

**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201804**

**学 号： U201814604**

**姓 名： 黄俊淇**

**指导教师： 袁凌**

**报告日期： 2019年 月 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](#_Toc24554156)

[1.1 问题描述 2](#_Toc24554157)

[1.1.1 顺序表的基本概念 2](#_Toc24554158)

[1.1.2 函数定义 3](#_Toc24554159)

[1.2 系统设计 4](#_Toc24554160)

[1.3 系统实现 5](#_Toc24554161)

[1.3.1编程环境 5](#_Toc24554162)

[1.3.2 头文件及预定义常量说明 5](#_Toc24554163)

[1.3.3 函数实现 6](#_Toc24554164)

[1.4 系统测试 10](#_Toc24554165)

[1.4.1系统演示操作 10](#_Toc24554166)

[1.4.2测试计划 11](#_Toc24554167)

[1.4.3系统演示 12](#_Toc24554168)

[1.4.4异常数据 19](#_Toc24554169)

[1.5 实验小结 20](#_Toc24554170)

[2 基于链式存储结构的线性表实现 21](#_Toc24554171)

[2.1 问题描述 21](#_Toc24554172)

[2.1.1链表的基本概念 21](#_Toc24554173)

[2.1.2函数定义 21](#_Toc24554174)

[2.2 系统设计 22](#_Toc24554175)

[2.3 系统实现 24](#_Toc24554176)

[2.3.1 编程环境 24](#_Toc24554177)

[2.3.2 头文件及预定义常量说明 24](#_Toc24554178)

[2.3.3 函数实现 24](#_Toc24554179)

[2.4 系统测试 29](#_Toc24554180)

[2.4.1系统演示操作 29](#_Toc24554181)

[2.4.2测试计划 30](#_Toc24554182)

[2.4.3系统演示 31](#_Toc24554183)

[2.5 实验小结 35](#_Toc24554184)

[3 基于二叉链表的二叉树实现 36](#_Toc24554185)

[3.1 问题描述 36](#_Toc24554186)

[3.2 系统设计 36](#_Toc24554187)

[3.3 系统实现 36](#_Toc24554188)

[3.4 系统测试 36](#_Toc24554189)

[3.5 实验小结 36](#_Toc24554190)

[4 基于邻接表的图实现 37](#_Toc24554191)

[4.1 问题描述 37](#_Toc24554192)

[4.2 系统设计 37](#_Toc24554193)

[4.3 系统实现 37](#_Toc24554194)

[4.4 系统测试 37](#_Toc24554195)

[4.5 实验小结 37](#_Toc24554196)

[参考文献 38](#_Toc24554197)

[附录A 基于顺序存储结构线性表实现的源程序 39](#_Toc24554198)

[附录B 基于链式存储结构线性表实现的源程序 47](#_Toc24554199)

[附录C 基于二叉链表二叉树实现的源程序 56](#_Toc24554200)

[附录D 基于邻接表图实现的源程序 58](#_Toc24554201)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

采用顺序表的物理结构，构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示。定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等基本运算对应的函数，并给出适当的操作提示显示，可选择以文件的形式进行存储和加载，即将生成的线性表存入到相应的文件中，也可以从文件中获取线性表进行操作。通过输入编号该实现对多表进行操作。

## 顺序表的基本概念

线性表是最常用且最简单的一种数据结构，即n个数据元素的有限序列。线性表中元素的个数n定义为线性表的长度，n=0时成为空表。在非空表中的每个数据元素都有一个确定的位置，如a1是第一个数据元素，an是最后一个数据元素，ai是第i个数据元素。线性表的存储结构为线性结构即为顺序表。

## 函数定义

依据最小完备性和常用性相结合的原则，以函数形式定义了包括线性表的初

始化表、加载表、保存表、销毁表、清空表、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、遍历表 14 个基本运算，

要求分别定义函数来实现上述功能，具体功能运算如下：

（1）初始化表：函数名称是Initalist(L)；初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。

（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。

（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FALSE。

（5）求表长：函数名称是ListLength(L)；初始条件是线性表已存在；操作结果是返回L中数据元素的个数。

（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，1≤i≤ListLength(L)；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,compare())；初始条件是线性表已存在；操作结果是返回L中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。

（8）获得前驱：函数名称是PriorElem(L,cur\_e,pre\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre\_e无定义。

（9）获得后继：函数名称是NextElem(L,cur\_e,next\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败，next\_e无定义。

（10）插入元素：函数名称是ListInsert(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

（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，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

（12）遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。

（13）加载文件：函数名称是Readfile（L），初始条件是线性表L已存在；操作结果是将文件中的元素加载到线性表中。

（14）加载文件：函数名称是Writefile（L），初始条件是线性表L已存在；操作结果是将线性表中的元素保存到文件中。

（15）切换顺序表：函数名称是Changelist（int \*n），初始条件是线性表L已存在；操作结果是将顺序表切换到指定编号的顺序表。

## 1.2 系统设计

**1.2.1 数据结构定义**

线性表的数据结构定义如下：

typedef struct { //顺序表（顺序结构）的定义

ElemType \*elem; //定义整型指针，为存储空间基址

int length; //线性表的长度

int listsize; //当前分配的存储容量(以 sizeof(ElemType)为单位)

}SqList;

要实现同时对多个线性表管理，只需要定义一个结构数组即可。

**1.2.2 演示系统**

菜单共设置15个函数，由switch语句实现。用户通过菜单来实现相应操作：当输入在1～15之间，进入循环，完成相应操作，由break语句进入下一个循环；当用户输入0，则退出本系统。在第一次进入循环while时首先会询问用户对哪个线性表进行操作，直至退出演示系统或用户选择切换线性表之前一直对指定线性表进行操作。演示系统结构如图1-1。

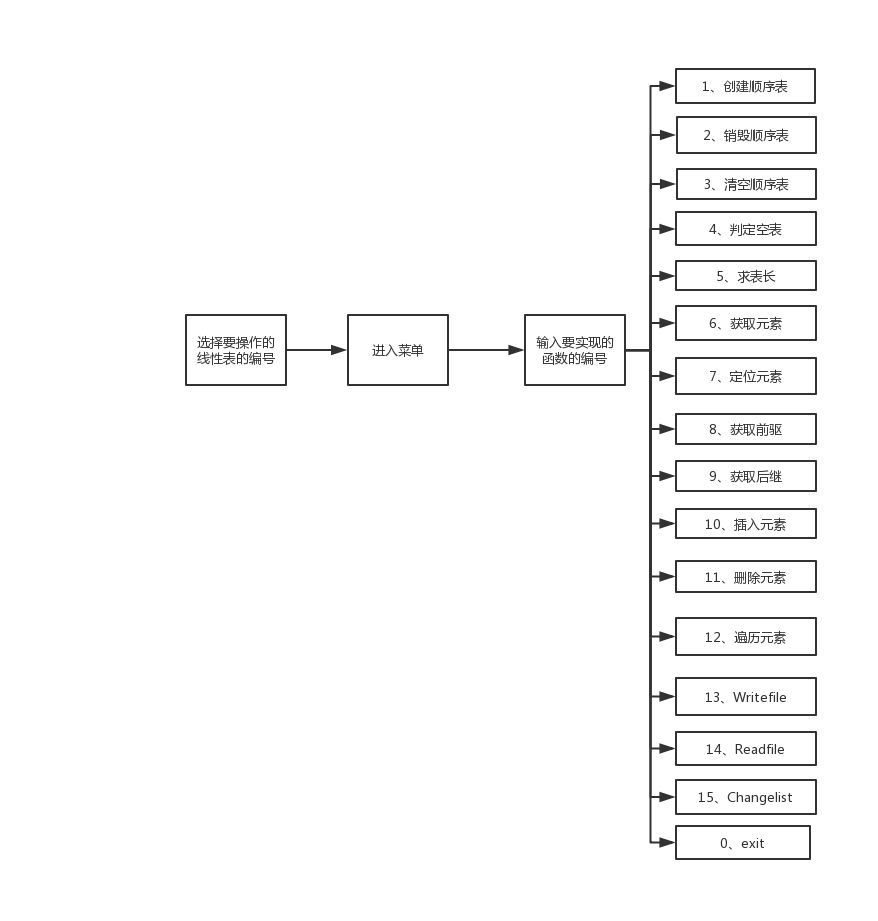


图1-1演示系统

## 1.3 系统实现

## 1.3.1编程环境

编译软件为Xcode10.3

## 1.3.2 头文件及预定义常量说明

1.头文件

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

2.预定义常量

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

3.类型表达式

typedef int status;

typedef int ElemType;

## 1.3.3 函数实现

（1）创建顺序表：函数参数为指向结构的指针，结构类型为Sqlist。用malloc函数给指定顺序表分配LISTSIZE大小的空间，将首地址返回给L->elem；由于顺序表为空，则L->listlength=0。

（2）销毁顺序表：函数参数为指向结构的指针，结构类型为Sqlist。用free函数将指定顺序表已分配的空间释放。若成功销毁，将L->listlength=0，返回OK，若首地址不存在即顺序表未创建，则返回ERROR。

（3）清空顺序表：函数参数为指向结构的指针，结构类型为Sqlist。若顺序表未创建则返回ERROR；若顺序表已创建，则L->listlength=0。

（4）判断是否为空表：函数参数为指向结构的指针，结构类型为Sqlist。若顺序表未创建则返回ERROR；若顺序表已创建，如果L->listlength=0，则返回OK，否则返回ERROR。

（5）求表长：函数参数为指向结构的指针，结构类型为Sqlist。若顺序表未创建则返回ERROR；若顺序表已创建，则返回L->listlength。

（6）获取元素：函数参数为指向结构的指针，结构类型为Sqlist以及指向类型为Elemtype的元素的指针。若输入不合法即小于1或者大于L->listlength，则返回OVERFLOW；否则将指定元素赋值给指向类型为Elemtype的元素的指针指向的元素。

（7）定位元素：函数参数为指向结构的指针，结构类型为Sqlist以及指向类型为Elemtype的元素的指针。流程图如图1-2。

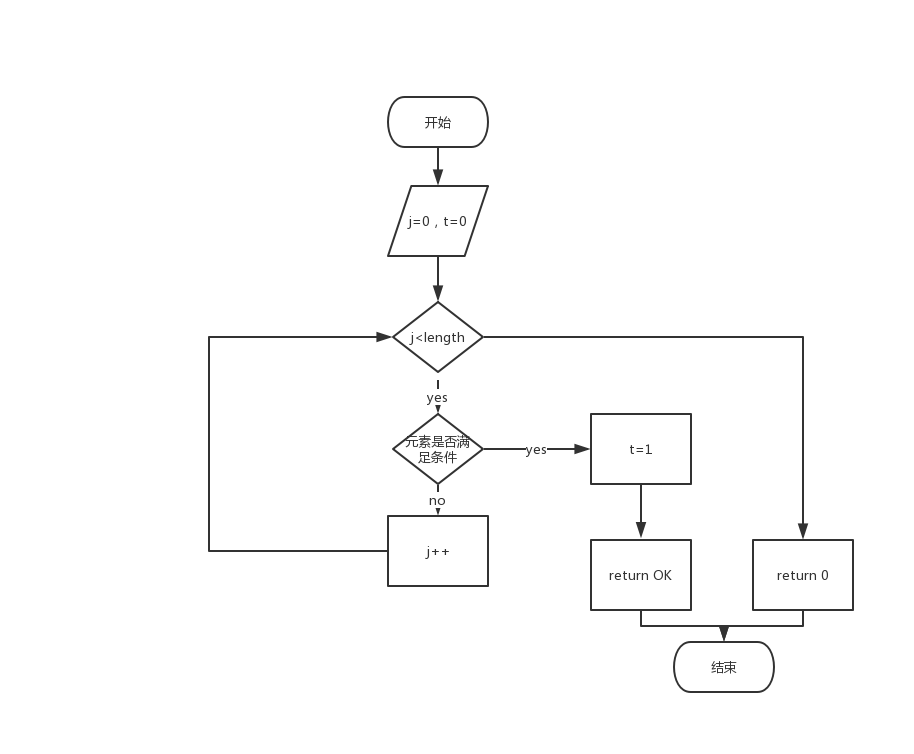


图1-2定位元素

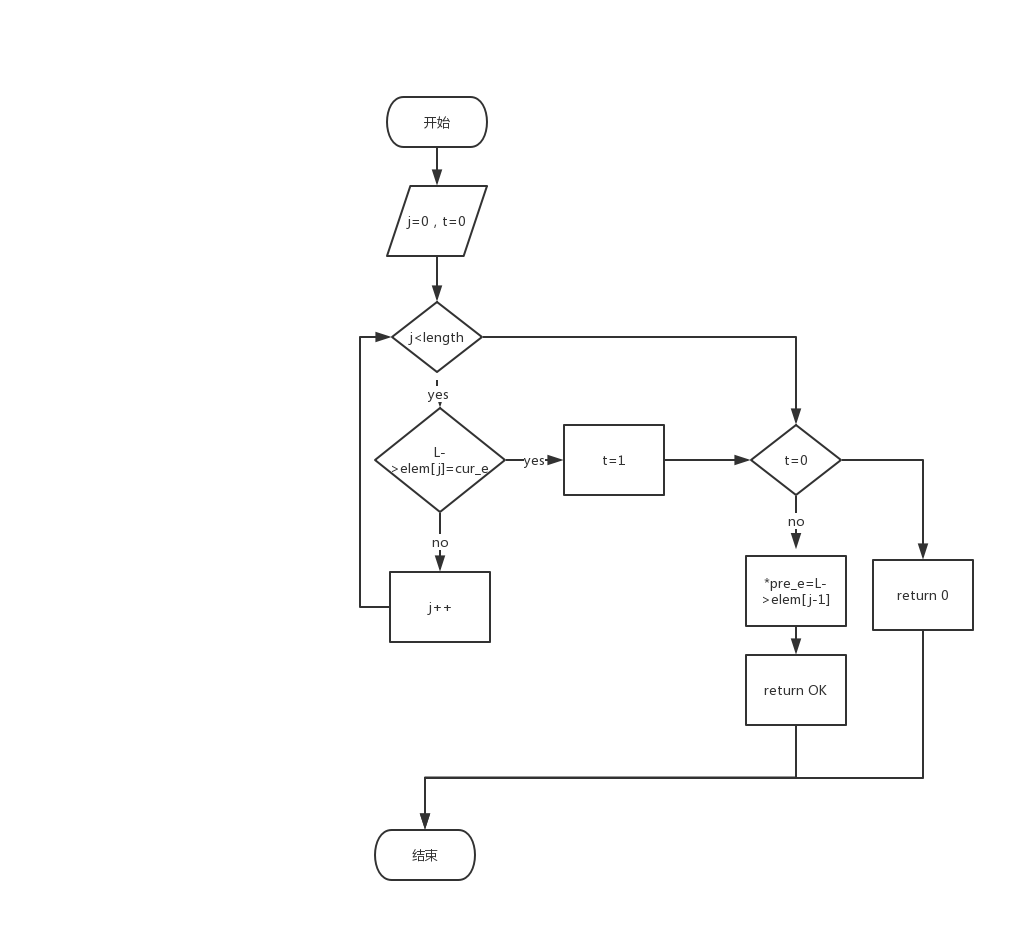


图1-3查找前驱元素

（8）查找前驱元素：函数参数为指向结构的指针，结构类型为Sqlist，指向类型为Elemtype的元素的指针，以及需要查找的元素。流程图如图1-3。

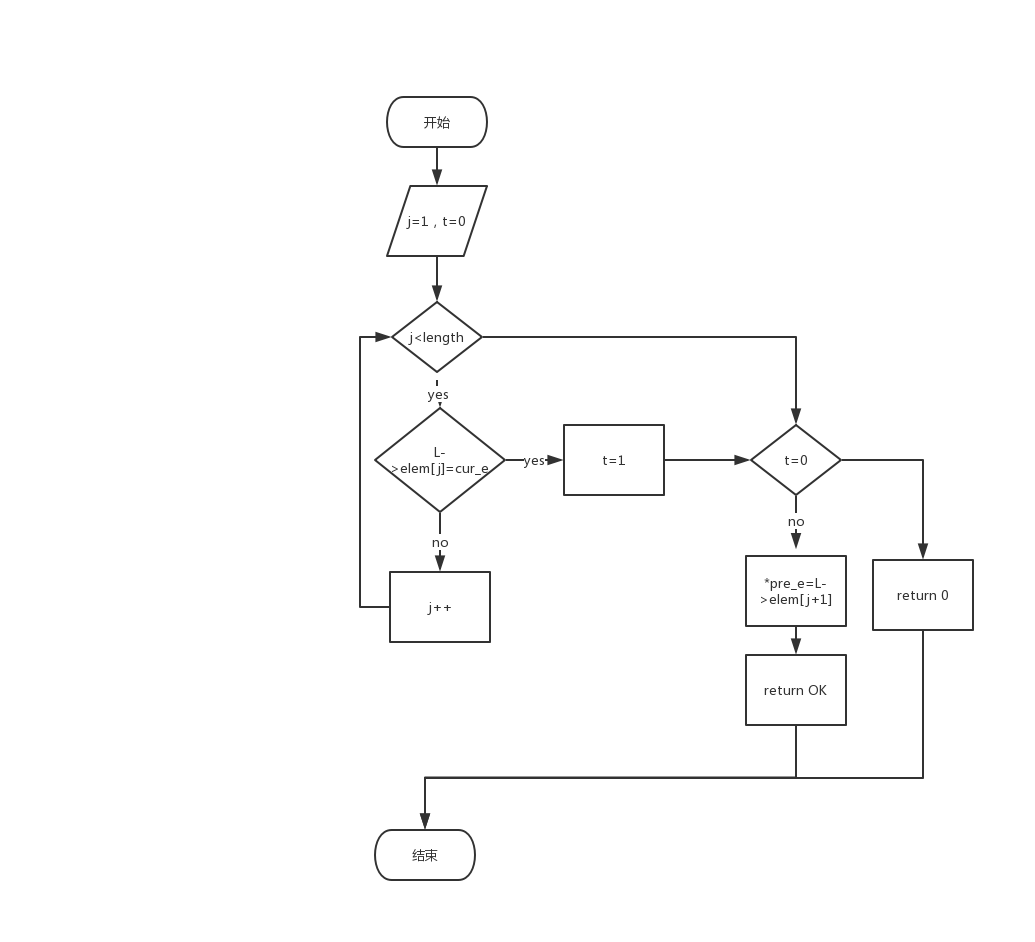


图1-4查找后继元素

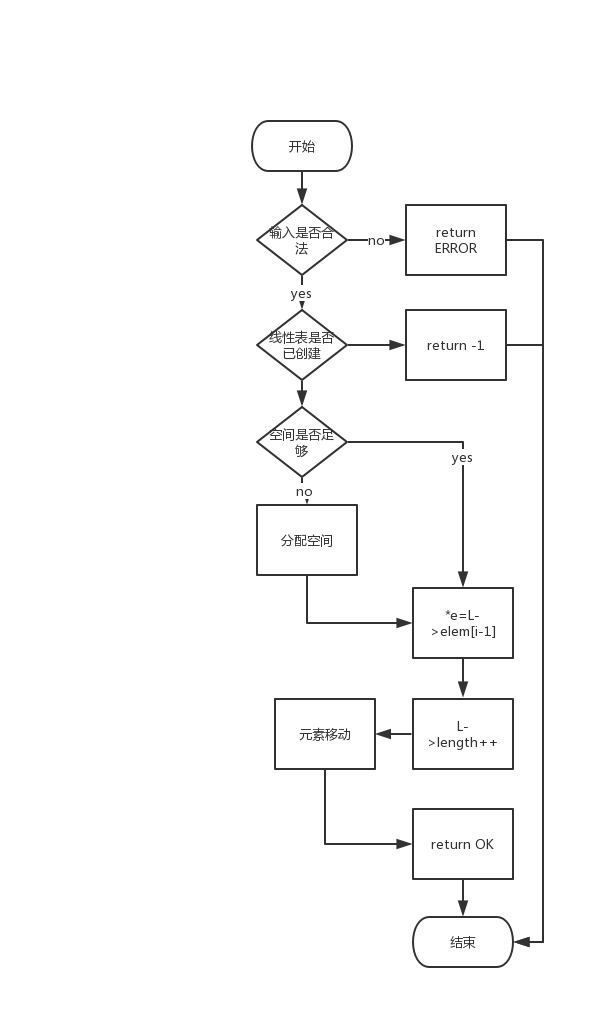


图1-5插入元素

（9）查找后继元素：函数参数为指向结构的指针，结构类型为Sqlist，指向类型为Elemtype的元素的指针，以及需要查找的元素。流程图如图1-4。

（10）插入元素：函数参数为指向结构的指针，结构类型为Sqlist，需要插入的位置，以及需要插入的元素。流程图如图1-5。

（11）删除元素：函数参数为指向结构的指针，结构类型为Sqlist，需要删除的位置，以及需要删除的元素。流程图如图1-6。

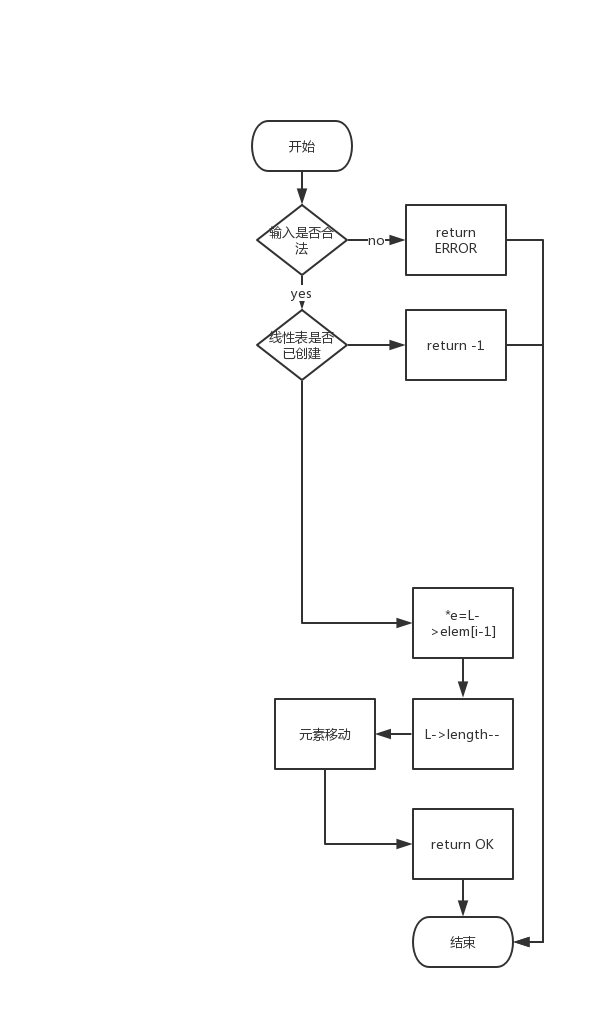


图1-6删除元素

（12）遍历：函数参数为指向结构的指针，结构类型为Sqlist。利用循环一次输出表中元素。

（13）写入文件：函数参数为指向结构的指针，结构类型为Sqlist。如果路径错误，则返回ERROR；否则，将表中元素通过循环依次写入文件，当碰到表尾则退出循环，返回OK，关闭文件。

（14）读取文件：函数参数为指向结构的指针，结构类型为Sqlist。如果路径错误，则返回ERROR；否则，将文件中元素通过循环依次读取存入表中，当碰到文件尾则退出循环，返回OK，关闭文件。

（15）切换顺序表：函数参数为int型指针。如果输入不合法，则返回ERROR；否则将用户输入的数字赋值到指针指向的元素，通过改变编号来切换顺序表，返回OK。

**1.4 系统测试**

## 1.4.1系统演示操作

（1）.系统一开始会显示菜单提示用户输入选择对1-99号线性表哪一个进行操作。如图1-7所示。



图1-7选择线性表

（2）.选择对线性表1进行操作，进入菜单演示界面，如图1-8所示。

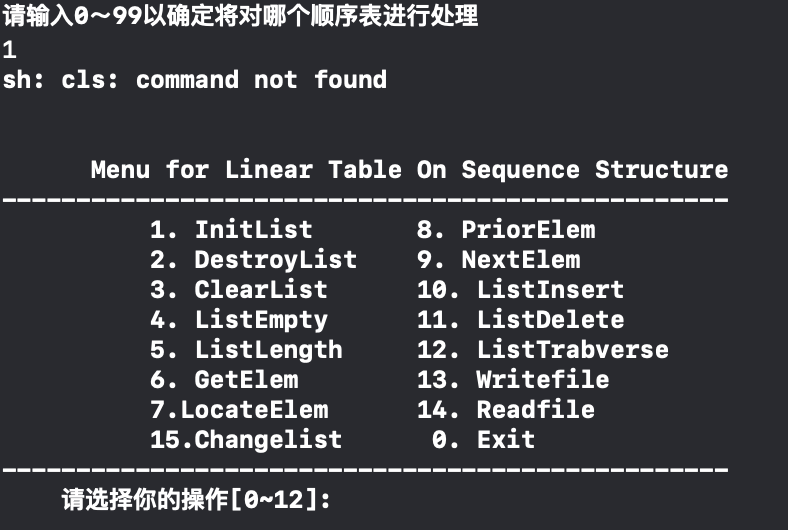


图1-8选择编号1线性表

（3）.输入0，退出演示系统，结束操作，如图1-9所示。

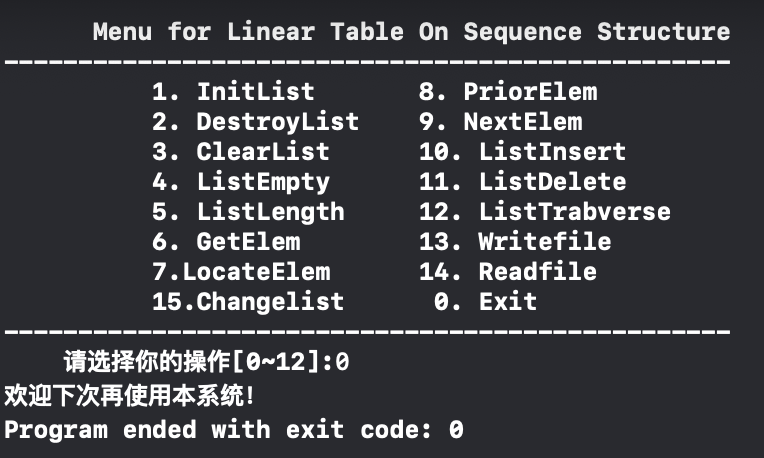


图1-9退出系统

## 1.4.2测试计划

表一 测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能及序号 | 输入要管理的线性表序号 | 输入函数的参数（具体元素） | 预计输出 | 此时线性表的状态 |
| 1.InitList | 1 | L的地址 | 线性表初始化成功 | 分配了连续的物理存储空间，表长度为0，表尺寸为空间大小 |
| 2.DestroyList | 1 | L的地址 | 销毁成功 | 线性表连续物理空间被释放， |
| 3.ClearList | 1 | L的地址 | 清空成功 | 线性表的物理空间保留，但表长置为0. |
| 10.ListInsert(多次调用) | 1 | L的地址，插入位置1，2，3，4，5，插入元素1，2，3，4，5 | 插入成功 | 线性表，序列为：1，2，3，4，5 |
| 4.ListEmpty | 1 | L的地址 | 该表不是空表 | 线性表，序列为：1，2，3，4，5 |
| 5.ListLength | 1 | L的地址 | 该表长为5 | 线性表，序列为：1，2，3，4，5 |
| 6.GetElem | 1 | 输入数据元素的位序为3 | 输出线性表的第三个元素为3 | 线性表，序列为：1，2，3，4，5 |
| 8.PriorElem | 1 | 输入数据元素的值为4 | 找到4的前驱元素是3 | 线性表，序列为：1，2，3，4，5 |
| 9.NextElem | 1 | 输入数据元素的值为2 | 找到2的后继元素是3 | 线性表，序列为：1，2，3，4，5v |
| 11.ListDelete | 1 | 输入数据元素的置为3 | 删除成功，删除的元素是3 | 线性表，序列为1245 |
| 7.LocateElem | 1 | 输入数据元素的值为4 | 您要查找的元素是第3元素 | 线性表，序列为1245 |
| 13.Writefile | 1 | 输入保存到的文件名为1.txt | 写入成功 | 文件显示1，2，4，5 |
| 14.Readfile | 1 | 输入要加载的文件名为2.txt | 读取成功 | 顺序表序列为1，2，3，4，5，6，7，8，9，10 |
| 15.Changelist | 1 | 输入编号2 | 切换成功 | 不存在线性表空间 |

**1.4.3系统演示**

（1）.输入对线性表1进行操作，进入菜单演示界面，执行功能1，初始化线性表，测试结果如图1-10所示。

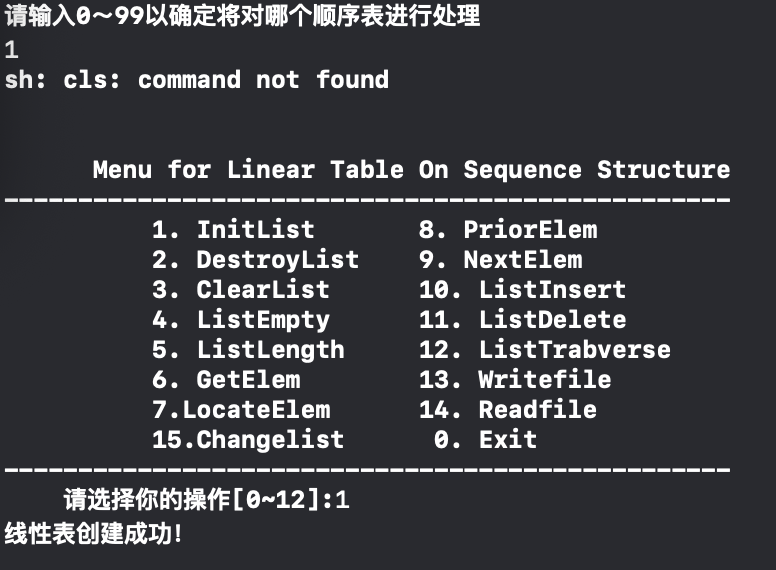


图1-10创建顺序表

（2）.执行功能2，销毁线性表，测试结果如图1-11所示。

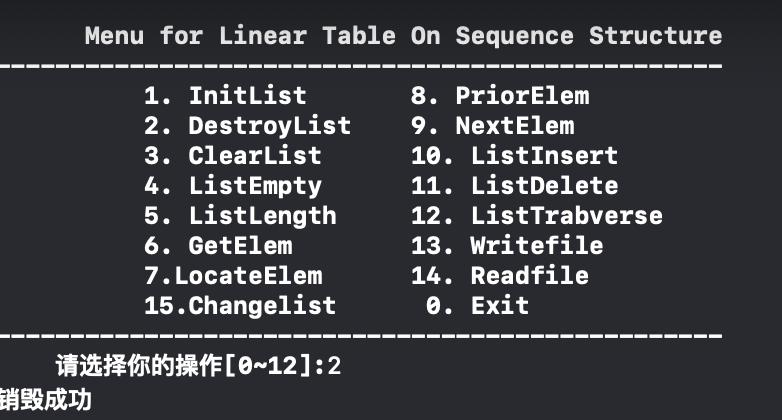


图1-11销毁顺序表

（3）.执行功能10，往线性表中添加数据元素1,2,3,4,5，测试结果如图1-12所示。

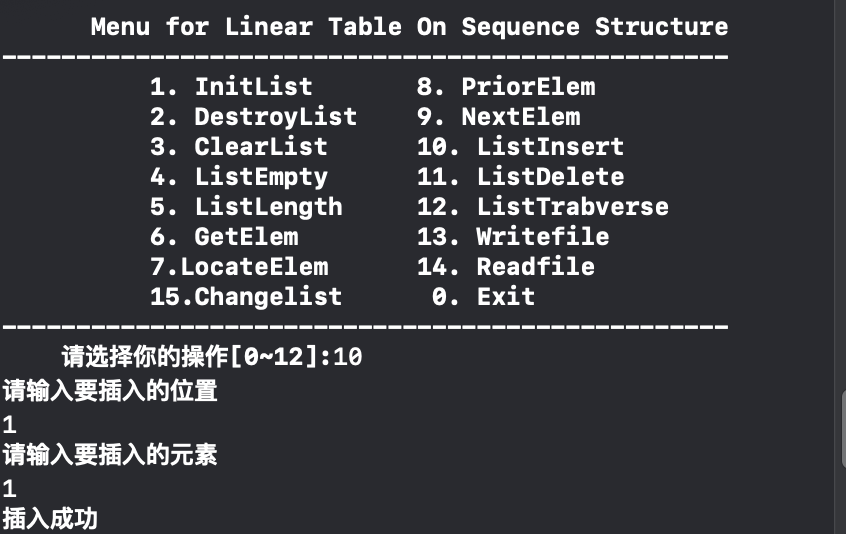


图1-12插入元素

（4）.执行功能4，判断线性表是否为空，测试结果如图1-13所示。



图1-13判定空表

（5）.执行功能5，求线性表的长度，测试结果如图1-14所示。

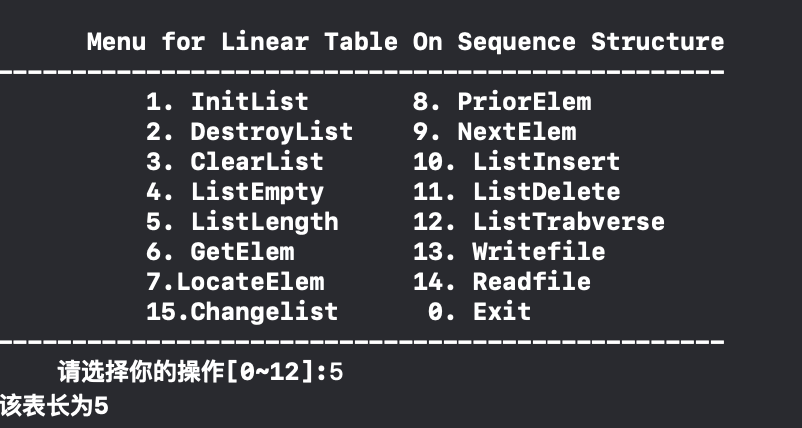


图1-14求表长

（6）.执行功能6，查找第三个数据元素的数值，测试结果如图1-15所示。

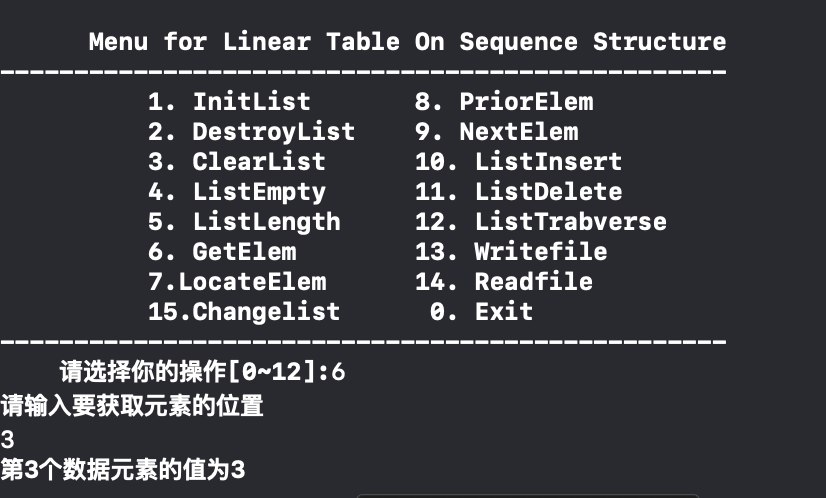


图1-15获取元素

（7）.执行功能8，确定值为4的数据元素前驱数据元素，测试结果如图1-16所示。



图1-16查找前驱

（8）.执行功能9，确定值为2的数据元素的后继数据元素，测试结果如图1-17所示。

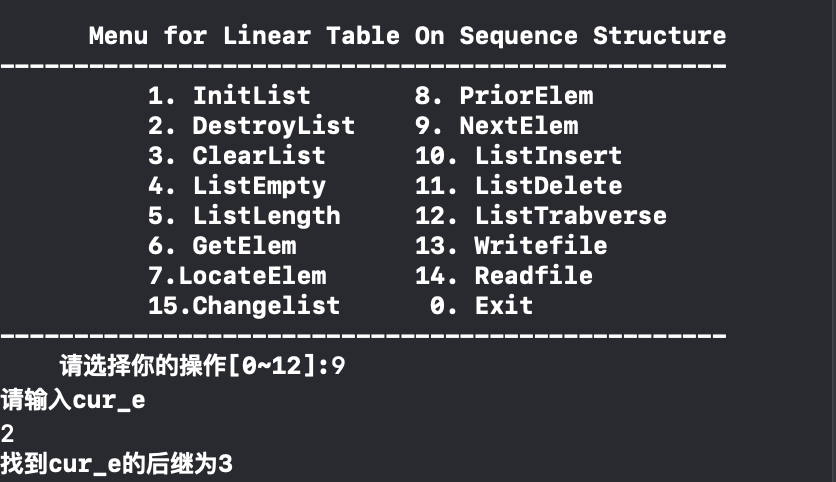


图1-17查找后继

（9）.执行功能11，删除第三个数据元素，测试结果如图1-18所示。

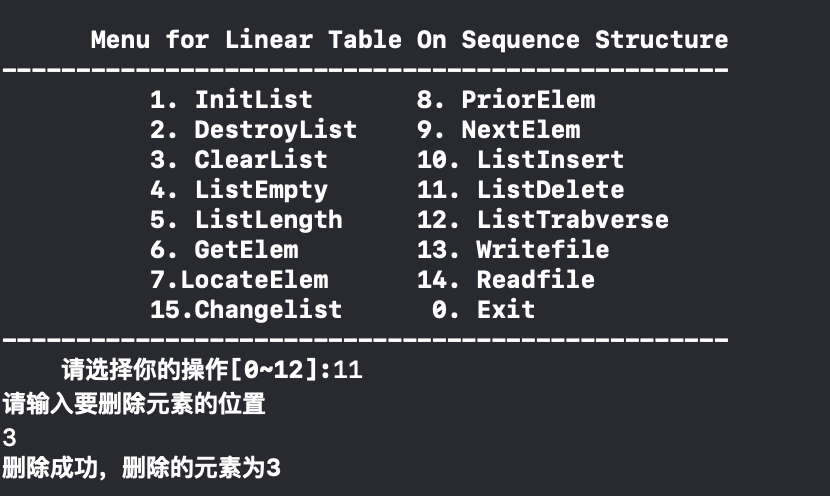


图1-18删除元素

（10）.执行功能13，保存线性表中的数据元素值到文件名为1.txt的文件中，测试结果如图1-19所示。

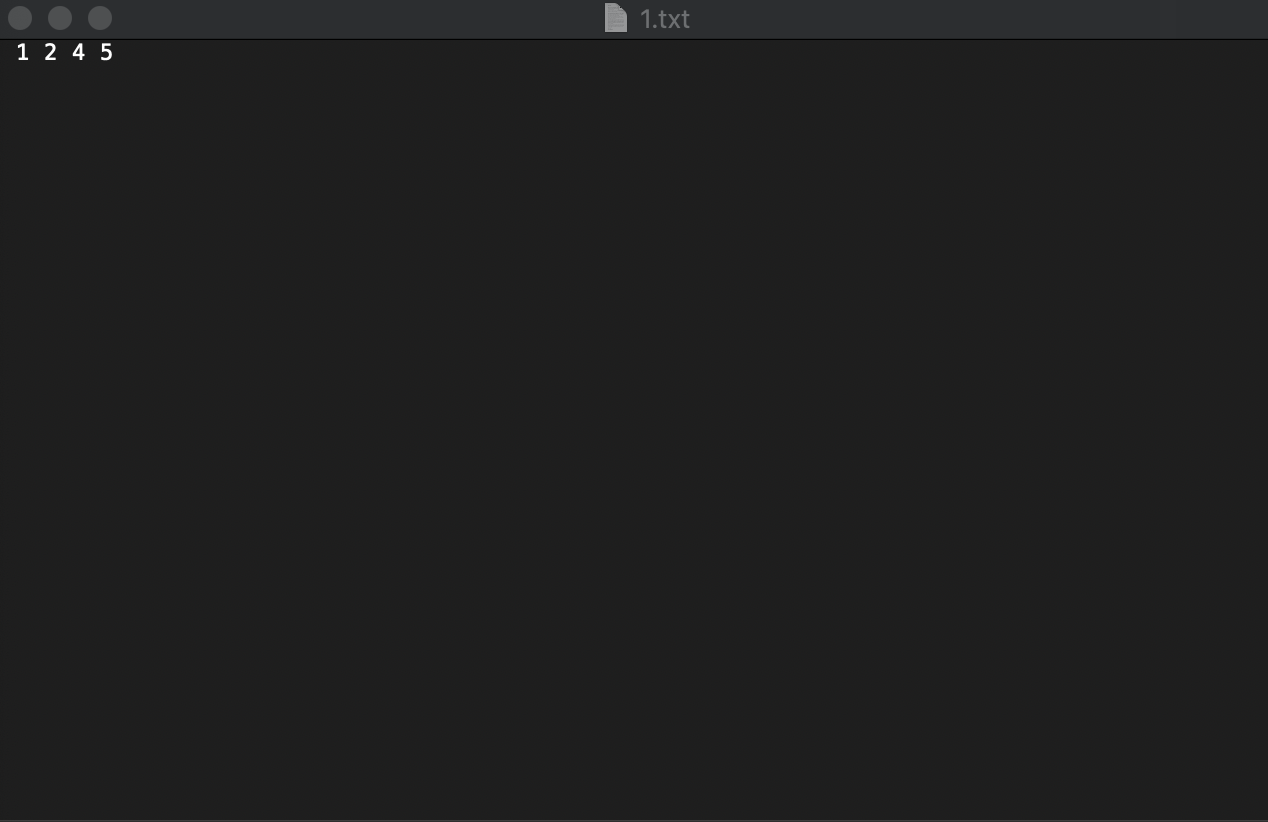


图1-19写入文件

（11）.执行功能3，14，将2.txt中元素读取到顺序表，测试结果如图1-20所示。



图1-20清空顺序表

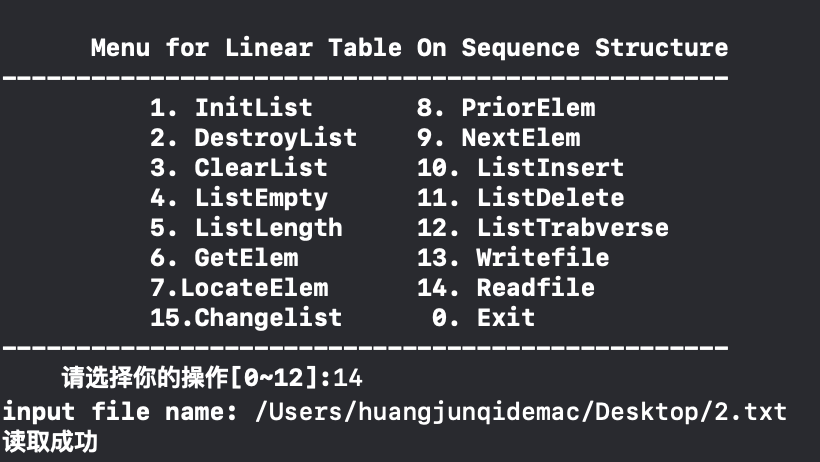


图1-21读取文件

（12）.执行功能12，遍历并输出线性表中的元素，应该输出1，2，3，4，5，6，7，8，9，10，测试结果如图1-21所示。

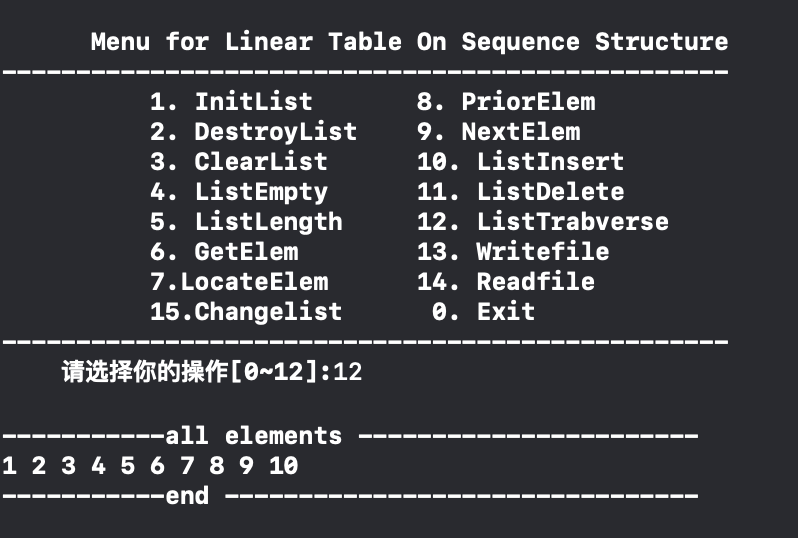


图1-22遍历元素

（13）.执行功能7，查找元素6，测试结果如图1-23所示。

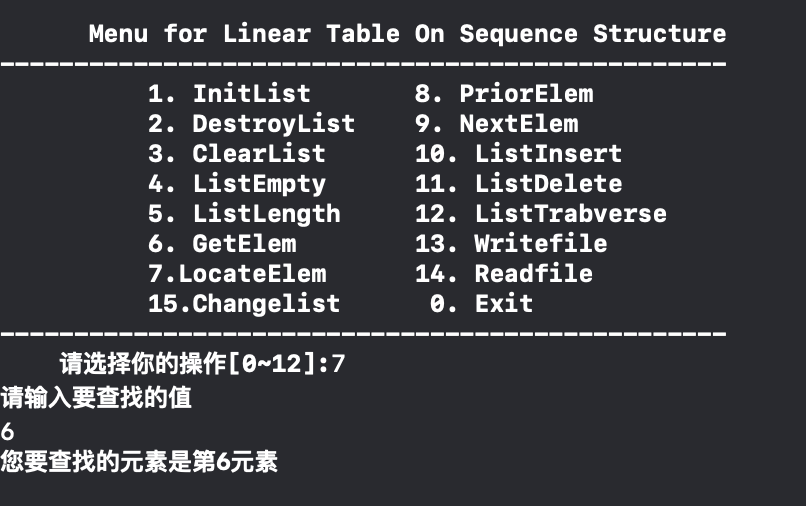


图1-23查找元素

（14）.执行功能15，输入编号2，测试结果如图1-24所示。

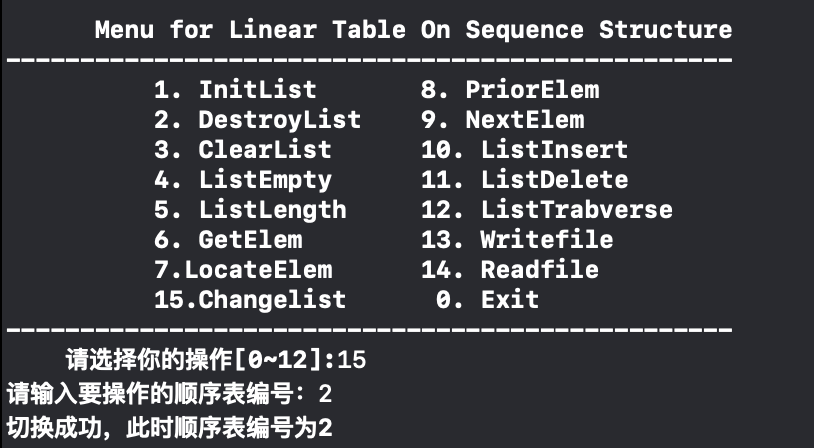


图1-24切换顺序表

**1.4.4异常数据**

（1）查找前驱异常情况：初始线性表中已有三个元素，依次是1，2，3，若查找第一个元素的前驱元素，测试结果如图1-25.

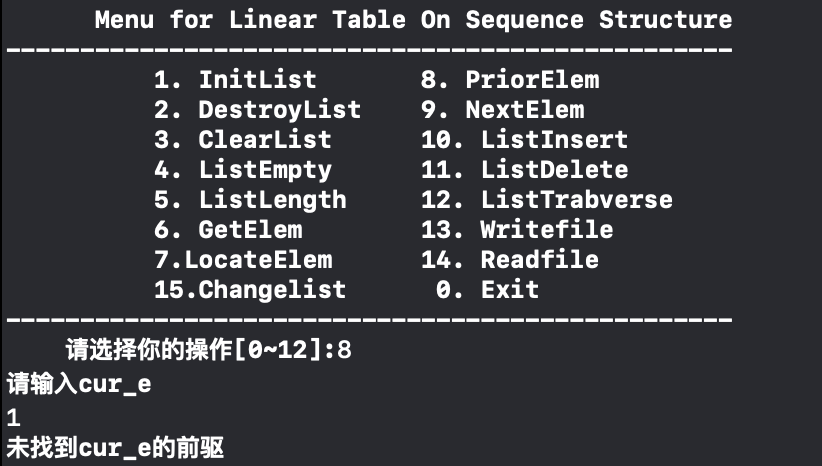


图1-25查找前驱异常

（2）查找后继异常情况：初始线性表中已有三个元素，依次是1，2，3，若查找最后一个元素的后继元素，测试结果如图1-26.

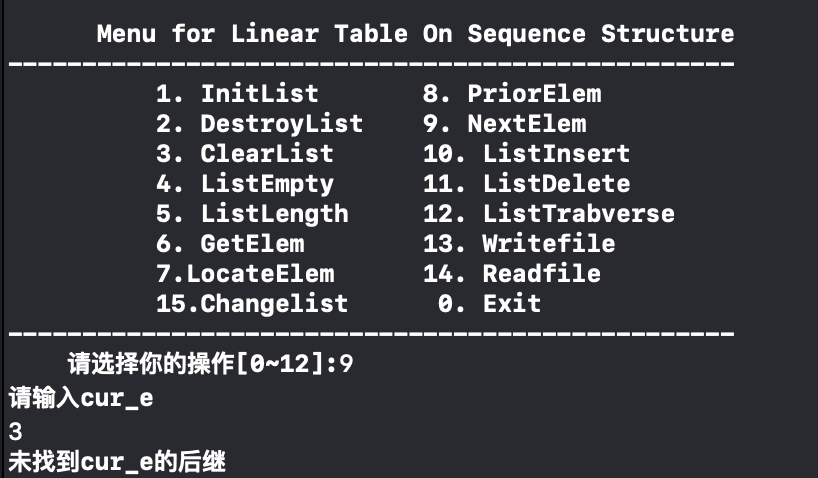


图1-26查找后继异常

（3）输入不合法：以插入元素为例，若输入不合法，测试结果如图1-27.

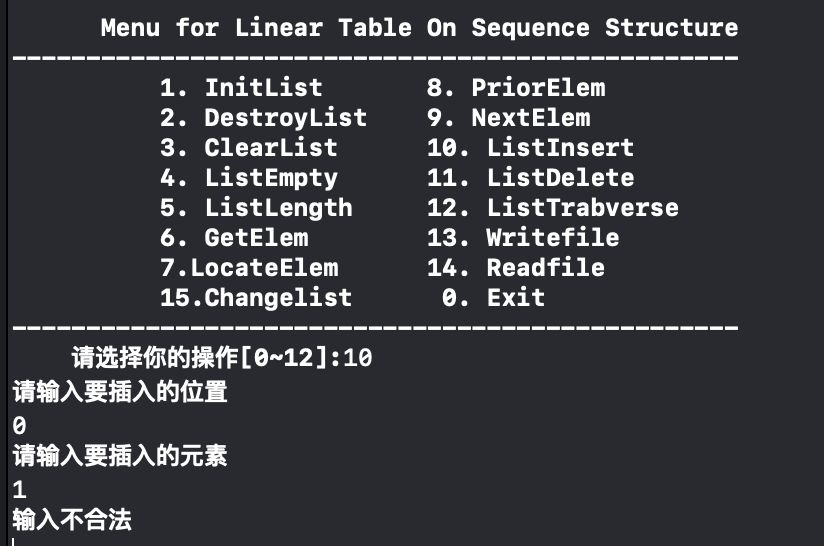


图1-27输入不合法

## 1.5 实验小结

本次实验中，还是存在了考虑情况不全的问题，当切换线性表之后，执行插入操作却显示插入成功，而此时还未创建线性表。原因是在插入函数里未设置这样情况的出口。改进方法是在函数里增加if语句来判断此时线性表的地址是否为空。如果为空则返回ERROR，否则返回OK。在main函数里，当函数返回值返回OK才显示插入成功。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

采用链式的物理结构，构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示。定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等基本运算对应的函数，并给出适当的操作提示显示，可选择以文件的形式进行存储和加载，即将生成的线性表存入到相应的文件中，也可以从文件中获取线性表进行操作。通过输入编号该实现对多表进行操作。

## 2.1.1链表的基本概念

线性表是最常用且最简单的一种数据结构，即n个数据元素的有限序列。线性表中元素的个数n定义为线性表的长度，n=0时成为空表。在非空表中的每个数据元素都有一个确定的位置，如a1是第一个数据元素，an是最后一个数据元素，ai是第i个数据元素。线性表的存储结构为链式结构即为链表。

## 2.1.2函数定义

依据最小完备性和常用性相结合的原则，以函数形式定义了包括线性表的初

始化表、加载表、保存表、销毁表、清空表、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、遍历表 14 个基本运算，

要求分别定义函数来实现上述功能，具体功能运算如下：

（1）初始化表：函数名称是Initalist(L)；初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。

（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。

（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FALSE。

（5）求表长：函数名称是ListLength(L)；初始条件是线性表已存在；操作结果是返回L中数据元素的个数。

（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，1≤i≤ListLength(L)；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,compare())；初始条件是线性表已存在；操作结果是返回L中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。

（8）获得前驱：函数名称是PriorElem(L,cur\_e,pre\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre\_e无定义。

（9）获得后继：函数名称是NextElem(L,cur\_e,next\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败，next\_e无定义。

（10）插入元素：函数名称是ListInsert(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

（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，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

（12）遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。

（13）加载文件：函数名称是Readfile（L），初始条件是线性表L已存在；操作结果是将文件中的元素加载到线性表中。

（14）加载文件：函数名称是Writefile（L），初始条件是线性表L已存在；操作结果是将线性表中的元素保存到文件中。

（15）切换顺序表：函数名称是Changelist（int \*n），初始条件是线性表L已存在；操作结果是将线性表切换到指定编号的线性表。

## 2.2 系统设计

**2.2.1 数据结构定义**

线性表的数据结构定义如下：

typedef struct Node{

ElemType data;

struct Node \*next;

}Node,\*Lnode;

要实现同时对多个线性表管理，只需要定义一个结构指针数组，用来存放不同链表的头指针即可。

**2.2.2 演示系统**

菜单共设置15个函数，由switch语句实现。用户通过菜单来实现相应操作：当输入在1～15之间，进入循环，完成相应操作，由break语句进入下一个循环；当用户输入0，则退出本系统。在第一次进入循环while时首先会询问用户对哪个线性表进行操作，直至退出演示系统或用户选择切换线性表之前一直对指定线性表进行操作。演示系统结构如图2-1。

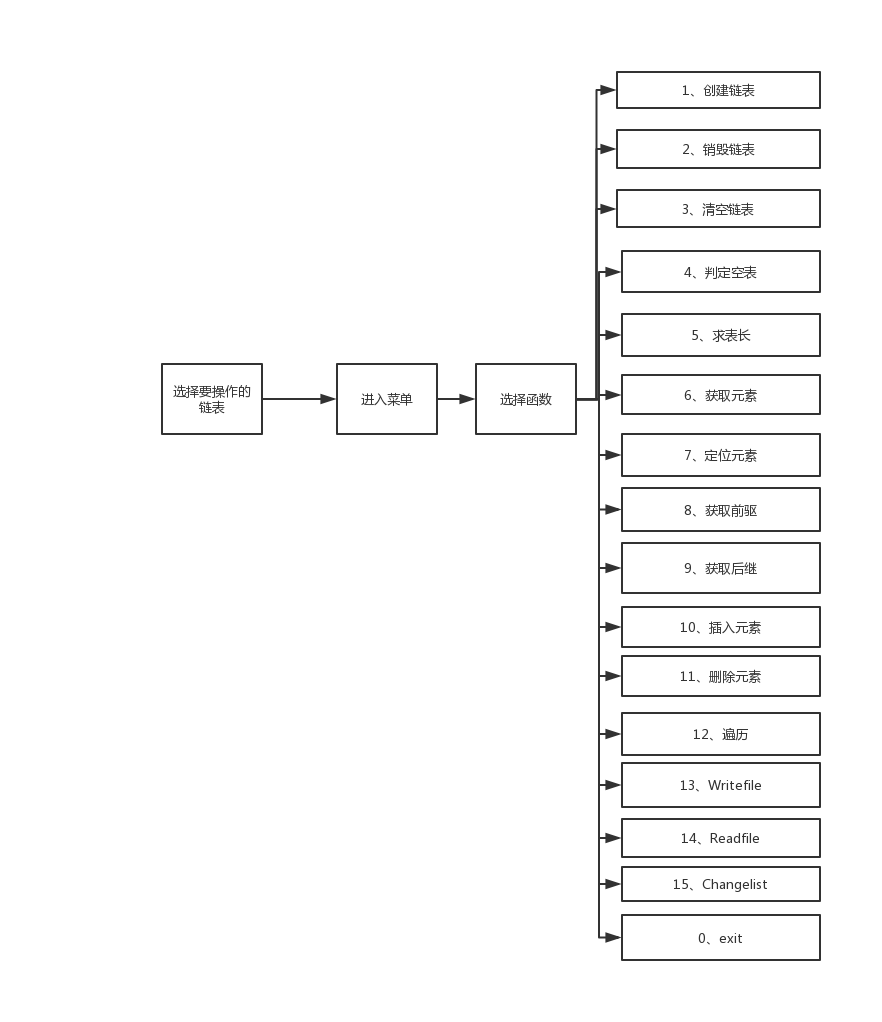


图2-1演示系统

## 2.3 系统实现

## 2.3.1 编程环境

编译软件为Xcode10.3

## 2.3.2 头文件及预定义常量说明

1.头文件

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

2.预定义常量

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

3.类型表达式

typedef int status;

typedef int ElemType;

## 2.3.3 函数实现

（1）创建链表：函数参数为指向结构的二级指针，类型为Lnode \*。用malloc函数为表头申请一个大小为Node的空间，表头的数据域存放链表的大小，将表头的指针域暂时设为Null，返回OK。

（2）销毁链表：函数参数为指向结构的二级指针，类型为Lnode \*。若链表未创建，则返回ERROR；若链表已创建，则利用循环用free函数释放链表，返回OK。

（3）清空链表：函数参数为指向结构的二级指针，类型为Lnode \*。若链表未创建，则返回ERROR；利用循环，从表头下一个元素开始依次释放，直到表尾，令（\*L）->data=0，返回OK。

（4）判断是否为空表：函数参数为指向结构的二级指针，类型为Lnode \*。若链表未创建，则返回-1；若链表已创建，若（\*L）->data=0，返回TRUE，否则返回FALSE。

（5）求表长：函数参数为指向结构的二级指针，类型为Lnode \*。若链表未创建，则返回-1；若链表已创建，返回（\*L）->data。

（6）获取元素：函数参数为指向结构的二级指针，类型为Lnode \*以及指向类型为Elemtype的元素的指针。若输入不合法即小于1或者大于（\*L）->data，则返回OVERFLOW；否则将指定元素赋值给指向类型为Elemtype的元素的指针指向的元素。

（7）定位元素：函数参数为指向结构的二级指针，类型为Lnode \*以及指向类型为Elemtype的元素的指针。若链表未创建，则返回-1；定位元素流程图如图2-2。

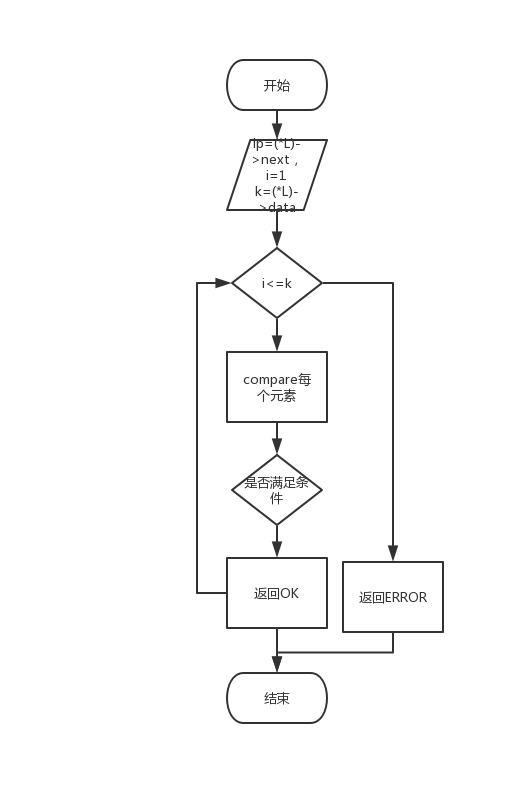


图2-2定位元素

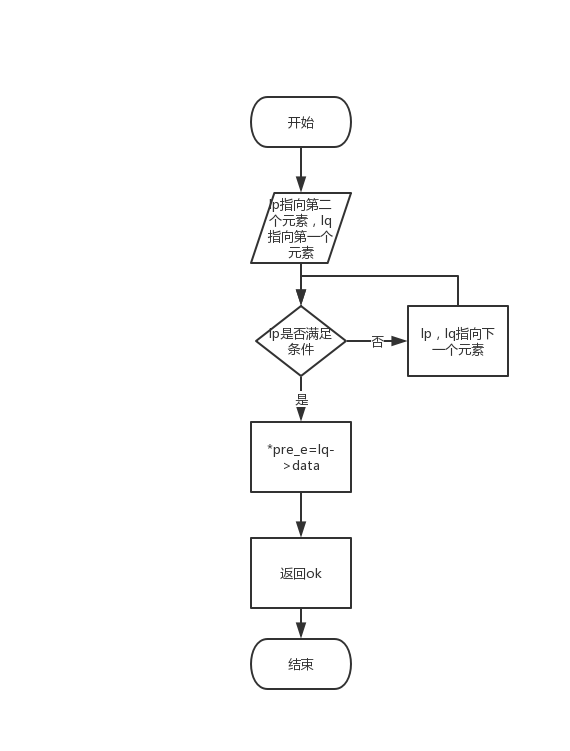


图2-3查找前驱元素

（8）查找前驱元素：函数参数为指向结构的二级指针，类型为Lnode \*指向类型为Elemtype的元素的指针，以及需要查找的元素。若链表未创建，则返回-1；若未找到返回0；查找前驱元素流程图如图2-3。

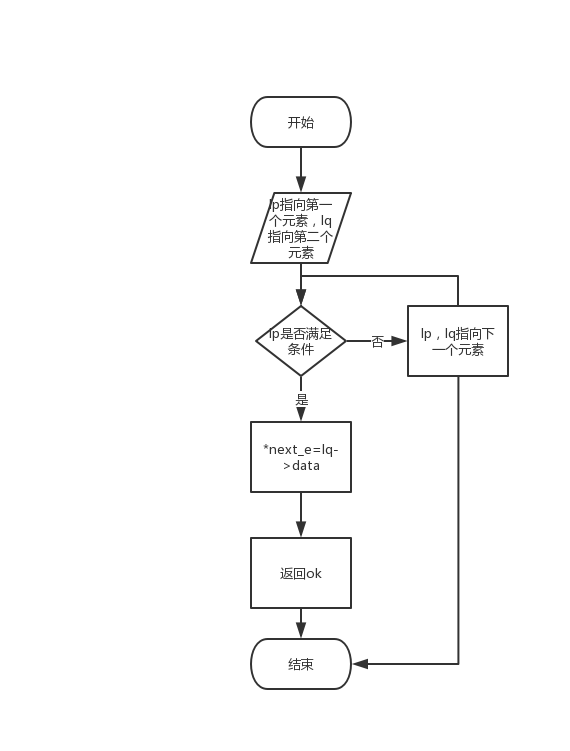


图2-4查找后继元素

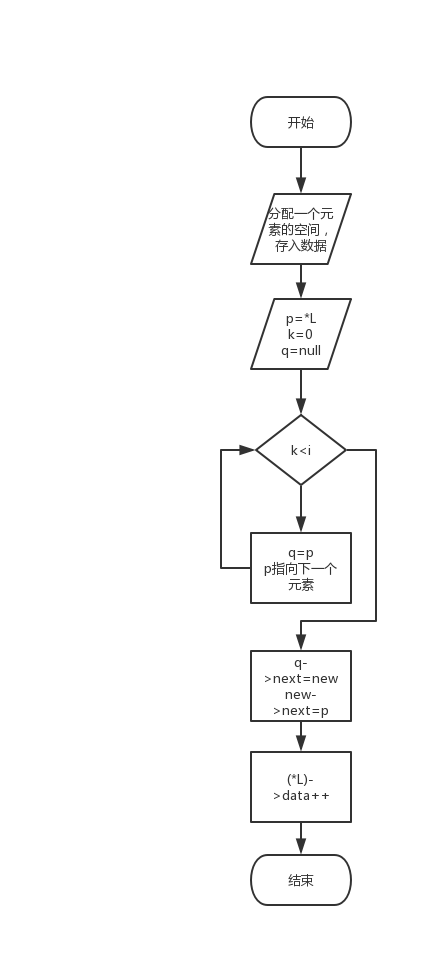


图2-5插入元素

（9）查找后继元素：函数参数为指向结构的二级指针，类型为Lnode \*，指向类型为Elemtype的元素的指针，以及需要查找的元素。若链表未创建，则返回-1；若未找到返回0；查找后继元素流程图如图2-4。

（10）插入元素：函数参数为指向结构的二级指针，类型为Lnode \*，需要插入的位置，以及需要插入的元素。若链表未创建，则返回-1；若空间不够则追加分配空间；插入元素流程图如图2-5。

（11）删除元素：函数参数为指向结构的二级指针，类型为Lnode \*，需要删除的位置，以及需要删除的元素。若链表未创建，则返回-1；删除元素流程图如图2-6。

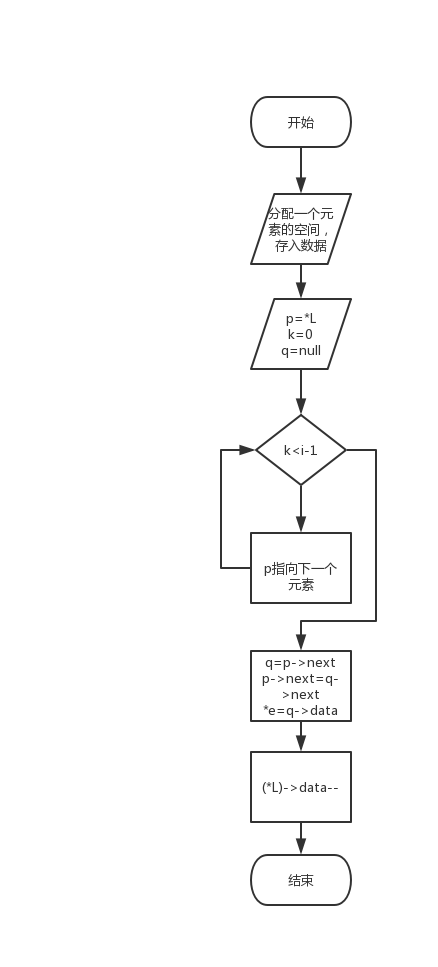


图2-6删除元素

（12）遍历：函数参数为指向结构的二级指针，类型为Lnode \*。利用循环一次输出表中元素。

（13）写入文件：函数参数为指向结构的二级指针，类型为Lnode \*。如果路径错误，则返回ERROR；否则，将表中元素通过循环依次写入文件，当碰到表尾则退出循环，返回OK，关闭文件。

（14）读取文件：函数参数为指向结构的二级指针，类型为Lnode \*。如果路径错误，则返回ERROR；否则，将文件中元素通过循环依次读取存入表中，当碰到文件尾则退出循环，返回OK，关闭文件。

（15）切换链表：函数参数为int型指针。如果输入不合法，则返回ERROR；否则将用户输入的数字赋值到指针指向的元素，通过改变编号来切换顺序表，返回OK。

## 2.4 系统测试

## 2.4.1系统演示操作

（1）.系统一开始会显示菜单提示用户输入选择对1-99号线性表哪一个进行操作。如图2-7所示。



图2-7选择线性表

（2）.选择对线性表1进行操作，进入菜单演示界面，如图2-8所示。

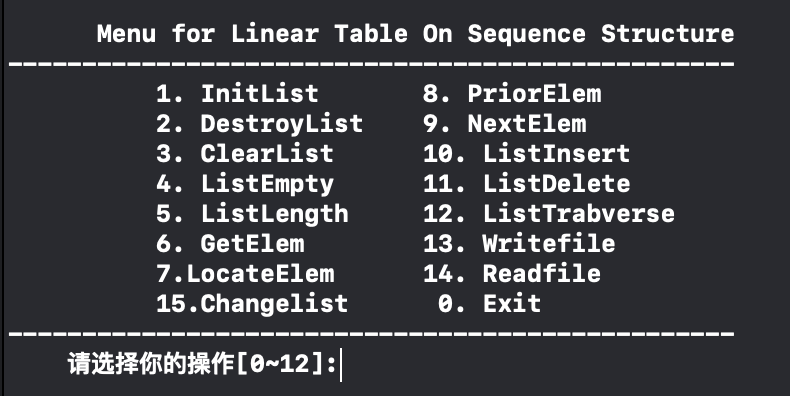


图2-8选择编号1线性表

（3）.输入0，退出演示系统，结束操作，如图2-9所示。

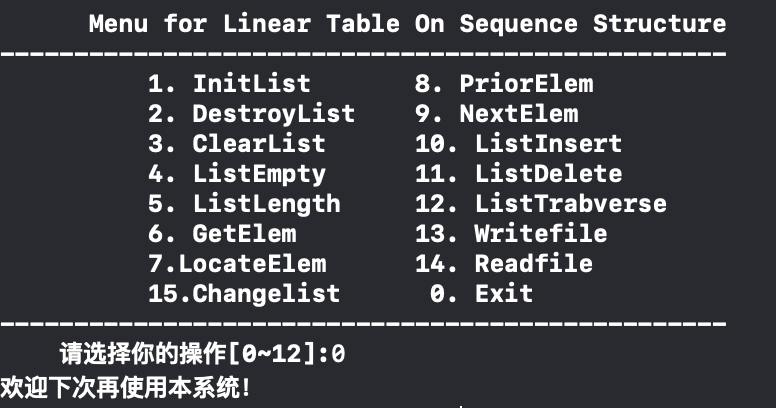


图2-9退出系统

## 2.4.2测试计划

表一 测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能及序号 | 输入要管理的线性表序号 | 输入函数的参数（具体元素） | 预计输出 | 此时线性表的状态 |
| 1.InitList | 1 | L的地址 | 线性表初始化成功 | 分配了表头空间，（\*L）->data=0,  (\*L)->next=null |
| 2.DestroyList | 1 | L的地址 | 销毁成功 | 表头为null |
| 3.ClearList | 1 | L的地址 | 清空成功 | (\*L)->data=0 |
| 10.ListInsert(多次调用) | 1 | L的地址，插入位置1，2，3，4，5，插入元素1，2，3，4，5 | 插入成功 | 线性表，序列为：1，2，3，4，5 |
| 4.ListEmpty | 1 | L的地址 | 该表不是空表 | 线性表，序列为：1，2，3，4，5 |
| 5.ListLength | 1 | L的地址 | 该表长为5 | 线性表，序列为：1，2，3，4，5 |
| 6.GetElem | 1 | 输入数据元素的位序为3 | 输出线性表的第三个元素为3 | 线性表，序列为：1，2，3，4，5 |
| 8.PriorElem | 1 | 输入数据元素的值为4 | 找到4的前驱元素是3 | 线性表，序列为：1，2，3，4，5 |
| 9.NextElem | 1 | 输入数据元素的值为2 | 找到2的后继元素是3 | 线性表，序列为：1，2，3，4，5v |
| 11.ListDelete | 1 | 输入数据元素的置为3 | 删除成功，删除的元素是3 | 线性表，序列为1245 |
| 7.LocateElem | 1 | 输入数据元素的值为4 | 您要查找的元素是第3元素 | 线性表，序列为1245 |
| 13.Writefile | 1 | 输入保存到的文件名为1.txt | 写入成功 | 文件显示1，2，4，5 |
| 14.Readfile | 1 | 输入要加载的文件名为2.txt | 读取成功 | 线性表序列为1，2，3，4，5，6，7，8，9，10 |
| 15.Changelist | 1 | 输入编号2 | 切换成功 | 不存在线性表空间 |

**2.4.3系统演示**

（1）.输入对线性表1进行操作，进入菜单演示界面，执行功能1，初始化线性表，测试结果如图2-10所示。

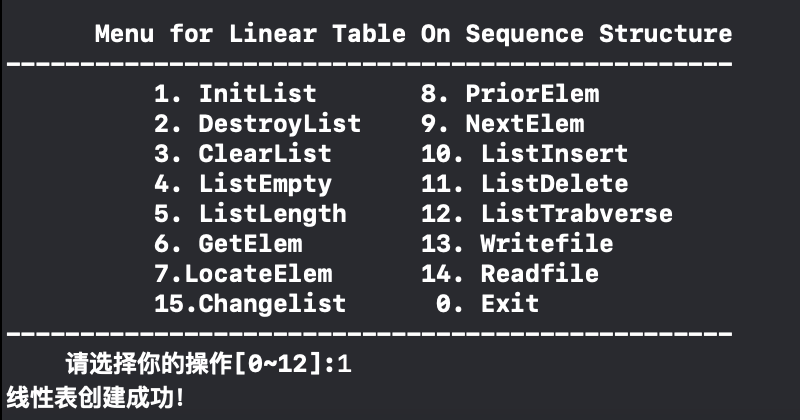


图2-10创建链表

（2）.执行功能2，销毁线性表，测试结果如图2-11所示。

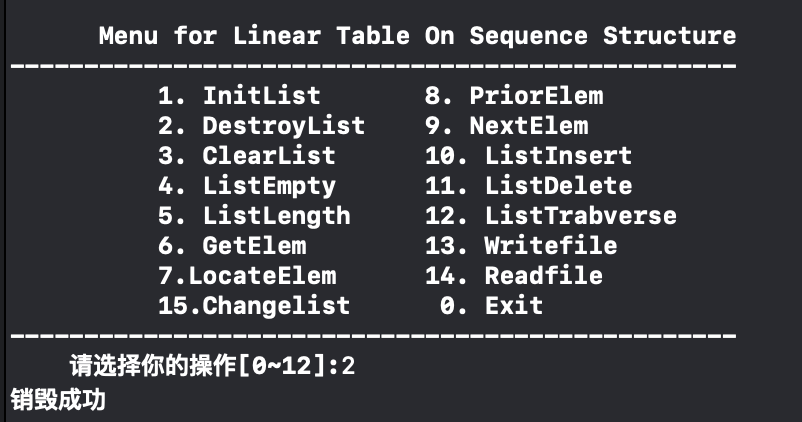


图2-11销毁链表

（3）.执行功能10，往线性表中添加数据元素1,2,3,4,5，测试结果如图2-12所示。

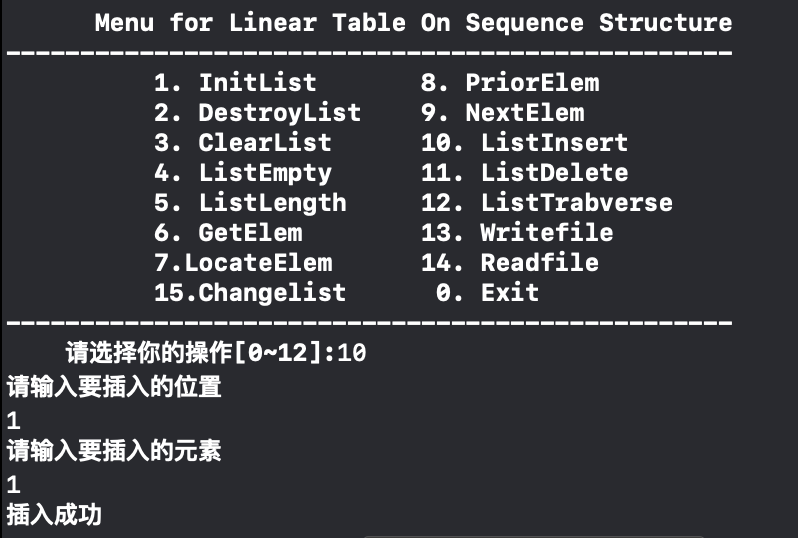


图2-12插入元素

（4）.执行功能4，判断线性表是否为空，测试结果如图2-13所示。

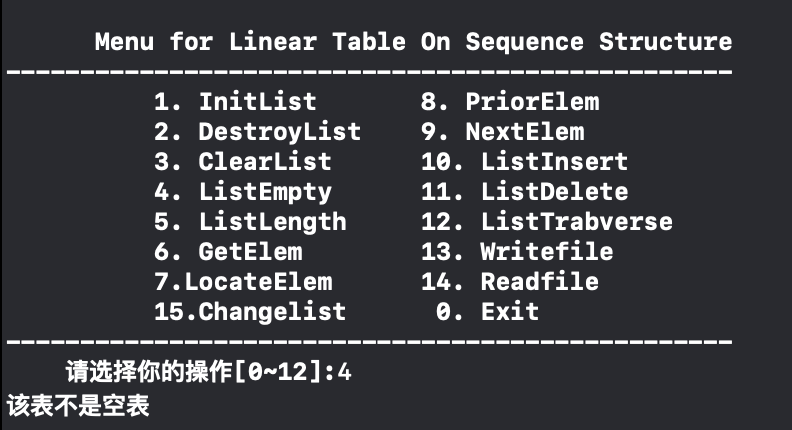


图2-13判定空表

（5）.执行功能5，求线性表的长度，测试结果如图2-24所示。

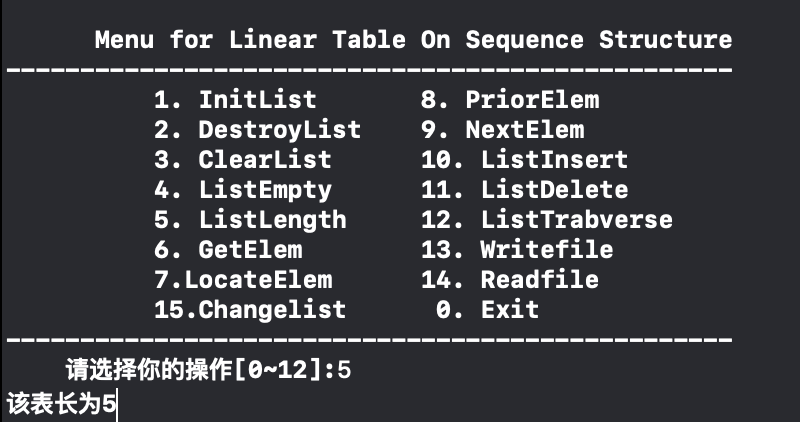


图2-14求表长

（6）.执行功能6，查找第三个数据元素的数值，测试结果如图2-15所示。

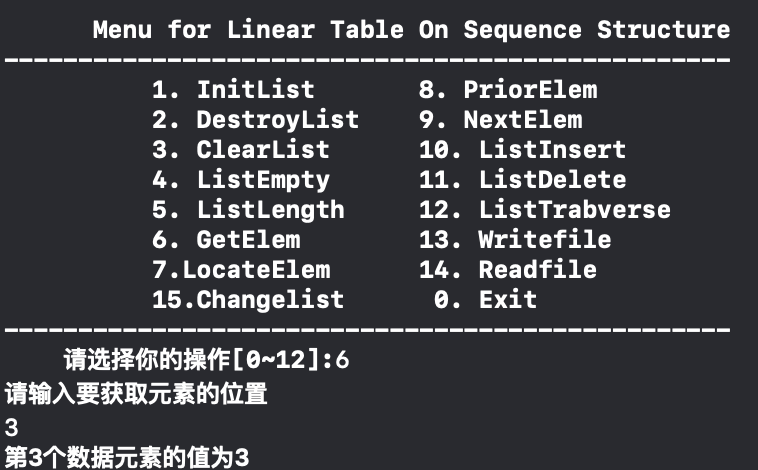


图2-15获取元素

（7）.执行功能8，确定值为4的数据元素前驱数据元素，测试结果如图2-16所示。

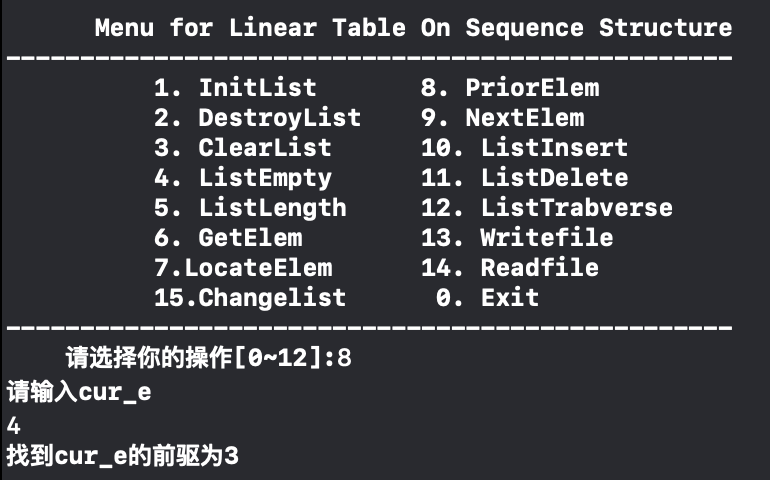


图2-16查找前驱

（8）.执行功能9，确定值为2的数据元素的后继数据元素，测试结果如图2-17所示。



图2-17查找后继

（9）.执行功能11，删除第三个数据元素，测试结果如图2-18所示。

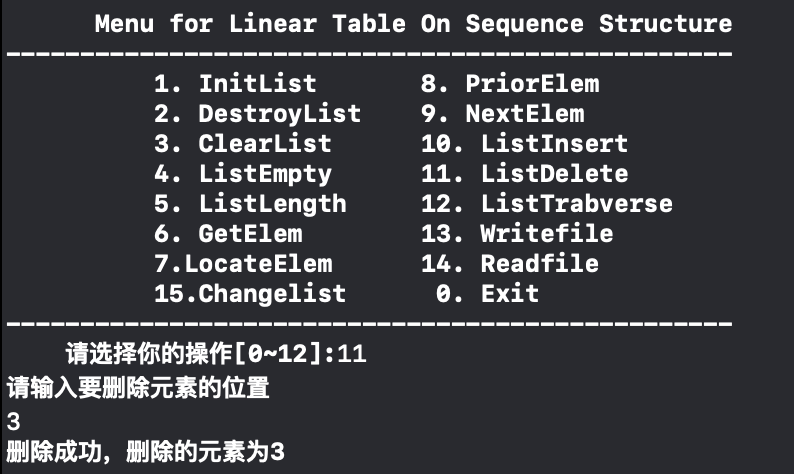


图2-18删除元素

（10）.执行功能13，保存线性表中的数据元素值到文件名为1.txt的文件中，测试结果如图2-19所示。

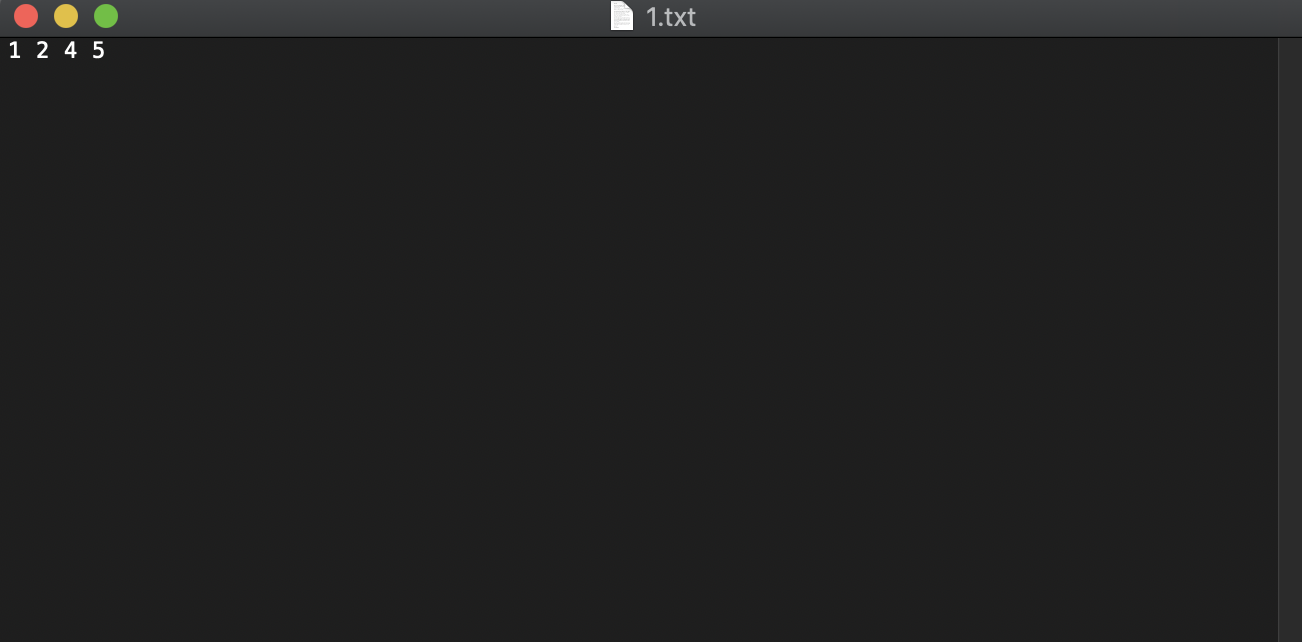


图2-19写入文件

（11）.执行功能3，14，将2.txt中元素读取到链表，测试结果如图2-20所示。

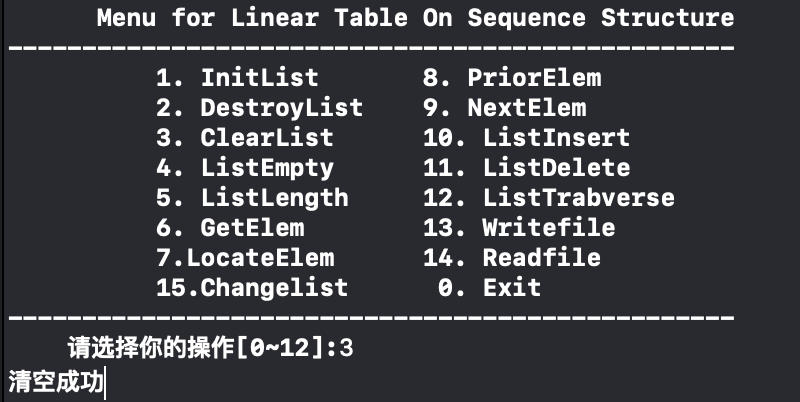


图2-20清空链表

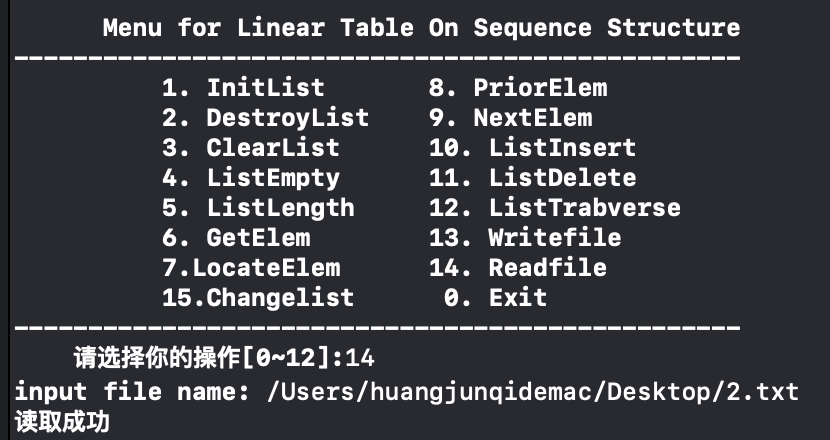


图2-21读取文件

（12）.执行功能12，遍历并输出线性表中的元素，应该输出1，2，3，4，5，6，7，8，9，10，测试结果如图2-22所示。

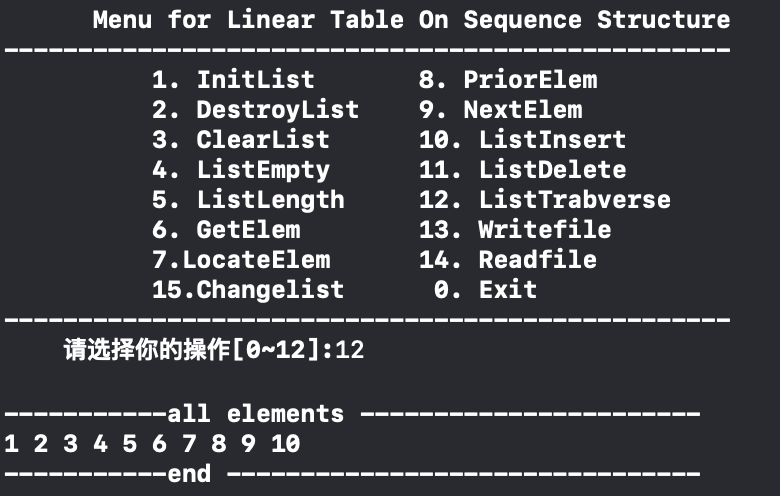


图2-22遍历元素

（13）.执行功能7，查找元素6，测试结果如图2-23所示。

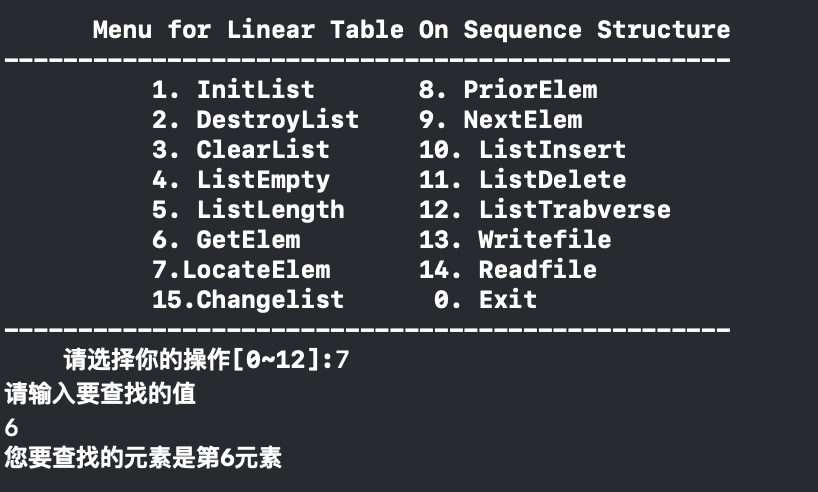


图2-23查找元素

（14）.执行功能15，输入编号2，测试结果如图2-24所示。

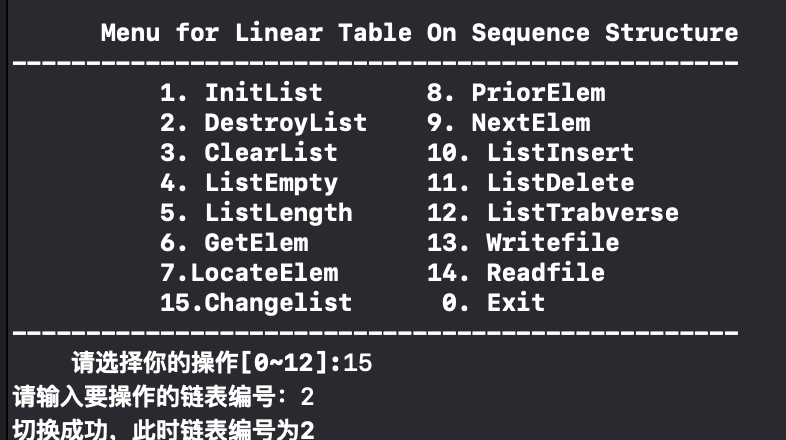


图2-24切换链表

**2.4.4异常数据**

（1）查找前驱异常情况：初始线性表中已有三个元素，依次是1，2，3，若查找第一个元素的前驱元素，测试结果如图2-25.

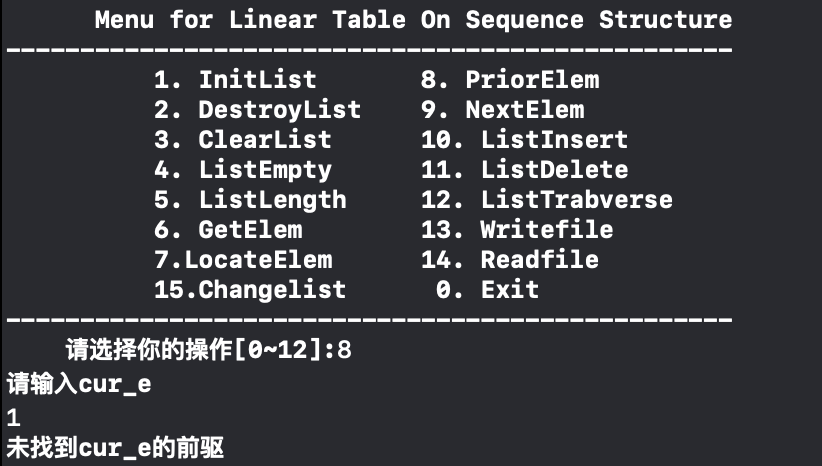


图2-25查找前驱异常

（2）查找后继异常情况：初始线性表中已有三个元素，依次是1，2，3，若查找最后一个元素的后继元素，测试结果如图2-26.

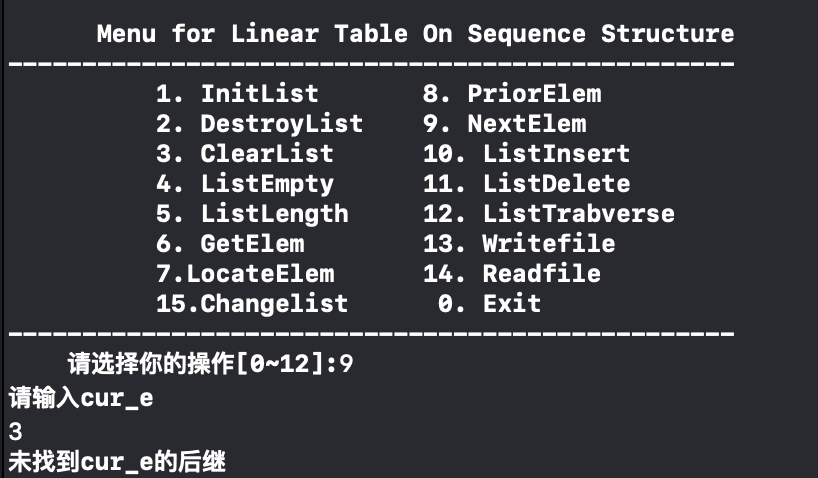


图2-26查找后继异常

（3）输入不合法：以插入元素为例，若输入不合法，测试结果如图2-27.

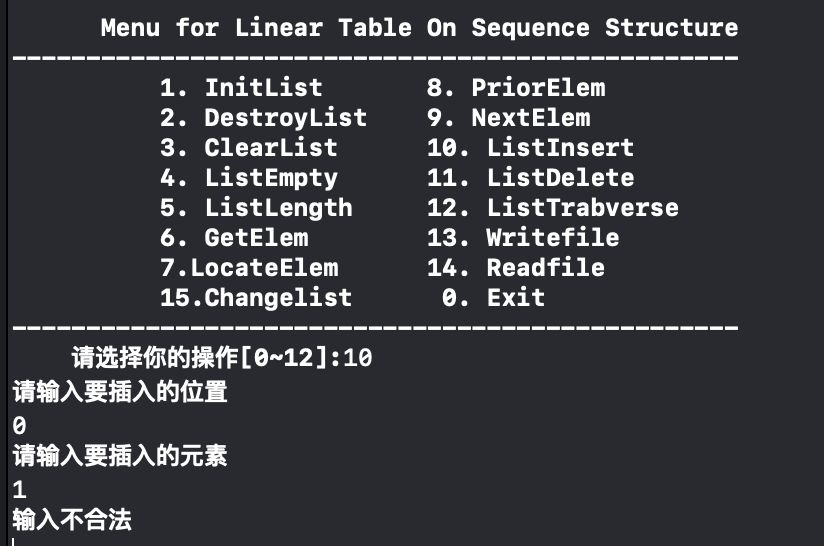


图2-27输入不合法

（4）不存在可执行线性表：以查找元素为例，若未创建线性表，不会显示未找到元素，测试结果如图2-28

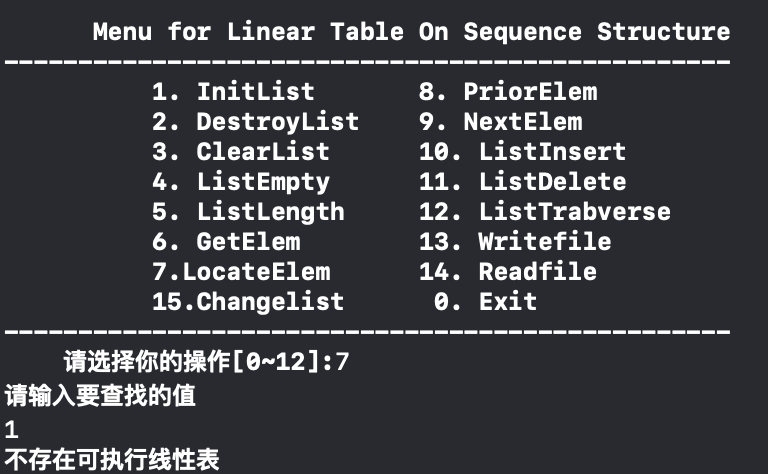


图2-28不存在可执行线性表

## 2.5 实验小结

本次实验中，几个函数都有同样的问题——语句的顺序会影响输出结果。比如：在功能6，获取元素时，若未找到元素，函数返回值为-1，导致主函数中会显示“您要查找的元素是第-1元素。”解决方法是，主函数对于-1的判断放在后面，即当判断返回值为正数，显示您已找到元素，当判断返回值为0，显示未创建链表，当判断返回值为-1，显示未找到元素。这个判断顺序不能改变。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

采用二叉链表作为二叉树的物理结构，实现二叉树的基本运算，数据元素的类型名可自行定义。要求构造一个具有菜单的功能演示系统，其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示。

## 3.1.1二叉树的基本概念

二叉树是一种树型结构，即n个结点的有限集，它的特点是每个结点至多只有两棵子树（即二叉树中不存在度大于2的结点），并且，二叉树的子树有左右之分，其次序不能任意颠倒。

## 3.1.2函数定义

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

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

**注：**①要求T中各结点关键字具有唯一性。后面各操作的实现，也都要满足一棵二叉树中关键字的唯一性，不再赘述；②CreateBiTree中根据definition生成T，不应在CreateBiTree中输入二叉树的定义。

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

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

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

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

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

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

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

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

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

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

**注：**前序、中序和后序三种遍历算法，要求至少一个用非递归算法实现。

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

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

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

（15）加载文件：函数名称是Readfile（T），初始条件是二叉树T已存在；操作结果是将文件中的元素加载到二叉树中。

（16）加载文件：函数名称是Writefile（T），初始条件是二叉树T已存在；操作结果是将二叉树中的元素保存到文件中。

（17）切换顺序表：函数名称是Changelist（int \*n），初始条件是二叉树T已存在；操作结果是将线性表切换到指定编号的二叉树。

## 3.2 系统设计

## 3.2.1数据结构定义

typedef struct BiTnode{

char name;

struct BiTnode \*lchild,\*rchild;

status data;

}BiTnode,\*BiTree;

要实现同时对多个二叉树管理，只需要定义一个结构指针数组，用来存放不同二叉树的根指针即可。

## 3.2.2演示系统

菜单共设置17个函数，由switch语句实现。用户通过菜单来实现相应操作：当输入在1～17之间，进入循环，完成相应操作，由break语句进入下一个循环；当用户输入0，则退出本系统。在第一次进入循环while时首先会询问用户对哪个二叉树进行操作，直至退出演示系统或用户选择切换二叉树之前一直对指定二叉树进行操作。演示系统结构如图3-1。

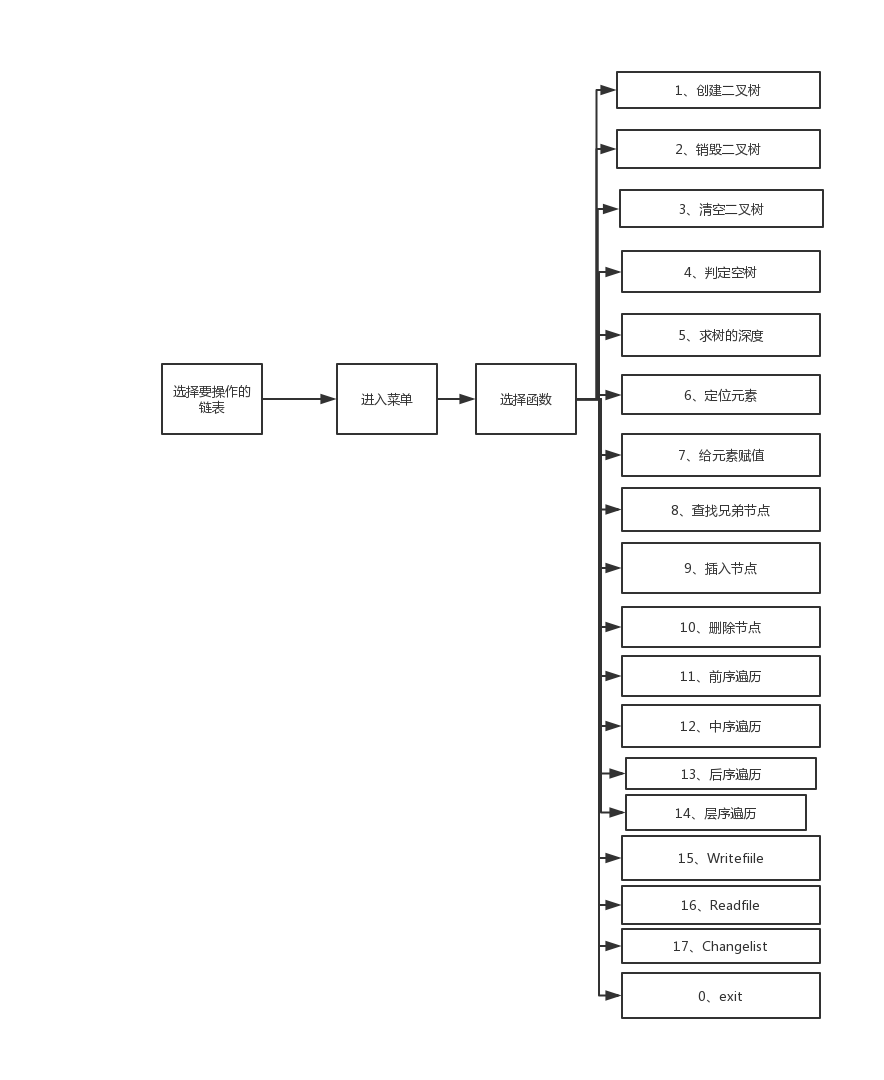


图3-1系统演示

## 3.3 系统实现

## 3.3.1 编程环境

编译软件为Xcode10.3

## 3.3.2 头文件及预定义常量说明

1.头文件

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

2.预定义常量

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define MAXSIZE 100

#define MAXLENG 100

#define QMAXSIZE 100

3.类型表达式

typedef int status;

typedef BiTree QElemType;

## 3.3.3函数实现

（1）创建二叉树：函数的参数是指向二叉树根结点的结构类型指针T，按照先序次序输入二叉树中结点的值，如果第一个字符为#，则T为空二叉树。否则，malloc函数分配一个单元的空间作为树的根结点，并为其赋值，采取递归的方式继续创建根结点的左子树和右子树。

（2）销毁二叉树：函数的参数是指向二叉树根结点的结构类型指针T，采取递归的方式先销毁二叉树的左子树，在销毁二叉树的右子树，最后用free函数释放掉根结点对应的内存空间。

（3）清空二叉树：函数的参数同上，将根结点的左右孩子指针置为空，此时其左右子树的存储空间并未释放掉。

（4）判定空二叉树：，对于一个二叉树，若根结点不存在则为空二叉树，否则不是空二叉树，那么只需要判断指向根结点的结构类型指针T是否为空即可。

（5）求二叉树的深度：采取递归的方式求二叉树的深度，如果根结点的左右孩子都不存在，则返回树的深度为1，否则的话返回根结点左右子树的深度的最大加上一。

（6）获得节点指针：函数的参数是二叉树的头指针以及主函数中输入的关键字，通过递归先序遍历二叉树，即先比较根结点的关键字与给定是否一致，若一致，返回该结点的指针，否则继续分别递归遍历其左子树和右子树。

（7）给节点赋值：函数的参数是二叉树的头指针，主函数中输入的关键字，以及要赋值的value，通过递归先序遍历二叉树，即先比较根结点的关键字与给定是否一致，若一致，将value赋值给指定关键字的节点，否则继续分别递归遍历其左子树和右子树。

（8）查找兄弟节点：函数参数是二叉树的头指针，主函数中输入的关键字。查找右兄弟：通过递归先序遍历二叉树，若某节点的左子树的关键字是要查找的关键字，则返回该节点的右节点指针并打印其其关键字；同理，可以查找左兄弟。若没有查找到某节点的兄弟，则返回NULL。

（9）插入子树：函数参数是二叉树头指针，主函数中输入的关键字，判断插入左右的LR，要插入的节点c。在主函数中先给c分配空间，赋予关键字和要存储的数据。通过先序遍历查找指定关键字的节点，若未找到则return ERROR，否则，通过LR判断要插入的是该节点的左子树还是右子树，若节点已存在左子树（或右子树），将原有左子树（或右子树）赋给c->rchild。

（10）删除子树：函数参数是指向二叉树的指针，主函数中输入的关键字。通过先序遍历查找到指定关键字的节点，若未找到，则return ERROR；否则，判断节点的度：若其左子树，右子树都不为空，则直接free；若其左子树，右子树中只有一个为空，则将不为空的子树替换该节点；若其左，右子树中只都不为空：若左子树不为空，利用循环找到该节点的左子树的最右节点，将其右子树赋给最右节点的右子树，将该节点的左子树替换该节点，free该节点。

（11）递归遍历：函数参数是二叉树的根指针，以及函数指针Visit。首先访问根结点，并对其执行遍历操作，操作成功后采取递归的方式遍历根结点的左子树，返回值为OK时继续遍历其右子树，最终遍历完成。递归方法前序遍历、中序遍历以及后序遍历的根本区别在与访问根结点的操作是在两次递归操作之间之前还是之后。

（12）非递归前序遍历：函数的形参是二叉树的头指针，仿照递归过程堆栈的使用，首先将根结点的值进栈，执行循环体，栈非空，弹出根结点并执行访问操作，将其右子树根结点指针进栈，再将其左子树根结点指针进堆栈，执行循环。

（13）层序遍历：函数的形参是二叉树的头指针，借助队列，使根结点进入队列，根结点出队列，执行访问操作，此时将根结点的左右子树根结点依次进入队列，即队列中的某个元素出队列时将其左右子树根结点放进队列，循环即可一层一层的访问二叉树。

（14）写入文件：函数参数为指向结构的二级指针，类型为BiT ree\*。如果路径错误，则返回ERROR；否则，将二叉树中元素通过循环依次写入文件，当碰到表尾则退出循环，返回OK，关闭文件。

（15）读取文件：函数参数为指向结构的二级指针，类型为BiTree \*。如果路径错误，则返回ERROR；否则，将文件中元素通过循环依次读取存入二叉树中，当碰到文件尾则退出循环，返回OK，关闭文件。

（16）切换二叉树：函数参数为int型指针。如果输入不合法，则返回ERROR；否则将用户输入的数字赋值到指针指向的元素，通过改变编号来切换二叉树，返回OK。

## 3.4 系统测试

## 3.4.1系统演示操作

（1）.系统一开始会显示菜单提示用户输入选择对1-99号线性表哪一个进行操作。如图3-2所示。



图3-2选择二叉树

（2）.选择二叉树1进行操作，进入菜单演示界面，如图3-3所示。



图3-3选择编号1线性表

（3）.输入0，退出演示系统，结束操作，如图3-4所示。



图3-4退出系统

## 3.4.2测试计划

表一 测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能及序号 | 输入要管理的二叉树序号 | 输入函数的参数（具体元素） | 预计输出 | 此时二叉树的状态 |
| 1. CreateBiTree | 1 | a 1 | 初始化成功 | 含有根节点，关键字为a，存放数据为1 |
| 3. ClearBiTree | 1 | 无 | 清空成功 | T！=null |
| 9. InsertChild (多次调用) | 1 | b 2 LR=0  c 3 LR=1 | 插入成功 | a的左孩子为b，存放数据为2，右孩子为c，存放数据为3 |
| 4. BiTreeEmpty | 1 | 无 | 该树不是空树 | 同上 |
| 5. BiTreeDepth | 1 | 无 | 深度为2 | 同上 |
| 6. LocateNode | 1 | b | 成功找到该节点，存放数据2 | 同上 |
| 8. Brother | 1 | b | 成功找到右兄弟，其关键字为c | 同上 |
| 7. Assign | 1 | b 4 | 成功赋值 | 同上 |
| 11.PreOrder | 1 | 无 | a：1 b：4 c：3 | 同上 |
| 12.MidOrder | 1 | 无 | b：4 a：1 c：3 | 同上 |
| 13.PostOrder | 1 | 无 | b：4 c：3 a：1 | 同上 |
| 10.DeleteNode | 1 | a | 成功删除 | 同上 |
| 14.LevelOrder | 1 | 无 | b：4 c：3 | 同上 |
| 15.Writefile | 1 | 输入存放地址 | 写入成功 | 同上 |
| 2.DestroyBiTree | 1 | 无 | 销毁成功 | T=null |
| 17. Changetree | 1 | 输入编号2 | 切换成功 | 不存在二叉树 |

**3.4.3系统演示**

（1）执行功能1，创建二叉树，根节点关键字为a，存放数据1，测试结果如图3-4所示



图3-4初始化二叉树

（2）执行功能3，清空二叉树，测试结果如图3-5所示



图3-5清空二叉树

（3）执行功能1，再执行功能9，把节点b插入到a的左孩子，节点c插入到a的右孩子，测试结果如图3-6所示

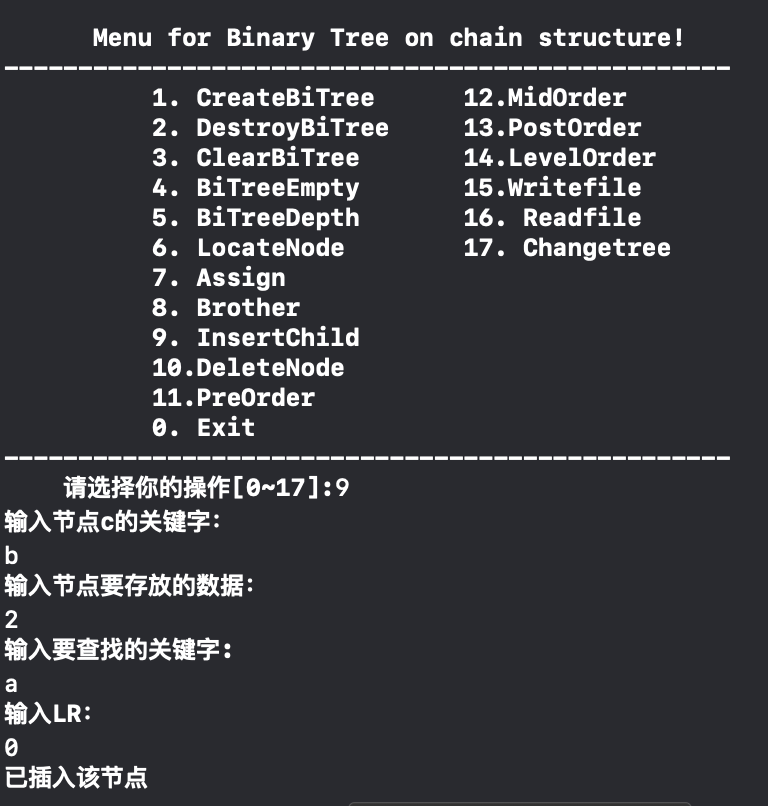




图3-6插入节点

（4）执行功能4，判断是否为空树，测试结果如图3-7所示

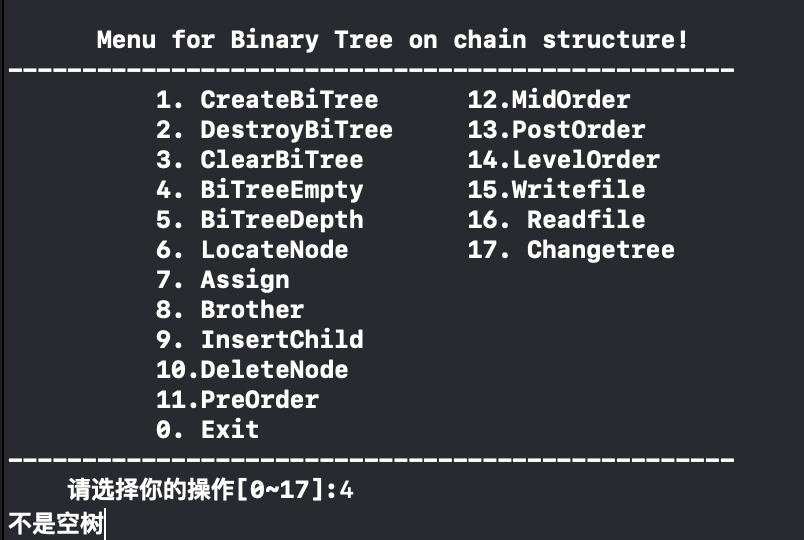


图3-7判断空树

（5）执行功能5，求树的深度，测试结果如图3-8所示



图3-8求树的深度

（6）执行功能6，定位b节点，返回数据，测试结果如图3-9所示



图3-9定位节点

（7）执行功能8，查找b的兄弟，测试结果如图3-10所示



图3-10查找兄弟节点

（8）执行功能7，给节点b赋值，测试结果如图3-11所示



图3-11给节点赋值

（9）执行功能11，前序遍历，测试结果如图3-12所示



图3-12前序遍历

（10）执行功能12，中序遍历，测试结果如图3-13所示



图3-13中序遍历

（11）执行功能13，后序遍历，测试结果如图3-14所示



图3-14后序遍历

（12）执行功能10，删除节点a，测试结果如图3-15所示



图3-15删除节点

（13）执行功能14，层序遍历，测试结果如图3-16所示



图3-16层序遍历

（14）执行功能15，将二叉树数据存入文件，测试结果如图3-17所示



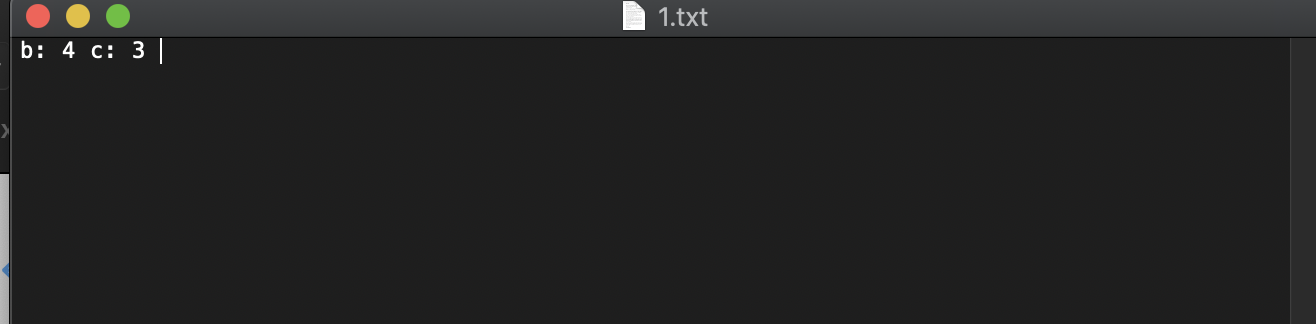


图3-17写入文件

（15）执行功能2，销毁二叉树，测试结果如图3-18所示

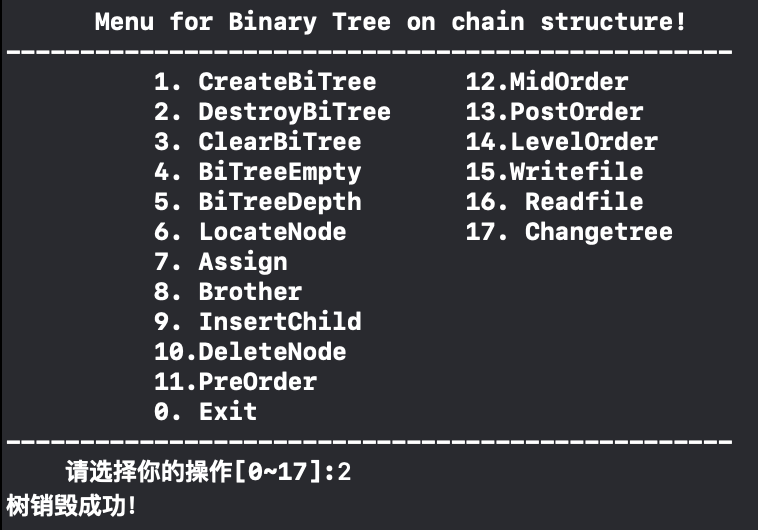


图3-18销毁二叉树

（16）执行功能17，切换二叉树，测试结果如图3-19所示



图3-19切换二叉树

**3.4.4异常数据**

（1）查找不存在节点，以功能6为例，测试结果如图3-20所示



图3-20不存在要查找的节点

（2）无兄弟节点，测试结果如图3-21所示



图3-21无兄弟节点

（3）删除不存在的节点测试结果如图3-22所示



图3-22删除不存在节点

（4）不存在可执行二叉树，以功能11为例，测试结果如图3-23所示

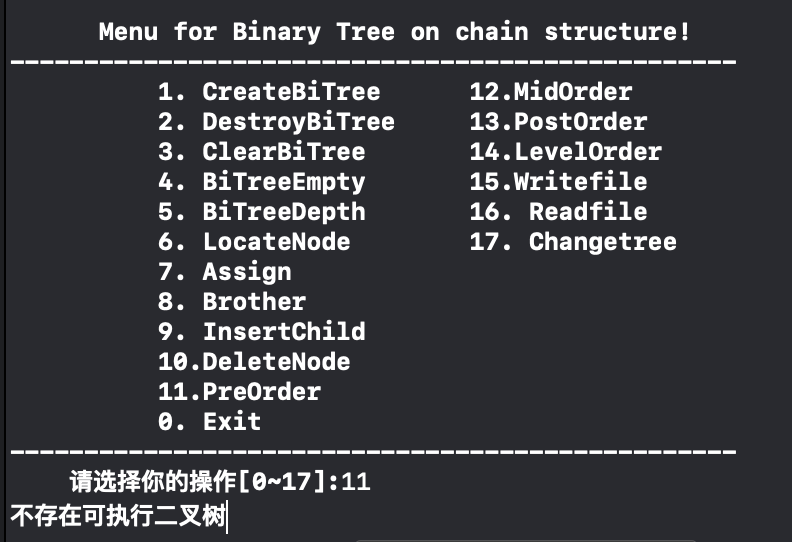


图3-23不存在可执行二叉树

## 3.5 实验小结

本次实验最难的是删除节点，因为要删除的节点的度有三种情况，度为0和度为1的情况比较好解决，对于删除度为2的节点，关键在于要用循环先找到要删除节点的左孩子的最有节点，将右孩子连接到左孩子的最右节点，再将要删除节点替换问它的左孩子。

# 4 基于邻接表的图实现

## 4.1 问题描述

## 4.2 系统设计

## 4.3 系统实现

## 4.4 系统测试

## 4.5 实验小结

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

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

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

#include <stdio.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

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);

status DestroyList(SqList \*L);

status ClearList(SqList \*L);

status ListEmpty(SqList \*L);

status ListLength(SqList \*L);

status GetElem(SqList \*L,int i,ElemType \*e);

status LocateElem(SqList \*L,ElemType e); //简化过

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

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

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

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

status ListTrabverse(SqList \*L); //简化过

status Writefile(SqList \*L);

status Readfile(SqList \*L);

status Changelist(int \*N);

status compare(int i,ElemType e);

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

int main(void){

SqList List[100];//多表存储于一个结构数组

int N;

printf("请输入0～99以确定将对哪个顺序表进行处理\n");

scanf("%d",&N);//N为顺序表编号

int op=1,i;

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

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. Writefile\n");

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

printf(" 15.Changelist 0. Exit\n");

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

printf(" 请选择你的操作[0~12]:");

scanf("%d",&op);

switch(op){

case 1:

if(InitList(&List[N])==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(&List[N])==OK) printf("销毁成功");

else printf("不存在可执行线性表");

getchar();getchar();

break;

case 3:

if(ClearList(&List[N])==OK) printf("清空成功");

else if(ClearList(&List[N])==ERROR) printf("不存在可执行线性表");

getchar();getchar();

break;

case 4:

if(ListEmpty(&List[N])==TRUE) printf("该表为空表");

else if(ListEmpty(&List[N])==FALSE) printf("该表不是空表");

else printf("不存在可执行线性表");

getchar();getchar();

break;

case 5:

if(ListLength(&List[N])==-1) printf("不存在可执行线性表");

else printf("该表长为%d",ListLength(&List[N]));

getchar();getchar();

break;

case 6:

printf("请输入要获取元素的位置\n");

scanf("%d",&i);

if(GetElem(&List[N],i,&e)==OVERFLOW) printf("输入不合法\n");

else

printf("第%d个数据元素的值为%d",i,e);

getchar();getchar();

break;

case 7:

printf("请输入要查找的值\n");

scanf("%d",&e);

if(LocateElem(&List[N],e)==0) printf("不存在需要查找的元素\n");

else printf("您要查找的元素是第%d元素\n",LocateElem(&List[N],e));

getchar();getchar();

break;

case 8:

printf("请输入cur\_e\n");

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

if(PriorElem(&List[N],cur\_e,&pre\_e)==1) printf("找到cur\_e的前驱为%d\n",pre\_e);

else printf("未找到cur\_e的前驱\n");

getchar();getchar();

break;

case 9:

printf("请输入cur\_e\n");

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

if(NextElem(&List[N],cur\_e,&next\_e)==1) printf("找到cur\_e的后继为%d\n",next\_e);

else printf("未找到cur\_e的后继\n");

getchar();getchar();

break;

case 10:

printf("请输入要插入的位置\n");

scanf("%d",&i);

printf("请输入要插入的元素\n");

scanf("%d",&e);

if(ListInsert(&List[N],i,e)==OK) {

printf("插入成功\n");

break;}

else if(ListInsert(&List[N],i,e)==ERROR) printf("输入不合法\n");

else if(ListInsert(&List[N],i,e)==-1) printf("不存在该线性表\n");

else printf("分配空间不足\n");

getchar();getchar();

break;

case 11:

printf("请输入要删除元素的位置\n");

scanf("%d",&i);

if(ListDelete(&List[N],i,&e)==OK) printf("删除成功，删除的元素为%d\n",e);

else printf("输入不合法\n");

getchar();getchar();

break;

case 12:

if(!ListTrabverse(&List[N])) printf("线性表是空表！\n");

getchar();getchar();

break;

case 0:

break;

case 13:

if(Writefile(&List[N])==OK) printf("写入成功\n");

else printf("写入失败\n");

getchar();getchar();

break;

case 14:

if(Readfile(&List[N])==OK) printf("读取成功\n");

else printf("读取失败\n");

getchar();getchar();

break;

case 15:

if(Changelist(&N)==OK) printf("切换成功，此时顺序表编号为%d\n",N);

else printf("输入不合法\n");

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

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

status InitList(SqList \*L){//创建顺序表

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

if(!L->elem) return OVERFLOW;

L->length=3;//测试

L->listsize=LIST\_INIT\_SIZE;

L->elem[0]=1;//测试

L->elem[1]=2;//测试

L->elem[2]=3;//测试

return OK;

}

status DestroyList(SqList \*L){//销毁顺序表

if((\*L).elem==NULL) return ERROR;

else{

free((\*L).elem);

(\*L).elem=NULL;//销毁空间，包括空间的z地址

L->length=0;

L->listsize=0;

return OK;

}

}

status ClearList(SqList \*L){//清空顺序表

if((\*L).elem==NULL) return ERROR;//特殊情况

else{

L->length=0;

return OK;

}

}

status ListEmpty(SqList \*L){//判断是否为空表

if(L->elem==NULL) return -1;//特殊情况

else{

if(L->length==0) return TRUE;

else return FALSE;}

}

status ListLength(SqList \*L){//求表长

if((\*L).elem==NULL) return -1;//特殊情况

else return L->length;

}

status GetElem(SqList \*L,int i,ElemType \*e){//获取元素

if(i<1||i>ListLength(L)) return OVERFLOW;//不合法输入

else{

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

return OK;

}

}

status LocateElem(SqList \*L,ElemType e){//通过compare函数定位元素的位置

int j,t=0;//t用于判断元素是否存在

for(j=0;j<(L->length);j++)

{

if(compare(L->elem[j],e)){

t=1;

break;

}

}

if(t==1) return j+1;

else return 0;

}

status PriorElem(SqList \*L,ElemType cur\_e,ElemType \*pre\_e){//查找前驱

int j,t=0;//t用于判断除了第一个外是否有前驱

for(j=1;j<L->length;j++)

{

if(L->elem[j]==cur\_e){

t=1;

break;

}

}

if(t==0) return ERROR;

else{

\*pre\_e=L->elem[j-1];

return OK;

}

}

status NextElem(SqList \*L,ElemType cur\_e,ElemType \*next\_e){//查找后继

int j,t=0;//t用于判断除了第一个外是否有后继

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

{

if(L->elem[j]==cur\_e){

t=1;

break;

}

}

if(t==0) return ERROR;

else{

\*next\_e=L->elem[j+1];

return OK;

}

}

status ListInsert(SqList \*L,int i,ElemType e){//将元素插入到指定位置

ElemType \*newbase;

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

if(L->elem==NULL) return -1;

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

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

if(!newbase) exit(OVERFLOW);//分配空间不足出口

L->elem = newbase;

L->listsize+=LISTINCREMENT;

}

int j;

for(j=L->length-1;j>=i-1;j--)

L->elem[j+1]=L->elem[j];

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

L->length++;

return OK;

}

status ListDelete(SqList \*L,int i,ElemType \*e){//删除顺序表中元素，并返回该值

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

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

int j;

for(j=i-1;j<L->length;j++)

L->elem[j]=L->elem[j+1];

L->length--;

return OK;

}

status ListTrabverse(SqList \*L){//将顺序表元素输出

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 Writefile(SqList \*L){//将顺序表中的元素写入到文件

int i;

FILE \*fp;char filename[30];

printf("input file name: ");

scanf("%s",filename);

//写文件的方法

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

{

printf("File open error\n ");

return ERROR;

}

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

fprintf(fp, " %d", L->elem[i]);//fwrite针对二进制

//这里是1次性写入，对于其它物理结构，可通过遍历，逐个访问数据元素

//并写入到文件中。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 Readfile(SqList \*L){//将文件中中的元素写入顺序表

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 ERROR;

}

while(!feof(fp)){

fscanf(fp,"%d",&(L->elem[L->length]));//fwrite函数针对二进制，在txt文件中会显示乱码

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;

}

status Changelist(int \*N){//切换顺序表

int k;

printf("请输入要操作的顺序表编号：");

scanf("%d",&k);

if(k<0||k>99) return ERROR;

\*N=k;

return OK;

}

status compare(int i,ElemType e){//比较函数

if(i==e) return OK;

else return ERROR;

}

# 附录B 基于链式存储结构线性表实现的源程序

#include <stdio.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

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 Node{

ElemType data;

struct Node \*next;

}Node,\*Lnode;

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

status InitList(Lnode \*L);

status DestroyList(Lnode \*L);

status ClearList(Lnode \*L);

status ListEmpty(Lnode \*L);

status ListLength(Lnode \*L);

status GetElem(Lnode \*L,int i,ElemType \*e);

status LocateElem(Lnode \*L,ElemType e); //简化过

status PriorElem(Lnode \*L,ElemType cur\_e,ElemType \*pre\_e);

status NextElem(Lnode \*L,ElemType cur\_e,ElemType \*next\_e);

status ListInsert(Lnode \*L,int i,ElemType e);

status ListDelete(Lnode \*L,int i,ElemType \*e);

status ListTrabverse(Lnode \*L); //简化过

status Writefile(Lnode \*L);

status Readfile(Lnode \*L);

status Changelist(int \*N);

status compare(int i,ElemType e);

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

int main(void){

Lnode List[100];//多表存储于一个结构数组

int N;

printf("请输入0～99以确定将对哪个链表进行处理\n");

scanf("%d",&N);//N为顺序表编号

int op=1,i;

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

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. Writefile\n");

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

printf(" 15.Changelist 0. Exit\n");

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

printf(" 请选择你的操作[0~12]:");

scanf("%d",&op);

switch(op){

case 1:

if(InitList(&List[N])==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(&List[N])==OK) printf("销毁成功");

else printf("不存在可执行线性表");

getchar();getchar();

break;

case 3:

if(ClearList(&List[N])==OK) printf("清空成功");

else if(ClearList(&List[N])==ERROR) printf("不存在可执行线性表");

getchar();getchar();

break;

case 4:

if(ListEmpty(&List[N])==TRUE) printf("该表为空表");

else if(ListEmpty(&List[N])==FALSE) printf("该表不是空表");

else printf("不存在可执行线性表");

getchar();getchar();

break;

case 5:

if(ListLength(&List[N])==-1) printf("不存在可执行线性表");

else printf("该表长为%d",ListLength(&List[N]));

getchar();getchar();

break;

case 6:

printf("请输入要获取元素的位置\n");

scanf("%d",&i);

if(GetElem(&List[N],i,&e)==OVERFLOW) printf("输入不合法\n");

if(GetElem(&List[N],i,&e)==-1) printf("输入不合法\n");

else

printf("第%d个数据元素的值为%d",i,e);

getchar();getchar();

break;

case 7:

printf("请输入要查找的值\n");

scanf("%d",&e);

if(LocateElem(&List[N],e)==0) printf("不存在需要查找的元素\n");

else if(LocateElem(&List[N],e)==-1)printf("不存在可执行线性表");

else if(LocateElem(&List[N],e)) printf("您要查找的元素是第%d元素\n",LocateElem(&List[N],e));

getchar();getchar();

break;

case 8:

printf("请输入cur\_e\n");

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

if(PriorElem(&List[N],cur\_e,&pre\_e)==1) printf("找到cur\_e的前驱为%d\n",pre\_e);

else printf("未找到cur\_e的前驱\n");

getchar();getchar();

break;

case 9:

printf("请输入cur\_e\n");

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

if(NextElem(&List[N],cur\_e,&next\_e)==1) printf("找到cur\_e的后继为%d\n",next\_e);

else printf("未找到cur\_e的后继\n");

getchar();getchar();

break;

case 10:

printf("请输入要插入的位置\n");

scanf("%d",&i);

printf("请输入要插入的元素\n");

scanf("%d",&e);

if(ListInsert(&List[N],i,e)==OK) {

printf("插入成功\n");

break;}

else if(ListInsert(&List[N],i,e)==ERROR) printf("输入不合法\n");

else if(ListInsert(&List[N],i,e)==-1) printf("不存在该线性表\n");

else printf("分配空间不足\n");

getchar();getchar();

break;

case 11:

printf("请输入要删除元素的位置\n");

scanf("%d",&i);

if(ListDelete(&List[N],i,&e)==OK) printf("删除成功，删除的元素为%d\n",e);

else printf("输入不合法\n");

getchar();getchar();

break;

case 12:

if(!ListTrabverse(&List[N])) printf("线性表是空表！\n");

getchar();getchar();

break;

case 0:

break;

case 13:

if(Writefile(&List[N])==OK) printf("写入成功\n");

else printf("写入失败\n");

getchar();getchar();

break;

case 14:

if(Readfile(&List[N])==OK) printf("读取成功\n");

else printf("读取失败\n");

getchar();getchar();

break;

case 15:

if(Changelist(&N)==OK) printf("切换成功，此时顺序表编号为%d\n",N);

else printf("输入不合法\n");

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

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

status InitList(Lnode \*L){//创建链表

Lnode s;

s=(Node \*)malloc(sizeof(Node));

s->data = 0;

s->next = NULL;

\*L=s;

return OK;

}

status DestroyList(Lnode \*L){//销毁链表

Lnode Lp;

if((\*L)==NULL) return ERROR;

while((\*L)!= NULL)

{

Lp = \*L;

(\*L) = (\*L)->next;

free(Lp);

}

return OK;

}

status ClearList(Lnode \*L){//清空链表

Lnode Lp=NULL, Lq=NULL;

if ((\*L)==NULL) return ERROR;

Lp =(\*L)->next;

while (Lp!= NULL)

{

Lq = Lp;

Lp = Lp->next;

free(Lq);

}

(\*L)->data = 0;

(\*L)->next=NULL;

return OK;

}

status ListEmpty(Lnode \*L){//判断是否为空表

if ((\*L)==NULL) return -1;

if((\*L)->data==0)

return TRUE;

else return FALSE;

}

status ListLength(Lnode \*L){//求表长

if((\*L)==NULL) return -1;

if((\*L)->data==0)

return ERROR;

else return (\*L)->data;

}

status GetElem(Lnode \*L,int i,ElemType \*e){//获取元素

if (i<1 || i>(\*L)->data) return -2;

if((\*L)==NULL) return -1;

Lnode Lp;

Lp = \*L;

int k;

for (k = 0; k < i; k++)

{

Lp = Lp->next;

}

\*e = Lp->data;

return OK;

}

status LocateElem(Lnode \*L,ElemType e){//通过compare函数定位元素的位置

Lnode Lp;

if ((\*L)==NULL) return -1;

int k=(\*L)->data;

int i;

Lp=(\*L)->next;

for(i=1;i<=k;i++){

if(compare(Lp->data,e)==OK) return i;

Lp=Lp->next;

}

return ERROR;

}

status PriorElem(Lnode \*L,ElemType cur\_e,ElemType \*pre\_e){//查找前驱

if((\*L)->data==0||(\*L)->data==1) return 0;

Lnode Lp,Lq;

Lp=(\*L)->next->next;

Lq=(\*L)->next;

while(Lp!=NULL){

if(Lp->data==cur\_e){

\*pre\_e=Lq->data;

return OK;

}

Lp=Lp->next;

Lq=Lq->next;

}

return ERROR;

}

status NextElem(Lnode \*L,ElemType cur\_e,ElemType \*next\_e){//查找后继

if((\*L)->data==0||(\*L)->data==1) return 0;

Lnode Lp,Lq;

Lp=(\*L)->next;

Lq=(\*L)->next->next;

while(Lq!=NULL){

if(Lp->data==cur\_e){

\*next\_e=Lq->data;

return OK;

}

Lp=Lp->next;

Lq=Lq->next;

}

return ERROR;

}

status ListInsert(Lnode \*L,int i,ElemType e){//将元素插入到指定位置

if((\*L)==NULL) return -1;

if (i<1 ||i>(\*L)->data + 1) return ERROR;

Lnode p = \*L,q = NULL,newNode;

newNode = (Lnode)malloc(2\*sizeof(Lnode));

newNode->data= e;

int k=0;

while (k<i)

{

q = p;

p = p->next;

k++;

}

q->next = newNode;

newNode->next = p;

(\*L)->data+=1;

return OK;

}

status ListDelete(Lnode \*L,int i,ElemType \*e){//删除链表中元素，并返回该值

if (i<1 || i>(\*L)->data) return ERROR;

Lnode p = \*L, q;

int k = 0;

while (p!=NULL&&k<i-1)

{

p = p->next;

k++;

}

q = p->next;

p->next = q->next;

\*e=q->data;

free(q);

(\*L)->data-=1;

return OK;

}

status ListTrabverse(Lnode \*L){//将链表元素输出

Lnode p;

if((\*L)->data==0) return ERROR;

p=(\*L)->next;

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

while(p!=NULL){

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

p=p->next;

}

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

return OK;

}

status Writefile(Lnode \*L){//将链表中的元素写入到文件

Lnode p;

p=(\*L)->next;

FILE \*fp;char filename[30];

printf("input file name: ");

scanf("%s",filename);

//写文件的方法

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

{

printf("File open error\n ");

return ERROR;

}

while(p!=NULL){

fprintf(fp,"%d ",p->data);

p=p->next;

}//fwrite针对二进制

//这里是1次性写入，对于其它物理结构，可通过遍历，逐个访问数据元素

//并写入到文件中。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 Readfile(Lnode \*L){//将文件中中的元素写入链表

ClearList(L);

if((\*L)==NULL) return -1;

(\*L)->data = 0;

Lnode p = \*L;

FILE \*fp;char filename[30];

printf("input file name: ");

scanf("%s",filename);

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

{

printf("File open error\n ");

return ERROR;

}

while (!feof(fp))

{

p->next = (Lnode)malloc(sizeof(Lnode));

p=p->next;

fscanf(fp,"%d",&(p->data));

(\*L)->data++;

}

p->next = NULL;

fclose(fp);

return OK;

}

status Changelist(int \*N){//切换链表

int k;

printf("请输入要操作的顺序表编号：");

scanf("%d",&k);

if(k<0||k>99) return ERROR;

\*N=k;

return OK;

}

status compare(int i,ElemType e){//比较函数

if(i==e) return OK;

else return ERROR;

}

# 附录C 基于二叉链表二叉树实现的源程序

#include <stdio.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 MAXSIZE 100

#define MAXLENG 100

#define QMAXSIZE 100

typedef int status;

typedef struct BiTnode{

char name;

struct BiTnode \*lchild,\*rchild;

status data;

}BiTnode,\*BiTree;//二叉树定义，name是关键字，data存放节点数据

typedef BiTree QElemType;

typedef struct

{

QElemType \*base;

int front; // 头指针,若队列不空,指向队列头元素

int rear; //尾指针,若队列不空,指向队列尾元素的下一个位置

}SqQueue;

status PreCreate(BiTree \*T) {

//按先序次序输入二叉树中结点的值（一个字母），'#'字符表示空树，

//采用递归调用的方式构造二叉树链表表示的二叉树为T

char ch;

printf("请输入关键字：\n");

scanf("%c", &ch);

getchar();

if (ch == '#') {

\*T = NULL;

return ERROR;

}

else {

if (!((\*T) = (BiTnode\*)malloc(sizeof(BiTnode)))) exit(OVERFLOW);

(\*T)->name = ch;

printf("请输入结点值：\n");

scanf("%d",&((\*T)->data));

getchar();

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

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

return OK;

}

}

void CreateBiTree(BiTree \*T,int definition){

if(definition==1)

PreCreate(T);

}

status DestroyBiTree(BiTree \*T) {

if (\*T) {

if ((\*T)->lchild)

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

if ((\*T)->rchild)

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

free(\*T);

\*T = NULL;

}

return OK;

}//采用递归的方式销毁二叉树，思想和前序遍历二叉树是一致的

status ClearBiTree(BiTree \*T) {

if (\*T) {

(\*T)->lchild = NULL;

(\*T)->rchild = NULL;

return OK;

}

else return ERROR;

}//清空保留根节点

status BinTreeEmpty(BiTree \*T)

{

if(\*T == NULL)//空树是指根为空

return TRUE;

else

return FALSE;

}

status BiTreeDepth(BiTree \*T){

int depthL; // 左子树深度

int depthR; // 右子树深度

if((\*T)==NULL) return ERROR;

if ((\*T)->lchild==NULL&&(\*T)->rchild==NULL)

return 1;

depthL = BiTreeDepth(&(\*T)->lchild);

depthR = BiTreeDepth(&(\*T)->rchild);

if (depthL > depthR)

return 1 + depthL;

else

return 1 + depthR;

}

BiTree LocateNode(BiTree \*T,char e){

BiTree p,st[100];

int top=0;

st[top++]=\*T;

while(top){

p=st[--top];

if(p->name==e) return p;

else{

if(p->rchild!=NULL)

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

if(p->lchild!=NULL)

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

}

}

return NULL;

}

status Assign(BiTree \*T,char e,int \*value){

BiTree p;

p=LocateNode(T,e);

if(p!=NULL){

p->data=\*value;

return OK;

}

else return ERROR;

}

BiTree LeftSibling(BiTree T, char e) {

//若e无左孩子，返回NULL

BiTree T1=NULL;

if (T) {

if (T->rchild!=NULL&&T->rchild->name == e) return T->lchild;

T1 = LeftSibling(T->lchild, e); //递归调用对左子树进行操作

if (T1 != NULL) return T1;

T1 = LeftSibling(T->rchild, e); //递归调用对右子树进行操作

if (T1 != NULL) return T1;

}

return NULL;

}

BiTree RightSibling(BiTree T, char e) {

//若e无右孩子，返回NULL

BiTree T1 = NULL;

if (T) {

if (T->lchild!=NULL&&T->lchild->name== e) return T->rchild;

T1 = RightSibling(T->lchild, e); //递归调用对左子树进行操作

if (T1 != NULL) return T1;

T1 = RightSibling(T->rchild, e); //递归调用对右子树进行操作

if (T1 != NULL) return T1;

}

return NULL;

}

status InsertChild(BiTree \*T, char e, int LR, BiTree c) {

BiTree p = NULL;

p=LocateNode(T,e);

if (!(\*T)||!p)

return -1;

if (LR == 0) {

c->rchild=p->lchild;

p->lchild = c;

}//若LR为0，插入c为T中p所指结点的左子树

else {

c->rchild = p->rchild;

p->rchild = c;

}//若LR为1，插入c为T中p所指结点的右子树

return OK;

}

BiTree Parent(BiTree T, char e) {

//若e是T的非根结点，返回它的双亲结点指针

//否则返回NULL

BiTree T1;

if (T) {

if ((T->lchild!=NULL&&T->lchild->name == e) ||(T->rchild!=NULL&& T->rchild->name == e)) return T;

T1 = Parent(T->lchild, e);

if (T1 != NULL) return T1;

T1 = Parent(T->rchild, e);

if (T1 != NULL) return T1;

}

return NULL;

}

status DeleteNode(BiTree \*T,char e){

BiTree p,q,r;

p=LocateNode(T,e);

q=Parent(\*T,e);

if(\*T==NULL) return ERROR;

if(p==NULL) return ERROR;

if(p->lchild==NULL&&p->rchild==NULL){

if(q->lchild==p) q->lchild=NULL;

if(q->rchild==p) q->rchild=NULL;

free(p);

}

if((p->lchild!=NULL&&p->rchild==NULL)){

if(q->lchild==p) q->lchild=p->lchild;

free(p);

}

if(p->lchild==NULL&&p->rchild!=NULL){

if(q->rchild==p) q->rchild=p->rchild;

free(p);

}

if(p->lchild!=NULL&&p->rchild!=NULL){

if(q==NULL){

(\*T)->lchild->rchild=(\*T)->rchild;

(\*T)=(\*T)->lchild;

}

else if(q->lchild==p) {

r=p->lchild;

while(r->rchild!=NULL)

{

r=r->rchild;

}

r->rchild=p->rchild;

q->lchild=p->lchild;

}

else if(q->rchild==p) {

q->rchild=p->lchild;

q->rchild->rchild=p->rchild;

}

free(p);

}

return OK;

}

status Visit(BiTree T){

if(T==NULL) return ERROR;

printf("结点关键字：%c ",T->name);

printf("存放数据为:%d ",T->data);

return OK;

}

status Pre\_order(BiTree T, status (\*Visit)(BiTree T)) {

if (T) {

if (Visit(T))

if (Pre\_order(T->lchild, Visit))

if (Pre\_order(T->rchild, Visit)) return OK;

return ERROR;

}

else return OK;

}

status Mid\_order(BiTree T, status(\*Visit)(BiTree T)) {

if (T) {

if (Mid\_order(T->lchild, Visit))

if (Visit(T))

if (Mid\_order(T->rchild, Visit)) return OK;

return ERROR;

}

else return OK;

}

status Post\_order(BiTree T, status(\*Visit)(BiTree T)) {

if (T) {

if (Post\_order(T->lchild, Visit))

if (Post\_order(T->rchild, Visit))

if (Visit(T)) return OK;

return ERROR;

}

else return OK;

}

//遍历二叉树的非递归算法,建立递归工作栈

status Pre\_order\_b(BiTree T, status(\*Visit)(BiTree T)) {

BiTree st[MAXSIZE];

BiTree p;

int top = 0; //置空栈

if (T) {

st[top++] = T;

while (top)

{

p = st[--top]; //弹出栈顶指针

if(!Visit(p)) return ERROR;

if (p->rchild != NULL) st[top++] = p->rchild; //将右子树的根结点存入栈中

if (p->lchild != NULL) st[top++] = p->lchild; //将左子树的根结点存入栈中

}

}

return OK;

}//先序遍历的另一种算法思想是直接

//在第一次访问根结点时对根结点进行操作

status Mid\_order\_b(BiTree T, status(\*Visit)(BiTree T)) {

BiTree st[MAXLENG]; //定义指针栈，并假设二叉树的深度不超过10

int top = 0; //置空栈,top为栈顶元素上一个单元的地址

do {

while (T) //根指针T表示的为非空二叉树

{

if (top == MAXLENG) exit(OVERFLOW);

st[top++] = T; //根指针进栈

T = T->lchild; //T移向左子树

} //循环结束表示以栈顶元素的指向为

//根结点的二叉树的左子树遍历结束

if (top) //为非空栈

{

T = st[--top]; //弹出根指针

if(!Visit(T)) return ERROR; //对根结点进行操作

T = T->rchild;

}

} while (top || T); //父结点未访问或者右子树未遍历

return OK;

}

//用一个堆栈并设置标志来实现后续非递归遍历

status Post\_order\_b(BiTree T, status(\*Visit)(BiTree T)) {

BiTnode \*st[MAXLENG], \*pre;

int flag, top = 0;

if (T) {

do {

while (T) {

if (top == MAXLENG) exit(OVERFLOW);

st[top++] = T;

T = T->lchild;

}//循环结束时表示以栈顶元素为指向的

//根节点的二叉树的左子树遍历完毕

pre = NULL;

flag = 1;//设置标志位，用于结束内层循环

while (top&&flag)

{

T = st[top - 1]; //访问左子树根结点

if (T->rchild == pre) {

if (!Visit(T)) return ERROR;

top--;//弹出已访问的根结点

pre = T;

}

else

{

T = T->rchild;

flag = 0;//用于跳出内层循环

}

}

} while (top);

}

return OK;

}

status InitQueue(SqQueue \*Q)

{

Q->base = (QElemType \*)malloc(QMAXSIZE\*sizeof(QElemType));

if (!Q->base) /\* 存储分配失败 \*/

exit(OVERFLOW);

Q->front = Q->rear = 0;

return OK;

}

status QueueEmpty(SqQueue Q){ /\* 若队列Q为空队列,则返回TRUE,否则返回FALSE \*/

if (Q.front == Q.rear) /\* 队列空的标志 \*/

return TRUE;

else

return FALSE;

}

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

if ((Q->rear + 1) % QMAXSIZE == Q->front) /\* 队列满 \*/

return ERROR;

Q->base[Q->rear] = e;

Q->rear = (Q->rear + 1) % QMAXSIZE;

return OK;

}

status DeQueue(SqQueue \*Q, QElemType \*e)

{ /\* 若队列不空,则删除Q的队头元素,用e返回其值,并返回OK;否则返回ERROR \*/

if (Q->front == Q->rear) /\* 队列空 \*/

return ERROR;

\*e = Q->base[Q->front];

Q->front = (Q->front + 1) % QMAXSIZE;

return OK;

}

status LevelOrderTraverse(BiTree T, status(\*Visit)(BiTree T)){

//采用二叉链表存储结构，Visit是对数据元素操作的应用函数

//层序遍历二叉树T算法(利用队列)，对每个数据元素调用函数Visit

SqQueue Q;

BiTree P=T;

InitQueue(&Q);// 初始化队列

if (T) EnQueue(&Q, T);

while (Q.front != Q.rear)

{

DeQueue(&Q, &P);

Visit(P);

printf("\n");

if (P->lchild) EnQueue(&Q, P->lchild); // 若存在左孩子，左孩子进队列

if (P->rchild) EnQueue(&Q, P->rchild); // 若存在右孩子，右孩子进队列

}

return OK;

}

status Writefile(BiTree T){//将链表中的元素写入到文件

FILE \*fp;char filename[30];

printf("input file name: ");

scanf("%s",filename);

//写文件的方法

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

{

printf("File open error\n ");

return ERROR;

}

BiTree st[1000];

int rear=0,front=0,q;

st[rear++]=T;

while(rear!=front)

{

q=rear-1;

fprintf(fp,"%c: ",st[front]->name);

fprintf(fp,"%d ",st[front++]->data);

if(st[q]->lchild!=NULL) {

st[rear]=st[q]->lchild;

rear++;

}

if(st[q]->rchild!=NULL) {

st[rear]=st[q]->rchild;

rear++;

}

}

//fwrite针对二进制

//这里是1次性写入，对于其它物理结构，可通过遍历，逐个访问数据元素

//并写入到文件中。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 Readfile(BiTree \*T){

FILE \*fp;char filename[30];

printf("input file name: ");

scanf("%s",filename);

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

{

printf("File open error\n ");

return ERROR;

}

char name;

int data;

while(!feof(fp)){

name=fscanf(fp,"%c",&name);

if(name=='#') break;

else{

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

(\*T)->name=name;

data=fscanf(fp,"%d",&data);

}

}

return OK;

}

status Changetree(int \*N){//切换

int k;

printf("请输入要操作的链表编号：");

scanf("%d",&k);

if(k<0||k>99) return ERROR;

\*N=k;

return OK;

}

int main(void) {

BiTree T[100];

BiTree c;

int op = 1;

int LR=0,value=0;

int N;

char e;

printf("请输入0～99来确定要进行操作的树\n");

scanf("%d",&N);

while (op) {

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

printf(" Menu for Binary Tree on chain structure!\n");

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

printf(" 1. CreateBiTree 12.MidOrder\n");

printf(" 2. DestroyBiTree 13.PostOrder \n");

printf(" 3. ClearBiTree 14.LevelOrder \n");

printf(" 4. BiTreeEmpty 15.Writefile \n");

printf(" 5. BiTreeDepth 16. Readfile\n");

printf(" 6. LocateNode 17. Changetree\n");

printf(" 7. Assign \n");

printf(" 8. Brother \n");

printf(" 9. InsertChild \n");

printf(" 10.DeleteNode \n");

printf(" 11.PreOrder \n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~17]:");

scanf("%d", &op);

switch (op) {

case 1:getchar();

if (PreCreate(&T[N]) == OK) printf("树初始化成功！\n");

else printf("创建失败\n");

getchar(); getchar();

break;

case 2:getchar();

if(T[N]==NULL) printf("不存在可执行二叉树");

else if (DestroyBiTree(&T[N]) == OK) printf("树销毁成功！\n");

getchar(); getchar();

break;

case 3: getchar();

if(T[N]==NULL) printf("不存在可执行二叉树");

else if(ClearBiTree(&T[N])==OK) printf("清空成功");

else printf("清空失败");

getchar(); getchar();

break;

case 4:getchar();

if(T[N]==NULL) printf("不存在可执行二叉树");

else if(BinTreeEmpty(&T[N])==TRUE) printf("是空树");

else printf("不是空树");

getchar(); getchar();

break;

case 5:getchar();

if(BiTreeDepth(&T[N])==ERROR) printf("不存在可执行二叉树");

else printf("深度为%d",BiTreeDepth(&T[N]));

getchar(); getchar();

break;

case 6:getchar();

if(T[N]==NULL) printf("不存在可执行二叉树");

else{

printf("输入要查找的关键字：");

scanf("%c",&e);

if(LocateNode(&T[N],e)==NULL) printf("不存在关键字为%c的节点",e);

else printf("成功找到该节点,存储的数据为%d",LocateNode(&T[N],e)->data);

}

getchar(); getchar();

break;

case 7:getchar();

if(T[N]==NULL) printf("不存在可执行二叉树");

else{

printf("请输入要查找的关键字：");

scanf("%c",&e);

printf("请输入value的值：");

scanf("%d",&value);

if(Assign(&T[N],e,&value)==OK) printf("查找成功");

else printf("不存在关键字为%c的节点",e);

}

getchar(); getchar();

break;

case 8:getchar();

if(T[N]==NULL) printf("不存在可执行二叉树");

else{

printf("请输入要查找的关键字：");

scanf("%c",&e);

if(LeftSibling(T[N],e)!=NULL) printf("成功找到左兄弟,其关键字为%c\n",Parent(T[N],e)->lchild->name);

else if(RightSibling(T[N],e)!=NULL) printf("成功找到右兄弟,,其关键字为%c\n",Parent(T[N],e)->rchild->name);

else printf("该节点无兄弟");

}

getchar(); getchar();

break;

case 9:getchar();

if(T[N]==NULL) printf("不存在可执行二叉树");

else{

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

printf("输入节点c的关键字：\n");

scanf("%c",&(\*c).name);

printf("输入节点要存放的数据：\n");

scanf("%d",&(\*c).data);

printf("输入要查找的关键字:\n");

getchar();

scanf("%c",&e);

printf("输入LR：\n");

scanf("%d",&LR);

c->lchild=NULL;

c->rchild=NULL;

if(InsertChild(&T[N],e,LR,c)==OK) printf("已插入该节点");

else printf("不存在关键字为%c的结点\n",e);

}

getchar(); getchar();

break;

case 10:getchar();

if(T[N]==NULL) printf("不存在可执行二叉树");

else{

printf("请输入要查找的关键字：");

scanf("%c",&e);

if(DeleteNode(&T[N],e)==ERROR) printf("不存在该节点");

else printf("已删除该节点");

}

getchar(); getchar();

break;

case 11:getchar();

if(T[N]==NULL) {

printf("不存在可执行二叉树");

getchar(); getchar();

break;}

if(Pre\_order\_b(T[N],Visit)==OK) printf("\n前序遍历成功");

getchar(); getchar();

break;

case 12:getchar();

if(T[N]==NULL) {

printf("不存在可执行二叉树");

getchar(); getchar();

break;}

if(Mid\_order(T[N],Visit)==OK) printf("\n中序遍历成功");

getchar(); getchar();

break;

case 13:getchar();

if(T[N]==NULL) {

printf("不存在可执行二叉树");

getchar(); getchar();

break;}

if(Post\_order(T[N],Visit)==OK) printf("\n后序遍历成功");

getchar(); getchar();

break;

case 14:getchar();

if(T[N]==NULL) printf("不存在可执行二叉树");

if(LevelOrderTraverse(T[N],Visit)==ERROR) printf("遍历失败");

else printf("遍历成功");

getchar(); getchar();

break;

case 15:getchar();

if(Writefile(T[N])==OK) printf("写入成功\n");

else printf("写入失败\n");

getchar();getchar();

break;

case 16:getchar();

if(Readfile(&T[N])==OK) printf("写入成功\n");

else printf("写入失败\n");

getchar();getchar();

break;

case 17:getchar();

if(Changetree(&N)==OK) printf("切换成功，此时二叉树编号为%d\n",N);

else printf("输入不合法\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");

return 0;

}//end of main()

# 附录D 基于邻接表图实现的源程序