

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级：计算机科学与技术201806**

**学 号： U201814670**

**姓 名： 李田田**

**指导教师： 李剑军**

**报告日期： 2019年 12月 24 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](#_Toc28598554)

[1.1 问题描述 2](#_Toc28598555)

[1.2 系统设计 4](#_Toc28598556)

[1.3 系统实现 8](#_Toc28598557)

[1.4 系统测试 11](#_Toc28598558)

[1.5 实验小结 24](#_Toc28598559)

[2 基于链式存储结构的线性表实现 26](#_Toc28598560)

[2.1 问题描述 26](#_Toc28598561)

[2.2 系统设计 27](#_Toc28598562)

[2.3 系统实现 32](#_Toc28598563)

[2.4 系统测试 35](#_Toc28598564)

[2.5 实验小结 48](#_Toc28598565)

[3 基于二叉链表的二叉树实现 50](#_Toc28598566)

[3.1 问题描述 50](#_Toc28598567)

[3.2 系统设计 52](#_Toc28598568)

[3.3 系统实现 54](#_Toc28598569)

[3.4 系统测试 61](#_Toc28598570)

[3.5 实验小结 72](#_Toc28598571)

[4 基于邻接表的图实现 73](#_Toc28598572)

[4.1 问题描述 73](#_Toc28598573)

[4.2 系统设计 75](#_Toc28598574)

[4.3 系统实现 78](#_Toc28598575)

[4.4 系统测试 84](#_Toc28598576)

[4.5 实验小结 97](#_Toc28598577)

[参考文献 98](#_Toc28598578)

[附录A 基于顺序存储结构线性表实现的源程序 99](#_Toc28598579)

[附录B 基于链式存储结构线性表实现的源程序 109](#_Toc28598580)

[附录C 基于二叉链表的二叉树实现的源程序 120](#_Toc28598581)

[附录D 基于邻接表的图实现的源程序 136](#_Toc28598582)

# 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，并且能对多个线性表进行选择。在主函数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。

定义了线性表的创建、销毁、清空、判定空表、求表长、获得元素、查找元素、获得前驱、获得后继、插入元素、删除元素、遍历的函数，并给出输入的操作提示，以文件形式进行存储和加载，将生成的线性表存入到相应的文件，也可以从文件中获取线性表进行操作。

1.1.1 基本运算

1. 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。
2. 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，并释放线性表的空间。
3. 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。
4. 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NO。
5. 求表长：函数名称是ListLength(L)；初始条件是线性表已存在，直接返回L->length的值；操作结果是返回L中数据元素的个数。
6. 获得元素：函数名称是GetElem(L,i,\*e)；初始条件是线性表已存在，i满足1≤i≤ListLength(L)，若i非法则返回ERROR；操作结果是用e返回L中第i个数据元素的值。
7. 查找元素：函数名称是LocateElem(L,e)；初始条件是线性表已存在；操作结果是返回L中第1个与e相等的数据元素的位序，若这样的数据元素不存在，则返回值为ERROR。
8. 获得前驱：函数名称是PriorElem(L,cur,\*pre\_e)；初始条件是线性表L已存在，通过LocateElem(L,e)函数查找元素cur；操作结果是若cur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。
9. 获得后继：函数名称是NextElem(L,cur,\*next\_e)；初始条件是线性表L已存在，通过LocateElem(L,e)函数查找元素cur；操作结果是若cur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。
10. 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，i满足1≤i≤ListLength(L)+1，若i非法，则返回ERROR；操作结果是在L的第i个位置之前插入新的数据元素e。
11. 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，i满足1≤i≤ListLength(L)，若i非法，则返回ERROR；操作结果：删除L的第i个数据元素，用e返回其值。
12. 遍历表：函数名称是ListTraverse(L)，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输出。
13. 写入文件：函数名称是Write(L)，初始条件是线性表L已存在；输入文件名，将文件打开并写入；操作结果是将L的数据依次写入到文件里。
14. 读取文件：函数名称是Read(\*L)，初始条件是线性表L已存在；输入文件名，将文件打开并读取；操作结果是将L的数据依次读取到线性表里。

1.1.2 演示系统

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。

## 1.2 系统设计

1.2.1 头文件、常量及结构说明

1. 头文件

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

1. 预定义常量

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define NO -1

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

1. 类型表达式

typedef int status;

typedef int ElemType; //数据元素类型定义

1. 定义数据的物理结构如下：

typedef struct{ //顺序表（顺序结构）的定义

ElemType \*elem;

int length;

int listsize;

}SqList;

同时因为需要实现多线性表的操作，构造一个结构数组：SqList M[100];

1.2.2 算法设计

1. 初始化表：传入参数结构型变量L的地址，在函数中，首先使用malloc函数分配LISTSIZE大小的连续内存空间，并将首地址返回赋值给L.elem，将L.length初始化为0。
2. 销毁表：传入参数结构体变量L的地址，首先使用free函数释放掉以L.elem为首地址的连续内存空间，L.elem为NULL，再将L.length，L.listsize重新赋值为0。
3. 清空表：传入参数结构体变量L的地址，因为只需要清空操作并不释放内存空间，故直接将线性表的长度置为0。
4. 判定空表：传入参数结构体变量L，直接判断L.length是否为0。
5. 求表长：传入参数结构体变量L，直接返回L.length即可。
6. 获得元素：传入参数结构体变量L，抽象数据e的地址和元素位置i，因为是顺序线性表，可直接访问数组L.elem[i-1]，随后返回该值。
7. 查找元素：传入参数结构体变量L，需要查找的元素e。从头查找每个元素是否与该元素相等，返回第一个相同的位置。
8. 获得前驱： 传入参数结构体变量L，查找前驱的元素cur和前驱指针pre\_e。首先调用LocateElem函数判断该元素在线性表中是否存在，若存在返回元素位置。再判断元素位置是否为第1个或者LocateElem函数的返回值为ERROR，如都不符合则返回其前驱元素L.elem[i-2]。
9. 获得后继：传入参数结构体变量L，查找前驱的元素cur和前驱指针next\_e。首先调用LocateElem函数判断该元素在线性表中是否存在，若存在返回元素位置。再判断元素位置是否为最后1个或者LocateElem函数的返回值为ERROR，如都不符合则返回其后继元素L.elem[i]。
10. 插入元素：传入参数结构体变量L，需要插入的数据e和元素位置i。具体如流程图所示。



图1-1 插入元素流程图

1. 删除元素：传入参数结构体变量L，需要删除的元素位置i和抽象数据e的地址。具体如流程图所示。



图1-2 删除元素流程图

1. 遍历表：传入参数结构体变量L，直接用一个循环来对线性表中的每一个元素进行操作。

1.2.3 理论分析

表1-1 函数算法复杂度分析

|  |  |
| --- | --- |
| 测试功能功能及序号 | 算法复杂度 |
| 1. InitList | O(1) |
| 2. DestroyList | O(1) |
| 3. ClearList | O(1) |
| 4. ListEmpty | O(1) |
| 5. ListLength | O(1) |
| 6. GetElem | O(1) |
| 7. LocateElem | O(n) |
| 8. PriorElem | O(n) |
| 9. NextElem | O(n) |
| 10. ListInsert | O(n) |
| 11. ListDelete | O(n) |
| 12. ListTrabverse | O(n) |

## 1.3 系统实现

1.3.1 系统演示

整个系统的流程及结构如下图所示。



图1-3 系统结构图

1.3.2 部分函数功能实现

1. 在获得前驱和后继时，通过调用LocateElem(L,cur)函数还判断线性表中是否有该元素，并令i等于该函数返回值，随后再判断i是否合法以保证前驱和后继的正确性。

获得前驱：

i=LocateElem(L,cur);//找到cur的位置

if(i==1||i==ERROR)

return ERROR;

获得后继：

i=LocateElem(L,cur);

if(i==L.length||i==ERROR)

return ERROR;

1. 插入元素：根据之前的算法设计流程图实现操作。首先判定要插入的位置i的合法性，若不合法，返回ERROR，若合法，则判断空间是否足够，不足够则进行连续空间的重新分配，足够则直接进行插入新结点的操作，插入位置之后的元素从表尾依次进行右移。最后顺序表的长度增加1，返回OK。

status ListInsert(SqList \*L,int i,ElemType e){//在顺序线性表中第i个元素之前插入新的元素e，1<=i<=listlenth+1

ElemType \*newbase;

ElemType \*q,\*p;

ElemType t;

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

return ERROR;//i不合法

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

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

if(!newbase)

exit(OVERFLOW);//存储分配失败

L->elem=newbase;//新基址

L->listsize+=LISTINCREMENT;

}

q=&(L->elem[i-1]);

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

\*(p+1)=\*p;//插入位置及之后的元素右移

L->elem[i-1]=e;

++(L->length);

return OK;

}

1. 删除元素：根据之前的算法设计流程图实现操作。首先判定要插入的位置i的合法性，若不合法，返回ERROR，若合法，进行删除结点的操作，插入位置之后的元素依次进行左移。最后顺序表的长度减1，返回OK。

status ListDelete(SqList \*L,int i,ElemType \*e){//在顺序线性表中删除第i个元素，并用e返回其值

ElemType \*q,\*p;

if(i<1||(i>L->length))

return ERROR;//i不合法

e=L->elem[i-1];//被删除元素值赋给e

p=L->elem+L->length-1;//令p为表尾位置

for(q=&(L->elem[i]);q<=p;q++)

\*(q-1)=\*q;//被删元素之后的元素依次前移

--(L->length);

return OK;

}

1.3.3 用文件读写

文件的读取可用多个线性表进行操作，例如用线性表1写入文件，再创建线性表2读取数据，使之与线性表1 相同。

1. 文件的写入因为不要求打开文件能直接看到数据，而是用另外一个线性表直接读取，故编写代码时直接用fwrite进行二进制写入，而不采用fprintf函数(该函数写入可直接看到数据)。

写入文件：

status Wirte(SqList L) {

FILE \*fp=NULL;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

if ((fp=fopen(filename,"wb"))==NULL){

printf("File open error\n ");

return FALSE;

}

//写文件的方法

fwrite(L.elem,sizeof(ElemType),L.length,fp);

//这里是1次性写入，对于其它物理结构，

//也可以先写入表长，再写入全部元素,这样读入会更方便

fclose(fp);

return OK;

}

1. 文件的读取因为是顺序表，所以直接while循环即可将数据从文件中保存到顺序表中。

读取文件：

status Read(SqList \*L) {

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

//读文件的方法

L->length=0;

if ((fp=fopen(filename,"rb"))==NULL){

printf("File open error\n ");

return FALSE;

}

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

L->length++;

//这里从文件中逐个读取数据元素恢复顺序表，对于不同的物理结构，可通过读取的数据元素恢复内存中的物理结构。

fclose(fp);

return OK;

}

## 1.4 系统测试

1.4.1 测试计划

表1-2 测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能功能及序号 | 测试的线性表 | 输入的参数 | 预计结果 | 线性表实际状态 |
| 选择线性表 | 1 | 1 | 对线性表1进行操作 | 选择线性表1，并未创建 |
| 1. InitList | 1 | 无 | 创建线性表1 | 分配了连续的空间，表长为0 |
| 10. ListInsert（多次调用） | 1 | 1 1；2 2；3 3；4 5 | 线性表1 写入数据 | 线性表序列为：1，2，3，5 |
| 4. ListEmpty | 1 | 无 | 线性表非空 | 线性表序列为：1，2，3，5 |
| 5. ListLength | 1 | 无 | 线性表长度 | 线性表序列为：1，2，3，5 |
| 6. GetElem | 1 | 想要获得的元素位置：3 | 第3个位置的元素 | 线性表序列为：1，2，3，5 |
| 7. LocateElem | 1 | 需要查找的元素：2 | 与该元素相同的元素的位置 | 线性表序列为：1，2，3，5 |
| 8. PriorElem | 1 | 需要获得前驱的元素：3 | 该元素的前驱 | 线性表序列为：1，2，3，5 |
| 9. NextElem | 1 | 需要获得后继的元素：3 | 该元素后继 | 线性表序列为：1，2，3，5 |
| 10. ListInsert | 1 | 4 4； | 在第四个位置插入元素4 | 线性表序列为：1，2，3，4，5 |
| 11. ListDelete | 1 | 删除元素的位置：5 | 删除第五个位置的元素 | 线性表序列为：1，2，3，4 |
| 12. ListTrabverse | 1 | 无 | 遍历线性表1 | 线性表序列为：1，2，3，4 |
| 13. Write | 1 | 文件名：data | 将线性表1写入文件 | 线性表序列为：1，2，3，4 |
| 3. ClearList | 1 | 无 | 清空线性表1 | 表长为0 |
| 2. DestroyList | 1 | 无 | 销毁线性表1 | 线性表连续空间被释放 |
| 0. Exit | 1 | 无 | 退出操作 | 无线性表 |
| 选择线性表 | 2 | 2 | 对线性表2进行操作 | 选择线性表2，并未创建 |
| 1. InitList | 2 | 无 | 创建线性表2 | 分配了连续的空间，表长为0 |
| 14. Read | 2 | 文件名：data | 从该文件读入数据 | 线性表2序列：1，2，3，4 |
| 12. ListTrabverse | 2 | 无 | 遍历线性表2 | 线性表2序列：1，2，3，4 |
| 0. Exit | 2 | 无 | 退出操作 | 无线性表 |
| 退出选择线性表 | 无 | 退出输入0 | 退出选择 | 运行结束 |

1.4.2 测试结果

执行1.4.1表格的测试计划，运行结果截图如下所示。

1. 选择线性表1，进入操作。

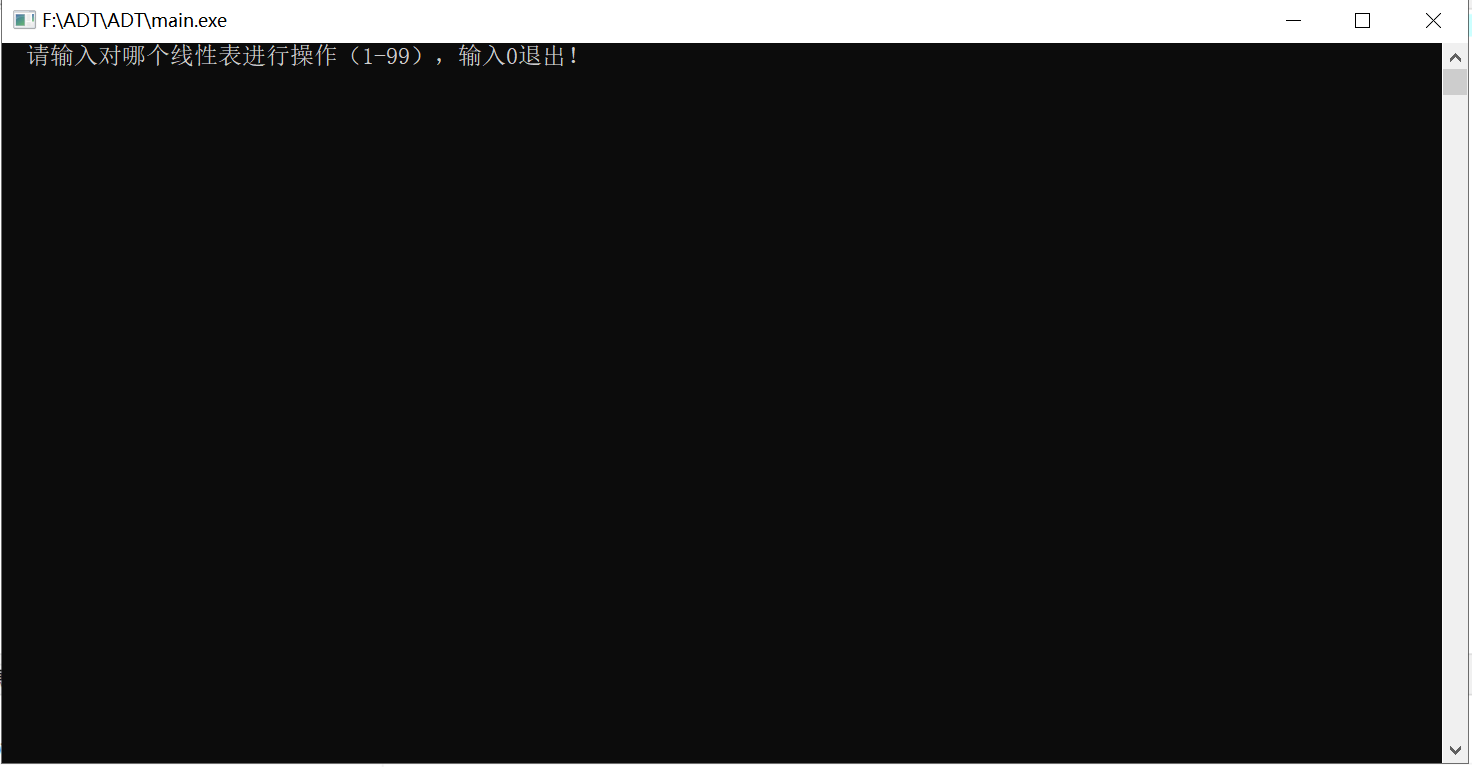


图1-4 测试-选择线性表

1. 执行功能1，创建线性表1。

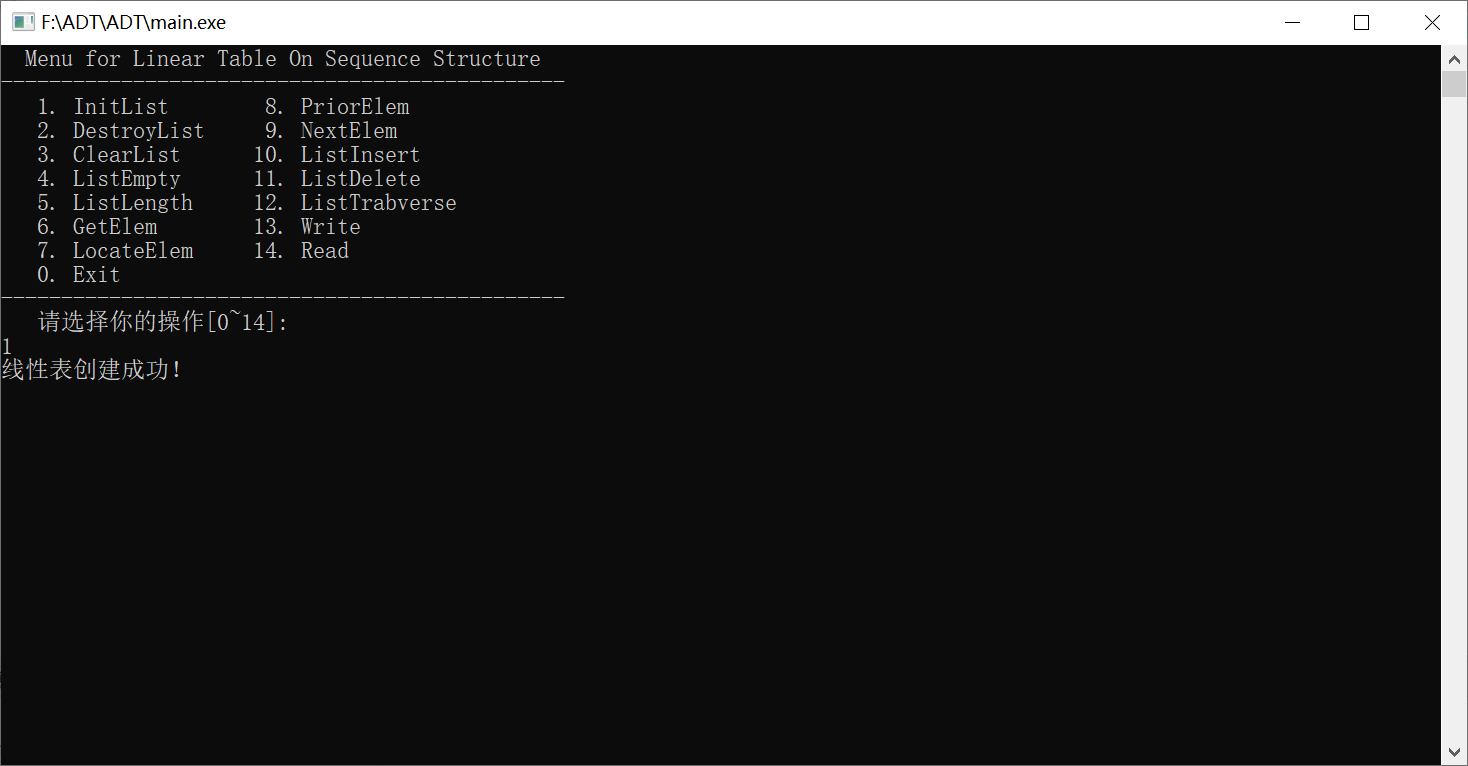


图1-5 测试-1创建线性表

1. 多次执行功能10，向线性表1添加元素1、2、3、5。

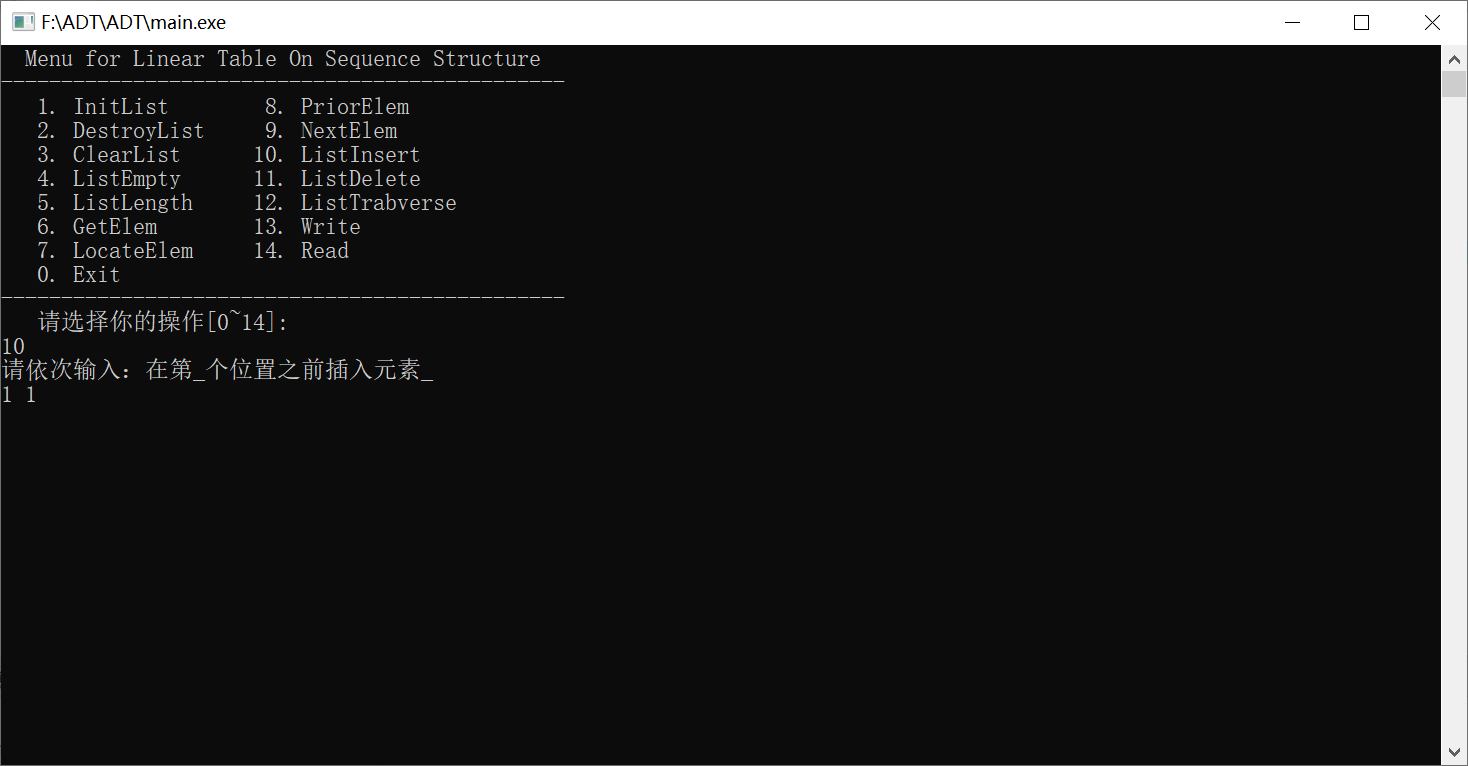


图1-6 测试-10插入元素

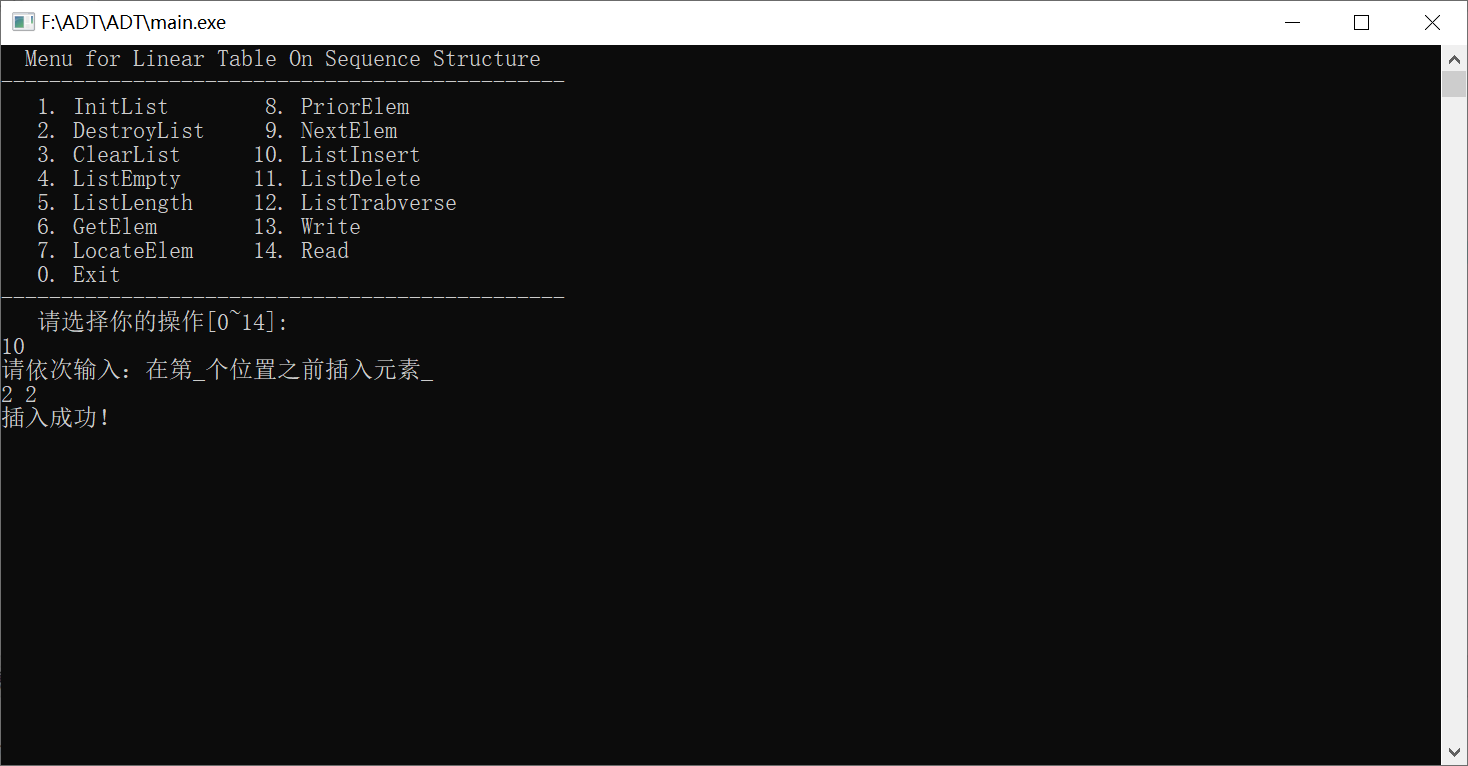


图1-7 测试-10插入元素

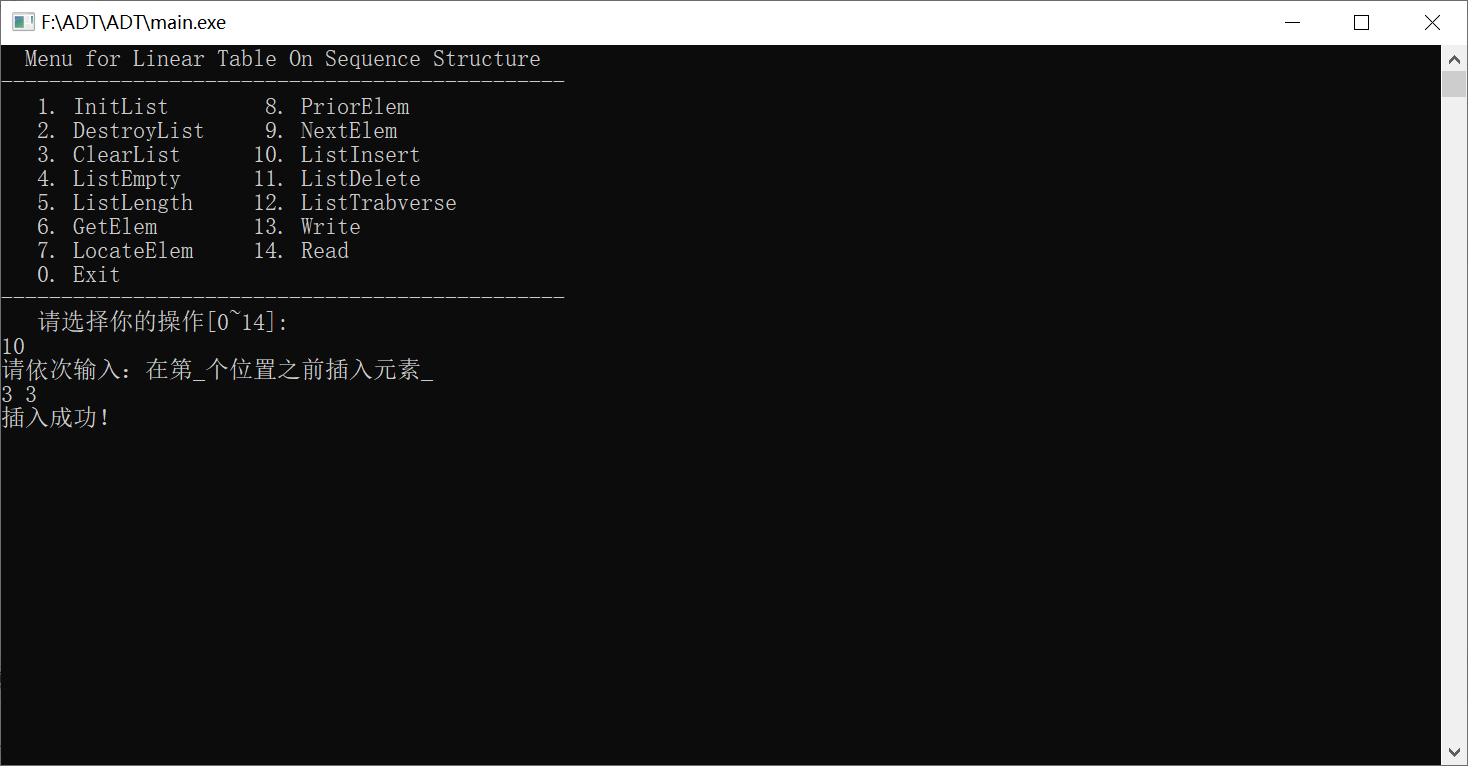


图1-8 测试-10插入元素

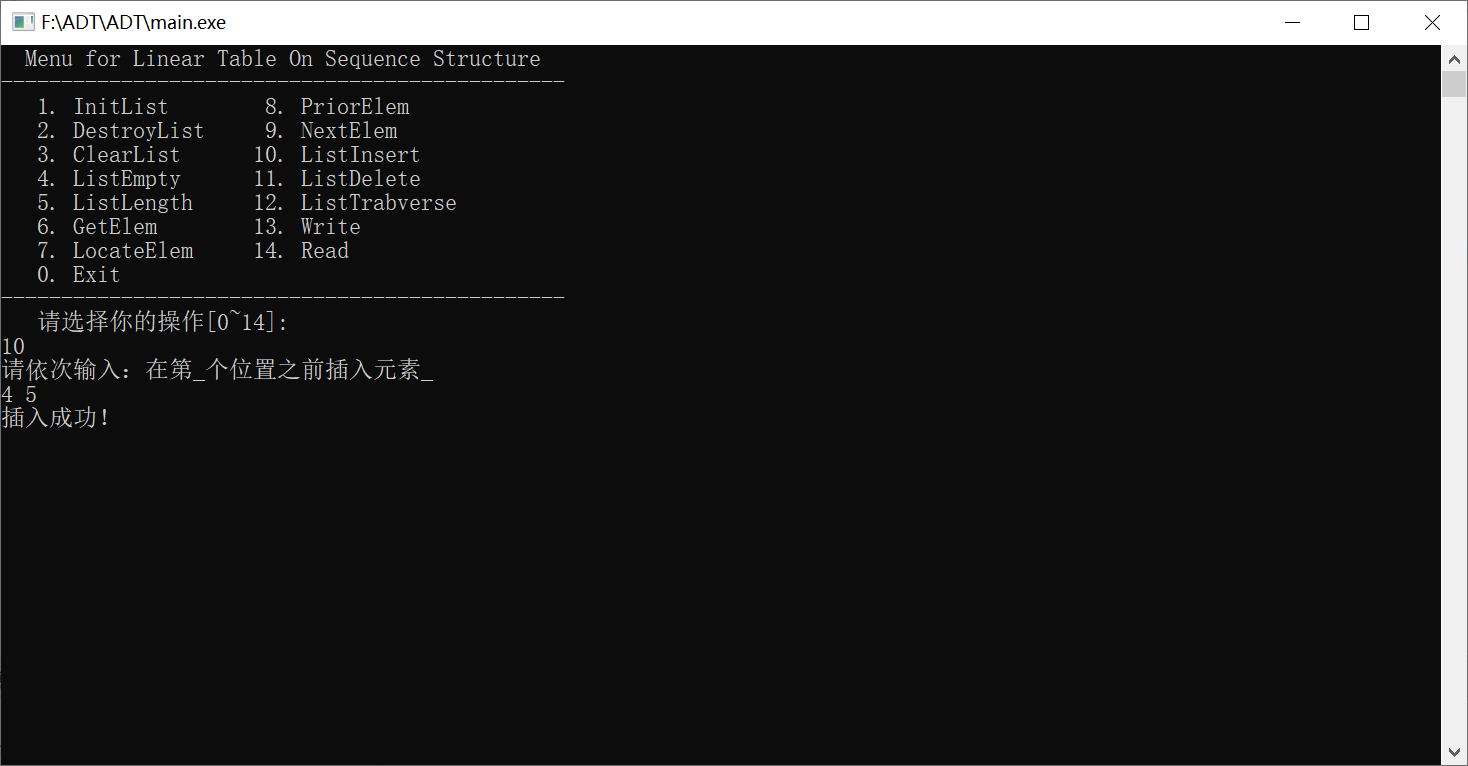


图1-9 测试-10插入元素

1. 执行功能4，判断线性表是否为空。

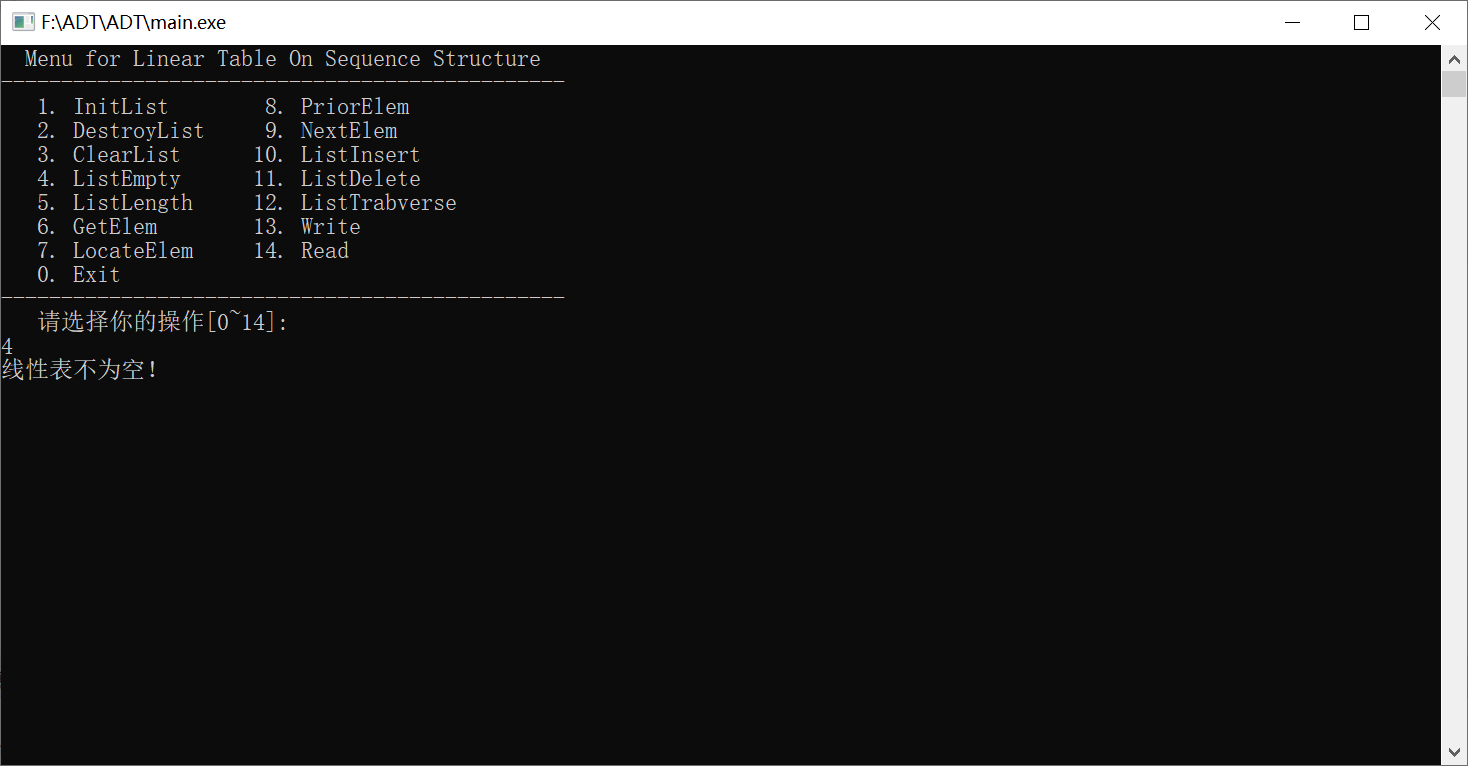


图1-10 测试-4判断线性表非空

1. 执行功能5，求线性表表长。

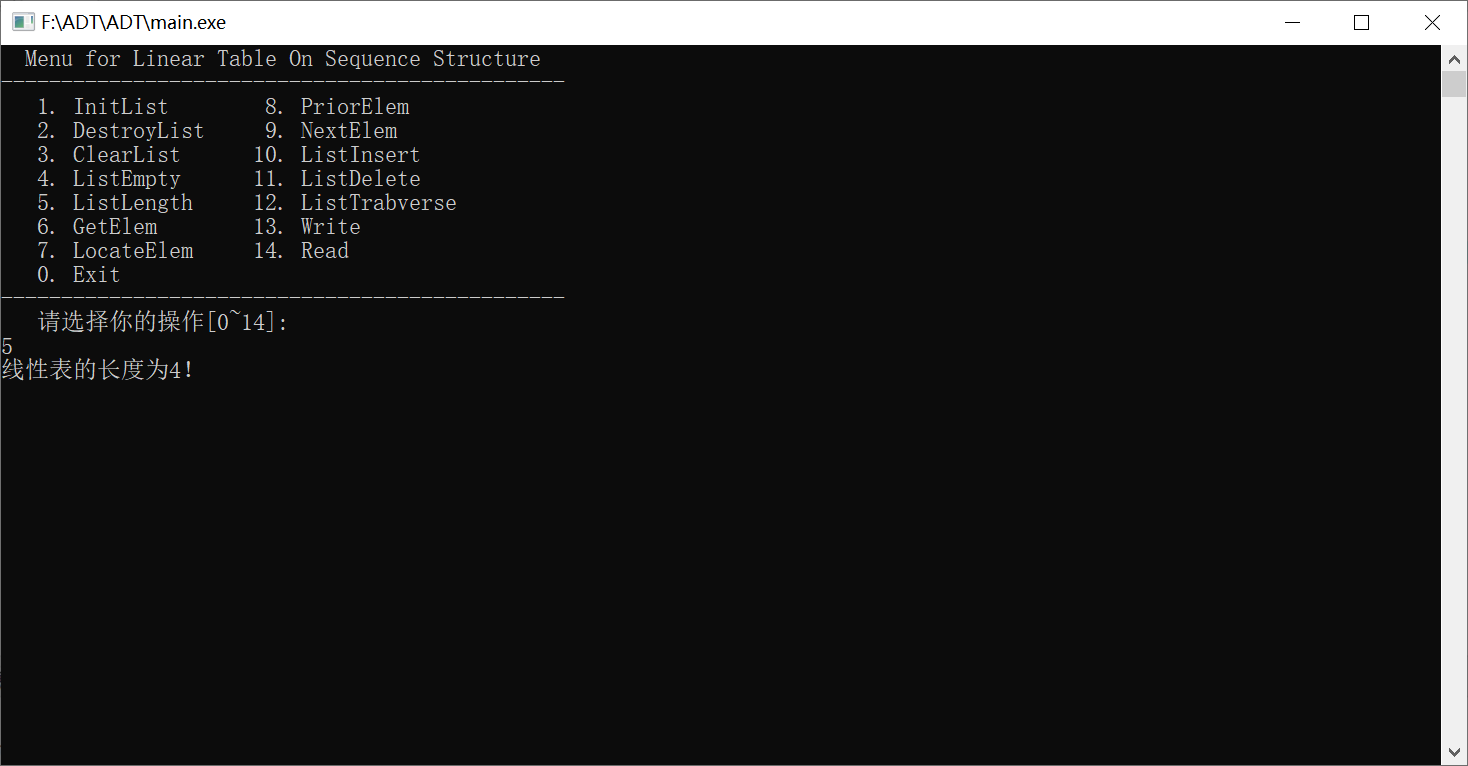


图1-11 测试-5求表长

1. 执行功能6，查找第3个元素的值。

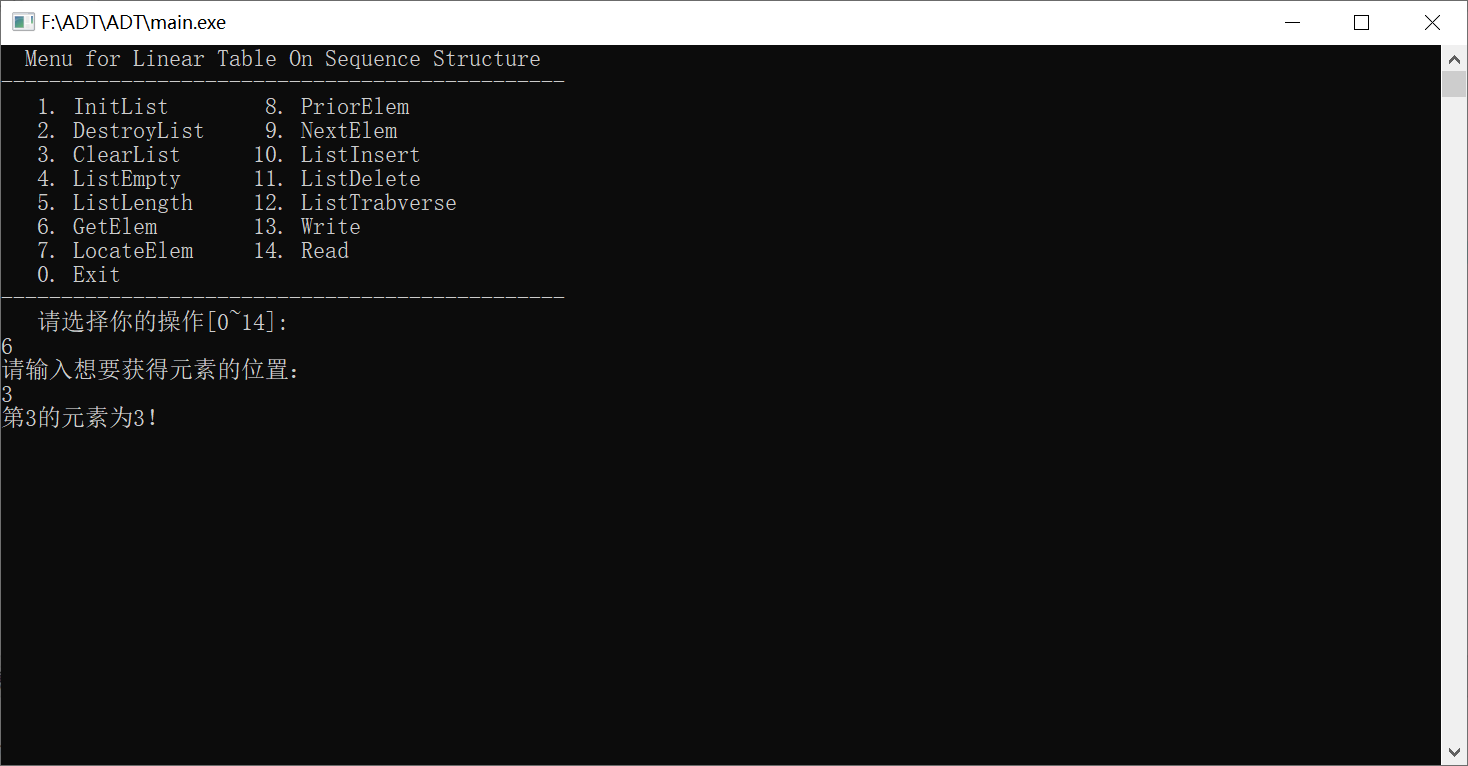


图1-12 测试-6获得元素

1. 执行功能7，查找元素2。

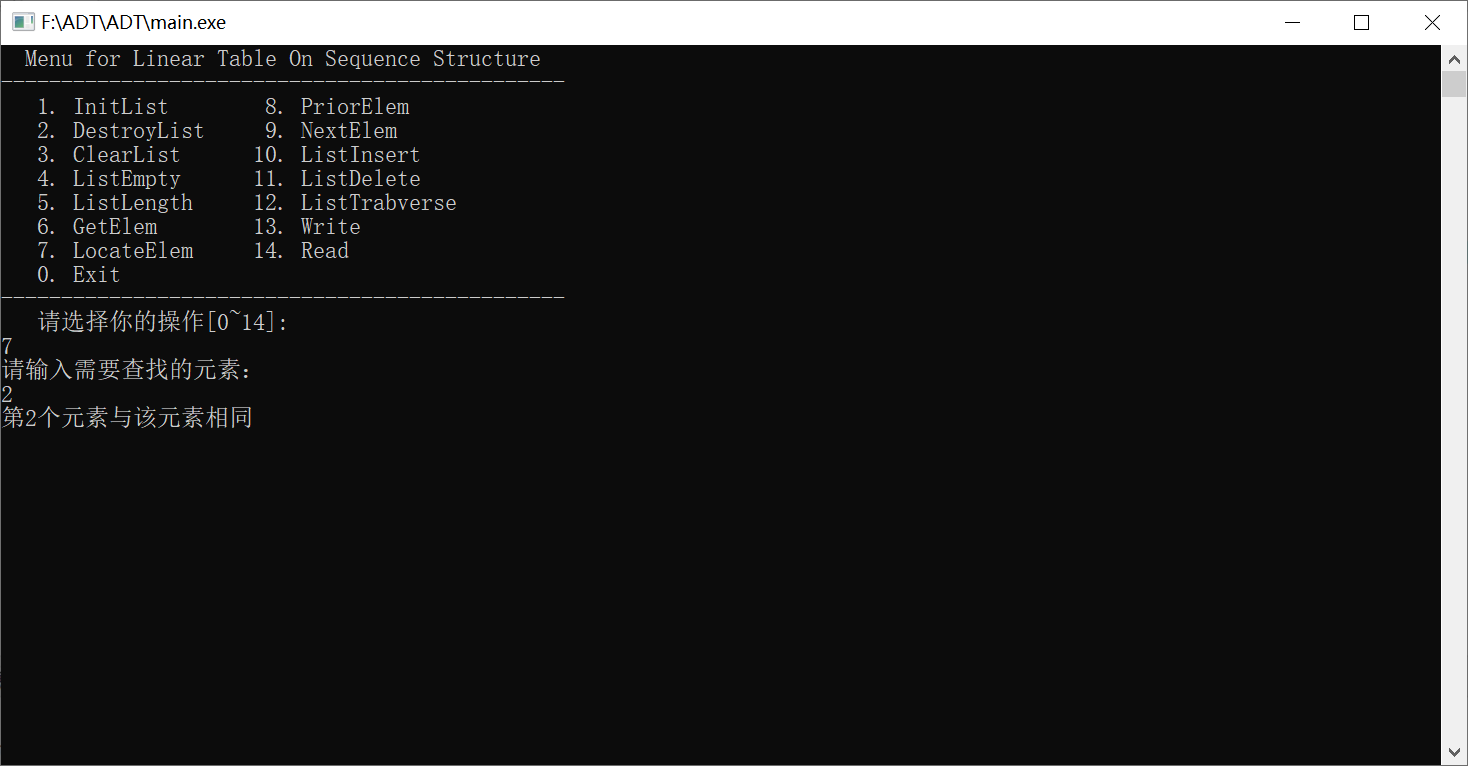


图1-13 测试-7查找元素

1. 执行功能8，获得3的前驱。

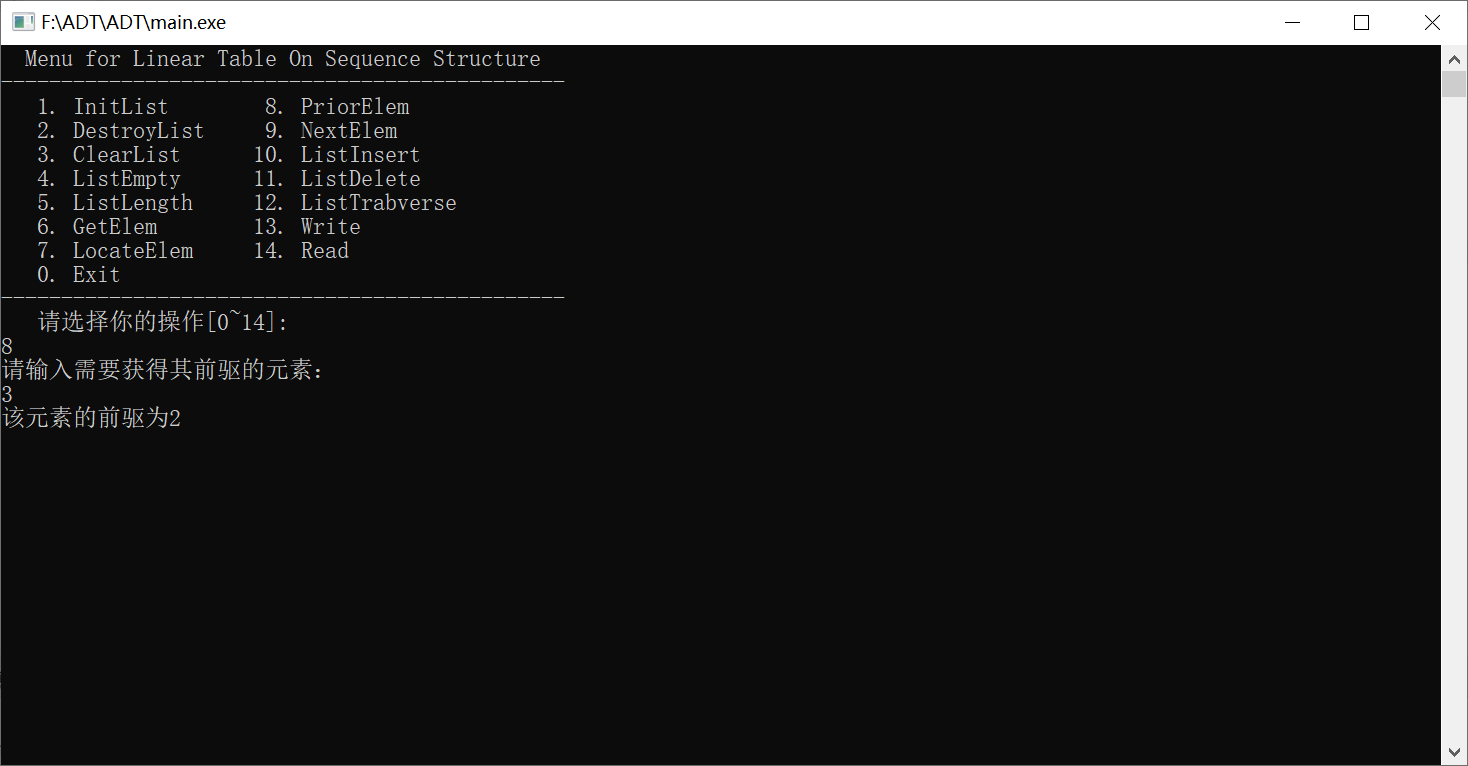


图1-14 测试-8获得前驱

1. 执行功能9，获得3的后继。

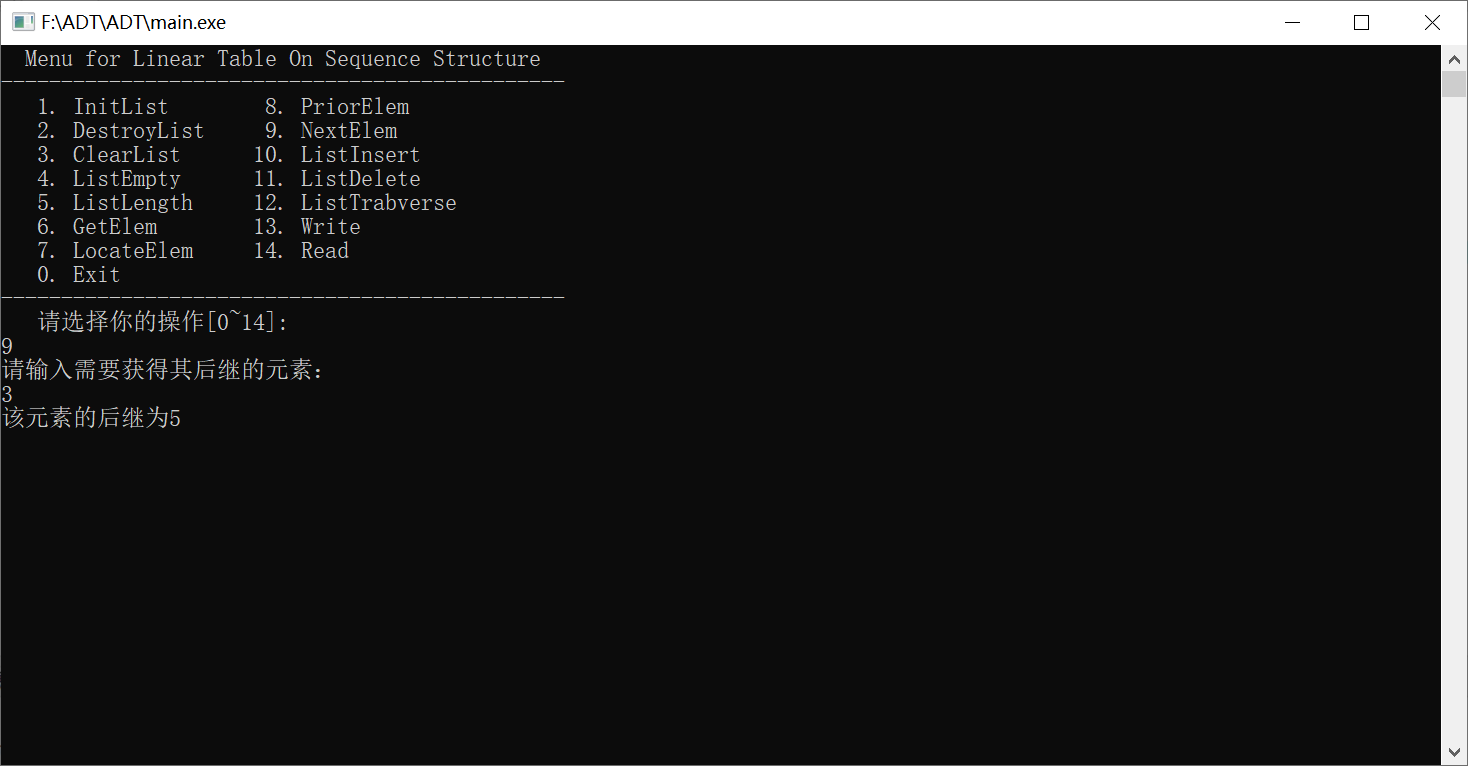


图1-15 测试-9获得后继

1. 执行功能10，在第四个位置插入元素4。

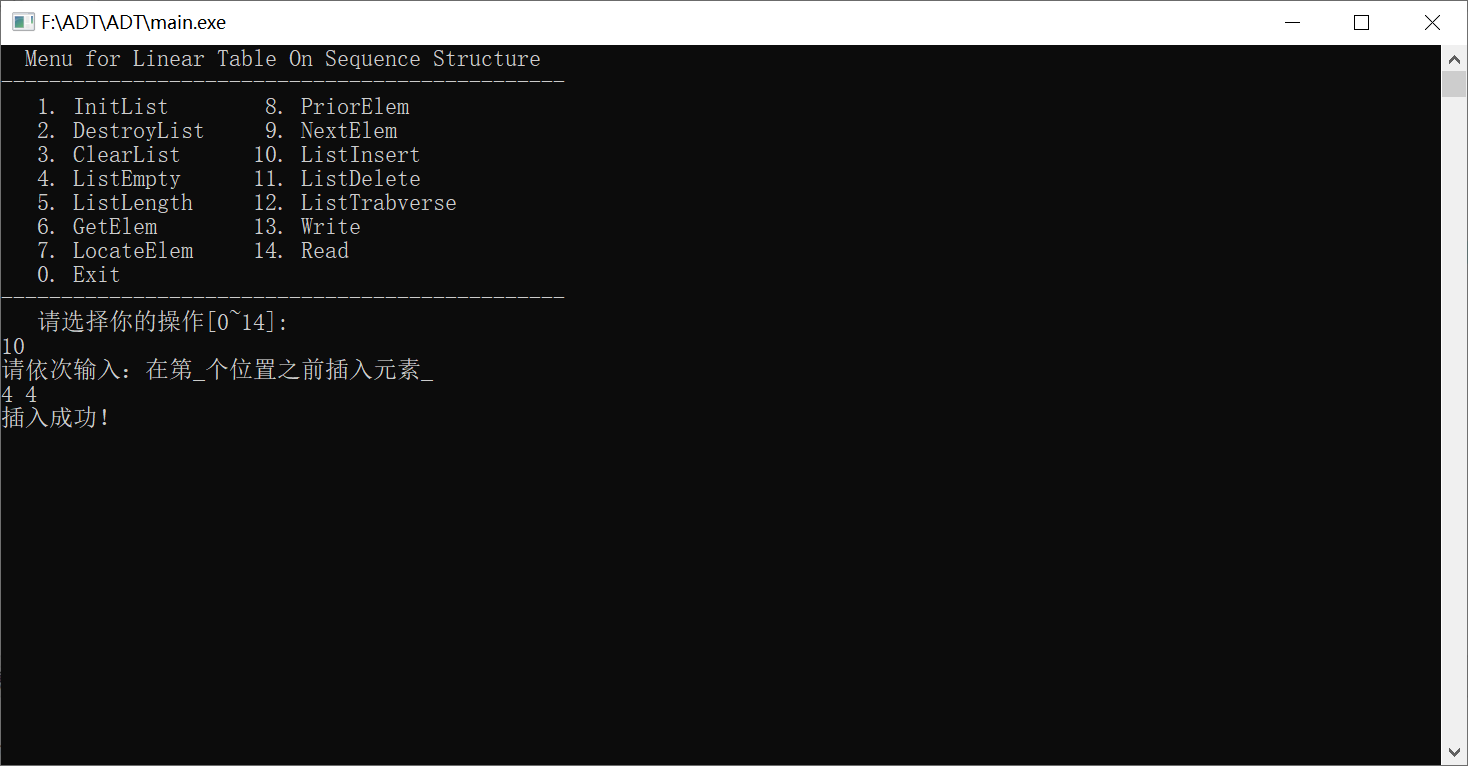


图1-16 测试-10插入元素

1. 执行功能11，删除第5个元素。

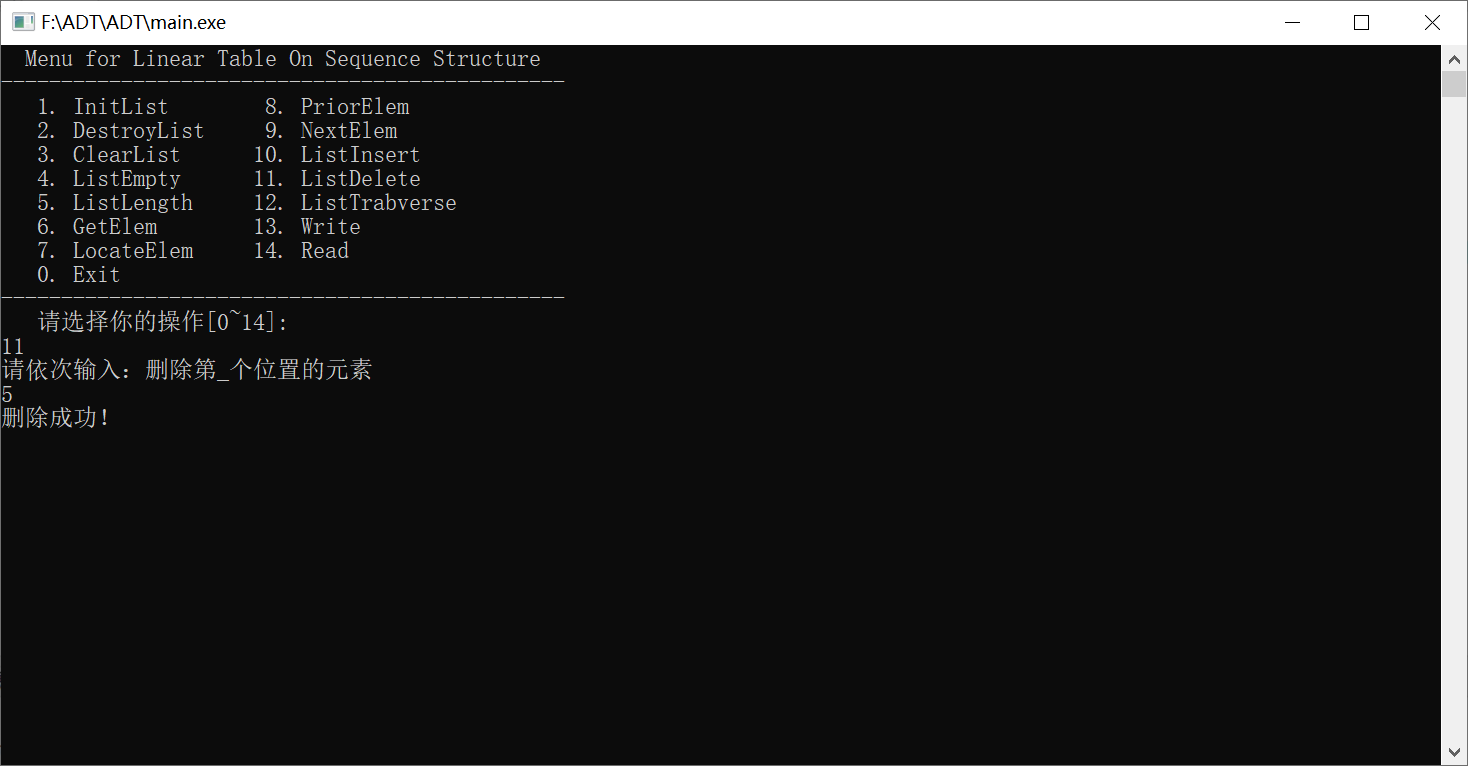


图1-17 测试-11删除元素

1. 执行功能12，遍历线性表1。

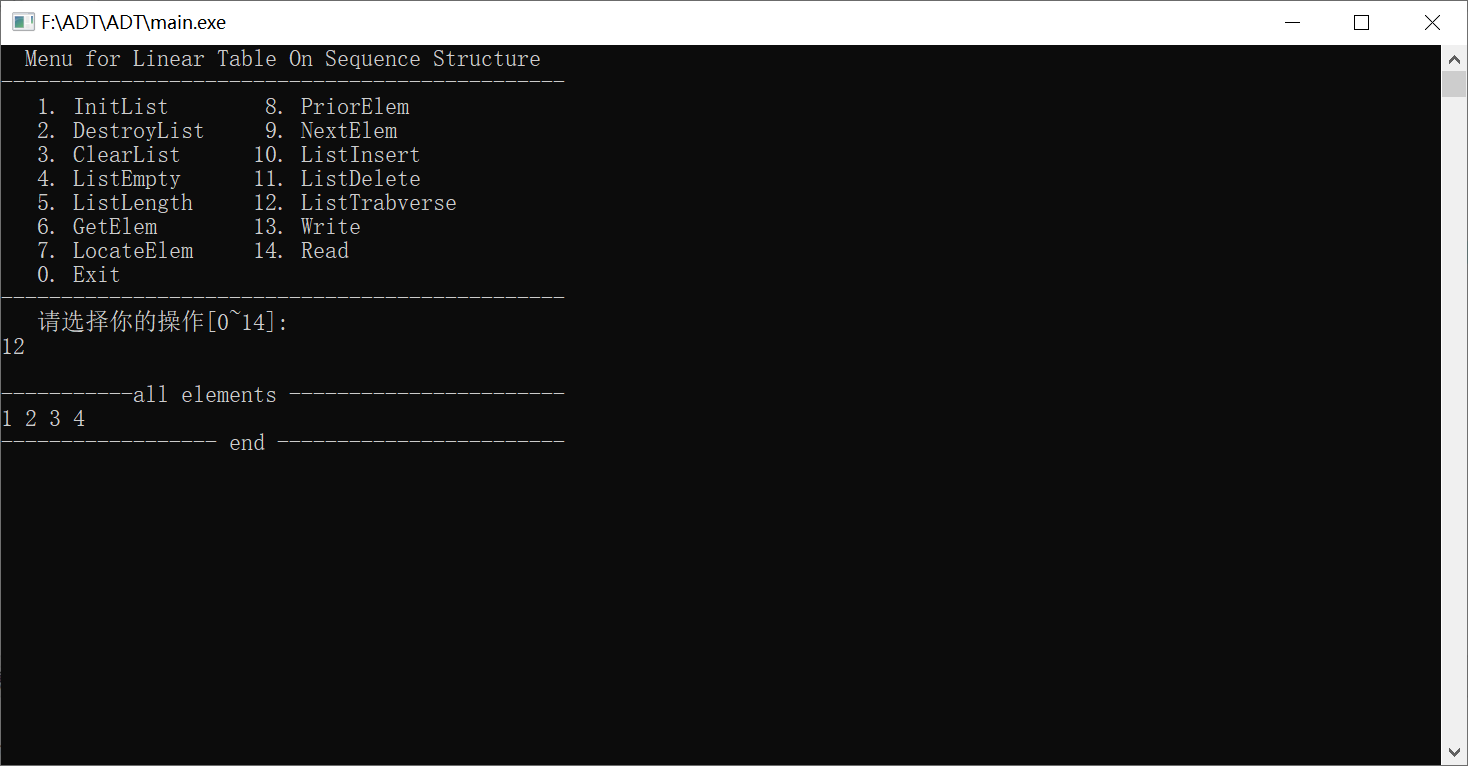


图1-18 测试-12遍历

1. 执行功能13，将数据写入文件data。

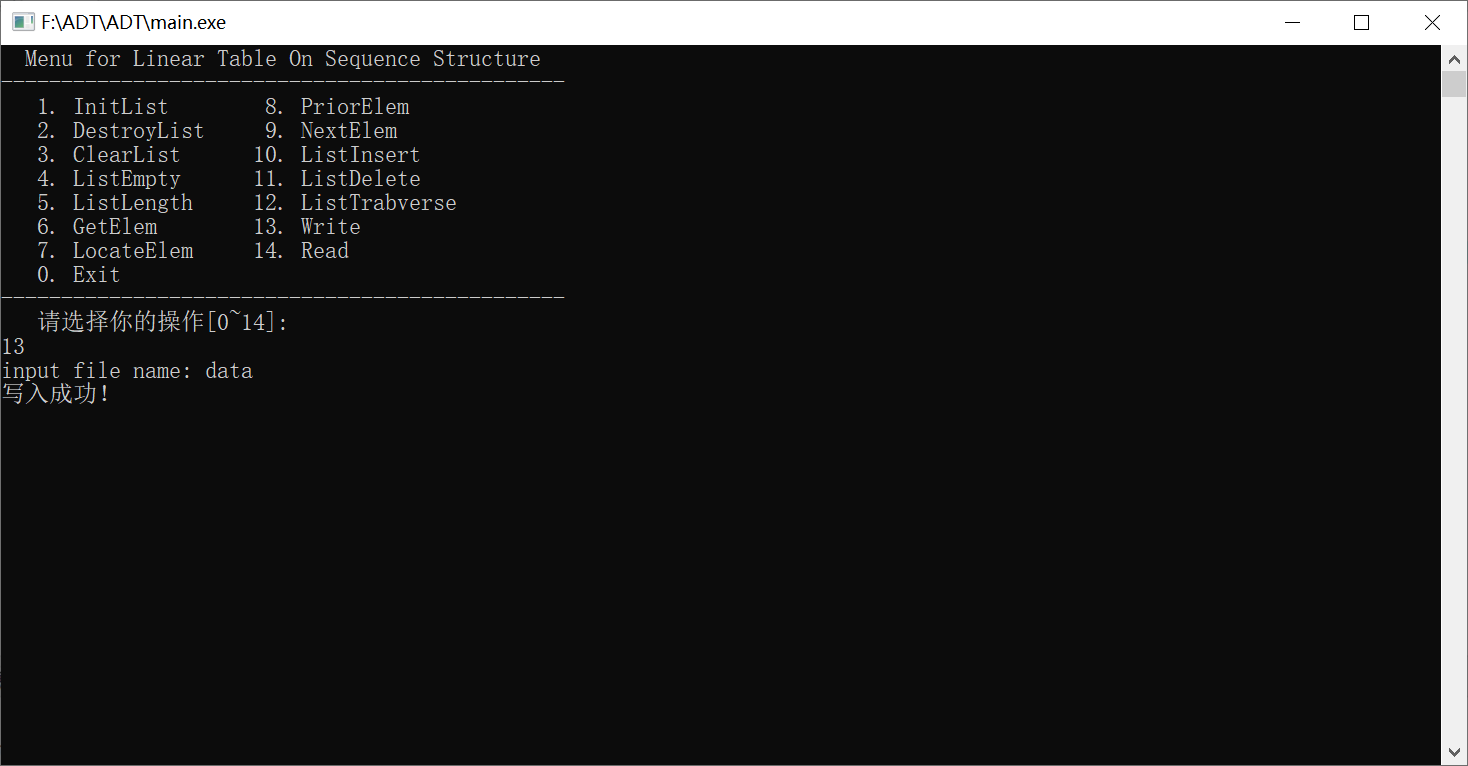


图1-19 测试-13写入文件

1. 执行功能3，清空线性表1。

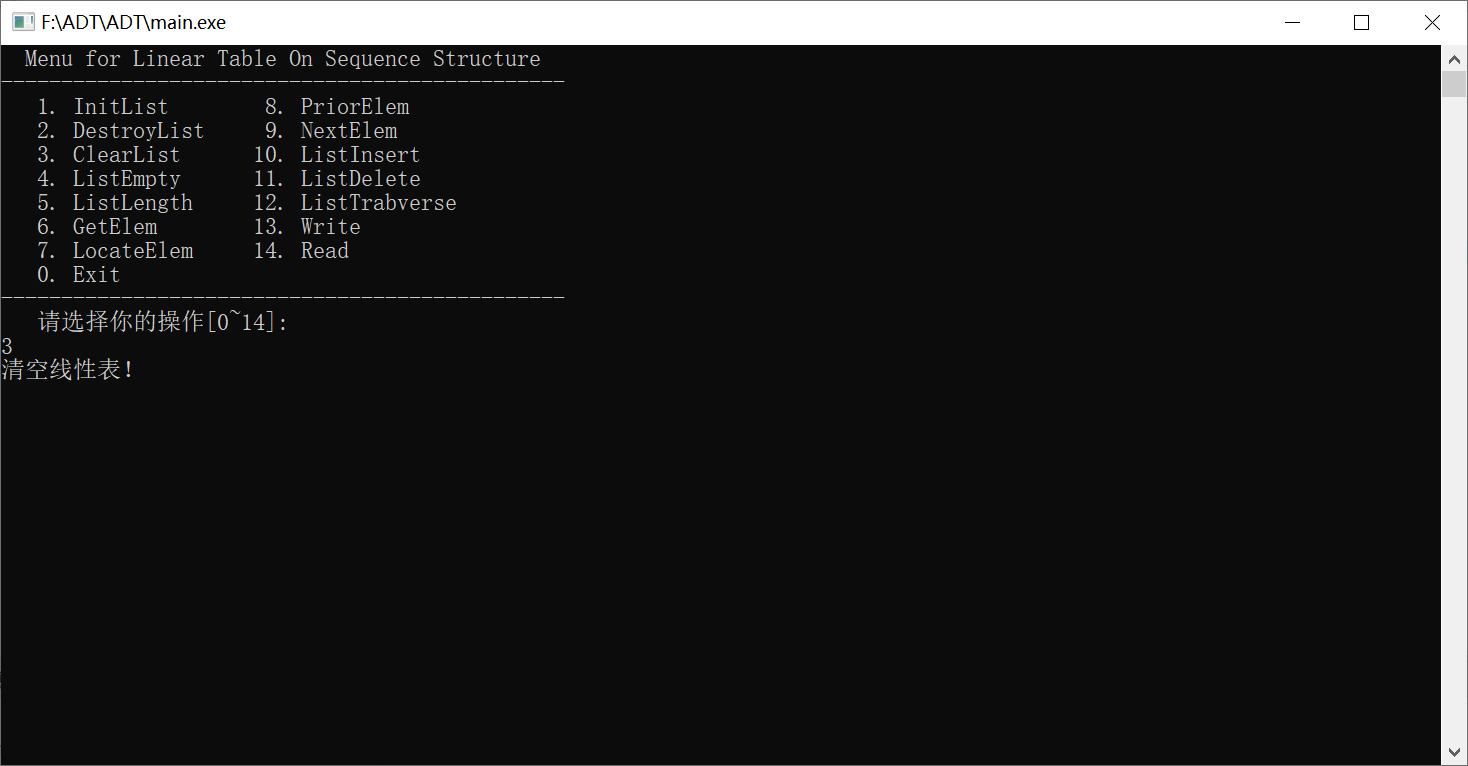


图1-10 测试-3清空线性表

1. 执行功能2，销毁线性表2，及销毁后其他操作不再可以进行。

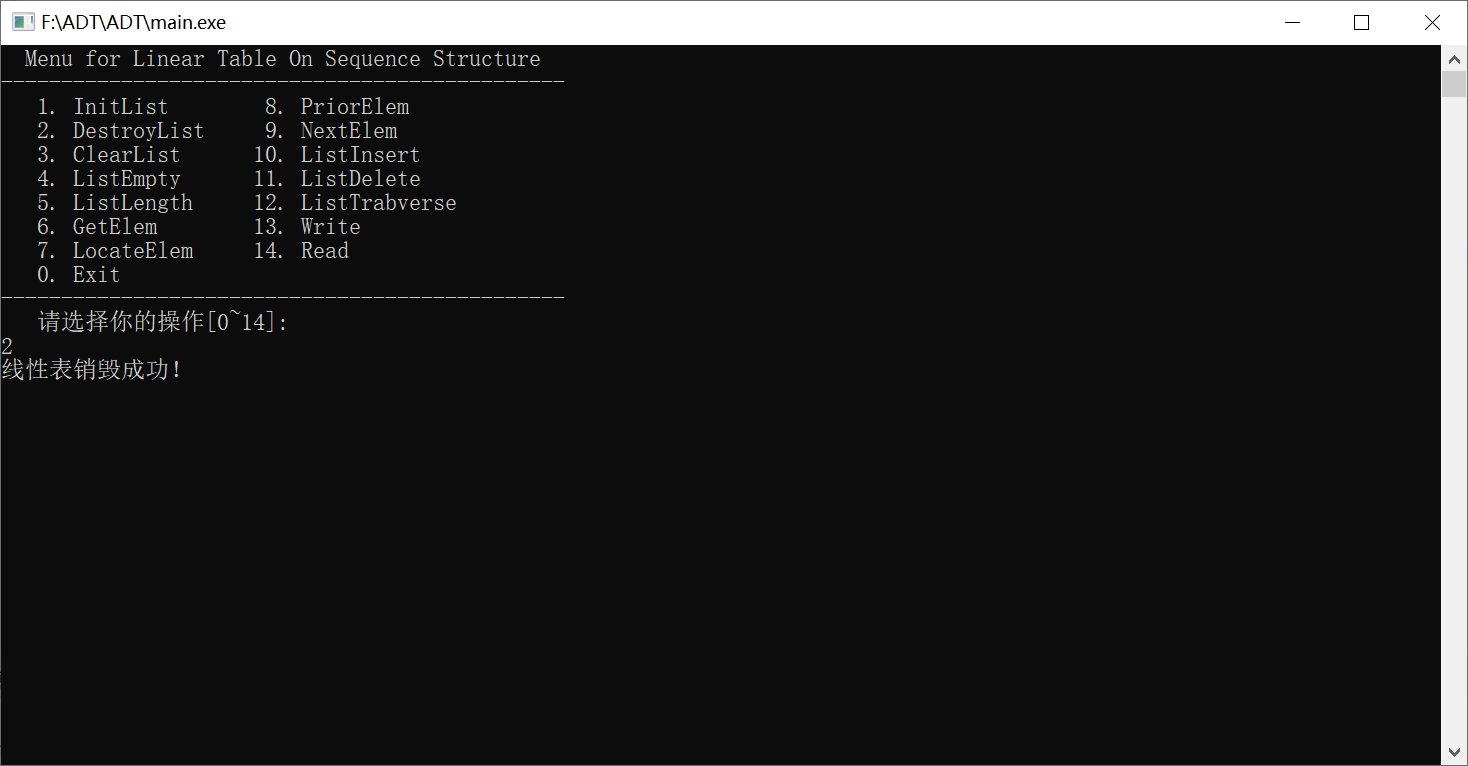


图1-21 测试-2销毁线性表

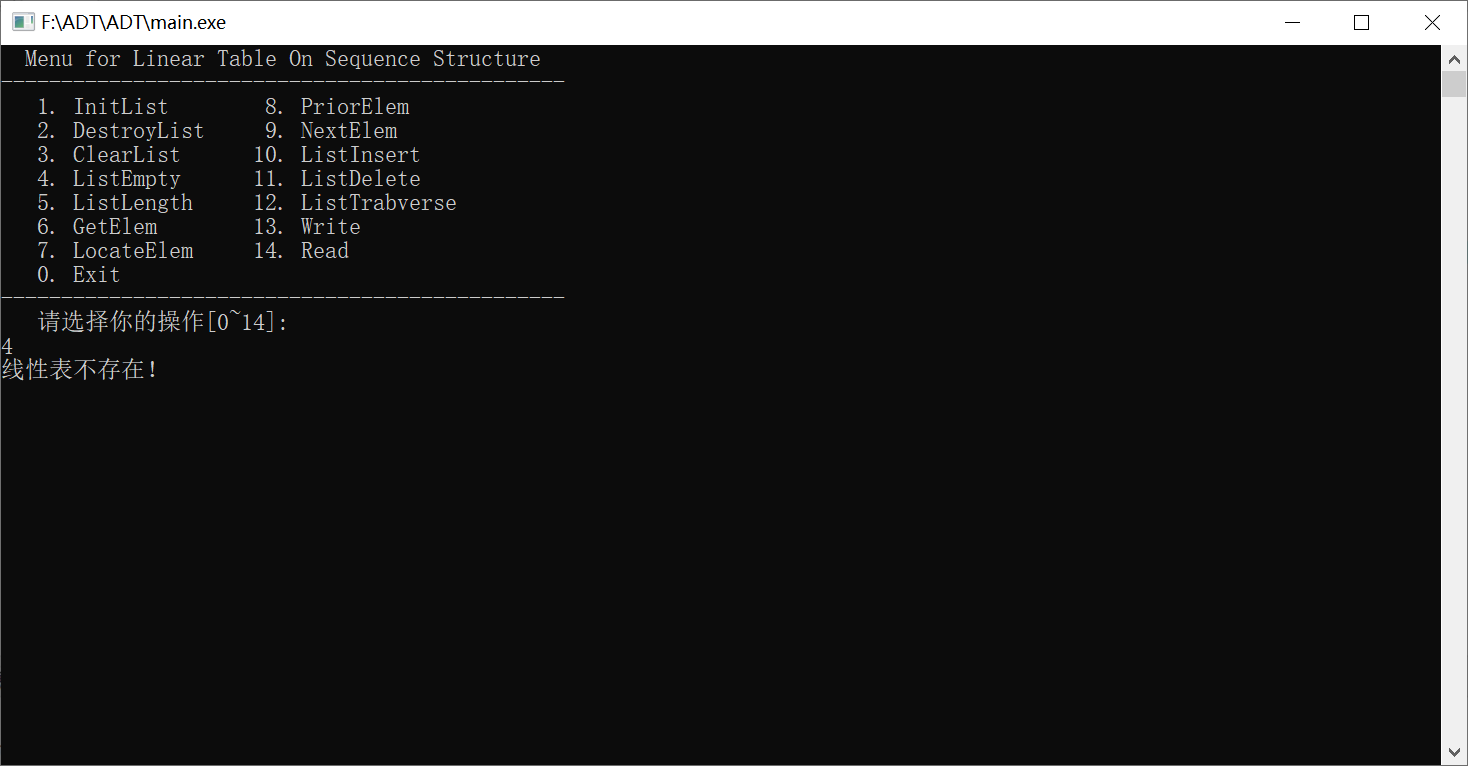


图1-22 测试-2销毁线性表后其他操作不可进行

1. 执行功能0，退出对线性表1的操作。

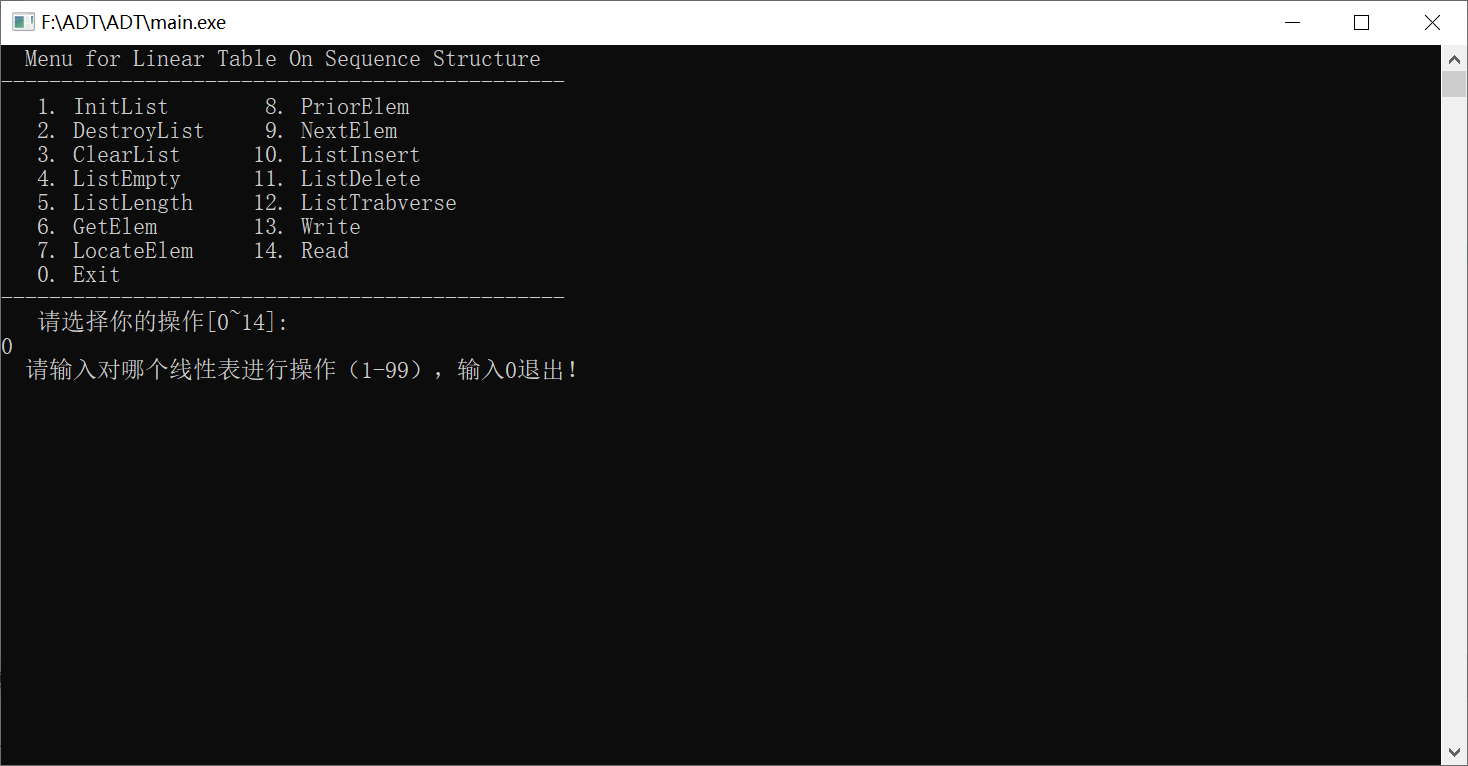
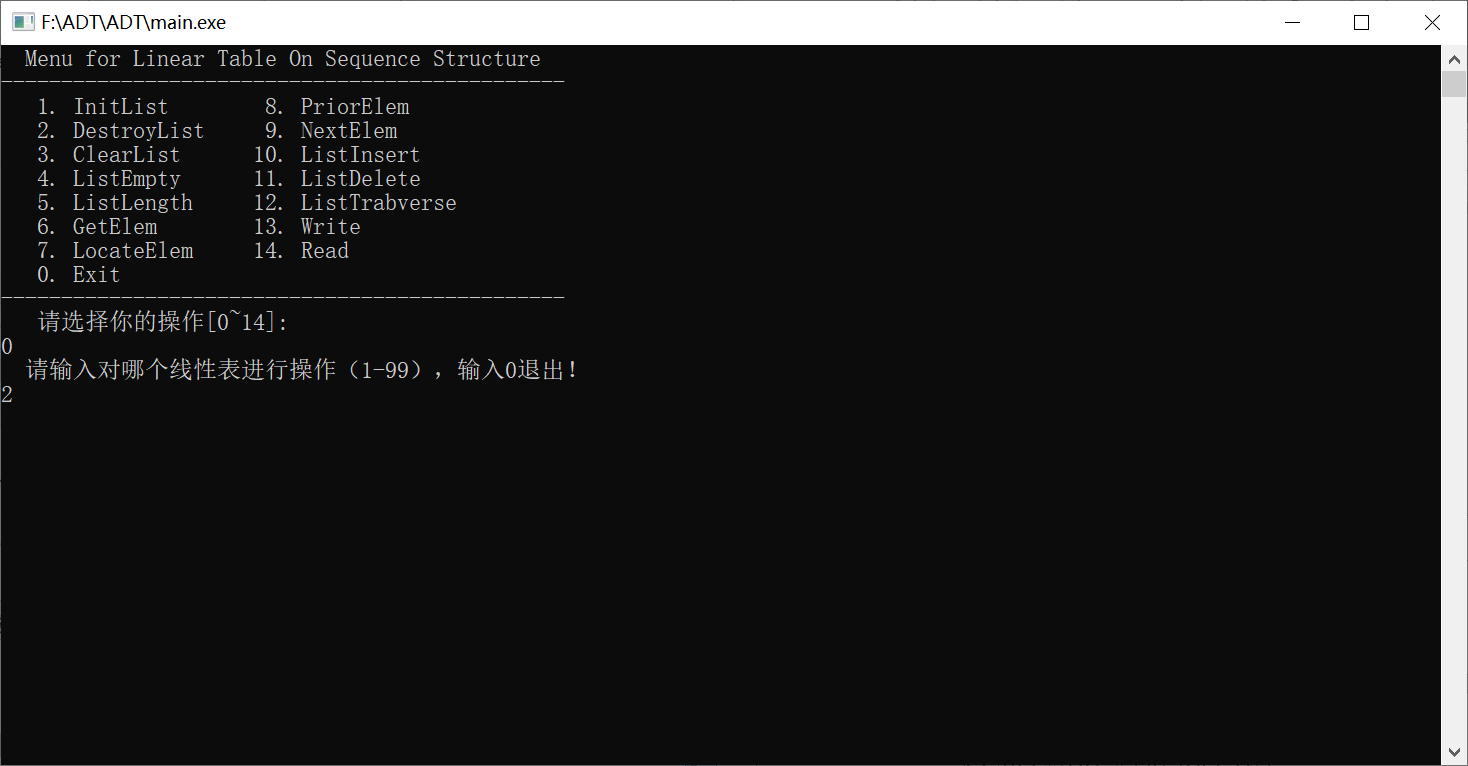
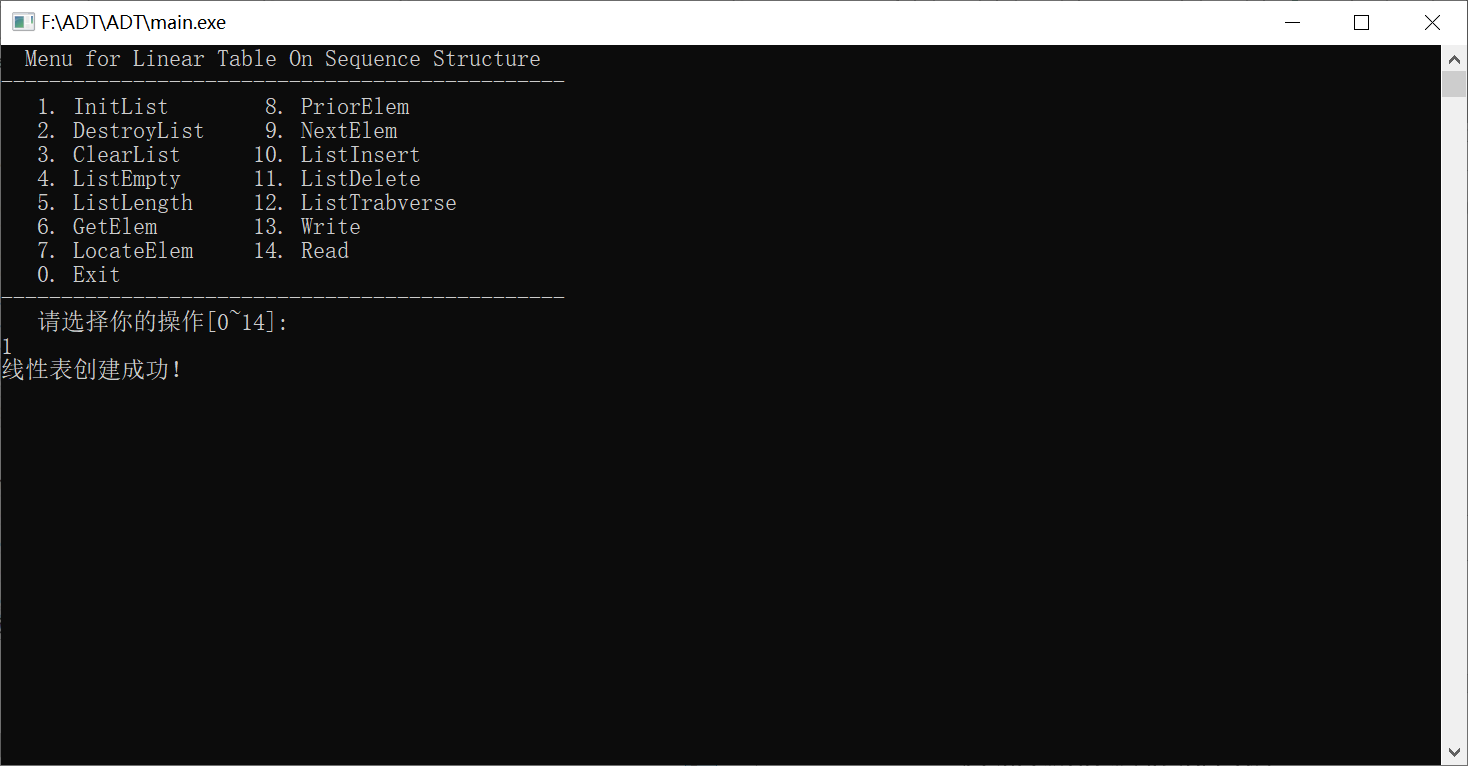


图1-23 测试-0退出操作

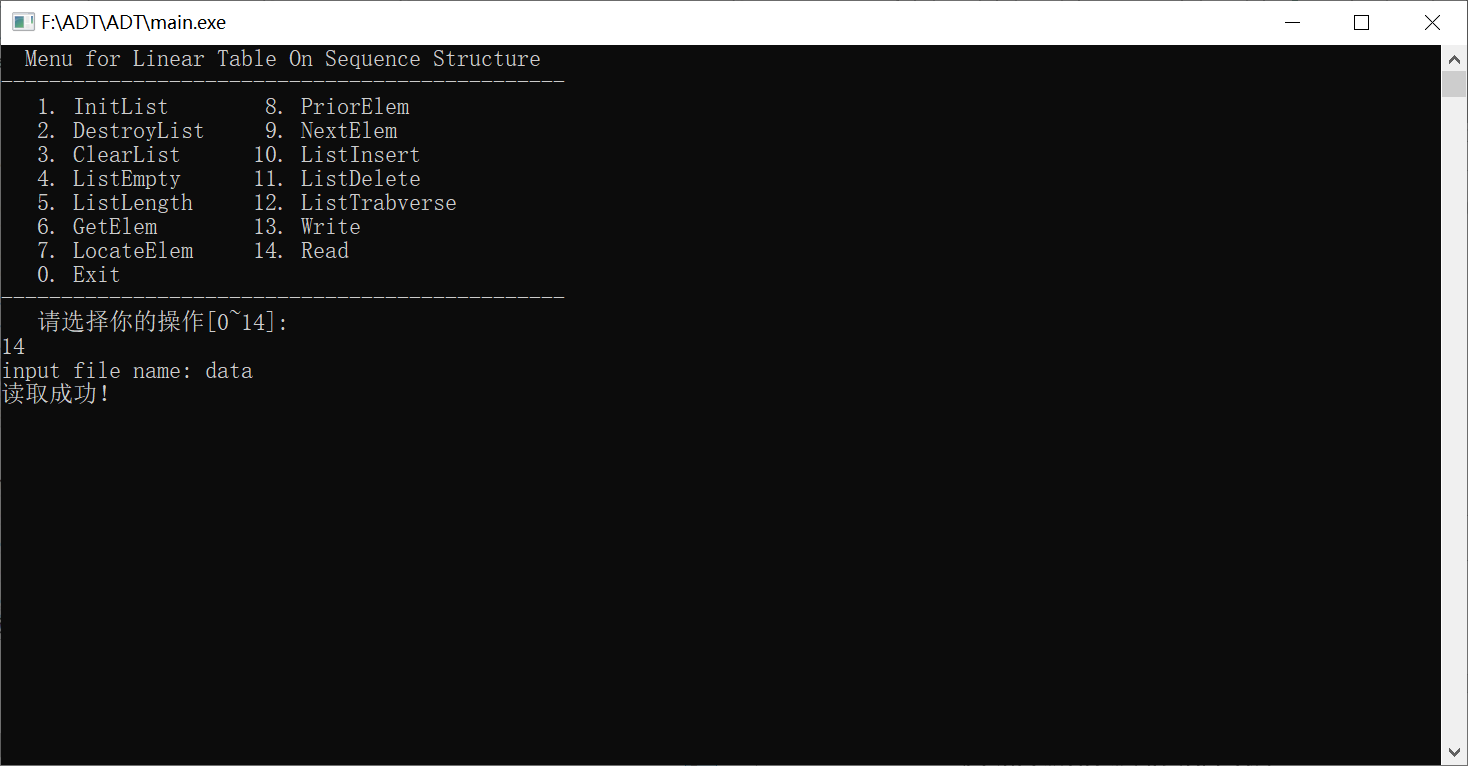
1. 重新选择线性表2。

图1-24 选择线性表2

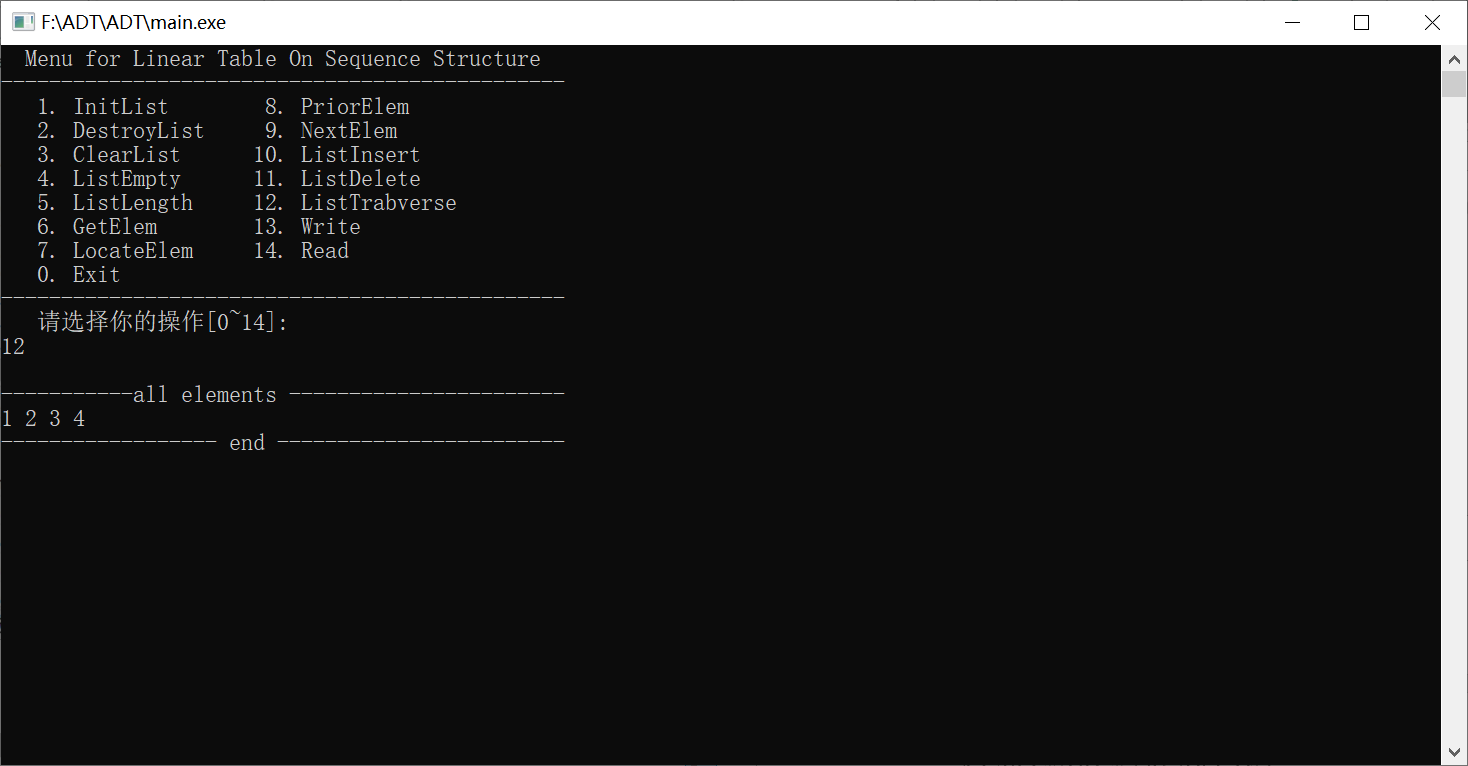
1. 执行功能1，创建线性表2。

图1-25 测试-1创建线性表2

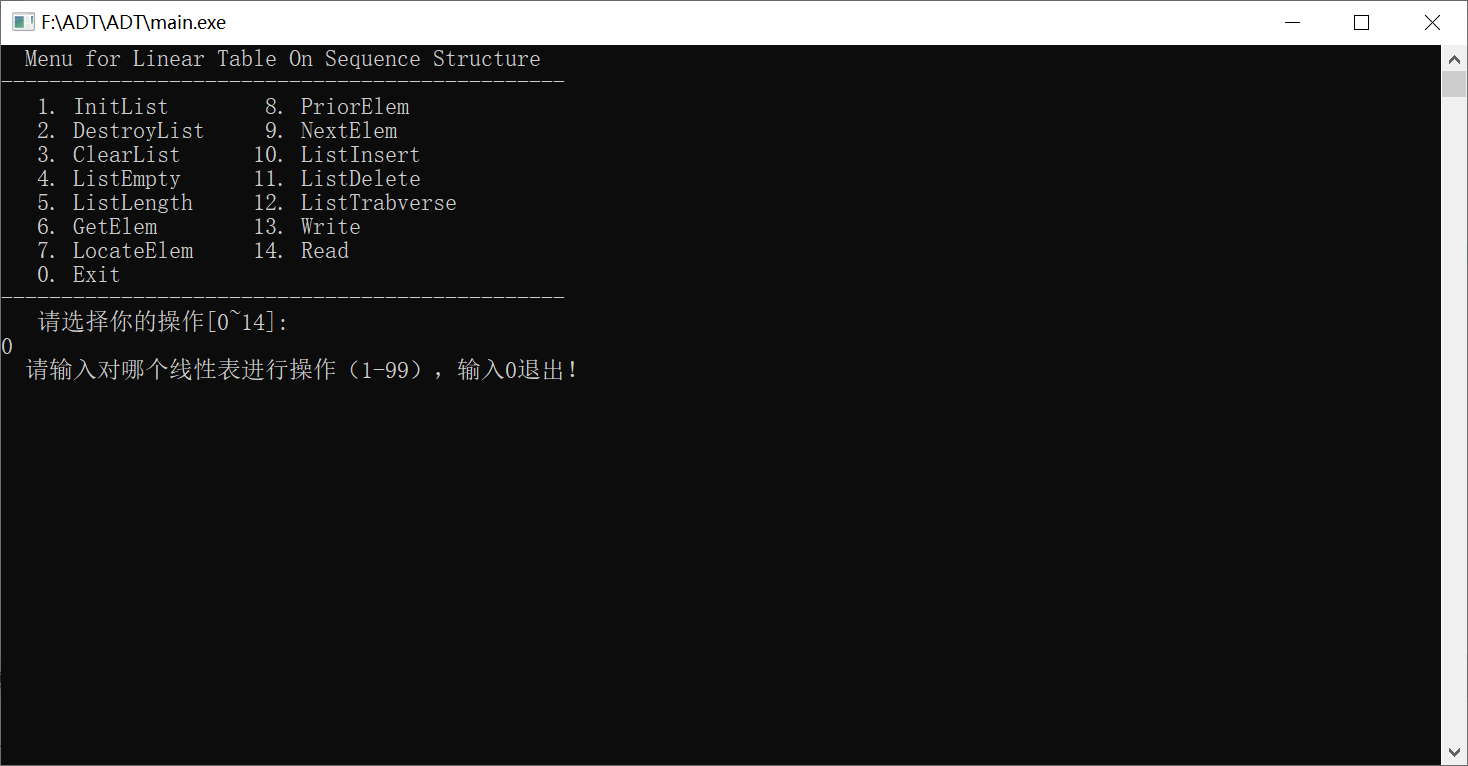
1. 执行功能14，从文件data中读取数据到线性表2。

图1-26 测试-14读入文件

1. 执行功能12，遍历线性表2。

图1-27 测试-12遍历线性表2

1. 执行功能0，退出对线性表2的操作。

图1-28 测试-0退出操作

1. 退出系统。

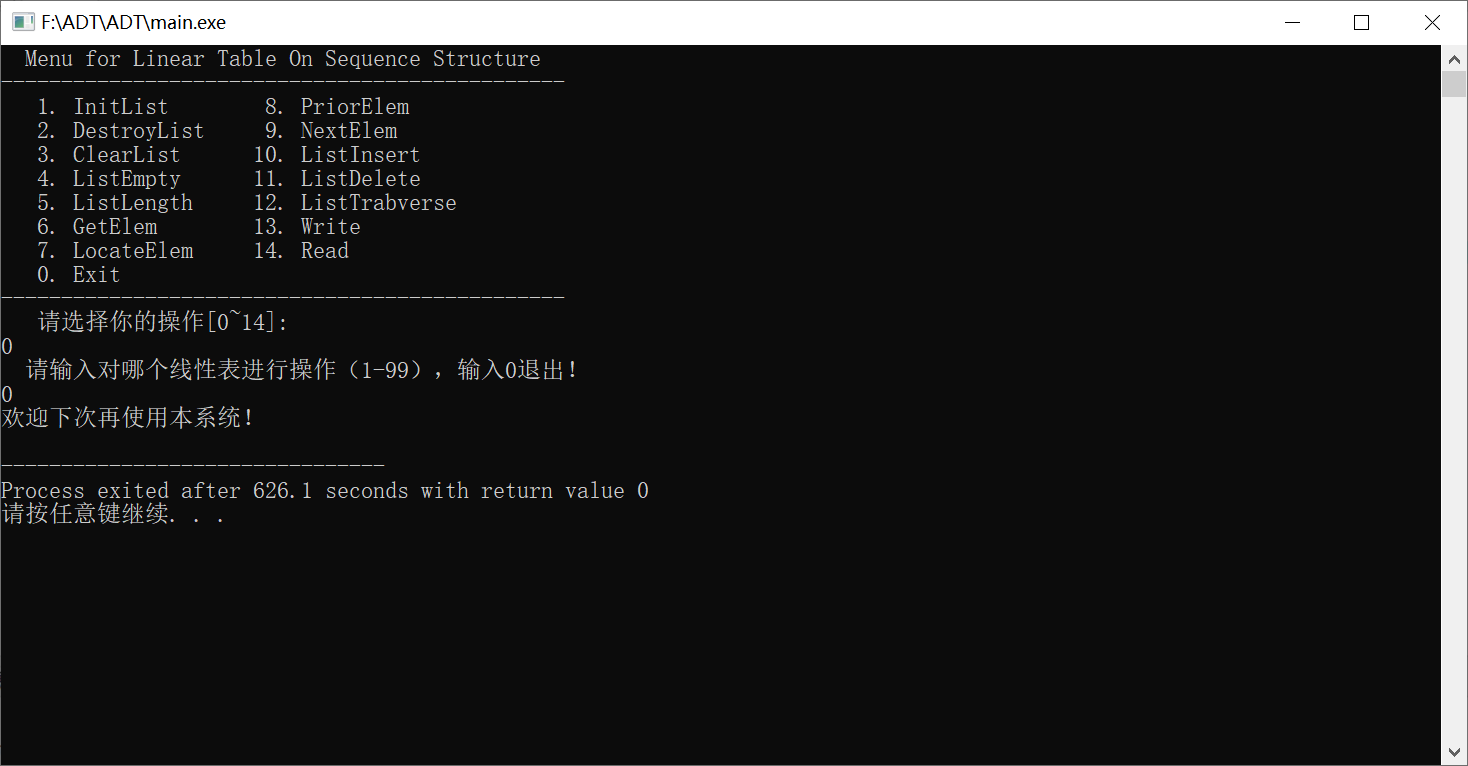


图1-29 退出系统

## 1.5 实验小结

1. 在C语言里没有C++所谓的“按引用传递”。在C语言中是不存在引用的，也就是说C语言中&表示的不是引用，而是取地址符。所以编译器出现的错误提示表明&在这里不能这样用，如何解决这个问题呢？常规的解决方法：用指针来取代引用，在主函数中传进来地址；
2. 明白了C语言中结构体中“.”与“->”的区别。一般情况下用“.”，只需要声明一个结构体。格式是，结构体类型名+结构体名。然后用结构体名加“.”加域名就可以引用域了。因为自动分配了结构体的内存。如同 int a;一样。而用“->”，则要声明一个结构体的指针，还要手动开辟一个该结构体的内存，然后把返回的指针给声明的结构体指针，才能用“->”正确引用。否则内存中只分配了指针的内存，没有分配结构体的内存，导致想要的结构体实际上是不存在。这时候用“->”引用自然出错了，因为没有结构体，自然没有结构体的域了。
3. C语言标准允许main函数为void类型。按照C++的标准中main必须是int类型。
4. 读写文件复习了fwrite和fread的用法，同时通过查阅相关资料了解fprint函数可以将二进制转化后再写入文件，文件打开后可以看到数字，而fwrite函数在写入文件时不会转化，打开后会看到乱码，除非将数据转换为char型写入。
5. 销毁线性表后仍能插入数据，获取表中元素，获取表长。
6. Read要改变当前顺序表，需要传入L的地址。因此深刻明白了函数参数指针有无的区别。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

采用链表的物理结构，构造一个具有菜单的功能演示系统，并且能对多个线性表进行选择。在主函数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。

定义了线性表的创建、销毁、清空、判定空表、求表长、获得元素、查找元素、获得前驱、获得后继、插入元素、删除元素、遍历的函数，并给出输入的操作提示，以文件形式进行存储和加载，将生成的线性表存入到相应的文件，也可以从文件中获取线性表进行操作。

2.1.1 基本运算

1. 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，直接构造头结点。
2. 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，并释放线性表的空间，单链表需要依次将所有空间释放。
3. 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，留下头结点。
4. 判定空表：函数名称是ListEmpty(\*L)；初始条件是线性表L已存在；判断首结点是否为空，操作结果是若L为空表则返回TRUE,否则返回NO。
5. 求表长：函数名称是ListLength(\*L)；初始条件是线性表已存在，遍历计算线性表的长度；操作结果是返回L中数据元素的个数。
6. 获得元素：函数名称是GetElem(\*L,i,\*e)；初始条件是线性表已存在，i满足1≤i≤ListLength(L)，若i非法则返回ERROR；操作结果是用e返回L中第i个数据元素的值。
7. 查找元素：函数名称是LocateElem(\*L,e)；初始条件是线性表已存在；操作结果是返回L中第1个与e相等的数据元素的位序，若这样的数据元素不存在，则返回值为ERROR。
8. 获得前驱：函数名称是PriorElem(\*L,cur,\*pre\_e)；初始条件是线性表L已存在，通过LocateElem(\*L,e)函数查找元素cur；操作结果是若cur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。
9. 获得后继：函数名称是NextElem(\*L,cur,\*next\_e)；初始条件是线性表L已存在，通过LocateElem(\*L,e)函数查找元素cur；操作结果是若cur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。
10. 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，i满足1≤i≤ListLength(L)+1，若i非法，则返回ERROR；操作结果是在L的第i个位置之前插入新的数据元素e。
11. 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，i满足1≤i≤ListLength(L)，若i非法，则返回ERROR；操作结果：删除L的第i个数据元素，用e返回其值。
12. 遍历表：函数名称是ListTraverse(\*L)，初始条件是线性表L已存在；操作结果是通过遍历依次对L的每个数据元素调用输出。
13. 写入文件：函数名称是Write(\*L)，初始条件是线性表L已存在；输入文件名，将文件打开并写入；操作结果是将L的数据依次写入到文件里。
14. 读取文件：函数名称是Read(\*\*L)，初始条件是线性表L已存在；输入文件名，将文件打开并读取；操作结果是将L的数据依次读取到线性表里。

2.1.2 演示系统

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。

## 2.2 系统设计

2.2.1 头文件、常量及结构说明

1. 头文件

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

1. 预定义常量

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define NO -1

1. 类型表达式

typedef int status;

typedef int ElemType; //数据元素类型定义

1. 定义数据的物理结构如下：

typedef struct LNode{ //链式表（链式结构）的定义

ElemType data;

struct LNode \*next;

}LinkList;

同时因为需要实现多线性表的操作，构造一个结构数组：LinkList \*M[100];

2.2.2 算法设计

1. 初始化表：传入参数结构型指针L的地址，在函数中，首先使用malloc函数创建头结点，且将头结点置为空。
2. 销毁表：传入参数结构体指针L的地址，声明一个结构体指针中间变量p与头指针相等，若p存在，遍历单链表，释放每个结点的空间，再令p为空，将其赋给结构体指针L。否则返回ERROR。
3. 清空表：传入参数结构体指针L，因为只需要清空操作并不释放内存空间。声明一个结构体指针中间变量p与头结点相等，若p存在，遍历单链表，释放每个结点的空间，再令头结点为空。否则返回ERROR。
4. 判定空表：传入参数结构体指针L，直接判断头结点是否为NULL。
5. 求表长：传入参数结构体指针L，定义结构体指针p，令其等于首结点，通过遍历计算个数，用i计数，返回i值。
6. 获得元素：传入参数结构体指针L，抽象数据e的地址和元素位置i，判断i值是否合法，其中1≤i≤ListLength(L),若不合法返回ERROR，合法则定义结构体指针p，使其指向首结点，通过遍历使p为第i位，令e的值为第i位元素的值，返回OK。
7. 查找元素：传入参数结构体变量L，需要查找的元素e。定义结构体指针p，使其指向首结点，通过遍历从头查找每个元素与该元素e是否相等，相等则返回第一个与其相同的元素的位置，否则继续遍历。若没有相同的元素则返回ERROR。
8. 获得前驱： 传入参数结构体变量L，查找前驱的元素cur和前驱指针pre\_e。首先调用LocateElem函数判断该元素在线性表中是否存在并判断是否是第一个元素，若不存在或者是第一个元素则返回ERROR。若存在返回元素位置，定义两个结构体指针p、q，使p指向头结点，q指向首结点；通过遍历使q指向与cur相同的元素，此时p指向其前驱，使pre\_e的值为p->data，返回OK。
9. 获得后继：传入参数结构体变量L，查找前驱的元素cur和前驱指针next\_e。首先调用LocateElem函数判断该元素在线性表中是否存在并判断是否是最后一个元素，若不存在或者是最后一个元素则返回ERROR。若存在返回元素位置，定义一个结构体指针p，使p指向首结点；通过遍历使p指向与cur相同的元素，则pre\_e的值为p->next->data，返回OK。
10. 插入元素：传入参数结构体变量L，需要插入的数据e和元素位置i。定义三个结构体指针t、p、q，使p指向头结点，q指向首结点，t为新结点并分配空间，将新的元素值赋给新结点；首先判定要插入的位置i的合法性，若不合法，返回ERROR，若合法，遍历到i-1位，进行插入新结点的操作，返回OK。具体如流程图所示。



图2-1 插入元素流程图

1. 删除元素：传入参数结构体变量L，需要删除的元素位置i和抽象数据e的地址。定义两个结构体指针tp、q，使p指向头结点，q指向首结点。首先判定要插入的位置i的合法性，若不合法，返回ERROR，若合法，进行删除结点的操作，通过遍历到i-1位，删除第i位，返回OK。具体如流程图所示。



图2-2 删除元素流程图

1. 遍历表：传入参数结构体变量L，直接用一个循环来输出线性表中的每一个元素。

2.2.3 理论分析

表2-1 函数算法复杂度分析

|  |  |
| --- | --- |
| 测试功能功能及序号 | 算法复杂度 |
| 1. InitList | O(1) |
| 2. DestroyList | O(n) |
| 3. ClearList | O(n) |
| 4. ListEmpty | O(1) |
| 5. ListLength | O(n) |
| 6. GetElem | O(n) |
| 7. LocateElem | O(n) |
| 8. PriorElem | O(n) |
| 9. NextElem | O(n) |
| 10. ListInsert | O(n) |
| 11. ListDelete | O(n) |
| 12. ListTrabverse | O(n) |

## 2.3 系统实现

2.3.1 系统演示

整个系统的流程及结构如下图所示。



图2-3 系统结构图

2.3.2 部分函数功能实现

1. 在获得前驱和后继时，通过调用LocateElem(\*L,cur)函数还判断线性表中是否有该元素，并判断于是位置是否存在前驱和后继以保证前驱和后继的正确性。

获得前驱：

if(LocateElem(L,cur)==1||LocateElem(L,cur)==ERROR)

//遍历单链表，查找是否有相同元素并判断位置合法

获得后继：

if(LocateElem(L,cur)==ListLength(L)||LocateElem(L,cur)==ERROR)

//遍历单链表，查找是否有相同元素并判断位置合法

1. 插入元素：根据之前的算法设计流程图实现操作。定义三个结构体指针t、p、q，使p指向头结点，q指向首结点，t为新结点并分配空间，将新的元素值赋给新结点；首先判定要插入的位置i的合法性，若不合法，返回ERROR，若合法，遍历到i-1位，进行插入新结点的操作，返回OK。改变指针域时注意语句先后顺序。

status ListInsert(LinkList \*L,int i,ElemType e){

//在链式线性表中第i个元素之前插入新的元素e

int j;

LinkList \*t;//定义新的结点并分配空间

t=(LinkList\*)malloc(sizeof(LinkList));

t->data=e;//将新的元素值赋给新结点

LinkList \*p=L;

LinkList \*q=L->next;

if(i>ListLength(L)+1||i<1)//判断i是否合法

return ERROR;

else{

for(j=1;j<i;j++){//遍历到i-1位

p=q;

q=q->next;

}

t->next=p->next;

p->next=t;

}

return OK;

}

1. 删除元素：根据之前的算法设计流程图实现操作。定义两个结构体指针tp、q，使p指向头结点，q指向首结点。首先判定要插入的位置i的合法性，若不合法，返回ERROR，若合法，进行删除结点的操作，通过遍历到i-1位，删除第i位，返回OK。注意释放最后的无用指针空间。

status ListDelete(LinkList \*L,int i,ElemType \*e){

//在链式线性表中删除第i个元素，并用e返回其值

int j;

LinkList \*p=L;

LinkList \*q=L->next;

if(i>ListLength(L)||i<1)//判断i是否合法

return ERROR;

else{

for(j=1;j<i;j++){//遍历到i-1位

p=q;

q=q->next;

}

p->next=q->next;

free(q);

}

return OK;

}

2.3.3 用文件读写

文件的读取可用多个线性表进行操作，例如用线性表1写入文件，再创建线性表2读取数据，使之与线性表1 相同。

1. 文件的写入：文件的写入与顺序表相似。因为不要求打开文件能直接看到数据，而是用另外一个线性表直接读取，故编写代码时直接用fwrite进行二进制写入，而不采用fprintf函数(该函数写入可直接看到数据)。

写入文件：

status Wirte(LinkList \*L) {

FILE \*fp=NULL;

char filename[30];

LinkList \*p=L->next;

printf("input file name: ");

scanf("%s",filename);

if ((fp=fopen(filename,"wb"))==NULL){

printf("File open error\n ");

return FALSE;

}

//写文件的方法

for(;p!=NULL;){//遍历单链表,写入文件

fwrite(&(p->data),sizeof(ElemType),1,fp);

p=p->next;

}

fclose(fp);

return OK;

}

1. 文件的读取：单链表不同于顺序表，不能直接用while循环。于是定义结构体指针t、p、q，t为头指针的地址(双重指针可修改一级指针的地址)，q为首结点，用while循环读取时进行线性表L的构造，最后释放无用指针，返回OK。

读取文件**：**

status Read(LinkList \*\*L) {

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

//读文件的方法

if ((fp=fopen(filename,"rb"))==NULL){

printf("File open error\n ");

return FALSE;

}

LinkList \*t=\*L;

LinkList \*p,\*q;

p=(LinkList\*)malloc(sizeof(LinkList));

q=(LinkList\*)malloc(sizeof(LinkList));//为指针分配空间

t->next=q;//q为首结点

while(fread(&(q->data),sizeof(ElemType),1,fp)){//从文件中读取数据到单链表中

p->next=q;

q->next=NULL;

p=p->next;

q=(LinkList\*)malloc(sizeof(LinkList));

}

//这里从文件中逐个读取数据元素恢复链式表，

free(q);//释放指针q

fclose(fp);

return OK;

}

## 2.4 系统测试

2.4.1 测试计划

表2-2 测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能功能及序号 | 测试的线性表 | 输入的参数 | 预计结果 | 线性表实际状态 |
| 选择线性表 | 1 | 1 | 对线性表1进行操作 | 选择线性表1，并未创建 |
| 1. InitList | 1 | 无 | 创建线性表1 | 创建了头结点 |
| 10. ListInsert（多次调用） | 1 | 1 1；2 2；3 3；4 5 | 线性表1 写入数据 | 线性表序列为：1，2，3，5 |
| 4. ListEmpty | 1 | 无 | 线性表非空 | 线性表序列为：1，2，3，5 |
| 5. ListLength | 1 | 无 | 线性表长度 | 线性表序列为：1，2，3，5 |
| 6. GetElem | 1 | 想要获得的元素位置：3 | 第3个位置的元素 | 线性表序列为：1，2，3，5 |
| 7. LocateElem | 1 | 需要查找的元素：2 | 与该元素相同的元素的位置 | 线性表序列为：1，2，3，5 |
| 8. PriorElem | 1 | 需要获得前驱的元素：3 | 该元素的前驱 | 线性表序列为：1，2，3，5 |
| 9. NextElem | 1 | 需要获得后继的元素：3 | 该元素后继 | 线性表序列为：1，2，3，5 |
| 10. ListInsert | 1 | 4 4； | 在第四个位置插入元素4 | 线性表序列为：1，2，3，4，5 |
| 11. ListDelete | 1 | 删除元素的位置：5 | 删除第五个位置的元素 | 线性表序列为：1，2，3，4 |
| 12. ListTrabverse | 1 | 无 | 遍历线性表1 | 线性表序列为：1，2，3，4 |
| 13. Write | 1 | 文件名：data | 将线性表1写入文件 | 线性表序列为：1，2，3，4 |
| 3. ClearList | 1 | 无 | 清空线性表1 | 表长为0 |
| 2. DestroyList | 1 | 无 | 销毁线性表1 | 线性表空间被释放 |
| 0. Exit | 1 | 无 | 退出操作 | 无线性表 |
| 选择线性表 | 2 | 2 | 对线性表2进行操作 | 选择线性表2，并未创建 |
| 1. InitList | 2 | 无 | 创建线性表2 | 创建头结点 |
| 14. Read | 2 | 文件名：data | 从该文件读入数据 | 线性表2序列：1，2，3，4 |
| 12. ListTrabverse | 2 | 无 | 遍历线性表2 | 线性表2序列：1，2，3，4 |
| 0. Exit | 2 | 无 | 退出操作 | 无线性表 |
| 退出选择线性表 | 无 | 退出输入0 | 退出选择 | 运行结束 |

2.4.2 测试结果

执行2.4.1表格的测试计划，运行结果截图如下所示。

1. 选择线性表1，进入操作。

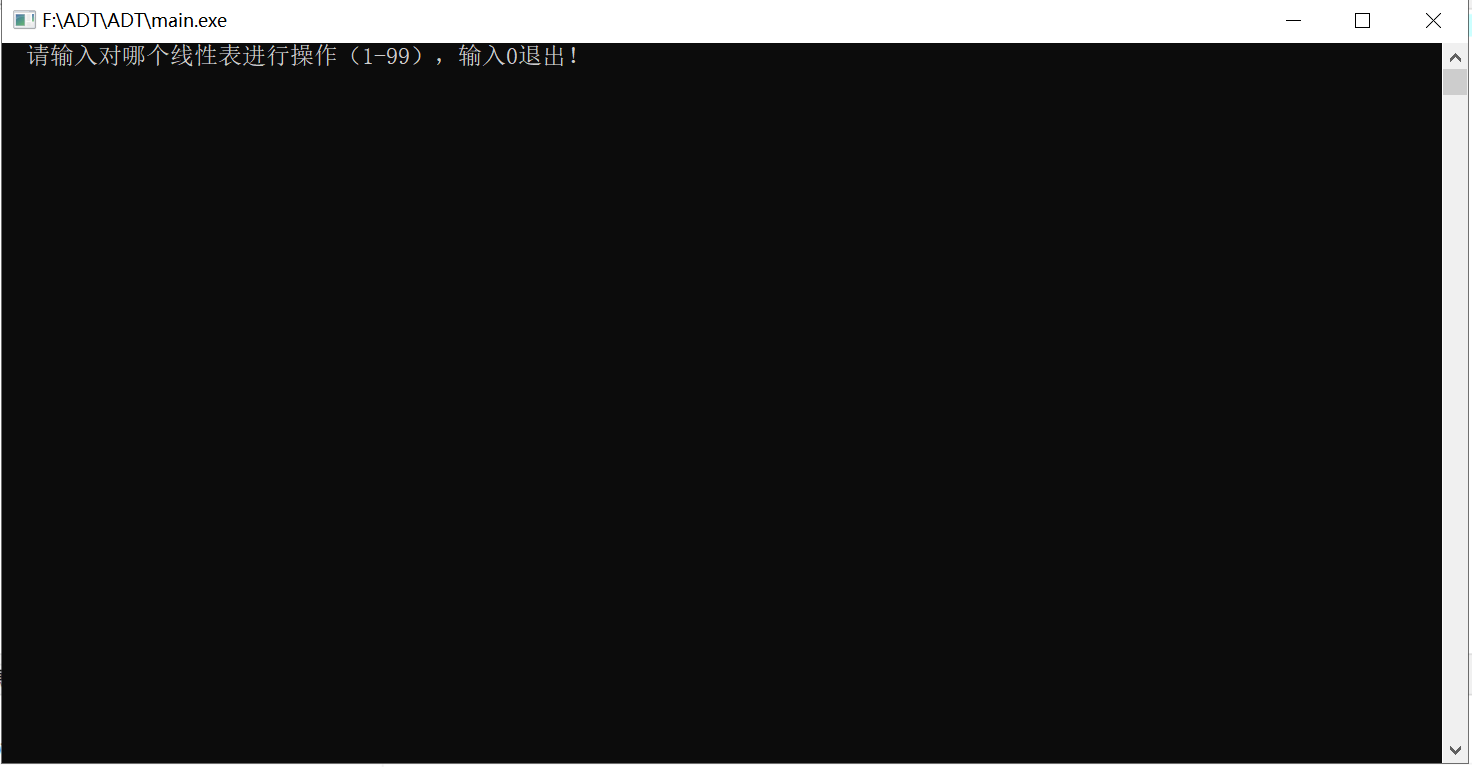


图2-4 测试-选择线性表

1. 执行功能1，创建线性表1。

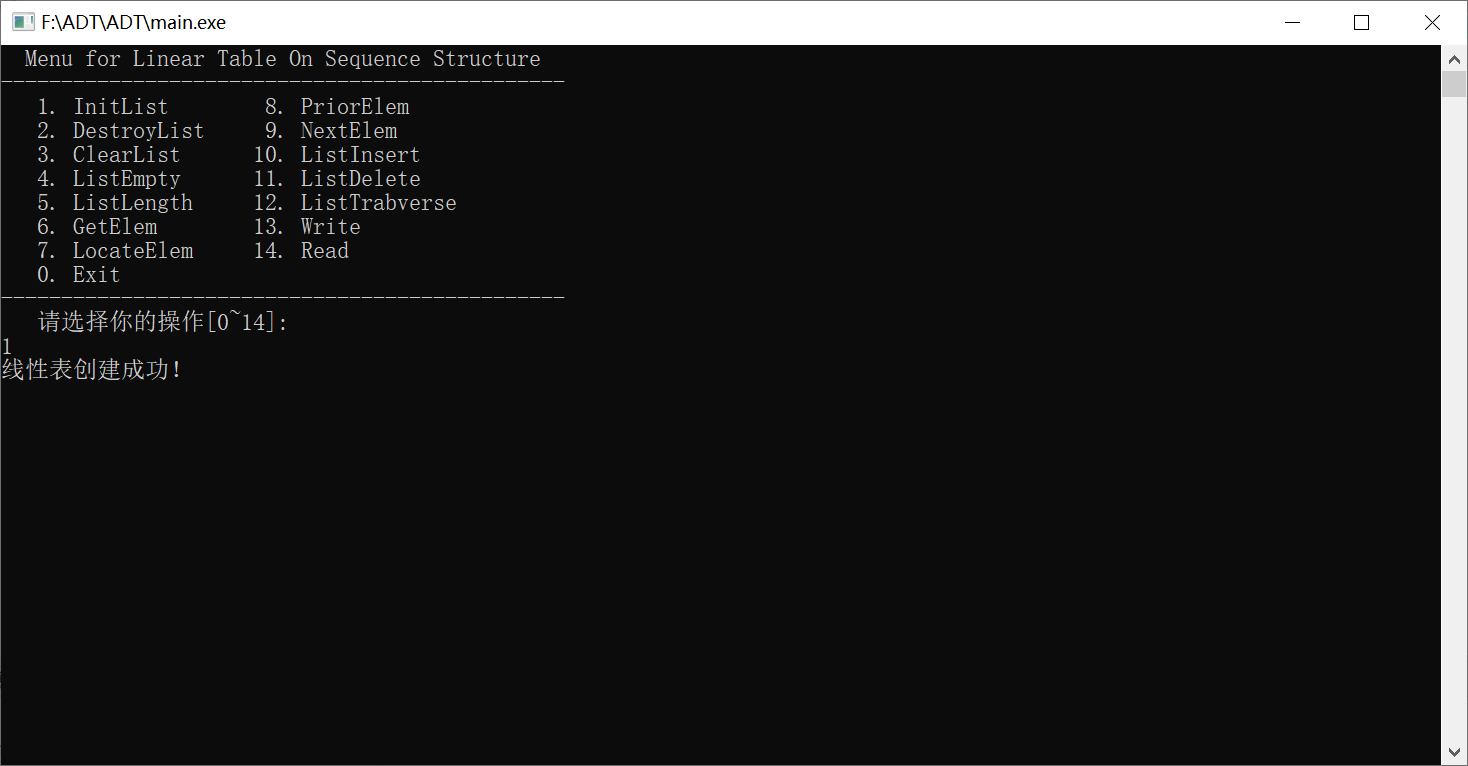


图2-5 测试-1创建线性表

1. 多次执行功能10，向线性表1添加元素1、2、3、5。

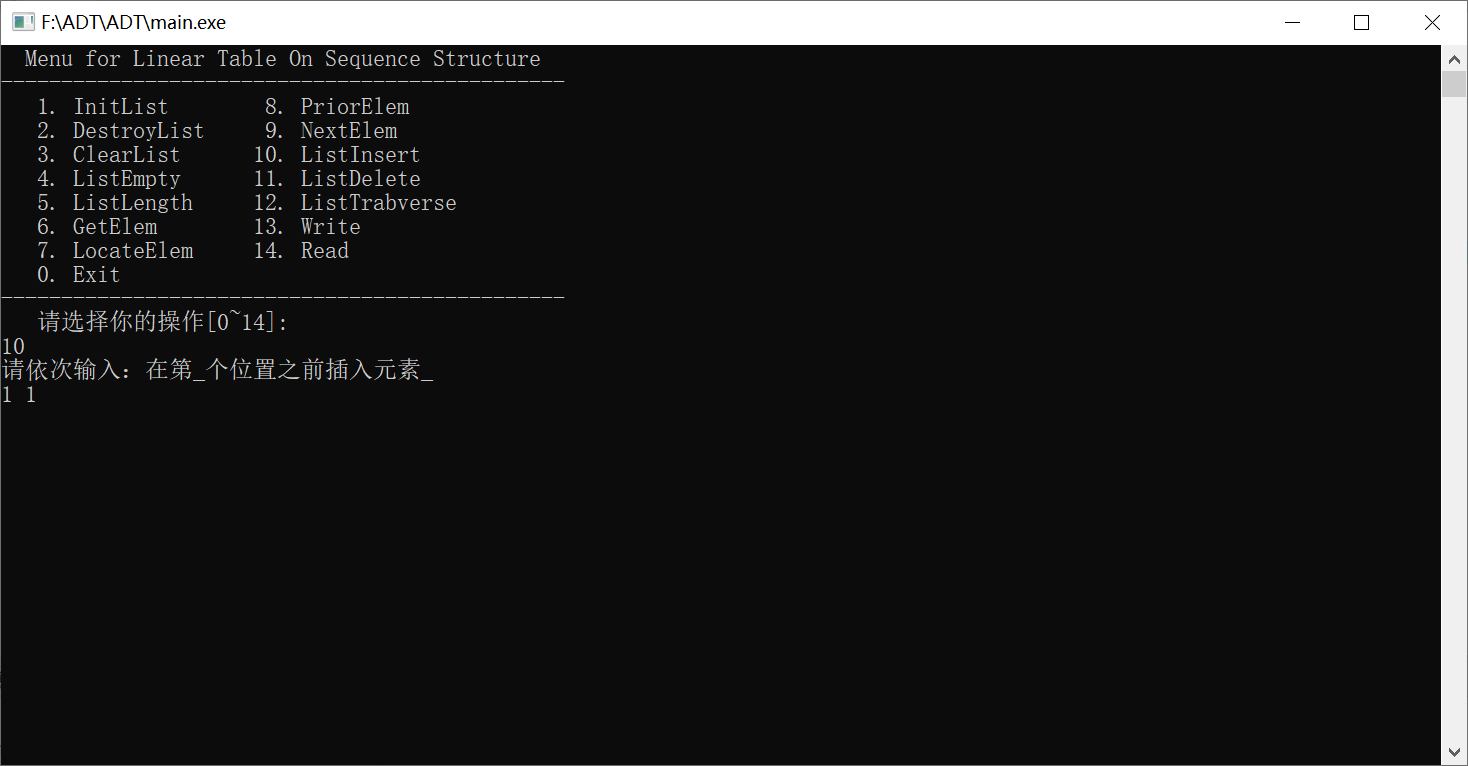


图2-6 测试-10插入元素

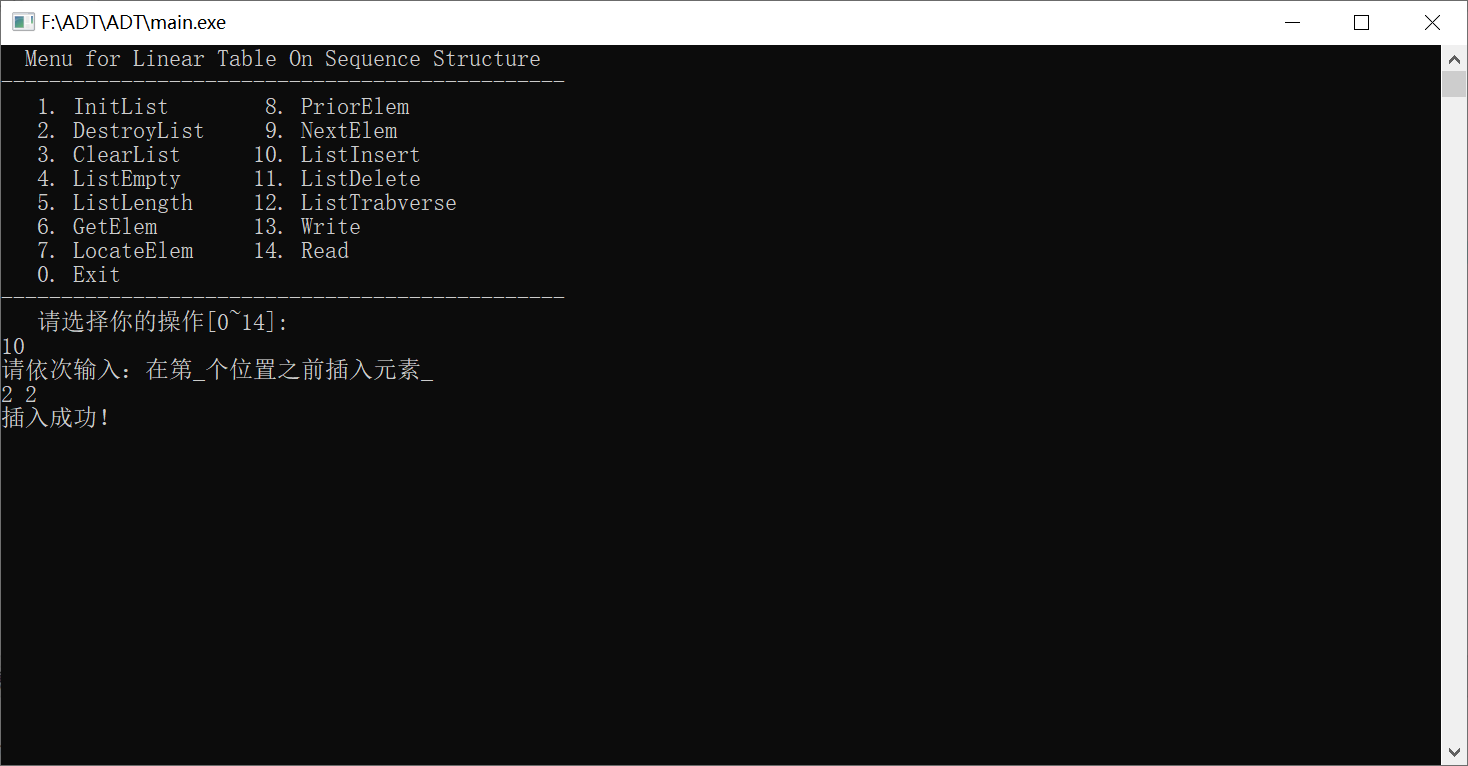


图2-7 测试-10插入元素

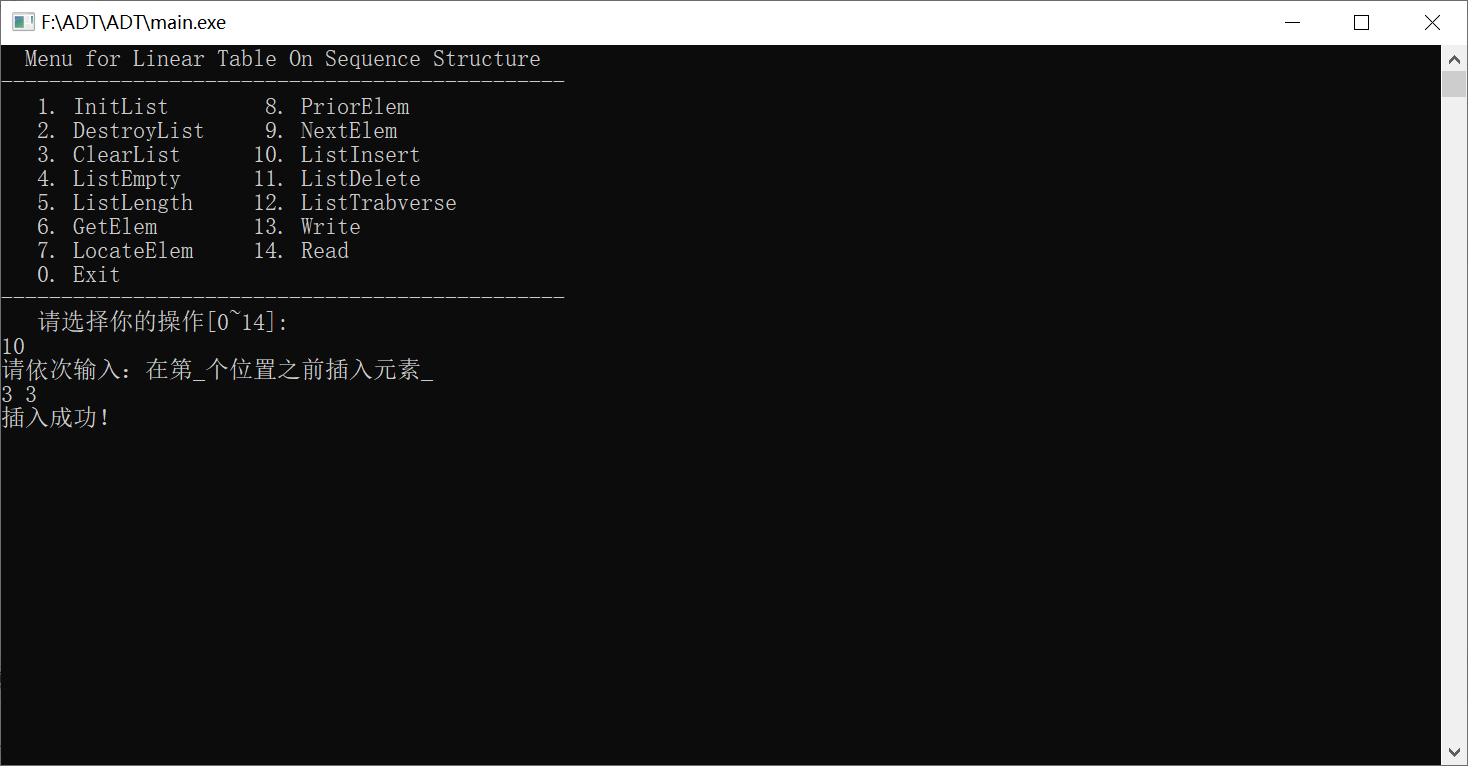


图2-8 测试-10插入元素

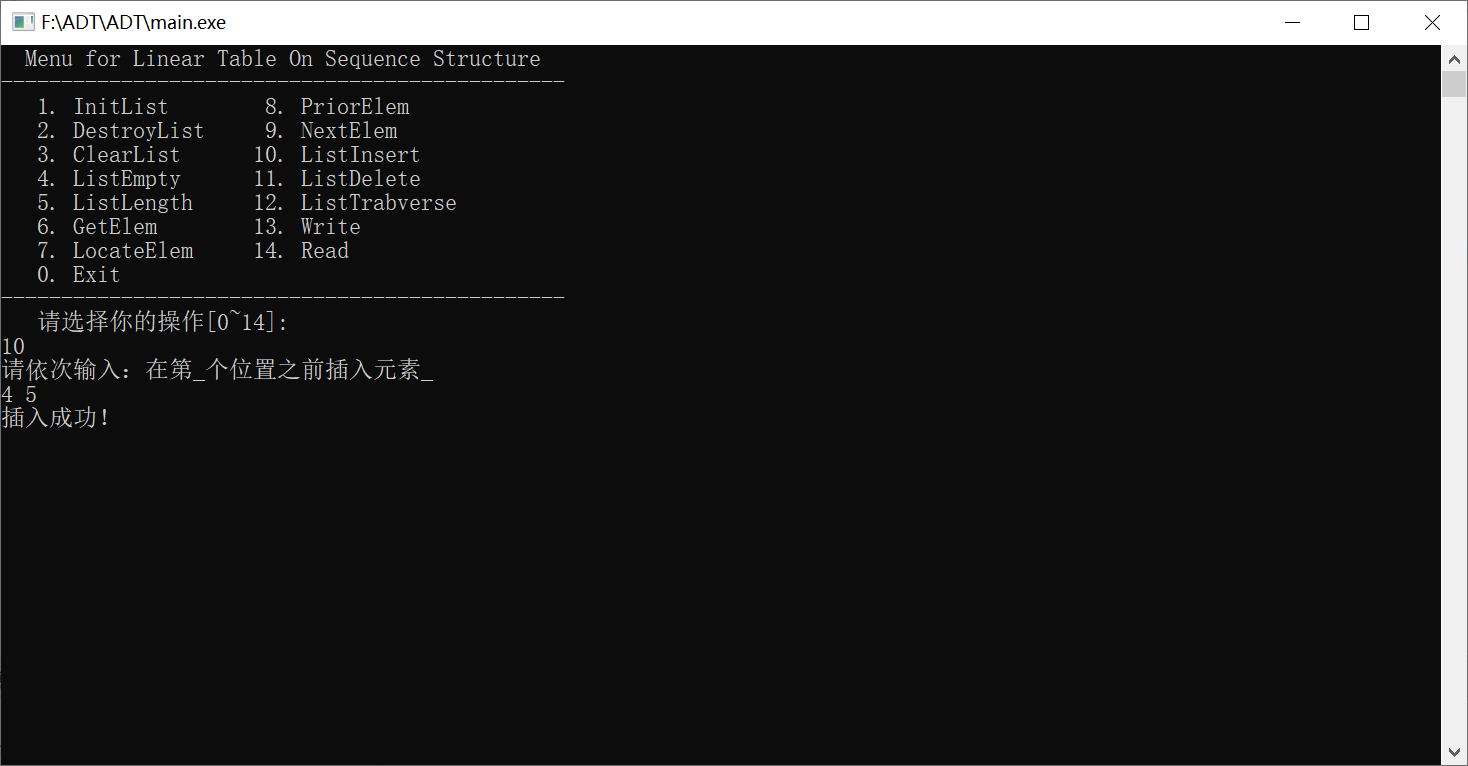


图2-9 测试-10插入元素

1. 执行功能4，判断线性表是否为空。

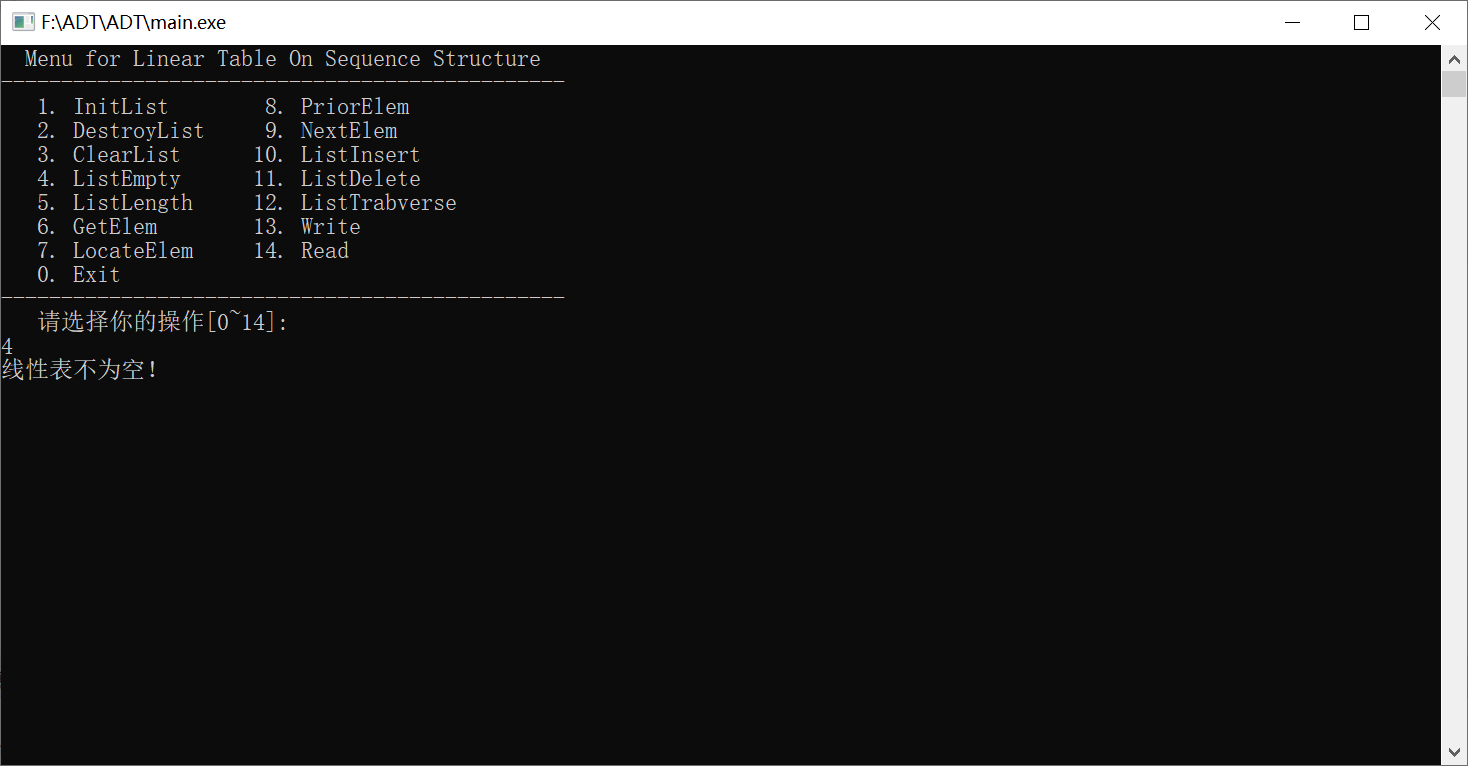


图2-10 测试-4判断线性表非空

1. 执行功能5，求线性表表长。

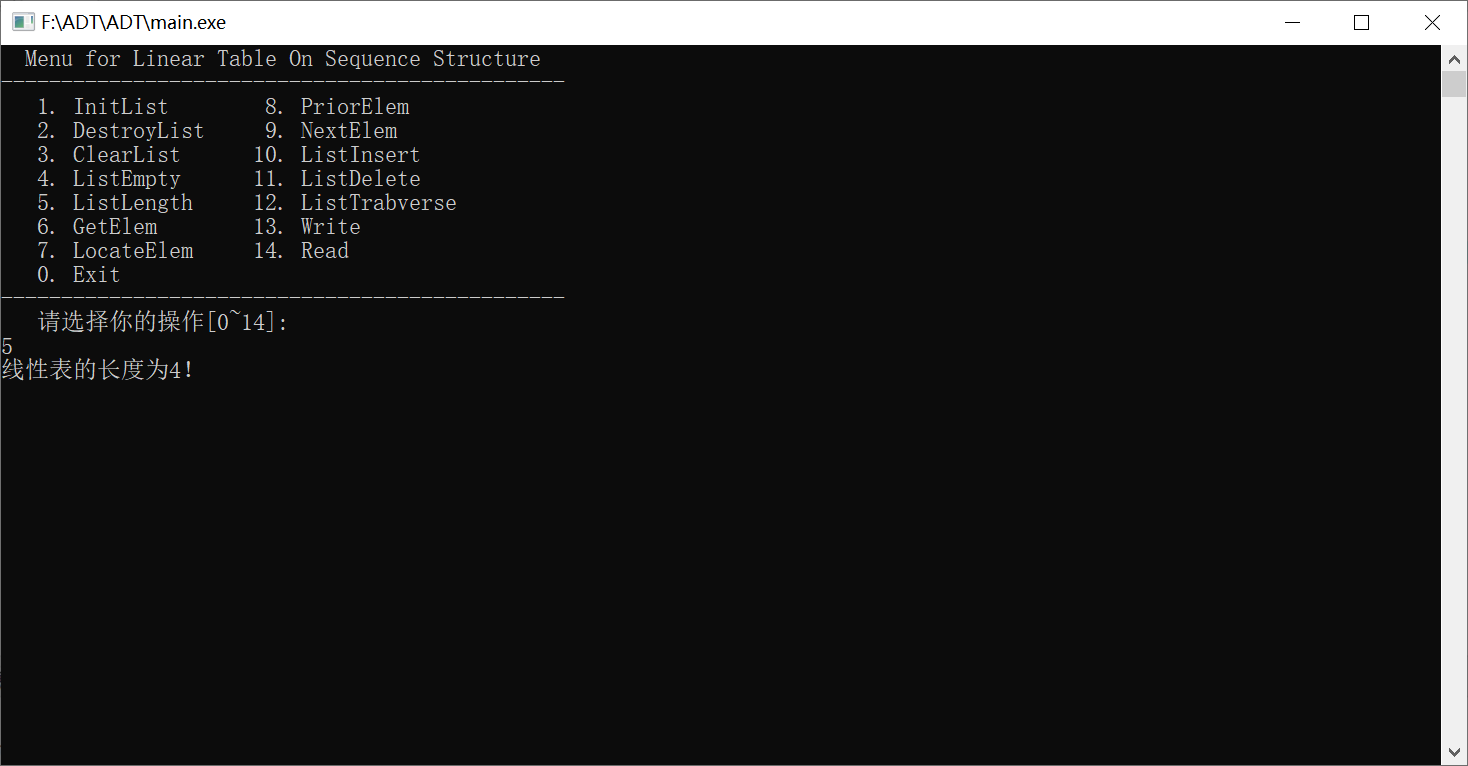


图2-11 测试-5求表长

1. 执行功能6，查找第3个元素的值。

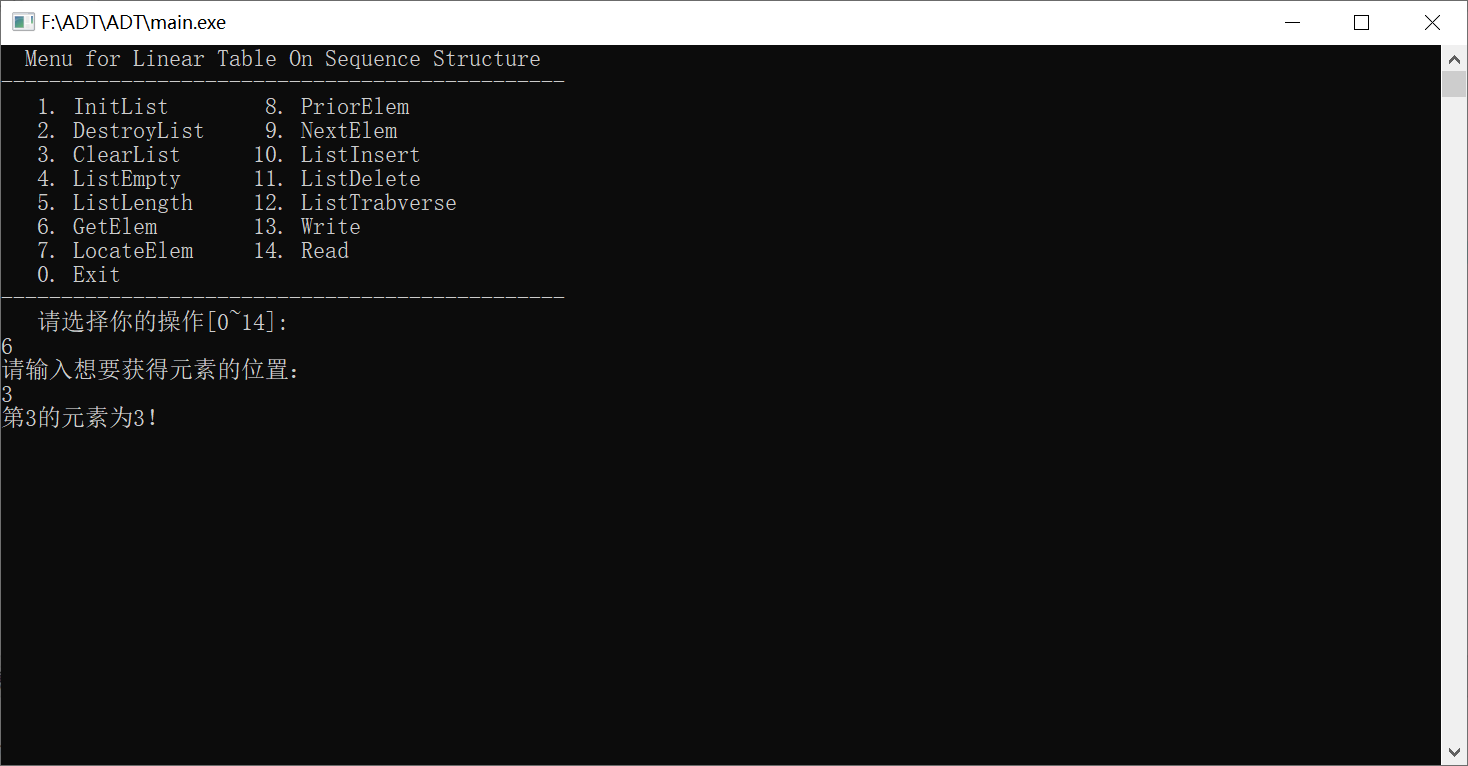


图2-12 测试-6获得元素

1. 执行功能7，查找元素2。

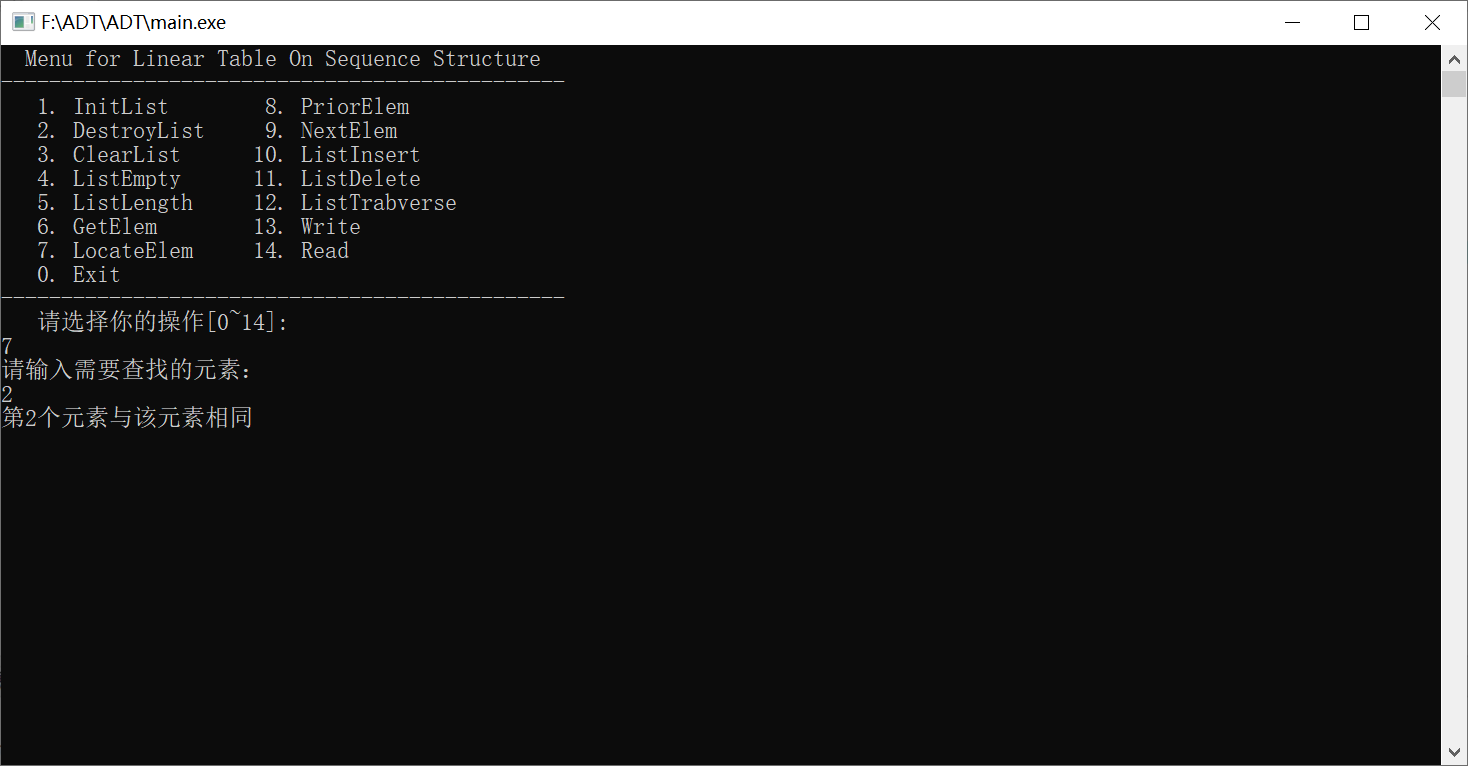


图2-13 测试-7查找元素

1. 执行功能8，获得3的前驱。

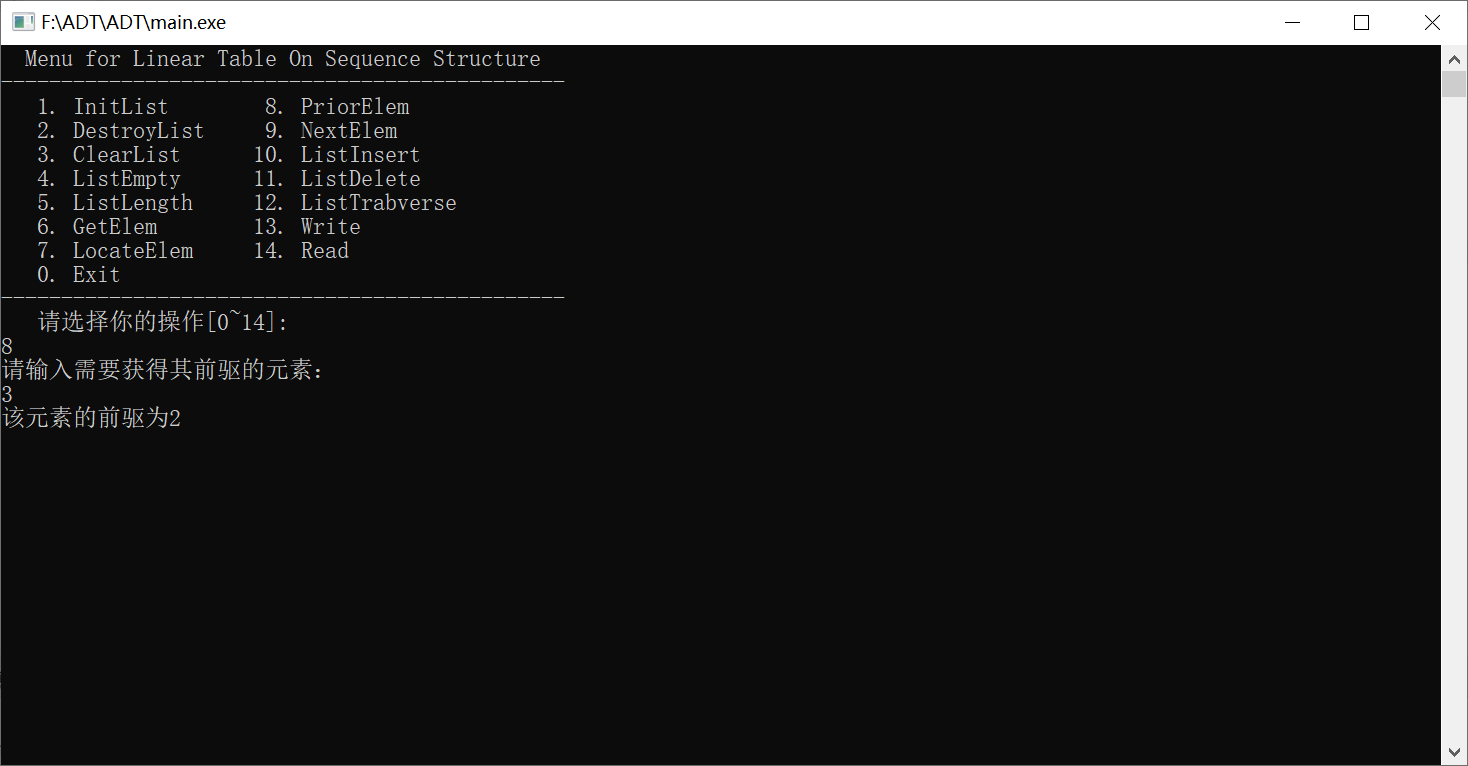


图2-14 测试-8获得前驱

1. 执行功能9，获得3的后继。

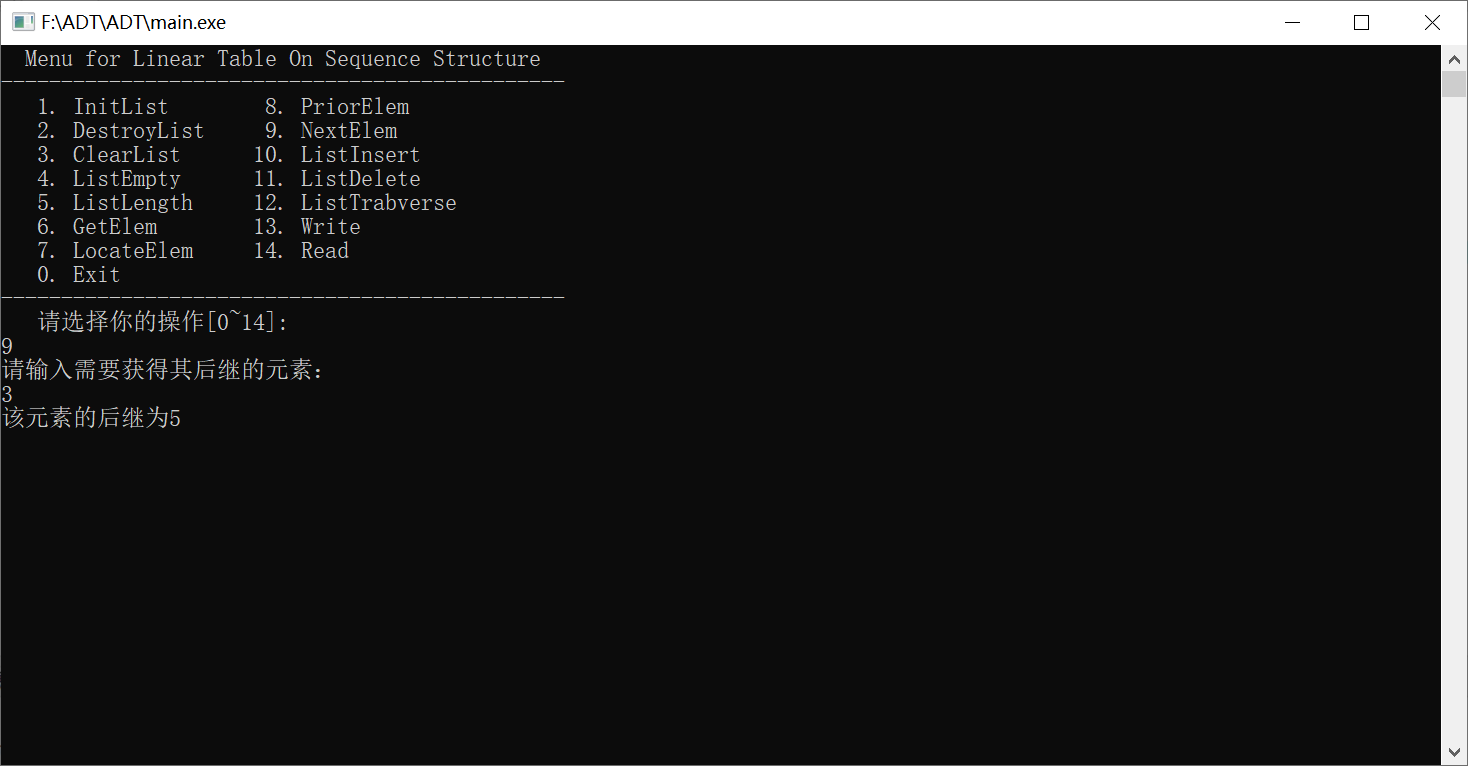


图2-15 测试-9获得后继

1. 执行功能10，在第四个位置插入元素4。

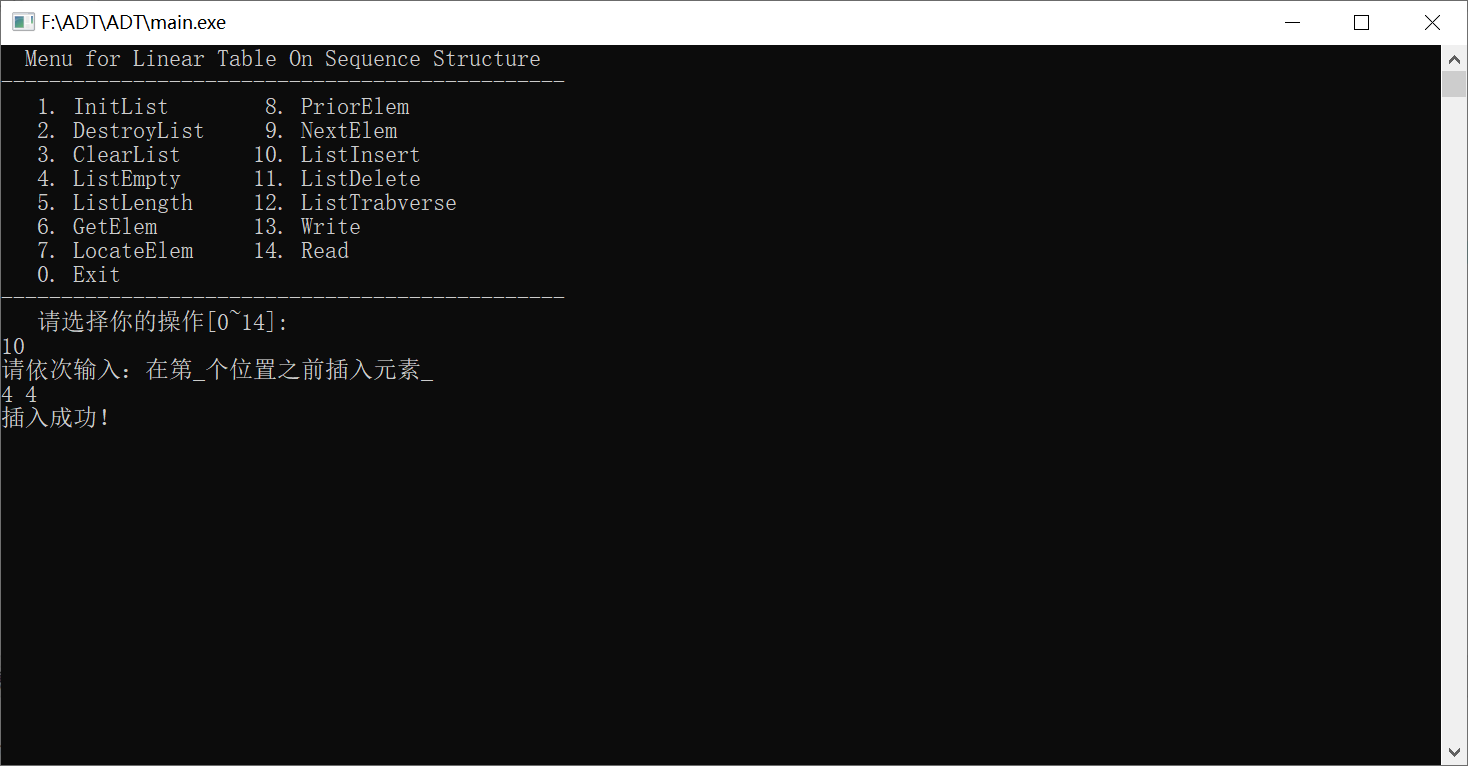


图2-16 测试-10插入元素

1. 执行功能11，删除第5个元素。

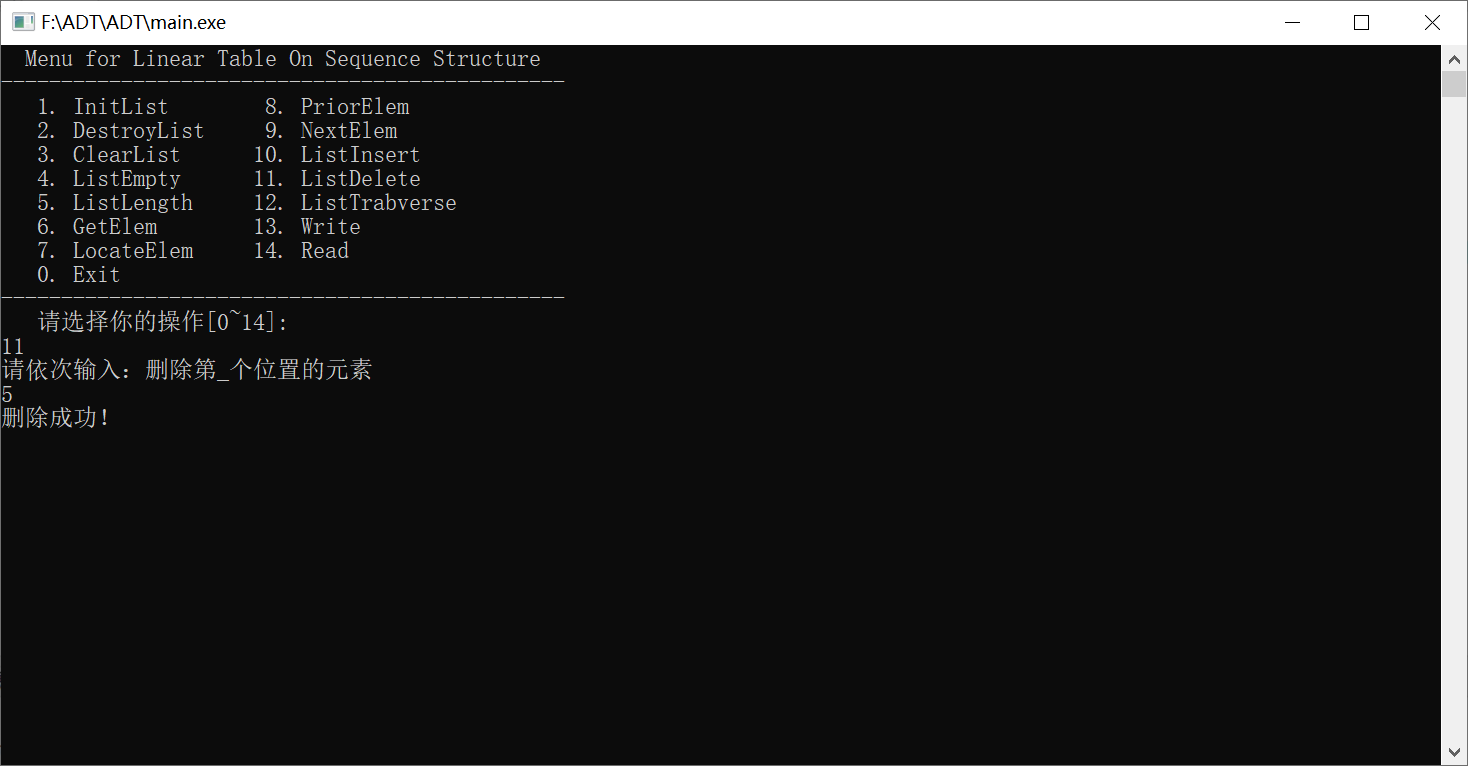


图2-17 测试-11删除元素

1. 执行功能12，遍历线性表1。

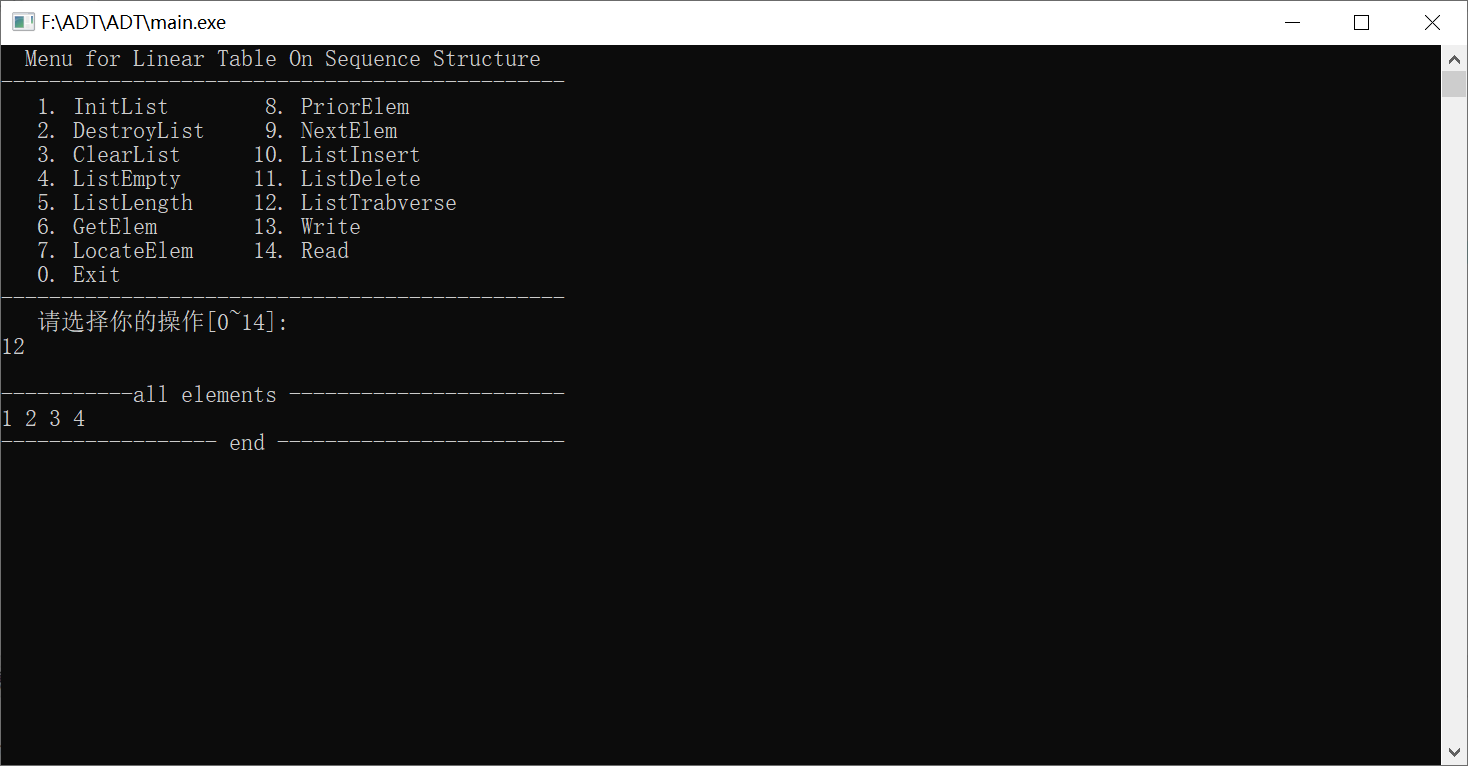


图2-18 测试-12遍历

1. 执行功能13，将数据写入文件data。

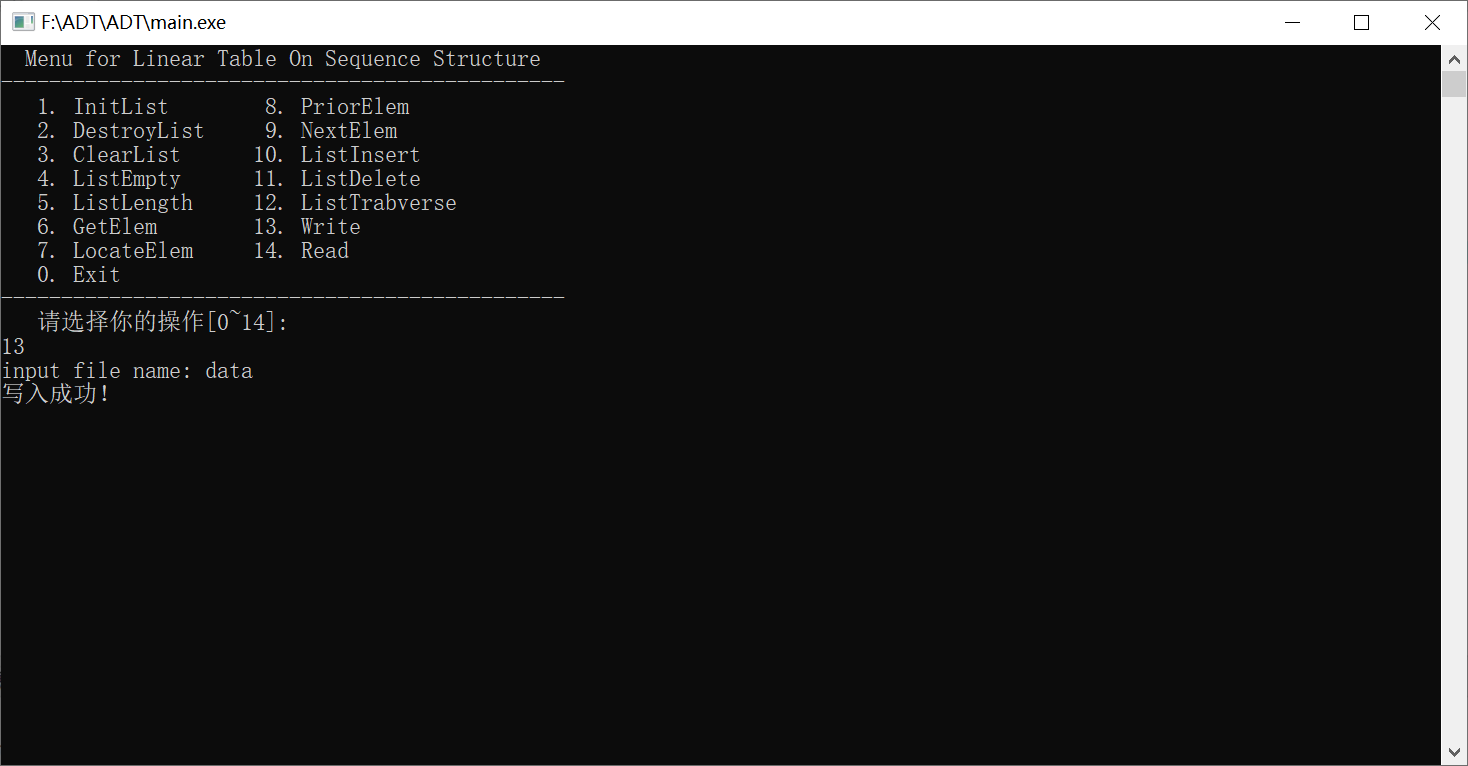


图2-19 测试-13写入文件

1. 执行功能3，清空线性表1。

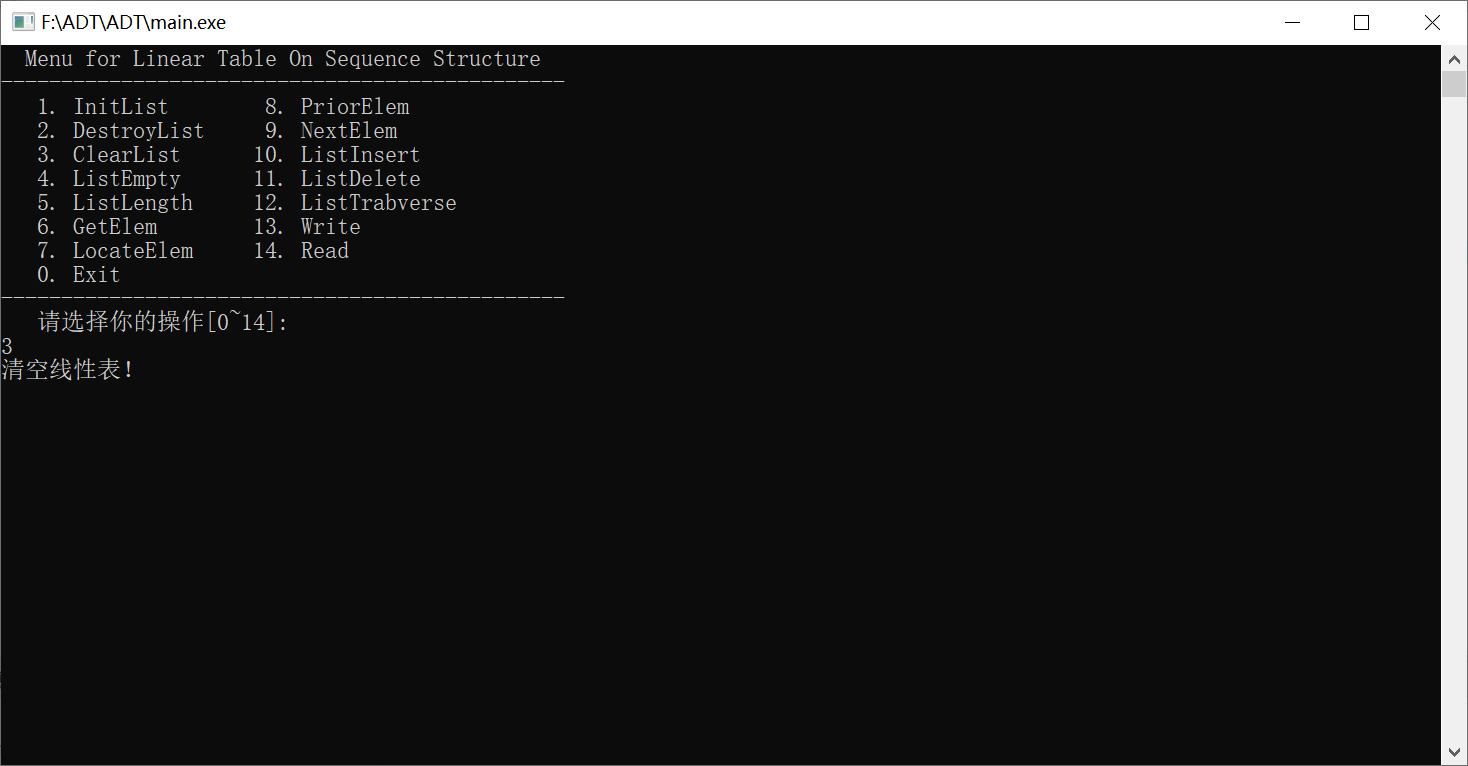


图2-10 测试-3清空线性表

1. 执行功能2，销毁线性表2，及销毁后其他操作不再可以进行。

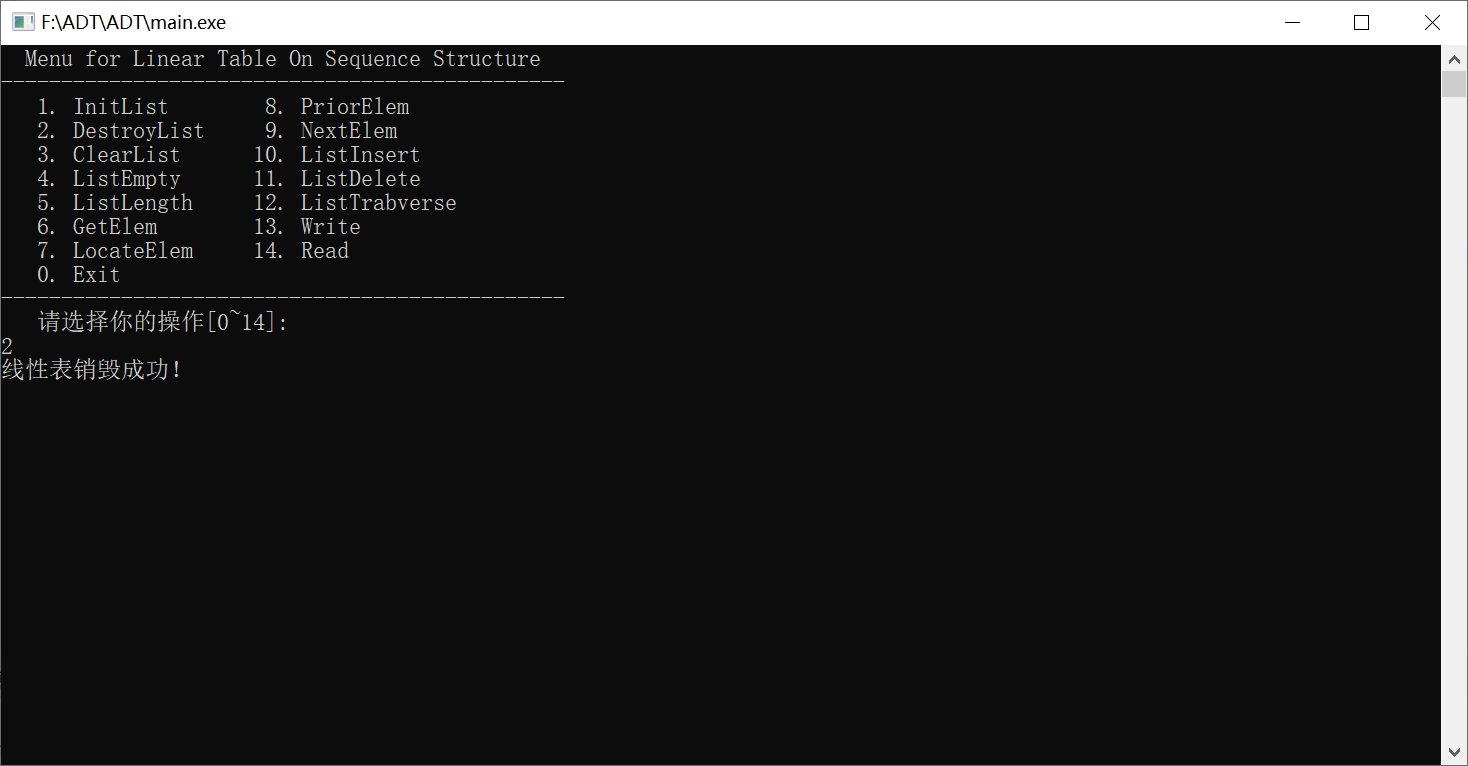


图2-21 测试-2销毁线性表

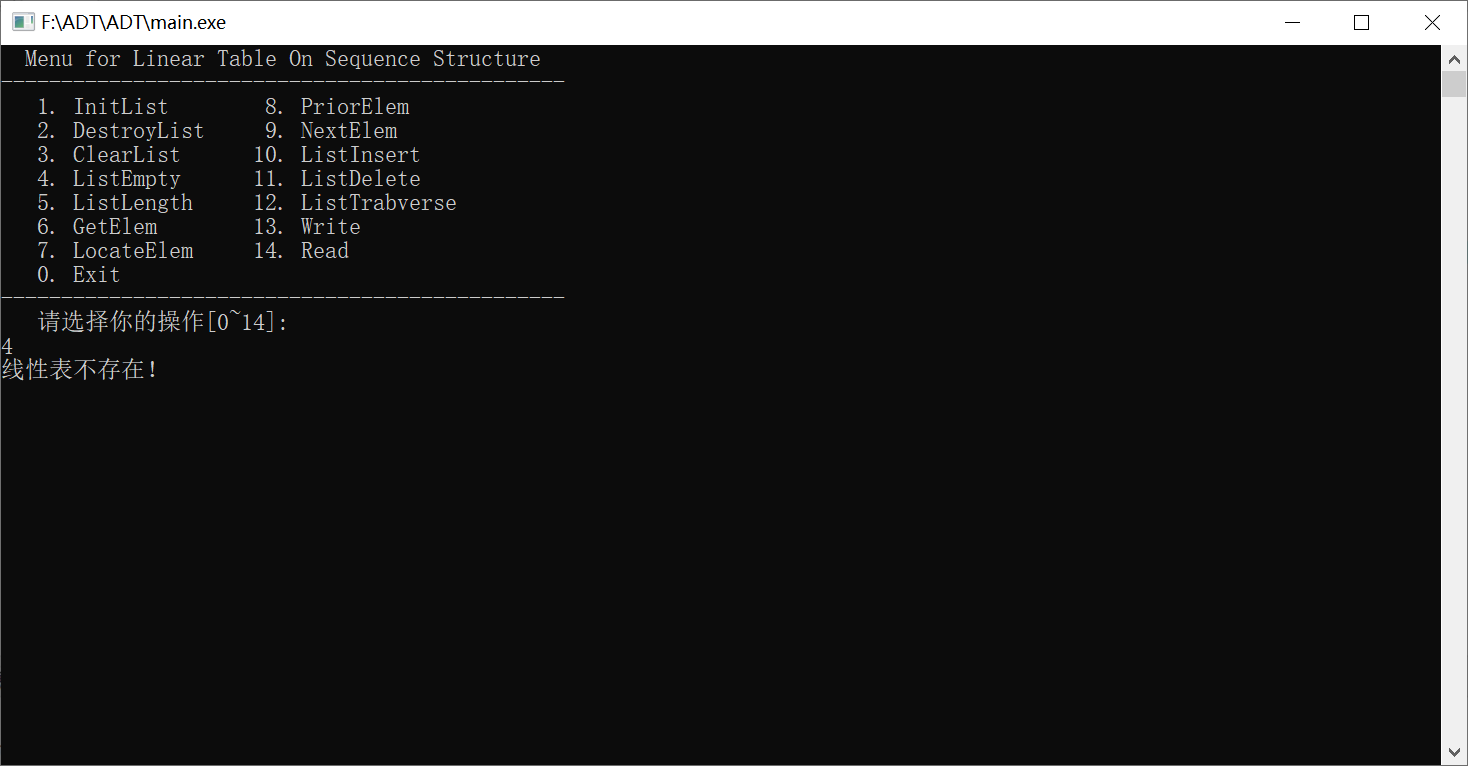


图2-22 测试-2销毁线性表后其他操作不可进行

1. 执行功能0，退出对线性表1的操作。

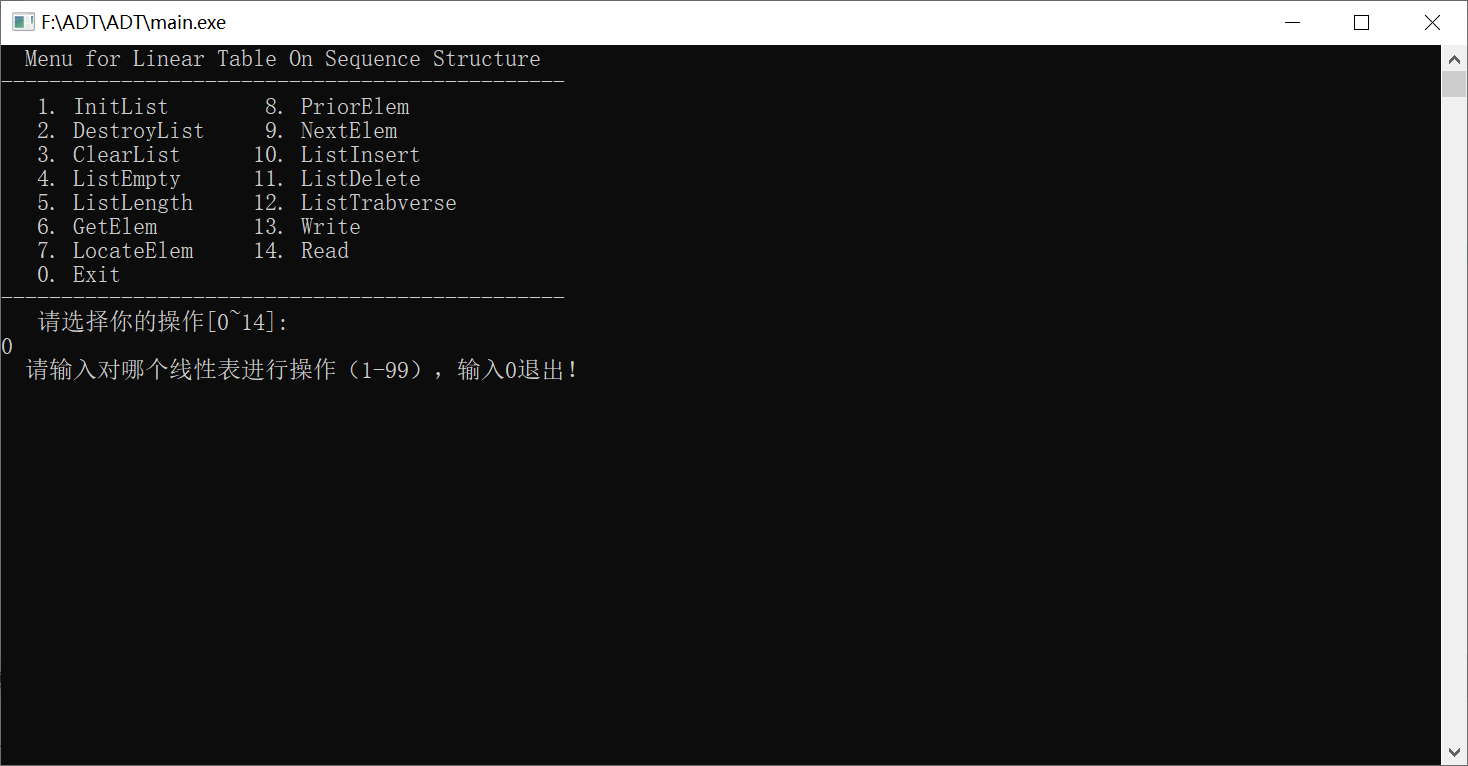
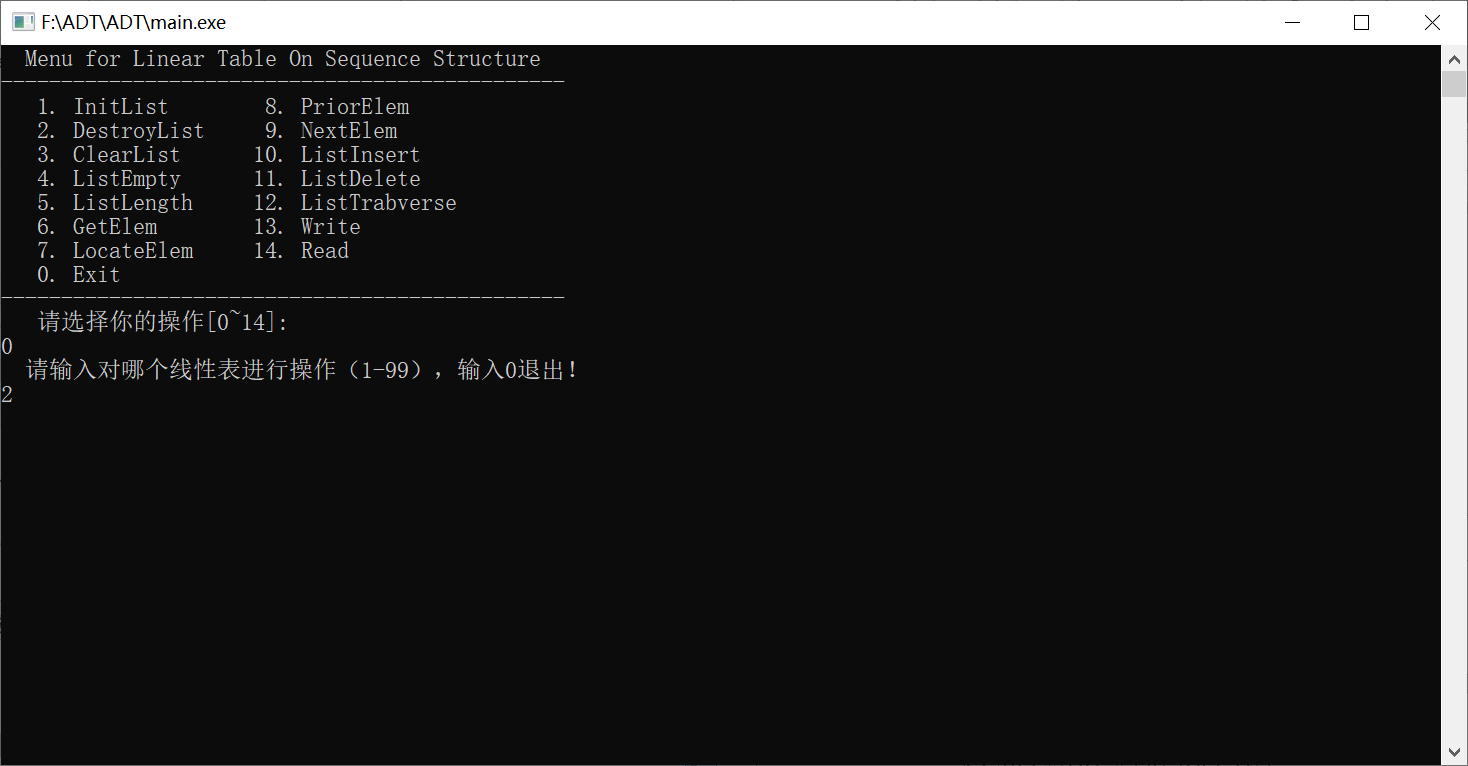
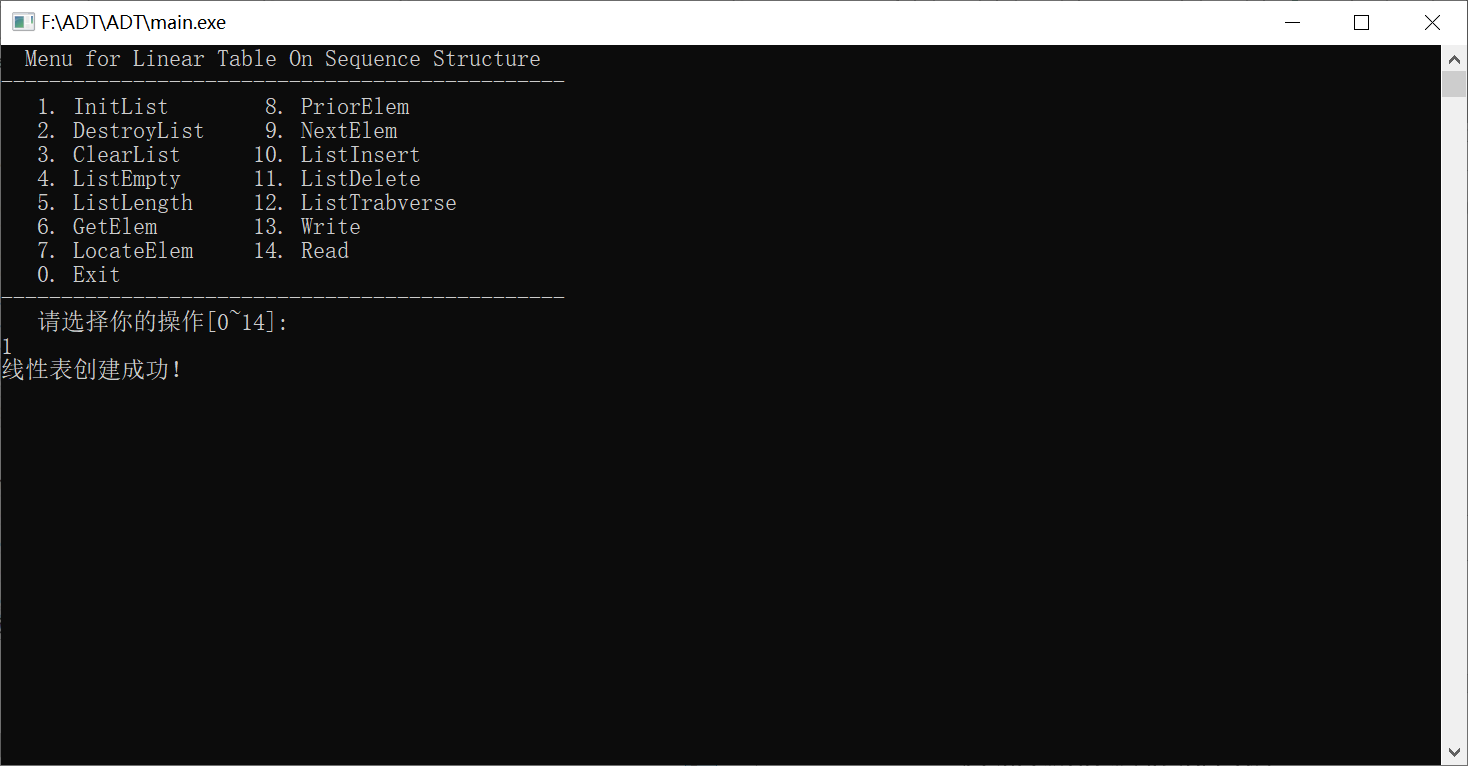


图2-23 测试-0退出操作

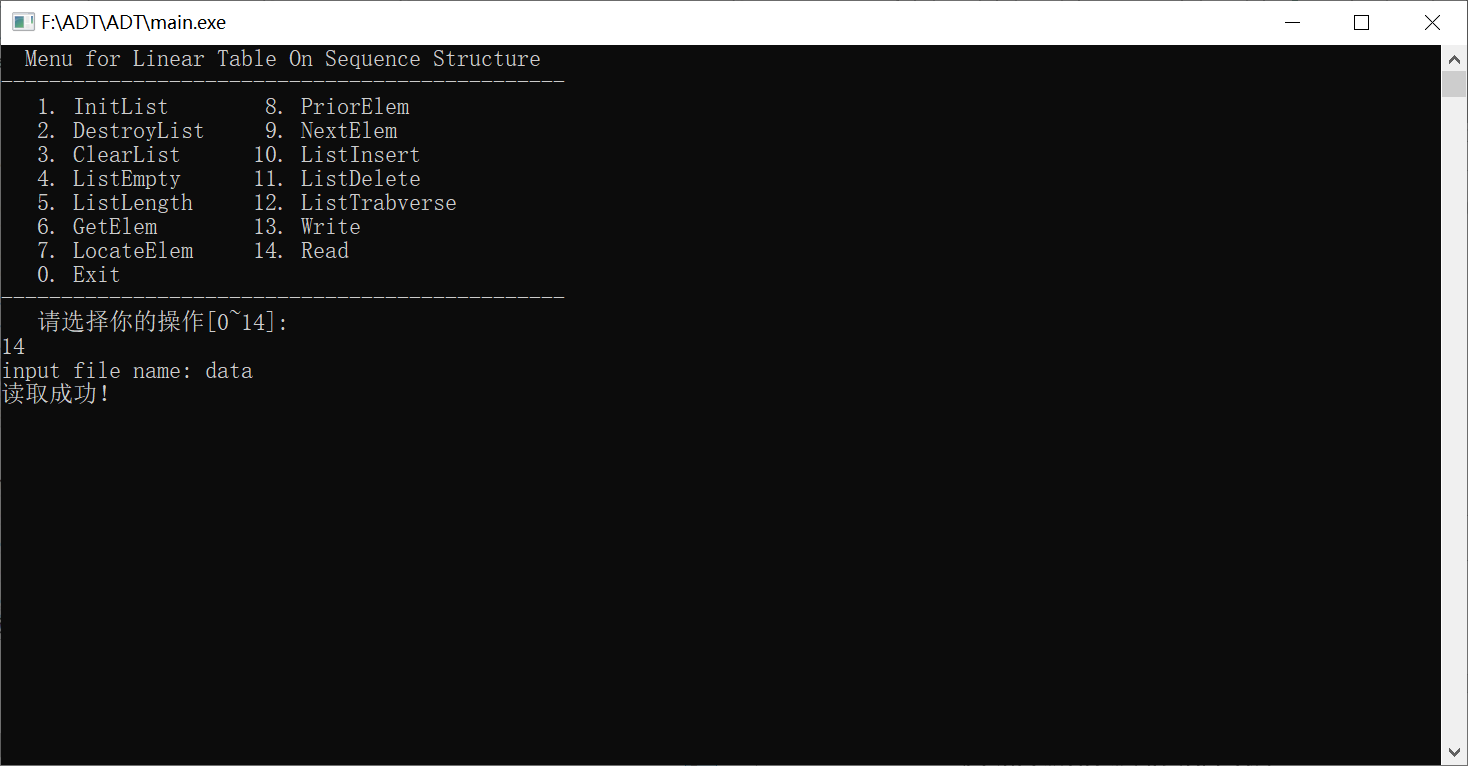
1. 重新选择线性表2。

图2-24 选择线性表2

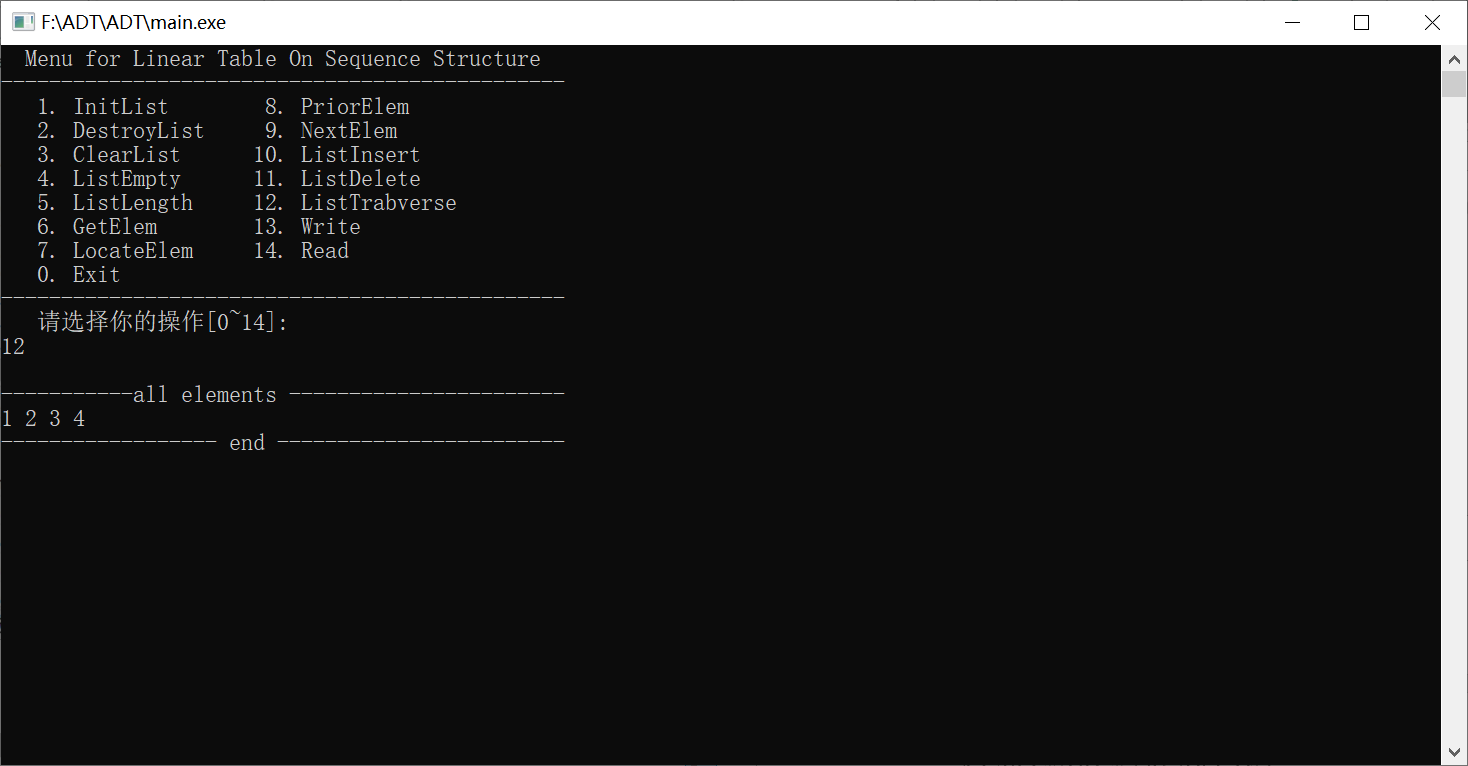
1. 执行功能1，创建线性表2。

图2-25 测试-1创建线性表2

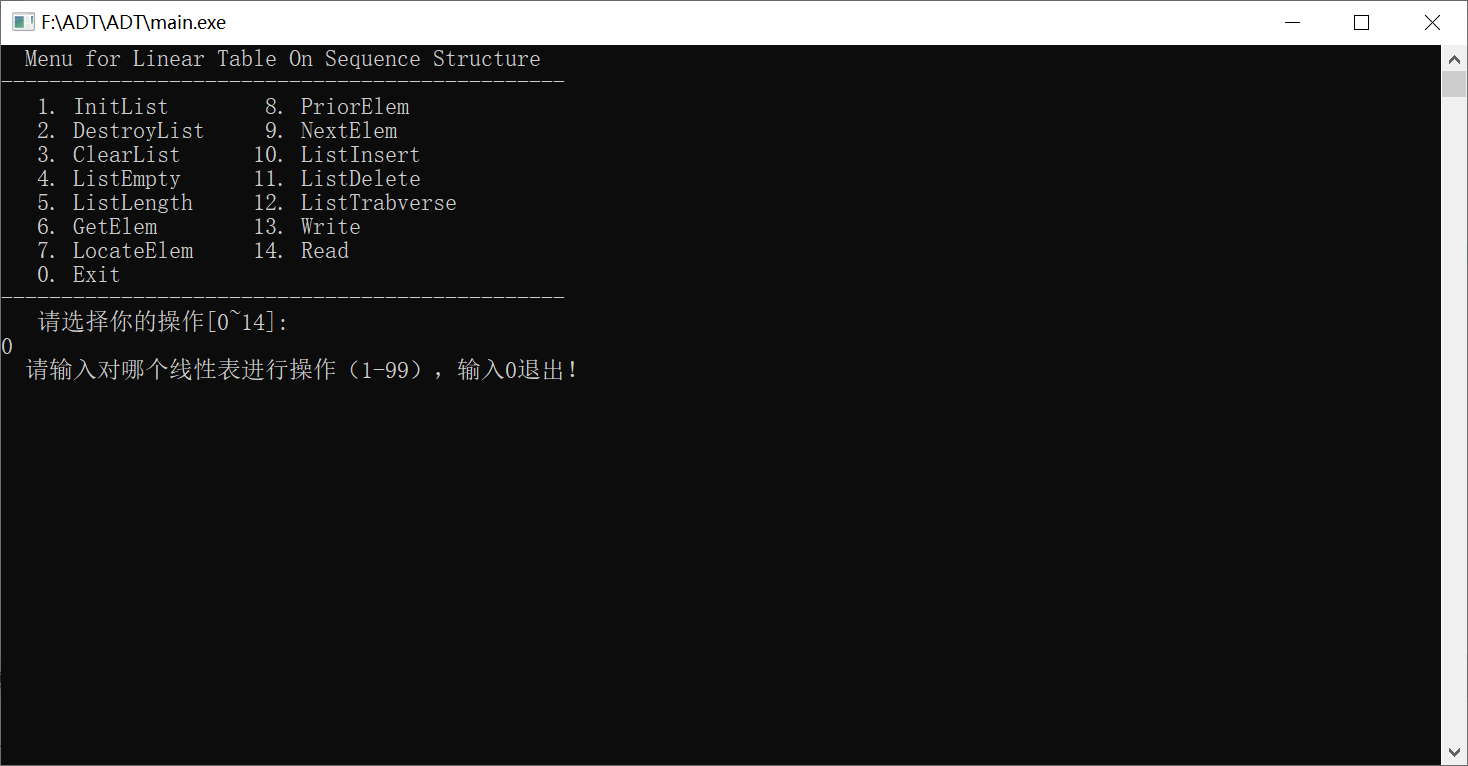
1. 执行功能14，从文件data中读取数据到线性表2。

图2-26 测试-14读入文件

1. 执行功能12，遍历线性表2。

图2-27 测试-12遍历线性表2

1. 执行功能0，退出对线性表2的操作。

图2-28 测试-0退出操作

1. 退出系统。

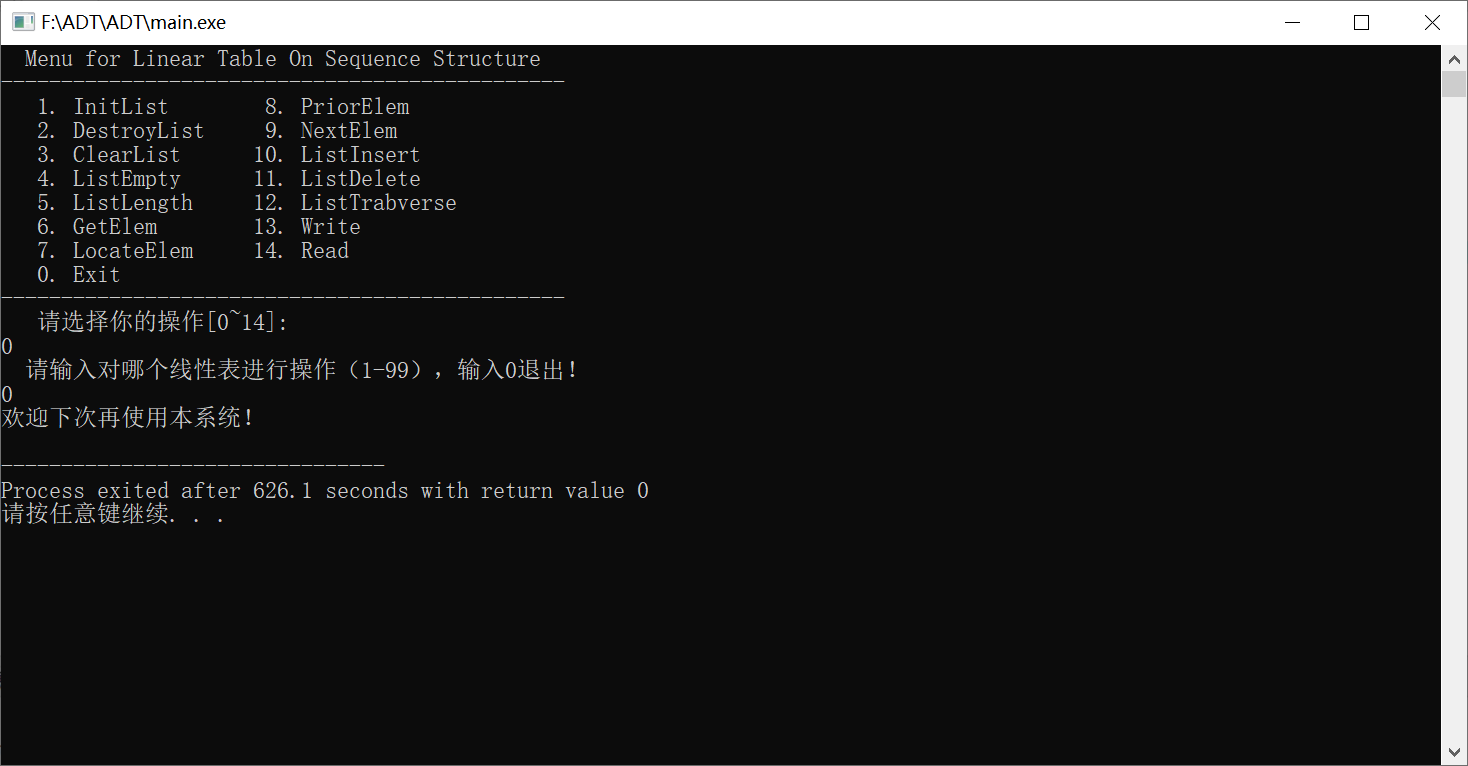


图2-29 退出系统

## 2.5 实验小结

1. 使用二级指针，很方便就修改了传入的结点一级指针的值。 如果用一级指针，则只能通过指针修改指针所指内容，却无法修改指针的值，也就是指针所指的内存块。所以创建链表和销毁链表需要二级指针引用。
2. 销毁和清空线性表时要将其无用空间全部释放，以免造成存储空间的浪费。
3. 读取文件时因为要将数据读到单链表中，所以在读取数据时要进行单链表的创建。
4. 写读取文件代码时对fread函数产生了疑惑，于是再一次查找了该函数用法。fread(void \*buffer, size\_t size, size\_t count, FILE \*stream)，其中 buffer是读取的数据存放的内存的指针（可以是数组，也可以是新开辟的空间，buffer就是一个索引），size是每次读取的字节数，count是读取次数，stream是要读取的文件的指针。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

采用链表的物理结构，构造一个具有菜单的功能演示系统，并且能对多个二叉树进行选择。在主函数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。

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。并给出输入的操作提示，以文件形式进行存储和加载，将生成的二叉树存入到相应的文件，也可以从文件中获取二叉树的关键字和数据进行操作。

3.1.1 基本运算

1. 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，definition为指向字符串的双重指针形式，字符串为带空子树的二叉树前序遍历序列；操作结果是按definition构造二叉树T。
2. 销毁二叉树：函数名称是DestroyBiTree(T)；初始条件是二叉树T已存在；传入二叉树指针的地址，二叉树存在头结点，将头结点一并销毁；操作结果是销毁二叉树T。
3. 清空二叉树：函数名称是ClearBiTree (T)；初始条件是二叉树T存在；二叉树存在头结点，操作结果是将二叉树T清空。
4. 判定空二叉树：函数名称是BiTreeEmpty(T)；初始条件是二叉树T存在；判断首结点是否存在，操作结果是若T为空二叉树则返回TRUE，否则返回FALSE。
5. 求二叉树深度：函数名称是BiTreeDepth(T)；初始条件是二叉树T存在；递归左右子树判断大小；操作结果是返回T的深度。
6. 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。
7. 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，调用查找结点函数寻找e结点；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。
8. 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，构造寻找父母结点的函数；先寻找父母结点再判断有无兄弟结点；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。
9. 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
10. 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。
11. 前序遍历：函数名称是PreOrderTraverse(T)；初始条件是二叉树T存在；操作结果：先序遍历，对每个结点遍历一次且一次，一旦调用失败，则操作失败。
12. 中序遍历：函数名称是InOrderTraverse(T)；初始条件是二叉树T存在；此函数用非递归；操作结果是中序遍历t，对每个结点遍历一次且一次，一旦调用失败，则操作失败。
13. 后序遍历：函数名称是PostOrderTraverse(T)；初始条件是二叉树T存在；操作结果是后序遍历t，对每个结点遍历一次且一次，一旦调用失败，则操作失败。
14. 按层遍历：函数名称是LevelOrderTraverse(T,Visit))；初始条件是二叉树T存在；操作结果是层序遍历t，对每个结点遍历一次且一次，一旦调用失败，则操作失败。

3.1.2 演示系统

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。二叉树读取函数时与前两次实验略有不同，此处不需要创建二叉树，可直接读取。

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，以高效保存线性表数据逻辑结构(D,{R})的完整信息；②需要设计ercs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。

## 3.2 系统设计

3.2.1 头文件、常量及结构说明

1. 头文件

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

1. 预定义常量

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define OVERFLOW -2

1. 类型表达式

typedef int status; //数据元素类型定义

typedef char ElemType;

typedef char kElemType;

1. 定义数据的物理结构如下：

typedef struct Node{//定义二叉树的结构类型

ElemType data;

char key;

struct Node \*lchild;

struct Node \*rchild;

}Tree;

同时因为需要实现多二叉树的操作，构造一个结构数组：Tree \*M[100];

3.2.2 算法设计

1. 创建二叉树：main函数里已创建了头结点；由main函数传入首结点和definition指针，数组里面为带空子树的前序遍历序列，‘#’为空结点。如果字符为‘#’，该结点不存在；若字符不为‘#’，则对结点进行分配空间并给关键字赋值。随后进行左子树和右子树的递归创建。此处传入的是二叉树指针的地址，需要用到双重指针，才能改变二叉树。
2. 销毁二叉树：传入二叉树指针的地址，即用双重指针修改二叉树。该函数调用清空二叉树函数进行空间的释放，最后再释放头指针。
3. 清空二叉树：传入首结点指针而不是头结点；如果二叉树本来就为空， 直接返回OK；如果不为空，则对其左子树和右子树的递归释放空间，再释放该结点的空间。返回OK。
4. 判定空二叉树：直接判断首结点是否存在；若存在则不为空返回TRUE，否则返回FALSE。
5. 求二叉树深度：函数传入的参数为首结点指针；声明三个变量depth、depthleft、depthright ；判断首结点是否为空，为空返回0；不为空进行左右子树的递归求深度，最后判断depthleft、depthright的大小，depth为较大的数加1；函数返回值为depth
6. 查找结点：传入头指针；声明tree类型指针p；如果结点为空返回NULL；否则判断该结点的关键字是否为e，如果不是，令p=LocateNode(T->lchild,e)，若p递归不为NULL，则返回p，否则返回LocateNode(T->rchild,e)。
7. 结点赋值：先调用 LocateNode(T,e)函数查找关键字为e的结点；如果返回值不为NULL，则对该结点赋值，否则返回ERROR；
8. 获得兄弟结点：传入首结点；首先构造一个查找父母结点的函数parents(T，e)，该函数与查找结点函数类似，不做赘述；查找兄弟结点时，先调用parents(T，e)，令p为其返回值，如果p不为NULL，则判断它的左右孩子是否符合条件，符合返回另外一个孩子，不符合则返回NULL。
9. 插入结点：如果结点存在，则判断关键字是否为e；如果为e，当LR为0时，该插入的结点c为左孩子，插入的结点c右孩子就为空，当LR为1时，该插入 的结点c为右孩子，插入的结点c左孩子就为空；如果不为e，则进行递归。
10. 删除结点：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。
11. 前序遍历：传入首结点； 如果结点存在，输出该结点的关键字和数据，再进行左子树和右子树的遍历，返回OK；
12. 中序遍历：传入首结点；采用非递归算法，与栈相似；声明一个Tree的指针数组。先将左孩子进栈，再将最上层的左孩子输出，T指向右孩子，循环直到栈为空并且T为空则结束。
13. 后序遍历：传入首结点； 如果结点存在，进行左子树和右子树的遍历，再输出该结点的关键字和数据，返回OK；
14. 按层遍历：传入首结点；采用非递归算法，与队相似；声明一个Tree的指针数组。先将T的关键字和数据输出，再将不为空的左孩子和右孩子进队，base++，每进队一个top++；直到base大于top结束。

3.2.3 理论分析

表3-1 函数算法复杂度分析

|  |  |
| --- | --- |
| 测试功能功能及序号 | 算法复杂度(右n个结点，以最坏情况看) |
| 1. CreatBiTree | O(n) |
| 2. DestroyBiTree | O(n) |
| 3. ClearBiTree | O(n) |
| 4. BiTreeEmpty | O(1) |
| 5. BiTreeDepth | O(n) |
| 6. LocateNode | O(n) |
| 7.Assign | O(n) |
| 8. GetSibling | O(n) |
| 9. InsertNode | O(n) |
| 10. DeleteNode | O(n) |
| 11. PreOrderTraverse | O(n) |
| 12. InOrderTraverse | O(n) |
| 13. PostOrderTraverse | O(n) |
| 14. LevelOrderTraverse | O(n) |

## 3.3 系统实现

3.3.1 系统演示

整个系统的流程及结构如下图所示。



图3-1 系统结构图

3.3.2 部分函数功能实现

1. 删除结点： 在删除结点时构造了三个函数DeleteNode(T,e) 、Deletel(T) 、和Deleter(T)；在DeleteNode(T,e)里判断删除左孩子还是右孩子并进行递归操作。在Deletel(T)里删除左孩子，分三种情况：结点的度为0时直接删除，结点的度为1时将唯一的子树顶替该结点，结点度为2时先遍历到被删除结点的左子树的最右边的孩子，则被删除结点的右子树为该结点的右孩子。

status DeleteNode(Tree \*T,kElemType e){//在删除关键字为e的结点

if((T->lchild)&&(T->lchild->key==e)){

Deletel(T);

return OK;

}

else if((T->rchild)&&(T->rchild->key==e)){

Deleter(T);

return OK;

}

else{

if(T->lchild)

if(DeleteNode(T->lchild,e)==OK)

return OK;

if(T->rchild)

if(DeleteNode(T->rchild,e)==OK)

return OK;

}

}

status Deletel(Tree \*T){//删除结点

Tree \*p;

Tree \*q;

Tree \*t;

q=T;

p=T->lchild;

t=p->lchild;

if(p->lchild==NULL){

if(p->rchild==NULL){//结点度为0

T->lchild=NULL;

return OK;

}

else{

p=p->rchild;//结点度为1，有右子树

q->lchild=p;

return OK;

}

}

else{

if(p->rchild==NULL){//结点度为1，有左子树

p=p->lchild;

q->lchild=p;

return OK;

}

else{//结点度为2

while(t->rchild!=NULL)

t=t->rchild;

t->rchild=p->rchild;

p=p->lchild;

q->lchild=p;

return OK;

}

}

}

status Deleter(Tree \*T){//删除结点

Tree \*p;

Tree \*q;

Tree \*t;

q=T;

p=T->rchild;

if(p->lchild==NULL){

if(p->rchild==NULL){//结点度为0

T->rchild=NULL;

return OK;

}

else{

p=p->rchild;//结点度为1，有右子树

q->rchild=p;

return OK;

}

}

else{

if(p->rchild==NULL){//结点度为1，有左子树

p=p->lchild;

q->rchild=p;

return OK;

}

else{//结点度为2

t=p->lchild;

while(t->rchild!=NULL)

t=t->rchild;

t->rchild=p->rchild;

p=p->lchild;

q->rchild=p;

return OK;

}

}



图3-2 deletel函数流程图

3.3.3 用文件读写

文件的读取可用多个二叉树进行操作，例如用二叉树1写入文件，再创建二叉树2读取数据，使之与二叉树1 相同。

1. 文件的写入：在Write函数中打开文件，并调用TreeWrite。调用完后关闭文件。

在TreeWrite函数中，如果该结点为空，写入‘ ’，如果结点不为空，则依次写入结点的关键字和数据值，在依次进行递归左子树和右子树。

写入文件：

status Wirte(Tree \*T) {

FILE \*fp=NULL;

char c='#';

char filename[30];

printf("input file name: ");

scanf("%s",filename);

if ((fp=fopen(filename,"wb"))==NULL){

printf("File open error\n ");

return FALSE;

}

//写文件的方法

TreeWrite(T->lchild,fp);//遍历二叉树,写入文件

fclose(fp);

return OK;

}

status TreeWrite(Tree \*T,FILE \*fp){

char c=' ';

if(T==NULL){

fwrite(&c,sizeof(kElemType),1,fp);

return;

}

else {

fwrite(&(T->key),sizeof(kElemType),1,fp);

fwrite(&(T->data),sizeof(kElemType),1,fp);

TreeWrite(T->lchild,fp);

TreeWrite(T->rchild,fp);

}

}

1. 文件的读取：通过Read函数打开文件，构造头结点，将文件中的数据读取到tree[]数组中去，再调用TreeRead函数。

设置一个全局变量Arch，来记录数组进行到哪一位。若为空字符，则该结点为空；若不为空，则读取该结点的关键字和数据构建树，每判定一个字符，将Arch增加1。最后进行左子树与右子树递归创建。

刚开始时，TreeRead函数传入了一个双重指针，但是由于tree是一个数组，此处&数组名与一级指针意义并不一样，所以不能直接对数组名取址当成双重指针使用。而创建函数是因为main函数里声明的definition为一个指针，可以对其取址传入双重指针。

int Arch;

status Read(Tree \*\*T) {

FILE \*fp;

char tree[100];

char single;

int i=0;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

//读文件的方法

if ((fp=fopen(filename,"rb"))==NULL){

printf("File open error\n ");

return FALSE;

}

(\*T)=(Tree \*)malloc(sizeof(Tree));//构造头结点

(\*T)->rchild=NULL;

(\*T)->key=0;

while(fread(&single,sizeof(ElemType),1,fp))

tree[i++]=single;

tree[i]='#';

Arch=0;

TreeRead(&((\*T)->lchild),&tree);//这里从文件中逐个读取数据元素恢复二叉树

fclose(fp);

return OK;

}

**函数写法1**：

status TreeRead(Tree \*\*T,char \*definition){

if(definition[Arch]=='#')

return OK;

if(definition[Arch]==' '){//结点不存在

(\*T)=NULL;

Arch++;

}

else{

if(!((\*T)=(Tree \*)malloc(sizeof(Tree))))

exit(OVERFLOW);

else{

(\*T)->key=definition[Arch];//赋关键字

Arch++;

(\*T)->data=definition[Arch];

Arch++;

TreeRead(&((\*T)->lchild),definition);

TreeRead(&((\*T)->rchild),definition);

}

}

return OK;

}

**函数写法2：(将definition改成一级指针)**

status TreeRead(Tree \*\*T,char \*definition){

if(\*definition=='#')

return OK;

if(\*definition==' '){//结点不存在

(\*T)=NULL;

definition++;

}

else{

if(!((\*T)=(Tree \*)malloc(sizeof(Tree))))

exit(OVERFLOW);

else{

(\*T)->key=\*definition;//赋关键字

definition++;

(\*T)->data=\*\*definition;

definition++;

TreeRead(&((\*T)->lchild),definition);

TreeRead(&((\*T)->rchild),definition);

}

}

return OK;

}

## 3.4 系统测试

3.4.1 测试计划

表3-2 测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能功能及序号 | 测试的二叉树 | 输入的参数 | 预计结果 | 二叉树实际状态 |
| 选择二叉树 | 1 | 1 | 对二叉树1进行操作 | 选择二叉树1，并未创建 |
| 1. CreatBiTree | 1 | ABC DEH G F # | 创建二叉树1 | 创建了二叉树 |
| 7. Assign  （多次调用） | 1 | A 1；B 2；C 3；D 4;E 5;F 6;G 7;H 8; | 二叉树1 写入数据 | 二叉树各结点赋值 |
| 4. BiTreeEmpty | 1 | 无 | 二叉树非空 | 状态不变 |
| 5. BiTreeDepth | 1 | 无 | 二叉树长度 | 状态不变 |
| 6. LocateNode | 1 | 想要获得的结点的关键字：D | 关键字为D的结点 | 状态不变 |
| 8. GetSibling | 1 | 需要获得兄弟结点的关键字：D | 该结点的兄弟结点 | 状态不变 |
| 9. InsertNode | 1 | 要插入的位置的关键字、左(0)还是右(1)、要插入结点的关键字、数据：D 1 I 9 | 插入结点 | I结点插入D结点的右子树 |
| 10. DeleteNode | 1 | 需要删除的结点的关键字：D | 删除结点 | D结点删除，E结点为B结点的右子树，D的右子树变为G的右子树 |
| 11. PreOrderTraverse | 1 | 无 | 前序遍历 | 状态不变 |
| 12. InOrderTraverse | 1 | 无 | 中序遍历 | 状态不变 |
| 13. PostOrderTraverse | 1 | 无 | 后序遍历 | 状态不变 |
| 14. LevelOrderTraverse | 1 | 无 | 按层遍历 | 状态不变 |
| 15. Write | 1 | 文件名：DATA | 将二叉树1写入文件 | 状态不变 |
| 3. ClearBiTree | 1 | 无 | 清空二叉树1 | 表长为0 |
| 2. DestroyBiTree | 1 | 无 | 销毁二叉树1 | 二叉树空间被释放 |
| 0. Exit | 1 | 无 | 退出操作 | 无二叉树 |
| 选择二叉树 | 2 | 2 | 对二叉树2进行操作 | 选择二叉树2，并未创建 |
| 16. Read | 2 | 文件名：DATA | 从该文件读入数据 | 二叉树2序列与销毁前的二叉树1相同 |
| 11. PreOrderTraverse | 2 | 无 | 遍历二叉树2 | 二叉树2序列：1，2，3，4 |
| 退出选择二叉树 | 无 | 退出输入0 | 退出选择 | 运行结束 |

3.4.2 测试结果

执行3.4.1表格的测试计划，运行结果截图如下所示。

1. 选择二叉树1，进入操作。

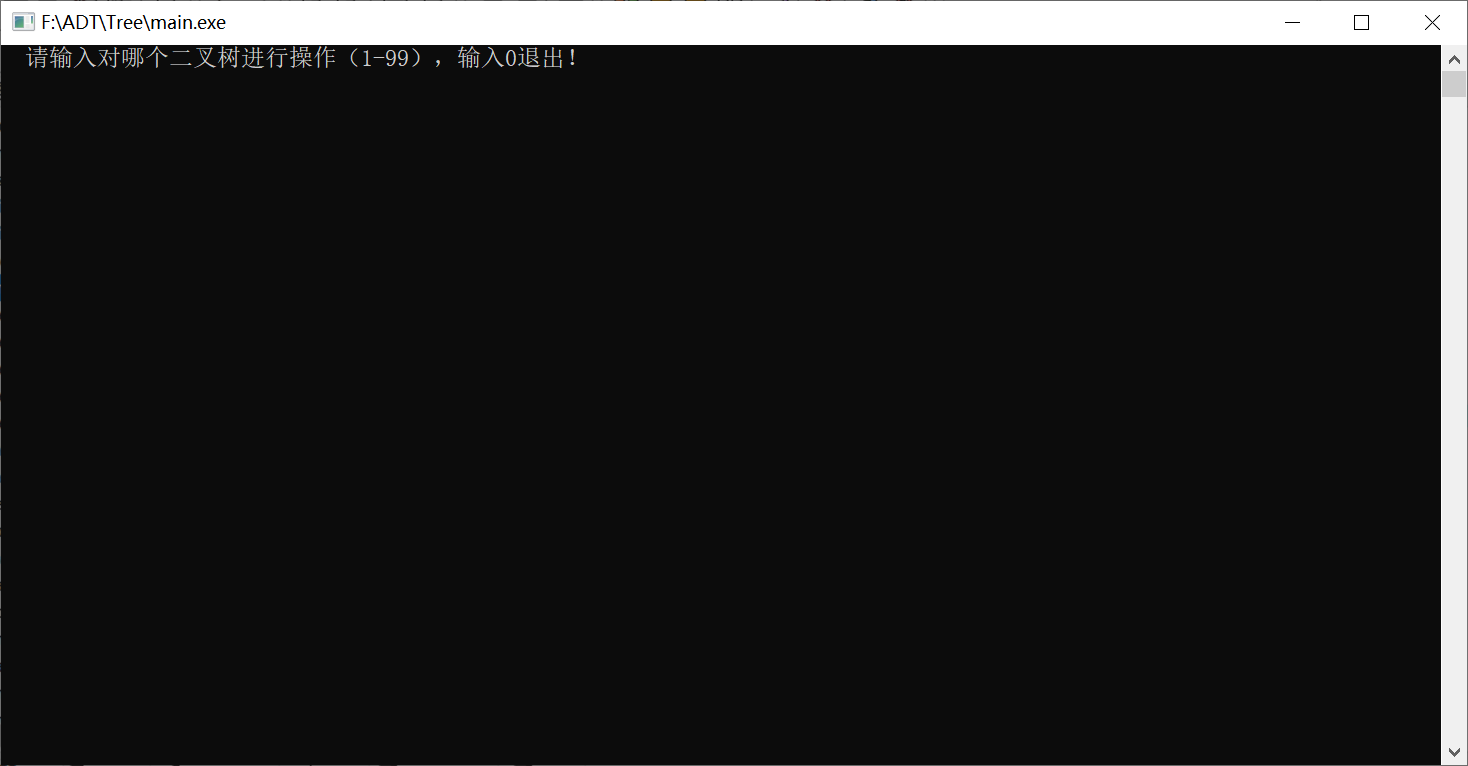


图3-3 测试-选择二叉树

1. 执行功能1，创建二叉树1。

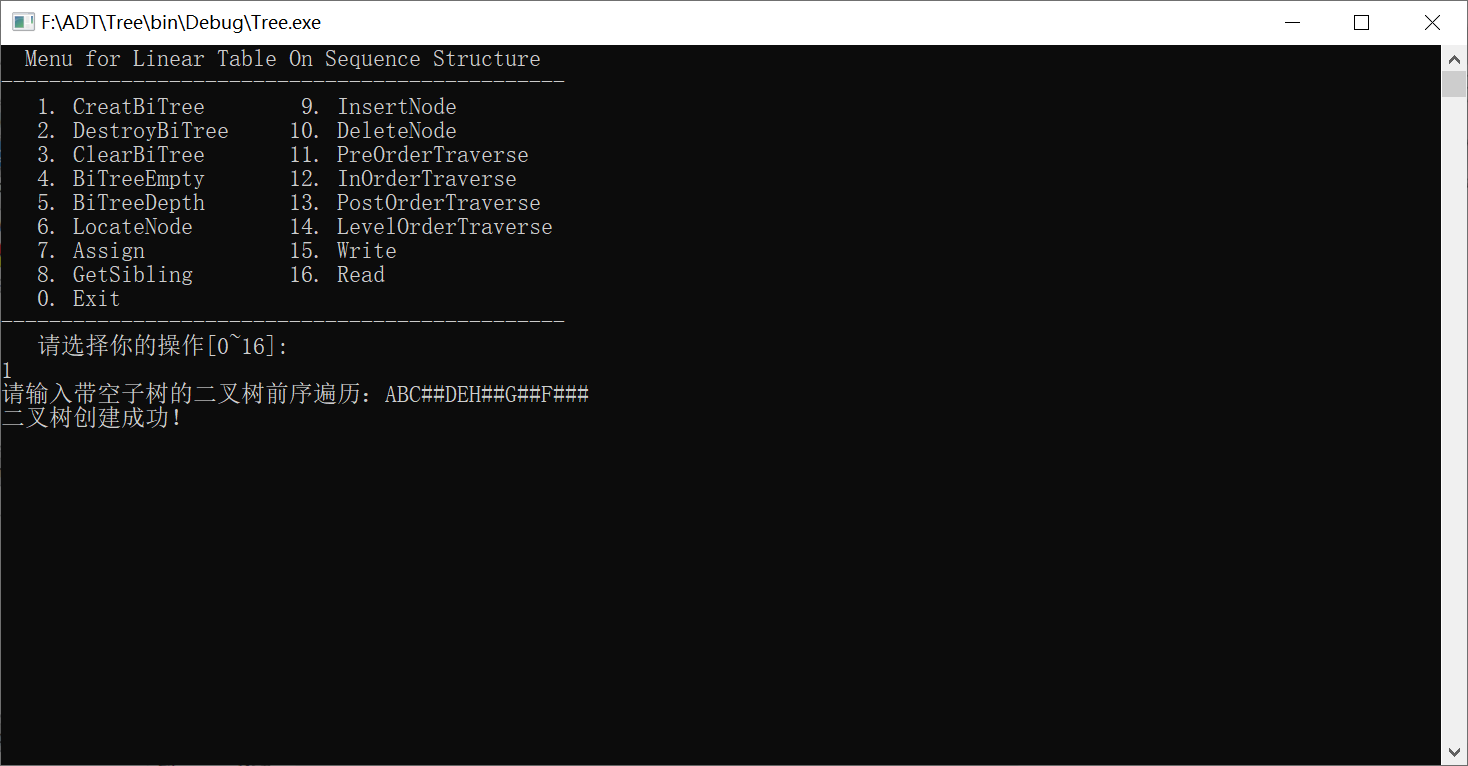


图3-4 测试-创建二叉树

1. 多次执行功能7，给结点A、B、C、D、E、F、G、H赋值1、2、3、4、5、6、7、8。

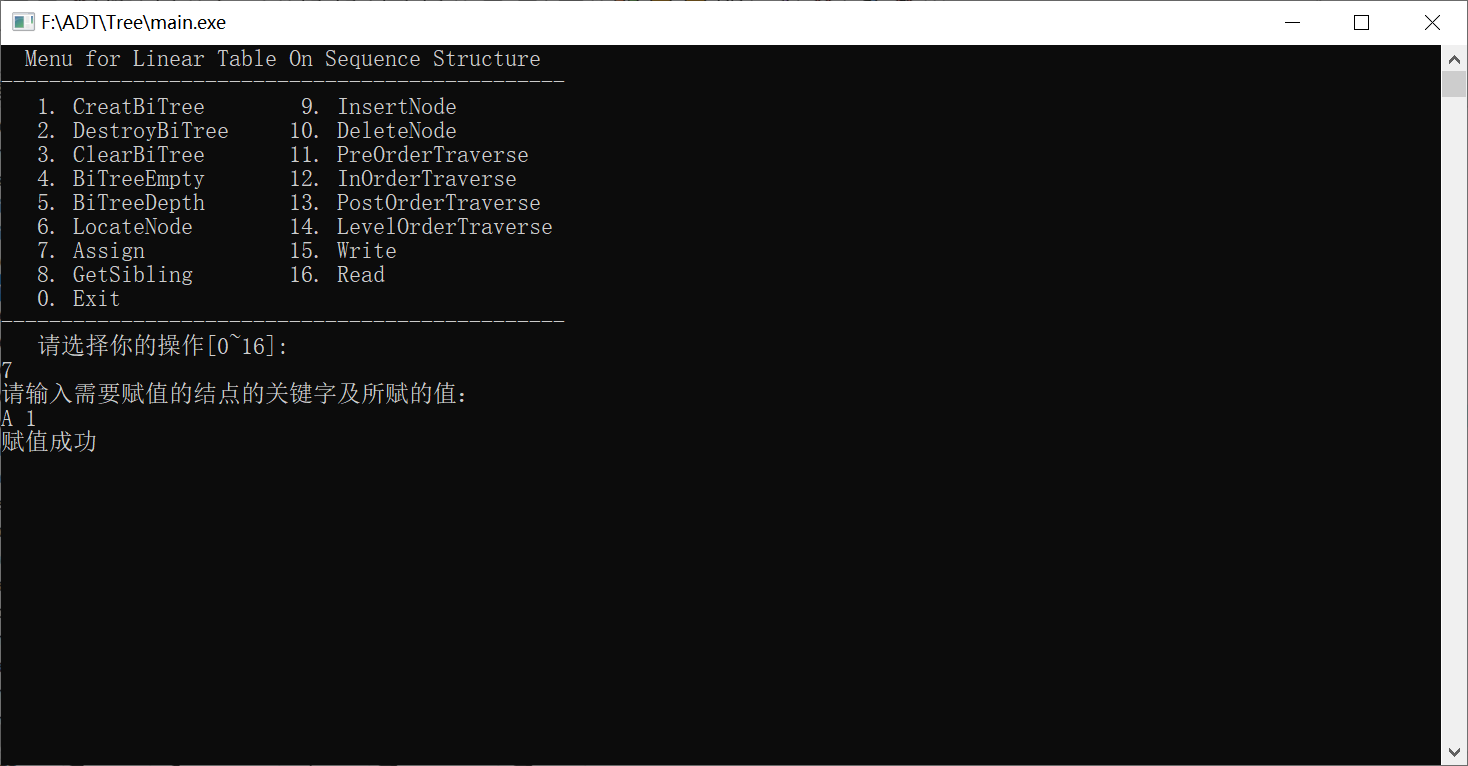


图3-5 测试-结点赋值

1. 执行功能4，判断二叉树是否为空。

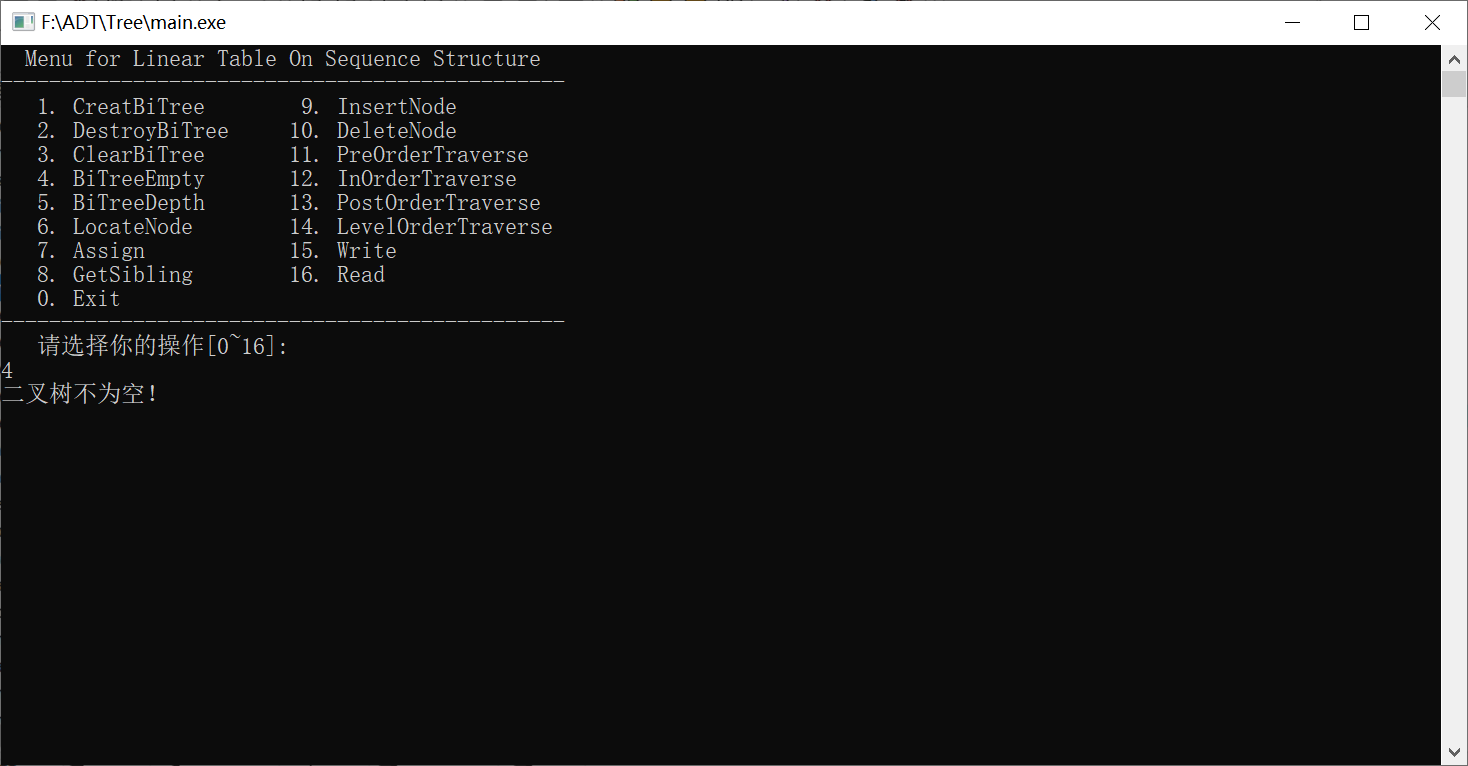


图3-6 测试-4判断二叉树非空

1. 执行功能5，求二叉树深度。

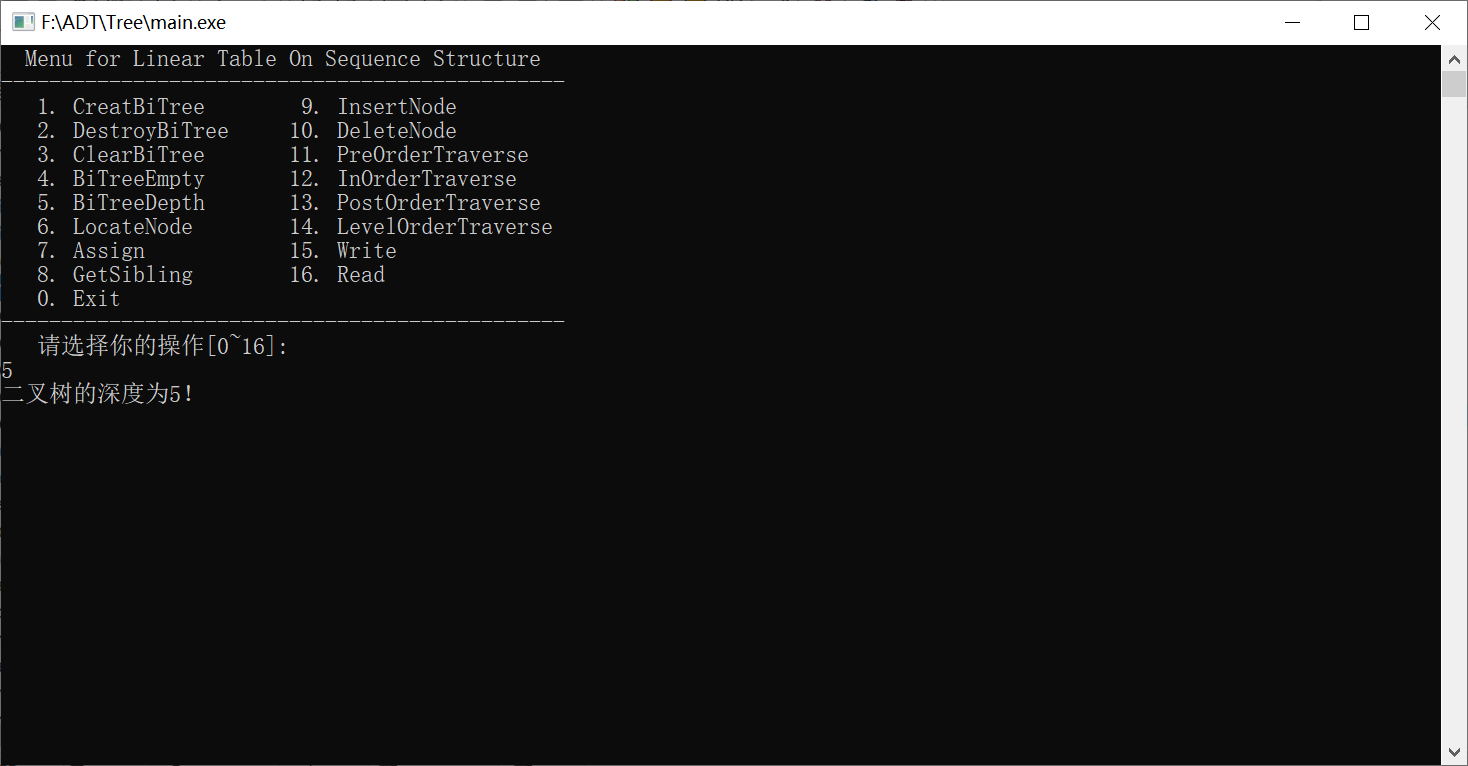


图3-7 测试-5求深度

1. 执行功能6，查找关键字D。

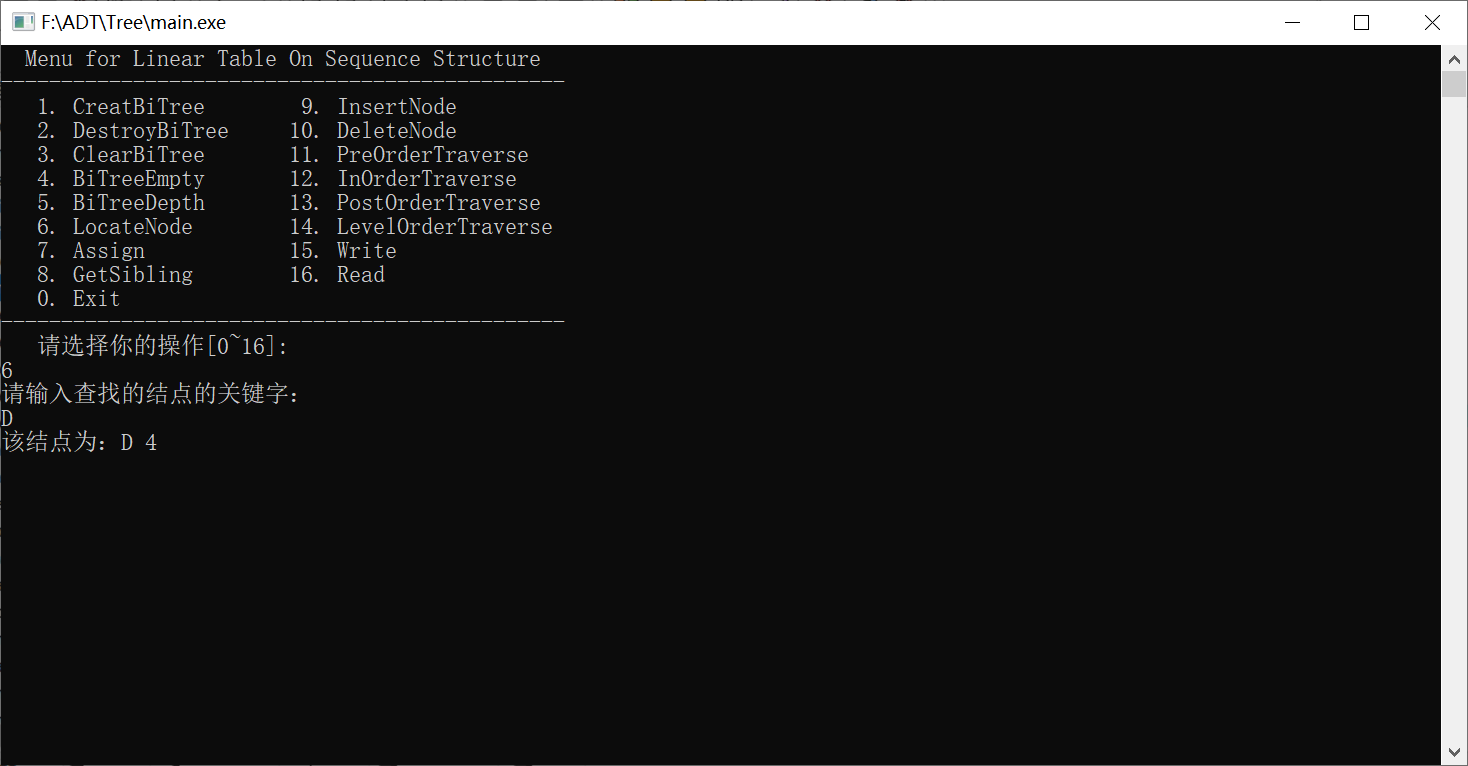


图3-8 测试-查找结点

1. 执行功能8，查找兄弟结点。

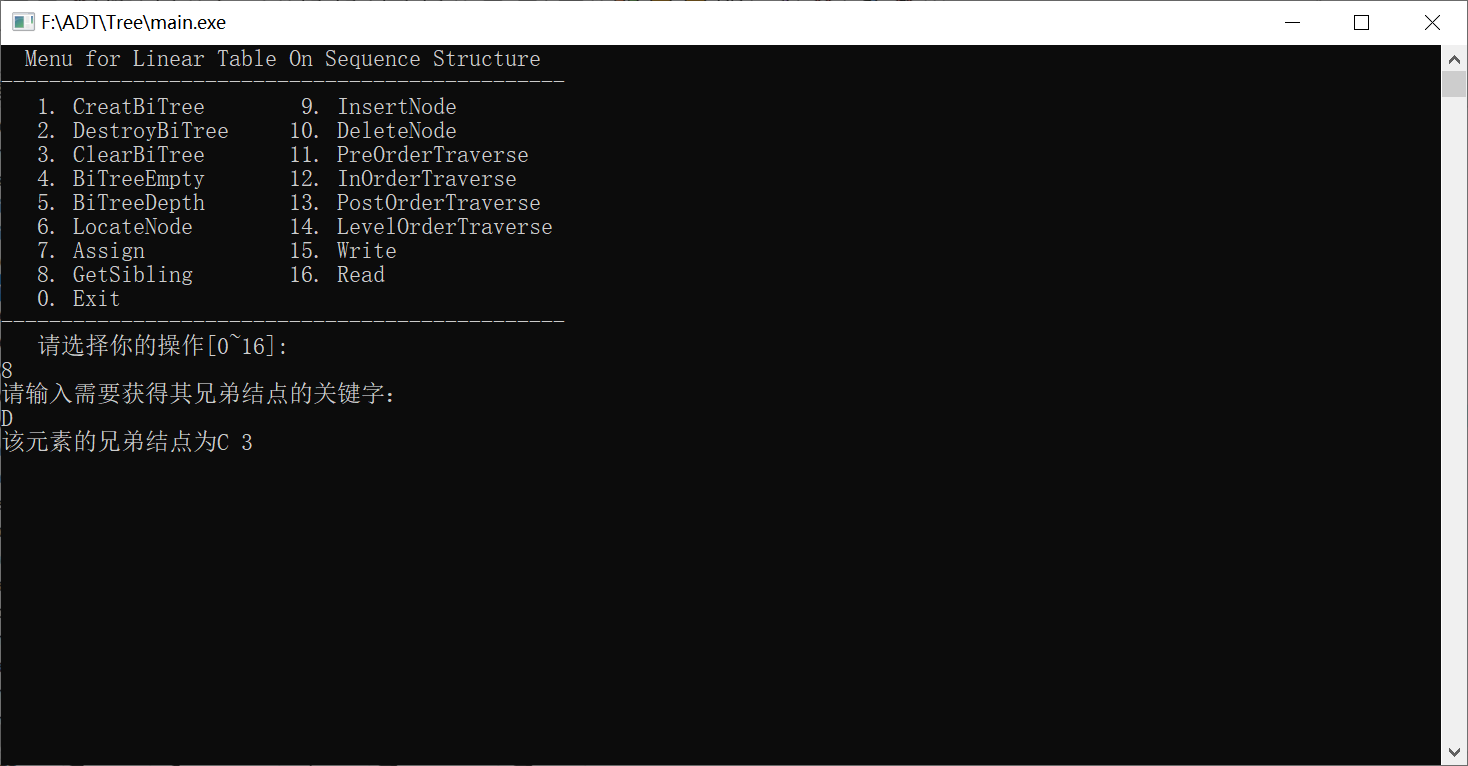


图3-9测试-查找兄弟结点

1. 执行功能9，插入结点。

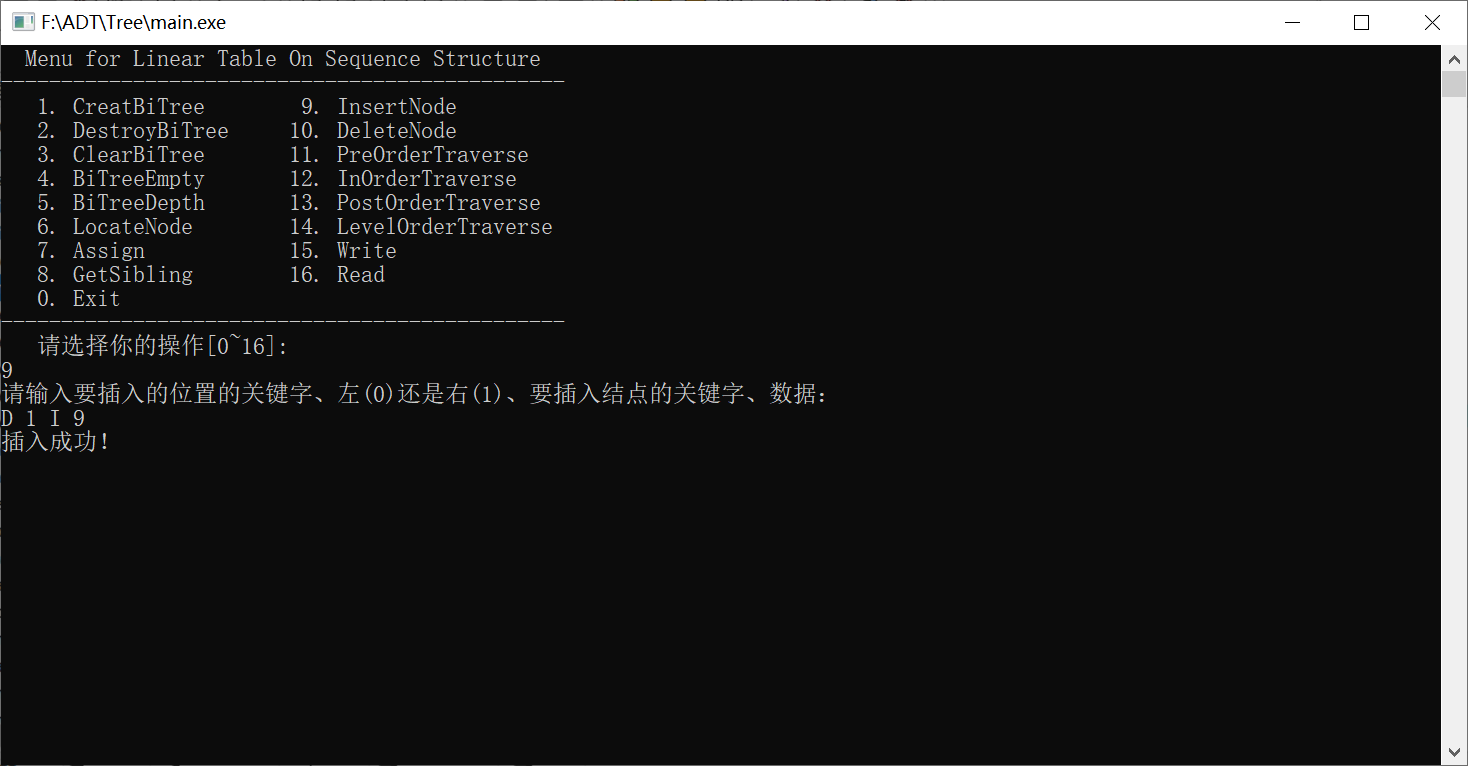


图3-10 测试-插入结点

1. 执行功能10，删除结点。

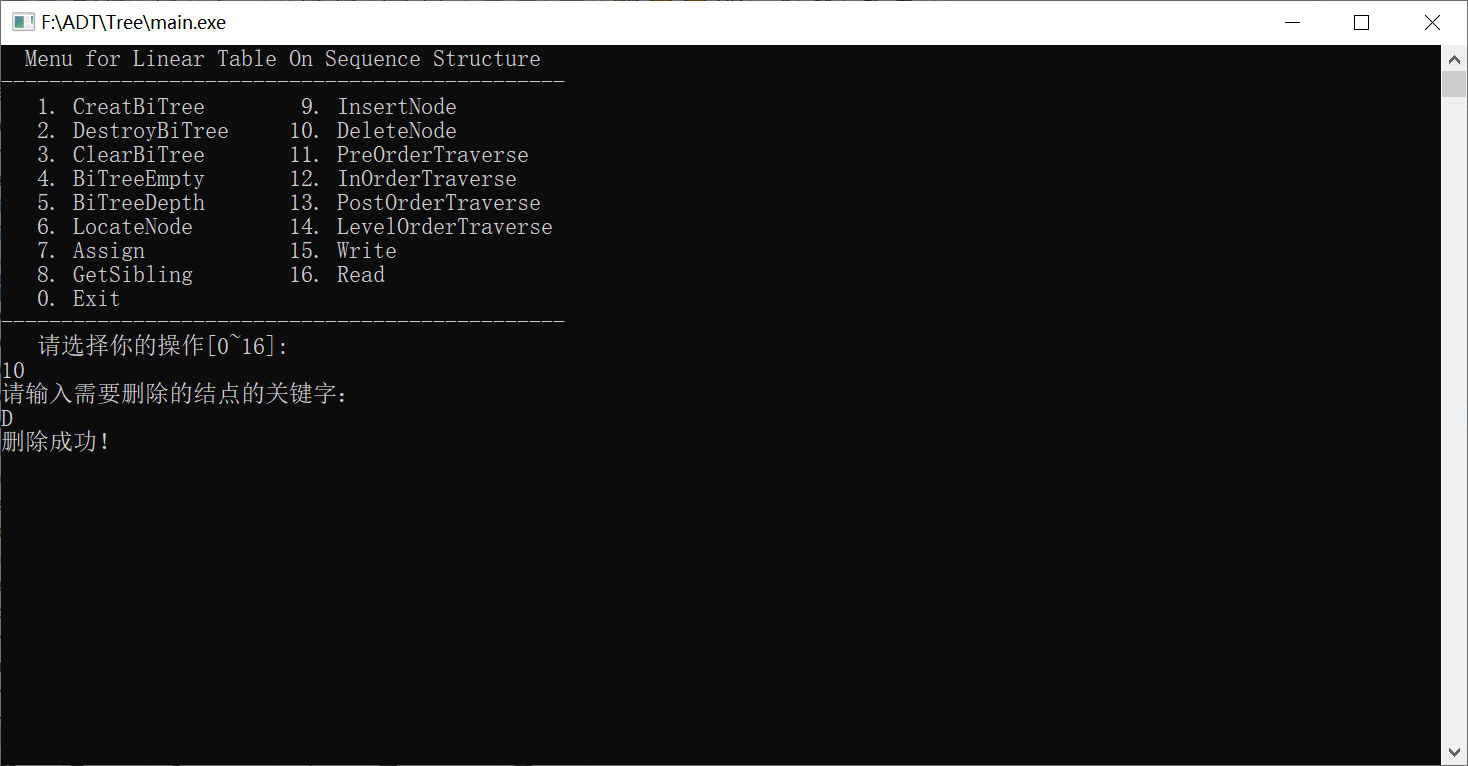


图3-11测试-删除结点

1. 执行功能11，前序遍历二叉树。

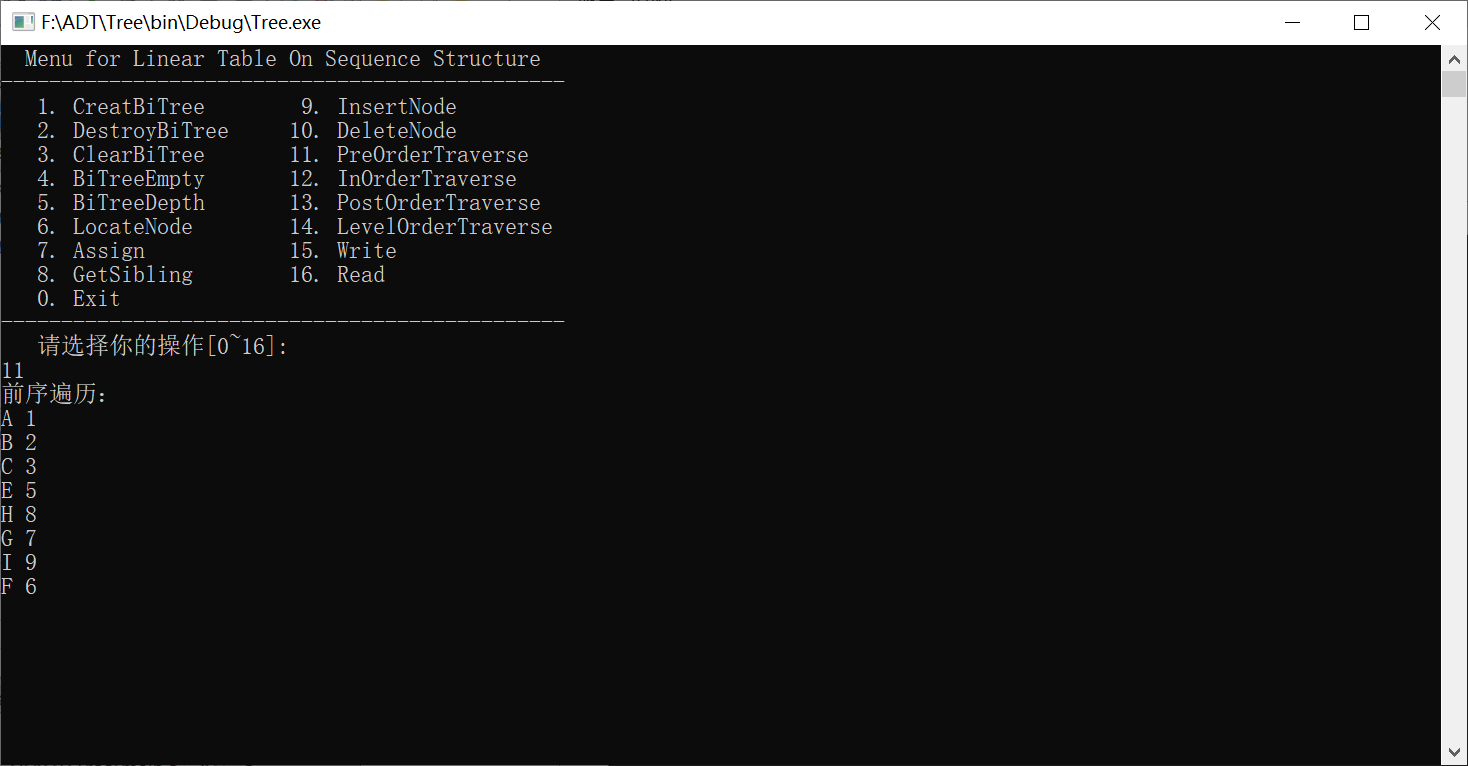


图3-12测试-前序遍历

1. 执行功能12，中序遍历二叉树。

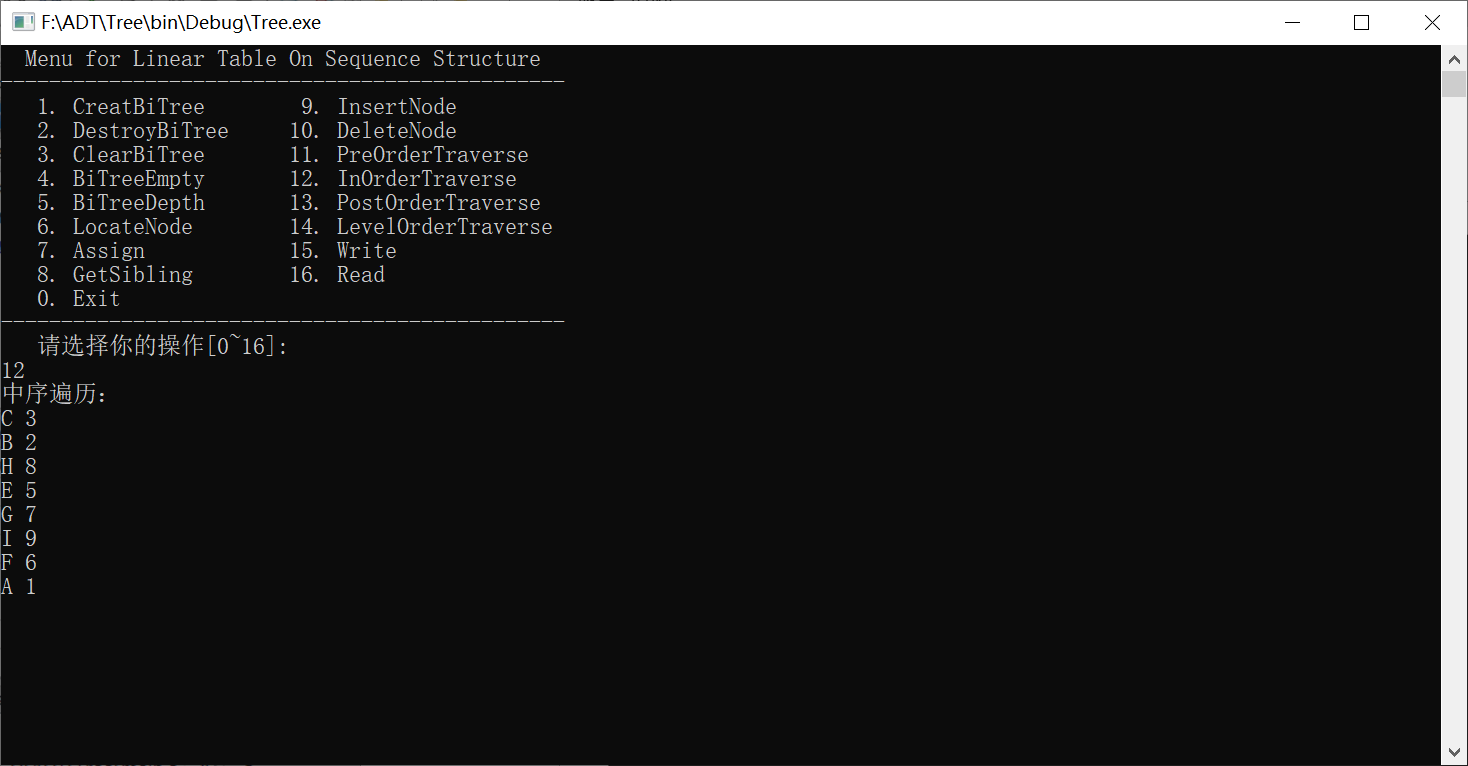


图3-13 测试-中序遍历

1. 执行功能13，后序遍历二叉树。

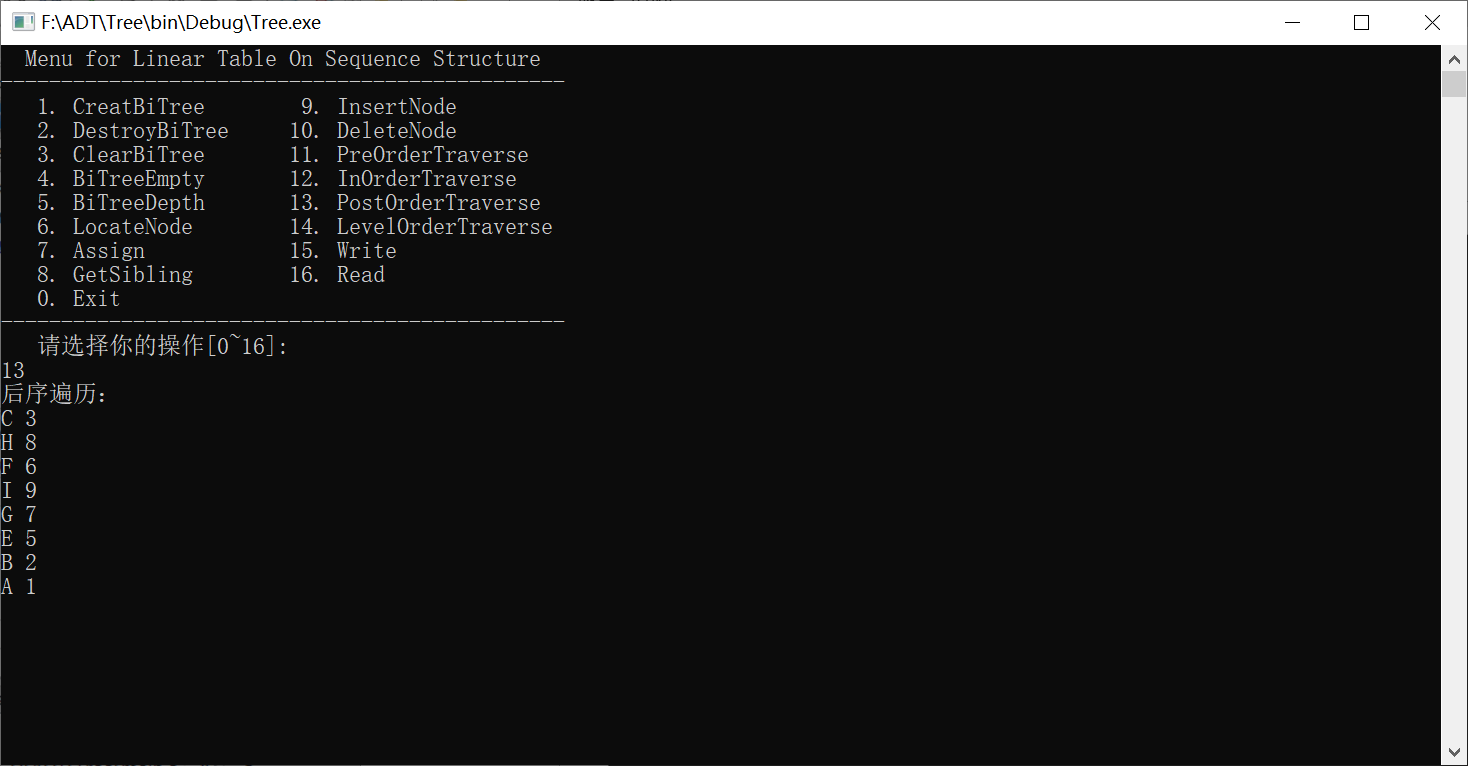


图3-14 测试-12遍历

1. 执行功能14，按层遍历二叉树。

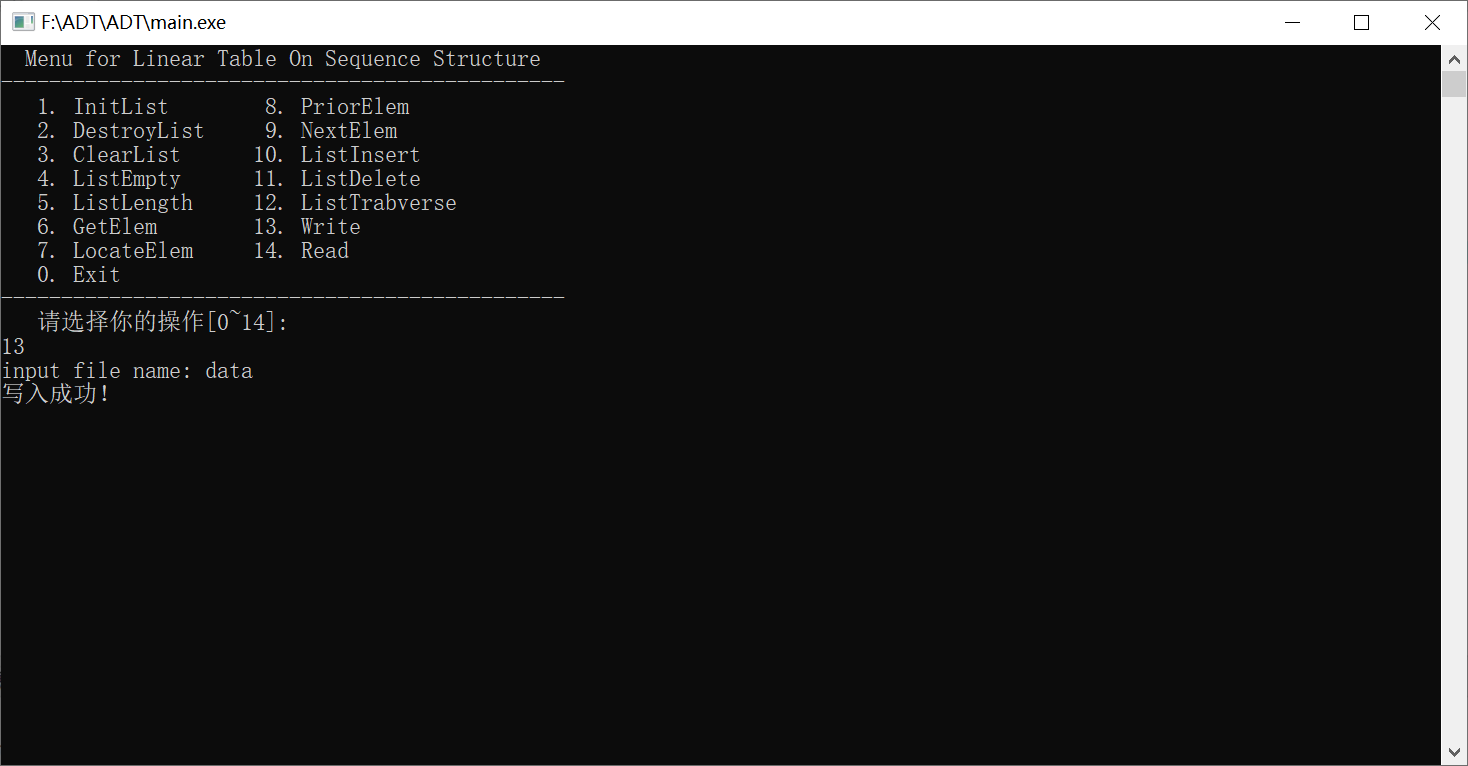


图3-15 测试-按层遍历

1. 执行功能15，写入文件。

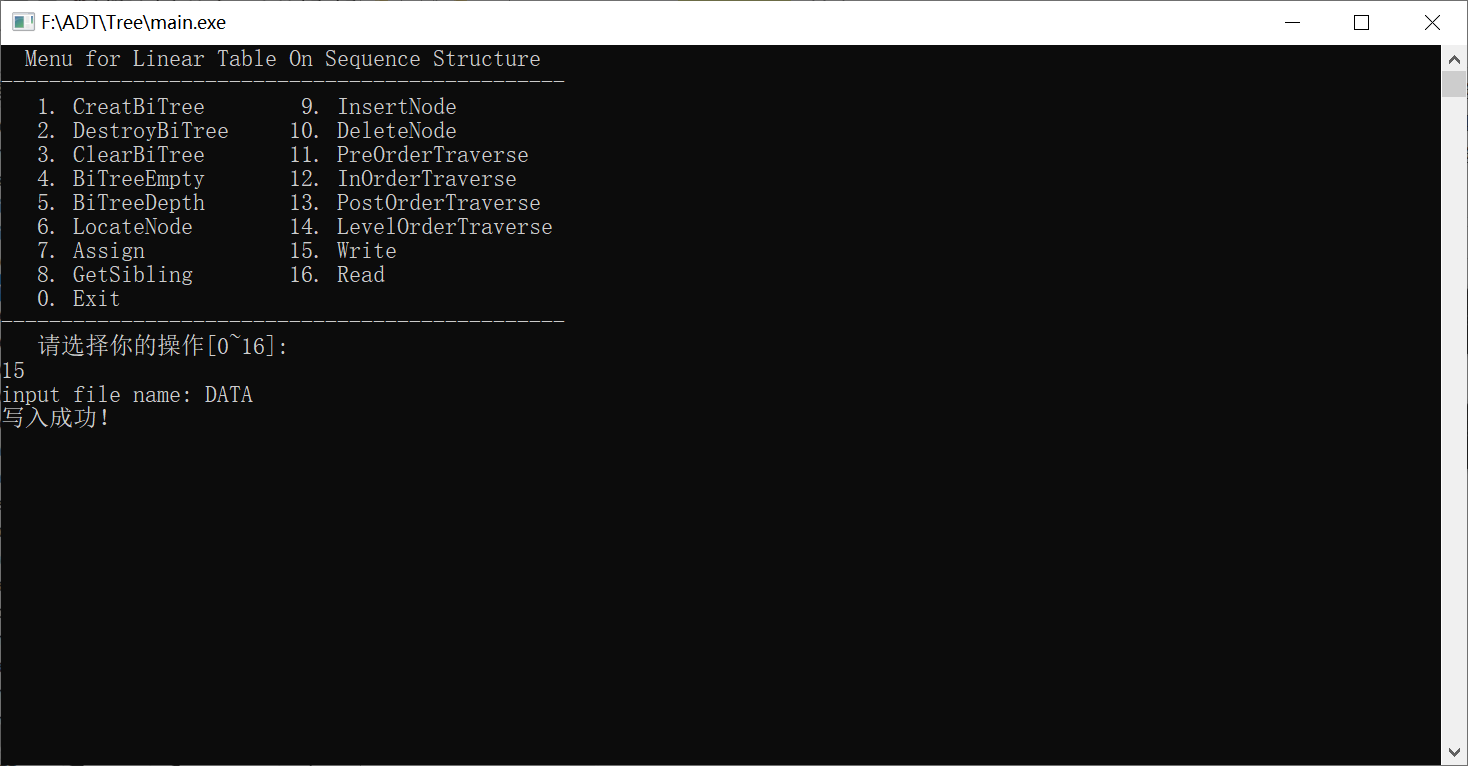


图3-16 测试-写入文件

1. 执行功能3，清空二叉树。

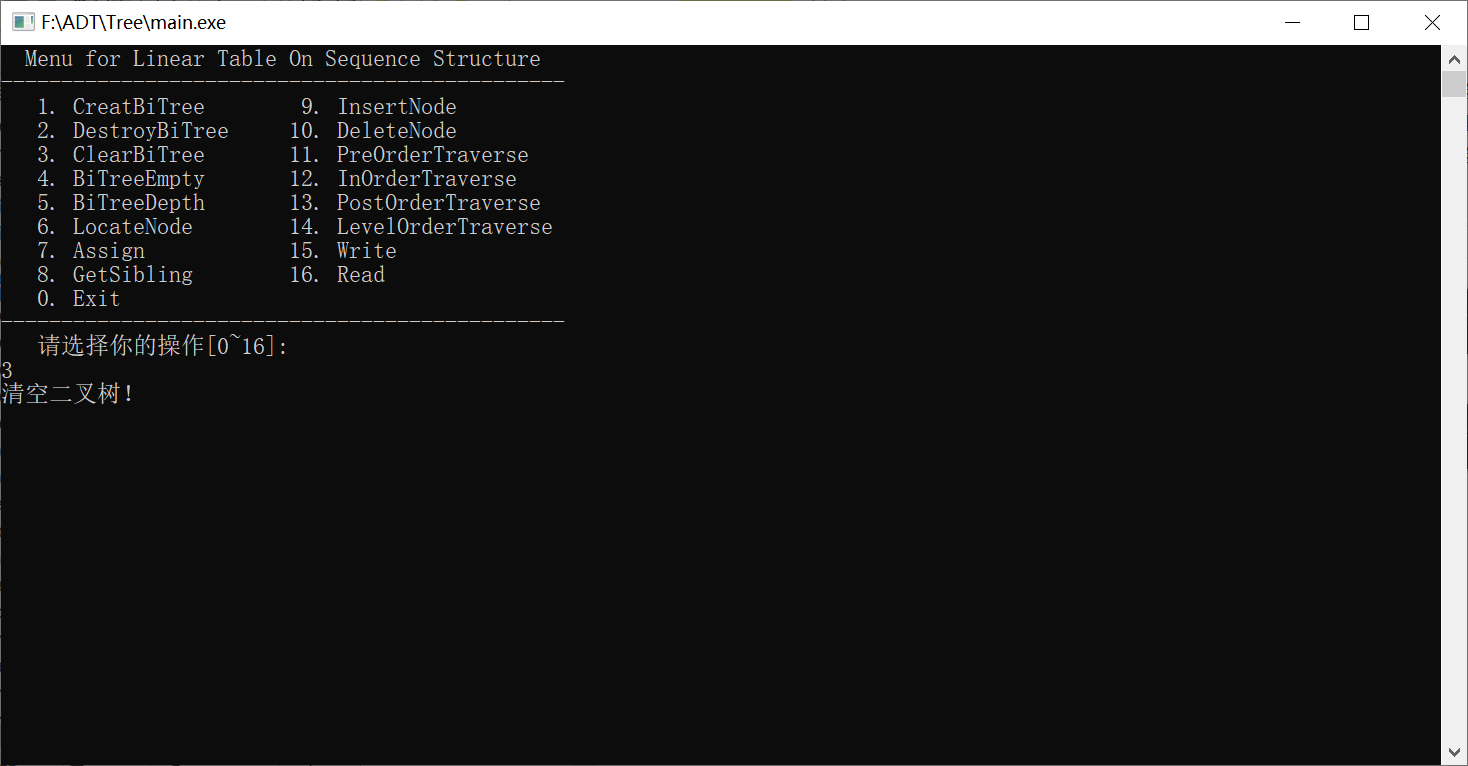


图3-17 测试-清空二叉树

1. 执行功能2，销毁二叉树2。

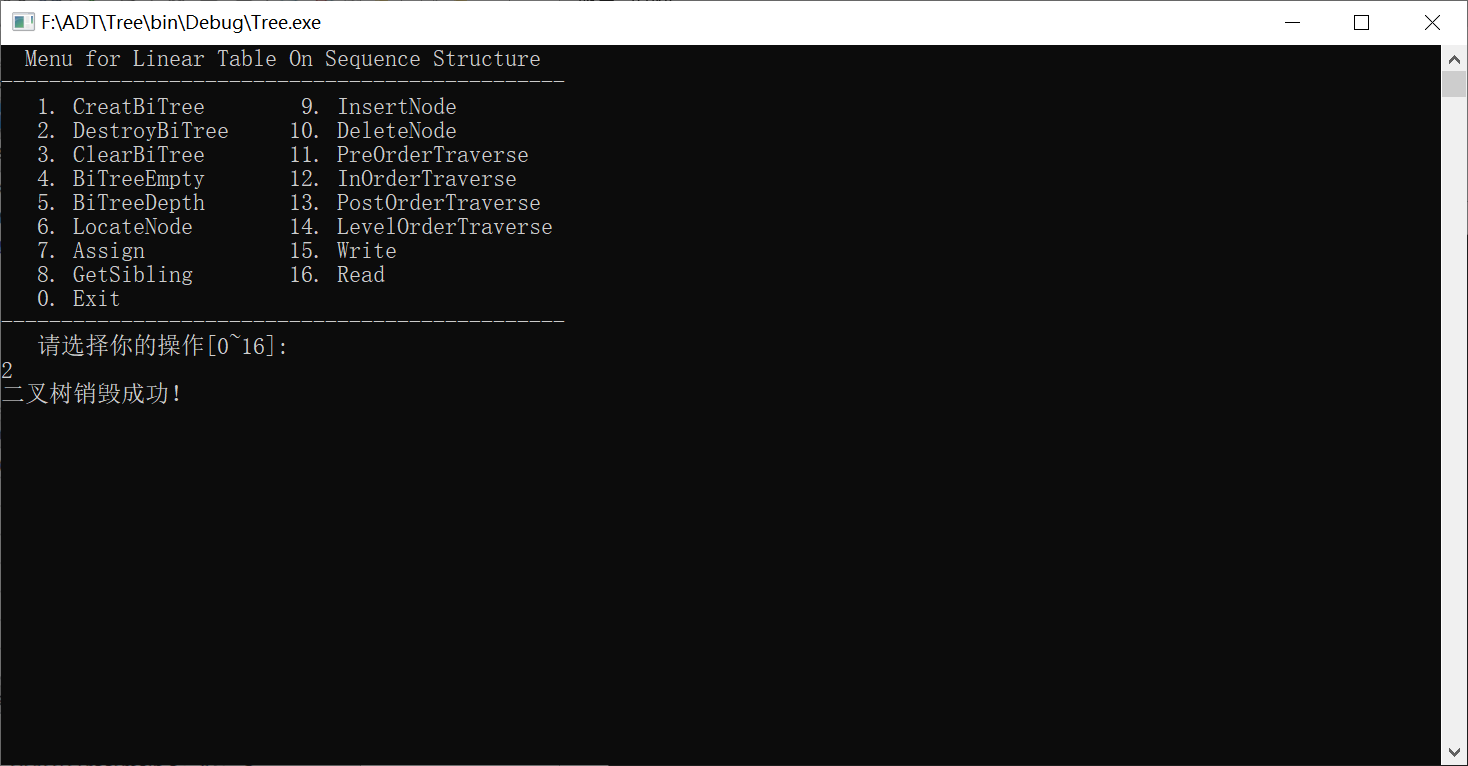


图3-18 测试-2销毁二叉树

1. 执行功能0，退出对二叉树1的操作, 重新选择二叉树2。

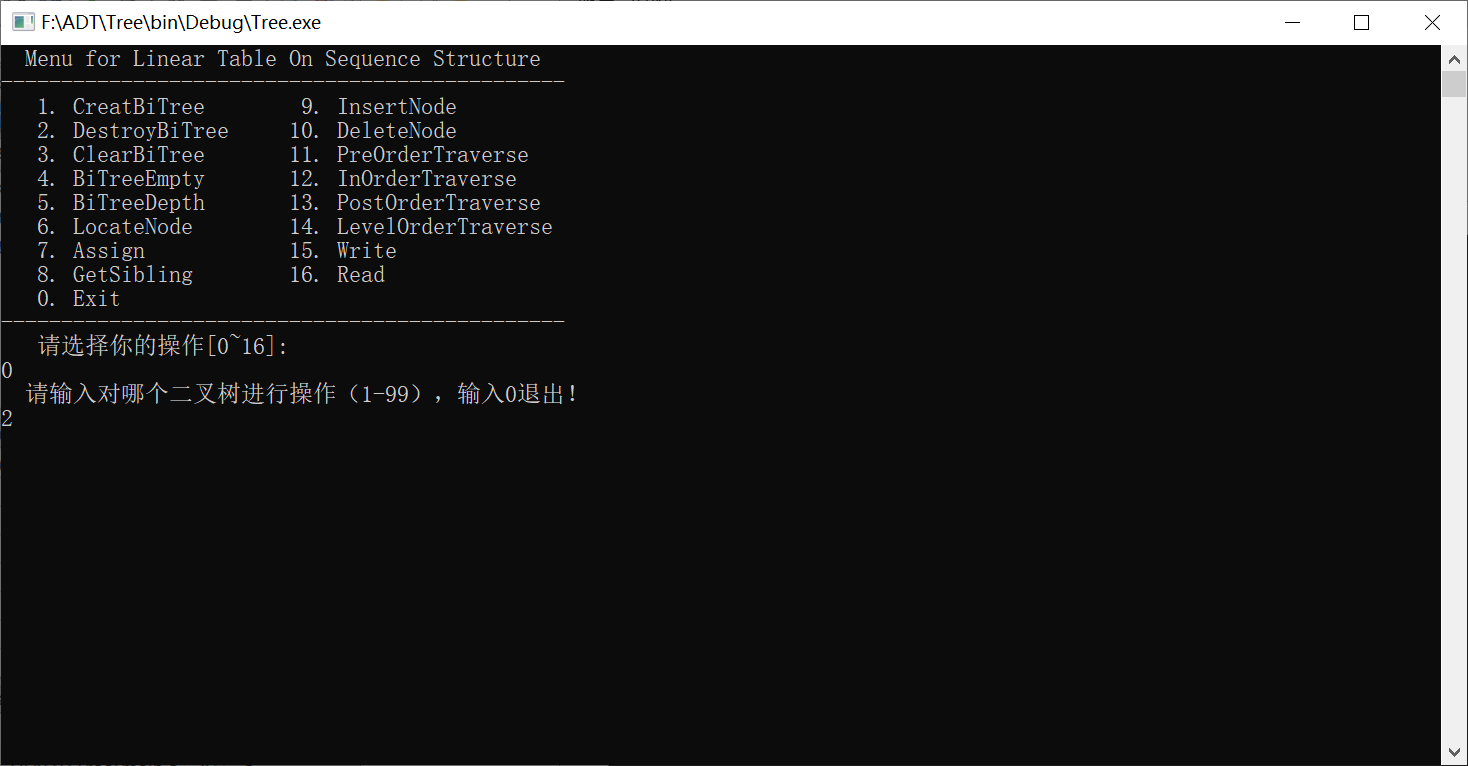
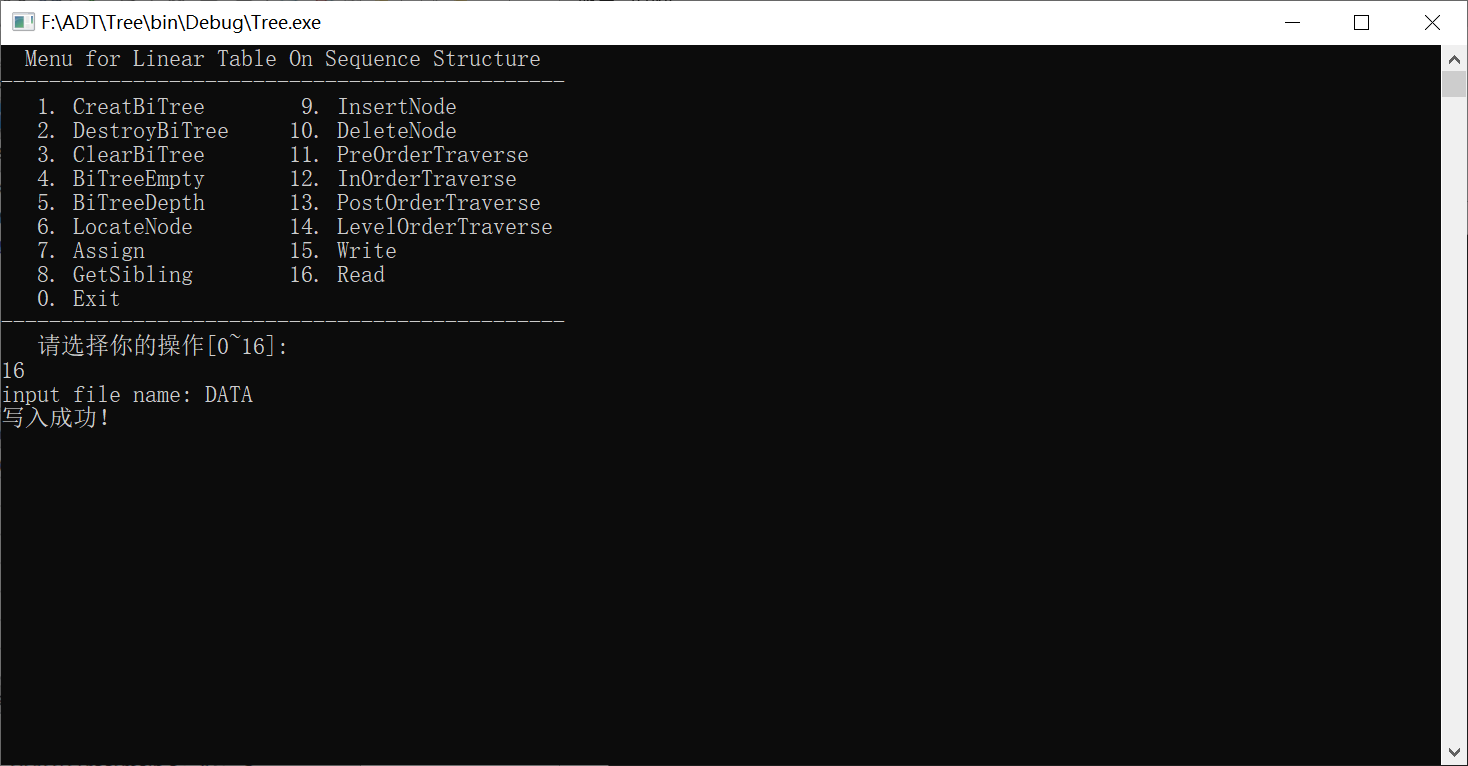
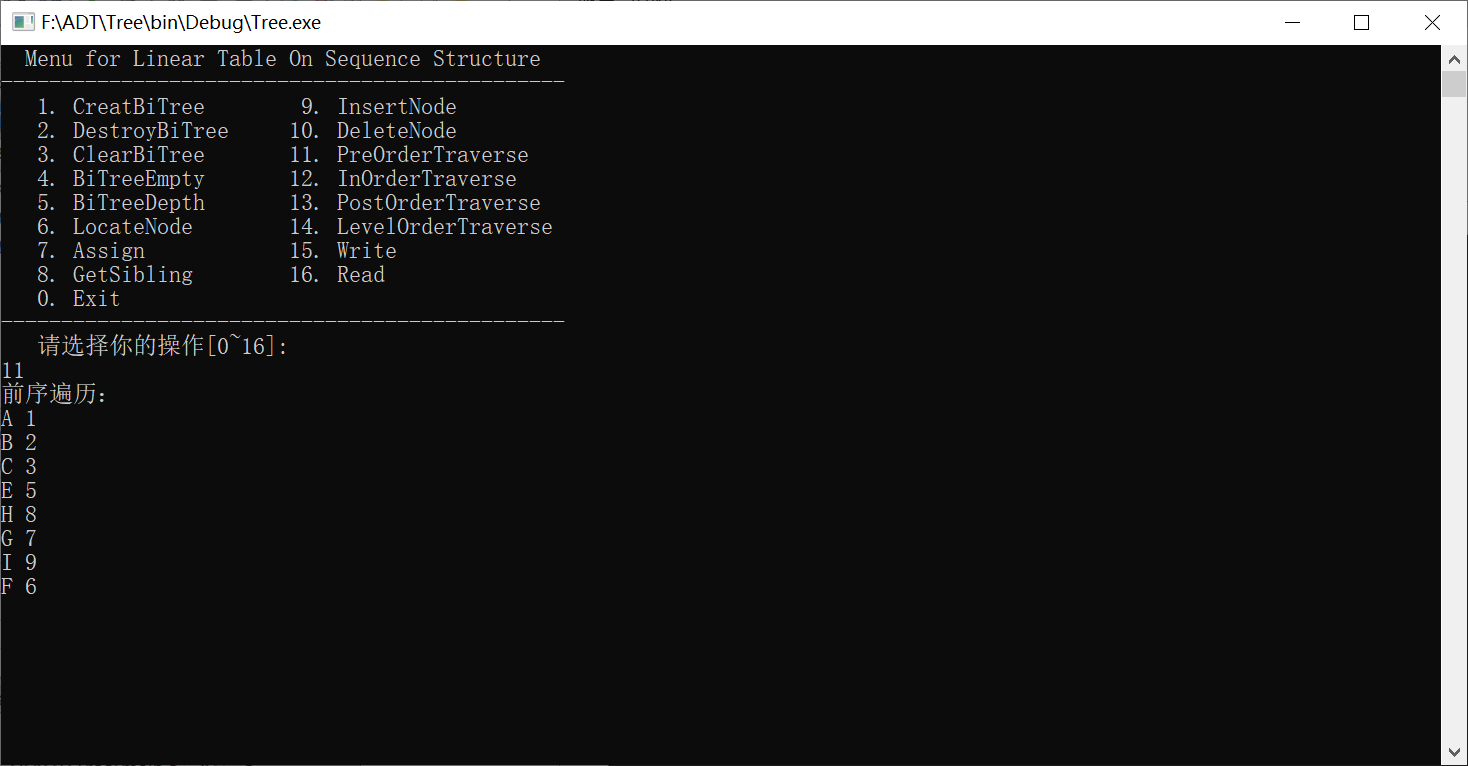


图3-19 测试-选择二叉树2

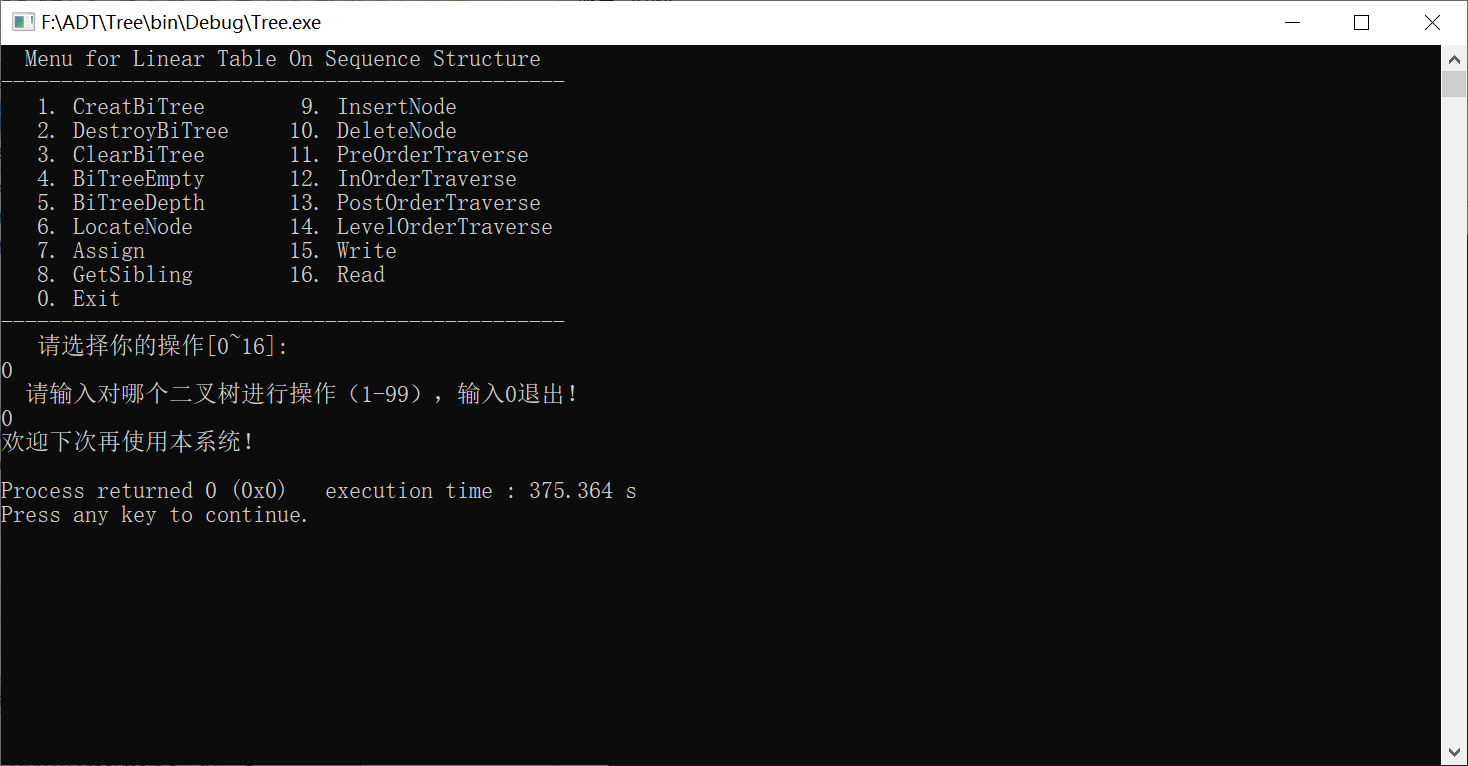
1. 执行功能16，读取文件。

图3-20 测试-读取文件

1. 执行功能11，前序遍历二叉树2。

图3-21 测试-前序遍历二叉树2

1. 执行功能0，退出对二叉树2的操作，退出系统。

图3-22 测试-0退出操作

## 3.5 实验小结

1. 当出现sizeof,和&操作符时，数组名不再当成指向一个元素的常量指针来使用，而指针仍当成指向一个元素的变量指针来使用。刚开始时，TreeRead函数传入了一个双重指针，但是由于tree是一个数组，此处&数组名与一级指针意义并不一样，所以不能直接对数组名取址当成双重指针使用。而创建函数是因为main函数里声明的definition为一个指针，可以对其取址传入双重指针。
2. 当函数有递归调用时，要在函数的开头判断指针是否为空。不然进行调用时会出现对空指针、空数据域的引用，导致程序中断。
3. 在创建函数时，一开始出现了必须要有结束符号的情况，后来发现在空结点的情况后没有加返回值，导致如果没有结束符就会一直递归调用，无法结束。非常感想李老师帮助我找到问题的所在。
4. 初步写程序时并没有加头结点，后来在写删除函数时发现没有头结点则首结点的情况需要单独列出，故又返回将二叉树加了头结点。通过此次发现头结点的用处很大。

# 4 基于邻接表的图实现

## 4.1 问题描述

采用链表与顺序表的物理结构，构造一个具有菜单的功能演示系统，并且能对多个图进行选择。在主函数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。

依据最小完备性和常用性相结合的原则，以函数形式定义了图的创建、销毁、查找顶点、顶点赋值、插入删除顶点和弧、深度遍历、广度遍历等12种基本运算。并给出输入的操作提示，以文件形式进行存储和加载，将生成的图存入到相应的文件，也可以从文件中获取图的位序、关键字和值进行操作。

4.1.1 基本运算

注明：本次实验中图的位序从0开始，不随删除插入而改变(与关键字类似)，关键字为字符型。

1. 创建图：函数名称是CreateCraph(G,V,VR)；V是图的顶点集，VR是图的关系集；两者都是字符型数组，故需将VR的字符转换成位序，再进行创建邻接表操作；操作结果是按V和VR的定义构造图G。

创建邻接表时，头插法和尾插法都进行了尝试。且因为没有头结点，第一邻接点的需要另外判断插入。

1. 销毁图：函数名称是DestroyGraph(G)；初始条件图G已存在；对于所有的顶点及顶点的邻接表依次进行销毁；操作结果是销毁图G。
2. 查找顶点：函数名称是LocateVex(G,u)；初始条件是图G存在，u是和G中顶点关键字类型相同的给定值；操作结果是若u在图G中存在，返回关键字为u的顶点位序，否则返回其它信息。
3. 顶点赋值：函数名称是PutVex (G,u,value)；初始条件是图G存在，u是和G中顶点关键字类型相同的给定值；操作结果是对关键字为u的顶点赋值value，失败返回ERROR。
4. 获得第一邻接点：函数名称是FirstAdjVex(G, u)；初始条件是图G存在，u是G中顶点的关键字；操作结果是返回u对应顶点的第一个邻接顶点的位序，如果u的顶点没有邻接顶点，否则返回无邻接点。
5. 获得下一邻接点：函数名称是NextAdjVex(G, v, w)；初始条件是图G存在，v是G中一个顶点的关键字，w对应v的邻接顶点的位序；操作结果是返回v的（相对于w）下一个邻接顶点的位置信息，如果w是最后一个邻接顶点，返回其为最后一个邻接顶点。
6. 插入顶点：函数名称是InsertVex(G,vex)；初始条件是图G存在，vex为G中顶点的结构；操作结果是在图G中增加新顶点v。（在这里也保持顶点关键字的唯一性）
7. 删除顶点：函数名称是DeleteVex(G,u)；初始条件是图G存在，u是G的一个顶点的关键字；操作结果是在图G中删除顶点u和与u相关的弧。
8. 插入弧：函数名称是InsertArc(G,u,z)；初始条件是图G存在，u、z是G的顶点的关键字；操作结果是在图G中增加弧<u,z>。
9. 删除弧：函数名称是DeleteArc(G, u,z)；初始条件是图G存在u、z是G的顶点的关键字；操作结果是在图G中删除弧<u,z>。
10. 深度优先搜索遍历：函数名称是DFSTraverse(G)；初始条件是图G存在；操作结果是图G进行深度优先搜索遍历，用数组visited[]进行标记；依次对图中的每一个顶点访问一次，且仅访问一次。
11. 广深度优先搜索遍历：函数名称是BFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行广度优先搜索遍历，，用数组visited[]进行标记；依次对图中的每一个顶点访问一次，且仅访问一次。

4.1.2 演示系统

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。图读取文件函数时与二叉树类似，不需要创建图，可直接读取。

演示系统可选择实现图的文件形式保存。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ercs文件保存和加载操作合理模式。

演示系统可选择实现多个图管理。

## 4.2 系统设计

4.2.1 头文件、常量及结构说明

1. 头文件

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

1. 预定义常量

#define TRUE 1

#define OK 1

#define ERROR -1

#define OVERFLOW -2

#define FALSE 0

#define max 20

#define MAX 100

1. 类型表达式

typedef int status; //数据元素类型定义

typedef int ElemType;

int visited[100];

1. 定义数据的物理结构如下：

typedef struct ArcNode{

int adjvex;//该弧指向顶点的位序

struct ArcNode \*nextarc;//指向下一条弧

}ArcNode;//弧

typedef struct VNode{

int num;

int value;//值

char key;//顶点信息

ArcNode \*firstarc;//指向第一条依附该顶点的弧的指针

}VNode,AdjList[MAX];

typedef struct{

AdjList vertices;

int vexnum,arcnum;//图的当前顶点数和弧数

}Graph;

typedef struct QNode{//队列结点结构

ElemType data;

struct QNode \*next;

}QNode, \*QueuePtr;

typedef struct{//队列结构

QueuePtr front;

QueuePtr rear;

}LinkQueue;

同时因为需要实现多图的操作，构造一个结构数组：Graph M[100];

4.2.2 算法设计

1. 创建图：将顶点集V[]、弧的集合VR[]传入该函数，先进行顶点构造；将弧的顶点关键字转换成位序；声明邻接表结点指针类型p、q，该函数构造邻接表时尝试了头插法和尾插法，运行的为头插法。
2. 销毁图：传入图指针的地址，即用双重指针修改图。该函数调用DestoryList进行弧的销毁，再进行顶点的销毁；在DestoryList函数中传入邻接表结点指针类型，对于邻接表遍历释放空间。
3. 查找结点：对于顶点进行遍历查找，若关键字一样，则返回位序，若没有该顶点，则返回ERROR。
4. 结点赋值：先调用 LocateVex(G,u)函数查找关键字为u的结点；如果返回值不为ERROR，则对该结点赋值，否则返回ERROR；
5. 获得第一邻接点：传入顶点关键字u；先调用 LocateVex(G,u)函数查找关键字为u的结点；如果返回值不为ERROR，再判断是否有邻接点，如果存在，则返回第一邻接点的位序，不存在则返回-2；若没有该顶点则返回ERROR。
6. 获得下一邻接点： 传入顶点关键字u，位序w；声明邻接表结点指针p；先调用 LocateVex(G,u)函数查找关键字为u的结点，如果存在该顶点则令p指向第一邻接点，再依次查找邻接点的位序是w的邻接点，如果下一邻接点存在则返回下一邻接点位序，如果下一邻接点不存在返回NULL；如果不存在顶点u则返回ERROR。
7. 插入顶点：传入顶点类型vex；在顺序表最末尾即vertices[G->vexnum]处加上被插入的顶点vex，G->vexnum++；返回OK。
8. 删除顶点： 先遍历所有顶点后的邻接表；调用删除弧的函数；声明邻接表结点指针p、q，如果结点存在，令p指向顶点的第一邻接点，依次删除所有含有该顶点的弧。再删除该结点，该结点后面的顶点依次前移；G->arcnum--，返回OK；如果该顶点不存在，返回ERROR。
9. 插入弧：传入弧头和弧尾顶点的关键字u、z；调用Locatehao(G,u) 找出它们的序号，如果存在则找到顶点u的最末尾的邻接点，然后插入该弧，G->arcnum++；返回OK。
10. 删除弧：传入弧头和弧尾顶点的关键字u、z；调用Locatehao(G,u) 找出它们的序号，如果存在则找到遍历顶点u的邻接表，找到顶点z的位序，随后删除；G->arcnum--；返回OK，否则返回ERROR。
11. 深度优先遍历：访问初始结点v，并对v进行已访问的标记（visited[v]= 1）；查找v的第一个邻接结点w。若w未被访问，则执行④，否则结束遍历。④若w未被访问，则对w进行深度优先遍历（递归），查找w的下一个邻接结点，并循环执行递归，直到遍历结束。最后判断是否所有的顶点都进行了访问，如果没有，则从未进行访问的顶点再次调用该函数。
12. 广度优先遍历：需要运到队列作为辅助。访问初始结点v，并对v进行已访问的标记。将结点v送入队列。若队列已空，则结束算法，队列非空则继续。出队列，取得头结点u。查询u的第一个邻接结点w。若w不存在，则转到判断队列是否为空，再依次执行操作；若w存在并w未被访问，则访问w并标记w；再让结点w入队；查找结点u在w之后的下一个邻接结点，循环进行。
13. 由关键字查找序号Locatehao：对于顶点进行遍历查找，若关键字一样，则返回i，若没有该顶点，则返回ERROR。
14. 由位序查找关键字Locatekey：对于顶点进行遍历查找，若位序一样，则返回关键字，若没有该顶点，则返回ERROR。
15. 由位序查找序号Locatekey2：对于顶点进行遍历查找，若位序一样，则返回序号，若没有该顶点，则返回ERROR。

4.2.3 理论分析

表4-1 函数算法复杂度分析

|  |  |
| --- | --- |
| 测试功能功能及序号 | 算法复杂度(右n个结点，以最坏情况看) |
| 1. CreatGraph | O(n) |
| 2. DestroyGraph | O(n) |
| 3. LocateVex | O(n) |
| 4. PutVex | O(n) |
| 5. FirstAdjVex | O(1) |
| 6. NextAdjVex | O(n) |
| 7. InsertVex | O(1) |
| 8. DeleteVex | O(n2) |
| 9. InsertArc | O(n) |
| 10. DeleteArc | O(n) |
| 11. DFSTraverse | O(n) |
| 12. BFSTraverse | O(n2) |

## 4.3 系统实现

4.3.1 系统演示

整个系统的流程及结构如下图所示。



图4-1 系统结构图

4.3.2 部分函数功能实现

1. 删除顶点：删除顶点先访问初始结点v，并对v进行已访问的标记（visited[v]= 1）；查找v的第一个邻接结点w。若w未被访问，则执行④，否则结束遍历。④若w未被访问，则对w进行深度优先遍历（递归），查找w的下一个邻接结点，并循环执行递归，直到遍历结束。最后判断是否所有的顶点都进行了访问，如果没有，则从未进行访问的顶点再次调用该函数。为了避免空指针的使用，故在判断时会有些麻烦。流程图如下图所示，因为图有些复杂，需要放大才能看清。

status DeleteVex(Graph \*G,char u){

int i,v;

ArcNode \*p;

ArcNode \*q;

v=LocateVex(\*G,u);

if(v!=ERROR){

for(i=0;i<G->vexnum;i++){

p=G->vertices[i].firstarc;

if((p!=NULL)&&(p->nextarc!=NULL)){

if(p->adjvex==v){

G->vertices[i].firstarc=p->nextarc;

}

else{

p=G->vertices[i].firstarc;

q=p->nextarc;

while(q!=NULL){

if(q->adjvex!=v){

p=p->nextarc;

q=q->nextarc;

}

else{

p->nextarc=q->nextarc;

q=NULL;

}

}

}

}

else

if((p!=NULL)&&(p->nextarc=NULL))

if(p->adjvex==v)

p=NULL;

}

DestoryList(G->vertices[v].firstarc);

for(i=v;i<G->vexnum;i++){//顶点v后面的顶点前移

G->vertices[i]=G->vertices[i+1];

}

G->vexnum--;

return OK;

}

else

return ERROR;

}

图4-2 DeleteVex函数流程图

1. 广度优先遍历： 需要运到队列作为辅助。访问初始结点v，并对v进行已访问的标记。将结点v送入队列。若队列已空，则结束算法，队列非空则继续。出队列，取得头结点u。查询u的第一个邻接结点w。若w不存在，则转到判断队列是否为空，再依次执行操作；若w存在并w未被访问，则访问w并标记w；再让结点w入队；查找结点u在w之后的下一个邻接结点，循环进行。

此处主要是关于队列的应用及其一系列操作。

status BFSTraverse(Graph G, LinkQueue Q){

int v,u,i,num;

char k;

ArcNode \*temp;

for(v=0;v<G.vexnum;v++)

visited[v]=0;

for(v=0;v<G.vexnum;v++){

if(!visited[v]){

visited[v]=1;

printf("%c %d\n", G.vertices[v].key,G.vertices[v].value);

num=LocateVex(G,G.vertices[v].key);//位序

EnQueue(&Q,num);

while(!QueueEmpty(Q)){

DeQueue(&Q,&u);

i=Locatekey2(G,u);//由位序找序号

for(temp=G.vertices[i].firstarc;temp!=NULL;temp=temp->nextarc){

k=Locatekey(G,temp->adjvex);

num=LocateVex(G,k);

i=Locatehao(G,k);

if (visited[i]!=1) {

visited[i]=1;

printf("%c %d\n", G.vertices[i].key,G.vertices[i].value);

EnQueue(&Q,num);

}

}

}

}

}

return OK;

}

status InitQueue(LinkQueue \*Q){//构造一空的队列Q

Q->front=Q->rear=(QueuePtr)malloc(sizeof(QNode));

if (!Q->front)

exit(OVERFLOW);

Q->front->next=NULL;

return OK;

}

status QueueEmpty(LinkQueue Q){//若Q为空,返回TRUE,不空则返回FALSE

if(Q.front==Q.rear)

return TRUE;

else

return FALSE;

}

status EnQueue(LinkQueue \*Q,ElemType e){//插入元素e为Q的新的队尾元素

QueuePtr p=(QueuePtr)malloc(sizeof(QNode));

if (!p)//存储分配失败

exit(OVERFLOW);

p->data=e;

p->next=NULL;

Q->rear->next=p;

Q->rear=p;

return OK;

}

status DeQueue(LinkQueue \*Q,ElemType \*e){//若队列不空,删除Q的队头元素,用e返回其值,并返回OK,否则返回ERROR

QueuePtr p;

if (Q->front==Q->rear)

return ERROR;

p=Q->front->next;

\*e=p->data;

Q->front->next=p->next;

if(Q->rear==p)

Q->rear=Q->front;

free(p);

return OK;

}

4.3.3 用文件读写

文件的读取可用多个图进行操作，例如用图1写入文件，再创建图2读取数据，使之与图1 相同。重点是关于读取文件时图的创建以及结束的判断条件。

1. 文件的写入：在Write函数中打开文件，依次将顶点的关键字、值、位序写入文件；再从第一个顶点开始将邻接点转化成弧依次写入文件。

写入文件：

status Write(Graph G){//写入文件

FILE \*fp=NULL;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

if ((fp=fopen(filename,"wb"))==NULL){

printf("File open error\n ");

return FALSE;

}

//写文件的方法

int i;

int j;

ArcNode \*p;

fwrite(&G.vexnum,sizeof(int),1,fp);

fwrite(&G.arcnum,sizeof(int),1,fp);

for(i=0;i<G.vexnum;i++){

fwrite(&G.vertices[i].key,sizeof(char),1,fp);

fwrite(&G.vertices[i].value,sizeof(int),1,fp);

fwrite(&G.vertices[i].num,sizeof(int),1,fp);

}

for(i=0;i<G.vexnum;i++){

p=G.vertices[i].firstarc;

while(p!=NULL){

fwrite(&G.vertices[i].num,sizeof(int),1,fp);

j=Locatekey2(G,p->adjvex);

fwrite(&G.vertices[j].num,sizeof(int),1,fp);

p=p->nextarc;

}

}

fclose(fp);

return OK;

}

1. 文件的读取：通过Read函数打开文件，先将顶点的关键字、位序、值的信息读取，再从文件中将弧读取并判断构造邻接表，原理与创建类似，此处也采用尾插法。因为fread函数有自己的判断条件，所以读取时不需要其他判断条件。

status Read(Graph \*G){//读取文件

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

if ((fp=fopen(filename,"rb"))==NULL){

printf("File open error\n ");

return FALSE;

}

//读文件的方法

int i=0;

int a=-1;

int j;

ArcNode \*p;

ArcNode \*q;

fread(&G->vexnum,sizeof(int),1,fp);

fread(&G->arcnum,sizeof(int),1,fp);

for(i=0;i<G->vexnum;i++){

fread(&G->vertices[i].key,sizeof(char),1,fp);

fread(&G->vertices[i].value,sizeof(int),1,fp);

fread(&G->vertices[i].num,sizeof(int),1,fp);

}

while(fread(&a,sizeof(int),1,fp)){

j=Locatekey2(\*G,a);

q=G->vertices[j].firstarc;

if(q==NULL){

p=(ArcNode\*)malloc(sizeof(ArcNode));

fread(&p->adjvex,sizeof(int),1,fp);

p->nextarc=NULL;

G->vertices[j].firstarc=p;

}

else{

while((q->nextarc)!=NULL){

q=q->nextarc;

}

p=(ArcNode\*)malloc(sizeof(ArcNode));

fread(&p->adjvex,sizeof(int),1,fp);

p->nextarc=NULL;

q->nextarc=p;

}

}

fclose(fp);

return OK;

}

## 4.4 系统测试

4.4.1 测试计划

表4-2 测试计划

|  |  |  |  |
| --- | --- | --- | --- |
| 测试功能功能及序号 | 测试的图 | 输入的参数 | 结果 |
| 选择图 | 1 | 1 | 对图1进行操作 |
| 1. CreatGraph | 1 | 8 8;  12345678;  12 13 24 25 36 37 48 85; | 创建图1 |
| 3. LocateVex | 1 | 8 | 查找到顶点8 |
| 4. PutVex | 1 | 5 10； | 顶点5的值为10 |
| 5. FirstAdjVex | 1 | 1 | 顶点1的第一邻接点的位序为1 |
| 6. NextAdjVex | 1 | 顶点关键字1；另一顶点位序1； | 下一邻接点的位序为2 |
| 7. InsertVex | 1 | 9 0； | 插入顶点9，值为0 |
| 9. InsertArc  (插入两次) | 1 | 69；  97； | 插入两个弧 |
| 10. DeleteArc | 1 | 85； | 删除弧85 |
| 8. DeleteVex | 1 | 需要删除的结点的关键字：6 | 删除结点6 |
| 11. DFSTraverse | 1 | 1 | 深度优先遍历 |
| 12. BFSTraverse | 1 | 无 | 广度优先遍历 |
| 8. DeleteVex | 1 | 1 | 删除顶点1 |
| 11. DFSTraverse | 1 | 2 | 从顶点2开始深度优先遍历 |
| 12. BFSTraverse | 1 | 无 | 广度优先遍历 |
| 13. Write | 1 | 文件名：DATA | 将图1写入文件 |
| 2. DestroyBiTree | 1 | 无 | 销毁图1 |
| 0. Exit | 1 | 无 | 退出操作 |
| 选择图 | 2 | 2 | 对图2进行操作 |
| 14. Read | 2 | 文件名：DATA | 从该文件读入数据 |
| 12. BFSTraverse | 2 | 无 | 遍历图2 |
| 退出选择图 | 无 | 退出输入0 | 退出选择 |

4.4.2 测试结果

执行3.4.1表格的测试计划，运行结果截图如下所示。

1. 选择图1，进入操作。

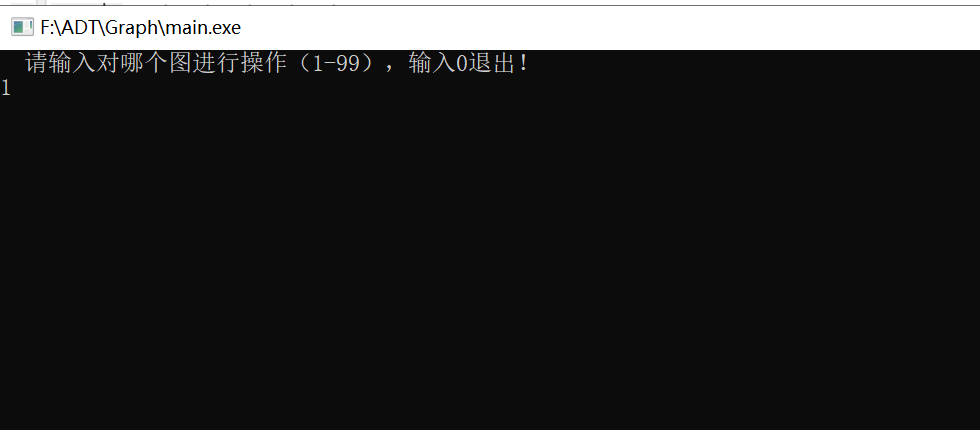


图4-3 测试-选择图

1. 执行功能1，创建图1。

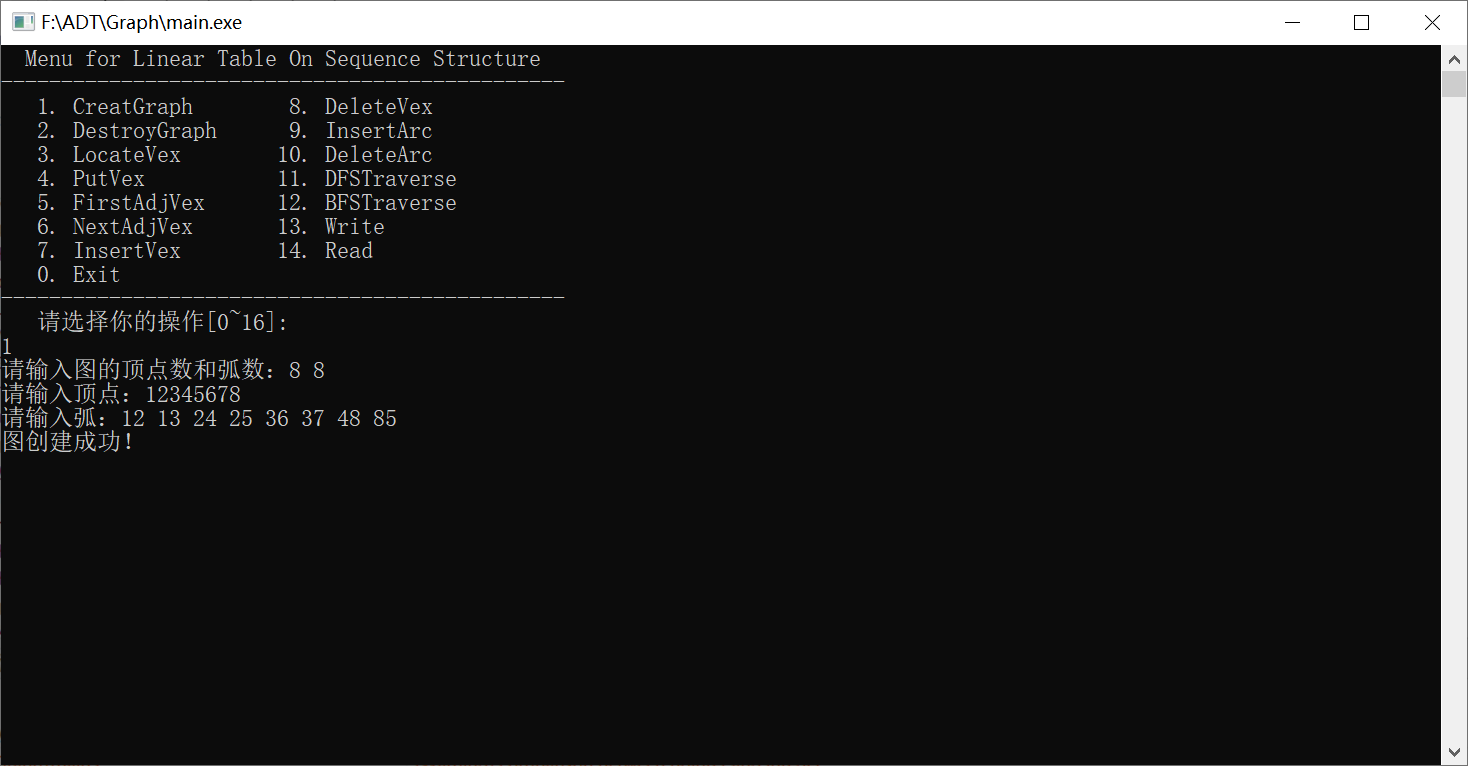


图4-4 测试-创建图

1. 执行功能3，查找结点8。

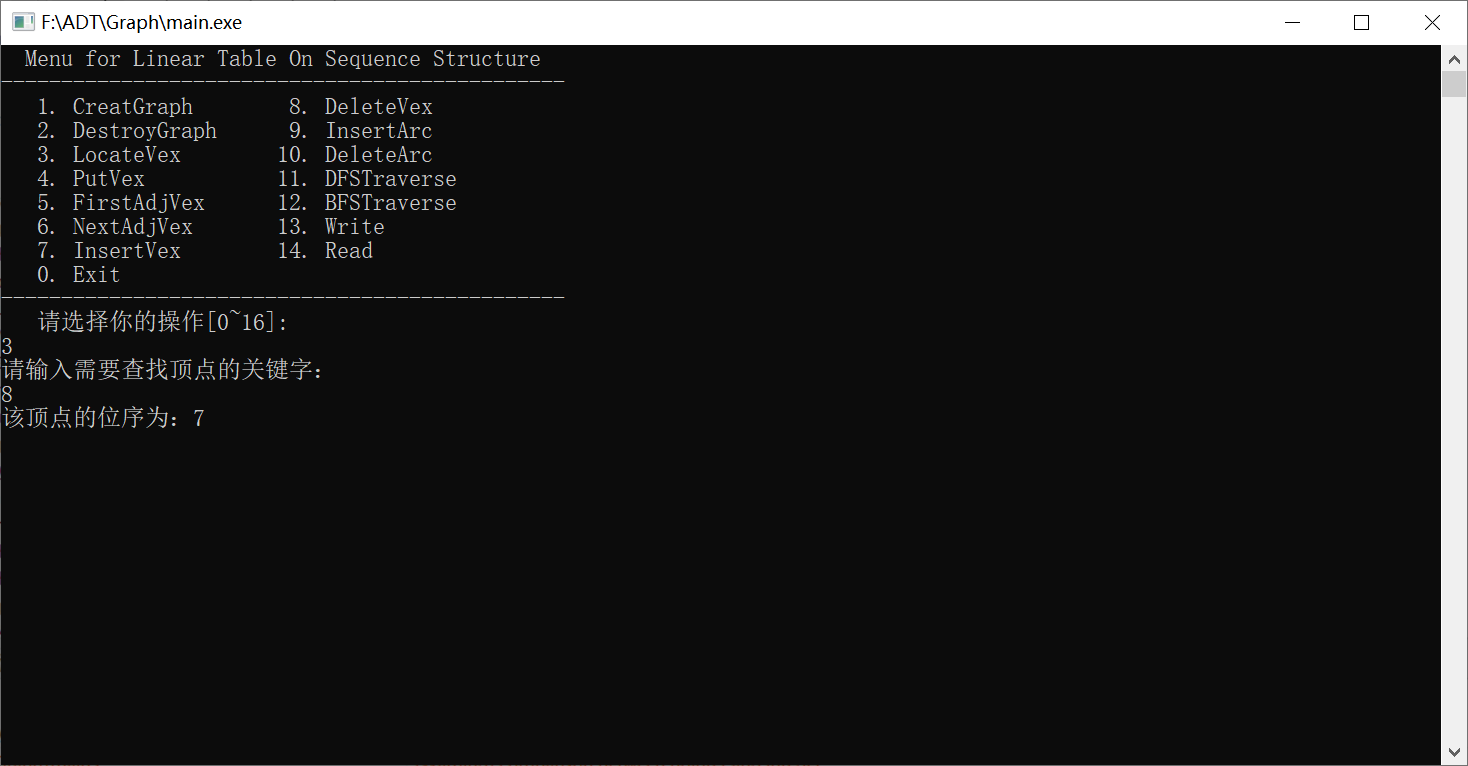


图4-5 测试-查找顶点

1. 执行功能4，顶点赋值。

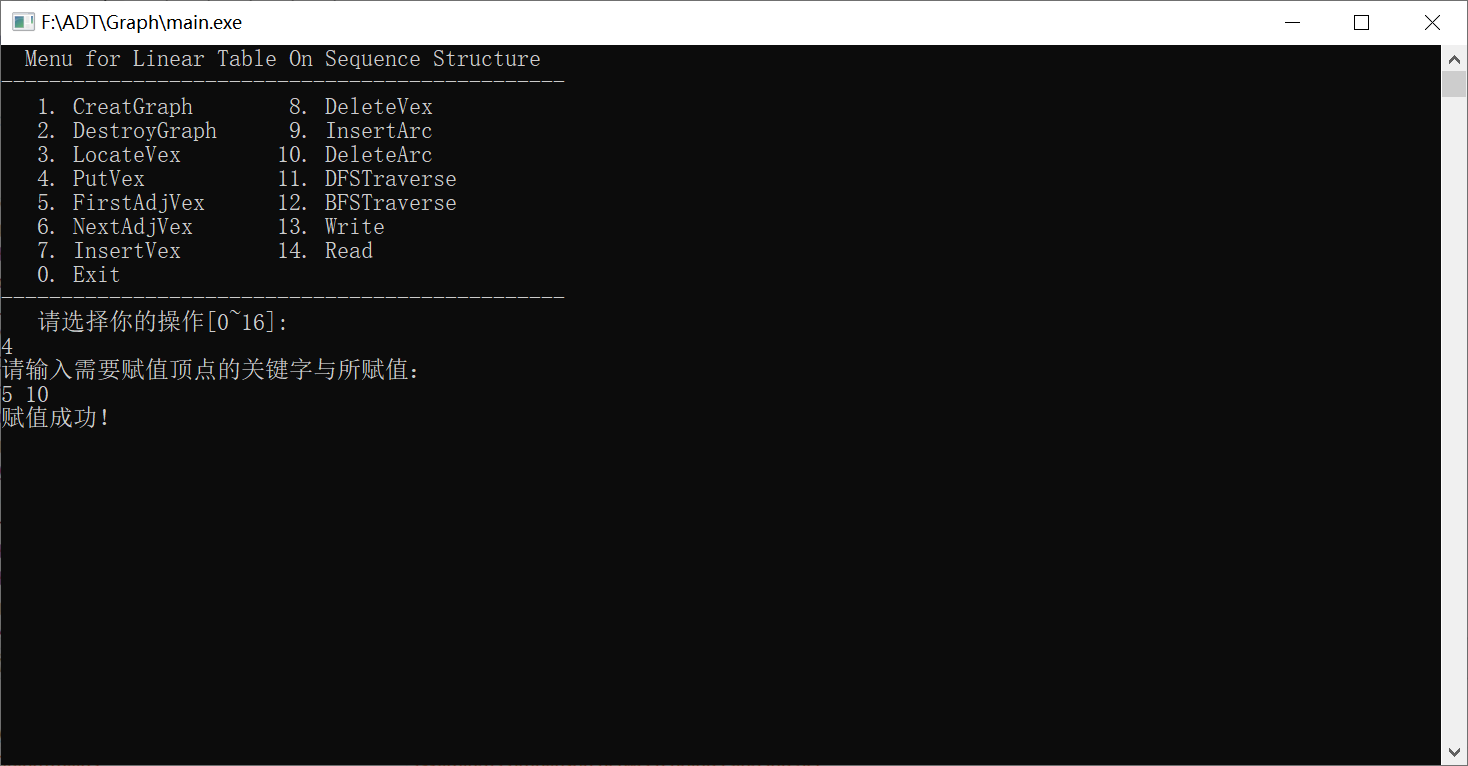


图4-6 测试-顶点赋值

1. 执行功能5，获得第一邻接点。

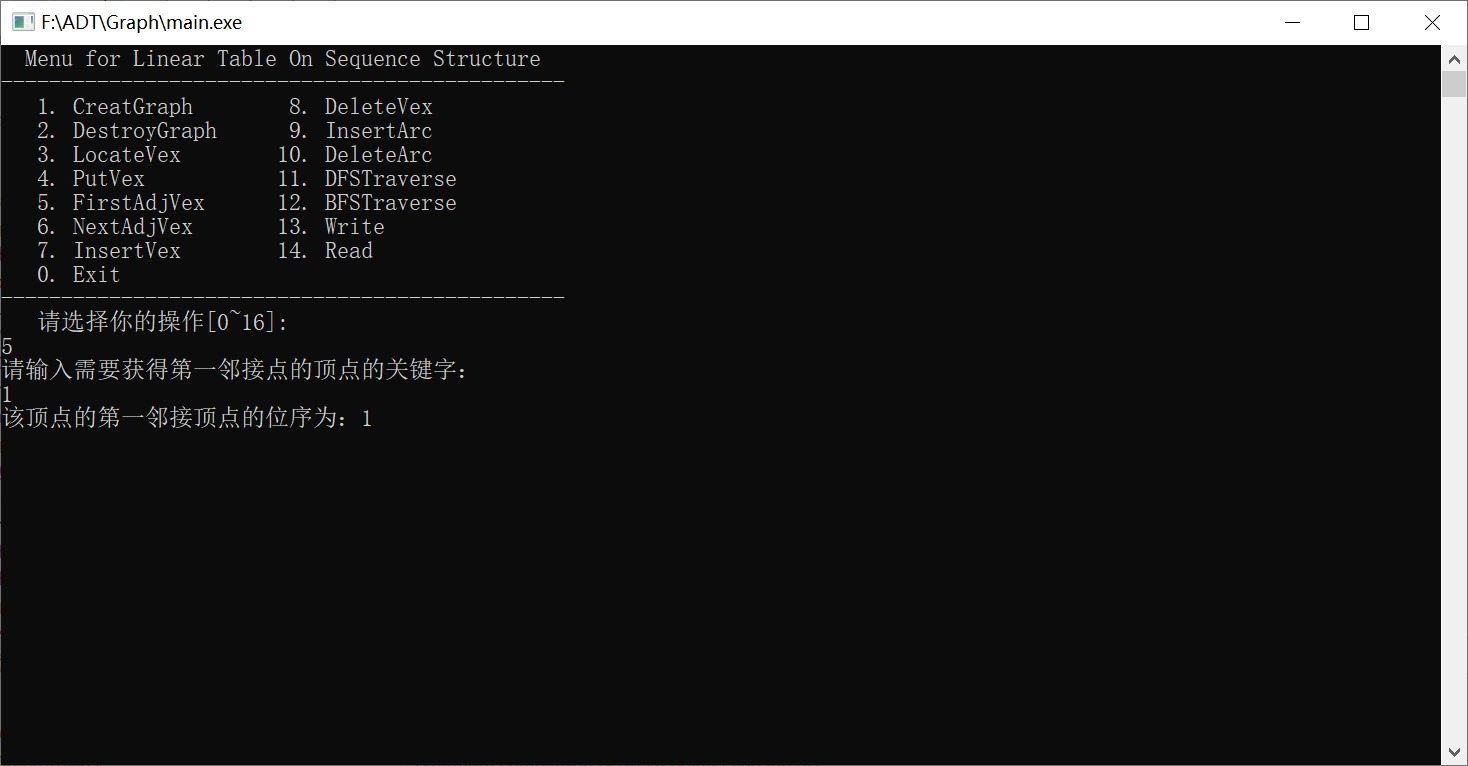


图4-7 测试-获得第一邻接点

1. 执行功能6，获得下一邻接点。

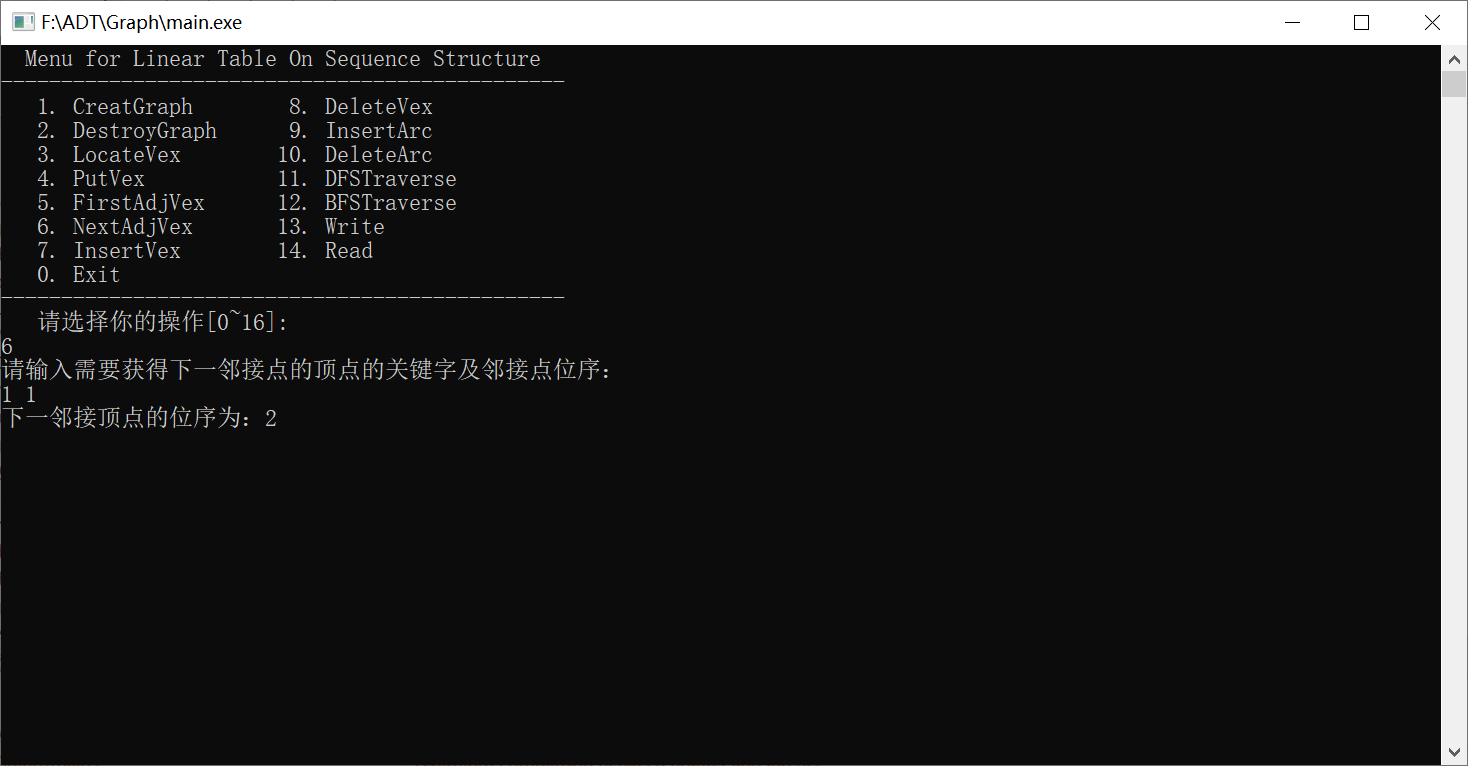


图4-8 测试-获得下一邻接点

1. 执行功能7，插入顶点。

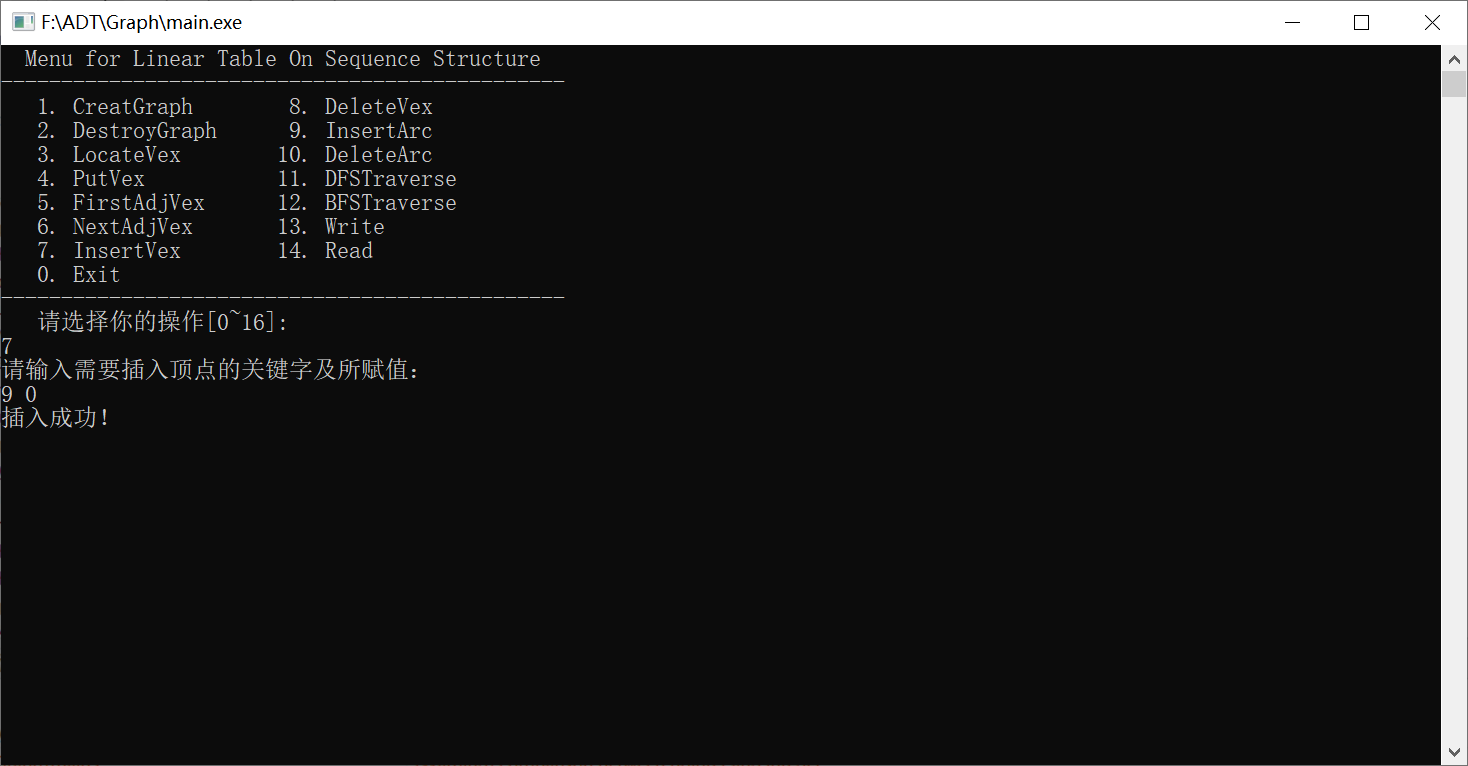


图4-9测试-插入顶点

1. 执行功能9，插入两个弧。

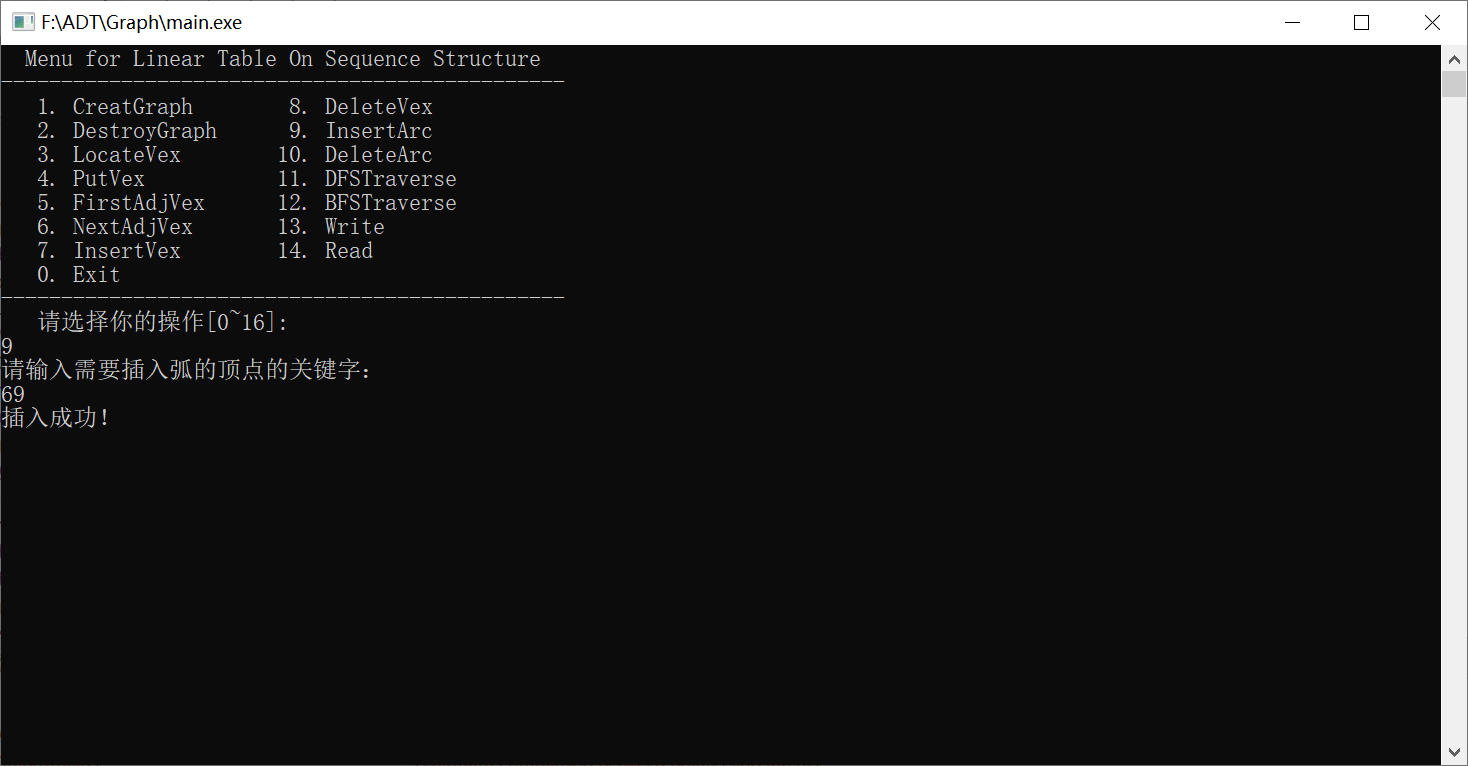


图4-10 测试-插入弧

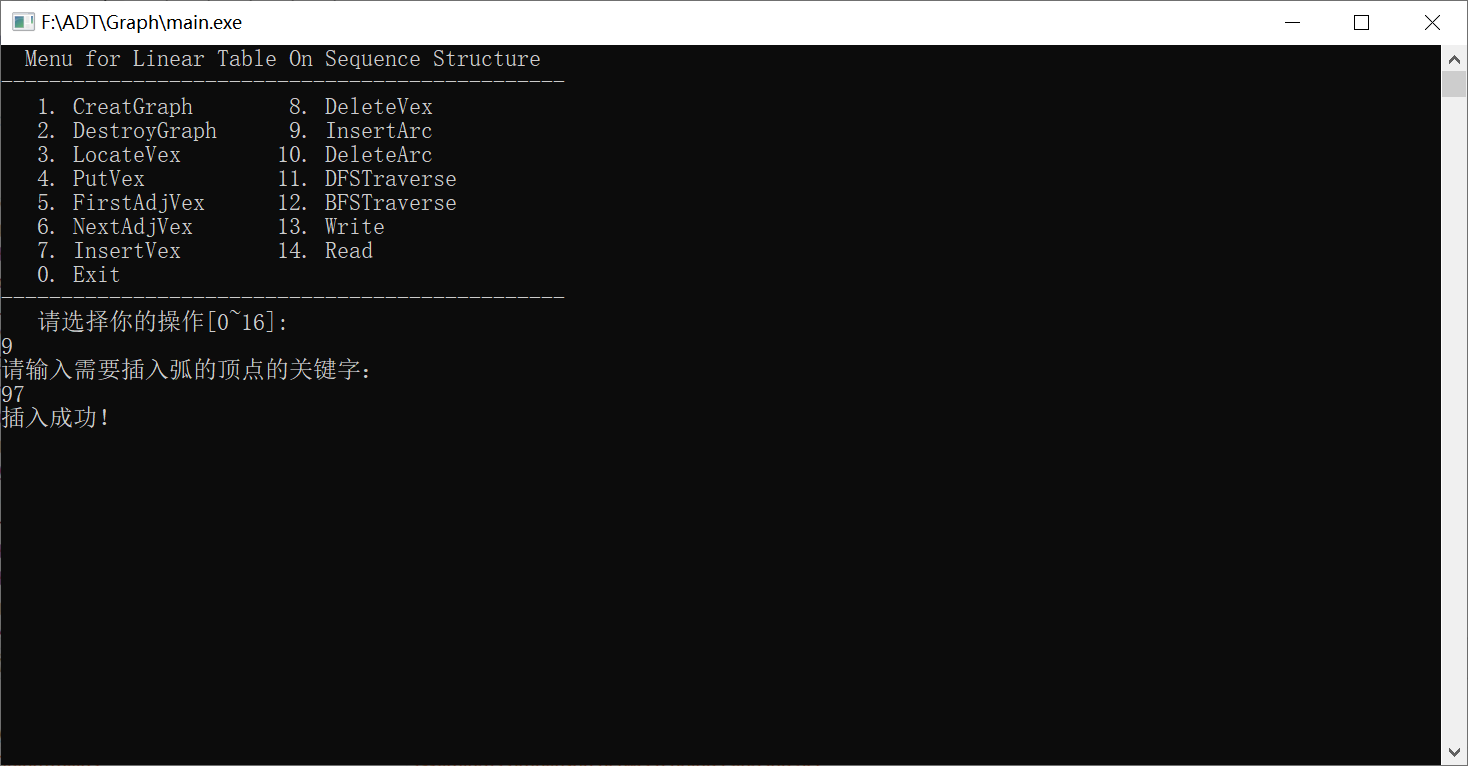


图4-11 测试-插入弧

1. 执行功能10，删除弧。

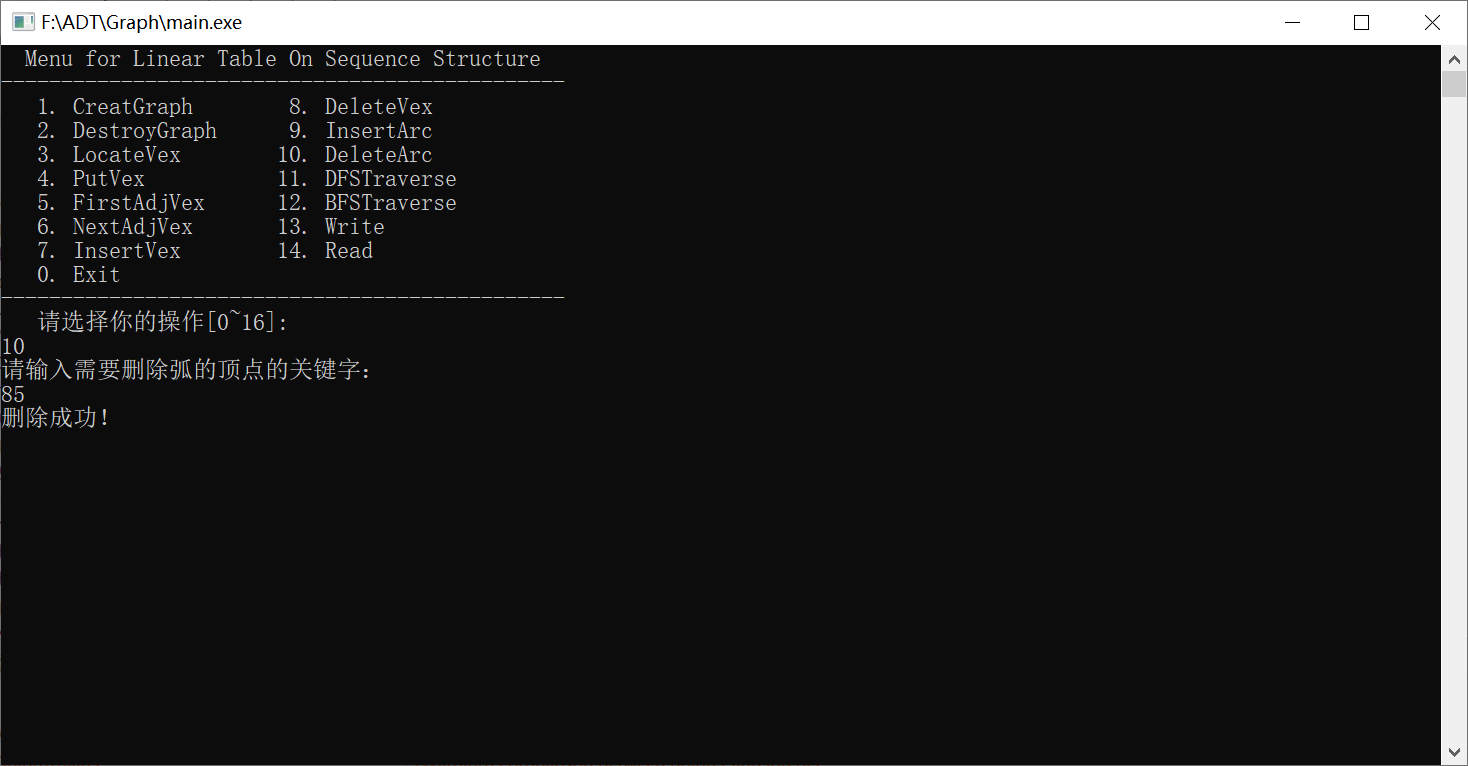


图4-12测试-删除弧

1. 执行功能8，删除顶点a和6。

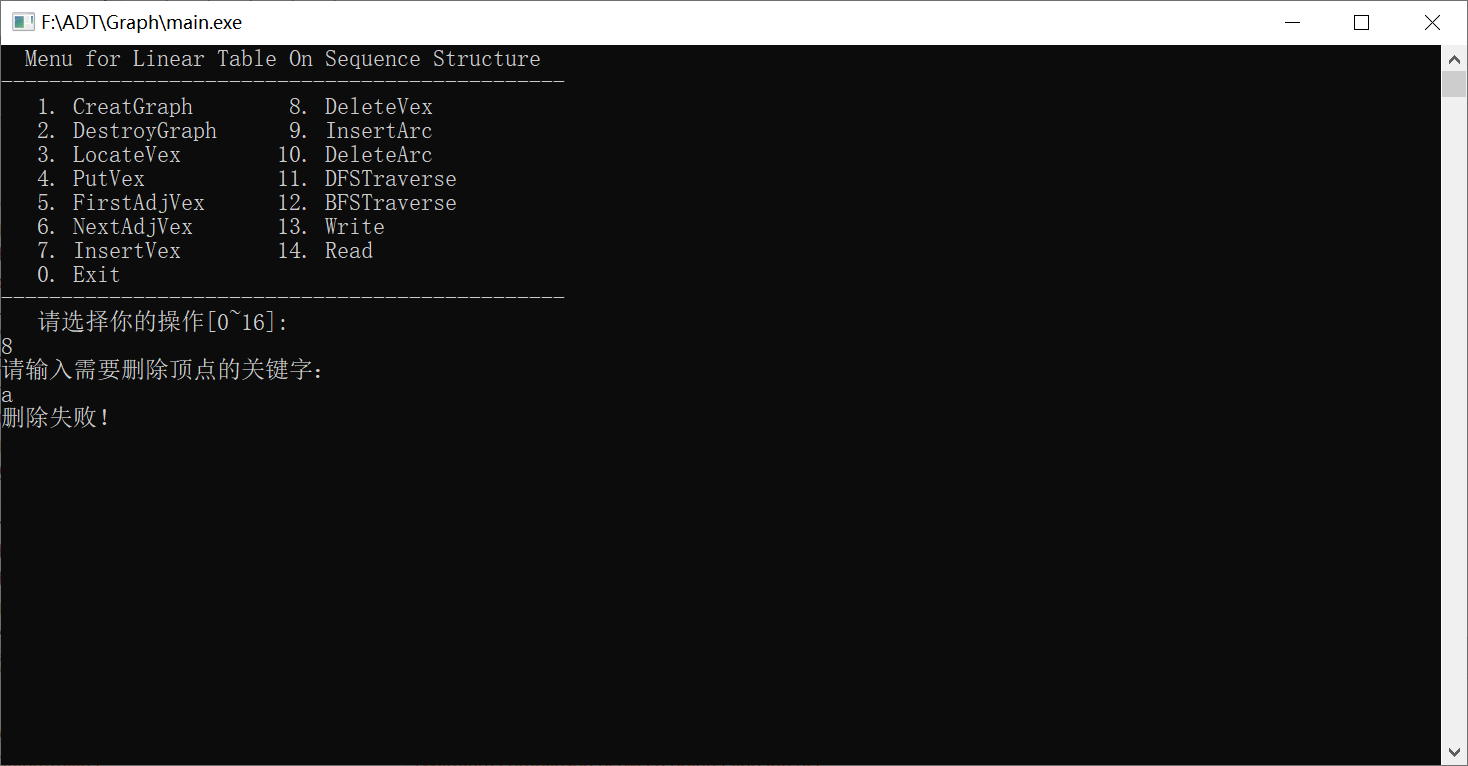


图4-13测试-删除顶点

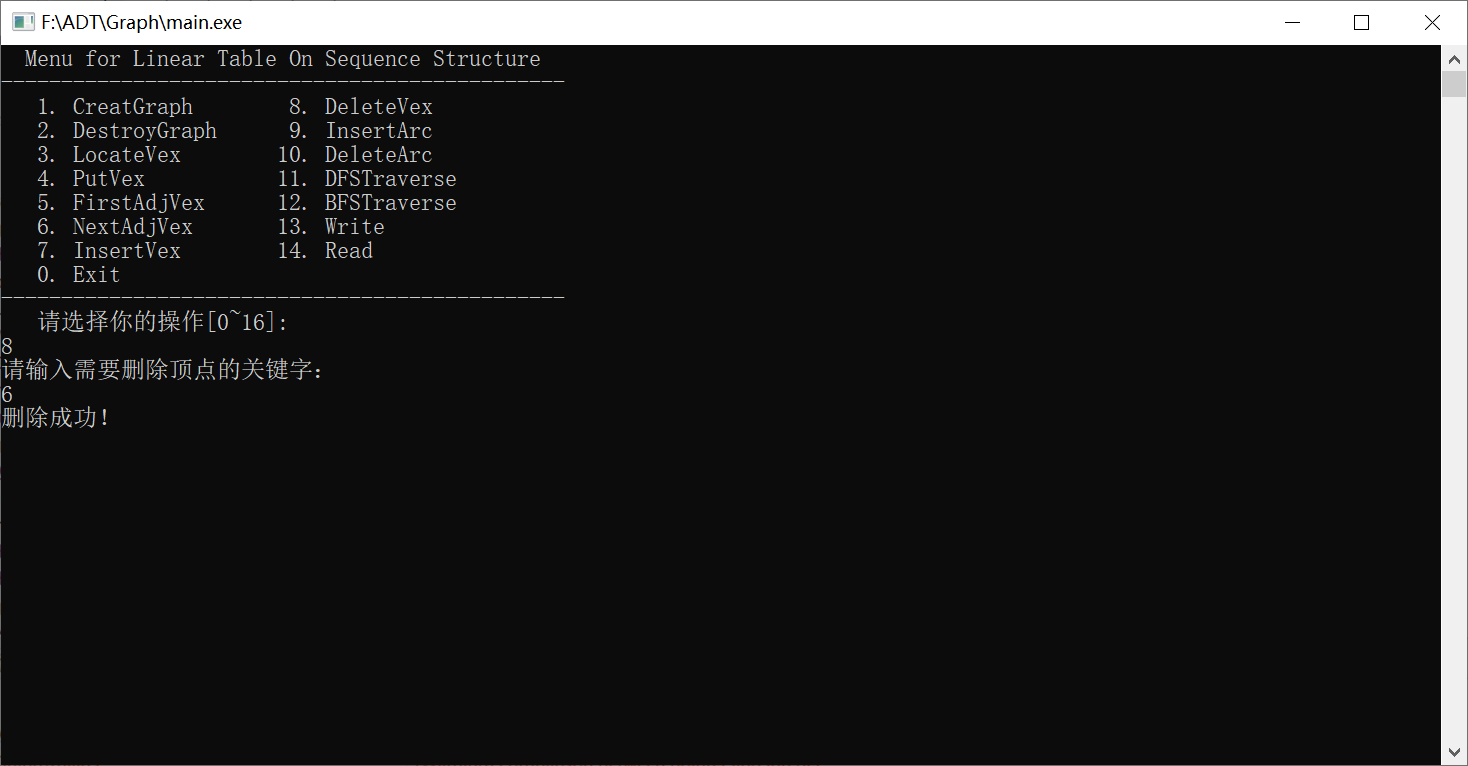


图4-14测试-删除顶点

1. 执行功能11，深度优先遍历。

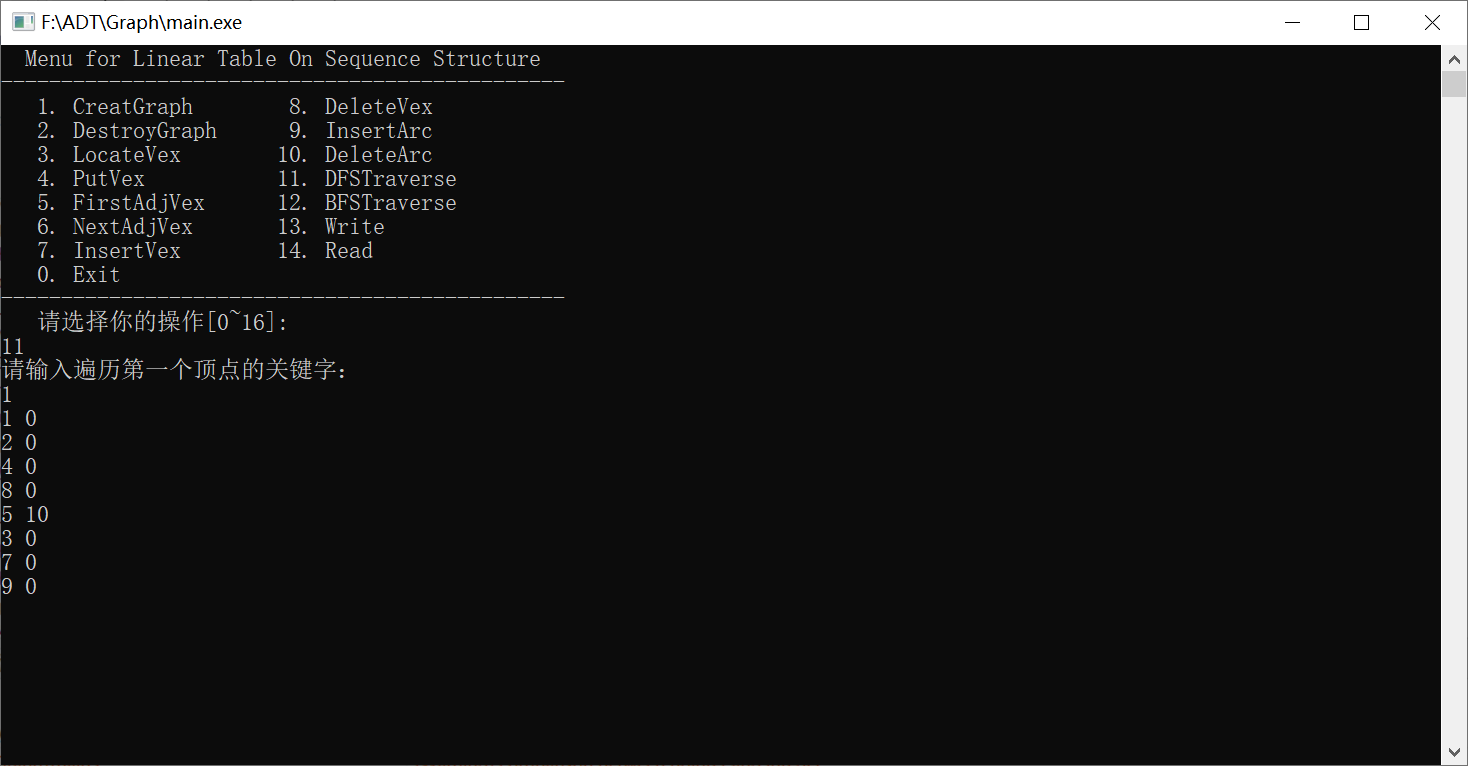


图4-15测试-深度优先遍历

1. 执行功能12，广度优先遍历。

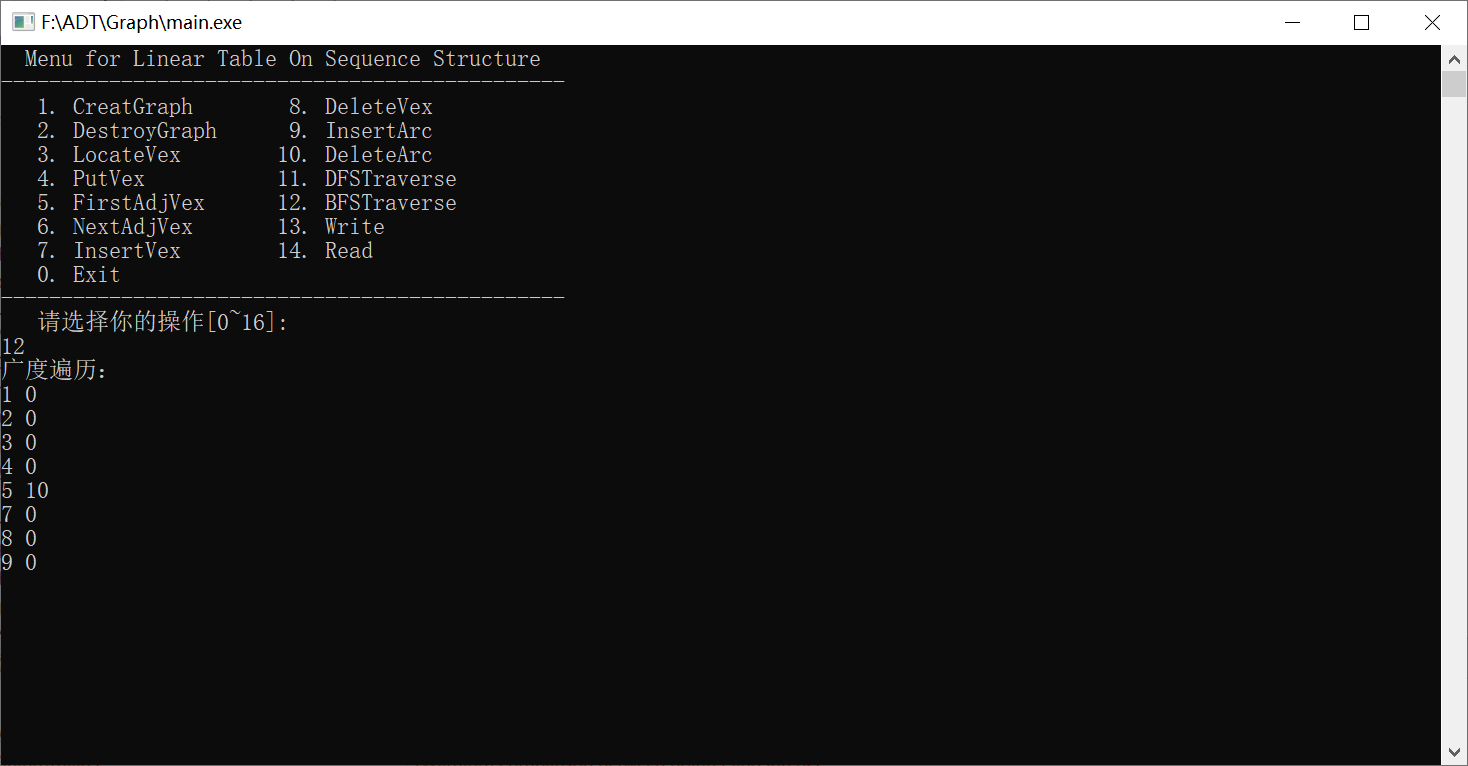


图4-16 测试-广度优先遍历

1. 执行功能8，删除顶点1。

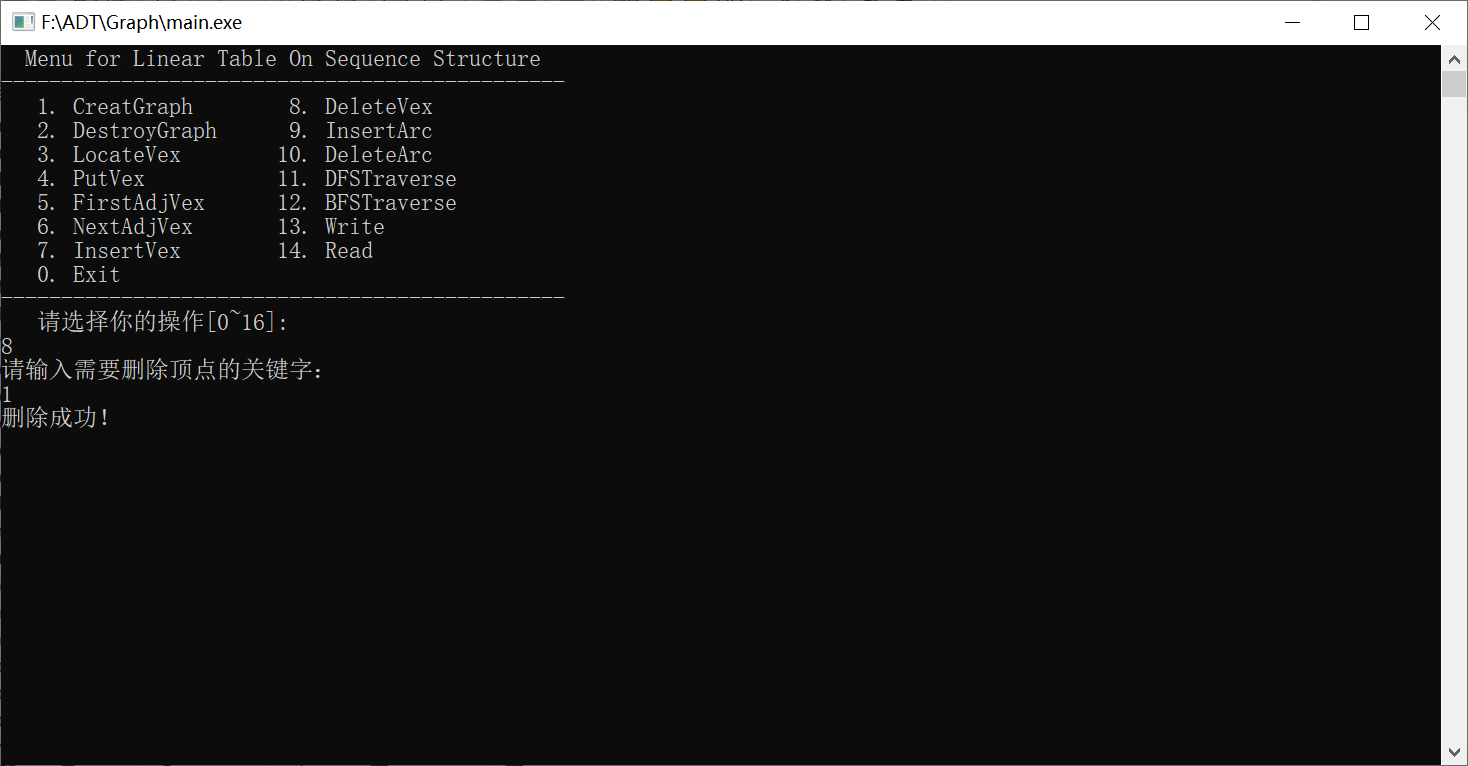


图4-17 测试-删除顶点

1. 执行功能11，深度优先遍历。

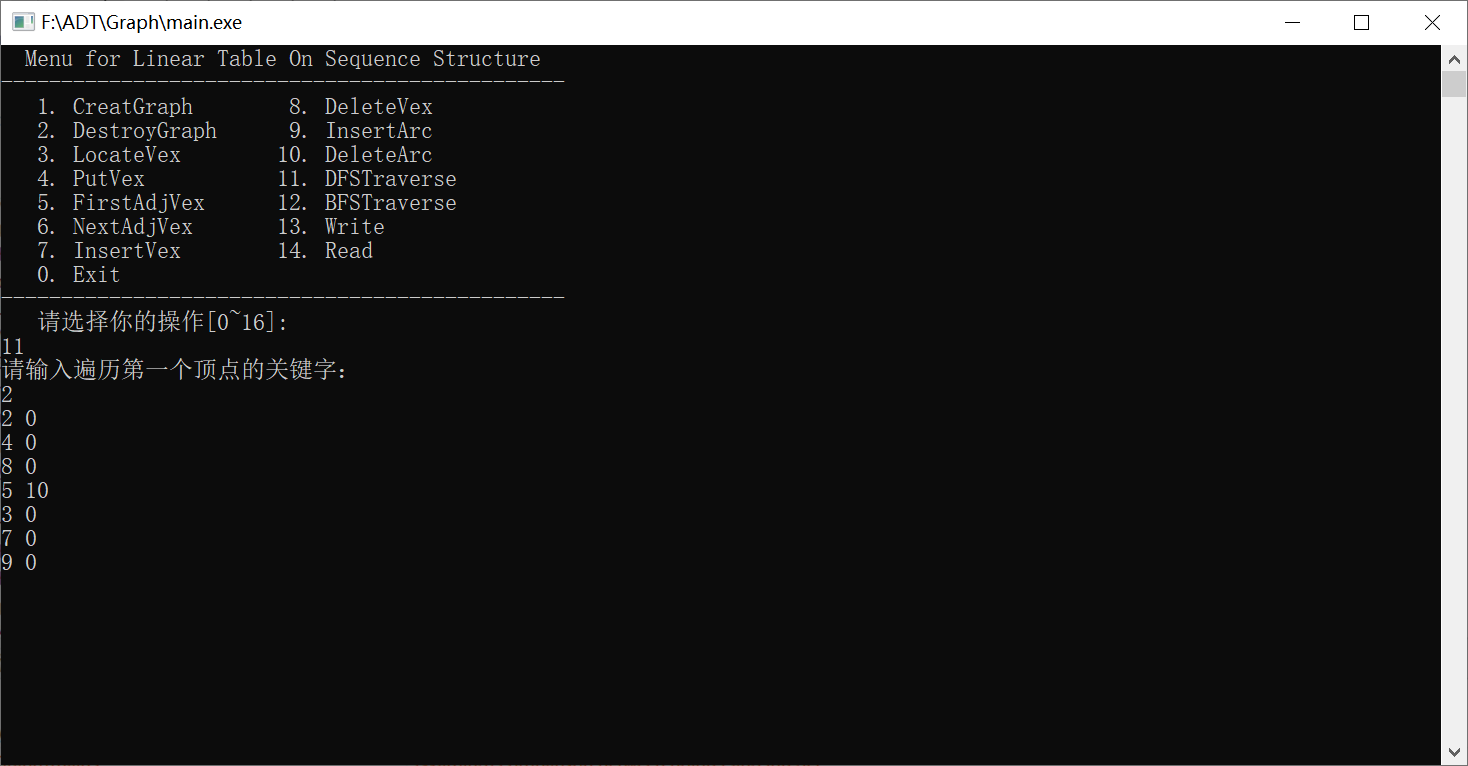


图4-18测试-深度优先遍历

1. 执行功能12，广度优先遍历。

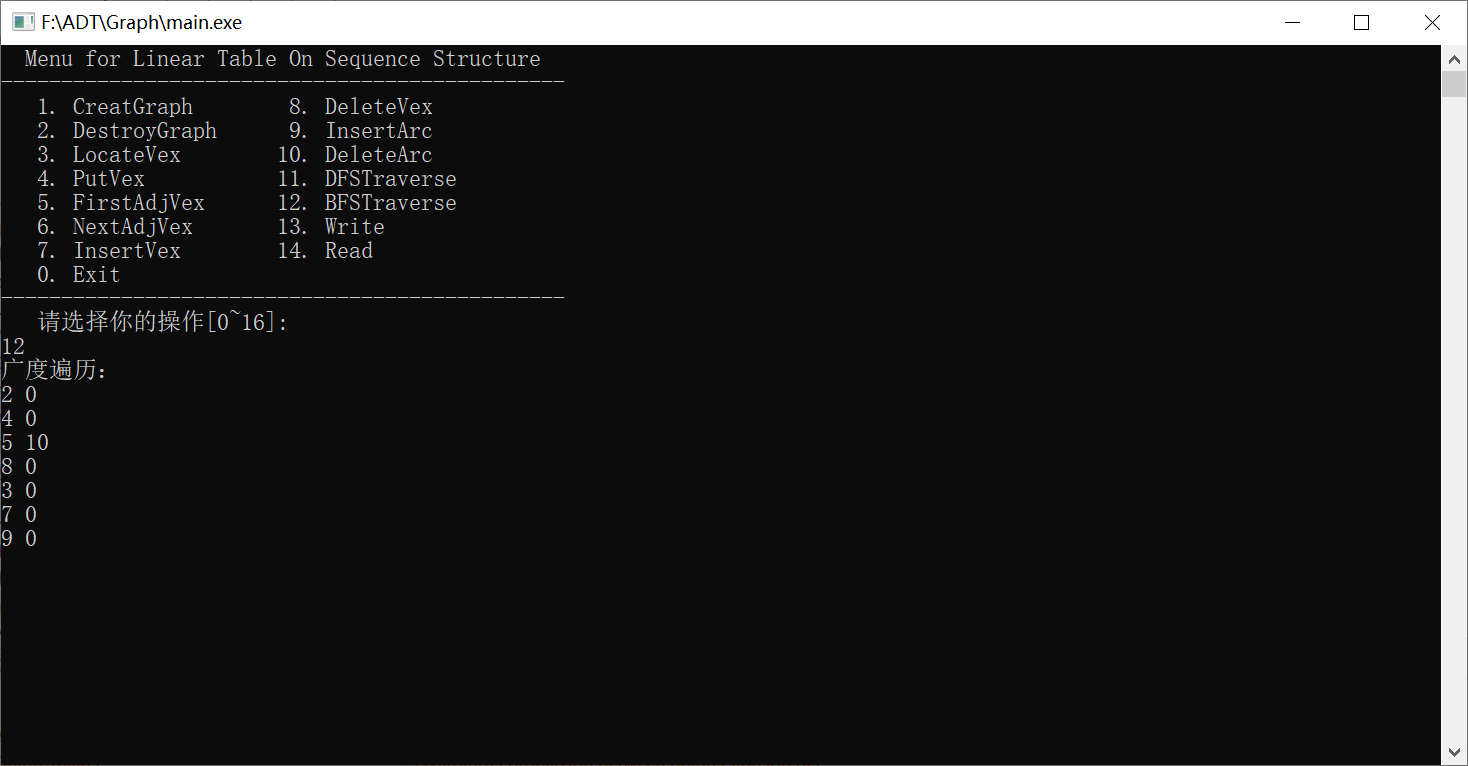


图4-19 测试-广度优先遍历

1. 执行功能13，写入文件。

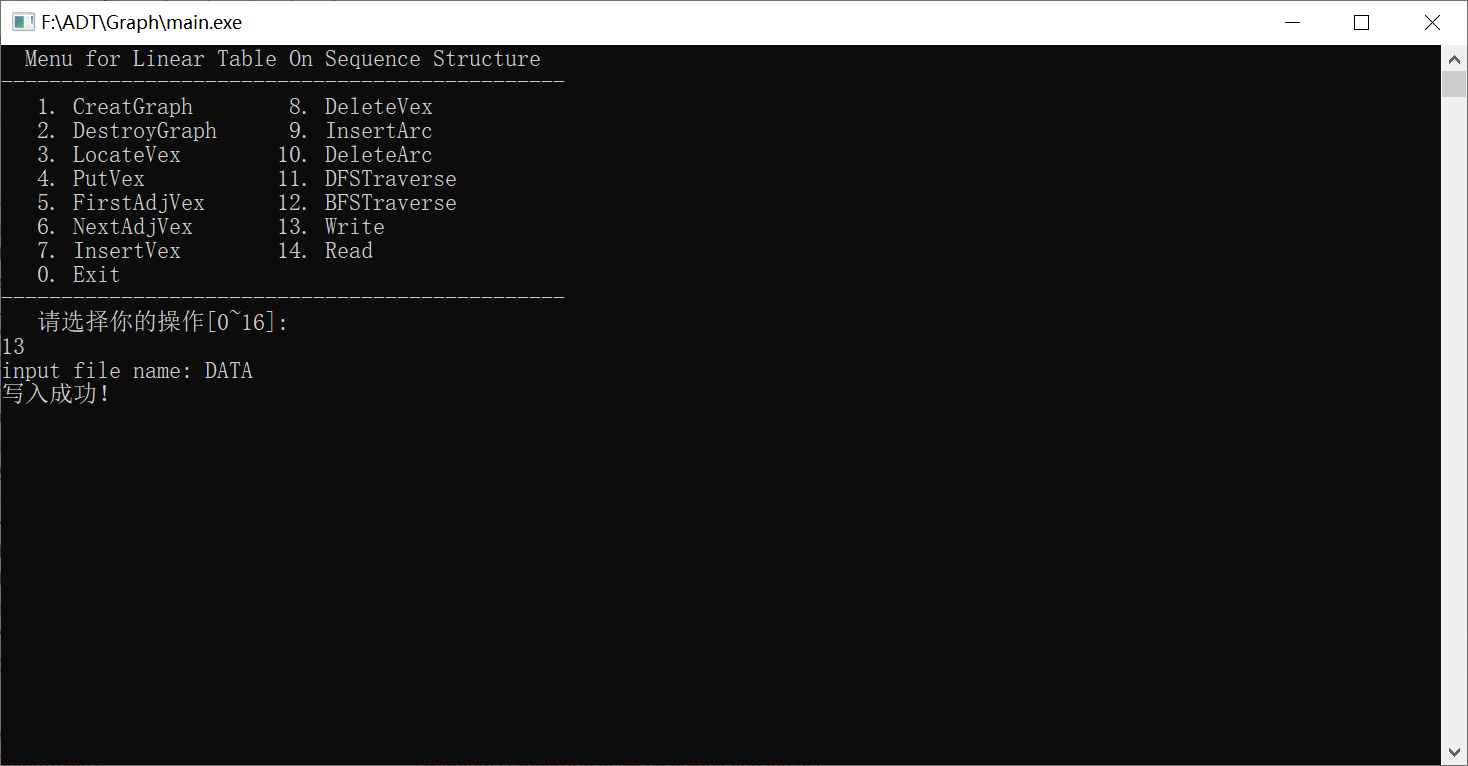


图4-20 测试-写入文件

1. 执行功能2，销毁图2。

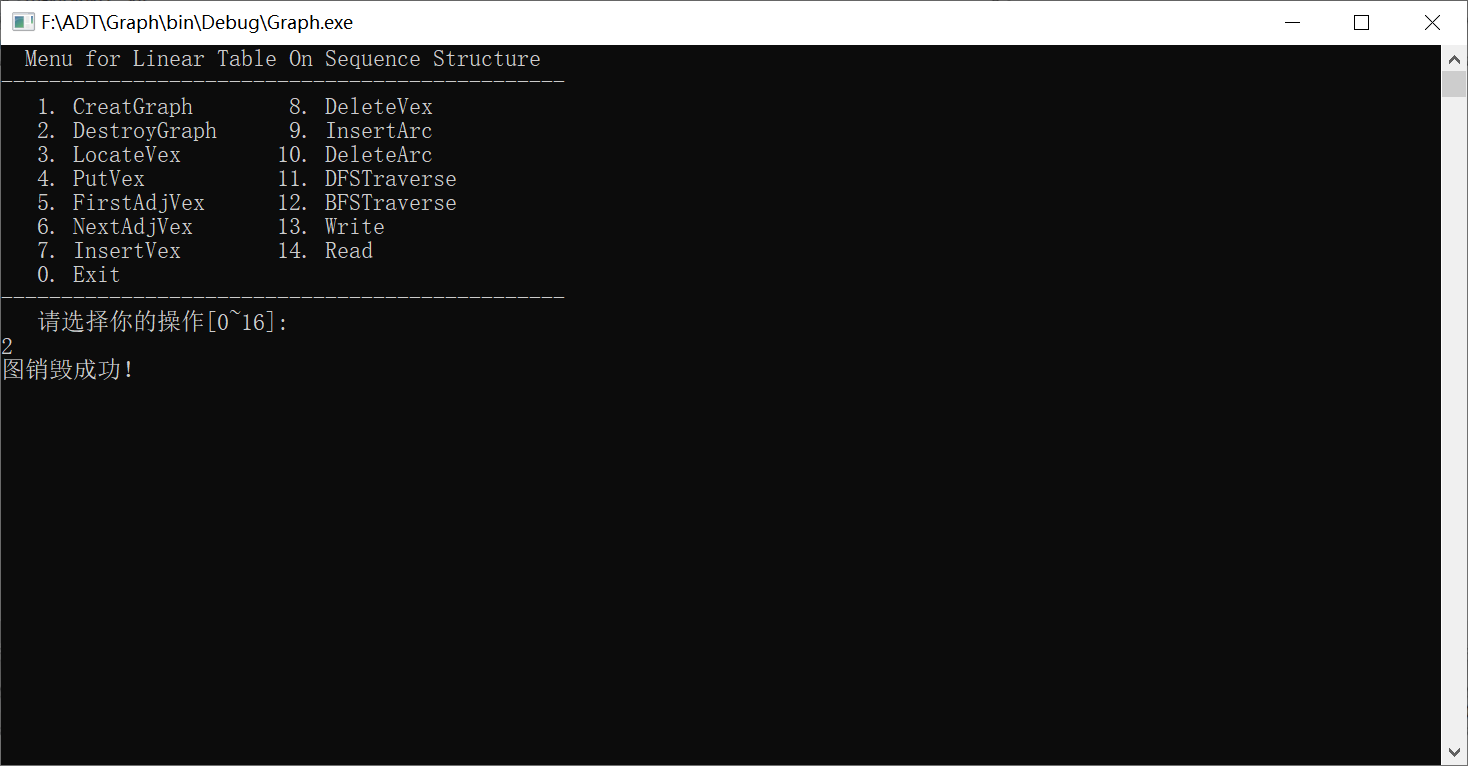


图4-21 测试-2销毁图

1. 执行功能0，退出对图1的操作, 重新选择图2。

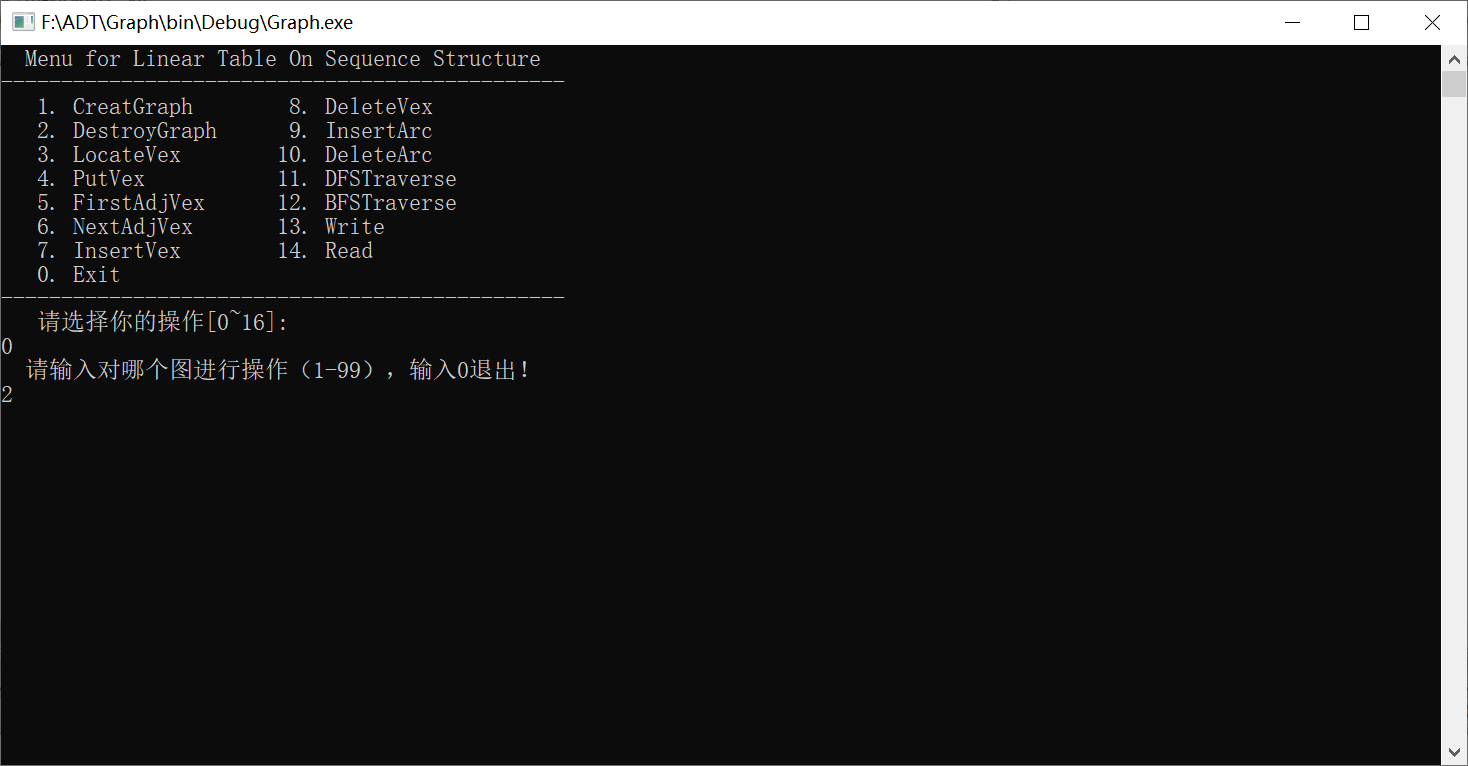


图4-22 测试-选择图2

1. 执行功能14，读取文件。

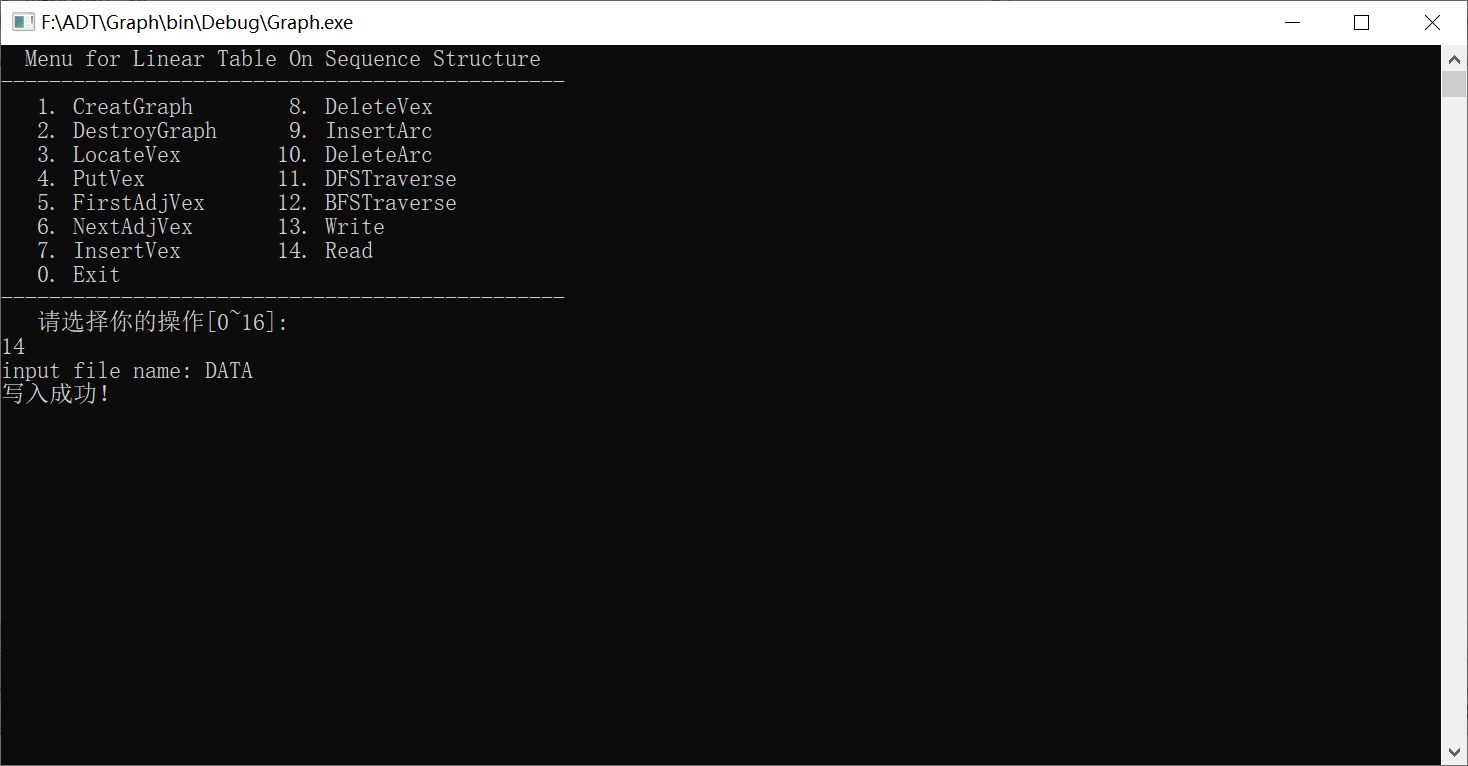


图4-23 测试-读取文件

1. 执行功能12，广度优先遍历图2。

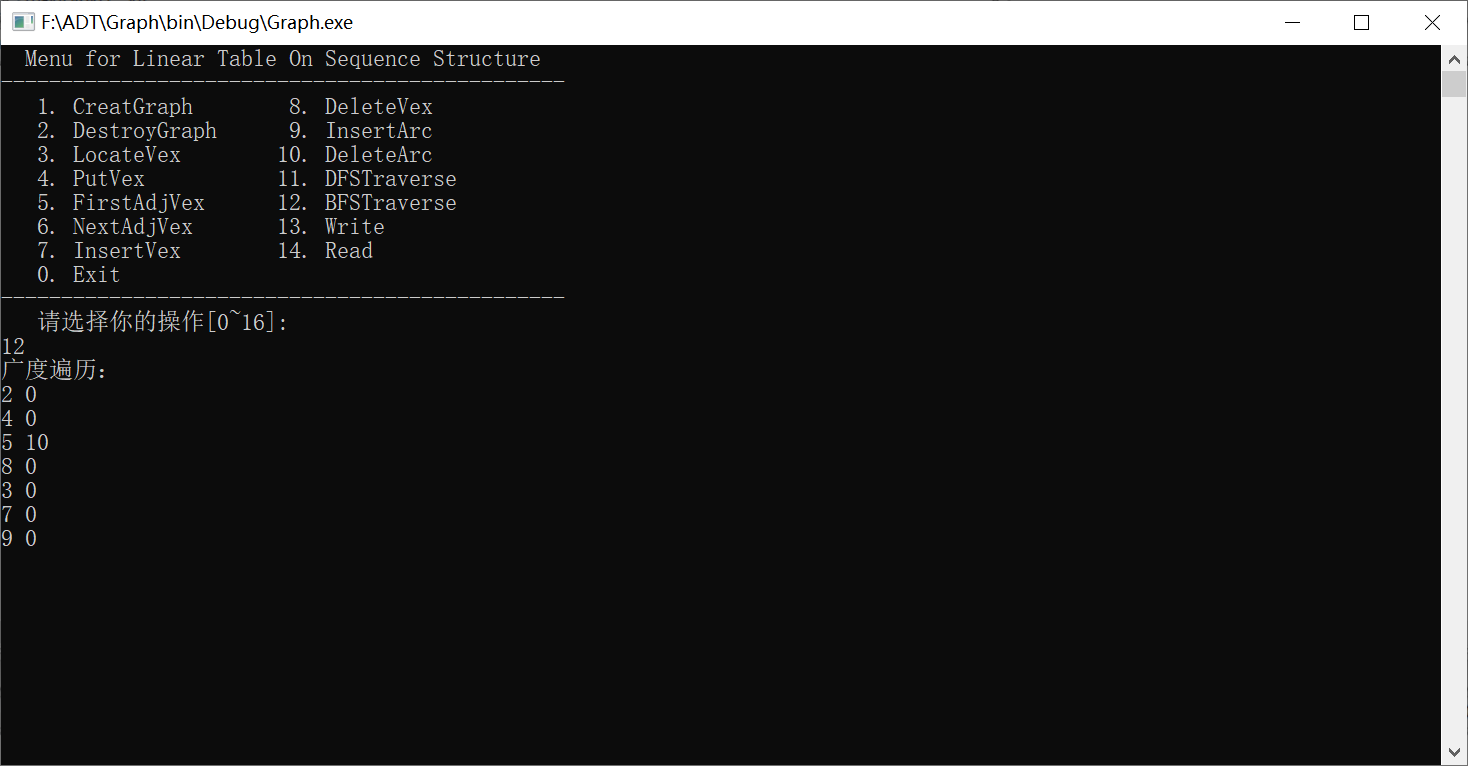


图4-24 测试-广度优先遍历

1. 执行功能0，退出对图2的操作，退出系统。

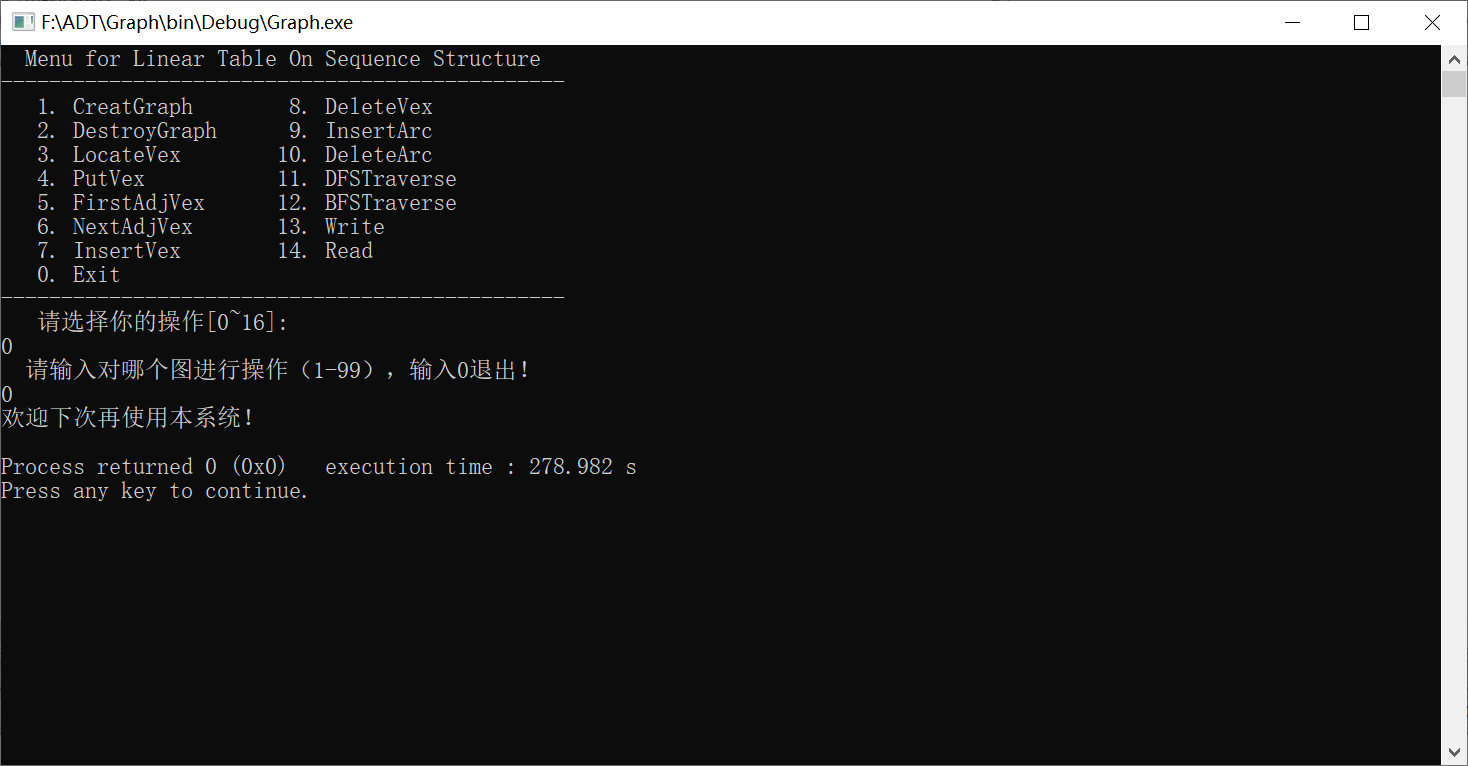


图4-25 测试-0退出操作

## 4.5 实验小结

1. 指针使用时一定要注意空指针的情况。在该次实验过程中，因为有些地方没有注意判断指针是不是空指针，调试时出现黄框。首先在创建图时，一开始选用头插法，后来写完代码后又尝试了尾插法，发现尾插法指针运用比头插法需求高一点，容易出现了空指针的情况。
2. 我构造的图的信息有点多，顶点含有关键字、位序还有序号。关键字和位序随着顶点一起，是唯一且固定的。位序即G.vertices[i]中的i值，在插入和删除顶点时顶点的位序会改变，所以实验中另外写了由关键字查找位序、序号和由位序、序号查找关键字的函数，以便于实验中运用。
3. 深度优先遍历和广度优先遍历涉及到了栈和队列的应用。栈和队列的运用一开始不是特别熟练，经过这次练习深入了许多。深度优先遍历采用了递归，因此输入时需要输入第一个开始的顶点。广度优先遍历不用。
4. 该次实验代码编写的过程中，对前几次的代码中的知识进行了一定的复习，C语言的“->”和“.”的运用，顺序表和链表的区别和共同点，比如顺序表创建、销毁、插入和删除时需要传入一级指针，而链表只在创建、销毁或者清空时传入二级指针。其余都是一级指针。
5. 写代码之前一定要先将代码的结构整理清楚，比如说该实验中图的信息、顶点的信息、顶点与弧邻接表的联系等等。只有明白了联系和结构才能更高效地写代码。

# 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

#define NO -1

typedef int status;

typedef int ElemType; //数据元素类型定义

/\*-------page 22 on textbook -------\*/

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct{ //顺序表（顺序结构）的定义

ElemType \*elem;

int length;

int listsize;

}SqList;

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

status InitList(SqList \*L);//创建

status DestroyList(SqList \*L);//销毁

status ClearList(SqList \*L);//清空

status ListEmpty(SqList L);//判断空表

status ListLength(SqList L);//求表长

status GetElem(SqList L,int i,ElemType \*e);//获得元素

int LocateElem(SqList L,ElemType e); //查找元素

status PriorElem(SqList L,ElemType cur,ElemType \*pre\_e);//获得前驱

status NextElem(SqList L,ElemType cur,ElemType \*next\_e);//获得后驱

status ListInsert(SqList \*L,int i,ElemType e);//插入

status ListDelete(SqList \*L,int i,ElemType \*e);//删除

status ListTrabverse(SqList L); //遍历

status Wirte(SqList L);//写入文件

status Read(SqList \*L);//读取文件

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

int main(void){

SqList M[100]; SqList L;//声明所需变量

int t,i;

ElemType p,cur;

ElemType \*pre\_e,\*next\_e,\*e;

t=1;

while(t){

int op=1;

printf(" 请输入对哪个线性表进行操作（1-99），输入0退出！\n");

scanf("%d",&t);//多个线性表的实现

if(t==0)

break;//0时退出

else {

if(t>99||t<0){

printf("选择失败！\n");

getchar();

}

else {

M[t].elem=NULL;

L=M[t];

while(op){

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

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

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

printf(" 1. InitList 8. PriorElem \n");

printf(" 2. DestroyList 9. NextElem\n");

printf(" 3. ClearList 10. ListInsert\n");

printf(" 4. ListEmpty 11. ListDelete\n");

printf(" 5. ListLength 12. ListTrabverse\n");

printf(" 6. GetElem 13. Write\n");

printf(" 7. LocateElem 14. Read\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~14]:\n");

scanf("%d",&op);

switch(op){

case 1:

if(InitList(&L)==OK)

printf("线性表创建成功！\n");

else

printf("线性表创建失败！\n");

getchar();getchar();

break;

case 2:

if(DestroyList(&L)==OK)

printf("线性表销毁成功！\n");

else

printf("线性表销毁失败！\n");

getchar();getchar();

break;

case 3:

if(L.elem!=NULL){

ClearList(&L);

printf("清空线性表！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 4:

if(L.elem!=NULL){

if(ListEmpty(L)==TRUE)

printf("线性表为空！\n");

else

printf("线性表不为空！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 5:

if(L.elem!=NULL){

ListLength(L);

printf("线性表的长度为%d！\n",L.length);

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 6:

if(L.elem!=NULL){

printf("请输入想要获得元素的位置：\n");

scanf("%d",&i);

if(GetElem(L,i,&e)!=ERROR){

printf("第%d的元素为%d！\n",i,e);

}

else

printf("该位置不存在\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 7:

if(L.elem!=NULL){

printf("请输入需要查找的元素：\n");

scanf("%d",&p);

if(LocateElem(L,p))

printf("第%d个元素与该元素相同\n",LocateElem(L,p));

else

printf("不存在相同的元素\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 8:

if(L.elem!=NULL){

printf("请输入需要获得其前驱的元素：\n");

scanf("%d",&cur);

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

printf("该元素的前驱为%d",pre\_e);

else

printf("操作失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 9:

if(L.elem!=NULL){

printf("请输入需要获得其后继的元素：\n");

scanf("%d",&cur);

if(NextElem(L,cur,&next\_e))

printf("该元素的后继为%d",next\_e);

else

printf("操作失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 10:

if(L.elem!=NULL){

printf("请依次输入：在第\_个位置之前插入元素\_\n");

scanf("%d",&i);

getchar();

scanf("%d",&p);

if(ListInsert(&L,i,p)==OK)

printf("插入成功！\n");

else

printf("插入失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 11:

if(L.elem!=NULL){

printf("请依次输入：删除第\_个位置的元素\n");

scanf("%d",&i);

if(ListDelete(&L,i,&e)==OK)

printf("删除成功！\n");

else

printf("删除失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 12:

if(L.elem!=NULL){

if(ListTrabverse(L)==0)

printf("线性表是空表！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 13:

if(L.elem!=NULL){

if(Wirte(L)==OK)

printf("写入成功！\n");

else

printf("写入失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 14:

if(L.elem!=NULL){

if(Read(&L)==OK)

printf("读取成功！\n");

else

printf("读取失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 0:

break;

}

}//end of switch

}

}

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

status InitList(SqList \*L){//构造一个空的顺序线性表L

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

if(!L->elem)

exit(OVERFLOW);

L->length=0;

L->listsize=LIST\_INIT\_SIZE;

return OK;

}

status DestroyList(SqList \*L){//销毁线性表L

if(L->elem){

free(L->elem);//释放线性表空间

L->elem=NULL;

L->length=0;

L->listsize=0;

return OK;

}

else

return ERROR;

}

status ClearList(SqList \*L){//将L置为空表

if(L->elem!=NULL){

L->length=0;

return OK;

}

else

printf("线性表不存在！\n");

return NO;

}

status ListEmpty(SqList L){//判断L是否为空

if(L.length==0)

return TRUE;

else

return FALSE;

}

status ListLength(SqList L){//计算L中元素个数

return L.length;

}

status GetElem(SqList L,int i,ElemType \*e){//用e返回第i个元素的值

if(L.length==0||i<1||i>L.length)

return ERROR;//顺序线性表为空或i的值不合法

\*e=L.elem[i-1];

return OK;

}

int LocateElem(SqList L,ElemType e){//返回L中与e相等的元素的位序，无返回0

int i;

if(L.elem!=NULL){

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

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

return ++i;

return ERROR;

}

} //简化过

status PriorElem(SqList L,ElemType cur,ElemType \*pre\_e){//返回cur元素的前驱

int i;

i=LocateElem(L,cur);//找到cur的位置

if(i==1||i==ERROR)

return ERROR;

else{

\*pre\_e=L.elem[i-2];

return OK;

}

}

status NextElem(SqList L,ElemType cur,ElemType \*next\_e){//返回cur元素的后继

int i;

i=LocateElem(L,cur);

if(i==L.length||i==ERROR)

return ERROR;

else{

\*next\_e=L.elem[i];

return OK;

}

}

status ListInsert(SqList \*L,int i,ElemType e){//在顺序线性表中第i个元素之前插入新的元素e，1<=i<=listlenth+1

ElemType \*newbase;

ElemType \*q,\*p;

ElemType t;

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

return ERROR;//i不合法

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

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

if(!newbase)

exit(OVERFLOW);//存储分配失败

L->elem=newbase;//新基址

L->listsize+=LISTINCREMENT;

}

q=&(L->elem[i-1]);

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

\*(p+1)=\*p;//插入位置及之后的元素右移

L->elem[i-1]=e;

++(L->length);

return OK;

}

status ListDelete(SqList \*L,int i,ElemType \*e){//在顺序线性表中删除第i个元素，并用e返回其值

ElemType \*q,\*p;

if(i<1||(i>L->length))

return ERROR;//i不合法

e=L->elem[i-1];//被删除元素值赋给e

p=L->elem+L->length-1;//令p为表尾位置

for(q=&(L->elem[i]);q<=p;q++)

\*(q-1)=\*q;//被删元素之后的元素依次前移

--(L->length);

return OK;

}

status ListTrabverse(SqList L){//遍历表

int i;

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

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

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

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

return L.length;

}

status Wirte(SqList L) {

FILE \*fp=NULL;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

if ((fp=fopen(filename,"wb"))==NULL){

printf("File open error\n ");

return FALSE;

}

//写文件的方法

fwrite(L.elem,sizeof(ElemType),L.length,fp);

//这里是1次性写入，对于其它物理结构，

//也可以先写入表长，再写入全部元素,这样读入会更方便

fclose(fp);

return OK;

}

status Read(SqList \*L) {

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

//读文件的方法

L->length=0;

if ((fp=fopen(filename,"rb"))==NULL){

printf("File open error\n ");

return FALSE;

}

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

L->length++;

//这里从文件中逐个读取数据元素恢复顺序表，对于不同的物理结构，可通过读取的数据元素恢复内存中的物理结构。

fclose(fp);

return OK;

}

# 附录B 基于链式存储结构线性表实现的源程序

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

/\*----------预定义--------\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define NO -1

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef struct LNode{ //链式表（链式结构）的定义

ElemType data;

struct LNode \*next;

}LinkList;

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

status InitList(LinkList \*\*L);//创建

status DestroyList(LinkList \*\*L);//销毁

status ClearList(LinkList \*L);//清空

status ListEmpty(LinkList \*L);//判断空表

status ListLength(LinkList \*L);//求表长

status GetElem(LinkList \*L,int i,ElemType \*e);//获得元素

int LocateElem(LinkList \*L,ElemType e); //查找元素

status PriorElem(LinkList \*L,ElemType cur,ElemType \*pre\_e);//获得前驱

status NextElem(LinkList \*L,ElemType cur,ElemType \*next\_e);//获得后驱

status ListInsert(LinkList \*L,int i,ElemType e);//插入

status ListDelete(LinkList \*L,int i,ElemType \*e);//删除

status ListTrabverse(LinkList \*L); //遍历

status Wirte(LinkList \*L);//写入文件

status Read(LinkList \*\*L);//读取文件

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

int main(void){

LinkList \*M[100]; LinkList \*L;//声明所需变量

int t,i;

ElemType p,cur;

ElemType \*pre\_e,\*next\_e,\*e;

t=1;

while(t){

int op=1;

printf(" 请输入对哪个线性表进行操作（1-99），输入0退出！\n");

scanf("%d",&t);//多个线性表的实现

if(t==0)

break;//0时退出

else {

if(t>99||t<0){

printf("选择失败！\n");

getchar();

}

else {

M[t]=NULL;

L=M[t];

while(op){

system("cls"); //菜单目录;

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

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

printf(" 1. InitList 8. PriorElem \n");

printf(" 2. DestroyList 9. NextElem\n");

printf(" 3. ClearList 10. ListInsert\n");

printf(" 4. ListEmpty 11. ListDelete\n");

printf(" 5. ListLength 12. ListTrabverse\n");

printf(" 6. GetElem 13. Write\n");

printf(" 7. LocateElem 14. Read\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~14]:\n");

scanf("%d",&op);

switch(op){//进入switch选择

case 1:

if(InitList(&L)==OK)

printf("线性表创建成功！\n");

else

printf("线性表创建失败！\n");

getchar();getchar();

break;

case 2:

if(DestroyList(&L)==OK)

printf("线性表销毁成功！\n");

else

printf("线性表销毁失败！\n");

getchar();getchar();

break;

case 3:

if(L!=NULL){

ClearList(L);

printf("清空线性表！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 4:

if(L!=NULL){

if(ListEmpty(L)==TRUE)

printf("线性表为空！\n");

else

printf("线性表不为空！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 5:

if(L!=NULL){

i=ListLength(L);

printf("线性表的长度为%d！\n",i);

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 6:

if(L!=NULL){

printf("请输入想要获得元素的位置：\n");

scanf("%d",&i);

if(GetElem(L,i,&e)!=ERROR){

printf("第%d的元素为%d！\n",i,e);

}

else

printf("该位置不存在\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 7:

if(L!=NULL){

printf("请输入需要查找的元素：\n");

scanf("%d",&p);

if(LocateElem(L,p))

printf("第%d个元素与该元素相同\n",LocateElem(L,p));

else

printf("不存在相同的元素\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 8:

if(L!=NULL){

printf("请输入需要获得其前驱的元素：\n");

scanf("%d",&cur);

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

printf("该元素的前驱为%d",pre\_e);

else

printf("操作失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 9:

if(L!=NULL){

printf("请输入需要获得其后继的元素：\n");

scanf("%d",&cur);

if(NextElem(L,cur,&next\_e))

printf("该元素的后继为%d",next\_e);

else

printf("操作失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 10:

if(L!=NULL){

printf("请依次输入：在第\_个位置之前插入元素\_\n");

scanf("%d",&i);

getchar();

scanf("%d",&p);

if(ListInsert(L,i,p)==OK)

printf("插入成功！\n");

else

printf("插入失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 11:

if(L!=NULL){

printf("请依次输入：删除第\_个位置的元素\n");

scanf("%d",&i);

if(ListDelete(L,i,&e)==OK)

printf("删除成功！\n");

else

printf("删除失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 12:

if(L!=NULL){

if(ListTrabverse(L)==0)

printf("线性表是空表！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 13:

if(L!=NULL){

if(Wirte(L)==OK)

printf("写入成功！\n");

else

printf("写入失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 14:

if(L!=NULL){

if(Read(&L)==OK)

printf("读取成功！\n");

else

printf("读取失败！\n");

}

else

printf("线性表不存在！\n");

getchar();getchar();

break;

case 0:

break;

}

}//end of switch

}

}

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

status InitList(LinkList \*\*L){//构造一个线性表L

\*L = (LinkList\*)malloc(sizeof(LinkList)); //创建一个头结点

LinkList \*p=\*L;

p->next=NULL;

return OK;

}

status DestroyList(LinkList \*\*L){//销毁线性表L

LinkList \*p=\*L;

LinkList \*q;

if(p){

for(;p!=NULL;){//遍历单链表

q=p;

p=p->next;

free(q);//释放线性表空间

}

p=NULL;

\*L=p;

return OK;

}

else

return ERROR;

}

status ClearList(LinkList \*L){//将L置为空表

LinkList \*p;

LinkList \*q;

if(L!=NULL){

for(p=L->next;p!=NULL;){//遍历单链表

q=p;

p=p->next;

free(q);//释放线性表空间

}

L->next=NULL;//将头节点置为空

return OK;

}

else

printf("线性表不存在！\n");

return NO;

}

status ListEmpty(LinkList \*L){//判断L是否为空

if(L->next==NULL)

return TRUE;

else

return FALSE;

}

status ListLength(LinkList \*L){//计算L中元素个数

int i;

LinkList \*p=L->next;

for(i=0;p!=NULL;i++)//遍历单链表，计数

p=p->next;

return i;

}

status GetElem(LinkList \*L,int i,ElemType \*e){//用e返回第i个元素的值

int j;

if(i<1||i>ListLength(L))

return ERROR;//链式线性表为空或i的值不合法

LinkList \*p=L->next;//p指向首结点

for(j=1;j<i;j++)//遍历到第i位

p=p->next;

\*e=p->data;

return OK;

}

int LocateElem(LinkList \*L,ElemType e){//返回L中与e相等的元素的位序，无返回0

int i;

LinkList \*p=L->next;//p指向首结点

for(i=1;p!=NULL;i++){//遍历单链表，查找是否有相同元素

if(p->data==e)

return i;

else

p=p->next;

}

return ERROR;

}

status PriorElem(LinkList \*L,ElemType cur,ElemType \*pre\_e){//返回cur元素的前驱

int i;

LinkList \*p=L;//p指向头结点

LinkList \*q=p->next;//q指向首结点

if(LocateElem(L,cur)==1||LocateElem(L,cur)==ERROR)//遍历单链表，查找是否有相同元素并判断位置合法

return ERROR;

else{

for(;q!=NULL;){//遍历到相同元素的前一个结点

if(q->data==cur){

\*pre\_e=p->data;

return OK;

}

else{

p=q;

q=q->next;

}

}

return ERROR;

}

}

status NextElem(LinkList \*L,ElemType cur,ElemType \*next\_e){//返回cur元素的后继

int i;

LinkList \*p=L->next;//p指向首结点

if(LocateElem(L,cur)==ListLength(L)||LocateElem(L,cur)==ERROR)//遍历单链表，查找是否有相同元素并判断位置合法

return ERROR;

else{

for(;p!=NULL;){//遍历到相同元素的位置

if(p->data==cur){

\*next\_e=p->next->data;

return OK;

}

else

p=p->next;

}

return ERROR;

}

}

status ListInsert(LinkList \*L,int i,ElemType e){//在链式线性表中第i个元素之前插入新的元素e

int j;

LinkList \*t;//定义新的结点并分配空间

t=(LinkList\*)malloc(sizeof(LinkList));

t->data=e;//将新的元素值赋给新结点

LinkList \*p=L;

LinkList \*q=L->next;

if(i>ListLength(L)+1||i<1)//判断i是否合法

return ERROR;

else{

for(j=1;j<i;j++){//遍历到i-1位

p=q;

q=q->next;

}

t->next=p->next;

p->next=t;

}

return OK;

}

status ListDelete(LinkList \*L,int i,ElemType \*e){//在链式线性表中删除第i个元素，并用e返回其值

int j;

LinkList \*p=L;

LinkList \*q=L->next;

if(i>ListLength(L)||i<1)//判断i是否合法

return ERROR;

else{

for(j=1;j<i;j++){//遍历到i-1位

p=q;

q=q->next;

}

p->next=q->next;

free(q);

}

return OK;

}

status ListTrabverse(LinkList \*L){//遍历表

LinkList \*p=L->next;//p指向首结点

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

for(;p!=NULL;){//遍历单链表

printf("%d ",p->data);

p=p->next;

}

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

return OK;

}

status Wirte(LinkList \*L) {

FILE \*fp=NULL;

char filename[30];

LinkList \*p=L->next;

printf("input file name: ");

scanf("%s",filename);

if ((fp=fopen(filename,"wb"))==NULL){

printf("File open error\n ");

return FALSE;

}

//写文件的方法

for(;p!=NULL;){//遍历单链表,写入文件

fwrite(&(p->data),sizeof(ElemType),1,fp);

p=p->next;

}

fclose(fp);

return OK;

}

status Read(LinkList \*\*L) {

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

//读文件的方法

if ((fp=fopen(filename,"rb"))==NULL){

printf("File open error\n ");

return FALSE;

}

LinkList \*t=\*L;

LinkList \*p,\*q;

p=(LinkList\*)malloc(sizeof(LinkList));

q=(LinkList\*)malloc(sizeof(LinkList));//为指针分配空间

t->next=q;//q为首结点

while(fread(&(q->data),sizeof(ElemType),1,fp)){//从文件中读取数据到单链表中

p->next=q;

q->next=NULL;

p=p->next;

q=(LinkList\*)malloc(sizeof(LinkList));

}//这里从文件中逐个读取数据元素恢复链式表

free(q);//释放指针q

fclose(fp);

return OK;

}

# 附录C 基于二叉链表的二叉树实现的源程序

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

/\*----------预定义--------\*/

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define OVERFLOW -2

typedef int status; //数据元素类型定义

typedef char ElemType;

typedef char kElemType;

typedef struct Node{//定义二叉树的结构类型

ElemType data;

char key;

struct Node \*lchild;

struct Node \*rchild;

}Tree;

status CreatBiTree(Tree \*\*T,kElemType \*\*definition);//创建

status DestroyBiTree(Tree \*\*T);//销毁

status ClearBiTree(Tree \*\*T);//清空

status BiTreeEmpty(Tree \*T);//判断空表

status BiTreeDepth(Tree \*T);//求二叉树深度

Tree \*LocateNode(Tree \*T,kElemType e); //查找结点

status Assign(Tree \*T,kElemType e,ElemType value);//结点赋值

status parents(Tree \*T,kElemType e);

status GetSibling(Tree \*T,kElemType e);//获得兄弟结点

status InsertNode(Tree \*T,kElemType e,int LR,Tree \*c);//插入结点

status DeleteNode(Tree \*T,kElemType e);//删除关键字为e的结点

status Delecel(Tree \*T);//删除结点

status Delecer(Tree \*T);//删除结点

status PreOrderTraverse(Tree \*T);//前序遍历

status InOrderTraverse(Tree \*T);//中序遍历

status PostOrderTraverse(Tree \*T);//后序遍历

status LevelOrderTraverse(Tree \*T);//按层遍历

status Write(Tree \*T);

status TreeWrite(Tree \*T,FILE \*fp);

status Read(Tree \*\*T);

status TreeRead(Tree \*\*T,char \*definition);

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

int main(void){

Tree \*M[100];

Tree \*T;//声明所需变量

Tree \*p;

Tree \*c;

int o,i,LR;

ElemType value;

kElemType e;

char \*definition;

o=1;

while(o){

int op=1;

printf(" 请输入对哪个二叉树进行操作（1-99），输入0退出！\n");

scanf("%d",&o);//多个线性表的实现

if(o==0)

break;//0时退出

else {

if(o>99||o<0){

printf("选择失败！\n");

getchar();

}

else {

M[o]=NULL;

T=M[o];

while(op){

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

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

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

printf(" 1. CreatBiTree 9. InsertNode \n");

printf(" 2. DestroyBiTree 10. DeleteNode\n");

printf(" 3. ClearBiTree 11. PreOrderTraverse\n");

printf(" 4. BiTreeEmpty 12. InOrderTraverse\n");

printf(" 5. BiTreeDepth 13. PostOrderTraverse\n");

printf(" 6. LocateNode 14. LevelOrderTraverse\n");

printf(" 7. Assign 15. Write\n");

printf(" 8. GetSibling 16. Read\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~16]:\n");

scanf("%d",&op);

switch(op){

case 1:

printf("请输入带空子树的二叉树前序遍历：");

getchar();

scanf("%[^\n]",definition);

T=(Tree \*)malloc(sizeof(Tree));

T->rchild=NULL;

T->key=0;

if(CreatBiTree(&(T->lchild),&definition)==OK)

printf("二叉树创建成功！\n");

else

printf("二叉树创建失败！\n");

getchar();getchar();

break;

case 2:

if(DestroyBiTree(&T)==OK)

printf("二叉树销毁成功！\n");

else

printf("二叉树销毁失败！\n");

getchar();getchar();

break;

case 3:

if(T!=NULL){

ClearBiTree(&(T->lchild));

printf("清空二叉树！\n");

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 4:

if(T!=NULL){

if(BiTreeEmpty(T)==TRUE)

printf("二叉树为空！\n");

else

printf("二叉树不为空！\n");

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 5:

if(T!=NULL){

i=BiTreeDepth(T->lchild);

printf("二叉树的深度为%d！\n",i);

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 6:

if(T!=NULL){

printf("请输入查找的结点的关键字：\n");

getchar();

scanf("%c",&e);

p=LocateNode(T,e);

if(p!=NULL){

printf("该结点为：%c %c\n",p->key,p->data);

}

else

printf("该结点不存在\n");

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 7:

if(T!=NULL){

printf("请输入需要赋值的结点的关键字及所赋的值：\n");

getchar();

scanf("%c %c",&e,&value);

if(Assign(T,e,value)==OK)

printf("赋值成功\n");

else

printf("赋值失败\n");

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 8:

if(T!=NULL){

printf("请输入需要获得其兄弟结点的关键字：\n");

getchar();

scanf("%c",&e);

p=GetSibling(T->lchild,e);

if(p!=NULL)

printf("该元素的兄弟结点为%c %c",p->key,p->data);

else

printf("无兄弟结点\n");

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 9:

if(T!=NULL){

c=(Tree\*)malloc(sizeof(Tree));

printf("请输入要插入的位置的关键字、左(0)还是右(1)、要插入结点的关键字、数据：\n");

getchar();

scanf("%c %d %c %c",&e,&LR,&(c->key),&(c->data));

if(InsertNode(T,e,LR,c)==OK)

printf("插入成功！\n");

else

printf("插入失败！\n");

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 10:

if(T!=NULL){

printf("请输入需要删除的结点的关键字：\n");

getchar();

scanf("%c",&e);

if(DeleteNode(T,e)==OK)

printf("删除成功！\n");

else

printf("删除失败！\n");

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 11:

if(T!=NULL){

printf("前序遍历：\n");

PreOrderTraverse(T->lchild);

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 12:

if(T!=NULL){

printf("中序遍历：\n");

InOrderTraverse(T->lchild);

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 13:

if(T!=NULL){

printf("后序遍历：\n");

PostOrderTraverse(T->lchild);

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 14:

if(T!=NULL){

printf("按层遍历：\n");

LevelOrderTraverse(T->lchild);

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 15:

if(T!=NULL){

if(Wirte(T)==OK)

printf("写入成功！\n");

else

printf("写入失败！\n");

}

else

printf("二叉树不存在！\n");

getchar();getchar();

break;

case 16:

if(Read(&T)==OK)

printf("写入成功！\n");

else

printf("写入失败！\n");

getchar();getchar();

break;

case 0:

break;

}

}//end of switch

}

}

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

status CreatBiTree(Tree \*\*T,kElemType \*\*definition){//创建二叉树，definition为数组

if(\*\*definition=='#'){//结点不存在

(\*T)=NULL;

(\*definition)++;

return OK;

}

else{

if(!((\*T)=(Tree \*)malloc(sizeof(Tree))))

exit(OVERFLOW);

else{

(\*T)->key=\*\*definition;//赋关键字

(\*T)->data='x';

(\*definition)++;

CreatBiTree(&((\*T)->lchild),definition);

CreatBiTree(&((\*T)->rchild),definition);

}

}

return OK;

}

status DestroyBiTree(Tree \*\*T){//销毁二叉树T

Tree \*p=\*T;

ClearBiTree(T);

free(p);

(\*T)=NULL;

return OK;

}

status ClearBiTree(Tree \*\*T){//将二叉树清空

if((\*T)==NULL)

return OK;

else{

if((\*T)->lchild!=NULL)

ClearBiTree(&(\*T)->lchild);

if((\*T)->rchild!=NULL)

ClearBiTree(&(\*T)->rchild);

free((\*T));

(\*T)=NULL;

return OK;

}

}

status BiTreeEmpty(Tree \*T){//判断T是否为空

if(T->lchild==NULL)

return TRUE;

else

return FALSE;

}

status BiTreeDepth(Tree \*T){//计算T深度

int depth,depthleft,depthright;

if(T==NULL)

depth=0;

else{

depthleft=BiTreeDepth(T->lchild);

depthright=BiTreeDepth(T->rchild);

depth=1+(depthleft>depthright?depthleft:depthright);

}

return depth;

}

Tree\* LocateNode(Tree \*T,kElemType e){//查找结点

Tree \*p;

if(T==NULL)

return NULL;

else if(T->key==e){

return T;

}

else{

p=LocateNode(T->lchild,e);

if(p!=NULL)

return p;

else{

return LocateNode(T->rchild,e);

}

}

}

status Assign(Tree \*T,kElemType e,ElemType value){//找到关键字为e的结点并赋值

Tree \*p;

p=LocateNode(T,e);

if(p!=NULL){

p->data=value;

return OK;

}

else

return ERROR;

}

status parents(Tree \*T,kElemType e){//寻找关键字为e的父母结点

Tree \*p;

if(T==NULL)

return NULL;

else if((T->lchild)&&(T->lchild->key==e)){

return T;

}

else if ((T->rchild)&&(T->rchild->key==e)){

return T;

}

else{

p=parents(T->lchild,e);

if(p!=NULL)

return p;

else{

return parents(T->rchild,e);

}

}

}

status GetSibling(Tree \*T,kElemType e){//寻找关键字为e的兄弟结点

Tree \*p;

p=parents(T->lchild,e);

if(p){

if((p->lchild)&&(p->lchild->key==e)){

if(p->rchild)

return p->rchild;

else

return NULL;

}

else {

if((p->rchild)&&(p->rchild->key==e))

if(p->lchild)

return p->lchild;

else

return NULL;

}

}

}

status InsertNode(Tree \*T,kElemType e,int LR,Tree \*c){//在关键字为e的结点后插入结点

if(T!=NULL){

if(T->key==e){

if(LR==0){

c->lchild=T->lchild;

c->rchild=NULL;

T->lchild=c;

}

else{

c->rchild=T->rchild;

c->lchild=NULL;

T->rchild=c;

}

}

else{

InsertNode(T->lchild,e,LR,c);

InsertNode(T->rchild,e,LR,c);

}

return OK;

}

}

status DeleteNode(Tree \*T,kElemType e){//在删除关键字为e的结点

if((T->lchild)&&(T->lchild->key==e)){

Deletel(T);

return OK;

}

else if((T->rchild)&&(T->rchild->key==e)){

Deleter(T);

return OK;

}

else{

if(T->lchild)

if(DeleteNode(T->lchild,e)==OK)

return OK;

if(T->rchild)

if(DeleteNode(T->rchild,e)==OK)

return OK;

}

}

status Deletel(Tree \*T){//删除结点

Tree \*p;

Tree \*q;

Tree \*t;

q=T;

p=T->lchild;

t=p->lchild;

if(p->lchild==NULL){

if(p->rchild==NULL){//结点度为0

T->lchild=NULL;

return OK;

}

else{

p=p->rchild;//结点度为1，有右子树

q->lchild=p;

return OK;

}

}

else{

if(p->rchild==NULL){//结点度为1，有左子树

p=p->lchild;

q->lchild=p;

return OK;

}

else{//结点度为2

while(t->rchild!=NULL)

t=t->rchild;

t->rchild=p->rchild;

p=p->lchild;

q->lchild=p;

return OK;

}

}

}

status Deleter(Tree \*T){//删除结点

Tree \*p;

Tree \*q;

Tree \*t;

q=T;

p=T->rchild;

if(p->lchild==NULL){

if(p->rchild==NULL){//结点度为0

T->rchild=NULL;

return OK;

}

else{

p=p->rchild;//结点度为1，有右子树

q->rchild=p;

return OK;

}

}

else{

if(p->rchild==NULL){//结点度为1，有左子树

p=p->lchild;

q->rchild=p;

return OK;

}

else{//结点度为2

t=p->lchild;

while(t->rchild!=NULL)

t=t->rchild;

t->rchild=p->rchild;

p=p->lchild;

q->rchild=p;

return OK;

}

}

}

status PreOrderTraverse(Tree \*T){

if(T){

printf("%c %c\n",T->key,T->data);

PreOrderTraverse(T->lchild);

PreOrderTraverse(T->rchild);

}

return OK;

}

status InOrderTraverse(Tree \*T){

Tree \*st[100];

int top=0;

do{

while(T){

if(top==100)

exit(OVERFLOW);

st[++top]=T;

T=T->lchild;

}

if(top){

T=st[top--];

printf("%c %c\n",T->key,T->data);

T=T->rchild;

}

}while(top||T);

return OK;

}

status PostOrderTraverse(Tree \*T){

if(T){

PostOrderTraverse(T->lchild);

PostOrderTraverse(T->rchild);

printf("%c %c\n",T->key,T->data);

}

return OK;

}

status LevelOrderTraverse(Tree \*T){

Tree \*que[100];

Tree \*p;

int top=0;

int base=0;

if(T){

que[top]=T;

while(base<=top){

p=que[base];

printf("%c %c\n",p->key,p->data);

if(p->lchild)

que[++top]=p->lchild;

if(p->rchild)

que[++top]=p->rchild;

base++;

}

}

}

status Wirte(Tree \*T) {

FILE \*fp=NULL;

char c='#';

char filename[30];

printf("input file name: ");

scanf("%s",filename);

if ((fp=fopen(filename,"wb"))==NULL){

printf("File open error\n ");

return FALSE;

}

//写文件的方法

TreeWrite(T->lchild,fp);//遍历二叉树,写入文件

fclose(fp);

return OK;

}

status TreeWrite(Tree \*T,FILE \*fp){

char c=' ';

if(T==NULL){

fwrite(&c,sizeof(kElemType),1,fp);

return;

}

else {

fwrite(&(T->key),sizeof(kElemType),1,fp);

fwrite(&(T->data),sizeof(kElemType),1,fp);

TreeWrite(T->lchild,fp);

TreeWrite(T->rchild,fp);

}

}

int Arch;

status Read(Tree \*\*T) {

FILE \*fp;

char tree[100];

char single;

int i=0;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

//读文件的方法

if ((fp=fopen(filename,"rb"))==NULL){

printf("File open error\n ");

return FALSE;

}

(\*T)=(Tree \*)malloc(sizeof(Tree));//构造头结点

(\*T)->rchild=NULL;

(\*T)->key=0;

while(fread(&single,sizeof(ElemType),1,fp))

tree[i++]=single;

tree[i]='#';

Arch=0;

TreeRead(&((\*T)->lchild),&tree);//这里从文件中逐个读取数据元素恢复二叉树

fclose(fp);

return OK;

}

status TreeRead(Tree \*\*T,char \*definition){//第一种写法

if(definition[Arch]=='#')

return OK;

if(definition[Arch]==' '){//结点不存在

(\*T)=NULL;

Arch++;

}

else{

if(!((\*T)=(Tree \*)malloc(sizeof(Tree))))

exit(OVERFLOW);

else{

(\*T)->key=definition[Arch];//赋关键字

Arch++;

(\*T)->data=definition[Arch];

Arch++;

TreeRead(&((\*T)->lchild),definition);

TreeRead(&((\*T)->rchild),definition);

}

}

return OK;

}

/\*第二种写法

status TreeRead(Tree \*\*T,char \*definition){

if(\*definition=='#')

return OK;

if(\*definition==' '){//结点不存在

(\*T)=NULL;

definition++;

}

else{

if(!((\*T)=(Tree \*)malloc(sizeof(Tree))))

exit(OVERFLOW);

else{

(\*T)->key=\*definition;//赋关键字

definition++;

(\*T)->data=\*\*definition;

definition++;

TreeRead(&((\*T)->lchild),definition);

TreeRead(&((\*T)->rchild),definition);

}

}

return OK;

}

\*/

# 附录D 基于邻接表的图实现的源程序

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

/\*----------预定义--------\*/

#define TRUE 1

#define OK 1

#define ERROR -1

#define OVERFLOW -2

#define FALSE 0

#define max 20

#define MAX 100

typedef int status; //数据元素类型定义

typedef int ElemType;

int visited[100];

typedef struct ArcNode{

int adjvex;//该弧指向顶点的位序

struct ArcNode \*nextarc;//指向下一条弧

}ArcNode;//弧

typedef struct VNode{

int num;

int value;//值

char key;//顶点信息

ArcNode \*firstarc;//指向第一条依附该顶点的弧的指针

}VNode,AdjList[MAX];

typedef struct{

AdjList vertices;

int vexnum,arcnum;//图的当前顶点数和弧数

}Graph;

typedef struct QNode{//队列结点结构

ElemType data;

struct QNode \*next;

}QNode, \*QueuePtr;

typedef struct{//队列结构

QueuePtr front;

QueuePtr rear;

}LinkQueue;

status InitQueue(LinkQueue \*Q); //构造一个空队列

status QueueEmpty(LinkQueue Q); //判断一个队列是否为空

status EnQueue(LinkQueue \*Q, ElemType e); //入队列

status DeQueue(LinkQueue \*Q, ElemType \*e); //出队列

status CreatGraph(Graph \*G,char V[],char VR[][2]);//创建图

status DestroyGraph(Graph \*G);//销毁

status DestroyList(ArcNode \*arc);//销毁顶点的弧链表

status LocateVex(Graph G,char u);//由关键字查找查找顶点位序

status PutVex(Graph \*G,char u,int value);//顶点赋值

status FirstAdjVex(Graph G,char u);//获得第一邻接点

status NextAdjVex(Graph G,char u,int w);//获得下一邻接点

status InsertVex(Graph \*G,VNode vex);//插入顶点

status DeleteVex(Graph \*G,char u);//删除顶点

status InsertArc(Graph \*G,char u,char z);//插入弧

status DeleteArc(Graph \*G,char u,char z);//删除弧

status DFSTraverse(Graph G,char u);//深度优先遍历

char Locatekey(Graph G,int v);//由位序查找关键字

status Locatekey2(Graph G,int v);//由位序查找序号

status Locatehao(Graph G,char u);//由关键字查找序号

status BFSTraverse(Graph G, LinkQueue Q);//广度优先遍历

status Write(Graph G);//写入文件

status Read(Graph \*G);//读取文件

int main(void){

Graph M[100];

Graph G;//声明所需变量

char V[100];

char VR[100][2];

int o,i,w,p;

char u,z;

ElemType value;

VNode vex;

LinkQueue Q;

o=1;

while(o){

int op=1;

printf(" 请输入对哪个图进行操作（1-99），输入0退出！\n");

scanf("%d",&o);//多个线性表的实现

if(o==0)

break;//0时退出

else {

if(o>99||o<0){

printf("选择失败！\n");

getchar();

}

else {

G=M[o];

while(op){

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

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

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

printf(" 1. CreatGraph 8. DeleteVex \n");

printf(" 2. DestroyGraph 9. InsertArc\n");

printf(" 3. LocateVex 10. DeleteArc\n");

printf(" 4. PutVex 11. DFSTraverse\n");

printf(" 5. FirstAdjVex 12. BFSTraverse\n");

printf(" 6. NextAdjVex 13. Write\n");

printf(" 7. InsertVex 14. Read\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~16]:\n");

scanf("%d",&op);

switch(op){

case 1:

printf("请输入图的顶点数和弧数：");

getchar();

scanf("%d %d",&(G.vexnum),&(G.arcnum));

printf("请输入顶点：");

getchar();

for(i=0;i<(G.vexnum);i++)

scanf("%c",&V[i]);

printf("请输入弧：");

getchar();

for(i=0;i<(G.arcnum);i++){

scanf("%c",&VR[i][0]);

scanf("%c",&VR[i][1]);

getchar();

}

if(CreatGraph(&G,V,VR)==OK)

printf("图创建成功！\n");

else

printf("图创建失败！\n");

getchar();

break;

case 2:

if(DestroyGraph(&G)==OK)

printf("图销毁成功！\n");

else

printf("图销毁失败！\n");

getchar();getchar();

break;

case 3:

if(G.vexnum!=0){

printf("请输入需要查找顶点的关键字：\n");

getchar();

scanf("%c",&u);

p=LocateVex(G,u);

if(p!=ERROR){

printf("该顶点的位序为：%d\n",p);

}

else

printf("该顶点不存在\n");

}

else

printf("图不存在！\n");

getchar();getchar();

break;

case 4:

if(G.vexnum!=0){

printf("请输入需要赋值顶点的关键字与所赋值：\n");

getchar();

scanf("%c %d",&u,&value);

if(PutVex(&G,u,value)==OK)

printf("赋值成功！\n");

else

printf("赋值失败！\n");

}

else

printf("图不存在！\n");

getchar();getchar();

break;

case 5:

if(G.vexnum!=0){

printf("请输入需要获得第一邻接点的顶点的关键字：\n");

getchar();

scanf("%c",&u);

p=FirstAdjVex(G,u);

if(p==ERROR)

printf("顶点不存在！\n");

else

if(p==-2)

printf("该顶点没有邻接顶点！\n");

else

printf("该顶点的第一邻接顶点的位序为：%d\n",p);

}

else

printf("图不存在！\n");

getchar();getchar();

break;

case 6:

if(G.vexnum!=0){

printf("请输入需要获得下一邻接点的顶点的关键字及邻接点位序：\n");

getchar();

scanf("%c %d",&u,&w);

p=NextAdjVex(G,u,w);

if(p==ERROR)

printf("顶点不存在或顶点没有位序为%d的邻接点！\n",w);

else

if(p==NULL)

printf("位序为%d的顶点为最后一个邻接点！\n",w);

else

printf("下一邻接顶点的位序为：%d\n",p);

}

else

printf("图不存在！\n");

getchar();getchar();

break;

case 7:

if(G.vexnum!=0){

printf("请输入需要插入顶点的关键字及所赋值：\n");

getchar();

scanf("%c %d",&(vex.key),&(vex.value));

p=InsertVex(&G,vex);

if(p==OK)

printf("插入成功！\n");

else

printf("插入失败！\n");

}

else

printf("图不存在！\n");

getchar();getchar();

break;

case 8:

if(G.vexnum!=0){

printf("请输入需要删除顶点的关键字：\n");

getchar();

scanf("%c",&u);

p=DeleteVex(&G,u);

if(p==OK)

printf("删除成功！\n");

else

printf("删除失败！\n");

}

else

printf("图不存在！\n");

getchar();getchar();

break;

case 9:

if(G.vexnum!=0){

printf("请输入需要插入弧的顶点的关键字：\n");

getchar();

scanf("%c%c",&u,&z);

p=InsertArc(&G,u,z);

if(p==OK)

printf("插入成功！\n");

else

printf("插入失败！\n");

}

else

printf("图不存在！\n");

getchar();getchar();

break;

case 10:

if(G.vexnum!=0){

printf("请输入需要删除弧的顶点的关键字：\n");

getchar();

scanf("%c%c",&u,&z);

p=DeleteArc(&G,u,z);

if(p==OK)

printf("删除成功！\n");

else

printf("删除失败！\n");

}

else

printf("图不存在！\n");

getchar();getchar();

break;

case 11:

if(G.vexnum!=0){

printf("请输入遍历第一个顶点的关键字：\n");

getchar();

scanf("%c",&u);

DFSTraverse(G,u);

for(i=0;i<G.vexnum;i++){

if(visited[i]==0)

DFSTraverse(G,G.vertices[i].key);

}

for(i=0;i<G.vexnum;i++){

visited[i]=0;

}

}

else

printf("图不存在！\n");

getchar();getchar();

break;

case 12:

if(G.vexnum!=0){

InitQueue(&Q);

printf("广度遍历：\n");

BFSTraverse(G,Q);

for(i=0;i<G.vexnum;i++){

visited[i]=0;

}

}

else

printf("图不存在！\n");

getchar();getchar();

break;

case 13:

if(G.vexnum!=0){

if(Write(G)==OK)

printf("写入成功！\n");

else

printf("写入失败！\n");

}

else

printf("图不存在！\n");

getchar();getchar();

break;

case 14:

if(Read(&G)==OK)

printf("写入成功！\n");

else

printf("写入失败！\n");

getchar();getchar();

break;

case 0:

break;

}

}//end of switch

}

}

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

status CreatGraph(Graph \*G,char V[],char VR[][2]){

int i,j,k;

ArcNode \*p;

ArcNode \*q;

char u;

k=0;

int a[100][2]={0};

for(i=0;i<(G->vexnum);i++){//顶点构造

G->vertices[i].key=V[i];

G->vertices[i].num=i;

G->vertices[i].firstarc=NULL;

}

for(i=0;i<(G->arcnum);i++){//关键字转换成位序

u=VR[i][0];

a[i][0]=LocateVex(\*G,u);

u=VR[i][1];

a[i][1]=LocateVex(\*G,u);

}

/\*for(i=0;i<(G->vexnum);i++){//尾插法

q=G->vertices[i].firstarc;

for(j=0;j<(G->arcnum);j++){

if(a[j][0]==i){

p=(ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex=a[j][1];

p->nextarc=q;

q=p;

G->vertices[i].firstarc=p;

}

}

}\*///邻接表创建

for(j=0;j<(G->arcnum);j++){

k=a[j][0];

q=G->vertices[k].firstarc;

if(q==NULL){

p=(ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex=a[j][1];

p->nextarc=NULL;

G->vertices[k].firstarc=p;

}

else{

while((q->nextarc)!=NULL){

q=q->nextarc;

}

p=(ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex=a[j][1];

p->nextarc=NULL;

q->nextarc=p;

}

}

return OK;

}

status DestroyGraph(Graph \*G){

int i;

for(i=0;i<G->vexnum;i++){//对于所有的顶点

if(G->vertices[i].firstarc){

if(DestoryList(G->vertices[i].firstarc)==OK);//销毁顶点的弧链表

else

return ERROR;

}

}

free(G->vertices);

G->vexnum=0;//顶点数为0

G->arcnum=0;//边或者弧数为0

return OK;

}

status DestoryList(ArcNode \*arc){

ArcNode \*p=arc;

ArcNode \*q;

if(arc){

for(;arc!=NULL;){//遍历单链表

q=arc;

arc=arc->nextarc;

free(q);//释放线性表空间

}

arc=NULL;

return OK;

}

else

return ERROR;

}

status LocateVex(Graph G,char u){

int i;

for(i=0;i<(G.vexnum);i++){//遍历查找

if(G.vertices[i].key==u){

return G.vertices[i].num;

}

}

return ERROR;

}

status Locatehao(Graph G,char u){//由关键字查找序号

int i;

for(i=0;i<(G.vexnum);i++){

if(G.vertices[i].key==u){

return i;

}

}

return ERROR;

}

status PutVex(Graph \*G,char u,int value){//顶点赋值

int p;

p=LocateVex(\*G,u);//查找顶点位序

if(p!=ERROR){

G->vertices[p].value=value;

return OK;

}

else

return ERROR;

}

status FirstAdjVex(Graph G,char u){

int p;

p=LocateVex(G,u);

if(p!=ERROR){//顶点v存在

if(G.vertices[p].firstarc)//顶点有邻接点

return G.vertices[p].firstarc->adjvex;//返回第一邻接顶点的位序

else

return -2;//顶点无邻接点

}

else

return ERROR;//顶点v不存在

}

status NextAdjVex(Graph G,char u,int w){

int i;

int t;

ArcNode \*p;

t=Locatehao(G,u);

if(t!=ERROR){//顶点v存在

p=G.vertices[t].firstarc;//p为第一邻接点

for(i=0;p!=NULL;i++){//查找位序为w的邻接点

if(p->adjvex==w){

if(p->nextarc)//p不为最后一个邻接点

return p->nextarc->adjvex;

else

return NULL;

}

p=p->nextarc;

}

return ERROR;//没有位序为w的邻接点

}

else

return ERROR;

}

status InsertVex(Graph \*G,VNode vex){//插入顶点

G->vertices[G->vexnum].key=vex.key;

G->vertices[G->vexnum].value=vex.value;

G->vertices[G->vexnum].firstarc=NULL;

G->vertices[G->vexnum].num=G->vexnum;

G->vexnum=G->vexnum+1;//顶点数增加

return OK;

}

status DeleteVex(Graph \*G,char u){//删除顶点

int i,v;

ArcNode \*p;

ArcNode \*q;

v=LocateVex(\*G,u);

if(v!=ERROR){

for(i=0;i<G->vexnum;i++){//删除含有该顶点的弧

p=G->vertices[i].firstarc;

if((p!=NULL)&&(p->nextarc!=NULL)){

if(p->adjvex==v){

G->vertices[i].firstarc=p->nextarc;

}

else{

p=G->vertices[i].firstarc;

q=p->nextarc;

while(q!=NULL){

if(q->adjvex!=v){

p=p->nextarc;

q=q->nextarc;

}

else{

p->nextarc=q->nextarc;

q=NULL;

}

}

}

}

else

if((p!=NULL)&&(p->nextarc==NULL))

if(p->adjvex==v)

p=NULL;

}

DestoryList(G->vertices[v].firstarc);

for(i=v;i<G->vexnum;i++){//顶点v后面的顶点前移

G->vertices[i]=G->vertices[i+1];

}

G->vexnum--;

return OK;

}

else

return ERROR;

}

status InsertArc(Graph \*G,char u,char z){//插入弧

int x,y;

ArcNode \*p;

x=Locatehao(\*G,u);

y=LocateVex(\*G,z);

p=(ArcNode\*)malloc(sizeof(ArcNode));

if((x!=ERROR)&&(y!=ERROR)){//顶点存在

p->adjvex=y;

if(G->vertices[x].firstarc){

p->nextarc=G->vertices[x].firstarc->nextarc;

G->vertices[x].firstarc=p;

}

else{

p->nextarc=G->vertices[x].firstarc;

G->vertices[x].firstarc=p;

}

}

G->arcnum++;//弧数量增加

return OK;

}

status DeleteArc(Graph \*G,char u,char z){//删除弧

int x,y;

ArcNode \*p;

ArcNode \*q;

x=Locatehao(\*G,u);

y=LocateVex(\*G,z);

if((x!=ERROR)&&(y!=ERROR)){//判断顶点是否存在

p=G->vertices[x].firstarc;

q=p->nextarc;

if((p!=NULL)&&(p->nextarc!=NULL)){//避免空指针的使用

if(p->adjvex==y){

G->vertices[x].firstarc=p->nextarc;

p=NULL;

}

else{

p=G->vertices[x].firstarc;

q=p->nextarc;

while(q!=NULL){

if(q->adjvex!=y){

p=p->nextarc;

q=q->nextarc;

}

else{

p->nextarc=q->nextarc;

q=NULL;

}

}

}

}

else

if((p!=NULL)&&(p->nextarc=NULL))

if(p->adjvex==y)

p=NULL;

}

G->arcnum--;

return OK;

}

status DFSTraverse(Graph G,char u){//深度优先遍历

int v;

char z;

ArcNode \*p;

v=Locatehao(G,u);

if(visited[v]==0){

printf("%c %d\n", G.vertices[v].key,G.vertices[v].value);

visited[v]=1;//标记已访问

p=G.vertices[v].firstarc;

while(p!=NULL){

if(visited[p->adjvex]==0){

z=Locatekey(G,p->adjvex);

DFSTraverse(G,z);

}

p=p->nextarc;

}

}

}

char Locatekey(Graph G,int v){//由位序找关键字

int i;

for(i=0;i<(G.vexnum);i++){//顶点构造

if(G.vertices[i].num==v){

return G.vertices[i].key;

}

}

return ERROR;

}

status Locatekey2(Graph G,int v){//由位序找序号

int i;

for(i=0;i<(G.vexnum);i++){//顶点构造

if(G.vertices[i].num==v){

return i;

}

}

return ERROR;

}

status BFSTraverse(Graph G, LinkQueue Q){//广度优先遍历

int v,u,i,num;

char k;

ArcNode \*temp;

for(v=0;v<G.vexnum;v++)

visited[v]=0;

for(v=0;v<G.vexnum;v++){

if(!visited[v]){

visited[v]=1;

printf("%c %d\n", G.vertices[v].key,G.vertices[v].value);

num=LocateVex(G,G.vertices[v].key);//位序

EnQueue(&Q,num);

while(!QueueEmpty(Q)){

DeQueue(&Q,&u);

i=Locatekey2(G,u);//由位序找序号

for(temp=G.vertices[i].firstarc;temp!=NULL;temp=temp->nextarc){

k=Locatekey(G,temp->adjvex);

num=LocateVex(G,k);

i=Locatehao(G,k);

if (visited[i]!=1) {

visited[i]=1;

printf("%c %d\n", G.vertices[i].key,G.vertices[i].value);

EnQueue(&Q,num);

}

}

}

}

}

return OK;

}

status InitQueue(LinkQueue \*Q){//构造一空的队列Q

Q->front=Q->rear=(QueuePtr)malloc(sizeof(QNode));

if (!Q->front)

exit(OVERFLOW);

Q->front->next=NULL;

return OK;

}

status QueueEmpty(LinkQueue Q){//若Q为空,返回TRUE,不空则返回FALSE

if(Q.front==Q.rear)

return TRUE;

else

return FALSE;

}

status EnQueue(LinkQueue \*Q,ElemType e){//插入元素e为Q的新的队尾元素

QueuePtr p=(QueuePtr)malloc(sizeof(QNode));

if (!p)//存储分配失败

exit(OVERFLOW);

p->data=e;

p->next=NULL;

Q->rear->next=p;

Q->rear=p;

return OK;

}

status DeQueue(LinkQueue \*Q,ElemType \*e){//若队列不空,删除Q的队头元素,用e返回其值,并返回OK,否则返回ERROR

QueuePtr p;

if (Q->front==Q->rear)

return ERROR;

p=Q->front->next;

\*e=p->data;

Q->front->next=p->next;

if(Q->rear==p)

Q->rear=Q->front;

free(p);

return OK;

}

status Write(Graph G){//写入文件

FILE \*fp=NULL;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

if ((fp=fopen(filename,"wb"))==NULL){

printf("File open error\n ");

return FALSE;

}

//写文件的方法

int i;

int j;

ArcNode \*p;

fwrite(&G.vexnum,sizeof(int),1,fp);//写入图的信息

fwrite(&G.arcnum,sizeof(int),1,fp);

for(i=0;i<G.vexnum;i++){//写入顶点信息

fwrite(&G.vertices[i].key,sizeof(char),1,fp);

fwrite(&G.vertices[i].value,sizeof(int),1,fp);

fwrite(&G.vertices[i].num,sizeof(int),1,fp);

}

for(i=0;i<G.vexnum;i++){

p=G.vertices[i].firstarc;

while(p!=NULL){

fwrite(&G.vertices[i].num,sizeof(int),1,fp);

j=Locatekey2(G,p->adjvex);

fwrite(&G.vertices[j].num,sizeof(int),1,fp);

p=p->nextarc;

}

}

fclose(fp);

return OK;

}

status Read(Graph \*G){//读取文件

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s",filename);

if ((fp=fopen(filename,"rb"))==NULL){

printf("File open error\n ");

return FALSE;

}

//读文件的方法

int i=0;

int a=-1;

int j;

ArcNode \*p;

ArcNode \*q;

fread(&G->vexnum,sizeof(int),1,fp);

fread(&G->arcnum,sizeof(int),1,fp);

for(i=0;i<G->vexnum;i++){//构造顶点

fread(&G->vertices[i].key,sizeof(char),1,fp);

fread(&G->vertices[i].value,sizeof(int),1,fp);

fread(&G->vertices[i].num,sizeof(int),1,fp);

}

while(fread(&a,sizeof(int),1,fp)){//将弧读取并构造邻接表，尾插法

j=Locatekey2(\*G,a);

q=G->vertices[j].firstarc;

if(q==NULL){

p=(ArcNode\*)malloc(sizeof(ArcNode));

fread(&p->adjvex,sizeof(int),1,fp);

p->nextarc=NULL;

G->vertices[j].firstarc=p;

}

else{

while((q->nextarc)!=NULL){

q=q->nextarc;

}

p=(ArcNode\*)malloc(sizeof(ArcNode));

fread(&p->adjvex,sizeof(int),1,fp);

p->nextarc=NULL;

q->nextarc=p;

}

}

fclose(fp);

return OK;

}