

**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201903**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期： 2020年 11月17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 4](#_Toc58965114)

[1.1 问题描述 4](#_Toc58965115)

[1.1.1 12个基本函数需求实现 4](#_Toc58965116)

[1.1.2 演示系统 5](#_Toc58965117)

[1.1.3 文件读写功能 5](#_Toc58965118)

[1.2 系统设计 5](#_Toc58965119)

[1.2.1 系统总体设计 5](#_Toc58965120)

[1.2.2 算法设计 7](#_Toc58965121)

[1.3 系统实现 11](#_Toc58965122)

[1.3.1 程序实现环境 11](#_Toc58965123)

[1.3.2 头文件及预定义说明 11](#_Toc58965124)

[**1.4 系统测试** 11](#_Toc58965125)

[1.5 实验小结 21](#_Toc58965126)

[2基于链式存储结构的线性表实现 22](#_Toc58965127)

[2.1 问题描述 22](#_Toc58965128)

[2.1.1 12个基本函数需求实现 22](#_Toc58965129)

[2.1.2 演示系统 23](#_Toc58965130)

[2.1.3 文件读写功能 23](#_Toc58965131)

[2.2 系统设计 23](#_Toc58965132)

[2.2.1 系统总体设计 23](#_Toc58965133)

[2.2.2 算法设计 25](#_Toc58965134)

[2.3 系统实现 29](#_Toc58965135)

[2.3.1 程序实现环境 29](#_Toc58965136)

[2.3.2 头文件及预定义说明 29](#_Toc58965137)

[**2.4 系统测试** 30](#_Toc58965138)

[2.5 实验小结 34](#_Toc58965139)

[3基于二叉链表的二叉树实现 36](#_Toc58965140)

[3.1 问题描述 36](#_Toc58965141)

[3.1.1 基本函数需求实现 36](#_Toc58965142)

[3.1.2 演示系统 37](#_Toc58965143)

[3.1.3 文件读写功能 38](#_Toc58965144)

[3.2 系统设计 38](#_Toc58965145)

[3.2.1 系统总体设计 38](#_Toc58965146)

[3.2.2 算法设计 39](#_Toc58965147)

[3.3 系统实现 43](#_Toc58965148)

[3.3.1 程序实现环境 43](#_Toc58965149)

[3.3.2 头文件及预定义说明 43](#_Toc58965150)

[**3.4 系统测试** 44](#_Toc58965151)

[3.5 实验小结 49](#_Toc58965152)

[4基于邻接表的图实现 50](#_Toc58965153)

[4.1 问题描述 50](#_Toc58965154)

[4.1.1 基本函数需求实现 50](#_Toc58965155)

[4.1.2 演示系统 51](#_Toc58965156)

[4.1.3 文件读写功能 51](#_Toc58965157)

[4.2 系统设计 52](#_Toc58965158)

[4.2.1 系统总体设计 52](#_Toc58965159)

[4.2.2 算法设计 53](#_Toc58965160)

[4.3 系统实现 56](#_Toc58965161)

[4.3.1 程序实现环境 56](#_Toc58965162)

[4.3.2 头文件及预定义说明 56](#_Toc58965163)

[**4.4 系统测试** 57](#_Toc58965164)

[4.5 实验小结 63](#_Toc58965165)

[参考文献 64](#_Toc58965166)

[附录A 基于顺序存储结构线性表实现的源程序 65](#_Toc58965167)

[附录B 基于链式存储结构线性表实现的源程序 79](#_Toc58965168)

[附录C 基于二叉链表二叉树实现的源程序 94](#_Toc58965169)

[附录D 基于邻接表图实现的源程序 113](#_Toc58965170)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

通过实验达到⑴加深对线性表的概念、基本运算的理解；⑵熟练掌握线性表的逻辑结构与物理结构的关系；⑶物理结构采用顺序表,熟练掌握线性表的基本运算的实现。

## 1.1.1 12个基本函数需求实现

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。

⑴初始化表：函数名称是InitList(L)；初始条件是线性表L不存在；操作结果是构造一个空的线性表。

⑵销毁表：函数名称是DestroyList(L)；初始条件是线性表L已存在；操作结果是销毁线性表L。

⑶清空表：函数名称是ClearList(L)；初始条件是线性表L已存在；操作结果是将L重置为空表。

⑷判定空表：函数名称是ListEmpty(L)；初始条件是线性表L已存在；操作结果是若L为空表则返回TRUE,否则返回FALSE。

⑸求表长：函数名称是ListLength(L)；初始条件是线性表已存在；操作结果是返回L中数据元素的个数。

⑹获得元素：函数名称是GetElem(L,i,e)；初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用e返回L中第i个数据元素的值。

⑺查找元素：函数名称是LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回L中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。

⑻获得前驱：函数名称是PriorElem(L,e,pre)；初始条件是线性表L已存在；操作结果是若e是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。

⑼获得后继：函数名称是NextElem(L,e,next)；初始条件是线性表L已存在；操作结果是若e是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。

⑽插入元素：函数名称是ListInsert(L,i,e)；初始条件是线性表L已存在，1≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

⑿遍历表：函数名称是ListTrabverse (L);初始条件是线性表L已存在；操作结果是依次输出L的每个数据元素。

## 1.1.2 演示系统

演示系统采用菜单形式呈现，有两层目录，实现多个线性表管理。欢迎菜单有包括退出系统、进入次级菜单的五个操作，次级菜单实现了顺序表的12个基本操作，以及文件写入、读取功能，还有返还初级菜单的功能。

## 1.1.3 文件读写功能

除了12种顺序表基本操作，本实验还新增了写入文件以及读取文件的功能

(1)写入文件：函数名称是SaveList(L,FileName);初始条件是线性表L存在；操作结果是把顺序表L中的数据按顺序写入指定文件中。

(2)读取文件：函数名称是LoadList(L,FileName);初始条件是存在指定文件；操作结果是把指定文件中的数据存入顺序表中。

## 1.2 系统设计

## 1.2.1 系统总体设计

系统main函数中switch作为主干，循环运行。

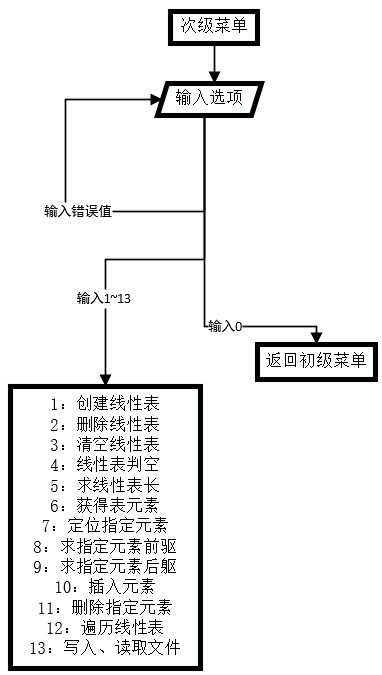
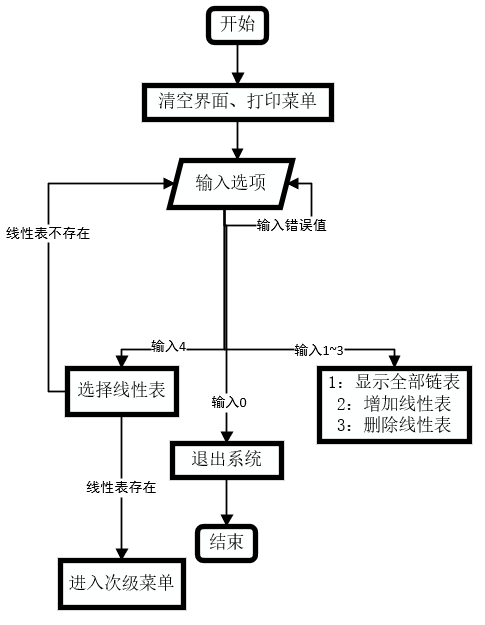


图1-2-1 系统总体设计思路图

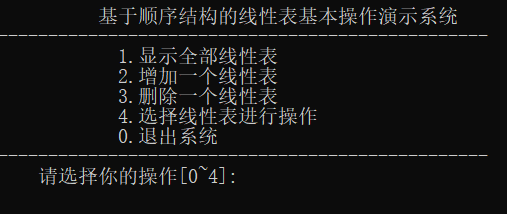


图1-2-2 初级菜单图

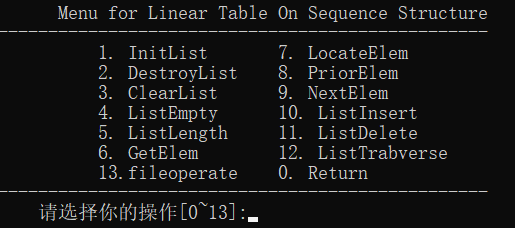


图1-2-3 次级菜单图

## 1.2.2 算法设计

⑴初始化表：status InitList(SqList \*L)；线性表L存在时，返回INFEASIBLE；线性表不存在时，为L.elem分配空间并将L.length置零，L.listsize设为LIST\_INIT\_SIZE，返回OK。

时空复杂度分析：T(n)=O(1);S(n)=O(1).

⑵销毁表：status DestroyList(SqList \*L);线性表L不存在时，返回INFEASIBLE；线性表L存在时，释放L.elem，并使L.elem指向NULL，L.length归零，返回OK。

时空复杂度分析：T(n)=O(1);S(n)=O(1).

⑶清空表：status ClearList(SqList \*L);线性表L不存在时，返回INFEASIBLE；线性表L存在时，将L.elem指向NULL，并将L.length归零，返回OK。

时空复杂度分析：T(n)=O(1);S(n)=O(1).

⑷判定空表：status ListEmpty(SqList L)；线性表L不存在时，返回INFEASIBLE;线性表L存在时，利用L.length判断线性表是否为空，为空时返回OK；否则返回ERROR。

时空复杂度分析：T(n)=O(1);S(n)=O(1).

⑸求表长：int ListLength(SqList L)；线性表L不存在时，返回INFEASIBLE；线性表存在时，返回L.length。

时空复杂度分析：T(n)=O(1);S(n)=O(1).

⑹获得元素：status GetElem(SqList L,int i,ElemType \*e)；输入线性表L、元素序号i、存储元素变量e。线性表L不存在时，返回INFEASBILE；线性表存在时，若i不符合条件（1≤i≤ListLength(L)），返回ERROR；否则，将目标元素赋值给e，返回OK。

时空复杂度分析：T(n)=O(1);S(n)=O(1).

⑺查找元素：status LocateElem(SqList L,ElemType e)；线性表L不存在时，返回INFEASIBLE；线性表存在时，循环比较L.elem中元素与e，查找到时返回位置序号；未查找到时返回ERROR。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑻获得前驱：status PriorElem(SqList L,ElemType e,ElemType \*pre)；

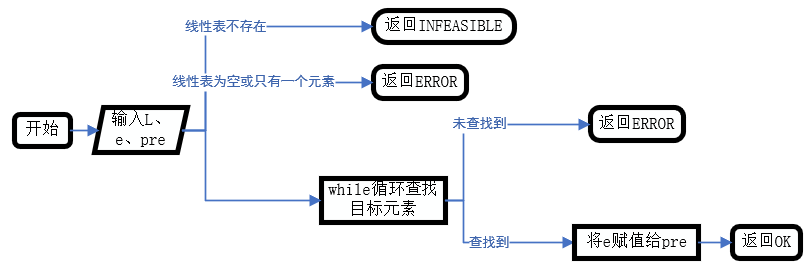


图1-2-4 获得前驱算法设计图

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑼获得后继：status NextElem(SqList L,ElemType e,ElemType \*next)；与获得前驱操作近同，只是将边缘条件更改，此处略过。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑽插入元素：status ListInsert(SqList \*L,int i,ElemType e)；

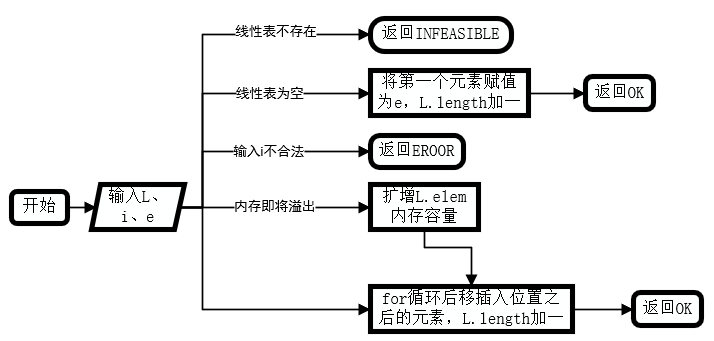


图1-2-5 插入元素算法设计图

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑾删除元素：status ListDelete(SqList \*L,int i,ElemType \*e)；

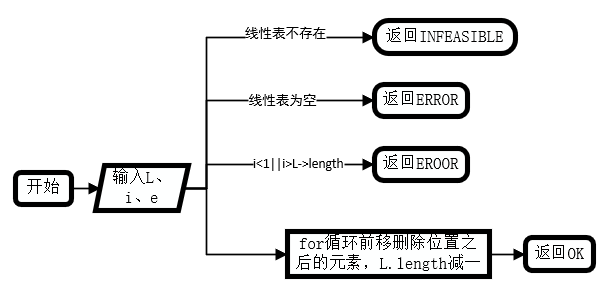


图1-2-6 删除元素算法设计图

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑿遍历表：status ListTrabverse(SqList L);依次输出L.elem指向的所有元素，返回L.length。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

(13)写入文件：status SaveList(SqList L,char FileName[]);输入线性表L、文件名FileName。如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

(14)读取文件：status LoadList(SqList \*L,char FileName[]);输入线性表L、文件名Filename。如果线性表L不存在，为L.elem分配空间，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

(15)多线性表管理-增加线性表：status AddList(LISTS \*Lists,char ListName[]);输入多线性表Lists、线性表名。为线性表分配空间，归零该线性表长度，为该线性表命名，多线性表长度加一。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

(16)多线性表管理-移除线性表：status RemoveList(LISTS \*Lists,char ListName[]);

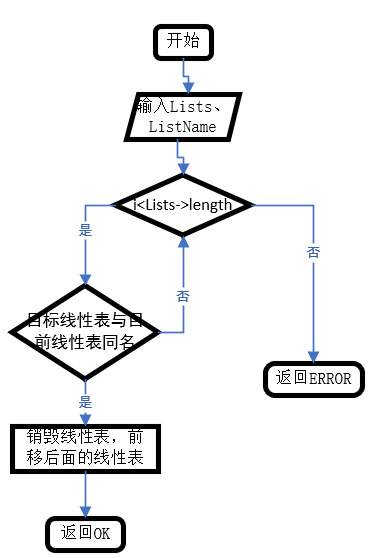


图1-2-7 移除线性表算法设计图

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

(17)多线性表管理-查找线性表：int LocateList(LISTS Lists,char ListName[]);输入多线性表Lists、线性表名ListName。找到同名线性表后返回逻辑序号，未找到则返回0.

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

## 1.3 系统实现

## 1.3.1 程序实现环境

WIN10系统下使用CodeBlocks编辑，编程语言为C语言。

## 1.3.2 头文件及预定义说明

(1)预定义：

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef struct{

ElemType \*elem;

int length;

int listsize;

}SqList;

typedef struct{

struct

{ char name[30];

SqList L;

}elem[10];

int length;

int listsize;

}LISTS;

(2)头文件：

1.func.h：包含所有需要使用的函数定义，在main.c中被调用。

2.def.h:包含所有预定义，在func.h中被调用。

**1.4 系统测试**

测试中线性表名字分别为101、102、103，所含元素皆为1、2、3、4这四个元素。测试如下：

1. 显示全部线性表（如表1-4-1）

表1-4-1 显示线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.无线性表时 | 无线性表 |  |
| 2.有三个线性表时 | 显示线性表名字及元素 |  |

1. 增加线性表（如表1-4-2）

表1-4-2 增加线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.无线性表时，选择操作1 | 无线性表 |  |
| 2.选择操作2，输入名称101 | 创建线性表成功 |  |
| 3.选择操作1 | 显示线性表101 |  |

1. 删除线性表测试（见表1-4-3）

表1-4-3 删除线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.有线性表时，选择操作1 | 显示线性表101 |  |
| 2.选择操作3，输入名称101 | 创建线性表成功 |  |
| 3.选择操作1 | 无线性表 |  |
| 4.选择操作3，输入错误表名 | 删除失败 |  |

1. 选择线性表测试（见表1-4-4）

表1-4-4 选择线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作1 | 线性表101、102 |  |
| 2.已有线性表101、102，选择101 | 进入次级菜单 |  |

1. 清除线性表、求表长、获取元素测试（见表1-4-5）

表1-4-5线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作12，遍历线性表 | 线性表已有1、2、3、4 |  |
| 2.选择操作5 | 线性表长度为4 |  |
| 3.选择操作6，获取第2个元素 | 成功提取元素，第二个元素为2 |  |
| 4.选择操作6，获取不合法元素 | 输入i不合法 |  |
| 5.选择操作3 | 线性表清除成功！ |  |
| 6.选择操作5 | 线性表长度为0 |  |

1. 查找元素、获取前驱、获取后驱测试（见表1-4-6）

表1-4-6 线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作12，遍历线性表 | 线性表已有1、2、3、4 |  |
| 2.选择操作7，查找元素3 | 查找元素在位置3处 |  |
| 3.选择操作7，查找非法元素100 | e不存在 |  |
| 4.选择操作8，获取元素2前驱 | 前驱为1，储存在pre中 |  |
| 5.选择操作8，获取元素1前驱 | 没有前驱 |  |
| 6.选择操作8，获取非法元素100前驱 | 没有前驱 |  |
| 7.选择操作9，获取元素2后驱 | 后驱为3，储存在next中 |  |
| 8.选择操作9，获取元素4后驱 | 没有后驱 |  |
| 9.选择操作9，获取非法元素0后驱 | 没有后驱 |  |

1. 插入元素、删除元素测试（见表1-4-7）

表1-4-7 线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作12，遍历线性表 | 线性表已有1、2、3、4 |  |
| 2.选择操作10，于第5个元素前插入5 | 元素插入成功 |  |
| 3.选择操作10，于第9个元素前插入6 | 插入位置不正确 |  |
| 4．选择操作12 | 显示1、2、3、4、5 |  |
| 5.选择操作11，删除第5个元素 | 元素删除成功，删除元素5保存在e中 |  |
| 6.选择操作11，删除非法位置11 | 删除位置不正确 |  |
| 7.选择操作12 | 显示1、2、3、4 |  |

1. 写入、读取文件操作测试（见表1-4-8）

表1-4-7 线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作12，遍历线性表 | 线性表已有1、2、3、4 |  |
| 2.选择操作13，选择读入操作，文件名myfile | 元素插入成功 |  |
| 3.返回初级菜单，选择线性表102，选择操作13，读取操作，输入文件名myfile | 读取成功 |  |
| 4.选择操作12 | 显示1、2、3、4 |  |
| 5.返回初级菜单，选择操作1 | 显示101、102及其所含元素 |  |

## 1.5 实验小结

本次实验实现了线性表的各个操作，首先在educoder上编写了所需的16个函数，在之后将各个函数组装为一个系统，以菜单形式呈现。

在课堂上粗浅地学习了线性表之后，本次实验令我更加深入地理解了线性表。除此之外，还对预定义、头文件、函数引用有了更深的了解，更加明白了代码可读性、易编辑性的意义。

在实验中，遇到了较多问题，在自己思索、询问同学、网上查询之后也成功地解决了，对数据结构这门课也有了初步的认识。之后的实验会更加努力，希望能有更多的收获。

# 2基于链式存储结构的线性表实现

## 2.1 问题描述

通过实验达到⑴加深对线性表的概念、基本运算的理解；⑵熟练掌握线性表的逻辑结构与物理结构的关系；⑶物理结构采用单链表,熟练掌握线性表的基本运算的实现。

## 2.1.1 12个基本函数需求实现

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。

⑴初始化表：函数名称是InitList(L)；初始条件是线性表L不存在；操作结果是构造一个空的线性表。

⑵销毁表：函数名称是DestroyList(L)；初始条件是线性表L已存在；操作结果是销毁线性表L。

⑶清空表：函数名称是ClearList(L)；初始条件是线性表L已存在；操作结果是将L重置为空表。

⑷判定空表：函数名称是ListEmpty(L)；初始条件是线性表L已存在；操作结果是若L为空表则返回TRUE,否则返回FALSE。

⑸求表长：函数名称是ListLength(L)；初始条件是线性表已存在；操作结果是返回L中数据元素的个数。

⑹获得元素：函数名称是GetElem(L,i,e)；初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用e返回L中第i个数据元素的值。

⑺查找元素：函数名称是LocateElem(L,e)；初始条件是线性表已存在；操作结果是返回L中第1个与e相同数据元素的位序，若这样的数据元素不存在，则返回值为0。

⑻获得前驱：函数名称是PriorElem(L, e,pre)；初始条件是线性表L已存在；操作结果是若e是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。

⑼获得后继：函数名称是NextElem(L, e,next)；初始条件是线性表L已存在；操作结果是若e是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。

⑽插入元素：函数名称是ListInsert(L,i,e)；初始条件是线性表L已存在，1≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

⑿遍历表：函数名称是ListTraverse(L)，初始条件是线性表L已存在；操作结果是依次输出L的每个数据元素。

## 2.1.2 演示系统

演示系统采用菜单形式呈现，有两层目录，实现多个线性表管理。欢迎菜单有包括退出系统、进入次级菜单的五个操作，次级菜单实现了顺序表的12个基本操作，以及文件写入、读取功能，还有返还初级菜单的功能。

## 2.1.3 文件读写功能

除了12种顺序表基本操作，本实验还新增了写入文件以及读取文件的功能

(1)写入文件：函数名称是SaveList(L,FileName);初始条件是线性表L存在；操作结果是把顺序表L中的数据按顺序写入指定文件中。

(2)读取文件：函数名称是LoadList(L,FileName);初始条件是存在指定文件；操作结果是把指定文件中的数据存入顺序表中。

## 2.2 系统设计

## 2.2.1 系统总体设计

系统main函数中switch作为主干，循环运行。

图2-2-1 系统总体设计思路图

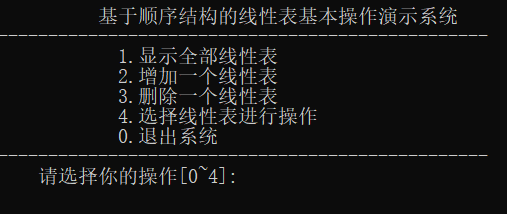


图2-2-2 初级菜单图

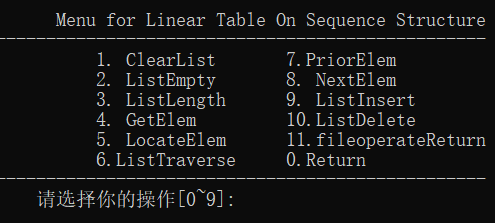


图2-2-3 次级菜单图

## 2.2.2 算法设计

⑴初始化表：status InitList(LinkList &L)；输入线性表L，线性表L存在时，返回INFEASIBLE；线性表不存在时，为L分配空间，并将L->next指向空，返回OK。

时空复杂度分析：T(n)=O(1);S(n)=O(1).

⑵销毁表：status DestroyList(LinkList &L);线性表L不存在时，返回INFEASIBLE；线性表L存在时，依次释放各个节点，返回OK。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑶清空表：status ClearList(LinkList \*L);线性表L不存在时，返回INFEASIBLE；线性表L存在时，将首节点保存，依次释放之后的各个节点，再将首结点指向空，返回OK。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑷判定空表：status ListEmpty(LinkList L)；线性表L不存在时，返回INFEASIBLE;线性表L存在时，检测L指向的结点是否为空，为空时返回OK；否则返回ERROR。

时空复杂度分析：T(n)=O(1);S(n)=O(1).

⑸求表长：int ListLength(LinkList L)；线性表L不存在时，返回INFEASIBLE；线性表存在时，while(p)循环i++，最后返回i-1。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑹获得元素：status GetElem(LinkList L,int i,ElemType &e)；输入线性表L、元素序号i、存储元素变量e。线性表L不存在时，返回INFEASBILE；线性表存在时，若i不符合条件（1≤i≤ListLength(L)），返回ERROR；否则，遍历链表并将目标元素赋值给e，返回OK。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑺查找元素：status LocateElem(LinkList L,ElemType e)；线性表L不存在时，返回INFEASIBLE；线性表存在时，遍历L，比较L->data与e，查找到时返回位置序号；未查找到时或线性表为空时返回ERROR。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑻获得前驱：status PriorElem(LinkList L,ElemType e,ElemType &pre)；



图2-2-4 获得前驱算法设计图

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑼获得后继：status NextElem(LinkList L,ElemType e,ElemType &next)；与获得前驱操作近同，只是将边缘条件更改，此处略过。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑽插入元素：status ListInsert(LinkList &L,int i,ElemType e)；



图2-2-5 插入元素算法设计图

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑾删除元素：status ListDelete(LinkList &L,int i,ElemType &e)；



图2-2-6 删除元素算法设计图

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑿遍历表：status ListTrabverse(LinkList L);L不存在时，返回INFEASIBLE；L存在时，p指向L头结点，遍历L，输出p->data，返回OK。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

(13)写入文件：status SaveList(LinkList L,char FileName[]);输入线性表L、文件名FileName。如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

(14)读取文件：status LoadList(LinkList &L,char FileName[]);输入线性表L、文件名Filename。如果线性表L存在，将其覆盖，重新分配空间，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

(15)多线性表管理-增加线性表：status AddList(LISTS \*Lists,char ListName[]);输入多线性表Lists、线性表名。为线性表分配空间，归零该线性表长度，为该线性表命名，多线性表长度加一。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

(16)多线性表管理-移除线性表：status RemoveList(LISTS &Lists,char ListName[]);

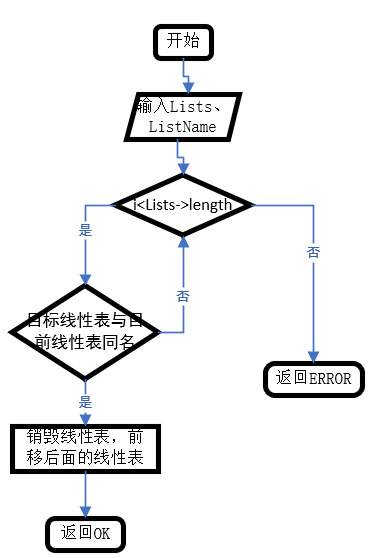


图2-2-7 移除线性表算法设计图

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

(17)多线性表管理-查找线性表：int LocateList(LISTS Lists,char ListName[]);输入多线性表Lists、线性表名ListName。找到同名线性表后返回逻辑序号，未找到则返回0.

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

## 2.3 系统实现

## 2.3.1 程序实现环境

WIN10系统下使用CodeBlocks编辑，编程语言为C语言。

## 2.3.2 头文件及预定义说明

(1)预定义：

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -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

#define LISTS\_INIT\_SIZE 10

typedef int ElemType;

typedef struct LNode{ //单链表（链式结构）结点的定义

ElemType data;

struct LNode \*next;

}LNode,\*LinkList;

typedef struct{

struct{

char name[30];

LinkList L;

}elem[LISTS\_INIT\_SIZE];

int length; //因为已经确认elem[10]，所以长度已经确定为10.

int listsize;

}LISTS;

(2)头文件：

1.func.h：包含所有需要使用的函数声明，在main.cpp中被调用。

2.def.h:包含所有预定义，在func.h中被调用。

(3)cpp文件：

1.func.cpp：包含所有函数定义，

**2.4 系统测试**

测试中线性表名字分别为101、102、103，所含元素皆为1、2、3、4这四个元素。测试如下：

（1）显示全部线性表（如表2-4-1）

表2-4-1 显示线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.无线性表时 | 无线性表 |  |
| 2.有三个线性表时 | 显示线性表名字及元素 |  |

（2）增加线性表（如表2-4-2）

表2-4-2 增加线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.无线性表时，选择操作1 | 无线性表 |  |
| 2.选择操作2，输入名称101 | 创建线性表成功 |  |
| 3.选择操作1 | 显示线性表101 |  |

（3）删除线性表测试（见表2-4-3）

表2-4-3 删除线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.有线性表时，选择操作1 | 显示线性表101 |  |
| 2.选择操作3，输入名称101 | 创建线性表成功 |  |
| 3.选择操作1 | 无线性表 |  |
| 4.选择操作3，输入错误表名 | 删除失败 |  |

（4）选择线性表测试（见表2-4-4）

表2-4-4 选择线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作1 | 线性表101、102 |  |
| 2.已有线性表101、102，选择101 | 进入次级菜单 |  |
| 3.输入错误名称103 | 无该线性表 |  |

（5）清除线性表、判空、求表长、获取元素测试（见表2-4-5）

表2-4-5线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作6，遍历线性表 | 线性表已有1、2、3、4 |  |
| 2.选择操作3 | 线性表长度为4 |  |
| 3.选择操作4，获取第2个元素 | 成功提取元素，第二个元素为2 |  |
| 4.选择操作4，获取不合法元素 | 输入i不合法 |  |
| 5.选择操作2 | 线性表非空 |  |
| 6.选择操作1 | 线性表清除成功！ |  |
| 7.选择操作3 | 线性表长度为0 |  |
| 8.选择操作2 | 线性表未空 |  |

（6）查找元素、获取前驱、获取后驱测试（见表2-4-6）

表2-4-6 线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作6，遍历线性表 | 线性表已有1、2、3、4 |  |
| 2.选择操作5，查找元素4 | 查找元素在位置4处 |  |
| 3.选择操作5，查找非法元素99 | e不存在 |  |
| 4.选择操作7，获取元素2前驱 | 前驱为1，储存在pre中 |  |
| 5.选择操作7，获取元素1前驱 | 没有前驱 |  |
| 6.选择操作7，获取非法元素100前驱 | 没有前驱 |  |
| 7.选择操作8，获取元素2后驱 | 后驱为3，储存在next中 |  |
| 8.选择操作8，获取元素4后驱 | 没有后驱 |  |
| 9.选择操作8，获取非法元素0后驱 | 没有后驱 |  |

（7）插入元素、删除元素测试（见表2-4-7）

表2-4-7 线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作6，遍历线性表 | 线性表已有1、2、3、4 |  |
| 2.选择操作9，于第5个元素前插入5 | 元素插入成功 |  |
| 3.选择操作9，于第9个元素前插入6 | 插入位置不正确 |  |
| 4．选择操作6 | 显示1、2、3、4、5 |  |
| 5.选择操作10，删除第5个元素 | 元素删除成功，删除元素5保存在e中 |  |
| 6.选择操作11，删除非法位置11 | 删除位置不正确 |  |
| 7.选择操作6 | 显示1、2、3、4 |  |

（8）写入、读取文件操作测试（见表2-4-8）

表2-4-7 线性表测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作6，遍历线性表 | 线性表已有1、2、3、4 |  |
| 2.选择操作11，选择读入操作，文件名myfile | 元素插入成功 |  |
| 3.返回初级菜单，选择线性表102，选择操作11，读取操作，输入文件名myfile | 读取成功 |  |
| 4.选择操作6 | 显示1、2、3、4 |  |
| 5.选择操作11，读取操作，输入错误文件名getchar | 文件不存在 |  |
| 5.返回初级菜单，选择操作1 | 显示101、102及其所含元素 |  |

## 2.5 实验小结

本次实验实现了非顺序线性表的各个操作，首先在educoder上编写了所需的16个函数，在之后将各个函数组装为一个系统，以菜单形式呈现。

大一的C语言学习中，我对链表掌握的一直不是很熟练，如今通过实现不同功能，对链表的了解也更深了。当然对于链表形式的顺序表也有了更深的了解，收获颇丰。

这次实验和上次的实验在组装程序上近似，节省了不少时间，实验做得也更快一些。在实验中，遇到的问题更少了，以此改进了不少算法，让一些函数变得更加美观简洁。在自己思索、询问同学、网上查询之后也成功地解决了，对数据结构这门课也有了更好的认识。之后的实验会更加努力，希望能有更多的收获。

# 3基于二叉链表的二叉树实现

## 3.1 问题描述

通过实验达到⑴加深对二叉树的概念、基本运算的理解；⑵熟练掌握二叉树的逻辑结构与物理结构的关系；⑶以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。

## 3.1.1 基本函数需求实现

依据最小完备性和常用性相结合的原则，以函数形式定义了二叉树的创建二叉树、清空二叉树和求二叉树深度等12种基本运算。具体运算功能定义和说明如下。

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

⑵清空二叉树：函数名称是ClearBiTree (T)；初始条件是二叉树T存在；操作结果是将二叉树T清空。

⑶求二叉树深度：函数名称是BiTreeDepth(T)；初始条件是二叉树T存在；操作结果是返回T的深度。

⑷查找结点：函数名称是LocateNode(T,e)；初始条件是二叉树T已存在，e是和T中结点关键字类型相同的给定值；操作结果是返回查找到的结点指针，如无关键字为e的结点，返回NULL。

⑸结点赋值：函数名称是Assign(T,e,value)；初始条件是二叉树T已存在，e是和T中结点关键字类型相同的给定值；操作结果是关键字为e的结点赋值为value。

⑹获得兄弟结点：函数名称是GetSibling(T,e)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值；操作结果是返回关键字为e的结点的（左或右）兄弟结点指针。若关键字为e的结点无兄弟，则返回NULL。

⑺插入结点：函数名称是InsertNode(T,e,LR,c)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值，LR为0或1，c是待插入结点；操作结果是根据LR为0或者1，插入结点c到T中，作为关键字为e的结点的左或右孩子结点，结点e的原有左子树或右子树则为结点c的右子树。特殊情况，c插入作为根结点时，可以考虑LR为-1时，作为根结点插入，原根结点作为c的右子树。

⑻删除结点：函数名称是DeleteNode(T,e)；初始条件是二叉树T存在，e是和T中结点关键字类型相同的给定值。操作结果是删除T中关键字为e的结点；同时，如果关键字为e的结点度为0，删除即可；如关键字为e的结点度为1，用关键字为e的结点孩子代替被删除的e位置；如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为e的左子树中最右结点的右子树。

⑼前序遍历：函数名称是PreOrderTraverse(T,Visit())；初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）；操作结果：先序遍历，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⑽中序遍历：函数名称是InOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是一个函数指针的形参（可使用该函数对结点操作）；操作结果是中序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

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

⑿按层遍历：函数名称是LevelOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是层序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

## 3.1.2 演示系统

演示系统采用菜单形式呈现，有两层目录，实现多个线性表管理。欢迎菜单有包括退出系统、进入次级菜单的五个操作，次级菜单实现了顺序表的12个基本操作，以及文件写入、读取功能，还有返还初级菜单的功能。

## 3.1.3 文件读写功能

(1)写入文件：函数名称是SaveBiTree(T,FileName);初始条件是二叉树T存在；操作结果是把二叉树T中的数据按顺序写入指定文件中。

(2)读取文件：函数名称是LoadBiTree(T,FileName);初始条件是存在指定文件；操作结果是把指定文件中的数据存入二叉树中。

## 3.2 系统设计

## 3.2.1 系统总体设计

系统main函数中switch作为主干，循环运行。



图3-2-1 系统总体设计思路图

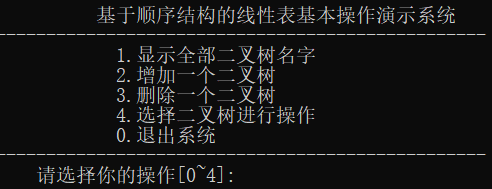


图3-2-2 初级菜单图

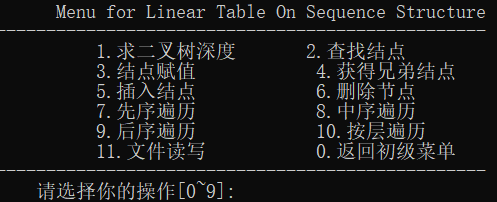


图3-2-3 次级菜单图

## 3.2.2 算法设计

⑴创建二叉树status CreateBiTree(BiTree &T,TElemType definition[])：输入带空枝的二叉树先根遍历序列definition；将根节点指针赋值给T并返回OK。



图3-2-4 创建二叉树算法设计图

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

⑵清空二叉树status ClearBiTree(BiTree &T)：输入二叉树T；将二叉树设置成空，并删除所有结点，释放结点空间。

时空复杂度分析：T(n)=O(n);S(n)=O(n).

⑶求二叉树深度int BiTreeDepth(BiTree T)：输入二叉树T；，若无二叉树，返回0,否则返回二叉树深度。

时空复杂度分析：T(n)=O(n);S(n)=O(n).

⑷查找结点BiTNode\* LocateNode(BiTree T,KeyType e)：输入二叉树T以及结点关键字e；返回查找到的结点指针，如无关键字为e的结点，返回NULL。

时空复杂度分析：T(n)=O(n);S(n)=O(n).

⑸结点赋值status Assign(BiTree& T, KeyType e, TElemType value)：输入二叉树T、赋值结点关键字e、修改值value；将关键字为e的结点赋值为value。

时空复杂度分析：T(n)=O(n);S(n)=O(n).

⑹获得兄弟结点BiTNode\* GetSibling(BiTree T,KeyType e)：输入二叉树T以及需要获得兄弟结点的结点关键字；返回关键字为e的结点的（左或右）兄弟结点指针，若关键字为e的结点无兄弟，则返回NULL。

时空复杂度分析：T(n)=O(n);S(n)=O(n).

⑺插入结点status InsertNode(BiTree &T,KeyType e,int LR,TElemType c)：输入二叉树T、插入位置结点关键字、插入方式、插入值c； LR为0或者1，插入结点c到T中，作为关键字为e的结点的左或右孩子结点，结点e的原有左子树或右子树则为结点c的右子树。特殊情况，c插入作为根结点时，LR为-1时，作为根结点插入，原根结点作为c的右子树。

时空复杂度分析：T(n)=O(n);S(n)=O(n).

⑻删除结点status DeleteNode(BiTree &T,KeyType e)：输入二叉树T、需要删除的结点关键字e；如果关键字为e的结点度为0，删除即可；如关键字为e的结点度为1，用关键字为e的结点孩子代替被删除的e位置；如关键字为e的结点度为2，用e的左孩子代替被删除的e位置，e的右子树作为e的左子树中最右结点的右子树。

时空复杂度分析：T(n)=O(n);S(n)=O(n).

⑼前序遍历status PreOrderTraverse(BiTree T,void (\*visit)(BiTree))：输入二叉树T、访问形参visit；先序遍历，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

时空复杂度分析：T(n)=O(n);S(n)=O(n).

⑽中序遍历status InOrderTraverse(BiTree T,void (\*visit)(BiTree))：输入二叉树T、访问形参visit；中序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。（采用非递归算法进行）



图3-2-5 中序遍历算法设计图

时空复杂度分析：T(n)=O(n);S(n)=O(n).

⑾后序遍历status PostOrderTraverse(BiTree T,void (\*visit)(BiTree))：输入二叉树T、访问形参visit；后序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

时空复杂度分析：T(n)=O(n);S(n)=O(n).

⑿按层遍历：函数名称是LevelOrderTraverse(T,Visit))；输入二叉树T、访问形参visit；层序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。



图3-2-6 层序遍历算法设计图

(13)写入文件status SaveBiTree(BiTree T, char FileName[])：输入二叉树T、文件名FileName；如果二叉树T存在，将元素写到FileName文件中，返回OK。

时空复杂度分析：T(n)=O(n);S(n)=O(n).

(14)读取文件status LoadBiTree(BiTree& T, char FileName[])：输入二叉树T、文件名Filename；如果二叉树T存在，将其覆盖，重新分配空间，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

时空复杂度分析：T(n)=O(n);S(n)=O(n).

(15)多线性表管理-增加线性表status AddList(LISTS \*Lists,char ListName[])：输入多线性表Lists、线性表名；为线性表分配空间，归零该线性表长度，为该线性表命名，多线性表长度加一。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

(16)多线性表管理-移除线性表status RemoveList(LISTS &Lists,char ListName[])：

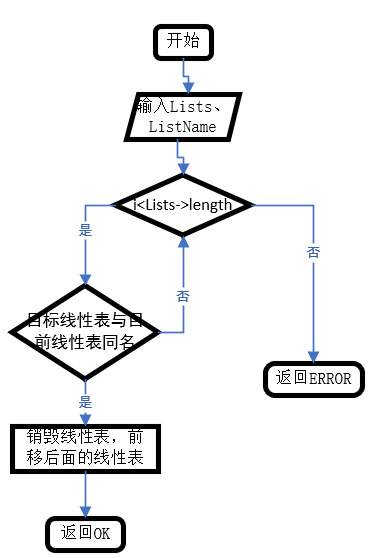


图3-2-7 移除线性表算法设计图

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

(17)多线性表管理-查找线性表int LocateList(LISTS Lists,char ListName[])：输入多线性表Lists、线性表名ListName；找到同名线性表后返回逻辑序号，未找到则返回0。

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

## 3.3 系统实现

## 3.3.1 程序实现环境

WIN10系统下使用CodeBlocks编辑，编程语言为C语言。

## 3.3.2 头文件及预定义说明

(1)预定义：

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LISTS\_INIT\_SIZE 10

typedef int status;

typedef int KeyType;

typedef struct {

KeyType key;

char others[20];

} TElemType; //二叉树结点类型定义

typedef struct BiTNode{ //二叉链表结点的定义

TElemType data;

struct BiTNode \*lchild,\*rchild;

} BiTNode, \*BiTree;

typedef struct{

struct{

char name[30];

BiTree T;

}elem[LISTS\_INIT\_SIZE];

int length; //因为已经确认elem[10]，所以长度已经确定为10.

int listsize;

}LISTS;

(2)头文件：

1.func.h：包含所有需要使用的函数声明，在main.cpp中被调用。

2.def.h:包含所有预定义，在func.h中被调用。

(3)cpp文件：

1.func.cpp：包含所有函数定义，

**3.4 系统测试**

测试中二叉树名字分别为101、102，101和102以definition（1 a 2 b 0 null 0 null 3 c 4 d 0 null 0 null 5 e 0 null 0 null -1 null）建立二叉树。测试如下：

（1）显示全部二叉树（如表3-4-1）

表3-4-1 显示二叉树测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.无二叉树时 | 无二叉树 |  |
| 2.有两个二叉树时 | 显示二叉树名字 |  |

（2）增加二叉树（如表3-4-2）

表3-4-2 增加二叉树测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.无二叉树时，选择操作1 | 无二叉树 |  |
| 2.选择操作2，输入名称101以及definition | 创建二叉树成功 |  |
| 3.选择操作1 | 显示二叉树 |  |

（3）删除二叉树测试（见表3-4-3）

表3-4-3 删除二叉树测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.有二叉树时，选择操作1 | 显示二叉树101 |  |
| 2.选择操作3，输入名称101 | 删除二叉树成功 |  |
| 3.选择操作1 | 无线性表 |  |
| 4.选择操作3，输入错误表名 | 删除失败 |  |

（4）选择二叉树测试（见表3-4-4）

表3-4-4 选择二叉树测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作1 | 二叉树101、102 |  |
| 2.已有二叉树101、102，选择101 | 进入次级菜单 |  |
| 3.输入错误名称103 | 无该二叉树 |  |

（5）先序遍历、中序遍历、后序遍历、按层遍历、求二叉树深度测试（见表3-4-5）

表3-4-5二叉树测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作7 | 如右 |  |
| 2.选择操作8 | 如右 |  |
| 3.选择操作9 | 如右 |  |
| 4.选择操作10 | 如右 |  |
| 5.选择操作1 | 二叉树深度为3 |  |

（6）查找结点、结点赋值、获得兄弟结点测试（见表3-4-6）

表3-4-6 二叉树测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作10，遍历二叉树 | a,b,c,d,e |  |
| 2.选择操作2，查找4 | 查找元素4，d |  |
| 3.选择操作2，查找非法关键字99 | 查找失败 |  |
| 4.选择操作3，修改关键字为4的结点 | 已成功赋值 |  |
| 5.选择操作3，修改关键字为8的结点 | 赋值失败 |  |
| 6.选择操作10，遍历二叉树 | a,b,c,h,e |  |
| 7.选择操作3，赋值重复关键字 | 赋值失败 |  |
| 8.选择操作4，查找关键字5的兄弟结点 | 6,h |  |
| 9.选择操作4，查找关键字1的兄弟结点 | 查找失败 |  |

（7）插入结点、删除结点测试（见表3-4-7）

表3-4-7 二叉树测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作10，遍历二叉树 | a,b,c,d,e |  |
| 2.选择操作5 | 插入成功 |  |
| 3.选择操作5 | 插入成功 |  |
| 4.选择操作5 | 插入成功 |  |
| 5.选择操作6 | 结点删除成功 |  |
| 6.选择操作6 | 结点删除成功 |  |
| 7.选择操作6 | 结点删除成功 |  |

（8）写入、读取文件操作测试（见表3-4-8）

表3-4-7 二叉树测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作10，遍历二叉树 | a,b,c,d,e |  |
| 2.选择操作11，选择读入操作，文件名myfile | 读入成功 |  |
| 3.返回初级菜单，选择线性表102，选择操作11，读取操作，输入文件名myfile | 读取成功 |  |
| 4.选择操作11，读取操作，输入错误文件名 | 文件不存在 |  |

## 3.5 实验小结

本次实验实现了二叉树的各个操作，首先在educoder上编写了所需的13个函数，在之后将各个函数组装为一个系统，以菜单形式呈现。

本次学习的二叉树数据结构是一个全新的结构，初次了解，运用并不是很熟练，所以编程过程比较磕磕绊绊。但是在全部完成之后也对于二叉树这一结构有了更深的理解，收获颇丰。

遇到的问题较多，通过在网上查询以及询问他人，也都得到了解决。本次实验所碰到最难理解的部分在于非递归算法，在仔细钻研之后也感觉对于栈和队列的使用方法更加掌握了。

# 4基于邻接表的图实现

## 4.1 问题描述

通过实验达到⑴加深对图的概念、基本运算的理解；⑵熟练掌握图的逻辑结构与物理结构的关系；⑶以邻接表作为物理结构，熟练掌握图基本运算的实现。

## 4.1.1 基本函数需求实现

依据最小完备性和常用性相结合的原则，以函数形式定义了创建图、销毁图、查找顶点、获得顶点值和顶点赋值等12种基本运算。具体运算功能定义和说明如下。具体运算功能定义如下。

(1)创建图：函数名称是CreateGraph(G,V,VR)；初始条件是V是图的顶点集VR是图的关系集；操作结果是按V和VR的定义构造图G。

注：①要求图G中顶点关键字具有唯一性。后面各操作的实现，也都要满足一个图中关键字的唯一性，不再赘述；② V和VR对应的是图的逻辑定义形式，比如V为顶点序列，VR为关键字对的序列。不能将邻接矩阵等物理结构来代替V和VR。

(2)销毁图：函数名称是DestroyGraph(G)；初始条件图G已存在；操作结果是销毁图G。

(3)查找顶点：函数名称是LocateVex(G,u)；初始条件是图G存在，u是和G中顶点关键字类型相同的给定值；操作结果是若u在图G中存在，返回关键字为u的顶点位置序号（简称位序），否则返回其它表示“不存在”的信息。

(4)顶点赋值：函数名称是PutVex (G,u,value)；初始条件是图G存在，u是和G中顶点关键字类型相同的给定值；操作结果是对关键字为u的顶点赋值value。

(5)获得第一邻接点：函数名称是FirstAdjVex(G, u)；初始条件是图G存在，u是G中顶点的位序；操作结果是返回u对应顶点的第一个邻接顶点位序，如果u的顶点没有邻接顶点，否则返回其它表示“不存在”的信息。

(6)获得下一邻接点：函数名称是NextAdjVex(G, v, w)；初始条件是图G存在，v和w是G中两个顶点的位序，v对应G的一个顶点,w对应v的邻接顶点；操作结果是返回v的（相对于w）下一个邻接顶点的位序，如果w是最后一个邻接顶点，返回其它表示“不存在”的信息。

(7)插入顶点：函数名称是InsertVex(G,v)；初始条件是图G存在，v和G中的顶点具有相同特征；操作结果是在图G中增加新顶点v。（在这里也保持顶点关键字的唯一性）

(8)删除顶点：函数名称是DeleteVex(G,v)；初始条件是图G存在，v是和G中顶点关键字类型相同的给定值；操作结果是在图G中删除关键字v对应的顶点以及相关的弧。

(9)插入弧：函数名称是InsertArc(G,v,w)；初始条件是图G存在，v、w是和G中顶点关键字类型相同的给定值；操作结果是在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>。

(10)删除弧：函数名称是DeleteArc(G,v,w)；初始条件是图G存在，v、w是和G中顶点关键字类型相同的给定值；操作结果是在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>。

(11)深度优先搜索遍历：函数名称是DFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

(12)广深度优先搜索遍历：函数名称是BFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

## 4.1.2 演示系统

演示系统采用菜单形式呈现，有两层目录，实现多个图管理。欢迎菜单有包括退出系统、进入次级菜单的五个操作，次级菜单实现了图的12个基本操作，以及文件写入、读取功能，还有返还初级菜单的功能。

## 4.1.3 文件读写功能

(1)写入文件：函数名称是SaveGraph(G,FileName);初始条件是图G存在；操作结果是把图G中的数据按顺序写入指定文件中。

(2)读取文件：函数名称是LoadGragh(G,FileName);初始条件是存在指定文件；操作结果是把指定文件中的数据存入图中。

## 4.2 系统设计

## 4.2.1 系统总体设计

系统main函数中switch作为主干，循环运行。



图4-2-1 系统总体设计思路图

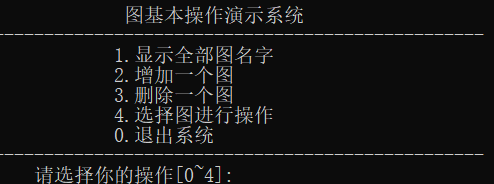


图4-2-2 初级菜单图

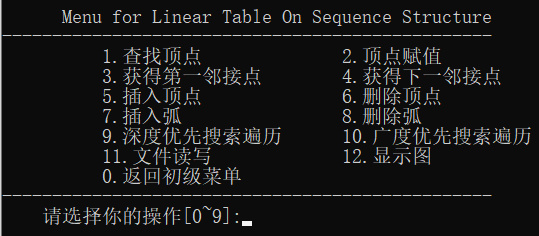


图4-2-3 次级菜单图

## 4.2.2 算法设计

⑴创建图status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2])：输入图的顶点集V以及关系集VR||按V和VR的定义构造图G并返回OK。



图4-2-4 创建图算法设计图

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

⑵销毁图status DestroyGraph(ALGraph &G)：输入图G||将图销毁。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑶查找顶点int LocateVex(ALGraph G,KeyType u)：输入图G、顶点关键字u||根据u在图G中查找顶点，查找成功返回位序，否则返回-1。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑷顶点赋值status PutVex(ALGraph &G,KeyType u,VertexType value)：输入图G、顶点关键字u以及修改值value||根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK；如果查找失败或关键字不唯一，返回ERROR。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑸获得第一邻接点int FirstAdjVex(ALGraph G,KeyType u)：输入图G、顶点关键字u||根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，否则返回-1。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑹获得下一邻接点int NextAdjVex(ALGraph G,KeyType v,KeyType w)：输入图G、顶点关键字v以及上一个弧连接点w||根据v在图G中查找顶点，查找成功返回顶点v的邻接顶点相对于w的下一邻接顶点的位序，查找失败返回-1。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑺插入顶点status InsertVex(ALGraph &G,VertexType v)：输入图G以及插入顶点值v||在图G中插入顶点v，成功返回OK,否则返回ERROR。

时空复杂度分析：T(n)=O(1);S(n)=O(1).

⑻删除顶点status DeleteVex(ALGraph &G,KeyType v)：输入图G、需要删除的顶点v||在图G中删除关键字v对应的顶点以及相关的弧，成功返回OK,否则返回ERROR。



图4-2-5 删除顶点算法设计图

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

⑼插入弧status InsertArc(ALGraph &G,KeyType v,KeyType w)：输入图G、弧两端顶点v、w||在图G中增加弧<v,w>，成功返回OK,否则返回ERROR。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑽删除弧status DeleteArc(ALGraph &G,KeyType v,KeyType w)：输入图G、弧两端顶点v、w||在图G中删除弧<v,w>，成功返回OK,否则返回ERROR。



图4-2-6 删除弧算法设计图

时空复杂度分析：T(n)=O(n);S(n)=O(1).

⑾深度优先搜索遍历status DFSTraverse(ALGraph &G,void (\*visit)(VertexType))：输入图G、访问形参visit||对图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。



图4-2-7 深度优先搜索遍历算法设计图

时空复杂度分析：T(n)=O(n2);S(n)=O(n).

⑿广度优先搜索遍历status BFSTraverse(ALGraph &G,void (\*visit)(VertexType)): 输入图G、访问形参visit||对图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。



图4-2-8 广度优先搜索遍历算法设计图

(13)写入文件status SaveGraph(ALGraph G, char FileName[])：输入图G、文件名FileName||将图的数据写入到文件FileName中并返回OK。

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

(14)读取文件status LoadGraph(ALGraph &G, char FileName[])：输入图G、文件名FileName||读入文件FileName的图数据，将原图覆盖，创建图的邻接表，并返回OK。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

(15)多图管理-增加图status GraghsAdd(GraghsLink &Graghs,char ListName[])：输入多图Graghs、新增图名||在链表中新增一个含图结点，创建成功则返回OK，否则返回ERROR。

时空复杂度分析：T(n)=O(n);S(n)=O(1).

(16)多图管理-移除图status GraghsRemove(GraghsLink &Graghs,char ListName[])：输入多图Graghs、删除图名||在链表中删除同名结点，删除成功返回OK，否则返回ERROR。

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

(17)多图管理-查找图GraghsLink GraghsLocate(GraghsLink Graghs,char ListName[])：输入多图Graghs、查找图名ListName；找到同名图后返回结点指针，未找到则返回NULL。

时空复杂度分析：T(n)=O(n2);S(n)=O(1).

## 4.3 系统实现

## 4.3.1 程序实现环境

WIN10系统下使用CodeBlocks编辑，编程语言为C语言。

## 4.3.2 头文件及预定义说明

(1)预定义：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 20

#define LISTS\_INIT\_SIZE 20

typedef int status;

typedef int KeyType;

typedef enum {DG,DN,UDG,UDN} GraphKind;

typedef struct {

KeyType key;

char others[20];

} VertexType; //顶点类型定义

typedef struct ArcNode { //表结点类型定义

int adjvex; //顶点位置编号

struct ArcNode \*nextarc; //下一个表结点指针

} ArcNode;

typedef struct VNode{ //头结点及其数组类型定义

VertexType data; //顶点信息

ArcNode \*firstarc; //指向第一条弧

} VNode,AdjList[MAX\_VERTEX\_NUM];

typedef struct { //邻接表的类型定义

AdjList vertices; //头结点数组

int vexnum,arcnum; //顶点数、弧数

GraphKind kind; //图的类型

} ALGraph;

typedef struct GraghNode{

char name[30];

ALGraph G;

struct GraghNode \*next;

}GraghNode,\*GraghsLink;

(2)头文件：

1.func.h：包含所有需要使用的函数声明，在main.cpp中被调用。

2.def.h:包含所有预定义，在func.h中被调用。

(3)cpp文件：

1.func.cpp：包含所有函数定义，

**4.4 系统测试**

测试中图名字分别为101、102，101和102以如下顶点及关系（5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1）建立无向图。测试如下：

（1）显示全部图（如表4-4-1）

表4-4-1 显示图测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.无图时 | 无图 |  |
| 2.有两个图时 | 显示图名字 |  |

（2）增加图（如表4-4-2）

表4-4-2 增加图测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.无图时 | 无图 |  |
| 2.选择操作2，输入101以及图值 | 创建图成功 |  |
| 3.选择操作1 | 显示二叉树 |  |

（3）删除图测试（见表4-4-3）

表4-4-3 删除图测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.有图，选择操作1 | 显示图101 |  |
| 2.选择操作3，输入名称101 | 删除图成功 |  |
| 3.选择操作1 | 无线性表 |  |
| 4.选择操作3，输入错误表名 | 删除失败 |  |

（4）选择图测试（见表4-4-4）

表4-4-4 选择图测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作1 | 图101、102 |  |
| 2.已有二叉树101、102，选择101 | 进入次级菜单 |  |
| 3.输入错误名称103 | 无该二叉树 |  |

（5）查找顶点、获得第一邻接点、获得下一邻接点测试（见表4-4-5）

表4-4-5图测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作12 | 如右 |  |
| 2.选择操作1，输入5 | 位序为0 |  |
| 3.选择操作1，输入无效值10 | 查找失败 |  |
| 4.选择操作3，输入5 | 位序为2 |  |
| 5.选择操作3，输入无效值10 | 查找失败 |  |
| 6．选择操作4，输入5和7 | 位序为3 |  |
| 7.选择操作4，输入5和10 | 查找失败 |  |

（6）顶点赋值、插入顶点、删除顶点测试（见表4-4-6）

表4-4-6 图测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作12 | 如右 |  |
| 2.选择操作2，修改5，为10 有向图 | 已成功赋值2 |  |
| 3.选择操作2，赋值非法关键字9 | 无该顶点 |  |
| 4.选择操作2，修改关键字为8的顶点为重复值 | 赋值失败 |  |
| 5.选择操作5，插入顶点1 三叉树 | 插入成功 |  |
| 6.选择操作5，插入重复关键字 | 插入顶点失败 |  |
| 7.选择操作6，删除关键字为10的顶点 | 顶点删除成功 |  |
| 8.选择操作6，删除无效关键字11 | 删除失败 |  |

（7）插入弧、删除弧测试（见表4-4-7）

表4-4-7 图测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择操作12 | 如右 |  |
| 2.选择操作7，输入5 8 | 插入成功 |  |
| 3.选择操作7，输入已存在弧 5 7 | 插入失败 |  |
| 4.选择操作7，输入无效弧 5 11 | 插入失败 |  |
| 5.选择操作8，输入5 8 | 删除成功 |  |
| 6.选择操作8，输入不存在弧6 8 | 删除成功 |  |

（8）深度、广度优先搜索遍历操作测试（见表4-4-8）

表4-4-8 图测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.图1测试，选择操作12 | 如右 |  |
| 2.选择操作9 | 如右 |  |
| 3.选择操作10 | 如右 |  |
| 4.图2测试，选择操作12 | 如右 |  |
| 5.选择操作9 | 如右 |  |
| 6.选择操作10 | 如右 |  |

（9）文件读写操作测试（见表4-4-9）

表4-4-9 图测试

|  |  |  |
| --- | --- | --- |
| 测试步骤 | 理论结果 | 运行结果 |
| 1.选择图101，操作12 | 如右 |  |
| 2.选择操作11，选择读入，文件名myfile | 读入成功 |  |
| 3.选择图102，操作12 | 如右 |  |
| 4.选择操作11，读取文件myfile | 如右 |  |
| 5.选择操作11，读取无效文件 | 文件不存在 |  |

## 4.5 实验小结

本次实验实现了图的各个操作，首先在educoder上编写了所需的32个函数，在之后将各个函数组装为一个系统，以菜单形式呈现。

本次学习的图数据结构比较二叉树结构实际运用更为简单，但是毕竟初次了解，也花了一定时间才熟悉，全部完成之后也对于图这一结构有了更深的理解，收获颇丰。

上三次的实验的多表系统都是采用线性表结构实现，这样会有大小限制的问题，于是这次改用链表储存多表，储存方式有好处也有坏处，但对自己的能力也是种锻炼。

遇到的问题还是不少，但感觉八次课上完，已经有了更强的解决问题能力。本次实验所碰到最难理解的部分在于深度和广度优先搜索遍历算法，也是极为重要的，可能还是需要多多使用才能更加深入理解。

# 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

“main.c”

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include"func.h"

/\*-----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);

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

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

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

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

status ListTrabverse(SqList L); //简化过

status AddList(LISTS \*Lists,char ListName[]);

status RemoveList(LISTS \*Lists,char ListName[]);

int LocateList(LISTS Lists,char ListName[]);

status SaveList(SqList L,char FileName[]);

status LoadList(SqList \*L,char FileName[]);

int main(void)

{

int op,op1=1;

char charname[10],filename[20];

LISTS Lists;

Lists.length=0;

int judge,judge1; //存储返回值的中间量

int elemnum; //存储选取线性表的序号

int e; //寄存元素的变量

int i; //需要使用的序号变量

int pre,next; //前驱与后驱存放变量

while(op1){

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

printf(" 基于顺序结构的线性表基本操作演示系统\n");

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

printf(" 1.显示全部线性表\n");

printf(" 2.增加一个线性表\n");

printf(" 3.删除一个线性表\n");

printf(" 4.选择线性表进行操作\n");

printf(" 0.退出系统\n");

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

printf(" 请选择你的操作[0~4]:");

scanf("%d",&op1);

switch(op1){

case 1:

if(Lists.length==0)

printf("无线性表\n");

else

{

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

{

printf("%s ",Lists.elem[i].name);

ListTrabverse(Lists.elem[i].L);

printf("\n");

}

}

getchar();getchar();

break;

case 2:

printf("请输入你要创建的线性表名\n");

scanf("%s",charname);

judge1=AddList(&Lists,charname);

if(judge1==OK)

printf("创建线性表成功\n");

else printf("创建线性表失败\n");

getchar();getchar();

break;

case 3:

printf("请输入你要删除的线性表名\n");

scanf("%s",charname);

judge1=RemoveList(&Lists,charname);

if(judge1==OK)

printf("删除成功\n");

else printf("删除失败\n");

getchar();getchar();

break;

case 4:

op=1;

printf("输入选择的线性表\n");

scanf("%s",charname);

elemnum=LocateList(Lists,charname);

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(" 13.fileoperate 0. Return\n");

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

printf(" 请选择你的操作[0~13]:");

scanf("%d",&op);

switch(op){

case 1:

judge=InitList(&Lists.elem[elemnum-1].L);

if(judge==OK) printf("线性表创建成功！\n");

else if(judge==ERROR) printf("已经存在线性表。\n");

else printf("线性表创建失败！\n");

getchar();getchar();

break;

case 2:

if(DestroyList(&Lists.elem[elemnum-1].L)==OK) printf("线性表删除成功！\n");

else printf("线性表删除失败！\n");

getchar();getchar();

break;

case 3:

if(ClearList(&Lists.elem[elemnum-1].L)==OK) printf("线性表清除成功！\n");

else printf( "线性表清除失败！\n");

getchar();getchar();

break;

case 4:

judge=ListEmpty(Lists.elem[elemnum-1].L);

if(judge==TRUE) printf("线性表为空！\n");

else if(judge==FALSE) printf("线性表非空！\n");

else printf("线性表不存在！\n");

getchar();getchar();

break;

case 5:

judge=ListLength(Lists.elem[elemnum-1].L);

if(judge>=0) printf("线性表长度为%d\n",judge);

else printf("线性表不存在！\n");

getchar();getchar();

break;

case 6:

printf("输入你想获取的元素序号。\n");

scanf("%d",&i);

judge=GetElem(Lists.elem[elemnum-1].L,i,&e);

if(judge==OK) printf("成功提取出元素，第%d个元素为%d\n",i,e);

else if(judge==ERROR) printf("输入i不合法呢！\n");

else printf("L不存在呢！\n");

getchar();getchar();

break;

case 7:

printf("输入你想查找的元素\n");

scanf("%d",&e);

judge=LocateElem(Lists.elem[elemnum-1].L,e);

if(judge>0)

printf("查找元素在位置%d处\n",judge);

else if(judge==0)

printf("e不存在\n");

else printf("L不存在\n");

getchar();getchar();

break;

case 8:

printf("输入你想查找前驱的元素\n");

scanf("%d",&e);

judge=PriorElem(Lists.elem[elemnum-1].L,e,&pre);

if(judge==OK)

printf("前驱为%d，储存在pre中",pre);

else if(judge==ERROR)

printf("没有前驱");

else printf("L不存在");

getchar();getchar();

break;

case 9:

printf("输入你想查找后驱的元素\n");

scanf("%d",&e);

judge=NextElem(Lists.elem[elemnum-1].L,e,&next);

if(judge==OK)

printf("后驱为%d，储存在next中",next);

else if(judge==ERROR)

printf("没有后驱");

else printf("L不存在");

getchar();getchar();

break;

case 10:

printf("输入你想要插入的元素\n");

scanf("%d",&e);

if(ListLength(Lists.elem[elemnum-1].L)==0)

printf("因为无元素，所以直接作为第一个元素插入。\n");

else{printf("你想要插入在第几个元素之前\n");

scanf("%d",&i);}

judge=ListInsert(&Lists.elem[elemnum-1].L,i,e);

if(judge==OK)

printf("元素插入成功\n");

else if(judge==ERROR)

printf("插入位置不正确\n");

else printf("L不存在\n");

getchar();getchar();

break;

case 11:

printf("输入想要删除的元素序号！\n");

scanf("%d",&i);

judge=ListDelete(&Lists.elem[elemnum-1].L,i,&e);

if(judge==OK)

printf("元素删除成功，删除元素%d保存在e中\n",e);

else if(judge==ERROR)

printf("删除位置不正确\n");

else printf("L不存在\n");

getchar();getchar();

break;

case 12:

if(!ListTrabverse(Lists.elem[elemnum-1].L)) printf("线性表是空表！\n");

getchar();getchar();

break;

case 13:

printf("输入1以进行读入操作，输入2以进行读取操作。\n");

scanf("%d",&judge1);

printf("输入文件名。\n");

scanf("%s",filename);

if(judge1==1)

{

if(SaveList(Lists.elem[elemnum-1].L,filename)==OK)

printf("读入成功。\n");

else printf("读入失败。\n");

}

else if(judge1==2)

{

judge=LoadList(&Lists.elem[elemnum-1].L,filename);

if(judge==OK)

printf("读取成功。\n");

else if(judge==ERROR)

printf("线性表不存在。\n");

else printf("读取失败。\n");

}

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

break;

case 0:

break;

}

}

printf("欢迎下次再使用本系统！\n");

return 0;

} //end of main()

“def.h”

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

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -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;

typedef struct{

struct{char name[30];

SqList L;

}elem[10];

int length; //因为已经确认elem[10]，所以长度已经确定为10.

int listsize;

}LISTS;

“func.h”

#include"def.h"

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

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

status InitList(SqList \*L){

if(L->elem)

return ERROR;

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

if(!L->elem) exit(OVERFLOW);

L->length=0;

L->listsize=LIST\_INIT\_SIZE;

return OK;

}

status ListTrabverse(SqList L){

int i;

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

return L.length;

}

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。

{

if(L->elem)

{

free(L->elem);

L->elem=NULL;

L->length=0;

L->listsize=0;

return OK;

}

else

return INFEASIBLE;

}

int ListLength(SqList L)

{

if(L.elem)

return L.length;

else

return INFEASIBLE;

}

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INFEASIBLE。

{

if(L->elem==NULL)

return INFEASIBLE;

L->length=0;

return OK;

}

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INFEASIBLE。

{

if(L.elem==NULL)

return INFEASIBLE;

else

{

if(L.length)

return FALSE;

else

return TRUE;

}

}

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INFEASIBLE。

{

if(L.elem==NULL)

return INFEASIBLE;

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

return ERROR;

\*e=\*(L.elem+i-1);

return OK;

}

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INFEASIBLE。

{

if(L.elem==NULL)

return INFEASIBLE;

ElemType \*p;

int i=1;

p=L.elem;

while(i<=L.length&&!(\*p++==e)) i++;

if(i <= L.length)

return i; //返回位置

else

return ERROR;

}

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INFEASIBLE。

{

if(L.elem==NULL)

return INFEASIBLE;

ElemType \*p;

int i=1;

p=L.elem;

if(\*(L.elem)==e)

return ERROR;

if(L.length==1)

return ERROR;

while(\*p!=e&&i<L.length)

{p++;i++;}

if(i>0&&i<=L.length)

{

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

return OK;

}

else

{

return ERROR;

}

}

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INFEASIBLE。

{

if(L.elem==NULL)

return INFEASIBLE;

if(L.length==1)

return ERROR;

ElemType \*p,i=1;

p=L.elem;

while(\*p!=e&&i<=L.length)

{p++;i++;}

if(i==L.length||i==L.length+1)

return ERROR;

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

return OK;

}

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INFEASIBLE。

{

if(L->elem==NULL)

return INFEASIBLE;

ElemType \*newbase,\*q,\*p;

if(L->length==0)

{

L->elem[0]=e;

L->length++;

return OK;

}

if(i<1||i>L->length+1) //L->length初始为0

return ERROR;

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

{

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

if(!newbase)

return ERROR;

L->elem=newbase; //新基址

L->listsize+=LISTINCREMENT; //增加存储容量

}

q=L->elem+i-1; // q为插入位置

for(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)

// 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INFEASIBLE。

{

if(L->elem==NULL)

return INFEASIBLE;

ElemType \*q,\*p;

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

return ERROR;

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

q=L->elem+i-1;

\*e=\*q;

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

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

--L->length;

return OK;

}

status SaveList(SqList L,char FileName[])

// 如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

{

if(L.elem==NULL)

return INFEASIBLE;

ElemType i=0;

FILE \*fp1;

if((fp1=fopen(FileName,"w+"))==NULL)

return ERROR;

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

{

fprintf(fp1,"%d ",L.elem[i]);

}

fclose(fp1);

return OK;

}

status LoadList(SqList \*L,char FileName[])

// 如果线性表L不存在，将FileName文件中的数据读入到线性表L中，返回OK，否则返回INFEASIBLE。

{

FILE \*fp2;

if((fp2=fopen(FileName,"r+"))==NULL)

return ERROR;

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

if(!L->elem)

return ERROR;

L->length=0;

L->listsize=LIST\_INIT\_SIZE;

ElemType i=0;

while((fscanf(fp2,"%d",&L->elem[i])!=EOF))

i++;

L->length+=i;

fclose(fp2);

return OK;

}

status AddList(LISTS \*Lists,char ListName[])

// 只需要在Lists中增加一个名称为ListName的空线性表，线性表数据又后台测试程序插入。

{

int j=-1;

SqList L1;

L1.elem=(ElemType \*)malloc(LIST\_INIT\_SIZE\*sizeof(ElemType));

L1.length=0;

L1.listsize=LIST\_INIT\_SIZE;

Lists->elem[Lists->length].L=L1;

while(Lists->elem[Lists->length].name[j]=ListName[++j]);

if(Lists->elem[Lists->length].L.elem==NULL)

return INFEASIBLE;

else

{

Lists->length++;

return OK;

}

}

status RemoveList(LISTS \*Lists,char ListName[])

// Lists中删除一个名称为ListName的线性表

{

int i,j,flag;

char ch;

for(i=0;i<Lists->length;i++) //p查找需要删除的线性表

{

j=0;

flag=1;

while(ch=ListName[j]) //判断线性表名是否符合要求

{

if((Lists->elem[i]).name[j]!=ch)

flag=0;

j++;

}

if(flag==1)

{

DestroyList(&Lists->elem[i].L);

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

{

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

}

Lists->length--;

return OK;

}

}

return ERROR;

}

int LocateList(LISTS Lists,char ListName[])

// 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回0

{

int i,j,flag;

char ch;

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

{

j=0;

flag=1;

while(ch=ListName[j])

{

if((Lists.elem[i]).name[j]!=ch)

flag=0;

j++;

}

if(flag==1)

return i+1;

}

if(i==Lists.length)

return 0;

}

# 附录B 基于链式存储结构线性表实现的源程序

“main.cpp”

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include"func.h"

int main(void)

{

int op,op1=1;

char charname[10],filename[20];

LISTS Lists;

Lists.length=0;

int judge,judge1; //存储返回值的中间量

int elemnum; //存储选取线性表的序号

int e; //寄存元素的变量

int i; //需要使用的序号变量

for(i=0;i<LISTS\_INIT\_SIZE; i++) {

Lists.elem[i].L = NULL;

}

int pre,next; //前驱与后驱存放变量

while(op1){

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

printf(" 基于顺序结构的线性表基本操作演示系统\n");

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

printf(" 1.显示全部线性表\n");

printf(" 2.增加一个线性表\n");

printf(" 3.删除一个线性表\n");

printf(" 4.选择线性表进行操作\n");

printf(" 0.退出系统\n");

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

printf(" 请选择你的操作[0~4]:");

scanf("%d",&op1);

switch(op1){

case 1:

if(Lists.length==0)

printf("无线性表\n");

else

{

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

{

printf("%s ",Lists.elem[i].name);

ListTraverse(Lists.elem[i].L);

printf("\n");

}

}

getchar();getchar();

break;

case 2:

printf("请输入你要创建的线性表名\n");

scanf("%s",charname);

judge=LocateList(Lists,charname);

if(judge)

printf("已存在同名表。\n");

else

{

judge1=AddList(Lists,charname);

if(judge1==OK)

printf("创建线性表成功\n");

else printf("创建线性表失败\n");

}

getchar();getchar();

break;

case 3:

printf("请输入你要删除的线性表名\n");

scanf("%s",charname);

judge1=RemoveList(Lists,charname);

if(judge1==OK)

printf("删除成功\n");

else printf("删除失败\n");

getchar();getchar();

break;

case 4:

op=1;

printf("输入选择的线性表\n");

scanf("%s",charname);

elemnum=LocateList(Lists,charname);

if(elemnum)

{

while(op){

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

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

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

printf(" 1. ClearList 7.PriorElem\n");

printf(" 2. ListEmpty 8. NextElem\n");

printf(" 3. ListLength 9. ListInsert \n");

printf(" 4. GetElem 10.ListDelete\n");

printf(" 5. LocateElem 11.fileoperateReturn\n");

printf(" 6.ListTraverse 0.Return\n");

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

printf(" 请选择你的操作[0~9]:");

scanf("%d",&op);

switch(op){

case 1:

if(ClearList(Lists.elem[elemnum-1].L)==OK) printf("线性表清除成功！\n");

else printf( "线性表清除失败！\n");

getchar();getchar();

break;

case 2:

judge=ListEmpty(Lists.elem[elemnum-1].L);

if(judge==TRUE) printf("线性表为空！\n");

else if(judge==FALSE) printf("线性表非空！\n");

else printf("线性表不存在！\n");

getchar();getchar();

break;

case 3:

judge=ListLength(Lists.elem[elemnum-1].L);

if(judge>=0) printf("线性表长度为%d\n",judge);

else printf("线性表不存在！\n");

getchar();getchar();

break;

case 4:

printf("输入你想获取的元素序号。\n");

scanf("%d",&i);

judge=GetElem(Lists.elem[elemnum-1].L,i,e);

if(judge==OK) printf("成功提取出元素，第%d个元素为%d\n",i,e);

else if(judge==ERROR) printf("输入i不合法呢！\n");

else printf("L不存在呢！\n");

getchar();getchar();

break;

case 5:

printf("输入你想查找的元素\n");

scanf("%d",&e);

judge=LocateElem(Lists.elem[elemnum-1].L,e);

if(judge>0)

printf("查找元素在位置%d处\n",judge);

else if(judge==0)

printf("e不存在\n");

else printf("L不存在\n");

getchar();getchar();

break;

case 6:

if(ListTraverse(Lists.elem[elemnum-1].L)==1);

else

printf("线性表无元素");

getchar();getchar();

break;

case 7:

printf("输入你想查找前驱的元素\n");

scanf("%d",&e);

judge=PriorElem(Lists.elem[elemnum-1].L,e,pre);

if(judge==OK)

printf("前驱为%d，储存在pre中",pre);

else if(judge==ERROR)

printf("没有前驱");

else printf("L不存在");

getchar();getchar();

break;

case 8:

printf("输入你想查找后驱的元素\n");

scanf("%d",&e);

judge=NextElem(Lists.elem[elemnum-1].L,e,next);

if(judge==OK)

printf("后驱为%d，储存在next中",next);

else if(judge==ERROR)

printf("没有后驱");

else printf("L不存在");

getchar();getchar();

break;

case 9:

printf("输入你想要插入的元素\n");

scanf("%d",&e);

if(ListLength(Lists.elem[elemnum-1].L)==0)

{

printf("因为无元素，所以直接作为第一个元素插入。\n");

i=1;

}

else{printf("你想要插入在第几个元素之前\n");

scanf("%d",&i);}

judge=ListInsert(Lists.elem[elemnum-1].L,i,e);

if(judge==OK)

printf("元素插入成功\n");

else if(judge==ERROR)

printf("插入位置不正确\n");

else printf("L不存在\n");

getchar();getchar();

break;

case 10:

printf("输入想要删除的元素序号！\n");

scanf("%d",&i);

judge=ListDelete(Lists.elem[elemnum-1].L,i,e);

if(judge==OK)

printf("元素删除成功，删除元素%d保存在e中\n",e);

else if(judge==ERROR)

printf("删除位置不正确\n");

else printf("L不存在\n");

getchar();getchar();

break;

case 11:

printf("输入1以进行读入操作，输入2以进行读取操作。\n");

scanf("%d",&judge1);

printf("输入文件名。\n");

scanf("%s",filename);

if(judge1==1)

{

if(SaveList(Lists.elem[elemnum-1].L,filename)==OK)

printf("读入成功。\n");

else printf("读入失败。\n");

}

else if(judge1==2)

{

judge=LoadList(Lists.elem[elemnum-1].L,filename);

if(judge==OK)

printf("读取成功。\n");

else if(judge==ERROR)

printf("文件不存在。\n");

else printf("读取失败。\n");

}

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

} //end of if(elemnum)

else

{

printf("无该线性表\n");

getchar();getchar();

break;

}

case 0:

break;

}

}

printf("欢迎下次再使用本系统！\n");

return 0;

} //end of main()

“func.cpp”

#include "func.h"

status InitList(LinkList &L)

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

{

if(L)

return INFEASIBLE;

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

L->next=NULL;

return OK;

}

status DestroyList(LinkList &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。

{

if(L==NULL)

return INFEASIBLE;

LinkList p;

while(L)

{

p=L->next;

free(L);

L=p;

}

return OK;

}

status ClearList(LinkList &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INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

LinkList p=L->next,q;

while(p)

{

q=p->next;

free(p);

p=q;

}

L->next=NULL;

return OK;

}

status ListEmpty(LinkList 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INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

else if(L->next==NULL)

return TRUE;

else return FALSE;

}

int ListLength(LinkList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

LinkList p=L;

int i=0; //循环计数变量

while(p)

{

p=p->next;

i++;

}

return i-1;

}

status GetElem(LinkList 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INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

LinkList p=L;

if(i<1)

return ERROR;

while(p&&(i>=0))

{

if(i==0)

{

e=p->data;

return OK;

}

p=p->next;

i--;

}

if(i)

return ERROR;

}

status LocateElem(LinkList 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；如果e不存在，返回ERROR；当线性表L不存在时，返回INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

LinkList p=L->next;

int i=0; //位置序号

while(p)

{

i++;

if(p->data==e)

return i;

p=p->next;

}

if(p==NULL)

return ERROR;

}

status PriorElem(LinkList 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INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

LinkList p=L->next,q=L;

if(p==NULL)

return ERROR;

if(p->data==e)

return ERROR;

q=p;p=p->next;

while(p)

{

if(p->data==e)

{

pre=q->data;

return OK;

}

q=p;

p=p->next;

}

if(p==NULL)

return ERROR;

}

status NextElem(LinkList 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INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

LinkList p=L->next,q=L;

if(p==NULL) //线性表判空

return ERROR;

while(p)

{

q=p;

p=p->next;

if(q->data==e&&p!=NULL)

{

next=p->data;

return OK;

}

}

if(p==NULL)

return ERROR;

}

status ListInsert(LinkList &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INFEASIBLE。

{

if (!L)

return INFEASIBLE;

LinkList p = L;

while (p && i-- > 0) {

if (i == 0) {

LinkList temp = p->next;

p->next = (LinkList)malloc(sizeof(LNode));

p->next->data = e;

p->next->next = temp;

return OK;

}

p = p->next;

}

return ERROR;

}

status ListDelete(LinkList &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INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

LinkList p=L->next,q=L;

int j=1; //循环变量

if(i<1)

return ERROR;

while(p)

{

if(j==i)

{

e=p->data;

q->next=p->next;

free(p);

return OK;

}

else

{

q=p;

p=p->next;

j++;

}

}

if(p==NULL)

return ERROR;

}

status ListTraverse(LinkList 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。

{

if(L==NULL)

return INFEASIBLE;

LinkList p=L->next;

if(p==NULL)

return OK;

while(p)

{

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

p=p->next;

}

return OK;

}

status SaveList(LinkList L,char FileName[])

// 如果线性表L存在，将线性表L的的元素写到FileName文件中，返回OK，否则返回INFEASIBLE。

{

if(L==NULL)

return INFEASIBLE;

FILE \*fp1;

LinkList p=L->next;

fp1=fopen(FileName,"w+");

while(p)

{

fprintf(fp1,"%d ",p->data);

p=p->next;

}

fclose(fp1);

return OK;

}

status LoadList(LinkList &L,char FileName[])

// 线性表存在就直接覆盖

{

/\*if(L)

return INFEASIBLE;\*/

FILE \*fp2;

if((fp2=fopen(FileName,"r"))==NULL)

return ERROR;

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

LinkList p,q=L,s;

while(!feof(fp2))

{

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

fscanf(fp2,"%d",&p->data);

s=q;

q->next=p;

q=p;

}

free(p);

s->next=NULL;

fclose(fp2);

return OK;

}

status AddList(LISTS &Lists,char ListName[])

// 只需要在Lists中增加一个名称为ListName的空线性表，线性表数据又后台测试程序插入。

{

if(Lists.length==LISTS\_INIT\_SIZE)

return ERROR;

int i = 0;

do {

Lists.elem[Lists.length].name[i]=ListName[i];

} while (ListName[i++]);

InitList(Lists.elem[Lists.length].L);

Lists.length++;

return OK;

}

status RemoveList(LISTS &Lists,char ListName[])

// Lists中删除一个名称为ListName的线性表

{

int i,j,flag;

char ch;

for(i=0;i<Lists.length;i++) //p查找需要删除的线性表

{

j=0;

flag=1;

while(ch=ListName[j]) //判断线性表名是否符合要求

{

if((Lists.elem[i]).name[j]!=ch)

flag=0;

j++;

}

if(flag==1)

{

DestroyList(Lists.elem[i].L);

for(j=i;j<(Lists.length-1);j++)

{

Lists.elem[j]=Lists.elem[j+1];

}

Lists.length--;

return OK;

}

}

return ERROR;

}

int LocateList(LISTS Lists,char ListName[])

// 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回0

{

int i,j,flag;

char ch;

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

{

j=0;

flag=1;

while(ch=ListName[j])

{

if((Lists.elem[i]).name[j]!=ch)

flag=0;

j++;

}

if(flag==1)

return i+1;

}

if(i==Lists.length)

return 0;

}

“func.h”

#include <stdio.h>

#include <stdlib.h>

#include"def.h"

status InitList(LinkList &L);

status DestroyList(LinkList &L);

status ClearList(LinkList &L);

status ListEmpty(LinkList L);

int ListLength(LinkList L);

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

status LocateElem(LinkList L,ElemType e);

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

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

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

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

status ListTraverse(LinkList L);

status SaveList(LinkList L,char FileName[]);

status LoadList(LinkList &L,char FileName[]);

status AddList(LISTS &Lists,char ListName[]);

status RemoveList(LISTS &Lists,char ListName[]);

int LocateList(LISTS Lists,char ListName[]);

“def.h”

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -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

#define LISTS\_INIT\_SIZE 10

typedef int ElemType;

typedef struct LNode{ //单链表（链式结构）结点的定义

ElemType data;

struct LNode \*next;

}LNode,\*LinkList;

typedef struct{

struct{

char name[30];

LinkList L;

}elem[LISTS\_INIT\_SIZE];

int length; //因为已经确认elem[10]，所以长度已经确定为10.

int listsize;

}LISTS;

# 附录C 基于二叉链表二叉树实现的源程序

“main.cpp”

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include"func.h"

int main(void)

{

int op,op1=1;

char charname[10],filename[20];

LISTS Lists;

Lists.length=0;

int judge,judge1; //存储返回值的中间量

int elemnum; //存储选取二叉树的序号

int i; //需要使用的序号变量

BiTree tempT; //中间存储二叉树指针

for(i=0;i<LISTS\_INIT\_SIZE; i++) {

Lists.elem[i].T = NULL;

}

while(op1){

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

printf(" 基于二叉树的基本操作演示系统\n");

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

printf(" 1.显示全部二叉树名字\n");

printf(" 2.增加一个二叉树\n");

printf(" 3.删除一个二叉树\n");

printf(" 4.选择二叉树进行操作\n");

printf(" 0.退出系统\n");

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

printf(" 请选择你的操作[0~4]:");

scanf("%d",&op1);

switch(op1){

case 1:

if(Lists.length==0)

printf("无二叉树\n");

else

{

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

{

printf("%s ",Lists.elem[i].name);

printf("\n");

}

}

getchar();getchar();

break;

case 2:

printf("请输入你要创建的二叉树名\n");

scanf("%s",charname);

judge=LocateList(Lists,charname);

if(judge)

printf("已存在同名树。\n");

else

{

judge1=AddList(Lists,charname);

if(judge1==OK)

printf("创建二叉树成功\n");

else printf("创建二叉树失败\n");

}

getchar();getchar();

break;

case 3:

printf("请输入你要删除的二叉树名\n");

scanf("%s",charname);

judge1=RemoveList(Lists,charname);

if(judge1==OK)

printf("删除成功\n");

else printf("删除失败\n");

getchar();getchar();

break;

case 4:

op=1;

printf("输入选择的二叉树\n");

scanf("%s",charname);

elemnum=LocateList(Lists,charname);

if(elemnum)

{

while(op){

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

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

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

printf(" 1.求二叉树深度 2.查找结点\n");

printf(" 3.结点赋值 4.获得兄弟结点\n");

printf(" 5.插入结点 6.删除节点 \n");

printf(" 7.先序遍历 8.中序遍历\n");

printf(" 9.后序遍历 10.按层遍历\n");

printf(" 11.文件读写 0.返回初级菜单\n");

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

printf(" 请选择你的操作[0~9]:");

scanf("%d",&op);

switch(op){

case 1:

judge=BiTreeDepth(Lists.elem[elemnum-1].T);

printf( "二叉树深度为%d\n",judge);

getchar();getchar();

break;

case 2:

printf("输入你想获取的元素的关键字。\n");

scanf("%d",&i);

tempT=LocateNode(Lists.elem[elemnum-1].T,i);

if(tempT) printf("成功提取，为%d,%s\n",tempT->data.key,tempT->data.others);

else printf("查找失败\n");

getchar();getchar();

break;

case 3:

TElemType value;

printf("输入想要赋值的结点关键字。\n");

scanf("%d",&i);

printf("输入修改的value.\n");

scanf("%d%s",&value.key,value.others);

judge=Assign(Lists.elem[elemnum-1].T,i,value);

if(judge) printf("已成功赋值\n");

else printf("赋值失败！\n");

getchar();getchar();

break;

case 4:

printf("输入你想获取兄弟节点的结点关键字。\n");

scanf("%d",&i);

tempT=GetSibling(Lists.elem[elemnum-1].T,i);

if(tempT) printf("成功提取，为%d,%s\n",tempT->data.key,tempT->data.others);

else printf("查找失败\n");

getchar();getchar();

break;

case 5:

printf("输入e、LR、c\n");

printf("e是和T中结点关键字类型相同的给定值,LR为0或1,c是待插入结点\n");

printf("根据LR为0或者1，插入结点c到T中,作为关键字为e的结点的左或右孩子结点，结点e的原有左子树或右子树则为结点c的右子树\n");

printf("当LR为-1时，作为根结点插入,原根结点作为c的右子树。\n");

int e,LR;

TElemType c;

scanf("%d%d%d%s",&e,&LR,&c.key,c.others);

judge=InsertNode(Lists.elem[elemnum-1].T,e,LR,c);

if(judge)

printf("插入成功\n");

else printf("插入结点失败\n");

getchar();getchar();

break;

case 6:

printf("输入想要删除的节点关键字\n");

scanf("%d",&i);

judge=DeleteNode(Lists.elem[elemnum-1].T,i);

if(judge)

printf("结点删除成功\n");

else printf("无该结点\n");

getchar();getchar();

break;

case 7:

printf("先序遍历结果如下：\n");

PreOrderTraverse(Lists.elem[elemnum-1].T,visit);

printf("\n");

getchar();getchar();

break;

case 8:

printf("中序遍历结果如下\n");

InOrderTraverse(Lists.elem[elemnum-1].T,visit);

printf("\n");

getchar();getchar();

break;

case 9:

printf("后序遍历结果如下\n");

PostOrderTraverse(Lists.elem[elemnum-1].T,visit);

printf("\n");

getchar();getchar();

break;

case 10:

printf("按层遍历结果如下\n");

LevelOrderTraverse(Lists.elem[elemnum-1].T,visit);

printf("\n");

getchar();getchar();

break;

case 11:

printf("输入1以进行读入操作，输入2以进行读取操作。\n");

scanf("%d",&judge1);

printf("输入文件名。\n");

scanf("%s",filename);

if(judge1==1)

{

if(SaveBiTree(Lists.elem[elemnum-1].T,filename)==OK)

printf("读入成功。\n");

else printf("读入失败。\n");

}

else if(judge1==2)

{

judge=LoadBiTree(Lists.elem[elemnum-1].T,filename);

if(judge==OK)

printf("读取成功。\n");

else if(judge==ERROR)

printf("文件不存在。\n");

else printf("读取失败。\n");

}

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

} //end of if(elemnum)

else

{

printf("无该二叉树\n");

getchar();getchar();

break;

}

case 0:

break;

}

}

printf("欢迎下次再使用本系统！\n");

return 0;

} //end of main()

“func.cpp”

#include "func.h"

int hasDuplicates(TElemType\* definition) //判断是否重复

{

for (int i = 0; definition[i].key != -1; i++) {

for (int j = i + 1; definition[j].key != -1; j++) {

if (definition[i].key == definition[j].key && definition[i].key)

return TRUE;

}

}

return FALSE;

}

void visit(BiTree T)

{

printf(" %d,%s",T->data.key,T->data.others);

}

status CreateBiTree(BiTree &T,TElemType definition[])

/\*根据带空枝的二叉树先根遍历序列definition构造一棵二叉树，将根节点指针赋值给T并返回OK，

如果有相同的关键字，返回ERROR。此题允许通过增加其它函数辅助实现本关任务\*/

{

if (hasDuplicates(definition))

return ERROR;

int i=0;

BiTree st[20];

int nodenum[20]={0};

int top=0,root=0;

while(definition[i].key!=-1)

{

if(definition[i].key==0)

nodenum[i]=2;

st[i]=(BiTNode\*)malloc(sizeof(BiTNode));

st[i]->data=definition[i];

st[i]->lchild=NULL;

st[i]->rchild=NULL;

i++;

}

T=st[0];

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

{

if(st[top]->data.key!=0)

{

if(nodenum[root]==0)

{

nodenum[root]++;

st[root]->lchild=st[top];

root=top;

}

if(nodenum[root]==1)

{

nodenum[root]++;

st[root]->rchild=st[top];

root=top;

}

}

if(st[top]->data.key==0)

{

nodenum[root]++;

nodenum[top]=2;

}

while(nodenum[root]==2&&root>0)

root--;

}

return OK;

}

status ClearBiTree(BiTree &T)

//将二叉树设置成空，并删除所有结点，释放结点空间

{

if(T==NULL)

return 0;

if(T->lchild)

{

ClearBiTree(T->lchild);

T->lchild=NULL;

}

if(T->rchild)

{

ClearBiTree(T->rchild);

T->rchild=NULL;

}

if(T)

{

free(T);

T=NULL;

}

}

int BiTreeDepth(BiTree T)

//求二叉树T的深度

{

int i,j; //分别存储左右的深度

if(T)

{

i=BiTreeDepth(T->lchild);

j=BiTreeDepth(T->rchild);

if(i>=j)

return i+1;

else

return j+1;

}

else

return 0;

}

BiTNode\* LocateNode(BiTree T,KeyType e)

//查找结点

{

if (!T)

return NULL;

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

return T;

BiTree temp = LocateNode(T->lchild,e);

if(temp)

return temp;

return LocateNode(T->rchild, e);

}

status Assign(BiTree& T, KeyType e, TElemType value)

{

//实现结点赋值。此题允许通过增加其它函数辅助实现本关任务

BiTree p=LocateNode(T, e);

if(!p) return ERROR;

BiTree duplicate = LocateNode(T,value.key);

if (duplicate&&p!=duplicate)

return ERROR;

p->data = value;

return OK;

}

BiTNode\* GetSibling(BiTree T,KeyType e)

//实现获得兄弟结点

{

if(!T)

return NULL;

if(T->lchild&&T->lchild->data.key==e)

return T->rchild;

if(T->rchild&&T->rchild->data.key==e)

return T->lchild;

BiTree temp=GetSibling(T->lchild,e);

if(temp)

return temp;

return GetSibling(T->rchild,e);

}

status InsertNode(BiTree &T,KeyType e,int LR,TElemType c)

//插入结点。此题允许通过增加其它函数辅助实现本关任务

{

BiTree temp;

temp=LocateNode(T,e);

if(!temp)

return ERROR;

if(e==c.key)

return ERROR;

if(LR==0)

{

BiTree newnode;

newnode=(BiTree)malloc(sizeof(BiTNode));

newnode->data=c;

newnode->lchild=NULL;

newnode->rchild=temp->lchild;

temp->lchild=newnode;

return OK;

}

if(LR==1)

{

BiTree newnode;

newnode=(BiTree)malloc(sizeof(BiTNode));

newnode->data=c;

newnode->lchild=NULL;

newnode->rchild=temp->rchild;

temp->rchild=newnode;

return OK;

}

if(LR==-1)

{

BiTree newnode;

newnode=(BiTree)malloc(sizeof(BiTNode));

newnode->data=c;

newnode->lchild=NULL;

newnode->rchild=T;

T=newnode;

return OK;

}

}

BiTNode\* GetParent(BiTree T,KeyType e)

{

if(!T)

return NULL;

if(T->lchild)

{

if(T->lchild->data.key==e)

return T;

}

if(T->rchild)

{

if(T->rchild->data.key==e)

return T;

}

BiTree temp=GetParent(T->lchild,e);

if(temp)

return temp;

return GetParent(T->rchild,e);

}

status DeleteNode(BiTree &T,KeyType e)

//删除结点。此题允许通过增加其它函数辅助实现本关任务

{

BiTree temp=LocateNode(T,e);

if(!temp)

return ERROR;

BiTree father=GetParent(T,e);

if(!temp->lchild&&!temp->rchild)

{

if(!father)

{

free(temp);

temp=NULL;

return OK;

}

if(father->lchild->data.key==temp->data.key)

father->lchild=NULL;

if(father->rchild->data.key==temp->data.key)

father->rchild=NULL;

free(temp);

temp=NULL;

return OK;

}

if(temp->lchild&&!temp->rchild)

{

BiTree temp2=temp->lchild;

temp->lchild=temp->lchild->lchild;

temp->rchild=temp->lchild->rchild;

temp->data=temp->lchild->data;

free(temp2);

temp2=NULL;

return OK;

}

if(temp->rchild&&!temp->lchild)

{

BiTree temp2=temp->rchild;

temp->lchild=temp->rchild->lchild;

temp->data=temp->rchild->data;

temp->rchild=temp->rchild->rchild;

free(temp2);

temp2=NULL;

return OK;

}

if(temp->lchild&&temp->rchild)

{

BiTree temp2=temp->lchild;

BiTree ptr=temp->lchild;

while(ptr->rchild)

{

ptr=ptr->rchild;

}

ptr->rchild=temp->rchild;

temp->rchild=temp->lchild->rchild;

temp->data=temp->lchild->data;

temp->lchild=temp->lchild->lchild;

free(temp2);

temp2=NULL;

return OK;

}

}

status PreOrderTraverse(BiTree T,void (\*visit)(BiTree))

//先序遍历二叉树T（递归）

{

if (T)

{

visit(T);

PreOrderTraverse(T->lchild,visit);

PreOrderTraverse(T->rchild,visit);

}

}

status InOrderTraverse(BiTree T,void (\*visit)(BiTree))

//中序遍历二叉树T（非递归）

{

/\*if(T) //递归算法

{

InOrderTraverse(T->lchild,visit);

visit(T);

InOrderTraverse(T->rchild,visit);

}\*/

BiTree st[100];

int top=0;

do{

while(T){

if(top==100)

exit(OVERFLOW);

st[top++]=T;

T=T->lchild;

}

if(top)

{

T=st[--top];

visit(T);

T=T->rchild;

}

}while(top||T);

}

status PostOrderTraverse(BiTree T,void (\*visit)(BiTree))

//后序遍历二叉树T

{

if(T)

{

PostOrderTraverse(T->lchild,visit);

PostOrderTraverse(T->rchild,visit);

visit(T);

}

}

status LevelOrderTraverse(BiTree T,void (\*visit)(BiTree))

//按层遍历二叉树T

{

BiTree temp[100];

int inp=0;int oup=0; //input&output

temp[inp++]=T;

while(inp>oup)

{

if(temp[oup])

{

visit(temp[oup]);

temp[inp++]=temp[oup]->lchild;

temp[inp++]=temp[oup]->rchild;

}

oup++;

}

}

status PreOrderSave(BiTree T, FILE\* fp)

{

//先序遍历输出二叉树T

if (!T) {

fprintf(fp, "0 NULL ");

return OK;

} else

fprintf(fp, "%d %s ", T->data.key, T->data.others);

PreOrderSave(T->lchild, fp);

PreOrderSave(T->rchild, fp);

return OK;

}

int PreOrderLoad(BiTree f, FILE\* fp, int flag)

{

//文件读取先序创建二叉树T

TElemType data;

if (fscanf(fp, "%d %s", &data.key, data.others) != 2)

return OK;

if (data.key == 0) {

if (flag) {

f->rchild = NULL;

} else {

f->lchild = NULL;

}

return OK;

}

BiTree child = (BiTree)malloc(sizeof(BiTNode));

child->data.key = data.key;

strcpy(child->data.others, data.others);

if (flag) {

f->rchild = child;

} else {

f->lchild = child;

}

PreOrderLoad(child, fp, 0);

PreOrderLoad(child, fp, 1);

return OK;

}

status SaveBiTree(BiTree T, char FileName[])

{

//将二叉树的结点数据写入到文件FileName中

FILE\* fp = NULL;

if ((fp = fopen(FileName, "w")) == NULL) {

return ERROR;

}

status flag = PreOrderSave(T, fp);

fclose(fp);

return flag;

}

status LoadBiTree(BiTree& T, char FileName[])

{

//读入文件FileName的结点数据，创建二叉树

/\*if (T)

return ERROR;\*/

FILE\* fp = NULL;

if ((fp = fopen(FileName, "r")) == NULL) {

return ERROR;

}

TElemType data;

fscanf(fp, "%d", &data.key);

fscanf(fp, "%s", data.others);

if (data.key == 0) {

T = NULL;

return OK;

}

T = (BiTree)malloc(sizeof(BiTNode));

T->data.key = data.key;

strcpy(T->data.others, data.others);

PreOrderLoad(T, fp, 0);

PreOrderLoad(T, fp, 1);

fclose(fp);

return OK;

}

status AddList(LISTS &Lists,char ListName[])

// 只需要在Lists中增加一个名称为ListName的树，并输入创建数据。

{

if(Lists.length==LISTS\_INIT\_SIZE)

return ERROR;

int i = 0;

do {

Lists.elem[Lists.length].name[i]=ListName[i];

} while (ListName[i++]);

TElemType definition[100];

printf("输入先序遍历序列definition：\n");

int ans;

i=0;

do {

scanf("%d%s",&definition[i].key,definition[i].others);

} while (definition[i++].key!=-1);

ans=CreateBiTree(Lists.elem[Lists.length].T,definition);

if(ans==OK)

{

Lists.length++;

return OK;

}

else

return ERROR;

}

status RemoveList(LISTS &Lists,char ListName[])

// Lists中删除一个名称为ListName的线性表

{

int i,j,flag;

char ch;

for(i=0;i<Lists.length;i++) //p查找需要删除的线性表

{

j=0;

flag=1;

while(ch=ListName[j]) //判断线性表名是否符合要求

{

if((Lists.elem[i]).name[j]!=ch)

flag=0;

j++;

}

if(flag==1)

{

ClearBiTree(Lists.elem[i].T);

for(j=i;j<(Lists.length-1);j++)

{

Lists.elem[j]=Lists.elem[j+1];

}

Lists.length--;

return OK;

}

}

return ERROR;

}

int LocateList(LISTS Lists,char ListName[])

// 在Lists中查找一个名称为ListName的线性表，成功返回逻辑序号，否则返回0

{

int i,j,flag;

char ch;

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

{

j=0;

flag=1;

while(ch=ListName[j])

{

if((Lists.elem[i]).name[j]!=ch)

flag=0;

j++;

}

if(flag==1)

return i+1;

}

if(i==Lists.length)

return 0;

}

“func.h”

#include <stdio.h>

#include <stdlib.h>

#include"def.h"

#include "string.h"

int hasDuplicates(TElemType\* definition);

status CreateBiTree(BiTree &T,TElemType definition[]);

status ClearBiTree(BiTree &T);

int BiTreeDepth(BiTree T);

BiTNode\* LocateNode(BiTree T,KeyType e);

status Assign(BiTree& T, KeyType e, TElemType value);

BiTNode\* GetSibling(BiTree T,KeyType e);

status InsertNode(BiTree &T,KeyType e,int LR,TElemType c);

BiTNode\* GetParent(BiTree T,KeyType e);

status DeleteNode(BiTree &T,KeyType e);

status PreOrderTraverse(BiTree T,void (\*visit)(BiTree));

status InOrderTraverse(BiTree T,void (\*visit)(BiTree));

status PostOrderTraverse(BiTree T,void (\*visit)(BiTree));

status LevelOrderTraverse(BiTree T,void (\*visit)(BiTree));

status PreOrderSave(BiTree T, FILE\* fp);

int PreOrderLoad(BiTree f, FILE\* fp, int flag) ;

status SaveBiTree(BiTree T, char FileName[]);

status LoadBiTree(BiTree& T, char FileName[]);

void visit(BiTree T);

status AddList(LISTS &Lists,char ListName[]);

status RemoveList(LISTS &Lists,char ListName[]);

int LocateList(LISTS Lists,char ListName[]);

“def.h”

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LISTS\_INIT\_SIZE 10

typedef int status;

typedef int KeyType;

typedef struct {

KeyType key;

char others[20];

} TElemType; //二叉树结点类型定义

typedef struct BiTNode{ //二叉链表结点的定义

TElemType data;

struct BiTNode \*lchild,\*rchild;

} BiTNode, \*BiTree;

typedef struct{

struct{

char name[30];

BiTree T;

}elem[LISTS\_INIT\_SIZE];

int length; //因为已经确认elem[10]，所以长度已经确定为10.

int listsize;

}LISTS;

# 附录D 基于邻接表图实现的源程序

“main.cpp”

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include"func.h"

int main(void)

{

int op,op1=1;

char charname[10],filename[20];

GraghsLink Graghs;

Graghs=(GraghsLink)malloc(sizeof(GraghNode));

Graghs->next=NULL;

GraghsLink GraghLoc;

int judge,judge1; //存储返回值的中间量

int i; //需要使用的序号变量

KeyType v,w;

while(op1){

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

printf(" 图基本操作演示系统\n");

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

printf(" 1.显示全部图名字\n");

printf(" 2.增加一个图\n");

printf(" 3.删除一个图\n");

printf(" 4.选择图进行操作\n");

printf(" 0.退出系统\n");

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

printf(" 请选择你的操作[0~4]:");

scanf("%d",&op1);

switch(op1){

case 1:

if(GraghsLength(Graghs)==0)

printf("无图\n");

else

{

GraghsTraverse(Graghs);

printf("\n");

}

/\*if(Lists.length==0)

printf("无图\n");

else

{

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

{

printf("%s ",Lists.elem[i].name);

printf("\n");

}

}\*/

getchar();getchar();

break;

case 2:

printf("请输入你要创建的图名\n");

scanf("%s",charname);

/\*judge=LocateList(Lists,charname);

if(judge)

printf("已存在同名图。\n");

else

{

printf("输入图的顶点以及弧\n");

judge1=AddList(Lists,charname);

if(judge1==OK)

printf("创建图成功\n");

else printf("创建图失败\n");

}\*/

judge1=GraghsAdd(Graghs,charname);

if(judge1==OK)

printf("创建图成功\n");

else printf("创建图失败\n");

getchar();getchar();

break;

case 3:

printf("请输入你要删除的图名\n");

scanf("%s",charname);

judge1=GraghsRemove(Graghs,charname);

if(judge1==OK)

printf("删除成功\n");

else printf("删除失败\n");

getchar();getchar();

break;

case 4:

op=1;

printf("输入选择的图\n");

scanf("%s",charname);

GraghLoc=GraghsLocate(Graghs,charname);

if(GraghLoc)

{

while(op){

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

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

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

printf(" 1.查找顶点 2.顶点赋值\n");

printf(" 3.获得第一邻接点 4.获得下一邻接点\n");

printf(" 5.插入顶点 6.删除顶点 \n");

printf(" 7.插入弧 8.删除弧\n");

printf(" 9.深度优先搜索遍历 10.广度优先搜索遍历\n");

printf(" 11.文件读写 12.显示图\n");

printf(" 0.返回初级菜单\n");

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

printf(" 请选择你的操作[0~9]:");

scanf("%d",&op);

switch(op){

case 1:

printf("输入你想获取的顶点的关键字。\n");

scanf("%d",&i);

judge=LocateVex(GraghLoc->G,i);

if(judge==-1)

printf("查找失败。\n");

else printf("该顶点位序为%d\n",judge);

getchar();getchar();

break;

case 2:

VertexType value;

printf("输入想要赋值的顶点关键字。\n");

scanf("%d",&i);

if(LocateVex(GraghLoc->G,i)==-1)

{

printf("无该顶点。\n");

getchar();getchar();

break;

}

printf("输入修改的value.\n");

scanf("%d%s",&value.key,value.others);

judge=PutVex(GraghLoc->G,i,value);

if(judge) printf("已成功赋值\n");

else printf("赋值失败！\n");

getchar();getchar();

break;

case 3:

printf("输入你想获取第一邻接点的顶点关键字。\n");

scanf("%d",&i);

judge=FirstAdjVex(GraghLoc->G,i);

if(judge!=-1) printf("成功提取，位序为%d\n",judge);

else printf("查找失败\n");

getchar();getchar();

break;

case 4:

printf("输入你想获取下一邻接点的顶点关键字以及结点关键字。\n");

scanf("%d%d",&v,&w);

judge=NextAdjVex(GraghLoc->G,v,w);

if(judge!=-1) printf("成功提取，位序为%d\n",judge);

else printf("查找失败\n");

getchar();getchar();

break;

case 5:

printf("输入想要插入的顶点值\n");

scanf("%d%s",&value.key,value.others);

judge=InsertVex(GraghLoc->G,value);

if(judge)

printf("插入成功\n");

else printf("插入顶点失败\n");

getchar();getchar();

break;

case 6:

printf("输入想要删除的顶点关键字\n");

scanf("%d",&v);

judge=DeleteVex(GraghLoc->G,v);

if(judge)

printf("顶点删除成功\n");

else printf("删除失败\n");

getchar();getchar();

break;

case 7:

printf("输入想要插入的弧的两端关键字\n");

scanf("%d%d",&v,&w);

judge=InsertArc(GraghLoc->G,v,w);

if(judge)

printf("插入成功\n");

else printf("插入失败\n");

getchar();getchar();

break;

case 8:

printf("输入想要删除的弧的两端关键字\n");

scanf("%d%d",&v,&w);

judge=DeleteArc(GraghLoc->G,v,w);

if(judge)

printf("删除成功\n");

else printf("删除失败\n");

getchar();getchar();

break;

case 9:

printf("深度优先遍历结果如下\n");

DFSTraverse(GraghLoc->G,visit);

printf("\n");

getchar();getchar();

break;

case 10:

printf("广度优先遍历结果如下\n");

BFSTraverse(GraghLoc->G,visit);

printf("\n");

getchar();getchar();

break;

case 11:

printf("输入1以进行读入操作，输入2以进行读取操作。\n");

scanf("%d",&judge1);

printf("输入文件名。\n");

scanf("%s",filename);

if(judge1==1)

{

if(SaveGraph(GraghLoc->G,filename)==OK)

printf("读入成功。\n");

else printf("读入失败。\n");

}

else if(judge1==2)

{

judge=LoadGraph(GraghLoc->G,filename);

if(judge==OK)

printf("读取成功。\n");

else if(judge==ERROR)

printf("文件不存在。\n");

else printf("读取失败。\n");

}

getchar();getchar();

break;

case 12:

ShowGragh(GraghLoc->G);

printf("\n");

getchar();getchar();

break;

case 0:

break;

}//end of switch

}//end of while

} //end of if(elemnum)

else

{

printf("无该图\n");

getchar();getchar();

break;

}

case 0:

break;

}

}

printf("欢迎下次再使用本系统！\n");

return 0;

} //end of main()

“func.cpp”

#include "func.h"

int hasDuplicate(ALGraph G,KeyType u) //查重函数

{

int i;

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

{

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

return 0;

}

return 1;

}

void visit(VertexType v)

{

printf(" %d %s",v.key,v.others);

}

void ShowGragh(ALGraph G)

{

int i;

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

{

ArcNode \*p=G.vertices[i].firstarc;

printf("%d %s",G.vertices[i].data.key,G.vertices[i].data.others);

while (p)

{

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

p=p->nextarc;

}

printf("\n");

}

}

status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2])

/\*根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR

如果有相同的关键字，返回ERROR。此题允许通过增加其它函数辅助实现本关任务\*/

{

int i=0,j,one,two;

int a[100]={0}; //记录是否有重复或者未出现过的结点

while(V[i].key!=-1)

{

a[V[i].key]++;

if(a[V[i].key]>1)

return ERROR;

i++;

if(i>MAX\_VERTEX\_NUM)

return ERROR;

}

i=0;

do {

G.vertices[i].data=V[i];

G.vertices[i].firstarc=NULL;

} while (V[i++].key!=-1);

G.vexnum=i-1;

i=0;

while (VR[i][0]!=-1)

{

if(a[VR[i][0]]!=1||a[VR[i][1]]!=1)

return ERROR;

ArcNode \*p;

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

if (VR[i][0]==G.vertices[j].data.key)

{

one=j; break;

}

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

if (VR[i][1]==G.vertices[j].data.key)

{

two=j; break;

}

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

p->adjvex=two;p->nextarc=G.vertices[one].firstarc;G.vertices[one].firstarc=p;

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

p->adjvex=one;p->nextarc=G.vertices[two].firstarc;G.vertices[two].firstarc=p;

i++;

}

G.arcnum=i;

return OK;

}

status DestroyGraph(ALGraph &G)

/\*销毁无向图G\*/

{

int i;

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

{

free(G.vertices[i].firstarc);

G.vertices[i].firstarc=NULL;

}

G.vexnum=0;

G.arcnum=0;

return OK;

}

int LocateVex(ALGraph G,KeyType u)

//根据u在图G中查找顶点，查找成功返回位序，否则返回-1；

{

int i;

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

{

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

return i;

}

return -1;

}

status PutVex(ALGraph &G,KeyType u,VertexType value)

//根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK；

//如果查找失败或关键字不唯一，返回ERROR

{

int i;

if(hasDuplicate(G,value.key)==0)

return ERROR;

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

{

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

{

G.vertices[i].data=value;

return OK;

}

}

return ERROR;

}

int FirstAdjVex(ALGraph G,KeyType u)

//根据u在图G中查找顶点，查找成功返回顶点u的第一邻接顶点位序，否则返回-1；

{

// 请在这里补充代码，完成本关任务

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

int loc=LocateVex(G,u);

if(loc==-1)

return -1;

if(G.vertices[loc].firstarc==NULL)

return -1;

return G.vertices[loc].firstarc->adjvex;

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

}

int NextAdjVex(ALGraph G,KeyType v,KeyType w)

//根据v在图G中查找顶点，查找成功返回顶点v的邻接顶点相对于w的下一邻接顶点的位序，查找失败返回-1；

{

int loc=LocateVex(G,v);

if(loc==-1)

return -1;

int loc1=LocateVex(G,w);

if(loc1==-1)

return -1;

ArcNode\* temp=G.vertices[loc].firstarc;

while(loc1!=temp->adjvex)

temp=temp->nextarc;

if(!temp||!temp->nextarc)

return -1;

return temp->nextarc->adjvex;

}

status InsertVex(ALGraph &G,VertexType v)

//在图G中插入顶点v，成功返回OK,否则返回ERROR

{

if(hasDuplicate(G,v.key)==0)

return ERROR;

if(MAX\_VERTEX\_NUM==G.vexnum)

return ERROR;

G.vertices[G.vexnum].data=v;

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

G.vexnum++;

return OK;

}

int ArcDetect(ALGraph G,KeyType v,KeyType w)

{ //v和w为序列值而非关键字

ArcNode \*det;

det=G.vertices[v].firstarc;

while(det)

{

if(det->adjvex==w)

return OK;

det=det->nextarc;

}

return ERROR;

}

int DeleteArcNode(ALGraph &G,KeyType v,KeyType w)

{ //v和w为序列值而非关键字

ArcNode \*p,\*q;

p=G.vertices[v].firstarc;

if(p->adjvex==w)

{

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

free(p);

p=NULL;

return OK;

}

q=p;p=p->nextarc;

while(p)

{

if(p->adjvex==w)

{

q->nextarc=p->nextarc;

free(p);

p=NULL;

return OK;

}

q=p;p=p->nextarc;

}

return ERROR;

}

status DeleteVex(ALGraph &G,KeyType v)

//在图G中删除关键字v对应的顶点以及相关的弧，成功返回OK,否则返回ERROR

{

int loc=LocateVex(G,v);

if(loc==-1)

return ERROR;

if(G.vexnum<=1)

return ERROR;

int i;

for(i=0;i<G.vexnum;i++) //删除所有相关弧

DeleteArc(G,v,G.vertices[i].data.key);

ArcNode \*p;

for(i=0;i<G.vexnum;i++) //将弧中指向结点前移

{

p=G.vertices[i].firstarc;

while(p)

{

if(p->adjvex>loc)

p->adjvex--;

p=p->nextarc;

}

}

for(i=loc;i<G.vexnum-1;i++) //将结点补齐缺口

G.vertices[i]=G.vertices[i+1];

G.vexnum--;

return OK;

}

status InsertArc(ALGraph &G,KeyType v,KeyType w)

//在图G中增加弧<v,w>，成功返回OK,否则返回ERROR

{

int loc1=LocateVex(G,v);

int loc2=LocateVex(G,w);

if(loc1==-1||loc2==-1)

return ERROR;

ArcNode \*p,\*q;

if(ArcDetect(G,loc1,loc2))

return ERROR;

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

p->adjvex=loc2;

p->nextarc=G.vertices[loc1].firstarc;

G.vertices[loc1].firstarc=p;

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

q->adjvex=loc1;

q->nextarc=G.vertices[loc2].firstarc;

G.vertices[loc2].firstarc=q;

G.arcnum++;

return OK;

}

status DeleteArc(ALGraph &G,KeyType v,KeyType w)

//在图G中删除弧<v,w>，成功返回OK,否则返回ERROR

{

int loc1=LocateVex(G,v);

int loc2=LocateVex(G,w);

if(loc1==-1||loc2==-1)

return ERROR;

if(!ArcDetect(G,loc1,loc2))

return ERROR;

if(DeleteArcNode(G,loc1,loc2)&&DeleteArcNode(G,loc2,loc1))

return OK;

return ERROR;

}

int visited[MAX\_VERTEX\_NUM];

void DFS(ALGraph G,int i,void (\*visit)(VertexType))

{

visited[i]=1;

visit(G.vertices[i].data);

int j;

for(j=FirstAdjVex(G,G.vertices[i].data.key);j>=0;j=NextAdjVex(G,G.vertices[i].data.key,G.vertices[j].data.key))

if(!visited[j])

DFS(G,j,visit);

}

status DFSTraverse(ALGraph &G,void (\*visit)(VertexType))

//对图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次

{

int i;

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

visited[i]=0;

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

if(!visited[i])

DFS(G,i,visit);

return OK;

}

status BFSTraverse(ALGraph &G,void (\*visit)(VertexType))

//对图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次

{

int visited[MAX\_VERTEX\_NUM];

int i,j;

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

visited[i]=0;

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

{

if(!visited[i])

visit(G.vertices[i].data);

visited[i]=1;

for(j=FirstAdjVex(G,G.vertices[i].data.key);j>=0;j=NextAdjVex(G,G.vertices[i].data.key,G.vertices[j].data.key))

if(!visited[j])

{

visit(G.vertices[j].data);

visited[j]=1;

}

}

return OK;

}

status SaveGraph(ALGraph G, char FileName[])

//将图的数据写入到文件FileName中

{

FILE \*fp1;

fp1=fopen(FileName,"w+");

int i,j;

int loc1,loc2;

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

fprintf(fp1,"%d %s ",G.vertices[i].data.key,G.vertices[i].data.others);

fprintf(fp1,"-1 nil ");

int a[MAX\_VERTEX\_NUM]={0};

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

a[G.vertices[i].data.key]=1;

for(i=0;i<MAX\_VERTEX\_NUM-1;i++)

if(a[i])

for(j=i+1;j<MAX\_VERTEX\_NUM;j++)

if(a[j])

{

loc1=LocateVex(G,i);

loc2=LocateVex(G,j);

if(ArcDetect(G,loc1,loc2))

fprintf(fp1,"%d %d ",i,j);

}

fprintf(fp1,"-1 -1");

fclose(fp1);

return OK;

}

status LoadGraph(ALGraph &G, char FileName[])

//读入文件FileName的图数据，创建图的邻接表

{

FILE \*fp2;

if((fp2=fopen(FileName,"r"))==NULL)

return ERROR;

VertexType V[21];

KeyType VR[100][2];

int i=0;

do {

fscanf(fp2,"%d%s",&V[i].key,V[i].others);

} while(V[i++].key!=-1);

i=0;

do {

fscanf(fp2,"%d%d",&VR[i][0],&VR[i][1]);

} while(VR[i++][0]!=-1);

if (CreateGraph(G,V,VR)==ERROR)

{

printf("输入数据错，无法创建");

return ERROR;

}

fclose(fp2);

return OK;

}

status CreateGraph1(ALGraph &G)

{

VertexType V[21];

KeyType VR[100][2];

int i=0;

do {

scanf("%d%s",&V[i].key,V[i].others);

} while(V[i++].key!=-1);

i=0;

do {

scanf("%d%d",&VR[i][0],&VR[i][1]);

} while(VR[i++][0]!=-1);

if (CreateGraph(G,V,VR)==ERROR)

{

printf("输入数据错，无法创建");

return ERROR;

}

return OK;

}

status GraghsAdd(GraghsLink &Graghs,char ListName[])

{

GraghsLink p =Graghs;

int i=0;

while (p->next)

p=p->next;

GraghsLink temp;

temp=(GraghsLink)malloc(sizeof(GraghNode));

do {

temp->name[i]=ListName[i];

} while (ListName[i++]);

temp->next=NULL;

printf("输入图数据集。\n");

printf("推荐测试集：5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1\n");

if(!CreateGraph1(temp->G))

return ERROR;

p->next=temp;

return OK;

}

status GraghsRemove(GraghsLink &Graghs,char ListName[])

{

if(Graghs==NULL)

return INFEASIBLE;

GraghsLink p=Graghs->next,q=Graghs;

int j,flag; //循环变量

char ch;

while(p)

{

j=0;

flag=1;

while(ch=ListName[j]) //判断线性表名是否符合要求

{

if(p->name[j]!=ch)

flag=0;

j++;

}

if(flag==1)

{

DestroyGraph(p->G);

q->next=p->next;

free(p);

return OK;

}

else

{

q=p;

p=p->next;

}

}

if(p==NULL)

return ERROR;

}

GraghsLink GraghsLocate(GraghsLink Graghs,char ListName[])

{

if(Graghs==NULL)

return NULL;

GraghsLink p=Graghs->next;

int j,flag;

char ch;

while(p)

{

j=0;

flag=1;

while(ch=ListName[j])

{

if(p->name[j]!=ch)

flag=0;

j++;

}

if(flag==1)

return p;

p=p->next;

}

if(p==NULL)

return NULL;

}

int GraghsLength(GraghsLink Graghs)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

if(Graghs==NULL)

return INFEASIBLE;

GraghsLink p=Graghs;

int i=0; //循环计数变量

while(p)

{

p=p->next;

i++;

}

return i-1;

}

status GraghsTraverse(GraghsLink Graghs)

// 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。

{

if(Graghs==NULL)

return INFEASIBLE;

GraghsLink p=Graghs->next;

if(p==NULL)

return OK;

while(p)

{

printf("%s ",p->name);

p=p->next;

}

return OK;

}

“func.h”

#include <stdio.h>

#include <stdlib.h>

#include"def.h"

#include "string.h"

int hasDuplicate(ALGraph G,KeyType u);

void visit(VertexType v);

void ShowGragh(ALGraph G);

status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2]);

status DestroyGraph(ALGraph &G);

int LocateVex(ALGraph G,KeyType u);

status PutVex(ALGraph &G,KeyType u,VertexType value);

int FirstAdjVex(ALGraph G,KeyType u);

int NextAdjVex(ALGraph G,KeyType v,KeyType w);

status InsertVex(ALGraph &G,VertexType v);

int ArcDetect(ALGraph G,KeyType v,KeyType w);

int DeleteArcNode(ALGraph &G,KeyType v,KeyType w);

status DeleteArc(ALGraph &G,KeyType v,KeyType w);

status DeleteVex(ALGraph &G,KeyType v);

status InsertArc(ALGraph &G,KeyType v,KeyType w);

status DeleteArc(ALGraph &G,KeyType v,KeyType w);

void DFS(ALGraph G,int i,void (\*visit)(VertexType));

status DFSTraverse(ALGraph &G,void (\*visit)(VertexType));

status BFSTraverse(ALGraph &G,void (\*visit)(VertexType));

status SaveGraph(ALGraph G, char FileName[]);

status LoadGraph(ALGraph &G, char FileName[]);

int GraghsLength(GraghsLink Graghs);

status GraghsAdd(GraghsLink &Graghs,char ListName[]);

status GraghsTraverse(GraghsLink Graghs);

status GraghsRemove(GraghsLink &Graghs,char ListName[]);

GraghsLink GraghsLocate(GraghsLink Graghs,char ListName[]);

“def.h”

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 20

#define LISTS\_INIT\_SIZE 20

typedef int status;

typedef int KeyType;

typedef enum {DG,DN,UDG,UDN} GraphKind;

typedef struct {

KeyType key;

char others[20];

} VertexType; //顶点类型定义

typedef struct ArcNode { //表结点类型定义

int adjvex; //顶点位置编号

struct ArcNode \*nextarc; //下一个表结点指针

} ArcNode;

typedef struct VNode{ //头结点及其数组类型定义

VertexType data; //顶点信息

ArcNode \*firstarc; //指向第一条弧

} VNode,AdjList[MAX\_VERTEX\_NUM];

typedef struct { //邻接表的类型定义

AdjList vertices; //头结点数组

int vexnum,arcnum; //顶点数、弧数

GraphKind kind; //图的类型

} ALGraph;

typedef struct GraghNode{

char name[30];

ALGraph G;

struct GraghNode \*next;

}GraghNode,\*GraghsLink;