

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术201806**

**学 号： U201814659**

**姓 名： 张鹏**

**指导教师： 袁凌**

**报告日期： 2019年 11月 05 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 1](#_Toc24293243)

[1.1 问题描述 1](#_Toc24293244)

[1.2 系统设计 1](#_Toc24293246)

[1.3 系统实现 10](#_Toc24293249)

[1.4 系统测试 11](#_Toc24293250)

[1.5 实验小结 22](#_Toc24293251)

[2 基于链式存储结构的线性表实现 23](#_Toc24293252)

[2.1 问题描述 23](#_Toc24293253)

[2.2 系统设计 23](#_Toc24293255)

[2.3 系统实现 30](#_Toc24293258)

[2.4 系统测试 30](#_Toc24293259)

[2.5 实验小结 42](#_Toc24293260)

[3 基于二叉链表的二叉树实现 43](#_Toc24293261)

[3.1 问题描述 43](#_Toc24293262)

[3.2 系统设计 43](#_Toc24293263)

[3.3 系统实现 43](#_Toc24293264)

[3.4 系统测试 43](#_Toc24293265)

[3.5 实验小结 43](#_Toc24293266)

[4 基于邻接表的图实现 44](#_Toc24293267)

[4.1 问题描述 44](#_Toc24293268)

[4.2 系统设计 44](#_Toc24293269)

[4.3 系统实现 44](#_Toc24293270)

[4.4 系统测试 44](#_Toc24293271)

[4.5 实验小结 44](#_Toc24293272)

[参考文献 45](#_Toc24293273)

[附录A 基于顺序存储结构线性表实现的源程序 46](#_Toc24293274)

[附录B 基于链式存储结构线性表实现的源程序 63](#_Toc24293275)

[附录C 基于二叉链表二叉树实现的源程序 79](#_Toc24293276)

[附录D 基于邻接表图实现的源程序 81](#_Toc24293277)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

采用顺序表作线性表的物理存储结构，也就是线性表中的相邻元素的物理地址也是相邻的，完成线性表的基本操作。使用ElemType抽象为存储的元素数据类型。

构造一个具有菜单的功能演示系统，对于用户输入的各种操作给予相应的回答，并且能够给出适当的操作提示。

此外，还需要实现对于多个表的管理，以及从文件中读表，保存当前表到文件。

## 1.1.1 需要实现的功能

1. 线性表的基本运算：包括线性表的初始化、销毁、清空，以及和与表相关的查询操作，元素的查找、插入、删除等操作。

2. 文件读写：需要能够从文件加载表的数据以及将表数据保存到文件，这里使用二进制文件进行操作。

3. 多表管理：能够存储多个表，实现多表的删除、添加，以及能够在多个不同的表之间切换。

## 1.2 系统设计

本系统为实现基于顺序表的线性表的实现，系统总体由三大模块组成。具体如图1-1所示。

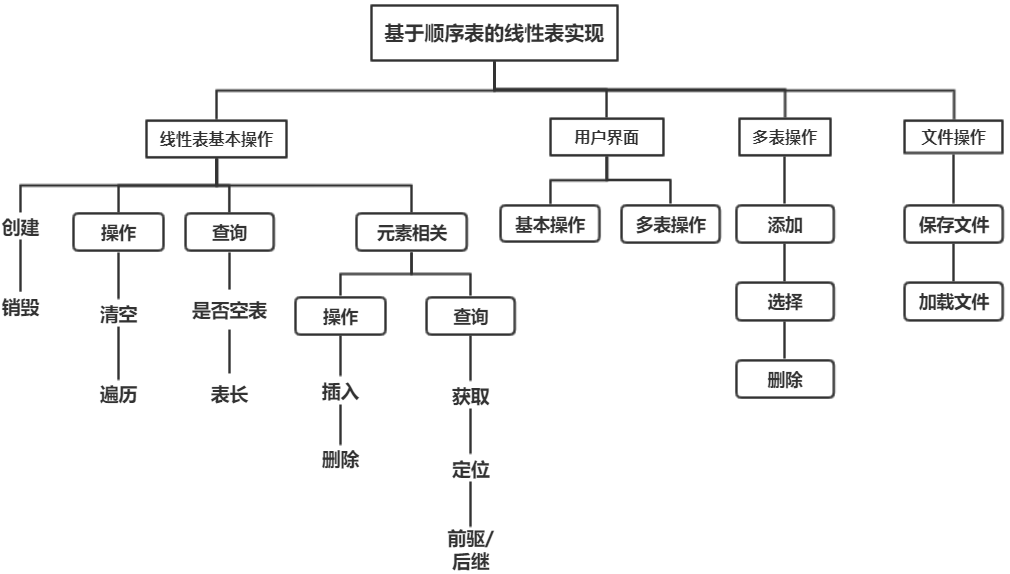


图1-1 系统总体设计

顺序表的结构体定义如下：

typedef struct { //顺序表（顺序结构）的定义

ElemType \*elem;

char name[41]; //表名字 在初始化表的时候写入

int length;

int listsize;

} SqList;

采用线性表的存储结构，用表名来区分不同的表，在结构中存下表长度以及容量。

系统菜单设计为数字选择样式，在选择不同的功能会出现相应的提示，如果输入的值不合法，系统能够告诉用户。

系统主要完成线性表的相关操作，除了基本的增删查改之外，还能对表中元素求前驱后继，以及定位元素。除此之外，系统额外实现了从文件中读取表单，以及保存表到文件的功能，更加方便用户进行读写。

一个表的使用往往不能满足要求，故添加了多表功能，对多个表进行管理。多表的定义如下：

struct table {

  SqList **\***list;

  struct table **\***next;

};

typedef struct table Table;

## 1.2.1 相关常量定义

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define NOTFOUND -3

typedef int status;

typedef int ElemType; //数据元素类型定义

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

## 1.2.2 函数设计

(1) status InitList(SqList \*\*L);

初始条件：线性表L 不存在

算法设计：函数接收指向L的指针的地址，为（\*L）和（\*L->elem）分配空间，如果分配失败返回OVERFLOW，否则初始化表长为0，容量为LIST\_INIT\_SIZE。并且读取name，返回OK。

操作结果：创建一个空的线性表，并且指针L指向该表。

(2) status DestroyList(SqList \*\*L);

初始条件：线性表L已存在。

算法设计：释放存储空间并销毁顺序表，将\*L置为NULL。

操作结果：销毁一个已经存在的顺序表L。

(3) status ClearList(SqList \*L);

初始条件：线性表L已存在。

算法设计：将表的长度设置为0

操作结果：使得表中元素无法访问，做到清空效果

(4) status ListEmpty(SqList \*L);

初始条件：线性表L已存在。

算法设计：判断L->length是否为0。

操作结果：空则返回TRUE，否则返回FALSE。

(5) int ListLength(SqList \*L);

初始条件：线性表L已存在。

算法设计：返回线性表L的表长，即返回L中的length元素。

操作结果：返回L中数据元素的个数。

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

初始条件：线性表已存在，1≤i≤ListLength(L)。

算法设计：先判断序号i是否在有效范围内（1≤i≤Length），根据序号找到线性表L中第i个元素的地址，并将地址赋值给指针e；

操作结果：指针e指向的元素即为L中第i个数据元素。

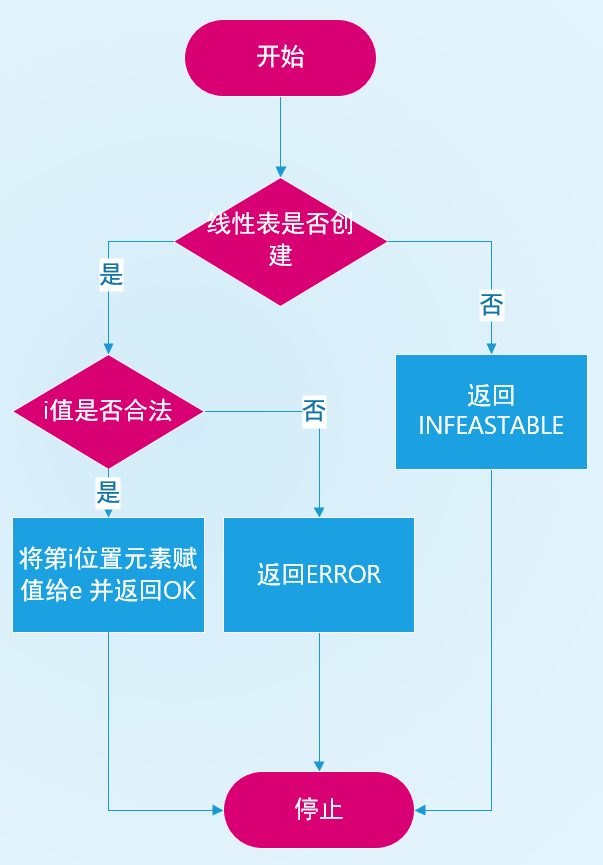


图1-2 GetElem流程图

(7) status LocateElem(SqList \*L, ElemType e);

初始条件：线性表L已存在。

算法设计：遍历线性表L，找到第一个和元素e的相等的元素。

操作结果：返回L中第1个与e相等的的数据元素的序号，若与之相等的数据元素不存在，则返回值为 NOTFOUND。

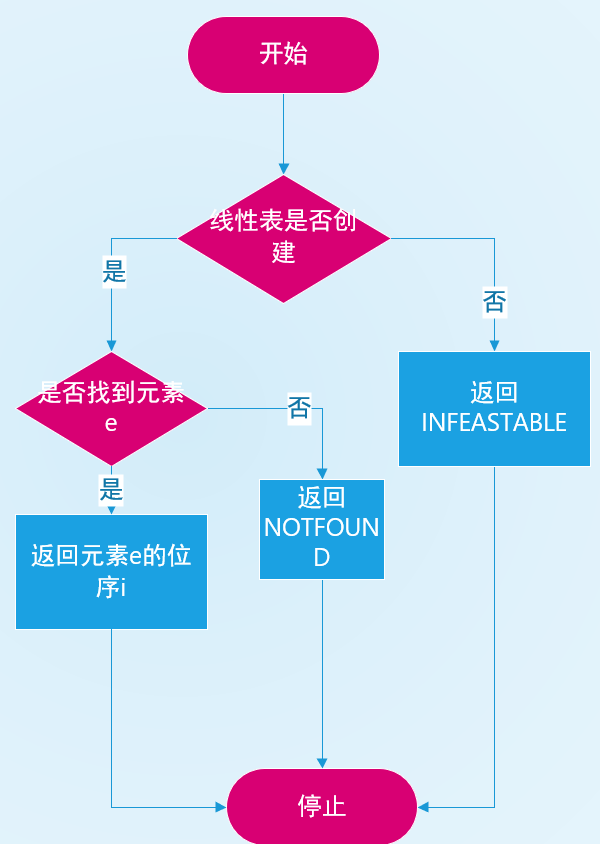


图1-3 LocateElem流程图

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

初始条件：线性表L已存在。

算法设计：遍历线性表L找到第一个和元素cur的相等的元素，如果其有前驱，将前驱的地址赋值给 pre\_e，函数返回OK；如果没有找到cur，则返回NOTFOUND,否则函数返回ERROR，pre\_e 无意义。

操作结果：若cur是L的数据元素，且不是第一个，则用pre\_e 返回它的前驱元素，否则操作失败，pre\_e 无定义。

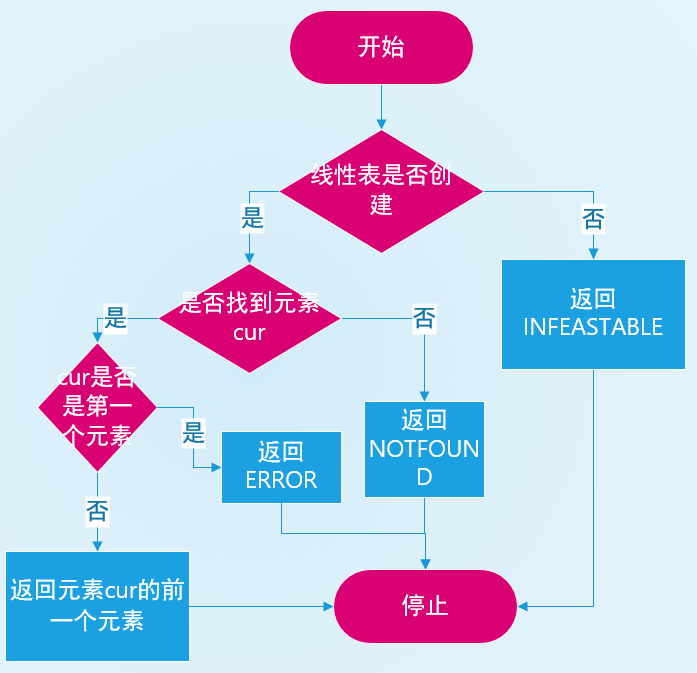


图1-4 PriorElem流程图

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

初始条件：线性表L已存在。

算法设计：遍历线性表L找到第一个和元素cur的相等的元素，如果其有后继，将后继的地址赋值给 next\_e，函数返回OK；如果没有找到cur，则返回NOTFOUND,否则函数返回FALSE，next\_e无意义。

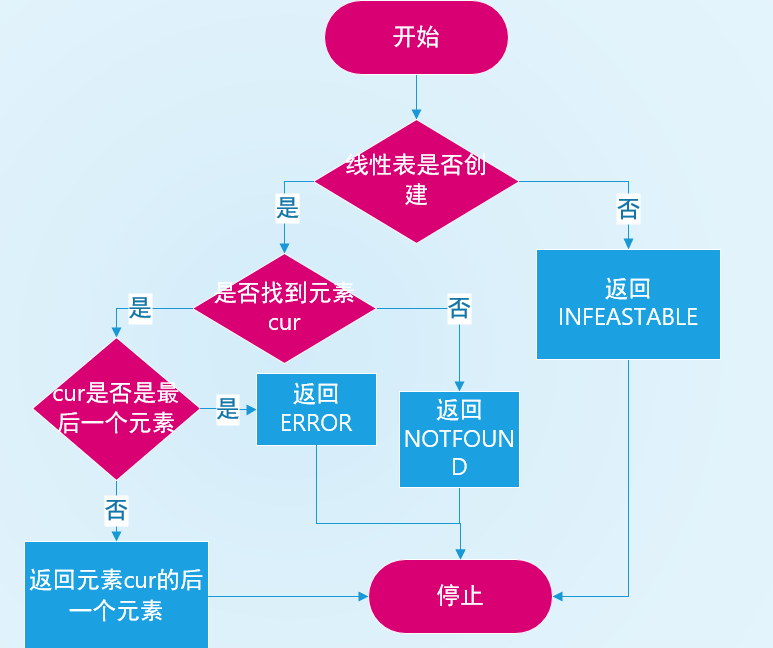
操作结果：若cur是L的数据元素，且不是最后一个，则用next\_e返回它的后继元素，否则操作失败，next\_e无定义。

图1-5 NextElem流程图

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

初始条件：线性表已存在，0≤i≤ListLength(L)。

算法设计：判断index是否在合理范围之内（1≤i≤ListLength(L)）。如果线性表已满，则重新分配存储空间。将第i个位置及其之后的元素向后面移动一个位置，将元素e放入第i个位置中，线性表长度加1。

操作结果：在L的第i个位置上插入新的数据元素e，L的长度加1。

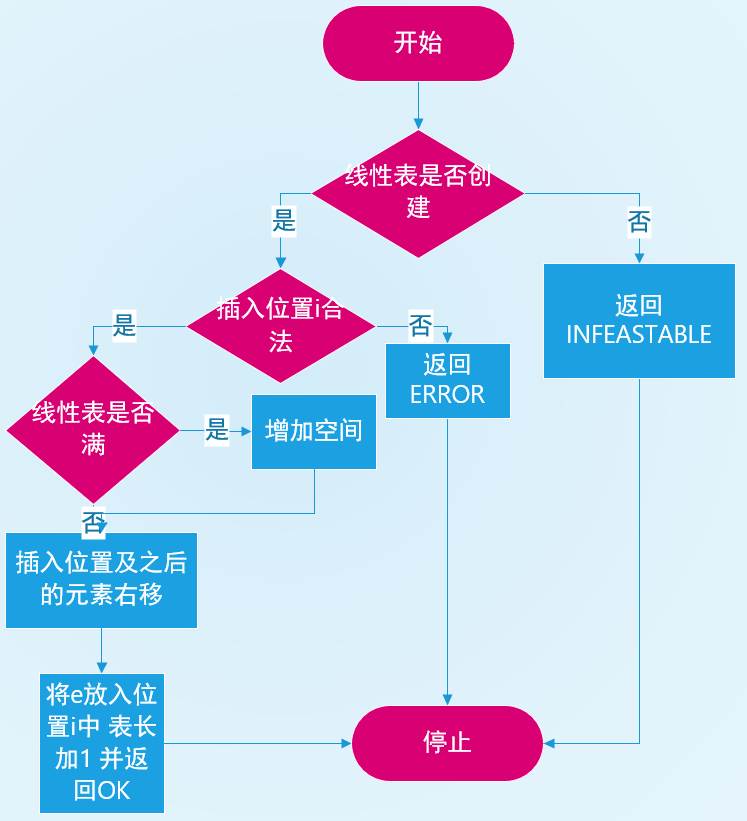


图1-6 ListInsert流程图

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

初始条件：线性表已存在，1≤i≤ListLength(L)。

算法设计：先判断i是否在合理范围之内（1≤i≤ListLength(L)），然后将第i个位置的值赋给指针引用变量e，之后第i个位置后面的全部元素前移一个位置, L的长度减1。

操作结果：删除L的位置为i的数据元素，用e返回其值，同时L的长度减1。

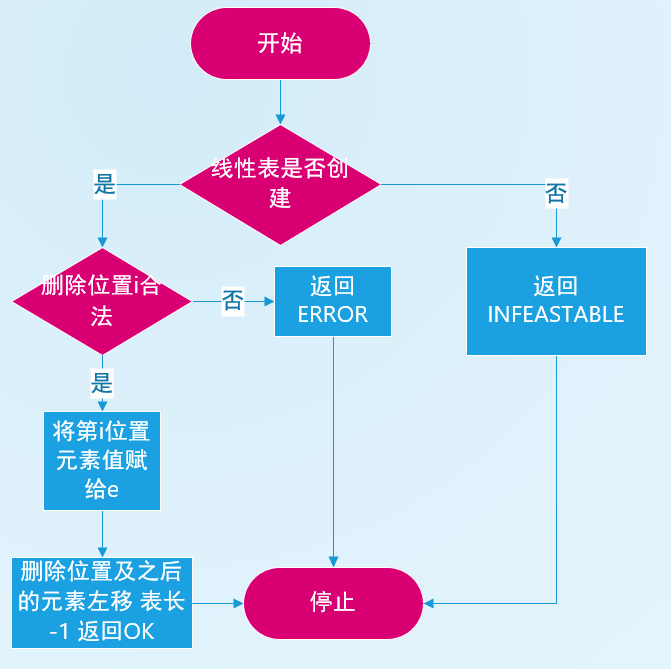


图1-7 ListDelete流程图

(12) status ListTraverse(SqList \*L);

初始条件：线性表L已存在。

算法设计：遍历表L中的每个元素，打印出每个元素的值。

操作结果：依次输出表L中的每个元素的值。

(13) status InputData(SqList \*L);

初始条件：线性表L已存在。

算法设计：在线性表的末尾添加值，每个元素以空格隔开，字符q作为结束。

操作结果：在线性表末尾添加输入的元素，线性表的长度相应增加。

(14) status SaveList(SqList \*L);

初始条件：线性表L已存在。

算法设计：将表中的元素以二进制的文件的形式保存。

操作结果：生成一个新的文件，保存当前表。

(15) status LoadList(SqList \*L);

初始条件：线性表L已存在。

算法设计：输入文件名称，如果文件不存在则返回ERROR，否则从文件中加载数据到当前表。

操作结果：读取已经保存的表。

(16) status AddList(Table \*head, SqList \*L);

初始条件：线性表L已存在。

算法设计：多表head开辟一块新的空间，保存当前L到多表中。

操作结果：保存当前表到多表。

(17) status ShowLists(Table \*head);

算法设计：从第一表开始，访问每个表的表长和表名字，依次按照顺序打印出信息。

操作结果：打印多表信息。

(18) status Choose(Table \*head, SqList \*\*L);

算法设计：首先调用ShowList(Table \*head)打印多表中的信息，根据输入的序号，找到存储对应线性表的指针，将其赋值给\*L。

操作结果：从多表中选取一个表加载。

(19) status Remove(Table \*head);

算法设计：首先调用ShowList(Table \*head)打印多表中的信息，根据输入的序号，找到其在多表中的位置，将存储空间归还。

操作结果：从多表中删除相应表。

## 1.3 系统实现

1. 初始化表：使用动态内存分配的方法，为线性表分配空间，并将线性表结构体中的值初始化。

2. 销毁表：将线性表中分配的空间free，然后将自身free，指向线性表的指针赋值为NULL。

3. 清空，判断空表，求表长：这几个操作都与线性表的长度有关，清空将线性表的长度置为0，判断空表只需判断长度是否为0，求表长则返回线性表长度。

4. 取元素：由于采用的顺序存储结构，所以可以做到直接取，只要计算出对应元素的地址即可。

5. 求位序：从头开始扫描线性表，直到碰到与要查找的值相同的元素，返回该元素的位序。

6. 求前驱和后继，与求位序类似，先求出要找的元素的位序，再由位序计算出前驱和后继，注意第一个元素没有前驱，最后一个元素没有后继。

7. 插入元素：如果线性表已满，先重新分配空间，由于采用的顺序存储结构，为了保持位序，需要将插入的位置及其之后的元素向后移动一个位置，如果表中有n个元素，则平均需要移动(n+1)/2次，时间复杂度为O(n)。

8. 删除元素：与插入类似，删除需要将删除位置之后的元素向前移动一个位置，如果表中有n个元素，则平均需要移动n/2次，时间复杂度为O(n)。

9. 遍历：使用循环操作从头到尾依次打印元素值。

## 1.4 系统测试

程序采用简易界面，如图所示，挑选ListEmpty, ListLength, GetElem, LocateElem, PriorElem, NextElem, ListInsert, ListDelete, ListTraverse这些重要功能进行测试。

测试用例分线性表未创建，线性表为空，线性表不为空三种情况。

假设表list1中已经存储{1，2，3，4，56，3，5，3，45};null 表实空表。

(1)ListEmpty测试

表1-1 ListEmpty测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| 空表 | 界面选4 | 线性表为空！ | 线性表为空！ |
| 非空表 | 界面选4 | 线性表不为空！ | 线性表不为空！ |
| 若表不存在 | 界面选4 | 线性表未创建 | 线性表未创建！ |

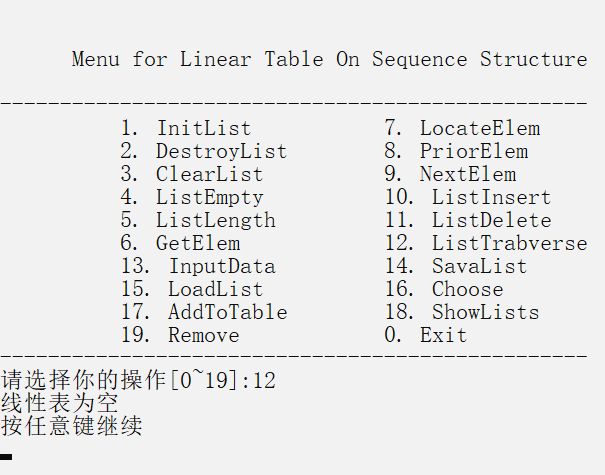


图1-8 判断空表（自身为空）

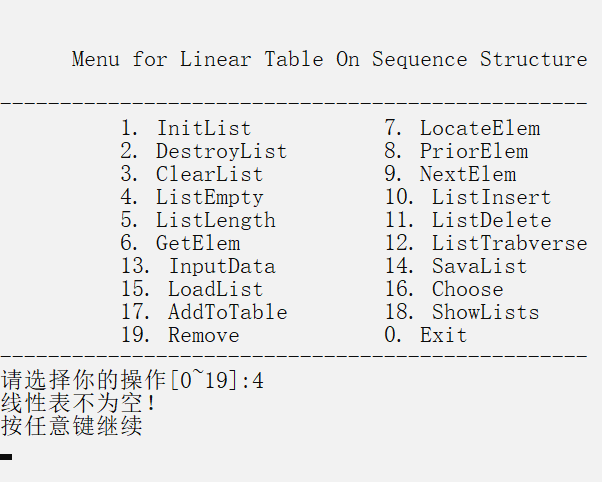


图1-9 判断空表（自身非空）

(2) ListLength测试

表1-2 ListLength测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选5 | 线性表表长为9 | 线性表表长为9 |
| null | 界面选5 | 线性表表长为0 | 线性表表长为0 |
| 若表不存在 | 界面选5 | 线性表不存在！ | 线性表不存在！ |

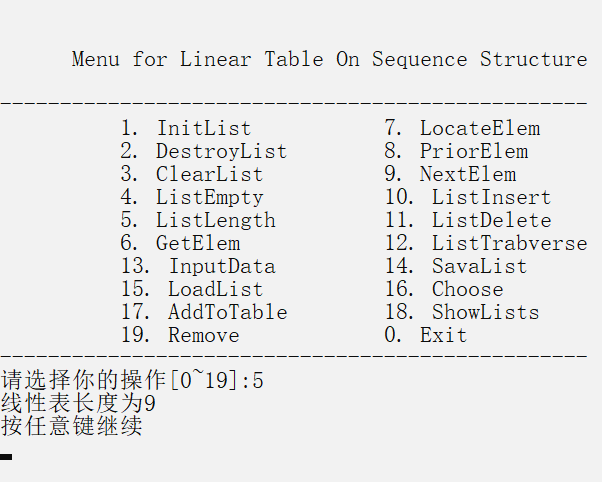


图1-10 求表长（用例List1）

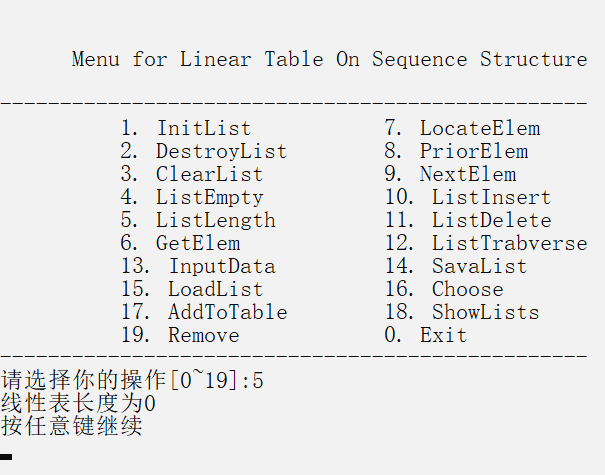


图1-11 求表长（用例null）

(3) GetElem测试

表1-3 GetElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选6  输入位置3 | 第3位元素是3 | 第3位元素是3 |
| List1 | 界面选6  输入位置12 | i值不合法 | i值不合法 |
| 若表不存在 | 界面选6 | 线性表未创建 | 线性表未创建 |

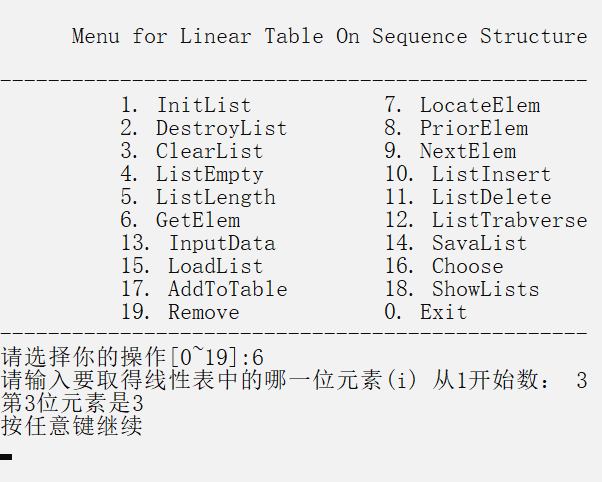


图1-12 取元素值（中间位置）

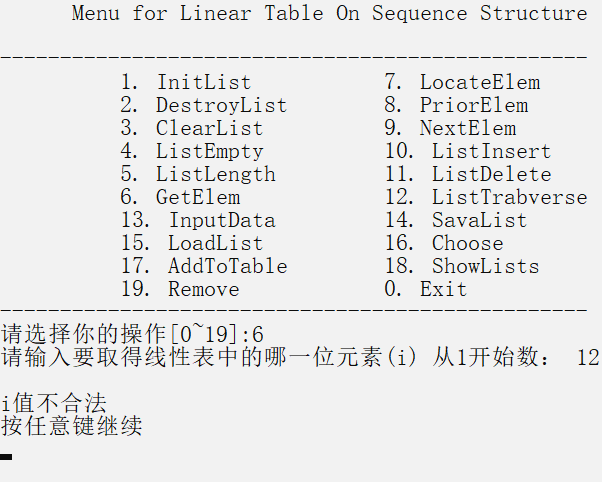


图1-13 取元素值（越界）

(4) LocateElem测试

表1-4 LocateElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选7  输入元素4 | 4元素位于第4个位置 | 4元素位于第4个位置 |
| List1 | 界面选7  输入元素20 | 表中没有这个元素 | 表中没有这个元素 |
| 若表不存在 | 界面选7 | 线性表未创建 | 线性表未创建 |

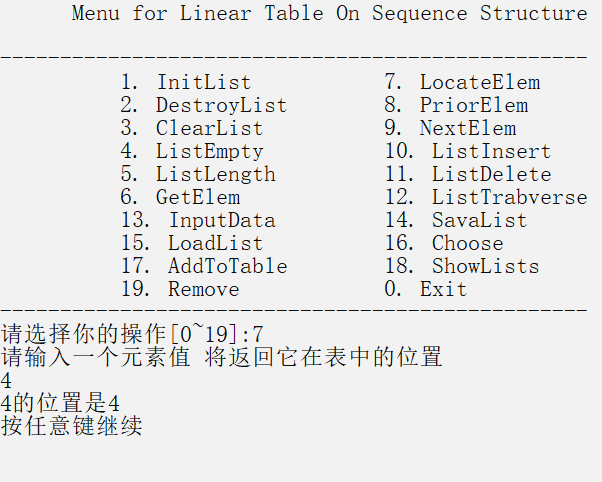


图1-14 返回位序（一般情况）



图1-15 返回位序（元素不存在）

(5) PriorElem测试

表1-5 PriorElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选8  输入元素2 | 2的前驱是1 | 2的前驱是1 |
| List1 | 界面选8  输入元素1 | 1为第一个元素,没有前驱 | 1为第一个元素,没有前驱 |
| List1 | 界面选8  输入元素13 | 表中不存在元素13 | 表中不存在元素13 |
| 若表不存在 | 界面选8 | 线性表未创建 | 线性表未创建 |

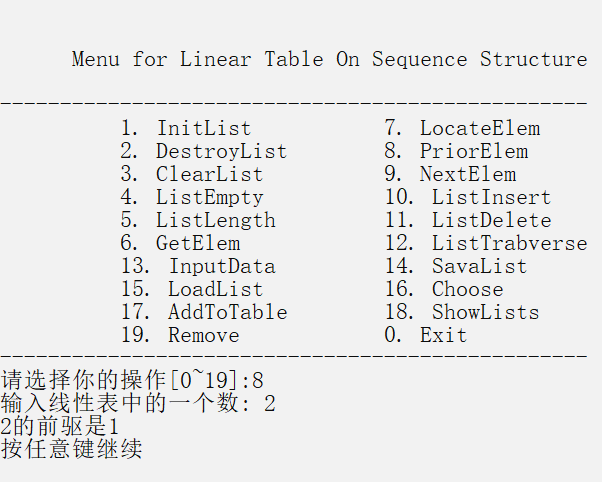


图1-16 求前驱（一般情况）

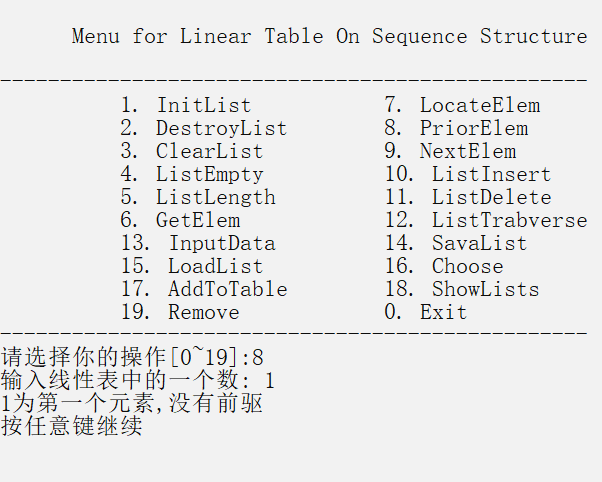


图1-17 求首元素前驱

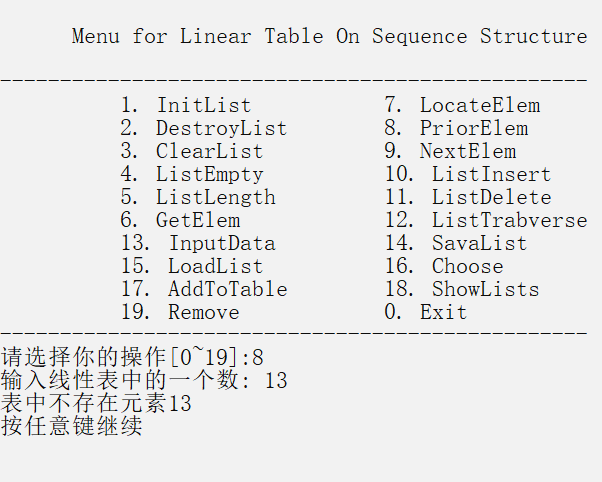


图1-18 求不存在元素前驱

(6) NextElem测试

表1-6 NextElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选9  输入元素4 | 4的后继是56 | 4的后继是56 |
| List1 | 界面选9  输入元素45 | 45为最后一个元素,没有后继 | 45为最后一个元素,没有后继 |
| List1 | 界面选9  输入元素选14 | 不存在元素14 | 不存在元素14 |
| 若表不存在 | 界面选9 | 线性表不存在！ | 线性表不存在！ |

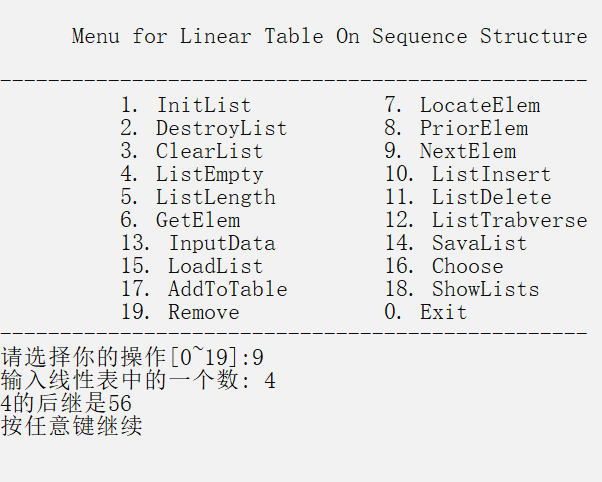


图1-20 求中间元素后继

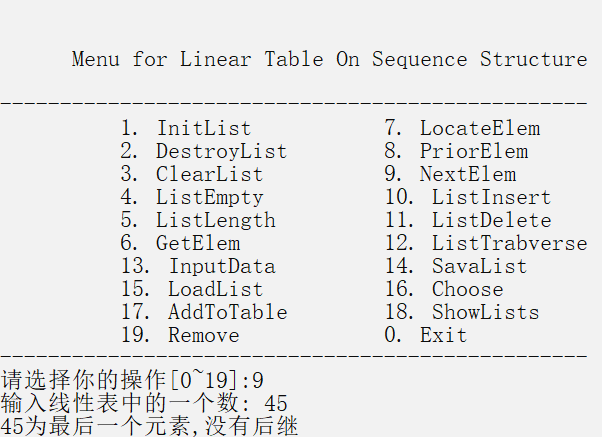


图1-21 求末尾元素后继

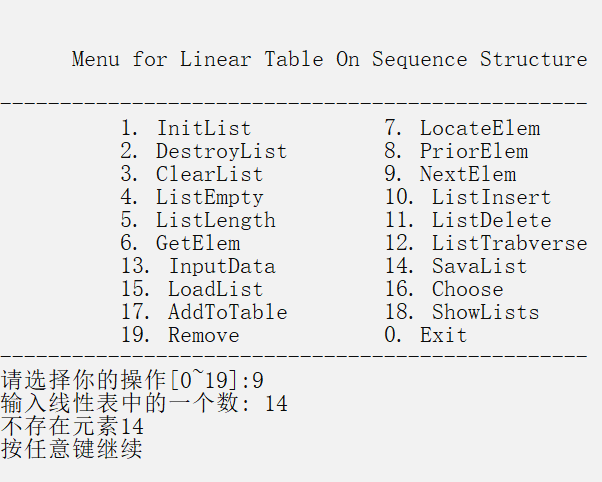


图1-21 求不存在元素后继

(7) ListInsert测试

表1-7 ListInsert测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选10  输入位置10  输入元素10 | 插入成功！ | 插入成功！ |
| List1 | 界面选10  插入位置20  输入元素2 | 插入位置不合法 | 插入位置不合法 |
| 若表不存在 | 界面选10 | 线性表未创建 | 线性表未创建 |



图1-22 插入元素



图1-23 插入位置不合法

(8) ListDelete测试

表1-8 ListDelete测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 选11、输入1 | 删除1成功！ | 删除1成功！ |
| List1 | 选11、输入12 | 删除位置不合法 | 删除位置不合法 |
| 若表不存在 | 选11 | 线性表不存在！ | 线性表不存在！ |

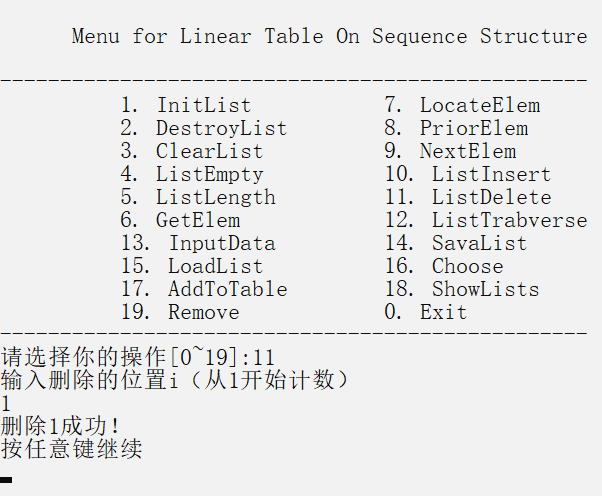


图1-24 删除元素

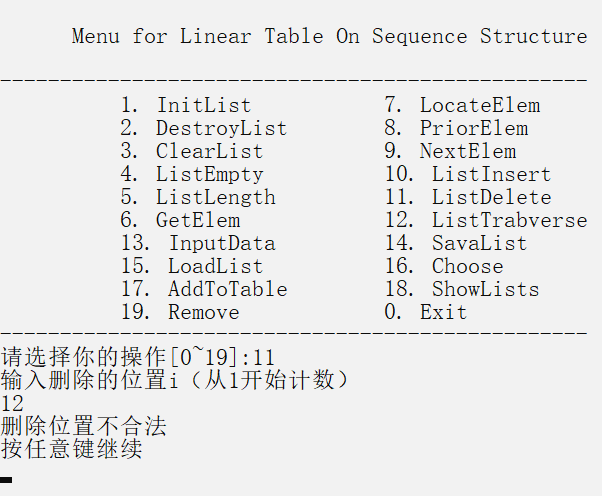


图1-25 删除位置不合法

(9) ListTraverse测试

表1-9 ListTraverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 测试结果 |
| List1 | 主界面选12 | 1 --> 2 --> 3 --> 4 --> 56 --> 3 --> 5 --> 3 --> 45 | 1 --> 2 --> 3 --> 4 --> 56 --> 3 --> 5 --> 3 --> 45 |
| null | 主界面选12 | 线性表为空 | 线性表为空 |
| 不存在的表 | 主界面选12 | 线性表未创建 | 线性表未创建 |

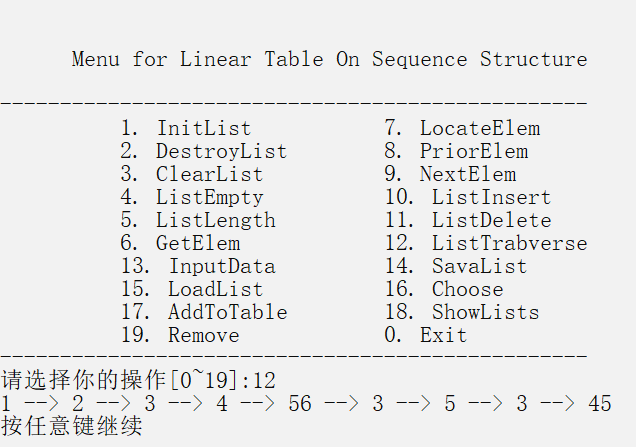


图1-26 遍历

## 1.5 实验小结

本次实验要求使用顺序表的存储方式实现线性表，总体来说难度不是很大，但是相比于之前，这是第一次写一个完整的程序，不仅要考虑到一般的正常情况，还要能够对各种可能发生的错误进行处理，更加强调整体性考验算法的健壮性。

一开始使用想使用数组存储多表，但是插入删除操作变得更加困难，几经修改之后，还是决定使用链表的形式进行多表的存储。虽然实现的功能都较为简单，但是实验过程中还是遇到了不少问题，当要求其他人对自己的程序进行测试时，总是能够发现自己未曾想到的问题，这让我知道，一个人考虑问题是很难做到面面俱到的。此外对代码的重新整理也是十分重要的，这种时候，写注释的习惯能够帮到自己很多。

最后，我由衷感谢老师、助教和同学在本次实验中对我的帮助。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

使用单链表作为线性表的物理存储结构，及线性表中相邻元素附加一个指针域，每个元素的指针域指向其后继节点，使用ElemType类型指代线性表中存储的元素类型。

构造一个具有菜单的功能演示系统，在主程序中完成对函数的调用，以及参数的准备，并且根据用户的输入基于不同的提示。

此外，还需要实现多个表的管理，以及从文件中加载，保存到文件的功能。

## 2.1.1 需要实现的功能

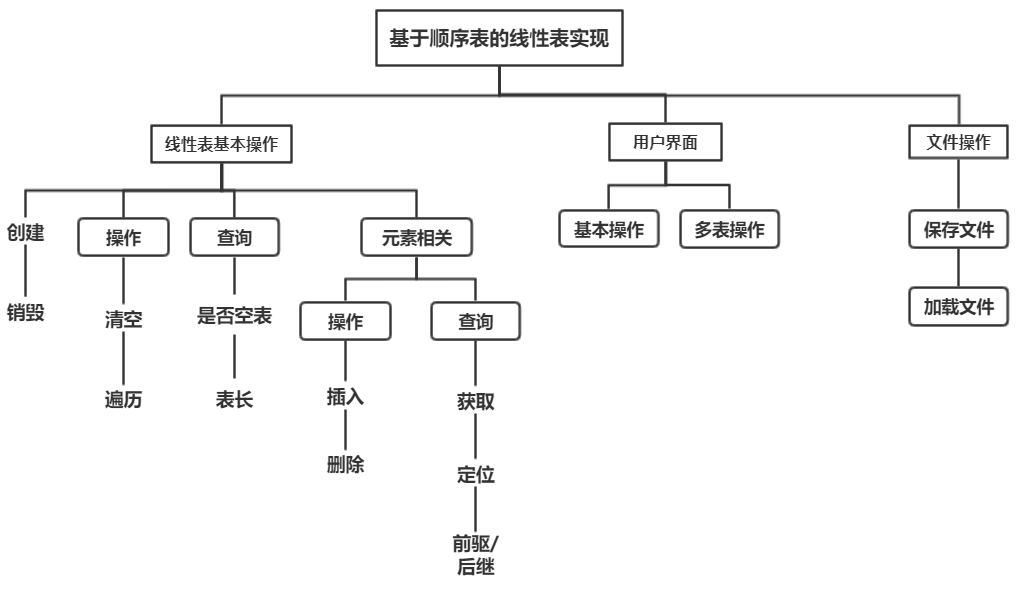
1. 线性表的基本运算：包括线性表的初始化、销毁、清空，以及和与表相关的查询操作，元素的查找、插入、删除等操作。

2. 文件读写：需要能够从文件加载表的数据以及将表数据保存到文件，这里使用二进制文件进行操作。

3. 多表管理：对每一个操作，可以选取对应的表ID

## 2.2 系统设计

本系统为实现基于单链表的线性表的实现，系统总体由三大模块组成。具体如图2-1所示。

图2-1 系统总体设计

单链表中每个节点定义如下：

typedef struct node

{

ElemType data;

struct node \*next;

} LNode;

typedef LNode \*PtrToLNode;

typedef PtrToLNode List;

使用带头节点的单链表实现对线性表的存储，头结点中的数据域储存的是线性表的长度，PtrToLNode为指向节点的指针，用List代表线性表，其只用来声明头节点。

系统菜单设计为数字选择样式，在选择不同的功能会出现相应的提示，如果输入的值不合法，系统能够告诉用户。

系统主要完成线性表的相关操作，除了基本的增删查改之外，还能对表中元素求前驱后继，以及定位元素。除此之外，系统额外实现了从文件中读取表单，以及保存表到文件的功能，更加方便用户进行读写，对于每一项基本的操作，系统会要求用户输入要操作的表的ID，从而简单的实现多个表的创建和管理。

## 2.2.1 相关常量定义

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR -3

#define INFEASTABLE -1

#define OVERFLOW -2

#define NOTFOUND -4

#define LISTNUM 3

typedef int status;

typedef int ElemType;

## 2.2.2 函数设计

status InitList(List \*L);

初始条件：线性表L不存在

算法设计：函数接收指向L的指针，为该指针使用动态内存分配分配空间，同时初始化数据域的值为0，指针域为NULL，该节点作为单链表的头节点使用，如果内存分配失败返回OVERFLOW。

操作结果：创建一个空的线性表，并且指针L指向该表。

status DestroyList(List \*L);

初始条件：线性表L已存在。

算法设计：从头节点开始，使用循环遍历每一个节点，并且对每一个节点进行free操作，归还内存空间，同时，使L指向NULL。

操作结果：销毁一个已经存在的顺序表L。

status ClearList(List L);

初始条件：线性表L已存在。

算法设计：将&(L->next)作为参数传递给DestroyList函数，使得头节点以后的节点全部销毁，只保留头节点，同时将头节点的data置为0。

操作结果：销毁除了头节点之外的所有节点。

status ListEmpty(List L);

初始条件：线性表L已存在。

算法设计：判断L->data是否为0。

操作结果：空则返回TRUE，否则返回FALSE。

status ListLength(List L);

初始条件：线性表L已存在。

算法设计：返回线性表L的表长，即返回L中的data元素。

操作结果：返回L中数据元素的个数。

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

初始条件：线性表已存在，1≤i≤ListLength(L)。

算法设计：先判断序号i是否在有效范围内（1≤i≤Length），使用一个循环结构找到第i个元素，用e将元素值带回。

操作结果：e将带回第i个数据元素。

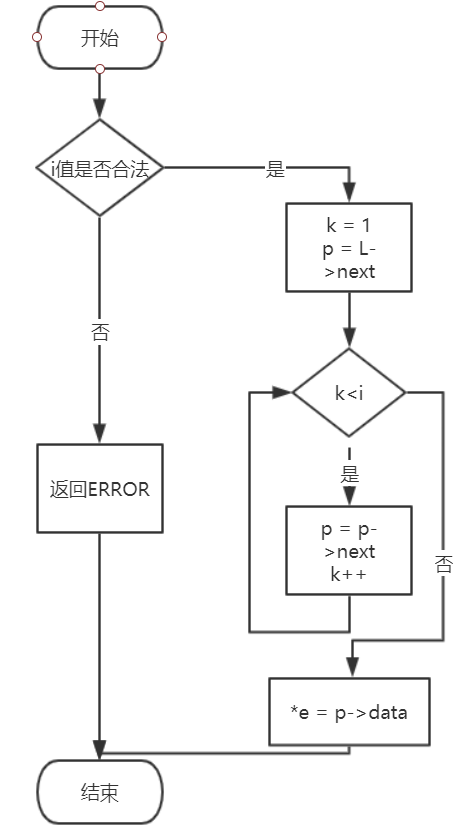


图2-2 GetElem流程图

int LocateElem(List L, int e);

初始条件：线性表L已存在。

算法设计：遍历线性表L，找到第一个和元素e的相等的元素。

操作结果：返回L中第1个与e相等的的数据元素的序号，若与之相等的数据元素不存在，则返回值为 NOTFOUND。

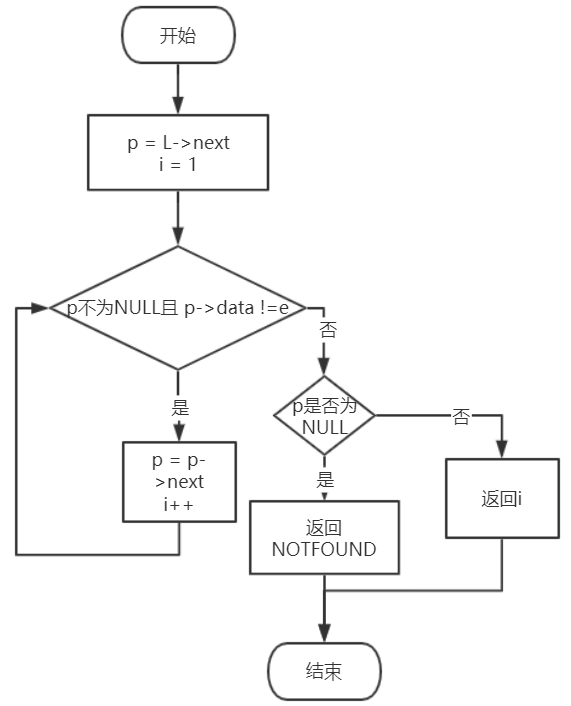


图2-3 LocateElem流程图

status PriorElem(List L, int cur\_e, int \*pre\_e);

初始条件：线性表L已存在。

算法设计：遍历线性表L找到p->next->data为cur的节点p，将p->data的值赋值给pre\_e，函数返回OK；如果没有找到cur，则返回NOTFOUND,否则函数返回ERROR，pre\_e 无意义。

操作结果：若cur是L的数据元素，且不是第一个，则用pre\_e 返回它的前驱元素，否则操作失败，pre\_e 无定义。

status NextElem(List L, int cur\_e, int \*next\_e);

初始条件：线性表L已存在。

遍历线性表L找到p->next->data为cur的节点p，将p->next->next->data的值赋值给next\_e，函数返回OK；如果没有找到cur，则返回NOTFOUND,否则函数返回ERROR，next\_e 无意义。

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

status ListInsert(List L, int i, int e);

初始条件：线性表已存在，0≤i≤ListLength(L)。

算法设计：判断index是否在合理范围之内（1≤i≤ListLength(L)+1）。使用一个循环，将p指向第i-1个节点的位置，在p之后插入新的节点，L->data加1。

操作结果：在L的第i个位置上插入新的数据元素e，L的长度加1。

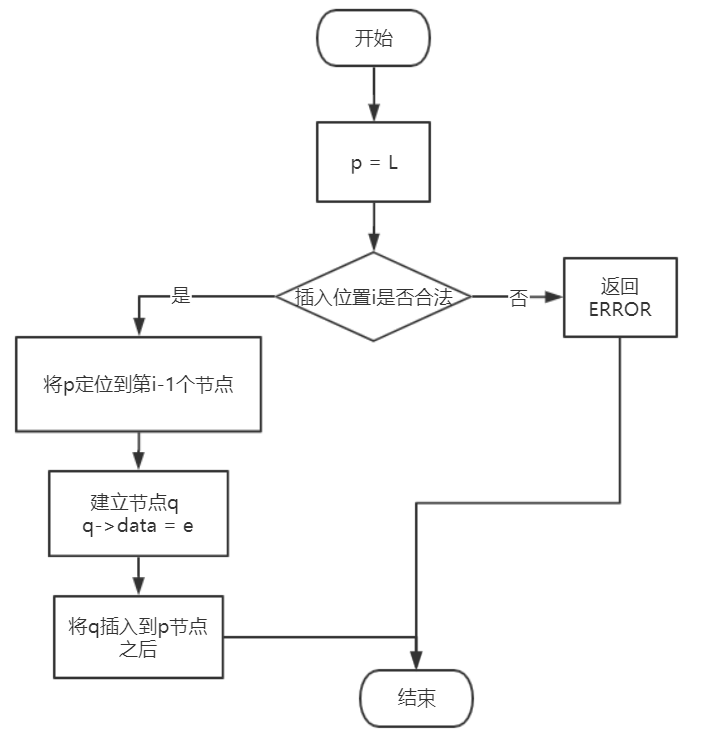


图2-4 ListInsert流程图

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

初始条件：线性表已存在，1≤i≤ListLength(L)。

算法设计：先判断i是否在合理范围之内（1≤i≤ListLength(L)），使用一个循环将p指向第i个位置之前的节点，用q指向p->next，将q->data赋值给e，然后在链表中将q节点删除，最后free(q)。

操作结果：删除L的位置为i的数据元素，用e返回其值，同时L的长度减1。

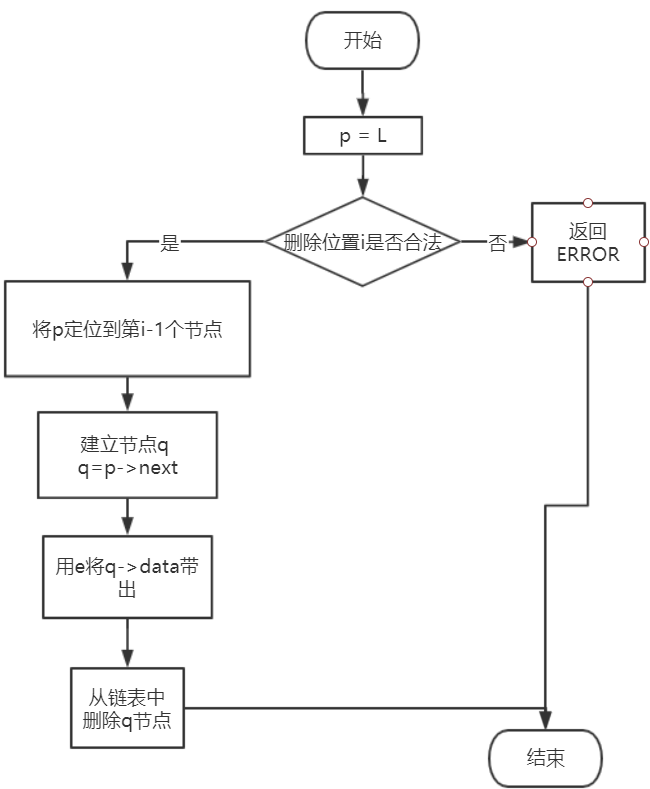


图2-5 ListDelete流程图

status ListTraverse(List L);

初始条件：线性表L已存在。

算法设计：遍历表L中的每个元素，打印出每个元素的值。

操作结果：依次输出表L中的每个元素的值。

status InputData(List L);

初始条件：线性表L已存在。

算法设计：在线性表的末尾添加值，每个元素以空格隔开，字符q作为结束。

操作结果：在线性表末尾添加输入的元素，线性表的长度相应增加。

status LoadList(List L);

初始条件：线性表L已存在。

算法设计：将表中的元素以二进制的文件的形式保存。

操作结果：生成一个新的文件，保存当前表。

status SaveList(List L);

初始条件：线性表L已存在。

算法设计：多表head开辟一块新的空间，保存当前L到多表中。

操作结果：保存当前表到多表。

## 2.3 系统实现

1. 初始化表：构造一个带头节点的空链表，使用L作为该节点的指针，同时L->data=0,L->next=NULL.

2. 销毁表：将单链表的每一个节点都free，同时将L置为NULL。

3. 清空：将L->next作为参数传递给Destroy()函数，同时将L->data置为0。

4. 判断空表，求表长：这两个操作都与线性表的长度有关，判断空表只需判断长度是否为0，求表长则返回线性表长度。

5. 取元素：由于采用的链式存储结构，需要使用一个循环结构，找到位序为i的节点并返回值。

6. 求位序：从头开始扫描线性表，直到碰到与要查找的值相同的元素，返回该元素的位序。

7. 求前驱和后继，与求位序类似，先求出要找的元素的前一个节点的位置，并用p指向，再由p指针返回要查找的值的前驱和后继。

8. 插入元素：如果需要在第i个位置上插入元素，则需要将p指向第i-1个元素，然后新建一个节点q，为该节点分配空间，q->data = e，然后将节点q插入到p后面，由于需要遍历链表所以时间复杂度为O(n)。

9. 删除元素：与插入类似，如果需要在第i个位置上插入元素，则需要将p指向第i-1个元素，q=p->next，然后将节点q的值赋值给e，最后将q节点free，由于需要遍历链表所以时间复杂度为O(n)。

10. 遍历：使用循环操作从头到尾依次打印元素值。

## 2.4 系统测试

程序采用简易界面，如图所示，挑选ListEmpty, ListLength, GetElem, LocateElem, PriorElem, NextElem, ListInsert, ListDelete, ListTraverse这些重要功能进行测试。

测试用例分线性表未创建，线性表为空，线性表不为空三种情况。

假设表list1中已经存储{1，2，3，4，56，3，5，3，45};null 表实空表。

 (1)ListEmpty测试

表2-1 ListEmpty测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| 空表 | 界面选4  Id 1 | 线性表为空！ | 线性表为空！ |
| 非空表 | 界面选4  Id 1 | 线性表不为空！ | 线性表不为空！ |
| 若表不存在 | 界面选4  Id 1 | 线性表未创建 | 线性表未创建！ |

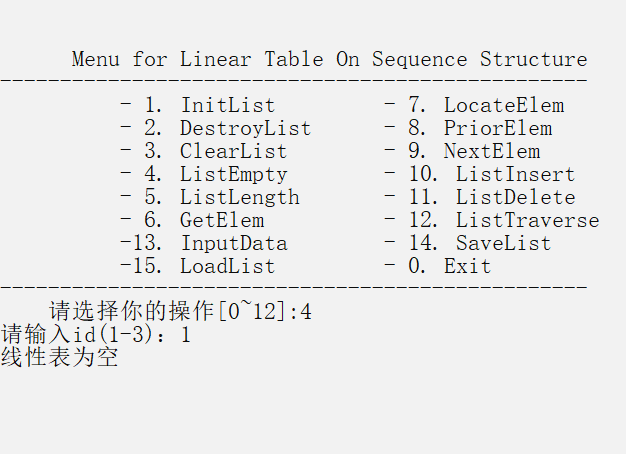


图2-6 判断空表（自身为空）

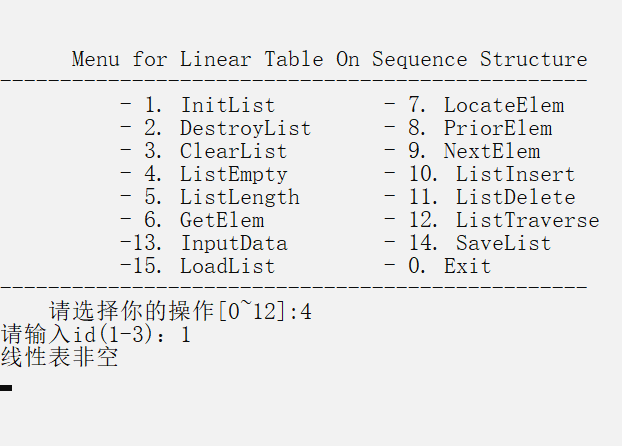


图2-7 判断空表（自身非空）

(2) ListLength测试

表2-2 ListLength测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选5  Id 1 | 线性表表长为9 | 线性表表长为9 |
| null | 界面选5  Id 1 | 线性表表长为0 | 线性表表长为0 |
| 若表不存在 | 界面选5  Id 1 | 线性表不存在！ | 线性表不存在！ |

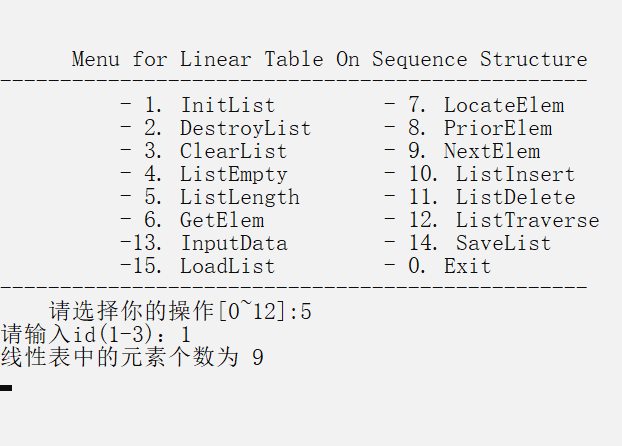


图2-8 求表长（用例List1）

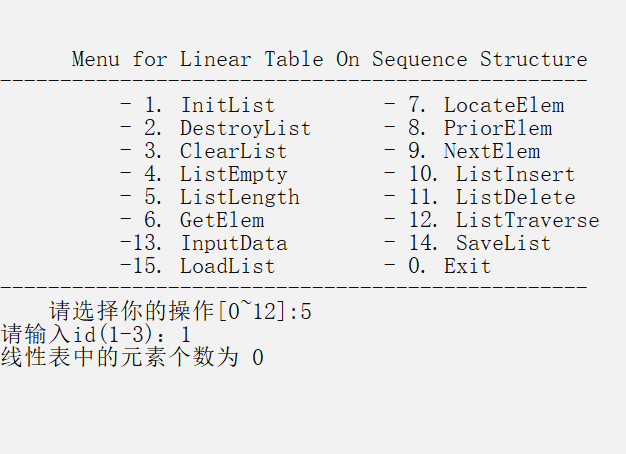


图2-9 求表长（用例null）

(3) GetElem测试

表2-3 GetElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选6  Id 1  输入位置3 | 第3位元素是3 | 第3位元素是3 |
| List1 | 界面选6  Id 1  输入位置12 | i值不合法 | i值不合法 |
| 若表不存在 | 界面选6  Id 1 | 线性表未创建 | 线性表未创建 |

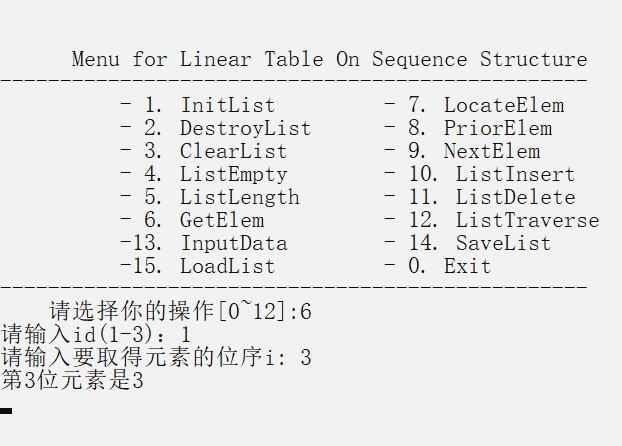


图2-10 取元素值（中间位置）

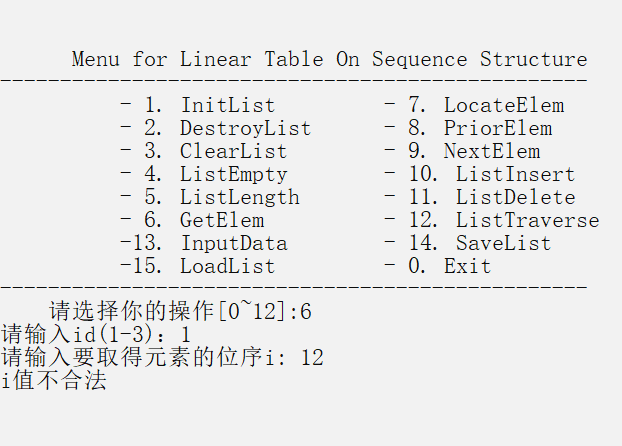


图2-11 取元素值（越界）

(4) LocateElem测试

表2-4 LocateElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选7  Id 1  输入元素4 | 4元素位于第4个位置 | 4元素位于第4个位置 |
| List1 | 界面选7  Id 1  输入元素20 | 线性表中无此元素 | 线性表中无此元素 |
| 若表不存在 | 界面选7 | 线性表未创建 | 线性表未创建 |

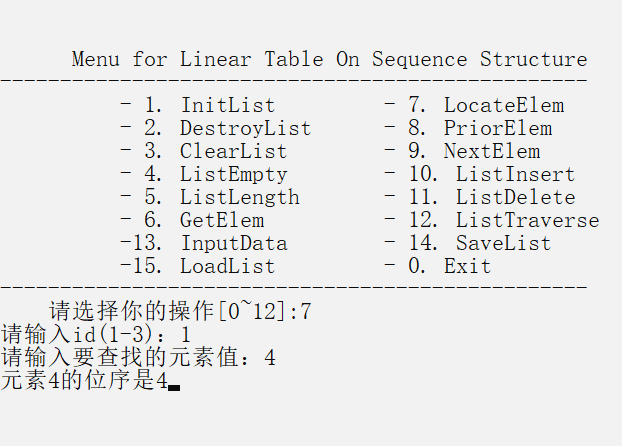


图2-12 返回位序（一般情况）

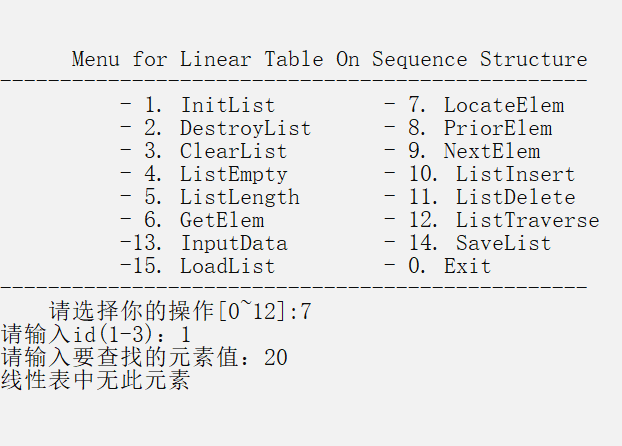


图2-13 返回位序（元素不存在）

(5) PriorElem测试

表1-5 PriorElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选8  Id 1  输入元素2 | 2的前驱是1 | 2的前驱是1 |
| List1 | 界面选8  Id 1  输入元素1 | 1为第一个元素,没有前驱 | 1为第一个元素,没有前驱 |
| List1 | 界面选8  Id 1  输入元素13 | 线性表中无此元素 | 线性表中无此元素 |
| 若表不存在 | 界面选8  Id 2 | 线性表未创建 | 线性表未创建 |

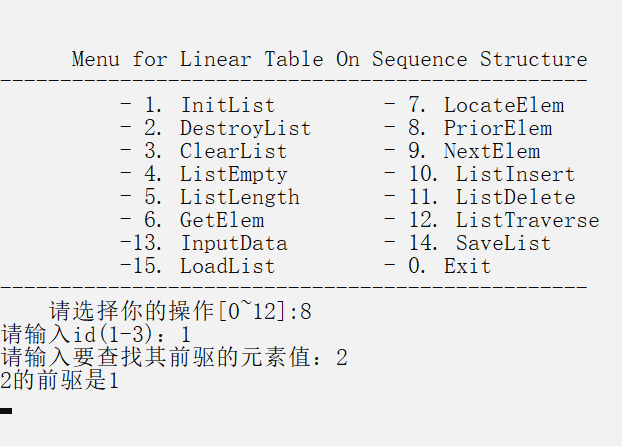


图2-14 求前驱（一般情况）

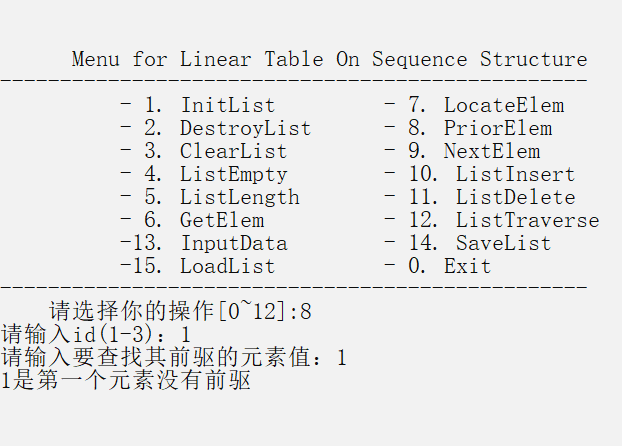


图2-15 求首元素前驱

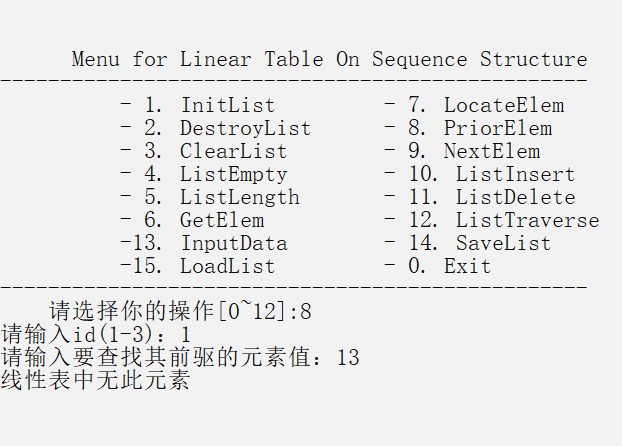


图2-16 求不存在元素前驱

(6) NextElem测试

表2-6 NextElem测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选9  Id 1  输入元素4 | 4的后继是56 | 4的后继是56 |
| List1 | 界面选9  Id 1  输入元素45 | 45为最后一个元素,没有后继 | 45为最后一个元素,没有后继 |
| List1 | 界面选9  Id 1  输入元素选14 | 线性表中无此元素 | 线性表中无此元素 |
| 若表不存在 | 界面选9  Id 2 | 线性表不存在！ | 线性表不存在！ |

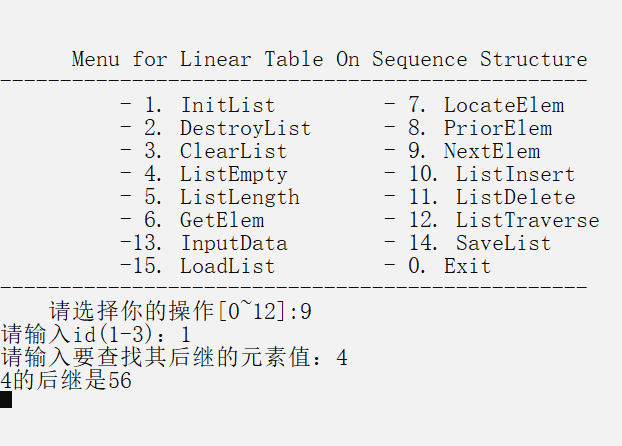


图2-17 求中间元素后继

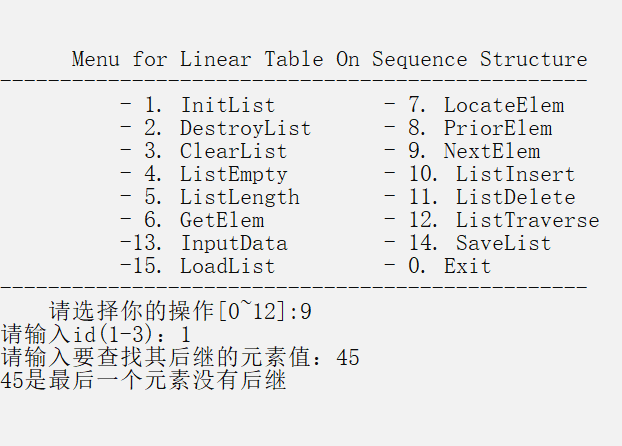


图2-18 求末尾元素后继

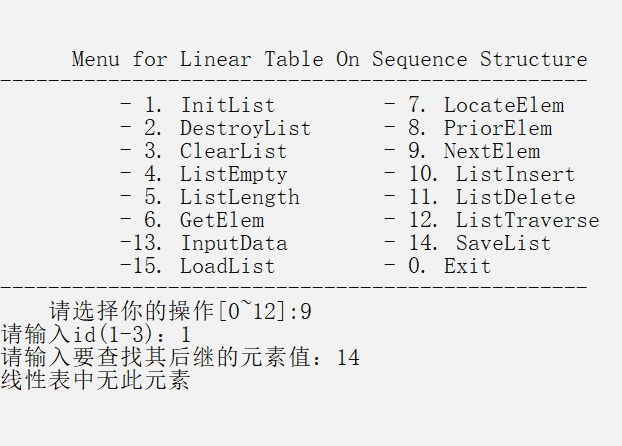


图2-19 求不存在元素后继

(7) ListInsert测试

表2-7 ListInsert测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 界面选10  Id 1  输入位置10  输入元素10 | 插入成功！ | 插入成功！ |
| List1 | 界面选10  Id 1  插入位置20  输入元素2 | 插入位置不合法 | 插入位置不合法 |
| 若表不存在 | 界面选10  Id 2 | 线性表未创建 | 线性表未创建 |

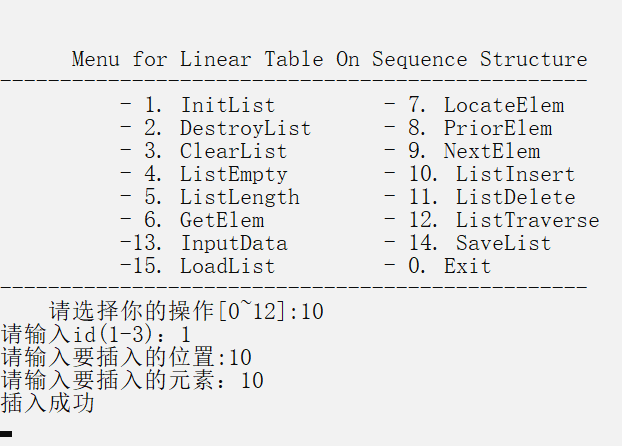


图2-20 插入元素

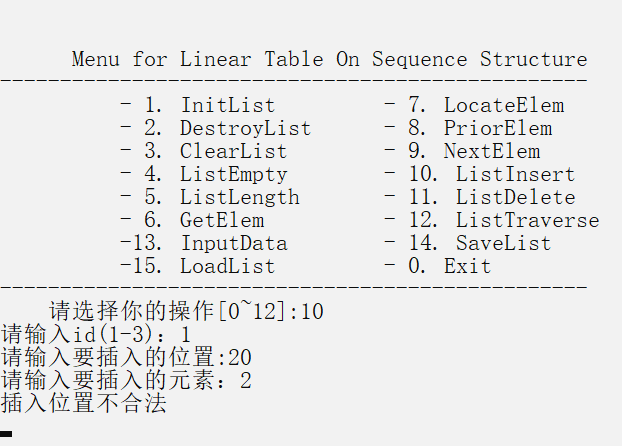


图2-21 插入位置不合法

(8) ListDelete测试

表2-8 ListDelete测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 输入 | 理论结果 | 测试结果 |
| List1 | 选11  Id 1  输入1 | 删除1成功！ | 删除1成功！ |
| List1 | 选11  Id 1  输入12 | 删除位置不合法 | 删除位置不合法 |
| 若表不存在 | 选11 | 线性表不存在！ | 线性表不存在！ |

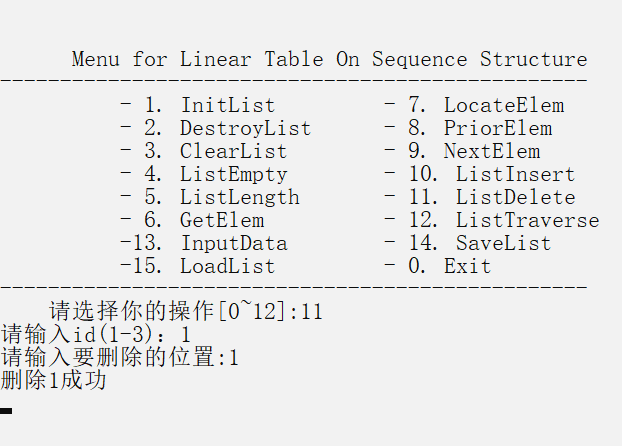


图2-22 删除元素

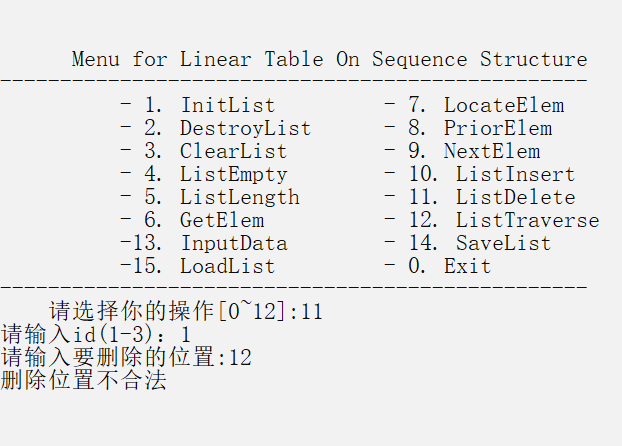


图2-23 删除位置不合法

(9) ListTraverse测试

表2-9 ListTraverse测试用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 测试结果 |
| List1 | 主界面选12  Id 1 | 1 --> 2 --> 3 --> 4 --> 56 --> 3 --> 5 --> 3 --> 45 | 1 --> 2 --> 3 --> 4 --> 56 --> 3 --> 5 --> 3 --> 45 |
| null | 主界面选12  Id 1 | 线性表为空 | 线性表为空 |
| 不存在的表 | 主界面选12 | 线性表未创建 | 线性表未创建 |

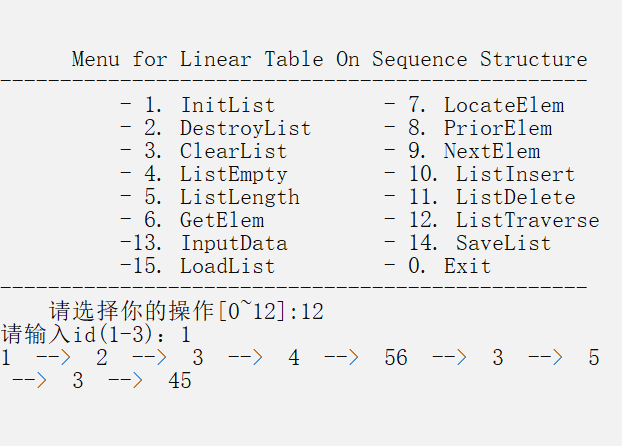


图2-24 遍历

## 2.5 实验小结

本次实验要求使用单链表的存储方式实现线性表，与上一次实验使用顺序表有一些差别，这主要体现在对元素的插入，删除，按位序查询上，由于顺序表要求在物理空间相邻，因此，插入和删除元素需要移动以保持物理空间上的相邻，而单链表则只需要改变指针的连接即可，在查询元素时，顺序表可以直接计算出元素位置，而单链表必须遍历。

对于同一种数据结构，可以采用不同的物理存储方式实现，而不同的方式之间有些操作简单，有些操作复杂，很难说清楚究竟是哪一个方式更好，但是就这两次的实验来说，我认为单链表实现线性表更加灵活

最后，我由衷感谢老师、助教和同学在本次实验中对我的帮助。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

## 3.2 系统设计

## 3.3 系统实现

## 3.4 系统测试

## 3.5 实验小结

# 4 基于邻接表的图实现

## 4.1 问题描述

## 4.2 系统设计

## 4.3 系统实现

## 4.4 系统测试

## 4.5 实验小结

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

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

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <malloc.h>

#include <stddef.h>

#include <stdio.h>

#include <stdlib.h>

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

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define NOTFOUND -3

typedef int status;

typedef int ElemType; //数据元素类型定义

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

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct { //顺序表（顺序结构）的定义

ElemType \*elem;

char name[41]; //表名字 在初始化表的时候写入

int length;

int listsize;

} SqList;

struct table {

SqList \*list;

struct table \*next;

};

typedef struct table Table;

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

status InitList(SqList \*\*L);

status DestroyList(SqList \*\*L);

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); //简化过

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 ListTraverse(SqList \*L); //简化过

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

status SaveList(SqList \*L); // 保存数据到文件

status LoadList(SqList \*L); // 从文件加载数据

status InputData(SqList \*L); // 手动输入数据

// 多表操作

status ShowLists(Table \*head); // 显示

status Remove(Table \*head); //删除

status AddList(Table \*head, SqList \*L); //增加

status Choose(Table \*head, SqList \*\*L); // 选择

int LengthofTable(Table \*head); // 求长度

int main(void) {

SqList \*L = NULL;

Table \*head = (Table \*)malloc(sizeof(Table));

head->next = NULL;

int op = 1;

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

// 此处为定义的一些临时变量 具体作用看不同的case

int i = 0;

int e = -1;

int pre\_e = -1;

int next\_e = -1;

status sta = OK; // 表示函数返回值

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

while (op) {

// linux

// system("clear");

// windows

system("cls");

printf("\n\n");

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

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

printf(" 1. InitList 7. LocateElem\n");

printf(" 2. DestroyList 8. PriorElem\n");

printf(" 3. ClearList 9. NextElem \n");

printf(" 4. ListEmpty 10. ListInsert\n");

printf(" 5. ListLength 11. ListDelete\n");

printf(" 6. GetElem 12. ListTraverse\n");

printf(" 13. InputData 14. SavaList\n");

printf(" 15. LoadList 16. Choose\n");

printf(" 17. AddToTable 18. ShowLists\n");

printf(" 19. Remove 0. Exit\n");

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

printf("请选择你的操作[0~19]:");

scanf("%d", &op);

switch (op) {

case 1:

// 初始化一个线性表

if (InitList(&L) == OVERFLOW)

printf("线性表创建失败！\n");

else

printf("线性表创建成功！\n");

printf("按任意键继续\n");

getchar();

getchar();

break;

case 2:

// 销毁线性表

if (DestroyList(&L) == INFEASTABLE)

printf("线性表销毁失败 因为线性表未创建！\n");

else

printf("线性表销毁成功！\n");

printf("按任意键继续\n");

getchar();

getchar();

break;

case 3:

// 清空线性表

if (ClearList(L) == OK)

printf("线性表清空成功！\n");

else

printf("线性表未创建，清空失败！\n");

printf("按任意键继续\n");

getchar();

getchar();

break;

case 4:

// 判断线性表是否为空

if (ListEmpty(L) == INFEASTABLE)

printf("线性表未创建！\n");

else if (ListEmpty(L) == FALSE)

printf("线性表不为空！\n");

else

printf("线性表为空！\n");

printf("按任意键继续\n");

getchar();

getchar();

break;

case 5:

// 求线性表长度

if (ListLength(L) != INFEASTABLE)

printf("线性表长度为%d\n", ListLength(L));

else

printf("线性表未创建\n");

printf("按任意键继续\n");

getchar();

getchar();

break;

case 6:

// 取得表中的第i位元素 从1开始数

printf("请输入要取得线性表中的哪一位元素(i) 从1开始数： ");

scanf("%d", &i);

sta = GetElem(L, i, &e); // sta表示函数状态

if (sta != OK) {

if (sta == INFEASTABLE)

printf("线性表未创建\n");

else if (sta == ERROR)

printf("i值不合法\n");

} else {

printf("第%d位元素是%d\n", i, e);

}

printf("按任意键继续\n");

getchar();

getchar();

break;

case 7:

//输入一个元素值 将返回它在表中的位置

printf("请输入一个元素值 将返回它在表中的位置\n");

scanf("%d", &e);

sta = LocateElem(L, e); // t表示函数的状态

if (sta == INFEASTABLE) {

printf("线性表未创建\n");

} else if (sta == NOTFOUND) {

printf("表中没有这个元素\n");

} else {

printf("%d的位置是%d\n", e, sta);

}

printf("按任意键继续\n");

getchar();

getchar();

break;

case 8:

// 求元素前驱

printf("输入线性表中的一个数: ");

scanf("%d", &e);

sta = PriorElem(L, e, &pre\_e); // sta 表示函数返回值

if (sta == OK)

printf("%d的前驱是%d\n", e, pre\_e);

else if (sta == INFEASTABLE)

printf("线性表未创建\n");

else if (sta == ERROR)

printf("%d为第一个元素,没有前驱\n", e);

else if (sta == NOTFOUND)

printf("表中不存在元素%d\n", e);

printf("按任意键继续\n");

getchar();

getchar();

break;

case 9:

// 求元素后继

printf("输入线性表中的一个数: ");

scanf("%d", &e);

sta = NextElem(L, e, &next\_e); // sta 表示函数返回值

if (sta == OK)

printf("%d的后继是%d\n", e, next\_e);

else if (sta == INFEASTABLE)

printf("线性表未创建\n");

else if (sta == ERROR)

printf("%d为最后一个元素,没有后继\n", e);

else if (sta == NOTFOUND)

printf("不存在元素%d\n", e);

printf("按任意键继续\n");

getchar();

getchar();

break;

case 10:

// 插入元素

printf("输入插入的位置i（从1开始计数）\n");

scanf("%d", &i);

printf("输入插入的值e\n");

scanf("%d", &e);

sta = ListInsert(L, i, e);

if (sta == OK)

printf("插入成功！\n");

else if (sta == INFEASTABLE)

printf("线性表未创建\n");

else if (sta == ERROR)

printf("插入位置不合法\n");

printf("按任意键继续\n");

getchar();

getchar();

break;

case 11:

// 删除元素

printf("输入删除的位置i（从1开始计数）\n");

scanf("%d", &i);

sta = ListDelete(L, i, &e);

if (sta == OK) {

printf("删除%d成功！\n", e);

} else if (sta == INFEASTABLE)

printf("线性表未创建\n");

else if (sta == ERROR)

printf("删除位置不合法\n");

printf("按任意键继续\n");

getchar();

getchar();

break;

case 12:

// 遍历元素

sta = ListTraverse(L);

if (sta == INFEASTABLE)

printf("线性表未创建\n");

else if (sta == ERROR)

printf("线性表为空\n");

printf("按任意键继续\n");

getchar();

getchar();

break;

case 13:

sta = InputData(L);

if (sta == INFEASTABLE) {

printf("线性表未创建\n");

}

printf("按任意键继续\n");

getchar();

getchar();

break;

case 14:

// 将当前表保存到文件

sta = SaveList(L);

if (sta == INFEASTABLE)

printf("线性表未创建\n");

else if (sta == ERROR)

printf("文件打开失败\n");

else

printf("保存为%s成功\n", L->name);

printf("按任意键继续\n");

getchar();

getchar();

break;

case 15:

// 从文件加载表 前提是先创建表

sta = LoadList(L);

if (sta == INFEASTABLE)

printf("线性表未创建\n");

else if (sta == ERROR)

printf("文件打开失败\n");

else

printf("加载成功\n");

getchar();

getchar();

break;

case 16:

// 从多表中选取一个表

sta = Choose(head, &L);

if (sta == ERROR)

printf("序号不合法");

else if (sta == OK)

printf("按任意键继续\n");

getchar();

getchar();

break;

case 17:

// 添加

sta = AddList(head, L);

if (sta == INFEASTABLE)

printf("线性表未创键\n");

printf("按任意键继续\n");

getchar();

getchar();

break;

case 18:

// 显示

sta = ShowLists(head);

printf("按任意键继续\n");

getchar();

getchar();

break;

case 19:

sta = Remove(head);

if (sta == ERROR)

printf("i值不合法\n");

else

printf("按任意键继续\n");

getchar();

getchar();

break;

case 0:

break;

default:

printf("请输入有效值\n");

getchar();

getchar();

break;

} // end of switch

} // end of while

printf("欢迎下次再使用本系统！\n");

return 0;

} // end of main()

status InitList(SqList \*\*L) {

\*L = (SqList \*)malloc(sizeof(SqList));

if (\*L == NULL) {

// L动态内存分配失败

return OVERFLOW;

}

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

if ((\*L)->elem == NULL) {

// L->elem动态内存分配失败

return OVERFLOW;

}

printf("请输入表名字 不超过40个字符\n");

scanf("%s", (\*L)->name);

(\*L)->length = 0;

(\*L)->listsize = LIST\_INIT\_SIZE;

return OK;

}

status DestroyList(SqList \*\*L) {

if ((\*L) == NULL) // 线性表未创建

return INFEASTABLE;

else if (!(\*L)->elem)

return INFEASTABLE;

(\*L)->listsize = (\*L)->length = 0;

free((\*L)->elem);

(\*L)->elem = NULL;

free(\*L);

\*L = NULL;

return OK;

}

status ClearList(SqList \*L) {

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

L->length = 0;

return OK;

}

status ListEmpty(SqList \*L) {

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

return L->length == 0; // L->length == 0则为空表

}

int ListLength(SqList \*L) {

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

return L->length;

}

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

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

if (i < 1 || i > L->length) // 请求越界

{

return ERROR;

}

\*e = \*(L->elem + i - 1); // 把e指向要取得的元素

return OK;

}

int LocateElem(SqList \*L, ElemType e) {

// 以1开始

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

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

if (e == L->elem[i - 1]) {

return i; //返回第位置i i从1开始

}

}

return NOTFOUND; //没有找到

}

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

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

// 寻找元素cur

int i = 1;

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

if (cur == L->elem[i - 1])

break;

if (i == L->length + 1) // 没找到

{

return NOTFOUND;

} else if (i == 1) // 如果cur为第一个元素 则没有前驱

{

return ERROR;

} else {

\*pre\_e = \*(L->elem + (i - 2));

return OK;

}

}

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

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

// 寻找元素cur

int i = 1;

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

if (cur == L->elem[i - 1]) {

break;

}

}

if (i == L->length + 1) // 没找到

{

return NOTFOUND;

} else if (i == L->length) // 如果cur为最后一个元素 则没有后继

{

return ERROR;

} else {

\*next\_e = \*(L->elem + (i));

return OK;

}

}

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

// 插入元素

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

if (i < 1 || i > L->length + 1) // 插入位置不合法

return ERROR;

if (L->length >= L->listsize) { // 空间不足 重新分配

ElemType \*newbase = (ElemType \*)realloc(

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

if (!newbase)

exit(OVERFLOW);

L->elem = newbase;

L->listsize += LISTINCREMENT;

}

ElemType \*q = &(L->elem[i - 1]); // q为要插入的位置

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

// 插入位置及之后的元素右移

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

\*q = e;

L->length++;

return OK;

}

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

// 在顺序线性表L中删除第i个元素 并用e返回值

// i的合法位置为1<= i <= L->length

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

if (i < 1 || i > L->length) // i值不合法

return ERROR;

ElemType \*p = &(L->elem[i - 1]); // p为要删除的元素地址

\*e = \*p;

ElemType \*q = &(L->elem[L->length - 1]); // q为表尾元素地址

for (p++; p <= q; p++) // 元素移动

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

L->length--;

return OK;

}

status ListTraverse(SqList \*L) {

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

if (L->length == 0)

// 空表

return ERROR;

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

// 遍历

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

if (i < L->length - 1)

printf(" --> ");

}

putchar('\n');

return OK;

}

status InputData(SqList \*L) {

// 手动输入数据

if (!L) // 线性表未创建

{

return INFEASTABLE;

} else if (!L->elem) {

return INFEASTABLE;

}

printf("请输入数据（整数），以空格分隔,以q退出\n");

int x = 0;

while ((scanf("%d", &x) == 1)) {

if (L->length >= L->listsize) { // 空间不足 重新分配

ElemType \*newbase = (ElemType \*)realloc(

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

if (!newbase)

exit(OVERFLOW);

L->elem = newbase;

L->listsize += LISTINCREMENT;

}

L->elem[L->length++] = x; // 写入数据

}

return OK;

}

status SaveList(SqList \*L) {

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

FILE \*fp;

// 写文件

if ((fp = fopen(L->name, "wb")) == NULL) //文件打开失败

{

printf("File open error\n ");

return ERROR;

}

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

fclose(fp);

return OK;

}

status LoadList(SqList \*L) {

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

//读文件

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s", filename);

L->length = 0;

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

printf("File open error\n ");

return ERROR;

}

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

L->length++;

fclose(fp);

return OK;

}

status AddList(Table \*head, SqList \*L) {

if (!L) // 线性表未创建

return INFEASTABLE;

else if (!L->elem)

return INFEASTABLE;

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

p->list = L;

p->next = NULL;

// 后进先出 带头节点

p->next = head->next;

head->next = p;

return OK;

}

status ShowLists(Table \*head) {

Table \*p = head->next;

int i = 1;

printf("序号\t表长\t表名\n");

while (p) {

if (p->list->elem == NULL) {

i++;

p = p->next;

continue;

}

printf("%d\t%d\t%s\n", i++, p->list->length, p->list->name);

p = p->next;

}

return OK;

}

int LengthofTable(Table \*head) {

int i = 0;

Table \*p = head->next;

while (p) {

i++;

p = p->next;

}

return i;

}

status Choose(Table \*head, SqList \*\*L) {

int i = 0;

Table \*p = head->next;

Table \*q = p;

ShowLists(head);

printf("请输入选择的表的序号：");

scanf("%d", &i);

if (i < 1 || i > LengthofTable(head)) // 序号不合法

return ERROR;

int k = i;

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

q = q->next;

}

if (q->list->elem == NULL)

return ERROR;

int j = 0;

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

p = p->next;

}

(\*L) = p->list;

return OK;

}

status Remove(Table \*head) {

int i = 0;

ShowLists(head);

printf("输入删除序号：");

scanf("%d", &i);

if (i < 1 || i > LengthofTable(head)) // 序号不合法

return ERROR;

Table \*p = head;

int j = 0;

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

p = p->next;

}

Table \*q = p->next; // 改变结构 删除节点

p->next = q->next;

free(q->list->elem);

free(q->list);

free(q);

return OK;

}

# 附录B 基于链式存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

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

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR -3

#define INFEASTABLE -1

#define OVERFLOW -2

#define NOTFOUND -4

#define LISTNUM 3

typedef int status;

typedef int ElemType; //数据元素类型定义

/\*------- 节点定义 -------\*/

typedef struct node

{

ElemType data;

struct node \*next;

} LNode;

typedef LNode \*PtrToLNode; // PtrToLNode 用来(PtrToLNode)malloc(sizeof(LNode));

// 由此定义线性表

typedef PtrToLNode List; // List用来代表一个表 （带头节点） List L = (List)malloc(sizeof(LNode)) 头节点的数据域用来存长度;

/\*-------------函数声明-----------------\*/

PtrToLNode GetAdress(List L, int e); // 返回某一个元素的前一个元素的地址值 不存在则返回最后一个节点的地址

status isinit(List L); // 判断L是否初始化

status InitList(List \*L); // 初始化表

status DestroyList(List \*L); // 摧毁表

status ClearList(List L); // 清空表

status ListEmpty(List L); // 表是否为空

status ListLength(List L); // 返回线性表长度

status GetElem(List L, int i, int \*e); // 用e返回第i位置的值

int LocateElem(List L, int e); // 返回e所在的位序

status PriorElem(List L, int cur\_e, int \*pre\_e); // 用pre\_e 带回 cur\_e 的前驱

status NextElem(List L, int cur\_e, int \*next\_e); // 用next\_e带回 cur\_e 的后继

status ListInsert(List L, int i, int e); // 在i的位置上插入e

status ListDelete(List L, int i, int \*e); // 删除i的位置的值 用e带出

status ListTraverse(List L); // 遍历

status InputData(List L); //手工输入数据

status LoadList(List L); // 加载

status SaveList(List L); // 保存

int main(void)

{

List L[LISTNUM] = {NULL};

int op = 1;

// 临时变量

int i = 0, e = 0;

int cur\_e = 0, next\_e = 0, pre\_e = 0;

status sta = OK;

int id = -1;

while (op)

{

system("cls");

printf("\n\n");

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

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

printf(" - 1. InitList - 7. LocateElem\n");

printf(" - 2. DestroyList - 8. PriorElem\n");

printf(" - 3. ClearList - 9. NextElem \n");

printf(" - 4. ListEmpty - 10. ListInsert\n");

printf(" - 5. ListLength - 11. ListDelete\n");

printf(" - 6. GetElem - 12. ListTraverse\n");

printf(" -13. InputData - 14. SaveList\n");

printf(" -15. LoadList - 0. Exit\n");

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

printf(" 请选择你的操作[0~12]:");

scanf("%d", &op);

switch (op)

{

case 1:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

sta = InitList(&L[id-1]);

if (sta == OK)

printf("线性表创建成功！\n");

else if (sta == OVERFLOW)

printf("线性表创建失败！\n");

getchar();

getchar();

break;

case 2:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

DestroyList(&L[id-1]);

printf("销毁成功\n");

}

getchar();

getchar();

break;

case 3:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

ClearList(L[id-1]);

printf("线性表清空成功\n");

}

getchar();

getchar();

break;

case 4:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

if (ListEmpty(L[id-1]) == TRUE)

printf("线性表为空\n");

else

printf("线性表非空\n");

}

getchar();

getchar();

break;

case 5:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

printf("线性表中的元素个数为 %d\n", ListLength(L[id-1]));

}

getchar();

getchar();

break;

case 6:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

printf("请输入要取得元素的位序i: ");

scanf("%d", &i);

sta = GetElem(L[id-1], i, &e);

if (sta == ERROR)

printf("i值不合法\n");

else if (sta == OK)

printf("第%d位元素是%d\n", i, e);

}

getchar();

getchar();

break;

case 7:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

printf("请输入要查找的元素值：");

scanf("%d", &e);

sta = LocateElem(L[id-1], e);

if (sta == NOTFOUND)

printf("线性表中无此元素\n");

else

printf("元素%d的位序是%d", e, sta);

}

getchar();

getchar();

break;

case 8:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

printf("请输入要查找其前驱的元素值：");

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

sta = PriorElem(L[id-1], cur\_e, &pre\_e);

if (sta == NOTFOUND)

printf("线性表中无此元素\n");

else if (sta == ERROR)

printf("%d是第一个元素没有前驱\n", cur\_e);

else

printf("%d的前驱是%d\n", cur\_e, pre\_e);

}

getchar();

getchar();

break;

case 9:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

printf("请输入要查找其后继的元素值：");

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

sta = NextElem(L[id-1], cur\_e, &next\_e);

if (sta == NOTFOUND)

printf("线性表中无此元素\n");

else if (sta == ERROR)

printf("%d是最后一个元素没有后继\n", cur\_e);

else

printf("%d的后继是%d\n", cur\_e, next\_e);

}

getchar();

getchar();

break;

case 10:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

printf("请输入要插入的位置:");

scanf("%d", &i);

printf("请输入要插入的元素：");

scanf("%d", &e);

sta = ListInsert(L[id-1], i, e);

if (sta == ERROR)

printf("插入位置不合法\n");

else

printf("插入成功\n");

}

getchar();

getchar();

break;

case 11:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

printf("请输入要删除的位置:");

scanf("%d", &i);

sta = ListDelete(L[id-1], i, &e);

if (sta == ERROR)

printf("删除位置不合法\n");

else

printf("删除%d成功\n", e);

}

getchar();

getchar();

break;

case 12:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

sta = ListTraverse(L[id-1]);

if (sta == ERROR)

printf("线性表为空表\n");

}

getchar();

getchar();

break;

case 13:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

printf("请输入数据，以q退出\n");

InputData(L[id-1]);

}

getchar();

getchar();

break;

case 14:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

if (SaveList(L[id-1]) == ERROR)

printf("文件打开失败\n");

else

printf("保存成功\n");

}

getchar();

getchar();

break;

case 15:

printf("请输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

while(id<1||id>LISTNUM)

{

printf("id值无效，请重新输入id(1-%d)：",LISTNUM);

scanf("%d",&id);

}

if (isinit(L[id-1]) == FALSE)

printf("线性表未创建\n");

else

{

if (LoadList(L[id-1]) == ERROR)

printf("文件打开失败\n");

else

printf("加载成功\n");

}

getchar();

getchar();

break;

case 0:

break;

} //end of switch

} //end of while

printf("欢迎下次再使用本系统！\n");

return 0;

} //end of main()

status isinit(List L)

{

if (L == NULL)

return FALSE;

else

return TRUE;

}

PtrToLNode GetAdress(List L, int e)

{

// 返回某一个元素的前一个元素的地址值 不存在则返回最后一个节点的地址

PtrToLNode p = L;

while (p->next && p->next->data != e)

p = p->next;

return p;

}

status InitList(List \*L)

{

// 分配空间

(\*L) = (List)malloc(sizeof(LNode)); // 创建一个带头节点的空链表

if ((\*L) == NULL)

return (OVERFLOW);

(\*L)->next = NULL;

(\*L)->data = 0; // 0表示当前表的长度为0

return OK; // 创建成功

} // InitList

status DestroyList(List \*L)

{

// L是这个表的头节点 最后把free(L)

// 然后把L为NULL

PtrToLNode temp1 = (\*L); // 临时变量指向头部

PtrToLNode temp2 = temp1;

while (temp1)

{

// 释放各个节点空间

temp2 = temp1->next;

free(temp1);

temp1 = temp2;

}

// 将L置为空

\*L = NULL;

return OK;

}

status ListEmpty(List L)

{

if (L->data == 0)

return TRUE;

else

return FALSE;

}

status ClearList(List L)

{

// 保留头节点 将头节点data设置为0

// free 其他节点

PtrToLNode t = L->next;

if (ListEmpty(L) == TRUE)

return OK;

else

DestroyList(&t);

L->next = NULL;

L->data = 0;

return OK;

}

int ListLength(List L)

{

// 返回线性表长度

return L->data;

}

status GetElem(List L, int i, int \*e)

{

if (i < 1 || i > ListLength(L)) //i 不合法

return ERROR;

int k = 0;

PtrToLNode p = L->next;

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

{

p = p->next;

}

\*e = p->data;

return OK;

}

int LocateElem(List L, int e)

{

int i = 1;

PtrToLNode p = L->next;

while (p && p->data != e)

{

p = p->next;

i++;

}

if (p) // 如果找到

return i;

else //没找到

return NOTFOUND;

}

status PriorElem(List L, int cur\_e, int \*pre\_e)

{

PtrToLNode p = GetAdress(L, cur\_e);

if (p->next == NULL) // 没有这一个元素

return NOTFOUND;

else if (p == L) // 第一个元素 没有前驱

return ERROR;

else

{

\*pre\_e = p->data;

return OK;

}

}

status NextElem(List L, int cur\_e, int \*next\_e)

{

PtrToLNode p = GetAdress(L, cur\_e);

if (p->next == NULL) // 没有这一个元素

return NOTFOUND;

else if (p->next->next == NULL) // 最后一个元素 没有后继

return ERROR;

else

{

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

return OK;

}

}

status ListInsert(List L, int i, int e)

{

//在i的位置上插入e

PtrToLNode p = L; // p为要插入的位置的前一个位置

if (i < 1 || i > ListLength(L) + 1) //i 不合法

return ERROR;

else

while (--i)

p = p->next;

PtrToLNode q = (PtrToLNode)malloc(sizeof(LNode));

q->data = e;

q->next = p->next;

p->next = q;

L->data++;

return OK;

}

status ListDelete(List L, int i, int \*e)

{

// 删除i的位置的值 用e带出

PtrToLNode p = L; // p为要删除的位置的前一个位置

if (i < 1 || i > ListLength(L)) //i 不合法

return ERROR;

else

while (--i)

p = p->next;

PtrToLNode q = p->next;

p->next = q->next;

\*e = q->data;

free(q);

L->data--;

return OK;

}

status ListTraverse(List L)

{

// 遍历

if (ListEmpty(L) == TRUE)

return ERROR;

else

{

PtrToLNode p = L->next;

while (p)

{

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

p = p->next;

if (p)

printf(" --> ");

}

}

printf("\n");

return OK;

}

status InputData(List L)

{

//手工输入数据 添到当前表的尾部

PtrToLNode p = L;

while (p->next) // 找到尾部

{

p = p->next;

}

int t;

while (scanf("%d", &t) == 1)

{

PtrToLNode q = (PtrToLNode)malloc(sizeof(LNode)); // 临时变量

q->data = t;

q->next = NULL;

p->next = q;

p = p->next;

L->data++;

}

p->next = NULL;

return OK;

}

status SaveList(List L)

{

char name[41];

printf("请输入文件名，少于40字符\n");

scanf("%s", name);

FILE \*fp;

// 写文件

if ((fp = fopen(name, "wb")) == NULL) //文件打开失败

{

printf("File open error\n ");

return ERROR;

}

PtrToLNode p = L->next;

while (p)

{

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

p = p->next;

}

fclose(fp);

return OK;

}

status LoadList(List L)

{

//读文件

FILE \*fp;

char filename[30];

printf("input file name: ");

scanf("%s", filename);

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

{

printf("File open error\n ");

return ERROR;

}

L->data = 0;

PtrToLNode p = L;

PtrToLNode q = (PtrToLNode)malloc(sizeof(LNode));

while (fread(&(q->data), sizeof(ElemType), 1, fp))

{

q->next = NULL;

p->next = q;

p = p->next;

L->data++;

q = (PtrToLNode)malloc(sizeof(LNode));

}

p->next = NULL;

fclose(fp);

return OK;

}

# 附录C 基于二叉链表二叉树实现的源程序

# 附录D 基于邻接表图实现的源程序