

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期： 2018年1月11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

1.1 问题描述 2

1.2 系统设计 5

1.3 系统实现 14

1.4 实验小结 29

2 基于链式存储结构的线性表实现 30

2.1 问题描述 30

2.2 系统设计 33

2.3 系统实现 42

2.4 实验小结 55

3 基于二叉链表的二叉树实现 56

3.1 问题描述 56

3.2 系统设计 60

3.3 系统实现 73

3.4 实验小结 92

4 基于邻接表的图实现 94

4.1问题描述 94

4.2 系统设计 97

4.3 系统实现 105

4.4 实验小结 122

参考文献 123

指导教师评定意见 124

附录A 基于顺序存储结构线性表实现的源程序 125

附录B 基于链式存储结构线性表实现的源程序 142

附录C 基于二叉链表二叉树实现的源程序 158

附录D 基于邻接表图实现的源程序 187

# 基于顺序存储结构的线性表实现

## 1.1 问题描述

采用顺序表作为线性表的物理结构，并以函数形式定义实现线性表的初始化表、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12种基本运算，且要求构造一个具有菜单的功能演示系统，在主程序中完成函数调用所需实参值的准备和函数执行结果。

1.1.1 线性表抽象数据类型

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

ADT SqList{

**数据对象:**D = { ai|ai ∈ ElemSet, i = 1,2, ……,n, n≥0 }

**数据关系:**R1 = { <ai-1,ai>|ai-1,ai∈D, i = 2,……,n }

**基本操作:**

IntiaList(&L)

初始条件:线性表L不存在.

操作结果:构造一个空的线性表L，成功构造返回OK，否则返回ERROR。

DestroyList(&L)

初始条件:线性表L已存在。

操作结果:销毁线性表L，当表尚未被创建时返回ERROR,否则销毁成

功返回OK。

ClearList(&L)

初始条件:线性表L已存在。

操作结果:将L重置为空表，成功置空返回OK，否则返回ERROR。

ListEmpty(L)

初始条件:线性表L已存在。

操作结果: 若L为空表则返回OK，否则返回ERROR

ListLength(L)

初始条件:线性表L已存在。

操作结果: 成功返回L中数据元素个数，失败返回ERROR。

GetElem(L,I,&e)

初始条件:线性表L已存在、非空且1≤i≤ListLength(L)。

操作结果: 找到L中第i个元素的值，成功则用e返回第i个元素的值，否则返回ERROR。

LocateElem(L,e)

初始条件:线性表L已存在且非空。

操作结果: 成功则返回第一个与e相同的数据的位序，不存在则返回0，其余情况返回ERROR。

PriorElem(L,cur,&pre\_e)

初始条件:线性表L已存在、非空且cur\_e不是第一个数据元素。

操作结果: 成功则返回与cur相同的第一个数据，并返回其第一个前驱；若无前驱以及其他情况则返回ERROR。

NextElem(L,cur,&next\_e)

初始条件:线性表L存在、非空且cur\_e不是最后一个数据元素。

操作结果: 成功则返回与cur\_e相同的第一个数据，并返回其后驱；若无后驱以及其他情况则返回ERROR。

ListInsert(&L,i,e)

初始条件:线性表L已存在且非空，1≤i≤ListLength(L)+1。

操作结果:在L的第i个位置之前插入新的数据元素e，成功插入返回OK，否则返回ERROR。

ListDelete(&L,i,e)

初始条件: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，成功删除返回OK，否则返回ERROR。

ListTraverse(L)

初始条件:线性表L已存在。

操作结果: 成功遍历返回线性表的长度，否则返回ERROR。

} ADT SqList

1.1.2 多线性表抽象数据类型

基于之前实现的线性表数据类型，设计了多线性表的数据对象、数据关系，并以定义了多线性表的初始化、销毁、遍历、切换共4种基本运算，具体数据和运算功能定义如下。

ADT MultiList{

**数据对象**:D = { ei|ei ∈ SqList, i = 1,2, ……,n, n≥0 }

**数据关系**:R1 = { <ei-1,ei>|ei-1,ei∈D, i = 2,……,n }

**基本操作**:

InitiaList2 (&head, &List)

初始条件:无

操作结果: 构造一个空的线性表,并使List只想该表,成功构造返回OK，否则返回ERROR。

DestroyList2(&head, & List)

初始条件:表MultiList非空。

操作结果: 成功销毁返回OK，否则返回ERROR.

MultiListTrabverse (&head)

初始条件:表MultiList非空。

操作结果:遍历MList中的所有线性表，成功遍历返回线性表的长度，否则返回ERROR.

ExList(&head)

初始条件:无。

操作结果: 进行多表切换操作, 一个指向切换后的活动表的指针.

} ADT MulList

1.1.3 演示系统与数据文件

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。

演示系统可同时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。

演示系统选择实现多个线性表管理。

## 1.2 系统设计

1.2.1 数据物理结构

在本程序中，数据原子类型ElemType被定义为int整型。

1.线性表的数据存储结构

线性表数据类型在程序中定义为结构体SqList，具体定义方式如下：

struct SqList{

ElemType \* elem;//指向线性表数据的线性存储区

int length;//表中元素的个数

int listsize;//表的最大元素个数

};

2.多线性表的数据存储结构

多线性表数据类型在程序中定义为结构体MultiList，具体定义方式如下：

struct MultiList {

SqList \*L; /\*指向一个线性表\*/

int num; /\*自动生成的表的编码\*/

struct MultiList \*next; /\*指向下一个表节点\*/

};

多线性表的存储结构如图1-1所示：

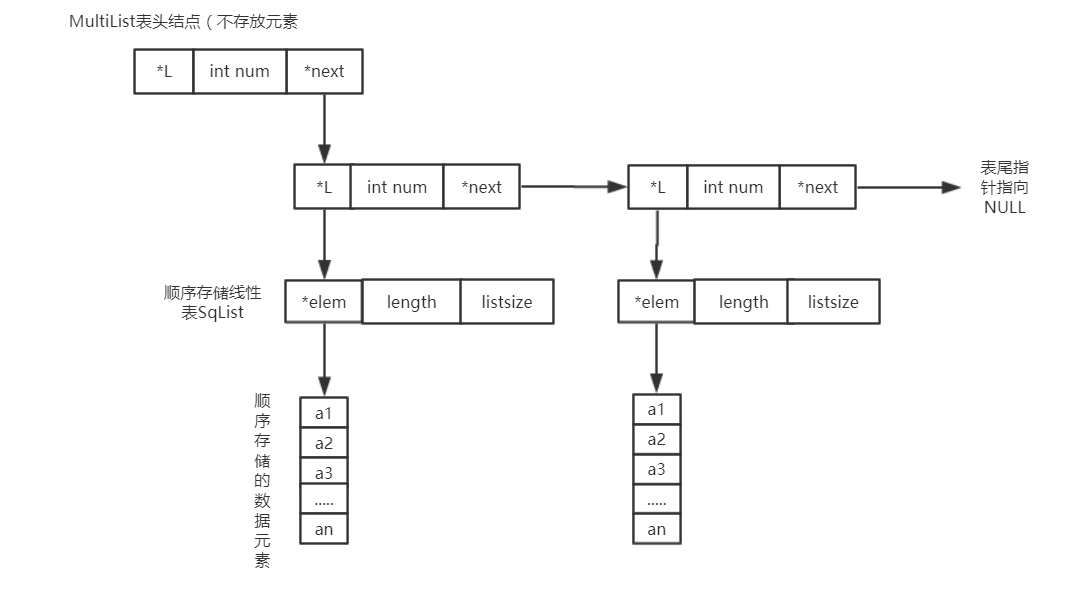


图1-1 多线性表的存储结构

1.2.2 演示系统

演示系统由用户操作界面与功能调用部分组成。

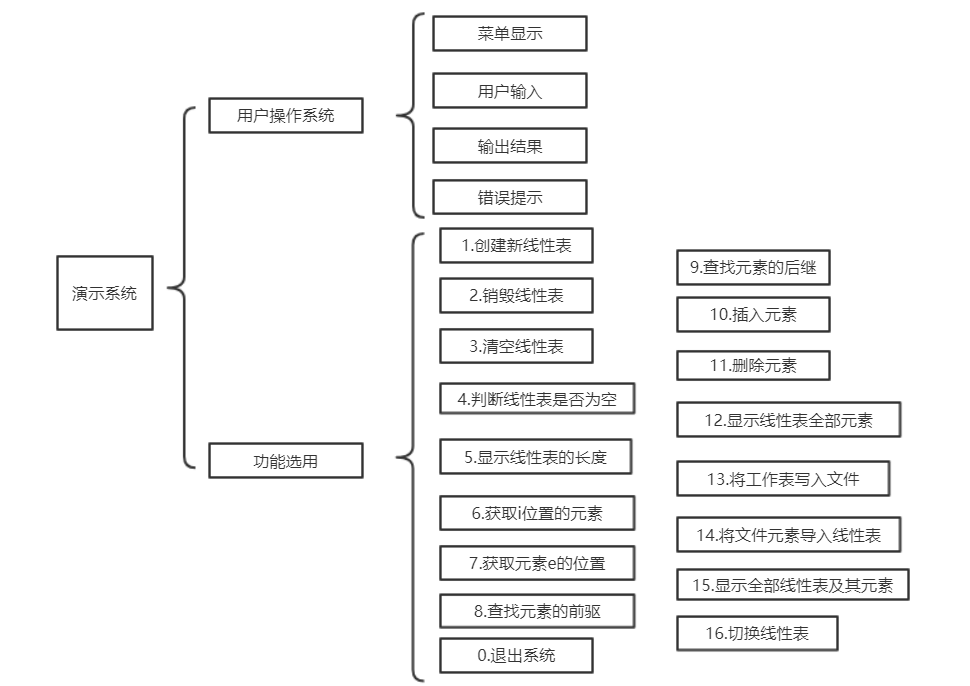


图1-2 演示系统模块结构图

1.2.3 数据文件存储格式

考虑到方便查看,以txt格式直接存放元素的值,相邻元素用空格隔开(其中第一个元素为该文件中元素的个数,即导入后的线性表表长)。

1.2.4 线性表运算实现算法

1.IntiaList(&L)

**算法思想：**首先判断L->elem是否为空指针(即L是否已初始化)，若不是则返回ERROR；否则使用malloc函数分配LIST\_INIT\_SIZE个ElemType类型的存储空间并使L->elem指向这一段空间，若分配失败则返回ERROR，否则L->length

=0,L->listsize=LIST\_INIT\_SIZE，返回OK。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

2.DestroyList(&L)

**算法思想：**首先判断L->elem是否为空指针(即L是否存在)，若是则返回ERROR；否则使用free函数释放L->elem指向的存储空间，释放L所指向的存储空间。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

3.ClearList(&L)

**算法思想：**首先判断L->elem是否为空指针(即L是否存在)，若是则返回ERROR；否则使用free函数释放L->elem指向的存储空间，然后使L->length = 0,成功则返回OK，否则返回ERROR。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

4.ListEmpty(L)

**算法思想：**首先判断L->elem是否为空指针(即L是否存在)，若是则返回

ERROR；然后判断若L.length=0，则输出L为空表且返回OK，否则输出L不为空表并返回ERROR。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

5.ListLength(L)

**算法思想：**首先判断L->elem是否为空指针(即L是否存在)，若是则返回

L.length，否则返回ERROR。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

6.GetElem(L,I,&e)

**算法思想：**首先判断L.length是否等于0(即L是否为空表)，若是则提示表

为空表并返回ERROR，然后判断i的输入是否合法（即是否符合1<=i<=L.length）,若不是则提示i的数值不合法，返回ERROR，否则然后将L.elem中第i-1位的元素送入e所指的存储单元中，返回OK。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

7.LocateElem(L,e)

**算法思想：**首先判断L.length是否等于0(即L是否为空表)，若是则

返回ERROR，然后遍历线性表L，当找到与e相同的数值或遍历结束时时停止，判断i的数值此刻是否小于L.length（即是否结束了线性表），若不符合则返回ERROR,否则返回i+1。

**时间复杂度分析：**假设表中有n个元素，且第一个与e相同的元素在表每个位置的概率相等（均为1/n），则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部变量i，算法的空间复杂度为常量阶O(1)。

8.PriorElem(L,cur,&pre\_e)

**算法思想：**算法流程图如下图1-3所示。

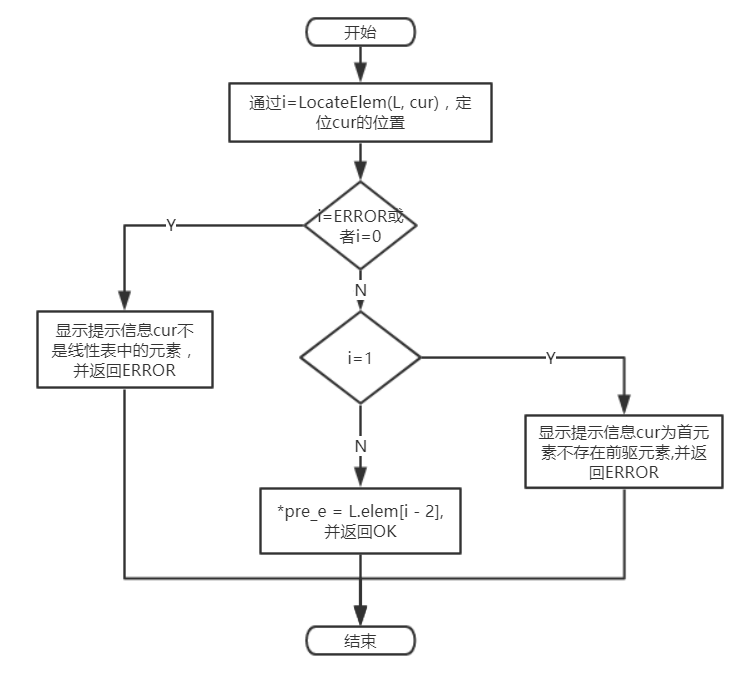


图1-3 PriorElem算法流程图

**时间复杂度分析：**假设表中有n个元素，且第一个与e相等的元素在表每个位置的概率相等（均为1/(n-1)），则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部变量i，算法的空间复杂度为常量阶O(1)。

9.NextElem(L,cur,&next\_e)

**算法思想：**算法流程图如下图1-4所示。

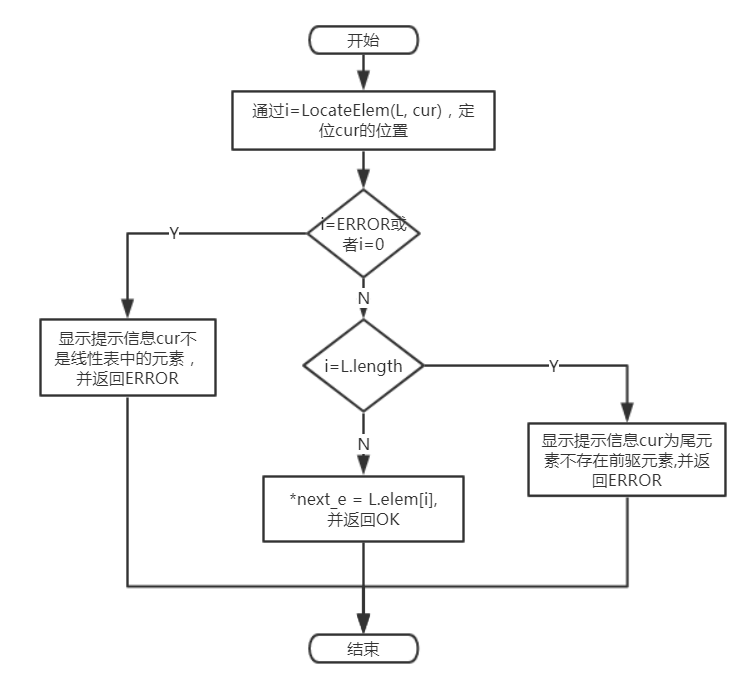


图1-4 NextElem算法流程图

**时间复杂度分析：**假设表中有n个元素，且第一个与e相等的元素在表每个位置的概率相等（均为1/(n-1)），则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部变量i，算法的空间复杂度为常量阶O(1)。

10.ListInsert(&L,i,e)

**算法思想：**算法流程图如下图1-5所示。

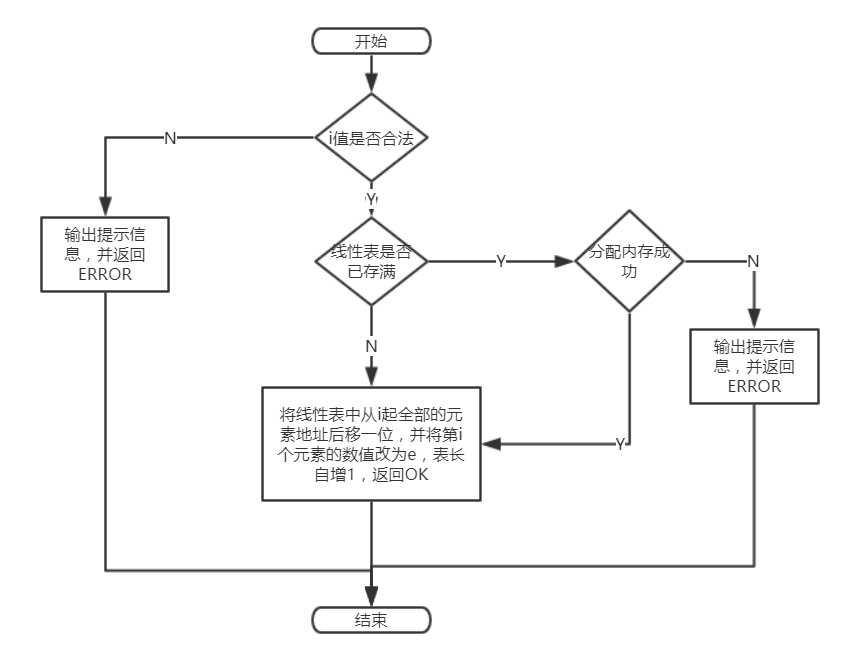


图1-5 ListInsert算法流程图

**时间复杂度分析：**假设表中有n个元素，且在每个位置插入的可能性相等(1/n+1)，则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

11.ListDelete(&L,i,e)

**算法思想：**算法流程图如下图1-6所示。

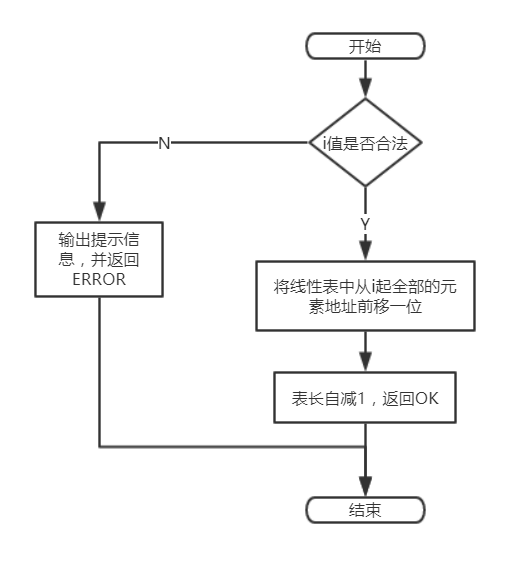


图1-6 ListDelete算法流程图

**时间复杂度分析：**删除算法的时间耗费主要在移动元素上，循环执行的次数L->length-i+1不定，假设表中有n个元素，且在每个位置删除的可能性相等(1/n+1)，则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

12.ListTraverse(L)

**算法思想：**从一号元素起输出表中元素，若是空表则输出空白并在最后显示提示信息，并返回表长L.length。

**时间复杂度分析：**假设表中有n个元素，则算法时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

13.SaveList(fp,L,filename)

**算法思想：**先输入文件名字，用fopen打开文件并返回文件指针fp，若打开失败则显示提示信息并返回ERROR，否则用fprintf依次写入线性表的表长以及各个元素，最后关闭fp指针并返回OK。

**时间复杂度分析：**假设表中有n个元素，则算法时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部变量i，算法的空间复杂度为常量阶O(1)。

14.LoadList(fp,L,filename)

**算法思想：**先输入文件名字，用fopen打开文件并返回文件指针fp，若打开失败则显示提示信息并返回ERROR，否则用fscanf先读入第一个元素，即线性表的表长，然后以表长为条件循环读入之后的每一个元素，并在读入每一个元素后调用ListInsert函数来进行插入，最后关闭fp指针并返回OK。

**时间复杂度分析：**假设表中有n个元素，则算法时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部变量i、e1、e，算法的空间复杂度为常量阶O(1)。

1.2.5 多线性表运算实现算法

1.InitiaList2 (&head, &List)

**算法思想：**首先使用指针q遍历至多线性表结构的最后一个元素，创建一个新的MultiList节点并使q->next指向它，调用IntiaList函数初始化新节点中的线性表，若初始化失败则返回ERROR,否则使新节点next指针指向空，返回OK。

**时间复杂度分析：**假设多线性表结构的总长度为n，则本算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

2.DestroyList2(&head, & List)

**算法思想：**先使指针p遍历至List所指的元素，q指向List的前一个元素，然后调用DestroyList函数销毁p->L,然后释放掉p的空间并使List指向空，返回OK。

**时间复杂度分析：**假设多线性表结构总长度为n，在每个表执行删除操作的概率相等(1/n)，则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

3.MultiListTrabverse (&head)

**算法思想：**从第一个多表节点开始先输出线性表的编号和表长（编号为自动生成的，由线性表在多表中的位置决定），依次使用ListTraverse函数对节点的线性表成员调用函数进行遍历，最后返回OK。

**时间复杂度分析：**假设有n个多表结构，每个节点下的线性表有ai(i=1,2,……,n)个元素，则算法的时间复杂度O(n)=a1+a2+…+an。



**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

4.ExList(&head)

**算法思想：**先调用MultiListTrabverse函数进行遍历，此时每一个表都对应一个编码，输入想要操作的表的编码，通过再次遍历找到该编码对应的含有所要线性表的结点，使指针List指向该节点，若List为空，则说明未找到编码所对应的线性表，显示提示信息，若不为空则返回List。

**时间复杂度分析：**假设有n个多表结构，且切换为每个表的操作概率相等(1/n)，则算法的时间复杂度为平方阶O(n2)。

**空间复杂度分析：**引入局部变量i，算法的空间复杂度为常量阶O(1)。

## 1.3 系统实现

1.3.1 实验环境

实验代码使用dev c++编写与编译.

1.3.2 演示系统操作

演示系统采用键盘操作，控制台输出操作提示与结果，第一行显示当前操作表（默认为无），中间显示可用的操作项，输入对应的操作指令即可执行，输入非0~16的操作号会提示错误。

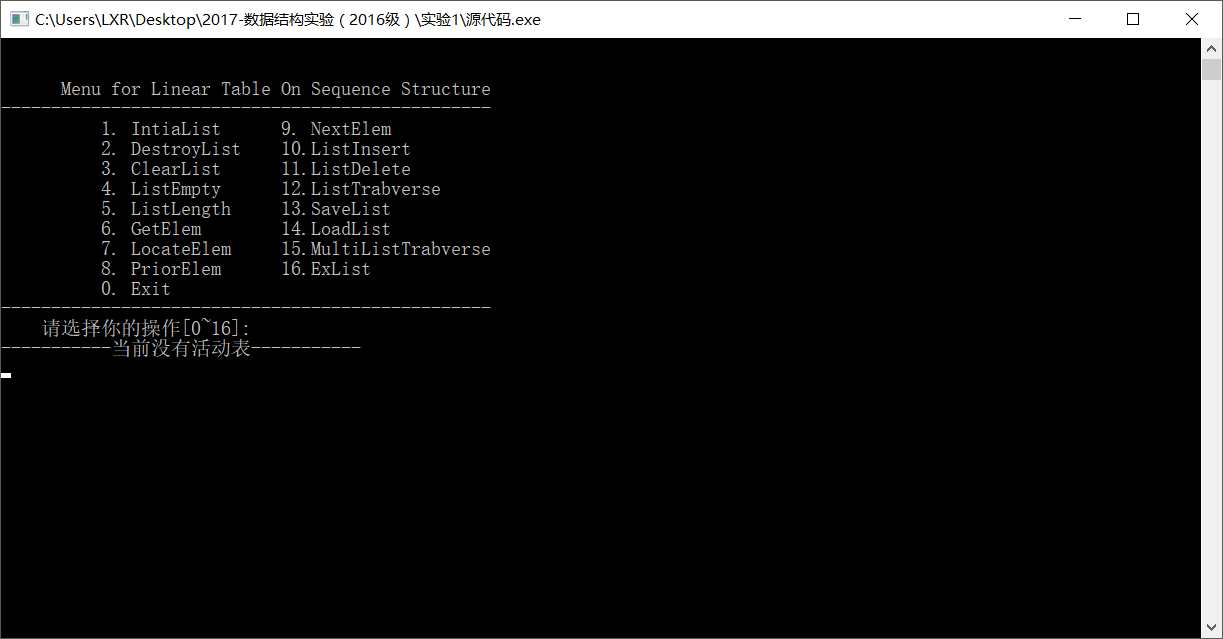


图1-7 启动程序后的操作菜单

输入操作指令0，程序退出

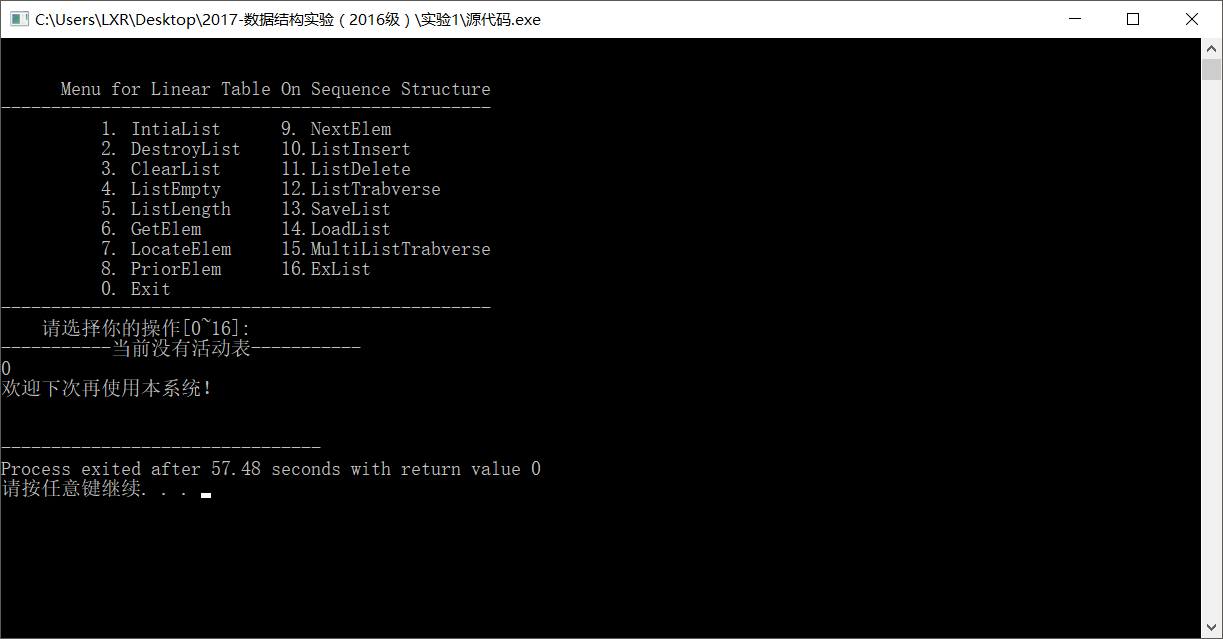


图1-8 输入操作0程序退出

1.3.3 测试计划

1.检测常规状态下系统的工作情况；

2.重点检查非法边界操作时的工作情况。

表1-1 演示系统测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 待测试功能 | 测试顺序 | 测试输入 | 预计输出 | 线性表状态 |
| IntiaList | 6  38  41 | 无 | 线性表创建成功！ | 6/38：表长0；编码1；  41：表长0，编码2 |
| DestroyList | 1  37 | 无 | 1：当前没有活动表  37：线性表销毁成功！ | 不存在 |
| ClearList | 2  33 | 无 | 2：当前没有活动表  33：线性表重置成功！ | 2：同1  33：表长0，编码1 |
| ListEmpty | 3  7  34 | 无 | 该表为空表； | 3：同1  7：同6  34：同33 |
| ListLength | 8  35 | 无 | 线性表长度为0 | 8：同6  35：同33 |
| GetElem | 9  22  23 | 0  2  0 | 9：该表为空表，查找失败；  22：位序为2的元素为5  23：i值不合法 | 9：同6  22-23：同20 |
| LocateElem | 10  24  25 | 10：1  24：4  15：0 | 10：不存在相同的元素  24：第1个元素与e相同  25：不存在与e相同的元素 | 10：同6  24-26：同20 |
| PriorElem | 11  26  27  28 | 1  4  5  6 | 1不是活动表中的元素  4为首元素没有前驱元素  5的前驱元素为4  6不是活动表中的元素 | 11：同6  26-28：同20 |
| NextElem | 29  30  31 | 4  5  6 | 4的后继元素为5  5为尾元素没有后继元素  6不是活动表中的元素 | 29-31：同20 |
| ListInsert | 13  14  15  16  17 | 3 1  4 2  5 3  6 0  6 6 | 13-15 无  16-17 i值不合法，插入失败 | 表中元素为3 4 5  表长为3  编码1 |
| ListDelete | 18  19  20 | 0  4  1 | 18-19删除元素失败  20 删除成功 | 18-19：同17  20：表中元素为4 5  表长为2  编码1 |
| ListTraverse | 12  21  36  40  44 | 无 | 12/36：输出空白加上提示信息  21/40/48：输出4 5 | 12：同6  21/40/48：同20  36：同34 |
| SaveList | 4  32 | data.txt | 4：无法读取  32：文件写入成功 | 4：同1  32：同20 |
| LoadList | 5  39 | data.txt | 5：无法读取  39：录入成功 | 5：同1  39：同20 |
| MultiListTrabverse | 42 | 无 | 输出各线性表 | 表1：元素4 5表长2  表2：空表  此时工作表为表2 |
| ExList | 43 | 1 | 切换为表1 | 表1：元素4 5表长2  表2：空表  此时工作表为表1 |

输入操作指令0，程序退出.

1.3.4 测试结果

测试用例一：

未初始化线性表时，操作对应测试计划1-5。

测试结果：（在进行其他操作时同样提示当前没有活动表）

1. 销毁线性表。

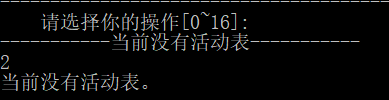


图1-9 销毁线性表

1. 清空线性表。



图1-10 清空线性表

1. 判断线性表是否为空表。

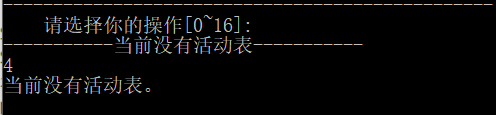


图1-11 判断线性表是否为空表

1. 将线性表内容写入文件。

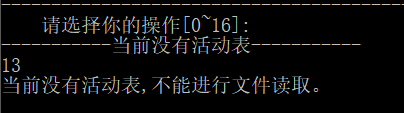


图1-12 文件写入

1. 将文件内容读入线性表。

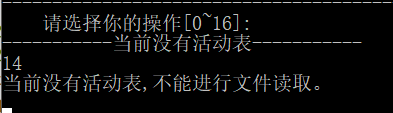


图1-13 文件读取

测试用例二：

空表，对应测试计划6-12

测试结果：

1. 初始化一个新表，并自动生成该表的编码1.

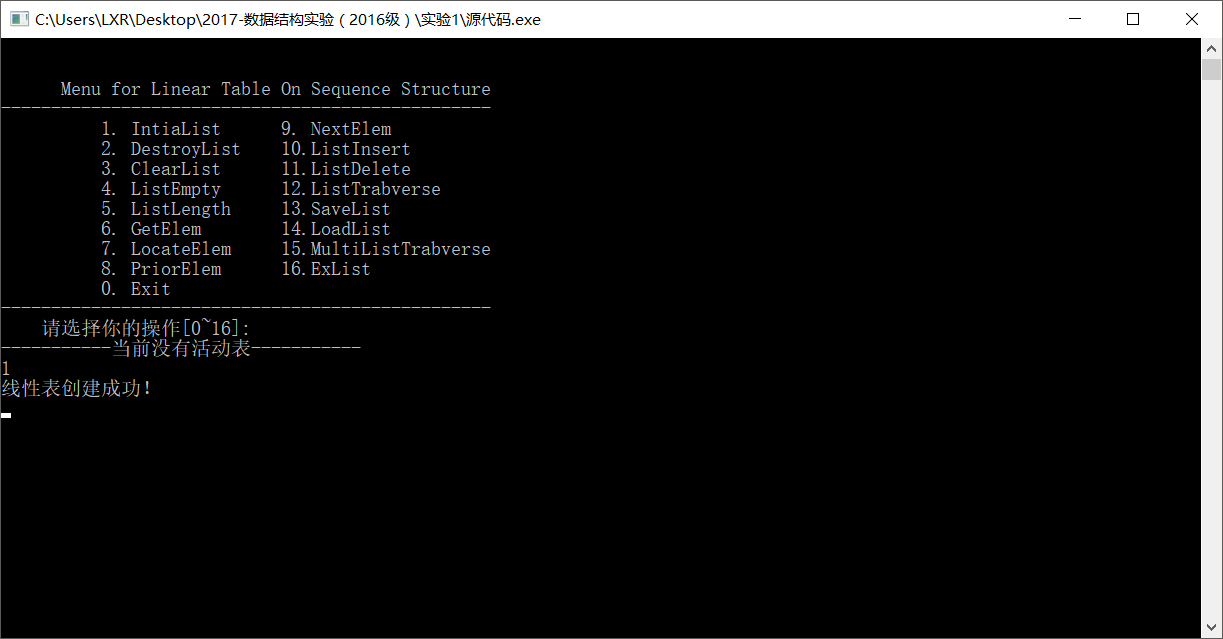


图1-14 初始化表

1. 判断表为空表。

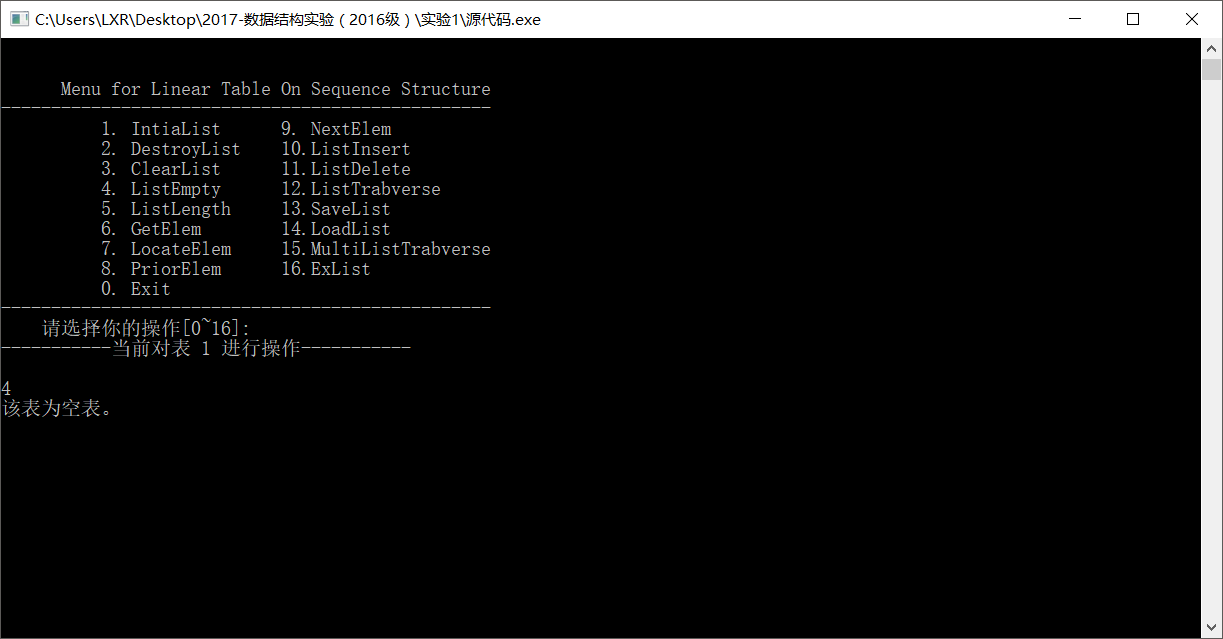


图1-15 判断新建的表是否是空表

1. 输出表长。

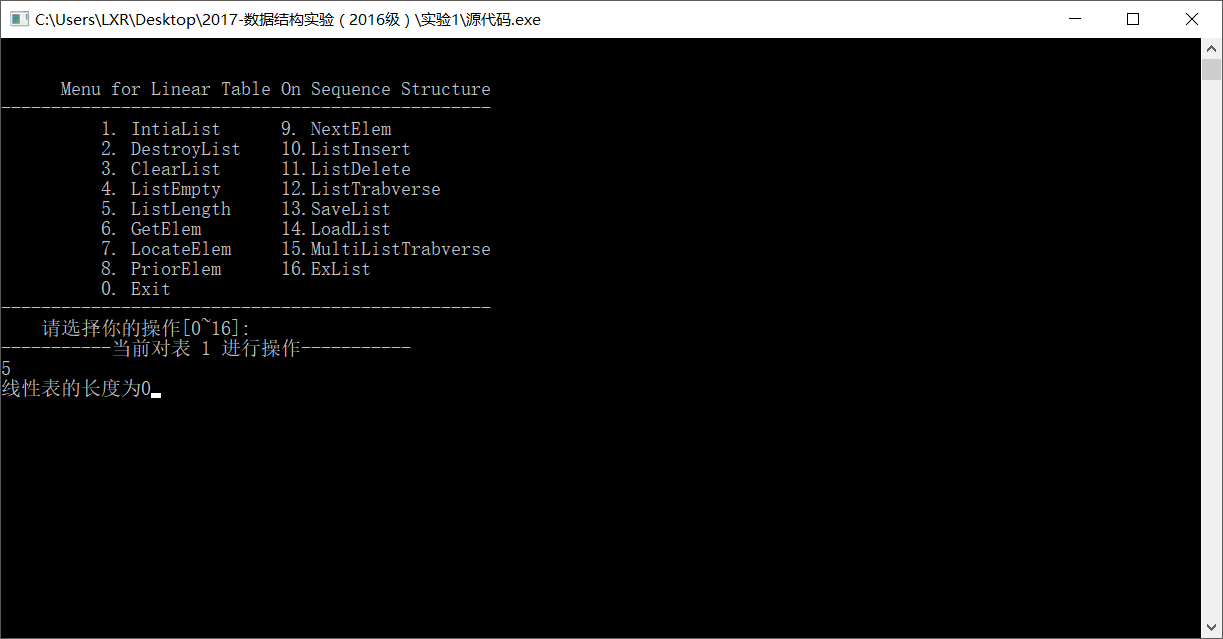


图1-16 输出表长

1. 查找某一元素的位序。

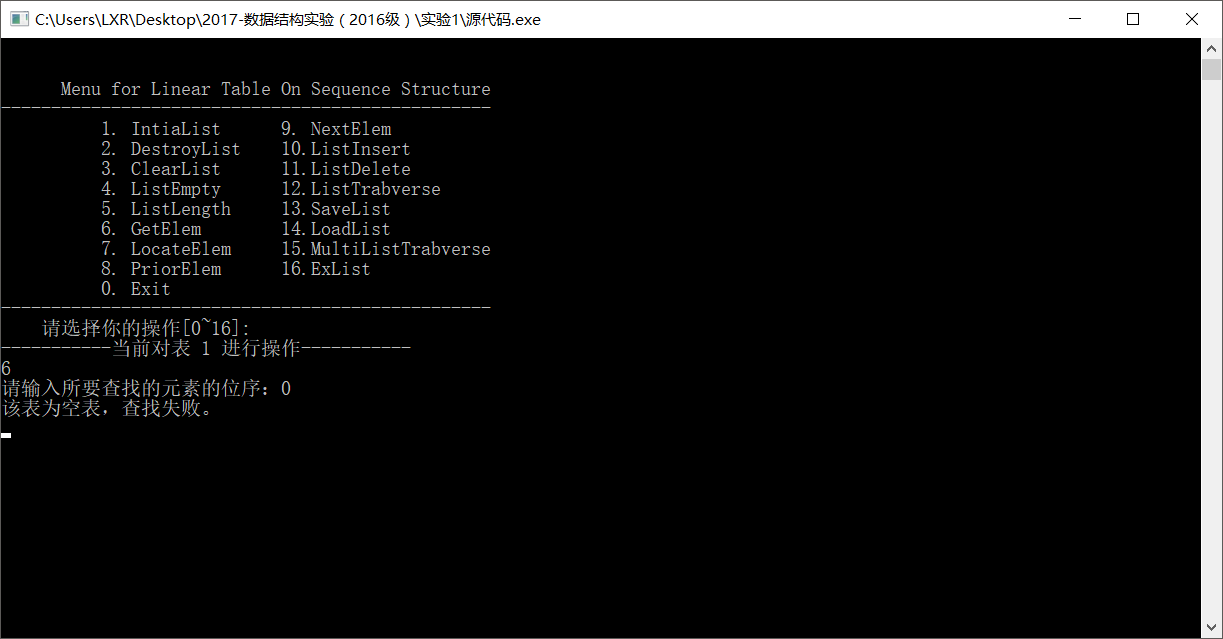


图1-17 查找位序

1. 定位元素位置。

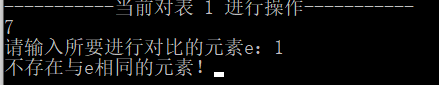


图1-16 定位元素位置

1. 查找某一元素的前驱元素。

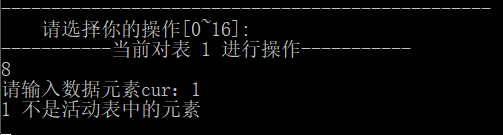


图1-17 查找前驱元素

1. 遍历该线性表。

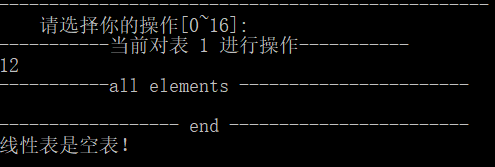


图1-18 遍历线性表

测试用例三：

先输入元素3、4、5并对线性表进行操作，对应测试计划13-37。

测试结果：

1. 依次输入3、4、5。

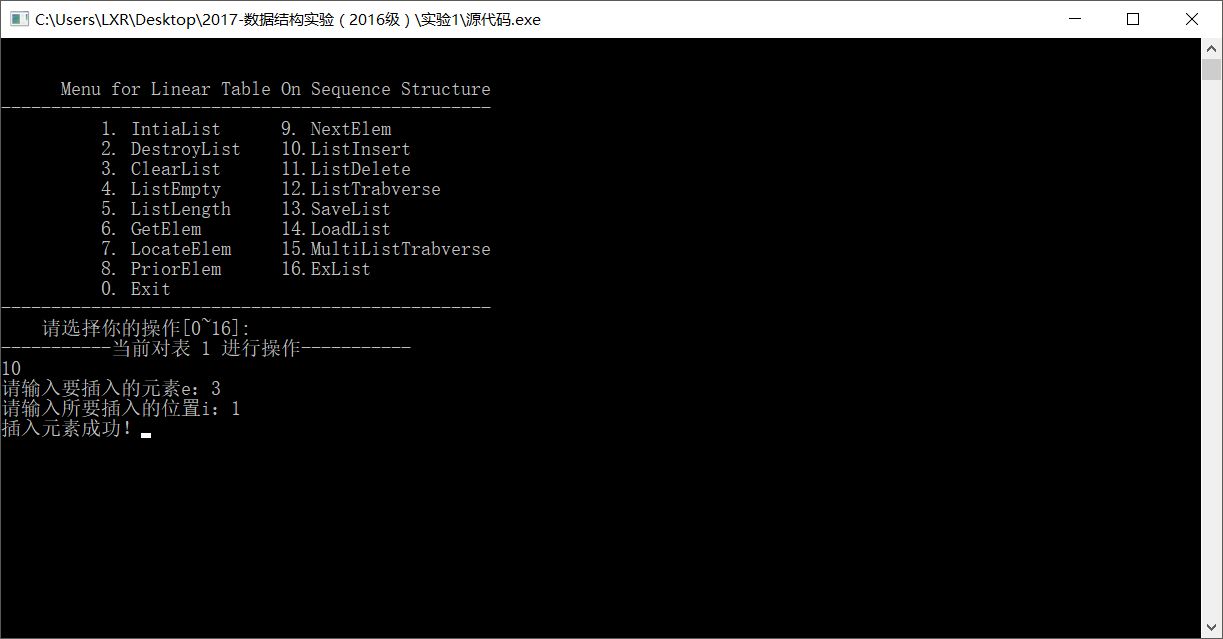


图1-19 插入元素3、插入位置1

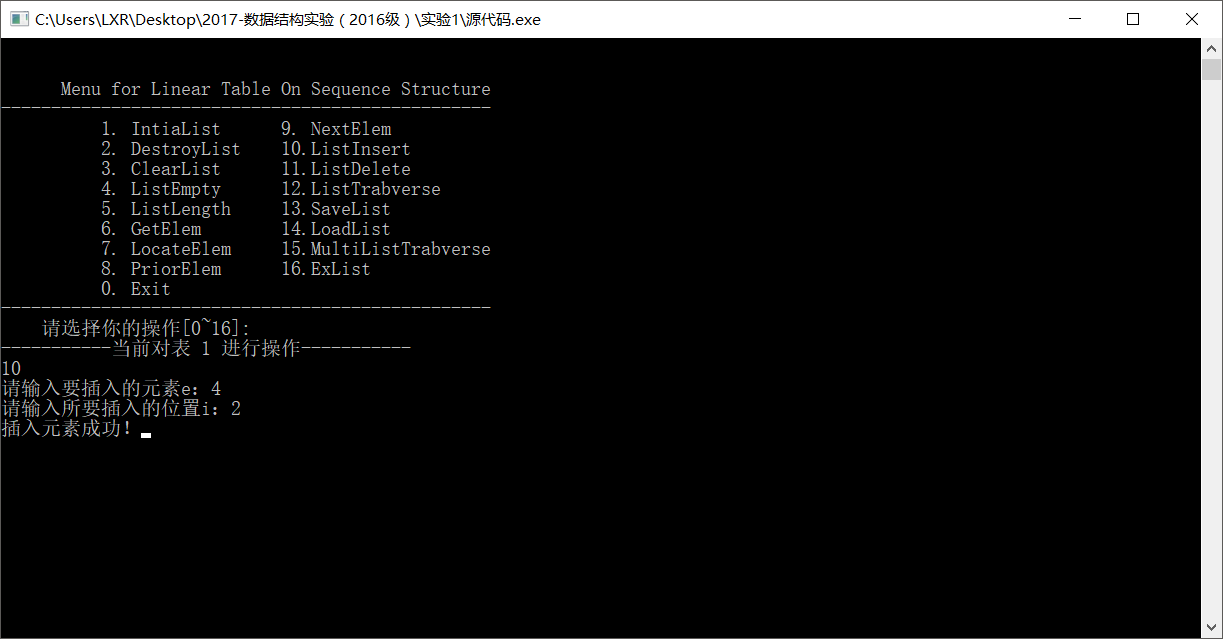


图1-20 插入元素4、插入位置2

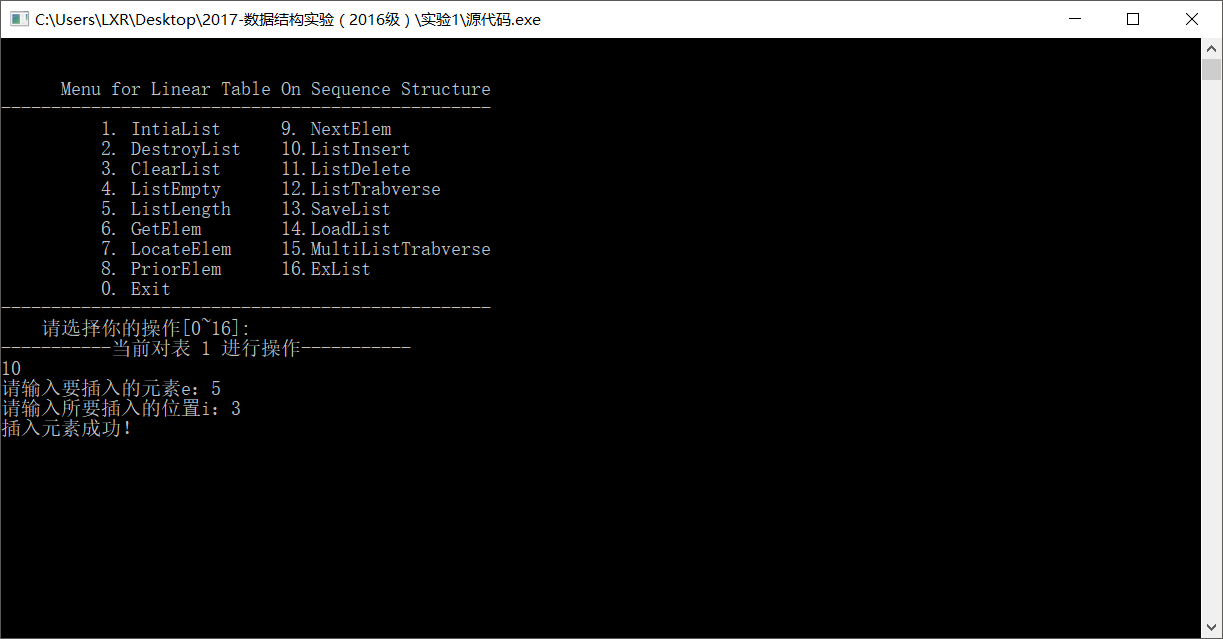


图1-21 插入元素5、插入位置3

1. 测试边界值插入，如位置为0或者6，则显示提示信息。

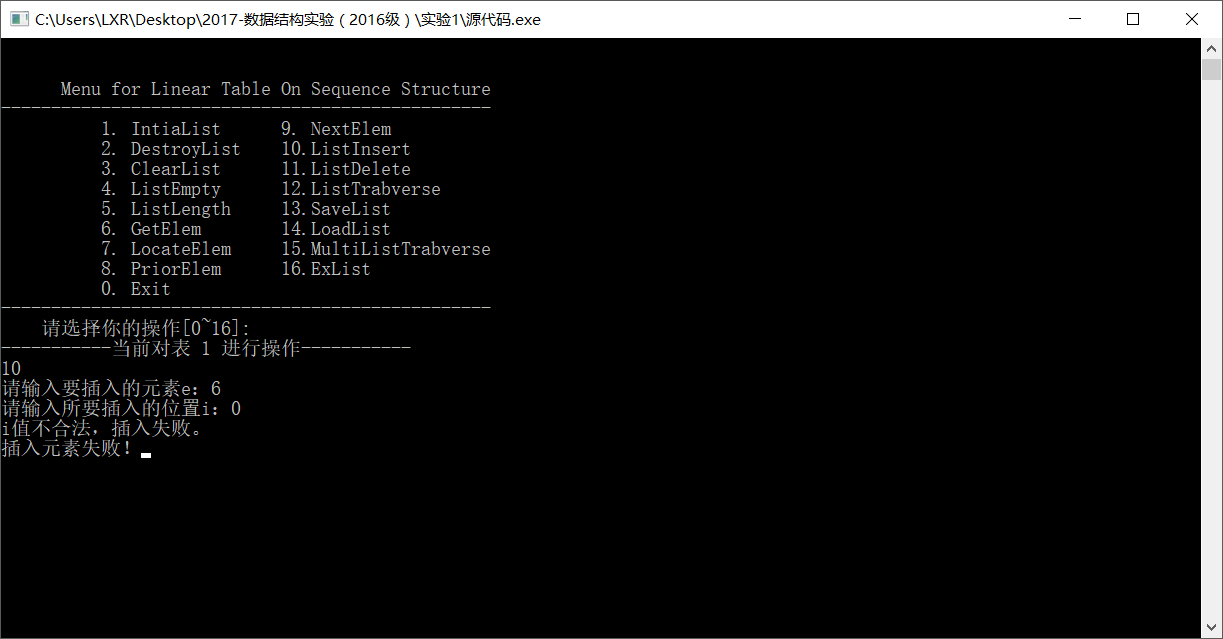


图1-22 若插入位置输入0，则插入失败

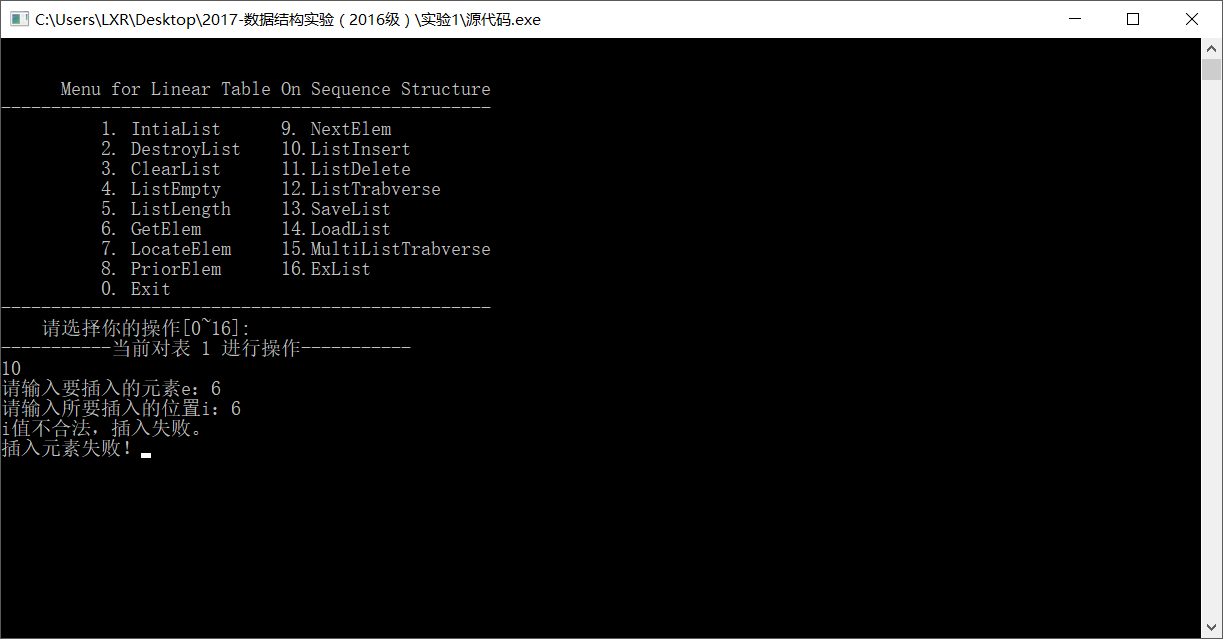


图1-23 输入一个比表长长2的i值，插入失败

（3）依此输入删除位序0、4、1，最后输出当前线性表的元素，用以验证删除成功。

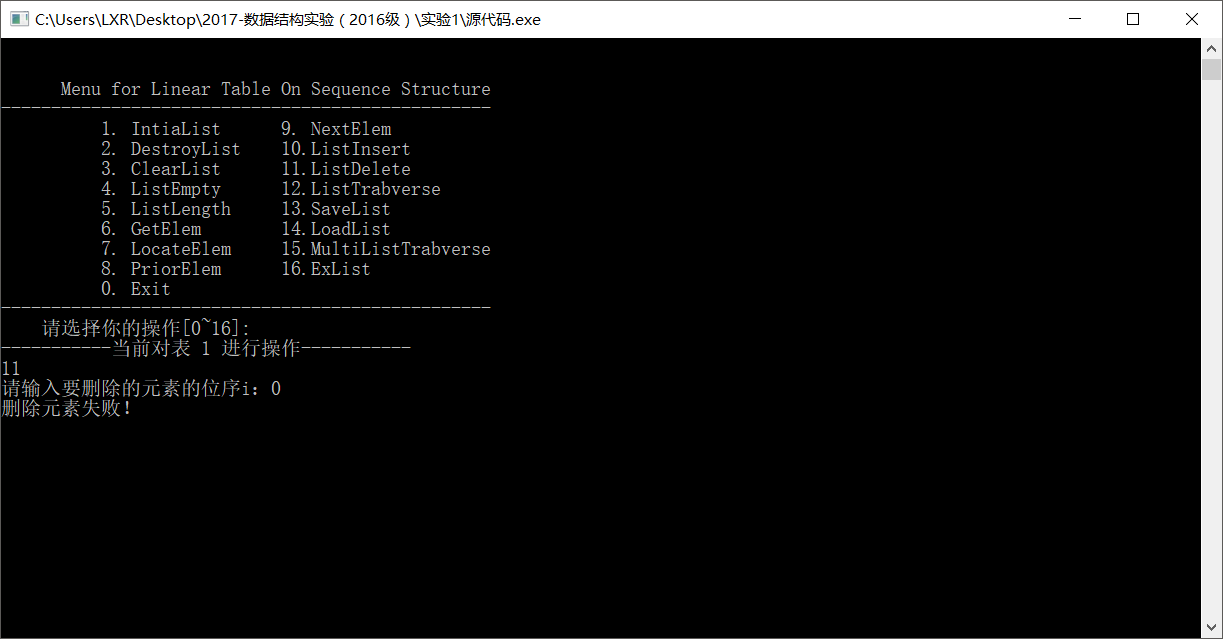


图1-24 输入删除位序0，则提示删除失败

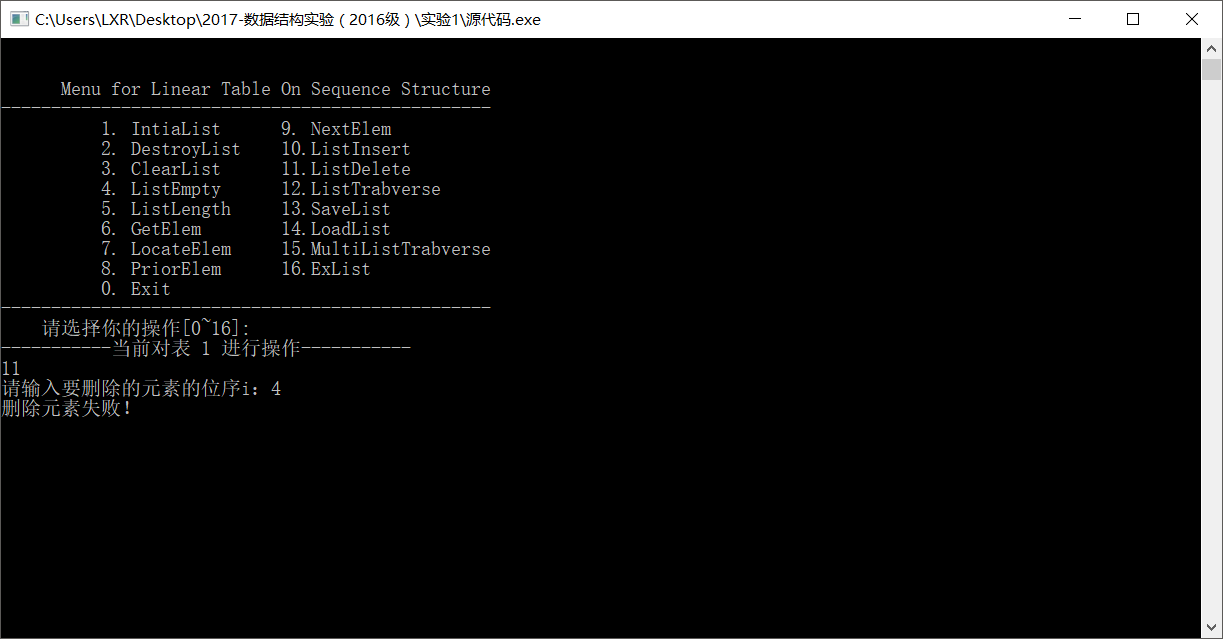


图1-25 输入删除位序4（即比表长大的位序），则提示删除失败

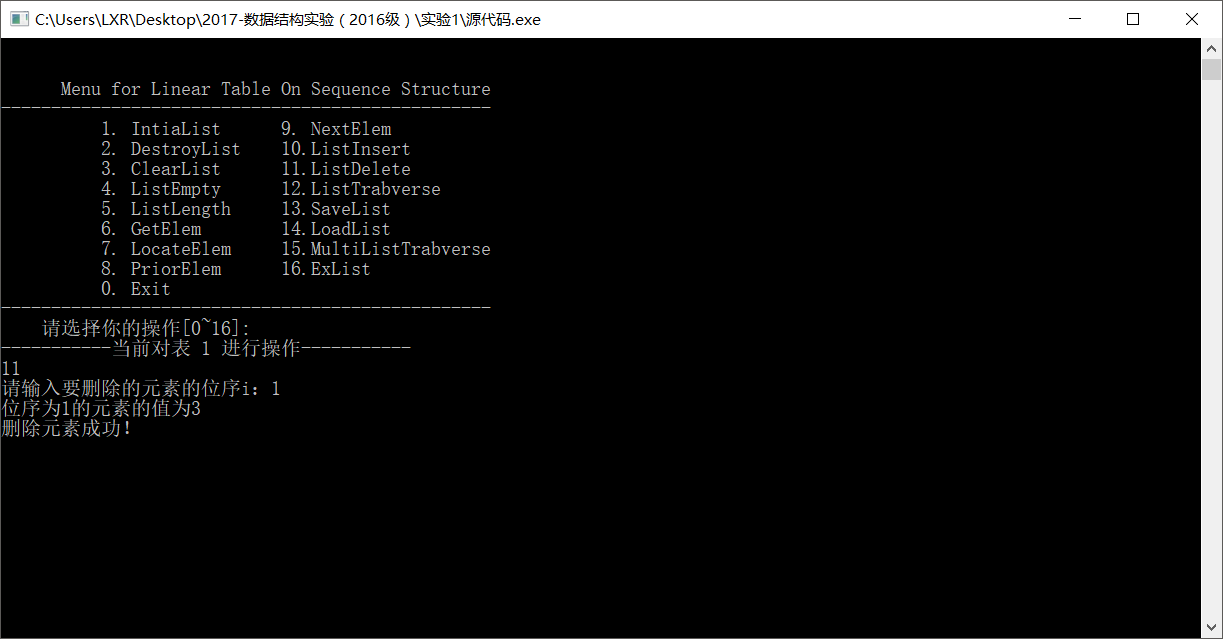


图1-26 输入删除位序1，输出原表位序1出的元素并提示删除成功

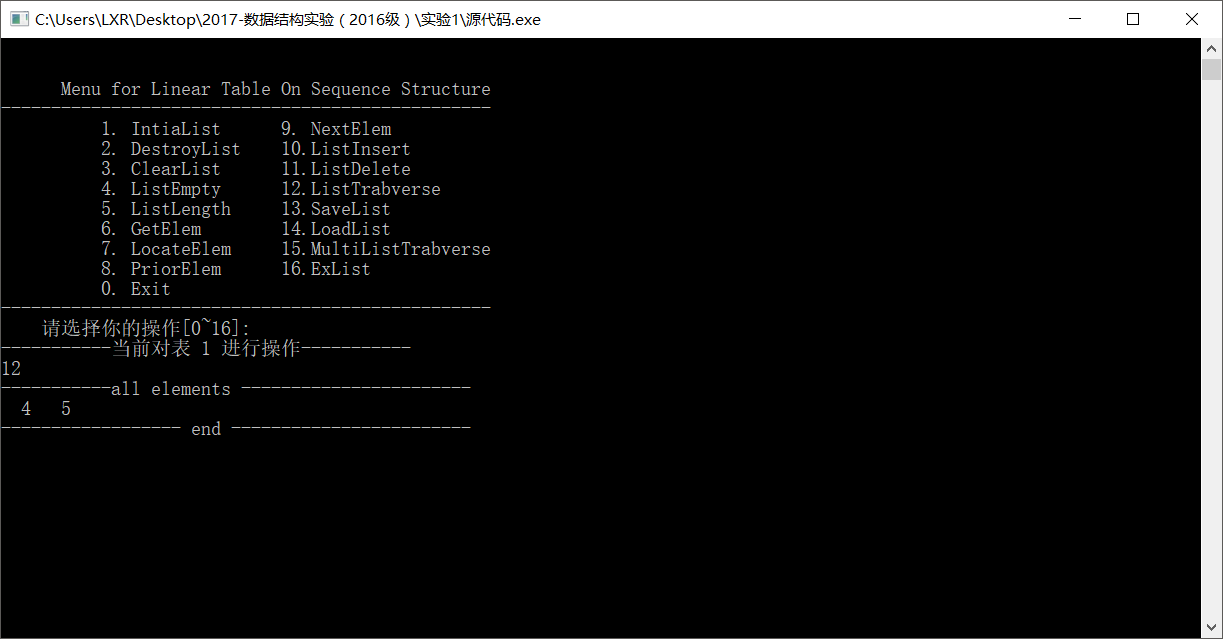


图1-27 遍历活动表并显示所有元素

（4）定位元素所在的位置，输入分别输入合法和不合法的i值。

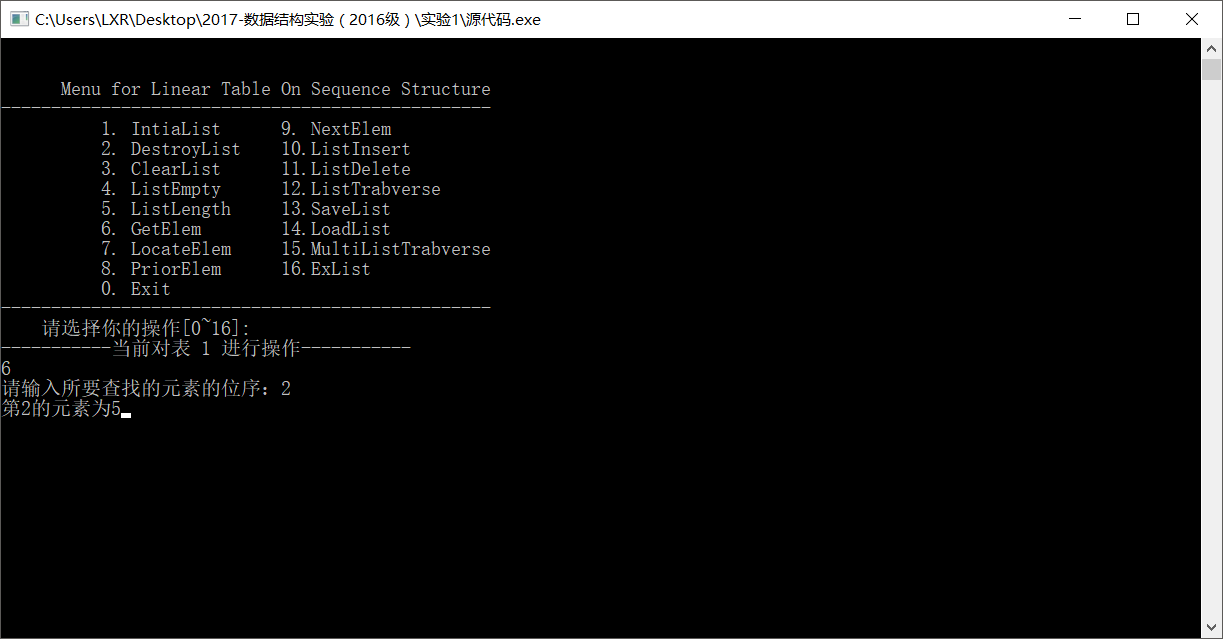


图1-28 查找合法的i的位序，输出正确的信息

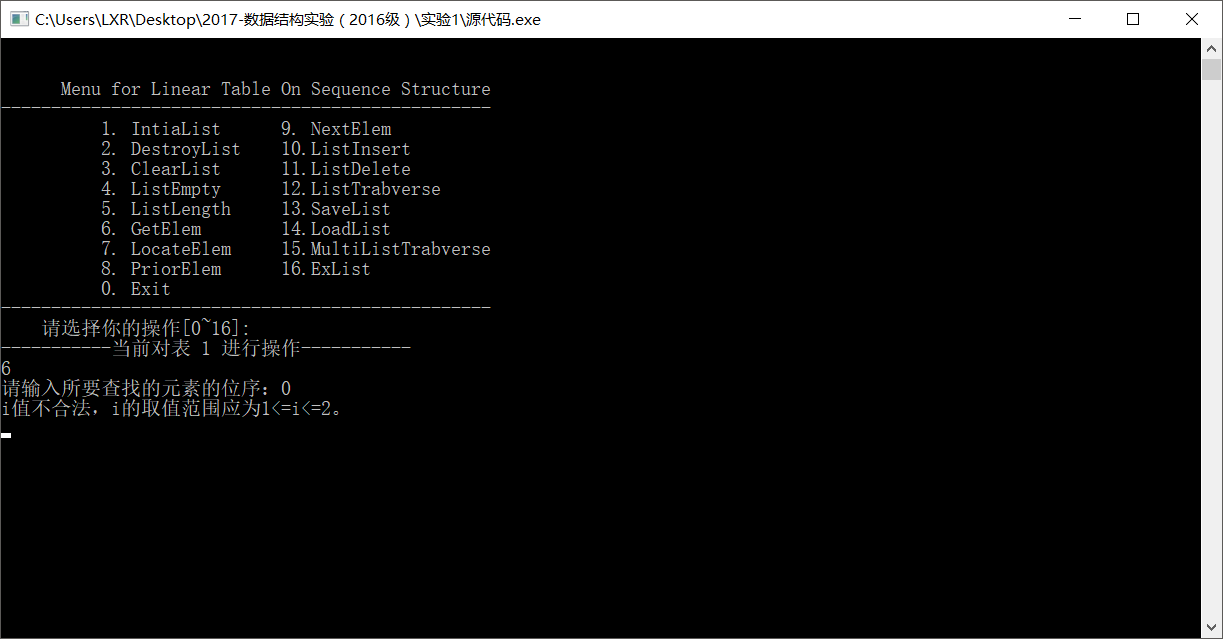


图1-29 查找元素的位序，输入不合法的i，提示错误并输出i的合法范围

1. 定位元素的位置，分别输入线性表中的元素4和非线性表中的元素0。

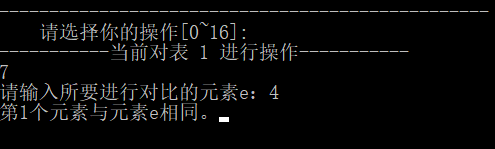
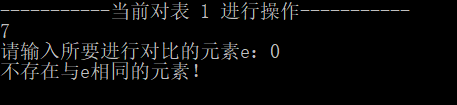


图1-30 定位元素4



1-31 定位元素0

（7）进行输出前驱元素的操作，分别输入线性表此刻的首元素、线性表中除了首元素以外的一个元素和一个不合法的i。

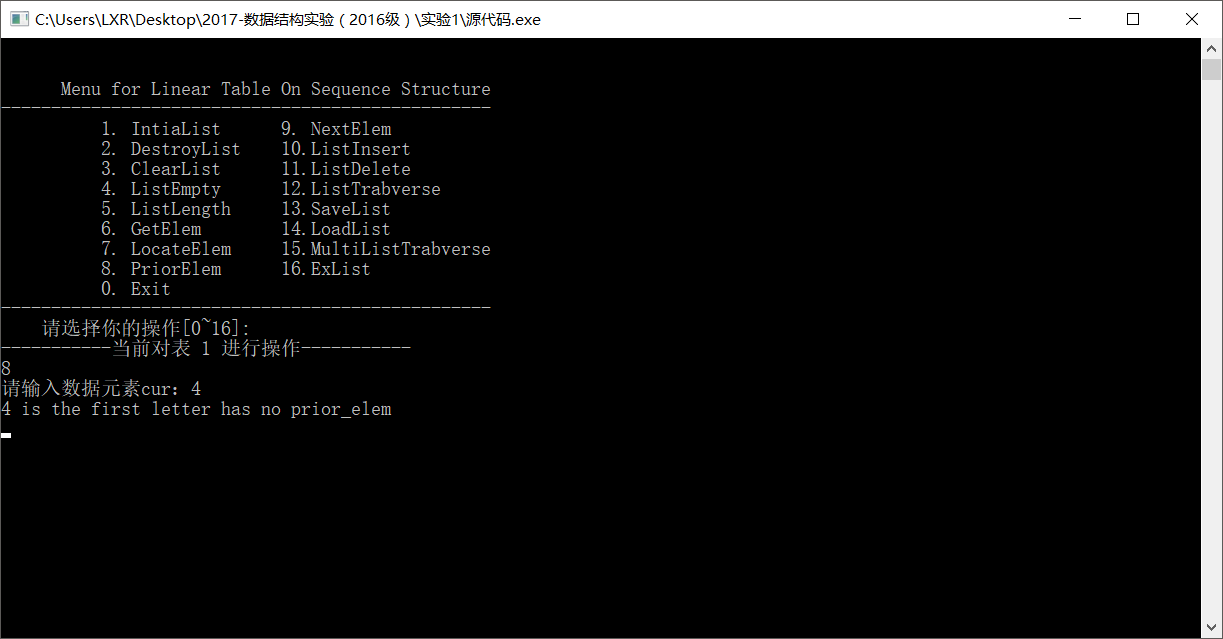


图1-31 查找首元素的前驱元素，输出提示信息

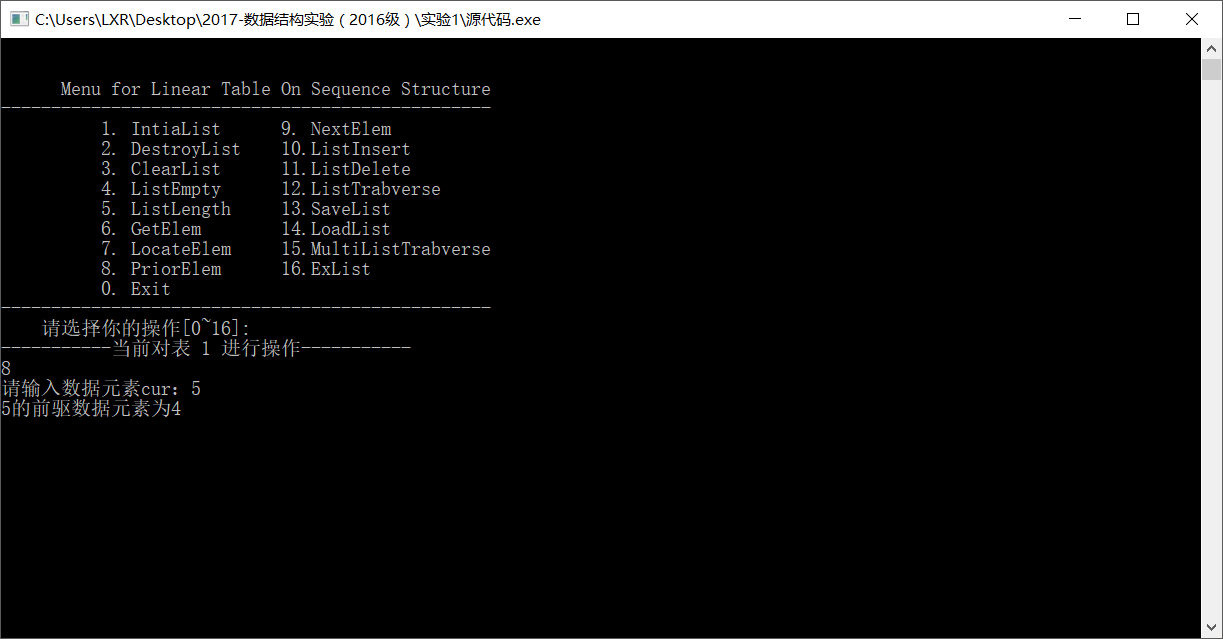


图1-32 查找5的前驱元素，输出其前驱元素为4

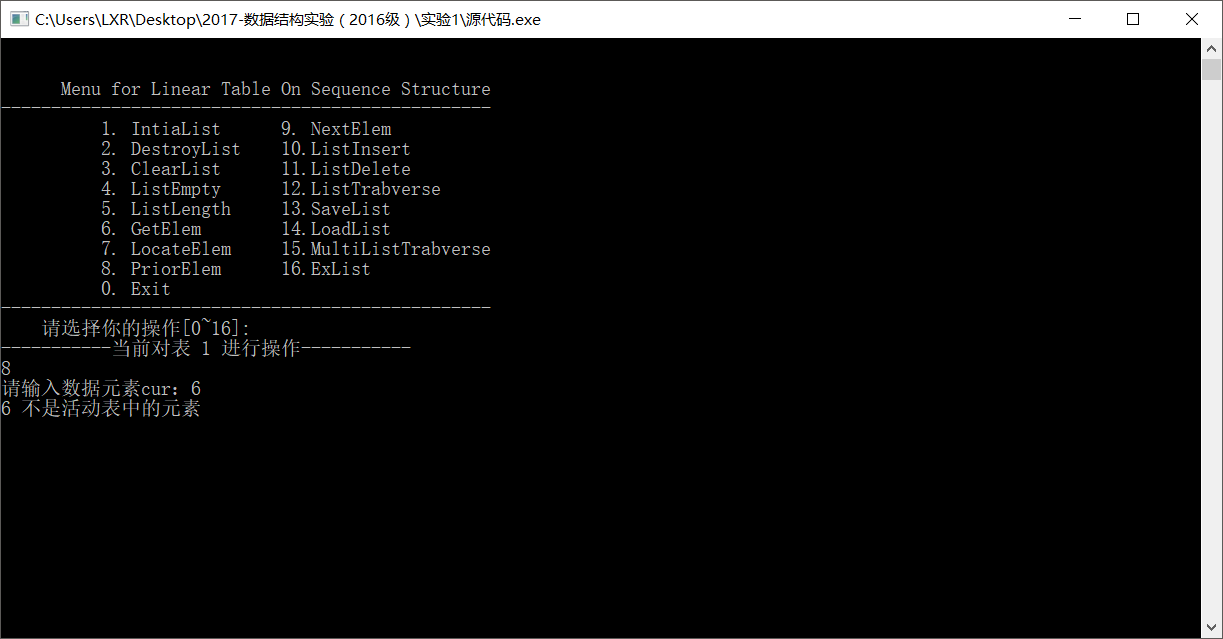


图1-33 输入一个不属于线性表中的元素，则提示错误信息

（8）进行输出后继元素的操作，依次输入线性表中除了尾元素以外的一个元素、线性表此刻的尾元素和一个不合法的i。

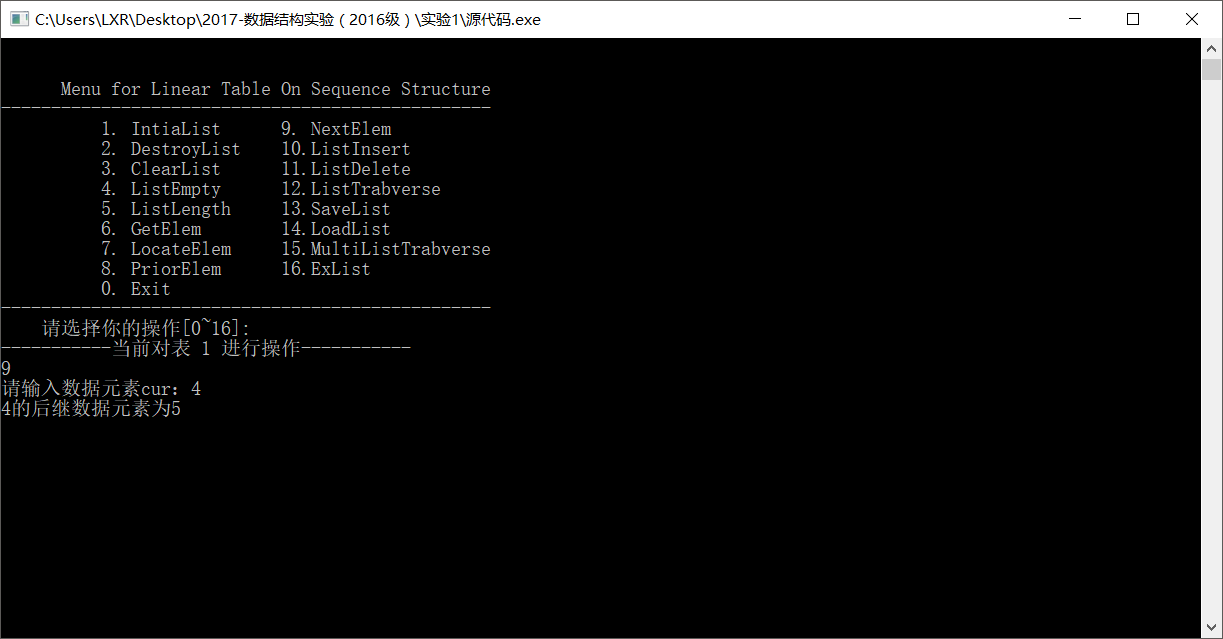


图1-34 查找元素4的后继元素，显示其后继元素为5

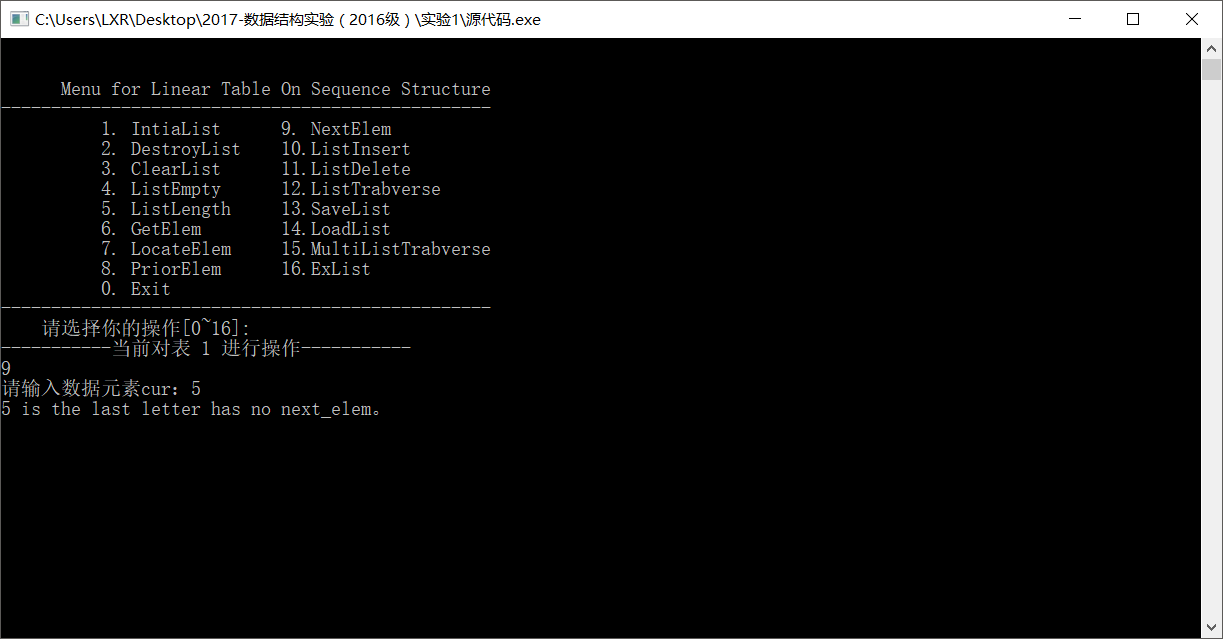


图1-35 查找尾元素的后继元素，输出提示信息

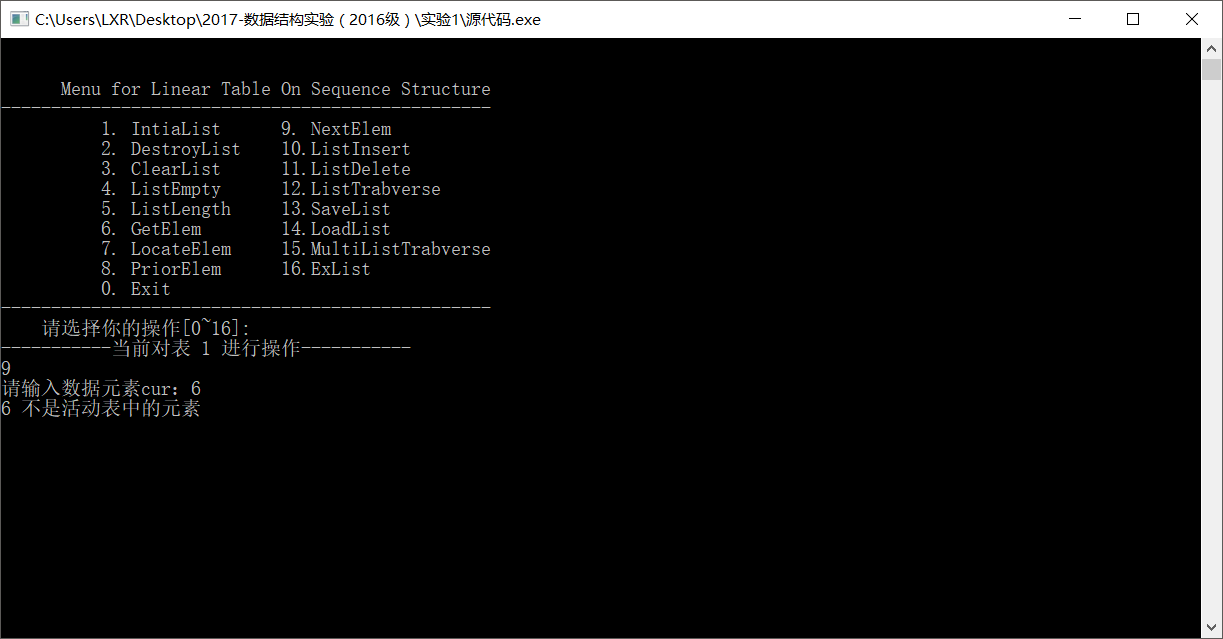


图1-36 输出一个不属于活动表的元素，提示错误信息

（9）将现在的线性表元素写入文件。

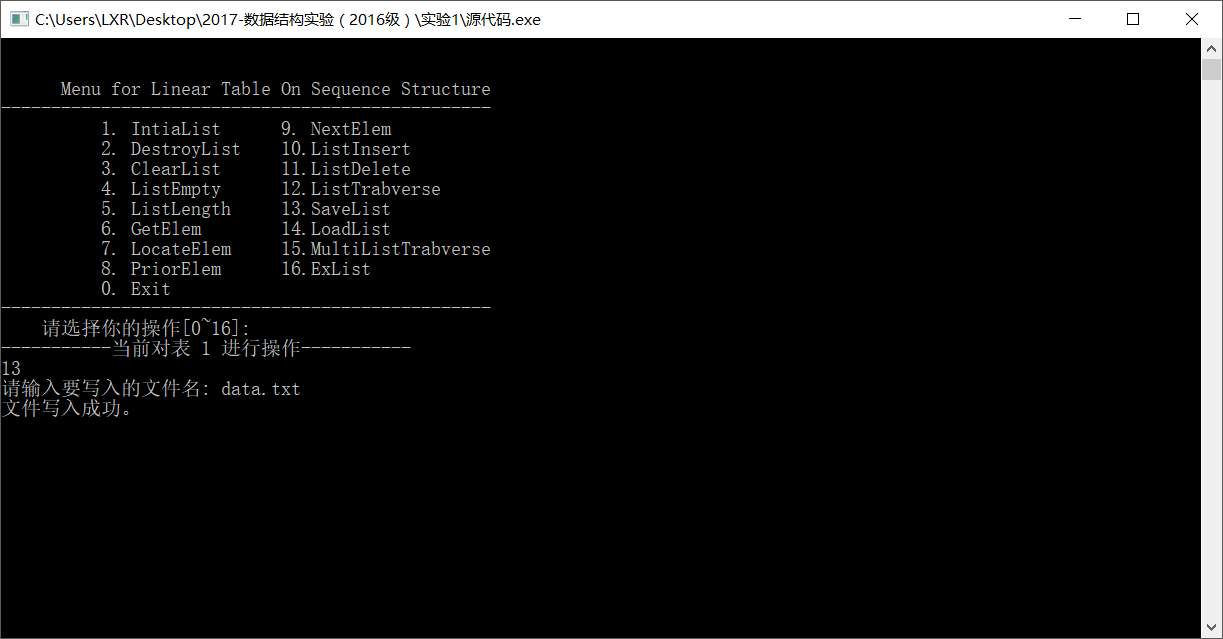


图1-37 将线性表写入名为data.txt的文件，若无该文件则自动创建

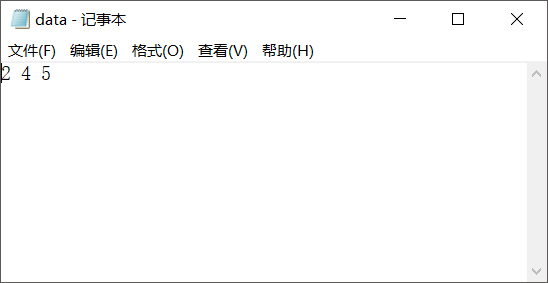


图1-38 文件中的线性表信息，第一个元素为线性表表长

（10）将活动表重置，并进行判断空表、表长显示、遍历显示的操作来检验重置是否成功。

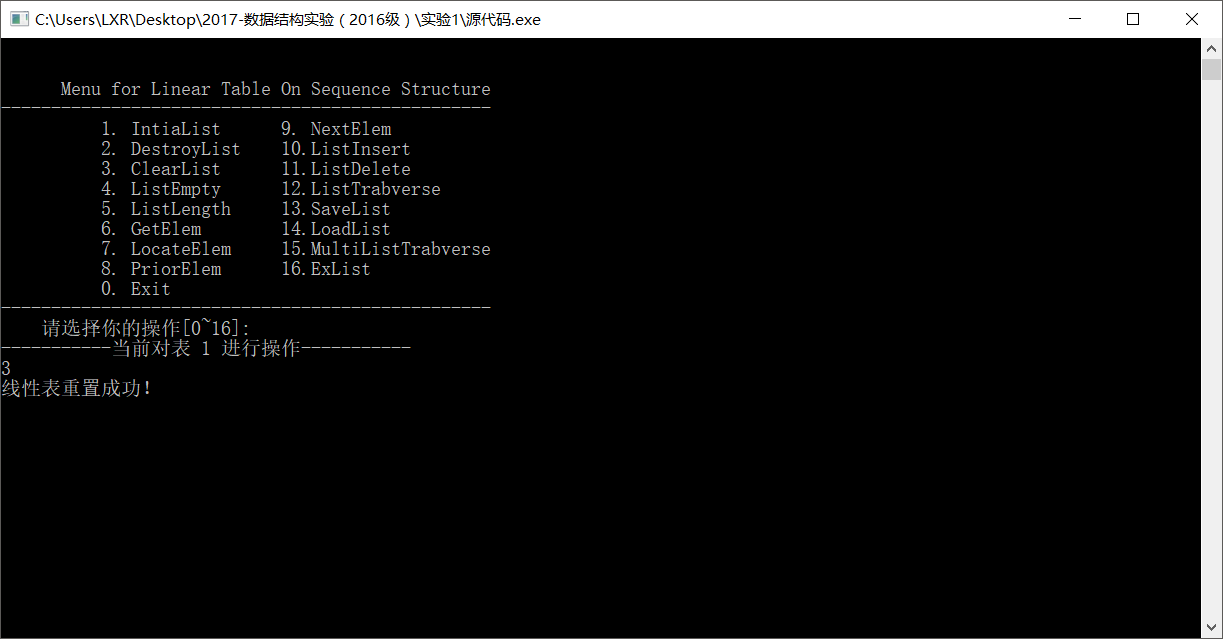


图1-39 重置当前活动表

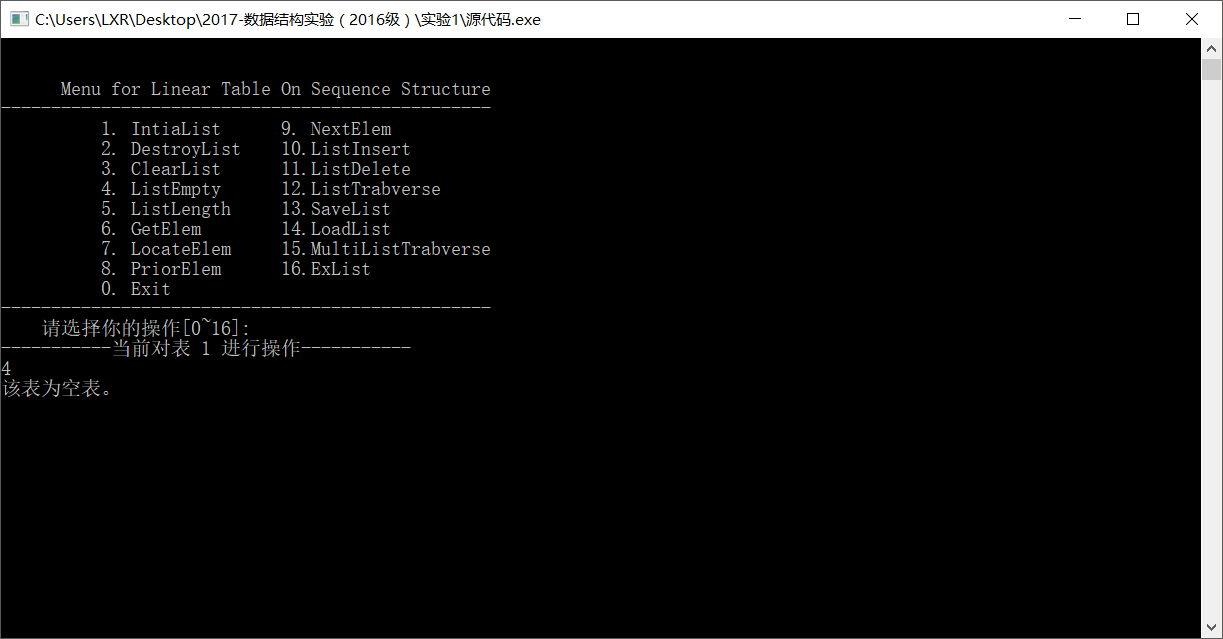


图1-40 判断重置后的表为空表

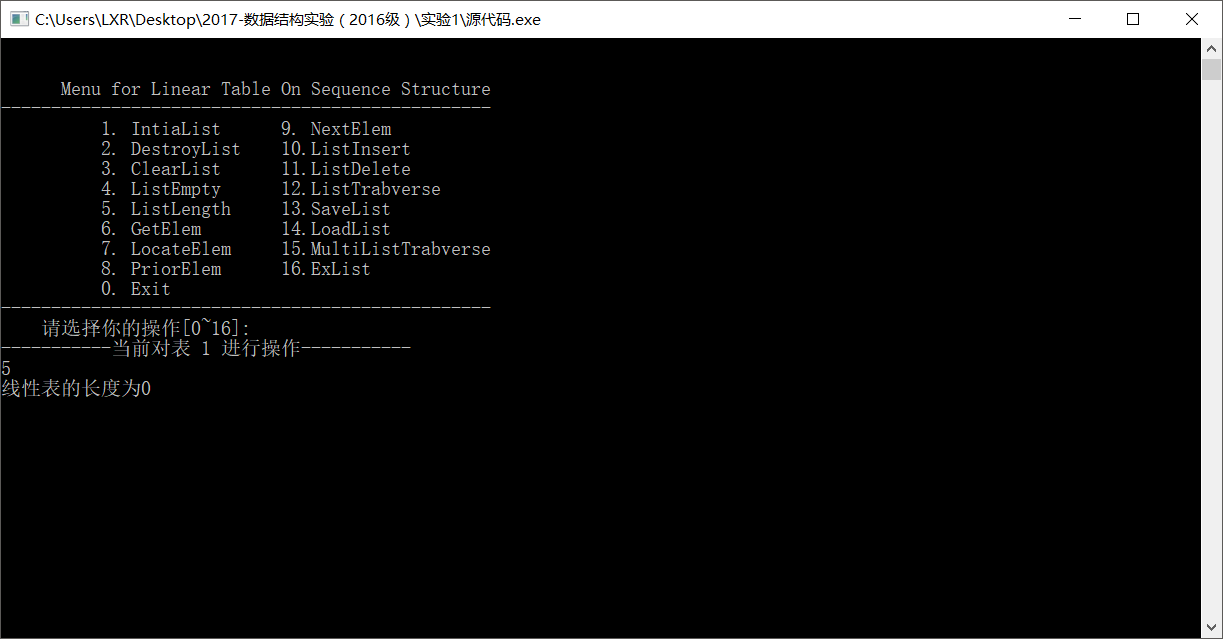


图1-41 输出重置后的表的表长

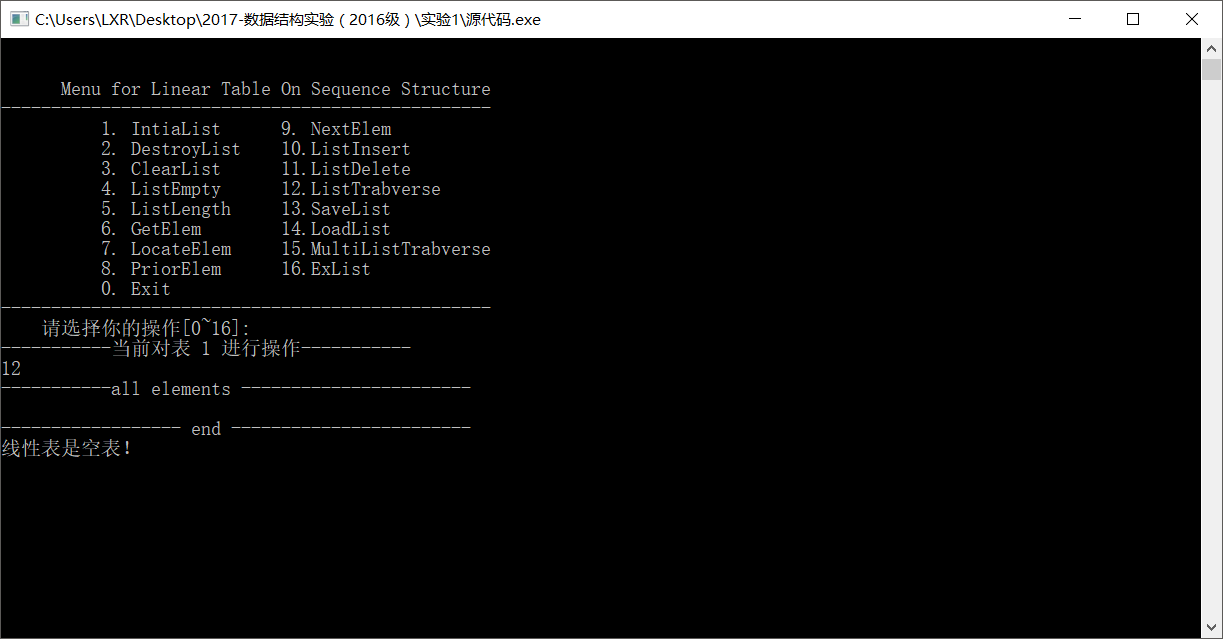


图1-42 对重置后的表进行遍历输出

（11）销毁目前的工作表。

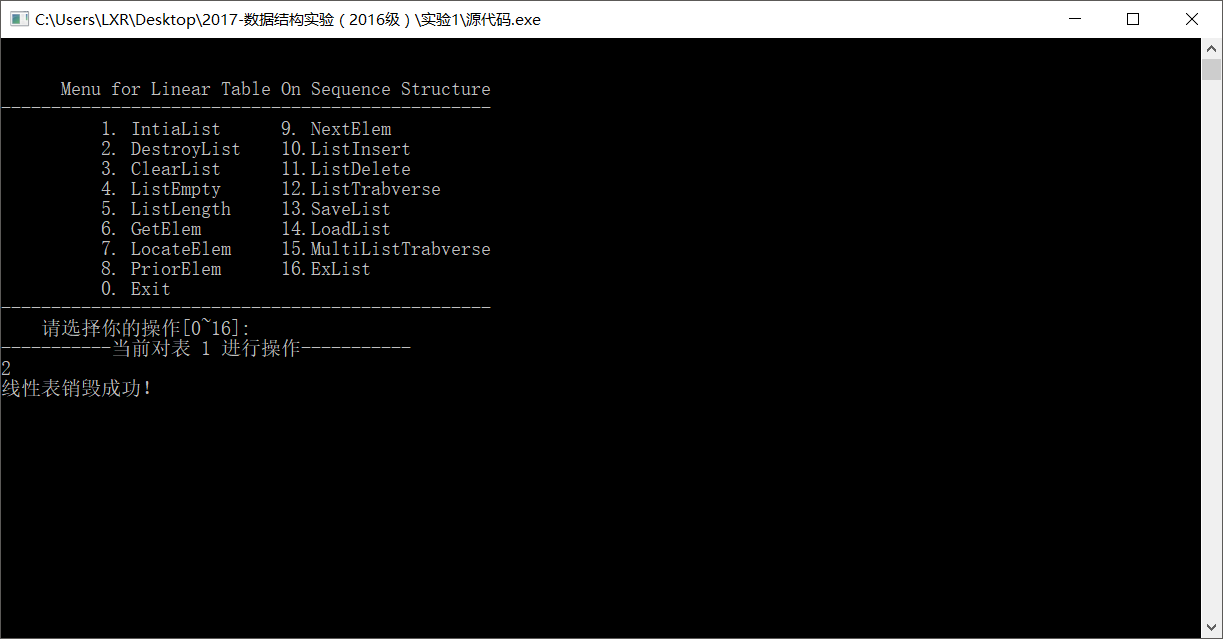


图1-43 销毁当前活动表

测试用例四：

新建两个表，自动生成编码1和编码2，对应测试计划38-44。

测试结果：

（1）新建一个表，调用LoadList函数将文件中的数据导入此刻的工作表中，并通过遍历显示导入成功。

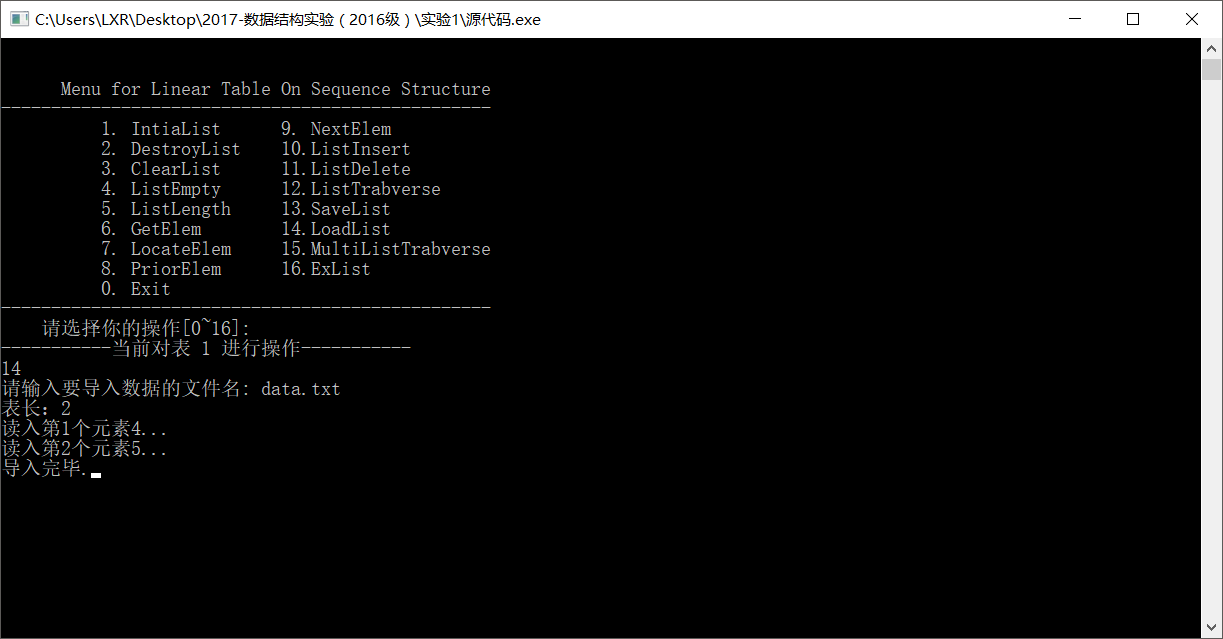


图1-44 用文件里的元素覆盖当前活动表

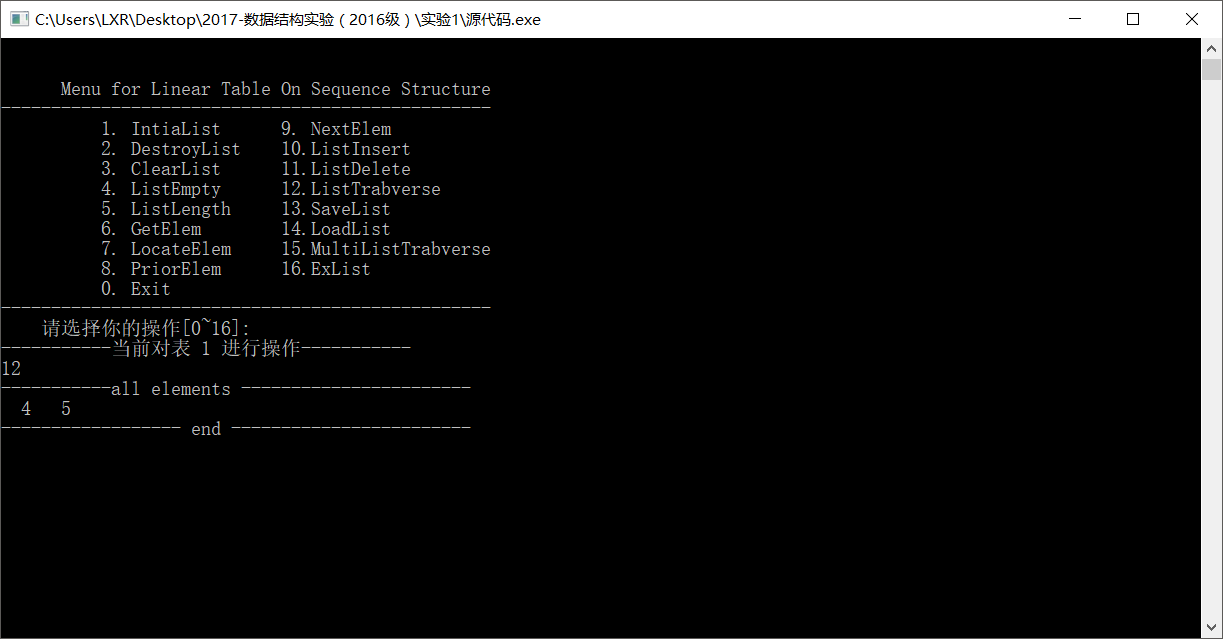


图1-45 遍历活动表并显示所有元素

（2）新建一个链表，其自动生成编码为2，调用MultiListTrabverse函数遍历此刻多表链表。

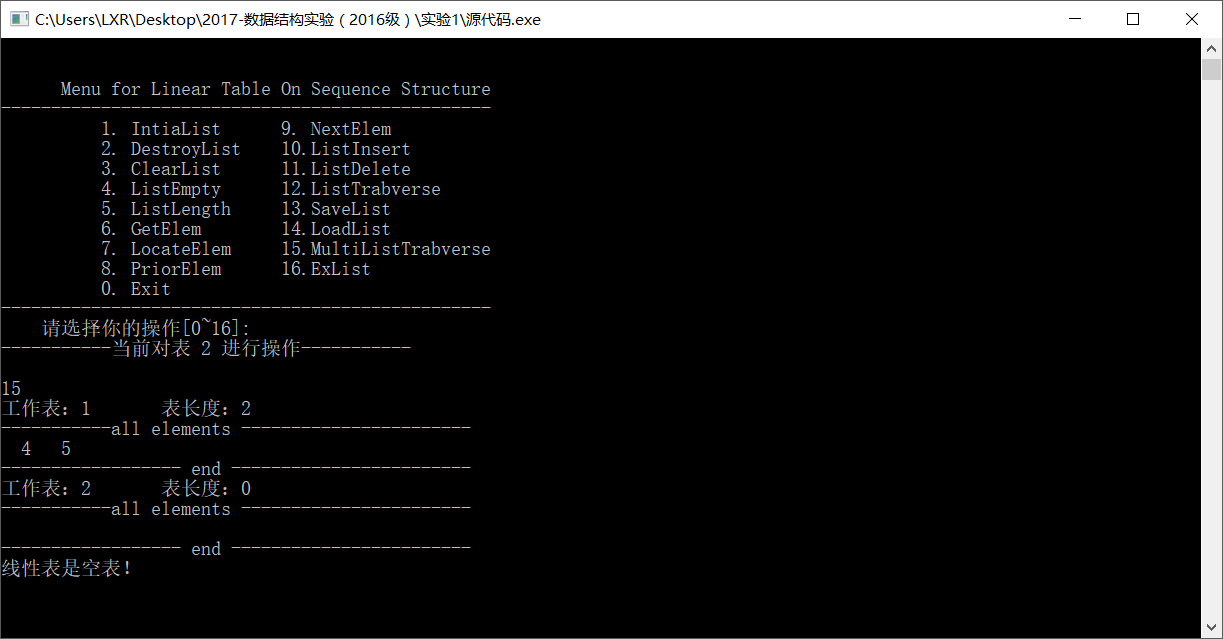


图1-46 使用MultiListTrabverse函数遍历此刻线性表中的数据元素

（3）调用ExList函数首先遍历显示所有的线性表，其次切换为1表，并进行遍历操作，以确定切换成功。

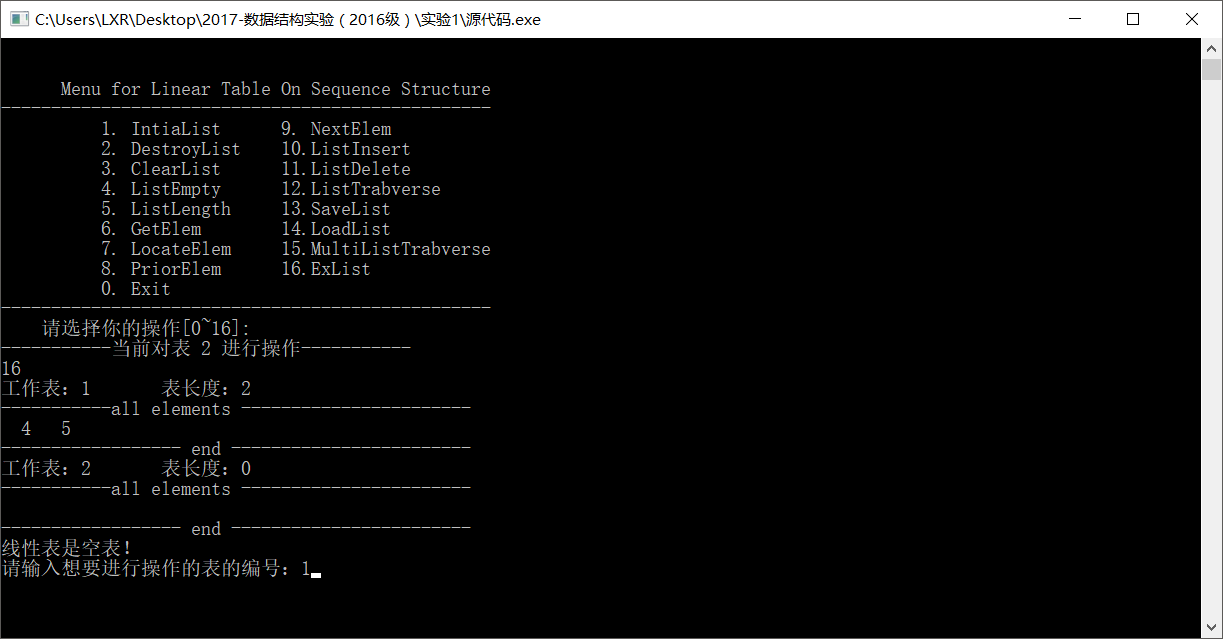


图1-47 使用ExList函数切换为表1

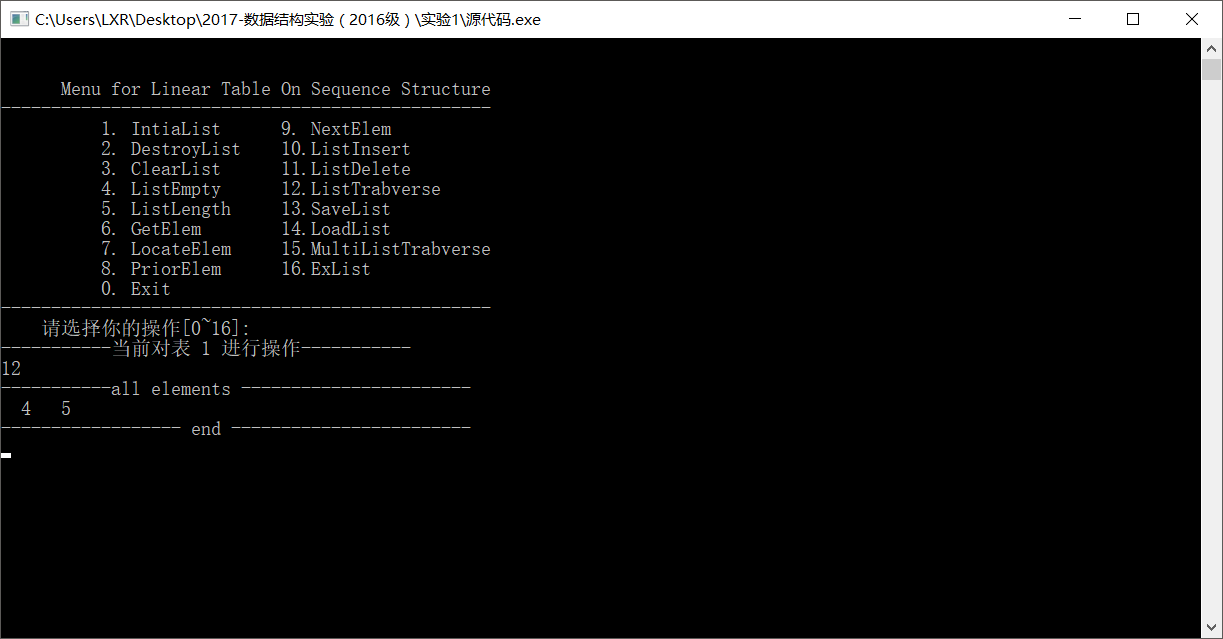


图1-48 遍历此刻的活动表，可判断线性表以切换

（4）输入操作指令0，程序退出。

## 1.4 实验小结

本次上机实验对线性表的算法和实现有了更深刻的理解，在实验中遇到的主要问题就是没有考虑到所有的路径，尤其是对一些非法的值和边界值的操作，总会出现输出错误的数字信息，或者程序停止运行等问题。在一遍遍重新思考整个系统的时候，才注意到了许多最开始编写没有注意到的问题，例如并不是所有的路径都有返回值、指针运用错误、在一些函数的定义中没有使用值传递等等。

总而言之就是在编写一个完善的程序前应先进行一个较为完善的分析，而不是急于开始编写，而一个较为完善的分析过程首先应当包括临界值的处理，以及空间的合理分配和回收，另有余力也要考虑一些更人性化的设计。

# **2 基于链式存储结构的线性表实现**

## 2.1 问题描述

采用链式结构作为线性表的物理结构，并以函数形式定义实现线性表的初始化表、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12种基本运算，且要求构造一个具有菜单的功能演示系统，在主程序中完成函数调用所需实参值的准备和函数执行结果。

2.1.1 线性表抽象数据类型

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

ADT SqList{

**数据对象:**D = { ai|ai ∈ ElemSet, i = 1,2, ……,n, n≥0 }

**数据关系:**R1 = { <ai-1,ai>|ai-1,ai∈D, i = 2,……,n }

**基本操作:**

IntiaList(&L)

初始条件:线性表L不存在.

操作结果:构造一个空的线性表L，成功构造返回OK，否则返回ERROR。

DestroyList(&L)

初始条件:线性表L已存在。

操作结果:销毁线性表L，当表尚未被创建时返回ERROR,否则销毁成

功返回OK。

ClearList(&L)

初始条件:线性表L已存在。

操作结果:将L重置为空表，成功置空返回OK，否则返回ERROR。

ListEmpty(L)

初始条件:线性表L已存在。

操作结果: 若L为空表则返回OK，否则返回ERROR

ListLength(L)

初始条件:线性表L已存在。

操作结果: 成功返回L中数据元素个数，失败返回ERROR。

GetElem(L,I,&e)

初始条件:线性表L已存在、非空且1≤i≤ListLength(L)。

操作结果: 找到L中第i个元素的值，成功则用e返回第i个元素的值，否则返回ERROR。

LocateElem(L,e)

初始条件:线性表L已存在且非空。

操作结果: 成功则返回第一个与e相同的数据的位序，不存在则返回0，其余情况返回ERROR。

PriorElem(L,cur,&pre\_e)

初始条件:线性表L已存在、非空且cur\_e不是第一个数据元素。

操作结果: 成功则返回与cur相同的第一个数据，并返回其第一个前驱；若无前驱以及其他情况则返回ERROR。

NextElem(L,cur,&next\_e)

初始条件:线性表L存在、非空且cur\_e不是最后一个数据元素。

操作结果: 成功则返回与cur\_e相同的第一个数据，并返回其后驱；若无后驱以及其他情况则返回ERROR。

ListInsert(&L,i,e)

初始条件:线性表L已存在且非空，1≤i≤ListLength(L)+1。

操作结果:在L的第i个位置之前插入新的数据元素e，成功插入返回OK，否则返回ERROR。

ListDelete(&L,i,e)

初始条件: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，成功删除返回OK，否则返回ERROR。

ListTraverse(L)

初始条件:线性表L已存在。

操作结果: 成功遍历返回线性表的长度，否则返回ERROR。

} ADT SqList

2.1.2 多线性表抽象数据类型

基于之前实现的线性表数据类型，设计了多线性表的数据对象、数据关系，并以定义了多线性表的初始化、销毁、切换共3种基本运算，具体数据和运算功能定义如下。

ADT MultiList{

**数据对象**:D = { ei|ei ∈ SqList, i = 1,2, ……,n, n≥0 }

**数据关系**:R1 = { <ei-1,ei>|ei-1,ei∈D, i = 2,……,n }

**基本操作**:

InitiaList2 (&head, &&List)

初始条件:无

操作结果: 构造一个空的线性表,并使List只想该表,成功构造返回OK，否则返回ERROR。

DestroyList2(&head, &&List)

初始条件:表MultiList非空。

操作结果: 成功销毁返回OK，否则返回ERROR.

ExList(&head)

初始条件:无。

操作结果: 先遍历多个线性表，并进行多表切换操作, 一个指向切换后的活动表的指针.

} ADT MulList

2.1.3 演示系统与数据文件

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。

演示系统可同时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。

演示系统选择实现多个线性表管理。

## 2.2 系统设计

2.2.1 数据物理结构

在本程序中，数据原子类型ElemType被定义为int整型。

1.线性表的数据存储结构

线性表数据类型在程序中定义为结构体ListNode，具体定义方式如下：

struct ListNode {

ElemType data; //头结点中存该线性表的长度，而其余结点存储数据

struct ListNode \*next; //指向线性表下一个结点

}；

2.多线性表的数据存储结构

多线性表数据类型在程序中定义为结构体MultiList，具体定义方式如下：

struct MultiList {

struct ListNode \*ListHead; //指向一个线性表的头节点

int num; //线性表的编码

struct MultiList \*next; //指向下一个表头节点

};

多线性表的存储结构如图2-1所示：



图2-1 多线性表的存储结构

2.2.2 演示系统

演示系统由用户操作界面与功能调用部分组成。

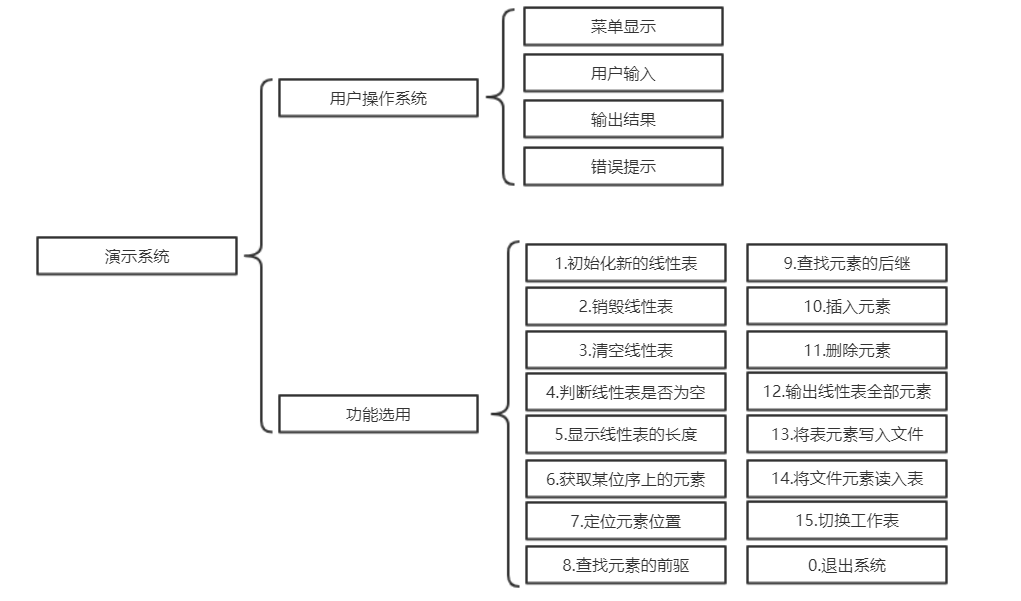


图2-2 演示系统模块结构图

2.2.3 数据文件存储格式

考虑到方便查看,以txt格式直接存放元素的值,相邻元素用空格隔开。

2.2.4 线性表运算实现算法

1.IntiaList(&L)

**算法思想：**首先判断L是否为空指针(即在IntiaList2中是否分配内存成功)，若是则返回ERROR；若不是则将L的next指针指向NULL，并将数据域赋值为0，即表示当前表长为0。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

2.DestroyList(&L)

**算法思想：**算法流程图如下图2-3所示。

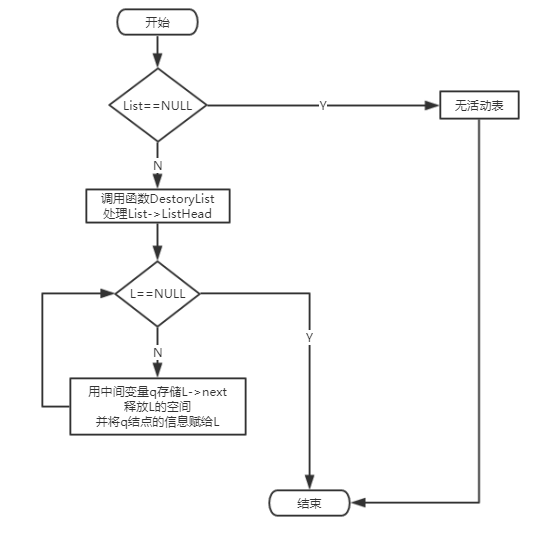


图2-3 DestoryList算法流程图

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

3.ClearList(&L)

**算法思想：**首先判断L->next是否为空指针(即L是否为空线性表)，若是则不必进行多余操作可直接返回OK；否则使用DestoryList函数销毁L->next指向的线性表，然后使L->data = 0,L->next=NULL,成功则返回OK，否则返回ERROR。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

4.ListEmpty(L)

**算法思想：**首先判断L->data是否等于0(即L是否为空表)，若是则返回OK，

否则返回ERROR。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

5.ListLength(L)

**算法思想：**返回L->data(头结点的数据域存放的是该线性表的长度)。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

6.GetElem(L,I,&e)

**算法思想：**首先判断L.data是否等于0(即L是否为空表)，若是则输出提

示信息并返回ERROR，然后判断i的输入是否合法（即是否符合1<=i<=L.length）,若不是则提示i的数值不合法，返回ERROR，否则声明指针q并表示L的地址，利用循环找到第i个结点，用e保存该结点的data，返回OK。

**时间复杂度分析：**假设表中有n个元素，且所查找的位序在表中的概率相等（均为1/n），则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

7.LocateElem(L,e)

**算法思想：**算法流程图如下图2-4所示。

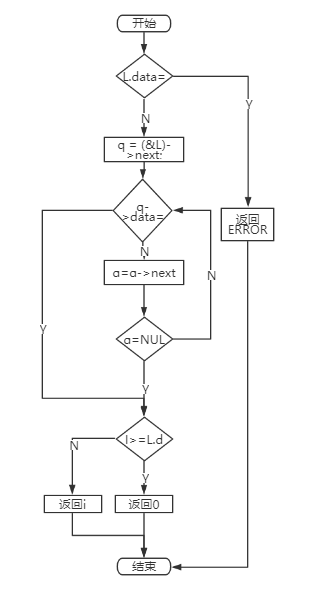


图2-4 LocateElem算法流程图

**时间复杂度分析：**假设表中有n个元素，且第一个与e相同的元素在表每个

位置的概率相等（均为1/n），则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部变量i，算法的空间复杂度为常量阶O(1)。

8.PriorElem(L,cur,&pre\_e)

**算法思想：**算法流程图如下图2-5所示。

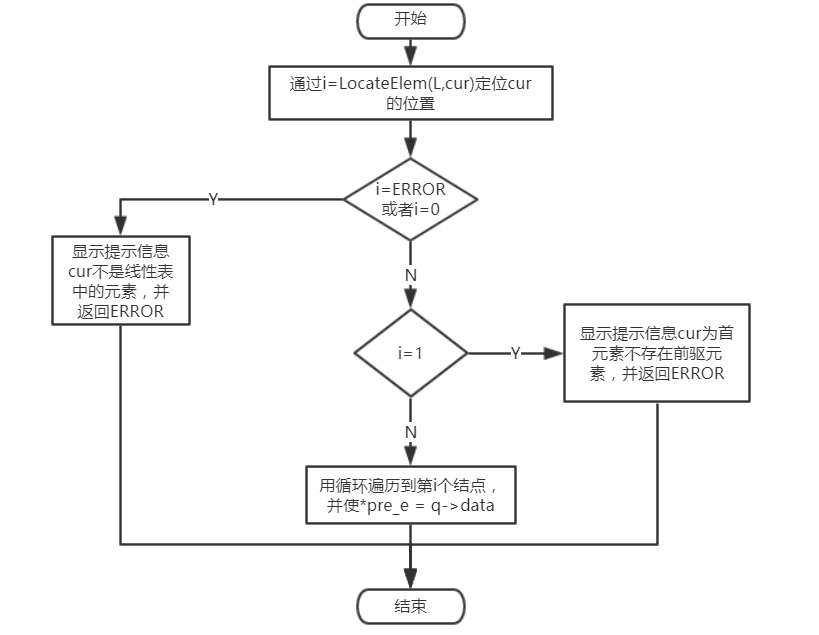


图2-5 PriorElem算法流程图

**时间复杂度分析：**假设表中有n个元素，且第一个与e相等的元素在表每个位置的概率相等（均为1/(n-1)），则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部变量i和局部结构指针q，算法的空间复杂度为常量阶O(1)。

9.NextElem(L,cur,&next\_e)

**算法思想：**算法流程图如下图2-6所示。

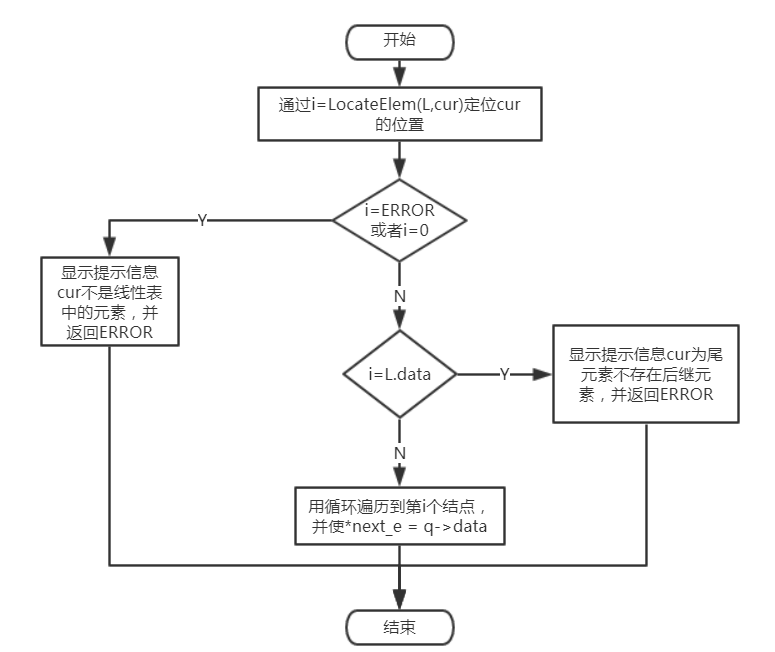


图2-6 NextElem算法流程图

**时间复杂度分析：**假设表中有n个元素，且第一个与e相等的元素在表每个位置的概率相等（均为1/(n-1)），则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部变量i和局部结构指针q，算法的空间复杂度为常量阶O(1)。

10.ListInsert(&L,i,e)

**算法思想：**算法流程图如下图2-7所示。

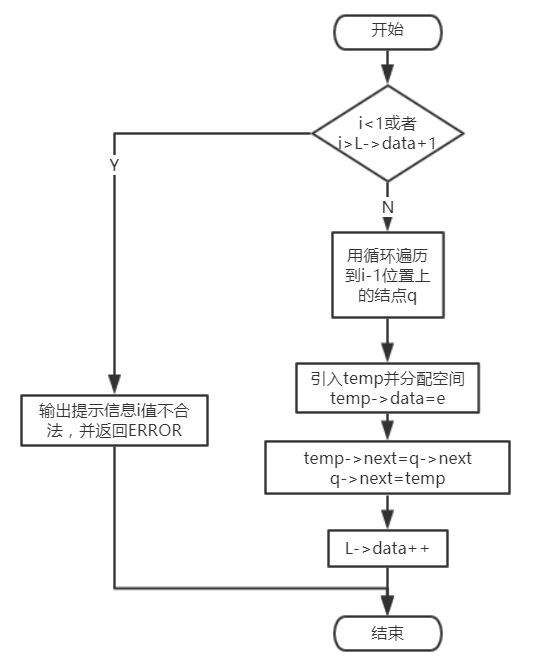


图2-7 ListInsert算法流程图

**时间复杂度分析：**假设表中有n个元素，且在每个位置插入的可能性相等(1/n+1)，则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入两个局部结构指针q和temp，并分配一次空间，算法的空间复杂度为常量阶O(1)。

11.ListDelete(&L,i,e)

**算法思想：**算法流程图如下图2-8所示。

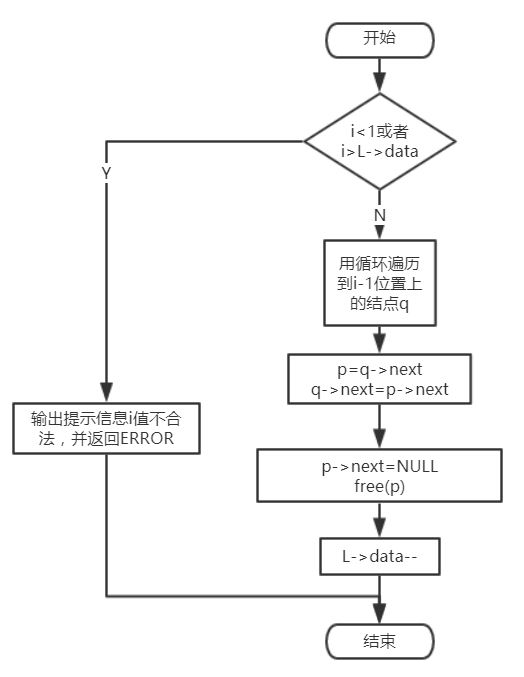


图2-8 ListDelete算法流程图

**时间复杂度分析：**删除算法的时间耗费主要在移动元素上，循环执行的次数L->data-i+1不定，假设表中有n个元素，且在每个位置删除的可能性相等(1/n+1)，则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部结构指针变量q和p，算法的空间复杂度为常量阶O(1)。

12.ListTraverse(L)

**算法思想：**先判断L.data是否等于0，若是则输出错误信息并返回ERROR，否则输出表长为L.data，令\*q=&L,利用for循环依次使q=q->data并输出q->data，并返回OK。

**时间复杂度分析：**假设表中有n个元素，则算法时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入变量i和局部结构指针变量q，算法的空间复杂度为常量阶O(1)。

13.SaveList(fp,L,filename)

**算法思想：**先输入文件名字，用fopen打开文件并返回文件指针fp，若打开失败则显示提示信息并返回ERROR，否则用fprintf依次写入线性表的每一个元素，最后关闭fp指针并返回OK。

**时间复杂度分析：**假设表中有n个元素，则算法时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部变量i和局部结构指针变量p，算法的空间复杂度为常量阶O(1)。

14.LoadList(fp,L,filename)

**算法思想：**先输入文件名字，用fopen打开文件并返回文件指针fp，若打开失败则显示提示信息并返回ERROR，使L->data=0,否则用fscanf先读入第一个元素，即线性表的表长，然后以表长为条件循环读入之后的每一个元素，并在读入每一个元素后调用ListInsert函数来进行插入，最后关闭fp指针并返回OK。

**时间复杂度分析：**假设表中有n个元素，则算法时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部变量i、e，算法的空间复杂度为常量阶O(1)。

2.2.5 多线性表运算实现算法

1.InitiaList2 (&head,&&List)

**算法思想：**引入 结构指针q使其等于head，用循环使q指向总链表的最后一个结点,即q->next=NULL。为q->next分配空间并使(\*List)=q->next,

(\*List)->num=q->num+1;为(\*List)->ListHead分配空间，并判断是否分配空间成功，令(\*List)->next=NULL,返回OK。

**时间复杂度分析：**假设多线性表结构的总长度为n，则本算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部结构指针变量q，并分配两次空间，算法的空间复杂度为常量阶O(1)。

2.DestroyList2(&head, &&List)

**算法思想：**算法流程图如下图2-9所示。



图2-9 DestroyList2算法流程图

**时间复杂度分析：**假设多线性表结构的总长度为n，在每个表执行删除操作的概率相等(1/n)，则本算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部结构指针变量p和q，算法的空间复杂度为常量阶O(1)。

3.ExList(&head)

**算法思想：**输入想要操作的表的编码，如果num<1则输出编码不合法，通过遍历找到该编码对应的含有所要线性表的结点，使指针List指向该节点，若List为空，则说明未找到编码所对应的线性表则输出编码不合法，最后返回List。

**时间复杂度分析：**假设有n个多表结构，且切换为每个表的操作概率相等(1/n)，则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**引入局部变量i和num以及局部结构指针变量List，算法的空间复杂度为常量阶O(1)。

## 2.3 系统实现

2.3.1 实验环境

实验代码使用dev c++编写与编译.

2.3.2 演示系统操作

演示系统采用键盘操作，控制台输出操作提示与结果，第一行显示当前操作表（默认为无），中间显示可用的操作项，输入对应的操作指令即可执行，输入非0~16的操作号会提示错误。

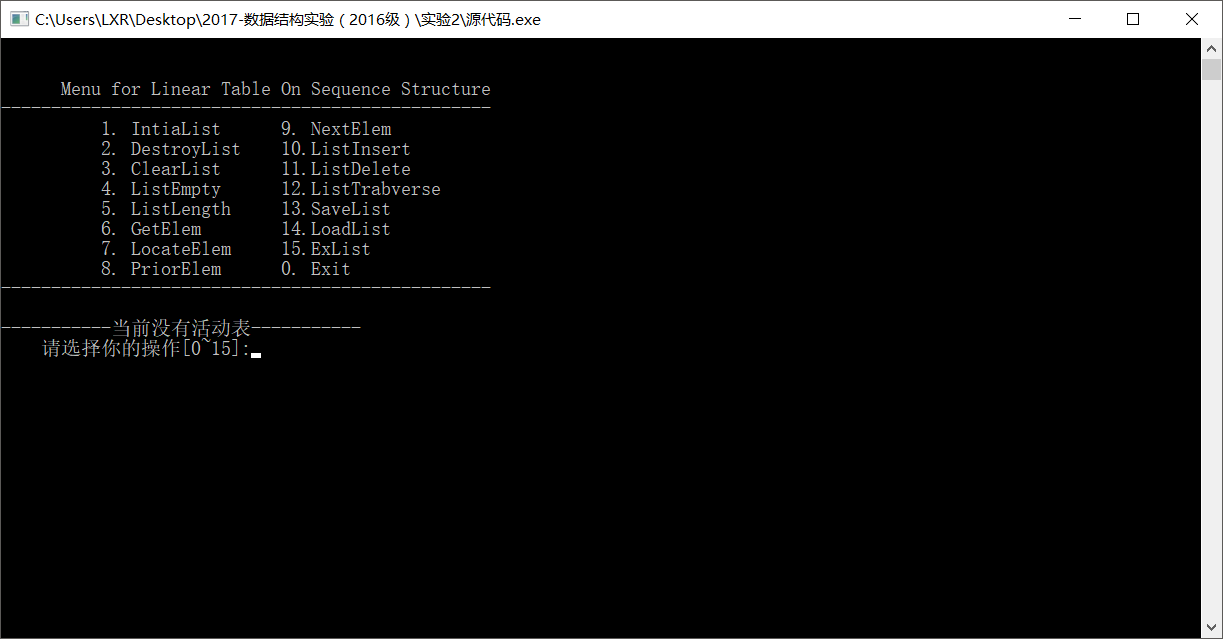


图2-10 启动程序后的操作菜单

输入操作指令0，程序退出

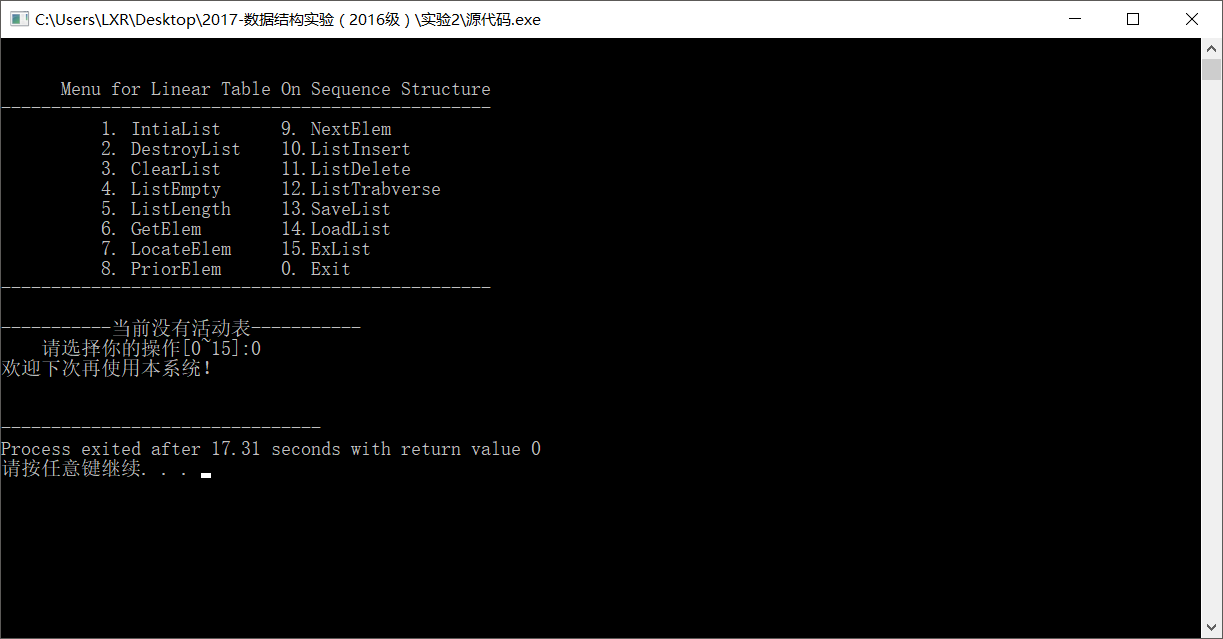


图2-11 输入操作0程序退出

2.3.3 测试计划

1.检测常规状态下系统的工作情况；

2.重点检查非法边界操作时的工作情况。

表2-1 演示系统测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 待测试功能 | 测试顺序 | 测试输入 | 预计输出 | 线性表状态 |
| IntiaList | 6  38  41 | 无 | 线性表创建成功！ | 6/38：表长0；编码1；  41：表长0，编码2 |
| DestroyList | 1  37 | 无 | 1：当前没有活动表  37：线性表销毁成功！ | 不存在 |
| ClearList | 2  33 | 无 | 2：当前没有活动表  33：线性表重置成功！ | 2：同1  33：表长0，编码1 |
| ListEmpty | 3  7  34 | 无 | 该表为空表； | 3：同1  7：同6  34：同33 |
| ListLength | 8  35 | 无 | 线性表长度为0 | 8：同6  35：同33 |
| GetElem | 9  22  23 | 0  2  0 | 9：该表为空表，查找失败；  22：第2个元素为9  23：i值不合法 | 9：同6  22-23：同20 |
| LocateElem | 10  24  25 | 10：1  24：9  25：0 | 10：不存在相同的元素  24：第2个元素与e相同  25：不存在与e相同的元素 | 10：同6  24-25：同20 |
| PriorElem | 11  26  27  28 | 1  8  9  7 | 1不是活动表中的元素  8为首元素没有前驱元素  9的前驱元素为8  7不是活动表中的元素 | 11：同6  26-28：同20 |
| NextElem | 29  30  31 | 8  9  7 | 8的后继元素为9  9为尾元素没有后继元素  7不是活动表中的元素 | 29-31：同20 |
| ListInsert | 13  14  15  16  17 | 1 7  2 8  3 9  0 6  6 6 | 13-15 无  16-17 i值不合法，插入失败 | 表中元素为7 8 9  表长为3  编码1 |
| ListDelete | 18  19  20 | 0  4  1 | 10-11删除元素失败  12 删除成功 | 18-19：同17  20：表中元素为8 9  表长为2  编码1 |
| ListTraverse | 12  21  36  40  43 | 无 | 12/36：输出空白加上提示信息  21/40/48：输出8 9 | 12：同6  21/40/48：同20  36：同34 |
| SaveList | 4  32 | data.txt | 4：无法读取  32：文件写入成功 | 4：同1  32：同20 |
| LoadList | 5  39 | data.txt | 5：无法读取  39：录入成功 | 5：同1  39：同20 |
| ExList | 42 | 1 | 切换为表1 | 表1：元素4 5表长  表2：空表  此时工作表为表1 |

输入操作指令0，程序退出.

2.3.4 测试结果

测试用例一：

未初始化线性表时，操作对应测试计划1-5。

测试结果：（在进行其他操作时同样提示当前没有活动表）

（1）销毁线性表。

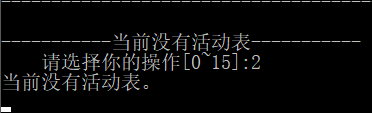


图2-12 销毁线性表

（2）清空线性表。

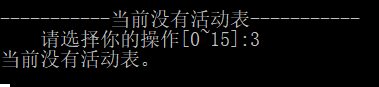


图2-13 清空线性表

（3）判断线性表是否为空表。

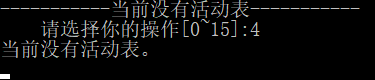


图2-14 判断线性表是否为空表

（4）将线性表内容写入文件。

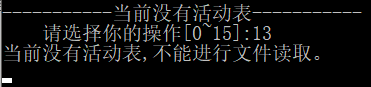


图2-15 文件写入

（5）将文件内容读入线性表。

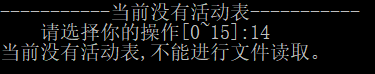


图2-16 文件读取

测试用例二：

空表，对应测试计划6-12

测试结果：

（1）初始化一个新表，并自动生成该表的编码1.

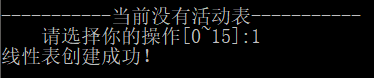


图2-17 初始化表

（2）判断表为空表。

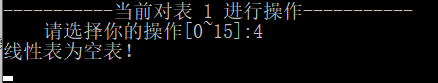


图2-18 判断新建的表是否是空表

（3）输出表长。

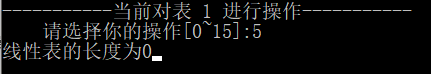


图2-19 输出表长

（4）查找某一元素的位序。

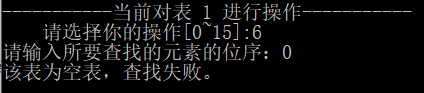


图2-20 查找位序

（5）定位元素位置。

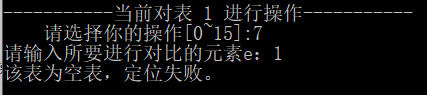


图2-21 定位元素位置

（6）查找某一元素的前驱元素。

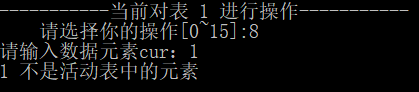


图2-22 查找前驱元素

（7）遍历该线性表。

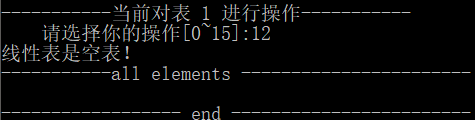


图2-23 遍历线性表

测试用例三：

先输入元素7、8、9并对线性表进行操作，对应测试计划13-37。

测试结果：

（1）依次输入7、8、9。

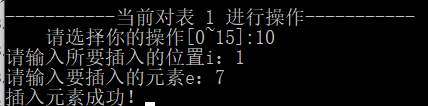


图2-24 插入位置1、插入元素7

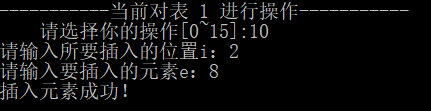


图2-25 插入位置2、插入元素8

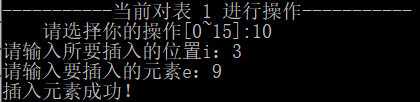


图2-26 插入位置3、插入元素9

（2）测试边界值插入，如位置为0或者6，则显示提示信息。

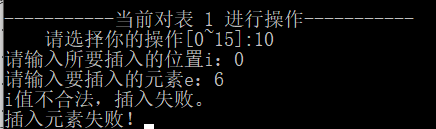


图2-27 若插入位置输入0，则插入失败



图2-28 输入一个比表长长2的i值，插入失败

（3）依此输入删除位序0、4、1，最后输出当前线性表的元素，用以验证删除成功。



图2-29 输入删除位序0，则提示删除失败

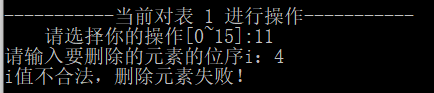


图2-30 输入删除位序4（即比表长大的位序），则提示删除失败

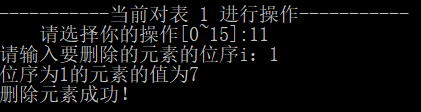


图2-31 输入删除位序1，输出原表位序1出的元素并提示删除成功

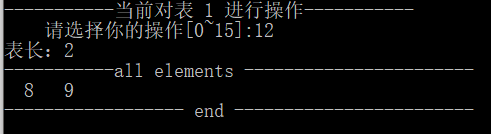


图2-32 遍历活动表并显示所有元素

（4）定位元素所在的位置，输入分别输入合法和不合法的i值。

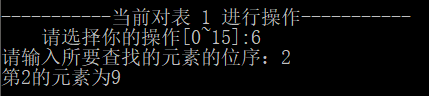


图2-33 查找合法的i的位序，输出正确的信息

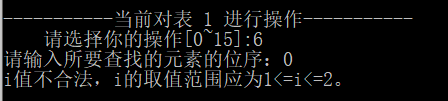


图2-34 查找元素的位序，输入不合法的i，提示错误并输出i的合法范围

（5）定位元素的位置，分别输入线性表中的元素4和非线性表中的元素0。

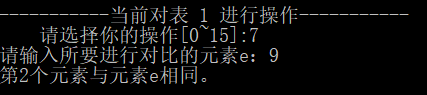
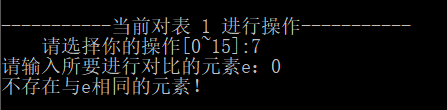


图2-35 定位元素9



2-36 定位元素0

（6）进行输出前驱元素的操作，分别输入线性表此刻的首元素、线性表中除了首元素以外的一个元素和一个不合法的i。

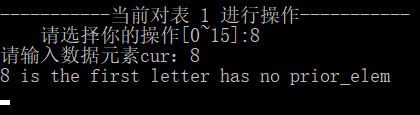


图2-37 查找首元素的前驱元素，输出提示信息

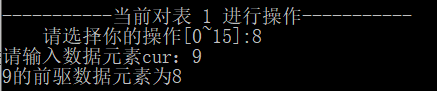


图2-38 查找9的前驱元素，输出其前驱元素为8

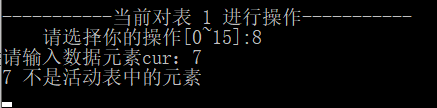


图2-39 输入一个不属于线性表中的元素，则提示错误信息

（7）进行输出后继元素的操作，依次输入线性表中除了尾元素以外的一个元素、线性表此刻的尾元素和一个不合法的i。

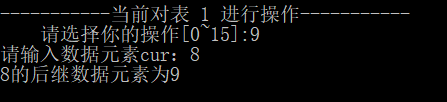


图2-40 查找元素8的后继元素，显示其后继元素为9

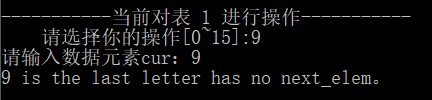


图2-41 查找尾元素的后继元素，输出提示信息

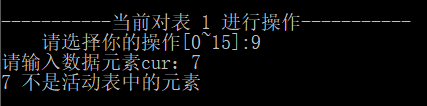


图2-42 输出一个不属于活动表的元素，提示错误信息

（8）将现在的线性表元素写入文件。

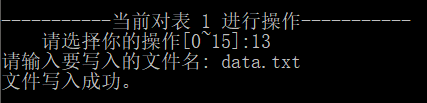


图2-43 将线性表写入名为data.txt的文件，若无该文件则自动创建

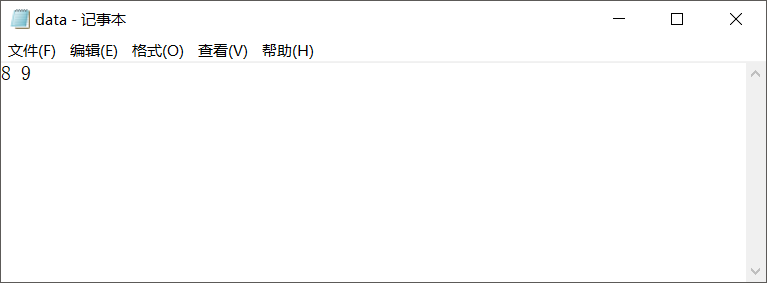


图2-44 文件中的线性表信息

（9）将活动表重置，并进行判断空表、表长显示、遍历显示的操作来检验重置是否成功。



图2-45重置当前活动表

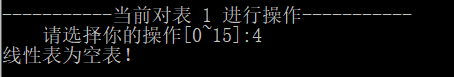


图2-46 判断重置后的表为空表

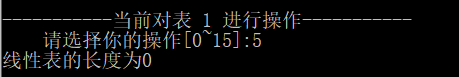


图2-47 输出重置后的表的表长

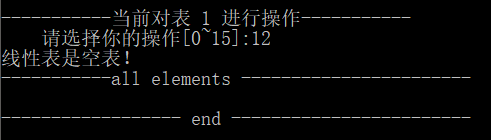


图2-48 对重置后的表进行遍历输出

（10）销毁目前的工作表。

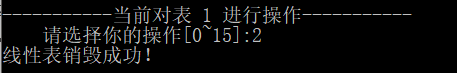


图2-49 销毁当前活动表

测试用例四：

新建两个表，自动生成编码1和编码2，对应测试计划38-44。

测试结果：

（1）新建一个表，调用LoadList函数将文件中的数据导入此刻的工作表中，并通过遍历显示导入成功。

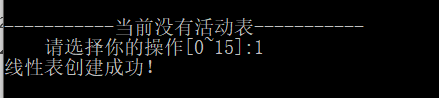


图2-50 新建线性表

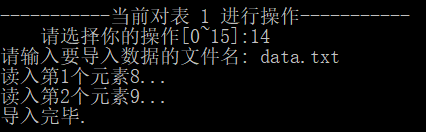


图2-51 用文件里的元素覆盖当前活动表

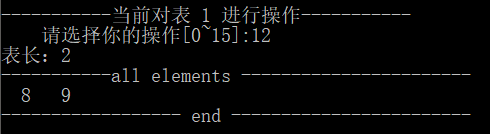


图2-52 遍历活动表并显示所有元素

（2）新建一个链表，其自动生成编码为2，调用MultiListTrabverse函数遍历此刻多表链表。

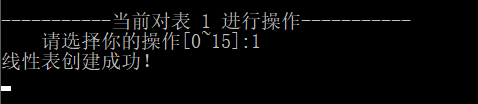


图2-53使用MultiListTrabverse函数遍历此刻线性表中的数据元素

（3）调用ExList函数首先遍历显示所有的线性表，其次切换为1表，并进行遍历操作，以确定切换成功。

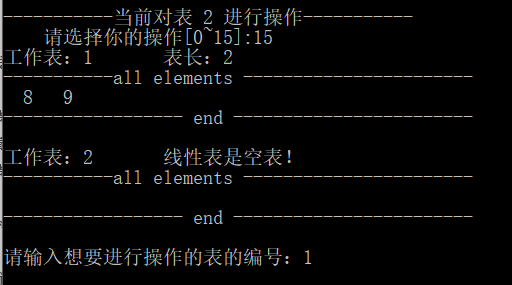


图2-54 使用ExList函数切换为表1

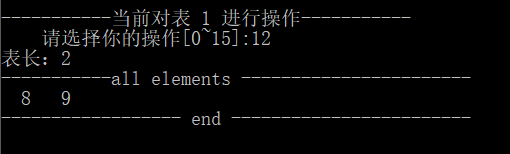


图2-55遍历此刻的活动表，可判断线性表以切换

（4）输入操作指令0，程序退出。

## 2.4 实验小结

本次实验遇到的主要问题是在最初写初始化函数时未使用双重指针，致使很多函数在检验时输出错误，最后一点点排查错误才找到了这处错误，觉得主要问题还是自己对指针还有疑问，对指针的使用较少，所以只有在使用时在会发现一些自己从未注意到过的错误。

其次还是第一次的问题，直到写报告时才发现一些运行错误问题，是自己在第一次实验后发现了问题却没有深刻反思自己甚至一错再错，也算是态度不认真的一种体现吧。总之不管做什么，首先一定要端正自己的态度吧。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

采用链表作为二叉树的物理结构，并以函数形式定义实现二叉树的初始化二叉树、销毁二叉树、创建二叉树、清空二叉树、判定空二叉树、求二叉树的深度、获得根结点、获得结点、结点赋值、获得双亲结点、获得左孩子结点、获得右孩子结点、获得左兄弟结点、获得右兄弟结点、插入子树、删除子树、前序遍历、中序遍历、后序遍历、层次遍历20种基本运算，以及文件读写和切换表3种拓展运算，且根据要求构造一个具有菜单的功能演示系统，在主程序中完成函数调用所需实参值的准备和函数执行结果。

3.1.1 二叉树抽象数据类型

依据最小完备性和常用性相结合的原则，设计了线性表的数据对象和数据关系，并定义了二叉树的初始化二叉树、销毁二叉树、创建二叉树、清空二叉树、判定空二叉树、求二叉树的深度、获得根结点、获得结点、结点赋值、获得双亲结点、获得左孩子结点、获得右孩子结点、获得左兄弟结点、获得右兄弟结点、插入子树、删除子树、前序遍历、中序遍历、后序遍历、层次遍历20种基本运算，具体数据和运算功能定义如下。

ADT BiTree{

**数据对象D：**D是具有相同特性的数据元素的集合

**数据关系R：**

若D=Ø，则D为空二叉树

若D!=Ø，则R={H}，H是如下的关系：

1. 在D中存在惟一的称为根的数据元素root，它在关系H下无前驱；
2. 若D-{root}!= Ø,则存在D-{root}={D1,Dr},且D1∩Dr=Ø；
3. 若D1!= Ø，则D1中存在惟一的元素x1,<root,x1>∈H,且存在D1上的关系H1∈H；若Dr!= Ø,则Dr中存在惟一的元素xr,<root,xr>∈H，且存在Dr上的关系Hr∈H；H={<root,x1>,<root,xr>,H1,Hr}；
4. (D1，{H1})是一颗符合本定义的二叉树，称为根的左子树，（Dr，{Hr}）是一颗符合本定义的二叉树，称为根的右子树。

**基本操作P:**

InitBiTree(&L):

初始条件:无。

操作结果:构造空的二叉树。

DestroyBiTree(L,F):

初始条件:顺序表L已存在。

操作结果:销毁顺序表L中的一个数据，即销毁一个二叉树。

CreateBiTree(definition)：

初始条件:二叉树存在且为空

操作结果:按definition构造二叉树。

ClearBiTree(T):

初始条件:二叉树存在。

操作结果:将二叉树清空。

BiTreeEmpty(T):

初始条件:二叉树存在。

操作结果:若二叉树为空二叉树则返回OK，否则返回ERROR。

BiTreeDepth(T):

初始条件:二叉树存在。

操作结果:返回二叉树的深度。

Root(T):

初始条件:二叉树存在。

操作结果:返回二叉树的根结点。

Value(T，e):

初始条件:二叉树存在。

操作结果:返回关键字为e的结点，并输出其数据，失败返回NULL。

Assign(T,e,c):

初始条件:二叉树存在。

操作结果:将关键字为e的结点的数据改为c，失败返回ERROR。

Parent(T,e):

初始条件:二叉树存在。

操作结果:返回关键字为e的结点的双亲结点，失败返回NULL。

LeftChild(T,e):

初始条件:二叉树存在。

操作结果:返回关键字为e的结点的左孩子结点，失败返回NULL。

RightChild(T,e):

初始条件:二叉树存在。

操作结果:返回关键字为e的结点的右孩子结点，失败返回NULL。

LeftSibling(T,e):

初始条件:二叉树存在。

操作结果:返回关键字为e的结点的左兄弟结点，失败返回NULL。

RightSibling(T,e):

初始条件:二叉树存在。

操作结果:返回关键字为e的结点的右兄弟结点，失败返回NULL。

InsertChild(T,e,LR,p):

初始条件: 二叉树存在，且构建子树p成功。

操作结果:将p子树通过change函数变化为一个右子树为空的子树后，将变化后的子树p插入到关键字为e的结点的左子树（LR=0）或右子树(LR=1)上。

change(T):

初始条件:二叉树存在。

操作结果:若二叉树的根结点的右子树不为空树，则通过树变二叉树的规则将二叉树转变为右子树为空的二叉树。

DeleteChild(T,e,LR):

初始条件:二叉树存在。

操作结果:删除二叉树中关键字为e的结点的左子树（LR=0）或右子树(LR=1)。

PreOrderTraverse(T):

初始条件:二叉树存在。

操作结果:输出先序遍历二叉树的序列。

InOrderTraverse(T):

初始条件:二叉树存在。

操作结果:输出中序遍历二叉树的序列。

PostOrderTraverse(T):

初始条件:二叉树存在。

操作结果:输出后序遍历二叉树的序列。

LevelOrderTraverse(T):

初始条件:二叉树存在。

操作结果:输出层次遍历二叉树的序列。

updata(T):

初始条件：二叉树存在且不为空树

操作结果：将二叉树中每一个结点的数据域中的Key型关键字赋值为其在完全二叉树中对应的位置。

} ADT BiTree

3.1.2 多二叉树抽象数据类型

基于之前实现的线性表数据类型，设计了多线性表的数据对象、数据关系，并以定义了多线性表的初始化、销毁、遍历、切换共4种基本运算，具体数据和运算功能定义如下。

ADT Forest{

**数据对象D:**D = { ei|ei ∈ SqList, i = 1,2, ……,n, n≥0 }

**数据关系R:**R1 = { <ei-1,ei>|ei-1,ei∈D, i = 2,……,n }

**基本操作P:**

SaveData(fp,T,filename):

初始条件:二叉树存在。

操作结果: 将二叉树中的数据按先序遍历顺序存入文件。

LoadData(fp,T,filename):

初始条件:二叉树存在且为空。

操作结果:读入文件字符串，并依此构建二叉树。

ForestTraverse(L):

初始条件:顺序表L已存在。

操作结果: 按先序遍历输出L中每个数据即每棵树的数据信息。

ExBiTree(L,name):

初始条件:无。

操作结果: 将当前活动树切换为树名为name的二叉树，若无则不进行改变。

}ADT Forest

3.1.3 演示系统与数据文件

构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

演示系统可选择实现二叉树的文件形式保存。其中，设计文件数据记录格式，以高效保存二叉树数据逻辑结构(D,{R})的完整信息；并设计了二叉树文件保存和加载操作合理模式。

演示系统基于顺序表实现的多个二叉树管理。采用顺序表的方式管理多个二叉树，线性表中的每个数据元素为一个二叉树的基本属性，包含有二叉树的名称和头结点。

## 3.2 系统设计

3.2.1 数据物理结构

在本程序中，数据关键字类型Key被定义为int整型,且由系统自动生成。

1. 每个节点的数据域的存储结构。

typedef struct TElemType {

Key i; //关键字，本系统采用完全二叉树中对应的位置

char c; //字符型数据

}TElemType;

1. 二叉树结点的存储结构。

typedef struct BiTNode {

TElemType data; //二叉树结点中的数据域

BiTNode \*lchild, \*rchild; //左右孩子指针

}BiTNode;

1. 二叉树的存储结构。

typedef struct BiTree {

char name[20]; //二叉树的名字

BiTNode\* HeadNode; //二叉树根结点

}BiTree;

1. 多个二叉树的顺序表存储结构。

typedef struct Forest {

BiTree \*elem; //数据域

int length; //二叉树的个数

int listsize;

}Forest;

整体存储结构如图3-1所示：

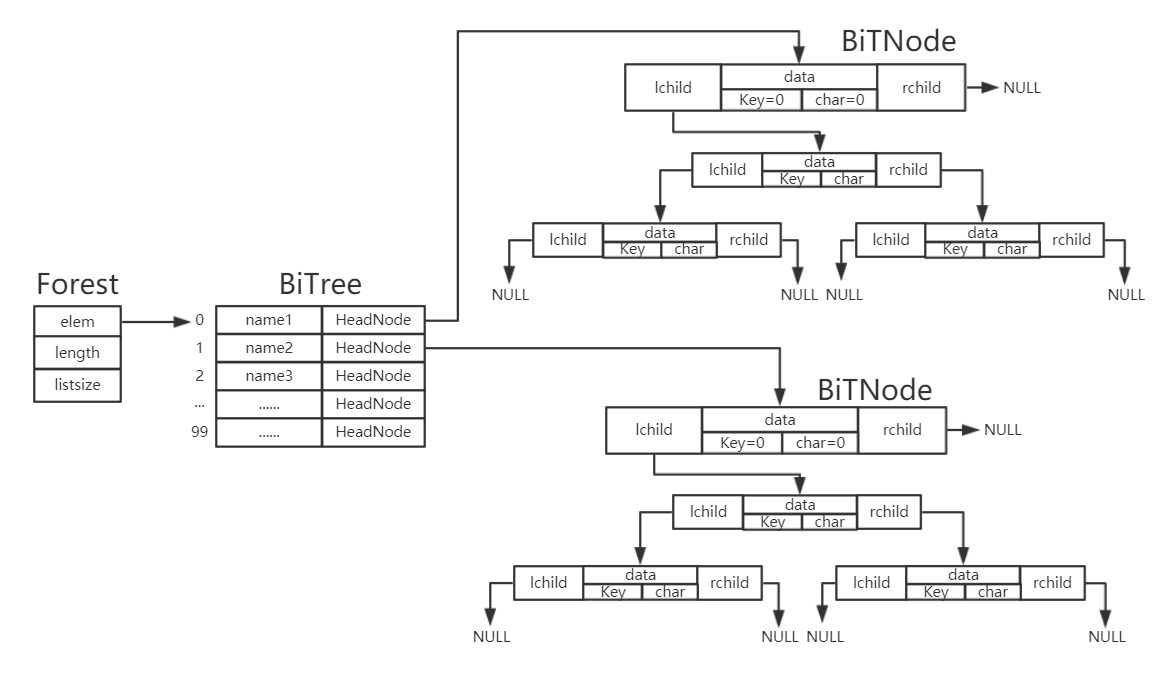


图3-1 多二叉树的存储结构

3.2.2 演示系统

演示系统由用户操作界面与功能调用部分组成，如图3-2所示。

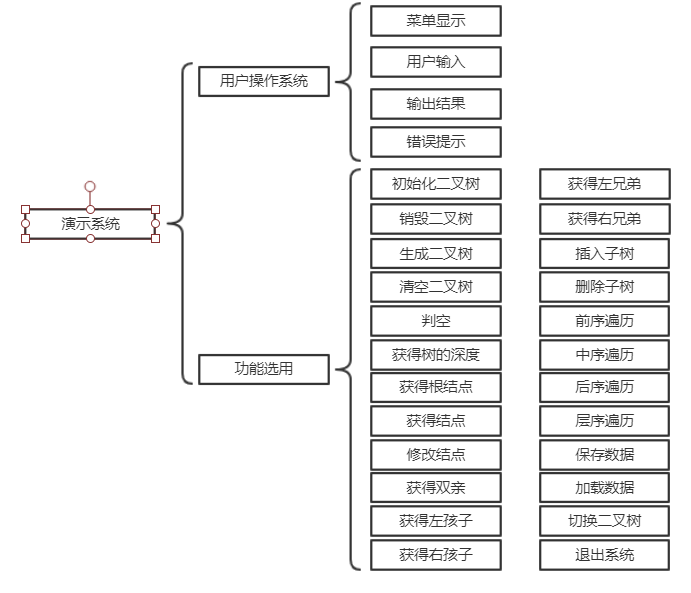


图3-2 演示系统模块结构图

3.2.3 数据文件存储格式

考虑到方便查看,以txt格式直接存放二叉树的数据，若为空则存入“#”。

3.2.4 二叉树运算实现算法

1. InitBiTree(&L):

**算法思想：**输入新树的名字，通过遍历顺序表确认拥有该名字的树是否已存在，若存在则输出提示信息并返回ERROR，若不存在则进行后续操作。若顺序表空间已满则开辟新空间，分配空间失败则输出提示信息并返回ERROR，否则L.listsize=L.listsize+LISTINCREMENT，并进行后序操作。对顺序表的内容进行赋值和开辟空间，顺序表长度L.length自增1，返回OK。

**时间复杂度分析：**假设顺序表中共有n个存储有二叉树的元素，且名字与任一二叉树重复的可能均相等，则算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

2.DestroyBiTree(L,F):

**算法思想：**遍历顺序表寻找到名字与F名字相同的元素，将该元素中树的根结点传入DestroyBiTree2函数。在退出DestroyBiTree2函数后利用循环将顺序表在该元素后面的元素依次前移，并将最后一个元素的头指针置空，最后令顺序表长度L.length自减1，返回OK。

**时间复杂度分析：**调用DestroyBiTree2函数，假设所需销毁的二叉树在顺序表中不同位置的概率均相同且该二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

3. DestroyBiTree2(T)：

**算法思想：**采用递归算法，若结点为NULL，返回OK，若非空则先销毁其左子树，再销毁其右子树。

**时间复杂度分析：**采用递归算法，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

4.CreateBiTree(definition)：

**算法思想：**首先判断definition的首元素，若为‘\n’或者‘#’ 则输出提示信息并返回ERROR，否则调用PreCreateBiTree函数。成功生成二叉树后将其根结点的数据域int型值赋值为1，之后返回这个根结点。

**时间复杂度分析：**调用PreCreateBiTree函数，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

5.PreCreateBiTree(&T,s,&i)：

**算法思想：**采用递归算法，其参数为二叉树结点T、字符串s以及定位字符串中元素位置的i，首先判断s[i]是否为‘#’，若是返回NULL，否则判断是否为'\n'或' '，若是则返回ERROR，否则进行其后操作。为T结点开空间并对其数据域进行赋值，之后调用函数生成T的左子树，失败返回ERROR，成功调用函数生成T的右子树，失败返回ERROR，成功返回OK。

**时间复杂度分析：**采用递归算法，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

6.ClearBiTree(T):

**算法思想：**若最初T结点为空，则返回OK，否则调用DestroyBiTree2函数销毁T->lchild(即二叉树根结点)，销毁失败返回ERROR，成功则将T->lchild指向NULL，返回OK。

**时间复杂度分析：**调用DestroyBiTree2函数，假设二叉树中有n个结点，算法的空间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

7.BiTreeEmpty(T):

**算法思想：**若T的左孩子结点为空则返回OK，否则返回ERROR。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

8.BiTreeDepth(T):

**算法思想：**采用递归算法，首先初始化一个临时变量deep，并赋值为0。若T非空则初始化临时变量leftdeep存储T的左子树的深度，初始化临时变量rightdeep存储T的右子树的深度，将leftdeep和rightdeep中较大的一个值加一后赋值给deep，最后返回deep。

**时间复杂度分析：**采用递归算法，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**声明了三个临时变量deep、liftdeep和rightdeep，假设二叉树中有n个结点，算法的空间复杂度为线性阶O(n)。

9.Root(T):

**算法思想：**直接返回T的左孩子结点，即为二叉树的根结点。

**时间复杂度分析：**算法的时间复杂度为常量阶O(1)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

10.Value(T,e):

**算法思想：**采用递归算法，首先判断T是否为NULL，若是返回T，否则判断T的数据域中的i值是否为e，若是则返回T，否则进行后续操作。声明一个结构指针temp存入在T的左子树中查找关键字为e的结点时的返回值，若temp为空，则为temp开辟空间并存入在T的右子树中查找关键字为e的结点时的返回值，最后返回temp。

**时间复杂度分析：**采用递归算法，假设二叉树中有n个结点且查找结果为每个结点的概率均相同，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**声明了一个临时变量temp，假设二叉树中有n个结点，算法的空间复杂度为线性阶O(n)。

11.Assign(T,e,c):

**算法思想：**声明结构指针temp，并存入调用Value函数查找关键字为e的结点的返回值，若temp为NULL，则返回ERROR，否则将temp的数据域c值更改为c，并返回OK。

**时间复杂度分析：**调用Value函数，假设二叉树中有n个结点且需更改的结点为每个结点的概率均相同，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

12.Parent(T,e):

**算法思想：**采用递归算法，先判断若T的左孩子非空且左孩子的关键字为e或者T的右孩子非空且右孩子的关键字e(即T为所找结点)，则返回T。否则声明一个结构指针temp，若T的左孩子非空则temp存入在T的左子树中查找关键字为e的结点的双亲结点时的返回值，若temp非空且T的右孩子非空则temp存入在T的右子树中查找关键字为e的结点的双亲结点时的返回值，最后返回temp。

**时间复杂度分析：**采用递归算法，假设二叉树中有n个结点且所要查找的结点为每个结点的概率相同，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(n)。

13.LeftChild(T,e):

**算法思想：**声明一个结构指针temp并存入调用Value函数查找关键字为e的结点的返回值，若temp为空则返回NULL，否则返回temp的左孩子结点。

**时间复杂度分析：**调用Value函数，假设二叉树中有n个结点且所要查找的结点为每个结点的概率相同，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(n)。

14.RightChild(T,e):

**算法思想：**声明一个结构指针temp并存入调用Value函数查找关键字为e的结点的返回值，若temp为空则返回NULL，否则返回temp的右孩子结点。

**时间复杂度分析：**调用Value函数，假设二叉树中有n个结点且所要查找的结点为每个结点的概率相同，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(n)。

15.LeftSibling(T,e):

**算法思想：**声明一个结构指针temp并存入调用Parent函数查找关键字为e的结点的双亲结点的返回值，若temp为空则返回NULL，否则返回temp的左孩子结点。

**时间复杂度分析：**调用Parent函数，假设二叉树中有n个结点且所要查找的结点为每个结点的概率相同，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(n)。

16.RightSibling(T,e):

**算法思想：**声明一个结构指针temp并存入调用Parent函数查找关键字为e的结点的双亲结点的返回值，若temp为空则返回NULL，否则返回temp的右孩子结点。

**时间复杂度分析：**调用Parent函数，假设二叉树中有n个结点且所要查找的结点为每个结点的概率相同，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(n)。

17.InsertChild(T,e,LR,p):

**算法思想：**算法流程图入3-3所示：

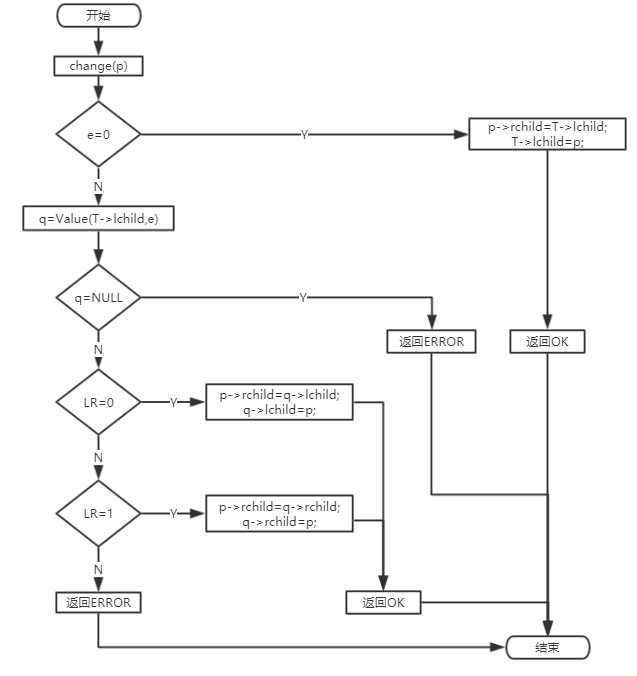


图3-3 InsertChild函数的算法流程图

**时间复杂度分析：**调用change、Value函数，假设二叉树中有n个结点且插入每个位置的概率相同，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

18.change(T):

**算法思想：**采用递归算法，先判断T是否为空，若为空则返回上一层，否则以T的左子树和右子树为参数依次调用change函数，若T的左孩子为空，则将T的左孩子指向原本的右孩子，否则将T的左孩子的右孩子指向T的右孩子，最后将T的右孩子指向空并退出该函数。

**时间复杂度分析：**采用递归算法，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

19.DeleteChild(T,e,LR):

**算法思想：**算法流程图如3-4所示：

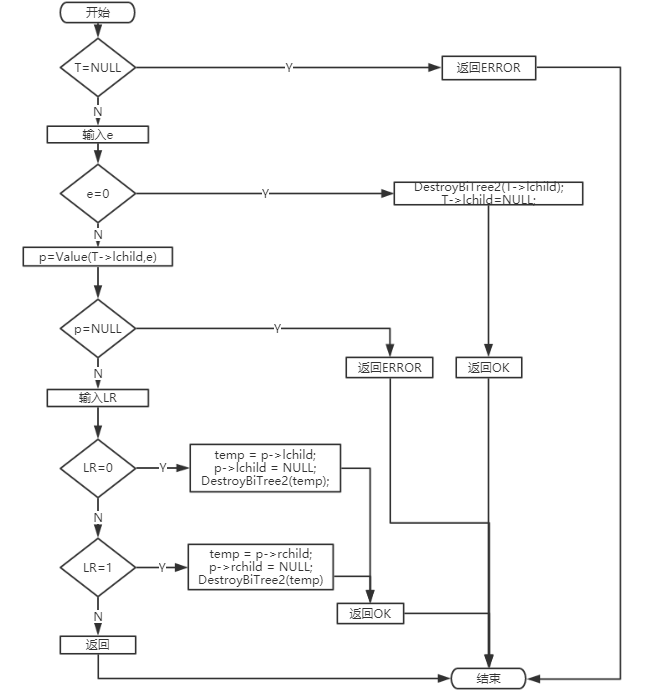


图3-4 DeleteChild函数的算法流程图

**时间复杂度分析：**调用change、DestroyBiTree2函数，假设二叉树中有n个结点且删除每个结点的概率相同，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(n)。

20.PreOrderTraverse(T):

**算法思想：**若T为空输出“#”，否则输出T的数据域中的c值，前序遍历T的左子树，前序遍历T的右子树，最后返回OK。

**时间复杂度分析：**采用递归算法，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

21.InOrderTraverse(T):

**算法思想：**若T为空输出“#”，否则中序遍历T的左子树，输出T的数据域中的c值，中序遍历T的右子树，最后返回OK。

**时间复杂度分析：**采用递归算法，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

22.PostOrderTraverse(T):

**算法思想：**若T为空输出“#”，否则后序遍历T的左子树，后序遍历T的右子树，输出T的数据域中的c值，最后返回OK。

**时间复杂度分析：** 采用递归算法，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

23.LevelOrderTraverse(T):

**算法思想：**借助队列实现层序遍历，算法流程图如3-5所示：

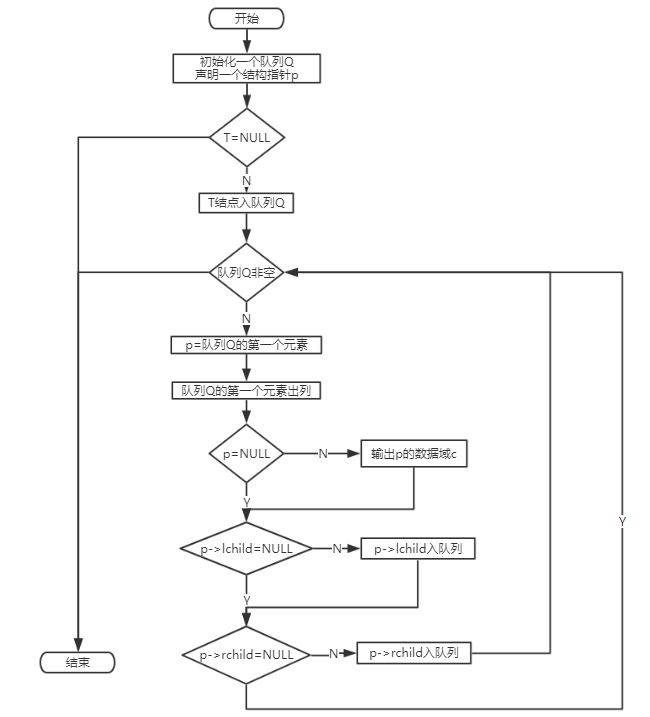


图3-5 LevelOrderTraverse函数的算法流程图

**时间复杂度分析：**假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

24.updata(T):

**算法思想：** 算法流程图如图3-6所示：

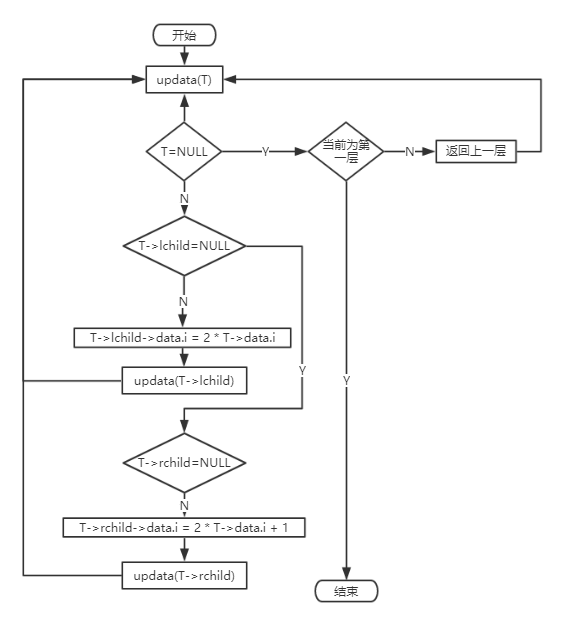


图3-6 updata函数的算法流程图

**时间复杂度分析：** 采用递归算法，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

3.2.5 多二叉树运算实现算法

1.SaveData(fp,T,filename):

**算法思想：**输入文件名filename，用fp存储fopen函数打开名为filename的文件的返回值，若为NULL则输出提示信息并返回ERROR，否则调用SaveData1函数，若返回则关闭文件指针fp并返回OK，否则关闭文件指针fp并返回ERROR。

**时间复杂度分析：** 调用SaveData1函数，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

2.SaveData1(fp,T,filename):

**算法思想：**采用递归算法，若T非空，则将T的数据域c值写入文件，之后依次以T的左孩子结点和T的右孩子结点为参数调用SaveData1函数，若T为空则将“#”写入文件。

**时间复杂度分析：** 采用递归算法，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

3.LoadData(fp,T,filename):

**算法思想：**输入文件名字filename，用fp存储fopen函数打开名为filename的文件的返回值，若为NULL则输出提示信息并返回ERROR，否则用字符串型临时变量temp存储文件中的字符串。若T非空则将T清空，清空失败返回ERROR,清空成功则进行后序操作。若T为空则以temp为参数调用CreateBiTree函数，判断若T==NULL且temp=='\0'或temp!='\0'且T!=NULL则关闭文件指针fp并返回OK。否则关闭指针fp并返回ERROR。

**时间复杂度分析：** 调用CreateBiTree函数和ClearBiTree函数，假设二叉树中有n个结点，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(n)。

4.ForestTraverse(L):

**算法思想：**遍历顺序表L并输出树的名字，若树为空则输出该树为空树，否则以前序遍历输出当前元素所存储二叉树的前序遍历序列。退出循环后输出遍历结束的提示信息并返回OK。

**时间复杂度分析：**调用PreOrderTraverse函数，假设线性表中有n个非空元素，且第i个二叉树中有ni个结点(0<i<n+1)，算法的时间复杂度为线性阶O(∑ni)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

5.ExBiTree(L,name):

**算法思想：**首先遍历顺序表L查找名字为name的元素位置，若没有查到则返回NULL，若查到则返回&(L.elem[i])。

**时间复杂度分析：**假设线性表中有n个非空元素，且所要查找的元素为每个元素的概率均相同，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

## 3.3 系统实现

3.3.1 实验环境

实验代码使用dev c++编写与编译.

3.3.2 演示系统操作

演示系统采用键盘操作，控制台输出操作提示与结果，第一行显示当前操作表（默认为无），中间显示可用的操作项，输入对应的操作指令即可执行，输入非0~23的操作号会提示错误。

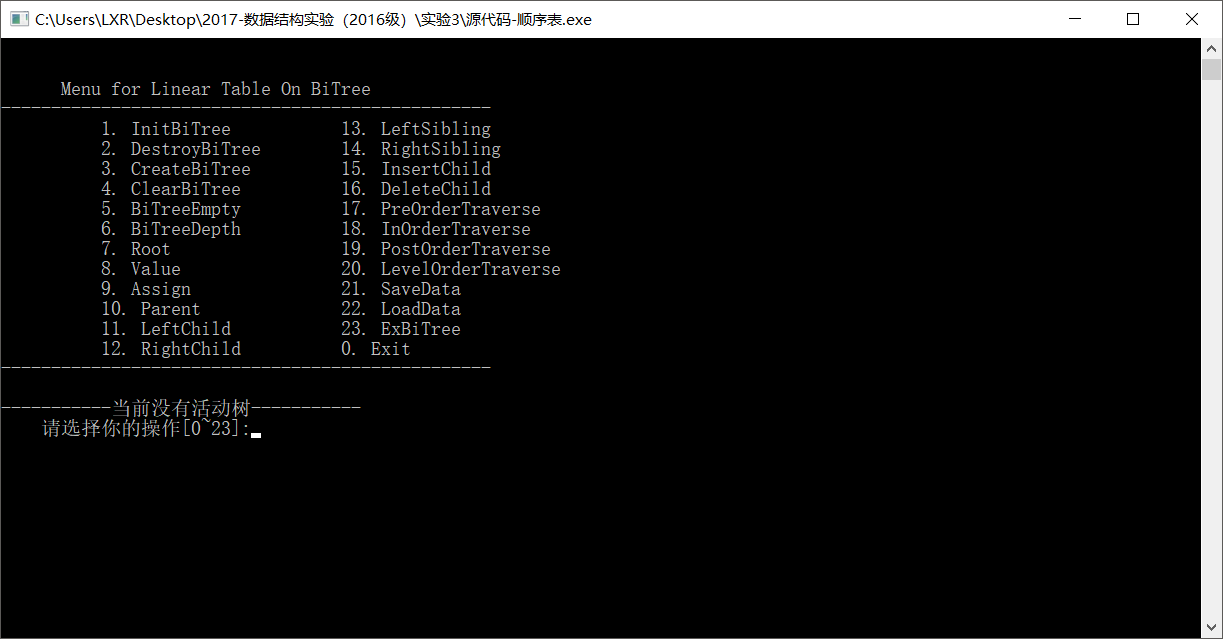


图3-7 启动程序后的操作菜单

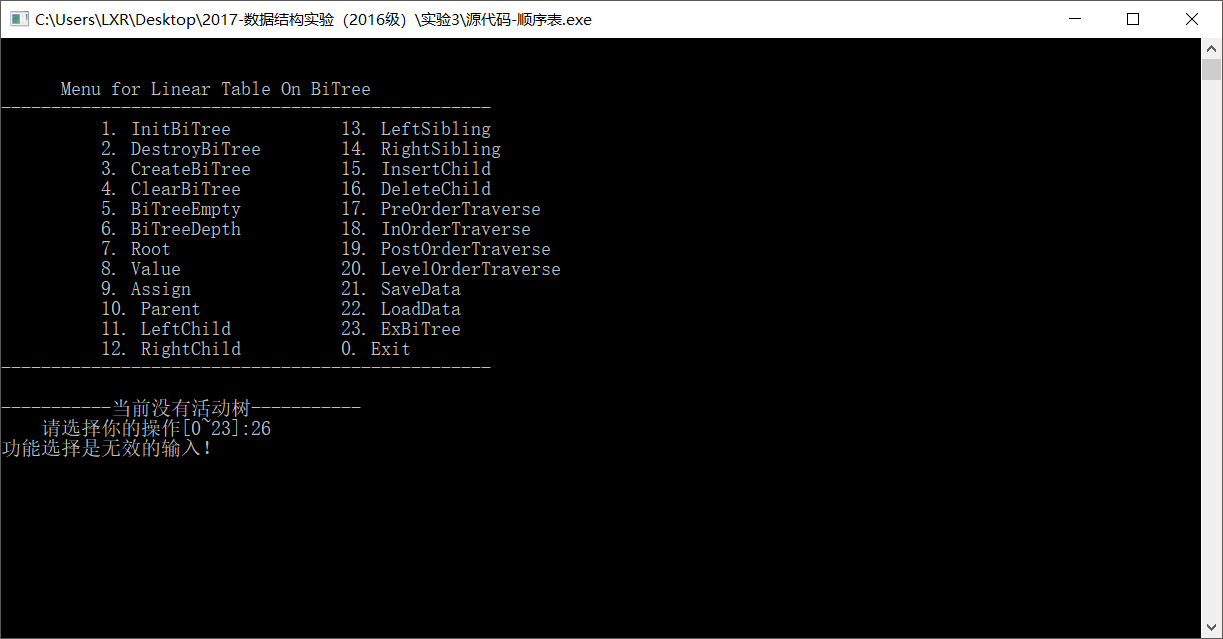


图3-8 输入错误的功能选择数字，显示提示信息

输入操作指令0，程序退出。

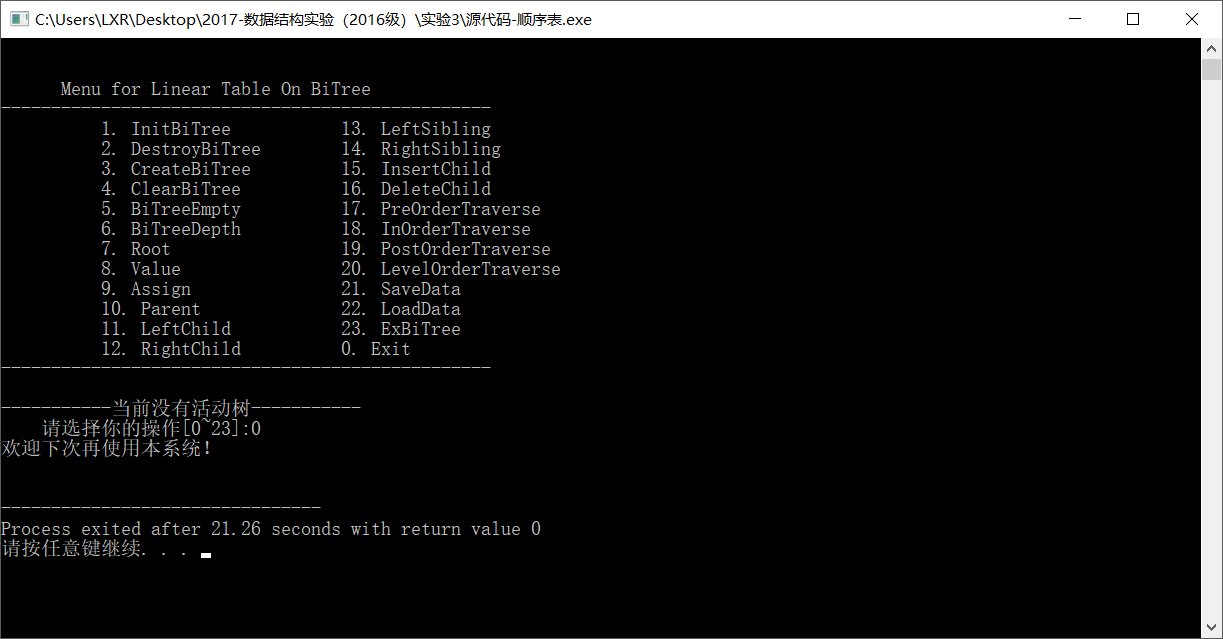


图3-9 输入操作0程序退出

3.3.3 测试计划

1.检测常规状态下系统的工作情况；

2.重点检查非法边界操作时的工作情况。

表3-1 演示系统测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能 | 测试顺序 | 测试输入 | 预计输出 | 二叉树状态 |
| InitBiTree | 4  13  59  60 | 4/60：zxc  13/59：asd | 4/13/60:创建一个空二叉树成功  59:构建失败 | 4/60：空树，名为zxc  13:空树，名为asd  59：同58 |
| DestroyBiTree | 1  11 | 无 | 1：无活动树  11：销毁成功 | 1：未初始化  11：销毁后不存在树 |
| CreateBiTree | 2  12  14  19  56  57  58  61 | 2/12：无  14:AB##C##  19：无  56：\n(输入换行符)  57：A#  58：A##  61:E## | 2/12：无活动树  14:创建成功  19：非空不能进行创建  56：未输入字符生成失败  57: 序列错误构建二叉树失败  58/61：构建成功 | 2/12：同1  14/19：树名asd，由先序序列AB##C##生成的二叉树  56/57：同53  58：树名asd，先序序列A##  61:树名zxc,先序序列E## |
| ClearBiTree | 3 | 3：无 | 3：无活动树 | 3：同1 |
| BiTreeEmpty | 5  20  55 | 无 | 5/55：为空  20：不为空 | 5：同4  20：同14  55：同53 |
| BiTreeDepth | 6  21 | 无 | 6：深度为0  21：深度为2 | 6：同4  21：同14 |
| Root | 7  22 | 无 | 7：根结点为空，该树为空二叉树  22：根结点存放的数据是A | 7：同4  22：同14 |
| Value | 8  23  24 | 8：无  23：2  24：0 | 8：为空不能查找  23：关键字为2的数据为B  24：没有找到 | 8：同4  23/24：同14 |
| Assign | 9  25 | 9：无  25：3 D | 9：为空不能修改  25：修改成功 | 9：同4  25：树名asd，原结点C的char数据域改为D |
| Parent | 27  28  29 | 27：1  28：2  29：4 | 27：为根不能查找  28：关键字：1 数据：A  29：没有找到 | 27/28/29：同25 |
| LeftChild | 30  31  32 | 30:1  31:2  32:4 | 30:2 B  31/32：没有找到结点或没有左孩子结点 | 30/31/32：同25 |
| RightChild | 33  34  35 | 33：1  34：3  35：0 | 33：3 D  34/35：没有找到结点或没有右孩子结点 | 33/34/35：同25 |
| LeftSibling | 36  37  38  39 | 36：1  37：2  38：3  39：4 | 36：为根，无左兄弟结点  37：为左孩子，无左兄弟结点  38：2 B  39：没有找到 | 36/37/38/39：同25 |
| RightSibling | 40  41  42  43 | 40：1  41：2  43：3  43：0 | 40：为根，无右兄弟结点  41：3 D  42：为右孩子，无右兄弟结点  43：没有找到 | 40/41/42/43：同25 |
| InsertChild | 45  46  47  48 | 45：A#  46：#  47：\n(输入换行符)  48：E#F## 1 0 | 45:序列错误构建子树失败  46：子树为空无法插入  47：未输入字符构建失败  48：插入成功 | 45/46/47：同25  48：树名asd，先序序列为AEF##B##D## |
| DeleteChild | 50  51  53 | 50:10  51:1 1  53：0 | 50:未找到结点删除失败  51：删除成功  53：删除成功 | 50：同48  51：树名asd，先序序列为AEF##B###  53：树名asd,空树 |
| PreOrderTraverse | 10  15  26  49  52  54  64  67 | 无 | 10/54：为空树  15：AB##C##  26/64：AB##D##  49: AEF##B##D##  52: AEF##B###  67:A## | 10：同4  15:同14  26/64:同25  49:同48  52:同51  54:同53  67:同58 |
| InOrderTraverse | 16 | 无 | 16：#B#A#C# | 16:同14 |
| PostOrderTraverse | 17 | 无 | 17：##B##CA | 17:同14 |
| LevelOrderTraverse | 18 | 无 | 18:ABC | 18:同14 |
| SaveData | 44 | data.txt | 写入成功 | 44：同25 |
| LoadData | 62  63 | 62：da.txt  63：data.txt | 62：文件打开失败  63:读入成功 | 62：同61  63:同25 |
| ExBiTree | 65  66 | 65:z  66：asd | 65:二叉树不存在，切换失败  66:切换成功 | 65：同25  66：同58 |
| Exit | 68 | 无 | 欢迎下次再使用本系统 | 68：无二叉树 |

输入操作指令0，程序退出。

3.3.4 测试结果

测试用例一：

未初始化二叉树时，操作对应测试计划1-3。

测试结果：（在进行其他操作时同样提示当前没有活动表）

（1）销毁二叉树，因当前没有初始化二叉树，所以会显示提示信息，并说明销毁二叉树失败。

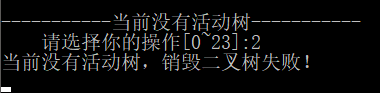


图3-10 销毁二叉树

（2）生成二叉树，因当前二叉树未初始化所以不能进行生成操作。

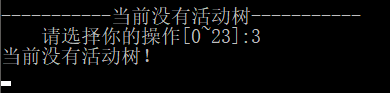


图3-11 生成二叉树

（3）清空二叉树，因当前二叉树未初始化所以无法进行清空操作。

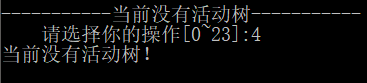


图3-12 请空二叉树

测试用例二：

初始化二叉树，此时二叉树为空树。操作对应测试用例4-12。

测试结果：

（1）调用初始化二叉树函数，初始化一个空的二叉树，树名为zxc。

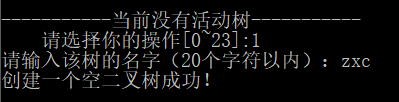


图3-13 初始化二叉树

（2）调用判定该二叉树是否为空的功能函数。

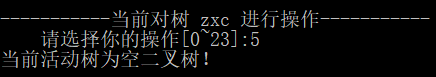


图3-14 判空

（3）调用返回二叉树深度的功能函数，当前二叉树是空树，所以深度为0。

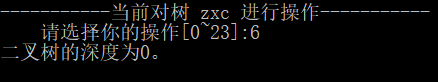


图3-15 返回二叉树的深度

（4）调用获得根结点的功能函数，当前二叉树是空树，所以根结点为空。

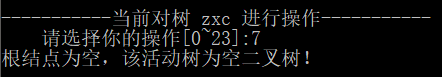


图3-16 返回根结点

（5）调用获得某一结点的功能函数，因为当前二叉树是空树，不能进行查找。

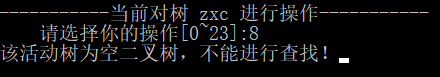


图3-17 获得结点

（6）调用修改某一结点的功能函数，因为当前二叉树是空树，不能进行修改。

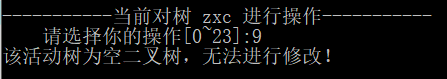


图3-18 修改结点

（7）调用前序遍历二叉树的功能函数，因为当前二叉树是空树，所以只输出提示信息该树为空二叉树。

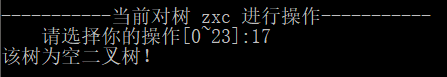


图3-19 前序遍历二叉树

（8）调用销毁二叉树的功能函数，虽然当前二叉树是空树，但可以进行销毁操作，销毁的结果的森林顺序表中该树所在位置的空间被释放，其后的非空数据依次前移。

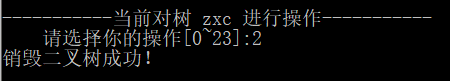


图3-20 销毁二叉树

（9）调用生成二叉树的功能函数，因当前二叉树已在上一步骤中被销毁，所以当前没有二叉树，不能进行生成二叉树的操作。

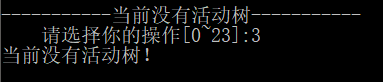


图3-21 生成二叉树

测试用例三：

初始化一个空二叉树，名称为asd。按definition生成一个二叉树，对二叉树进行一系列操作，对应测试用例13-58。

测试结果：

（1）调用初始化二叉树的功能函数，初始化一个名称为asd的空二叉树。

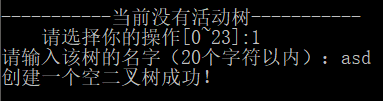


图3-22 初始化二叉树

（2）调用生成二叉树的功能函数，输入一个正确的先序遍历序列，其中空用‘#’表示且每个元素都是char类型，则构建二叉树成功。

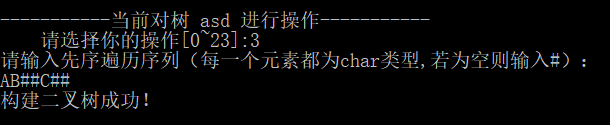


图3-23 生成二叉树

（3）依次调用先序遍历二叉树、中序遍历二叉树、后序遍历二叉树、层次遍历二叉树的功能函数，输出不同的遍历序列。

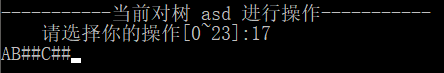


图3-24 前序遍历二叉树

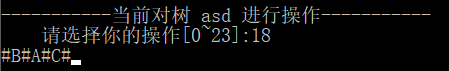


图3-25 中序遍历二叉树

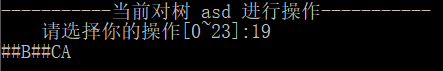


图3-26 后序遍历二叉树

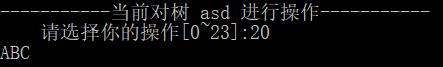


图3-27 层序遍历二叉树

（4）再次调用生成二叉树的功能函数，因为当前活动树中已经存入数据即当前活动树是非空二叉树，则不能重新进行生成操作。

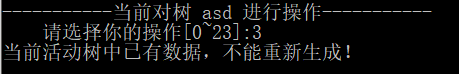


图3-28 生成二叉树

（5）调用判断二叉树是否为空的功能函数，因为当前活动树不是空二叉树，所以输出提示信息。

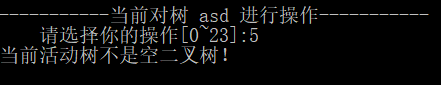


图3-29 判空

（6）调用返回二叉树深度的功能函数，因当前二叉树的前序序列为AB##C##，所以显然二叉树深度为2。

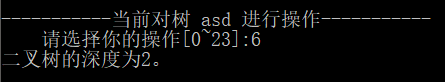


图3-30 返回二叉树的深度

（7）调用返回根结点的功能函数，当前根结点的数据域char型数据为A,Key型数据为1。

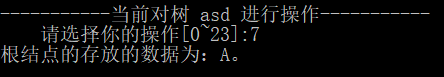


图3-31 返回根结点

（8）调用获得结点的功能函数，依次输入一个合法的关键字2和一个非法关键字0，其中非法指不存在含有该关键字的结点。在输入2时输出B；在输入0时输出没有找到。

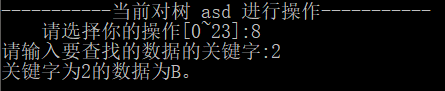


图3-32 获得结点

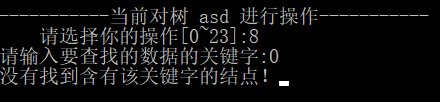


图3-33 获得结点

（9）调用修改结点数据域的功能函数，将关键字为3的结点的数据修改为D，并调用先序遍历二叉树的功能函数，通过观察输出的先序序列确认修改成功。

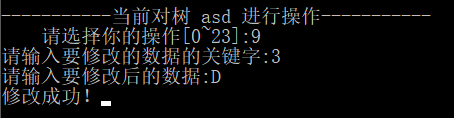


图3-34 修改结点

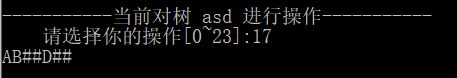


图3-35 修改结点

（10）调用返回双亲结点的功能函数，依次输入1、2、4。输入1时因找到的结点为根结点，所以输出提示信息该结点为根结点，无双亲结点；输入2时正确查找双亲结点，最后返回关键字为1、数据为A的结点；输入4时因4时非法的关键字，所以输出没有找到结点，查找失败。

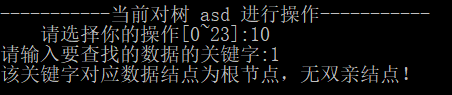


图3-36 获得双亲结点

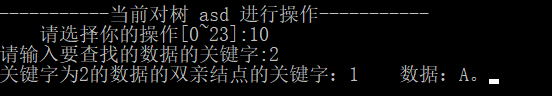


图3-37 获得双亲结点

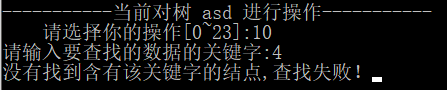


图3-38 获得双亲结点

（11）调用返回左孩子结点的功能函数，依次输入1、2、4。输入1时正确输出关键字为2、数据为B的左孩子结点；输入2时，因为关键字为2的结点无左孩子结点，所以输出提示信息，查找失败；输入4时，因为关键是非法，所以输出提示信息，查找失败。

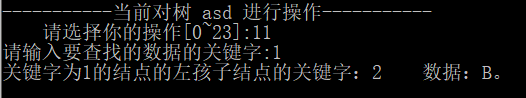


图3-39 获得左孩子结点

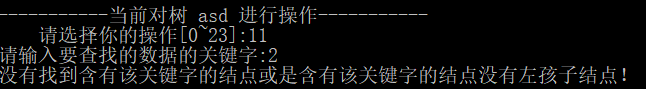


图3-40 获得左孩子结点

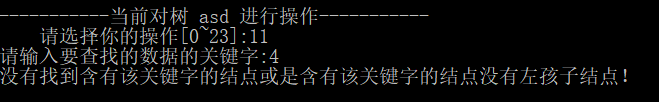


图3-41 获得左孩子结点

（12）调用返回右孩子结点的功能函数，依次输入1、3、0。输入1时正确输出关键字为3、数据为C的右孩子结点；输入3时，因为关键字为3的结点无右孩子结点，所以输出提示信息，查找失败；输入0时，因为关键是非法，所以输出提示信息，查找失败。

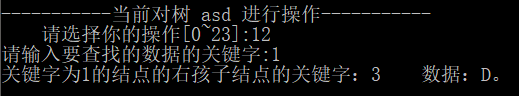


图3-42 获得右孩子结点

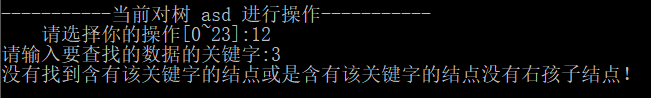


图3-43 获得右孩子结点

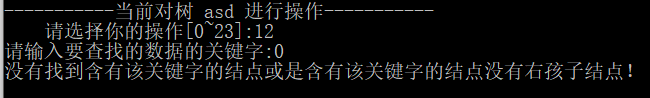


图3-44 获得右孩子结点

（13）调用返回左兄弟结点的功能函数，依次输入1、2、3、4。输入1时因为查找到的为根结点，所以无左兄弟结点且输出提示信息；输入2时，因为关键字为2的结点本身为其双亲结点的左孩子结点，所以无左兄弟结点并输出提示信息；输入3时，正确返回关键字为2、数据为B的左兄弟结点；输入4时，因为关键是非法，所以输出提示信息，查找失败。

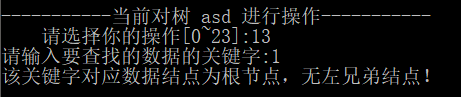


图3-45 获得左兄弟结点

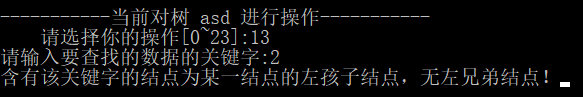


图3-46 获得左兄弟结点

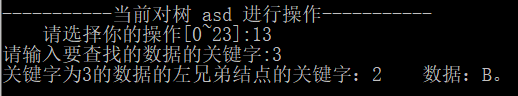


图3-47 获得左兄弟结点

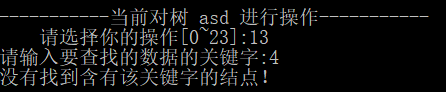


图3-48 获得左兄弟结点

（14）调用返回右兄弟结点的功能函数，依次输入1、2、3、0。输入1时因为查找到的为根结点，所以无右兄弟结点且输出提示信息；输入2时，正确返回关键字为3、数据为C的右兄弟结点；输入3时，因为关键字为3的结点本身为其双亲结点的右孩子结点，所以无右兄弟结点并输出提示信息；输入0时，因为关键是非法，所以输出提示信息，查找失败。

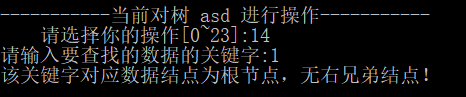


图3-49 获得右兄弟结点

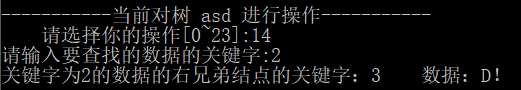


图3-50 获得右兄弟结点

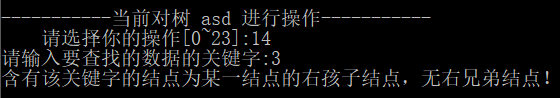


图3-51 获得右兄弟结点

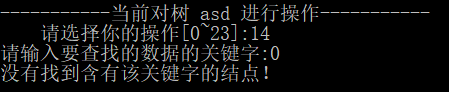


图3-52 获得右兄弟结点

（15）调用文件写入的功能函数，将当前二叉树的先序遍历序列写入文件data.txt，通过查看txt文件确认是否写入成功。

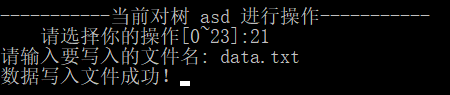


图3-53 录入数据

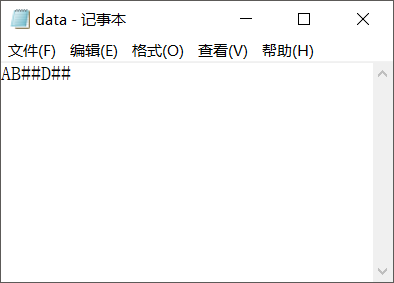


图3-54 data.txt文件

（16）调用插入子树的功能函数，依次输入子树的先序序列“A#”、“#”、“\n(换行符)”。输入A#时因A#时不完整的先序遍历序列所以提示构建二叉树失败；输入#时因单指一个空结点，所以生成一颗空树，但无法进行插入操作；输入换行符时因未输入任何字符所以构建二叉树失败。最后输入一个正确的先序遍历序列E#F##构建一个子树，再调用change函数用树变二叉树的方式将子树变为一个右子树为空的二叉树，将子树插入指定位置的同时，将以其指定位置上原有结点为根结点的子树连接在子树的右子树上，通过调用先序遍历的功能函数确认插入是否正确。

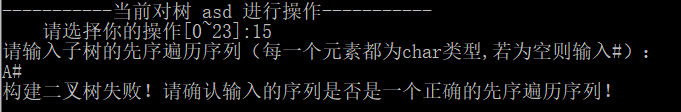


图3-55 插入子树

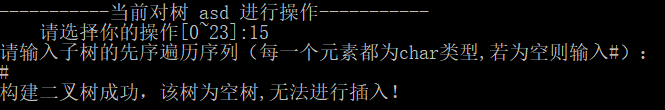


图3-56 插入子树

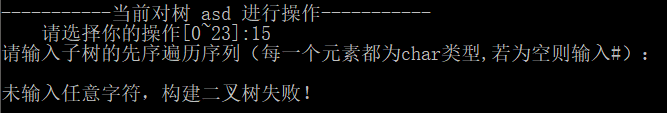


图3-57 插入子树

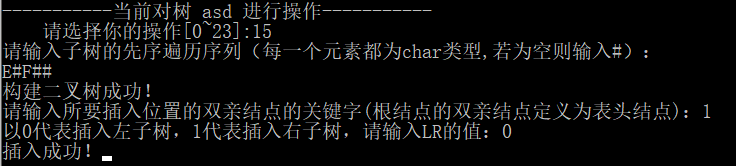


图3-58 插入子树

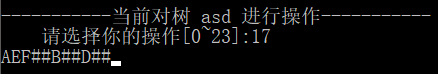


图3-59 插入子树

（17）调用删除子树的功能函数，依次输入10、1&1、0。输入10时，因10时非法的关键字所以输出提示信息该结点不存在删除失败；输入1&1时，删除的是根结点的左子树，通过调用先序遍历的功能函数确认删除成功；输入0时，因定义根结点的双亲结点为表头结点所以删除后的该活动树变成一个空树，通过调用先序遍历二叉树和判断二叉树是否为空的功能函数确认删除成功。

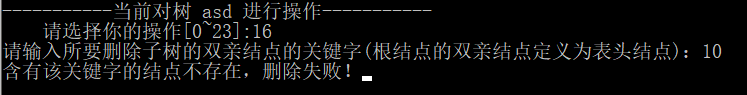


图3-60 删除子树

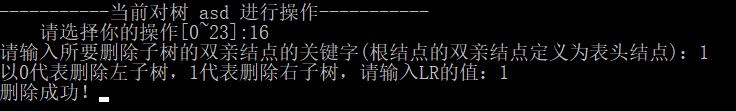


图3-61 删除子树

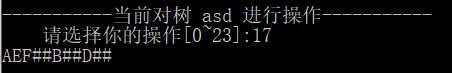


图3-62 前序遍历二叉树

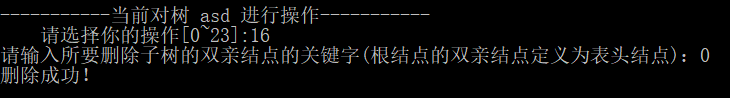


图3-63 删除子树

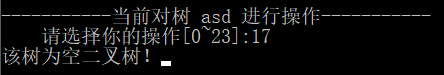


图3-64 前序遍历二叉树

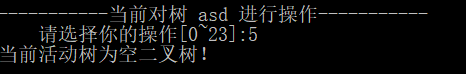


图3-65 判空

（18）调用生成二叉树的功能函数，依次输入子树的先序序列“\n(换行符)”、“A#”、“A##”、。输入换行符时因未输入任何字符所以构建二叉树失败；输入A#时因A#时不完整的先序遍历序列所以提示构建二叉树失败；输入A##时，正确生成一个仅有一个根结点且char型数据为A关键字为1的二叉树。

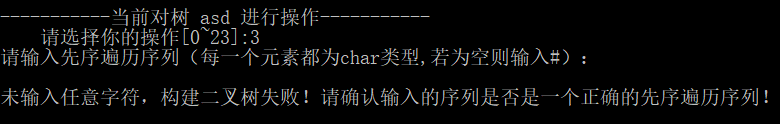


图3-66 生成二叉树

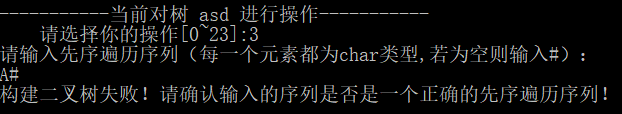


图3-67 生成二叉树

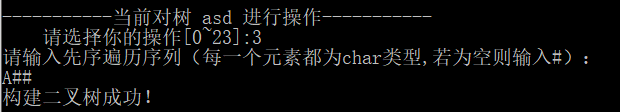


图3-68 生成二叉树

测试用例四：

多二叉树操作，再初始化一个名为asd的二叉树，进行文件读取和切换表的操作，对应测试用例59-67。

（1）调用初始化二叉树的功能函数，依次输入asd、zxc。输入asd时，因名为asd的二叉树已经存在，所以输出该名称已存在，创建失败；输入zxc时，正确初始化一个名为zxc的空二叉树。

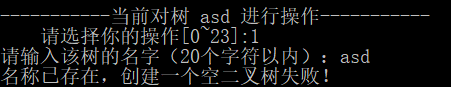


图3-69 初始化二叉树

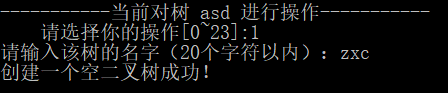


图3-70 初始化二叉树

（2）调用生成二叉树的功能函数，输入序列E##正确生成一个二叉树。

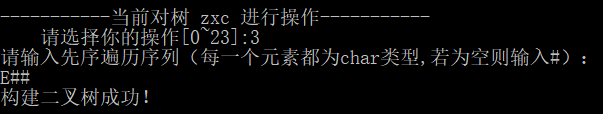


图3-71 生成二叉树

（3）调用从文件中加载数据的功能函数，输入错误的文件名字da.txt时显示错误信息，打开文件失败和录入数据失败；输入正确的data.txt后提示录入完毕且录入成功。通过调用先序遍历的功能函数确认原二叉树已被覆盖，目前的二叉树的先序序列即为文件中的序列。

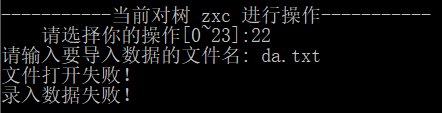


图3-72 加载二叉树

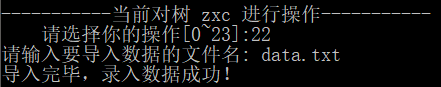


图3-73 加载二叉树

（4）调用切换表的功能函数，输入错误的名字z，则提示切换失败，可通过调用先序遍历的功能函数确认活动表未发生变化；输入正确的asd，则正确切换表，并可通过调用先序遍历的功能函数确认活动表已切换至表asd。

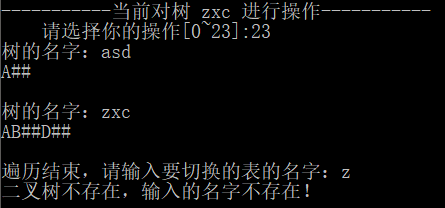


图3-74 切换二叉树

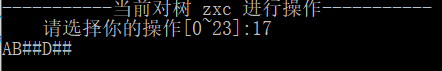


图3-75 前序二叉树

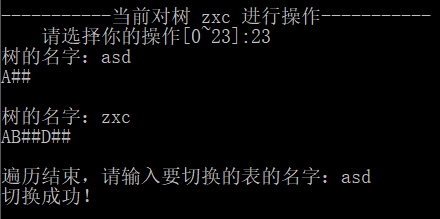


图3-76 切换二叉树

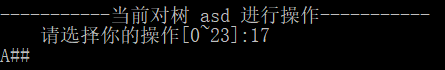


图3-77 前序遍历二叉树

测试用例五：

最后输入0，退出系统，对应测试用例68。

测试结果：

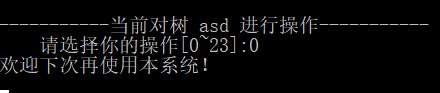


图3-78 退出系统

## 3.4 实验小结

这次实验主要是在生成二叉树、插入子树和层序遍历二叉树的功能函数中遇到问题。

起初在生成二叉树的递归中函数参数为int，虽未报错但生成的二叉树通过遍历看出时错误的，最后在经过单步调试后发现因参数类型为int，导致因决定元素位置而应递增的i值，出现了回溯的现象，之后传入i值地址作为参数才解决了问题。

插入子树的主要问题在于怎样将子树转变为一个右子树为空的二叉树，最初思维局限于自上而下，但明显自上而下的操作为避免重复处理则需要加很多的判断条件，在经过很多尝试后，最后采用自下而上的处理方式，以防止重复处理的现象。

在实现层序遍历时，因个人认为在书写二叉树的遍历序列时层序遍历时最简单最不易出错的，所以最初想用循环的方式实现，思考了一段时间觉得单循环是不能解决问题的，二叉树是一种比较特殊的结构，其层序遍历可借助队列来实现。因此也通过这次实验对队列中初始化队列、出队、入队、输出队列首元素、判断队列是否为空等基本操作进行了练习和使用。

# 4 基于邻接表的图实现

## 4.1问题描述

采用邻接表作为图的物理结构，并以函数形式定义实现图的创建、销毁、查找顶点、获得顶点值、顶点赋值、获得第一邻接点、获得下一邻接点、插入顶点、删除顶点、插入弧、删除弧、深度优先搜索遍历、广度优先搜索遍历13种基本算法，以及文件读写和切换图3种拓展算法，且根据要求构造一个具有菜单的功能演示系统，在主程序中完成函数调用所需实参值的准备和函数执行结果。

4.1.1 图抽象数据类型

依据最小完备性和常用性相结合的原则，设计了图的数据对象和数据关系，并定义了图的创建、销毁、查找顶点、获得顶点值、顶点赋值、获得第一邻接点、获得下一邻接点、插入顶点、删除顶点、插入弧、删除弧、深度优先搜索遍历、广度优先搜索遍历13种基本算法，具体数据和运算功能定义如下。

ADT Graph{

**数据对象V**：V是具有相同特性的数据元素的集合，称为顶点集。

**数据关系R**：R={VR}

VR={<v,w>|v,w∈V且P(v,w),<v,w>表示从v到w的弧，谓词P(v,w)定义了弧<v,w>的意义或信息}

**基本操作P**：

CreateCraph(&G,V,VR)：

初始条件：V是图的顶点集，VR是图中弧的集合。

操作结果：按V和VR的定义构造图。

DestroyGraph(G)：

初始条件：图G存在。

操作结果：销毁图G。

LocateVex(G,u)：

初始条件：图G存在，u是G中顶点有相同特征。

操作结果：若G种存在顶点u，则返回该顶点在图中的位置，

否则返回其他信息。

GetVex (G,v)：

初始条件：图G存在，v是G中某个顶点。

操作结果：返回v的值。

PutVex (G,v,u)：

初始条件：图G存在，v是G中某个顶点。

操作结果：对v赋值u。

FirstAdjVex(&G, v)：

初始条件：图G存在，v是G中某个顶点。

操作条件：返回v的第一个邻接顶点。若顶点在G中没有邻接

顶点，就返回“空”。

NextAdjVex(&G, v, w)：

初始条件：图G存在，v是G中某个顶点，w是v的邻接顶点。

操作结果：返回v的（相对于w的）下一个邻接顶点。若w是

v的最后一个邻接点，则返回“空”。

InsertVex(&G,v)：

初始条件：图G存在，v和图中顶点有相同特征。

操作结果：在图G中增添新顶点v。

DeleteVex(&G,v)：

初始条件：图G存在，v是G中某个顶点。

操作结果：删除G中顶点v及其相关的弧。

InsertArc(&G,v,w)：

初始条件：图G存在，v和w是G中两个顶点。

操作结果：在G中增添弧<v,w>。

DeleteArc(&G,v,w)：

初始条件：图G存在，v和w是G中两个顶点。

操作结果：在G中删除弧<v,w>。

DFSTraverse(G)：

初始条件：图G存在。

操作结果：对图进行深度优先遍历。

BFSTraverse(G)：

初始条件：图G存在。

操作结果：对图进行广度优先遍历。

}ADT Graph

4.1.2 多图抽象数据类型

基于之前实现的图数据类型，设计了多图的数据对象、数据关系，并以定义了多线性表的文件读写和切换图3种基本运算，具体数据和运算功能定义如下。

ADT ALGraph{

**数据对象V**：V是具有相同特性的数据元素的集合，称为顶点集。

**数据关系R**：R={VR}

VR={<v,w>|v,w∈V且P(v,w),<v,w>表示从v到w的弧，谓词P(v,w)定义了弧<v,w>的意义或信息}

**基本操作P**：

SaveData(fp,G,filename):

初始条件:图存在。

操作结果:将图中的数据按深度优先遍历顺序存入文件。

LoadData(fp,G,filename):

初始条件:图存在且为空。

操作结果:读入文件中数据，并依此构建图。

ExGraph(Head,name):

初始条件:无。

操作结果:将当前活动图切换为树名为name的图，若无则不

进行改变。

}ADT ALGraph

4.1.3 演示系统与数据文件

构造一个具有菜单的功能演示系统。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

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

演示系统基于链表实现的多个图管理。采用链表的方式管理多个二叉树，链表中的每个结点为一个图的基本属性，包含有图的名称、顶点集、顶点数、弧的数目以及指向下一个图的指针。

## 4.2 系统设计

4.2.1 数据物理结构

在本程序中，数据关键字类型VertexType被定义为char字符型。

1.每个图结点的存储结构。

typedef struct ALGraph {

VNode vertices[MAX\_VERTICE\_NUM];

int vexnum, arcnum; //图的当前顶点数和弧数

char name[20]; //图的名字

ALGraph \*next; //指向下一个图的指针

}ALGraph;

2.每个顶点结点的存储结构。

typedef struct VNode{

VertexType data; //顶点信息

ArcNode \*firstarc; //指向第一条依附该顶点的弧的指针

}VNode;

3.每一条弧的存储结构。

typedef struct ArcNode {

int adjvex; //该弧所指向的顶点的位置

struct ArcNode \*nextarc; //指向下一条弧的指针

}ArcNode;

整体存储结构如图4-1所示：

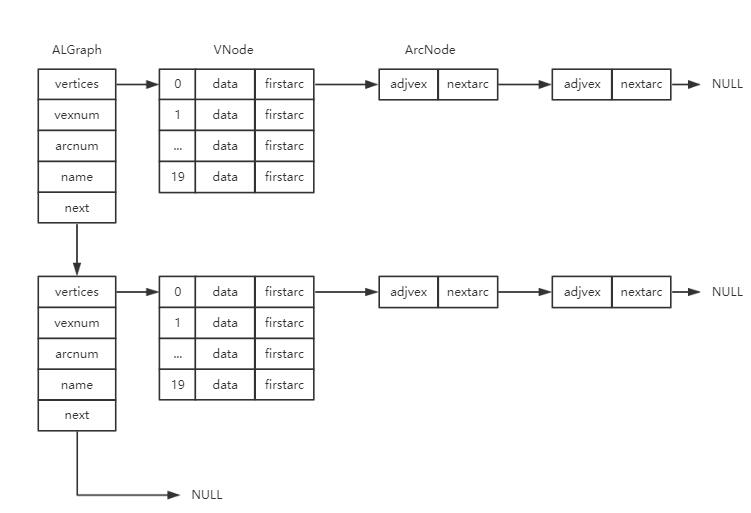


图4-1 多图的存储结构

4.2.2 演示系统

演示系统由用户操作界面与功能调用部分组成，如图4-2所示。

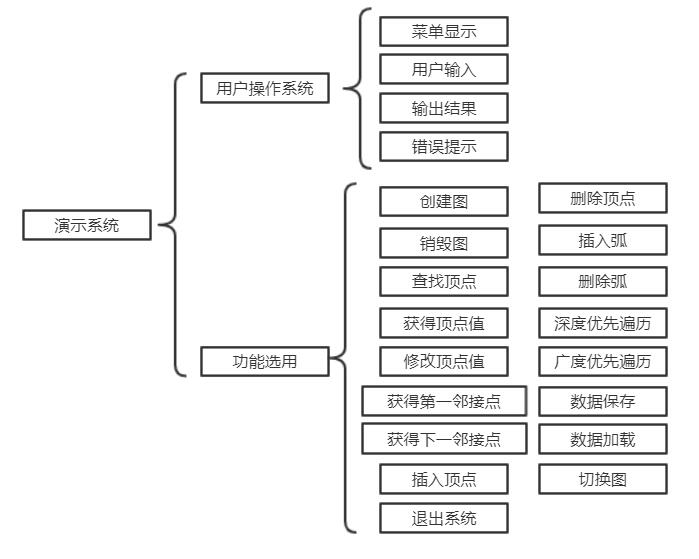


图4-2 演示系统模块结构图

4.2.3 数据文件存储格式

考虑到方便查看,以txt格式直接存放图的数据。

4.2.4 图运算实现算法

1.CreateCraph(&G,V,VR)

**算法思想：** 先将顶点集V中数据依次赋值给G中顶点数组中的数据域，并将其指针域的指针置空。循环对每一个顶点进行操作，若对应的VR[i][j]不为终止符‘\0’则查找VR[i][j]中的数据对应的顶点位置，若不存在返回NULL，并提示构建失败，若存在则新声明一个ArcNode的指针，用其数据域存储该顶点位置，其指针域指向空，并用先进先出的方法将该结点接入VR[i]的邻接链表中，最后若成功则将G返回给主函数中的G1，并将G1介入到多图链表的最后一个非空结点的后面。并令G=G1。

**时间复杂度分析：** 假设图G有v个顶点和e条弧，则算法的时间复杂度为O(v+e)。

**空间复杂度分析：** 假设图G有v个顶点和e条弧，因插入每一条弧是都需要进行空间的分配，所以算法的空间复杂度为O(e)。

2.DestroyGraph(G)

**算法思想：** 若G为空则返回ERROR,不能进行销毁操作。若G不为空则利用循环对G中每个结点进行操作，将结点中fistarc后连接的结点的存储全部释放并将fistarc指针指向空。最后令G的vexnum,arcnum,name,vertices全部初始化为0，返回OK后，在主函数中利用循环找到G1前面的结点G，并使G1->next=G1->next->next，G1=NULL。

**时间复杂度分析：** 假设图G有v个顶点和e条弧，则算法的时间复杂度为O(v+e)。

**空间复杂度分析：** 假设图G有v个顶点和e条弧，因插入每一条弧是都需要进行空间的分配，所以算法的空间复杂度为O(e)。

3.LocateVex(G,e)

**算法思想：** 算法流程图如图4-3所示：

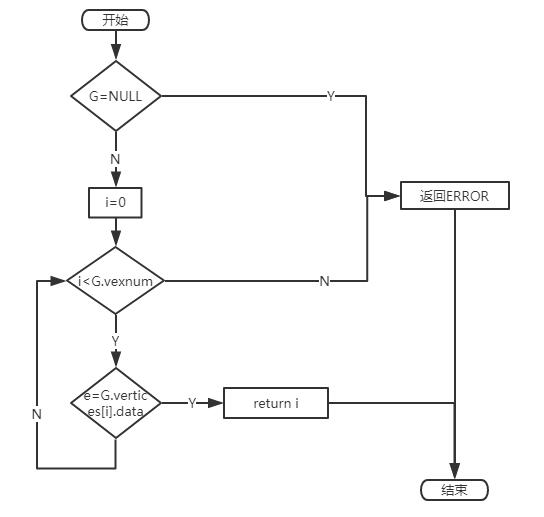


图4-3 LocateVex的算法流程图

**时间复杂度分析：** 假设图G有v个顶点，则算法的时间复杂度为O(v)。

**空间复杂度分析：** 算法的空间复杂度为0(1)。

4.GetVex (G,i)

**算法思想：** 算法流程图如图4-4所示：



图4-4 GetVex的算法流程图

**时间复杂度分析：** 算法的时间复杂度为O(1)。

**空间复杂度分析：** 算法的空间复杂度为0(1)。

5.PutVex (G,i,e)

**算法思想：** 调用GetVex函数查找位置为i的顶点的数据，若函数返回值为ERROR,则返回ERROR,否则将G->vertices[i].data赋值为e。

**时间复杂度分析：** 算法的时间复杂度为O(1)。

**空间复杂度分析：** 算法的空间复杂度为0(1)。

6.FirstAdjVex(&G, i)

**算法思想：** 调用GetVex函数查找位置为v的顶点的数据，若函数返回值为ERROR,则返回ERROR,否则判断G.vertices[v].fistarc是否为空，若是空返回-1，否则返回G.vertices[v].fistarc.adjvex。

**时间复杂度分析：** 算法的时间复杂度为O(1)。

**空间复杂度分析：** 算法的空间复杂度为0(1)。

7.NextAdjVex(&G, i, j)

**算法思想：** 算法流程图如图4-5所示：

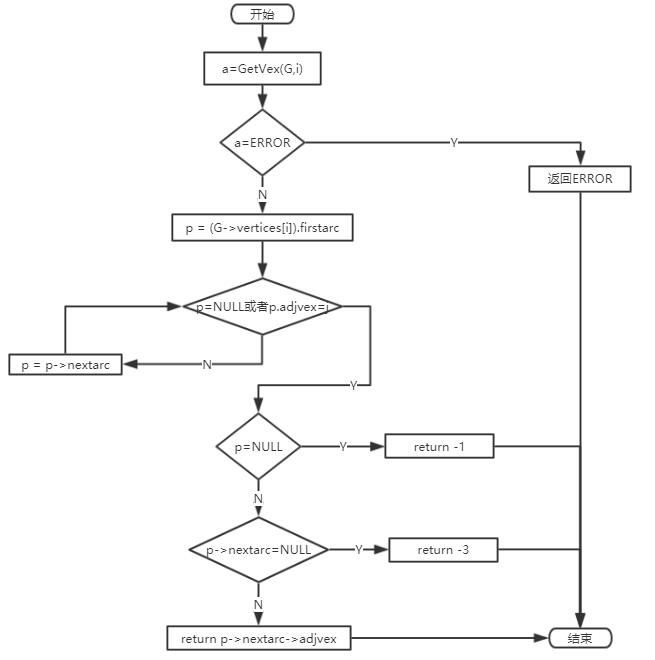


图4-5 NextAdjVex的算法流程图

**时间复杂度分析：** 假设图G有n个顶点且每个顶点指向ni个顶点，且查找每个顶点的每个邻接点可能性相同，则算法的时间复杂度为0((∑(ni+1)/2)/n)。

**空间复杂度分析：** 声明一个临时结构指针变量p，算法的空间复杂度为0(1)。

8.InsertVex(&G,e)

**算法思想：** 若G为空或G.vexnum= MAX\_VERTICE\_NUM则返回ERROR,否则令j=G.vexnum,将G.vertices[j].data=e,G.vertices[j].firstarc=NULL,最后将G.vexnum自增1并返回OK。

**时间复杂度分析：** 算法的时间复杂度为O(1)。

**空间复杂度分析：** 算法的空间复杂度为0(1)。

9.DeleteVex(&G,i)

**算法思想：** 调用GetVex函数查找位置为i的顶点，并用a接收返回值，若a=ERROR,则返回ERROR。利用循环将该顶点的邻接顶点结构指针逐个释放空间并置空，同时每次删除一条边都令G.arcnum--,令G.vexnum--。用循环遍历图中其他顶点的邻接顶点中是否有该顶点，若有则删除该邻接顶点结点。最后返回OK。

**时间复杂度分析：** 假设图G有n个顶点且每个顶点有ni个邻接点，且删除每个顶点的可能性相同，则算法的时间复杂度为0((∑(ni+1)/2)/n)。

**空间复杂度分析：** 假设图G有n个顶点且每个顶点有ni个邻接点，则算法的空间复杂度为0((∑ni)/n)。

10.InsertArc(&G,i,j)

**算法思想：** 首先判断图G中是否存在位置为i和j的两个顶点，若不是则返回ERROR，若是则声明结构指针p和q，用p存处j并令p的下一弧指针指向空，再令q=G.vertices[i].firstarc，若q=NULL,则G.vertices[i].firstarc=p,否则遍历至最后一个非空结点此时p->next=NULL,所以令p->next=q，最后令G.arcnum++,并返回OK。

**时间复杂度分析：** 假设图G有n个顶点且每个顶点指向ni个顶点，且删除每个顶点的可能性相同，则算法的时间复杂度为0((∑(ni+1)/2)/n)。

**空间复杂度分析：** 声明的临时结构指针变量p和q，算法的空间复杂度为0(1)。

11.DeleteArc(&G,i,j)

**算法思想：** 所要删除弧的弧头顶点位置为i，弧尾定点位置为j。首先判断位置为i的顶点是否存在，若存在则遍历其邻接表中的弧结构，找到或遍历结束就退出循环，此时p指向退出循环是最后的弧结构，若p=NULL,即没有找到则返回ERROR，若存在则删除该弧结点。

**时间复杂度分析：** 假设图G有n个顶点且每个顶点指向ni个顶点，且删除每条弧的可能性相同，则算法的时间复杂度为0((∑(ni+1)/2)/n)。

**空间复杂度分析：** 算法的空间复杂度为0(1)。

12.DFSTraverse(G)

**算法思想：** 若G为NULL则返回ERROR，否则将visited数组元素全部初始化为0，并利用循环对每个顶点进行判断，若未被访问就调用DFS函数。

**时间复杂度分析：** 调用DFS函数，假设图G有v个顶点和e条弧,算法的时间复杂度为0(v+e)。

**空间复杂度分析：**算法的孔明好考吗复杂度为0(1)。   
13.DFS(G,v)

**算法思想：** 输出位置为v的顶点的数据，并将其visited[v]=1,标志为已经遍历。令p指向v的第一个邻接点，若p不为空则判断若p的数据域存储的位置的顶点未遍历则调用DFS(G,p->adjvex)，否则令p=p->nextarc。

**时间复杂度分析：** 采用递归的算法思想，假设图G有v个顶点和e条弧,算法的时间复杂度为0(v+e)。

**空间复杂度分析：** 算法的空间复杂度为0(1)。

14.BFSTraverse(G)

**算法思想：** 借助队列实现图的广度优先搜索，先将将visited数组元素全部初始化为0，将位置为0的顶点出发，将顶点入队列，然后访问了队列头元素的邻接顶点，并将该顶点的邻接点入队列，最后将该顶点出队列，以队列为空作为条件循环，直到所有顶点都被访问过。

**时间复杂度分析：** 假设图G有v个顶点和e条弧,算法的时间复杂度为0(v+e)。

**空间复杂度分析：** 算法的空间复杂度为0(1)。

4.2.5 多图运算实现算法

1.SaveData(fp,G,filename)

**算法思想：**输入文件名filename，用fp存储fopen函数打开名为filename的文件的返回值，若为NULL则输出提示信息并返回ERROR，否则将G的信息写入文件，之后关闭文件指针fp并返回OK。

**时间复杂度分析：**假设图G有v个顶点和e条弧,算法的时间复杂度为0(v+e)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

2.LoadData(fp,G,filename)

**算法思想：** 输入文件名filename，用fp存储fopen函数打开名为filename的文件的返回值，若为NULL则输出提示信息并返回ERROR，否则将文件中的信息读入数组V和VR调用CreatGraph函数生成图，之后关闭文件指针fp并返回OK。

**时间复杂度分析：** 调用CreatGraph函数，假设图G有v个顶点和e条弧，则算法的时间复杂度为O(v+e)。

**空间复杂度分析：** 假设图G有v个顶点和e条弧，因插入每一条弧是都需要进行空间的分配，所以算法的空间复杂度为O(e)。

3.ExGraph(Head,name)

**算法思想：**首先遍历链表查找名字为name的元素位置，若没有查到则返回NULL，若查到则返回当前结构指针。

**时间复杂度分析：**假设多图中有n个图，且所要查找的图为每个图的概率均相同，算法的时间复杂度为线性阶O(n)。

**空间复杂度分析：**算法的空间复杂度为常量阶O(1)。

## 4.3 系统实现

4.3.1 实验环境

实验代码使用dev c++编写与编译.

4.3.2 演示系统操作

演示系统采用键盘操作，控制台输出操作提示与结果，第一行显示当前操作表（默认为无），中间显示可用的操作项，输入对应的操作指令即可执行，输入非0~16的操作号会提示错误。

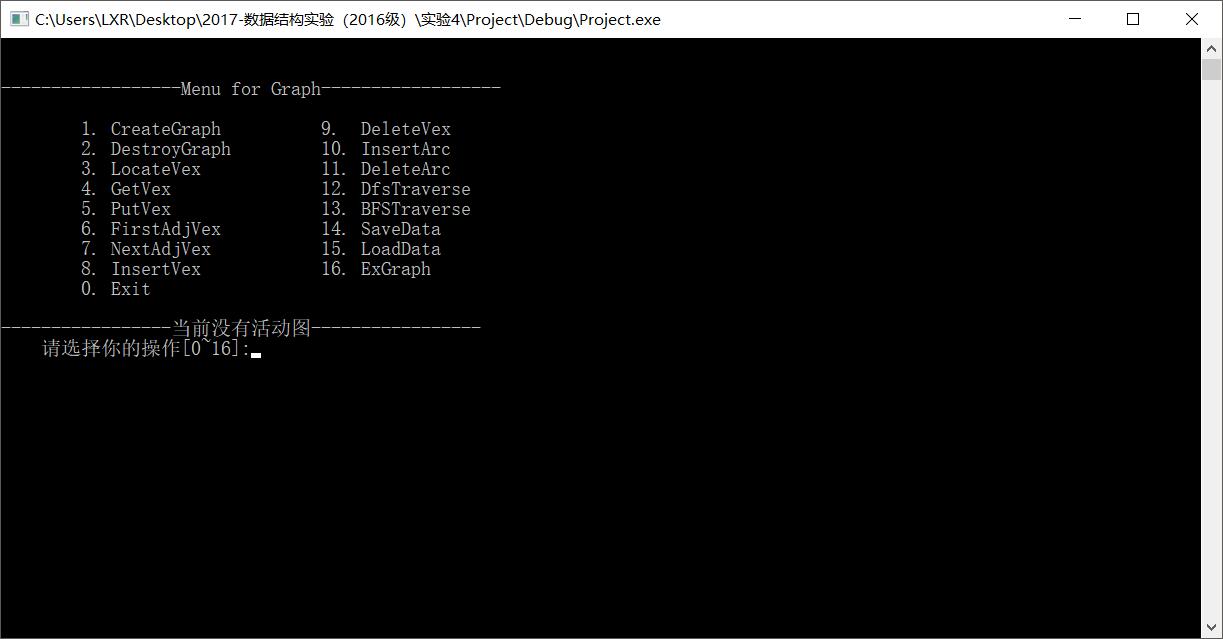


图4-6 启动程序后的操作菜单

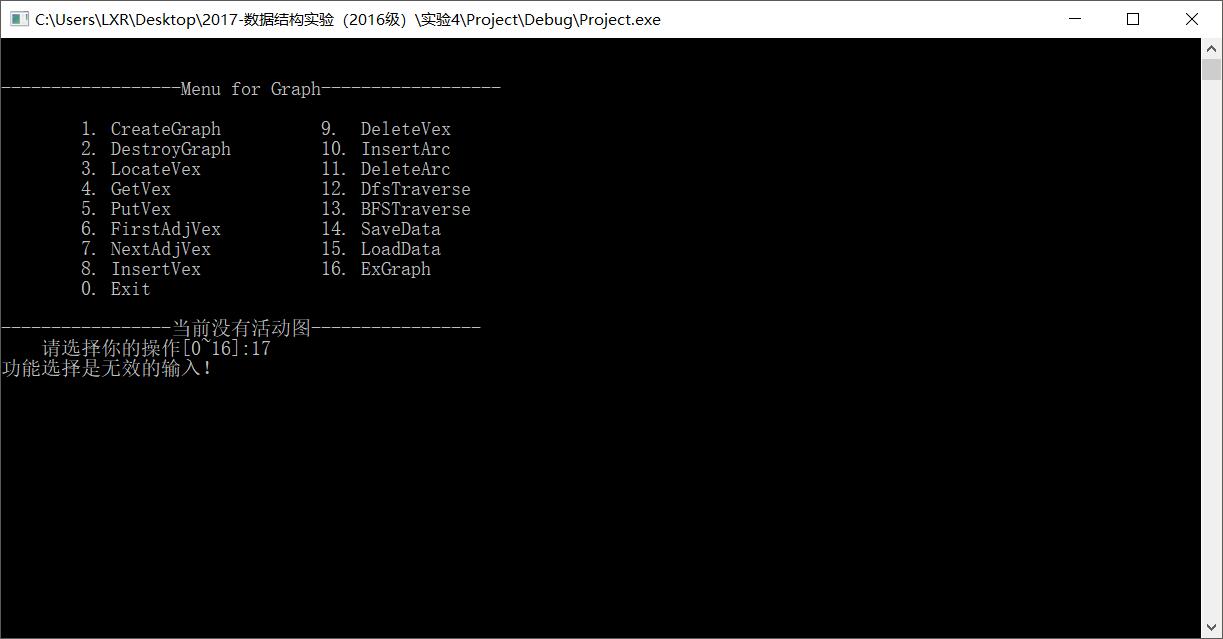


图4-7 输入错误的功能选择数字，显示提示信息

输入操作指令0，程序退出。

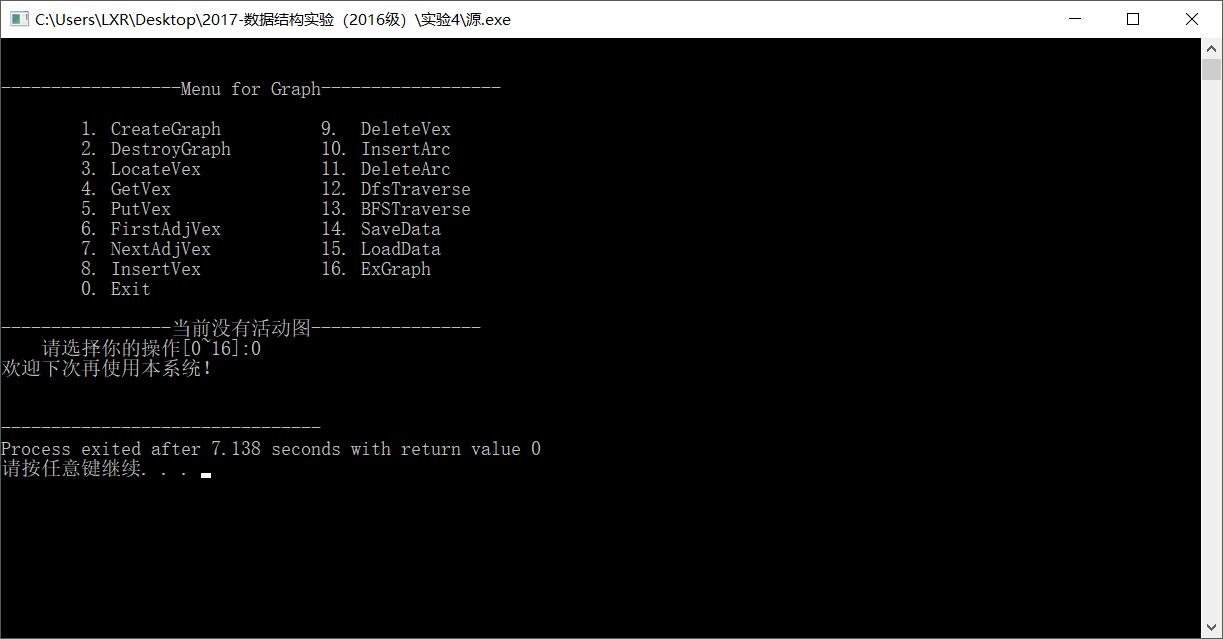


图4-8 输入操作0程序退出

4.3.3 测试计划

1.检测常规状态下系统的工作情况；

2.重点检查非法边界操作时的工作情况。

表3-1 演示系统测试计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试功能 | 测试顺序 | 测试输入 | 预计输出 | 图的状态 |
| CreateGraph | 4  5  6  7  11  42  43 | 4：asd 0 1  5:asd  4 4  abcd  b c a 0  6：asd 21 4  7：asd 0 0  11: zxc  4 3  abcd  b c a 0  42：zxc  43：asd  5 3  qwert  w e r 0 0 | 7/11：构建成功  4/5/6：构建失败 | 4/5/6：同1  7：图名为asd，顶点数和弧数都是0  11:图名为zxc,顶点集为abcd，弧数为3<a,b>、<b,c>、<c,a>  42:同38  43:图名为asd,顶点集为qwert，弧数为3<q,w><w,e><e,r> |
| DestroyGraph | 1  10 |  | 10：销毁成功 | 1/10：无活动图 |
| LocateVex | 2  12 13 14  28  31 | 2：0  12：0  13：a  14:d  28:f  31：b | 2/12/31：查找失败  13：0  14：3  28:4 | 2：同1  12/13/14：同11  28:同27  31:同30 |
| GetVex | 1. 8   15 16 17  21 | 3/8：0  15：0  16:4  17:-1  21:2 | 3/8/16/17：查找失败  15：a  21:e | 3：同1  8：同4  15/16/17：同11  21:同20 |
| PutVex | 18  19 20 | 18:-1 e  19:4 e  20:2 e | 18/19：修改失败  20：修改成功 | 20：图名为zxc,顶点集为abed，弧数为3<a,b>、<b,e>、<e,a> |
| FirstAdjVex | 22  23  24  32  35 | 22:1  23:2  24:3  32/35:0 | 22:2 e  23:0 a  24:无  32:无  35:3 f | 22/23/24：同20  32：同30  35:同34 |
| NextAdjVex | 25 26 | 25：0 1  26：0 2 | 25:无  26：位置为0的不存在位置为2的邻接点 | 25/26:同20 |
| InsertVex | 27 | 27：f | 27:插入成功 | 27：图名为zxc,顶点集为abedf，弧数为3<a,b>、<b,e>、<e,a> |
| DeleteVex | 29  30 | 29：5  30：1 | 29：删除失败  30:删除成功 | 29：同27  30：图名为zxc,顶点集为aedf，弧数为1<e,a> |
| InsertArc | 33  34 | 33:0 5  34:0 3 | 33:插入失败  34:插入成功 | 33：同30  34：图名为zxc,顶点集为aedf，弧数为2<e,a><a,f> |
| DeleteArc | 36  37  38 | 36:5 0  37:0 5  38:1 0 | 36/37:删除失败  38:删除成功 | 36/37：同34  38：图名为zxc,顶点集为aedf，弧数为1<a,f> |
| DfsTraverse | 9  39  44  47  49  51 |  | 9空白+遍历成功  39/47/51：afed  44/49:qwert | 9同4  39/47/49：同38  44:同43  51：同50 |
| BFSTraverse | 40 |  | 40：afed | 40:同38 |
| SaveData | 41 | data.txt | 41:写入成功 | 41：同38 |
| LoadData | 50 | data.txt | 48:录入成功 | 50:除名字为asd外与38相同 |
| ExGraph | 45  46  48 | 45:cv  46：zxc  48:asd | 45:切换失败  46/48：切换成功 | 45：同43  46:同38  48:同43 |
| Exit | 52 |  |  |  |

4.3.4 测试结果

测试用例一：

当前图未被构建时，操作对应测试计划1-3。

测试结果：（在进行其他操作时同样提示当前没有活动表）

（1）在无活动图的情况下销毁图，销毁失败。

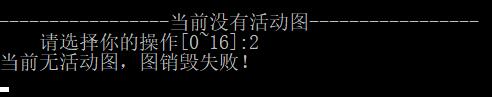


图4-9 销毁图

（2）在无活动图的情况下查找顶点，输入0，查找失败。

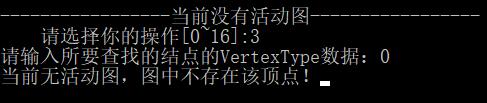


图4-10 获得顶点位置

（3）在无活动图的情况下获得顶点值，输入4，查找失败。

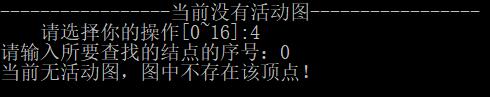


图4-11 获得顶点值

测试用例二：

活动图为一个顶点和弧数都是0的图，操作对应测试计划4-10。

测试结果：

（1）创建一个图，当输入的顶点数为0，而边数不为0时创建失败。

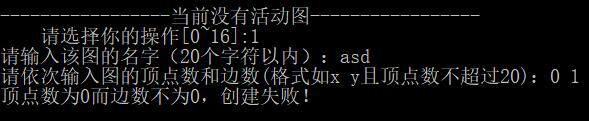


图4-11 创建图

（2）创建一个图，当输入的边数，和实际输入的边数不同时，创建失败。

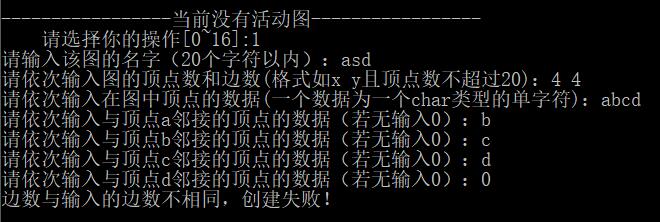


图4-12 创建图

（3）创建一个图，当输入的顶点数大于最大限制20时，创建失败。

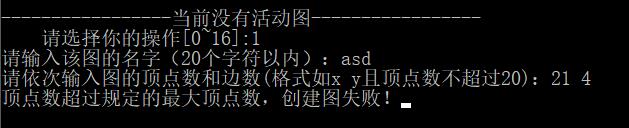


图4-13 创建图

（4）创建一个名为asd且顶点数和弧数都为0的图，创建成功。

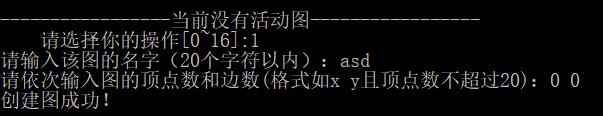


图4-14 创建图

（5）找到图中序号为0的顶点的数据，由于图asd的顶点集为空，所以查找失败。

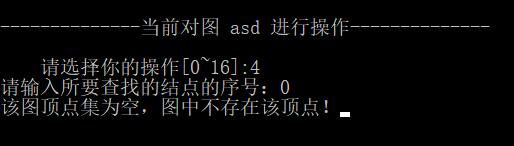


图4-15 查找顶点

（6）遍历图，因为asd中顶点集为空，所以遍历成功但输出的为空白。

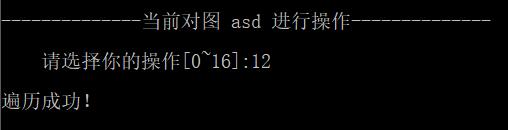


图4-16 遍历图

（7）销毁图，因为当前有活动图，所以销毁成功。

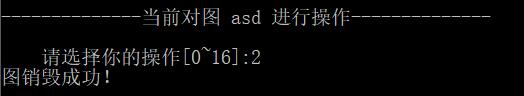


图4-17 销毁图

测试用例三：

活动图为一个有四个顶点（a、b、c、d）和弧数为3（<a,b>、<b,c>、<c,a>）的图，操作对应测试计划11-41。

测试结果：

（1）创建一个 名为zxc,有四个顶点三条弧的图，其中弧为<a,b>、<b,c>、<c,a>，创建成功。

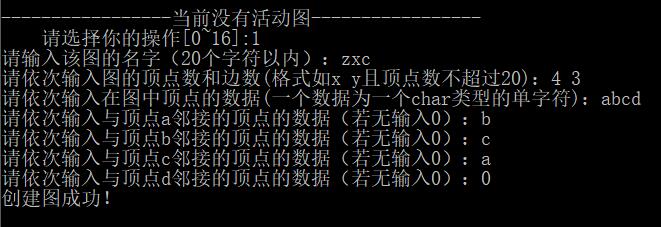


图4-17 创建图

（2）查找图zxc中数据为0的顶点，由于图中没有这个顶点所以查找失败。

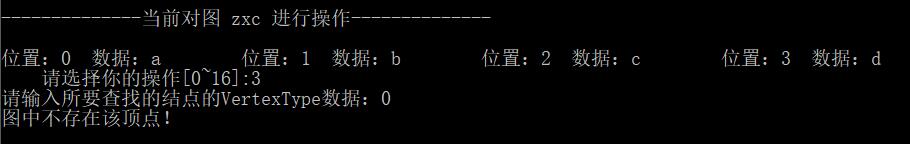


图4-18 查找顶点

（3）查找图zxc中数据为a的顶点，由创建图的过程可知，其序号为0。

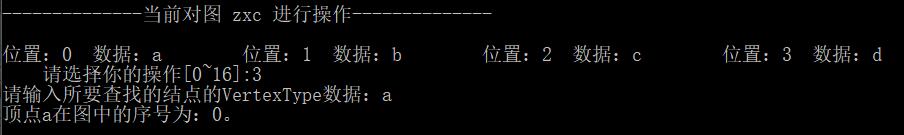


图4-19 查找顶点

（4）查找图zxc中数据为d的顶点，由创建图的过程可知，其序号为3。

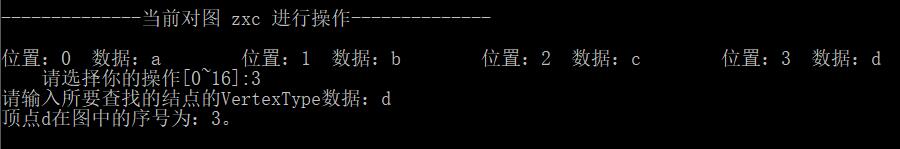


图4-20 查找顶点

（5）查找图zxc中位置为0的顶点的值，由创建图的过程可知，其值为a。

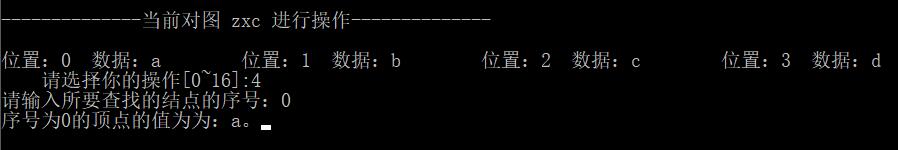


图4-21 获得顶点值

（6）查找图zxc中位置为4的顶点的值，由创建图的过程可知，图中不存在该顶点，所以查找失败。

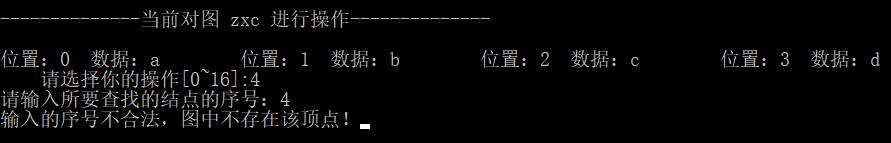


图4-22 获得顶点值

（7）查找图zxc中位置为-1的顶点的值，由创建图的过程可知，图中不存在该顶点，所以查找失败。

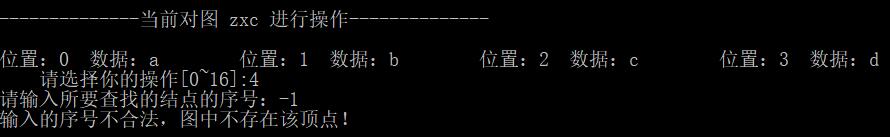


图4-23 获得顶点值

（8）修改图zxc中位置为-1的顶点的数据为e，由于图中没有位置为-1的顶点，修改失败。

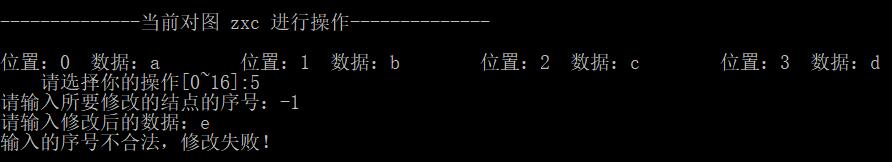


图4-24 修改顶点值

（9）修改图zxc中位置为4的顶点的数据为e，由于图中没有位置为4的顶点，修改失败。

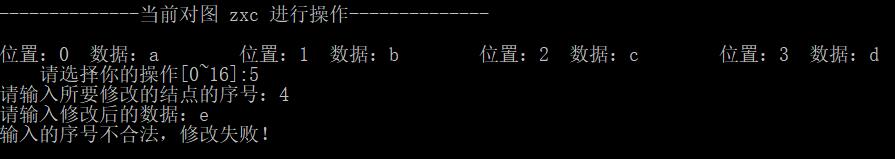


图4-25 修改顶点值

（10）修改图zxc中位置为2的顶点的数据为e，由于图中存在位置为2的顶点，修改成功。

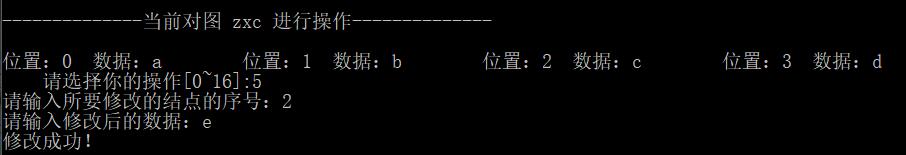


图4-26 修改顶点值

（11）查找图zxc中位置为2的顶点的值，经修改后，查找后的结果为e。

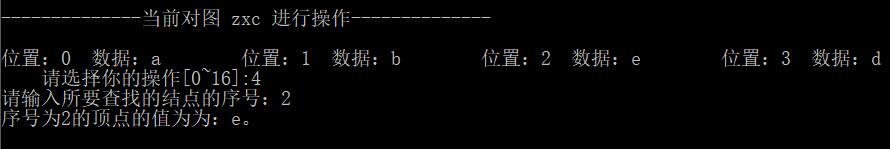


图4-27 获得顶点值

（12）查找图zxc中位置为1的顶点的第一邻接点，查找结果为序号为2，数据为e的顶点。

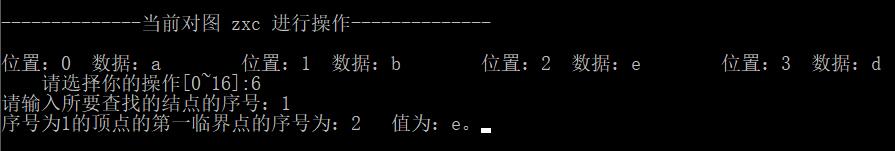


图4-28 获得第一邻接点

（13）查找图zxc中位置为2的顶点的第一邻接点，查找结果为序号为0，数据为a的顶点。

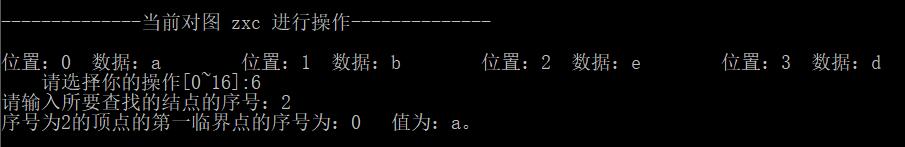


图4-29 获得第一邻接点

（14）查找图zxc中位置为3的顶点的第一邻接点，由于该顶点无邻接点，所以查找失败。

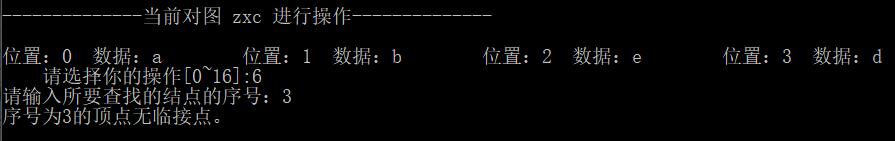


图4-30 获得第一邻接点

（15）查找图zxc中序号为0的顶点的序号为1的邻接点的下一邻接点，由于序号为0的顶点仅有序号为1的顶点这一个邻接点，所以无下一邻接点。

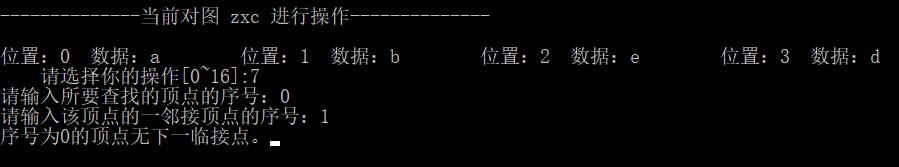


图4-31 获得下一邻接点

（16）查找图zxc中序号为0的顶点的序号为2的邻接点的下一邻接点，由于序号为0的顶点没有序号为2的顶点这一个邻接点，查找失败。

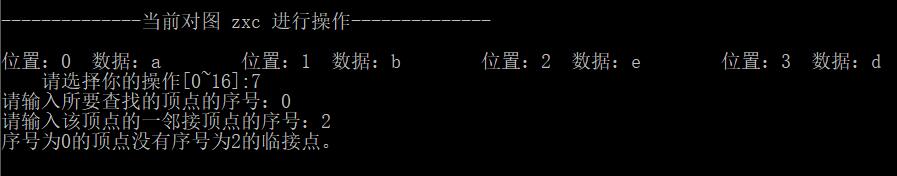


图4-32 获得下一邻接点

（17）为图zxc插入一个顶点值为f的顶点，插入成功。

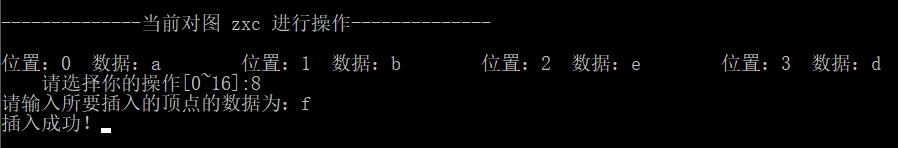


图4-33 插入顶点

（18）查找图zxc中顶点值为f的顶点的位置，经插入顶点后，查找结果为4。

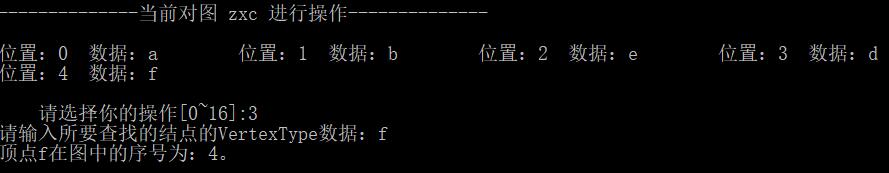


图4-34 查找顶点

（19）删除图zxc中序号为5的顶点，由于图zxc中不存在符合该条件的顶点，所以删除失败。

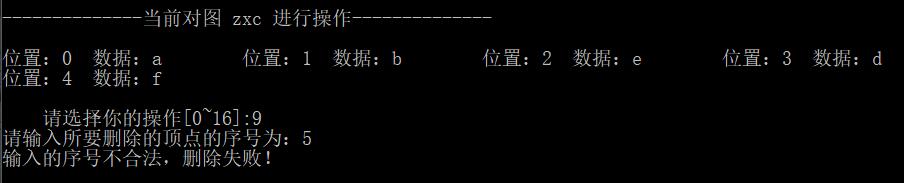


图4-35 删除顶点

（20）删除图zxc中序号为1的顶点，由于图zxc中存在符合该条件的顶点，所以删除成功。

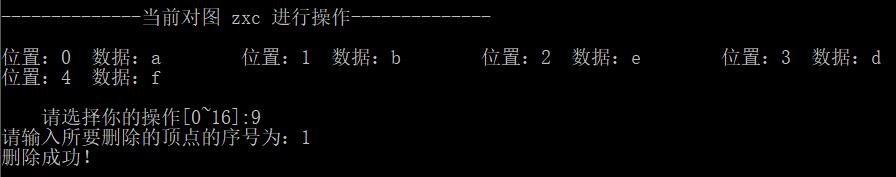


图4-36 删除顶点

（21）查找图zxc中顶点值为b的顶点，经删除后，图中不存在该顶点，查找失败。

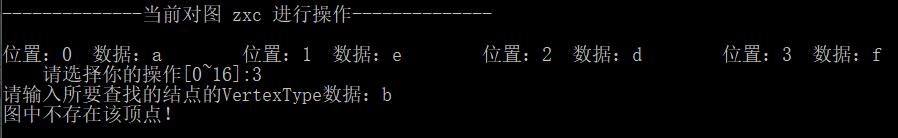


图4-37 查找顶点

（22）查找图zxc中位置为0的顶点的第一邻接点，由于原本的序号为1的顶点已被删除，以该顶点为弧头弧尾的边均已删除，所以该顶点无邻接点，查找失败。

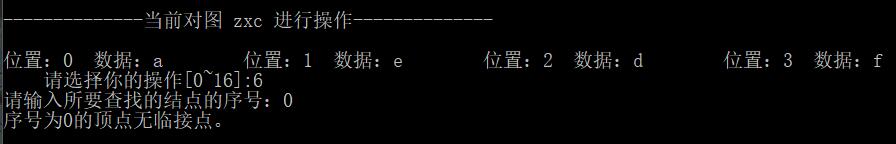


图4-38 获得第一邻接点

（23）为图zxc插入一个弧尾为序号为0的顶点，弧头为序号为5的顶点的弧，由于图中不存在序号为5的顶点，所以插入弧失败。

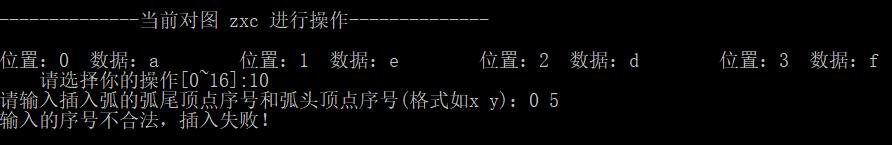


图4-39 插入弧

（24）为图zxc插入一个弧尾为序号为0的顶点，弧头为序号为3的顶点的弧，由于图中存在序号为0的顶点和序号为3的顶点，所以插入弧成功。

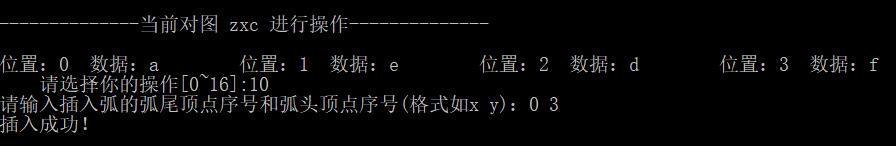


图4-40 插入弧

（25）查找图zxc中序号为0的顶点的第一邻接点，经插入弧<a,f>成功后，查找结果为f。

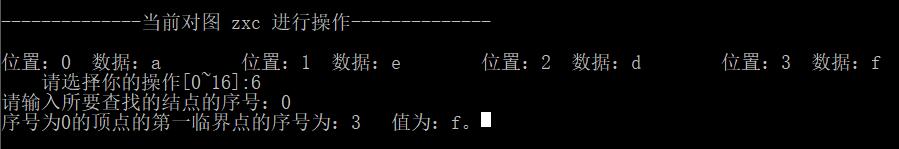


图4-41 获得第一邻接点

（26）删除图zxc中以为序号为5的顶点为弧尾，序号为0的顶点为弧头的弧，由于图中不存在序号为5的顶点，所以删除失败。

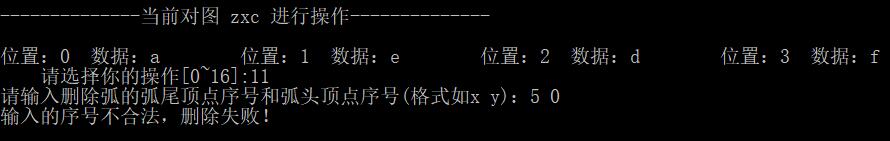


图4-42 删除弧

（27）删除图zxc中以为序号为0的顶点为弧尾，序号为5的顶点为弧头的弧，由于图中不存在序号为5的顶点，所以删除失败。

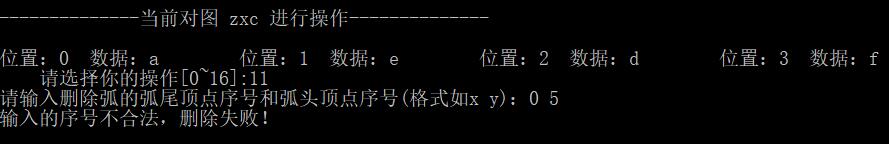


图4-43 删除弧

（28）删除图zxc中以为序号为1的顶点为弧尾，序号为0的顶点为弧头的弧，由于图中存在序号为1的顶点和序号为0的顶点，所以删除成功。

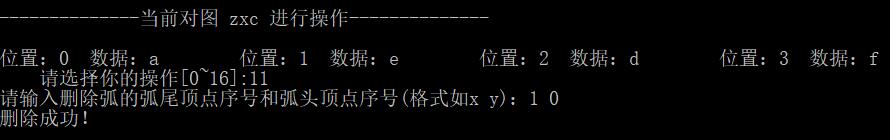


图4-44 删除弧

（29）深度优先遍历图zxc，为afed。

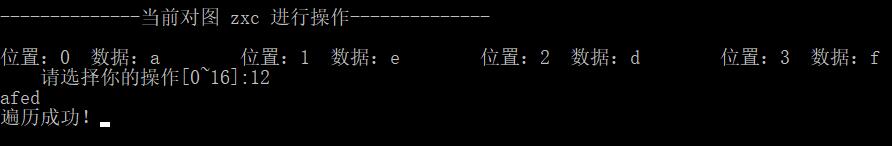


图4-45 深度优先遍历

（30）广度优先遍历图zxc，为aedf。

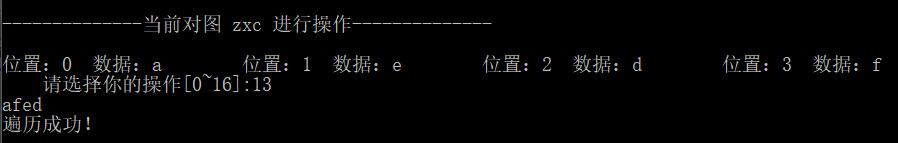


图4-46 广度优先遍历

（31）将图zxc的数据存入名为data.txt的文件中，第一行存顶点数和弧数，第二行存顶点集，之后依次存入与顶点邻接的数据构成的字符串，无邻接点存入一个字符0。

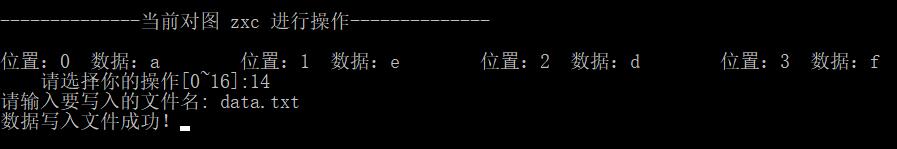


图4-47 数据保存



图4-48 数据保存

测试用例四：

再创建一个图，检验图切换和文件读写功能，操作对应测试计划42-51。

测试结果：

（1）再创建一个名为zxc的图，由于该名称已存在，创建失败。

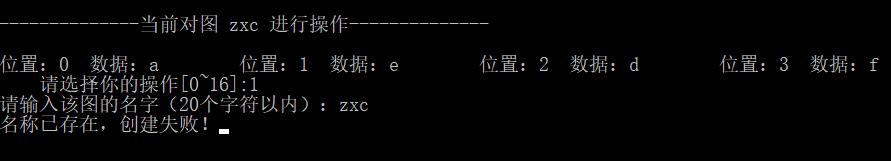


图4-49 创建图

（2）创建一个名为asd，顶点为qwert弧数为3的图，其中弧为<q,w><w,e><e,r>,的图，创建成功。

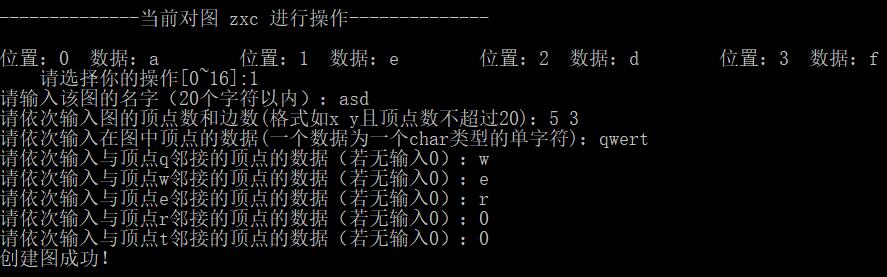


图4-50 创建图

（3）深度优先遍历图asd。

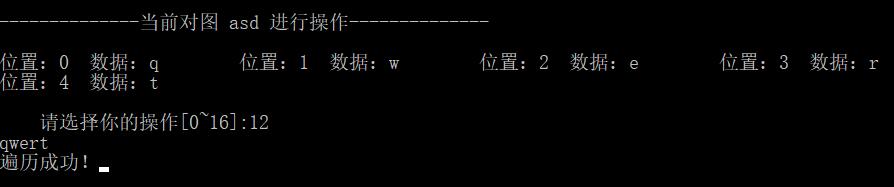


图4-51 深度优先遍历

（4）将工作图切换为名称为cv的图，由于不存在名为cv的图，切换失败。

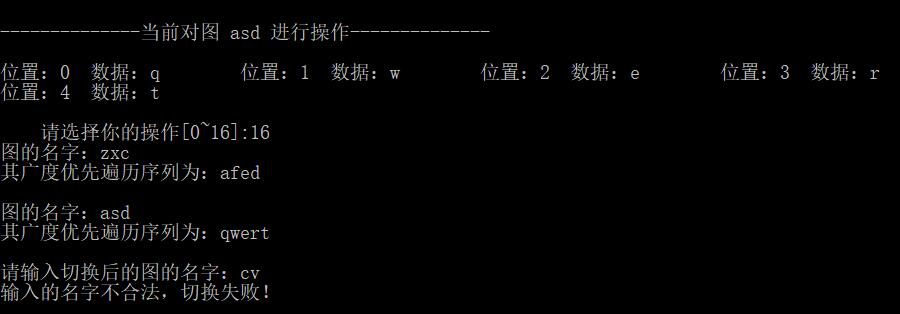


图4-52 切换图

（5）将工作图切换为名称为zxc的图，当前工作图为zxc。

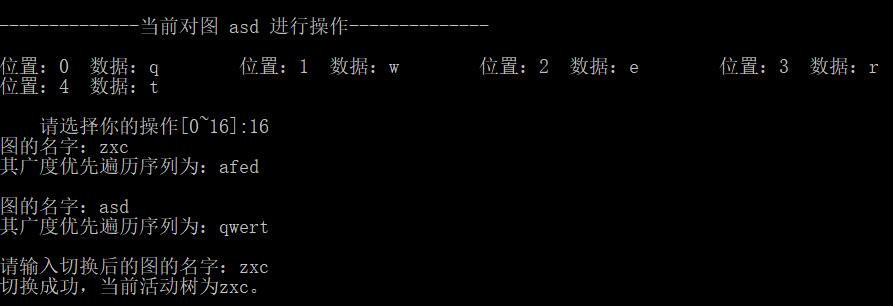


图4-53 切换图

（6）深度优先遍历工作图，由于已切换成功，所以遍历结果为afed。

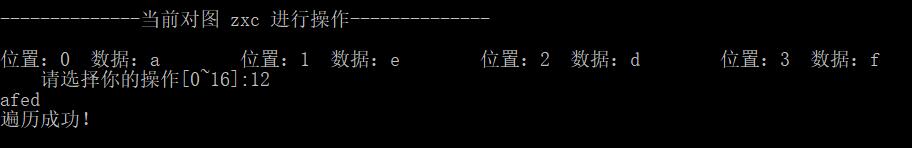


图4-54 深度优先遍历

（7）将工作图切换为名称为asd的图，当前工作图为asd。

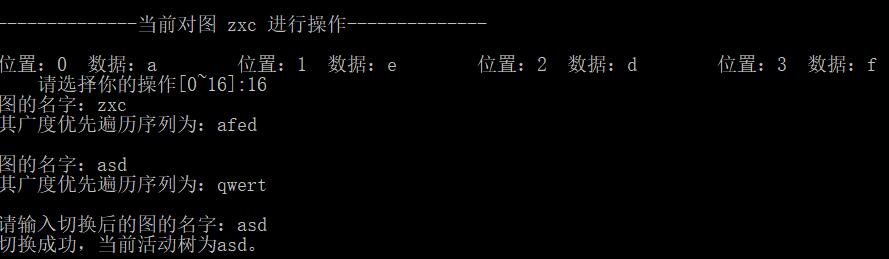


图4-55 切换图

（8）深度优先遍历工作图，由于已切换成功，所以遍历结果为qwert。

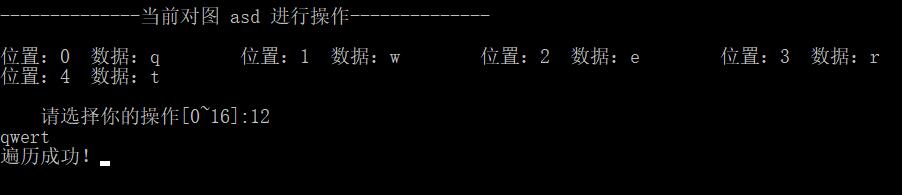


图4-56 深度优先遍历

（9）将文件data.txt中的数据加载到工作图中。

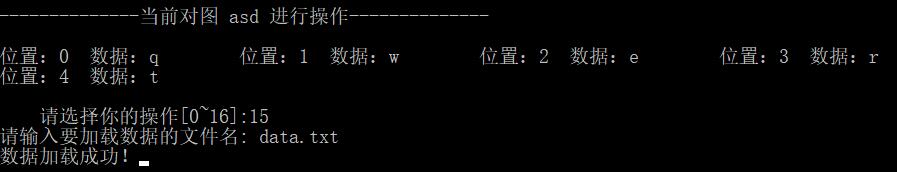


图4-57 数据加载

（10）深度优先遍历工作图，由于已加载数据成功，所以遍历结果为afed。

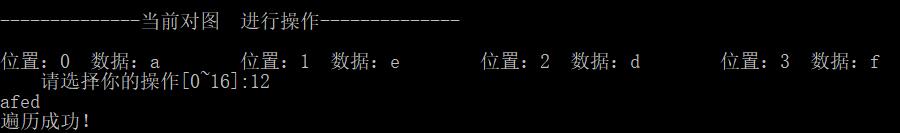


图4-58 深度优先遍历

（11）退出系统。

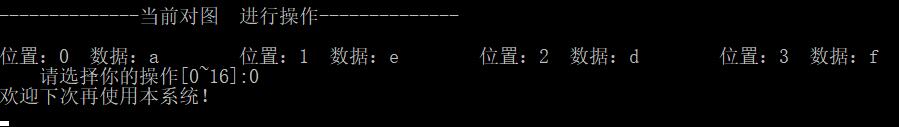


图4-59 退出系统

## 4.4 实验小结

本次实验的难点在于图的构建。最初写程序时没有按照任务书上的要求，自行定义函数的参数只有图G，在函数中输入一个数据，将一个数据存入图中，之后修改为任务书上的要求，参数重新定义为图G、顶点集V和顶点集VR，之后问题在于在构建图时终止条件，我选择的方式时将其余没有存数据的全部赋值为‘\0’，并以此为判断条件。

另外一个难点应该是两个遍历函数，最初自己写遍历时，没有考虑到如果图有两个及以上的连通分量，在连通图的情况下遍历正常，但在有多个连通分量的图中，仅遍历了位置为0的顶点所在的连通分量，最后参考了书本上关于两个遍历的伪码，对程序进行了修改，最后完成了要求。

# 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] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 指导教师评定意见

一、对实验报告的评语

|  |
| --- |
|  |

二、对实验报告评分

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 程序内容  (36.8分) | 程序规范  (9.2分) | 报告内容  (36.8分) | 报告规范  (9.2分) | 考勤  （8分） | 逾期扣分 | 合 计  (100分) |
| 得分 |  |  |  |  |  |  |  |

# 附录A 基于顺序存储结构线性表实现的源程序

#include<stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include<string.h>

#define FALSE -1

#define OK 1

#define ERROR -2

#define TRUE 1

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

struct SqList{

ElemType \* elem;//指向线性表数据的线性存储区

int length;//表中元素的个数

int listsize;//表的最大元素个数

};

struct MultiList {

SqList \*L; /\*指向一个线性表\*/

int num; /\*自动生成的表的编码\*/

struct MultiList \*next; /\*指向下一个表节点\*/

};

status IntiaList(SqList \*L);

status InitiaList2(MultiList \*head, MultiList \*\*List);

status DestroyList2(MultiList \*head, MultiList \*\*List);

status ClearList(SqList \*L);

status ListEmpty(SqList L);

int ListLength(SqList L);

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

status LocateElem(SqList L, ElemType e);

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

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

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

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

status ListTrabverse(SqList L);

status MultiListTrabverse(MultiList \*head);

void show();

void updata(MultiList \*head);

MultiList \*ExList(MultiList \*head);

status SaveList(FILE \*fp,SqList L, char filename[30]);

status LoadList(FILE \*fp,SqList \*L,char filename[30]);

int main(void) {

int op = 1;

int i = 0, cur;

ElemType e = 0,e1=0;

ElemType pre\_e = 0, next\_e = 0;

status a;

FILE \*fp;

char filename[30]; /\*存储文件名\*/

MultiList head; head.L = NULL; head.next = NULL;

MultiList \*List = head.next;

int num;

while (op) {

show();

if (List == NULL)

printf("\n-----------当前没有活动表-----------\n");

else

printf("\n-----------当前对表 %d 进行操作-----------\n", List->num);

scanf("%d", &op);getchar();

switch (op) {

case 1:

if (InitiaList2(&head, &List) == OK)

printf("线性表创建成功！\n");

else

printf("线性表创建失败！\n");

getchar();

break;

case 2:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

if (DestroyList2(&head, &List) != ERROR)

printf("线性表销毁成功！\n");

else

printf("线性表销毁失败！\n");

}

getchar();

break;

case 3:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

if (ClearList(List->L) == OK)

printf("线性表重置成功！\n");

else

printf("当前没有活动表！\n");

}

getchar();

break;

case 4:

if (List == NULL)

printf("当前没有活动表。\n");

else

ListEmpty(\*(List->L));

getchar();

break;

case 5:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

a=ListLength(\*(List->L));

if (a!= ERROR)

printf("线性表的长度为%d", a);

else

printf("当前没有活动表。\n");

}

getchar();

break;

case 6:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入所要查找的元素的位序：");

scanf("%d", &i);

if (GetElem(\*(List->L), i, &e) != ERROR)

printf("第%d的元素为%d", i, e);

getchar();

}

getchar();

break;

case 7:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入所要进行对比的元素e：");

scanf("%d", &e);

a = LocateElem(\*(List->L), e);

if (a != ERROR)

printf("第%d个元素与元素e相同。", a);

else

printf("不存在与e相同的元素！");

getchar();

}

getchar();

break;

case 8:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入数据元素cur：");

scanf("%d", &cur);

if (PriorElem(\*(List->L), cur, &pre\_e) != ERROR)

printf("%d的前驱数据元素为%d", cur, pre\_e);

getchar();

}

getchar();

break;

case 9:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入数据元素cur：");

scanf("%d", &cur);

if (NextElem(\*(List->L), cur, &next\_e) != ERROR)

printf("%d的后继数据元素为%d", cur, next\_e);

getchar();

}

getchar();

break;

case 10:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入要插入的元素e：");

scanf("%d", &e);

printf("请输入所要插入的位置i：");

scanf("%d", &i);

if (ListInsert(List->L, i, e) == OK)

printf("插入元素成功！");

else

printf("插入元素失败！");

getchar();

}

getchar();

break;

case 11:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入要删除的元素的位序i：");

scanf("%d", &i);

if (ListDelete(List->L, i, &e) == OK)

printf("位序为%d的元素的值为%d\n删除元素成功！", i, e);

else

printf("删除元素失败！");

getchar();

}

getchar();

break;

case 12:

if (List == NULL)

printf("当前没有活动表。\n");

else

ListTrabverse(\*(List->L));

getchar();

break;

case 13:

if (List == NULL)

printf("当前没有活动表,不能进行文件读取。\n");

else

SaveList(fp,\*List->L,filename);

getchar();

break;

case 14:

if (List == NULL)

printf("当前没有活动表,不能进行文件读取。\n");

else

LoadList(fp,List->L,filename);

getchar();

break;

case 15:

if (head.next == NULL)

printf("没有可操作的表。");

MultiListTrabverse(&head);

getchar();

break;

case 16:

if (head.next == NULL)

{

printf("没有可操作的表。");

getchar(); getchar();

break;

}

if(MultiListTrabverse(&head)==NULL)

break;

List=ExList(&head);

break;

case 0:

break;

default:

printf("功能选择是无效的输入。\n");

getchar();

}

updata(&head);

}

printf("欢迎下次再使用本系统！\n");

getchar();

}//end of main()

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：LoadList

\*函数参数：文件指针，线性表L,文件名filename[30]

\*函数功能：数据加载，为线性表输入元素。

\*返回值：加载成功返回OK，加载失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status LoadList(FILE \*fp,SqList \*L,char filename[30])

{

int e1,e;

printf("请输入要导入数据的文件名: ");

gets(filename);

if ((fp=fopen(filename, "r+")) == NULL)

{

printf("文件打开失败！\n");

return ERROR;

}

while (fscanf(fp, "%d", &e1) != EOF)

{

L->length = 0;

printf("表长：%d\n",e1);

for (int i = 0; i < e1; )

{

fscanf(fp, "%d ", &e);

i++;

printf("读入第%d个元素%d...\n", i, e);

ListInsert(L, i, e);

}

}

printf("导入完毕.");

fclose(fp);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：SaveList

\*函数参数：文件指针，线性表L,文件名filename[30]

\*函数功能：将L中的数据保存到fp指针指向的文件中

\*返回值：OK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status SaveList(FILE \*fp,SqList L,char filename[30])

{

printf("请输入要写入的文件名: ");

gets(filename);

if ((fp=fopen(filename, "w+"))== NULL)

{

printf("文件打开失败。");

getchar();

return ERROR;

}

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

{

if (i == -1)

{

fprintf(fp, "%d ", L.length);

continue;

}

fprintf(fp, "%d ", \*(L.elem + i));

}

printf("文件写入成功。");

fclose(fp);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函数名称：ExList

\* 函数参数：MultiList的头指针

\* 函数功能：进行多表切换操作

\* 返回值：一个指向切换后的活动表的指针

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

MultiList\* ExList(MultiList \*head)

{

int num;

printf("请输入想要进行操作的表的编号：");

scanf("%d",&num);

MultiList \* List = head;

for (int i = 0;i<num&&List!=NULL; i++)

List = List->next;

if(List==NULL)

{

printf("输入的编码不合法。\n");

getchar();getchar();

}

return List;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函数名称：updata

\* 函数参数：无

\* 函数功能：更新多表中的每个表对应的编号，便于切换表

\* 返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void updata(MultiList \*head)

{

MultiList \*p = head->next;

int i = 0;

while (p)

{

i++;

p->num = i;

p = p->next;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函数名称：show

\* 函数参数：无

\* 函数功能：显示伪菜单界面

\* 返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void show()

{

system("cls");

printf("\n\n");

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

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

printf(" 1. IntiaList 9. NextElem\n");

printf(" 2. DestroyList 10.ListInsert\n");

printf(" 3. ClearList 11.ListDelete\n");

printf(" 4. ListEmpty 12.ListTrabverse\n");

printf(" 5. ListLength 13.SaveList\n");

printf(" 6. GetElem 14.LoadList\n");

printf(" 7. LocateElem 15.MultiListTrabverse\n");

printf(" 8. PriorElem 16.ExList\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~16]:");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称:InitiaList

\*函数功能:初始化表L的数据指针elem，设置表长为L，表最大长度为LIST\_INIT\_SIZE

\*输入参数:L存放指向SqList的指针

\*输出参数:被初始化后的L指针

\*返回数值:当表已被初始化时返回ERROR,

\* 否则初始化表成功返回SUCCESS

\*调用说明:需要将L的elem成员置空才能初始化L

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status IntiaList(SqList \*L)

{

if(L->elem!=NULL)

return ERROR;

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

if (L->elem == NULL)//分配内存失败

return ERROR;

L->length = 0;//空表长度为0

L->listsize = LIST\_INIT\_SIZE;//初始存储容量

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函数名称：InitiaList2

\* 函数参数：MultiList的头指针head，遍历指针List

\* 函数功能：构造一个空的线性表

\* 返回值：成功构造返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status InitiaList2(MultiList \*head,MultiList \*\*List)

{

\*List = head;

MultiList \*q = \*List;

while (\*List)

{

q = \*List;

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

}

q->next = (MultiList \*)malloc(sizeof(MultiList));

q->next->L= (SqList \*)malloc(sizeof(SqList));

\*List = q->next;

(\*List)->L->elem = NULL;

if (IntiaList((\*List)->L) == ERROR)

{

return ERROR;

}

(\*List)->next = NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称:DestroyList

\*函数功能:释放表L的elem指针，使elem指针置空

\*输入参数:L存放指向SqList的指针

\*输出参数:无

\*返回数值:当表尚未被创建时返回ERROR,否则销毁成功返回OK

\*调用说明:无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status DestroyList(SqList \*L)

{

if (L->elem == NULL)

return ERROR;

free(L->elem);

free(L);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：DestroyList2

\*函数参数：MultiList的头指针head，遍历指针List，

\*函数功能：销毁MultiList的结点List，最后List为空。

\*返回值：成功销毁返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status DestroyList2(MultiList \*head, MultiList \*\*List)

{

MultiList \*p = head;

MultiList \*q = p;

while (p != \*List)

{

q = p;

p = p->next;

}

q->next = p->next;

DestroyList(p->L);

free(p);

\*List = NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ClearList

\*函数参数：线性表L的地址

\*函数功能：重置线性表

\*返回值：成功置空返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ClearList(SqList \*L)

{

if (L->elem)

{

free(L->elem);

L->length = 0;

return OK;

}

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListEmpty

\*函数参数：线性表L

\*函数功能：判断线性表是否为空

\*返回值：若L为空表则返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ListEmpty(SqList L)

{

if (L.elem)

{

if (L.length == 0)

{

printf("该表为空表。\n");

return OK;

}

else

{

printf("该表不为空表。\n");

return ERROR;

}

}

else

{

printf("不存在活动表。\n");

return ERROR;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListLength

\*函数参数：线性表L

\*函数功能：计算线性表L中数据元素个数；

\*返回值：成功返回L中数据元素个数，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int ListLength(SqList L)

{

if (L.elem)

return L.length;

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：GetElem

\*函数参数：线性表L，e为第i个数据的地址；

\*函数功能：查找并显示L中第i个元素的值；

\*返回值：成功则返回第i个元素的值，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

if (L.length == 0)

{

printf("该表为空表，查找失败。\n");

return ERROR;

}

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

{

printf("i值不合法，i的取值范围应为1<=i<=%d。\n", ListLength(L));

return ERROR;

}

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

return \*e;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：LocateElem

\*函数参数：线性表L，相对比的元素值；

\*函数功能：查找L中与e相同数据所在的位序；位序从1开始

\*返回值：成功则返回第一个与e相同的数据的位序，不存在则返回0，其余情况返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status LocateElem(SqList L, ElemType e)

{

int i = 0;

if (L.length == 0)

return ERROR;

while (L.elem[i] != e&&i<L.length) {

i++;

}

if (i >= L.length)

return ERROR;

else

return i + 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：PriorElem

\*函数参数：线性表L，查找的数据cur,前驱pre\_e；

\*函数功能：查找L中与cur相同的第一个数据，并返回其前驱

\*返回值：成功则返回与cur相同的第一个数据，并返回其第一个前驱；若无前驱以及其他情况则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

int i;

i = LocateElem(L, cur);

if(i==ERROR||i==0)

{

printf("%d 不是活动表中的元素\n",cur);

return ERROR;

}

else if (i>1) {

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

return OK;

}

else if (i == 1)

{

printf("%d is the first letter has no prior\_elem\n", cur);

return ERROR;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：NextElem

\*函数参数：线性表L，查找的数据cur\_e,后驱next\_e；

\*函数功能：查找L中与cur\_e相同的第一个数据，并返回其后驱

\*返回值：成功则返回与cur\_e相同的第一个数据，并返回其后驱；若无后驱以及其他情况则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

int i;

i = LocateElem(L, cur);

if(i==ERROR||i==0)

{

printf("%d 不是活动表中的元素\n",cur);

return ERROR;

}

else if (i>0) {

if (i < L.length) {

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

return OK;

}

else if(i==L.length)

{

printf("%d is the last letter has no next\_elem。\n", cur);

return ERROR;

}

}

else

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListInsert

\*函数参数：线性表L的地址，插入的位置i，插入的数据元素e。

\*函数功能：在L的第i个位置之前插入新的数据元素e；若已插满，则按照分配增量，分配更大的空间。

\*返回值：成功插入返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

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

{

printf("i值不合法，插入失败。\n");

return ERROR;

}

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

{

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

if (!newbase) return ERROR;//存储分配失败

L->elem = newbase;

L->listsize = L->listsize + LISTINCREMENT;

}

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

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

for (; p >= q; p--)

\*(p + 1) = \*p;

\*q = e;

L->length++;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListDelete

\*函数参数：线性表L的地址，删除的位序i，指针e的地址。

\*函数功能：删除L的第i个数据元素，用e返回其值。

\*返回值：成功删除返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

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

return ERROR;

ElemType \*p;

p = &L->elem[i - 1];

\*e = \*p;

ElemType \* q;

q = L->elem + L->length - 1;

for (p++; p <= q; p++)

\*(p - 1) = \*p;

L->length--;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListTrabverse

\*函数参数：线性表L

\*函数功能：依次显示线性表中的每个元素。

\*返回值：成功遍历返回线性表的长度，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ListTrabverse(SqList L) {

int i;

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

for (i = 0; i<L.length; i++) printf("%3d ", L.elem[i]);

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

if (L.length == 0)

{

printf("线性表是空表！\n");

return ERROR;

}

return L.length;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：MultiListTrabverse

\*函数参数：MultiList头指针head

\*函数功能：依次显示每一个线性表的编码、表长以及线性表中的每个元素。

\*返回值：成功遍历返回线性表的长度，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status MultiListTrabverse(MultiList \*head)

{

MultiList \*p = head->next;

while (p)

{

printf("工作表：%d\t表长度：%d\n", p->num, p->L->length);

ListTrabverse(\*p->L);

p = p->next;

}

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>

#include<string.h>

#define FALSE -1

#define OK 1

#define ERROR -2

#define TRUE 1

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

struct ListNode {

ElemType data;

struct ListNode \*next;

};

struct MultiList {

struct ListNode \*ListHead; /\*指向一个线性表的头节点\*/

int num; /\*线性表的编码\*/

struct MultiList \*next; /\*指向下一个表头节点\*/

};

void show(MultiList \*List);

status IntiaList(ListNode \*L);

status InitiaList2(MultiList \*head, MultiList \*\*List);

status DestroyList(ListNode\* L);

status DestroyList2(MultiList \*head, MultiList \*\*List);

status ClearList(ListNode \*L);

status ListEmpty(ListNode L);

int ListLength(ListNode L);

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

status LocateElem(ListNode L, ElemType e);

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

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

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

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

status ListTrabverse(ListNode L);

status MultiListTrabverse(MultiList head);

status SaveList(FILE \*fp, ListNode L, char filename[30]);

status LoadList(FILE \*fp, ListNode \*L, char filename[30]);

MultiList \*ExList(MultiList \*head);

int main()

{

struct MultiList\* head = (MultiList \*)malloc(sizeof(MultiList));

head->ListHead = NULL; head->next = NULL; head->num = 0;

struct MultiList\* List;

List = head->next;

FILE \*fp=NULL;

char filename[30]; /\*存储文件名\*/

int op = 1;

status a;

int i = 0, cur,num1,num2;

ElemType e = 0, e1 = 0;

ElemType pre\_e = 0, next\_e = 0;

while (op)

{

show(List);

scanf("%d", &op);

switch (op)

{

case 0:

break;

case 1:

if (InitiaList2(head, &List) == 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 (List == NULL)

printf("当前没有活动表。\n");

else

{

if (DestroyList2(head, &List) != ERROR)

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 (List == NULL)

printf("当前没有活动表。\n");

else

{

if (ClearList(List->ListHead) != ERROR)

printf("线性表置空成功！\n");

else

printf("线性表置空失败！\n");

}

getchar(); getchar();

break;

case 4:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

if (ListEmpty(\*(List->ListHead)) != ERROR)

printf("线性表为空表！\n");

else

printf("线性表不为空表！\n");

}

getchar(); getchar();

break;

case 5:

if (List == NULL)

printf("当前没有活动表。\n");

else

printf("线性表的长度为%d", ListLength(\*(List->ListHead)));

getchar(); getchar();

break;

case 6:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入所要查找的元素的位序：");

scanf("%d", &i);

if (GetElem(\*(List->ListHead), i, &e) != ERROR)

printf("第%d的元素为%d", i, e);

}

getchar();getchar();

break;

case 7:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入所要进行对比的元素e：");

scanf("%d", &e);

a = LocateElem(\*(List->ListHead), e);

if (a != ERROR&&a!=0)

printf("第%d个元素与元素e相同。", a);

else if(a==0)

printf("不存在与e相同的元素！");

else

printf("该表为空表，定位失败。\n");

}

getchar();getchar();

break;

case 8:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入数据元素cur：");

scanf("%d", &cur);

if (PriorElem(\*(List->ListHead), cur, &pre\_e) == OK)

printf("%d的前驱数据元素为%d", cur, pre\_e);

}

getchar();getchar();

break;

case 9:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入数据元素cur：");

scanf("%d", &cur);

if (NextElem(\*(List->ListHead), cur, &next\_e) != ERROR)

printf("%d的后继数据元素为%d", cur, next\_e);

}

getchar();getchar();

break;

case 10:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入所要插入的位置i：");

scanf("%d", &i);

printf("请输入要插入的元素e：");

scanf("%d", &e);

if (ListInsert(List->ListHead, i, e) == OK)

printf("插入元素成功！");

else

printf("插入元素失败！");

}

getchar();getchar();

break;

case 11:

if (List == NULL)

printf("当前没有活动表。\n");

else

{

printf("请输入要删除的元素的位序i：");

scanf("%d", &i);

if (ListDelete(List->ListHead, i, &e) == OK)

printf("位序为%d的元素的值为%d\n删除元素成功！", i, e);

else

printf("删除元素失败！");

}

getchar();getchar();

break;

case 12:

if (List == NULL)

printf("当前没有活动表。\n");

else

ListTrabverse(\*(List->ListHead));

getchar(); getchar();

break;

case 13:

if (List == NULL)

printf("当前没有活动表,不能进行文件读取。\n");

else

SaveList(fp, \*(List->ListHead), filename);

getchar();getchar();

break;

case 14:

if (List == NULL)

printf("当前没有活动表,不能进行文件读取。\n");

else

LoadList(fp, List->ListHead, filename);

getchar();getchar();

break;

case 15:

if (head->next== NULL)

{

printf("没有可操作的表。");

getchar(); getchar();

break;

}

if (MultiListTrabverse(\*head) == NULL)

break;

List = ExList(head);

getchar();

break;

default:

printf("功能选择是无效的输入。\n");

getchar(); getchar();

}

}

printf("欢迎下次再使用本系统！\n");

getchar(); getchar();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函数名称：ExList

\* 函数参数：MultiList的头指针

\* 函数功能：进行多表切换操作

\* 返回值：一个指向切换后的活动表的指针

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

MultiList\* ExList(MultiList \*head)

{

int num;

printf("请输入想要进行操作的表的编号：");

scanf("%d", &num);

MultiList \* List = head->next;

if(num<1)

{

printf("输入的编码不合法。\n");

getchar();

}

for (int i = 1; i<num&&List != NULL; i++)

List = List->next;

if (List == NULL)

{

printf("输入的编码不合法。\n");

getchar();

}

return List;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：MultiListTrabverse

\*函数参数：MultiList头指针head

\*函数功能：依次显示每一个线性表的编码、表长以及线性表中的每个元素。

\*返回值：成功遍历返回总线性表的长度，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status MultiListTrabverse(MultiList head)

{

MultiList \*p = (&head)->next;

if (p == NULL)

printf("当前无线性表存在。\n");

while (p)

{

printf("工作表：%d\t", p->num);

ListTrabverse(\*p->ListHead);

printf("\n");

p = p->next;

}

getchar();

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：LoadList

\*函数参数：文件指针，线性表L,文件名filename[30]

\*函数功能：数据加载，为线性表输入元素。

\*返回值：加载成功返回OK，加载失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status LoadList(FILE \*fp, ListNode \*L, char filename[30])

{

int e,i=0;

printf("请输入要导入数据的文件名: ");

scanf("%s",filename,30);

if ((fp=fopen(filename, "r+")) == NULL)

{

printf("文件打开失败！\n");

return ERROR;

}

L->data = 0;

while (fscanf(fp, "%d", &e) != EOF)

{

i++;

printf("读入第%d个元素%d...\n", i, e);

ListInsert(L, i, e);

}

getchar();

printf("导入完毕.");

fclose(fp);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：SaveList

\*函数参数：文件指针，线性表L的头结点，文件名filename[30]

\*函数功能：将L中的数据保存到fp指针指向的文件中

\*返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status SaveList(FILE \*fp, ListNode L, char filename[30])

{

printf("请输入要写入的文件名: ");

scanf("%s",filename);

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

{

printf("文件打开失败。");

getchar();

return ERROR;

}

ListNode\*p = &L;

for (int i = 0; i < L.data&&p!=NULL; i++)

{

p = p->next;

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

}

printf("文件写入成功。\n");

fclose(fp);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListTrabverse

\*函数参数：线性表L的头结点

\*函数功能：依次显示线性表中的每个元素。

\*返回值：成功遍历返回线性表的长度，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ListTrabverse(ListNode L)

{

ListNode\*q = &L;

int i;

if (L.data == 0)

{

printf("线性表是空表！\n");

}

else

printf("表长：%d\n", L.data);

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

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

{

q = q->next;

printf("%3d ", q->data);

}

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

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListDelete

\*函数参数：线性表L的头指针，删除的位序i，指针e的地址。

\*函数功能：删除L的第i个数据元素，用e返回其值。

\*返回值：成功删除返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

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

{

printf("i值不合法，");

return ERROR;

}

ListNode \*q = L;

for (int j = 0; (j + 1) != i; j++)

q = q->next;

ListNode \*p = q->next;

q->next = p->next;

\*e = p->data;

p->next = NULL;

free(p);

L->data--;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListInsert

\*函数参数：线性表L的头指针，插入的位置i，插入的数据元素e。

\*函数功能：在L的第i个位置之前插入新的数据元素e。

\*返回值：成功插入返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

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

{

printf("i值不合法，插入失败。\n");

return ERROR;

}

ListNode \*q = L;

for (int j = 0; (j + 1) != i; j++)

q = q->next;

ListNode \*temp=(ListNode\*)malloc(sizeof(ListNode));

temp->data = e;

temp->next=q->next;

q->next = temp;

L->data++;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：NextElem

\*函数参数：线性表L的头结点，查找的数据cur\_e,后驱next\_e；

\*函数功能：查找L中与cur\_e相同的第一个数据，并用next\_e返回其后继

\*返回值：成功则返回与cur\_e相同的第一个数据，并返回OK；若无后驱以及其他情况则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

int i= LocateElem(L, cur);

if (i == ERROR || i == 0)

{

printf("%d 不是活动表中的元素\n", cur);

return ERROR;

}

else if (i == L.data)

{

printf("%d is the last letter has no next\_elem。\n", cur);

return ERROR;

}

else

{

ListNode \*q = &L;

for (int j = 0; (j -1)!= i; j++)

q = q->next;

\*next\_e = q->data;

return OK;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：PriorElem

\*函数参数：线性表L的头结点，查找的数据cur,前驱pre\_e；

\*函数功能：查找L中与cur相同的第一个数据，并用pre\_e返回其前驱

\*返回值：成功则返回与cur相同的第一个数据，并返回OK；若无前驱以及其他情况则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

int i = LocateElem(L, cur);

if (i == ERROR || i == 0)

{

printf("%d 不是活动表中的元素\n", cur);

return ERROR;

}

else if (i == 1)

{

printf("%d is the first letter has no prior\_elem\n", cur);

return ERROR;

}

else {

ListNode \*q = &L;

for (int j = 0; (j+1) != i; j++)

q = q->next;

\*pre\_e = q->data;

return OK;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：LocateElem

\*函数参数：线性表L的头结点，相对比的元素值；

\*函数功能：查找L中与e相同数据所在的位序；位序从1开始

\*返回值：成功则返回第一个与e相同的数据的位序，不存在则返回0，其余情况返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status LocateElem(ListNode L, ElemType e)

{

int i = 1;

if (L.data == 0)

return ERROR;

ListNode \*q = (&L)->next;

for (; q->data != e; )

{

q = q->next;

i++;

if (q == NULL)break;

}

if (i > L.data)

return 0;

else

return i ;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：GetElem

\*函数参数：线性表L的头结点，e为第i个数据的地址；

\*函数功能：查找并用e返回L中第i个元素的值；

\*返回值：成功则返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

if (L.data == 0)

{

printf("该表为空表，查找失败。\n");

return ERROR;

}

if (i < 1 || i>L.data)

{

printf("i值不合法，i的取值范围应为1<=i<=%d。\n", ListLength(L));

return ERROR;

}

ListNode \*q = &L;

for (int j=0;j!=i;j++)

q = q->next;

\*e = q->data;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListLength

\*函数参数：线性表L的头结点

\*函数功能：计算线性表L中数据元素个数；

\*返回值：成功返回L中数据元素个数，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int ListLength(ListNode L)

{

return L.data;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ListEmpty

\*函数参数：线性表L的头结点

\*函数功能：判断线性表是否为空

\*返回值：若L为空表则返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ListEmpty(ListNode L)

{

if (L.data == 0)

return OK;

else

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：ClearList

\*函数参数：线性表L的头指针

\*函数功能：将线性表L重置为一个空表

\*返回值：成功置空返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status ClearList(ListNode \*L)

{

if (L->next == NULL)

return OK;

DestroyList(L->next);

L->data = 0;

L->next = NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称:DestroyList

\*函数功能:释放表L的每一个结点的空间及头指针的空间

\*输入参数:L存放指向ListNode的指针

\*输出参数:无

\*返回数值:失败返回ERROR,否则销毁成功返回OK

\*调用说明:无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status DestroyList(ListNode \*L)

{

ListNode \*q;

while (L)

{

q = L->next;

free(L);

L = q;

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称：DestroyList2

\*函数参数：MultiList的头指针head，遍历指针List，

\*函数功能：销毁MultiList的结点List，最后List为空。

\*返回值：成功销毁返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status DestroyList2(MultiList \*head, MultiList \*\*List)

{

MultiList \*p = head;

MultiList \*q = p;

while (p != \*List)

{

q = p;

p = p->next;

}

q->next = p->next;

if (DestroyList(p->ListHead) == ERROR)

return ERROR;

free(p);

\*List = NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*函数名称:InitiaList

\*函数功能:初始化表L,

\*输入参数:List->HeadList,

\*输出参数:被初始化后的L指针

\*返回数值:当分配内存失败时返回ERROR,

\* 否则初始化表成功返回SUCCESS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status IntiaList(ListNode \*L)

{

if (L == NULL)//分配内存失败

return ERROR;

L->next = NULL;

L->data = 0;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函数名称：InitiaList2

\* 函数参数：MultiList的头指针head，遍历指针List

\* 函数功能：构造一个空的MultiList节点

\* 返回值：成功构造返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status InitiaList2(MultiList \*head, MultiList \*\*List)

{

MultiList \*q = head;

while (q->next)

{

q = q->next;

}

q->next = (MultiList \*)malloc(sizeof(MultiList));

\*List = q->next;

(\*List)->num = q->num + 1;

(\*List)->ListHead = (ListNode\*)malloc(sizeof(ListNode));

if (IntiaList((\*List)->ListHead) == ERROR)

return ERROR;

(\*List)->next = NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函数名称：show

\* 函数参数：遍历指针List

\* 函数功能：显示伪菜单界面

\* 返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void show(MultiList \*List)

{

system("cls");

printf("\n\n");

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

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

printf(" 1. IntiaList 9. NextElem\n");

printf(" 2. DestroyList 10.ListInsert\n");

printf(" 3. ClearList 11.ListDelete\n");

printf(" 4. ListEmpty 12.ListTrabverse\n");

printf(" 5. ListLength 13.SaveList\n");

printf(" 6. GetElem 14.LoadList\n");

printf(" 7. LocateElem 15.ExList\n");

printf(" 8. PriorElem 0. Exit\n");

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

if (List == NULL)

printf("\n-----------当前没有活动表-----------\n");

else

printf("\n-----------当前对表 %d 进行操作-----------\n", List->num);

printf(" 请选择你的操作[0~15]:");

}

# 附录C 基于二叉链表二叉树实现的源程序

#include<stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include<string.h>

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

#define FALSE -1

#define OK 1

#define ERROR -2

#define TRUE 1

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int Key; //定义关键字类型为int

typedef struct TElemType {

Key i; //关键字，本系统采用完全二叉树中对应的位置

char c; //字符型数据

}TElemType;

typedef struct BiTNode {

TElemType data; //二叉树结点中的数据域

BiTNode \*lchild, \*rchild; //左右孩子指针

}BiTNode;

typedef struct BiTree {

char name[20]; //二叉树的名字

BiTNode\* HeadNode; //二叉树根结点

}BiTree;

typedef struct Forest {

BiTree \*elem; //数据域

int length; //二叉树的个数

int listsize;

}Forest;

//---------------------------------------队列的定义和基本操作，用于层次遍历二叉树---------------------------------------//

struct LinkQueueNode {

BiTNode \*data;

struct LinkQueueNode \*next;

};

struct LKQueue {

LinkQueueNode\*front, \*rear;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：InitQueue

函数参数：队列LQ

函数功能：初始化一个队列

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void InitQueue(LKQueue \*LQ)

{

LinkQueueNode\*p;

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

LQ->front = p;

LQ->rear = p;

LQ->front->next = NULL;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：EmptyQueue

函数参数：队列LQ

函数功能：判断队列是否为空

返回值：若队列为空返回1，否则返回0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int EmptyQueue(LKQueue\*LQ)

{

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

return 1;

else

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：EnQueue

函数参数：队列LQ，结点T

函数功能：将T结点插入队列尾端

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void EnQueue(LKQueue\*LQ, BiTNode\*T)

{

LinkQueueNode \*p;

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

p->data = T;

p->next = NULL;

LQ->rear->next = p;

LQ->rear = p;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：OutQueue

函数参数：队列LQ

函数功能：将队列第一个元素出列

返回值：成功返回1，失败即队列已为空时返回0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int OutQueue(LKQueue\*LQ)

{

LinkQueueNode \*p;

if (EmptyQueue(LQ))

{

return 0;

}

else

{

p = LQ->front->next;

LQ->front->next = p->next;

if (p->next == NULL)

LQ->rear = LQ->front;

free(p);

return 1;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：GetHead

函数参数：队列LQ

函数功能：返回队列第一个元素

返回值：返回第一个元素，若队列为空则返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BiTNode \*GetHead(LKQueue \*LQ)

{

LinkQueueNode \*p;

BiTNode \*T=NULL;

if (EmptyQueue(LQ))

return T;

else

{

p = LQ->front->next;

return p->data;

}

}

//---------------------------------------功能函数的声明---------------------------------------//

void show(BiTree\*F);//显示伪菜单

void updata(BiTNode \*F);//将对应的位置赋值给关键字

int InitBiTree(Forest \*L);//初始化一个空二叉树

int DestroyBiTree(Forest L,BiTree \*F);//销毁二叉树

int DestroyBiTree2(BiTNode \*T);

BiTNode \*CreateBiTree(char\*definition);//按definition的序列生成二叉树

int PreCreateBiTree(BiTNode \*\*T,char\*definition,int \*i);

int ClearBiTree(BiTNode \*T);//请空二叉树

int BiTreeEmpty(BiTNode T);//判断二叉树是否为空

int BiTreeDepth(BiTNode \*T);//返回二叉树的深度

BiTNode\* Root(BiTNode \*T);//返回二叉树的根结点

BiTNode\* Value(BiTNode \*T,Key e);//返回关键字为e的结点

int Assign(BiTNode \*T,Key e,char c);//修改关键字为e的结点

BiTNode\* Parent(BiTNode \*T,Key e);//返回关键字为e的结点的双亲结点

BiTNode\* LeftChild(BiTNode\*T,Key e);//返回关键字为e的结点的左孩子结点

BiTNode\* RightChild(BiTNode\*T,Key e);//返回关键字为e的结点的右孩子结点

BiTNode\* LeftSibling(BiTNode\*T,Key e);//返回关键字为e的结点的左兄弟结点

BiTNode\* RightSibling(BiTNode\*T,Key e);//返回关键字为e的结点的右兄弟结点

int InsertChild(BiTNode\*T, Key e, int LR,BiTNode\*p);//插入子树

void change(BiTNode \*T);//按照树变二叉树的规则将二叉树变为右子树为空的二叉树

int DeleteChild(BiTNode\*T, Key e, int LR);//删除子树

int PreOrderTraverse(BiTNode \*T);//先序遍历

int InOrderTraverse(BiTNode \*T);//中序遍历

int PostOrderTraverse(BiTNode \*T);//后序遍历

int LevelOrderTraverse(BiTNode \*T);//层次遍历

int SaveData(FILE \*fp, BiTNode\*T, char filename[30]);//保存数据

int SaveData1(FILE \*fp, BiTNode\*T, char filename[30]);

int LoadData(FILE \*fp, BiTNode\*\*T, char filename[30]);//加载数据

int ForestTraverse(Forest L);//遍历森林

BiTree \*ExBiTree(Forest L, char name[20]);//切换树

int main()

{

Forest L;

L.elem = (BiTree \*)malloc(LIST\_INIT\_SIZE \* sizeof(BiTree));

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

BiTree \*F=NULL,\*F1;//F为遍历指针，F1用于接收返回的树结点

BiTNode \*T;//用于接收返回的二叉树结点指针

FILE \*fp = NULL;//文件指针

char filename[30];//存储文件名

int op = 1;//功能选择

char s[100];//存储录入的字符串用于生成二叉树，或用于存储读入的二叉树名称

Key e;char a;//数据域临时变量

int LR;//左或右的选择

while (op)

{

show(F);

scanf("%d", &op); getchar();

switch (op)

{

case 0:

printf("欢迎下次再使用本系统！\n");

getchar();

break;

case 1:

if (InitBiTree(&L)== OK)

{

F=&(L.elem[L.length-1]);

printf("创建一个空二叉树成功！\n");

}

else

printf("创建一个空二叉树失败！\n");

getchar();

break;

case 2:

if (DestroyBiTree(L,F)== OK)

{

F=NULL;

printf("销毁二叉树成功！\n");

}

else

printf("销毁二叉树失败！\n");

getchar();

break;

case 3:

if (F == NULL)

printf("当前没有活动树！\n");

else

{

if (F->HeadNode->lchild != NULL)

printf("当前活动树中已有数据，不能重新生成！");

else

{

printf("请输入先序遍历序列（每一个元素都为char类型,若为空则输入#）：\n");

fgets(s, 100, stdin);

F->HeadNode->lchild = CreateBiTree(s);

if (s[0] != ' '&&s[0]!='#'&&F->HeadNode->lchild == NULL)

printf("构建二叉树失败！请确认输入的序列是否是一个正确的先序遍历序列！");

else

printf("构建二叉树成功！");

}

}

getchar();

break;

case 4:

if (F == NULL)

printf("当前没有活动树！\n");

else if (ClearBiTree(F->HeadNode) == OK)

printf("二叉树清空成功！");

else

printf("二叉树清空失败！");

getchar();

break;

case 5:

if (F == NULL)

printf("当前没有活动树！\n");

else if (BiTreeEmpty(\*(F->HeadNode)) == OK)

printf("当前活动树为空二叉树！");

else

printf("当前活动树不是空二叉树！");

getchar();

break;

case 6:

if (F == NULL)

printf("当前没有活动树！\n");

else

printf("二叉树的深度为%d。",BiTreeDepth(F->HeadNode->lchild));

getchar();

break;

case 7:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("根结点为空，该活动树为空二叉树！");

else

printf("根结点的存放的数据为：%c。", Root(F->HeadNode)->data.c);

getchar();

break;

case 8:

if (F == NULL)

printf("当前没有活动树！\n");

else if (F->HeadNode->lchild == NULL)

printf("该活动树为空二叉树，不能进行查找！");

else

{

printf("请输入要查找的数据的关键字:");scanf("%d", &e);getchar();

T = Value(F->HeadNode->lchild, e);

if (T!=NULL)

printf("关键字为%d的数据为%c。", e, T->data.c);

else

printf("没有找到含有该关键字的结点！");

}

getchar();

break;

case 9:

if (F == NULL)

printf("当前没有活动树！\n");

else if(F->HeadNode->lchild==NULL)

printf("该活动树为空二叉树，无法进行修改！");

else

{

printf("请输入要修改的数据的关键字:");scanf("%d", &e);getchar();

printf("请输入要修改后的数据:");scanf("%c", &a);getchar();

if (Assign(F->HeadNode->lchild, e, a) == OK)

printf("修改成功！");

else

printf("没有找到含有该关键字的结点！");

}

getchar();

break;

case 10:

if (F == NULL)

printf("当前没有活动树！\n");

else if(F->HeadNode->lchild==NULL)

printf("该活动树为空二叉树，无法进行查找！");

else

{

printf("请输入要查找的数据的关键字:"); scanf("%d", &e); getchar();

if (e == 1)

printf("该关键字对应数据结点为根节点，无双亲结点！");

else

{

T = Parent(F->HeadNode->lchild, e);

if (T != NULL)

printf("关键字为%d的数据的双亲结点的关键字：%d 数据：%c。", e, T->data.i, T->data.c);

else

printf("没有找到含有该关键字的结点,查找失败！");

}

}

getchar();

break;

case 11:

if (F == NULL)

printf("当前没有活动树！\n");

else if(F->HeadNode->lchild==NULL)

printf("该活动树为空二叉树，无法进行查找！");

else

{

printf("请输入要查找的数据的关键字:"); scanf("%d", &e); getchar();

T=LeftChild(F->HeadNode->lchild, e);

if (T == NULL)

printf("没有找到含有该关键字的结点或是含有该关键字的结点没有左孩子结点！");

else

printf("关键字为%d的结点的左孩子结点的关键字：%d 数据：%c。", e,T->data.i,T->data.c);

}

getchar();

break;

case 12:

if (F == NULL)

printf("当前没有活动树！\n");

else if(F->HeadNode->lchild==NULL)

printf("该活动树为空二叉树，无法进行查找！");

else

{

printf("请输入要查找的数据的关键字:"); scanf("%d", &e); getchar();

T = RightChild(F->HeadNode->lchild, e);

if (T == NULL)

printf("没有找到含有该关键字的结点或是含有该关键字的结点没有右孩子结点！");

else

printf("关键字为%d的结点的右孩子结点的关键字：%d 数据：%c。", e, T->data.i, T->data.c);

}

getchar();

break;

case 13:

if (F == NULL)

printf("当前没有活动树！\n");

else if(F->HeadNode->lchild==NULL)

printf("该活动树为空二叉树，无法进行查找！");

else

{

printf("请输入要查找的数据的关键字:"); scanf("%d", &e); getchar();

if (e == 1)printf("该关键字对应数据结点为根节点，无左兄弟结点！");

else

{

T = LeftSibling(F->HeadNode->lchild, e);

if (T == NULL)

printf("没有找到含有该关键字的结点！");

else if (T->data.i == e)

printf("含有该关键字的结点为某一结点的左孩子结点，无左兄弟结点！");

else

printf("关键字为%d的数据的左兄弟结点的关键字：%d 数据：%c。", e, T->data.i, T->data.c);

}

}

getchar();

break;

case 14:

if (F == NULL)

printf("当前没有活动树！\n");

else if(F->HeadNode->lchild==NULL)

printf("该活动树为空二叉树，无法进行查找！");

else

{

printf("请输入要查找的数据的关键字:"); scanf("%d", &e); getchar();

if (e == 1)printf("该关键字对应数据结点为根节点，无右兄弟结点！");

else

{

T = RightSibling(F->HeadNode->lchild, e);

if (T == NULL)

printf("没有找到含有该关键字的结点！");

else if (T->data.i == e)

printf("含有该关键字的结点为某一结点的右孩子结点，无右兄弟结点！");

else

printf("关键字为%d的数据的右兄弟结点的关键字：%d 数据：%c！", e, T->data.i, T->data.c);

}

}

getchar();

break;

case 15:

if (F == NULL)

printf("当前没有活动树！\n");

else

{

printf("请输入子树的先序遍历序列（每一个元素都为char类型,若为空则输入#）：\n");

fgets(s, 30, stdin);

if (s[0] == '\n')

printf("未输入任意字符，构建二叉树失败！");

else if(s[0]=='#')

{

F->HeadNode->lchild == NULL;

printf("构建二叉树成功，该树为空树,无法进行插入！");

}

else

{

T = CreateBiTree(s);

if (s[0] != ' '&&T == NULL)

printf("构建二叉树失败！请确认输入的序列是否是一个正确的先序遍历序列！");

else

{

printf("构建二叉树成功！\n");

printf("请输入所要插入位置的双亲结点的关键字(根结点的双亲结点定义为表头结点)："); scanf("%d", &e); getchar();

if(e!=0)

{

printf("以0代表插入左子树，1代表插入右子树，请输入LR的值："); scanf("%d", &LR); getchar();

}

if(InsertChild(F->HeadNode, e,LR,T)==OK)

printf("插入成功！");

else

printf("插入失败！");

}

}

}

getchar();

break;

case 16:

if (F == NULL)

printf("当前没有活动树！\n");

else if (DeleteChild(F->HeadNode, e, LR) == OK)

printf("删除成功！");

else

printf("删除失败！");

getchar();

break;

case 17:

if (F == NULL)

printf("当前没有活动树！");

else if (F->HeadNode->lchild == NULL)

printf("该树为空二叉树！");

else

PreOrderTraverse(F->HeadNode->lchild);

getchar();

break;

case 18:

if (F == NULL)

printf("当前没有活动树！");

else if (F->HeadNode->lchild == NULL)

printf("该树为空二叉树！");

else

InOrderTraverse(F->HeadNode->lchild);

getchar();

break;

case 19:

if (F == NULL)

printf("当前没有活动树！");

else if (F->HeadNode->lchild == NULL)

printf("该树为空二叉树！");

else

PostOrderTraverse(F->HeadNode->lchild);

getchar();

break;

case 20:

if (F == NULL)

printf("当前没有活动树！");

else if (F->HeadNode->lchild == NULL)

printf("该树为空二叉树！");

else

LevelOrderTraverse(F->HeadNode->lchild);

getchar();

break;

case 21:

if (F == NULL)

printf("当前没有活动树！\n");

else if (SaveData(fp,F->HeadNode->lchild,filename)== OK)

printf("数据写入文件成功！");

else

printf("数据写入文件失败！");

getchar();

break;

case 22:

if (F == NULL)

printf("当前没有活动树！\n");

else if (LoadData(fp, &(F->HeadNode->lchild), filename) == OK)

printf("录入数据成功！");

else

printf("录入数据失败！");

getchar();

break;

case 23:

if (L.elem == NULL)

printf("当前没有活动树！\n");

else

{

ForestTraverse(L);

printf("请输入要切换的表的名字："); scanf("%s", &s); getchar();

F1=ExBiTree(L, s);

if (F1 == NULL)

printf("输入的名字不存在！\n");

else

{

F=F1;

printf("切换成功！");

}

}

getchar();

break;

default:

printf("功能选择是无效的输入！\n");

getchar();

}

if(F!=NULL)

if(F->HeadNode!=NULL)

updata(F->HeadNode->lchild);

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：ForestTraverse

函数参数：存放森林的顺序表

函数功能：遍历全部的树

返回值：成功返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int ForestTraverse(Forest L)

{

int i;

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

{

printf("树的名字：%s\n", L.elem[i].name);

if(L.elem[i].HeadNode->lchild==NULL)

printf("该树为空树！");

else

PreOrderTraverse(L.elem[i].HeadNode->lchild);

printf("\n\n");

}

printf("遍历结束，");

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：ExBiTree

函数参数：存放森林的顺序表 ，要切换的树的名字

函数功能：切换工作树

返回值：切换成功则返回切换后森林结构的结点，否则返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BiTree \*ExBiTree(Forest L, char name[20])

{

int i;

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

{

if (strcmp(L.elem[i].name,name) == 0)

break;

}

if (i+1>L.length)

{

printf("二叉树不存在，");

return NULL;

}

else

return &(L.elem[i]);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：LoadData

函数参数：文件指针，二叉树的表头结点T,文件名filename[30]

函数功能：加载数据

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int LoadData(FILE \*fp, BiTNode\*\*T, char filename[30])

{

printf("请输入要导入数据的文件名: ");

scanf("%s", filename);getchar();

if ((fp=fopen(filename, "r+")) == NULL)

{

printf("文件打开失败！\n");

return ERROR;

}

char \*temp;

fgets(temp, 1024, fp);

if (\*T == NULL)

{

\*T=CreateBiTree(temp);

if((\*T==NULL&&temp=='\0')||temp!='\0'&&\*T!=NULL)

{

printf("导入完毕，");

fclose(fp);

return OK;

}

}

if (ClearBiTree((\*T)) == ERROR)

return ERROR;

\*T=CreateBiTree(temp);

if((\*T==NULL&&temp=='\0')||temp!='\0'&&\*T!=NULL)

{

printf("导入完毕，");

fclose(fp);

return OK;

}

fclose(fp);

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：SaveData

函数参数：文件指针，二叉树的表头结点T,文件名filename[30]

函数功能：保存数据

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int SaveData(FILE \*fp, BiTNode\*T, char filename[30])

{

printf("请输入要写入的文件名: "); scanf("%s", filename);getchar();

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

{

printf("文件打开失败！");

return ERROR;

}

if (SaveData1(fp, T, filename) == OK)

{

fclose(fp);

return OK;

}

fclose(fp);

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：SaveData1

函数参数：文件指针，二叉树的表头结点T,文件名filename[30]

函数功能：按先序遍历序列向文件filename中依次写入数据的关键字和字符

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int SaveData1(FILE \*fp, BiTNode\*T, char filename[30])

{

if (T) {

fprintf(fp, "%c",T->data.c);

SaveData1(fp, T->lchild, filename);

SaveData1(fp, T->rchild, filename);

}

else

fprintf(fp, "%c",'#');

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：LevelOrderTraverse

函数参数：指向二叉树的根结点的指针F->HeadNode->lchild

函数功能：按层遍历二叉树

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int LevelOrderTraverse(BiTNode \*T)

{

LKQueue\*Q;

BiTNode \*p;

InitQueue(Q);

if (T!= NULL)

{

EnQueue(Q, T);

while (!EmptyQueue(Q))

{

p = GetHead(Q);

OutQueue(Q);

if(p==NULL)

putchar('#');

else

printf("%c",p->data.c);

if (p->lchild != NULL)

EnQueue(Q, p->lchild);

if (p->rchild != NULL)

EnQueue(Q, p->rchild);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：PostOrderTraverse

函数参数：指向二叉树的根结点的指针F->HeadNode->lchild

函数功能：后序遍历二叉树

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int PostOrderTraverse(BiTNode \*T)

{

if (T)

{

PostOrderTraverse(T->lchild);

PostOrderTraverse(T->rchild);

printf("%c", T->data.c);

}

else

putchar('#');

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：InOrderTraverse

函数参数：指向二叉树的根结点的指针F->HeadNode->lchild

函数功能：中序遍历二叉树

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int InOrderTraverse(BiTNode \*T)

{

if (T)

{

InOrderTraverse(T->lchild);

printf("%c", T->data.c);

InOrderTraverse(T->rchild);

}

else

putchar('#');

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：PreOrderTraverse

函数参数：指向二叉树的根结点的指针F->HeadNode->lchild

函数功能：前序遍历二叉树

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int PreOrderTraverse(BiTNode \*T)

{

if (T) {

printf("%c", T->data.c);

PreOrderTraverse(T->lchild);

PreOrderTraverse(T->rchild);

}

else

putchar('#');

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DeleteChild

函数参数：二叉树的根结点，关键字e ，判断标志LR

函数功能：删除子树

返回值：成功插入返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int DeleteChild(BiTNode\*T, Key e, int LR)

{

if(T==NULL)

{

printf("该活动树为空二叉树，");

return ERROR;

}

printf("请输入所要删除子树的双亲结点的关键字(根结点的双亲结点定义为表头结点)："); scanf("%d", &e); getchar();

if(e==0)

{

DestroyBiTree2(T->lchild);

T->lchild=NULL;

return OK;

}

BiTNode\*p = Value(T->lchild, e);

if (p == NULL)

{

printf("含有该关键字的结点不存在，");

return ERROR;

}

else

{

printf("以0代表删除左子树，1代表删除右子树，请输入LR的值："); scanf("%d", &LR); getchar();

BiTNode\*temp;

if (LR == 0)

{

temp = p->lchild;

p->lchild = NULL;

if (DestroyBiTree2(temp) == OK)

return OK;

}

else if (LR == 1)

{

temp = p->rchild;

p->rchild = NULL;

if (DestroyBiTree2(temp) == OK)

return OK;

}

else

{

printf("输入的LR值不合法，");

return ERROR;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：change

函数参数：二叉树的根结点

函数功能：通过树变二叉树的方式将要插入的子树转变为没有右子树的树

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void change(BiTNode\*T)

{

if(T==NULL)

return ;

change(T->lchild);

change(T->rchild);

if(T->lchild==NULL)

T->lchild=T->rchild;

else

T->lchild->rchild=T->rchild;

T->rchild=NULL;

return ;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：get\_nums\_bitree

函数参数：二叉树的根结点

函数功能：统计二叉树的结点数

返回值：返回二叉树的结点数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int get\_nums\_bitree(BiTNode \*T)

{

if (T==NULL)

return 0;

else

return get\_nums\_bitree(T->lchild) + get\_nums\_bitree(T->rchild) + 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：InsertChild

函数参数：二叉树的根结点，关键字e ，判断标志LR， 代表子树的字符串s

函数功能：插入子树

返回值：成功插入返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int InsertChild(BiTNode\*T, Key e, int LR, BiTNode\*p)

{

change(p);

if(e==0)

{

p->rchild=T->lchild;

T->lchild=p;

return OK;

}

BiTNode\*q=Value(T->lchild,e);

if(q==NULL)

{

printf("e值不合法，");

return ERROR;

}

if(LR==0)

{

p->rchild=q->lchild;

q->lchild=p;

}

else if(LR==1)

{

p->rchild=q->rchild;

q->rchild=p;

}

else

{

printf("LR值不合法，");

return ERROR;

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：RightSibling

函数参数：二叉树的根结点，关键字e

函数功能：寻找右兄弟结点

返回值：如果查找到返回结点指针，否则返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BiTNode \*RightSibling(BiTNode\*T, Key e)

{

BiTNode\*temp = Parent(T, e);

if (temp == NULL)

return NULL;

else

return temp->rchild;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：LeftSibling

函数参数：二叉树的根结点，关键字e

函数功能：寻找左兄弟结点

返回值：如果查找到返回结点指针，否则返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BiTNode \*LeftSibling(BiTNode\*T, Key e)

{

BiTNode\*temp = Parent(T,e);

if (temp == NULL)

return NULL;

else

return temp->lchild;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：RightChild

函数参数：二叉树的根结点，关键字e

函数功能：寻找右孩子结点

返回值：如果查找到返回结点指针，否则返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BiTNode \* RightChild(BiTNode\*T, Key e)

{

BiTNode \*temp = Value(T, e);

if (temp == NULL)

return NULL;

return temp->rchild;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：LeftChild

函数参数：二叉树的根结点，关键字e

函数功能：寻找左孩子结点

返回值：如果查找到返回结点指针，否则返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BiTNode \* LeftChild(BiTNode\*T, Key e)

{

BiTNode \*temp=Value(T, e);

if (temp == NULL)

return NULL;

return temp->lchild;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：Parent

函数参数：二叉树的根结点，关键字e

函数功能：寻找双亲结点

返回值：如果查找到就返回结点指针，否则返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BiTNode \* Parent(BiTNode \*T, Key e)

{

if ((T->lchild != NULL && e == T->lchild->data.i)

|| (T->rchild != NULL && e == T->rchild->data.i)) {

return T;

}

else {

BiTNode\*temp=NULL;

if(T->lchild != NULL)

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

if(temp==NULL&&T->rchild != NULL)

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

return temp;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：Assign

函数参数：二叉树的根结点，关键字e，数据c

函数功能：将关键字为e的结点的数据改为c

返回值：如果查找到结点就返回OK，否则返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Assign(BiTNode \*T, Key e, char c)

{

BiTNode \*temp;

if ((temp = Value(T, e) )== NULL)

return ERROR;

temp->data.c = c;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：Value

函数参数：二叉树的根结点，关键字e

函数功能：返回二叉树相应结点的数据域

返回值：如果存在就返回二叉树相应结点，否则返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BiTNode \* Value(BiTNode \*T, Key e)

{

if (T == NULL)

return NULL;

if (T->data.i == e)

return T;

BiTNode \* temp;

temp = Value(T->lchild, e);

if (temp == NULL)

{

temp= (BiTNode\*)malloc(sizeof(BiTNode));

temp = Value(T->rchild, e);

}

return temp;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：Root

函数参数：二叉树的表头结点

函数功能：返回二叉树根节点

返回值：二叉树根节点

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BiTNode \*Root(BiTNode \*T)

{

return T->lchild;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：BiTreeDepth

函数参数：二叉树的根结点的指针

函数功能：返回二叉树的深度

返回值：返回二叉树的深度

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int BiTreeDepth(BiTNode \*T)

{

int deep = 0;

if (T)

{

int leftdeep = BiTreeDepth(T->lchild);

int rightdeep = BiTreeDepth(T->rchild);

deep = leftdeep >= rightdeep ? leftdeep + 1 : rightdeep + 1;

}

return deep;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：BiTreeEmpty

函数参数：二叉树的表头结点

函数功能：判断二叉树是否为空

返回值：是返回OK，不是返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int BiTreeEmpty(BiTNode T)

{

if ((&T)->lchild == NULL)

return OK;

else

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：ClearBiTre

函数参数：指向二叉树的表头结点的指针F->HeadNode

函数功能：清空二叉树

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int ClearBiTree(BiTNode \*T)

{

if(T==NULL)

return OK;

if (DestroyBiTree2(T->lchild) == ERROR)

return ERROR;

T->lchild = NULL;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：PreCreateBiTree

函数参数：二叉树的根结点T的地址，输入的字符串s，用于定位字符串元素的i

函数功能：用递归生成二叉树

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int PreCreateBiTree(BiTNode \*\*T, char\*s, int \*i)

{

if (s[\*i] == '#') \*T = NULL;

else if (s[\*i] == '\n'|| s[\*i] == ' ') return ERROR;

else

{

if (!((\*T) = ((BiTNode\*)malloc(sizeof(BiTNode))))) return ERROR;

(\*T)->data.c = s[\*i];

if ((PreCreateBiTree(&((\*T)->lchild), s, &(++(\*i))) == ERROR))

return ERROR;

if ((PreCreateBiTree(&((\*T)->rchild), s, &(++(\*i)))) == ERROR)

return ERROR;

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：CreateBiTree

函数参数：字符串definition

函数功能：按照输入的definition序列生成二叉树

返回值：成功返回root，失败返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

BiTNode \* CreateBiTree(char\*definition)

{

if (definition[0] == '\n')

{

printf("未输入任意字符，");

return NULL;

}

else if(definition[0]=='#')

{

printf("该树为空树，");

return NULL;

}

int i = 0;

BiTNode \*root = (BiTNode\*)malloc(sizeof(BiTNode));

if (PreCreateBiTree(&root, definition, &i) == ERROR)

return NULL;

root->data.i = 1;

return root;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DestroyBiTree2

函数参数：指向Forest的头结点L和遍历指针F

函数功能：销毁森林结构中F对应结点，且将并使F指向NULL，

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int DestroyBiTree(Forest L,BiTree \*F)

{

int i;

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

{

if (strcmp(L.elem[i].name,F->name) == 0)

break;

}

if (i+1>L.length)

{

printf("当前没有活动树，");

return ERROR;

}

DestroyBiTree2(L.elem[i].HeadNode->lchild);

free(L.elem[i].HeadNode);

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

{

strcpy(L.elem[i].name, L.elem[i+1].name);

L.elem[i].HeadNode = L.elem[i+1].HeadNode;

}

L.elem[L.length].HeadNode=NULL;

L.length--;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DestroyBiTree2

函数参数：根结点

函数功能：销毁二叉树T

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int DestroyBiTree2(BiTNode \*T)

{

if (T == NULL) {

return OK;

}

DestroyBiTree2(T->lchild);

DestroyBiTree2(T->rchild);

free(T);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：InitBiTree

函数参数：Forest的头指针

函数功能：构造空二叉树T

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int InitBiTree(Forest \*L)

{

int i;

char name[20];

printf("请输入该树的名字（20个字符以内）：");scanf("%s",name);getchar();

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

{

if (strcmp(L->elem[i - 1].name, name) == 0) break;

}

if (i <= L->length)

{

printf("名称已存在，");

return ERROR;

}

BiTree \*newbase;

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

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

if (!newbase)

{

printf("分配内存失败，");

return ERROR;

}

L->listsize += LISTINCREMENT;

L->elem = newbase;

}

strcpy(L->elem[L->length].name, name);

L->elem[L->length].HeadNode = (BiTNode\*)malloc(sizeof(BiTNode));

L->elem[L->length].HeadNode->data.i= 0;

L->elem[L->length].HeadNode->data.c= 0;

L->elem[L->length].HeadNode->lchild = L->elem[L->length].HeadNode->rchild = NULL;

(L->length)++;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：show

函数参数：遍历指针F

函数功能：显示伪菜单界面

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void show(BiTree\* F)

{

system("cls");

printf("\n\n");

printf(" Menu for BiTree On Chain Structure \n");

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

printf(" 1. InitBiTree 13. LeftSibling\n");

printf(" 2. DestroyBiTree 14. RightSibling\n");

printf(" 3. CreateBiTree 15. InsertChild\n");

printf(" 4. ClearBiTree 16. DeleteChild\n");

printf(" 5. BiTreeEmpty 17. PreOrderTraverse\n");

printf(" 6. BiTreeDepth 18. InOrderTraverse\n");

printf(" 7. Root 19. PostOrderTraverse\n");

printf(" 8. Value 20. LevelOrderTraverse\n");

printf(" 9. Assign 21. SaveData\n");

printf(" 10. Parent 22. LoadData\n");

printf(" 11. LeftChild 23. ExBiTree\n");

printf(" 12. RightChild 0. Exit\n");

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

if (F == NULL)

printf("\n-----------当前没有活动树-----------\n");

else

printf("\n-----------当前对树 %s 进行操作-----------\n", F->name);

printf(" 请选择你的操作[0~23]:");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：updata

函数参数：根结点

函数功能：将结点所在位置的赋值给结点数据域的key

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void updata(BiTNode \*T)

{

if (T == NULL)return;

if (T->lchild)

{

T->lchild->data.i = 2 \* T->data.i;

updata(T->lchild);

}

if (T->rchild)

{

T->rchild->data.i = 2 \* T->data.i + 1;

updata(T->rchild);

}

}

# 附录D 基于邻接表图实现的源程序

//---------------------------------------基于邻接表的有向图相关操作-------------------------------//

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<string.h>

#define FALSE -1

#define OK 1

#define ERROR -2

#define TRUE 1

#define INFEASTABLE -1

#define OVERFLOW -2

#define MAX\_VERTICE\_NUM 20

typedef enum { DG, DN, UDG, UND }GraphKind; //有向图,有向网,无向图,无向网

typedef int status;

typedef char VertexType;

typedef struct ArcNode {

int adjvex; //该弧所指向的顶点的位置

struct ArcNode \*nextarc; //指向下一条弧的指针

}ArcNode;

typedef struct VNode{

VertexType data; //顶点信息

ArcNode \*firstarc; //第一个表结点的地址,指向第一条依附该顶点的弧的指针

}VNode;

typedef struct ALGraph {

VNode vertices[MAX\_VERTICE\_NUM];

int vexnum, arcnum; //图的当前顶点数和弧数

char name[20]; //图的名字

ALGraph \*next; //指向下一个图的指针

}ALGraph;

int visited[MAX\_VERTICE\_NUM];

//---------------------------------------队列的定义和基本操作，用于广度优先遍历图---------------------------------------//

struct LinkQueueNode {

VNode\*data;

struct LinkQueueNode \*next;

};

struct LKQueue {

LinkQueueNode\*front, \*rear;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：InitQueue

函数参数：队列LQ

函数功能：初始化一个队列

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void InitQueue(LKQueue \*LQ)

{

LinkQueueNode\*p;

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

LQ->front = p;

LQ->rear = p;

LQ->front->next = NULL;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：EmptyQueue

函数参数：队列LQ

函数功能：判断队列是否为空

返回值：若队列为空返回1，否则返回0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int EmptyQueue(LKQueue\*LQ)

{

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

return 1;

else

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：EnQueue

函数参数：队列LQ，结点T

函数功能：将T结点插入队列尾端

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void EnQueue(LKQueue\*LQ, VNode\*T)

{

LinkQueueNode \*p;

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

p->data = T;

p->next = NULL;

LQ->rear->next = p;

LQ->rear = p;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：OutQueue

函数参数：队列LQ

函数功能：将队列第一个元素出列

返回值：成功返回1，失败即队列已为空时返回0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int OutQueue(LKQueue\*LQ)

{

LinkQueueNode \*p;

if (EmptyQueue(LQ))

{

return 0;

}

else

{

p = LQ->front->next;

LQ->front->next = p->next;

if (p->next == NULL)

LQ->rear = LQ->front;

free(p);

return 1;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：GetHead

函数参数：队列LQ

函数功能：返回队列第一个元素

返回值：返回第一个元素，若队列为空则返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

VNode \*GetHead(LKQueue \*LQ)

{

LinkQueueNode \*p;

VNode \*T = NULL;

if (EmptyQueue(LQ))

return T;

else

{

p = LQ->front->next;

return p->data;

}

}

//---------------------------------------功能函数的声明---------------------------------------//

void show(ALGraph \*G);

ALGraph \*CreateGraph(ALGraph \*Head, char V[MAX\_VERTICE\_NUM],char VR[MAX\_VERTICE\_NUM][MAX\_VERTICE\_NUM]);

status DestroyGraph(ALGraph \*G);

status LocateVex(ALGraph\*G, VertexType e);

VertexType GetVex(ALGraph\*G, int i);

status PutVex(ALGraph\*G, int i, VertexType e);

status FirstAdjVex(ALGraph\*G, int i);

status NextAdjVex(ALGraph\* G, int i,int j);

status InsertVex(ALGraph\* G, VertexType e);

status DeleteVex(ALGraph\* G, int i);

status InsertArc(ALGraph\* G, int i, int j);

status DeleteArc(ALGraph\* G, int i, int j);

status DFSTraverse(ALGraph\* G);

void DFS(ALGraph\* G, int v);

status BFSTraverse(ALGraph\* G);

status SaveData(FILE \*fp, ALGraph \*G, char filename[30]);

status LoadData(FILE \*fp, ALGraph \*G, char filename[30]);

ALGraph\* ExGraph(ALGraph\*Head, char name[20]);

int main() {

int op = 1;

VertexType e = 0;

status a,i,j;

ALGraph \*Head,\*G,\*G1;

Head = (ALGraph\*)malloc(sizeof(ALGraph));

Head->arcnum = 0; Head->name[20] = {'\0'}; Head->vertices[0]= {'\0'}; Head->vexnum = 0;

Head->next = NULL;G = Head->next;

FILE\*fp=NULL; char filename[30];

char name[20];

char V[MAX\_VERTICE\_NUM], VR[MAX\_VERTICE\_NUM][MAX\_VERTICE\_NUM];

while (op) {

show(G);

scanf("%d", &op); getchar();

switch (op)

{

case 1:

printf("请输入该图的名字（20个字符以内）："); scanf("%s", name, 20); getchar();

G1 = Head->next;

while (G1)

{

if (strcmp(G1->name, name) == 0) break;

G1 = G1->next;

}

if (G1!= NULL)

{

printf("名称已存在，创建失败！");

getchar();

break;

}

G1= (ALGraph\*)malloc(sizeof(ALGraph));

G1->next = NULL;

strcpy(G1->name,name);

printf("请依次输入图的顶点数和边数(格式如x y且顶点数不超过20)："); scanf("%d %d", &((\*G1).vexnum), &((\*G1).arcnum)); getchar();

if ((\*G1).vexnum > MAX\_VERTICE\_NUM)

{

printf("顶点数超过规定的最大顶点数，创建图失败！");

getchar();break;

}

if ((\*G1).vexnum == 0)

{

if ((\*G1).arcnum == 0)

{

G1->vertices[0] = { '\0' };

G = Head;

while (G->next)

G = G->next;

G->next = G1;

G = G1;

printf("创建图成功！");

getchar(); break;

}

else

{

printf("顶点数为0而边数不为0，创建失败！");

getchar(); break;

}

}

printf("请依次输入在图中顶点的数据(一个数据为一个char类型的单字符)："); scanf("%s", V, 20); getchar();

j = 0;

for (i = 0; (\*G1).arcnum != 0&&i < (\*G1).vexnum; i++)

{

printf("请依次输入与顶点%c邻接的顶点的数据（若无输入0）：", V[i]); scanf("%s", VR[i], 20); getchar();

if (VR[i][0] == '0')

VR[i][0] = '\0';

else

j = j + strlen(VR[i]);

}

for (; i < MAX\_VERTICE\_NUM; i++)

\*VR[i] = {'\0'};

if (j != (\*G1).arcnum)

{

printf("边数与输入的边数不相同，创建失败！");

getchar();break;

}

if ((G1 = CreateGraph(G1,V,VR)) != NULL)

{

G = Head;

while (G->next)

G = G->next;

G->next = G1;

G = G1;

printf("创建图成功！");

}

else

printf("创建图失败！");

getchar();break;

case 2:

if (DestroyGraph(G) == OK)

{

printf("图销毁成功！\n");

G1 = Head;

for (; G1->next != G; G1 = G1->next);

G1->next = G1->next->next;

free(G);

G->next = NULL;

G = NULL;

}

else

printf("图销毁失败！\n");

getchar();break;

case 3:

printf("请输入所要查找的顶点的VertexType数据：");scanf("%c", &e); getchar();

if ((a = LocateVex(G, e)) != ERROR)

printf("顶点%c在图中的序号为：%d。", e, a);

else

printf("图中不存在该顶点！");

getchar();break;

case 4:

printf("请输入所要查找的顶点的序号："); scanf("%d", &i); getchar();

if ((e = GetVex(G, i)) != ERROR)

printf("序号为%d的顶点的值为为：%c。", i, e);

else

printf("图中不存在该顶点！");

getchar();break;

case 5:

printf("请输入所要修改的顶点的序号："); scanf("%d", &i); getchar();

printf("请输入修改后的数据："); scanf("%c", &e); getchar();

if ((PutVex(G,i,e)) != ERROR)

printf("修改成功！");

else

printf("修改失败！");

getchar();break;

case 6:

printf("请输入所要查找的顶点的序号："); scanf("%d", &i); getchar();

if ((a = FirstAdjVex(G, i)) == ERROR)

printf("图中不存在该顶点！");

else if (a != -1)

printf("序号为%d的顶点的第一邻接点的序号为：%d 值为：%c。", i, a, G->vertices[a].data);

else

printf("序号为%d的顶点无邻接点。", i);

getchar();break;

case 7:

printf("请输入所要查找的顶点的序号："); scanf("%d", &i); getchar();

printf("请输入该顶点的一邻接顶点的序号："); scanf("%d", &j); getchar();

if ((a = NextAdjVex(G, i, j)) == ERROR)

printf("图中不存在该顶点！");

else if (a == -1)

printf("序号为%d的顶点没有序号为%d的邻接点。", i, j);

else if(a==-3)

printf("序号为%d的顶点无下一临接点。", i);

else

printf("序号为%d的顶点的下一临界点的序号为：%d 值为：%c。", i, a, G->vertices[a].data);

getchar();break;

case 8:

printf("请输入所要插入的顶点的数据为："); scanf("%c", &e); getchar();

if ((InsertVex(G, e))== OK)

printf("插入成功！");

else

printf("插入失败！");

getchar();break;

case 9:

printf("请输入所要删除的顶点的序号为："); scanf("%d", &i); getchar();

if (DeleteVex(G, i) == OK)

printf("删除成功！");

else

printf("删除失败！");

getchar();break;

case 10:

printf("请输入插入弧的弧尾顶点序号和弧头顶点序号(格式如x y)："); scanf("%d %d", &i, &j); getchar();

if (InsertArc(G, i, j) == OK)

printf("插入成功！");

else

printf("插入失败！");

getchar();break;

case 11:

printf("请输入删除弧的弧尾顶点序号和弧头顶点序号(格式如x y)："); scanf("%d %d", &i, &j); getchar();

if (DeleteArc(G, i, j) == OK)

printf("删除成功！");

else

printf("删除失败！");

getchar();break;

case 12:

if (DFSTraverse(G) == OK)

printf("\n遍历成功！");

else

printf("遍历失败！");

getchar();break;

case 13:

if (BFSTraverse(G) == OK)

printf("\n遍历成功！");

else

printf("遍历失败！");

getchar();break;

case 14:

if (SaveData(fp, G, filename) == OK)

printf("数据写入文件成功！");

else

printf("数据写入文件失败！");

getchar();break;

case 15:

if (LoadData(fp, G, filename) == OK)

printf("数据加载成功！");

else

printf("数据加载失败！");

getchar();break;

case 16:

if (G == NULL)

{

printf("当前没有图，切换失败！");

getchar(); break;

}

G1 = Head->next;

while (G1)

{

printf("图的名字：%s\n其广度优先遍历序列为：", G1->name);

BFSTraverse(G1);

printf("\n\n");

G1 = G1->next;

}

printf("请输入切换后的图的名字："); scanf("%s", name,20); getchar();

G1 = ExGraph(Head, name);

if (G1 == NULL)

printf("输入的名字不合法，切换失败！");

else

{

G = G1;

printf("切换成功，当前活动树为%s。",G->name);

}

getchar();break;

case 0:

printf("欢迎下次再使用本系统！\n");

getchar();break;

default:

{

printf("功能选择是无效的输入！\n");

getchar();

}

}

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：LoadData

函数参数：文件指针fp，活动图指针，文件名字filename

函数功能：将文件中的信息加载到活动图中

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status LoadData(FILE \*fp, ALGraph\*G, char filename[30])

{

if (G == NULL)

{

printf("当前无活动图，");

return ERROR;

}

printf("请输入要加载数据的文件名: "); scanf("%s", filename, 30); getchar();

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

{

printf("文件打开失败！");

return ERROR;

}

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

{

ArcNode \*p, \*q;

p = (\*G).vertices[i].firstarc;

while (p) {

q = p->nextarc;

free(p);

p = q;

}

}

(\*G).vexnum = 0; (\*G).arcnum = 0;

(\*G).vertices[MAX\_VERTICE\_NUM] = { '\0' };

char V[MAX\_VERTICE\_NUM], VR[MAX\_VERTICE\_NUM][MAX\_VERTICE\_NUM];

fscanf(fp, "%d %d", &((\*G).vexnum), &((\*G).arcnum));

fscanf(fp, "%s", V);

int i;

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

fscanf(fp, "%s", VR[i]);

for (;i < MAX\_VERTICE\_NUM; i++)

\*VR[i] = {'\0'};

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

for(int j=0;j<MAX\_VERTICE\_NUM;j++)

{

if(VR[i][j]=='0')

{

VR[i][j]='\0';

break;

}

}

G=CreateGraph(G,V,VR);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：SaveData

函数参数：文件指针fp，活动图结点，文件名字filename

函数功能：保存活动图的信息

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status SaveData(FILE \*fp, ALGraph \*G, char filename[30])

{

if (G == NULL)

{

printf("当前无活动图，");

return ERROR;

}

printf("请输入要写入的文件名: "); scanf("%s", filename,30); getchar();

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

{

printf("文件打开失败！");

return ERROR;

}

fprintf(fp,"%d %d\n",(\*G).vexnum,(\*G).arcnum);

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

fprintf(fp, "%c", GetVex(G,i));

fprintf(fp, "\n");

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

{

ArcNode \*p = (\*G).vertices[i].firstarc;

if(p==NULL) fprintf(fp,"0");

while (p)

{

fprintf(fp, "%c", GetVex(G, p->adjvex));

p = p->nextarc;

}

fprintf(fp, "\n");

}

fclose(fp);

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：ExGraph

函数参数：多图头指针，图的名字name

函数功能：切换活动图

返回值：成功返回名字为name的图的结构，失败返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ALGraph\* ExGraph(ALGraph\*Head, char name[20])

{

ALGraph\*L = Head->next;

while (L)

{

if (strcmp(L->name, name) == 0)

break;

L = L->next;

}

return L;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：BFSTraverse

函数参数：当前活动图G

函数功能：广度优先搜索图

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status BFSTraverse(ALGraph\* G)

{

if (G == NULL)

{

printf("当前无活动图，");

return ERROR;

}

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

visited[v] = 0;

LKQueue \*Q=(LKQueue \*)malloc(sizeof(LKQueue));

InitQueue(Q);

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

{

if (!visited[v])

{

printf("%c", (\*G).vertices[v].data);

visited[v] = 1;

EnQueue(Q, &(\*G).vertices[v]);

while (!EmptyQueue(Q))

{

int u = LocateVex(G, GetHead(Q)->data);

OutQueue(Q);

for (int w = FirstAdjVex(G, u); w >= 0; w = NextAdjVex(G, u, w))

if (!visited[w])

{

printf("%c", (\*G).vertices[w].data);

visited[w] = 1;

EnQueue(Q, &(\*G).vertices[w]);

}

}

}

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DFSTraverse

函数参数：当前活动图G,要序号v

函数功能：深度优先搜索图

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void DFS(ALGraph\* G, int v)

{

ArcNode \*p;

printf("%c", (\*G).vertices[v].data);

visited[v] = 1;

p = (\*G).vertices[v].firstarc;

while (p)

{

if (!visited[p->adjvex])

DFS(G, p->adjvex);

p = p->nextarc;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DFSTraverse

函数参数：当前活动图G

函数功能：深度优先搜索图

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status DFSTraverse(ALGraph\* G)

{

if (G == NULL)

{

printf("当前无活动图，");

return ERROR;

}

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

visited[v] = 0;

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

{

if (!visited[v])

DFS(G, v);

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DeleteArc

函数参数：当前活动图G、删除弧的弧尾i和弧头j

函数功能：删除弧

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status DeleteArc(ALGraph\* G, int i, int j)

{

VertexType a = GetVex(G, i);

if (a == ERROR)

return ERROR;

a = GetVex(G, j);

if (a == ERROR)

return ERROR;

ArcNode\*p=(\*G).vertices[i].firstarc;

if (p == NULL)

{

printf("弧头顶点序号为%d且弧尾顶点序号为%d的弧不存在，", i, j);

return ERROR;

}

if (p->adjvex == j)

{

(\*G).vertices[i].firstarc = p->nextarc;

free(p);

p = (\*G).vertices[i].firstarc;

(\*G).arcnum--;

return OK;

}

status flag = 0;

ArcNode\*q;

while (p)

{

q = p;

p = p->nextarc;

if (p->adjvex == j)

{

q->nextarc = p->nextarc;

free(p);

flag = 1;

break;

}

}

if (flag == 0)

{

printf("弧头顶点序号为%d且弧尾顶点序号为%d的弧不存在，",i,j);

return ERROR;

}

(\*G).arcnum--;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：InsertArc

函数参数：当前活动图G、插入弧的弧尾i和弧头j

函数功能：插入弧

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status InsertArc(ALGraph\* G, int i, int j)

{

VertexType a = GetVex(G, i);

if (a == ERROR)

return ERROR;

a = GetVex(G, j);

if (a == ERROR)

return ERROR;

ArcNode\*p,\*q;

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

p->adjvex = j;

p->nextarc= NULL;

q = (\*G).vertices[i].firstarc;

if(q==NULL)

(\*G).vertices[i].firstarc=p;

else

{

while (q->nextarc)

q = q->nextarc;

q->nextarc = p;

}

(\*G).arcnum++;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DeleteVex

函数参数：当前活动图G、所要删除顶点的序号i

函数功能：删除顶点

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status DeleteVex(ALGraph\* G, int i)

{

VertexType a = GetVex(G, i);

if (a == ERROR)

return ERROR;

ArcNode \*p, \*q=NULL;

p = (\*G).vertices[i].firstarc;

while (p)

{

q = p;

p = p->nextarc;

free(q);

(\*G).arcnum--;

}

int j;

for (j = i; j<(\*G).vexnum; j++)

(\*G).vertices[j] = (\*G).vertices[j + 1];

(\*G).vexnum--;

for (j = 0; j<(\*G).vexnum; j++)

{

p = (\*G).vertices[j].firstarc;

while (p)

{

if (p->adjvex == i)

{

if (p == (\*G).vertices[j].firstarc)

{

(\*G).vertices[j].firstarc = p->nextarc;

free(p);

p = (\*G).vertices[j].firstarc;

}

else {

q->nextarc = p->nextarc;

free(p);

p = q->nextarc;

}

(\*G).arcnum--;

}

else

{

if (p->adjvex>i)

p->adjvex--;

q = p;

p = p->nextarc;

}

}

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：InsertVex

函数参数：当前活动图G、所要插入顶点的数据e

函数功能：插入顶点

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status InsertVex(ALGraph\* G, VertexType e)

{

if (G == NULL)

{

printf("当前无活动图，");

return ERROR;

}

if (G->vexnum == MAX\_VERTICE\_NUM)

{

printf("当前图的顶点数已达到规定的最大值，");

return ERROR;

}

int j=G->vexnum;

G->vertices[j].data= e;

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

G->vexnum++;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：NextAdjVex

函数参数：当前活动图G、所要查找的顶点的序号i及其邻接结点j

函数功能：输出下一邻接顶点

返回值：成功返回顶点位置，失败返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status NextAdjVex(ALGraph\* G, int i, int j)

{

VertexType a = GetVex(G, i);

if (a == ERROR)

return ERROR;

ArcNode \*p;

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

while (p != NULL&&p->adjvex != j)

p = p->nextarc;

if (p == NULL)

return -1;

if (p->nextarc == NULL)

return -3;

return p->nextarc->adjvex;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：FirstAdjVex

函数参数：当前活动图G、所要查找的顶点的序号i

函数功能：输出第一邻接顶点

返回值：成功返回顶点位置，失败返回NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status FirstAdjVex(ALGraph\* G, int i)

{

VertexType a = GetVex(G, i);

if (a == ERROR)

return ERROR;

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

return -1;

return (G->vertices[i]).firstarc->adjvex;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：PutVex

函数参数：当前活动图G、所要修改的顶点的序号i以及修改后的数据e

函数功能：节点赋值

返回值：成功返回OK，失败返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status PutVex(ALGraph\* G, int i, VertexType e)

{

VertexType a = GetVex(G, i);

if (a == ERROR)

return ERROR;

G->vertices[i].data = e;

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：GetVex

函数参数：当前活动图G，以及所要查找的顶点的序号i

函数功能：查找顶点值

返回值：找到则返回顶点值，未找到返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

VertexType GetVex(ALGraph\*G, int i)

{

if (G == NULL)

{

printf("当前无活动图，");

return ERROR;

}

int j = G->vexnum;

if (j == 0)

{

printf("该图顶点集为空，");

return ERROR;

}

if(i>j-1||i<0)

{

printf("输入的序号不合法，");

return ERROR;

}

return G->vertices[i].data;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：LocateVex

函数参数：当前活动图G，以及所要查找的顶点的VertexType数据e

函数功能：查找顶点

返回值：找到则返回位置i，未找到返回ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status LocateVex(ALGraph\*G, VertexType e)

{

if(G == NULL)

{

printf("当前无活动图，");

return ERROR;

}

int i;

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

{

if (e == (\*G).vertices[i].data)

return i;

}

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：DestroyGraph

函数参数：遍历指针G

函数功能：销毁图

返回值：成功为OK，失败为ERROR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

status DestroyGraph(ALGraph \*G)

{

if (G == NULL)

{

printf("当前无活动图，");

return ERROR;

}

int i;

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

{

ArcNode \*p, \*q;

p = (\*G).vertices[i].firstarc;

while (p) {

q = p->nextarc;

free(p);

p = q;

}

}

(\*G).vexnum = 0;

(\*G).arcnum = 0;

(\*G).name[20] = {'\0'};

(\*G).vertices[MAX\_VERTICE\_NUM] = { '\0' };

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：CreateCraph

函数参数：多图的头指针Head

函数功能：创建图

返回值：当前图的结点

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ALGraph \*CreateGraph(ALGraph \*G, char V[MAX\_VERTICE\_NUM], char VR[MAX\_VERTICE\_NUM][MAX\_VERTICE\_NUM])

{

int i, j = 0;

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

{

(\*G).vertices[i].data = V[i];

(\*G).vertices[i].firstarc = NULL;

}

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

{

for (j = 0; VR[i][j] != '\0'; j++)

{

int a = LocateVex(G, VR[i][j]);

if (a == ERROR)

{

printf("数据为%c的顶点在G中不存在，", VR[i][j]);

return NULL;

}

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

p->adjvex = a; p->nextarc = NULL;

ArcNode\*q = (\*G).vertices[i].firstarc;

if (q == NULL)

(\*G).vertices[i].firstarc = p;

else

{

while (q->nextarc) q = q->nextarc;

q->nextarc = p;

}

}

}

for (; i < MAX\_VERTICE\_NUM; i++)

{

(\*G).vertices[i].data = 0;

(\*G).vertices[i].firstarc = NULL;

}

return G;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名称：show

函数参数：遍历指针G

函数功能：显示伪菜单界面

返回值：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void show(ALGraph\* G)

{

system("cls");

printf("\n\n");

printf("------------------Menu for Graph------------------\n\n");

printf("\t1. CreateGraph\t\t9. DeleteVex\n");

printf("\t2. DestroyGraph\t\t10. InsertArc\n");

printf("\t3. LocateVex\t\t11. DeleteArc\n");

printf("\t4. GetVex\t\t12. DfsTraverse\n");

printf("\t5. PutVex\t\t13. BFSTraverse\n");

printf("\t6. FirstAdjVex\t\t14. SaveData\n");

printf("\t7. NextAdjVex\t\t15. LoadData\n");

printf("\t8. InsertVex\t\t16. ExGraph\n");

printf("\t0. Exit\n\n");

if (G == NULL)

printf("-----------------当前没有活动图-----------------\n");

else

{

printf("\n--------------当前对图 %s 进行操作--------------\n\n", G->name);

for (int i = 1; i <= G->vexnum; i++)

{

printf("位置：%d 数据：%c\t", i-1, G->vertices[i-1].data);

if (i % 4 == 0)printf("\n");

if(i==G->vexnum&&i%4!=0)printf("\n\n");

}

}

printf(" 请选择你的操作[0~16]:");

}