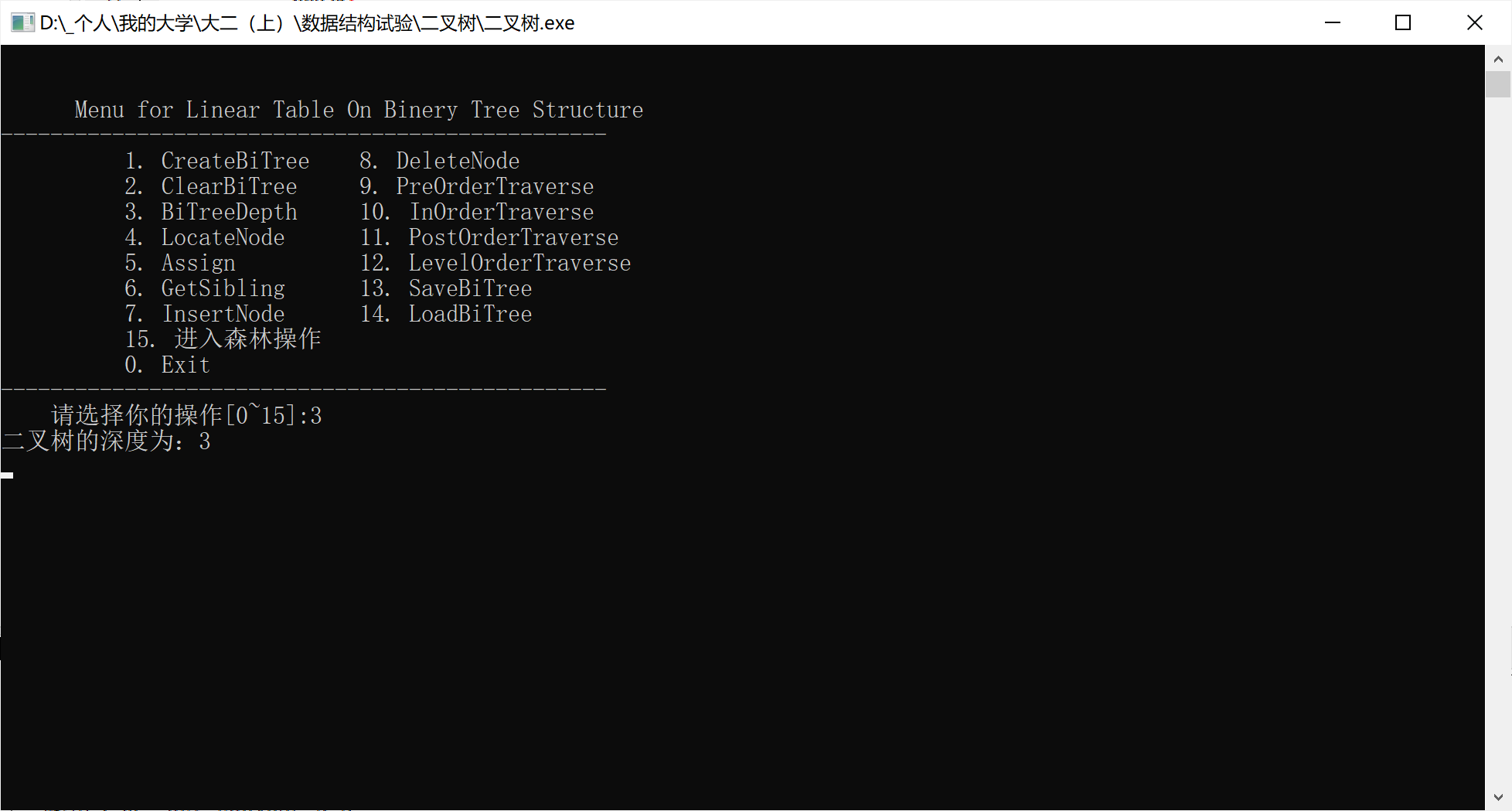


图2-6 BiTreeDepth测试1

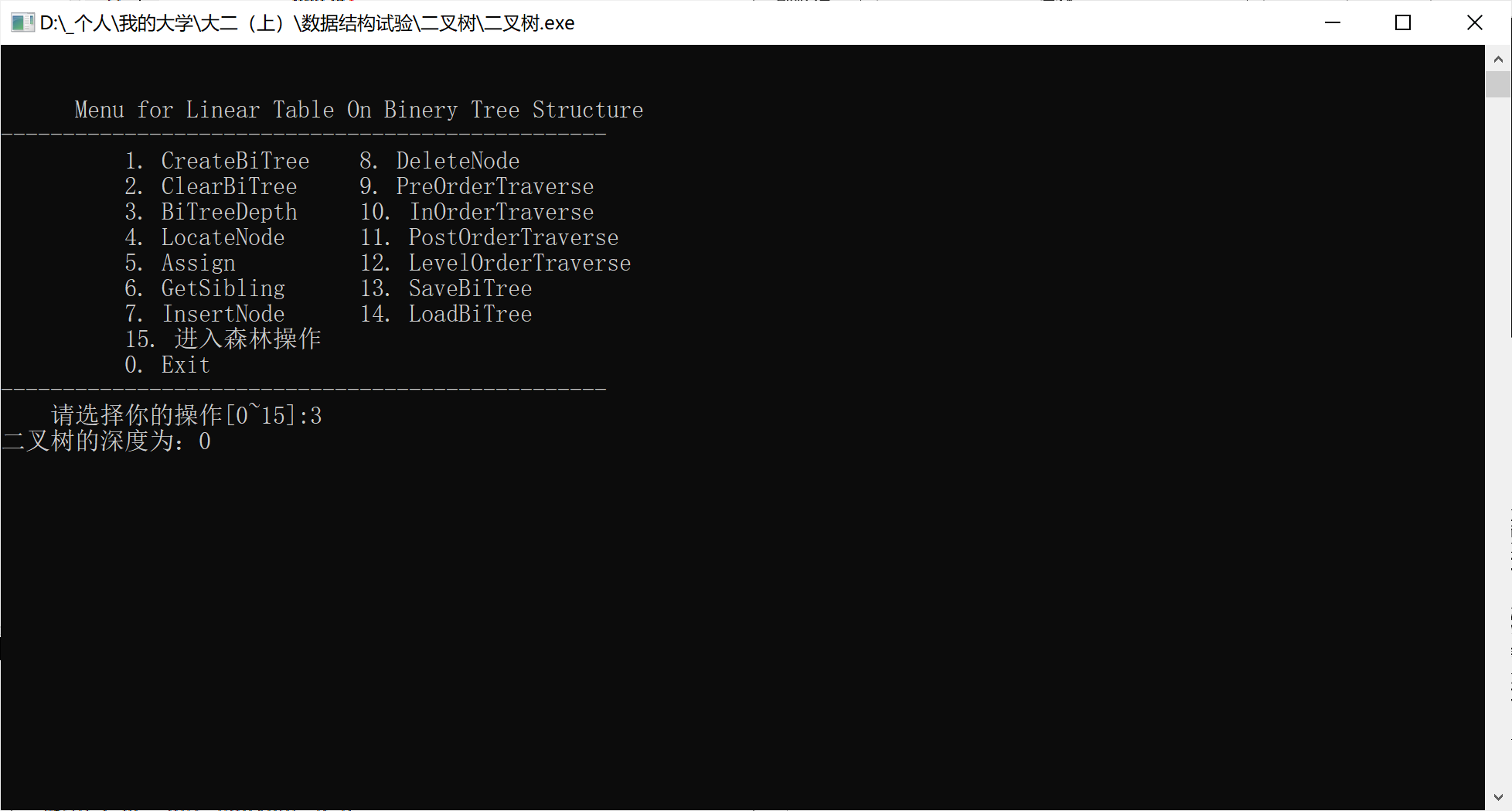


图2-7 BiTreeDepth测试2

（2）LocateNode测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| T | 界面选择4，输入1 | 1，g | 1，g |
| T | 界面选择4，输入7 | 查找失败 | 查找失败 |
| 空二叉树 | 界面选择4，输入1 | 查找失败 | 查找失败 |

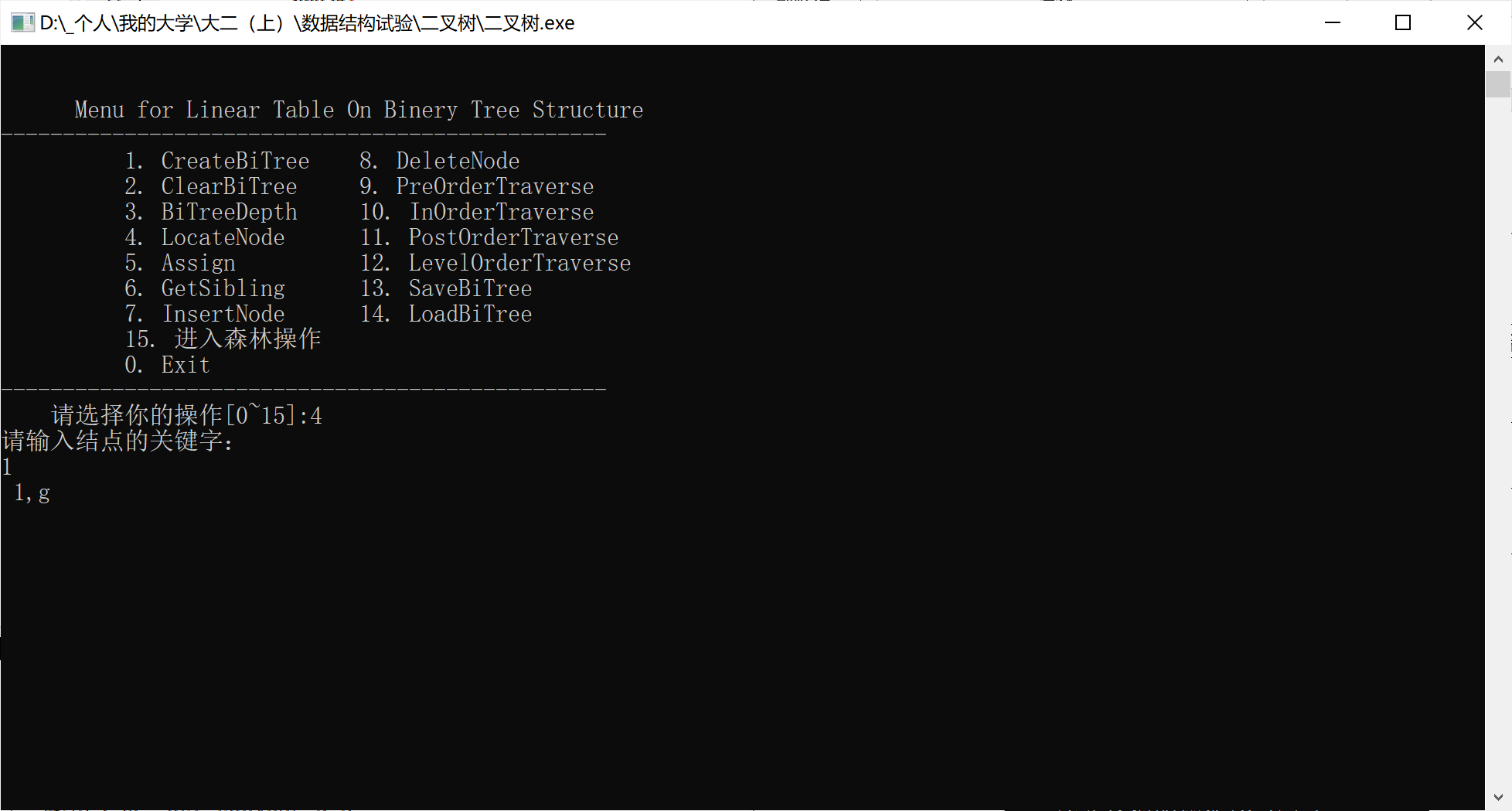


图2-8 LocateNode测试1

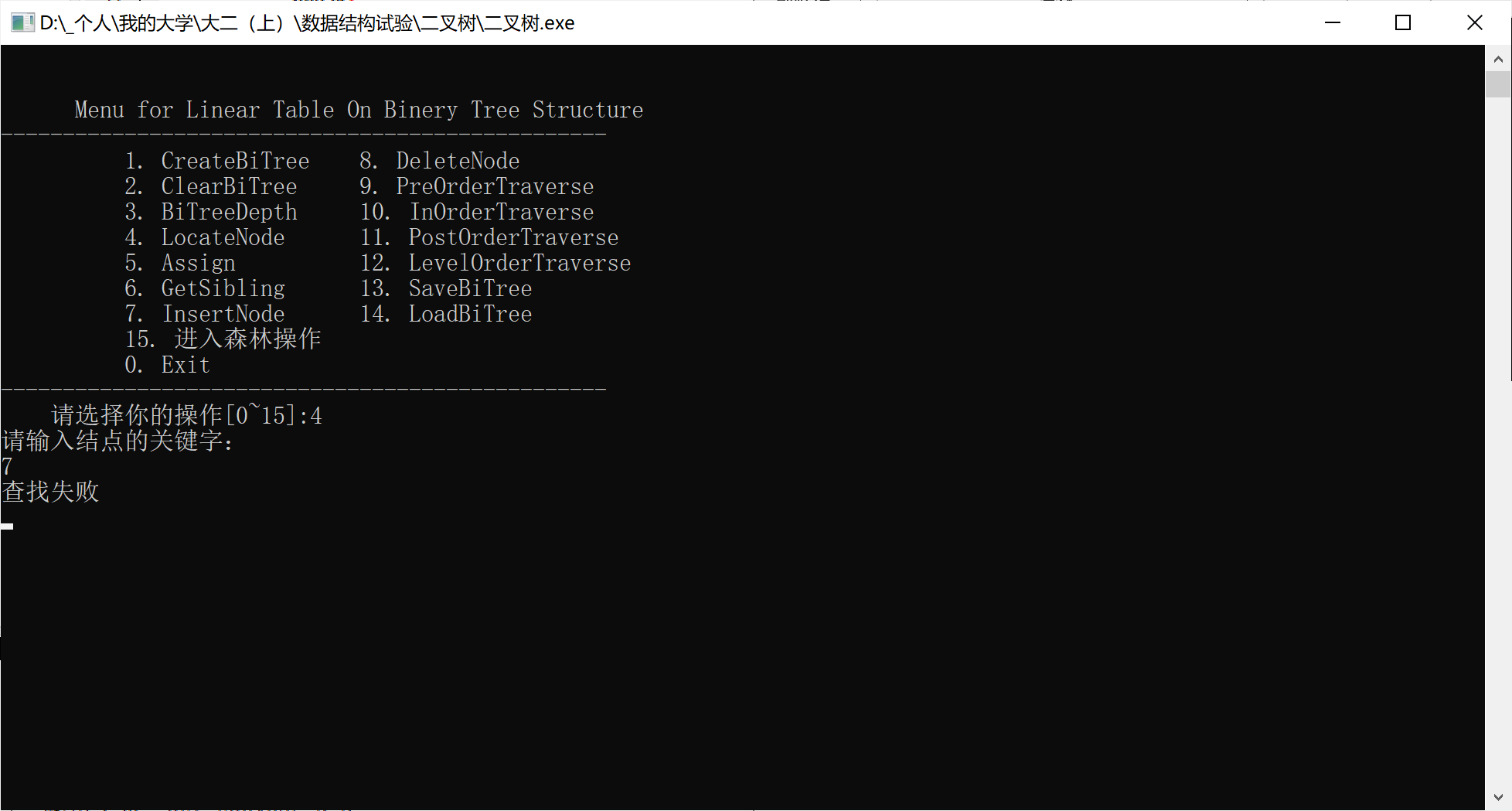


图2-9 LocateNode测试2

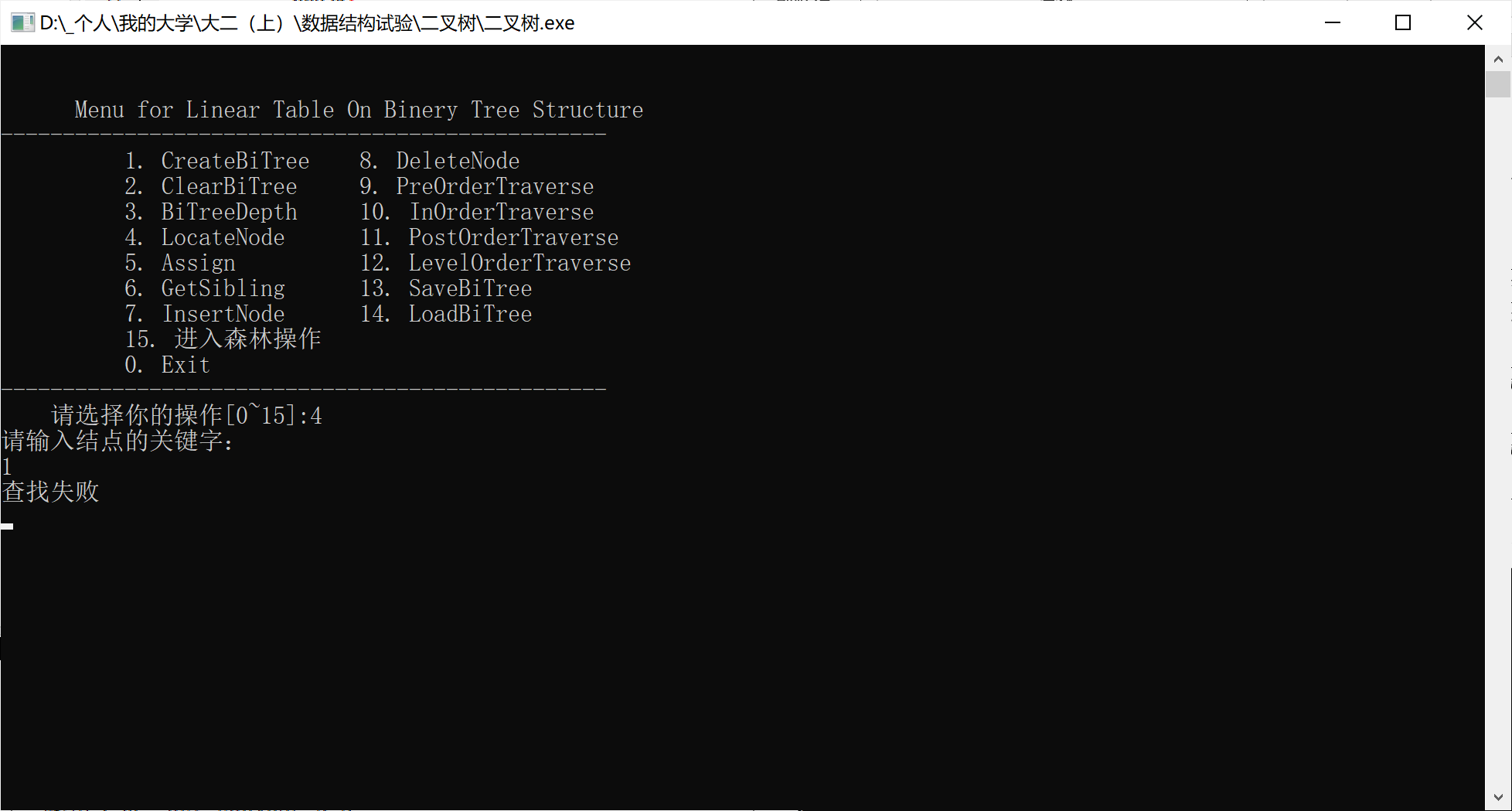


图2-10 LocateNode测试3

（3）PreOrderTraverse测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| T | 界面选择9 | 1,g 2,b 3,c 4,d 5,e | 1,g 2,b 3,c 4,d 5,e |
| 空二叉树 | 界面选择9 | 二叉树为空 | 二叉树为空 |

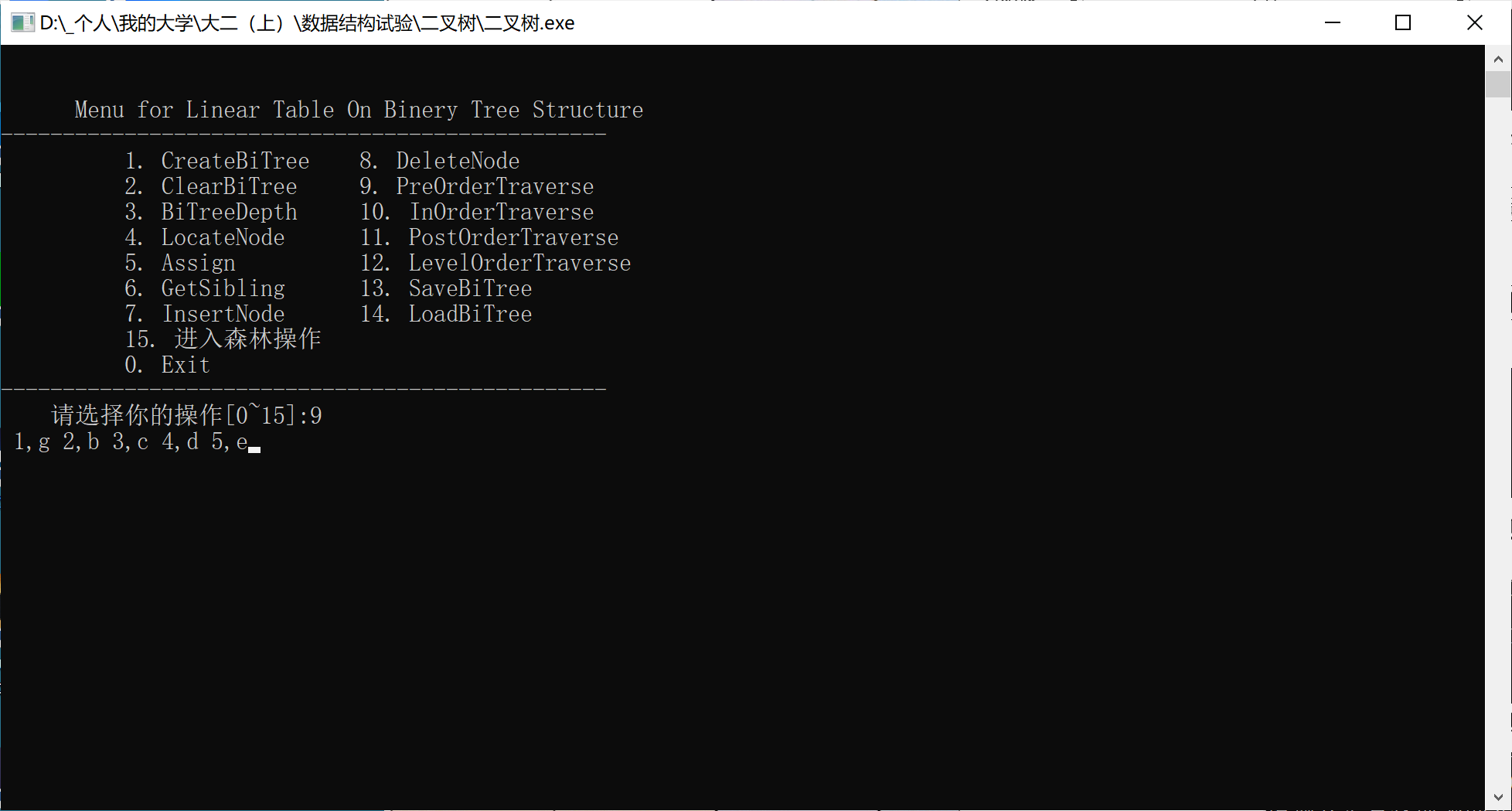


图2-11 PreOrderTraverse测试1

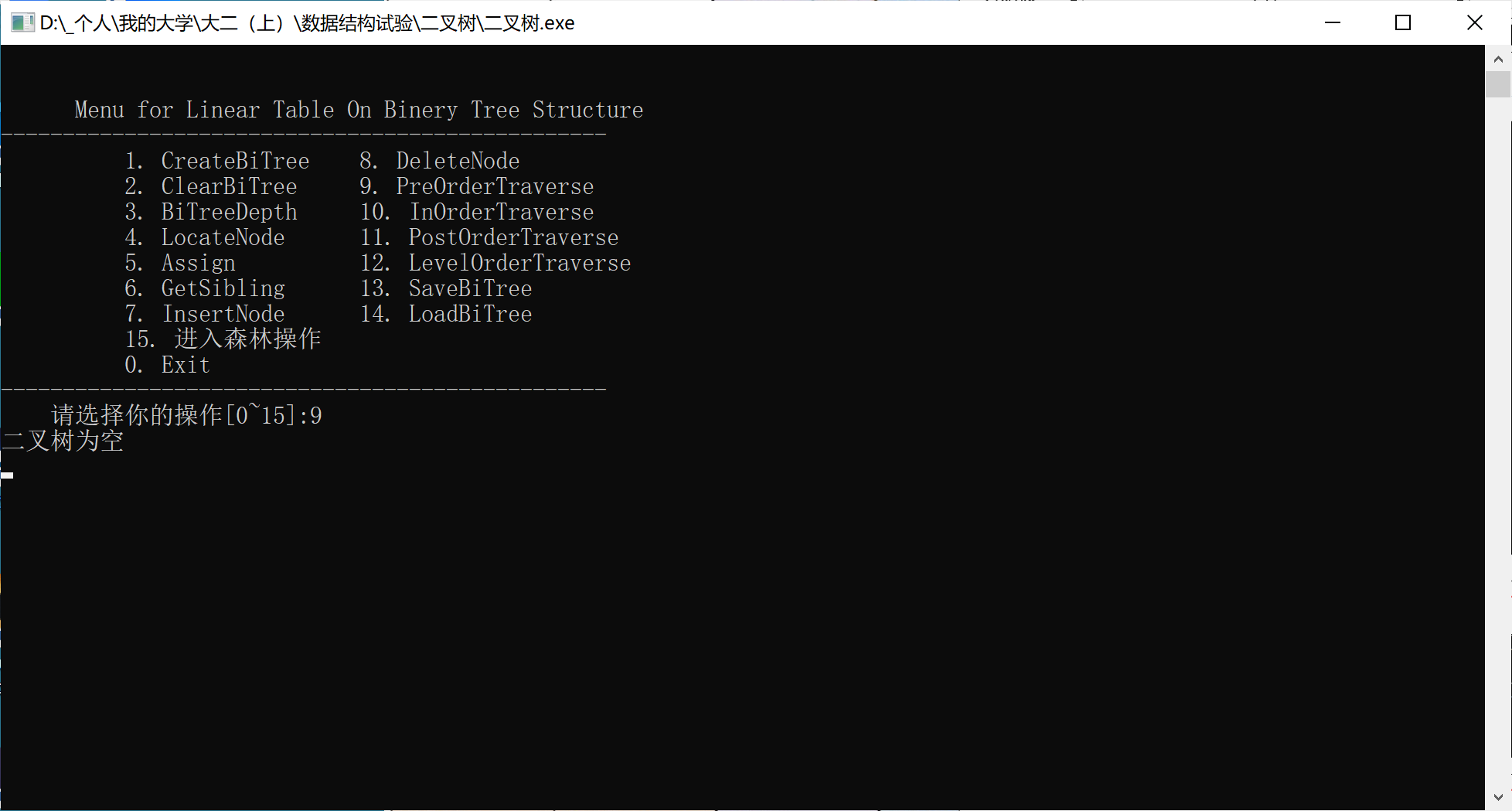


图2-12 PreOrderTraverse测试2

（4）LevelOrderTraverse测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| T | 界面选择12 | 1,g 2,b 3,c 4,d 5,e | 1,g 2,b 3,c 4,d 5,e |
| 空二叉树 | 界面选择12 | 二叉树为空 | 二叉树为空 |

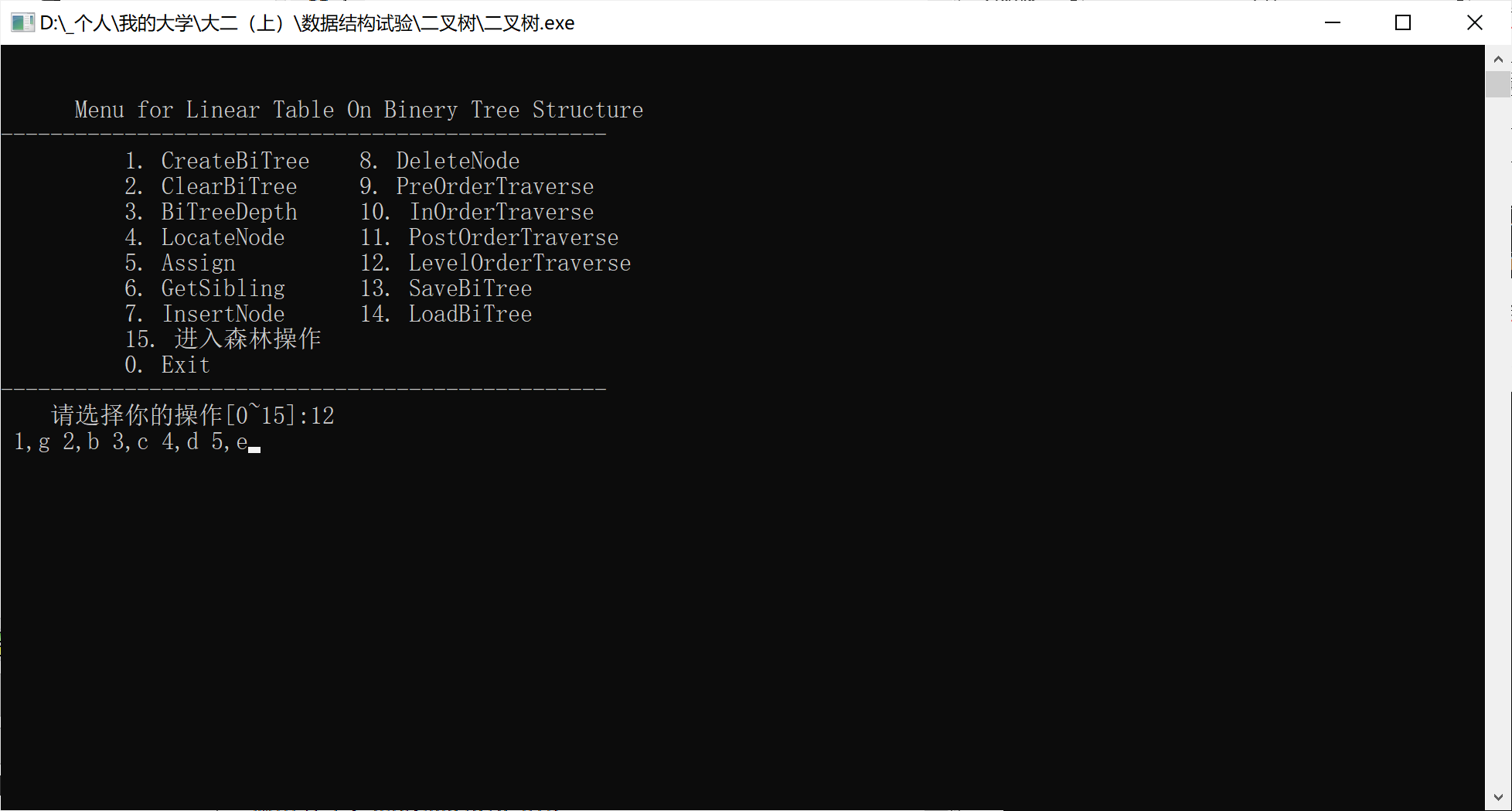


图2-13 LevelOrderTraverse测试1

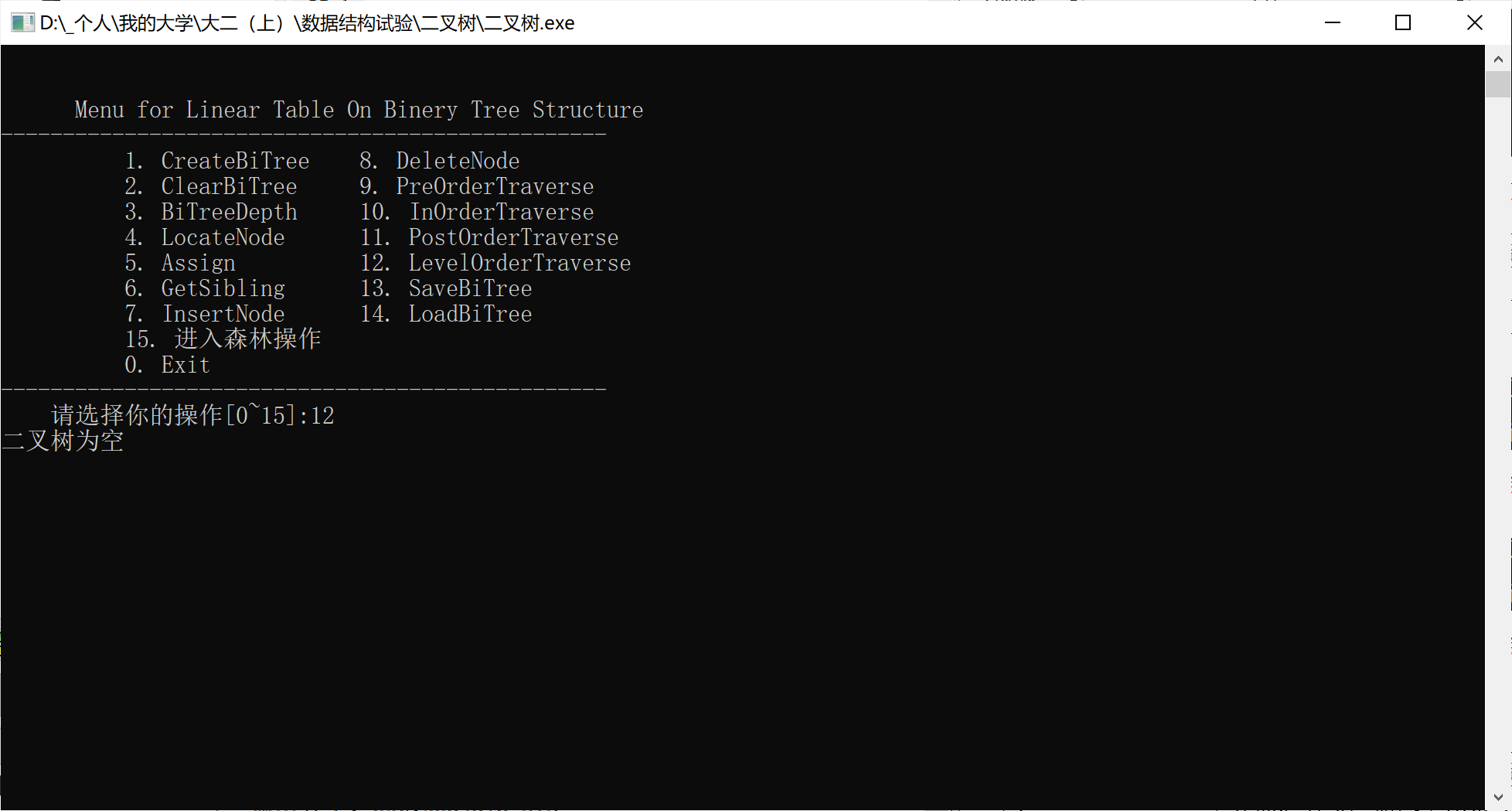


图2-14 LevelOrderTraverse测试2

多线性表：

(5)AddTree测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| 多二叉树tree1，tree2 | 界面选择15，界面选择1，输入tree1；  1 1 a 2 2 b 3 3 c 0 0 null；  tree2；  1 1 g 2 2 b 3 3 c 0 0 null | 创建成功  多二叉树如下：  tree1 1,a 2,b 3,c  tree2 1,g 2,b 3,c | 创建成功  多二叉树如下：  tree1 1,a 2,b 3,c  tree2 1,g 2,b 3,c |

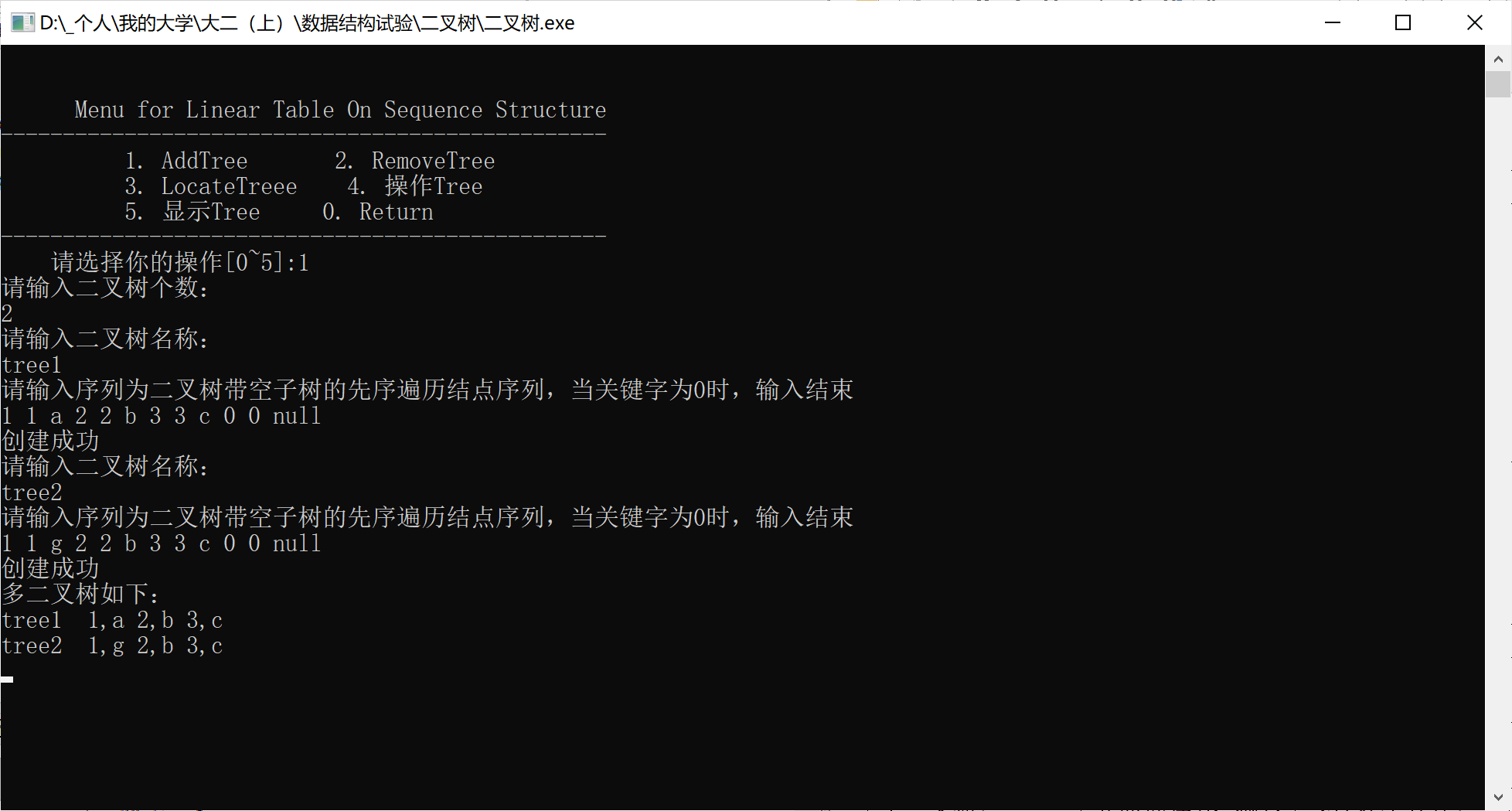


图2-15 AddTree测试1

（6）LocateTree测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 实际结果 |
| 多二叉树tree1，tree2 | 界面选择3,输入tree1 | 查找的二叉树如下：  tree1 1,a 2,b 3,c | 查找的二叉树如下：  tree1 1,a 2,b 3,c |
| 多二叉树tree1,tree2 | 界面选择3,输入tree2 | 查找的二叉树如下：  tree2 1,g 2,b 3,c | 查找的二叉树如下：  tree2 1,g 2,b 3,c |
| 多二叉树tree1,tree2 | 界面选择3,输入tree3 | 查找失败 | 查找失败 |

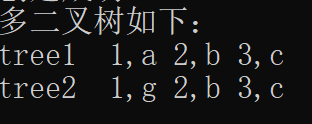


图2-16 LocateTree测试集展示

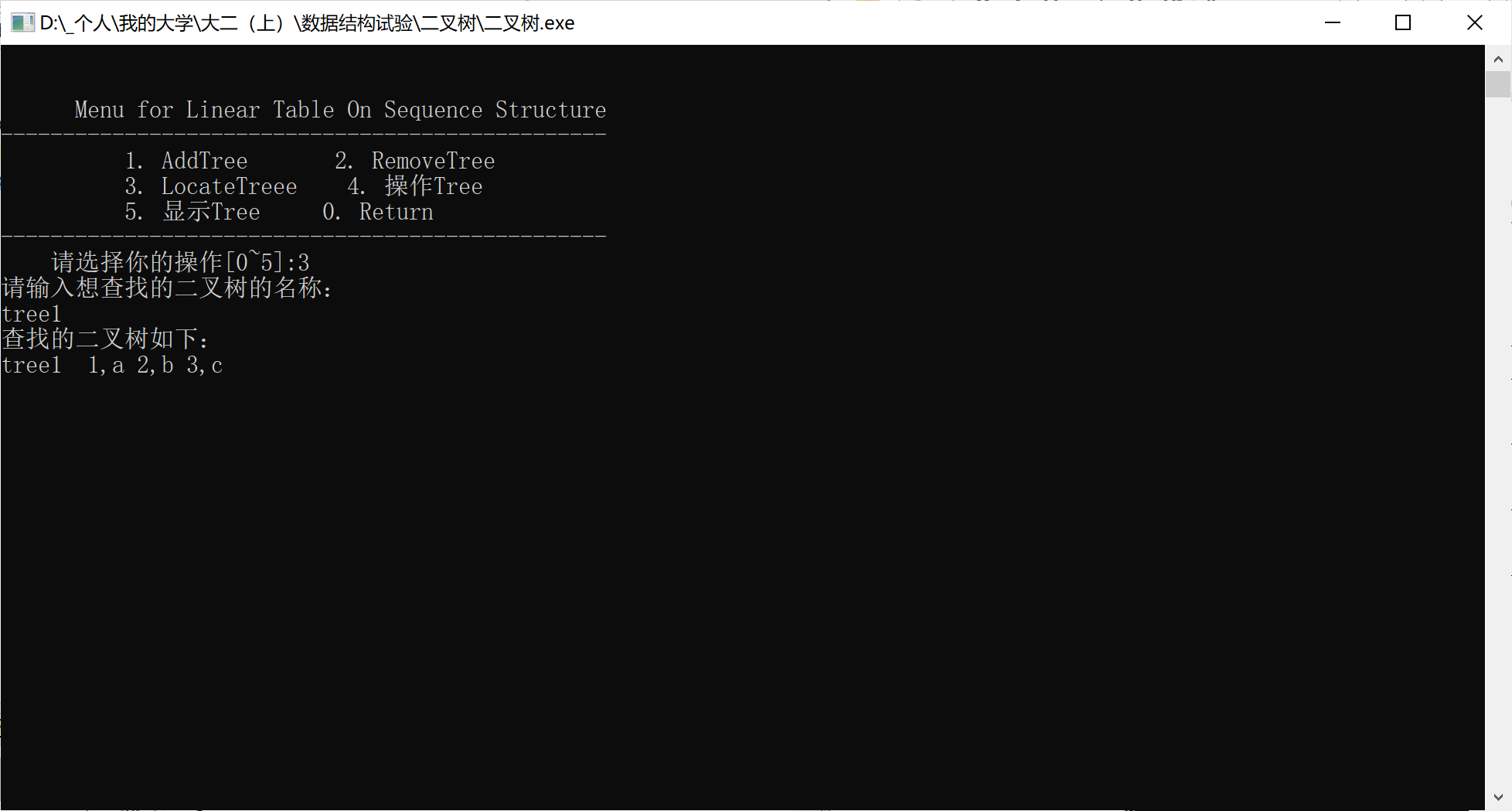


图2-17 LocateTree测试1

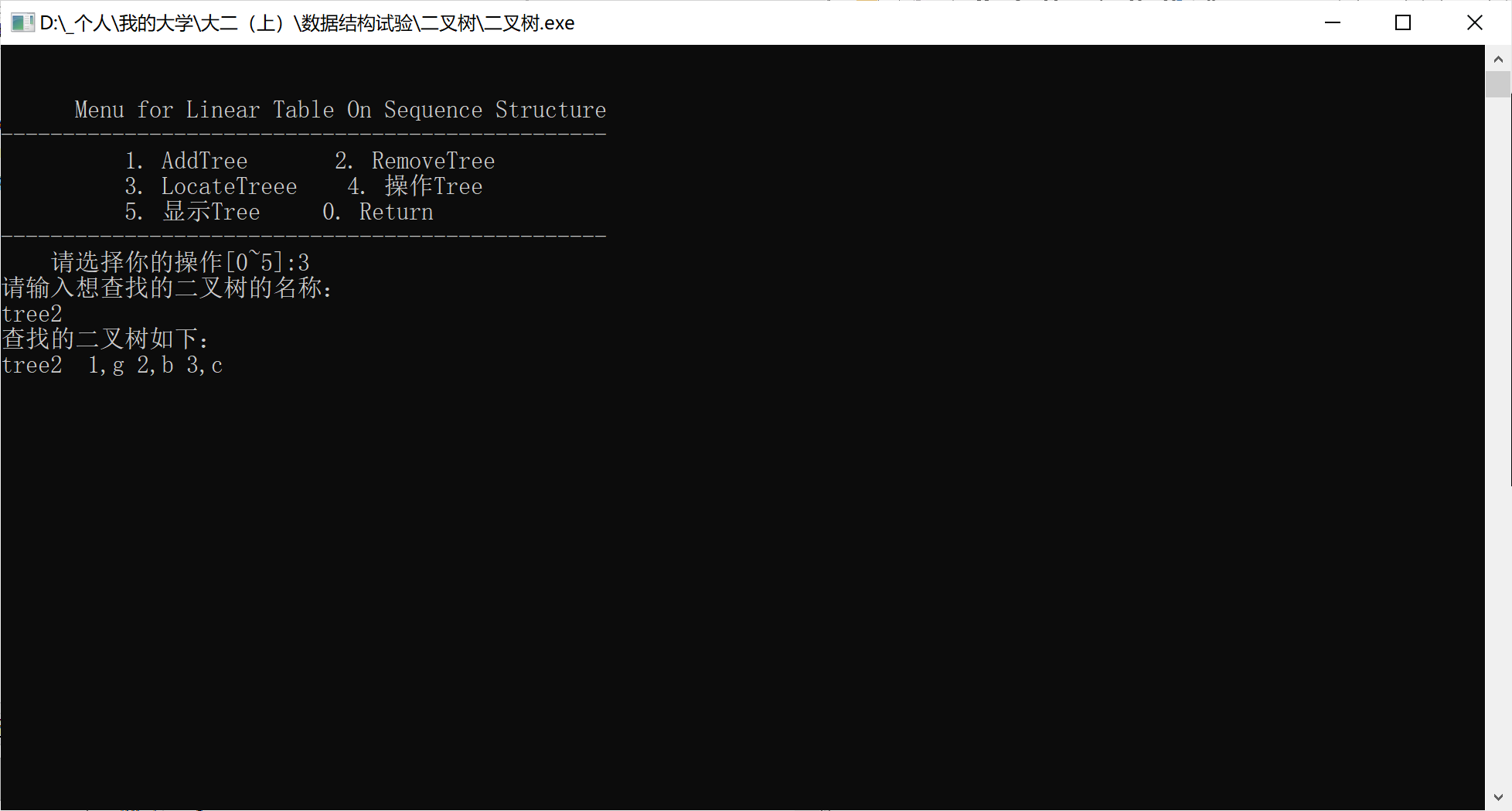


图2-18 LocateTree测试2

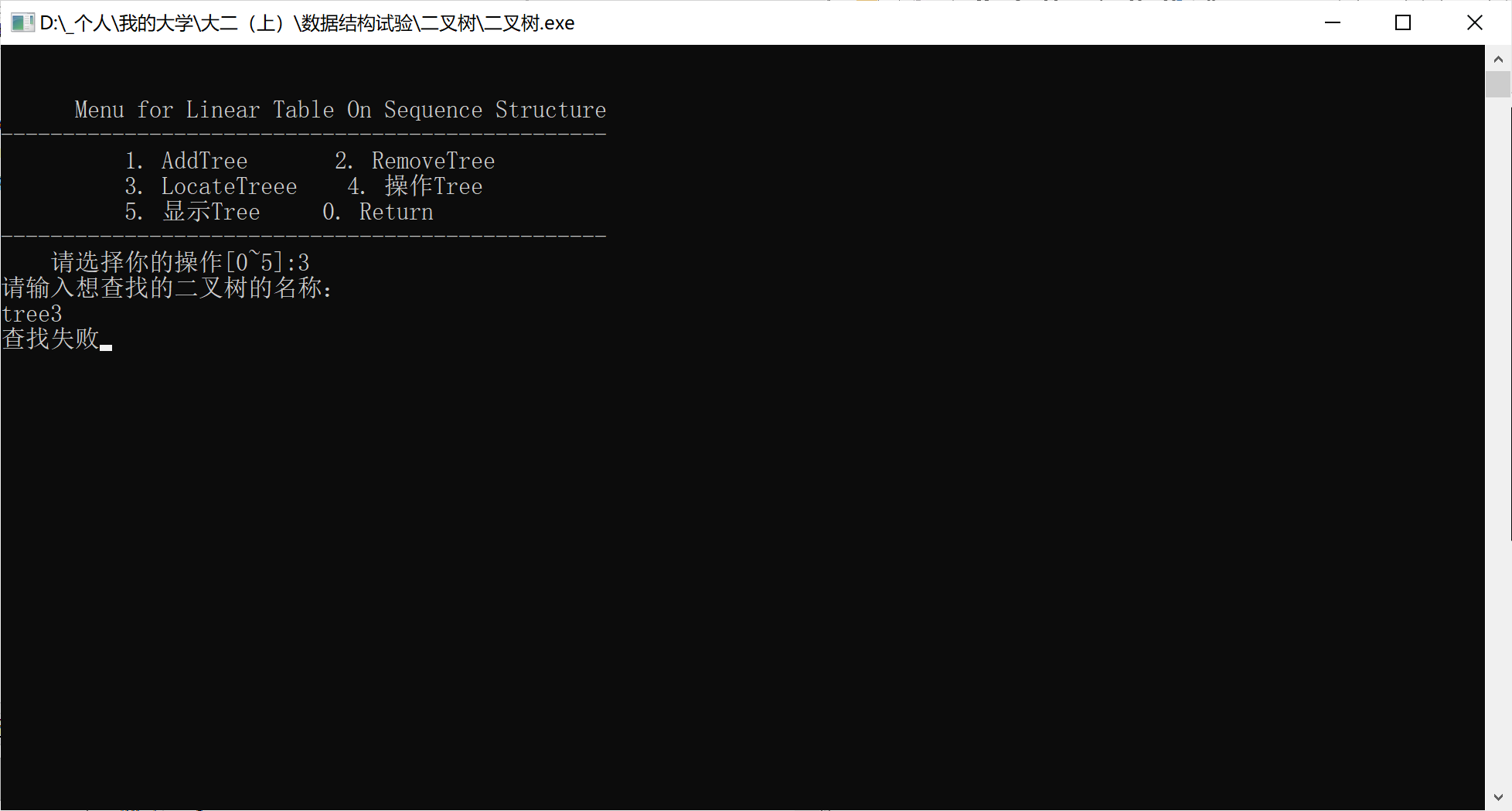


图2-19 LocateTree测试3

## 3.5 实验小结

这次的实验相较之于之前的两次实验难度有了明显的提高，主要在于删除结点和还有就是创建二叉树是对结点进行赋值并且保存好元素之间的关系，以及使用非递归算法进行二叉树的后续遍历。

前两次的线性表数据结构是线性结构，所以元素之间的关系表示起来就会简单一些，但是二叉树是非线性的结构，所以元素之间的关系会更加复杂，构建元素的物理结构和储存元素内容和关系就会比较复杂。

通过这次的实验，我对二叉树这一数据结构有了更深的理解和认识，知道了如何创建一棵二叉树，如何销毁，如何增加、删除、查找、修改结点，其中删除结点在写代码的时候思考的比较久，因为这其中包含了多种情况，并且要写额外的函数实现一些辅助的功能。还有就是实现二叉树的遍历，包括前序遍历，中序遍历和后续遍历，如果是使用递归的方法，那么实现起来会比较容易，这体现出了递归这一方法的优越性，如果是使用非递归方法，那么就要使用栈去操作，需要写栈相关的函数进行辅助，前序遍历和中序遍历会相对来说比较简答，但是后序遍历要多判断是第几次访问结点。

这次的二叉树实验涉及到了栈、队列等结构，是对之前学习的内容的一次综合的考验，在线上函数实现和线下系统组装中也有遇到一些问题，思考解决方法时会花费较多的时间，但是在解决之后收获颇丰，受益良多。

# 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

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

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

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

# 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