

**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（交换生）**

**学 号： X2020I1007**

**姓 名： 刘日星**

**指导教师： 郑渤龙**

**报告日期： 2021年 5月 6 日**

**计算机科学与技术学院**

目 录

[1 基于顺序存储结构的线性表实现 2](#_Toc458159879)

[1.1 问题描述 2](#_Toc458159880)

[1.2 系统设计 2](#_Toc458159882)

[1.3 系统实现 2](#_Toc458159883)

[1.4 实验小结 2](#_Toc458159884)

[2 基于链式存储结构的线性表实现 2](#_Toc458159885)

[2.1 问题描述 2](#_Toc458159886)

[2.2 系统设计 2](#_Toc458159887)

[2.3 系统实现 2](#_Toc458159888)

[2.4 实验小结 2](#_Toc458159889)

[3 基于二叉链表的二叉树实现 2](#_Toc458159890)

[3.1 问题描述 2](#_Toc458159891)

[3.2 系统设计 2](#_Toc458159892)

[3.3 系统实现 2](#_Toc458159893)

[3.4 实验小结 2](#_Toc458159894)

[4 基于二叉链表的二叉树实现 2](#_Toc458159895)

[4.1 问题描述 2](#_Toc458159896)

[4.2 系统设计 2](#_Toc458159897)

[4.3 系统实现 2](#_Toc458159898)

[4.4 实验小结 2](#_Toc458159899)

[参考文献 2](#_Toc458159900)

[附录A 基于顺序存储结构线性表实现的源程序 2](#_Toc458159901)

[附录B 基于链式存储结构线性表实现的源程序 2](#_Toc458159902)

[附录C 基于二叉链表二叉树实现的源程序 2](#_Toc458159903)

[附录D 基于邻接表图实现的源程序 2](#_Toc458159904)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述（说明此实验要解决的基本问题）

运用顺序表的存储结构（物理结构）构造一个具有线性表基本功能的演示系统。其系统定义了线性表的初始化表、销毁表、清空表、判断空表、求表长和获得表中元素等线性表的基本运算所对应的函数，并给予一些对应基本运算的操作提示。

## 1.1.1 线性表的基本概念

线性表是最常用且最简单的一种数据结构。换言之，一个线性表是n个数据元素的有限序列。在线性表中，n为元素的个数表示线性表的长度，当n为0时，线性表为空表（表长为0）。当线性表非空时，每个数据元素都有一个确定的位置以及固定的前驱和后继结点，例如a1是首元素（第一个数据元素），an是尾元素（最后一个数据元素），ai是表中第i个元素。线性表的存储结构又分为顺序存储结构和链式存储结构。

## 1.1.2 线性表的抽象数据类型及基本操作

抽象数据类型线性表的定义如下（含12种线性表的基本操作）：

ADT List{

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：

1. 初始化线性表：InitList(&L)

初始条件：线性表已存在或不存在；

操作结果：构造一个空的线性表L。

1. 销毁线性表：DestroyList(&L)

初始条件：线性表L已存在；

操作结果：销毁线性表L。

1. 清空线性表：ClearList(&L)

初始条件：线性表L已存在；

操作结果：将线性表L重置为空表。

1. 判断空表：ListEmpty(L)

初始条件：线性表L已存在；

操作结果：若线性表L为空表，则返回TRUE，否则返回FALSE。

1. 求表长：ListLength(L)

初始条件：线性表L已存在；

操作结果：返回线性表L中数据元素的个数。

1. 元素获取：GetElem(L,i,e)

初始条件：线性表L已存在，1≤i≤ListLength(L)；

操作结果：用e返回线性表L中第i个元素的值。

1. 元素查找：LocateElem(L,e)

初始条件：线性表L已存在；

操作结果：返回线性表L中第1个与e满足关系数据元素的位序。若这样的数据元素不存在，则返回值为0。

1. 获得前驱结点：PriorElem(L,e,&pre)

初始条件：线性表L已存在；

操作结果：若e是线性表L的数据元素，且不是第一个，则用pre返回它的前驱，否则操作失败，pre无定义。

1. 获得后继结点：NextElem(L,e,&next)

初始条件：线性表L已存在；

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

1. 插入元素：ListInsert(L,i,e)

初始条件：线性表L已存在，1≤i≤ListLength(L)+1；

操作结果：在线性表L中第i个位置之前插入新的数据元素e，线性表L的长度+1。

1. 删除元素：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：删除线性表L的第i个数据元素，并用e返回其值，线性表L的长度减1。

1. 遍历表：ListTraverse(L)

初始条件：线性表L已存在；

操作结果：依次对线性表L的每个数据元素进行扫描。如果数据元素不存在则返回线性表为空，否则按顺序返回线性表L中的每个元素。

}ADT List

## 1.2 系统设计（包括整体系统结构设计和数据结构设计等）

## 1.2.1 线性表顺序存储结构（物理结构）

线性表的顺序存储结构：

#define maxleng 100

typedef struct{ //顺序表（顺序结构）定义

ElemType elem[maxleng]; //定义指针为存储空间基地址

int length; //线性表长度

int listsize; //当前分配的存储容量

//（以sizeof(ElemType)为单位）

}SqList;

只需要定义一个结构数组就可以实现同时对多个线性表的管理。

## 1.2.2 整体系统设计

用一个while循环语句实现线性表的12个基本功能，使用12个case功能语句分别调用12个线性表基本功能的函数并在case里对线性表进行判断是否已存在，再通过使用一个switch语句将12个case语句组合成一个菜单栏实现对线性表基本功能的操作选择。

## 1.2.3 数据结构设计（算法设计）

线性表运算算法设计与思想：

1. InitList(&L)

设计思想： 给线性表L分配存储空间，并将线性表长度置为0；线性表L若已存在，显示创建失败。

1. DestroyList(&L)

设计思想： 将线性表L的表长置为0，销毁线性表L，释放线性表L的数据元素空间；若线性表L不存在，则销毁失败。

1. ClearList(&L)

设计思想：如果线性表L已存在则清空线性表内所有内容，表长置为0。

1. ListEmpty(L)

设计思想：判断线性表L是否为空，是则返回TRUE，否则返回FALSE。

1. ListLength(L)

设计思想：若线性表L已存在，则返回线性表L的表长。

1. GetElem(L,i,&e)

设计思想：判断线性表L是否为空，若为空显示数据元素获取失败，否则判断输入值i是否存在线性表L的数据位置范围内，然后将获得的值L.elem[i]赋值给e。

1. LocateElem(L,e)

设计思想：扫描线性表L，若数据元素e在线性表L中，返回e在线性表L中的位置序号，否则显示查找失败。

1. PriorElem(L,e,&pre)

设计思想：扫描线性表L中所有元素，若输入数据元素e为线性表L中第一个数据元素或未找到该数据元素则显示错误，否则将L.elem[i-1]的值赋予给pre。

1. NextElem(L,e,&next)

设计思想：扫描线性表L中所有元素，若输入数据元素e为线性表L中最后一个数据元素或未找到该数据元素则显示错误，否则将L.elem[i+1]的值赋予给next。

1. ListInsert(&L,i,e)

设计思想：在线性表L存在下，将数据元素e插入线性表L的第i个元素之前，如果第i个元素的位置序号不合理，则显示插入错误；如果线性表L空间已满则需要重新分配新的存储空间；将插入位置后的数据元素整体后移，线性表L的表长加1，再插入数据元素e。

1. ListDelete(&L,i,&e)

设计思想：在线性表L存在下，将线性表L中第i个数据元素删除；如果第i个数据元素的位置不合理则显示删除错误，否则将删除位置后的数据元素整体往前移并将被删除的数据元素的值赋值给e，线性表L的表长减1，显示e的值。

1. ListTraverse(L)

设计思想：若线性表L已存在，则使用一个循环函数依次对线性表L中的所有元素扫描一遍并输出，如果线性表L不存在，则显示线性表不存在。

## 1.3 系统实现（主要说明各个主要函数的实现思想，复杂函数可辅助流程图进行说明，函数和系统实现的源代码放在附录中）

## 1.3.1 主要函数实现

1.头文件：

#define<stdio.h>

#define<malloc.h>

#define<stdlib.h>

2.常量定义:

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define INIT\_LIST\_SIZE 100

#define LISTINCREMENT 10

3.表达式类型：

typedef int status;

typedef int ElemType;

4.主要函数实现：

(1)InitList: 初始化线性表，使用malloc函数给线性表L动态分配listsize大小的存储空间；若分配成功则将线性表L的长度置为0，将线性表L的存储空间容量listsize设置为INIT\_LIST\_SIZE大小(100)，返回OK；若分配不成功则返回INFEASTABLE。

(2)DestroyList: 销毁线性表，将线性表L的长度置为0，线性表L的存储空间同样置为0，然后使用free函数将线性表L.elem的空间释放，成功则返回OK，否则返回INFEASTABLE。

(3)ClearList: 清空线性表，将线性表L的长度置为空(0)；成功则返回OK，否则返回INFEASTABLE。

(4)ListEmpty: 判断线性表是否为空，使用一个if函数判断线性表L的长度是否为空，若为空返回TRUE，否则返回FALSE。

(5)ListLength: 返回线性表L的长度。

(6)GetElem: 获取数据元素的位置，输入一个位置序号i，判断该位置序号i是否小于1或超过线性表L的长度范围，若该位置序号i在线性表L内，则将线性表L中第i个元素的值赋值到e并返回OK，否则返回INFEASTABLE。

(7)LocateElem: 查找线性表L中的数据元素，输入一个数据元素e，通过使用while函数循环遍历整个线性表L，若数据元素e在线性表L里，返回该数据元素的位置序号并返回OK，否则显示数据元素e不存在并返回ERROR。

(8)PriorElem: 获取前驱节点，输入一个数据元素e，通过使用while函数循环遍历整个线性表L，若数据元素e在线性表L里，则将该数据元素e的前驱节点值L.elem[i-1]赋值给pre并返回OK，否则该数据元素没有前驱节点并返回ERROR。

(9)NextElem: 获取后继节点，输入一个数据元素e，通过使用while函数循环遍历整个线性表L，若数据元素e在线性表L里，则将该数据元素e的后继节点值L.elem[i+1]赋值给next并返回OK，否则该数据元素没有后继节点并返回ERROR。

(10)ListInsert: 插入数据元素，先判断线性表是否已满，若线性表L已满，则使用realloc函数给线性表L重新分配存储空间；然后将第i个数据元素及后面的数据元素整体后移1位，线性表L的长度加1并将数据元素e插入到第i个数据元素之前的空位且返回OK；若插入位置不正确则返回ERROR。

(11)ListDelete: 删除数据元素，输入一个要删除的数据元素位序i，若该数据元素位序小于1或超出线性表L的长度范围，则返回ERROR；否则将第i个数据元素从线性表L中删除并将被删除的第i个数据元素的值L.elem[i-1]赋值给e，然后将该数据元素位序后的数据元素依次往前移动1位，线性表L的长度减1并返回OK。

(12)ListTraverse: 遍历线性表，使用一个for循环函数依序扫描线性表L内的数据元素，并将所有的数据元素按顺序输出显示，最后返回OK。

## 1.3.2 源代码

（详细源代码见附录A）

* 1. **系统测试**（主要说明针对各个函数正常和异常的测试用例及测试结果）
* 开发环境：Windows 10
* 编译器：Visual Studio 2019
* 在程序的开始时，会提示用户选择一个线性表进行操作，如图1-1：

**图1-1**

* 文本

  描述已自动生成**文本

  描述已自动生成**选择完线性表后，便会出现一个主菜单，给予用户选择对线性表的操作如图1-2：

**图1-2**

1. InitList：初始化线性表，给线性表动态分配存储空间（默认存储空间容量为100），若分配成功，则显示线性表创建成功如图1-3：

文本

描述已自动生成**图1-3**

1. 文本

   描述已自动生成DestroyList：销毁线性表，将线性表表长和listsize置为0，使用free函数将线性表的存储空间释放掉，成功则显示线性表删除成功如图1-4，否则显示线性表不存在如图1-5：

**图1-4**

**图1-5**

文本

描述已自动生成

1. 文本

   描述已自动生成ClearList：清空线性表，将线性表的表长置为0，成功则显示线性表清空成功如图1-6：

**图1-6**

1. ListEmpty：判断线性表是否为空，如果线性表为空表长为0，则显示线性表为空如图1-7，否则显示线性表不为空；为了验证此线性表是否真文本

   描述已自动生成为空，可以使用遍历表功能检查，如图1-8：

文本

描述已自动生成**图1-7**

**图1-8**

1. ListLength：求表长，直接返回并显示线性表的长度如图1-9：

文本

描述已自动生成**图1-9**

1. 文本

   描述已自动生成GetElem：获取数据元素的位置，先提示输入想要获取的数据元素的位序，如果该位序不小于1且在线性表表长范围内，则将该数据元素的值赋值给e并返回和显示e如图1-10，否则显示该数据元素不在范围内如图1-11，可通过遍历线性表进行验证如图1-12：

**图1-10**

文本

描述已自动生成

文本

描述已自动生成**图1-11**

**图1-12**

1. LocateElem：数据元素查找，输入要查找的数据元素，遍历整个线性表，若线性表内含有此数据元素，则将该数据元素的位置赋值给i并显示i的值如图1-13，否则显示数据元素查找失败，该数据元素不存在如图1-14，验证线性表内数据元素可参考图1-12：

文本

描述已自动生成文本

描述已自动生成**图1-13**

**图1-14**

1. PriorElem：获取前驱节点，输入一个数据元素e，如果数据元素e不为首元素（线性表第一个数据元素）且在线性表范围内，遍历将该数据元素的前驱元素赋值给pre，并返回OK显示pre的值如图1-15，否则返回ERROR显示该数据元素没有前驱节点如图1-16，线性表包含的数据元素请参考图1-12：

文本

描述已自动生成**图1-15**文本

描述已自动生成

**图1-16**

1. NextElem：获取后继节点，输入一个数据元素e，如果数据元素e不为尾元素（线性表最后一个数据元素）且在线性表范围内，遍历将该数据元素的后继元素赋值给next，并返回OK显示next的值如图1-17，否则返回ERROR显示该数据元素没有后继节点如图1-18，线性表包含的数据元素请参考图1-12：

文本

描述已自动生成文本

描述已自动生成**图1-17**

**图1-18**

1. ListInsert：插入数据元素，按顺序输入插入数据元素的位置，然后输入插入数据元素的值，若输入的位置合理（位置序号不小于1且在线性表的长度范围内），则在该数据元素位置插入新的数据元素并返回OK显示新数据元素插入成功且线性表长度加1如图1-19，否则返回ERROR并显示新数据元素插入失败如图1-20，产生的新线性表如图1-21，原线性表数据元素请参考图1-文本

   描述已自动生成12：

文本

描述已自动生成**图1-19**

**图1-20**

**图1-21**文本

描述已自动生成

1. 文本

   描述已自动生成ListDelete：删除数据元素，按输入删除数据元素的位置，若输入的位置合理（位置序号不小于1且在线性表的长度范围内），则删除该数据元素并返回OK显示删除数据元素成功且线性表长度减1如图1-22，否则返回ERROR并显示数据元素删除失败如图1-23，删除数据元素后的线性表如图1-24，原线性表数据元素请参考图1-21：

**图1-22**

**文本

描述已自动生成**文本

描述已自动生成**图1-23**

**图1-24**

1. ListTraverse：遍历线性表，将线性表内所有的数据元素节点的值按顺序输出如图1-25：

**图1-25**文本

描述已自动生成

* 1. **实验小结**（重点说明在实验中取得的实际经验，例如调试中碰到的典型错误等，不要写套话）

本次实验为数据结构首次实验。实验难度不算太难，但对于没有正式在课堂上学过C语言的我来说，很多公式函数表达式等都需要我自学和查找资料例子。我在实验中碰到了一些问题总会花费我很多时间去逐个攻破，问题如下：

1. 条件思考得不够充分，导致很多时候程序运行到一个地方的时候，无法判断出错的地方，例如判断线性表是否存在，存在才能继续下一步操作，不存在则报错。
2. 小地方检查不够仔细，导致一些线性表操作出错，如删除一个元素，并将线性表表长减1，指针指向的数据元素不正确，导致删除了错误的数据元素，后来修改过后，能够正确删除数据元素。

本次实验让我对线性表的顺序结构基本操作加深了印象，对用C语言实现线性表的顺序存储结构有了更深层次的理解，能够弥补我在C语言代码运用及计算方面的欠缺，让我从此实验中发现到更多关于线性表顺序存储结构的新知识。

2 基于链式存储结构的线性表实现

## 2.1 问题描述

通过使用线性表的链式存储结构，构造一个具有链表操作基础功能的菜单，实现链表的基本运算。在主函数里定义了链表的初始化表、销毁表、清空表、判定空表和求表长等基本运算对应的函数，且给出相应的操作提示。更简单，清晰地表示出线性表链式存储结构与随机存储数据元素功能。

## 2.1.1 单链表的基本操作及运算

依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等12种基本运算，具体运算功能定义如下。

1. ⑴初始化线性表：InitList(&L)

初始条件：线性表已存在或不存在；

操作结果：构造一个空的线性表L。

1. 销毁线性表：DestroyList(&L)

初始条件：线性表L已存在；

操作结果：销毁线性表L。

1. 清空线性表：ClearList(&L)

初始条件：线性表L已存在；

操作结果：将线性表L重置为空表。

1. 判断空表：ListEmpty(L)

初始条件：线性表L已存在；

操作结果：若线性表L为空表，则返回TRUE，否则返回FALSE。

1. 求表长：ListLength(L)

初始条件：线性表L已存在；

操作结果：返回线性表L中数据元素的个数。

1. 元素获取：GetElem(L,i,e)

初始条件：线性表L已存在，1≤i≤ListLength(L)；

操作结果：用e返回线性表L中第i个元素的值。

1. 元素查找：LocateElem(L,e)

初始条件：线性表L已存在；

操作结果：返回线性表L中第1个与e满足关系数据元素的位序。若这样的数据元素不存在，则返回值为0。

1. 获得前驱结点：PriorElem(L,e,&pre)

初始条件：线性表L已存在；

操作结果：若e是线性表L的数据元素，且不是第一个，则用pre返回它的前驱，否则操作失败，pre无定义。

1. 获得后继结点：NextElem(L,e,&next)

初始条件：线性表L已存在；

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

1. 插入元素：ListInsert(L,i,e)

初始条件：线性表L已存在，1≤i≤ListLength(L)+1；

操作结果：在线性表L中第i个位置之前插入新的数据元素e，线性表L的长度+1。

1. 删除元素：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：删除线性表L的第i个数据元素，并用e返回其值，线性表L的长度减1。

1. 遍历表：ListTraverse(L)

初始条件：线性表L已存在；

操作结果：依次对线性表L的每个数据元素进行扫描。如果数据元素不存在则返回线性表为空，否则按顺序返回线性表L中的每个元素。

## 2.2 系统设计

## 2.2.1 线性表链式存储结构（物理结构）

线性表的链式存储结构（单链表）：

#define LIST\_INIT\_SIZE 100

typedef struct LNode{ //单链表（链式结构）的定义

ElemType data; //定义数据域

Struct LNode\* next; //定义指针域

}LNode, \*LinkList;

定义一个数组结构就可以实现对多个链表的管理。

## 2.2.2 结构系统设计

用一个while循环语句实现线性表的12个基本功能，使用12个case功能语句分别调用12个线性表基本功能的函数并在case里对链表进行判断是否已存在，再通过使用一个switch语句将12个case语句组合成一个菜单栏实现对单链表基本功能的操作选择。

## 2.2.3 算法设计

链表运算算法设计与思想：

1. InitList(&L)

设计思想： 创建一个表头结点的空间并且将data置为0，下一个指针指向空；链表L若已存在，显示创建失败。

1. DestroyList(&L)

设计思想： 遍历整个链表并释放每个节点的存储空间，最后将表头结点设置为空；若链表L不存在，则销毁失败。

1. ClearList(&L)

设计思想： 遍历整个链表并释放每个节点的存储空间并将表头结点L的下一个节点设置为空。

1. ListEmpty(L)

设计思想：判断链表的表头结点是否为空，是则返回TRUE，否则返回FALSE。

1. ListLength(L)

设计思想：若链表L已存在，则创建一个计数变量i，遍历整个线性表，每当扫描到一个节点元素则执行i+1，最后返回i的值。

1. GetElem(L,i,&e)

设计思想：判断链表L是否为空，若为空显示数据元素获取失败，否则判断输入值i是否存在链表L的数据位置范围内，然后遍历整个链表直到第i个节点并将第i个节点的值赋值给e，返回OK；否则返回ERROR。

1. LocateElem(L,e)

设计思想：遍历链表L，若数据元素e在链表L中，返回e在链表L中的位置序号，否则显示查找失败。

1. PriorElem(L,e,&pre)

设计思想： 遍历链表，若当前节点的next节点数据元素与所寻找的数据元素相等，则将当前节点的data值赋值给pre并返回OK；否则返回ERROR。

1. NextElem(L,e,&next)

设计思想： 遍历链表，若当前节点的数据元素与寻找的数据元素相等，并且next节点不为空，则将next节点的data值赋值给next并返回OK；否则返回ERROR。

1. ListInsert(&L,i,e)

设计思想： 创建一个新节点q，将e的值赋值给q->data，遍历链表，在第i个节点前插入q->data并返回OK，如果i的值超过了链表的存储大小，则返回ERROR。

1. ListDelete(&L,i,&e)

设计思想： 创建一个节点q，从表头节点开始遍历整个链表直到第i个节点停止并且将第i个节点的值赋予给节点q存储，若i的值超出链表的存储空间大小，则返回ERROR，否则执行删除操作且返回OK。

1. ListTraverse(L)

设计思想： 若链表表长为空，则返回线性表是空表并返回菜单， 否则利用while循环语句遍历整个链表并依次输出每个节点的data值。

## 2.3 系统实现

## 2.3.1 主要函数实现

1.头文件：

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

2.常量定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

3.表达式类型：

typedef int status;

typedef int ElemType;

4.函数实现：

1. InitList：给单链表L分配存储空间，若存储空间分配成功，将L的data域置为0，L的next域置为空并返回OK；否则返回ERROR；
2. DestroyList： 定义两个LinkList类型的指针p和q，p指向表头结点L，q为p的next节点，用p和q遍历表并释放每一个节点的存储空间； 最后将表头结点L置为空并返回OK；
3. ClearList：定义两个节点指针p和q，q为p的next节点，用p和q遍历表并释放每一个节点的存储空间，最后把表头节点L的next节点设置为空；
4. ListEmpty：判断表头节点L的next节点是否为空，是则返回TRUE，否则返回FALSE。
5. ListLength：创建一个计数变量名i和指向L->next节点的指针p，用p指针遍历表的每一个节点元素，每当p指针遍历到一个节点时将i+1，最后返回i的值。
6. GetElem：如果线性表存在，判断输入i是否在链表范围内，若在范围内则创立一个指针p指向链表L的next节点并用p指针遍历链表L直到第i个节点元素，将该节点元素的data值赋值给e并返回OK；否则返回ERROR。
7. LocateElem：创建一个计数变量i从1开始计数，同时定义一个节点p指向L->next，用p指针遍历链表中每个节点元素同时i+1，当p->next等于e时停下，若e存在链表L中则返回i，否则返回ERROR。
8. PriorElem：定义两个指针p和q，p指向L->next，q指向L的头节点； 用q和p遍历整个链表，如果p->next等于输入值e时，将它的前驱节点q->next的值赋值给pre并返回OK，否则返回ERROR。
9. NextElem：定义一个指针p指向链表L的头节点的下一个节点L->next，用p遍历链表，如果p->next等于输入值e时，将它的后继节点p->next的data值赋值给next并返回OK，否则返回ERROR。
10. ListInsert：定义一个指针p指向链表头节点L并定义一个计数变量j，用p指针遍历表中每一个节点元素时j+1位，定义一个指针q并将e赋值给q->data，当遍历表遍历到第i个节点时（i在链表L范围内）执行插入操作，否则返回ERROR。
11. ListDelete：定义一个指针q用于存储被删除的节点的值，同时定义一个指针p指向链表头节点L，用p遍历链表直到出现第i个节点出现后执行删除第i个节点并释放该节点的空间，否则返回ERROR。
12. ListTrabverse：若链表不存在则返回INFEASTABLE，否则定义一个p指针指向头节点L的下一个节点next，用p遍历整个链表并以此按顺序输出链表内每个节点的数据元素值。

## 2.3.2 源代码

（详细源代码见附录B）

## 2.4 系统测试

* 开发环境：Windows 10
* 编译器：Visual Studio 2019
* **文本

  描述已自动生成**在程序的开始时，会提示用户选择一个线性表进行操作，如图2-1：

**图2-1**

* 文本

  描述已自动生成选择完线性表后，便会出现一个主菜单，给予用户选择对线性表的操作如图2-2：

**图2-2**

1. 文本

   描述已自动生成InitList：初始化链表，若单链表不存在，则使用malloc函数构造一个新的空单链表并显示创建成功如图2-3，否则显示单链表创建失败；若单链表已存在，则提示单链表已存在，请清除原有单链表后再进行初始化单链表操作如图2-4：

文本

描述已自动生成**图2-3**

**图2-4**

1. 文本

   描述已自动生成DestroyList：销毁链表，若单链表已存在，则用指针p遍历整个单链表的所有节点元素，并用free函数将p指针遍历到的所有节点元素的存储空间释放，然后将表头节点L置为空并显示单链表销毁成功如图2-5，否则提示单链表不存在或单链表删除失败如图2-6：

电脑屏幕的截图

描述已自动生成**图2-5**

**图2-6**

1. 文本

   描述已自动生成ClearList：清空链表，如果单链表存在则用free函数将链表中所有节点的存储空间释放，若链表已清空则显示单链表清空成功如图2-7，否则显示单链表清空失败或单链表不存在：

**图2-7**

1. 文本

   描述已自动生成ListEmpty：判断链表是否为空，如果链表的头节点的下一个节点next为空，则显示单链表为空如图2-8，否则显示单链表不为空，可以使用遍历表的功能进行验证如图2-9：

文本

描述已自动生成**图2-8**

**图2-9**

1. 文本

   描述已自动生成ListLength：求表长，通过定义一个指针p指向L->next和创建一个计数变量i计算链表中含有数据元素的节点个数，用指针p每遍历到表中一个节点元素时进行i+1，在最后将i的值返回如图2-10，表中节点元素如图2-11：

文本

描述已自动生成**图2-10**

**图2-11**

1. 文本

   描述已自动生成GetElem：获取数据元素的位置，先提示输入想要获取的数据元素的位序i，如果i在单链表的范围内，则将单链表中第i个元素的值赋值给e并显示e的值如图2-12，否则显示该数据元素不在范围内如图2-13：

**图2-12**

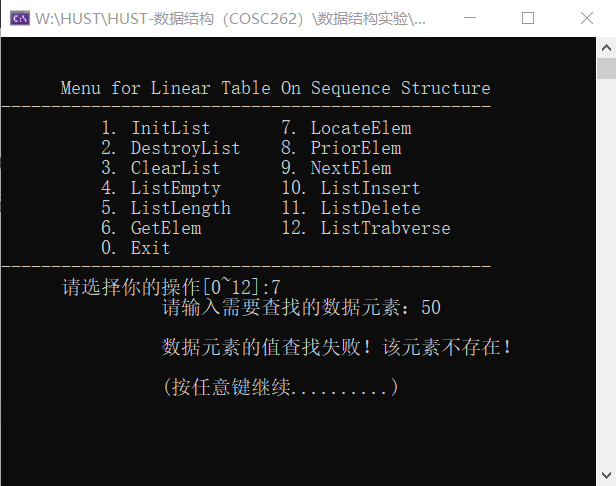
**文本

描述已自动生成图2-13**

1. 文本

   描述已自动生成LocateElem：数据元素查找，输入要查找的数据元素e，遍历整个链表，若链表内含有此数据元素，则将该数据元素的位序赋值给i并显示i的值如图2-14，否则显示数据元素查找失败，该数据元素不存在如图2-15：

**图2-14**

**图2-15**

1. PriorElem：获取前驱节点，输入一个数据元素e，如果数据元素e不为首节点元素（单链表表第一个节点元素）且在链表范围内，遍历链表将该节点元素的前驱节点元素赋值给pre，并显示pre的值如图2-16，否则显示该数据元素没有前驱节点如图2-17，单链表包含的节点元素请参考图2-11：

**图2-16**

**图2-17**

1. 文本

   描述已自动生成NextElem：获取后继节点，输入一个数据元素e，如果数据元素e不为尾节点元素（单链表表最后一个节点元素）且在链表范围内，遍历链表将该节点元素的后继节点元素赋值给next，并显示next的值如图2-18，否则显示该数据元素没有后继节点如图2-19，单链表包含的节点元素请参考图2-11：

**图2-18**

文本

描述已自动生成

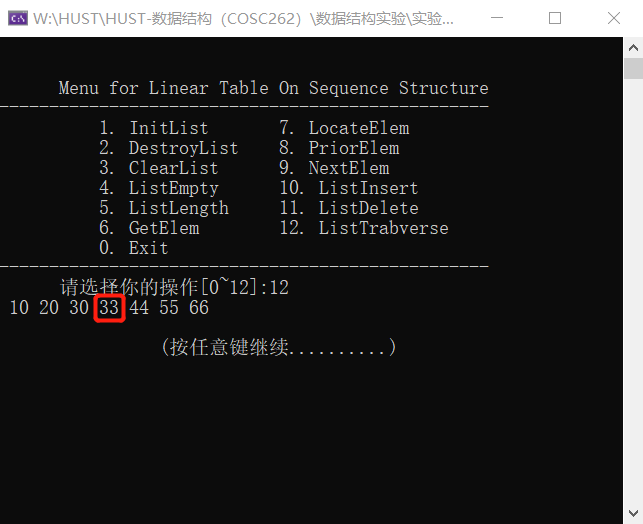
**图2-19**

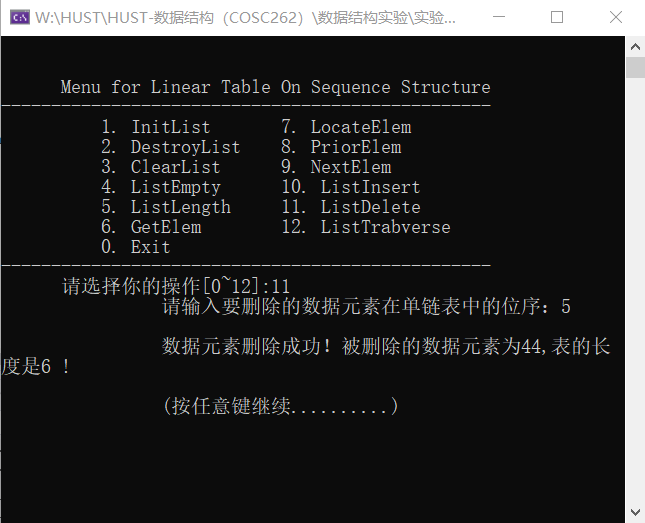
1. ListInsert：插入数据元素，输入要插入数据元素的节点位序i，然后输入插入数据元素的值，若单链表L的节点数据元素长度大于单链表L的存储空间，则给单链表L重新分配存储空间然后判断插入位置是否合理，若合理再将数据元素e插入到第i个节点元素之前，显示新数据元素插入成功以及单链表长度+1如图2-20，否则显示数据元素插入失败如图2-21，新单链表数据元素如图2-22，原单链表数据元素请参考图2-11：

文本

描述已自动生成**图2-20**

**图2-21**

**图2-22**

1. ListDelete：删除数据元素，输入删除数据元素的节点位序i，若输入的位置合理（在单链表范围内），则将第i个节点的数据元素的值赋值给e并将该节点的数据元素删除，然后显示删除数据元素成功且单链表长度减1如图2-23，否则显示数据元素删除失败如图2-24，删除数据元素后的单链表数据元素如图2-25，原单链表数据元素请参考图2-22：

文本

描述已自动生成**图2-23**

文本

描述已自动生成**图2-24**

**图2-25**

1. ListTraverse：遍历单链表，将单链表内所有的数据元素节点的值按顺序输出如图2-26：

**文本

描述已自动生成图2-26**

## 2.5 实验小结

本次实验为数据结构第二次实验。有了第一次实验的锻炼以后，能够更熟练更高效地完成第二次实验“单链表的基础功能操作”。但有时候还是会因为粗心大意而遇到问题，例如：

1. 当判断条件不满足时，单链表的操作结果就会截然不同，例如在写获取前驱节点元素时，因为没给判断指针q的data域是否等于0，若等于0则返回ERROR，导致后面当遇到查询0的时候，返回的是乱码。
2. 容易将指针指向的next域和data域混淆，这样就导致了代码报错，返回不出单链表的操作结果，经仔细检查逐个判断测试后，data域指的是当前指针的值而next域指的是下一个数据元素的指针。

经过这次的实验，让我对线性表的链式存储结构基本操作更加熟练运用，同时让我明白了线性表的链式存储结构要比线性表顺序存储结构运算更加快捷，更加具有高效性，因为单链表在插入和删除数据元素时不需要进行大量元素的移动。

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

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

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

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

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

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct { //顺序表（顺序结构）的定义

ElemType\* elem;

int length;

int listsize;

}SqList;

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

status InitList(SqList& L)

// 线性表L不存在，构造一个空的线性表，返回OK，否则返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

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

if (!L.elem)

return INFEASTABLE;

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

return OK;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status DestroyList(SqList& L)

// 如果线性表L存在，销毁线性表L，释放数据元素的空间，返回OK，否则返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L.elem) {

L.length = 0;

L.listsize = 0;

free(L.elem);

L.elem = NULL;

return OK;

}

else

{

return INFEASTABLE;

}

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ClearList(SqList& L)

// 如果线性表L存在，删除线性表L中的所有元素，返回OK，否则返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L.elem)

{

L.length = 0;

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListEmpty(SqList L)

// 如果线性表L存在，判断线性表L是否为空，空就返回TRUE，否则返回FALSE；如果线性表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if(L.elem)

{

if(L.length == 0)

{

return TRUE;

}

return FALSE;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListLength(SqList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L.elem)

{

return L.length;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status GetElem(SqList L, int i, ElemType& e)

// 如果线性表L存在，获取线性表L的第i个元素，保存在e中，返回OK；如果i不合法，返回ERROR；如果线性表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L.elem)

{

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

{

return ERROR;

}

e = L.elem[i];

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status LocateElem(SqList L, ElemType e)

// 如果线性表L存在，查找元素e在线性表L中的位置序号并返回OK；如果e不存在，返回ERROR；当线性表L不存在时，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L.elem)

{

int i = 1;

while (i < L.length + 1 && L.elem[i] != e)

i += 1;

if (i >= L.length+1)

{

return ERROR;

}

return i;

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status PriorElem(SqList L, ElemType e, ElemType& pre)

// 如果线性表L存在，获取线性表L中元素e的前驱，保存在pre中，返回OK；如果没有前驱，返回ERROR；如果线性表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L.elem)

{

int i = 1;

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

{

i++;

}

if (e == L.elem[1] || i >= L.length+1)

return ERROR;

pre = L.elem[i - 1];

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status NextElem(SqList L, ElemType e, ElemType& next)

// 如果线性表L存在，获取线性表L元素e的后继，保存在next中，返回OK；如果没有后继，返回ERROR；如果线性表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L.elem)

{

int i = LocateElem(L,e);

if (i != 0 && i != L.length)

{

next = L.elem[i + 1];

return OK;

}

return ERROR;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListInsert(SqList& L, int i, ElemType e)

// 如果线性表L存在，将元素e插入到线性表L的第i个元素之前，返回OK；当插入位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L.elem)

{

int a;

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

return ERROR;

if (L.length >= L.listsize)

{

ElemType\* newbase;

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

if (!newbase)

return OVERFLOW;

L.elem = newbase;

L.listsize = LISTINCREMENT;

}

for (a = L.length; a >= i - 1; a--)

L.elem[a + 1] = L.elem[a];

L.elem[i] = e;

L.length++;

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListDelete(SqList& L, int i, ElemType& e)

// 如果线性表L存在，删除线性表L的第i个元素，并保存在e中，返回OK；当删除位置不正确时，返回ERROR；如果线性表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L.elem)

{

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

return ERROR;

int\* p = &(L.elem[i]);

e = \*p;

int \*q = L.elem + L.length;

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

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

L.length--;

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListTraverse(SqList L)

// 如果线性表L存在，依次显示线性表中的元素，每个元素间空一格，返回OK；如果线性表L不存在，返回INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L.elem)

{

int i;

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

{

if (i == L.length+1)

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

else

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

}

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

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

void main(void) {

SqList L; int op = 1;

int u = 1;

int e, i, pre\_e, next\_e, current\_e, elem;

printf("请输入对哪一个线性表进行操作1-99，若输入0则退出程序！\n");

scanf("%d", &u);

if (u == 0)

return;

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. ListTrabverse\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~12]:");

scanf("%d", &op);

switch (op) {

case 1:

//printf("\n----IntiList功能待实现！\n");

if (InitList(L) == OK)

printf("\t\t线性表创建成功！\n");

else

{

printf("\t\t给线性表动态分配空间失败！\n");

printf("\t\t线性表创建失败！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 2:

if (L.elem)

{

if (DestroyList(L) == OK)

printf("\t\t线性表删除成功！\n");

else

printf("\t\t线性表删除失败！\n");

}

else

{

printf("\t\t线性表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 3:

if (L.elem)

{

if (ClearList(L) == OK)

printf("\t\t线性表清空成功！\n");

else

printf("\t\t线性表清空失败！\n");

}

else

{

printf("\t\t线性表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 4:

if (L.elem)

{

if (ListEmpty(L) == TRUE)

printf("\t\t此线性表为空！\n");

else

printf("\t\t此线性表不为空！\n");

}

else

{

printf("\t\t线性表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 5:

if(L.elem)

printf("\t\t线性表的长度为：%d\t\n", ListLength(L));

else

printf("\t\t线性表不存在！\n");

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 6:

if (L.elem)

{

printf("\t\t请输入想要获得的数据元素的位序：");

scanf("%d", &i);

if (GetElem(L, i, e) == ERROR)

printf("\n\t\t该数据元素不在范围内！\n");

else

printf("\n\t\t线性表L中第%d个数据元素的值是：%d\n", i, e);

}

else

{

printf("\t\t线性表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 7:

if (L.elem)

{

printf("\t\t请输入需要查找的数据元素：");

scanf("%d", &e);

if (LocateElem(L, e) == ERROR)

printf("\n\t\t数据元素的值查找失败！该元素不存在！\n");

else

printf("\n\t\t该数据元素值在线性表中为第%d个元素！\n", LocateElem(L, e));

}

else

{

printf("\t\t线性表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 8:

if (L.elem)

{

printf("\t\t请输入一个数据元素用于查找此前驱结点元素：");

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

if (PriorElem(L, current\_e, pre\_e) == ERROR)

printf("\n\t\t该数据元素没有前驱结点元素！\n");

else

printf("\n\t\t该数据元素%d的前驱结点元素为：%d\n", current\_e, pre\_e);

}

else

{

printf("\t\t线性表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 9:

if (L.elem)

{

printf("\t\t请输入一个数据元素用于查找此后继结点元素：");

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

if (NextElem(L, current\_e, next\_e) == ERROR)

printf("\n\t\t该数据元素没有后继结点元素！\n");

else

printf("\n\t\t该数据元素%d的后继结点元素为：%d\n", current\_e, next\_e);

}

else

{

printf("\t\t线性表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 10:

if (L.elem)

{

printf("\t\t请输入插入数据元素的位置：");

scanf("%d", &i);

getchar();

printf("\t\t请输入插入数据元素的值：");

scanf("%d", &e);

getchar();

if (ListInsert(L, i, e) == ERROR)

printf("\n\t\t新数据元素插入失败！\n");

else

printf("\n\t\t新数据元素插入成功！线性表的长度为：%d\n", L.length);

}

else

{

printf("\t\t线性表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 11:

if (L.elem)

{

printf("\t\t请输入要删除的数据元素在线性表中的位序：");

scanf("%d", &i);

if (ListDelete(L, i, e) == ERROR)

printf("\n\t\t数据元素删除失败！\n");

else

printf("\n\t\t数据元素删除成功！被删除的数据元素为%d,表的长度是%d !\n", e, L.length);

}

else

{

printf("\t\t线性表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 12:

//printf("\n----ListTrabverse功能待实现！\n");

if (ListTraverse(L))

{

if (L.length == 0)

printf("\t\t线性表是空表！\n");

}

else

{

printf("\t\t线性表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

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

# 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 0

#define INFEASTABLE -1

#define OVERFLOW -2

typedef int status;

typedef int ElemType; //数据元素类型定义

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

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct LNode{ //单链表（链式结构）的定义

ElemType data;

struct LNode\* next;

}LNode,\*LinkList;

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

status InitList(LinkList& L)

// 单链表L不存在，构造一个空的单链表，返回OK，否则返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (!L)

{

L = (LinkList)malloc(sizeof(LNode));

if (!L)

{

return ERROR;

}

L->data = 0;

L->next = 0;

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status DestroyList(LinkList& L)

// 如果单链表L存在，销毁单链表L，释放数据元素的空间，返回OK，否则返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L)

{

LinkList p, q;

p = L;

while (p)

{

q = p->next;

free(p);

p = q;

}

L = NULL;

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ClearList(LinkList& L)

// 如果单链表L存在，删除单链表L中的所有元素，返回OK，否则返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L)

{

LinkList p, q;

p = L->next;

while (p)

{

q = p;

p = p->next;

free(q);

}

L->next = 0;

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListEmpty(LinkList L)

// 如果单链表L存在，判断单链表L是否为空，空就返回TRUE，否则返回FALSE；如果单链表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L)

{

if (L->next == 0)

return TRUE;

return FALSE;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

int ListLength(LinkList L)

// 如果单链表L存在，返回单链表L的长度，否则返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L)

{

LinkList p = L->next;

int i = 0;

while (p)

{

i++;

p = p->next;

}

return i;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status GetElem(LinkList L, int i, ElemType& e)

// 如果单链表L存在，获取单链表L的第i个元素，保存在e中，返回OK；如果i不合法，返回ERROR；如果单链表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L)

{

LinkList p = L->next;

int j = 1;

while (p && j < i)

{

p = p->next;

j++;

}

if (!p || j > i)

return ERROR;

e = p->data;

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status LocateElem(LinkList L, ElemType e)

// 如果单链表L存在，查找元素e在单链表L中的位置序号；如果e不存在，返回ERROR；当单链表L不存在时，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L)

{

int i = 1;

LinkList p = L->next;

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

{

p = p->next;

i++;

}

if (!p)

return ERROR;

return i;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status PriorElem(LinkList L, ElemType e, ElemType& pre)

// 如果单链表L存在，获取单链表L中元素e的前驱，保存在pre中，返回OK；如果没有前驱，返回ERROR；如果单链表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L)

{

LinkList p = L->next;

LinkList q = L;

if (e == 1)

return ERROR;

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

{

p = p->next;

q = q->next;

}

if (!p || q->data == 0)

return ERROR;

p = q;

pre = q->data;

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status NextElem(LinkList L, ElemType e, ElemType& next)

// 如果单链表L存在，获取单链表L元素e的后继，保存在next中，返回OK；如果没有后继，返回ERROR；如果单链表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L)

{

LinkList p = L->next;

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

{

p = p->next;

}

if (!p || p->next == 0)

return ERROR;

next = p->next->data;

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListInsert(LinkList& L, int i, ElemType e)

// 如果单链表L存在，将元素e插入到单链表L的第i个元素之前，返回OK；当插入位置不正确时，返回ERROR；如果单链表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L)

{

LinkList p = L;

int j = 0;

while (p && j < i - 1)

{

p = p->next;

++j;

}

if (!p || p == 0 || i <= 0)

return ERROR;

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

q->data = e;

q->next = p->next;

p->next = q;

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListDelete(LinkList& L, int i, ElemType& e)

// 如果单链表L存在，删除单链表L的第i个元素，并保存在e中，返回OK；当删除位置不正确时，返回ERROR；如果单链表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L)

{

LinkList p = L;

int j = 0;

while (p && j < i - 1)

{

p = p->next;

j++;

}

if (!p || i - 1 < j || p == 0)

return ERROR;

LinkList q = p;

e = q->next->data;

q = p->next;

p->next = q->next;

free(q);

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

status ListTraverse(LinkList L)

// 如果单链表L存在，依次显示单链表中的元素，每个元素间空一格，返回OK；如果单链表L不存在，返回INFEASTABLE。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

if (L)

{

LinkList p = L->next;

while (p)

{

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

p = p->next;

}

printf("\n");

return OK;

}

return INFEASTABLE;

/\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

}

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

void main(void) {

LinkList L = 0; int op = 1;

int u = 1;

int i;

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

printf("请输入对哪一个单链表进行操作1-99，若输入0则退出程序！\n");

scanf("%d", &u);

if (u == 0)

return;

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. ListTrabverse\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~12]:");

scanf("%d", &op);

switch (op) {

case 1:

//printf("\n----IntiList功能待实现！\n");

if (!L)

{

if (InitList(L) == OK)

printf("\t\t单链表创建成功！\n");

else

{

printf("\t\t单链表创建失败！\n");

}

}

else

{

printf("\t\t单链表已存在，请清除原有单链表后再进行初始化单链表操作！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 2:

if (L)

{

if (DestroyList(L) == OK)

printf("\t\t单链表删除成功！\n");

else

printf("\t\t单链表删除失败！\n");

}

else

{

printf("\t\t单链表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 3:

if (L)

{

if (ClearList(L) == OK)

printf("\t\t单链表清空成功！\n");

else

printf("\t\t单链表清空失败！\n");

}

else

{

printf("\t\t单链表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 4:

if (L)

{

if (ListEmpty(L) == TRUE)

printf("\t\t此单链表为空！\n");

else

printf("\t\t此单链表不为空！\n");

}

else

{

printf("\t\t单链表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 5:

if (L)

printf("\t\t单链表的长度为：%d\t\n", ListLength(L));

else

printf("\t\t单链表不存在！\n");

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 6:

if (L)

{

printf("\t\t请输入想要获得的数据元素的位序：");

scanf("%d", &i);

if (GetElem(L, i, e) == ERROR)

printf("\n\t\t该数据元素不在范围内！\n");

else

printf("\n\t\t单链表L中第%d个数据元素的值是：%d\n", i, e);

}

else

{

printf("\t\t单链表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 7:

if (L)

{

printf("\t\t请输入需要查找的数据元素：");

scanf("%d", &e);

if (LocateElem(L, e) == ERROR)

printf("\n\t\t数据元素的值查找失败！该元素不存在！\n");

else

printf("\n\t\t该数据元素值在单链表中为第%d个元素！\n", LocateElem(L, e));

}

else

{

printf("\t\t单链表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 8:

if (L)

{

printf("\t\t请输入一个数据元素用于查找此前驱结点元素：");

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

if (PriorElem(L, current\_e, pre\_e) == ERROR)

printf("\n\t\t该数据元素没有前驱结点元素！\n");

else

printf("\n\t\t该数据元素%d的前驱结点元素为：%d\n", current\_e, pre\_e);

}

else

{

printf("\t\t单链表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 9:

if (L)

{

printf("\t\t请输入一个数据元素用于查找此后继结点元素：");

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

if (NextElem(L, current\_e, next\_e) == ERROR)

printf("\n\t\t该数据元素没有后继结点元素！\n");

else

printf("\n\t\t该数据元素%d的后继结点元素为：%d\n", current\_e, next\_e);

}

else

{

printf("\t\t单链表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 10:

if (L)

{

printf("\t\t请输入插入数据元素的位置：");

scanf("%d", &i);

getchar();

printf("\t\t请输入插入数据元素的值：");

scanf("%d", &e);

getchar();

if (ListInsert(L, i, e) == ERROR)

printf("\n\t\t新数据元素插入失败！\n");

else

printf("\n\t\t新数据元素插入成功！单链表的长度为：%d\n", ListLength(L));

}

else

{

printf("\t\t单链表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 11:

if (L)

{

printf("\t\t请输入要删除的数据元素在单链表中的位序：");

scanf("%d", &i);

if (ListDelete(L, i, e) == ERROR)

printf("\n\t\t数据元素删除失败！\n");

else

printf("\n\t\t数据元素删除成功！被删除的数据元素为%d,表的长度是%d !\n", e, ListLength(L));

}

else

{

printf("\t\t单链表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 12:

//printf("\n----ListTrabverse功能待实现！\n");

if (L)

{

if (ListTraverse(L))

{

if (ListLength(L) == 0)

printf("\t\t单链表是空表！\n");

}

else

{

printf("\t\t单链表不存在！\n");

}

}

else

{

printf("\t\t单链表不存在！\n");

}

getchar();

printf("\n\t\t(按任意键继续..........)\n");

getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

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

# 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