bl



**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期： 2020年 10月 23 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 3](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159879)

[1.1 问题描述 3](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159880)

[1.2 系统设计 5](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159882)

[1.3 系统实现 6](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159883)

[1.4 系统测试 13](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159883)

[1.5 实验小结 18](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159884)

[2 基于链式存储结构的线性表实现 19](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159885)

[2.1 问题描述 19](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159886)

[2.2 系统设计 20](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159887)

[2.3 系统实现 2](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159888)1

[2.4 系统测试 33](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159883)

[2.5 实验小结 48](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159889)

[3 基于二叉链表的二叉树实现 49](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159890)

[3.1 问题描述 49](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159891)

[3.2 系统设计 51](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159892)

[3.3 系统实现 53](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159893)

[3.4 系统测试 67](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159883)

[3.5 实验小结 80](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159894)

[4 基于邻接表的图实现 81](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159895)

[4.1 问题描述 81](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159896)

[4.2 系统设计 83](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159897)

[4.3 系统实现 84](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159898)

[4.4 系统测试 98](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159883)

[4.5 实验小结 110](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159899)

[参考文献 111](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159900)

[附录A 基于顺序存储结构线性表实现的源程序 112](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159901)

[附录B 基于链式存储结构线性表实现的源程序 12](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159902)7

[附录C 基于二叉链表二叉树实现的源程序 144](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159903)

[附录D 基于邻接表图实现的源程序 167](file:///C:\Users\spiri\Downloads\数据结构课程实验报告.docx#_Toc458159904)

# 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 线性表的运算定义

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,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,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()。

### 1.1.2 实验任务

采用顺序表作为线性表的物理结构，实现§1.2的基本运算。其中ElemType为数据元素的类型名，具体含义可自行定义。

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。

演示系统可选择实现线性表的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存线性表数据逻辑结构(D,{R})的完整信息；②需要设计线性表文件保存和加载操作合理模式。附录B提供了文件存取的参考方法。

演示系统可选择实现多个线性表管理。

演示系统的源程序应按照代码规范增加注释和排版，目标程序务必是可以独立于IDE运行的EXE文件。

## 1.2 系统设计

结合单线性表和多线性表的操作之间的关系设计出演示系统的交互，首先创建一个包含多个线性表的结构体Lists，可以包含多个有名字标志的线性表以便进行选中指定线性表进行基本操作，系统设计如图1-1所示

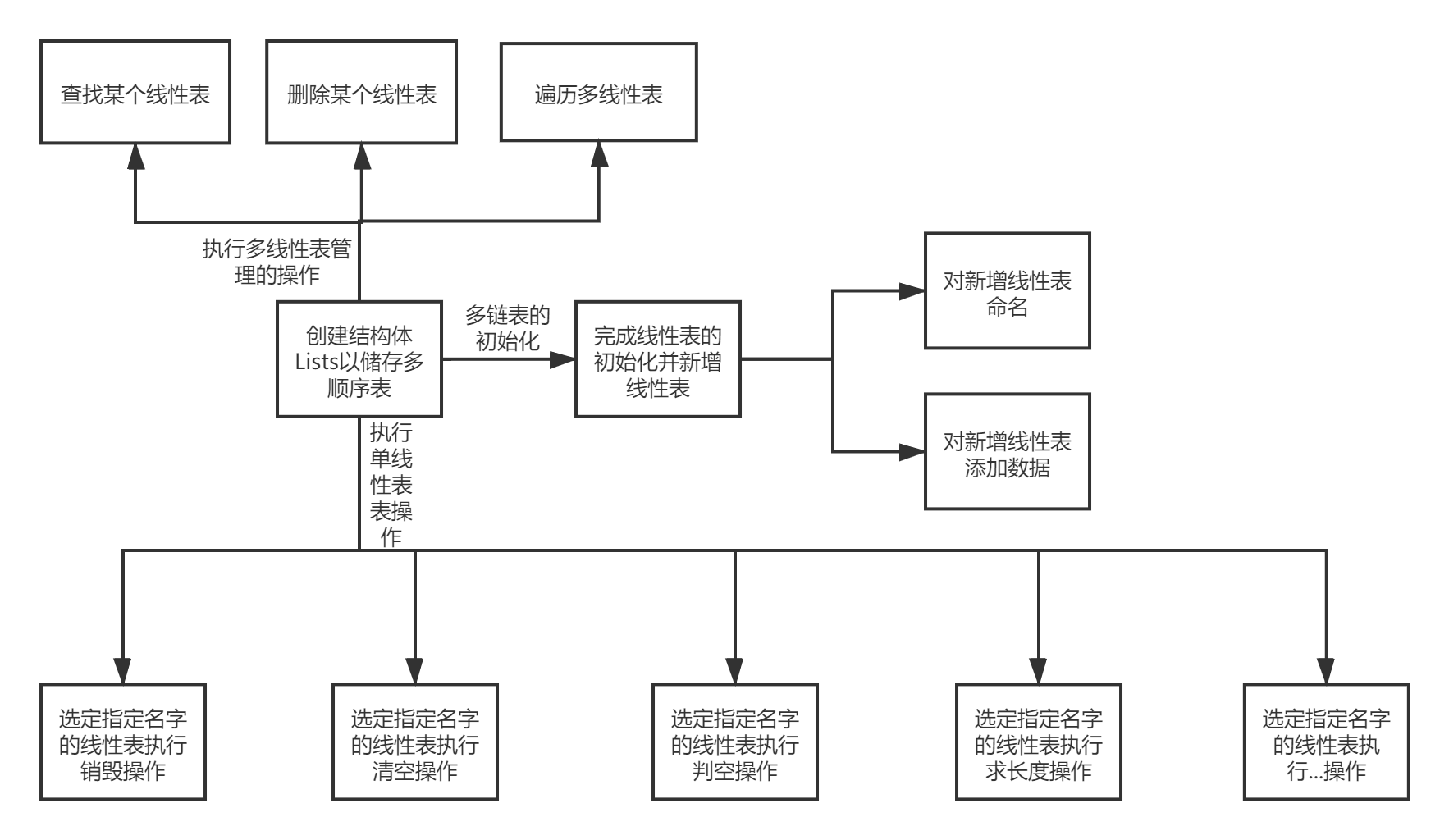


图1-1-1系统设计图

## 1.3 系统实现

### 1.3.1 主函数部分

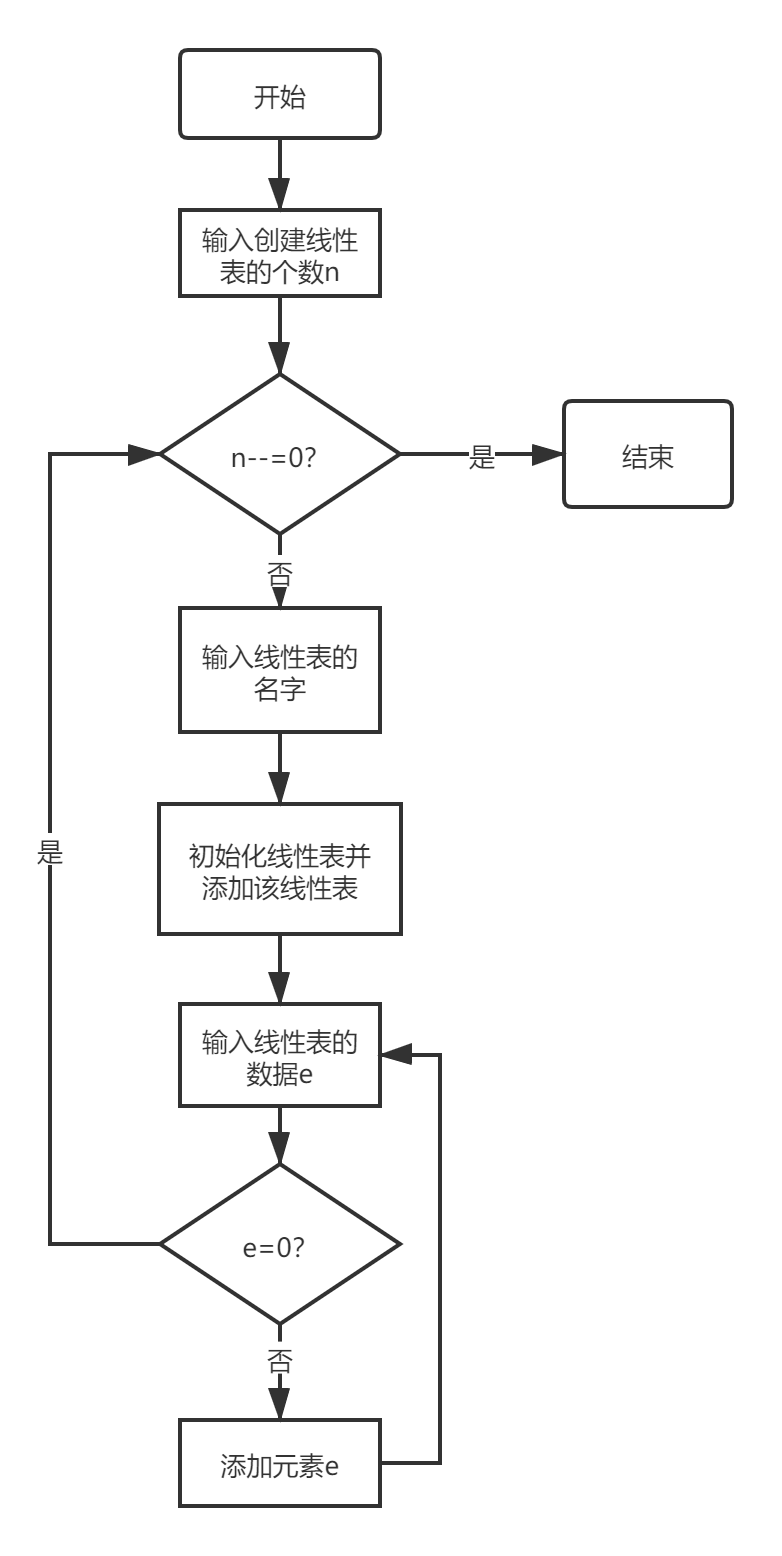
由于要实现构造出一个具有菜单功能的功能演示系统，在这其中要在主程序中完成函数的调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。所以在主函数部分要打印出相应的菜单界面，并且使用system("cls");函数在执行完一次功能之后使界面清空，使界面显得干净整洁，方便操作。同时为了实现选择对应功能来进行相应的操作，采用switch-case语句完成函数选择以完成相应的操作。

此外在具体功能的实现方面，由于要进行单线性表的操作，以及多线性表的管理，要在主函数中声明单顺序表SqList L，多顺序表表 LISTS Lists和储存单顺序表名字的字符数组char name[30]

### 1.3.2功能函数部分

1. 添加线性表：

添加多线性表是初始化一个线性表的延伸，因为在次函数中使用到了单个线性表和多线性表的初始化操作，单线性表加入到多线性表的操作。首先请求用户说明要新增的线性表的个数，接着请求用户输入这些线性表的名字和其储存的数据。



1. 清空线性表

首先请求用户输入要清空的线性表的名字，判断要清空的线性表是否存在，如果存在则执行清空操作，将这个线性表的长度置为零即可。

1. 线性表判空

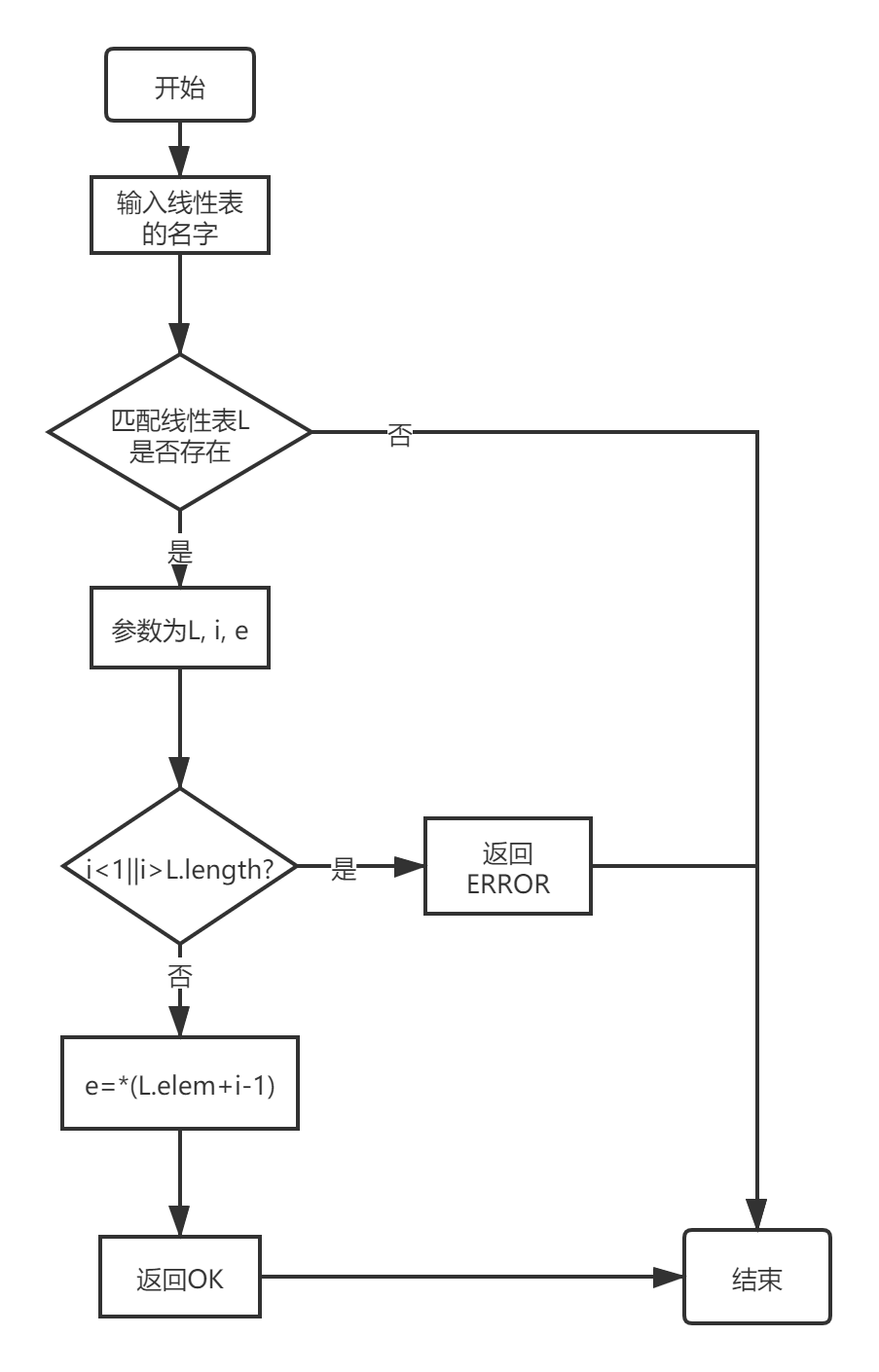
首先请求用户输入要进行判空操作的线性表的名字，判断要判空的线性表是否为存在，如果存在则执行判空操作，如果线性表的长度为0，则为空，如果不为0，则不是空线性表。

1. 线性表长度

首先请求用户输入要进行获取长度的线性表的名字，判断此线性表是否存在，如果存在则执行返回长度的操作

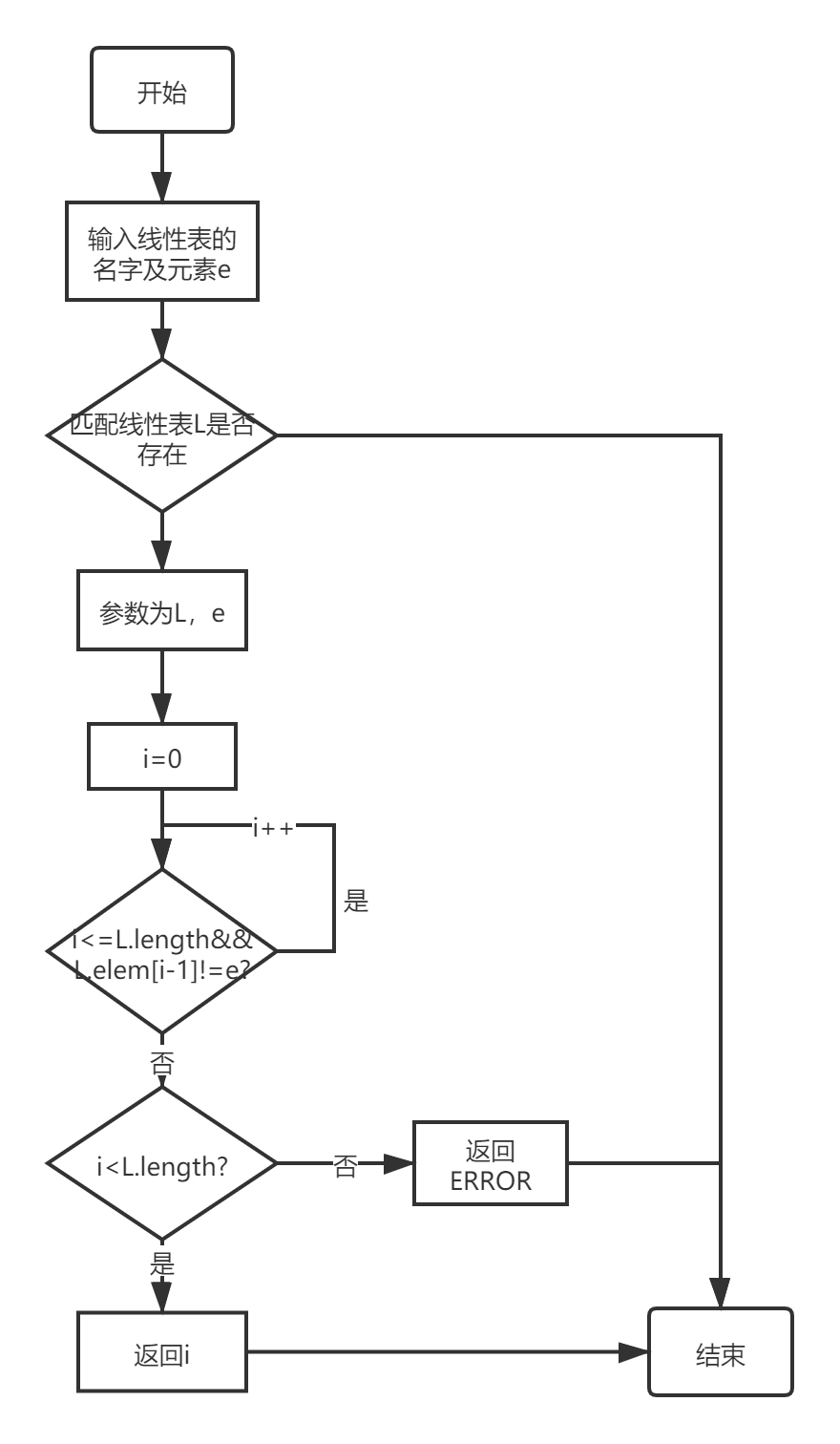
1. 获取元素

首先请求用户输入要进行获取元素操作的线性表的名字，判断此线性表是否存在，如果存在则进行再判断，如果获取元素的位置小于1或大于线性表的长度则返回错误，如果位置合法则进行返回OK，相应位置的值赋给e，e=\*(L.elem+i-1)。



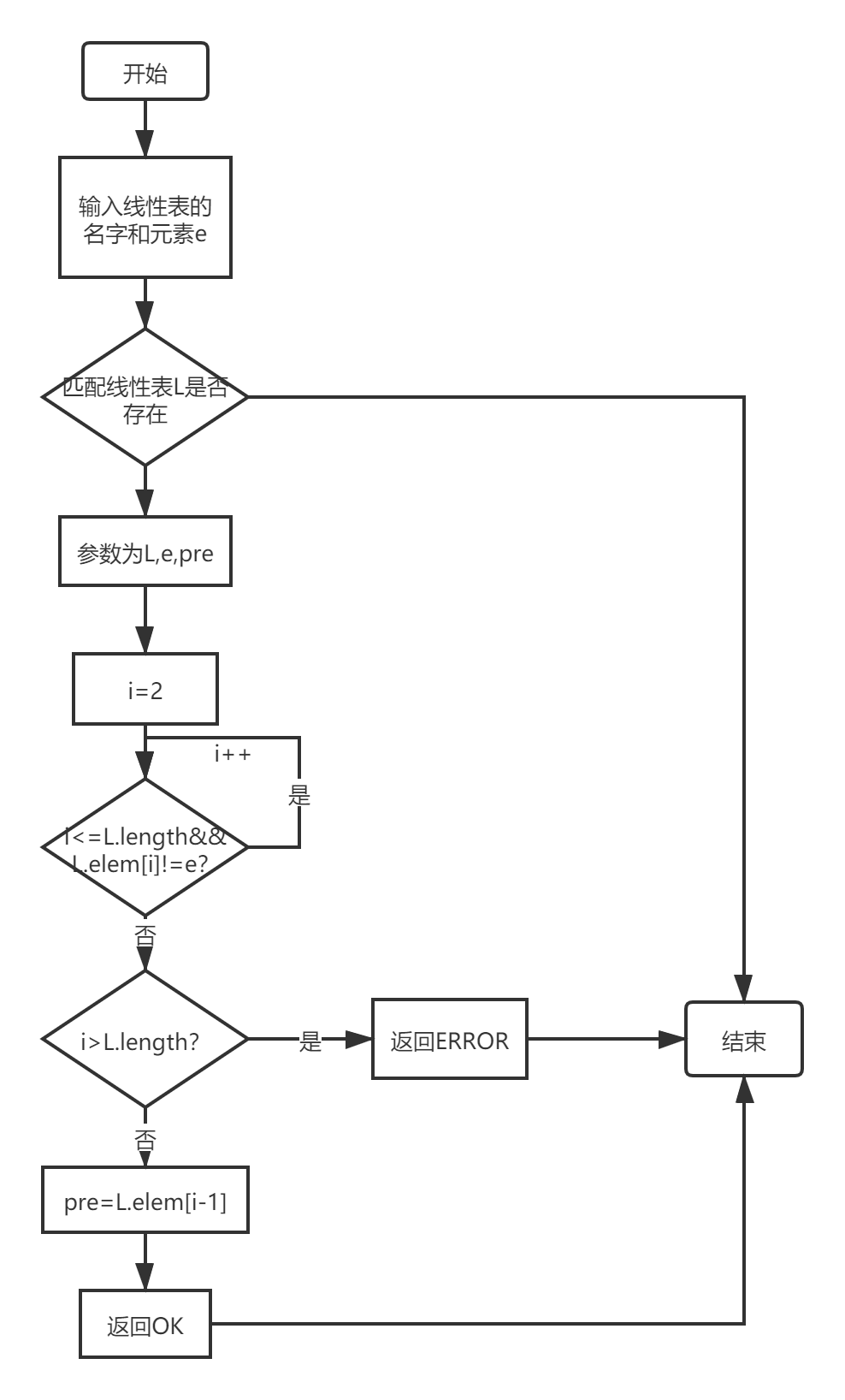
1. 查找元素

首先请求用户输入要进行查找元素操作的线性表的名字，判断此线性表是否存在，如果存在则进行查找元素e操作。定义一个初值为1标志元素位序的变量i，当i<=L.length&&L.elem[i-1]!=e时循环执行i++操作，此时如果i<=L.length，则返回i即是查找的元素的位置，否则返回0，标志没有查找到元素e。



1. 获取元素的前驱

首先请求用户输入要进行获取元素前驱的线性表的名字，判断此线性表是否存在，如果存在则执行获取元素前驱的操作，定义一个初值为2标志有前驱的元素位序的变量i，当i<=L.length&&L.elem[i]!=e时循环执行i++操作，此时如果i>L.length，返回错误信息表示没有相应的元素，如果合法，将元素的前驱赋给pre：pre=L.elem[i-1]，并返回OK。

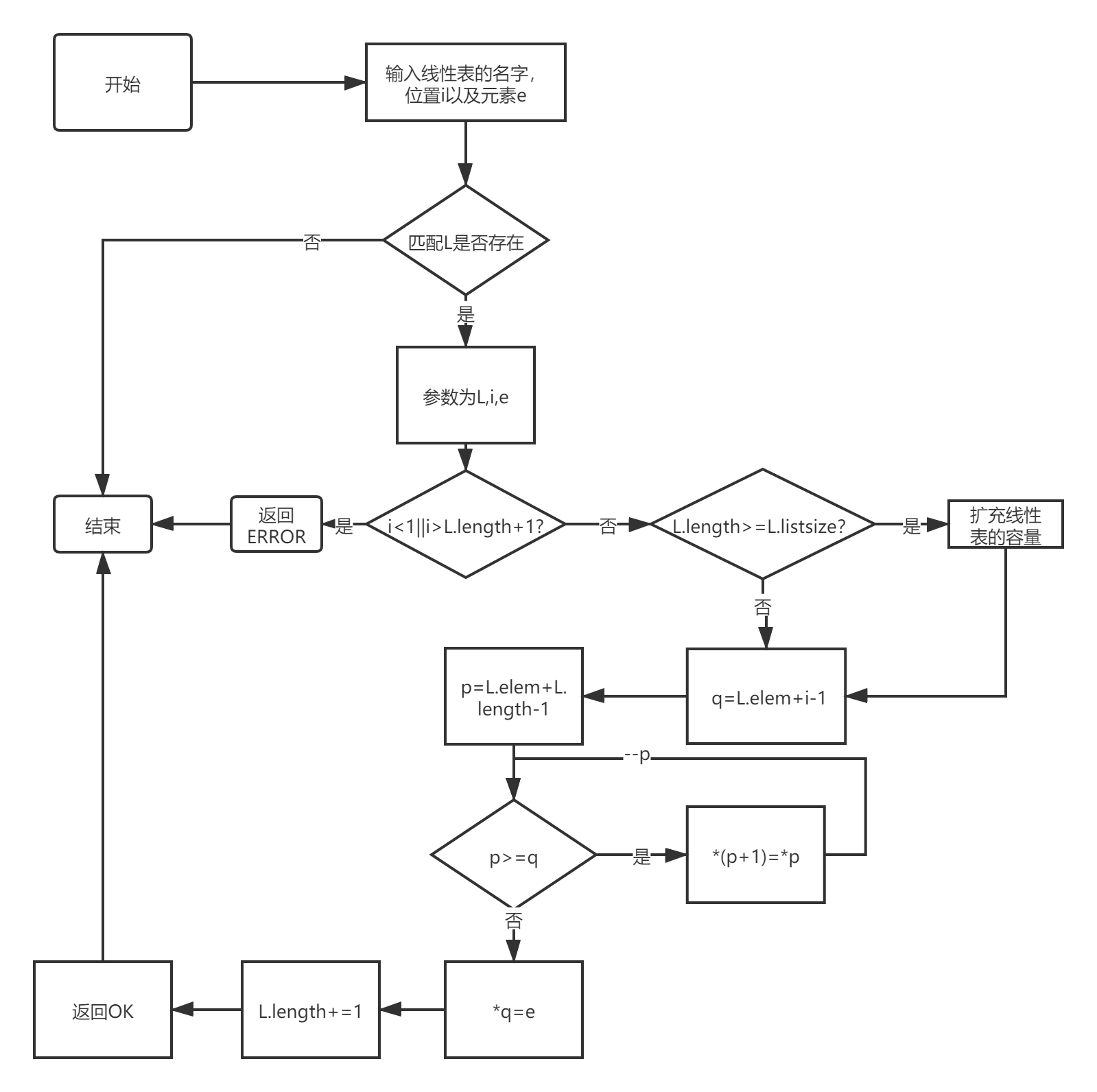


1. 获取元素的后继

首先请求用户输入要进行获取元素后继的线性表的名字，判断此线性表是否存在，如果存在则执行获取元素后继的操作，定义一个初值为2标志有前驱的元素位序的变量i，当i<=L.length&&L.elem[i]!=e时循环执行i++操作，此时如果i>L.length，返回错误信息表示没有相应的元素，如果合法，将元素的前驱赋给pre：pre=L.elem[i-1]，并返回OK。

1. 插入元素

首先请求用户输入要进行获取元素后继的线性表的名字，判断此线性表是否存在，如果存在则执行插入元素的操作，执行对插入元素的位置是否合法的判断操作，如果插入位置i<1或i>L.length+1，就返回ERROR的错误信息，之后进一步判断线性表的长度是否大于等于线性表的容量，则需要进行扩充线性表的容量。将插入位置之后的元素后移一位，将要插入的元素e插入到相关的位置中，且线性表的长度加1，返回OK。



1. 删除元素

首先请求用户输入要进行获取元素后继的线性表的名字，判断此线性表是否存在，如果存在则执行删除元素的操作，执行对删除元素的位置的判断，如果插入位置i<1或i>L.length，就返回ERROR的错误信息，之后取得被删除位置的地址p = L.elem + i – 1，并将要删除的元素赋给e，之后完成插入元素之后的元素向前移动一位，线性表的长度减1，返回OK。

1. 遍历线性表

首先请求用户输入要进行获取元素后继的线性表的名字，判断此线性表是否存在，如果存在则执行遍历线性表的操作，声明ElemType \*p，并给p赋值线性表数组的首元素地址：p=L.elem，然后循环执行访问元素的操作，遍历完成时候返回OK。

1. 保存线性表

请求用户输入要保存的线性表的名字，通过循环匹配查找判断此线性表是否存在，如果存在，再次请求用户输入要以文件形式进行保存的地址，打开文件，如果文件打开失败则返回ERROR的错误信息，如果打开成功，将线性表中的内容写入对应地址的文件中，并返回OK。

1. 加载线性表

请求用户输入要加载入线性表的名字，通过循环匹配查找判断此线性表是否存在，如果存在，则要先释放该线性表数据与的内存然后置空，接着在请求用户输入加载的文件的地址，打开文件，如果文件打开失败则返回ERROR的错误信息，如果打开成功，则将文件的内容写入进对应的线性表中，并返回OK。

1. 删除线性表

首先执行对应名字的线性表的循环查找匹配操作，如果匹配成功则执行删除操作，将删除位置之后的线性表向前移一位，并释放删除线性表的内存，完成删除操作之后跳出匹配查找的循环，如果完成查找匹配操作之后的次数等多线性表的长度表示没有查找到要删除的线性表返回ERROR的错误信息，否则标志查找成功，多线性表的长度减1，返回OK。

1. 查找线性表

执行对应名字的线性表的循环查找匹配操作，如果匹配成功，就返回线性表在多线性表中的位置，如果循环执行完毕后仍没有返回线性表的位置，表示没有找到相应的线性表

1. 遍历多线性表

设置初值i=0,变化范围为多线性表的长度List.length，步长为1递增，进行循环遍历单线性表，对于每个单链表首先打印单链表的名字，然后调用ListTraverse该函数遍历单链表储存的数据。

**1.4 系统测试**

系统环境：Windows10 专业版

系统版本号：20H2

测试用IDE：Clion

IDE版本号：2020.2.4

1. 测试1：创建一个线性表，并添加元素1，2，3

表1 测试创建一个线性表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例序号 | 操作名称 | 预计结果 | 实际结果 |
| 1 | 添加线性表a  元素为1，2，3 | 输出a 1 2 3 | a 1 2 3 |

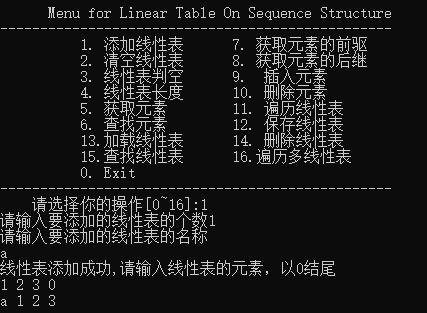


图1-4-1测试1结果图

1. 测试2：再创建一个线性表b，同样添加元素1，2，3

表2 测试创建另外一个线性表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 操作名称 | 预计结果 | 实际结果 |
| 1 | 添加线性表b元素为1，2，3 | 输出b 1 2 3 | b 1 2 3 |

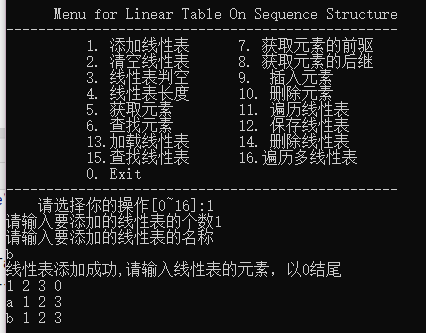


图1-4-2测试2结果图

1. 测试3：删除第二个线性表的第二个元素

表3 测试删除元素

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例序号 | 操作名称 | 预计结果 | 实际结果 |
| 1 | 删除线性表b的元素2 | 输出1 3 | 1 3 |

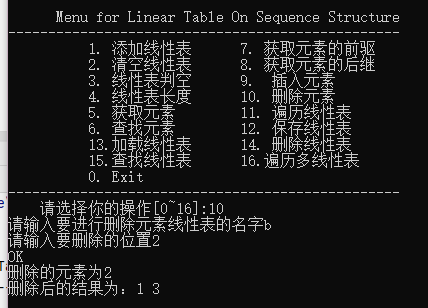


图1-4-3测试3结果图

1. 测试4：把第一个线性表以文件形式进行保存

表4 测试保存一个线性表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 操作名称 | 预计结果 | 实际结果 |
| 1 | 保存线性表a | 文件保存线性表a中的数据 | 见文件保存图 |

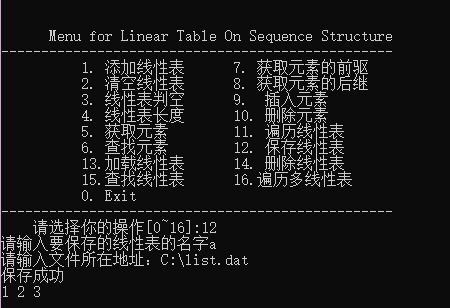


图1-4-4-1测试4操作演示图

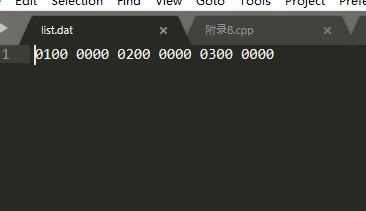


图1-4-4-2测试4文件保存结果图

1. 测试5：删除第一个线性表

表1 测试删除一个线性表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 操作名称 | 预计结果 | 实际结果 |
| 1 | 删除表a | 输出b 1 3 | 输出b 1 3 |

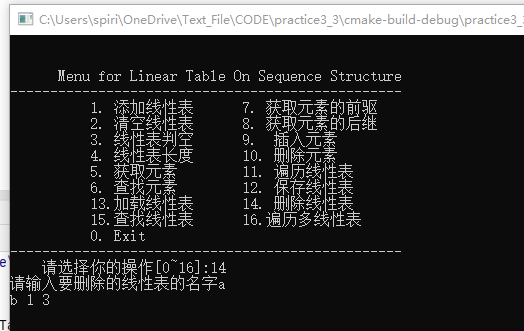


图1-4-5测试5结果图

1. 测试6：添加一个新的空线性表，并将读取刚刚保存的文件的内容载入此线性表中，并遍历多线性表查看加载的结果是否正确

表1 测试创建一个线性表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 操作名称 | 预计结果 | 实际结果 |
| 1 | 添加空线性表c  加载之间保存的文件 | 输出c 1 2 3 | 输出c 1 2 3 |

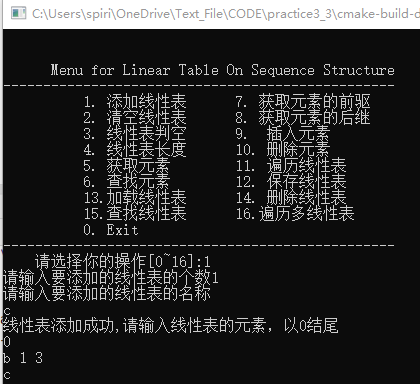


图1-4-6-1测试6添加空线性表结果图

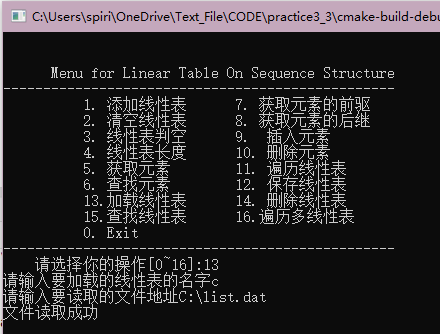


图1-4-6-2测试6加载线性表c结果图

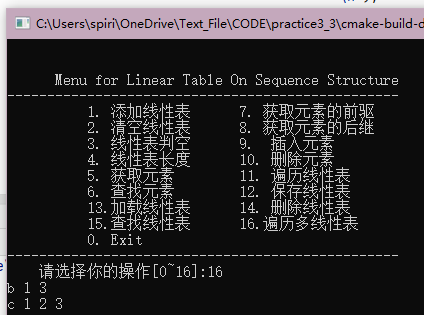


图1-4-6-3测试6遍历多线性表结果图

## 1.5 实验小结

本次实验的主要任务就是在线性表的顺序表示和实现下完成线性表的首相数据类型中的基本操作，包括线性表的初始化，销毁一个线性表，清空一个线性表，判断一个线性表是否为空，求一个线性表的长度，获取线性表中的某个元素，获取线性表中某一个元素的位置，获取线性表中某一个元素的前驱和后继，向线性表中插入一个元素，向线性表中删除一个元素，这些基本操作还可以相互组合实现一些更加复杂的操作，在本次实验中做重要的是做到多个线性表的管理，和线性表的文件保存和加载以及怎样如何设计出合适的管理方式，这就要求到了要设计出一种新的结构体,在这个结构体中，可以临时储存的单线性表，并对相应的线性表命名。通过这次实验的训练加深了我对线性表的概念基本运算的理解，较为熟练掌握线性表的逻辑结构与物理结构的关系，在物理结构采用顺序表的情况下，较为熟练的掌握线性表的基本元素运算的实现。

本次实验作为本次实验课的开头，对之后的实验起到了至关重要的作用，是一个开头，是之后实验的模板，因为不管数据结构如何改变，都是为了能够在满足需求的基础上完成对数据的管理，看了整篇数据结构实验之后，我发现每个实验的框架都有一定的联系，是可以相互对照参考并加以修改的，因此第一次实验虽然比较简单，但是它同样也不简单，好好完成它，可以为之后的实验减轻很多负担。

# 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，实现线性表的基本操作，对于各个操作的实现要求构造一个具有菜单的功能演示系统，其中要在主程序中完成对实现各个操作的函数完成调用，能够根据输入的指令实参等完成相应的准备活动并完成相应函数操作结果的展示，并能够在演示系统中给出适当的操作提示使得用户在使用操作系统的功能更加顺利。

对于操作中创建的线性表能够以文件的形式保存，同时也能够将文件中的数据读入线性表中完成文件操作，所以要求还要设计出文件保存和加载操作的合理模式，已完成上述文件操作。

该演示系统能够完成对多个线性表的管理，即在能够完成对历史创建的线性表进行临时储存，在后续的操作中选中历史线性表进行操作以完成对多线性表的管理。

要求在所有代码在编译完成相应的功能之后能有相对应的目标程序，此目标程序能够独立于IDE运行的EXE文件

### 2.1.1 线性表的基本操作定义

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,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,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()。

## 2.2 系统设计

结合单线性表和多线性表的操作之间的关系设计出演示系统的交互，首先创建一个包含多个线性表的结构体Lists，做为各个功能操作的主体，在这个结构体中，可以包含多个有以名字标志的线性表以便进行选中指定线性表进行相应的基本操作，此系统设计如图2-1-1所示：

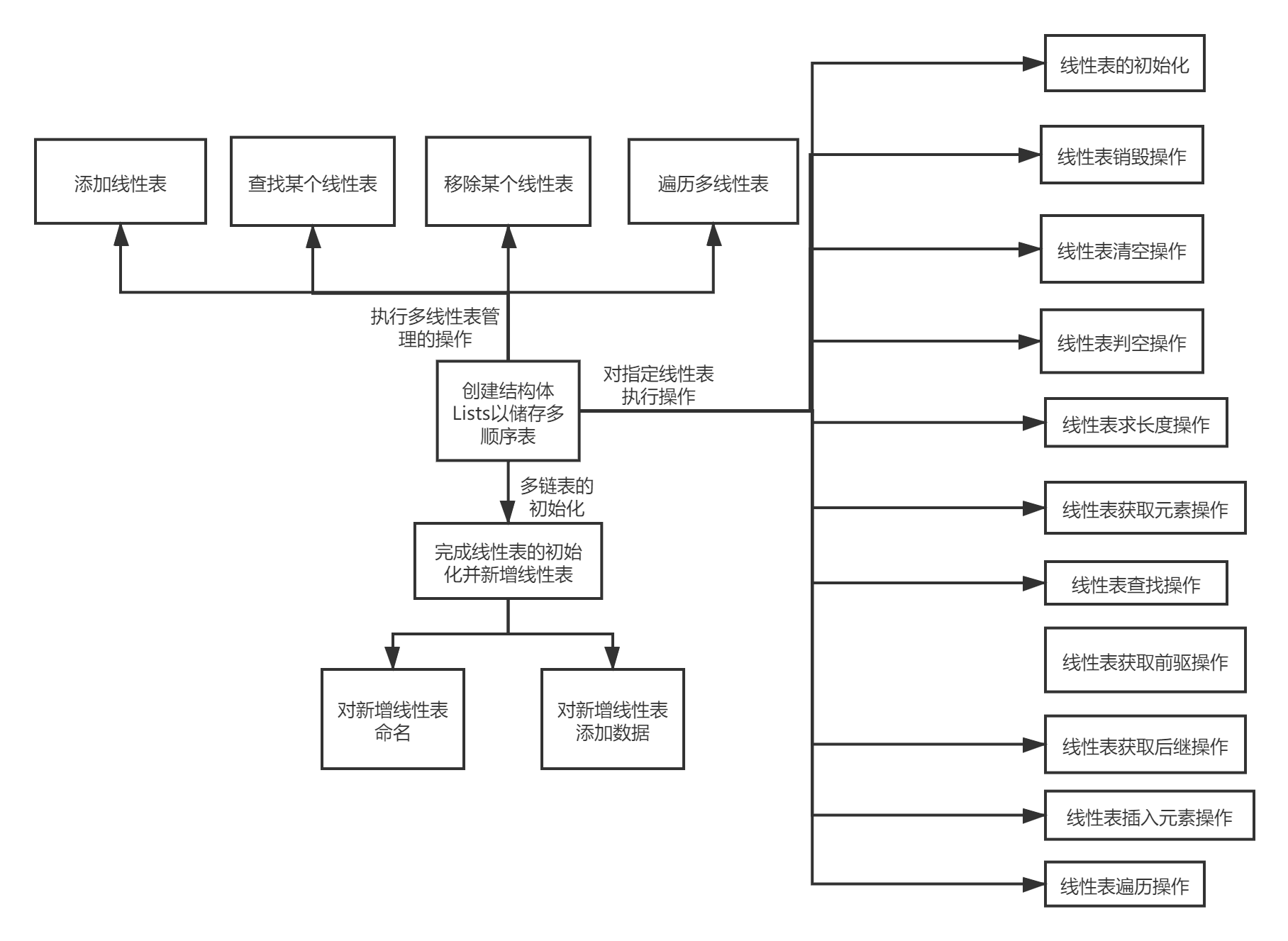


图2-1-1系统设计图

## 2.3 系统实现

### 2.3.1 一些在主函数前的准备工作：

1. 函数返回值宏设定：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

1. 线性表储存类型和相关函数返回值类型设定：

typedef int status;

typedef int ElemType;

1. 单线性表和多线性表存储结构的定义：

typedef struct LNode{ //线性表（链式结构）结点的定义

ElemType data;

struct LNode \*next;

}LNode,\*LinkList;

typedef struct{ //多线性表管理结构体的定义

struct {

char name[30];

LinkList L;

}elem [10];

int length;

int listsize;

}LISTS;

由于要实现构造出一个具有菜单功能的功能演示系统，在这其中要在主程序中完成函数的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。所以在主函数部分要打印出相应的菜单界面，并且使用system("cls");函数在执行完一次功能之后使

### 2.3.2 主函数部分

1. 演示系统的打印：

由于要实现构造出一个具有菜单功能的功能演示系统，在这其中要在主程序中完成函数的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。所以在主函数部分要打印出相应的菜单界面，并且使用system("cls");函数在执行完一次功能之后使界面清空，使界面显得干净整洁，方便进行下一次操作。

1. 演示系统的功能选择：

同时为了实现选择对应功能来进行相应的操作，采用switch-case语句完成选择特定的函数功能实现部分以完成相应的操作。

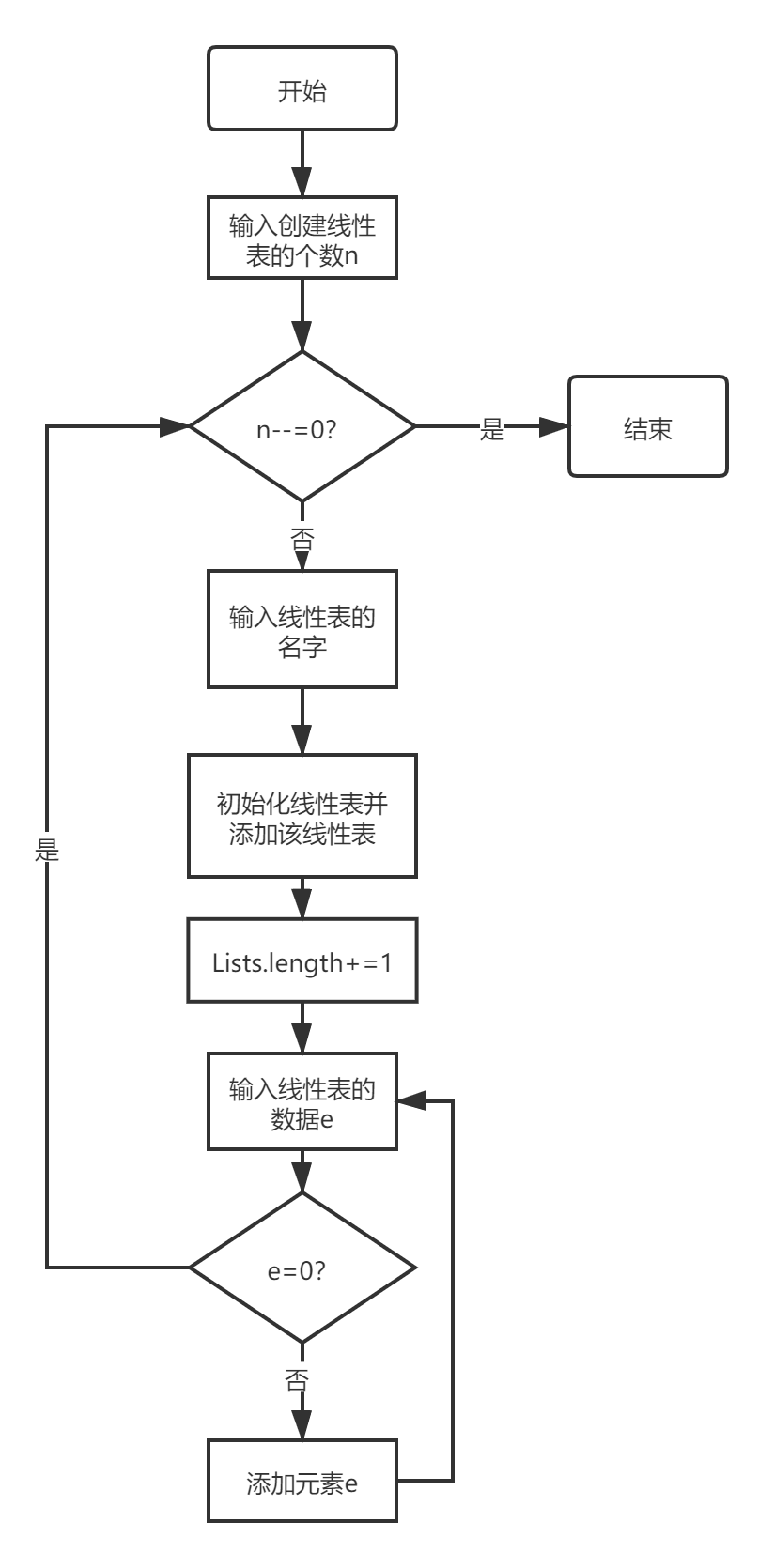
c）多线性表管理部分：

此外在具体功能的实现方面，由于要进行单线性表的操作，以及多线性表的管理，要在主函数中声明单链表LinkList L，多链表结构体 LISTS Lists和储存单顺序表名字的字符数组char name[30]，上述完成准备工作

### 2.3.3功能函数部分

1. 添加线性表：

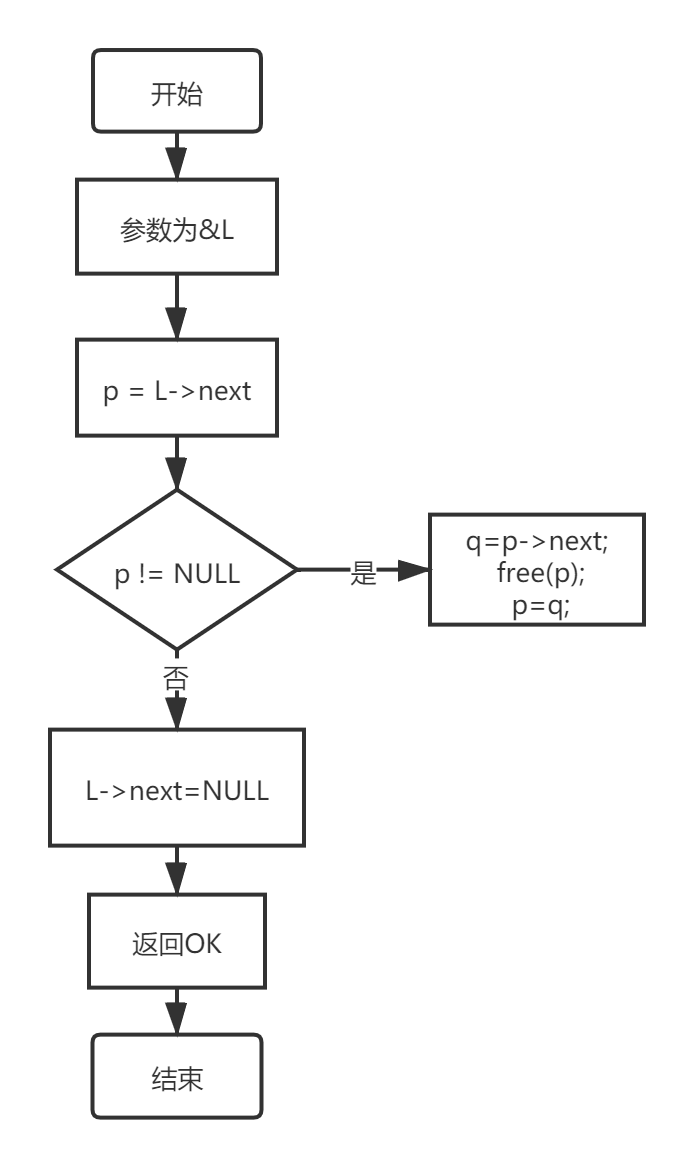
添加多线性表是初始化一个线性表的延伸，因为在此函数中使用到了单个线性表和多线性表的初始化操作，单线性表加入到多线性表的操作。首先用户输入要新增的线性表的个数，接着请求用户输入这些线性表的名字，借助AddList函数在多线性表结构中添加对应名字的线性表，随后请求用户输入要在该线性表中储存的数据，如果用户想要创造空表的话，不添加元素即可。在该操作中，对线性表实现初始化的一系列操作后，该线性表即自动被添加到多线性表管理结构中。



1. 清空线性表

首先请求用户输入要清空的线性表的名字，判断要清空的线性表是否存在，如果线性表存在则执行清空操作：

* 声明LinkList p = L->next, q;
* 循环判断p != NULL
* 进行节点内存释放操作：q=p->next; free(p); p=q; 回到上一步判断，判断是否还需要节点释放操作。
* 线性表置空：L->next=NULL;
* 函数返回OK



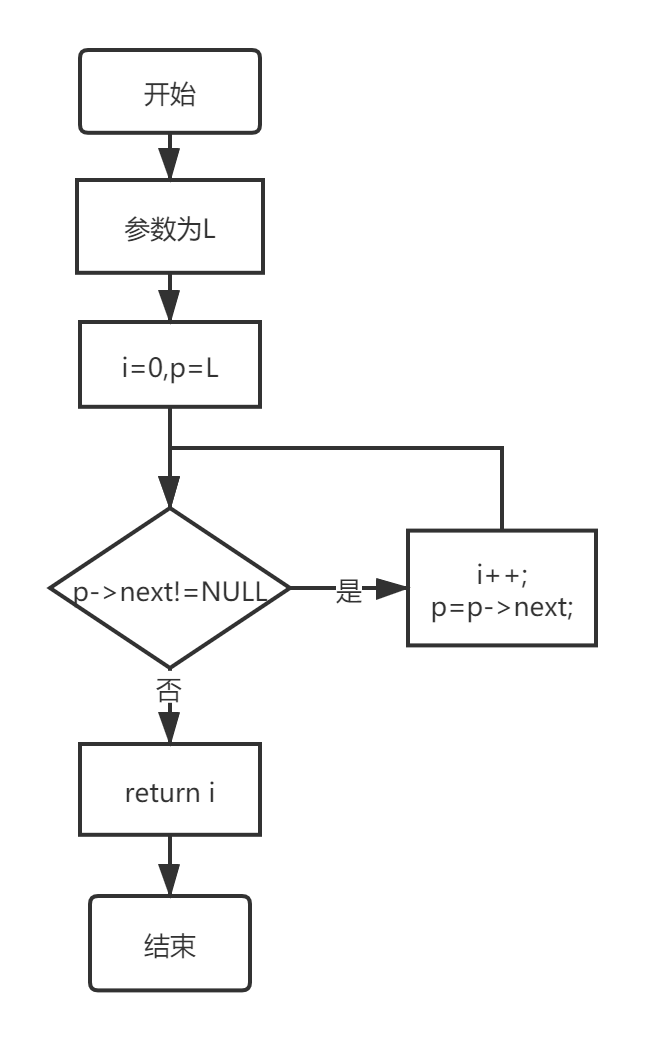
1. 线性表判空

首先请求用户输入要进行判空操作的线性表的名字，判断要判空的线性表是否为存在，如果线性表存在则执行判空操作，如果线性表的头节点的指针域为空，则该线性表为空，返回true，否则该线性表不是空线性表，返回false。

1. 线性表的长度

首先请求用户输入要进行获取长度的线性表的名字，判断此线性表是否存在，如果存在则执行返回长度的操作：

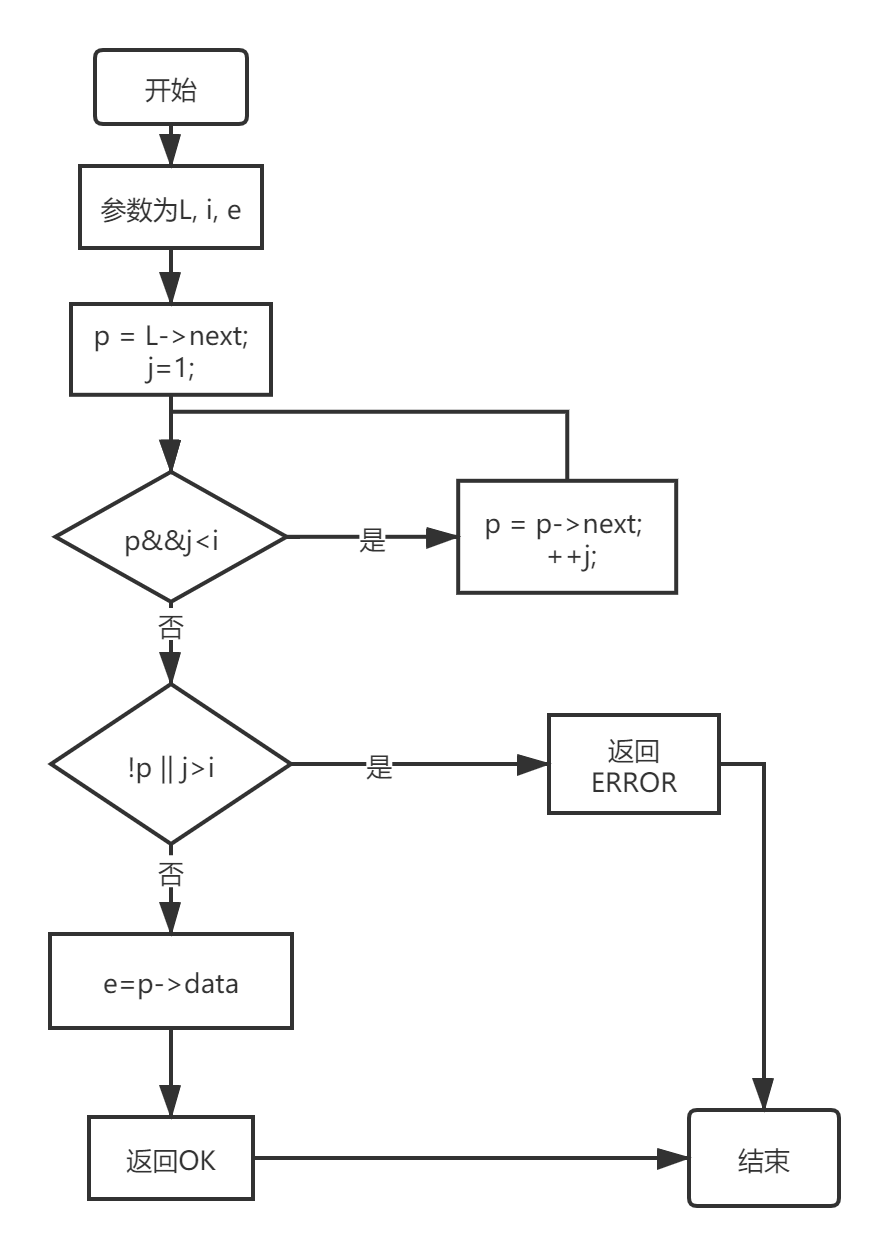
* 声明int i=0，用于计数
* 声明LinkList p=L;
* 循环判断p->next!=NULL
* 通过判断执行i++; p=p->next; 回到上一步再执行判断
* return i;



1. 获取元素

首先请求用户输入要进行获取元素操作的线性表的名字，判断此线性表是否存在，如果存在则进行如下操作：

* 声明LinkList p = L->next; int j =1;
* 循环判断p&&j<i
* 判断通过执行：p = p->next; ++j; 回到上步继续判断
* 判断!p || j>i是否成立，如果成立返回ERROR
* e=p->data;
* return OK



1. 查找元素

首先请求用户输入要进行查找元素操作的线性表的名字，判断此线性表是否存在，如果存在则进行查找元素e操作。

* 声明LinkList p=L->next, int i=0
* 循环判断p，
* 判断通过执行下列操作：

i++;

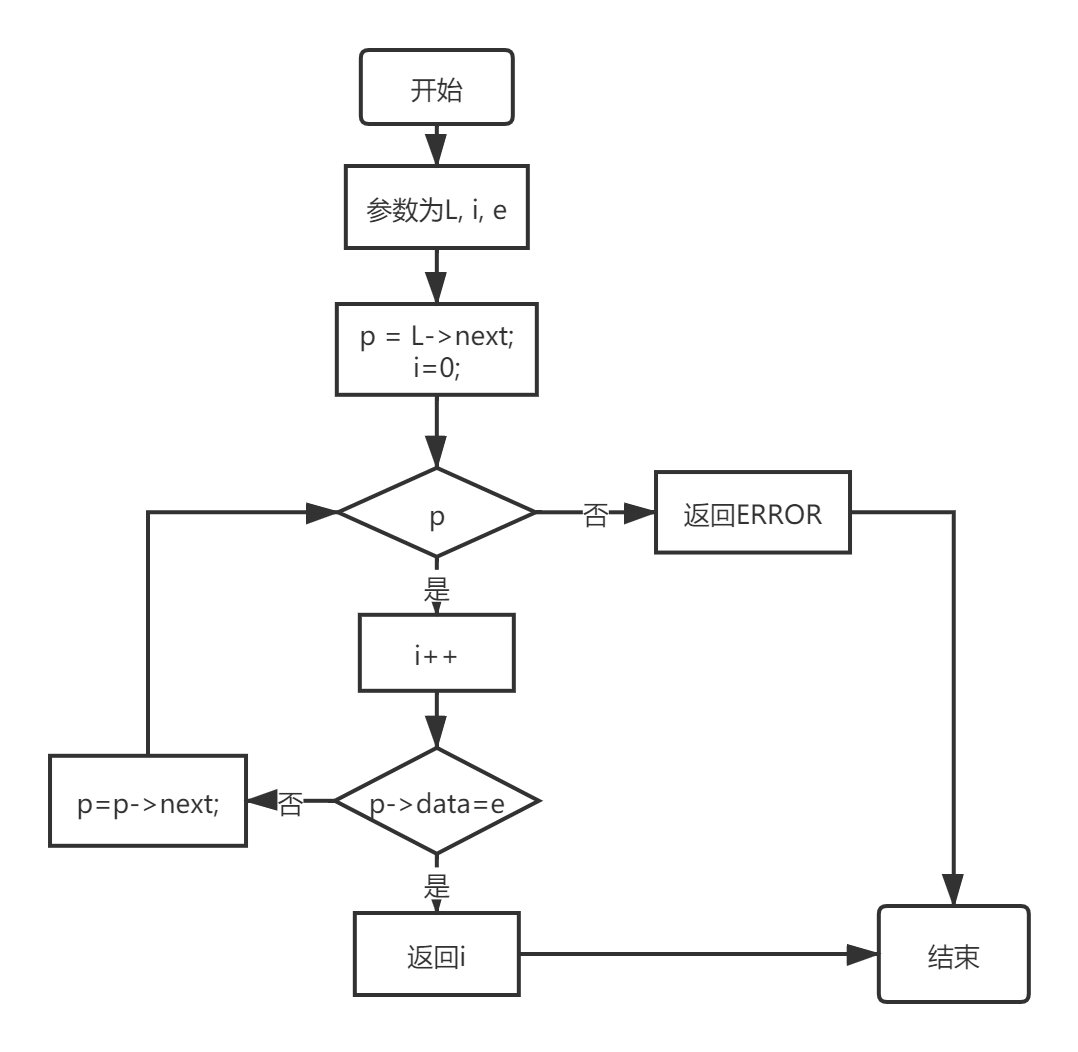
if(p->data==e) /\* 找到这样的数据元素 \*/

return i;

p=p->next;

回到上一步继续判断。

* return ERROR



1. 获取元素的前驱

首先请求用户输入要进行获取元素前驱的线性表的名字，判断此线性表是否存在，如果存在则执行获取元素前驱的操作：

* 判断L->next==NULL || L->next->data==e 如果成立，return ERROR;
* 声明LinkList p=L->next;
* 循环判断p->next
* 上一步判断通过执行：

if(p->next->data==e)

{

pre = p->data;

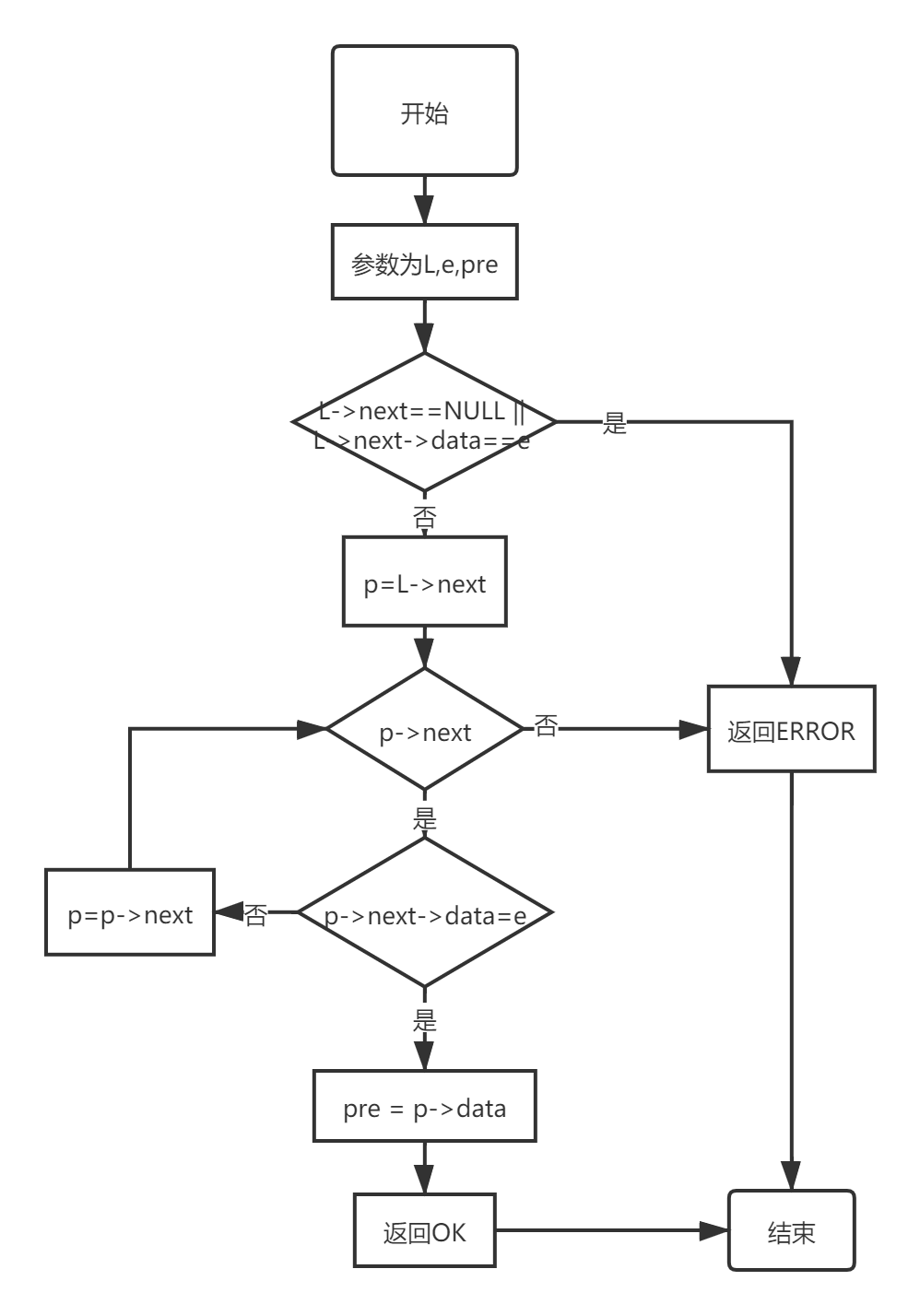
return OK;

}

p=p->next;

返回上一步再判断

* return ERROR



1. 获取元素的后继

首先请求用户输入要进行获取元素后继的线性表的名字，判断此线性表是否存在，如果存在则执行获取元素后继的操作：

* 判断L->next==NULL如果成立，return ERROR;
* 声明LinkList p=L->next;
* 循环判断p->next
* 上一步判断通过执行：

if(p->data==e)

{

next = p->next->data;

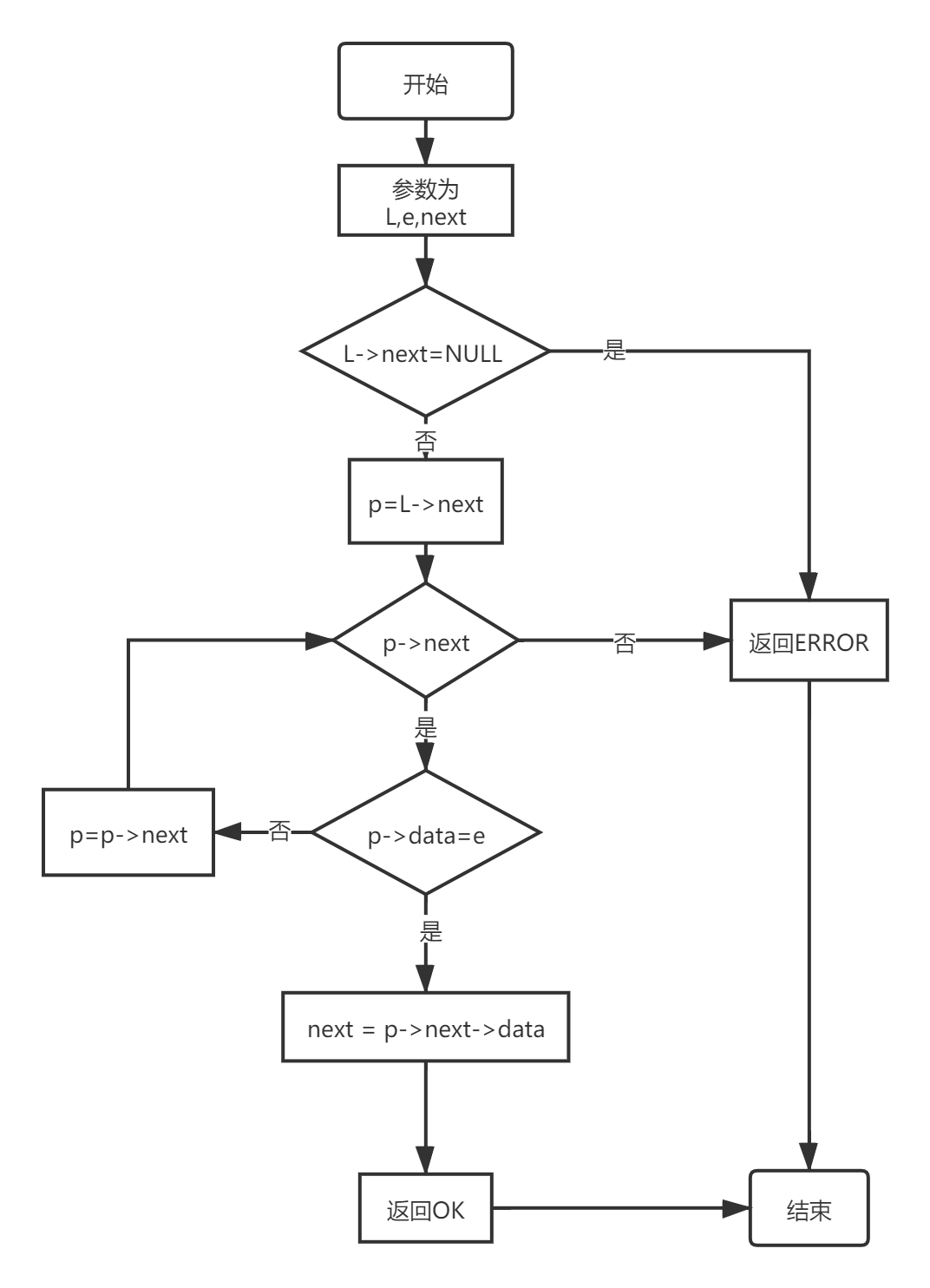
return OK;

}

p=p->next;

返回上一步再判断

* return ERROR



1. 插入元素

首先请求用户输入要进行获取元素后继的线性表的名字，判断此线性表是否存在，如果存在则执行插入元素的操作：

* 判断i<=0，如果成立返回ERROR
* 声明LinkList p=L; int j=0;
* 循环判断p&&j<i-1
* 判断通过执行：

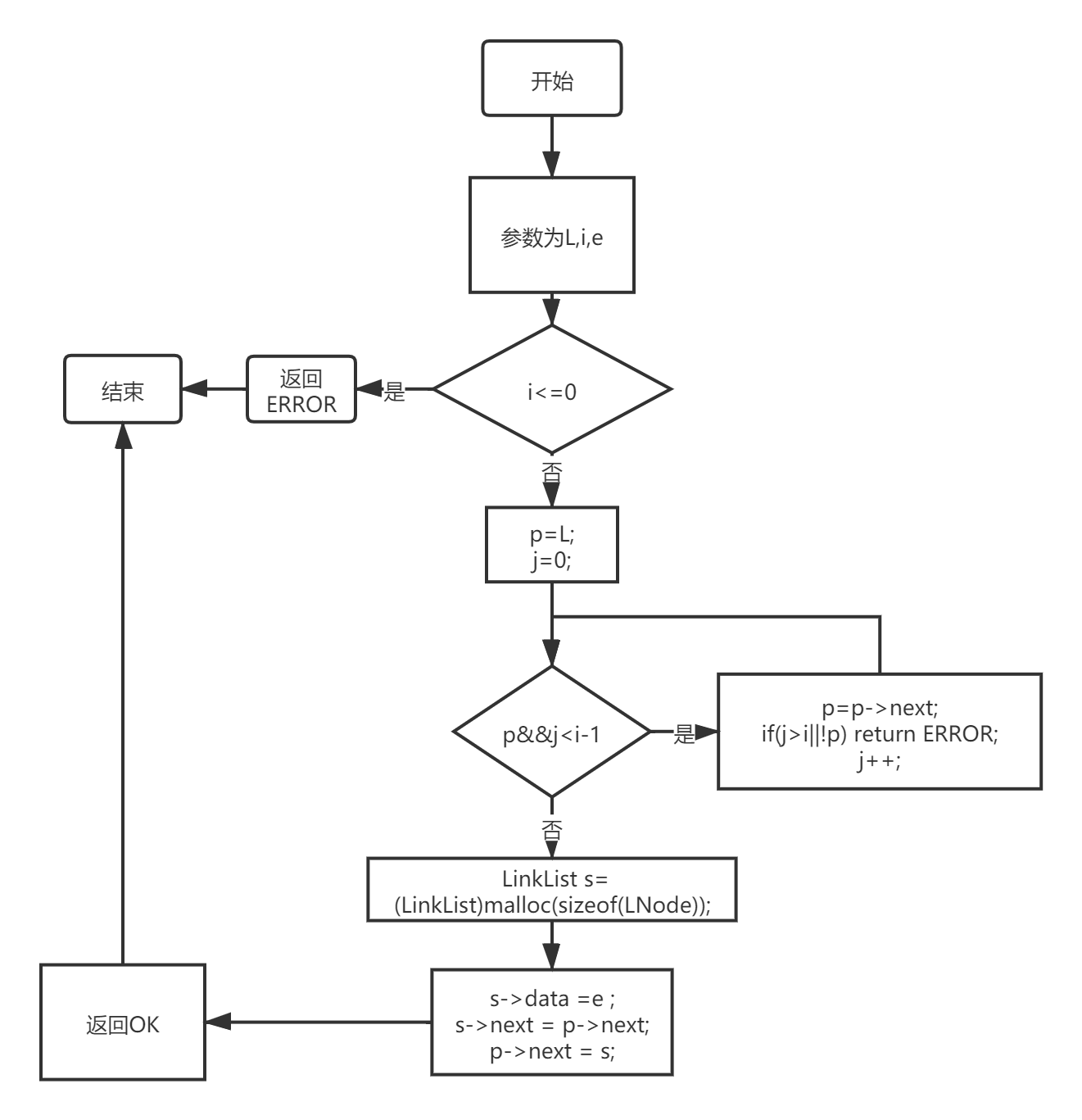
p=p->next;

if(j>i||!p) return ERROR;

j++;

回到上一步再进行判断

* 声明LinkList s=(LinkList)malloc(sizeof(LNode));
* 执行操作 s->data =e ; s->next = p->next; p->next = s;
* return OK;



1. 删除元素

首先请求用户输入要进行获取元素后继的线性表的名字，判断此线性表是否存在，如果存在则执行删除元素的操作：

* 声明LinkList p=L->next,q; int l=0, j=1;
* 求线性表的长度
* 判断线性表是否为空：p==NULL，如果为空返回ERROR
* 判断i值是否合法：i<1||i>l，如果不合法返回ERROR
* 循环找到要删除节点的前驱
* 获取删除结点的元素，释放删除结点的内存：

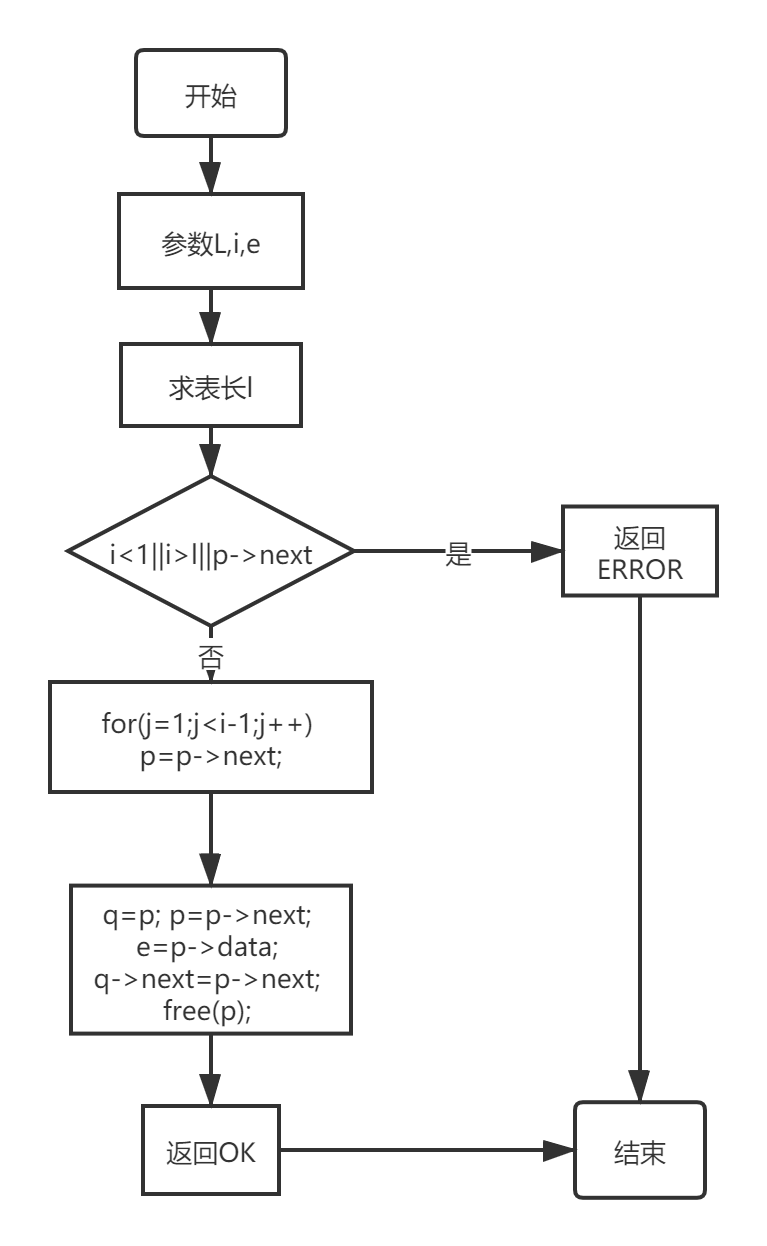
q=p; p=p->next;

e=p->data;

q->next=p->next;

free(p);

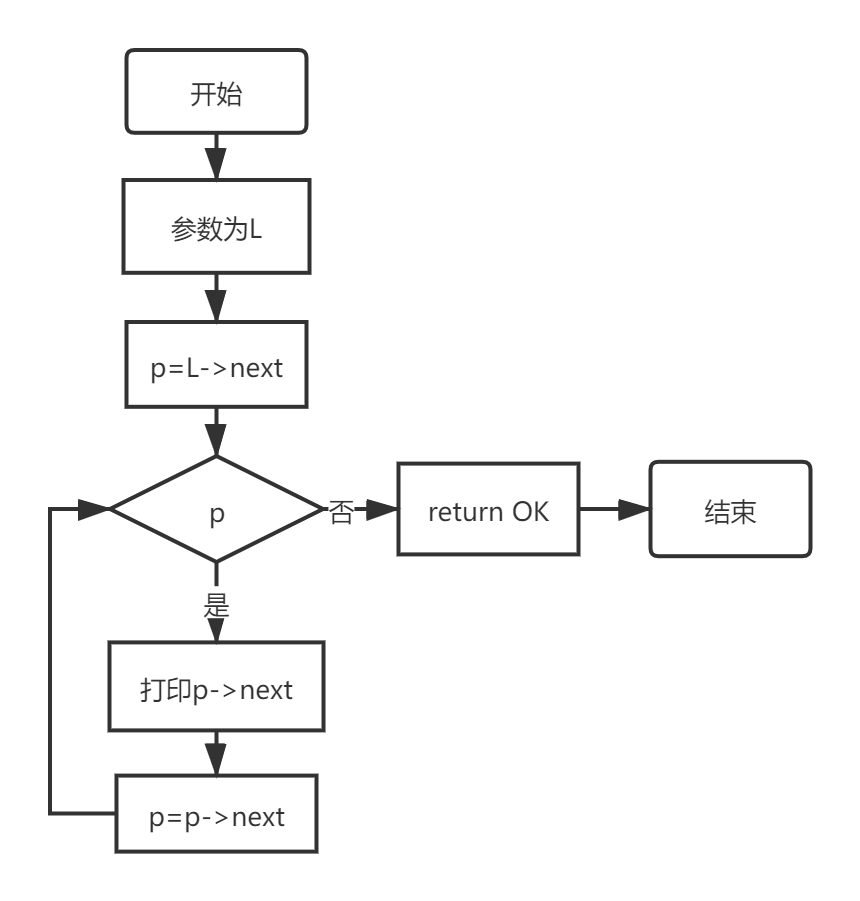
* return OK



1. 遍历线性表

首先请求用户输入要进行获取元素后继的线性表的名字，判断此线性表是否存在，如果存在则执行遍历线性表的操作：

* 声明LinkList p=L->next;
* 循环判断p
* 判断通过执行操作：  
  printf("%d ", p->data); p=p->next;
* 回到上一步再判断
* return OK



1. 保存线性表

请求用户输入要保存的线性表的名字，通过循环匹配查找判断此线性表是否存在，如果存在，再次请求用户输入要以文件形式进行保存的地址，打开文件，如果文件打开失败则返回ERROR的错误信息，如果打开成功，将线性表中的内容写入对应地址的文件中，并返回OK。

1. 加载线性表

请求用户输入要加载入线性表的名字，通过循环匹配查找判断此线性表是否存在，如果存在，则要先释放该线性表数据与的内存然后置空，接着在请求用户输入加载的文件的地址，打开文件，如果文件打开失败则返回ERROR的错误信息，如果打开成功，则将文件的内容写入进对应的线性表中，并返回OK。

1. 删除线性表

首先执行对应名字的线性表的循环查找匹配操作，如果匹配成功则执行删除操作，将删除位置之后的线性表向前移一位，并释放删除线性表的内存，完成删除操作之后跳出匹配查找的循环，如果完成查找匹配操作之后的次数等多线性表的长度表示没有查找到要删除的线性表返回ERROR的错误信息，否则标志查找成功，多线性表的长度减1，返回OK。

1. 查找线性表

执行对应名字的线性表的循环查找匹配操作，如果匹配成功，就返回线性表在多线性表中的位置，如果循环执行完毕后仍没有返回线性表的位置，表示没有找到相应的线性表

1. 遍历多线性表

设置初值i=0,变化范围为多线性表的长度List.length，步长为1递增，进行循环遍历单线性表，对于每个单链表首先打印单链表的名字，然后调用ListTraverse该函数遍历单链表储存的数据。

## 2.4 系统测试

系统环境：Windows10 专业版

系统版本号：20H2

测试用IDE：Clion

IDE版本号：2020.2.4

测试1：测试创建空线性表：

表1：测试创建空线性表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 创建名为a  的空线性表 | a |  |

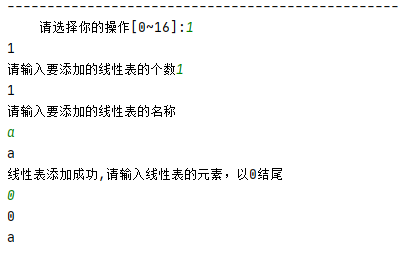


图2-4-1-1 测试1运行图

测试2：测试清空线性表：

表2：测试清空线性表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 清空空线性表 | 该线性表已经清空 |  |
| 2 | 清空内容为1，2，3的线性表 | 已清空该线性表 |  |

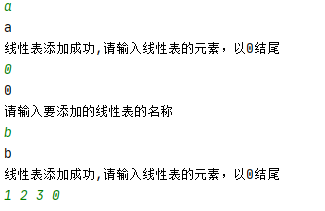


图2-4-2-1 测试2用例创建

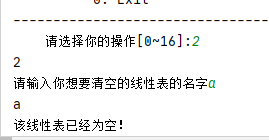


图2-4-2-2 测试2用例1运行结果图

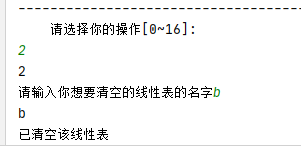


图2-4-2-3 测试2用例2运行结果图

测试3：测试判空线性表：

表3：测试判空线性表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 判断空线性表 | 该线性表为空 |  |
| 2 | 判断数据为1 2 3的线性表 | 该线性表不为空 |  |

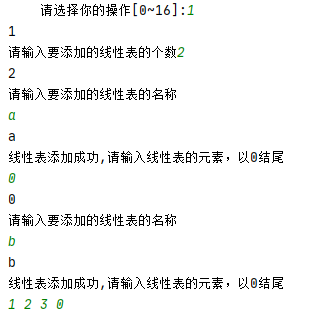


图2-4-3-1 测试3用例创建

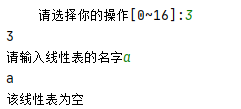


图2-4-3-2 测试3用例1运行结果图

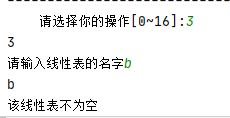


图3 测试2-4-3-3例2运行结果图

测试4：测试求线性表长度：

表4 ：测试线性表长度

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 求空表的长度 | 0 |  |
| 2 | 求数据为1 2 3的线性表的长度 | 3 |  |

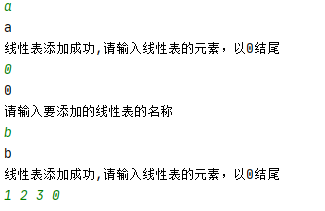


图2-4-4-1 测试4用例创建

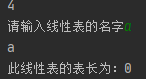


图2-4-4-2 测试4用例1运行结果图

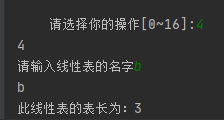


图2-4-4-3 测试4用例2运行结果图

测试5：测试获取元素：

表5：测试获取元素（已经创建线性表1 2 3 4）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 获取位置0  的元素 | ERROR |  |
| 2 | 获取位置3  的元素 | 3 |  |
| 3 | 获取位置6  的元素 | ERROR |  |

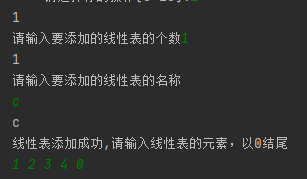


图2-4-5-1 测试5用例创建

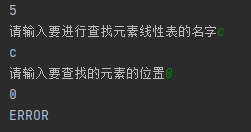


图2-4-5-2 测试5用例1运行结果图

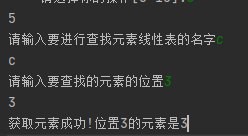


图2-4-5-3 测试5用例2运行结果图

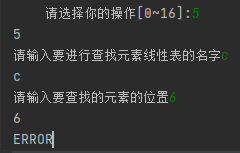


图2-4-5-4测试5例3运行结果图

测试6：测试查找元素：

表6：测试查找元素（已经创建线性表1 2 3 4）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 寻找元素8 | ERROR |  |
| 2 | 寻找元素3 | 3 |  |

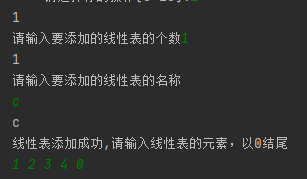


图2-4-6-1 测试6用例创建

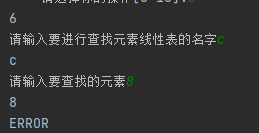


图2-4-6-2 测试6用例1运行结果图

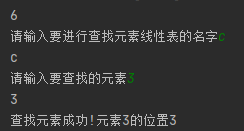


图2-4-6-3 测试6用例3运行结果图

测试7：测试获取元素的前驱：

表7：测试获取元素的前驱

（a为空表 c数据1 2 3 4的表）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 求空表a一个元素的前驱 | ERROR |  |
| 2 | 求所给表b元素1的前驱 | ERROR |  |
| 3 | 求给表b元素3前驱 | 2 |  |
| 4 | 求所给表b元素8前驱 | ERROR |  |

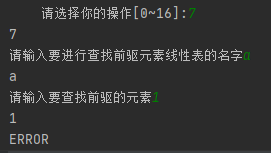


图2-4-7-1 测试7例1运行结果图

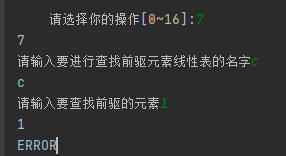


图2-4-7-2 测试7例3运行结果图

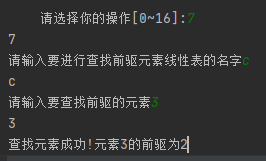


图2-4-7-3 测试7例3运行结果图

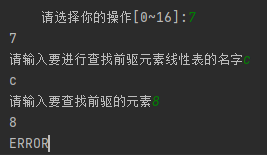


图2-4-7-4 测试7例4运行结果图

测试8：测试获取元素的后继：

表7：测试获取元素的前驱

（a为空表 c数据1 2 3 4的表）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 求空表a一个元素的后继 | ERROR |  |
| 2 | 求所给表b元素4的后继 | ERROR |  |
| 3 | 求给表b元素3的后继 | 4 |  |
| 4 | 求所给表b元素8的后继 | ERROR |  |

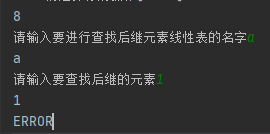


图2-4-8-1 测试8例1运行结果图

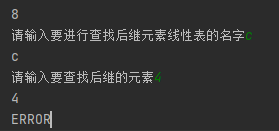


图2-4-8-2 测试8例2运行结果图

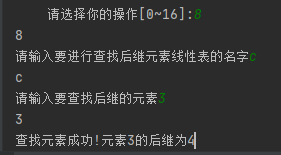


图2-4-8-3 测试8例3运行结果图

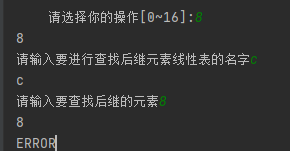


图2-4-8-4 测试8例4运行结果图

测试9：测试插入元素：

表9 测试插入元素

（a为空表 c数据1 2 3 4的表）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 空表a插入元素1 | 1 |  |
| 2 | 表c在位置0  插入元素1 | ERROR |  |
| 3 | 表c在位置2  插入元素1 |  |  |
| 4 | 表c在位置8  插入元素1 | ERROR |  |

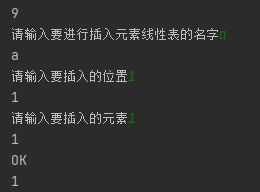


图2-4-9-1 测试9例1运行结果图

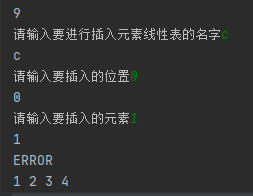


图2-4-9-2 测试9例2运行结果图

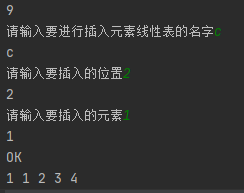


图2-4-9-3 测试9例3运行结果图

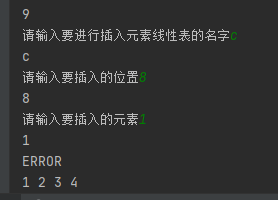


图2-4-9-4 测试9例4运行结果图

测试10：测试删除元素：

表10 测试删除元素

（a为空表 c数据1 2 3 4的表）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 删除表a第1个元素 | ERROR |  |
| 2 | 删除表b第0个元素 | ERROR |  |
| 3 | 删除表b第2个元素 | 1 3 4 |  |
| 4 | 删除表b第8个元素 | ERROR |  |

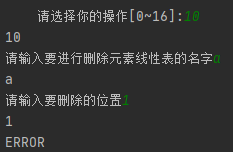


图2-4-10-1 测试10例1运行结果图

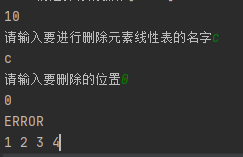


图2-4-10-2 测试10例2运行结果图

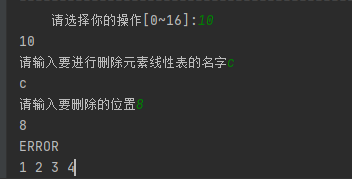


图2-4-10-3 测试10例3运行结果图

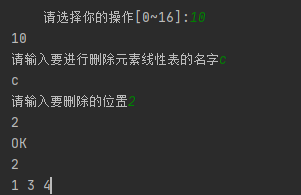


图2-4-10-4 测试10例4运行结果图

测试11：测试遍历线性表：

表11 测试遍历线性表

（a为空表 c数据1 2 3 4的表）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 遍历表a | 空线性表 |  |
| 2 | 遍历表c | 1 2 3 4 |  |

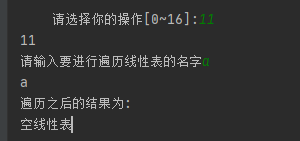


图2-4-11-1 测试11例1运行结果图

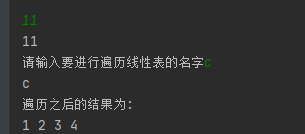


图2-4-11-2 测试11例2运行结果图

测试12：测试保存线性表：

表12 测试保存线性表

（c数据1 2 3 4的表）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 将表c的数据保存至文件list.dat中 | 保存成功 |  |

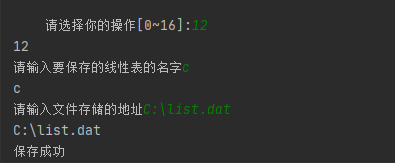


图2-4-12-1 测试12例1运行结果图

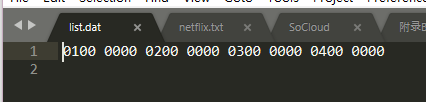


图2-4-12-2 测试12文件保存结果图

测试13：测试加载线性表：

表12 测试保存线性表

（文件list1.dat储存的内容为1 2 3 4 5，a为空表）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 文件list1.dat的数据加载进表a中 | 读取成功 |  |

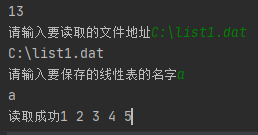


图2-4-13-1 测试13例1运行结果图

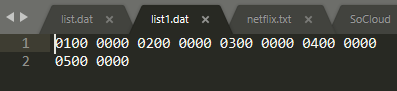


图2-4-13-2 测试12要读取文件数据图

测试14：测试删除线性表：

表14 测试删除元素

（a为空表 b数据1 2 3 的表c数据1 2 3 4的表）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 删除表e | 删除失败 |  |
| 2 | 删除表b | 删除成功 |  |

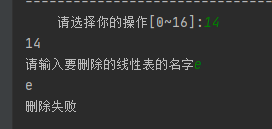


图2-4-14-1 测试13例1运行结果图

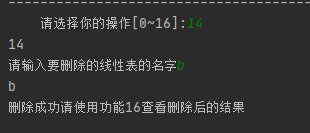


图2-4-14-2 测试13例2运行结果图

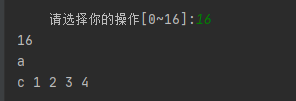


图2-4-14-3 测试13例2删除操作完成之后遍历结果图

测试15：测试查找线性表：

表6：测试查找元素

（a为空表 b数据1 2 3 的表c数据1 2 3 4的表）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 寻找表e | 查找失败 |  |
| 2 | 寻找表c | 查找成功 |  |

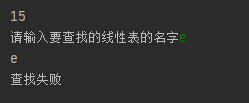


图2-4-15-1 测试15例1运行结果图

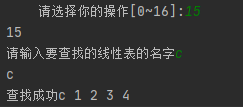


图2-4-15-2 测试15例2运行结果图

测试16：测试遍历多线性表：

表14 测试删除元素

（a为空表 b数据1 2 3 的表c数据1 2 3 4的表）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 遍历表 | a  b 1 2 3  c 1 2 3 4 |  |

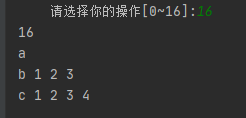


图2-4-16-1 测试16例1的运行结果图

## 2.5 实验小结

本次实验的主要任务就是在线性表的链式存储表示和实现下完成线性表的数据类型中的基本操作，包括线性表的初始化，销毁一个线性表，清空一个线性表，判断一个线性表是否为空，求一个线性表的长度，获取线性表中的某个元素，获取线性表中某一个元素的位置，获取线性表中某一个元素的前驱和后继，向线性表中插入一个元素，向线性表中删除一个元素，这些基本操作还可以相互组合实现一些更加复杂的操作，在本次实验中做重要的是做到多个线性表的管理，和线性表的文件保存和加载以及怎样如何设计出合适的管理方式，这就要求到了要设计出一种新的结构体,在这个结构体中，可以临时储存的单线性表，并对相应的线性表命名。与实验一不同的是通过这次实验的训练进一步加深了我对线性表的概念基本运算的理解，较为熟练的使用链式存储结构表示线性表，较为熟练的掌握线性表的基本元素运算的实现。

同时在完成实验的过程中，我也意识到自己也有很多不足，有很多疏忽之处造成程序有很多漏洞，在助教指出后改正，所以在以后的实验中要仔细思考，并尽可能的保证算法的健壮性。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

采用二叉链表作为二叉树的物理结构，二叉树的基本操作运算。其中ElemType为数据元素的类型名，具体含义可自行定义，但要求二叉树结点类型为结构型，至少包含二个部分，一个是能唯一标识一个结点的关键字，另一个是其它部分。

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。

演示系统可选择实现二叉树的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存二叉树数据逻辑结构(D,{R})的完整信息；②需要设计二叉树文件保存和加载操作合理模式。

演示系统可选择实现多个二叉树管理。可采用线性表的方式管理多个二叉树，线性表中的每个数据元素为一个二叉树的基本属性，至少应包含有二叉树的名称。基于顺序表实现的二叉树管理。

演示系统的源程序应按照代码规范增加注释和排版，目标程序务必是可以独立于IDE运行的EXE文件。

**3.1.1 二叉树的基本操作定义**

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。

## 3.2 系统设计

结合单二叉树和多二叉树的操作之间的关系设计出演示系统的交互，首先创建一个包含多个线性表的结构体wood，作为各个功能操作的主体，在这个结构体中，可以包含多个有以名字标志的二叉树以便进行选中指定二叉树进行相应的基本操作，此多树结构设计如图3-1-1所示：

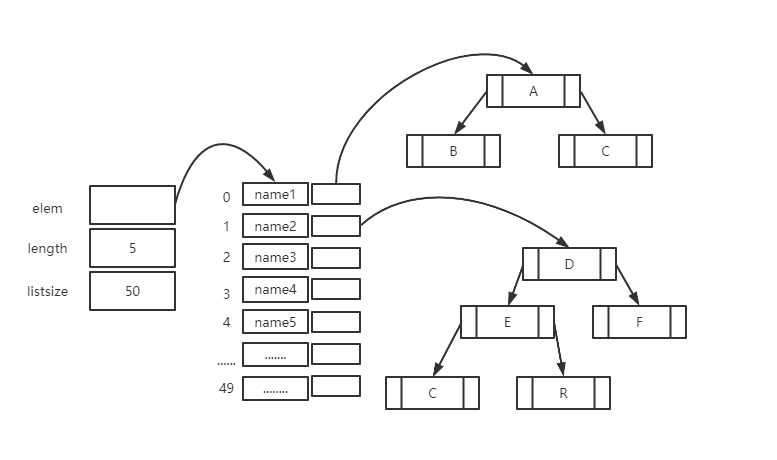


图3-1-1多二叉树管理的物理结构示意图

系统设计图如3-1-2所示：

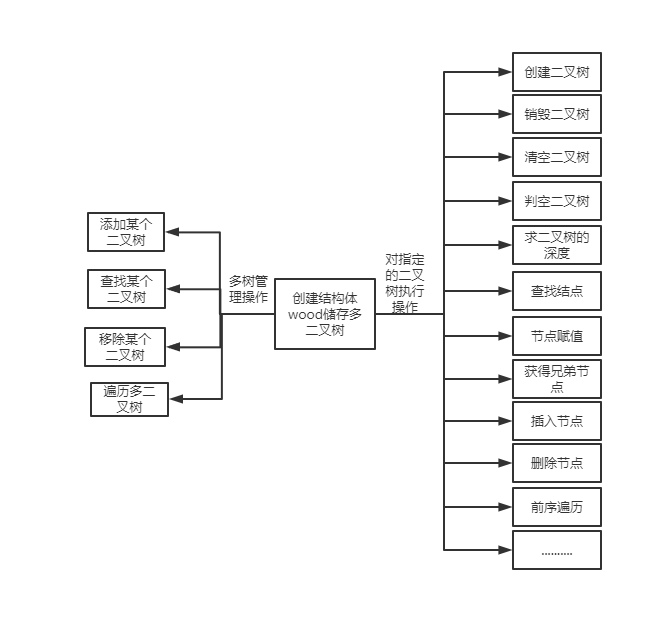


图3-1-2系统设计图

## 3.3 系统实现

**3.3.1一些在主函数前的准备**

1. 函数返回值宏设定：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

1. 二叉树储存类型和相关函数返回值类型设定：

typedef int status;

typedef int KeyType;

1. 二叉树和多二叉树相关存储结构的定义：

typedef struct {

KeyType key; //整型

char others[20]; //字符型

} TElemType; //二叉树结点类型定义

typedef struct BiTNode { //二叉链表的定义

TElemType data;

struct BiTNode \*lchild, \*rchild;

} BiTNode, \*BiTree;

typedef struct { //多二叉树存储结构定义

struct {

char name[30];

BiTree T;

} elem[100];

int length;

int listsize;

} wood;

typedef struct { //节点在完全二叉树位置及节点的数据

int pos;

TElemType data;

} DEF;

**3.3.2主函数部分**

1. 演示系统的打印：

由于要实现构造出一个具有菜单功能的功能演示系统，在这其中要在主程序中完成函数的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。所以在主函数部分要打印出相应的菜单界面，并且使用system("cls");函数在执行完一次功能之后使界面清空，使界面显得干净整洁，方便进行下一次操作。

1. 演示系统的功能选择：

同时为了实现选择对应功能来进行相应的操作，采用switch-case语句完成选择特定的函数功能实现部分以完成相应的操作。

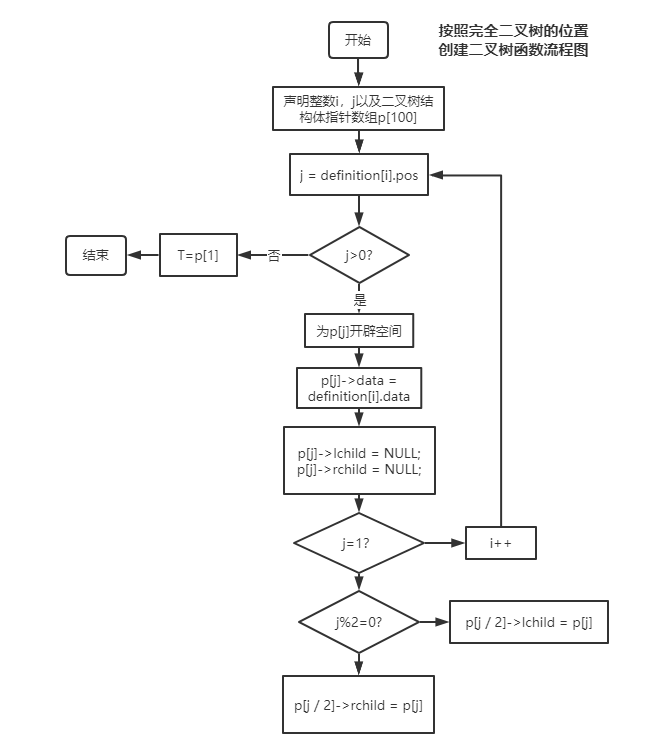
1. 多树管理部分：

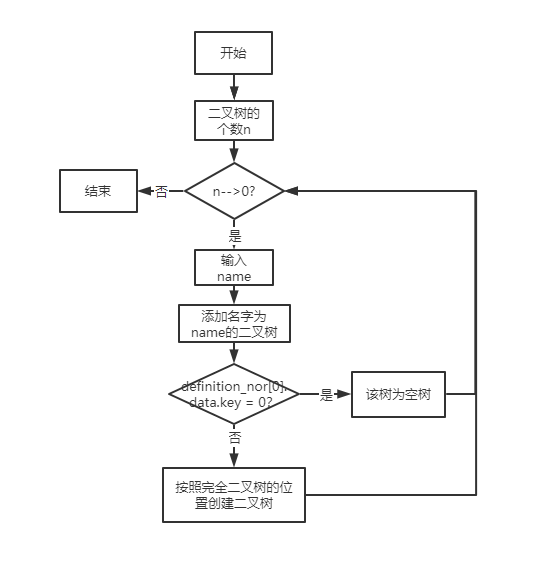
此外在具体功能的实现方面，由于要进行单二叉树的操作，以及多二叉树的管理，要在主函数中声明单链表BiTree T=NULL, node = NULL;，其中T是每棵树的根节点，node是子树中的节点，多二叉树结构体 wood Trees;，和储存每个二叉树名字的字符数组char name[30]，上述完成准备工作。

**3.3.3功能函数部分**

1. 添加二叉树

添加二叉树是初始化一个线性表的延伸，因为在此函数中使用到了单二叉树的初始化操作，单二叉树加入到多树的操作。首先请求用户说明要新增的二叉树的个数，接着请求用户输入这些二叉树的名字，借助Addtrees函数在多树结构中添加对应名字的二叉树，随后请求用户请以完全二叉树的位置输入节点，要求带有关键字和字符，以0 0 null结束，如果用户想要创造空二叉树的话，不添加节点即可。所以，在该操作中，对二叉树实现初始化的一系列操作后，该二叉树即自动被添加到多树管理结构中。

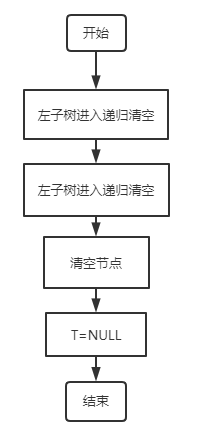




1. 清空二叉树

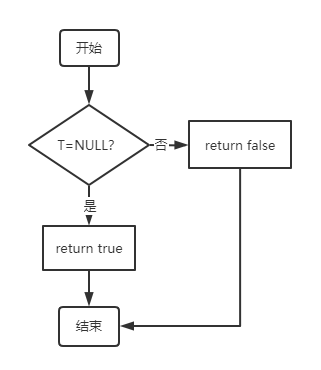
首先请求用户输入要要清空的二叉树的名字，判断要清空的二叉树是否存在，如果二叉树存在则执行清空操作：

* 左子树进入递归清空
* 右子树进入递归清空
* 清空节点
* 根节点T=NULL
* 返回OK



1. 判空二叉树

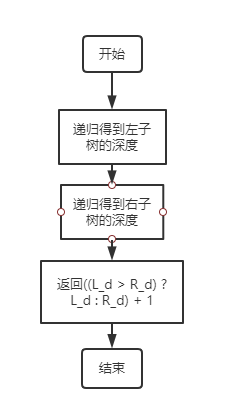
首先请求用户输入要进行判空操作的二叉树的名字，判断要判空的二叉树是否为存在，如果二叉树存在则执行判空操作，如果二叉树的根节点为空，则该二叉树为空，返回true，否则二叉树不为空，返回false。



1. 二叉树深度

首先请求用户输入要进行获取深度的二叉树的名字，判断此二叉树是否存在，如果存在则执行返回深度的操作：

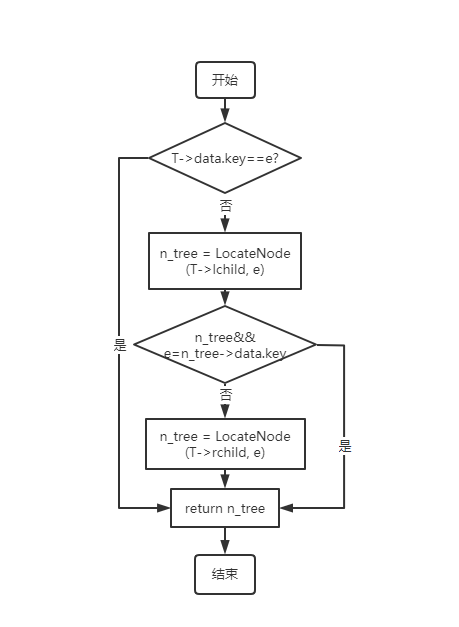
* 声明int L\_d表示左子树的深度
* 声明int R\_d表示右子树的深度
* 递归分别求根节点的左右子树的深度
* 返回(((L\_d > R\_d) ? L\_d : R\_d) + 1)



1. 查找结点

首先请求用户输入要进行获取元素操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行如下操作：

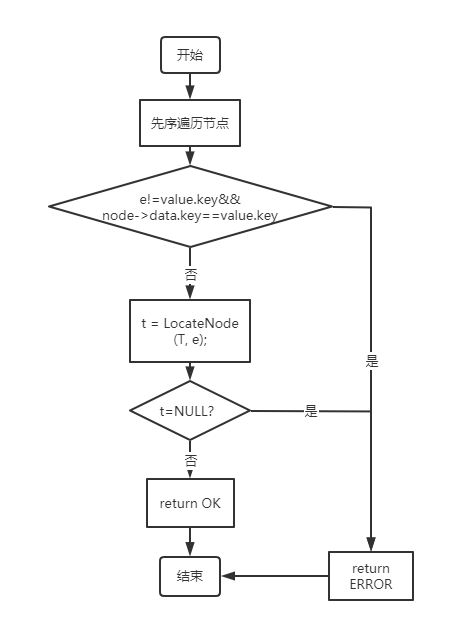
* 声明n\_tree作为关键值为e的节点的地址返回变量
* 如果根节点的关键值是e，使n\_tree=T，返回n\_tree
* 否则递归查找左右子树，其中如果关键值等于e的节点就在左子树则直接返回所以要在左右子树递归的中间加一个判断：if (n\_tree&&e==n\_tree->data.key) return n\_tree;，如果最终返回n\_tree的值，则代表关键字为e的节点在右子树。



1. 节点赋值

首先请求用户输入要进行赋值操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行以下操作：

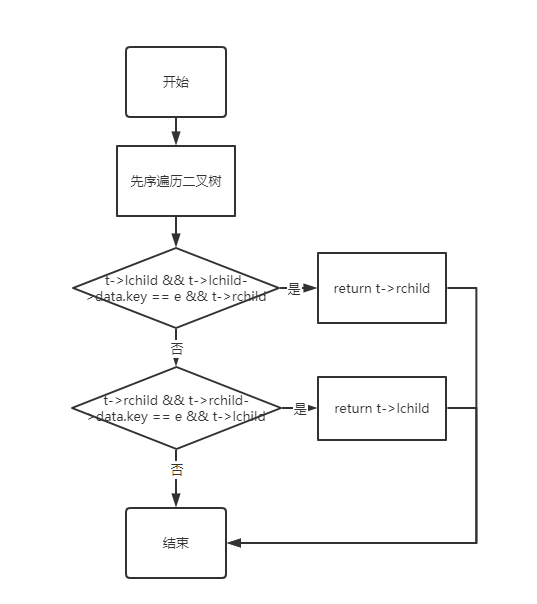
* 中序遍历二叉树
* 如果满足e!=value.key&&node->data.key==value.key则代表要改变的值不合法，重复，返回错误
* 判断key值是否存在，如果不存在，返回错误
* 如果都合法，则进行赋值操作
* 返回OK



1. 获得兄弟节点

首先请求用户输入要进行获取兄弟节点操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行以下操作：

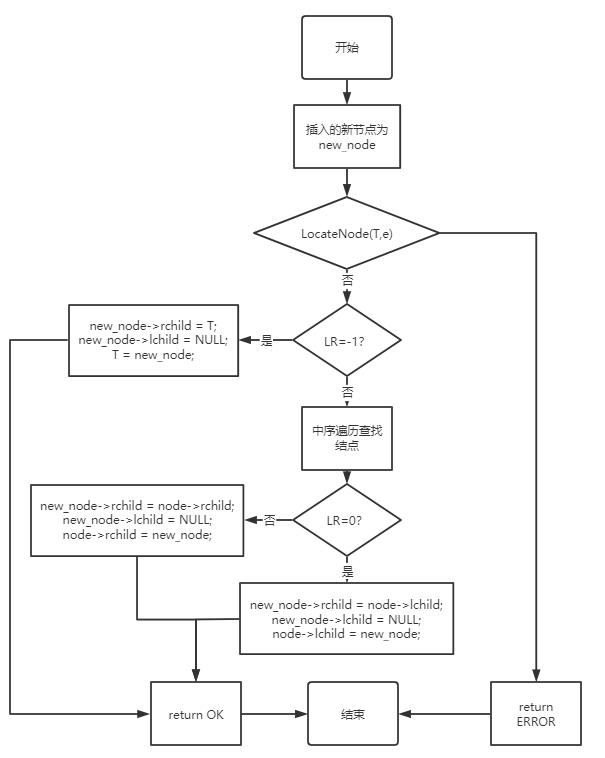
* 中序遍历寻找关键字的父节点
* 如果关键字节点和他的兄弟节点存在返回兄弟节点



1. 插入节点

首先请求用户输入要进行插入节点操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行以下操作：

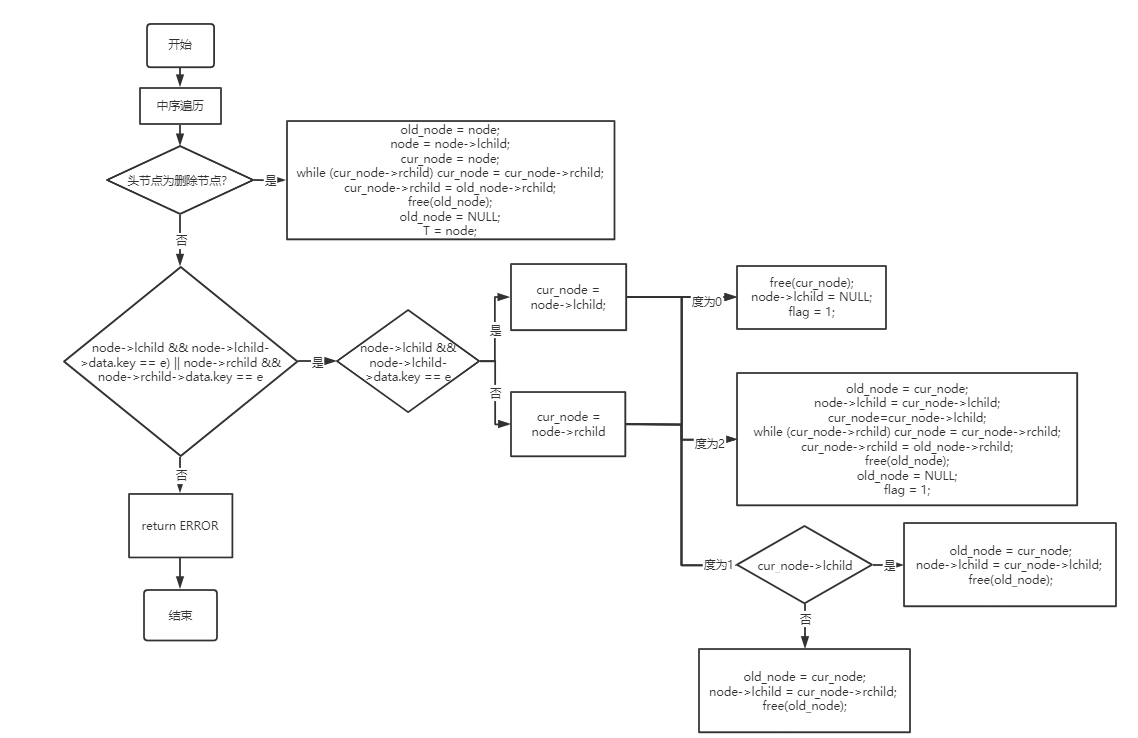
* 遍历二叉树判断插入之后的关键字是否重复
* 如果不重复进行插入操作
* 判断LR，如果LR等于-1进行插入头节点的操作
* 接着先序遍历二叉树寻找插入位置，并根据LR的值选择插入的位置



1. 删除节点

首先请求用户输入要进行删除节点操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行以下操作：

* 中序遍历二叉树分一下情况
* 如果是根节点，删除根节点，右子树作为左子树最右叶子的右子树
* 如果删除节点是左子树，度为0时直接删除，度为2时，删除节点的右子树作为左子树最右叶子的右子树，度为1时，叶子代替删除结点的位置
* 如果删除节点是右子树，度为0时直接删除，度为2时，删除节点的右子树作为左子树最右叶子的右子树，度为1时，叶子代替删除结点的位置



1. 前序遍历

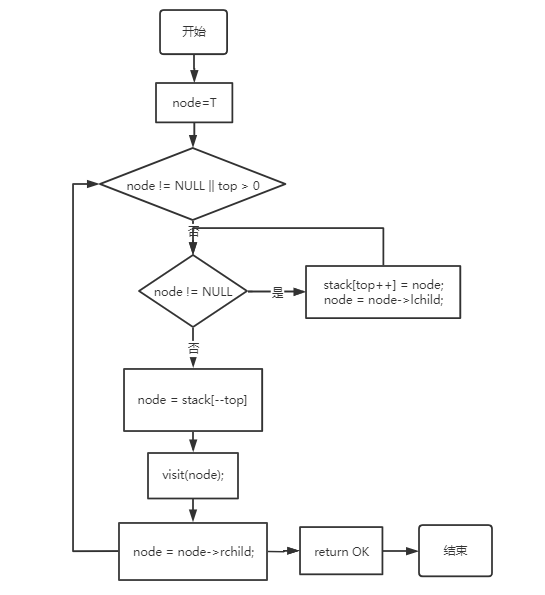
首先请求用户输入要进行前序遍历操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行以下操作

* 如果T不为空
* 打印节点信息
* 递归左子树打印节点信息
* 递归右子树打印节点信息

1. 中序遍历（非递归）

首先请求用户输入要进行中序遍历操作的二叉树的名字，判断此而二叉树是否存在，如果存在则进行以下操作：

* node=T，当node不为空且栈不为空时进行遍历：
* 当node不为空时node入栈，node=node->lchild
* 当node为空时，node出栈，打印node的信息，node=stack[top--]
* node=node->rchild回到第一步重复上述操作
* 当第一步的条件不满足时，遍历结束



1. 后序遍历

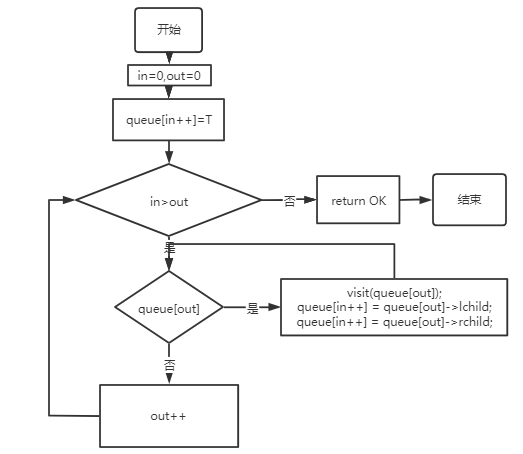
首先请求用户输入要进行后序遍历操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行以下操作：

* 如果T不为空
* 递归左子树打印节点信息
* 递归右子树打印节点信息
* 打印节点信息

1. 层序遍历

首先请求用户输入要进行后序遍历操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行以下操作：

* BiTree queue[100] 作为队列,in=0,out=0;
* queue[in++]=T
* 如果in>out，进行下一步
* 如果queue[out]存在，打印queue[out]的信息，并将他的两个孩子加入队列
* out++，回到第三步没重复上述过程



1. 保存二叉树

首先请求用户输入要进行保存操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行以下操作

* 如果T不为空，线序递归遍历存入节点
* 如果T为空，存入已经预设的节点

1. 加载二叉树

首先请求用户输入要进行加载操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行以下操作

* 循环读取文件写入TElemType的数组
* 按照educoder第一关的函数，重新创建二叉树：CreateBiTree(T, d)

1. 销毁二叉树

首先请求用户输入要进行销毁操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行后序递归销毁操作

1. 遍历森林

循环对森林中的每一棵树进行遍历操作

## 3.4 系统测试

系统环境：Windows10 专业版

系统版本号：20H2

测试用IDE：Clion

IDE版本号：2020.2.4

测试1：测试创建空树：

表1：测试创建空树

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 创建名为a  的空二叉树 | a |  |

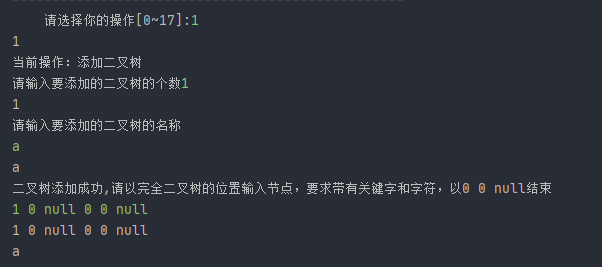


图3-4-1-1 测试1运行图

测试2：测试清空二叉树：

表2：测试清空二叉树

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 清空空二叉树 | 该二叉树已经清空 |  |
| 2 | 清空内容为1,a 2,b 3,c的二叉树 | 已清空该线性表 |  |

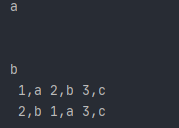


图3-4-2-1 测试2用例创建

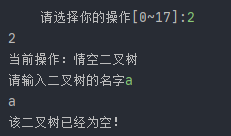


图3-4-2-2 测试2用例1运行结果图

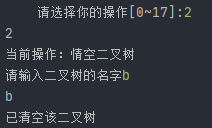


图3-4-2-3 测试2用例2运行结果图

测试3：测试判空二叉树：

表3：二叉树

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 判断空二叉树 | 该二叉树为空 |  |
| 2 | 判断数据为1,a 2,b 3,c的线性表 | 该二叉树不为空 |  |

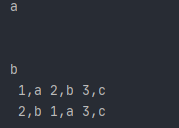


图3-4-3-1 测试3用例创建

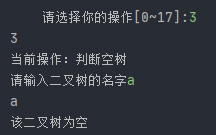


图3-4-3-2 测试3用例1运行结果图

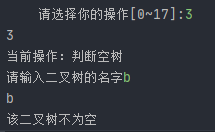


图3-2-4-3-3测试3例2运行结果图

测试4：测试求二叉树的深度：

表4 ：测试二叉树的深度

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 求空树的深度 | 0 |  |
| 2 | 求数据为1,a 2,b 3,c的完全二叉树的深度 | 2 |  |

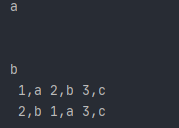


图3-4-4-1 测试4用例创建

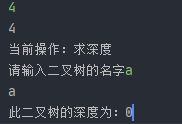


图3-4-4-2 测试4用例1运行结果图

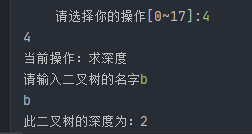


图3-4-4-3 测试4用例2运行结果图

测试5：测试查找节点：

表5：测试查找节点（已经创建线性表1 1 a 2 2 b 3 3 c 6 4 d 7 5 e）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 寻找元素10 | ERROR |  |
| 2 | 寻找元素3 | 3 c |  |

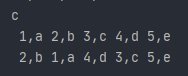


图3-4-5-1 测试5用例创建

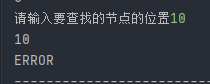


图3-4-5-2 测试5用例1运行结果图

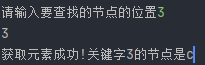


图3-4-5-3 测试5用例3运行结果图

测试6：测试节点赋值：

表6：测试赋值

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null 3 2 new | ERROR |  |
| 2 | 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null 10 20 new | ERROR |  |
| 3 | 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null 3 10 new | 1,a 2,b 10,new 4,d 5,e  2,b 1,a 4,d 10,new 5,e |  |
| 4 | 1 1 a 2 2 b 3 3 c 6 4 d 7 5 e 0 0 null 3 3 new | 1,a 2,b 3,new 4,d 5,e  2,b 1,a 4,d 3,new 5,e |  |

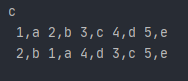


图3-4-6-1 测试6用例创建

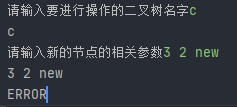


图3-4-6-2 测试6用例1运行结果图

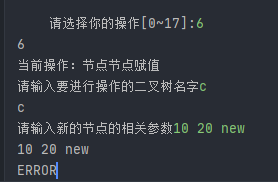


图3-4-6-3 测试6用例2运行结果图

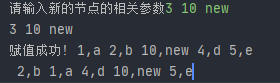


图3-4-6-4 测试6用例3运行结果图

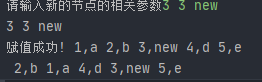


图3-4-6-4 测试6用例4运行结果图

测试7：测试获取兄弟节点：

表7：测试获取兄弟节点

（a为1 1 a 2 2 b 3 3 c 6 4 d 7 5 e

c为1 11 a 2 12 b 5 13 c 3 20 f 6 14 d 7 15 e）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 求a一个关键字为22的兄弟节点 | ERROR |  |
| 2 | 求c关键字为13的兄弟节点 | ERROR |  |
| 3 | 求a一个关键字为3的兄弟节点 | 2,b |  |
| 4 | 求a一个关键字为4的兄弟节点 | 5,e |  |

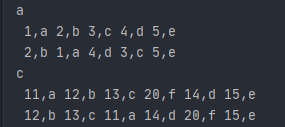


图3-4-7-1 测试7用例创建

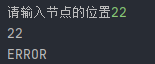


图3-4-7-2 测试7例1运行结果图

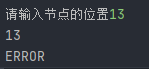


图3-4-7-3 测试7例2运行结果图

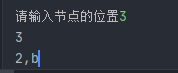


图3-4-7-4 测试7例3运行结果图

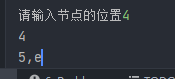


图3-4-7-5 测试7例4运行结果图

测试8：测试插入节点：

表7：测试插入节点

（a为1 1 a 2 2 b 3 3 c 6 4 d 7 5 e）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 输入 3 0 3 f | ERROR |  |
| 2 | 输入 10 0 6 f | ERROR |  |
| 3 | 输入 3 0 6 f | 1,a 2,b 3,c 6,f 4,d 5,e  2,b 1,a 6,f 4,d 3,c 5,e |  |
| 4 | 输入3 -1 6 f | 6,f 1,a 2,b 3,c 4,d 5,e  6,f 2,b 1,a 4,d 3,c 5,e |  |

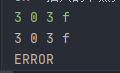


图3-4-8-1 测试8例1运行结果图

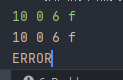


图3-4-8-2 测试8例2运行结果图

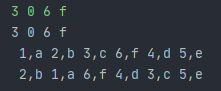


图3-4-8-3 测试8例3运行结果图

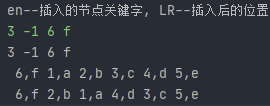


图3-4-8-4 测试8例4运行结果图

测试9：测试删除节点：

表9 测试删除节点

（a为1 1 a 2 2 b 3 3 c 6 4 d 7 5 e）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 删除 10 | 删除操作失败 |  |
| 2 | 删除1 | 1,a 2,b 6,f 3,c 5,e  2,b 6,f 1,a 3,c 5,e |  |
| 3 | 删除4 | 1,a 2,b 6,f 3,c 5,e  2,b 6,f 1,a 3,c 5,e |  |
| 4 | 删除2 | 1,a 6,f 3,c 4,d 5,e  6,f 1,a 4,d 3,c 5,e |  |

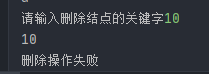


图3-4-9-1 测试9例1运行结果图

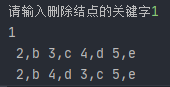


图3-4-9-2 测试9例2运行结果图

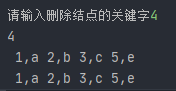


图3-4-9-3 测试9例3运行结果图

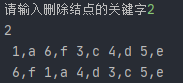


图3-4-9-4 测试9例4运行结果图

测试10：遍历操作：

表10 测试遍历操作

（a为1 1 a 2 2 b 3 3 c 6 4 d 7 5 e）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 对a先序遍历 | 1,a 2,b 6,f 3,c 4,d 5,e |  |
| 2 | 对a中序遍历 | 2,b 6,f 1,a 4,d 3,c 5,e |  |
| 3 | 对a后序遍历 | 6,f 2,b 4,d 5,e 3,c 1,a |  |
| 4 | 对a层序遍历 | 1,a 2,b 3,c 6,f 4,d 5,e |  |

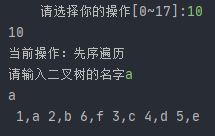


图3-4-10-1 测试10例1运行结果图

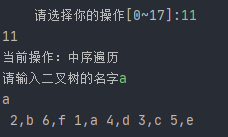


图3-4-10-2 测试10例2运行结果图

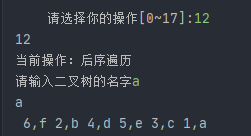


图3-4-10-3 测试10例3运行结果图

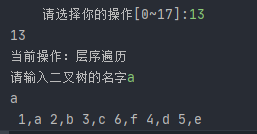


图3-4-10-4 测试10例4运行结果图

测试11：测试保存二叉树：

表12 测试保存线性表

（a为1 1 a 2 2 b 3 3 c 6 4 d 7 5 e）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 将a的数据保存至文件list.dat中 | 保存成功 |  |

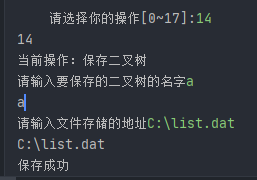


图3-4-11-1 测试11例1运行结果图

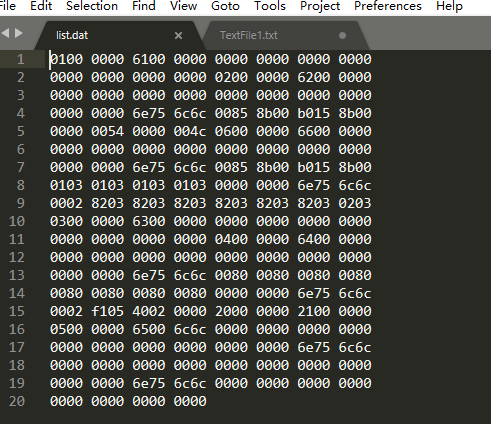
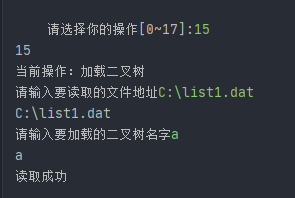


图3-4-11-2 测试11文件保存结果图

测试12：测试加载线性表：

表12 测试加载线性表

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 文件list1.dat的数据加载进表a中 | 读取成功 |  |



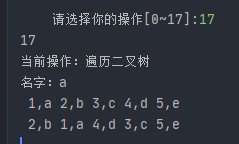


图3-4-12-1 测试12例1运行结果图

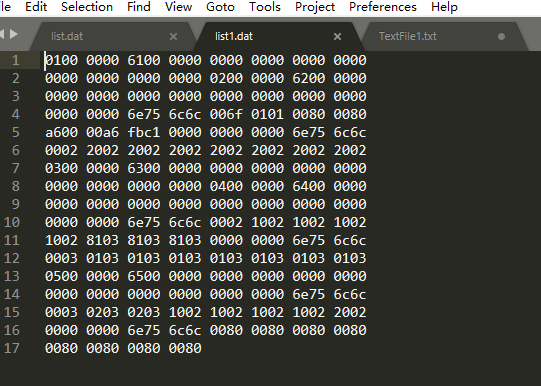


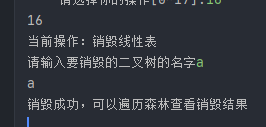
图3-4-12-2 测试12要读取文件数据图

测试13：测试销毁二叉树：

表13 测试销毁二叉树

（a为1 1 a 2 2 b 3 3 c 6 4 d 7 5 e）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 销毁二叉树a | 销毁成功 |  |



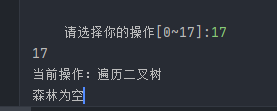


图3-4-13-1 测试13例1的运行结果图

测试14：测试遍历森林：

表14 测试遍历森林

（a为1 1 a 2 2 b 3 3 c 6 4 d 7 5 e

b为1 11 a 2 12 b 5 13 c 3 20 f 6 14 d 7 15 e

c为 1 1 a 2 2 b 3 3 c）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 遍历森林 | a  1,a 2,b 3,c 4,d 5,e  2,b 1,a 4,d 3,c 5,e  b  11,a 12,b 13,c 20,f 14,d 15,e  12,b 13,c 11,a 14,d 20,f 15,e  c  1,a 2,b 3,c  2,b 1,a 3,c |  |

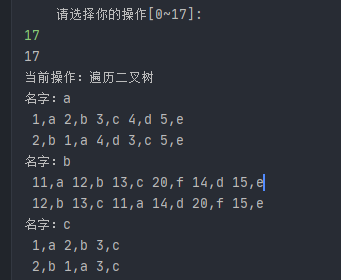


图2-4-16-1 测试16例1的运行结果图

## 3.5 实验小结

本次实验的主要任务就是在二叉树的链式存储表示和实现下完成二叉树的数据类型中的基本操作，包括二叉树的初始化，销毁一个二叉树，清空一个二叉树，判断一个二叉树是否为空，求一个二叉树的深度，获取二叉树中某一个节点的位置，获取二叉树中某一个节点的兄弟节点，向二叉树中插入一个二叉树，向二叉树中删除一个节点，对二叉树中的某个节点进行重新赋值，这些基本操作还可以相互组合实现一些更加复杂的操作，虽然此次实验对多树管理的要求并不高，因为经过前两次的训练之后，以及实验要求的用顺序表的方式来进行多表管理，应该使用的比较熟练了，而二叉树的的文件保存和加载重要的是在于如何读取文件后以合适的方式再次创建二叉树，这里我采用了讲读取的二叉树节点信息存在数组中，然后再次使用二叉树的创建函数完成对二叉树从文件中加载出来，而不是采用边读取边创建的方式。

在函数实现方面，遇到的较大的问题的就是，二叉树节点删除和节点重新赋值，这里面都涉及到，在删除节点中需要考虑的情况比较多，比如说要考虑到头节点的情况，考虑到左右子树的情况，要考虑到删除的节点是否是叶子的情况等等；而在节点赋值考虑情况比较容易使人绕进去，要考虑赋值之后的二叉树的关键字是否重复，而且要注意的是，要在赋值之前就要完成判断，这里就要排除要求改的关键字和修改后的关键字相同的请况，只有赋值之前完成对关键字合法与否的判断后才能进行赋值的操作，其实整个实验的难点，就是考虑关键字重复的情况要如何判断。

在此次实验中，由于对上述问题的探讨不够全面导致在和助教检查的时候出现很多错误，在助教旁边修改代码的时候，助教向我讲述的算法边界条件的问题使我获益颇多。

# 4 基于邻接表的图实现

## 4.1 问题描述

采用邻接表作为图的物理结构，实现图的基本运算，可任选无向图、有向图、无向网和有向网这四种图中的一种实现，本次实验选择的是实现无向图，其中ElemType为数据元素的类型名，具体含义可自行定义，但要求顶点类型为结构型，至少包含二个部分，一个是能唯一标识一个顶点的关键字，类型名为KeyType，另一个是其它部分，类型名为char。

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。

演示系统可选择实现图的文件形式保存。其中，①需要设计文件数据记录格式，以高效保存图的数据逻辑结构(D,{R})的完整信息；②需要设计图文件保存和加载操作合理模式。

演示系统可选择实现多个图管理。

演示系统的源程序应按照代码规范增加注释和排版，目标程序务必是可以独立于IDE运行的EXE文件。

**4.1.1 无向图的基本操作定义**

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。

## 4.2 系统设计

结合单二叉树和多二叉树的操作之间的关系设计出演示系统的交互，首先创建一个包含多个线性表的结构体Graphs，作为各个功能操作的主体，在这个结构体中，可以包含多个有以名字标志的二叉树以便进行选中指定二叉树进行相应的基本操作，此多树结构设计如图3-1-1所示：

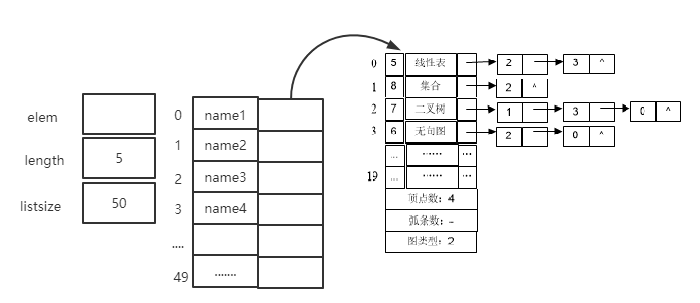


图4-1-1多图管理的物理结构示意图

系统设计图如4-1-2所示：

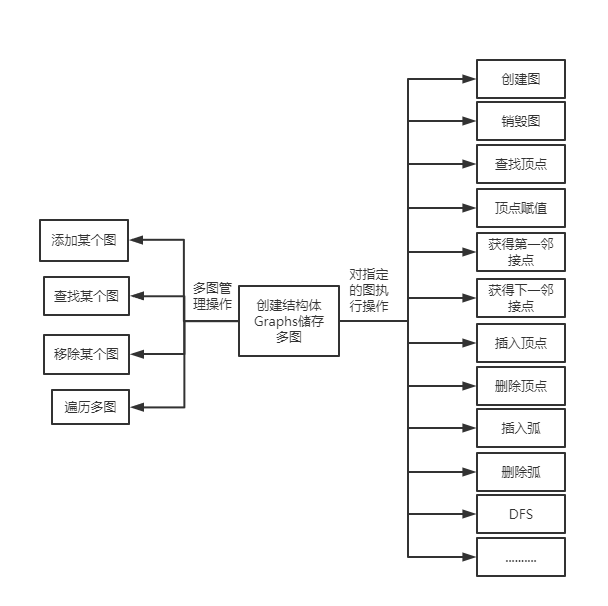


图3-1-2系统设计图

## 4.3 系统实现

**4.3.1一些在主函数前的准备**

* 1. 函数返回值宏设定：

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

* 1. 图储存类型和相关函数返回值类型，以及图的枚举类型设定：

typedef int status;

typedef int KeyType;

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

* 1. BFS遍历中所使用到的标记数组和队列结构的定义：

typedef int Boolean;

Boolean visited[MAX\_VERTEX\_NUM];

typedef struct{

int data[MAX\_VERTEX\_NUM];

int front;

int rear;

}Queue;

* 1. 图和多图相关存储结构的定义：

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 {

struct {

char name[30];

ALGraph graph;

} elem[100];

int length;

int listsize;

}Graphs;

**4.3.2主函数部分**

1. 演示系统的打印：

由于要实现构造出一个具有菜单功能的功能演示系统，在这其中要在主程序中完成函数的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。所以在主函数部分要打印出相应的菜单界面，并且使用system("cls");函数在执行完一次功能之后使界面清空，使界面显得干净整洁，方便进行下一次操作。

1. 演示系统的功能选择：

同时为了实现选择对应功能来进行相应的操作，采用switch-case语句完成选择特定的函数功能实现部分以完成相应的操作。

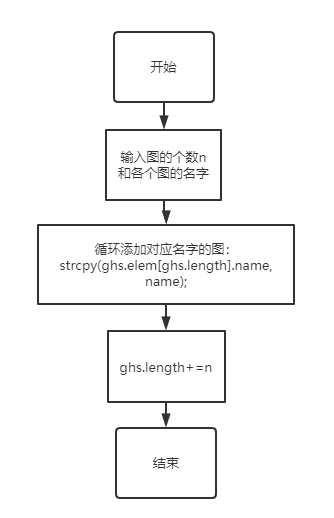
1. 多图管理部分：

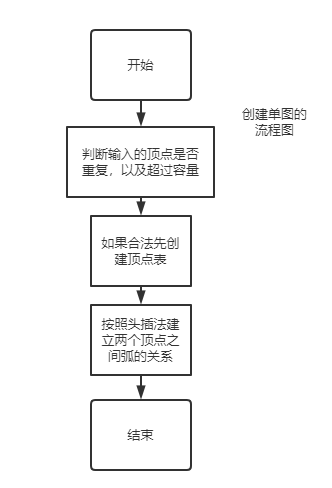
此外在具体功能的实现方面，由于要进行单图的操作，以及多图的管理，要在主函数中声明单图ALGraph G，多图结构体 Graphs ghs，和储存每个图的字符数组char name[30]，以及后面的文件存储地址的上述完成准备工作。

**4.3.3功能函数部分**

1. 添加图

添加图是初始化一个图的延伸，因为在此函数中使用到了单图的初始化操作，单图加入到多图的操作。首先请求用户说明要新增的图的个数，接着请求用户输入这些图的名字，借助AddGraphs函数在多图结构中添加对应名字的图，随后请求用户输入图中的顶点信息，要求带有关键字和字符，以-1 nil结束，接着输入结点之间的关系，以-1，-1结尾。在该操作中，对图实现初始化的一系列操作后，该图即自动被添加到多图管理结构中。





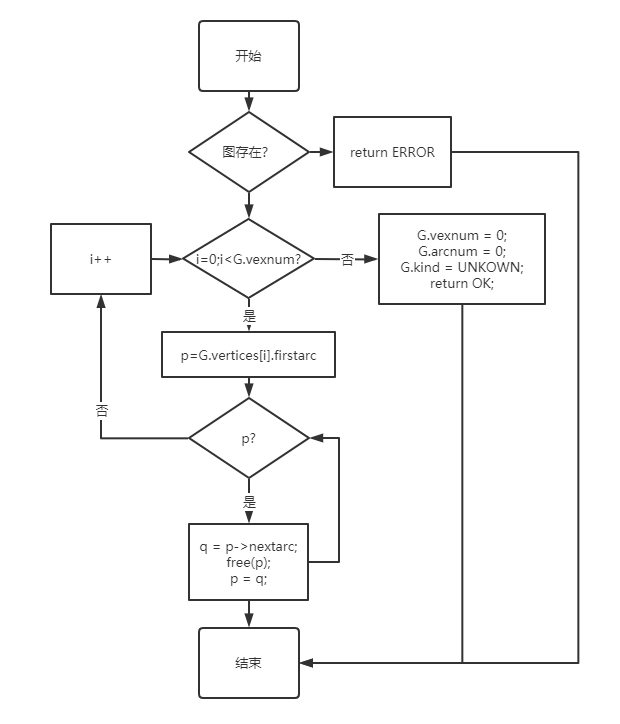
1. 销毁图

首先请求用户输入要销毁的图的名字，判断要销毁的图是否存在，如果图存在则执行销毁操作：

* 先循环释放邻接表中的弧：



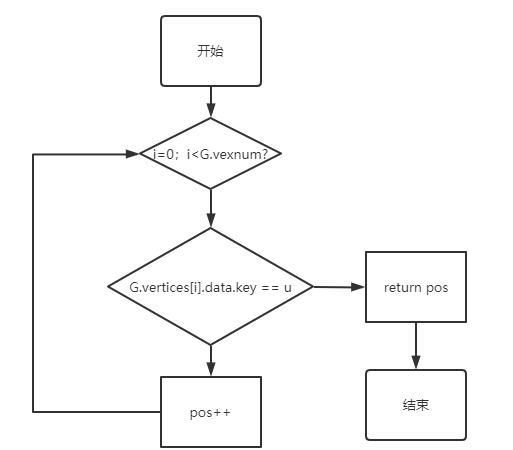
* 图的顶点数，边数置为0
* 图的类型置为UNKOWN
* 根节点T=NULL
* 释放图的名字
* 循环将销毁的图后面的图前移



1. 查找顶点

首先请求用户输入要进行获取元素操作的图的名字，判断此图是否存在，如果存在则进行如下操作：

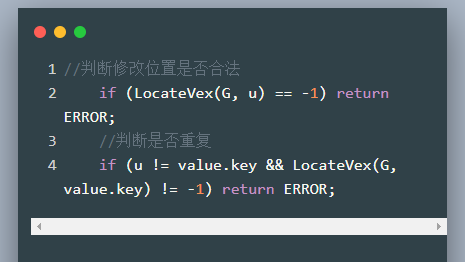
* 循环比对顶点的key是否与给定的值相等，如果相等返回序号pos
* 如果循环结束仍然找不到，返回-1



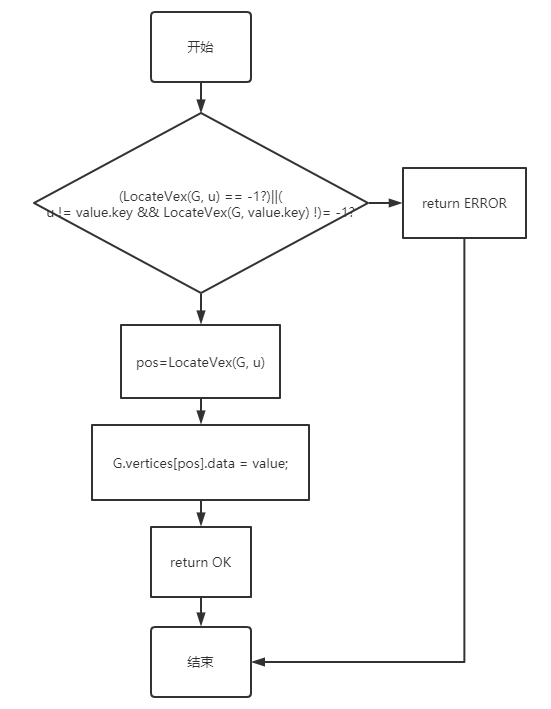
1. 顶点赋值

首先请求用户输入要进行相应操作的图的名字，判断此图是否存在，如果存在则进行如下操作：

* 首先判断位置是否合法



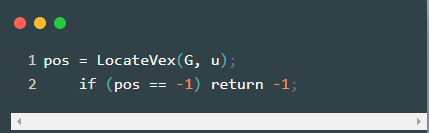
* 如果位置合法则进行赋值操作
* 先定位修改的key的位置pos
* 进行赋值操作G.vertices[pos].data = value;



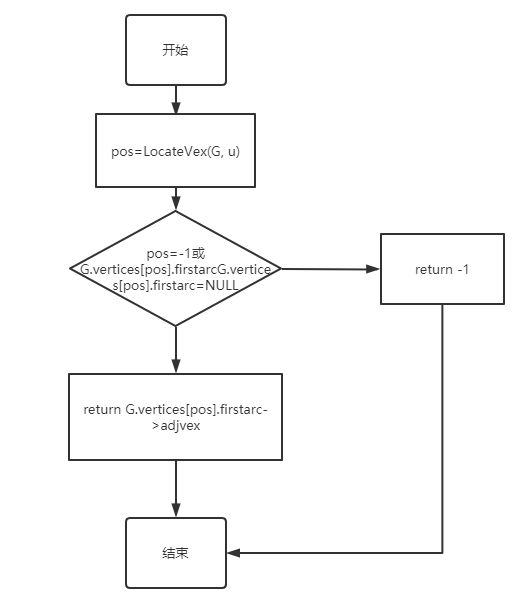
1. 获得第一邻接点

首先请求用户输入要进行相应操作的图的名字，判断此图是否存在，如果存在则进行如下操作：

* 判断相应关键字的序号是否存在：



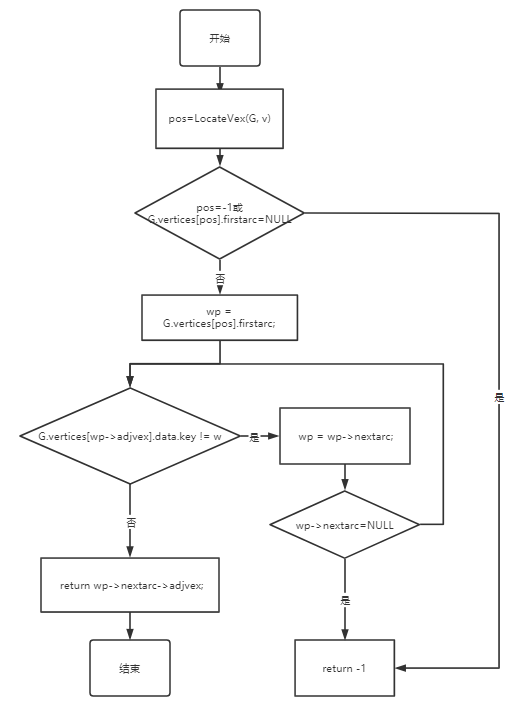
* 判断序号为pos的顶点是否有第一邻接点，如果没有返回-1
* 如果存在第一邻接点，返回第一邻接点的关键字



1. 获得下一邻接点

首先请求用户输入要进行相应操作的图的名字，判断此图是否存在，如果存在则进行如下操作：

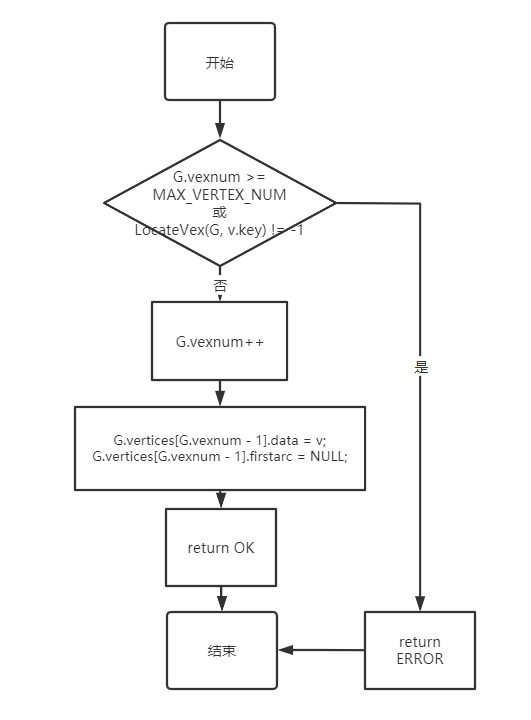
* 寻找v，如果找不到返回-1
* 寻找w，如果找不到，返回-1
* 如果找到w，返回w的位序



1. 插入顶点

首先请求用户输入要进行相应操作的图的名字，判断此图是否存在，如果存在则进行如下操作：

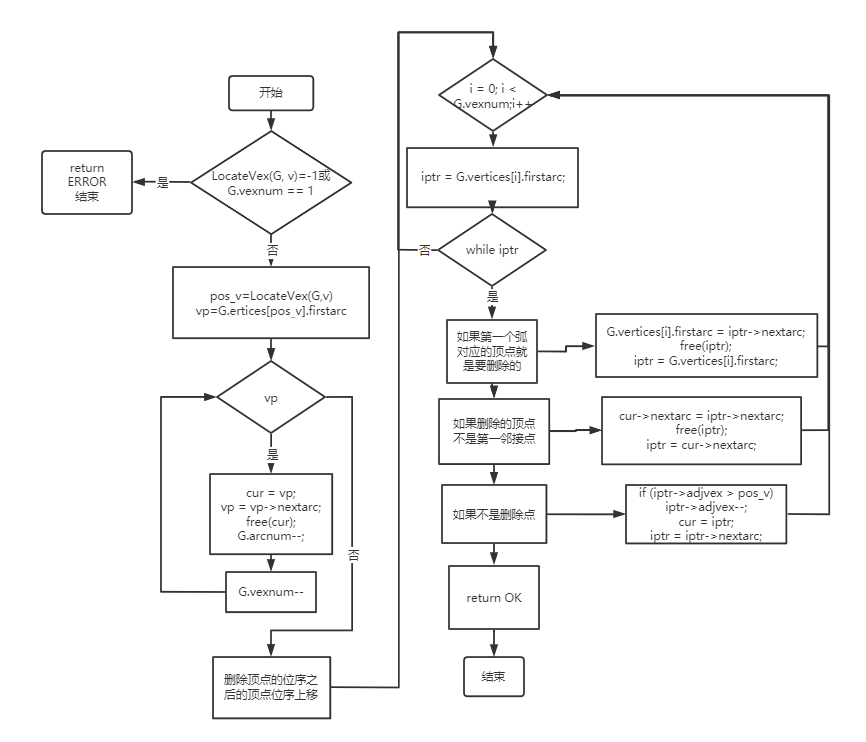
* 判断插入位置是否合法以及插入之后的关键字是否重复
* 如果都合法则进行插入操作
* 图的顶点数加1



1. 删除顶点

首先请求用户输入要进行相应操作的图的名字，判断此图是否存在，如果存在则进行如下操作：

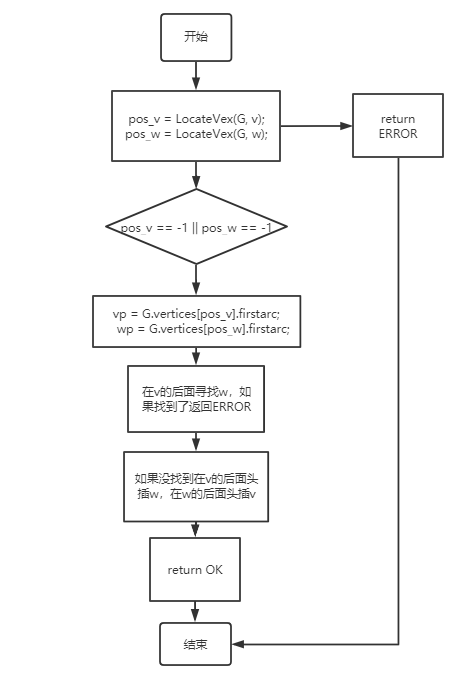
* 判断删除的关键字是否合法以及图的顶点数是否为1
* 如果条件符合删除条件则执行删除操作
* 首先删除相应顶点的行邻接表，图的节点数减一
* 删除结点之后，将删除之后邻接表的位序上移一位
* 遍历删除之后的邻接表，遇到删除节点的位序就删除，遇到非删除结点的位序就减一

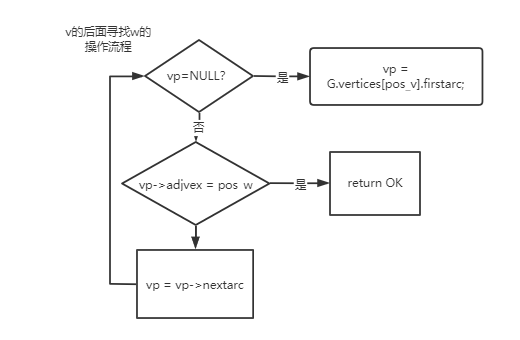


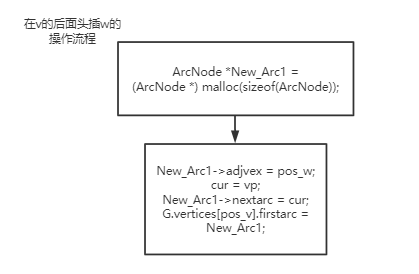
1. 插入弧

首先请求用户输入要进行相应操作的图的名字，判断此图是否存在，如果存在则进行如下操作：

* 判断插入弧的位置是否合法
* 在v的后面寻找w，如果有w，则弧已经存在，返回ERROR
* 如没有w，执行插入操作：在v的后面头插w，在w的后面插入v
* 图的弧数加一







1. 删除弧

首先请求用户输入要进行后序遍历操作的图的名字，判断此图是否存在，如果存在则进行以下操作：

* 判断弧的两个端点是否存在，以及弧是否存在，如果存在进行删除胡弧的操作
* 在v的后面查找w，删除弧
* 在w的后面寻找v，删除弧

1. 深度优先遍历

首先请求用户输入要进行后序遍历操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行以下操作：

* 首先初始化标记数组，赋值false，表示所有顶点都没有被遍历过
* 从序号为0的顶点开始遍历，遍历该邻接点之后的顶点，将遍历之后的顶点的序号对应的数组赋值为true，表示遍历过了
* 接下来遍历其他顶点与上述内容相似
* 遍历之后返回OK

1. 广度优先遍历

首先请求用户输入要进行后序遍历操作的图的名字，判断图是否存在，如果存在则进行以下操作：

* 首先初始化标记数组，赋值false，表示所有顶点都没有被遍历过
* 初始化队列Q
* 从序号为0的顶点开始遍历，将遍历之后的顶点的序号对应的数组赋值为true，表示遍历过了，将遍历之后的点入栈，如果栈不为空，栈顶顶点出栈，同时遍历与它所有相关的弧的列另一个顶点，并将它们入栈，遍历之后的需要把相应序号的数组记为true
* 接下来遍历其他顶点与上述内容相似
* 遍历之后返回OK

1. 保存图

首先请求用户输入要进行保存操作的图的名字，判断此图是否存在，如果存在则进行保存图写入指定文件地址的操作

1. 加载图

首先请求用户输入要进行加载操作的二叉树的名字，判断此二叉树是否存在，如果存在则进行将指定地址的文件加载出来写入图的操作

1. 遍历图

首先请求用户输入要进行遍历操作的图的名字，判断此图是否存在，如果存在则进行遍历图的邻接表的操作。

1. 遍历多图

循环对多图中的每一个图进行遍历操作。

## 4.4 系统测试

系统环境：Windows10 专业版

系统版本号：20H2

测试用IDE：Clion

IDE版本号：2020.2.4

测试1：创建图

表1：测试创建图

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 创建一个图 | 创建成功 | 见输出结果图示 |

（注：创见的图的内容为：1 A 2 B 3 C 4 D 8 F 6 E -1 nil 1 2 1 6 1 3 3 4 6 4 2 4 2 8 4 8 -1 -10）

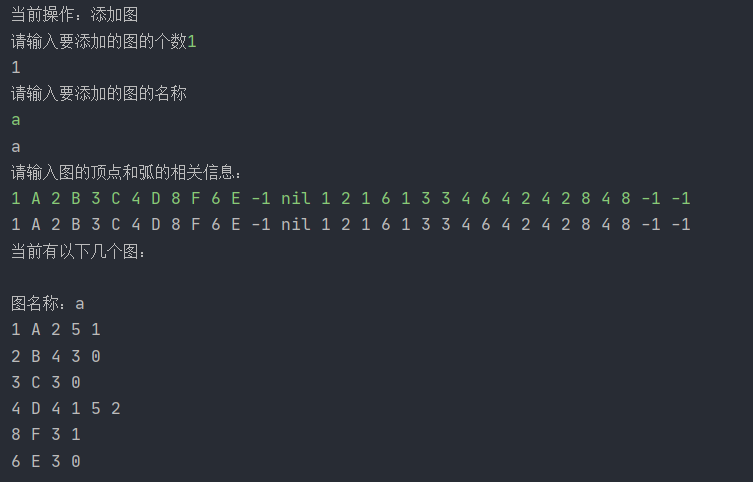


图4-4-1-1 测试1运行结果图

测试2：销毁图

表2：测试销毁图

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 销毁上一测试  创建的图 | 销毁成功 |  |

（销毁的结果可以通过查看遍历多图的操作查看）

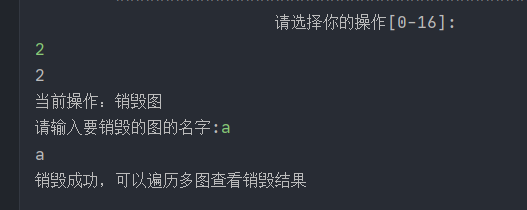
****

图4-4-2-1 测试1运行图

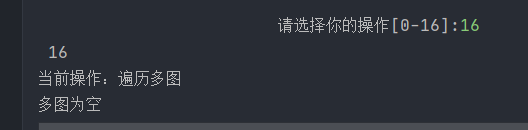
****

图4-4-2-2 测试1结果检验图

测试3：查找顶点

表3：测试查找顶点

（测试用图为测试1创建的图a）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 查找关键字为1的顶点的位置 | 查找成功，0 |  |
| 2 | 查找关键字为5的顶点的位置 | 查找失败 |  |
| 3 | 查找关键字为8的顶点的位置 | 查找成功 |  |

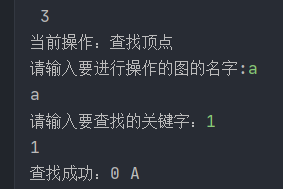


图4-4-3-1 测试3用例1运行图

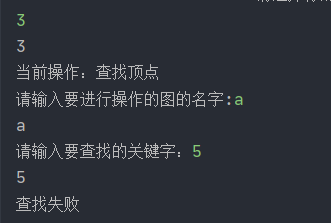


图4-4-3-2 测试3用例2运行图

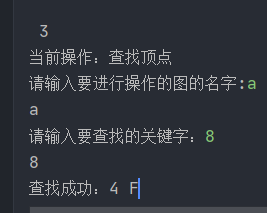


图4-4-3-3 测试3用例3运行图

测试4：顶点赋值

表4：测试顶点赋值

（测试用图为测试1创建的图并插入顶点9 G和10 O）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 赋值6 10 E | 赋值操作失败 |  |
| 2 | 赋值6 10 O | 修改成功 |  |

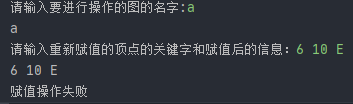


图4-4-4-1 测试4用例1运行图

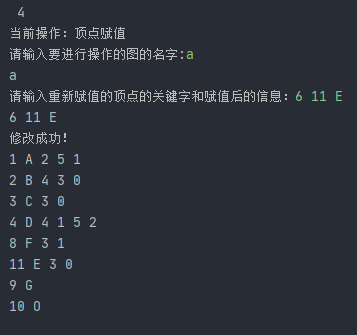


图4-4-4-2 测试4用例2运行图

测试5：获得第一邻接点

表5：测试获取第一邻接点

（测试用图为测试1创建的图a）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 获得2的第一邻接点 | 8 F |  |

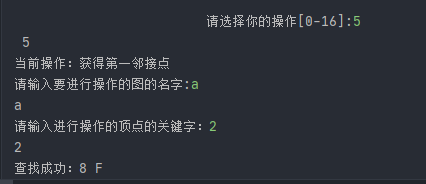


图4-4-5-1 测试5运行图

测试6：获得下一邻接点

表6：测试获得下一邻接点

（测试用图为测试1创建的图a）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 获得2 8的下一邻接点 | 查找成功 4 D |  |

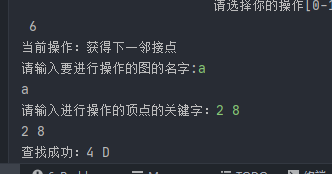


图4-4-6-1 测试6运行图

测试7：插入顶点

表7：测试插入顶点

（测试用图为测试1创建的图a）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 插入顶点9 G | 插入顶点成功 |  |
| 2 | 插入顶点10 O | 插入顶点成功 |  |

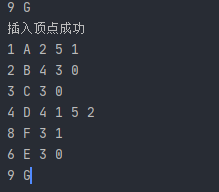


图4-4-7-1 测试7用例1运行图

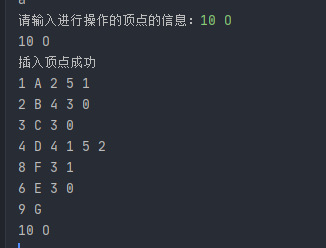


图4-4-7-2 测试7用例2运行图

测试8：删除顶点

表8：测试删除顶点

（测试用图为测试9修改之后的图）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 删除关键字为2的顶点 | 删除成功 |  |

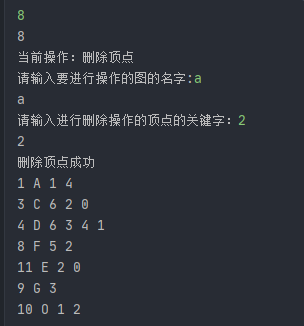


图4-4-8-1 测试8运行图

测试9：插入弧

表9：测试插入弧

（测试用图为测试4修改之后的图）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 插入弧8 9 | 插入成功 |  |
| 2 | 插入弧 4 10 | 插入成功 |  |
| 3 | 插入弧3 10 | 插入成功 |  |

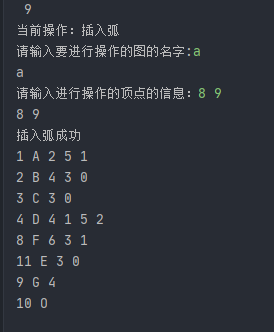


图4-4-9-1 测试9用例1运行图

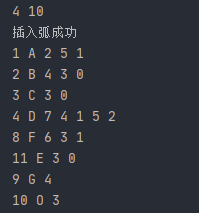


图4-4-9-2 测试9用例2运行图

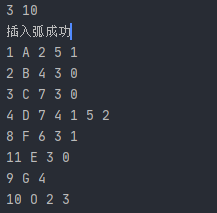


图4-4-9-3 测试9用例3运行图

测试10：删除弧

表10：测试删除弧

（测试用图为测试8修改之后的图）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 删除弧4 8 | 删除成功 |  |

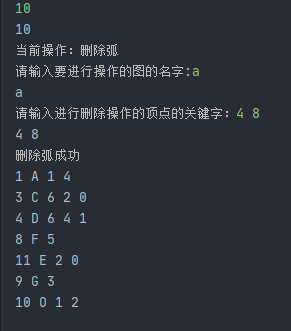


图4-4-10-1 测试10运行图

测试11：DFS

表11：测试DFS

（测试用图为测试1创建的图a）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 深度遍历测试图两遍 | 1 A 3 C 4 D 8 F 2 B 6 E |  |



图4-4-11-1 测试11运行图

测试12；BFS

表12：测试BFS

（测试用图为测试10修改之后的图）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 广度遍历测试图 | 1 A 3 C 11 E 10 O 4 D 8 F 9 G |  |

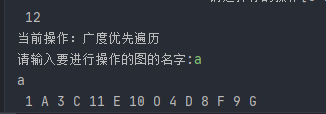


图4-4-12-1 测试12运行图

测试13：保存图

表13：测试保存图

（测试用图为测试10修改之后的图）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 保存测试图至C:\list.dat | 保存成功 |  |

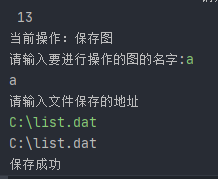
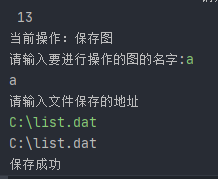


图4-4-13-1 测试13运行图



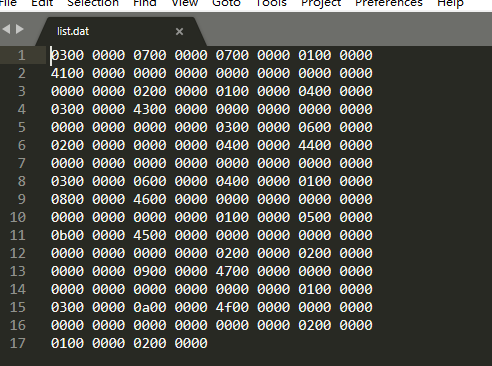


图4-4-13-2 测试13文件保存图

测试14：加载图

表14：测试加载图

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 加载测试13保存的内容 | 加载成功 |  |

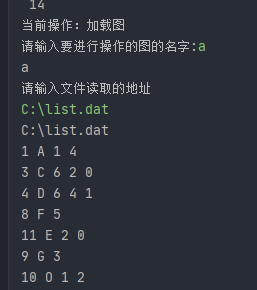


图4-4-14-1 测试14运行图

测试15：遍历图

表15：测试遍历图

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 遍历测试14加载之后的图 | 遍历结果符合实际 | 查看运行结果图 |

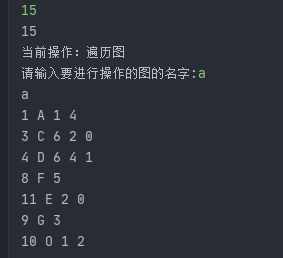


图4-4-5-1 测试15运行结果图

测试16：遍历多图

表16：测试遍历多图

（在测试15用图的基础上添加一个新图：

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）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试序号 | 测试内容 | 预计输出 | 实际输出 |
| 1 | 遍历供测试的多图 | 遍历结果符合实际 | 查看运行结果图 |

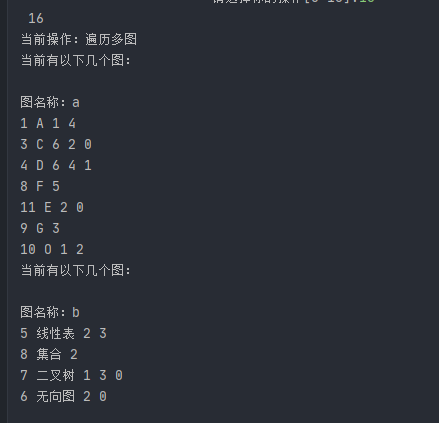


图4-4-16-1 测试16运行图

## 4.5 实验小结

本次实验的主要任务就是在图的邻接表存储表示和实现下完成图的数据类型中的基本操作，包括图的的初始化与创建，销毁一个图，查找图的顶点，顶点赋值，获得某个顶点的第一邻接点，获得某个顶点的下一邻接点，在图中插入一个顶点，删除图中的一个顶点，在图中插入弧，删除弧，对图分别进行深度优先遍历以及广度优先遍历，这些基本操作还可以相互组合实现一些更加复杂的操作，对于多图管理的实现，仍可以用顺序表的方式来进行多图管理。本次实验中较为困难的就是删除节点操作，虽然其本身并没有太大的难度，但是操作较为繁琐。接着就是DFS和BFS遍历操作，分别要求对栈和队列的相关操作掌握较为熟练并能够灵活使用。

本次实验的完成意味着本次实验课的完结，本门课程经过了运动会的一次中断和考试周的延后，最终还是顺利结课，经过本次课程的学习，我学的很多，不仅仅是对课上的内容的实践与巩固，但是我也很清楚，数据结构学习的知识光靠实验课上要求完成的内容是远远达不到的，所以在日后的学习中要对课上学习到的而在是实验中并没有要求的多加实践，达到实验要求的真正的要求的高度。

# 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

[5] 程杰 大话数据结构 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <malloc.h>

#include<stdio.h>

#include <stdlib.h>

#include<string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status; //函数返回值状态类型定义

typedef int ElemType; //数据元素类型定义

#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;

using namespace std;

//所要使用的函数声明以供使用

int InitList(SqList &L);

int DestroyList(SqList &L);

int ClearList(SqList &L);

status ListEmpty(SqList L);

int ListLength(SqList L);

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

int LocateElem(SqList L,ElemType e);

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

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

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

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

status ListTraverse(SqList L); //单个线性表的处理

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

status LoadList(SqList &L,char FileName[]); //线性表的存储读写操作

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

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

int LocateList(LISTS Lists,char ListName[]); //多链表的操作

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

int main(void){

LISTS Lists;

Lists.length=0;

int n;

char name[30], FileName[30];;

LISTS L\_a;

SqList L; int op=1; int j;

int f,e,i,pre,next,p;

int count=0;

int flag;

L.elem=NULL; //表L置为空

while(op){

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

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

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

printf(" 1. 添加线性表 7. 获取元素的前驱\n");

printf(" 2. 清空线性表 8. 获取元素的后继\n");

printf(" 3. 线性表判空 9. 插入元素\n");

printf(" 4. 线性表长度 10. 删除元素\n");

printf(" 5. 获取元素 11. 遍历线性表\n");

printf(" 6. 查找元素 12. 保存线性表\n");

printf(" 13.加载线性表 14. 删除线性表\n");

printf(" 15.查找线性表 16.遍历多线性表\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~16]:");

scanf("%d",&op);

switch(op) {

case 1:

printf("请输入要添加的线性表的个数");

scanf("%d", &n);

while(n--)

{

printf("请输入要添加的线性表的名称\n");

scanf("%s",name);

AddList(Lists,name); //增加函数的使用

printf("线性表添加成功,请输入线性表的元素，以0结尾\n");

scanf("%d",&e);

while (e)

{

ListInsert(Lists.elem[Lists.length-1].L,Lists.elem[Lists.length-1].L.length+1,e);

scanf("%d",&e);

}

}

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

{

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

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

printf("\n");

}

getchar();

getchar();

printf("\n");

break;

case 2: // 清空线性表 done

printf("请输入你想要清空的线性表的名字");

scanf("%s",name);

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

int j = strcmp(Lists.elem[i].name,name);

if(j==0){ //匹对陈功执行删除操作

flag=ClearList(Lists.elem[i].L);

if(flag==OK) printf("已清空该线性表");

if(flag==INFEASIBLE) printf("ERROR");

break;

}

}

getchar();

getchar();

break;

case 3: //判断是否为空 done

printf("请输入线性表的名字");

scanf("%s",name);

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

int j = strcmp(Lists.elem[i].name,name);

if(j==0){ //匹对陈功执行删除操作

flag=ListEmpty(Lists.elem[i].L);

if(flag==INFEASIBLE) printf("ERROR");

if(flag==true) printf("该线性表为空");

if(flag== false) printf("该线性表不为空");

break;

}

}

getchar();

getchar();

break;

case 4: //求表长 done

printf("请输入线性表的名字");

scanf("%s",name);

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

int j = strcmp(Lists.elem[i].name,name);

if(j==0){ //匹对陈功执行删除操作

flag=ListLength(Lists.elem[i].L);

printf("此线性表的表长为：%d",flag);

break;

}

}

getchar();

getchar();

break;

case 5: //获取元素 done

printf("请输入要进行查找元素线性表的名字");

scanf("%s", name);

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

int j = strcmp(Lists.elem[i].name,name);

if(j==0){ //匹对陈功执行删除操作

printf("请输入要查找的元素的位置");

scanf("%d", &i);

j = GetElem(Lists.elem[i].L, i, e); //查找函数的使用

if (j == OK) printf("获取元素成功!位置%d的元素是%d", i, e);

if (j == ERROR) printf("ERROR");

break;

}

}

getchar();

getchar();

break;

case 6: //查找元素位置 done

printf("请输入要进行查找元素线性表的名字");

scanf("%s", name);

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

int j = strcmp(Lists.elem[i].name,name);

if(j==0){ //匹对陈功执行查找操作

printf("请输入要查找的元素");

scanf("%d", &e);

j = LocateElem(Lists.elem[i].L,e); //查找函数的使用

if (j == ERROR) printf("ERROR");

printf("查找元素成功!元素%d的位置%d",e,j);

break;

}

}

getchar();

getchar();

break;

case 7: //查找前驱元素 done

printf("请输入要进行查找元素线性表的名字");

scanf("%s", name);

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

int j = strcmp(Lists.elem[i].name,name);

if(j==0){ //匹对陈功执行查找操作

printf("请输入要查找前驱的元素");

scanf("%d", &e);

j = PriorElem(Lists.elem[i].L,e,pre); //查找函数的使用

if (j == ERROR) printf("ERROR");

printf("查找元素成功!元素%d的前驱为%d",e,pre);

break;

}

}

getchar();

getchar();

break;

case 8: //查找后继元素 done

printf("请输入要进行查找元素线性表的名字");

scanf("%s", name);

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

int j = strcmp(Lists.elem[i].name,name);

if(j==0){ //匹对陈功执行查找操作

printf("请输入要查找后继的元素");

scanf("%d", &e);

j = NextElem(Lists.elem[i].L,e,next); //查找函数的使用

if (j == ERROR) printf("ERROR");

printf("查找元素成功!元素%d的后继为%d",e,next);

break;

}

}

getchar();getchar();

break;

case 9: //插入元素 done

printf("请输入要进行插入元素线性表的名字");

scanf("%s", name);

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

int j = strcmp(Lists.elem[i].name,name);

if(j==0){ //匹对陈功执行查找操作

printf("请输入要插入的位置");

scanf("%d",&p);

printf("请输入要插入的元素");

scanf("%d",&e);

j = ListInsert(Lists.elem[i].L,p,e); //查找函数的使用

printf("%s\n", j==OK? "OK" : j==ERROR? "ERROR" : "OVERFLOW");

printf("插入后的结果为：");

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

break;

}

}

getchar();getchar();

break;

case 10: //元素的删除操作 done

printf("请输入要进行删除元素线性表的名字");

scanf("%s", name);

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

int j = strcmp(Lists.elem[i].name,name);

if(j==0){ //匹对陈功执行查找操作

printf("请输入要删除的位置");

scanf("%d",&p);

j=ListDelete(Lists.elem[i].L,p,e);

if(j==ERROR) printf("ERROR\n");

if(j==OK) printf("OK\n删除的元素为%d\n",e);

printf("删除后的结果为：");

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

break;

}

}

getchar();getchar();

break;

case 11: //线性表的遍历操作 done

printf("请输入要进行遍历线性表的名字");

scanf("%s", name);

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

int j = strcmp(Lists.elem[i].name, name);

if(j==0) {

printf("遍历之后的结果为:\n");

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

if (j == OK && !Lists.elem[i].L.length) printf("空线性表，没有元素可以展示\n");

break;

}

}

getchar();getchar();

break;

case 12: //文件保存操作 done

i=0;

printf("请输入要保存的线性表的名字");

scanf("%s",name);

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

int j = strcmp(Lists.elem[i].name,name);

if(j==0) {

printf("请输入文件所在地址：");

scanf("%s",FileName);

j = SaveList(Lists.elem[i].L, FileName);

if(j==ERROR) printf("文件打开失败");

if(j==OK){

printf("保存成功\n");

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

}

break;

}

}

if(i>=Lists.length) printf("没有相应的线性表");

getchar();getchar();

break;

case 13: //线性表加载操作

printf("请输入要加载的线性表的名字");

scanf("%s",name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) {

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

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

printf("请输入要读取的文件地址");

scanf("%s", FileName);

j = LoadList(Lists.elem[i].L, FileName);

if (j == ERROR) printf("文件读取失败");

if (j == OK) printf("文件读取成功");

break;

}

}

getchar();getchar();

break;

case 14: //删除线性表 done

printf("请输入要删除的线性表的名字");

scanf("%s",name);

if (RemoveList(Lists,name)==OK)

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

{

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

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

putchar('\n');

}

else printf("删除失败");

printf("\n");

getchar();

getchar();

break;

case 15: //查找线性表

printf("请输入要查找的线性表的名字");

scanf("%s",name);

if (n=LocateList(Lists,name))

{

printf("%s ",Lists.elem[n-1].name);

ListTraverse(Lists.elem[n-1].L);

putchar('\n');

}

else printf("查找失败\n");

getchar();getchar();

break;

case 16: //遍历多线性表

if(Lists.length==0) printf("线性表的个数为零，展示为空");

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

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

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

printf("\n");

}

getchar();

getchar();

break;

case 0:

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

return 1;

}//end of main()

// 初始化线性表,构造一个空线性表

int InitList(SqList &L) //case 1

{

if(L.elem)

return INFEASIBLE;

else{

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

if (!L.elem)

return OVERFLOW;

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

return OK;

}

} //case1

//初始条件：线性表已存在,操作结果：销毁线性表

int DestroyList(SqList &L){

if(!L.elem)

return INFEASIBLE;

else{

free(L.elem);

L.elem=NULL;

L.length=0;

L.listsize=0;

return OK;

}

}

//case 2

/\*初始条件：线性表存在;操作结果：清空线性表

（将当前元素个数赋值0，遍历不出任何一个元素，相当于清空线性表）\*/

status ClearList(SqList &L){

if(!L.elem)

return INFEASIBLE;

else{

L.length=0;

return OK;

}

}

//case 3

/\*初始条件：线性表存在

操作结果：线性表为空返回true，不为空返回false\*/

status ListEmpty(SqList L){

if(!L.elem)

return INFEASIBLE;

else{

return (L.length == 0) ? true : false;

}

}

//case 4

//获取线性表当前元素个数

//初始条件：线性表已存在

//操作结果：返回线性表当前元素个数

int ListLength(SqList L){

if(!L.elem)

return INFEASIBLE;

else

return L.length;

}

//case 5

//获得指定位置的数据元素

//初始条件：线性表存在

//操作结果：获得指定位置的数据元素并赋值给e

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

if(!L.elem) return INFEASIBLE;

else{

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

return ERROR;

e=\*(L.elem+i-1);//（基址+i-1）即为第i个元素的地址

return OK;

}

}

//case 6

//定位元素（获得符合一定条件的数据元素的位序）

//初始条件：线性表已存在

//操作结果：返回L中第一个与e满足关系的元素的位序,若不存在返回0

int LocateElem(SqList L,ElemType e) {

if(!L.elem) return INFEASIBLE;

else{

ElemType \*p = L.elem; //P的初值为第一个元素的存储位置

int i = 1;//i的初值为第一个元素的位序

while (i <= L.length&&p[i-1]!=e)//越界或已找到满足条件的元素

++i;

//i的最大可能值为L.length+1

{

if (i <= L.length)

return i;

else

return 0;

}

}

}

//case 7

//返回要查找的位置元素的前驱

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

{

if(!L.elem) return INFEASIBLE;

int i=2;

ElemType \*p=L.elem+1;

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

{

p++;

i++;

}

if(i>L.length)

return ERROR;

pre=\*--p;

return OK;

}

//case 8

//初始条件：线性表已存在，数据元素存在后继

//操作结果：查找数据元素，若线性表中有该元素且后继存在，将后继拷贝给一个与数据元素数据类型相同的变量；若后继不存在，上述变量无定义

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

{

if (!L.elem) return INFEASIBLE;

int i;

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

if (L.elem[i] == e) { next = L.elem[i + 1]; return OK; }

}

return ERROR;

}

//case 9

//插入一个数据元素

//初始条件：线性表存在

//操作结果：在L中第i个元素之前插入新的元素e，L的长度加1

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

{

if(!L.elem) return INFEASIBLE;

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

if(i<1||i>L.length+1)//i值不合法，无法执行插入操作

return ERROR;

if(L.length>=L.listsize) //当前存储空间已满，增加分配空间

{

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

exit(0);//分配存储空间失败

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)

{ \*(p+1)=\*p;} //给插入位置之后的元素赋值达到之后元素后移一位的效果

\*q=e;//插入e

L.length+=1;

return OK;

}

//case10

//删除元素

//初始条件：线性表已存在

//操作结构：删除第i个数据元素并返回其值，L的长度减1

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) return INFEASIBLE;

ElemType \*p, \*q;

if (i < 1 || i > L.length)//i值不合法

return ERROR;

p = L.elem + i - 1;//p为被删除元素的位置

e = \*p;//被删除元素的值赋给e

q = L.elem + L.length - 1;//表尾元素的位置

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

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

L.length--;

return OK;

}

//case 11

status ListTraverse(SqList L) //case 11

{

if(!L.elem) return INFEASIBLE;

ElemType \*p=L.elem;

int i;

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

printf("%d ",p[i]);

return OK;

}

status SaveList(SqList L,char FileName[]) //case 12

// 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) return INFEASIBLE;

FILE \*fp;

if ((fp=fopen(FileName,"wb"))==NULL)

{

printf("File open error\n ");

return ERROR;

}

fwrite(L.elem,sizeof(ElemType),L.length,fp);

fclose(fp);

return OK;

}

status LoadList(SqList &L,char FileName[]) //case 13

// 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。

{

if(L.elem) return INFEASIBLE;

FILE \*fp;

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

L.length = 0;

L.listsize = LIST\_INIT\_SIZE;

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

{

printf("File open error\n ");

return ERROR;

}

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

L.length++;

fclose(fp);

return OK;

}

status AddList(LISTS &Lists,char ListName[]) //case 1

// 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 i=0;

Lists.elem[Lists.length].L.elem = (ElemType \*)malloc(LIST\_INIT\_SIZE \* sizeof(int));

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

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

if (!Lists.elem[Lists.length].L.elem)

return OVERFLOW;

while (ListName[i]!='\0'){

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

i++;

}

Lists.elem[Lists.length].name[i]='\0';

Lists.length++;

return OK;

}

status RemoveList(LISTS &Lists,char ListName[]) //case 14

// Lists中删除一个名称为ListName的线性表

{

int i;

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

int j = strcmp(Lists.elem[i].name,ListName);

if(j==0){ //匹对陈功执行删除操作

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

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

break;

}

}

if(i==Lists.length) return ERROR;

Lists.length--;

return OK;

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

}

int LocateList(LISTS Lists,char ListName[]) //case 15

// 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;

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

int j=strcmp(Lists.elem[i-1].name,ListName);

if (j==0)

return i;

}

return ERROR;

}

# 附录B 基于链式存储结构线性表实现的源程序

#include "stdio.h"

#include "stdlib.h"

#include "string.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 struct LNode { //线性表表（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[10];

int length;

int listsize;

} LISTS;

using namespace std;

//所要使用的函数声明以供使用

int InitList(LinkList &L);

status DestroyList(LinkList &L);

int ClearList(LinkList &L);

status ListEmpty(LinkList L);

int ListLength(LinkList L);

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

int LocateElem(LinkList L, ElemType e);

status PriorElem(LinkList L, ElemType cur\_e, ElemType &pre\_e);

status NextElem(LinkList L, ElemType cur\_e, ElemType &next\_e);

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

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

status ListTraverse(LinkList L); //单个线性表的处理

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

status LoadList(LinkList &L, char FileName[]); //线性表的存储读写操作

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

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

int LocateList(LISTS Lists, char ListName[]); //多链表的操作

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

int main(void) {

LISTS Lists;

Lists.length = 0;

int n;

int opt = 0;

char name[30];

char FileName[30];

LinkList L, L1, s;

int op = 1;

int j;

int f, e, i, pre, next, p;

int count = 0;

int flag;

L = NULL; //表L置为空

while (op) {

system("cls");

printf("\n\n");

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

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

printf(" 1. 添加线性表 7. 获取元素的前驱\n");

printf(" 2. 清空线性表 8. 获取元素的后继\n");

printf(" 3. 线性表判空 9. 插入元素\n");

printf(" 4. 线性表长度 10. 删除元素\n");

printf(" 5. 获取元素 11. 遍历线性表\n");

printf(" 6. 查找元素 12. 保存线性表\n");

printf(" 13.加载线性表 14. 删除线性表\n");

printf(" 15.查找线性表 16.遍历多线性表\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~16]:");

scanf("%d", &op);

switch (op) {

case 1:

printf("请输入要添加的线性表的个数");

scanf("%d", &n);

while (n--) {

printf("请输入要添加的线性表的名称\n");

scanf("%s", name);

AddList(Lists, name); //增加函数的使用

printf("线性表添加成功,请输入线性表的元素，以0结尾\n");

scanf("%d", &e);

i = 0;

while (e) {

ListInsert(Lists.elem[Lists.length - 1].L, i + 1, e);

scanf("%d", &e);

i++;

}

}

for (n = 0; n < Lists.length; n++) {

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

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

printf("\n");

}

getchar();

printf("\n");

break;

case 2: // 清空线性表 done

printf("请输入你想要清空的线性表的名字");

scanf("%s", name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) { //匹对陈功执行删除操作

flag = ClearList(Lists.elem[i].L);

if (flag == ERROR) printf("该线性表已经为空！");

if (flag == OK) printf("已清空该线性表");

if (flag == INFEASIBLE) printf("ERROR");

break;

}

}

getchar();

getchar();

break;

case 3: //判断是否为空 done

printf("请输入线性表的名字");

scanf("%s", name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) { //匹对陈功执行删除操作

flag = ListEmpty(Lists.elem[i].L);

if (flag == INFEASIBLE) printf("ERROR");

if (flag == true) printf("该线性表为空");

if (flag == false) printf("该线性表不为空");

break;

}

}

getchar();

getchar();

break;

case 4: //求表长 done

printf("请输入线性表的名字");

scanf("%s", name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) { //匹对陈功执行删除操作

flag = ListLength(Lists.elem[i].L);

printf("此线性表的表长为：%d", flag);

break;

}

}

getchar();

getchar();

break;

case 5: //查找元素 done

printf("请输入要进行查找元素线性表的名字");

scanf("%s", name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) { //匹对陈功执行删除操作

printf("请输入要查找的元素的位置");

scanf("%d", &p);

j = GetElem(Lists.elem[i].L, p, e); //查找函数的使用

if (j == OK) printf("获取元素成功!位置%d的元素是%d", p, e);

if (j == ERROR) printf("ERROR");

break;

}

}

getchar();

getchar();

break;

case 6: //查找元素位置 done

printf("请输入要进行查找元素线性表的名字");

scanf("%s", name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) { //匹对陈功执行查找操作

printf("请输入要查找的元素");

scanf("%d", &e);

j = LocateElem(Lists.elem[i].L, e); //查找函数的使用

if (j == ERROR) printf("ERROR");

else printf("查找元素成功!元素%d的位置%d", e, j);

break;

}

}

getchar();

getchar();

break;

case 7: //查找前驱元素 done

printf("请输入要进行查找前驱元素线性表的名字");

scanf("%s", name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) { //匹对陈功执行查找操作

printf("请输入要查找前驱的元素");

scanf("%d", &e);

j = PriorElem(Lists.elem[i].L, e, pre); //查找函数的使用

if (j == ERROR) printf("ERROR");

else printf("查找元素成功!元素%d的前驱为%d", e, pre);

break;

}

}

getchar();

getchar();

break;

case 8: //查找后继元素 done

printf("请输入要进行查找后继元素线性表的名字");

scanf("%s", name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) { //匹对陈功执行查找操作

printf("请输入要查找后继的元素");

scanf("%d", &e);

j = NextElem(Lists.elem[i].L, e, next); //查找函数的使用

if (j == ERROR) printf("ERROR");

else printf("查找元素成功!元素%d的后继为%d", e, next);

break;

}

}

getchar();

getchar();

break;

case 9: //插入元素 done

printf("请输入要进行插入元素线性表的名字");

scanf("%s", name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) { //匹对陈功执行查找操作

printf("请输入要插入的位置");

scanf("%d", &p);

printf("请输入要插入的元素");

scanf("%d", &e);

j = ListInsert(Lists.elem[i].L, p, e); //查找函数的使用

printf("%s\n", j == OK ? "OK" : j == ERROR ? "ERROR" : "OVERFLOW");

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

break;

}

}

getchar();

getchar();

break;

case 10: //元素的删除操作 done

printf("请输入要进行删除元素线性表的名字");

scanf("%s", name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) { //匹对陈功执行查找操作

printf("请输入要删除的位置");

scanf("%d", &p);

j = ListDelete(Lists.elem[i].L, p, e);

if (j == ERROR) printf("ERROR\n");

if (j == OK) printf("OK\n%d\n", e);

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

break;

}

}

getchar();

getchar();

break;

case 11: //线性表的遍历操作 done

printf("请输入要进行遍历线性表的名字");

scanf("%s", name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) {

printf("遍历之后的结果为:\n");

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

if (j == OK && !Lists.elem[i].L->next) printf("空线性表\n");

break;

}

}

getchar();

getchar();

break;

case 12: //文件保存 done

printf("请输入要保存的线性表的名字");

scanf("%s", name);

printf("请输入文件存储的地址");

scanf("%s", FileName);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) {

j = SaveList(Lists.elem[i].L, FileName);

printf("保存成功");

break;

}

}

if (i >= Lists.length) printf("没有相应的线性表");

getchar();

getchar();

break;

case 13:

printf("请输入要读取的文件地址");

scanf("%s", FileName);

printf("请输入要保存的线性表的名字");

scanf("%s", name);

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

j = strcmp(Lists.elem[i].name, name);

if (j == 0) {

while (Lists.elem[i].L) {

s = Lists.elem[i].L->next;

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

Lists.elem[i].L = s;

}

j = LoadList(Lists.elem[i].L, FileName);

if (j == OK) printf("读取成功");

}

break;

}

for (s = Lists.elem[i].L->next; s; s = s->next)

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

getchar();

getchar();

break;

case 14: //删除线性表 done

printf("请输入要删除的线性表的名字");

scanf("%s", name);

if (RemoveList(Lists, name) == OK) printf("删除成功请使用功能16查看删除后的结果\n");

else printf("删除失败");

getchar();

getchar();

break;

case 15: //查找线性表

printf("请输入要查找的线性表的名字");

scanf("%s", name);

if (n = LocateList(Lists, name)) {

printf("查找成功%s ", Lists.elem[n - 1].name);

ListTraverse(Lists.elem[n - 1].L);

} else printf("查找失败\n");

getchar();

getchar();

break;

case 16:

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

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

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

printf("\n");

}

if (Lists.length == 0) printf("多链表为空！");

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 1;

}//end of main()

// 初始化线性表,构造一个空线性表

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) return INFEASIBLE;

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

L->next = NULL;

return OK;

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

} //case1

//初始条件：线性表已存在,操作结果：销毁线性表

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) return INFEASIBLE;

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

while (q != NULL) {

free(p);

p = q;

q = p->next;

}

free(p);

L = NULL;

return OK;

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

}

//case 2

/\*初始条件：线性表存在;操作结果：清空线性表

（将当前元素个数赋值0，遍历不出任何一个元素，相当于清空线性表）\*/

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) return INFEASIBLE;

LinkList p = L->next, q;

if (!p) return ERROR;

while (p != NULL) {

q = p->next;

free(p);

p = q;

}

L->next = NULL;

return OK;

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

}

//case 3

/\*初始条件：线性表存在

操作结果：线性表为空返回true，不为空返回false\*/

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) return INFEASIBLE;

if (L->next == NULL)

return true;

else return false;

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

}

//case 4

//获取线性表当前元素个数

//初始条件：线性表已存在

//操作结果：返回线性表当前元素个数

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

int i = 0;//用于计数

if (!L) return INFEASIBLE;

LinkList p = L;

while (p->next != NULL) {

i++;

p = p->next;

}

return i;

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

}

//case 5

//获得指定位置的数据元素

//初始条件：线性表存在

//操作结果：获得指定位置的数据元素并赋值给e

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) return INFEASIBLE;

LinkList p = L->next;

int j = 1;

while (p && j < i) {

p = p->next;

++j;

}

if (!p || j > i) return ERROR;

e = p->data;

return OK;

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

}

//case 6

//定位元素（获得符合一定条件的数据元素的位序）

//初始条件：线性表已存在

//操作结果：返回L中第一个与e满足关系的元素的位序,若不存在返回0

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) return INFEASIBLE;

int i = 0;

LinkList p = L->next;

while (p) {

i++;

if (p->data == e) /\* 找到这样的数据元素 \*/

return i;

p = p->next;

}

return 0;

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

}

//case 7

//返回要查找的位置元素的前驱

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) return INFEASIBLE;

if (L->next == NULL) return ERROR;

if (L->next->data == e) return ERROR;

LinkList p = L->next;

while (p->next) {

if (p->next->data == e) {

pre = p->data;

return OK;

}

p = p->next;

}

return ERROR;

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

} //case 8

//初始条件：线性表已存在，数据元素存在后继

//操作结果：查找数据元素，若线性表中有该元素且后继存在，将后继拷贝给一个与数据元素数据类型相同的变量；若后继不存在，上述变量无定义

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) return INFEASIBLE;

if (L->next == NULL) return ERROR;

LinkList p = L->next; //此时p为第一个节点

while (p->next) //判断p是否是最后一个节点 如果p是最后一个节点则跳出循环 返回ERROR

{

if (p->data == e) //匹配节点 匹配成功返回后继

{

next = p->next->data;

return OK;

}

p = p->next;

}

return ERROR;

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

}

//case 9

//插入一个数据元素

//初始条件：线性表存在

//操作结果：在L中第i个元素之前插入新的元素e，L的长度加1

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) return INFEASIBLE;

if (i <= 0) return ERROR;

LinkList p = L;

int j = 0;

while (p && j < i - 1) {

p = p->next;

if (j > i || !p) return ERROR;

j++;

}

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

s->data = e;

s->next = p->next;

p->next = s;

return OK;

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

}

//case10

//删除元素

//初始条件：线性表已存在

//操作结构：删除第i个数据元素并返回其值，L的长度减1

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) return INFEASIBLE;

LinkList p = L->next, q;

int l = 0, j = 1;

while (p) //求线性表的长度

{

l++;

p = p->next;

}

p = L->next;

if (p == NULL) return ERROR; //如果线性表为空 返回错误

if (i < 1 || i > l) return ERROR; //如果i不合法返回错误

//执行删除元素的操作

for (j = 1; j < i - 1; j++) p = p->next;

q = p;

p = p->next; //获取要删除的节点p及它的前继q

e = p->data; //取要删除的节点的元素

q->next = p->next; //删除节点

free(p);

return OK;

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

}

//case 11

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) return INFEASIBLE;

LinkList p = L->next;

while (p) {

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

p = p->next;

}

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) return INFEASIBLE;

int length = 0;

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

while (s) {

length++;

s = s->next;

}

FILE \*fp;

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

printf("File open error\n ");

return 1;

}

while (length--) {

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

p = p->next;

}

fclose(fp);

return OK;

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

}

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) return INFEASIBLE;

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

L->next = NULL;

LNode \*s, \*r = L;

FILE \*fp;

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

printf("File open error\n ");

return 1;

}

s = (LNode \*) malloc(sizeof(LNode));

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

r->next = s;

r = s;

s = (LNode \*) malloc(sizeof(LNode));

}

r->next = NULL;

fclose(fp);

return OK;

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

}

status AddList(LISTS &Lists, char ListName[]) //case 14

// 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 i = 0;

Lists.elem[Lists.length].L = (LinkList) malloc(sizeof(LNode));

Lists.elem[Lists.length].L->next = NULL;

if (!Lists.elem[Lists.length].L)

return OVERFLOW;

while (ListName[i] != '\0') {

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

i++;

}

Lists.elem[Lists.length].name[i] = '\0';

Lists.length++;

return OK;

}

status RemoveList(LISTS &Lists, char ListName[]) //case 15

// Lists中删除一个名称为ListName的线性表

{

int i,n;

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

int j = strcmp(Lists.elem[i].name, ListName);

if (j == 0) { //匹对陈功执行删除操作

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

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

break;

}

}

if (i == Lists.length) return ERROR;

Lists.length--;

return OK;

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

}

int LocateList(LISTS Lists, char ListName[]) //case 16

// 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;

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

int j = strcmp(Lists.elem[i - 1].name, ListName);

if (j == 0)

return i;

}

return ERROR;

}

# 附录C 基于二叉链表二叉树实现的源程序

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

#include "ctype.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; //整型

char others[20]; //字符型

} TElemType; //二叉树结点类型定义

typedef struct BiTNode { //二叉链表的定义

TElemType data;

struct BiTNode \*lchild, \*rchild;

} BiTNode, \*BiTree;

typedef struct { //多二叉树存储结构定义

struct {

char name[30];

BiTree T;

} elem[100];

int length;

int listsize;

} wood;

typedef struct { //节点在完全二叉树位置及节点的数据

int pos;

TElemType data;

} DEF;

void visit(BiTree T) {

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

}

status CreateBiTree\_nor(BiTree &T, DEF definition[]) {

int i = 0, j;

static BiTNode \*p[100];

while (j = definition[i].pos) {

p[j] = (BiTNode \*) malloc(sizeof(BiTNode));

p[j]->data = definition[i].data;

p[j]->lchild = NULL;

p[j]->rchild = NULL;

if (j != 1)

if (j % 2) p[j / 2]->rchild = p[j];

else p[j / 2]->lchild = p[j];

i++;

}

T = p[1];

return OK;

}

//函数声明

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

status ClearBiTree(BiTree &T);

status TreeEmpty(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);

status DeleteNode(BiTree &T, KeyType e);

//三种排序至少有一种试非递归的

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 SaveBiTree(BiTree T, char FileName[]);

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

status Addtrees(wood &Trees, char name[]);

status TreesTraverse(wood trees);

status DestroyBiTree(BiTree &T);

int main(void) {

wood Trees; //森林

Trees.length = 0;

BiTree T=NULL, node = NULL; //根节点与节点

TElemType definition[100]; //节点数据数组

DEF definition\_nor[100];

TElemType e; //节点数据--查找兄弟节点用

int key; //节点的关键字

int en;

int LR; //左右子树的状态值

int n; //添加二叉树数目的临时变量

int ans; //函数返回状态值

char name[30];

char FileName[30];

int op = 1;

int j,f, i, p;;

int count = 0;

int flag;

while (op) {

system("cls");

printf("\n\n");

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

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

printf(" 1. 添加二叉树 9. 删除节点\n");

printf(" 2. 清空二叉树 10. 前序遍历\n");

printf(" 3. 判空二叉树 11. 中序遍历\n");

printf(" 4. 二叉树深度 12. 后序遍历\n");

printf(" 5. 查找结点 13. 层序遍历\n");

printf(" 6. 节点赋值 14. 保存二叉树\n");

printf(" 7. 获得兄弟节点 15. 加载二叉树\n");

printf(" 8. 插入节点 16. 销毁二叉树\n");

printf(" 17.\*遍历森林\* 0. \*Exit\*\n");

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

printf(" 请选择你的操作[0~17]:");

scanf("%d", &op);

switch (op) {

case 1:

printf("当前操作：添加二叉树\n");

printf("请输入要添加的二叉树的个数");

scanf("%d", &n);

while (n--) {

printf("请输入要添加的二叉树的名称\n");

scanf("%s", name);

Addtrees(Trees, name); //增加函数的使用

printf("二叉树添加成功,请以完全二叉树的位置输入节点，要求带有关键字和字符，以0 0 null结束\n");

i = 0;

do {

scanf("%d%d%s", &definition\_nor[i].pos, &definition\_nor[i].data.key,

definition\_nor[i].data.others);

} while (definition\_nor[i++].pos);

//创建空树

if (definition\_nor[0].data.key == 0) {

Trees.elem[Trees.length - 1].T = NULL;

continue;

}

ans = CreateBiTree\_nor(Trees.elem[Trees.length - 1].T, definition\_nor);

}

//遍历多线性表

TreesTraverse(Trees);

printf("\n");

getchar();

getchar();

break;

case 2: // 清空线性表 done

printf("当前操作：情空二叉树\n");

printf("请输入二叉树的名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) { //匹对陈功执行删除操作

flag = ClearBiTree(Trees.elem[i].T);

if (flag == ERROR) printf("该二叉树已经为空！");

if (flag == OK) printf("已清空该二叉树");

if (flag == INFEASIBLE) printf("ERROR");

break;

}

}

getchar();

getchar();

break;

case 3: //判断是否为空 done

printf("当前操作：判断空树\n");

printf("请输入二叉树的名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) { //匹对陈功执行删除操作

flag = TreeEmpty(Trees.elem[i].T);

if (flag == INFEASIBLE) printf("ERROR");

if (flag == true) printf("该二叉树为空");

if (flag == false) printf("该二叉树不为空");

break;

}

}

getchar();

getchar();

break;

case 4: //求二叉树的深度 done

printf("当前操作：求深度\n");

printf("请输入二叉树的名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) { //匹对陈功执行删除操作

flag = BiTreeDepth(Trees.elem[i].T);

printf("此二叉树的深度为：%d", flag);

break;

}

}

getchar();

getchar();

break;

case 5: //查找节点 done

printf("当前操作：查找节点\n");

printf("请输入二叉树的名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) { //匹对陈功执行删除操作

printf("请输入要查找的节点的位置");

scanf("%d", &key);

node = LocateNode(Trees.elem[i].T, key); //查找函数的使用

if (node) printf("获取元素成功!关键字%d的节点是%s", key, node->data.others);

if (!node) printf("ERROR");

break;

}

}

getchar();

getchar();

break;

case 6: //节点赋值 done

printf("当前操作：节点节点赋值\n");

printf("请输入要进行操作的二叉树名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) { //匹对陈功执行查找操作

printf("请输入新的节点的相关参数");

scanf("%d%d%s", &key, &e.key, e.others); //10 11 h

ans = Assign(Trees.elem[i].T, key, e);

if (ans == ERROR) printf("ERROR");

else {

printf("赋值成功!\n");

PreOrderTraverse(Trees.elem[i].T, visit);

printf("\n");

InOrderTraverse(Trees.elem[i].T, visit);

}

break;

}

}

getchar();

getchar();

break;

case 7: //获得兄弟节点

printf("当前操作：获得兄弟节点\n");

printf("请输入要进行操作的二叉树名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) { //匹对陈功执行查找操作

printf("请输入节点的位置");

scanf("%d", &en);

node = GetSibling(Trees.elem[i].T, en); //查找函数的使用

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

else printf("ERROR");

break;

}

}

getchar();

getchar();

break;

case 8: //插入节点 done

printf("当前操作：插入节点\n");

printf("请输入要进行操作的二叉树名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) { //匹对陈功执行查找操作

printf("请输入插入节点的关键字以及新节点的相关性质\n");

printf("en--插入的节点关键字, LR--插入后的位置, e.key--新节点关键字, e.others--新节点其他\n");

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

ans = InsertNode(Trees.elem[i].T, en, LR, e);

if (ans == OK) {

PreOrderTraverse(Trees.elem[i].T, visit);

printf("\n");

InOrderTraverse(Trees.elem[i].T, visit);

} else printf("ERROR");

break;

}

}

getchar();

getchar();

break;

case 9: //删除元素 done

printf("当前操作：删除元素\n");

printf("请输入线性表的名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) { //匹对陈功执行查找操作

printf("请输入删除结点的关键字");

scanf("%d", &en);

ans = DeleteNode(Trees.elem[i].T, en);

if (ans == OK) {

PreOrderTraverse(Trees.elem[i].T, visit);

printf("\n");

InOrderTraverse(Trees.elem[i].T, visit);

} else printf("删除操作失败");

break;

}

}

getchar();

getchar();

break;

case 10: //前序遍历 done

printf("当前操作：先序遍历\n");

printf("请输入二叉树的名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) { //匹对陈功执行查找操作

PreOrderTraverse(Trees.elem[i].T, visit);

break;

}

}

getchar();

getchar();

break;

case 11: //中序遍历操作 done

printf("当前操作：中序遍历\n");

printf("请输入二叉树的名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) {

InOrderTraverse(Trees.elem[i].T, visit);

break;

}

}

getchar();

getchar();

break;

case 12: //后序遍历操作 done

printf("当前操作：后序遍历\n");

printf("请输入二叉树的名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) {

PostOrderTraverse(Trees.elem[i].T, visit);

break;

}

}

getchar();

getchar();

break;

case 13: //层序遍历操作 done

printf("当前操作：层序遍历\n");

printf("请输入二叉树的名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) {

LevelOrderTraverse(Trees.elem[i].T, visit);

break;

}

}

getchar();

getchar();

break;

case 14: //保存二叉树 done

printf("当前操作：保存二叉树\n");

printf("请输入要保存的二叉树的名字");

scanf("%s", name);

printf("请输入文件存储的地址");

scanf("%s", FileName);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) {

j = SaveBiTree(Trees.elem[i].T, FileName);

printf("保存成功");

break;

}

}

if (i >= Trees.length) printf("没有相应的二叉树");

getchar();

getchar();

break;

case 15: //加载二叉树

printf("当前操作：加载二叉树\n");

printf("请输入要读取的文件地址");

scanf("%s", FileName);

printf("请输入要加载的二叉树名字");

scanf("%s", name);

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) {

DestroyBiTree(Trees.elem[i].T);

j = LoadBiTree(Trees.elem[i].T, FileName);

if (j == OK) printf("读取成功");

}

break;

}

if (i >= Trees.length) printf("没有相应的二叉树");

//遍历检验

getchar();

getchar();

break;

case 16: //销毁二叉树

printf("当前操作：销毁线性表\n");

printf("请输入要销毁的二叉树的名字");

scanf("%s", name);

if (Trees.length == 0) printf("没有可供删除的二叉树");

for (i = 0; i < Trees.length; i++) {

j = strcmp(Trees.elem[i].name, name);

if (j == 0) {

DestroyBiTree(Trees.elem[i].T);

free(Trees.elem[i].name);

Trees.length--;

while (i < Trees.length) {

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

i++;

}

}

printf("销毁成功，可以遍历森林查看销毁结果\n");

}

getchar();

getchar();

break;

case 17: //遍历多二叉树

printf("当前操作：遍历二叉树\n");

if (Trees.length == 0) printf("森林为空");

for (i = 0; i < Trees.length; i++) {

printf("名字：%s\n", Trees.elem[i].name);

PreOrderTraverse(Trees.elem[i].T, visit);

printf("\n");

InOrderTraverse(Trees.elem[i].T, visit);

printf("\n");

}

getchar();

getchar();

break;

case 0:

exit(0);

break;

}//end of switch

}//end of while

printf("欢迎下次再使用本系统！\n");

return 1;

}//end of main()

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

int flag;

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

static int i = 0, count = definition[0].key;

i++;

if (definition[i - 1].key == 0) T = NULL;

else if (definition[i - 1].key != 0) {

if (count != definition[i - 1].key) return ERROR;

T->data.key = definition[i - 1].key;

strcpy(T->data.others, definition[i - 1].others);

count++;

flag = CreateBiTree(T->lchild, definition);

flag = CreateBiTree(T->rchild, definition);

if (flag == ERROR) return ERROR;

}

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) return ERROR;

if (T) {

ClearBiTree(T->lchild);

ClearBiTree(T->rchild);

free(T);

T = NULL;

}

return OK;

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

}

status TreeEmpty(BiTree &T) {

if (T == NULL) return true;

else return false;

}

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

if (!T) return ERROR;

int L\_d = BiTreeDepth(T->lchild);

int R\_d = BiTreeDepth(T->rchild);

return (((L\_d > R\_d) ? L\_d : R\_d) + 1);

/\*\*\*\*\*\*\*\*\*\* 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 n\_tree = NULL;

if (T) {

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

n\_tree = T;

return n\_tree;

}

n\_tree = LocateNode(T->lchild, e);

if (n\_tree&&e==n\_tree->data.key) return n\_tree;

n\_tree = LocateNode(T->rchild, e);

}

return n\_tree;

/\*\*\*\*\*\*\*\*\*\* 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 tr=T;

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

//遍历是否重复

//错误分类 完成赋值之后重复

BiTree node;

int flag=0,status=0;

BiTree stack[100];

int top = 0;

node = T;

while (node != NULL || top > 0) {

while (node != NULL) {

stack[top++] = node;

node = node->lchild;

}

node = stack[--top];

if(e!=value.key&&node->data.key==value.key) return ERROR;

node = node->rchild;

}

// 赋值

t = LocateNode(T, e);

if (!t) return ERROR;

t->data = value;

//判断赋值之后是否重复

return OK;

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

/\*

\* 遍历 找到之后

\* 父节点的左右节点

\*/

//非递归中序遍历 实现 栈

BiTree stack[100];

int top = 0;

BiTree t = T;

while (t != NULL || top > 0) {

while (t != NULL) {

stack[top++] = t;

t = t->lchild;

}

t = stack[--top];

if (t->lchild && t->lchild->data.key == e && t->rchild) return t->rchild;

if (t->rchild && t->rchild->data.key == e && t->lchild) return t->lchild;

t = t->rchild;

}

return ERROR;

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

}

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

BiTree new\_node = (BiTree) malloc(sizeof(BiTNode)), node;

new\_node->data = c;

int flag=0,status=0;

BiTree stack[100];

int top = 0;

node = T;

//遍历是否重复

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

/\*

\*

\* while (node != NULL || top > 0) {

while (node != NULL) {

stack[top++] = node;

node = node->lchild;

}

node = stack[--top];

if(node->data.key==c.key) return ERROR;

node = node->rchild;

}

\*/

if (LR == -1) {

new\_node->rchild = T;

new\_node->lchild = NULL;

T = new\_node;

return OK;

}

/\*

\* LR 0 T 1R -1 ROOT

\* e->lr==c->r

\* e是序号

\* c是数据

\*/

/\*

\* 查找节点

\* 插入

\*/

node=T; top=0;

while (node != NULL || top > 0) {

while (node != NULL) {

stack[top++] = node;

node = node->lchild;

}

node = stack[--top];

if (node->data.key == e && node) {

if (LR == 0) {

new\_node->rchild = node->lchild;

new\_node->lchild = NULL;

node->lchild = new\_node;

} else {

new\_node->rchild = node->rchild;

new\_node->lchild = NULL;

node->rchild = new\_node;

}

status=1; //插入完成

}

node = node->rchild;

}

if(status==1) 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 \*\*\*\*\*\*\*\*\*/

//删除头节点

//查找父节点

//删除节点

//释放内存

//删除情况分类：

/\*

\* 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。

\*/

BiTree node, old\_node, cur\_node;

BiTree stack[100];

int top = 0;

node = T;

int flag = 0;

while (node != NULL || top > 0) {

while (node != NULL) {

stack[top++] = node;

node = node->lchild;

}

node = stack[--top];

//考虑根节点的情况

if (node->data.key == e && top == 0) {

old\_node = node;

node = node->lchild;

cur\_node = node;

while (cur\_node->rchild) cur\_node = cur\_node->rchild;

cur\_node->rchild = old\_node->rchild;

free(old\_node);

old\_node = NULL;

T = node;

return OK;

}

//考虑子树的情况

//查找父节点成功 执行删除操作

if ((node->lchild && node->lchild->data.key == e) || node->rchild && node->rchild->data.key == e) {

//为左子树

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

cur\_node = node->lchild;

//度为0

if (!(cur\_node->lchild || cur\_node->rchild)) {

free(cur\_node);

node->lchild = NULL;

flag = 1;

}

//度为2

else if (cur\_node->lchild && cur\_node->rchild) {

old\_node = cur\_node;

node->lchild = cur\_node->lchild;

cur\_node=cur\_node->lchild;

while (cur\_node->rchild) cur\_node = cur\_node->rchild;

cur\_node->rchild = old\_node->rchild;

free(old\_node);

old\_node = NULL;

flag = 1;

}

//度为1

else {

if (cur\_node->lchild) {

old\_node = cur\_node;

node->lchild = cur\_node->lchild; //两个if语句可以使用 ?: 代替

free(old\_node);

} else {

old\_node = cur\_node;

node->lchild = cur\_node->rchild;

free(old\_node);

}

flag = 1;

}

return OK;

}

//为右子树

if (node->rchild) {

cur\_node = node->rchild;

//度为0

if (!(cur\_node->lchild && cur\_node->rchild)) {

free(cur\_node);

node->rchild = NULL;

flag = 1;

}

//度为2

else if (cur\_node->lchild && cur\_node->rchild) {

old\_node = cur\_node;

node->rchild = cur\_node->lchild;

cur\_node=cur\_node->lchild;

while (cur\_node->rchild) cur\_node = cur\_node->rchild;

cur\_node->rchild = old\_node->rchild;

free(old\_node);

old\_node = NULL;

flag = 1;

}

//度为1

else {

if (cur\_node->lchild) {

old\_node = cur\_node;

node->rchild = cur\_node->lchild; //两个if语句可以使用 ?: 代替

free(old\_node);

} else {

old\_node = cur\_node;

node->rchild = cur\_node->lchild;

free(old\_node);

}

flag = 1;

}

return OK;

}

}

if (flag == 1) return OK;

node = node->rchild;

}

return ERROR;

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

}

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);

PreOrderTraverse(T->lchild, visit);

PreOrderTraverse(T->rchild, visit);

}

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

//非递归

BiTree node;

BiTree stack[100];

int top = 0;

node = T;

while (node != NULL || top > 0) {

while (node != NULL) {

stack[top++] = node;

node = node->lchild;

}

node = stack[--top];

visit(node);

node = node->rchild;

}

return OK;

//递归

/\*

if (T) {

InOrderTraverse(T->lchild, visit);

visit(T);

InOrderTraverse(T->rchild, visit);

}

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

{

// 请在这里补充代码，完成本关任务

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

//队列先进先出

//出的时候将其左右孩子放入队尾

BiTree node;

BiTree queue[100];

int in = 0;

int out = 0;

queue[in++] = T;

while (in > out) {

if (queue[out]) {

visit(queue[out]);

queue[in++] = queue[out]->lchild;

queue[in++] = queue[out]->rchild;

}

out++;

}

return OK;

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

}

status RecurSave(BiTree T, FILE \*fp) {

BiTree b\_node; //空白节点

char b[] = "null";

b\_node = (BiTree) malloc(sizeof(BiTNode));

b\_node->data.key = 0;

strcpy(b\_node->data.others, b);

if (T) {

fwrite(&T->data, sizeof(TElemType), 1, fp);

if (RecurSave(T->lchild, fp))

if (RecurSave(T->rchild, fp))

return OK;

else

return ERROR;

} else {

fwrite(&b\_node->data, sizeof(TElemType), 1, fp);

return OK;

}

}

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;

fp = fopen(FileName, "wb");

if (fp == NULL)

return ERROR;

if (RecurSave(T, fp) == OK) {

fclose(fp);

return OK;

} else {

fclose(fp);

return ERROR;

}

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

}

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;

TElemType d[100];

int i = 0;

int s;

fp = fopen(FileName, "rb");

if (fp == NULL)

return ERROR;

else {

while (fread(&d[i], sizeof(TElemType), 1, fp)) i++;

s = CreateBiTree(T, d);

if (s) return OK;

return ERROR;

}

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

}

status Addtrees(wood &Trees, char TreeName[])

// 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 i = 0;

Trees.elem[Trees.length].T = (BiTree) malloc(sizeof(BiTNode));

Trees.elem[Trees.length].T->rchild = NULL;

Trees.elem[Trees.length].T->lchild = NULL;

if (!Trees.elem[Trees.length].T)

return OVERFLOW;

strcpy(Trees.elem[Trees.length].name, TreeName);

Trees.length++;

return OK;

}

status TreesTraverse(wood Trees) {

int i = 0;

for (i = 0; i < Trees.length; i++) {

printf("%s\n", Trees.elem[i].name);

PreOrderTraverse(Trees.elem[i].T, visit);

printf("\n");

InOrderTraverse(Trees.elem[i].T, visit);

printf("\n");

}

return OK;

}

status DestroyBiTree(BiTree &T) {

if (T) {

DestroyBiTree(T->lchild);

DestroyBiTree(T->rchild);

free(T);

T = NULL;

}

return OK;

}

# 附录D 基于邻接表图实现的源程序

**主函数：**

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <cctype>

#include "def.h"

#include "stu.h"

int main() {

/\*变量定义\*/

Graphs ghs; //多图结构体的声明

ALGraph G; //单个图的声明

int op = 1; //功能选择的声明

int i, j, k; //循环变量

int n; //图的个数

int ukey, vkey, wkey; //要进行查找的关键字

int s; //函数返回状态值

char name[30]; //图的名字

char FileName[100]; //文件保存地址

VertexType value; //新的赋值顶点

//多图的初始化

for (i = 0; i <= 20; i++) {

ghs.elem[i].graph.kind = UDG;

}//全都定义为无向图

ghs.length = 0;

while (op) {

system("cls"); //用于清屏

printf("\n\n");

printf("\t\t Menu for Undirected Graph On Chain Structure \n");

printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t1. 创建图 \t\t\t2. 销毁图\n");

printf("\t\t\t3. 查找顶点 \t\t\t4. 顶点赋值\n");

printf("\t\t\t5. 获得第一邻接点\t\t 6. 获得下一邻接点\n");

printf("\t\t\t7. 插入顶点 \t\t\t8. 删除顶点\n");

printf("\t\t\t9. 插入弧 \t\t 10. 删除弧\n");

printf("\t\t\t11. 深度优先搜索遍历\t\t 12. 广度优先搜索遍历\n");

printf("\t\t\t13. 保存图 \t\t 14. 加载图\n");

printf("\t\t\t15. 遍历图\t\t 16. 遍历多图\n");

printf("\t\t\t\t\t\*\*0. 退出\*\*\n");

printf("\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\t\t\t请选择你的操作[0-16]: ");

scanf("%d", &op);

switch (op) {

case 1: //添加图

printf("当前操作：添加图\n");

printf("请输入要添加的图的个数");

scanf("%d", &n);

while (n--) {

printf("请输入要添加的图的名称\n");

scanf("%s", name);

AddGraphs(ghs, name); //增加图

printf("请输入图的顶点和弧的相关信息：\n");

VertexType V[30]; //顶点的信息

KeyType VR[100][2]; //弧的信息

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(ghs.elem[ghs.length - 1].graph, V, VR) == ERROR){

free(ghs.elem[ghs.length-1].name);

ghs.length--;

printf("输入数据错，无法创建\n");

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

//遍历多线性表

GraphsTraverse(ghs);

printf("\n");

getchar();

getchar();

break;

case 2: //销毁图

printf("当前操作：销毁图\n");

printf("请输入要销毁的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

DestroyGraph(ghs.elem[i].graph);

free(ghs.elem[i].name);

ghs.length--;

while (i < ghs.length) {

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

i++;

}

printf("销毁成功，可以遍历多图查看销毁结果\n");

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 3: //查找顶点

printf("当前操作：查找顶点\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

printf("请输入要查找的关键字：");

scanf("%d", &ukey);

s = LocateVex(ghs.elem[i].graph, ukey);

if (s != -1)

printf("查找成功：%d %s", s, ghs.elem[i].graph.vertices[s].data.others);

else

printf("查找失败");

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 4:

printf("当前操作：顶点赋值\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

printf("请输入重新赋值的顶点的关键字和赋值后的信息：");

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

s = PutVex(ghs.elem[i].graph, ukey, value);

if (s == OK){

printf("修改成功！\n");

Traverse(ghs.elem[i].graph);

}

else

printf("赋值操作失败");

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 5:

printf("当前操作：获得第一邻接点\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

printf("请输入进行操作的顶点的关键字：");

scanf("%d", &ukey);

s = FirstAdjVex(ghs.elem[i].graph, ukey);

if (s != -1)

printf("查找成功：%d %s", ghs.elem[i].graph.vertices[s].data.key, ghs.elem[i].graph.vertices[s].data.others);

else

printf("查找失败");

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 6:

printf("当前操作：获得下一邻接点\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

printf("请输入进行操作的顶点的关键字：");

scanf("%d%d", &vkey, &wkey);

s = NextAdjVex(ghs.elem[i].graph, vkey, wkey);

if (s != -1)

printf("查找成功：%d %s", ghs.elem[i].graph.vertices[s].data.key, ghs.elem[i].graph.vertices[s].data.others);

else

printf("无下一邻接顶点");

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 7:

printf("当前操作：插入顶点\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

printf("请输入进行操作的顶点的信息：");

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

s = InsertVex(ghs.elem[i].graph, value);

if (s != -1)

{

printf("插入顶点成功\n");

Traverse(ghs.elem[i].graph);

}

else

printf("插入操作失败");

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 8:

printf("当前操作：删除顶点\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

printf("请输入进行删除操作的顶点的关键字：");

scanf("%d", &vkey);

s = DeleteVex(ghs.elem[i].graph, vkey);

if (s == OK){

printf("删除顶点成功\n");

Traverse(ghs.elem[i].graph);

}

else

printf("删除操作失败");

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 9:

printf("当前操作：插入弧\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

printf("请输入进行操作的顶点的信息：");

scanf("%d%d", &vkey, &wkey);

s = InsertArc(ghs.elem[i].graph, vkey,wkey);

if (s == OK)

{

printf("插入弧成功\n");

Traverse(ghs.elem[i].graph);

}

else

printf("插入弧操作失败");

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 10:

printf("当前操作：删除弧\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

printf("请输入进行删除操作的顶点的关键字：");

scanf("%d%d", &vkey,&wkey);

s = DeleteArc(ghs.elem[i].graph, vkey,wkey);

if (s == OK){

printf("删除弧成功\n");

Traverse(ghs.elem[i].graph);

}

else

printf("删除操作失败");

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 11:

printf("当前操作：深度优先遍历\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

DFSTraverse(ghs.elem[i].graph,visit);

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 12:

printf("当前操作：广度优先遍历\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

BFSTraverse(ghs.elem[i].graph,visit);

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 13:

printf("当前操作：保存图\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

printf("请输入文件保存的地址\n");

scanf("%s",FileName);

s=SaveGraph(ghs.elem[i].graph,FileName);

if(s==OK){

printf("保存成功");

DestroyGraph(ghs.elem[i].graph);

}

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 14:

printf("当前操作：加载图\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

printf("请输入文件读取的地址\n");

scanf("%s",FileName);

LoadGraph(ghs.elem[i].graph,FileName);

Traverse(ghs.elem[i].graph);

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 15:

printf("当前操作：遍历图\n");

printf("请输入要进行操作的图的名字:");

scanf("%s", name);

//匹配名字

for (i = 0; i < ghs.length; i++) {

j = strcmp(ghs.elem[i].name, name);

if (j == 0) {

Traverse(ghs.elem[i].graph);

break;

}

}

if (i==ghs.length) printf("未匹配到相应名字的图");

getchar();

getchar();

break;

case 16:

printf("当前操作：遍历多图\n");

GraphsTraverse(ghs);

getchar();

getchar();

break;

case 0:

exit(0);

break;

}

}

printf("欢迎下次在使用本系统！\n");

return 1;

}

**头文件：**

***def.h:***

//

// Created by spiri on 2020/11/25.

//

#ifndef EX\_4\_DEF\_H

#define EX\_4\_DEF\_H

#endif //EX\_4\_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

typedef int status;

typedef int KeyType;

typedef int Boolean;

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

Boolean visited[MAX\_VERTEX\_NUM];

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{

int data[MAX\_VERTEX\_NUM];

int front;

int rear;

}Queue;

typedef struct {

struct {

char name[30];

ALGraph graph;

} elem[100];

int length;

int listsize;

} Graphs;

***stu.h:***

//

// Created by spiri on 2020/11/25.

//

#ifndef EX\_4\_STU\_H

#define EX\_4\_STU\_H

#endif //EX\_4\_STU\_H

void Traverse(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");

}

}

void visit(VertexType v) {

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

}

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

int i, pos = 0;

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

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

return pos;

}

pos++;

}

return -1;

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

}

status AddGraphs(Graphs &ghs, char name[]) {

strcpy(ghs.elem[ghs.length].name, name);

ghs.length++;

return OK;

}

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\*/

{

// 请在这里补充代码，完成本关任务

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

/\*

\* V--节点信息

\* VR--节点之间的关系

\*/

int i = 0, k = 0, j;

int pos\_i, pos\_j;

int flag = 0, p = 0, q = 0;

int num = 0;

G.vexnum = 0;

G.arcnum = 0;

//匹配元素是否重复，以及元素是否合法，容量是否合法

do {

q = p;

while (V[++q].key != -1) {

if (V[p].key == V[q].key) flag = 1;

}

num++;

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

if (num > MAX\_VERTEX\_NUM)

return ERROR;

//建立顶点表

if (flag == 0) {

do {

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

G.vertices[i].firstarc = NULL;

G.vexnum++;

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

//建立边表

while (VR[k][0] != -1) {

i = VR[k][0];

j = VR[k][1];

pos\_i = LocateVex(G, i);

pos\_j = LocateVex(G, j);

if (pos\_i==-1||pos\_j==-1) return ERROR;

if (!(pos\_j < G.vexnum && pos\_i < G.vexnum)) return ERROR;

ArcNode \*e = NULL;

//建立i与j的关系

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

e->adjvex = pos\_j;

e->nextarc = G.vertices[pos\_i].firstarc;

G.vertices[pos\_i].firstarc = e;

//建立j与i的关系

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

e->adjvex = pos\_i;

e->nextarc = G.vertices[pos\_j].firstarc;

G.vertices[pos\_j].firstarc = e;

G.arcnum++;

k++;

}

return OK;

}

return ERROR;

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

}

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

int i;

VertexType \*vp;

ArcNode \*p, \*q;

ALGraph \*g = &G;

if (!g) return ERROR;

//先释放边

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

p = G.vertices[i].firstarc;

while (p) {

q = p->nextarc;

free(p);

p = q;

}

}

//再释放顶点,图的信息,顶点数目,边数目,kind置为0

G.vexnum = 0;

G.arcnum = 0;

G.kind = UNKOWN;

return OK;

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

int pos,flag;

//判断修改位置是否存在

if (LocateVex(G, u) == -1) return ERROR;

flag=LocateVex(G, value.key);

//判断是否重复

if (u != value.key && flag != -1) return ERROR;

//如果合法执行赋值操作

//定位u

pos = LocateVex(G, u);

G.vertices[pos].data = value;

return OK;

/\*\*\*\*\*\*\*\*\*\* 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 pos;

pos = LocateVex(G, u);

if (pos == -1) return -1;

if (!G.vertices[pos].firstarc) return -1;

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

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

int pos;

ArcNode \*wp;

pos = LocateVex(G, v);

//找不到v return ERROR

//v的后面没有弧 返回错误信息

if (pos == -1) return -1;

if (!G.vertices[pos].firstarc) return -1;

//查找w

wp = G.vertices[pos].firstarc;

while (G.vertices[wp->adjvex].data.key != w) {

wp = wp->nextarc;

if (wp->nextarc == NULL) return -1; //找不到w

}

return wp->nextarc->adjvex;

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

//判断插入位置是否合法

if (G.vexnum >= MAX\_VERTEX\_NUM) return ERROR;

//判断插入之后是否重复

if (LocateVex(G, v.key) != -1) return ERROR;

//开始插入操作

G.vexnum++;

G.vertices[G.vexnum - 1].data = v;

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

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

//判断删除的关键字是否合法

if (LocateVex(G, v) == -1) return ERROR;

if (LocateVex(G, v) != -1) {

//删除之后仍然是图

if (G.vexnum == 1) return ERROR;

//关键字合法，执行删除操作

int i, pos\_v;

ArcNode \*vp, \*iptr, \*cur;

pos\_v = LocateVex(G, v);

vp = G.vertices[pos\_v].firstarc;

//删除vp对应的数组

while (vp) {

cur = vp;

vp = vp->nextarc;

free(cur);

G.arcnum--;

}

G.vexnum--;

for (i = pos\_v; i < G.vexnum; i++)

G.vertices[i] = G.vertices[i + 1]; //删除之后的数组上移

//遍历删除之后的邻接表，遇到v删除弧，遇到非v就-1

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

iptr = G.vertices[i].firstarc;

while (iptr) {

//如果等于pos\_v 执行删除弧的操作

if (iptr->adjvex == pos\_v) {

if (iptr == G.vertices[i].firstarc) {

//如果是第一个元素

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

free(iptr);

iptr = G.vertices[i].firstarc;

} else {

//如果不是第一个元素

cur->nextarc = iptr->nextarc;

free(iptr);

iptr = cur->nextarc;

}

} else {

//如果大于pos\_v，序号减一

if (iptr->adjvex > pos\_v)

iptr->adjvex--;

cur = iptr;

iptr = iptr->nextarc;

}

}

}

}

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

//首插法

//判断插入位置是否合法

int pos\_v, pos\_w;

pos\_v = LocateVex(G, v);

pos\_w = LocateVex(G, w);

if (pos\_v == -1 || pos\_w == -1) return ERROR; //弧顶点找不到

//如果弧已经重复 返回ERROR

ArcNode \*vp, \*wp, \*cur;

vp = G.vertices[pos\_v].firstarc;

wp = G.vertices[pos\_w].firstarc;

//找v后面是否有w

while (vp) {

if (vp->adjvex == pos\_w) return ERROR;

vp = vp->nextarc;

}

vp = G.vertices[pos\_v].firstarc;

//如果没有w执行插入操作

//在v后面头插w

ArcNode \*New\_Arc1 = (ArcNode \*) malloc(sizeof(ArcNode));

New\_Arc1->adjvex = pos\_w;

cur = vp;

New\_Arc1->nextarc = cur;

G.vertices[pos\_v].firstarc = New\_Arc1;

//在w后面头插v

ArcNode \*New\_Arc2 = (ArcNode \*) malloc(sizeof(ArcNode));

New\_Arc2->adjvex = pos\_v;

cur = wp;

New\_Arc2->nextarc = cur;

G.vertices[pos\_w].firstarc = New\_Arc2;

//插入完成

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 flag = 0;

int pos\_v, pos\_w;

pos\_v = LocateVex(G, v);

pos\_w = LocateVex(G, w);

if (pos\_v == -1 || pos\_w == -1) return ERROR; //弧顶点找不到

//如果弧不存在 返回ERROR

ArcNode \*vp, \*wp, \*cur;

vp = G.vertices[pos\_v].firstarc;

wp = G.vertices[pos\_w].firstarc;

//找v后面是否有w 如果没有 返回ERROR

while (vp) {

if (vp->adjvex == pos\_w) {

flag = 1;

break;

}

vp = vp->nextarc;

}

vp = G.vertices[pos\_v].firstarc;

if (flag == 0) return ERROR;

//所有都合法执行删除弧的操作：

//在v后面删除w

//查找 w

//如果第一个就是则不适用循环

if (vp->adjvex == pos\_w) {

free(vp);

G.vertices[pos\_v].firstarc = NULL;

} else {

while (vp->nextarc->adjvex != pos\_w)

vp = vp->nextarc;

cur = vp->nextarc;

//链到v的next

vp->nextarc = cur->nextarc;

free(cur);

}

//在w后面删除v

//查找 v

//如果第一个就是则不适用循环

if (wp->adjvex == pos\_v) {

free(wp);

G.vertices[pos\_w].firstarc = NULL;

} else {

while (wp->nextarc->adjvex != pos\_v)

wp = wp->nextarc;

cur = wp->nextarc;

//链到w的next

wp->nextarc = cur->nextarc;

free(cur);

}

//删除完成

G.arcnum--;

return OK;

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

}

void DFS(ALGraph G, int i) {

ArcNode \*cur;

visited[i] = TRUE;

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

cur = G.vertices[i].firstarc;

while (cur) {

if (!visited[cur->adjvex])

DFS(G, cur->adjvex);

cur = cur->nextarc;

}

}

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 i;

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

visited[i] = FALSE;

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

if (!visited[i])

DFS(G, i);

return OK;

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

}

status InitQueue(Queue &Q) {

Q.front = 0;

Q.rear = 0;

return OK;

}

status EnQueue(Queue &Q, int e) {

if ((Q.rear + 1) % MAX\_VERTEX\_NUM == Q.front)

return ERROR;

Q.data[Q.rear] = e;

Q.rear = (Q.rear + 1) % MAX\_VERTEX\_NUM;

return OK;

}

status DeQueue(Queue &Q, int &e) {

if (Q.front == Q.rear)

return ERROR;

e = Q.data[Q.front];

Q.front = (Q.front + 1) % MAX\_VERTEX\_NUM;

return OK;

}

status QueueEmpty(Queue &Q) {

if (Q.rear == Q.front)

return TRUE;

else return ERROR;

}

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 i;

ArcNode \*cur;

Queue Q;

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

visited[i] = FALSE;

InitQueue(Q);

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

if (!visited[i]) {

visited[i] = TRUE;

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

EnQueue(Q, i);

while (!QueueEmpty(Q)) {

DeQueue(Q, i);

cur = G.vertices[i].firstarc;

while (cur) {

if (!visited[cur->adjvex]) {

visited[cur->adjvex] = TRUE;

visit(G.vertices[cur->adjvex].data);

EnQueue(Q, cur->adjvex);

}

cur = cur->nextarc;

}

}

}

}

return OK;

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

}

int OutDegree(VNode i) {

int num = 0;

ArcNode \*p;

p = i.firstarc;

while (p != NULL) {

num++;

p = p->nextarc;

}

return num;

}

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

int i, num;

FILE \*fp;

ArcNode \*cur;

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

printf("\t\t\t保存文件失败！\n");

return ERROR;

}

//保存邻接表的基本信息

fwrite(&G.kind, sizeof(int), 1, fp); //邻接表类型

fwrite(&G.vexnum, sizeof(int), 1, fp); //顶点数

fwrite(&G.arcnum, sizeof(int), 1, fp); //弧数

//保存每一个序号的节点和弧的信息

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

//保存每个序号的顶点信息

fwrite(&G.vertices[i].data, sizeof(VertexType), 1, fp);

num = OutDegree(G.vertices[i]);

fwrite(&num, sizeof(int), 1, fp);

if ((cur = G.vertices[i].firstarc) != NULL) {

fwrite(&cur->adjvex, sizeof(int), 1, fp);

while ((cur = cur->nextarc) != NULL) {

fwrite(&(cur->adjvex), sizeof(int), 1, fp);

}

}

}

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

int i, k = 0, num;

FILE \*fp;

ArcNode \*cur,\*nArc;

if (G.vexnum != 0) return ERROR;

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

printf("\t\t\t文件加载失败！\n");

return ERROR;

}

fread(&G.kind, sizeof(int), 1, fp);

fread(&G.vexnum, sizeof(int), 1, fp);

fread(&G.arcnum, sizeof(int), 1, fp);

while (fread(&G.vertices[k].data, sizeof(VertexType), 1, fp)) {

fread(&num, sizeof(int), 1, fp);

if (num == 0)

G.vertices[k].firstarc = NULL;

else {

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

fread(&(nArc->adjvex), sizeof(int), 1, fp);

G.vertices[k].firstarc=nArc;

cur=nArc;

for (i = 0; i < num-1; i++) {

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

fread(&(nArc->adjvex), sizeof(int), 1, fp);

cur->nextarc = nArc;

cur=nArc;

}

cur->nextarc=NULL;

}

k++;

}

fclose(fp);

return OK;

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

}

status GraphsTraverse(Graphs &ghs) {

int i, j;

if (ghs.length == 0){

printf("多图为空");

return OK;

}

printf("当前有以下几个图：\n");

for (i = 0; i < ghs.length; i++) {

//遍历每个图的信息

printf("\n图名称：%s\n", ghs.elem[i].name);

for (j = 0; j < ghs.elem[i].graph.vexnum; j++) {

//遍历邻接表

ArcNode \*p = ghs.elem[i].graph.vertices[j].firstarc;

printf("%d %s", ghs.elem[i].graph.vertices[j].data.key, ghs.elem[i].graph.vertices[j].data.others);

while (p) {

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

p = p->nextarc;

}

printf("\n");

}

}

return OK;

}