

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术201806**

**学 号： U201813676**

**姓 名： 刘 汉 鹏**

**指导教师： 许贵平**

**报告日期： 2020年 11月 22 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](#_Toc56931138)

[1.1 问题描述 2](#_Toc56931139)

[1.1.1 线性表的基本运算定义 2](#_Toc56931140)

[1.2 系统设计 3](#_Toc56931141)

[1.3 系统实现 7](#_Toc56931142)

[**1.4 系统测试** 7](#_Toc56931143)

[1.5 实验小结 8](#_Toc56931144)

[2 基于链式存储结构的线性表实现 9](#_Toc56931145)

[2.1 问题描述 9](#_Toc56931146)

[2.1.1 线性表的基本运算定义 9](#_Toc56931147)

[2.2 系统设计 10](#_Toc56931148)

[2.3 系统实现 14](#_Toc56931149)

[**2.4 系统测试** 14](#_Toc56931150)

[2.5 实验小结 15](#_Toc56931151)

[3.1 问题描述 16](#_Toc56931152)

[3.2 系统设计 16](#_Toc56931153)

[3.3 系统实现 18](#_Toc56931154)

[3.4 系统测试 18](#_Toc56931155)

[3.5 实验小结 19](#_Toc56931156)

[4 基于邻接表的图实现 20](#_Toc56931157)

[4.1 问题描述 20](#_Toc56931158)

[4.2 系统设计 20](#_Toc56931159)

[4.3 系统实现 21](#_Toc56931160)

[4.4 系统测试 22](#_Toc56931161)

[4.5 实验小结 22](#_Toc56931162)

[参考文献 23](#_Toc56931163)

[附录A 基于顺序存储结构线性表实现的源程序 24](#_Toc56931164)

[附录B 基于链式存储结构线性表实现的源程序 40](#_Toc56931165)

[附录C 基于二叉链表二叉树实现的源程序 53](#_Toc56931166)

[附录D 基于邻接表图实现的源程序 73](#_Toc56931167)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

采用顺序表作为线性表的物理结构，实现线性表的基本运算，具体运算描述见**1.1.1**。其中ElemType为数据元素的类型名，具体含义为int。

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

## 1.1.1 线性表的基本运算定义

依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的初始化表、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,cur\_e,pre\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre\_e无定义。

⑼获得后继：函数名称是NextElem(L,cur\_e,next\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败，next\_e无定义。

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

(13)线性表读写文件:函数名称是SaveList(L,FileName[]),初始条件是线性表L已经存在，FileName已知；操作结果是将L中的元素依次写到FileName文件中。

(14)增加一个新线性表:函数名称是AddList(Lists,ListName[]),初始条件是多线性表Lists已经存在；操作结果是Lists新增一个线性表，名字是ListName.

(15)移出一个线性表:函数名称是RemoveList(Lists,ListName[]),初始条件是多线性表Lists已经存在，ListName已知；操作结果是将ListName从多线性表中删除。

(16)查找线性表:函数名称是Locate(Lists,ListName[]),初始化条件是多线性表Lists已经存在，ListName已知；操作结果是返回ListName的位置。

## 1.2 系统设计

（1）ListLength(L)：图1-1

输入：线性表的表名L

输出：线性表的长度length

算法处理步骤：1）输入表名

2）(a)判断表是否存在，若不存在，返回INFEASIBLE；

（b）若表存在，对表进行遍历，当表的元素存在时，length++；

3）返回表长

算法复杂度：时间复杂度：O（n）

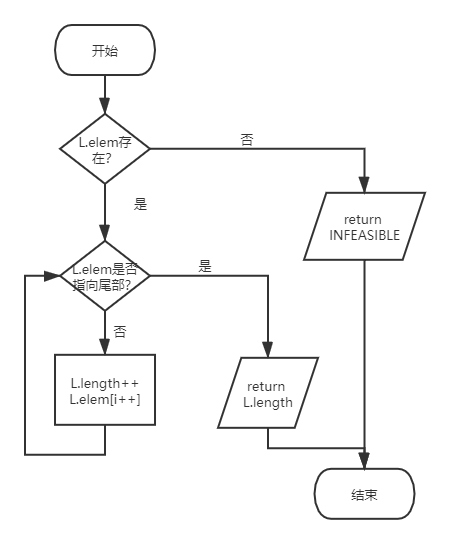
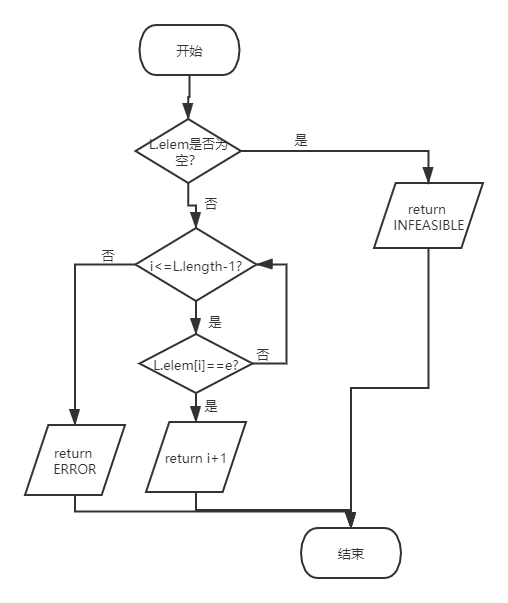
 

图1-1 ListLength(L) 图1-2 LocateElem(L,e)

（2）LocateElem(L,e)：图1-2

输入：线性表的表名L，想要确定位置的元素e

输出：元素e的位置

算法处理步骤：1)输入表名L，元素e

2）a)线性表L是否存在，若不存在，则返回INFEASIBLE；

b)若存在，从表头开始，对表进行遍历，判断L.elem[i]与元素e是否相等，若相等，则返回元素e的位置i+1；若不等，则将i+1,直到i指向表尾。

3）返回元素e的位置i+1.

算法复杂度：时间复杂度O（n）

（3）ListInsert(L,i,e):图1-3

输入：线性表的表名L，插入的位置i，插入的元素e

输出：插入元素后的表L

算法处理步骤：1）输入线性表的表名L，插入的位置i，插入的元素e；

2）判断线性表L是否不存在，若不存在，返回INFEASIBLE；

3）判断插入的位置i是否在1～length+1之间，若不是，则返回ERROR；

4）判断表L是否为满表，若是，则将表容一位；

5）从表尾开始将每个元素向后移动一位，直到第i个元素，将第i个元素的值赋为e，表长加1，返回OK。

算法复杂度：时间复杂度：O（n）

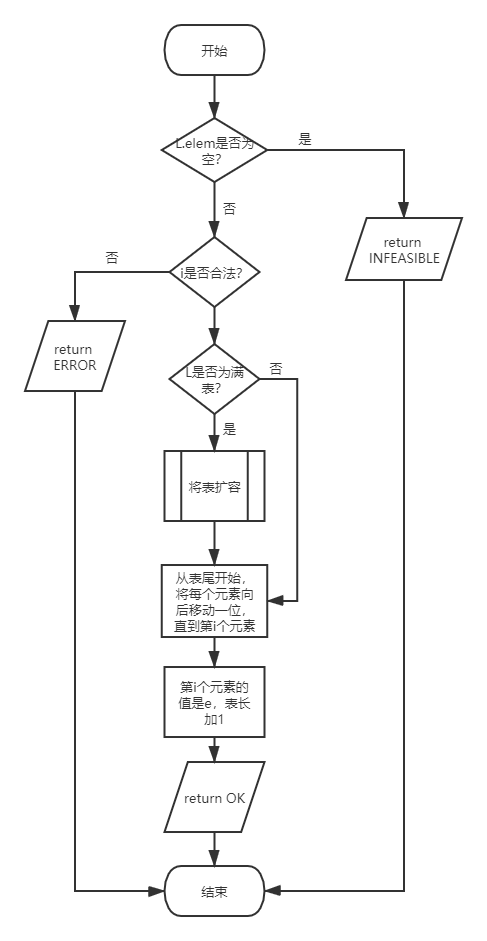
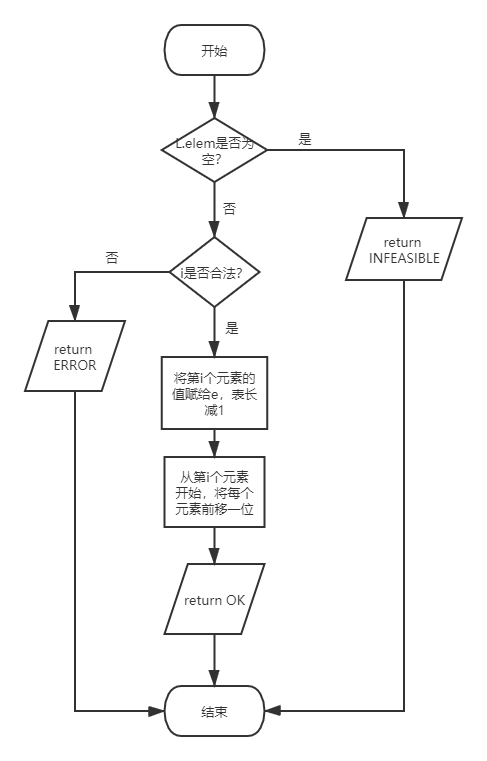
 

图1-3 ListInsert(L,i,e) 图1-4 ListDelete（L,i,e）

（4）ListDelete（L,i,e）:图1-4

输入：线性表的表名L，删除的位置i

输出：删除的元素e

算法处理步骤：1）输入线性表的表名L，删除的位置i

2）判断线性表L是否不存在，若不存在，返回INFEASIBLE；

3)判断删除的位置i是否在1～length之间，若不是，则返回ERROR；

4）将第i个元素的值赋给e，表长减1，从第i+1个元素开始，将每个元素向前移动一位，返回OK。

算法复杂：时间复杂度：O(n)

(5)AddList(Lists,ListName[]):图1-5

输入：要增加的表名ListName

输出：增加一个线性表后的总表

算法处理步骤：1)读取表的表名ListName

2)初始化一个表名为ListName的线性表，总表的长度加1

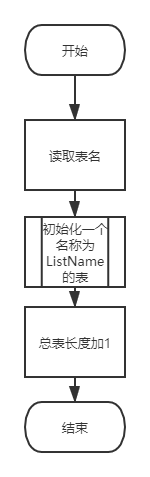
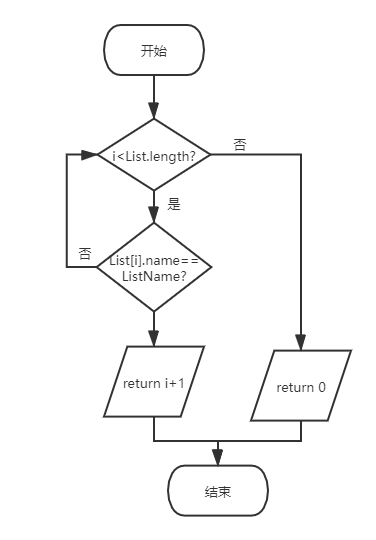
 

图1-5 AddList(Lists,ListName[]) 图1-6 LocateList(Lists,ListName[])

(6)LocateList(Lists,ListName[]):图1-6

输入：要确定位置的表名ListName

输出：线性表在总表的位置i

算法处理步骤：从表头开始对总表进行遍历，判断线性表的名称是否为ListName,若是则返回i+1,否则，返回 0

将封装好的子函数写入头文件中，然后利用swtich{case n:… break;}的结构，将程序进行可视化操作。（如图1-7）

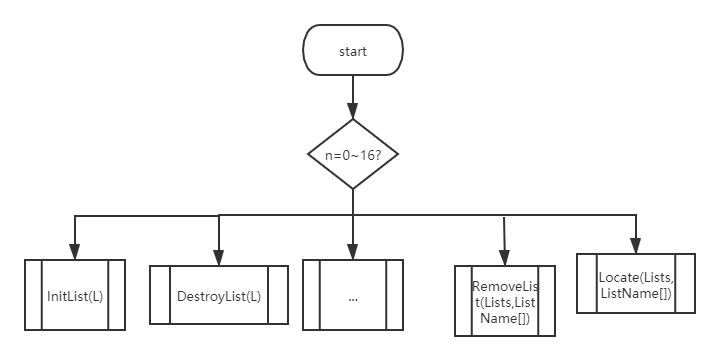


图1-7线性表操作演示流程示意图

## 1.3 系统实现

程序实现采用的集成环境是Visual Studio 2019,使用了C语言标准头文件stdio.h，主要采用了顺序编程结构。

**1.4 系统测试**

系统测试挑选了几个主要的程序进行了测试，测试结果如表一。经过在线下对所有函数的测试，所有函数对正确的输入有正确的输出，对错误的输入也有相应的报错提醒。

表1-1线性表测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 程序名称 | 程序输入 | 理论结果 | 运行结果 |
| ListInsert | 7,1 | 表头的元素是7 |  |
| ListInsert | 6,6 | 在表尾插入了6 |  |
| ListInsert | 9,3 | 第三个元素是9 |  |
| ListDetle | 3 | 返回的元素是9 |  |
| AddList | 湖北 | 多线性表存在一个名为湖北的线性表 |  |
| LocateList | 湖北 | 返回的位置为1 |  |
| RemoveList | 湖北 | 线性表存在，删除成功 |  |

## 1.5 实验小结

这次实验主要学习了如何对线性表进行一些基本的操作，如：初始化、插入、删除和查询等；以及进一步学会了利用多线性表对线性表进行集中管理，对线性表这种结构体有了更深的理解和认识；此次试验中没有单独编写头文件，希望能够吸取教训在接下来实验中有所改进。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

采用单链表作为线性表的物理结构，实现线性表的基本运算，具体运算描述见**2.1.1**。其中ElemType为数据元素的类型名，具体含义为int。

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

## 2.1.1 线性表的基本运算定义

依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的初始化表、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,cur\_e,pre\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre\_e无定义。

⑼获得后继：函数名称是NextElem(L,cur\_e,next\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败，next\_e无定义。

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

(13)线性表读写文件:函数名称是SaveList(L,FileName[]),初始条件是线性表L已经存在，FileName已知；操作结果是将L中的元素依次写到FileName文件中。

(14)增加一个新线性表:函数名称是AddList(Lists,ListName[]),初始条件是多线性表Lists已经存在；操作结果是Lists新增一个线性表，名字是ListName.

(15)移出一个线性表:函数名称是RemoveList(Lists,ListName[]),初始条件是多线性表Lists已经存在，ListName已知；操作结果是将ListName从多线性表中删除。

(16)查找线性表:函数名称是worklist(Lists,ListName[]),初始化条件是多线性表Lists已经存在，ListName已知；操作结果是返回ListName的单链表的指针。

## 2.2 系统设计

（1）ListLength(L)：图2-1

输入：线性表的表名L

输出：线性表的长度length

算法处理步骤：1）输入表名

2）(a)判断表是否存在，若不存在，返回INFEASIBLE；

（b）若表存在，对表进行遍历，当单链表的指针存在时，length++；

3）返回表长

算法复杂度：时间复杂度：O（n）

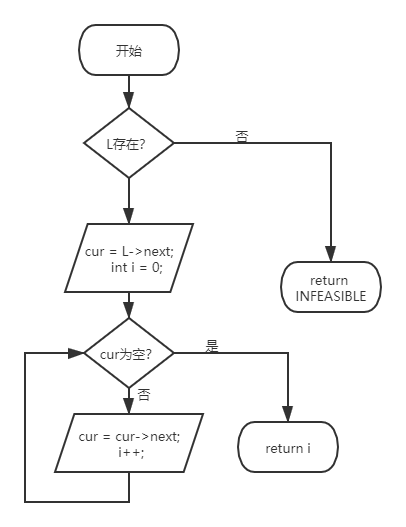
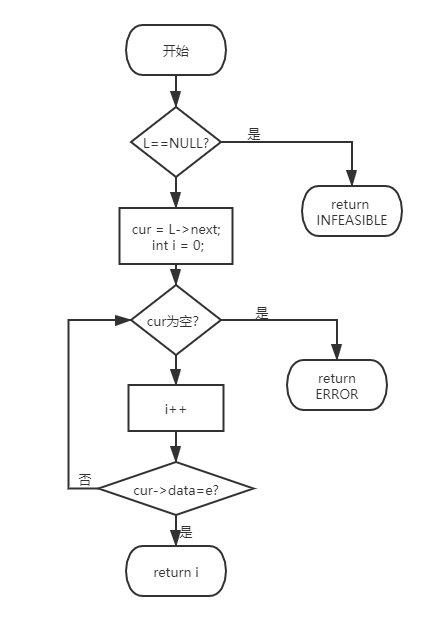
 

图2-1 ListLength(L) 图2-2 LocateElem(L,e)

（2）LocateElem(L,e)：图2-2

输入：线性表的表名L，想要确定位置的元素e

输出：元素e的位置

算法处理步骤：1)输入表名L，元素e

2）a)线性表L是否存在，若不存在，则返回INFEASIBLE；

b)若存在，从表头开始，对表进行遍历，判断cur->data与元素e是否相等，若相等，则返回元素e的位置i；若不等，则将链表指针后移,直到指针为空。

3）返回元素e的位置i+1.

算法复杂度：时间复杂度O（n）

（3）ListInsert(L,i,e):图2-3

输入：线性表的表名L，插入的位置i，插入的元素e

输出：插入元素后的表L

算法处理步骤：1）输入线性表的表名L，插入的位置i，插入的元素e；

2）判断线性表L是否不存在，若不存在，返回INFEASIBLE；

3）判断插入的位置i是否大于1，若不是，则返回ERROR；

4）判断是否为空表且在头结点插入，若是直接在表头后添加一个新的结点且结点的元素为e，后驱指针为空；

5）对链表进行遍历，查找第i个元素，当不是表尾插入时，前驱的后继为新节点，新节点的后继为当前结点，新节点的元素为e；当是表尾插入时，尾结点的后继为新节点，新节点的后继为NULL,元素为e,否则返回ERROR。

算法复杂度：时间复杂度：O（n）

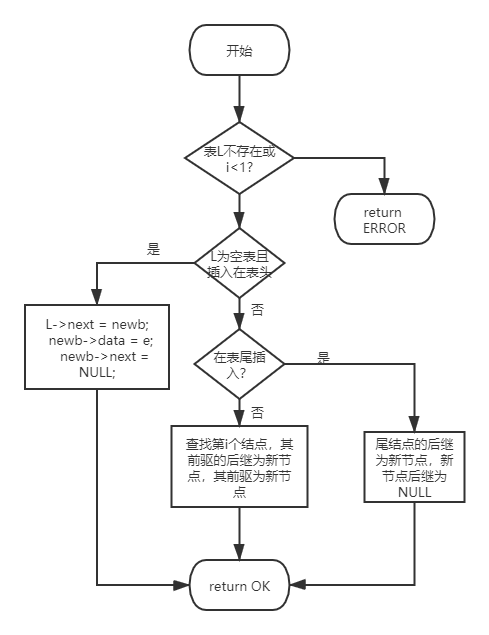
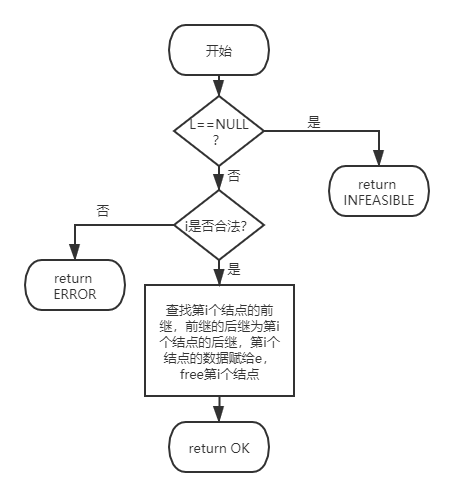
 

图2-3 ListInsert(L,i,e) 图2-4 ListDelete（L,i,e）

（4）ListDelete（L,i,e）:图2-4

输入：线性表的表名L，删除的位置i

输出：删除的元素e

算法处理步骤：1）输入线性表的表名L，删除的位置i

2）判断线性表L是否不存在，若不存在，返回INFEASIBLE；

3)判断删除的位置i是否在1～length之间，若不是，则返回ERROR；

4）对单链表进行遍历，第i个结点的前驱的后继为第i个结点的后继，将第i个结点的数据赋给e，表长减1，将第i个结点free，返回OK。

算法复杂：时间复杂度：O(n)

(5)AddList(Lists,ListName[]):图22-5

输入：要增加的表名ListName

输出：增加一个线性表后的总表

算法处理步骤：1)读取表的表名ListName

2)初始化一个表名为ListName的线性表，在总表的表尾添加一个单链表，总表的长度加1

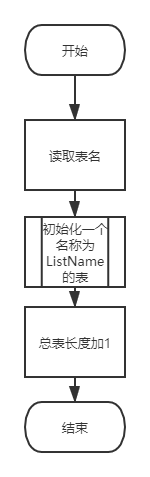
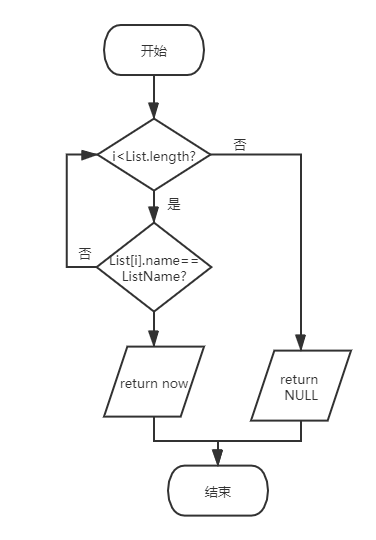
 

图2-5 AddList(Lists,ListName[]) 图2-6 worklist(Lists,ListName[])

(6)worklist(Lists,ListName[]):图2-6

输入：要确定位置的表名ListName

输出：线性表的表头指针

算法处理步骤：从表头开始对总表进行遍历，判断线性表的名称是否为ListName,若是则返回单链表的表头指针,否则，返回 0

将封装好的子函数写入头文件中，然后利用swtich{case n:… break;}的结构，将程序进行可视化操作。（如图1-7）

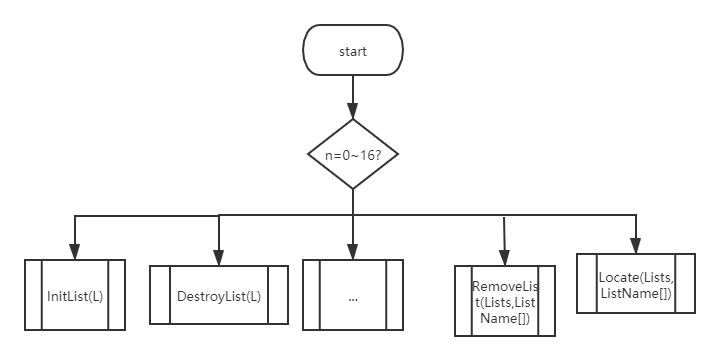


图2-7线性表操作演示流程示意图

## 2.3 系统实现

程序实现采用的集成环境是Visual Studio 2019,使用了C语言标准头文件stdio.h，stdlib.h和自己编写的头文件linklist.h，主要采用了顺序编程结构。

**2.4 系统测试**

系统测试挑选了几个主要的程序进行了测试，测试结果如表1-1。经过在线下对所有函数的测试，所有函数对正确的输入有正确的输出，对错误的输入也有相应的报错提醒。

表1-1线性表测试及结果表

|  |  |  |  |
| --- | --- | --- | --- |
| 程序名称 | 程序输入 | 理论结果 | 运行结果 |
| ListInsert | 7,1 | 表头的元素是7 |  |
| ListInsert | 6,6 | 在表尾插入了6 |  |
| ListInsert | 9,3 | 第三个元素是9 |  |
|  |  |  |  |
| AddList | a | 多线性表存在一个名为a的线性表 |  |
| worklist | a | 对线性表a进行操作 |  |
| RemoveList | a | 线性表存在，删除成功 |  |

## 2.5 实验小结

这次实验主要学习了如何对单链表进行一些基本的操作，如：初始化、插入、删除和查询等；以及进一步学会了利用多单链表对线性表进行集中管理，对以单链表为物理结构的线性表这种结构体有了更深的理解和认识。

改进:这次实验完善了自己的注释习惯，同时将完成的函数单独写在一个头文件中。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。

## 3.2 系统设计

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

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

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

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

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

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

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

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

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

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

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

特殊情况，c插入作为根结点？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。

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

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

**注：**前序、中序和后序三种遍历算法，要求至少一个用非递归算法实现。

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

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

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

## 3.3 系统实现

（1）前序遍历：函数名称是PreOrderTraverse(T,Visit())：采用递归形式。

1.若二叉树为空，则遍历结束。

2.否则，执行下列步骤：

（1）访问根结点；

（2）先序遍历根的左子树；

（3）先序遍历根的右子树。

时间复杂度：O（n）

空间复杂度：O（n），需要递归工作栈

（2）中序遍历：函数名称是InOrderTraverse(T,Visit)):非递归算法

1.设置栈S存放所经过的根结点指针信息；初始化S

2.遇到根结点并不访问，而是入栈

3.中序遍历它的左子树；左子树遍历结束后，将根结点指针退栈，并访问根结点，然后中序遍历它的右子树。

4．当需要退栈时，如果栈为空则结束。

时间复杂度：O（n）

（3）后序遍历：函数名称是PostOrderTraverse(T,Visit))：递归算法

1.若二叉树为空，则遍历结束。

2.否则，执行下列步骤：

（1）后序遍历根的左子树；

（2）后序遍历根的右子树；

（3）访问根结点。

时间复杂度：O（n）

空间复杂度：O（n），需要递归工作栈

（4）按层遍历：函数名称是LevelOrderTraverse(T,Visit))：

1.初始化队列Q；

2.当根结点非空时，根结点入队

3.队列Q非空，则队首结点出队，并访问，队首结点的左孩子非空则入队，队首结点的右孩子非空则入队。直到队列Q为空。

时间复杂度：O（n）

空间复杂度：O（n）

## 3.4 系统测试

系统测试挑选了几个主要的程序进行了测试，测试结果如表3-1。经过在线下对所有函数的测试，所有函数对正确的输入有正确的输出，对错误的输入也有相应的报错提醒。

|  |  |  |  |
| --- | --- | --- | --- |
| 程序名称 | 程序输入 | 理论结果 | 运行结果 |
| PreOrderTraverse(T,Visit()) | 1 a 2 b 0 null 0 null 3 c 4 d 0 null 0 null 5 e 0 null 0 null -1 null | 1,a 2,b 3,c 4,d 5,e |  |
| InOrderTraverse(T,Visit)) | 1 a 2 b 0 null 0 null 3 c 4 d 0 null 0 null 5 e 0 null 0 null -1 null | 2,b 1,a 4,d 3,c 5,e |  |
| PostOrderTraverse(T,Visit)) | 1 a 2 b 0 null 0 null 3 c 4 d 0 null 0 null 5 e 0 null 0 null -1 null | 2,b 4,d 5,e 3,c 1,a |  |
| LevelOrderTraverse(T,Visit)) | 1 a 2 b 0 null 0 null 3 c 4 d 0 null 0 null 5 e 0 null 0 null -1 null | 1,a 2,b 3,c 4,d 5,e |  |

表 3-1

## 3.5 实验小结

这次实验主要学习了如何对二叉树进行一些基本的操作，如：初始化、插入、删除和遍历等；其中学会了使用递归算法和非递归算法进行遍历。

# 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。

## 4.2 系统设计

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CreateCraph(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。

⑵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。

⑷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。

⑸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。

⑹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。

⑺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）

⑻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。

⑼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>。

⑽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>。

⑾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。

⑿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。

## 4.3 系统实现

（1）深度优先搜索遍历：函数名称是DFSTraverse(G,visit())；

1.访问顶点v；

2.依次从v的未被访问的领接点w出发，对图进行深度优先遍历；直到图中和v有路径想通的顶点都被访问；

3．若此时图中尚有顶点未被访问，则从一个未被访问的顶点出发，重新进行深度优先遍历，直到图中所有顶点均被访问过为止。

时间复杂度：O（n+e）

空间复杂度：O（n）

（2）广深度优先搜索遍历：函数名称是BFSTraverse(G,visit())

1.访问顶点v；

2.访问v的所有未被访问的邻接点w1,w2,w3,…wk;

3.依次从这些领接点出发，访问他们的所有未被访问的领接点；依次类推，直到由v可以到达的所有顶点都被访问过为止；

4.若此时图中还有顶点未被访问，则另选一个未被访问过的顶点作起始点，重复上述过程直到图中所有顶点均被访问到。

时间复杂度：O（e）

空间复杂度：O（n）

## 4.4 系统测试

系统测试挑选了几个主要的程序进行了测试，测试结果如表4-1。经过在线下对所有函数的测试，所有函数对正确的输入有正确的输出，对错误的输入也有相应的报错提醒。

|  |  |  |  |
| --- | --- | --- | --- |
| 程序名称 | 程序输入 | 理论结果 | 运行结果 |
| DFSTraverse(G,visit()) | 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 | 5 线性表 7 二叉树 8 集合 6 无向图 |  |
| BFSTraverse(G,visit()) | 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 | 5 线性表 7 二叉树 6 无向图 8 集合 |  |

表 4-1

## 4.5 实验小结

这次实验主要学习了如何对基于邻接表的图进行一些基本的操作，如：初始化、插入、删除和遍历等；其中学会了使用DFS和BFS对图进行遍历。

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

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

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

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

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASTABLE -1

#define OVERFLOW -2

#define INFEASIBLE -1

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 { //线性表的管理表定义

struct {

char name[30];

SqList L;

} elem[10];

int length;

int listsize;

}LISTS;

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

status InitList(SqList& L);

status DestroyList(SqList& L);

status ClearList(SqList&L);

status ListEmpty(SqList L);

int ListLength(SqList L);

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

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

status PriorElem(SqList L, ElemType 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 ListTraverse(SqList L); //简化过

status InitLists(LISTS& Lists);

status workLists(LISTS Lists, char ListName[], SqList& L);

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

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

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

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

int main(void) {

LISTS Lists;

int e;

int n;

int i;

SqList L; int op = 1;

while (op) {

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

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

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

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

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

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

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

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

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

printf(" 13. InitLists 14. ADDLists\n");

printf(" 15. RemoveList 16. LocateList\n");

printf(" 17. workLists\n");

printf(" 0. Exit\n");

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

printf("请选择你的操作[0~17]:");

scanf("%d", &op);

switch (op) {

case 1:

//printf("\n----IntiList功能待实现！\n");

if (InitList(L) == OK) printf("线性表创建成功！\n");

else printf("线性表创建失败！\n");

getchar(); getchar();

break;

case 2:

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

if (DestroyList(L) == OK) printf("线性表销毁成功\n");

else printf("线性表销毁失败\n");

getchar(); getchar();

break;

case 3:

//printf("\n----ClearList功能待实现！\n");

if (ClearList(L) == OK) printf("clear ok\n");

else printf("clear error\n");

getchar(); getchar();

break;

case 4:

//printf("\n----ListEmpty功能待实现！\n");

if (ListEmpty(L) == OK) printf("ListEmpty OK\n");

else printf("ListEmpty error\n");

getchar(); getchar();

break;

case 5:

//printf("\n----ListLength功能待实现！\n");

{

if (( ListLength(L)) != FALSE) printf("ListLength=%d\n",L.length);

else printf("ListLength error\n");

getchar(); getchar();

break;

}

case 6:

{

//printf("\n----GetElem功能待实现！\n");

printf("请输入元素位置\n");

scanf("%d", &n);

if (GetElem(L, n, e) == OK) printf("第%d号元素是:%d\n", n, e);

else printf("GetElem error\n");

getchar(); getchar();

break;

}

case 7:

{

//printf("\n----LocateElem功能待实现！\n");

printf("输入你需要查找的元素\n");

scanf("%d", &e);

if ((i = LocateElem(L, e) )!= FALSE) printf("元素%d的位置为%d\n", e, i);

else printf("Locate error\n");

getchar(); getchar();

break;

}

case 8:

{

//printf("\n----PriorElem功能待实现！\n");

printf("输入你需要确定前驱的元素\n");

scanf("%d", &e);

int pre;

if (PriorElem(L, e, pre) != FALSE) printf("%d 的前驱是：%d\n", e, pre);

else printf("PriorElem error\n");

getchar(); getchar();

break;

}

case 9:

{

//printf("\n----NextElem功能待实现！\n");

printf("输入你要确定后继的元素\n");

scanf("%d", &e);

int next;

if (NextElem(L, e, next) != FALSE) printf("%d 的后继是：%d\n", e, next);

else printf("NextElem error\n");

getchar(); getchar();

break;

}

case 10:

{

//printf("\n----ListInsert功能待实现！\n");

printf("输入你要插入的元素\n");

scanf("%d", &e);

printf("输入你要插入的位置\n");

scanf("%d", &n);

if (ListInsert(L, n, e) != FALSE) printf("插入成功\n");

else printf("插入失败\n");

getchar(); getchar();

break;

}

case 11:

{

//printf("\n----ListDelete功能待实现！\n");

printf("输入你要删除的位置\n");

scanf("%d", &i);

if (ListDelete(L, i, e) != FALSE) printf("删除第%d号位置的元素是：%d\n", i, e);

else printf("删除失败\n");

getchar(); getchar();

break;

}

case 12:

{

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

if (!ListTraverse(L)) printf("线性表是空表！\n");

getchar(); getchar();

break;

}

case 13:

{

if (InitLists(Lists) == OK) printf("Lists ok \n");

else printf("Lists error\n");

getchar(); getchar();

break;

}

case 14:

{

printf("输入你要插入的表的名字：");

char name[10] = { 0 };

scanf("%s", name);

if (AddList(Lists, name) != FALSE) printf("插入成功\n");

else printf("插入失败\n");

getchar(); getchar();

break;

}

case 15:

{

printf("输入你要删除的表的名字:");

char dname[10] = { 0 };

scanf("%s", dname);

if (RemoveList(Lists, dname) != FALSE) printf("delet ok \n");

else printf("delet erro\n");

getchar(); getchar();

break;

}

case 16:

{

printf("输入你要确定位置的表名:");

char qname[10] = { 0 };

scanf("%s", qname);

int q;

if ((q = LocateList(Lists, qname)) != FALSE) printf("位置是%d\n", q);

else printf("locate erro\n");

getchar(); getchar();

break;

}

case 17:

{

printf("输入你要操作的表的名字:");

char wname[10] = { 0 };

scanf("%s", wname);

if (workLists(Lists, wname, L) != FALSE) printf("work start\n");

else printf("work error\n");

getchar(); getchar();

break;

}

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

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

status InitList(SqList& L) {

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 ListTraverse(SqList L) {

int i;

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

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

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

return L.length;

}

\*/

status 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 INFEASIBLE;

/\*\*\*\*\*\*\*\*\*\* 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INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

else

{

for (int i = 0; i <= L.length - 1; i++)

{

L.elem[i] = 0;

}

L.length = 0;

return OK;

}

/\*\*\*\*\*\*\*\*\*\* 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INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

else if (L.length == 0)

{

return TRUE;

}

else return FALSE;

/\*\*\*\*\*\*\*\*\*\* 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INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

else

{

int i = 0;

while (L.elem[i] != 0)

{

i++;

L.length++;

}

return L.length;

}

/\*\*\*\*\*\*\*\*\*\* 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INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

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

{

return ERROR;

}

else

{

e = L.elem[i - 1];

return OK;

}

/\*\*\*\*\*\*\*\*\*\* 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INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

for (int i = 0; i <= L.length - 1; i++)

{

if (e == L.elem[i])

return i + 1;

}

return ERROR;

/\*\*\*\*\*\*\*\*\*\* 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INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

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

{

if (L.elem[i] == e)

{

if (i == 0)

return ERROR;

else

{

pre = L.elem[i - 1];

return OK;

}

}

}

return ERROR;

/\*\*\*\*\*\*\*\*\*\* 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INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL)

{

return INFEASIBLE;

}

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

{

if (L.elem[i] == e)

{

if (i == L.length - 1)

return ERROR;

else

{

next = L.elem[i + 1];

return OK;

}

}

}

return ERROR;

/\*\*\*\*\*\*\*\*\*\* 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INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL) return INFEASIBLE;

int j;

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

return ERROR;

if (L.length >= L.listsize)

{

int\* newbase;

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

if (newbase == NULL) return ERROR;

L.elem = newbase;

L.listsize += LISTINCREMENT;

}

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

{

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

}

L.elem[i - 1] = e;

L.length++;

return OK;

/\*\*\*\*\*\*\*\*\*\* 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INFEASIBLE。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL) return INFEASIBLE;

if ((i < 1) || (i > L.length)) return ERROR;

int\* p = &(L.elem[i - 1]);

e = \*p;

int\* q = L.elem + L.length - 1;

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

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

--L.length;

return OK;

/\*\*\*\*\*\*\*\*\*\* 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 == NULL) return INFEASIBLE;

if (L.length != 0)

{

for (int i = 0; i <= L.length - 2; i++)

{

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

}

printf("%d", L.elem[L.length - 1]);

}

return OK;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem == NULL) return INFEASIBLE;

FILE\* pp;

pp = fopen(FileName, "wb");

if (pp == NULL) return ERROR;

for (int i = 0; i <= L.length - 1; i++)

{

fwrite(&L.elem[i], sizeof(int), 1, pp);

}

fclose(pp);

return OK;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L.elem != NULL) return INFEASIBLE;

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

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

FILE\* fr;

fr = fopen(FileName, "rb");

if (fr == NULL) return ERROR;

while (fread(&L.elem[L.length], sizeof(int), 1, fr))

{

L.length++;

}

fclose(fr);

return OK;

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

}

status InitLists(LISTS& Lists)

{

Lists.length = 0;

Lists.listsize = LISTINCREMENT;

return OK;

}

status workLists(LISTS Lists,char ListName[],SqList&L)

{

for (int i = 0; i <= Lists.length - 1; i++)

{

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

{

L = Lists.elem[i].L;

L.length = Lists.elem[i].L.length;

L.listsize = Lists.elem[i].L.listsize;

L.elem = Lists.elem[i].L.elem;

return OK;

}

}

return FALSE;

}

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。

{

// 请在这里补充代码，完成本关任务

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

strcpy(Lists.elem[Lists.length].name, ListName);

int\* newbase;

newbase = (int\*)malloc(LIST\_INIT\_SIZE \* sizeof(int));

Lists.elem[Lists.length].L.elem = newbase;

Lists.elem[Lists.length].L.length = 0;

Lists.elem[Lists.length].L.listsize = LIST\_INIT\_SIZE;

Lists.length++;

return OK;

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

}

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

// Lists中删除一个名称为ListName的线性表

{

// 请在这里补充代码，完成本关任务

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

int i = 0;

bool coms;

for (i = 0; i <= Lists.length - 1; i++)

{

for (int j = 0; j <= 4; j++)

{

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

coms = TRUE;

else

{

coms = FALSE;

break;

}

}

if (coms == TRUE)

break;

}

if (coms == FALSE)

return ERROR;

else

{

for (int k = i; k <= Lists.length - 2; k++)

{

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

}

Lists.length--;

return OK;

}

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

}

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

{

// 请在这里补充代码，完成本关任务

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

int i = 0;

bool coms=FALSE;

for (i = 0; i <= Lists.length - 1; i++)

{

for (int j = 0; j <= 4; j++)

{

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

coms = TRUE;

else

{

coms = FALSE;

break;

}

}

if (coms == TRUE)

break;

}

if (coms == FALSE)

return 0;

else

return i + 1;

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

}

# 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 \*/

“linklist.h”

#pragma once

#include "stdio.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

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 LISTS{ //单链表的管理表定义

char name;

LinkList cur;

struct LISTS\* next;

}LISTS,\*LinkLists;

status InitLists(LinkLists& L)

{

//多表不存在啊，构建一个空的多表

if (L == NULL)

{

LinkLists now = new(LISTS);

L = now;

L->name = '\*';

L->next = NULL;

return OK;

}

else return FALSE;

}

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。

{

// 请在这里补充代码，完成本关任务

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

if(L==NULL)

{

LinkList newbase=new(LNode);

L=newbase;

L->next = NULL;

return OK;

}

else return INFEASIBLE;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;//线性表为空，无法发销毁

LinkList cur = L->next;//需要销毁的链表

while (cur)

{

LinkList p = cur;

cur = cur->next;

delete p;//销毁当前节点

}

L = NULL;//销毁表头

return OK;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;

LinkList p;

LinkList cur = L->next;//需要清空的链表

while (cur)

{

p = cur;

cur = cur->next;

delete p;//清空当前节点

p = NULL;

}

L->next = NULL;

return OK;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;//线性表L不存在

LinkList cur = L->next;

if (cur == NULL) return TRUE;//线性表为空

return FALSE;//线性表不为空

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;//线性表不存在

LinkList cur = L->next;

int i = 0;

while (cur)

{

cur = cur->next;

i++;

}

return i;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;//线性表不存在

int j = 0;

LinkList cur = L->next;

while (cur)

{

j++;

if (i == j)

{

e = cur->data;

return OK;

}

cur = cur->next;

}

return ERROR;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;//线性表不存在

LinkList cur;

cur = L->next;

int i = 0;

while (cur)

{

i++;

if (cur->data == e)

{

return i;

}

cur = cur->next;

}

return ERROR;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;//线性表不存在

LinkList prior = L;//前驱指针

LinkList cur = L->next;//当前节点

while (cur)

{

if (cur->data == e)//判断e是否存在，若存在返回前驱

{

if (prior == L) return ERROR;

pre = prior->data;

return OK;

}

prior = cur;

cur = cur->next;

}

return ERROR;//查找失败

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;//线性表不存在

LinkList cur = L->next;//当前节点

if (cur == NULL) return ERROR;//线性表长度为0

LinkList nextl = cur->next;//后继节点

while (nextl)

{

if (cur->data == e)

{

if (nextl == NULL) return ERROR;//尾节点存在后继元素

next = nextl->data;

return OK;

}

cur = nextl;//当前节点后移

nextl = nextl->next;//后继指针后移

}

return ERROR;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;//线性表不存在

if (i < 1) return ERROR;//i不合法，插入失败

int curnum = 0;//当前节点的位置

LinkList prior = L;//前驱节点

LinkList cur = L->next;//当前节点

if (cur == NULL && i == 1)//线性表为空，且插在头结点

{

LinkList newb = new(LNode);

L->next = newb;

newb->data = e;

newb->next = NULL;

return OK;

}

while (cur)

{

curnum++;

if (curnum == i)

{

LinkList newbase = (LinkList)malloc(sizeof(LinkList));//添加新节点

if (newbase == NULL) return ERROR;//插入失败

prior->next = newbase;

newbase->data = e;

newbase->next = cur;

return OK;

}

prior = cur;

cur = cur->next;

}

if (i == curnum + 1)//在尾节点插入

{

LinkList newc = (LinkList)malloc(sizeof(LinkList));

if (newc == NULL) return ERROR;//插入失败

prior->next = newc;

newc->data = e;

newc->next = NULL;

return OK;

}

return ERROR;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;// 线性表不存在

int length = ListLength(L);

if (i < 1||i>length) return ERROR;//i不合法，删除失败

LinkList DeletePrior =NULL;//前驱节点

LinkList DeletCur = L;//当前节点

while (i--)

{

DeletePrior = DeletCur;

DeletCur = DeletCur->next;

}

e = DeletCur->data;

DeletePrior->next = DeletCur->next;

free(DeletCur);

return OK;

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

}

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。

{

// 请在这里补充代码，完成本关任务

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

if (L == NULL) return INFEASIBLE;//线性表不存在

LinkList cur = L->next;//当前节点

while (cur!=NULL)

{

printf("%d ", cur->data);//输出当前元素

if (cur->next == NULL) return OK;

else cur = cur->next;//指针后移

}

return OK;

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

}

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。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 1 \*\*\*\*\*\*\*\*\*/

if (L == NULL) return INFEASIBLE;//线性表不存在

FILE\* fp = fopen(FileName, "wt");//打开文件

LinkList cur = L->next;//当前节点

while (cur)

{

fprintf(fp, "%d ", cur->data);//写入当前元素

cur = cur->next;

}

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 1 \*\*\*\*\*\*\*\*\*\*/

}

status LoadList(LinkList& 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。

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 2 \*\*\*\*\*\*\*\*\*/

if (L != NULL) return INFEASIBLE;//线性表存在不能写入

L = (LinkList)malloc(sizeof(LinkList));//生成表头节点

FILE\* fp = fopen(FileName, "rt");//打开文件

LinkList cur = L;//插入的位置的前驱

while (feof(fp) == 0)

{

LinkList newbase = (LinkList)malloc(sizeof(LinkList));//插入节点

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

cur->next = newbase;

cur = newbase;

}

cur->next = NULL;

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 2 \*\*\*\*\*\*\*\*\*\*/

}

status addlist(LinkLists& L,char& name)//添加单链表

{

LinkLists newlist = (LISTS\*)malloc(sizeof(LISTS));//生成新的单链表

if (newlist == NULL) return FALSE;

newlist->name = name;

newlist->cur = NULL;

newlist->next = NULL;

LinkLists tail = L;

while (tail->next != NULL)

{

tail = tail->next;

}

tail->next = newlist;

return OK;

}

status deletelist(LinkLists& L, char& name)//删除名字为name的链表

{

if (L == NULL) return FALSE;

LinkLists prior =L;//前驱

LinkLists now = L->next;//当前

//查找需要删除的表

while (now->name != name && now != NULL)

{

prior = now;

now = now->next;

}

if (now == NULL) return FALSE;//单链表不存在

prior->next = now->next;

now->next = NULL;

delete now;

return OK;

}

LinkLists worklist(LinkLists& L, char& name)//对名为name的单链表进行操作

{

if (L == NULL) return L;//多表不存在

LinkLists now = L->next;

//查找需要操作的单链表

while (now->name != name && now != NULL)

{

now = now->next;

}

if (now == NULL) return FALSE;//单链表不存在

return now;

}

“link.cpp”

#include "linklist.h"

int main(void) {

LinkLists L=NULL;

LinkLists Ls = NULL;

LinkLists temp = NULL;

char name;

int op = 1;

int e;

int i;

status s;

int length;

char filename[] = "./data.txt";

while (op) {

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

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

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

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

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

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

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

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

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

printf(" 13.SaveList 14. LoadList\n");

printf(" 15.InitLists 16.addlisst\n");

printf(" 17.deletelist 18.worklist\n");

printf(" 0. Exit\n");

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

printf("请选择你的操作[0~18]:");

scanf("%d", &op);

switch (op) {

case 1:

//printf("\n----IntiList功能待实现！\n");

if (InitList(L->cur) == OK) printf("单链表创建成功！\n");

else printf("单链表创建失败！\n");

getchar(); getchar();

break;

case 2:

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

if (DestroyList(L->cur) == OK) printf("单链表销毁成功!\n");

else printf("单链表销毁失败！\n");

getchar(); getchar();

break;

case 3:

//printf("\n----ClearList功能待实现！\n");

if (ClearList(L->cur) == OK) printf("单链表清除成功！\n");

else printf("单链表清除失败！\n");

getchar(); getchar();

break;

case 4:

//printf("\n----ListEmpty功能待实现！\n");

if (ListEmpty(L->cur) == TRUE) printf("单链表为空！\n");

else if (ListEmpty(L->cur) == INFEASIBLE) printf("单链表判空失败!\n");

else printf("单链表不为空");

getchar(); getchar();

break;

case 5:

//printf("\n----ListLength功能待实现！\n");

length=ListLength(L->cur);

if (length != INFEASIBLE) printf("单链表长度为：%d\n", length);

else printf("listlength error!\n");

getchar(); getchar();

break;

case 6:

//printf("\n----GetElem功能待实现！\n");

printf("输入你要获得的元素的位置：");

scanf("%d", &i);

s = GetElem(L->cur, i, e);

if (s == INFEASIBLE) printf("单链表为空！\n");

else if (s == ERROR) printf("获取元素失败\n");

else printf("第%d位置的元素是：%d \n", i, e);

getchar(); getchar();

break;

case 7:

//printf("\n----LocateElem功能待实现！\n");

printf("输入你要确定位置的元素:\n");

scanf("%d", &e);

s = LocateElem(L->cur, e);

if (s == INFEASIBLE) printf("单链表为空！\n");

else if (s == ERROR) printf("查找失败！\n");

else printf("元素%d的位置是%d", e, s);

getchar(); getchar();

break;

case 8:

//printf("\n----PriorElem功能待实现！\n");

printf("输入你需要确定前驱的元素\n");

scanf("%d", &e);

int pre;

if (PriorElem(L->cur, e, pre) != FALSE) printf("%d 的前驱是：%d\n", e, pre);

else printf("PriorElem error\n");

getchar(); getchar();

break;

case 9:

//printf("\n----NextElem功能待实现！\n");

printf("输入你要确定后继的元素\n");

scanf("%d", &e);

int next;

if (NextElem(L->cur, e, next) != FALSE) printf("%d 的后继是：%d\n", e, next);

else printf("NextElem error\n");

getchar(); getchar();

break;

case 10:

//printf("\n----ListInsert功能待实现！\n");

printf("输入你要插入的元素\n");

scanf("%d", &e);

printf("输入你要插入的位置\n");

scanf("%d", &i);

if (ListInsert(L->cur, i, e) != FALSE) printf("插入成功\n");

else printf("插入失败\n");

getchar(); getchar();

break;

case 11:

//printf("\n----ListDelete功能待实现！\n");

printf("输入你要删除的位置\n");

scanf("%d", &i);

s = ListDelete(L->cur, i, e);

if (s== OK) printf("删除第%d号位置的元素是：%d\n", i, e);

else printf("删除失败\n");

getchar(); getchar();

break;

case 12:

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

s = ListTraverse(L->cur);

if (s!=OK) printf("线性表是空表！\n");

getchar(); getchar();

break;

case 13:

s = SaveList(L->cur, filename);

if (s == OK) printf("线性表存储成功！\n");

else printf("线性表存储失败!\n");

getchar(); getchar();

break;

case 14:

s = LoadList(L->cur, filename);

if (s == OK) printf("线性表读取成功！\n");

else printf("线性表读取失败！\n");

getchar(); getchar();

break;

case 15:

s = InitLists(Ls);

if (s == OK) printf("多表创建成功\n");

else printf("多表创建失败！\n");

getchar(); getchar();

break;

case 16:

printf("请输入你要添加的单链表的名字:");

getchar();

name = getchar();

s = addlist(Ls, name);

if (s == OK) printf("添加成功！\n");

else printf("添加失败！\n");

getchar(); getchar();

break;

case 17:

printf("请输入你要删除的单链表的名字:");

getchar();

name = getchar();

s = deletelist(Ls, name);

if (s == OK) printf("删除成功!\n");

else printf("删除失败！\n");

getchar(); getchar();

break;

case 18:

printf("请输入你要操作的单链表的名字：");

getchar();

name = getchar();

L = worklist(Ls, name);

if( L->name==name) printf("操作成功！\n");

else printf("操作失败！\n");

getchar(); getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

# 附录C 基于二叉链表二叉树实现的源程序

“def.h”

#pragma once

#include "stdio.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 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 BiTList {//二叉树线性表元素

char name;

BiTree T;

};

typedef struct list {

BiTList Tlist[10];

int length=0;

int listsize = 10;

};

“stu.h”

#pragma once

#include <stack>

#include<queue>

#include<iostream>

#include <fstream>

using namespace std;

int n = 0;

int flag = 0;

status check(TElemType definition[])

{

int i = 0;

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

{

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

{

int j = i + 1;

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

{

if (definition[j].key != 0)

{

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

{

return ERROR;

}

}

j++;

}

}

i++;

}

return 1;

}

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\*/

{

// 请在这里补充代码，完成本关任务

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

if (flag == 0)

{

if (check(definition) == ERROR) return ERROR;

flag = 1;

}

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

if (T == NULL) exit(OVERFLOW);

else

{

if (definition[n].key != -1)

{

if (definition[n].key != 0)

T->data = definition[n];

else

{

T = NULL;

n++;

return OK;

}

}

n++;

CreateBiTree(T->lchild, definition);

CreateBiTree(T->rchild, definition);

}

return OK;

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

}

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

{

// 请在这里补充代码，完成本关任务

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

if (T)

{

ClearBiTree(T->lchild);

ClearBiTree(T->rchild);

free(T);

T = NULL;

}

return OK;

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

}

int BiTreeDepth(BiTree T)

//求二叉树T的深度

{

// 请在这里补充代码，完成本关任务

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

int length = 0;

if (T)

{

int llength = BiTreeDepth(T->lchild);

int rlength = BiTreeDepth(T->rchild);

length= (llength >= rlength ? llength : rlength) + 1;

}

return length;

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

}

BiTNode\* LocateNode(BiTree T, KeyType e)

//查找结点

{

// 请在这里补充代码，完成本关任务

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

BiTree search = NULL;

if (T)

{

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

{

search = T;

return search;

}

search = LocateNode(T->lchild, e);

if (search == NULL)

{

search = LocateNode(T->rchild, e);

}

}

return search;

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

}

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

{

// 请在这里补充代码，完成本关任务

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

BiTree temp = NULL;

temp = LocateNode(T, e);

if (temp&&temp->data.key<=value.key)

{

temp->data = value;

return OK;

}

else

return 0;

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

}

BiTNode\* GetSibling(BiTree T, KeyType e)

//实现获得兄弟结点

{

// 请在这里补充代码，完成本关任务

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

BiTree brother = NULL;

if (T)

{

if(T->lchild)

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

{

brother = T->rchild;

return brother;

}

if(T->rchild)

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

{

brother = T->lchild;

return brother;

}

brother = GetSibling(T->lchild, e);

brother = GetSibling(T->rchild, e);

}

return brother;

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

}

BiTree Getparent(BiTree& T, KeyType e)

{

BiTree parent = NULL;

if (T)

{

if (T)

{

if (T->lchild)

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

{

parent = T;

return parent;

}

if (T->rchild)

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

{

parent=T;

return parent;

}

parent = Getparent(T->lchild, e);

parent = Getparent(T->rchild, e);

}

}

return parent;

}

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

{

// 请在这里补充代码，完成本关任务

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

if (LocateNode(T, c.key)) return ERROR;

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

{

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

temp->data = c;

temp->rchild = T;

T = temp;

return OK;

}

if (LR == -1)

{

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

temp->data = c;

temp->lchild = NULL;

temp->rchild = T;

T = temp;

return OK;

}

BiTree search = LocateNode(T, e);

if (search == NULL) return ERROR;

if (LR == 0)

{

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

temp->data = c;

temp->lchild = NULL;

temp->rchild = search->lchild;

search->lchild = temp;

return OK;

}

if (LR == 1)

{

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

temp->data = c;

temp->lchild = NULL;

temp->rchild = search->rchild;

search->rchild = temp;

return OK;

}

return ERROR;

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

}

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

{

// 请在这里补充代码，完成本关任务

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

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

{

BiTree p = T;

if (T->lchild != NULL)

{

BiTree temp = T->rchild;

T = T->lchild;

BiTree rr = T;

while (rr->rchild)

{

rr = rr->rchild;

}

rr->rchild = temp;

free(p);

return OK;

}

else

{

T = T->rchild;

free(p);

return OK;

}

}

BiTree parent = Getparent(T, e);

if (parent == NULL) return ERROR;

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

{

BiTree p = parent->lchild;

if (p->lchild != NULL)

{

parent->lchild = p->lchild;

BiTree rr =p;

while (rr->rchild)

{

rr = rr->rchild;

}

rr->rchild = p->rchild;

free(p);

return OK;

}

else

{

parent->lchild = p->rchild;

free(p);

return OK;

}

}

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

{

BiTree p = parent->rchild;

if (p->lchild != NULL)

{

parent->rchild = p->lchild;

BiTree rr = p;

while (rr->rchild)

{

rr = rr->rchild;

}

rr->rchild = p->rchild;

free(p);

return OK;

}

else

{

parent->rchild = p->rchild;

free(p);

return OK;

}

}

return ERROR;

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

}

void visit(BiTree T)

{

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

}

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

//先序遍历二叉树T

{

// 请在这里补充代码，完成本关任务

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

if (T)

{

visit(T);

if (PreOrderTraverse(T->lchild, visit));

if (PreOrderTraverse(T->rchild, visit)) return OK;

return ERROR;

}

else return OK;

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

}

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

//中序遍历二叉树T

{

// 请在这里补充代码，完成本关任务

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

stack<BiTree> s;

s.push(T);

while (!s.empty())

{

BiTree p =s.top();

while (p)

{

s.push(p->lchild);

p = s.top();

}

p = s.top();

s.pop();

if (!s.empty())

{

p = s.top();

s.pop();

visit(p);

s.push(p->rchild);

}

}

return OK;

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

}

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

//后序遍历二叉树T

{

// 请在这里补充代码，完成本关任务

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

if (T)

{

PostOrderTraverse(T->lchild,visit);

PostOrderTraverse(T->rchild,visit);

visit(T);

}

return OK;

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

}

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

//按层遍历二叉树T

{

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

queue<BiTree> s;

if (T) s.push(T);

while (!s.empty())

{

BiTree p = s.front();

s.pop();

visit(p);

if (p->lchild) s.push(p->lchild);

if (p->rchild) s.push(p->rchild);

}

return OK;

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

}

void iterSave(FILE\* fp, BiTree ptr) {

if (ptr) {

fprintf(fp, "%d ", ptr->data.key);

fprintf(fp, "%s", ptr->data.others);

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

iterSave(fp, ptr->lchild);

iterSave(fp, ptr->rchild);

}

else {

fprintf(fp, "%d", 0);

fprintf(fp,"%s", " #");

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

}

}

status SaveBiTree(BiTree T, char FileName[])

//将二叉树的结点数据写入到文件FileName中

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 1 \*\*\*\*\*\*\*\*\*/

FILE\* fp = fopen(FileName, "w");

if (fp == NULL)

{

fp = fopen(FileName, "wb");

}

iterSave(fp, T);

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 1 \*\*\*\*\*\*\*\*\*\*/

}

status iterLoad(FILE\* fp,BiTree& T)

{

int key=0;

char other[20] = {};

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

int i = 0;

fscanf(fp, "%c", other);

while (\*(other + i) != '$')

{

i++;

fscanf(fp, "%c", other + i);

}

if (key == 0) T = NULL;

else

{

if (!(T = (BiTNode\*)malloc(sizeof(BiTNode))))

exit(OVERFLOW);

T->data.key = key;

for (int k = 0; k < 20; k++)

{

T->data.others[k] ='\0';

}

int j = 1;

while (\*(other + j) != '$')

{

\*(T->data.others + j-1) = \*(other + j);

j++;

}

iterLoad(fp, T->lchild);

iterLoad(fp, T->rchild);

}

return OK;

}

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

//读入文件FileName的结点数据，创建二叉树

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 2 \*\*\*\*\*\*\*\*\*/

FILE\* fp = fopen(FileName, "r");

if (fp == NULL)

{

return ERROR;

}

iterLoad(fp, T);

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 2 \*\*\*\*\*\*\*\*\*\*/

}

status Initdata(TElemType definition[])

{

int i = 0;

do {

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

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

return OK;

}

status AddTree(list &L,char name)

{

if (L.length <= 10)

{

L.Tlist[L.length].name = name;

L.length++;

return OK;

}

else return FALSE;

}

status DeleteTree(list &L,char name)

{

int i = 0;

do

{

if (L.Tlist[i].name == name)

{

free(L.Tlist[i].T);

L.Tlist[i].name = NULL;

L.length--;

return OK;

}

i++;

} while (i < L.length);

if (i == L.length)

{

return FALSE;

}

else return OK;

}

int WorkTree(list &L, char name)

{

n = 0;

flag = 0;

int i = 0;

do

{

if (L.Tlist[i].name == name)

{

return i;

}

i++;

} while (i < L.length);

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

{

return -1;

}

}

“main.h”

#include "def.h"

#include "stu.h"

#include "string.h"

char Filename[] = "./data.txt";

int main(void) {

BiTree T=NULL ;

TElemType definition[100] ;

list L;

int length;

int key;

int ans;

BiTNode\* p = NULL;

TElemType e;

int LR;

char name;

int op=1;

int now=0;

printf("\n");

while (op) {

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

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

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

printf(" 1. CreateBiTree 10.InOrderTraverse\n");

printf(" 2. ClearBiTree 11.PostOrderTraverse\n");

printf(" 3. BiTreeDepth 12.LevelOrderTraverse \n");

printf(" 4. LocateNode 13.SaveBiTree\n");

printf(" 5. Assign 14.LoadBiTree\n");

printf(" 6. GetSibling 15.Initdata\n");

printf(" 7. InsertNode 16.AddTree\n");

printf(" 8. DeleteNode 17.DeleteTree\n");

printf(" 9. PreOrderTraverse 18.WorkTree\n");

printf(" 0. Exit\n");

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

printf("请选择你的操作[0~18]:");

scanf("%d", &op);

switch (op) {

case 1:

if (CreateBiTree(L.Tlist[now].T,definition) == OK) printf("二叉树创建成功！\n");

else printf("二叉树创建失败！\n");

getchar(); getchar();

break;

case 2:

if (ClearBiTree(L.Tlist[now].T) == OK) printf("二叉树销毁成功!\n");

else printf("二叉树销毁失败！\n");

getchar(); getchar();

break;

case 3:

length = BiTreeDepth(L.Tlist[now].T);

printf("二叉树的深度：%d\n", length);

getchar(); getchar();

break;

case 4:

printf("输入你要查询的结点的关键字：");

cin >> key;

p = LocateNode(L.Tlist[now].T, key);

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

else printf("查找失败\n");

getchar(); getchar();

break;

case 5:

printf("输入你要赋值的关键字： ");

cin >> key;

printf("输入修改后的值：");

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

ans = Assign(L.Tlist[now].T, key, e);

if (ans == OK)

{

printf("赋值操作成功");

}

else printf("赋值操作失败");

getchar(); getchar();

break;

case 6:

printf("输入你要获得兄弟结点的关键字：");

scanf("%d", &key);

p = GetSibling(L.Tlist[now].T, key);

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

else printf("无兄弟结点");

getchar(); getchar();

break;

case 7:

printf("输入你要插入的位置(0是左,1是右,-1是根结点):\n");

scanf("%d", &LR);

printf("输入你要插入结点的关键字：");

scanf("%d", &key);

printf("输入要插入的数据：");

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

ans = InsertNode(L.Tlist[now].T, key, LR, e);

if (ans == OK)

{

printf("插入操作成功");

}

else printf("插入操作失败");

getchar(); getchar();

break;

case 8:

printf("输入你要删除结点的关键字：");

scanf("%d", &key);

ans = DeleteNode(L.Tlist[now].T, key);

if (ans == OK)

{

printf("删除操作成功");

}

else printf("删除操作失败");

getchar(); getchar();

break;

case 9:

printf("先序遍历二叉树的结果：\n");

PreOrderTraverse(L.Tlist[now].T, visit);

getchar(); getchar();

break;

case 10:

printf("中序遍历二叉树的结果：\n");

InOrderTraverse(L.Tlist[now].T, visit);

getchar(); getchar();

break;

case 11:

printf("后序遍历二叉树的结果：\n");

PostOrderTraverse(L.Tlist[now].T, visit);

getchar(); getchar();

break;

case 12:

printf("层次遍历二叉树的结果：\n");

LevelOrderTraverse(L.Tlist[now].T,visit);

getchar(); getchar();

break;

case 13:

ans=SaveBiTree(L.Tlist[now].T, Filename);

if (ans == OK) printf("存储成功\n");

else printf("存储失败\n");

getchar(); getchar();

break;

case 14:

ans = LoadBiTree(L.Tlist[now].T, Filename);

if (ans == OK) printf("读取成功\n");

else printf("读取失败\n");

getchar(); getchar();

break;

case 15:

printf("输入二叉树先序遍历的数据:\n");

ans = Initdata(definition);

if (ans == OK) printf("输入成功");

else printf("输入失败");

getchar(); getchar();

break;

case 16:

printf("请输入你要添加的二叉树的名字:");

getchar();

scanf("%c", &name);

ans = AddTree(L, name);

if (ans == OK) printf("添加成功");

else printf("添加失败");

getchar(); getchar();

break;

case 17:

printf("请输入你要删除的二叉树的名字:");

getchar();

scanf("%c", &name);

ans = DeleteTree(L, name);

if (ans == OK) printf("删除成功");

else printf("删除失败");

getchar(); getchar();

break;

case 18:

printf("请输入你要操作的二叉树的名字：");

getchar();

scanf("%c", &name);

now= WorkTree(L, name);

if (now!=-1) printf("操作成功");

else printf("操作失败");

getchar(); getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

return 0;

}//end of main()

# 附录D 基于邻接表图实现的源程序

“def.h”

#pragma once

#pragma once

#include "stdio.h"

#include "stdlib.h"

#include <queue>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 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 Glist//多表管理

{

char name;

ALGraph G;//图

};

“stu.h”

#pragma once

#pragma once

#include "def.h"

status CreateCraph(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);

status DeleteVex(ALGraph& G, KeyType v);

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

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

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

void visit(VertexType v);

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

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

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

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

int workGraph(Glist GList[], char name);

status addGraph(Glist GList[], char name);

status deleteGraph(Glist GList[], char name);

“grap.cpp”

#include "stu.h"

int Glength = 0;

bool visited[MAX\_VERTEX\_NUM];

using namespace std;

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

{

if (V[0].key == -1) return ERROR;//判断顶点个数是否为0

int hash[MAX\_VERTEX\_NUM] = { 0 };//储存关键字的位置

G.vexnum = 0;

G.kind = UDG;//无向图

while (V[G.vexnum].key != -1)//顶点

{

G.vexnum++;//顶点个数增加

if (hash[V[G.vexnum - 1].key] != 0) return ERROR;

G.vertices[G.vexnum - 1].data = V[G.vexnum - 1];//存储顶点

G.vertices[G.vexnum - 1].firstarc = NULL;//弧的初始化

hash[V[G.vexnum - 1].key] = G.vexnum;//顶点位置加1

}

if (G.vexnum > 20) return ERROR;

G.arcnum = 0;//弧的个数

while (VR[G.arcnum][0] != -1)//弧

{

int temp1 = hash[VR[G.arcnum][0]];//顶点1在顶点数组中的位置加1

int temp2 = hash[VR[G.arcnum][1]];//顶点2在顶点数组中的位置加1

if (temp1 == 0 | temp2 == 0) return ERROR;

ArcNode\* p1 = (ArcNode\*)malloc(sizeof(ArcNode));//生成弧1

ArcNode\* p2 = (ArcNode\*)malloc(sizeof(ArcNode));//生成弧2

//弧1赋值

p1->adjvex = temp2 - 1;

p1->nextarc = G.vertices[temp1 - 1].firstarc;

G.vertices[temp1 - 1].firstarc = p1;

//弧2赋值

p2->adjvex = temp1 - 1;

p2->nextarc = G.vertices[temp2 - 1].firstarc;

G.vertices[temp2 - 1].firstarc = p2;

G.arcnum++;//弧的个数加1

}

return OK;

}

status DestroyGraph(ALGraph& G)

/\*销毁无向图G\*/

{

// 请在这里补充代码，完成本关任务

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

for (int i = 0; i < G.vexnum; i++)//对每个顶点进行销毁操作

{

while (G.vertices[i].firstarc)//弧不为空则释放

{

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

free(G.vertices[i].firstarc);//释放空间

G.vertices[i].firstarc = p;

}

}

G.vexnum = 0;

G.arcnum = 0;

return OK;

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

}

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；

{

// 请在这里补充代码，完成本关任务

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

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

{

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

}

return ERROR;

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

}

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

{

// 请在这里补充代码，完成本关任务

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

if (LocateVex(G, value.key) != ERROR) return ERROR;

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

{

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

{

G.vertices[i].data = value;

return OK;

}

}

return ERROR;

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

}

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 temp = 0;

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

{

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

{

temp = G.vertices[i].firstarc->adjvex;

return temp;

}

}

return -1;

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

}

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

//根据u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；

{

// 请在这里补充代码，完成本关任务

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

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

{

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

{

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

while (p->nextarc)

{

if (G.vertices[p->adjvex].data.key == w)

{

int temp = p->nextarc->adjvex;

return temp;

}

else

{

p = p->nextarc;

}

}

return -1;

}

}

return -1;

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

}

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

{

// 请在这里补充代码，完成本关任务

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

if (G.vexnum == MAX\_VERTEX\_NUM) //顶点已满

return ERROR;

for (int i = 0; i < G.vexnum; i++)//查询关键字是否唯一

{

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

{

return ERROR;

}

}

G.vertices[G.vexnum].data = v;//插入关键字

G.vexnum++;

return OK;

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

}

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

{

// 请在这里补充代码，完成本关任务

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

int place = -1;//记录v的位序

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

{

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

{

place = i;

break;

}

}

if (place == 0 && G.vexnum == 1) return ERROR;

if (place == -1)

return ERROR;//v不存在

else//删除顶点及其有关的弧

{

for (int i = 0; i < G.vexnum; i++)//删除弧

{

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

if (p && p->adjvex == place)

{

G.vertices[i].firstarc = G.vertices[i].firstarc->nextarc;

p->nextarc = NULL;

free(p);

G.arcnum--;

}

if (G.vertices[i].firstarc)

{

if (G.vertices[i].firstarc->adjvex > place)

{

G.vertices[i].firstarc->adjvex--;

}

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

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

while (p)

{

if (p->adjvex == place)

{

ArcNode\* temp = p;

p = p->nextarc;

father->nextarc = temp->nextarc;

temp->nextarc = NULL;

free(temp);

G.arcnum--;

}

else

{

if (p->adjvex > place)

p->adjvex--;

father = p;

p = p->nextarc;

}

}

}

}

ArcNode\* tp = G.vertices[place].firstarc;

while (tp)

{

ArcNode\* tp0 = tp->nextarc;

tp->nextarc = NULL;

free(tp);

tp = tp0;

}

for (int i = place; i < G.vexnum - 1; i++)//删除顶点

{

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

ArcNode\* temp1 = G.vertices[i + 1].firstarc;

G.vertices[i].firstarc = temp1;

}

G.vertices[G.vexnum - 1].data.key = NULL;

G.vertices[G.vexnum - 1].firstarc = NULL;

G.vexnum--;

}

return OK;

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

}

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

{

// 请在这里补充代码，完成本关任务

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

int place0 = -1;//记录v的位序

int place1 = -1;//记录w的位序

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

{

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

{

place0 = i;

}

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

{

place1 = i;

}

}

ArcNode\* p = G.vertices[place0].firstarc;

while (p)

{

if (p->adjvex == place1)

return ERROR;

p = p->nextarc;

}

if (place0 == -1 || place1 == -1) return ERROR;

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

newnode->adjvex = place1;

newnode->nextarc = G.vertices[place0].firstarc;

G.vertices[place0].firstarc = newnode;

ArcNode\* newnode1 = (ArcNode\*)malloc(sizeof(ArcNode));

newnode1->adjvex = place0;

newnode1->nextarc = G.vertices[place1].firstarc;

G.vertices[place1].firstarc = newnode1;

G.arcnum++;

return OK;

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

}

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

{

// 请在这里补充代码，完成本关任务

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

int deleteflag = 0;

int place0 = -1;//记录v的位序

int place1 = -1;//记录w的位序

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

{

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

{

place0 = i;

}

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

{

place1 = i;

}

}

if (place0 == -1 || place1 == -1) return ERROR;

if (G.vertices[place0].firstarc->adjvex == place1)

{

ArcNode\* p = G.vertices[place0].firstarc;

G.vertices[place0].firstarc = p->nextarc;

free(p);

deleteflag = 1;

}

else

{

ArcNode\* father = G.vertices[place0].firstarc;

ArcNode\* p = father->nextarc;

while (p)

{

if (p->adjvex == place1)

{

father->nextarc = p->nextarc;

free(p);

deleteflag = 1;

break;

}

else

{

father = p;

p = p->nextarc;

}

}

}

if (deleteflag != 1) return ERROR;

if (G.vertices[place1].firstarc->adjvex == place0)

{

ArcNode\* p = G.vertices[place1].firstarc;

G.vertices[place1].firstarc = p->nextarc;

free(p);

}

else

{

ArcNode\* father = G.vertices[place1].firstarc;

ArcNode\* p = father->nextarc;

while (p)

{

if (p->adjvex == place0)

{

father->nextarc = p->nextarc;

free(p);

break;

}

else

{

father = p;

p = p->nextarc;

}

}

}

G.arcnum--;

return OK;

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

}

void visit(VertexType v)

{

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

}

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

{

visited[v] = TRUE;

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

for (int w = G.vertices[v].firstarc->adjvex; w >= 0; w = NextAdjVex(G, G.vertices[v].data.key, G.vertices[w].data.key))

{

if (!visited[w]) DFS(G, w, 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

{

// 请在这里补充代码，完成本关任务

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

int v = 0;

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

{

visited[v] = FALSE;

}

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

{

if (!visited[v])

{

DFS(G, v, visit);

}

}

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

}

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

{

// 请在这里补充代码，完成本关任务

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

int v = 0;

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

{

visited[v] = FALSE;

}

queue<int> Q;//置空辅助队列

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

{

if (!visited[v])

{

visited[v] = TRUE;

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

Q.push(v);

while (!Q.empty())

{

int u = Q.front();

Q.pop();

for (int w = G.vertices[u].firstarc->adjvex; w >= 0; w = NextAdjVex(G, G.vertices[v].data.key, G.vertices[w].data.key))

{

if (!visited[w])

{

visited[w] = TRUE;

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

Q.push(w);

}

}

}

}

}

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

}

status SaveGraph(ALGraph G, char FileName[])

//将图的数据写入到文件FileName中

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 1 \*\*\*\*\*\*\*\*\*/

FILE\* fp = fopen(FileName, "wb");

VertexType V[MAX\_VERTEX\_NUM];//储存顶点信息

int VR[1000][2];//储存弧

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

{

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

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

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

}//储存顶点

int j = 0;

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

{

ArcNode\* p = G.vertices[i].firstarc;//第一临接点

while (p)

{

if (p->adjvex > i)//储存弧的信息

{

VR[j][0] = G.vertices[i].data.key;

VR[j][1] = G.vertices[p->adjvex].data.key;

j++;

}

p = p->nextarc;

}

}

VR[j][0] = -1;

VR[j][1] = -1;

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

{

fprintf(fp, "%d", VR[i][0]);

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

fprintf(fp, "%d", VR[i][1]);

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

}//储存弧

fclose(fp);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 1 \*\*\*\*\*\*\*\*\*\*/

}

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

//读入文件FileName的图数据，创建图的邻接表

{

// 请在这里补充代码，完成本关任务

/\*\*\*\*\*\*\*\*\*\* Begin 2 \*\*\*\*\*\*\*\*\*/

VertexType V[MAX\_VERTEX\_NUM];//储存顶点信息

char other[20] = {};

int VR[1000][2];//储存弧

FILE\* fp = fopen(FileName, "rb");

int i = 0;

fscanf(fp, "%d", &V[i].key);

fscanf(fp, "%c", other);

int j = 0;

while (\*(other + j) != '$')

{

j++;

fscanf(fp, "%c", other + j);

}

int k = 0;

for (k = 0; \*(other + k) != '$'; k++)

{

\*(V[i].others + k) = \*(other + k);

}

for (; k < 20; k++)

{

\*(V[i].others + k) = NULL;

}

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

{

i++;

char other[20] = {};

fscanf(fp, "%d", &V[i].key);

fscanf(fp, "%c", other);

int j = 0;

while (\*(other + j) != '$')

{

j++;

fscanf(fp, "%c", other + j);

}

int k = 0;

for (k = 0; \*(other + k) != '$'; k++)

{

\*(V[i].others + k) = \*(other + k);

}

for (; k < 20; k++)

{

\*(V[i].others + k) = NULL;

}

}//读取顶点信息

i = 0;

char o = ' ';

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

fscanf(fp, "%c", &o);

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

fscanf(fp, "%c", &o);

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

{

i++;

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

fscanf(fp, "%c", &o);

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

fscanf(fp, "%c", &o);

}

for (int i = 0; VR[i][0] != -1; i++)

{

int min0 = VR[i][0];

int min1 = VR[i][1];

int min = 10 \* min0 + min1;

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

{

if (VR[j][0] \* 10 + VR[j][1] < min)

{

int temp0 = VR[j][0];

int temp1 = VR[j][1];

VR[j][0] = VR[i][0];

VR[j][1] = VR[i][1];

VR[i][0] = temp0;

VR[i][1] = temp1;

}

}

}

CreateCraph(G, V, VR);

return OK;

/\*\*\*\*\*\*\*\*\*\* End 2 \*\*\*\*\*\*\*\*\*\*/

}

int workGraph(Glist GList[], char name)

{

for (int i = 0; i < Glength; i++)

{

if (GList[i].name == name)

{

return i;

}

}

return -1;

}

status addGraph(Glist GList[], char name)

{

GList[Glength].name = name;

Glength++;

return OK;

}

status deleteGraph(Glist GList[], char name)

{

int flag = 0;

int i = 0;

for (i; i < Glength; i++)

{

if (GList[i].name == name)

{

flag = 1;

break;

}

}//查找到需要删除的位置

if (flag == 0) return ERROR;

DestroyGraph(GList[i].G);

ALGraph\* temp,\*p;

for (; i < Glength - 1; i++)

{

temp = &GList[i].G;

p = &GList[i + 1].G;

\*temp = \*p;

GList[i].name = GList[i + 1].name;

}

DestroyGraph(GList[Glength - 1].G);

Glength--;

return OK;

}

“main.cpp”

#include "def.h"

#include "stu.h"

#include "string.h"

char Filename[] = "./data.txt";

int main(void) {

printf("ok\n");

Glist GList[10];//多图数组

int now = 0;//当前操作的图

char name;

int op = 1;

VertexType V[30];

KeyType VR[100][2];

int key = 0;

int locate = 0;

VertexType value;

int s = 0;

int w = 0;

int gi = 0;

printf("\n");

while (op) {

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

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

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

printf(" 1. CreateCraph 10.DeleteArc\n");

printf(" 2. DestroyGraph 11.DFSTraverse\n");

printf(" 3. LocateVex 12.BFSTraverse \n");

printf(" 4. PutVex 13.SaveGraph\n");

printf(" 5. FirstAdjvex 14.LoadGraph\n");

printf(" 6. NextAdjvex 15.WorkGraph\n");

printf(" 7. InsertVex 16.AddGraph\n");

printf(" 8. DeleteVex 17.DeleteGraph\n");

printf(" 9. InsertArc\n");

printf(" 0. Exit\n");

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

printf("请选择你的操作[0~18]:");

scanf("%d", &op);

switch (op) {

case 1:

gi = 0;

do {

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

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

gi = 0;

do {

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

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

if (CreateCraph(GList[now].G, V, VR) == ERROR) printf("输入数据错，无法创建");

else printf("二叉树创建成功！\n");

getchar(); getchar();

break;

case 2:

if (DestroyGraph(GList[now].G) == OK) printf("无向图销毁成功!\n");

else printf("无向图销毁失败！\n");

getchar(); getchar();

break;

case 3:

printf("输入你要查找的顶点的关键字：");

scanf("%d", &key);

locate = LocateVex(GList[now].G, key);

printf("%d %s\n", GList[now].G.vertices[locate].data.key, GList[now].G.vertices[locate].data.others);

getchar(); getchar();

break;

case 4:

printf("输入你要赋值的顶点的关键字：");

scanf("%d", &key);

printf("输入你修改后的内容：");

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

s = PutVex(GList[now].G, key, value);

if (s) printf("修改成功\n");

else printf("修改失败\n");

getchar(); getchar();

break;

case 5:

printf("输入你要查询第一邻接点的关键字： ");

scanf("%d", &key);

s = FirstAdjVex(GList[now].G, key);

if (s == -1)printf("查找失败\n");

else

{

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

}

getchar(); getchar();

break;

case 6:

printf("输入你要获得邻接点顶点关键字：");

scanf("%d", &key);

printf("输入你要邻接点的关键字：");

scanf("%d", &w);

s = NextAdjVex(GList[now].G, key, w);

if (s == -1)printf("查找失败\n");

else

{

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

}

getchar(); getchar();

break;

case 7:

printf("输入你要插入的顶点:\n");

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

s = InsertVex(GList[now].G, value);

if (s == OK)

{

printf("插入操作成功");

}

else printf("插入操作失败");

getchar(); getchar();

break;

case 8:

printf("输入你要删除顶点的关键字：");

scanf("%d", &key);

s = DeleteVex(GList[now].G, key);

if (s == OK)

{

printf("删除操作成功");

}

else printf("删除操作失败");

getchar(); getchar();

break;

case 9:

printf("输入你要插入的弧：\n");

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

s = InsertArc(GList[now].G, key, w);

if (s == OK) printf("插入成功\n");

else printf("插入失败\n");

getchar(); getchar();

break;

case 10:

printf("输入你要删除的弧：\n");

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

s = DeleteArc(GList[now].G, key, w);

if (s == OK) printf("删除成功\n");

else printf("删除失败\n");

getchar(); getchar();

break;

case 11:

printf("深度遍历图的结果：\n");

DFSTraverse(GList[now].G, visit);

getchar(); getchar();

break;

case 12:

printf("广度遍历图的结果：\n");

BFSTraverse(GList[now].G, visit);

getchar(); getchar();

break;

case 13:

s = SaveGraph(GList[now].G, Filename);

if (s == OK) printf("存储成功\n");

else printf("存储失败\n");

getchar(); getchar();

break;

case 14:

s = LoadGraph(GList[now].G, Filename);

if (s == OK) printf("读取成功\n");

else printf("读取失败\n");

getchar(); getchar();

break;

case 16:

printf("请输入你要添加的图的名字:");

getchar();

scanf("%c", &name);

s = addGraph(GList, name);

if (s == OK) printf("添加成功");

else printf("添加失败");

getchar(); getchar();

break;

case 17:

printf("请输入你要删除的图的名字:");

getchar();

scanf("%c", &name);

s = deleteGraph(GList, name);

if (s == OK) printf("删除成功");

else printf("删除失败");

getchar(); getchar();

break;

case 15:

printf("请输入你要操作的图的名字：");

getchar();

scanf("%c", &name);

now = workGraph(GList, name);

if (now != -1) printf("操作成功");

else printf("操作失败");

getchar(); getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

return 0;

}//end of main()