

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术202003**

**学 号： U202015381**

**姓 名： 苏宜静**

**指导教师： 陈奇**

**报告日期： 2021年 6月 30 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 5](#_Toc75600691)

[1.1 问题描述 5](#_Toc75600692)

[1.1.1 问题1：线性表ADT的单链表实现 5](#_Toc75600693)

[1.1.2 问题2：实现线性表的文件形式保存和读取 5](#_Toc75600694)

[1.1.3 问题3：实现多线性表管理 5](#_Toc75600695)

[1.1.4 问题4：构造具有菜单的功能演示系统 6](#_Toc75600696)

[1.2 系统设计 6](#_Toc75600697)

[1.2.1 顺序表实现下线性表ADT的物理结构设计 6](#_Toc75600698)

[1.2.2 文件格式设计 7](#_Toc75600699)

[1.2.3 多线性表管理模式设计 7](#_Toc75600700)

[1.2.4 具有菜单功能的演示系统设计 7](#_Toc75600701)

[1.3 系统实现 9](#_Toc75600702)

[1.3.1 项目工程说明 9](#_Toc75600703)

[1.3.2 线性表ADT运算的顺序表实现 9](#_Toc75600704)

[1.3.3 文件输入输出实现 19](#_Toc75600705)

[1.3.4 多线性表管理实现 20](#_Toc75600706)

[1.3.5 演示系统实现 22](#_Toc75600707)

[1.4 系统测试 24](#_Toc75600708)

[1.4.1 测试计划 24](#_Toc75600709)

[1.4.2 测试结果 28](#_Toc75600710)

[1.4.3 测试分析 36](#_Toc75600711)

[1.5 实验小结 36](#_Toc75600712)

[2 基于链式存储结构的线性表实现 38](#_Toc75600713)

[2.1 问题描述 38](#_Toc75600714)

[2.1.1 问题1：线性表ADT的单链表实现 38](#_Toc75600715)

[2.1.2 问题2：实现线性表的文件形式保存和读取 38](#_Toc75600716)

[2.1.3 问题3：实现多线性表管理 38](#_Toc75600717)

[2.1.4 问题4：构造具有菜单的功能演示系统 39](#_Toc75600718)

[2.2 系统设计 39](#_Toc75600719)

[2.2.1 单链表实现下线性表ADT的物理结构设计 39](#_Toc75600720)

[2.2.2 文件格式设计 40](#_Toc75600721)

[2.2.3 多线性表管理模式设计 40](#_Toc75600722)

[2.2.4 具有菜单功能的演示系统设计 40](#_Toc75600723)

[2.3 系统实现 42](#_Toc75600724)

[2.3.1 项目工程说明 42](#_Toc75600725)

[2.3.2 线性表ADT运算的单链表实现 43](#_Toc75600726)

[2.3.3 文件输入输出实现 54](#_Toc75600727)

[2.3.4 多线性表管理实现 55](#_Toc75600728)

[2.3.5 演示系统实现 57](#_Toc75600729)

[2.4 系统测试 58](#_Toc75600730)

[2.4.1 测试计划 58](#_Toc75600731)

[2.4.2 测试结果 62](#_Toc75600732)

[2.4.3 测试分析 70](#_Toc75600733)

[2.5 实验小结 70](#_Toc75600734)

[3 基于二叉链表的二叉树实现 72](#_Toc75600735)

[3.1 问题描述 72](#_Toc75600736)

[3.1.1 问题1：二叉树ADT的二叉链表实现 72](#_Toc75600737)

[3.1.2 问题2：实现二叉树的文件形式保存和读取 72](#_Toc75600738)

[3.1.3 问题3：实现多二叉树管理 73](#_Toc75600739)

[3.1.4 问题4：构造具有菜单的功能演示系统 73](#_Toc75600740)

[3.2 系统设计 73](#_Toc75600741)

[3.2.1 二叉链表实现下二叉树ADT的物理结构设计 73](#_Toc75600742)

[3.2.2 文件格式设计 74](#_Toc75600743)

[3.2.3 多二叉树管理模式设计 74](#_Toc75600744)

[3.2.4 具有菜单的功能演示系统设计 75](#_Toc75600745)

[3.3 系统实现 76](#_Toc75600746)

[3.3.1 项目工程说明 76](#_Toc75600747)

[3.3.2 二叉树ADT运算的二叉链表实现 77](#_Toc75600748)

[3.3.3 文件输入输出实现 90](#_Toc75600749)

[3.3.4 多二叉树管理实现 92](#_Toc75600750)

[3.3.5 演示系统实现 93](#_Toc75600751)

[3.4 系统测试 95](#_Toc75600752)

[3.4.1 测试计划 95](#_Toc75600753)

[3.4.2 测试结果 101](#_Toc75600754)

[3.4.3 测试分析 114](#_Toc75600755)

[3.5 实验小结 114](#_Toc75600756)

[4 基于邻接表的图实现 115](#_Toc75600757)

[4.1 问题描述 115](#_Toc75600758)

[4.1.1 问题1：无向图ADT的邻接表实现 115](#_Toc75600759)

[4.1.2 问题2：实现无向图的文件形式保存和读取 115](#_Toc75600760)

[4.1.3 问题3：实现多线性表管理 115](#_Toc75600761)

[4.1.4 问题4：构造具有菜单的功能演示系统 116](#_Toc75600762)

[4.2 系统设计 116](#_Toc75600763)

[4.2.1 邻接表实现下无向图ADT的物理结构设计 116](#_Toc75600764)

[4.2.2 文件格式设计 117](#_Toc75600765)

[4.2.3 多无向图管理模式设计 117](#_Toc75600766)

[4.2.4 具有菜单功能的演示系统设计 118](#_Toc75600767)

[4.3 系统实现 119](#_Toc75600768)

[4.3.1 项目工程说明 119](#_Toc75600769)

[4.3.2 无向图ADT运算的邻接表实现 120](#_Toc75600770)

[4.3.3 文件输入输出实现 131](#_Toc75600771)

[4.3.4 多无向图管理实现 132](#_Toc75600772)

[3.3.5 演示系统实现 134](#_Toc75600773)

[4.4 系统测试 136](#_Toc75600774)

[4.4.1 测试计划 136](#_Toc75600775)

[4.4.2 测试结果 141](#_Toc75600776)

[4.4.3 测试分析 153](#_Toc75600777)

[4.5 实验小结 153](#_Toc75600778)

[参考文献 155](#_Toc75600779)

[附录A 基于顺序存储结构线性表实现的源程序 156](#_Toc75600780)

[附录B 基于链式存储结构线性表实现的源程序 183](#_Toc75600781)

[附录C 基于二叉链表二叉树实现的源程序 209](#_Toc75600782)

[附录D 基于邻接表图实现的源程序 243](#_Toc75600783)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

本实验以顺序表作为线性表的物理结构，通过实现线性表的基本操作，掌握线性表的基本概念，熟练掌握线性表的逻辑结构和物理结构的关系，掌握线性表基本运算的实现。依据最小完备性和常用性相结合的原则，实验中采用函数形式定义和实现线性表的初始化、销毁表、清空表、判定空表、求表长、获取元素等12种基本运算，并利用线性表进行文件读写和进行线性表管理。本实验将以上述内容为主要目标，使用 C++语言构建问题解决方案。

### 1.1.1 问题1：线性表ADT的单链表实现

根据实验要求，本次实验中要以顺序表作为线性表的物理结构，通过函数形式定义和实现线性表的基本功能，掌握和了解线性表的代码实现和功能操作思想。基本运算包括初始化表、销毁表、清空表、判空表、求表长、获取元素、查找元素、获取前驱元素、获取后继元素、插入元素、删除元素、遍历表等12种，基本可以囊括线性表的各项基本功能和管理。

### 1.1.2 问题2：实现线性表的文件形式保存和读取

根据实验要求，实验者需要实现将线性表内容保存在指定文件中、将指定文件中的数据内容读取到线性表中两种操作。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。

### 1.1.3 问题3：实现多线性表管理

根据实验要求，实验者要实现多个线性表管理，包括线性表添加、线性表删除、线性表查找等3个功能。线性表集合中对每个线性表进行唯一命名，便于选择进入对某个线性表的操作；线性表集合应当设置上限，达到上限后不能再进行添加线性表操作；对不存在的线性表进行删除或查找操作应返回操作错误提示信息；对命名已存在的线性表实行添加操作时返回操作错误提示。

### 1.1.4 问题4：构造具有菜单的功能演示系统

根据实验要求，需要实验者自行设计和构造可以完整展现线性表ADT功能的演示系统，可以通过用户选择实现对线性表的基本功能操作。演示系统中应包括功能选项、用户需要输入内容提示、操作结果提示等，便于用户直接使用和操作系统，进行合法输入。同时，因为演示系统包括多线性表管理和单线性表功能操作，应当设计好主菜单和子菜单的层级关系，使用户操作结果可以在两层菜单中传递和保存。

## 1.2 系统设计

根据实验要求，系统设计主要包括以下几个方面。

（1）设计顺序表实现下的线性表ADT的物理结构。

（2）设计文件格式。

（3）设计多线性表管理模式。

（4）设计具有菜单的功能演示系统。

### 1.2.1 顺序表实现下线性表ADT的物理结构设计

顺序表ADT的物理结构如下：

|  |
| --- |
| typedef struct{ //顺序表（顺序结构）的定义  ElemType \* elem; //存放抽象数据元素的顺序表  int length; //线性表表长  int listsize; //线性表分配空间大小  }SqList; |

此处使用预定义的方式实现线性表ADT的物理结构，便于后续对线性表的定义和使用。

### 1.2.2 文件格式设计

实验中使用二进制文件格式进行对线性表中数据元素的存储和读取，有利于数据形式在文件中保存和读取到线性表中。

### 1.2.3 多线性表管理模式设计

多线性表存放管理模式定义如下：

|  |
| --- |
| typedef struct{ //线性表的管理表定义  struct { char name[30]; //线性表名称  SqList L;  } elem[10]; //存放线性表的集合  int length; //线性表集合的长度  int listsize; //线性表集合分配的空间  }LISTS; |

使用数组方式建立线性表集合，同时记录线性表集合长度和空间，便于对多线性表的管理和选择。

### 1.2.4 具有菜单功能的演示系统设计

在演示系统中，为实现多线性表管理和单线性表操作，设计多级菜单和多、单线性表系统。演示系统结构如图1-1所示。



图1-1 演示系统结构

演示系统包括在多线性表选择下的主菜单、二级菜单与临时单线性表选择。多线性表管理菜单包含添加、删除、查找、进入二级菜单、退出5个选项，进入二级菜单前需要输入待操作的线性表名称，如该线性表不存在则返回错误提示。二级菜单包含线性表的12个基本功能操作和线性表的文件操作。临时单线性表选项下可以创建不在线性表集合中的无命名的临时线性表，进入选项后可进行对线性表的12个基本功能操作和线性表的文件操作。

针对系统各层级的进入和退出、线性表功能的选择，通过switch语句，由用户输入数字化选择op（演示系统中有对应功能提示表），根据用户输入进入不同层级或功能选择。在主函数中针对需要获取用户输入的函数设计对应提示，以便用户知晓需要输入以实现函数正常功能的内容；针对正常情况（或合法输入）和异常情况（或非法输入）设计对应提示，以便用户知晓操作是否成功及操作结果，有利于用户下一步操作。

## 1.3 系统实现

本次实验使用的开发环境为Microsoft Visual Studio Code 1.56.2。

系统实现方面主要包括如下内容。

（1）项目工程说明。

（2）线性表ADT运算的顺序表实现。

（3）文件输入输出实现。

（4）多线性表管理实现。

（5）演示系统实现。

### 1.3.1 项目工程说明

项目工程包含4个标准库头文件：stdio.h、stdlib.h、malloc.h、string.h。

项目工程中预定义的常量和类型声明如下：

|  |
| --- |
| #define TRUE 1  #define FALSE 0  #define OK 1  #define ERROR 0  #define INFEASIBLE -1  #define OVERFLOW -2 //预定义的常量  typedef int status; //函数返回结果标志  typedef int ElemType; //数据元素类型定义 |

### 1.3.2 线性表ADT运算的顺序表实现

相关常量定义和顺序表elem、顺序表结构SqList定义如下：

|  |
| --- |
| #define LIST\_INIT\_SIZE 100 //线性表存储结构分配的空间  #define LISTINCREMENT 10 //线性表空间分配增量  typedef struct{ //顺序表（顺序结构）的定义  ElemType \* elem; //存放抽象数据元素的顺序表  int length; //线性表表长  int listsize; //线性表分配空间大小  }SqList; |

使用的自定义函数如表1-1：

表1-1 自定义线性表函数表

|  |  |  |
| --- | --- | --- |
| 序号 | 函数定义 | 函数功能 |
| 1 | status InitList(SqList &L) | 初始化线性表 |
| 2 | status DestroyList(SqList &L) | 销毁线性表 |
| 3 | status ClearList(SqList &L) | 清空线性表 |
| 4 | status ListEmpty(SqList L) | 线性表判空 |
| 5 | status ListLength(SqList L) | 求线性表表长 |
| 6 | status GetElem(SqList L,int i,ElemType &e) | 获取元素 |
| 7 | int LocateElem(SqList L,ElemType e) | 查找元素 |
| 8 | status PriorElem(SqList L,ElemType e,ElemType &pre) | 获取前驱元素 |
| 9 | status NextElem(SqList L,ElemType e,ElemType &next) | 获取后继元素 |
| 10 | status ListInsert(SqList &L,int i,ElemType e) | 插入元素 |
| 11 | status ListDelete(SqList &L,int i,ElemType &e) | 删除元素 |
| 12 | status ListTraverse(SqList L) | 遍历线性表 |
| 13 | status SaveList(SqList L,char FileName[]) | 线性表存档 |
| 14 | status LoadList(SqList &L,char FileName[]) | 线性表读档 |
| 15 | status AddList(LISTS &Lists,char ListName[]) | 增加线性表 |
| 16 | status RemoveList(LISTS &Lists,char ListName[]) | 删除线性表 |
| 17 | int LocateList(LISTS Lists,char ListName[]) | 查找线性表 |
| 18 | status InitInput(SqList &L) | 进入二级菜单 |
| 19 | void SecondaryTable(SqList &L,int op) | 线性表数据输入 |

**1.初始化线性表**

函数功能概述：线性表L不存在，构造一个空的线性表，返回OK，否则返回INFEASIBLE。

具体实现：

（1）首先判断线性表是否存在，如线性表存在，返回INFEASIBLE。

（2）如线性表不存在，利用动态分配函数malloc为线性表分配空间，定义线性表的空间大小和长度。判断线性表是否初始化成功，成功则返回OK，失败则返回ERROR。

复杂度分析：时间复杂度：O（1）；空间复杂度：O（1）。

**2.销毁线性表**

函数功能概述：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。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回INFEASIBLE。

（2）如线性表存在，将线性表指向置为空，释放线性表的空间，将线性表的长度和空间大小置为0。判断线性表是否被销毁，成功则返回OK，失败则返回ERROR。

复杂度分析：时间复杂度：O（1）；空间复杂度：O（1）。

**3.清空线性表**

函数功能概述：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。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回INFEASIBLE。

（2）如线性表存在，将线性表长度置为0，返回OK。

注：清空线性表与销毁线性表的区别在于，清空线性表只是将线性表中的元素内容清除，长度重置为0，而线性表依然占据存储空间，不能将线性表的空间释放。

复杂度分析：时间复杂度：O（1）；空间复杂度：O（1）。

**4.线性表判空**

函数功能概述：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。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回INFEASIBLE。

（2）如线性表存在，判断线性表长度L.length是否为0。L.length是创建线性表和添加元素时对线性表长度的记录，L.length为0则线性表为空，返回TRUE，否则线性表不为空，返回FALSE。

复杂度分析：时间复杂度：O（1）；空间复杂度：O（1）。

**5.求线性表长度**

函数功能概述：如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回INFEASIBLE。

（2）如线性表存在，通过L.length获取线性表的长度并返回L.length。

复杂度分析：时间复杂度：O（1）；空间复杂度：O（1）。

**6.获取元素**

函数功能概述：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。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回INFEASIBLE。

（2）如线性表存在，根据用户输入的逻辑顺序i，如i值合法可由顺序表下标索引查找得到线性表的第i个元素，弹出该元素保存在e中，返回OK；如i值不合法，返回ERROR。

注意，用户输入的i为逻辑序号，线性表下标使用物理序号进行定位，因此最后i对应的元素为顺序表中下标为i-1的元素。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。



图1-2 获取元素流程图

**7.查找元素**

函数功能概述：如果线性表L存在，查找元素e在线性表L中的位置序号并返回该序号；如果e不存在，返回0；当线性表L不存在时，返回INFEASIBLE。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回INFEASIBLE。

（2）如线性表存在，根据用户输入元素，遍历线性表查找该元素是否存在于线性表中。使用flag标记是否查找到该元素。若线性表中某元素与待查找元素相同，标记flag，返回该元素在线性表中的逻辑序号，即i+1；如遍历循环结束后flag仍未被标记，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**8.获取前驱元素**

函数功能概述：如果线性表L存在，获取线性表L元素e的前驱，保存在pre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。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回不可行提示。

（2）如线性表存在，根据用户输入元素，通过i遍历线性表查找该元素是否存在于线性表中。如元素存在于线性表中且其位置合法（存在前驱元素），则弹出其后继元素，保存在pre中，返回OK；否则，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。



图1-3 获取前驱元素流程图

**9.获取后继元素**

函数功能概述：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。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回INFEASIBLE。

（2）如线性表存在，根据用户输入元素，通过i遍历线性表查找该元素是否存在于线性表中。如元素存在于线性表中且其位置合法（存在后继元素），则弹出其后继元素，保存在next中，返回OK；否则，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。



图1-4 获取后继元素流程图

**10.插入元素**

函数功能概述：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。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回不可行提示。

（2）如线性表存在，根据用户输入的逻辑序号i，判断i值是否合法且线性表长度L.length是否已达到线性表空间L.listsize大小。如i值合法且线性表长度未达到线性表空间大小，通过循环将第i个及第i个之后的元素后移一位，再将待插入元素e插入到原本第i个元素的位置，线性表长度加一。随插入元素增多，线性表长度L.length可能等于线性表空间大小L.listsize，此时线性表中已无空间存放新元素，需要另外开辟空间。即，如i值合法且线性表长度达到线性表空间大小时，通过realloc函数增加线性表空间，L.listsize增加，再重复上述移位插入操作。插入成功返回OK；如i值不合法，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。



图1-5 插入元素流程图

**11.删除元素**

函数功能概述：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。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回INFEASIBLE。

（2）如线性表存在，通过用户输入元素逻辑序号i，当i合法时，通过顺序表下标弹出第i个元素，保存在e中，从第i个元素即下标i-1处起，其后所有元素遍历前移一位，线性表长度减一，返回OK。若i不合法，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。



图1.3-11 删除元素流程图

**12.遍历线性表**

函数功能概述：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。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回INFEASIBLE。

（2）如线性表存在，判断线性表长度是否为零，即线性表是否为空，为空则返回ERROR；否则通过顺序表下标i遍历线性表，依次打印线性表中的元素，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**13.进入二级菜单**

函数功能概述：将所选的多线性表集合中的待操作线性表传递进入二级菜单中，实现对单线性表的基本操作。

具体实现：

包含对所有基本功能函数的选择，根据用户输入选择系统功能实现操作，利用函数使参数传递更简单。

### 1.3.3 文件输入输出实现

线性表的逻辑结构较简单，只需要将线性表中元素依次写入到文件中或依次读取文件中元素，即可实现文件的输入输出。

**1.线性表存档**

函数功能概述：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。

具体实现：

（1）首先判断线性表是否存在，如线性表不存在，返回INFEASIBLE。

（2）如线性表存在，根据用户输入的文件路径和名称打开对应文件，利用顺序表下标i遍历线性表，依次将线性表中元素以二进制写入到文件中，至遍历完线性表长度退出循环，关闭文件，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**2.线性表读档**

函数功能概述：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。

具体实现：

（1）首先判断线性表是否存在，如线性表存在，进行读档操作会使线性表中原有元素失去，因此不能进行读档操作，返回INFEASIBLE。

（2）如线性表不存在，根据用户输入的文件路径和名称打开对应文件，使用malloc函数为线性表分配空间，将线性表空间L.listsize置为LIST\_INIT\_SIZE，线性表长度L.length置为0。通过下标i对创建的线性表依次进行元素读取。当对应文件中文件指针指向元素存在时，若线性表长度未达到线性表空间大小，直接将文件中数据读入到线性表中，线性表长度和i值增加；若线性表长度达到线性表空间大小，使用realloc函数为线性表增加空间，空间大小L.listsize增加，再重复线性表长度L.length和i值增加的操作。至文件数据全部读取后退出循环，关闭文件，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

### 1.3.4 多线性表管理实现

多线性表管理主要包括对线性表集合的线性表进行增加、删除、查找操作，同时包括实验员为从一级菜单进入二级菜单使用的进入单线性表操作函数。对多线性表的管理使用数组实现。

**1.增加空线性表**

函数功能概述：在Lists中增加一个名称为ListName的空线性表，线性表数据另行插入。

具体实现：

（1）当线性表集合中线性表个数等于线性表集合的空间时，返回OVERFLOW。

（2）当线性表集合中线性表个数小于线性表集合的空间时，利用线性表集合的下标i遍历线性表集合，使用strcmp函数判断线性表集合中各个线性表名称是否与待增加的线性表的名称相同，使用flag对相同名称线性表进行标记。若被标记，则返回ERROR，以避免线性表集合中出现同名线性表；若未被标记，将待添加线性表的名字复制给线性表集合中的新添加位置的线性表，并为线性表分配空间，将线性表初始空间大小置为LIST\_INIT\_SIZE，线性表长度置为0，线性表集合长度加一，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**2.删除线性表**

函数功能概述：在Lists中删除一个名称为ListName的线性表。

具体实现：

利用线性表集合的下标i遍历线性表集合，利用strcmp函数判断集合中是否有与待删除线性表名称相同的线性表，有则用flag标记并退出遍历循环。如flag未被标记，返回ERROR；如flag被标记，从当前i值开始，让后一个线性表覆盖前一个线性表，清空集合末尾最后一个线性表，线性表集合的长度Lists.length减一，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**3.查找线性表**

函数功能概述：在Lists中查找一个名称为ListName的线性表，查找成功返回逻辑序号，否则返回ERROR。

具体实现：

利用线性表集合的下标i遍历线性表集合，利用strcmp函数判断集合中是否有与待删除线性表名称相同的线性表，有则用flag标记并退出遍历循环。如flag未被标记或线性表长度为零，返回ERROR；否则，返回待查找线性表在线性表集合中的逻辑序号i+1。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**4.线性表数据输入**

函数功能概述：选择是否向线性表中输入数据，是则由用户输入以0为结束标志的一组数据，返回OK，否则返回0。

具体实现：

提示用户选择是否输入数据，若选择不输入，返回0；若选择输入，在用户输入值不为0时循环输入数据，利用ListInsert函数将输入元素插入到对应线性表尾部，输入完成后打印输入成功提示并返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

### 1.3.5 演示系统实现

主菜单演示界面设计如下：

|  |
| --- |
| system("cls"); printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("--------------------------------------------------------------\n");  printf(" -1.MainTable/MultiList -2.SingleList\n");  printf(" -3.Exit\n");  printf("--------------------------------------------------------------\n");  printf(" 请选择所在的菜单层级：\n");  printf(" -1：主菜单实现多线性表管理，-2：临时单线性表基本操作。\n"); |

多线性表管理界面设计如下：

|  |
| --- |
| system("cls"); printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("---------------------------------------------------------------------\n");  printf(" 15.AddList 16.RemoveList 17.LocateList\n");  printf(" 18.SecondaryTable\n");  printf(" 0. Exit\n");  printf("---------------------------------------------------------------------\n");  for(i=0;i<5;i++)  {  if(Lists.elem[i].L.elem==NULL&&Lists.elem[i+5].L.elem==NULL)  printf("\t线性表%d：未创建 \t线性表%d：未创建",i+1,i+6);  else if(Lists.elem[i].L.elem==NULL&&Lists.elem[i+5].L.elem!=NULL)  printf("\t线性表%d：未创建 \t线性表%d：%s",i+1,i+6,Lists.elem[i+5].name);  else if(Lists.elem[i].L.elem!=NULL&&Lists.elem[i+5].L.elem==NULL)  printf("\t线性表%d：%s \t线性表%d：未创建",i+1,Lists.elem[i].name,i+6);  else  printf("\t线性表%d：%s \t线性表%d：%s",i+1,Lists.elem[i].name,i+6,Lists.elem[i+5].name);  printf("\n");  }  printf(" 请选择你的操作[0或15~18]:"); |

单线性表操作界面设计如下：

|  |
| --- |
| system("cls"); printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("-------------------------------------------------------------------------\n");  printf(" 1. InitList 7. LocateElem 13.SaveList\n");  printf(" 2. DestroyList 8. PriorElem 14.LoadList\n");  printf(" 3. ClearList 9. NextElem\n");  printf(" 4. ListEmpty 10. ListInsert\n");  printf(" 5. ListLength 11. ListDelete\n");  printf(" 6. GetElem 12. ListTrabverse\n");  printf(" 0. Exit\n");  printf("-------------------------------------------------------------------------\n");  printf(" 请选择你的操作[0~14]:"); |

## 1.4 系统测试

根据实验要求，系统测试主要包括以下内容。

（1）测试计划。

（2）测试结果。

（3）测试分析。

### 1.4.1 测试计划

测试计划包含3个模块：多线性表管理菜单测试，单线性表操作菜单测试，文件操作测试。

测试模块1：多线性表管理菜单测试（如表1-2所示）。

表1-2 多线性表管理菜单测试内容

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试序号 | 测试前总体情况描述 | 测试描述 | 测试操作 | 预计输出 |
| 1 | 线性表集合有线性表 | 测试添加线性表功能 | 选择添加表，提供合法输入 | 添加线性表成功，线性表集合菜单尾部显示新添加的线性表名称，提示是否输入线性表元素 |
| 2 | 线性表集合中有3个线性表 | 测试添加线性表功能 | 选择添加表，输入已存在的表名称 | 添加线性表失败，提示线性表已存在 |
| 3 | 线性表集合已满 | 测试添加线性表功能 | 选择添加表，提供合法输入 | 添加线性表失败，提示线性表集合已满 |
| 4 | 线性表集合有3个线性表 | 测试删除线性表功能 | 选择删除表，删除全部线性表 | 删除线性表成功，线性表集合菜单显示所有线性表均未创建 |
| 5 | 线性表集合有3个线性表 | 测试删除线性表功能 | 选择删除表，输入集合中不存在的线性表名称 | 删除线性表失败，提示集合中无此线性表 |
| 6 | 线性表集合有3个线性表 | 测试查找线性表功能 | 选择查找表，查找第3个线性表 | 查找线性表成功，提示线性表在集合中位置 |
| 7 | 线性表集合有3个线性表 | 测试查找线性表功能 | 选择查找表，查找不存在于集合中的表 | 查找线性表失败，提示集合中无此线性表 |
| 8 | 线性表集合有3个线性表 | 测试进入二级菜单功能 | 选择进入二级菜单，选择不存在于集合中的表名称 | 进入二级菜单失败，提示集合中无此线性表 |

测试模块2：单线性表操作菜单测试（如表1-3所示）。

表1-3 单线性表操作菜单测试内容

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试序号 | 测试前总体情况描述 | 测试描述 | 测试操作 | 预计输出 |
| 1 | 多线性表进入单线性表操作，线性表中有初始元素 | 测试初始化功能 | 选择初始化，提供合法输入 | 初始化失败，提示线性表已存在 |
| 2 | 线性表已存在 | 测试销毁功能 | 选择销毁 | 销毁成功，提示线性表被销毁 |
| 3 | 线性表已被销毁 | 测试销毁功能 | 选择销毁 | 销毁成功，提示线性表不存在 |
| 4 | 线性表已被销毁 | 测试初始化、遍历功能 | 选择初始化，提供与原线性表数据不同的合法输入；选择遍历 | 初始化成功，提示创建新线性表，提示是否输入线性表元素；遍历成功，显示后一次输入的数据元素 |
| 5 | 表中有元素 | 测试清空功能 | 选择清空 | 清空成功 |
| 6 | 表中有元素 | 测试判空功能 | 选择判空 | 显示线性表为空 |
| 7 | 表中无元素 | 测试求表长功能 | 选择求表长 | 求表长成功，显示表长 |
| 8 | 表中有元素 | 测试获取元素功能 | 选择获取元素，提供合法输入 | 获取元素成功，显示位置及元素 |
| 9 | 表中有元素 | 测试获取元素功能 | 选择获取元素，输入0或大于线性表长度的位置 | 获取元素失败，提示元素位置不合法 |
| 10 | 表中有元素 | 测试查找元素功能 | 选择查找元素，提供合法输入 | 查找元素成功，显示元素及位置 |
| 11 | 表中有元素 | 测试查找元素功能 | 选择查找元素，输入不在线性表中的元素 | 查找元素成功，提示元素不存在于线性表中 |
| 12 | 表中有元素 | 测试获取前驱元素功能 | 选择获取前驱元素，提供合法输入 | 获取前驱元素成功，显示待查找元素和前驱元素 |
| 13 | 表中有元素 | 测试获取前驱元素功能 | 选择获取前驱元素，输入首位元素数据 | 获取前驱元素失败，提示待查找元素无前驱元素 |
| 14 | 表中有元素 | 测试获取后继元素功能 | 选择获取后继元素，提供合法输入 | 获取后继元素成功，显示待查找元素和后继元素 |
| 15 | 表中有元素 | 测试获取后继元素功能 | 选择获取后继元素，输入末位元素数据 | 获取后继元素失败，提示待查找元素无后继元素 |
| 16 | 表中有元素 | 测试插入元素、删除元素功能 | 选择插入元素，提供合法输入；选择删除元素，提供合法输入 | 插入和删除元素成功 |
| 17 | 表中有元素 | 测试插入元素、删除元素功能 | 选择插入元素，输入不合法插入位置；选择删除元素，输入不在线性表中的元素 | 插入和删除元素失败 |
| 18 | 表中有元素 | 测试插入元素、删除元素、遍历功能 | 选择遍历 | 遍历成功，显示上次操作后线性表中元素顺序，符合插入和删除预期 |
| 19 | 表中无元素 | 测试遍历功能 | 选择遍历 | 提示线性表无元素 |

测试模块3：文件操作菜单测试（如表1-4所示）。

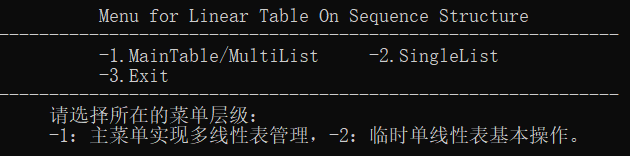
表1-4 文件操作菜单测试内容

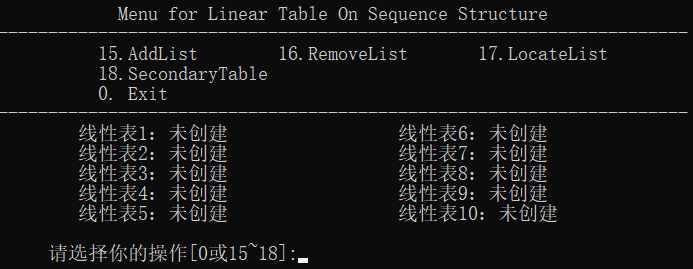
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试序号 | 测试前总体情况描述 | 测试描述 | 测试操作 | 预计输出 |
| 1 | 线性表存在 | 测试存档功能 | 选择存档 | 存档成功，在指定文件下可见线性表中数据 |
| 2 | 线性表不存在 | 测试存档功能 | 选择存档 | 存档失败，提示线性表不存在 |
| 3 | 线性表存在 | 测试读档功能 | 选择读档 | 读档失败，提示线性表已存在，读档会失去线性表中原有元素 |
| 4 | 线性表不存在 | 测试读档功能 | 选择读档 | 读档成功，遍历线性表可见线性表中元素与指定文件中相同 |

### 1.4.2 测试结果

根据测试计划，测试结果包含3个部分：多线性表管理菜单测试结果，单线性表操作菜单测试结果，文件操作测试结果。

各初始菜单选择界面如图1-7所示。





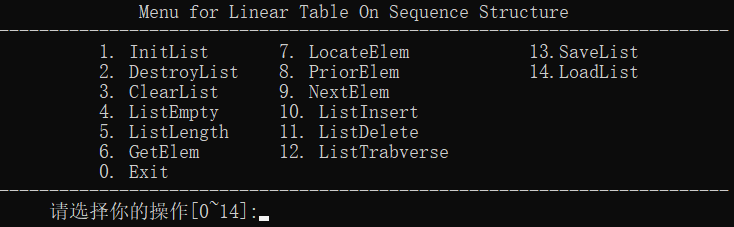
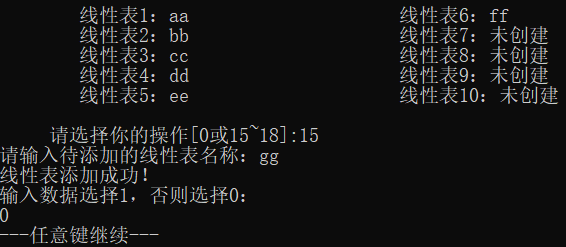


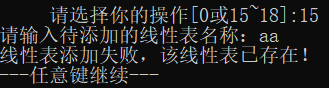
图1-7 线性表系统菜单界面展示

多线性表管理菜单测试结果如图1-8所示。

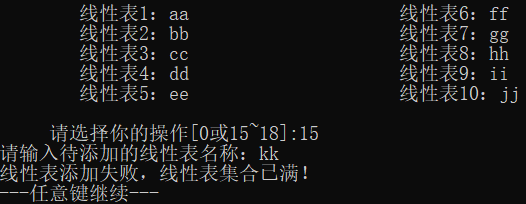




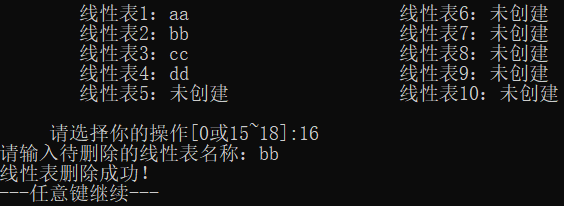
测试序号1

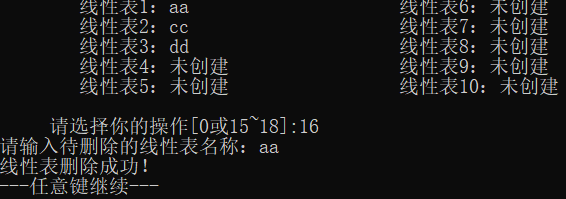


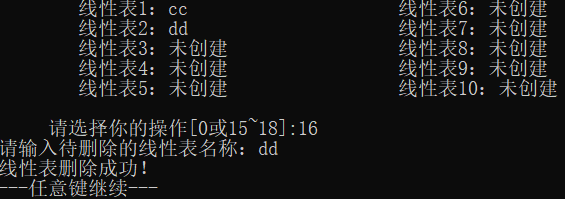
测试序号2

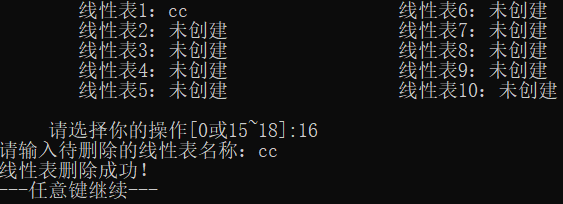


测试序号3

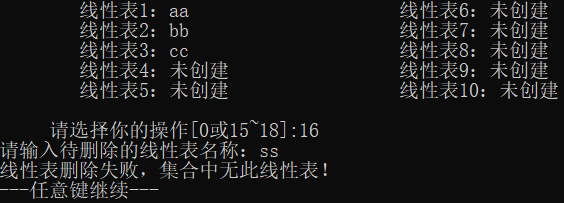




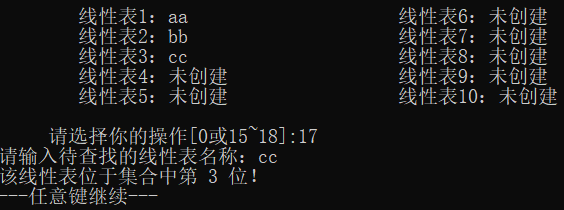




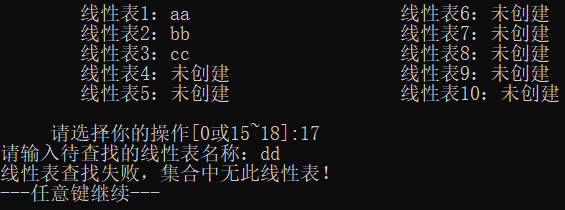
测试序号4



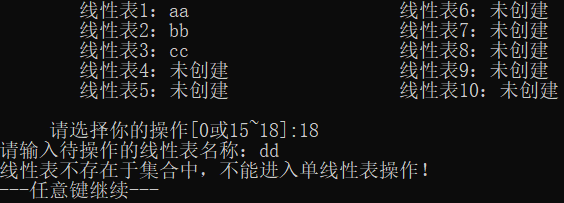
测试序号5



测试序号6



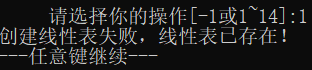
测试序号7



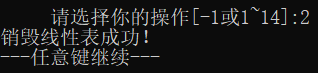
测试序号8

图1-8 多线性表管理菜单测试结果

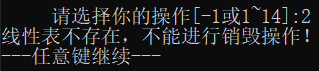
单线性表操作菜单测试结果如图1-9所示。



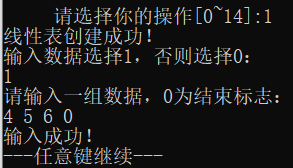
测试序号1

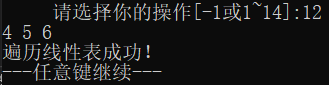


测试序号2

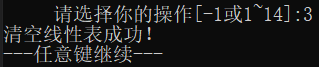


测试序号3

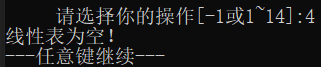




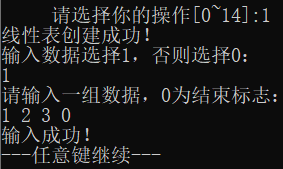
测试序号4



测试序号5

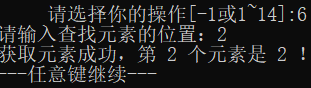


测试序号6

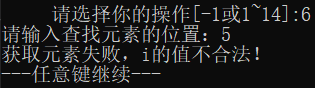




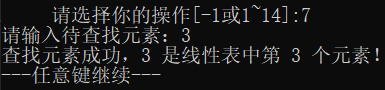
测试序号7



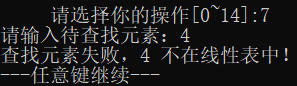
测试序号8



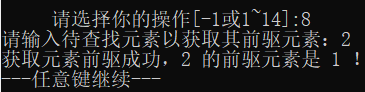
测试序号9



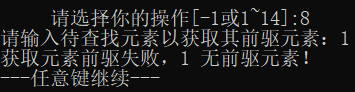
测试序号10



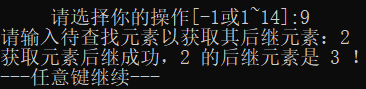
测试序号11



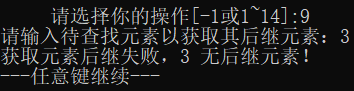
测试序号12



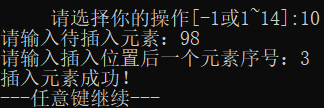
测试序号13

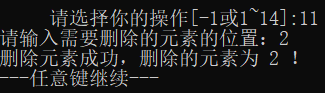


测试序号14

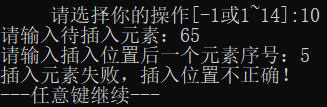


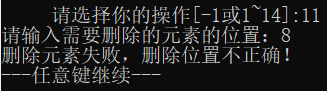
测试序号15



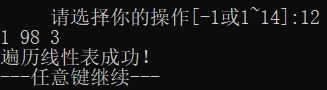


测试序号16

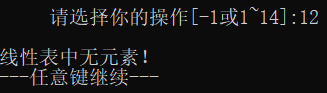




测试序号17



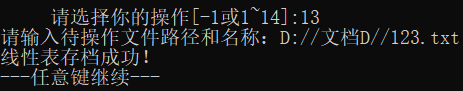
测试序号18

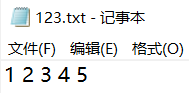


测试序号19

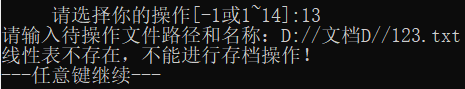
图1-9 单线性表操作菜单测试结果

线性表文件操作菜单测试结果如图1-10所示。

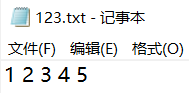


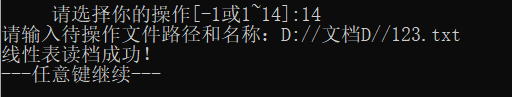


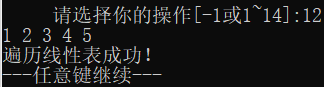
测试序号1



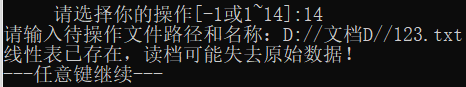
测试序号2







测试序号3



测试序号4

图1-10 线性表文件操作菜单测试结果

### 1.4.3 测试分析

根据实验要求，以测试计划的3个模块的测试结果为根本依据，将测试结果总结如下：

多线性表管理菜单测试：总计9项测试内容全部通过，通过率为100%。

单线性表操作菜单测试：总计19项测试内容全部通过，通过率为100%。

文件操作菜单测试：总计4项内容全部通过，通过率为100%。

## 1.5 实验小结

本次实验包含17个必需函数，使用switch对函数功能进行选择，还要通过if和while进行菜单层级改变，代码本身会非常长，这时良好的代码风格如正确的缩进、适当的注释等就会有利于我们自己检查代码。我按照操作序号在每个函数前添加了注释，以数字作为标识会比文字更易于查找到需要检查的函数；对每个函数功能实现的描述也有利于我及时了解某个函数需要用户输入使用的参数、返回值等等。

在检查和调试的过程中，遇到的最重要的问题就是两级菜单中线性表的传递的问题。我由于不够细致，在菜单设计的之初没有注意到线性表需要传递，导致进入二级菜单后线性表中的内容要么保留上一次操作的线性表内容，要么被清空，主菜单中线性表集合也有前一个线性表被后一个线性表覆盖的问题（集合长度没有传递好）。在发现问题后又忽略了参数传递的特性，直接使用类似L=List.elem[i].L的语句，在首次进入二级菜单时可以正常对线性表进行操作，但是退出二级菜单后线性表中元素并不能保存到集合中对应的线性表中，再次进入时线性表会被清空。部分同学使用了直接将线性表集合中线性表作为各函数参数的办法传递线性表，但是这样意味着每一个函数的参数及函数内部都需要做调整，非常繁琐。最后我选择了创建一个二级菜单函数，并单独设置主菜单进入二级菜单的入口（原本是将-2作为对应进入二级菜单和单线性表操作的入口），将二级菜单中所有操作选择放进该函数中，同时将线性表集合中下标索引的线性表作为参数传递进入该函数中，参数名称与其他各函数名称相同（SqList L），这样只需微调新定义的函数即可实现操作。

这次实验的菜单设计比较考验耐心和细心，要做出优秀的菜单系统需要我们有良好的系统设计思考和代码实现、调试的能力，让我认识到了自己在系统思考、细节把握上的不足。希望在以后的学习中可以不断改进和完善自己。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

本次实验的主要内容是采用单链表作为线性表的物理结构，实现线性表抽象

数据类型（Abstract Data Type，缩略名 ADT）的基本运算，构造一个具有菜单的

功能演示系统，并要求实现线性表的文件形式保存和线性表管理。本实验将以上述内容为主要目标，使用 C++语言构建问题解决方案。

### 2.1.1 问题1：线性表ADT的单链表实现

根据实验要求，本次实验中要以单链表作为线性表的物理结构，通过函数形式定义和实现线性表的基本功能，掌握和了解线性表的代码实现和功能操作思想。基本运算包括初始化表、销毁表、清空表、判空表、求表长、获取元素、查找元素、获取前驱元素、获取后继元素、插入元素、删除元素、遍历表等12种，基本可以囊括线性表的各项基本功能和管理。

### 2.1.2 问题2：实现线性表的文件形式保存和读取

根据实验要求，实验者需要实现将线性表内容保存在指定文件中、将指定文件中的数据内容读取到线性表中两种操作。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。

### 2.1.3 问题3：实现多线性表管理

根据实验要求，实验者要实现多个线性表管理，包括线性表添加、线性表删除、线性表查找等3个功能。线性表集合中对每个线性表进行唯一命名，便于选择进入对某个线性表的操作；线性表集合应当设置上限，达到上限后不能再进行添加线性表操作；对不存在的线性表进行删除或查找操作应返回操作错误提示信息；对命名已存在的线性表实行添加操作时返回操作错误提示。

### 2.1.4 问题4：构造具有菜单的功能演示系统

根据实验要求，需要实验者自行设计和构造可以完整展现线性表ADT功能的演示系统，可以通过用户选择实现对线性表的基本功能操作。演示系统中应包括功能选项、用户需要输入内容提示、操作结果提示等，便于用户直接使用和操作系统，进行合法输入。同时，因为演示系统包括多线性表管理和单线性表功能操作，应当设计好主菜单和子菜单的层级关系，使用户操作结果可以在两层菜单中传递和保存。

## 2.2 系统设计

根据实验要求，系统设计主要包括以下几个方面。

（1）设计单链表实现下的线性表ADT的物理结构。

（2）设计文件格式。

（3）设计多线性表管理模式。

（4）设计具有菜单的功能演示系统。

### 2.2.1 单链表实现下线性表ADT的物理结构设计

单链表ADT的物理结构如下：

|  |
| --- |
| typedef struct LNode{ //单链表（链式结构）结点的定义  ElemType data; //单链表数据元素存放  struct LNode \*next; //单链表下一结点指向  }LNode,\*LinkList; |

此处使用预定义的方式实现线性表ADT的物理结构，便于后续对线性表的定义和使用。

### 2.2.2 文件格式设计

实验中使用二进制文件格式进行对线性表中数据元素的存储和读取，有利于数据形式在文件中保存和读取到线性表中。

### 2.2.3 多线性表管理模式设计

多线性表存放管理模式定义如下：

|  |
| --- |
| typedef struct{ //线性表的管理表定义  struct { char name[30]; //线性表名称  LinkList L; //线性表头指针  } elem[10]; //存放线性表的集合  int length; //线性表集合的长度  int listsize; //线性表集合分配的空间  }LISTS; |

使用数组方式建立线性表集合，同时记录线性表集合长度和空间，便于对多线性表的管理和选择。

### 2.2.4 具有菜单功能的演示系统设计

在演示系统中，为实现多线性表管理和单线性表操作，设计多级菜单和多、单线性表系统。演示系统结构如图2-1所示。



图2-1 演示系统结构

演示系统包括在多线性表选择下的主菜单、二级菜单与临时单线性表选择。多线性表管理菜单包含添加、删除、查找、进入二级菜单、退出5个选项，进入二级菜单前需要输入待操作的线性表名称，如该线性表不存在则返回错误提示。二级菜单包含线性表的12个基本功能操作和线性表的文件操作。临时单线性表选项下可以创建不在线性表集合中的无命名的临时线性表，进入选项后可进行对线性表的12个基本功能操作和线性表的文件操作。

针对系统各层级的进入和退出、线性表功能的选择，通过switch语句，由用户输入数字化选择op（演示系统中有对应功能提示表），根据用户输入进入不同层级或功能选择。在主函数中针对需要获取用户输入的函数设计对应提示，以便用户知晓需要输入以实现函数正常功能的内容；针对正常情况（或合法输入）和异常情况（或非法输入）设计对应提示，以便用户知晓操作是否成功及操作结果，有利于用户下一步操作。

## 2.3 系统实现

本次实验使用的开发环境为Microsoft Visual Studio Code 1.56.2。

系统实现方面主要包括如下内容。

（1）项目工程说明。

（2）线性表ADT运算的单链表实现。

（3）文件输入输出实现。

（4）多线性表管理实现。

（5）演示系统实现。

### 2.3.1 项目工程说明

项目工程包含4个标准库头文件：stdio.h、stdlib.h、malloc.h、string.h。

项目工程中预定义的常量和类型声明如下：

|  |
| --- |
| #define TRUE 1  #define FALSE 0  #define OK 1  #define ERROR 0  #define INFEASIBLE -1  #define OVERFLOW -2 //函数返回值提示  typedef int status; //函数返回结果标志  typedef int ElemType; //数据元素类型定义 |

### 2.3.2 线性表ADT运算的单链表实现

相关常量定义和单链表LNode、单链表指针LinkList定义如下：

|  |
| --- |
| #define LIST\_INIT\_SIZE 100  #define LISTINCREMENT 10  typedef int ElemType;  typedef struct LNode{ //单链表（链式结构）结点的定义  ElemType data; //单链表数据元素存放  struct LNode \*next; //单链表下一结点指向  }LNode,\*LinkList; |

使用的自定义函数如表2-1：

表2-1 自定义线性表函数表

|  |  |  |
| --- | --- | --- |
| 序号 | 函数定义 | 函数功能 |
| 1 | status InitList(LinkList &L) | 初始化线性表 |
| 2 | status DestroyList(LinkList &L) | 销毁线性表 |
| 3 | status ClearList(LinkList &L) | 清空线性表 |
| 4 | status ListEmpty(LinkList L) | 线性表判空 |
| 5 | int ListLength(LinkList L) | 求线性表表长 |
| 6 | status GetElem(LinkList L,int i,ElemType &e) | 获取元素 |
| 7 | status LocateElem(LinkList L,ElemType e) | 查找元素 |
| 8 | status PriorElem(LinkList L,ElemType e,ElemType &pre) | 获取前驱元素 |
| 9 | status NextElem(LinkList L,ElemType e,ElemType &next) | 获取后继元素 |
| 10 | status ListInsert(LinkList &L,int i,ElemType e) | 插入元素 |
| 11 | status ListDelete(LinkList &L,int i,ElemType &e) | 删除元素 |
| 12 | status ListTraverse(LinkList L) | 遍历线性表 |
| 13 | status SaveList(LinkList L,char FileName[]) | 线性表存档 |
| 14 | status LoadList(LinkList &L,char FileName[]) | 线性表读档 |
| 15 | status AddList(LISTS &Lists,char ListName[]) | 增加线性表 |
| 16 | status RemoveList(LISTS &Lists,char ListName[]) | 删除线性表 |
| 17 | int LocateList(LISTS Lists,char ListName[]) | 查找线性表 |
| 18 | void SecondaryTable(LinkList &L,int op) | 进入二级菜单 |
| 19 | status InitInput(LinkList &L) | 线性表数据输入 |

**1.初始化线性表**

函数功能概述：线性表L不存在，构造一个空的线性表，返回OK，否则返回INFEASIBLE。

具体实现：

（1）由头指针L判断线性表是否存在，存在则返回INFEASIBLE。

（2）若线性表不存在即头指针指向为空，用malloc函数为头结点分配空间，返回OK。

复杂度分析：时间复杂度：O（1）；空间复杂度：O（1）。

**2.销毁线性表**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，定义指针p以遍历线性表，p初始指向头指针后一个空间，每次将p的后一个空间连接到L后以防数据丢失，释放p指向空间，清除指针值，再将p指向原先p的后一个空间。

（3）释放头指针指向空间，将L指针值清除为0。返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**3. 清空线性表**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，定义指针p以遍历线性表，p初始指向头指针后一个空间，每次将p的后一个空间连接到L后以防数据丢失，释放p指向空间，再将p指向原先p的后一个空间。返回OK。

（注意：销毁线性表与清空线性表的区别在于，清空线性表只需将各结点指向的空间释放，不需将指针值置为NULL，同时头结点空间也应该保留）

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**4.线性表判空**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，判断头结点的后一结点是否不存在，是则返回TRUE，否则返回FALSE。

复杂度分析：时间复杂度：O（1）；空间复杂度：O（1）。

**5.求线性表表长**

函数功能概述：如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，定义表长len=0，表指针t=L，若t的下一结点存在，则len++。返回len。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**6.获取元素**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，根据用户输入元素逻辑位置i，若i<1，元素位置不合法，返回ERROR。否则，定义结构体指针t=L，for循环每次使i--，t指向下一结点。

（3）当i=0时，t指向的元素即为所求元素，e=t->data，返回OK；若i减至0之前t的后一个结点为空，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。



图2-2 获取元素流程图

**7.查找元素**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，根据用户输入待查找元素e，定义i=0，结构体指针t=L，for循环每次使i++，t指向下一结点。

（3）若e=t->data，返回i；若遍历线性表后仍未查找到线性表中与e相等的元素，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**8.获取前驱元素**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，根据用户输入待查找元素e，定义i=0，结构体指针t=L，for循环每次使i++，t指向下一结点.

（3）若e=t->next->data且i!=0，pre=t->data，返回OK；若遍历线性表后仍未查找到线性表中与e相等的元素，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。



图2-3 获取前驱元素流程图

**9.获取后继元素**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，根据用户输入待查找元素e，定义i=0，结构体指针t=L，for循环每次使i++，t指向下一结点。

（3）若e=t->data，next=t->next->data，返回OK；若遍历线性表后仍未查找到线性表中与e相等的元素，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。



图2-4 获取后继元素流程图

**10.插入元素**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，根据用户输入待查找元素e和待插入位置i，定义j=0，结构体指针t=L，for循环每次使j++，t指向下一结点。

（3）若j+1=i，定义结构体指针tag，开辟新的单个结点存储空间，将元素e存储到tag指向结点中。将tag链到线性表中t的后一结点位置，返回OK；若遍历线性表后仍未查找到线性表中i的前一位置，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。



图2-5 插入元素流程图

**11.删除元素**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，根据用户输入待删除元素位置i，定义j=0，结构体指针t=L，for循环每次使j++，t指向下一结点。

（3）若j+1=i，定义结构体指针tag=t->next，将t->next->data保存到e中，将t与t->next->next链接，释放tag所指向空间，返回OK；若遍历线性表后仍未查找到线性表中i的前一位置，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。



图2-6 删除元素流程图

**12.遍历线性表**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，定义flag=0，结构体指针t=L->next，for循环每次使t指向下一结点，可进入for循环则将flag置为1。当t->next不为空时，每次打印t->data。

（3）退出循环后若flag=1，返回OK；否则返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**13.输入线性表元素**

函数功能概述：选择是否向线性表中输入数据，是则由用户输入以0为结束标志的一组数据，返回OK，否则返回0。

具体实现：

用户选择是否输入数据。若选择输入，创建先进先出链表接收用户输入，至用户输入0退出函数；否则直接退出函数。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

### 2.3.3 文件输入输出实现

线性表的逻辑结构较简单，只需要将线性表中元素依次写入到文件中或依次读取文件中元素，即可实现文件的输入输出。

**1.线性表存档**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，不存在则返回INFEASIBLE。

（2）若线性表存在，定义结构体指针t=L->next，文件指针fp。以二进制写入方式打开文件。while循环每次将线性表中t->data中元素写入文件，t=t->next。当t为空时结束循环，关闭文件，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**2.线性表读档**

函数功能概述：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。

具体实现：

（1）由头指针L判断线性表是否存在，存在则返回INFEASIBLE。

（2）若线性表不存在，为头结点L开辟新的存储空间，L->next=NULL。定义结构体指针t和文件指针fp。以二进制读取方式打开文件，为L->next分配空间，t=t->next，t->next=NULL。

（3）while循环每次将文件中的数据读取为t->data，文件未读取结束前每次为t->next分配空间，t指向下一结点。

（4）退出循环后将t后一结点置为NULL，关闭文件，返回OK。

### 2.3.4 多线性表管理实现

多线性表管理主要包括对线性表集合的线性表进行增加、删除、查找操作，同时包括实验员为从一级菜单进入二级菜单使用的进入单线性表操作函数。对多线性表的管理使用数组实现。

**1.增加线性表**

函数功能概述：在Lists中增加一个名称为ListName的空线性表，线性表数据由主函数插入。

具体实现：

（1）判断线性表集合长度Lists.length是否小于线性表集合空间Lists.listsize，否则返回OVERFLOW。

（2）若Lists.length<Lists.listsize，根据用户输入的线性表名称ListName，遍历线性表集合用strcmp函数判断是否有重名线性表，有则用flag标记并退出循环。

（3）若未被标记，将用户输入名称通过strcpy函数复制到线性表集合最后一位，为对应线性表头结点分配空间，头结点后一结点置为NULL，集合长度Lists.length++，返回OK；若被标记，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**2.删除线性表**

函数功能概述：在Lists中删除一个名称为ListName的线性表。

具体实现：

（1）根据用户输入的线性表名称ListName，用数组下标i遍历线性表集合用strcmp函数判断是否有重名线性表，有则用flag标记并退出循环。

（2）若被标记，调用销毁函数销毁当前线性表，从i位置起依次将集合中后一个线性表覆盖前一个，若Lists.elem[Lists.length-1]的线性表未被销毁，调用销毁函数销毁之，将其名称置为空，头结点置为NULL，释放其头结点，线性表长度Lists.length--，返回OK；若未被标记，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**3.查找线性表**

函数功能概述：在Lists中查找一个名称为ListName的线性表，查找成功返回逻辑序号，否则返回ERROR。

具体实现：

（1）根据用户输入的线性表名称ListName，用数组下标i遍历线性表集合用strcmp函数判断是否有重名线性表，有则用flag标记并退出循环。

（2）若未被标记或线性表集合长度为0，返回ERROR；否则返回i+1（线性表在集合中的逻辑序号）。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**4.进入二级菜单**

函数功能概述：将所选的多线性表集合中的待操作线性表传递进入二级菜单中，实现对单线性表的基本操作。

具体实现：

使用while函数实现循环输入，switch函数进行功能选择以实现线性表基本操作。

### 2.3.5 演示系统实现

主菜单演示界面设计如下：

|  |
| --- |
| system("cls"); printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("--------------------------------------------------------------\n");  printf(" -1.MainTable/MultiList -2.SingleList\n");  printf(" -3.Exit\n");  printf("--------------------------------------------------------------\n");  printf(" 请选择所在的菜单层级：\n");  printf(" -1：主菜单实现多线性表管理，-2：临时单线性表基本操作。\n");  scanf("%d",&op); |

多线性表管理界面设计如下：

|  |
| --- |
| system("cls"); printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("---------------------------------------------------------------------\n");  printf(" 15.AddList 16.RemoveList 17.LocateList\n");  printf(" 18.SecondaryTable\n");  printf(" 0. Exit\n");  printf("---------------------------------------------------------------------\n");  for(i=0;i<5;i++)  {  if(Lists.elem[i].L==NULL&&Lists.elem[i+5].L==NULL)  printf("\t线性表%d：未创建 \t线性表%d：未创建",i+1,i+6);  else if(Lists.elem[i].L==NULL&&Lists.elem[i+5].L!=NULL)  printf("\t线性表%d：未创建 \t线性表%d：%s",i+1,i+6,Lists.elem[i+5].name);  else if(Lists.elem[i].L!=NULL&&Lists.elem[i+5].L==NULL)  printf("\t线性表%d：%s \t线性表%d：未创建",i+1,Lists.elem[i].name,i+6);  else  printf("\t线性表%d：%s \t线性表%d：%s",i+1,Lists.elem[i].name,i+6,Lists.elem[i+5].name);  printf("\n");  }  printf("\n 请选择你的操作[0或15~18]:"); |

单线性表操作界面设计如下：

|  |
| --- |
| system("cls"); printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("-------------------------------------------------------------------------\n");  printf(" 1. InitList 7. LocateElem 13.SaveList\n");  printf(" 2. DestroyList 8. PriorElem 14.LoadList\n");  printf(" 3. ClearList 9. NextElem\n");  printf(" 4. ListEmpty 10. ListInsert\n");  printf(" 5. ListLength 11. ListDelete\n");  printf(" 6. GetElem 12. ListTrabverse\n");  printf(" 0. Exit\n");  printf("-------------------------------------------------------------------------\n");  printf(" 请选择你的操作[0~14]:"); |

## 2.4 系统测试

根据实验要求，系统测试主要包括以下内容。

（1）测试计划。

（2）测试结果。

（3）测试分析。

### 2.4.1 测试计划

测试计划包含3个模块：多线性表管理菜单测试，单线性表操作菜单测试，文件操作测试。

测试模块1：多线性表管理菜单测试（如表2-2所示）。

表2-2 多线性表管理菜单测试内容

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试序号 | 测试前总体情况描述 | 测试描述 | 测试操作 | 预计输出 |
| 1 | 线性表集合有线性表 | 测试添加线性表功能 | 选择添加表，提供合法输入 | 添加线性表成功，线性表集合菜单尾部显示新添加的线性表名称，提示是否输入线性表元素 |
| 2 | 线性表集合中有3个线性表 | 测试添加线性表功能 | 选择添加表，输入已存在的表名称 | 添加线性表失败，提示线性表已存在 |
| 3 | 线性表集合已满 | 测试添加线性表功能 | 选择添加表，提供合法输入 | 添加线性表失败，提示线性表集合已满 |
| 4 | 线性表集合有3个线性表 | 测试删除线性表功能 | 选择删除表，删除全部线性表 | 删除线性表成功，线性表集合菜单显示所有线性表均未创建 |
| 5 | 线性表集合有3个线性表 | 测试删除线性表功能 | 选择删除表，输入集合中不存在的线性表名称 | 删除线性表失败，提示集合中无此线性表 |
| 6 | 线性表集合有3个线性表 | 测试查找线性表功能 | 选择查找表，查找第3个线性表 | 查找线性表成功，提示线性表在集合中位置 |
| 7 | 线性表集合有3个线性表 | 测试查找线性表功能 | 选择查找表，查找不存在于集合中的表 | 查找线性表失败，提示集合中无此线性表 |
| 8 | 线性表集合有3个线性表 | 测试进入二级菜单功能 | 选择进入二级菜单，选择不存在于集合中的表名称 | 进入二级菜单失败，提示集合中无此线性表 |

测试模块2：单线性表操作菜单测试（如表2-3所示）。

表2-3 单线性表操作菜单测试内容

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试序号 | 测试前总体情况描述 | 测试描述 | 测试操作 | 预计输出 |
| 1 | 多线性表进入单线性表操作，线性表中有初始元素 | 测试初始化功能 | 选择初始化，提供合法输入 | 初始化失败，提示线性表已存在 |
| 2 | 线性表已存在 | 测试销毁功能 | 选择销毁 | 销毁成功，提示线性表被销毁 |
| 3 | 线性表已被销毁 | 测试销毁功能 | 选择销毁 | 销毁成功，提示线性表不存在 |
| 4 | 线性表已被销毁 | 测试初始化、遍历功能 | 选择初始化，提供与原线性表数据不同的合法输入；选择遍历 | 初始化成功，提示创建新线性表，提示是否输入线性表元素；遍历成功，显示后一次输入的数据元素 |
| 5 | 表中有元素 | 测试清空功能 | 选择清空 | 清空成功 |
| 6 | 表中有元素 | 测试判空功能 | 选择判空 | 显示线性表为空 |
| 7 | 表中无元素 | 测试求表长功能 | 选择求表长 | 求表长成功，显示表长 |
| 8 | 表中有元素 | 测试获取元素功能 | 选择获取元素，提供合法输入 | 获取元素成功，显示位置及元素 |
| 9 | 表中有元素 | 测试获取元素功能 | 选择获取元素，输入0或大于线性表长度的位置 | 获取元素失败，提示元素位置不合法 |
| 10 | 表中有元素 | 测试查找元素功能 | 选择查找元素，提供合法输入 | 查找元素成功，显示元素及位置 |
| 11 | 表中有元素 | 测试查找元素功能 | 选择查找元素，输入不在线性表中的元素 | 查找元素成功，提示元素不存在于线性表中 |
| 12 | 表中有元素 | 测试获取前驱元素功能 | 选择获取前驱元素，提供合法输入 | 获取前驱元素成功，显示待查找元素和前驱元素 |
| 13 | 表中有元素 | 测试获取前驱元素功能 | 选择获取前驱元素，输入首位元素数据 | 获取前驱元素失败，提示待查找元素无前驱元素 |
| 14 | 表中有元素 | 测试获取后继元素功能 | 选择获取后继元素，提供合法输入 | 获取后继元素成功，显示待查找元素和后继元素 |
| 15 | 表中有元素 | 测试获取后继元素功能 | 选择获取后继元素，输入末位元素数据 | 获取后继元素失败，提示待查找元素无后继元素 |
| 16 | 表中有元素 | 测试插入元素、删除元素功能 | 选择插入元素，提供合法输入；选择删除元素，提供合法输入 | 插入和删除元素成功 |
| 17 | 表中有元素 | 测试插入元素、删除元素功能 | 选择插入元素，输入不合法插入位置；选择删除元素，输入不在线性表中的元素 | 插入和删除元素失败 |
| 18 | 表中有元素 | 测试插入元素、删除元素、遍历功能 | 选择遍历 | 遍历成功，显示上次操作后线性表中元素顺序，符合插入和删除预期 |
| 19 | 表中无元素 | 测试遍历功能 | 选择遍历 | 提示线性表无元素 |

测试模块3：文件操作菜单测试（如表2-4所示）。

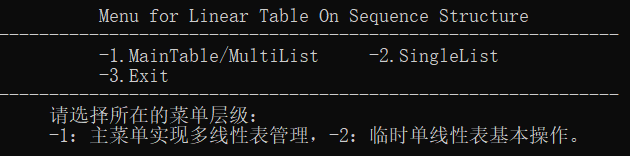
表2-4 文件操作菜单测试内容

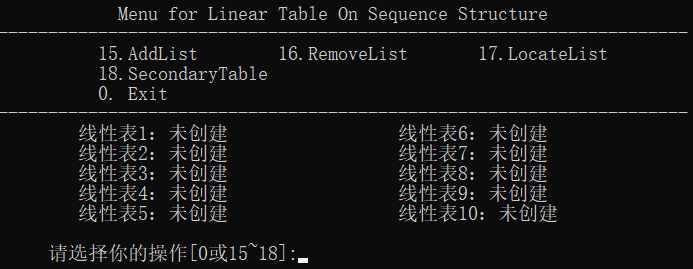
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试序号 | 测试前总体情况描述 | 测试描述 | 测试操作 | 预计输出 |
| 1 | 线性表存在 | 测试存档功能 | 选择存档 | 存档成功，在指定文件下可见线性表中数据 |
| 2 | 线性表不存在 | 测试存档功能 | 选择存档 | 存档失败，提示线性表不存在 |
| 3 | 线性表存在 | 测试读档功能 | 选择读档 | 读档失败，提示线性表已存在，读档会失去线性表中原有元素 |
| 4 | 线性表不存在 | 测试读档功能 | 选择读档 | 读档成功，遍历线性表可见线性表中元素与指定文件中相同 |

### 2.4.2 测试结果

根据测试计划，测试结果包含3个部分：多线性表管理菜单测试结果，单线性表操作菜单测试结果，文件操作测试结果。

各初始菜单选择界面如图2-7所示。





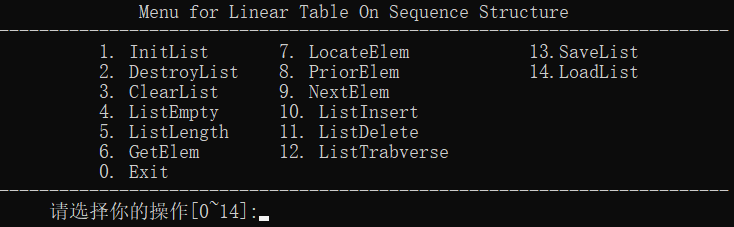
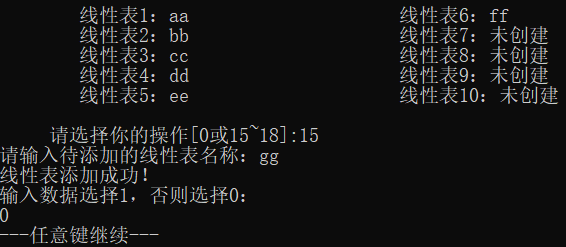


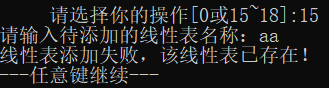
图2-7 线性表系统菜单界面展示

多线性表管理菜单测试结果如图2-8所示。

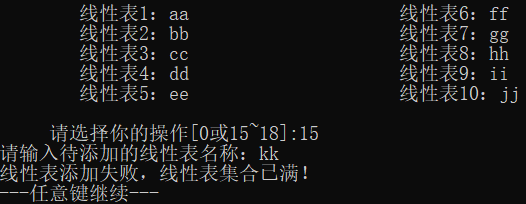




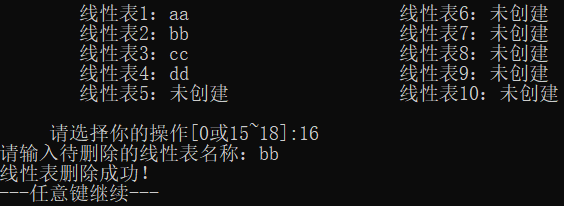
测试序号1

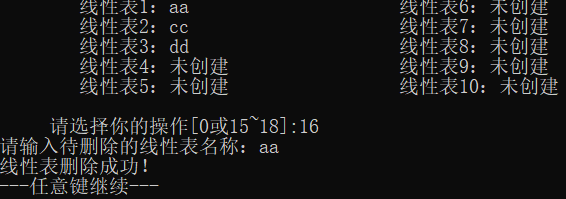


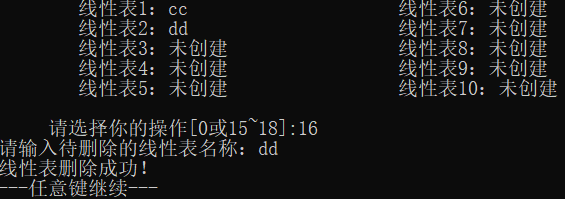
测试序号2

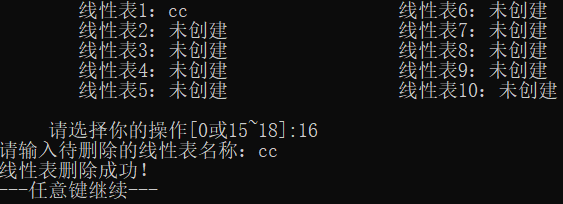


测试序号3

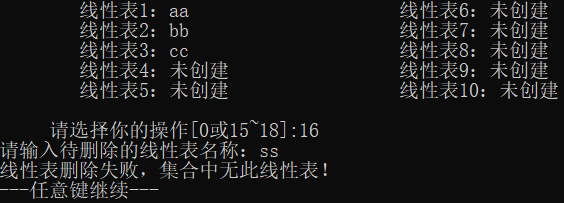




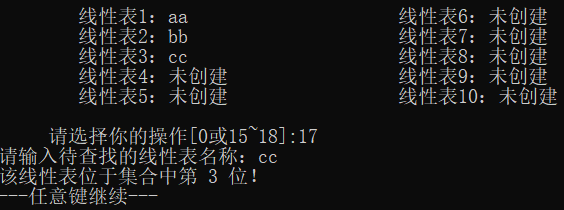




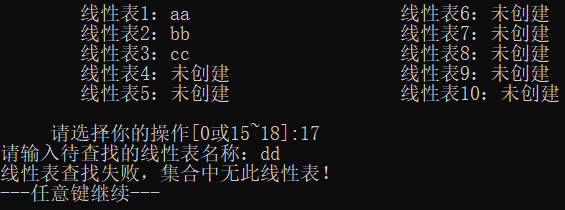
测试序号4



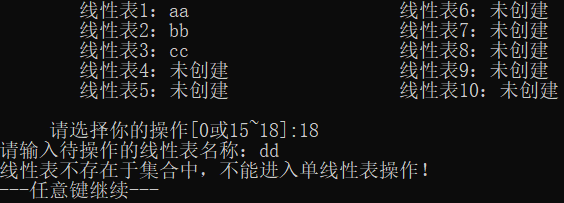
测试序号5



测试序号6



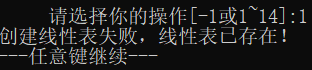
测试序号7



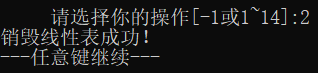
测试序号8

图2-8 多线性表管理菜单测试结果

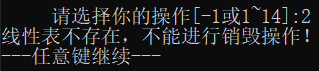
单线性表操作菜单测试结果如图2-9所示。



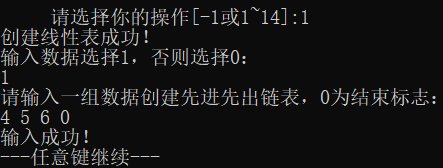
测试序号1

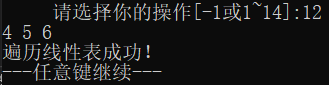


测试序号2

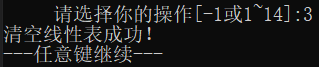


测试序号3

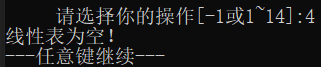




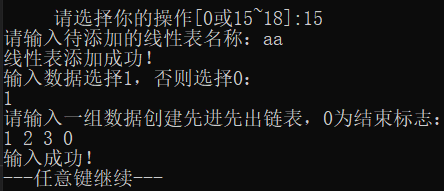
测试序号4



测试序号5

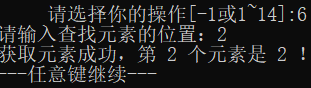


测试序号6

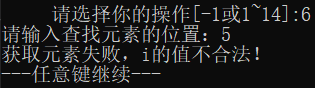




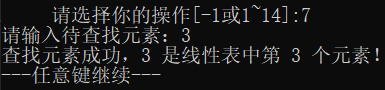
测试序号7



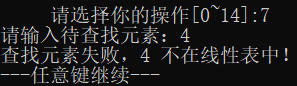
测试序号8



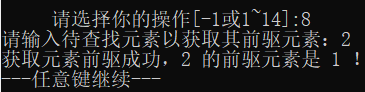
测试序号9



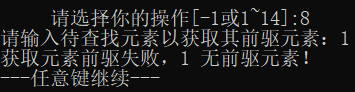
测试序号10



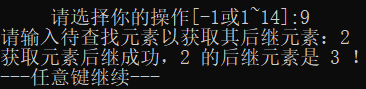
测试序号11



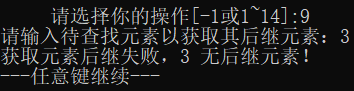
测试序号12



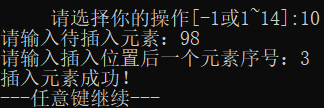
测试序号13

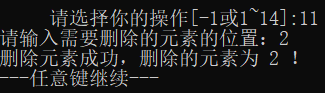


测试序号14

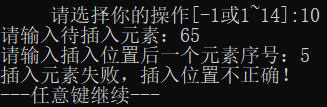


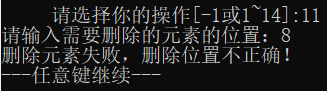
测试序号15



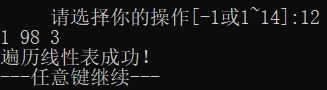


测试序号16

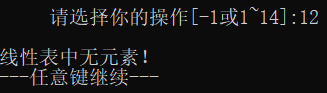




测试序号17



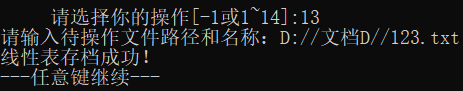
测试序号18

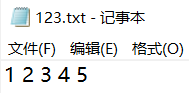


测试序号19

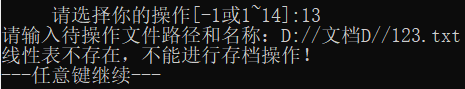
图2-9 单线性表操作菜单测试结果

线性表文件操作菜单测试结果如图2-10所示。

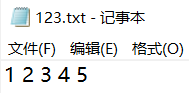


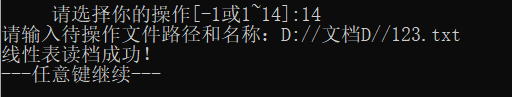


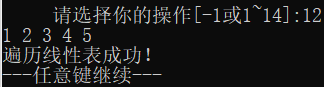
测试序号1



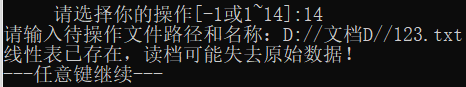
测试序号2







测试序号3



测试序号4

图2-10 线性表文件操作菜单测试结果

### 2.4.3 测试分析

根据实验要求，以测试计划的3个模块的测试结果为根本依据，将测试结果总结如下：

多线性表管理菜单测试：总计9项测试内容全部通过，通过率为100%。

单线性表操作菜单测试：总计19项测试内容全部通过，通过率为100%。

文件操作菜单测试：总计4项内容全部通过，通过率为100%。

## 2.5 实验小结

本次实验是第2次数据结构实验，在第1次以顺序表实现线性表操作的基础上改为用单链表实现。各模块操作及功能基本相同，但需要在语法上进行修改调整。基本可以套用第1次实验的框架，但是具体实现上也需要更加注意，比如销毁线性表时就需要将每个结点都释放，而不能只释放头结点空间。相对来说，在调试程序过程中痛苦减少了很多，于是有很多时候用来微调程序上不够完善的地方。在实验中同时发现了由多线性表管理进入单线性表操作时，如果将线性表销毁，返回多线性表管理界面可见线性表已不存在于集合中，但是线性表名称无法删除。即，如销毁名为aa的线性表后，再选择进入名为aa的线性表，仍可进入（正确结果应当报告线性表不存在）。最后发现只要在删除线性表函数中将末位元素名称的第0位置为’\0’即可。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

通过本次实验达到目的：

⑴加深对二叉树的概念、基本运算的理解；

⑵熟练掌握二叉树的逻辑结构与物理结构的关系；

⑶以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。

本次实验将实现二叉树抽象数据类型（Abstract Data Type，缩略名 ADT）的基本运算，构造一个具有菜单的功能演示系统，并要求实现二叉树的文件形式保存和二叉树管理。本实验将以上述内容为主要目标，使用 C++语言构建问题解决方案。

### 3.1.1 问题1：二叉树ADT的二叉链表实现

根据本次实验要求，本次实验中要以二叉链表作为二叉树的物理结构，通过函数形式定义和实现二叉树的基本功能，掌握和了解二叉树的代码实现和功能操作思想。基本运算包括创建二叉树、销毁二叉树、清空二叉树、二叉树判空、求二叉树深度、查找结点、结点赋值、获取兄弟结点、插入结点、删除结点、先序遍历、中序遍历、后序遍历、按层遍历等14种，基本可以囊括二叉树的各项基本功能和管理。

### 3.1.2 问题2：实现二叉树的文件形式保存和读取

根据实验要求，实验者需要实现将线性表内容保存在指定文件中、将指定文件中的数据内容读取到线性表中两种操作。其中，①需要设计文件数据记录格式，以高效保存二叉树各结点数据信息并便于查找；②需要设计二叉树文件保存和加载操作合理模式。

### 3.1.3 问题3：实现多二叉树管理

根据实验要求，实验者要实现多个二叉树管理，包括二叉树添加、二叉树删除、二叉树查找等3个功能。二叉树集合中对每个二叉树进行唯一命名，便于选择进入对某个二叉树的操作；二叉树集合应当设置上限，达到上限后不能再进行添加二叉树操作；对不存在的二叉树进行删除或查找操作应返回操作错误提示信息；对命名已存在的二叉树实行添加操作时返回操作错误提示。

### 3.1.4 问题4：构造具有菜单的功能演示系统

根据实验要求，需要实验者自行设计和构造可以完整展现二叉树ADT功能的演示系统，可以通过用户选择实现对二叉树的基本功能操作。演示系统中应包括功能选项、用户需要输入内容提示、操作结果提示等，便于用户直接使用和操作系统，进行合法输入。同时，因为演示系统包括多二叉树管理和单二叉树功能操作，应当设计好主菜单和子菜单的层级关系，使用户操作结果可以在两层菜单中传递和保存。

## 3.2 系统设计

根据实验要求，系统设计主要包括以下几个方面。

（1）设计二叉链表实现下的二叉树ADT的物理结构。

（2）设计文件格式。

（3）设计多二叉树管理模式。

（4）设计具有菜单的功能演示系统。

### 3.2.1 二叉链表实现下二叉树ADT的物理结构设计

二叉树ADT的物理结构如下：

|  |
| --- |
| typedef struct  {  KeyType key;//二叉树结点关键字定义  char others[20];//二叉树结点字符串  } TElemType; //二叉树结点类型定义  typedef struct BiTNode  {  TElemType data;  struct BiTNode \*lchild, \*rchild;//二叉树左右子树定义  } BiTNode, \*BiTree;//二叉链表结点的定义 |

此处使用预定义的方式实现二叉树ADT的物理结构，便于后续对二叉树的定义和使用。

### 3.2.2 文件格式设计

实验中使用二进制文件格式进行对二叉树中结点位置、关键字、内容的存储和读取，有利于数据形式在文件中保存和读取到二叉树中。

### 3.2.3 多二叉树管理模式设计

多二叉树存放管理模式定义如下：

|  |
| --- |
| typedef struct  { //二叉树的管理表定义  struct  {  char name[30]; //二叉树名称  BiTree T;//二叉树根结点指针  } elem[10]; //存放二叉树的集合  int length; //二叉树集合的长度  int listsize; //二叉树集合分配的空间  } LISTS; |

使用数组方式建立二叉树集合，同时记录二叉树集合长度和空间，便于对多二叉树的管理和选择。

### 3.2.4 具有菜单的功能演示系统设计

在演示系统中，为实现多二叉树管理和单二叉树操作，设计多级菜单和多、单二叉树系统。演示系统结构如图3-1所示。



图3-1 演示系统结构

演示系统包括在多二叉树选择下的主菜单、二级菜单与临时单二叉树选择。多二叉树管理菜单包含添加、删除、查找、进入二级菜单、退出5个选项，进入二级菜单前需要输入待操作的二叉树名称，如该二叉树不存在则返回错误提示。二级菜单包含二叉树的14个基本功能操作和二叉树的文件操作。临时单二叉树选项下可以创建不在二叉树集合中的无命名的临时二叉树，进入选项后可进行对二叉树的14个基本功能操作和二叉树的文件操作。

针对系统各层级的进入和退出、二叉树功能的选择，通过switch语句，由用户输入数字化选择op（演示系统中有对应功能提示表），根据用户输入进入不同层级或功能选择。在主函数中针对需要获取用户输入的函数设计对应提示，以便用户知晓需要输入以实现函数正常功能的内容；针对正常情况（或合法输入）和异常情况（或非法输入）设计对应提示，以便用户知晓操作是否成功及操作结果，有利于用户下一步操作。

## 3.3 系统实现

本次实验使用的开发环境为Microsoft Visual Studio Code 1.56.2。

系统实现方面主要包括如下内容。

（1）项目工程说明。

（2）二叉树ADT运算的二叉链表实现。

（3）文件输入输出实现。

（4）多二叉树管理实现。

（5）演示系统实现。

### 3.3.1 项目工程说明

项目工程包含4个标准库头文件：stdio.h、stdlib.h、math.h、string.h。

项目工程中预定义的常量和类型声明如下：

|  |
| --- |
| #define TRUE 1  #define FALSE 0  #define OK 1  #define ERROR 0  #define INFEASIBLE -1  #define OVERFLOW -2  #define Maxlength 100  typedef int status;  typedef int KeyType;  typedef struct  {  int pos;  TElemType data;  } DEF;//用于存放用户输入的位置、关键字、字符串信息 |

### 3.3.2 二叉树ADT运算的二叉链表实现

相关常量定义和单链表LNode、单链表指针LinkList定义如下：

|  |
| --- |
| #define LIST\_INIT\_SIZE 100  #define LISTINCREMENT 10  typedef struct LNode{ //单链表（链式结构）结点的定义  ElemType data; //单链表数据元素存放  struct LNode \*next; //单链表下一结点指向  }LNode,\*LinkList; |

使用的自定义函数如表3-1：

表3-1 自定义二叉树函数表

|  |  |  |
| --- | --- | --- |
| 序号 | 函数定义 | 函数功能 |
| 1 | status CreateBiTree(BiTree &T, DEF definition[]) | 创建二叉树 |
| 2 | status DestroyBiTree(BiTree &T) | 销毁二叉树 |
| 3 | status ClearBiTree(BiTree &T) | 清空二叉树 |
| 4 | status BiTreeEmpty(BiTree T) | 二叉树判空 |
| 5 | int BiTreeDepth(BiTree T) | 求二叉树深度 |
| 6 | BiTNode \*LocateNode(BiTree T, KeyType e) | 查找结点 |
| 7 | status Assign(BiTree T, KeyType e, TElemType value) | 结点赋值 |
| 8 | BiTNode \*GetSibling(BiTree T, KeyType e) | 获取兄弟结点 |
| 9 | status InsertNode(BiTree &T, KeyType e, int LR, TElemType c) | 插入结点 |
| 10 | status DeleteNode(BiTree &T, KeyType e) | 删除结点 |
| 11 | status PreOrderTraverse(BiTree T, void (\*visit)(BiTree)) | 先序遍历 |
| 12 | status InOrderTraverse(BiTree T, void (\*visit)(BiTree)) | 中序遍历 |
| 13 | status PostOrderTraverse(BiTree T, void (\*visit)(BiTree)) | 后序遍历 |
| 14 | status LevelOrderTraverse(BiTree T, void (\*visit)(BiTree)) | 层序遍历 |
| 15 | status SaveBiTree(BiTree T, char FileName[]) | 二叉树存档 |
| 16 | status LoadBiTree(BiTree T, char FileName[]) | 二叉树读档 |
| 17 | status AddList(LISTS &Lists, char ListName[]) | 增加二叉树 |
| 18 | status RemoveList(LISTS &Lists, char ListName[]) | 删除二叉树 |
| 19 | int LocateList(LISTS Lists, char ListName[]) | 查找二叉树 |
| 20 | void SecondaryTable(BiTree &T,int op) | 进入二级菜单 |
| 21 | BiTNode \*LocateTree(BiTree T, BiTree t); | 查找父亲结点 |
| 22 | void visit(BiTree T); | 访问结点 |

**1.创建二叉树**

函数功能概述：二叉树不存在，构造新的不为空的、关键字不重复的二叉树，成功则返回OK，若关键字重复则返回ERROR，若二叉树已存在则返回INFEASIBLE。

具体实现：

（1）由根结点T判断二叉树是否存在，存在则返回INFEASIBLE。

（2）若二叉树不存在，根据主函数中用户输入并存储在definition结构体数组中的位置、关键字、字符串，遍历definition，使用数组v标记关键字是否出现过（v数组初始值均为0，出现某关键字i后v[i]=1），若出现重复的关键字，返回ERROR。

（3）若无重复关键字，遍历definition，使用结点数组p，对每一位置的结点，使用malloc函数开辟存储空间并赋值，将其左右子树置为NULL。当结点位置不为1时，结点的父亲结点位置是该结点位置的1/2。若位置值为奇数，则该结点为其父亲结点的右子树，反之为左子树。据此构建二叉树。

（4）将根结点指针T指向p[1]，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。



图3-2 创建二叉树流程图

**2.销毁二叉树**

函数功能概述：将二叉树设置成空，并删除所有结点，释放结点空间,并释放头结点。

具体实现：

（1）若二叉树T不存在，返回INFEASIBLE。

（2）若二叉树存在，判断其左子树是否存在，存在则使用DestroyBiTree(T->lchild)进行递归，右子树同操作。

（3）递归进行到叶子结点时，释放此时的结点T空间，将T置为NULL，返回OK。再回退到递归上一层。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**3.清空二叉树**

函数功能概述：将二叉树设置成空，并删除所有结点，释放结点空间。

具体实现：

（1）若二叉树T不存在，返回INFEASIBLE。

（2）若二叉树存在，判断其左子树是否存在，存在则使用DestroyBiTree(T->lchild)进行递归，右子树同操作。

（3）递归进行到叶子结点时，释放此时的结点T空间，将T置为NULL，返回OK。再回退到递归上一层。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

（对二叉树而言，销毁和清空操作的结果相同，根结点最终都被置为NULL）

**4.二叉树判空**

函数功能概述：判断二叉树是否为空树，是则返回TRUE，否则返回FALSE。

具体实现：若二叉树T存在，返回TRUE，否则返回FALSE。

复杂度分析：时间复杂度：O（1）；空间复杂度：O（1）；

**5.求二叉树深度**

函数功能概述：递归求二叉树T深度并返回深度值。

具体实现：

（1）若当前结点T为空，返回0。

（2）若当前结点T不为空，递归使用lh、rh分别记录当前结点T左右子树深度，返回lh、rh中较大值加一。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。



图3-3 求二叉树深度流程图

**6.查找结点**

函数功能概述：e是和T中结点关键字类型相同的给定值；根据e查找符合条件的结点，返回该结点指针，如无关键字为e的结点，返回NULL。

具体实现：

（1）若当前结点T为空，返回NULL。

（2）若当前结点T不为空，判断当前结点T的关键字T->data.key是否与用户输入的e相同，相同则返回该结点。

（3）若不相同，定义结点指针p，置为空。若结点的左子树存在，递归查找左子树，p接收函数返回值。若结点的右子树存在且p不存在，递归查找右子树，p接收函数返回值。返回p。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。



图3-4 查找元素流程图

**7.结点赋值**

函数功能概述：e是和T中结点关键字类型相同的给定值；查找结点关键字等于e的结点，将结点值修改成value，返回OK（要求结点关键字保持唯一性）。如果查找失败，返回ERROR。

具体实现：

（1）若结点为空，返回NULL。

（2）若结点不为空，定义结点指针数组data、结点指针p，将data作为栈，top指向栈顶位置向上一个单元。

（3）将根结点进栈。top不为0时，由p指向栈顶元素。定义标记数组v记录关键字是否出现过。初始v中全部元素为0，出现一个关键字p->data.key时，将v[p->data.key]置为1。依先序遍历记录每个结点关键字。

（4）再次利用栈进行先序遍历，当栈顶元素p关键字与e相同时，若value.key未被标记或与p关键字相同，进行结点赋值操作，返回OK；否则返回ERROR。

复杂度分析：时间复杂度：O（n2）；空间复杂度：O（n）。

**8.获取兄弟结点**

函数功能概述：e是和T中结点关键字类型相同的给定值；查找结点关键字等于e的结点的兄弟结点，返回其兄弟结点指针。如果查找失败，返回NULL。

具体实现：

（1）根据用户输入的关键字e，当结点的左右子树均非空时，若左子树关键字T->lchild->data.key=e，返回右子树结点；若右子树关键字T->rchild->data.key=e，返回右子树结点。

（2）定义结点指针p=NULL，若当前结点左子树非空，对左子树进行递归获取，p接收返回值。若当前结点右子树非空且p为空，对右子树进行递归获取，p接收返回值。返回p。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**9.插入结点**

函数功能概述：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；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，返回OK。如果插入失败，返回ERROR。特别地，当LR为-1时，作为根结点插入，原根结点作为c的右子树。

具体实现：

（1）根据用户输入关键字e、待插入结点c的关键字和字符串、插入方式选择LR，定义结点指针p、t，使用查找结点函数LocateNode查找是否有与待插入结点c关键字相同的结点，有则返回ERROR。

（2）若原二叉树中无与待插入结点c关键字相同的结点，使用查找结点函数LocateNode查找关键字与e相同的结点，用p指向该结点。若p指向为空即二叉树中无此待操作结点，返回ERROR。

（3）若p非空，根据LR：

a.LR=0时，t指向p的左子树。使用malloc函数为p的左子树重新开辟空间，将c的关键字、字符串赋给新左子树结点，新左子树结点的左子树置为空，右子树置为t。

b.LR=1时，t指向p的右子树。使用malloc函数为p的右子树重新开辟空间，将c的关键字、字符串赋给新右子树结点，新右子树结点的左子树置为空，右子树置为t。

c.LR=-1时，使用malloc函数为t开辟空间，将c的关键字、字符串赋给t结点，t结点的左子树置为空，右子树置为原根结点T。

（4）返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**10.删除结点**

函数功能概述：e是和T中结点关键字类型相同的给定值。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。成功删除结点后返回OK，否则返回ERROR。

具体实现：

（1）根据用户输入的关键字e，使用查找结点函数LocateNode定位对应结点，用p指向该结点。

（2）若p为空，返回ERROR。

（3）若p非空，p指向根结点时，结点指针pre置为NULL，否则使用查找父亲结点函数LocateTree将pre置为p的父亲结点。

（4）根据p的左右子树情况：

a.p的左右子树均不存在：

若pre为NULL，不执行操作；

若pre的左子树为p，执行pre->lchild=NULL。

若pre的右子树为p，执行pre->rchild=NULL。

b.p的左右子树均存在：使结点指针tmp指向p的左子树的最右结点，并执行tmp->rchild=p->rchild。

若pre为NULL，将pre置为p的左子树，根结点赋给pre。

若pre的左子树为p，执行pre->lchild=p->lchild。

若pre的右子树为p，执行pre->rchild=p->lchild。

c.仅p的左子树存在：

若pre为NULL，将pre置为p的左子树，根结点赋给pre。

若pre的左子树为p，执行pre->lchild=p->lchild。

若pre的右子树为p，执行pre->rchild=p->lchild。

**11.先序遍历**

函数功能概述：先序遍历二叉树T，对每个结点访问一次且仅一次。

具体实现：

（1）定义结点指针p，结点数组data作为栈，定义top指向栈顶向上一个单元。

（2）若根结点为空，返回INFEASIBLE。否则，将根结点入栈。当top不为0时，p指向栈顶结点，访问p结点。当p的右子树不为空时，将右子树入栈。当p的左子树不为空时，将左子树入栈。循环直至top=0。返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。



图3-5 先序遍历流程图

**12.中序遍历**

函数功能概述：中序遍历二叉树T，对每个结点访问一次且仅一次。

具体实现：

（1）定义结点指针p，结点数组data作为栈，定义top指向栈顶向上一个单元。

（2）若根结点为空，返回INFEASIBLE。否则，p指向根结点T，进入do-while循环。当p不为空时进行while循环：若栈满，退出并返回OVERFLOW；否则将p进栈，p不断指向其左子树直至p为空退出循环。循环结束后，当top不为0时，p指向栈顶元素，访问p，p指向其右子树。当top=0且p为空时退出do-while循环。返回OK。

复杂度分析：时间复杂度：O（）；空间复杂度：O（）。



图3-6 中序遍历流程图

**13.后序遍历**

函数功能概述：后序遍历二叉树T，对每个结点访问一次且仅一次。

具体实现：

（1）定义结点指针p、结点指针pre，定义标记变量flag，结点数组data作为栈，定义top指向栈顶向上一个单元。

（2）若根结点为空，返回INFEASIBLE。否则，p指向根结点T，进入do-while循环。当p不为空时进行while循环：若栈满，退出并返回OVERFLOW；否则将p进栈，p不断指向其左子树直至p为空退出循环。循环结束后，置pre=NULL，flag=1，当top和flag均不为0时进入while循环：p指向栈顶结点，若p的右子树与pre相等，访问p，退栈，pre指向p；否则，p指向右子树，flag置为0，循环直至top=flag=0。当top=0时，退出do-while循环。返回OK。

复杂度分析：时间复杂度：O（）；空间复杂度：O（n）。



图3-7 后序遍历流程图

**14.层序遍历**

函数功能概述：按层遍历二叉树T，对每个结点访问一次且仅一次。

具体实现：

（1）定义结点指针栈data，出栈和入栈变量in、out记录出入栈的结点。

（2）将根结点入栈，in++。

（3）当in>out时，进入while循环。当data[out]存在时，访问结点data[out]。data[in++]依次存储data的左子树和右子树就诊。

**15.查找父亲结点**

函数功能概述：根据子树t查找其父亲结点，查找成功则返回父亲结点指针，否则返回NULL。

具体实现：

（1）若当前结点T为空，返回NULL。

（2）若当前结点T不为空，判断当前结点T的关键字左右孩子之一是否与待查找结点t相同，相同则返回该结点T。

（3）若不相同，定义结点指针p，置为空。若结点的左子树存在，递归查找左子树，p接收函数返回值。若结点的右子树存在且p不存在，递归查找右子树，p接收函数返回值。返回p。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**16.访问结点**

函数功能概述：对当前结点执行特定操作。

具体实现：打印当前结点的关键字和字符串。

复杂度分析：时间复杂度：O（1）；空间复杂度：O（1）。

### 3.3.3 文件输入输出实现

文件存取二叉树时，选择依次存储二叉树位置、关键字、字符串，位置由遍历二叉树时i确定，根据二叉树排列特定可确定每一结点位置。存储位置可使读档时构造二叉树更便捷。

**1.二叉树存档**

函数功能概述：将二叉树的结点数据写入到文件FileName中。

具体实现：

（1）若T不存在，返回INFEASIBLE。

（2）若T存在，定义结点指针栈node，将node[1]指向根结点T。定义depth记录二叉树深度。

（3）若文件不可打开，返回ERROR。

（4）当文件可打开时，通过while循环使i从1开始至指示结点位置， 当node[i]不为NULL时，将i、node[i]->data.key、node[i]->data.others写入文件中。当前结点的左子树node[i]->lchild不为空时，执行node[2\*i]=node[i]->lchild；当前结点的右子树node[i]->rchild不为空时，执行node[2\*i+1]=node[i]->lchild。

（5）循环结束后关闭文件，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**2.二叉树读档**

函数功能概述：读入文件FileName的结点数据，创建二叉树。

具体实现：

（1）若T不为空，返回INFEASIBLE。

（2）若T为空，定义node指针栈，将栈全部置为NULL。定义int型变量a、b和char型数组string暂存文件中位置、关键字、字符串。

（3）当文件未读取结束时，进行while循环，为node[a]开辟空间，node[a]的关键字node[a]->data.key置为b，复制字符串string至node[a]->data.others中，node[a]的左右子树置为NULL。

（4）i从2至Maxlength进行循环，当node[i/2]不为空时，若i为奇数，执行node[i/2]->rchild=node[i]；若i为偶数，执行node[i/2] ->lchild=node[i]。

（5）循环结束，将根结点T指向node[1]，关闭文件，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

### 3.3.4 多二叉树管理实现

多二叉树管理主要包括对二叉树集合的二叉树进行增加、删除、查找操作，同时包括实验员为从一级菜单进入二级菜单使用的进入单二叉树操作函数。对多二叉树的管理使用数组实现。

**1.增加二叉树**

函数功能概述：在Lists中增加一个名称为ListName的二叉树。

具体实现：

（1）判断二叉树集合长度Lists.length是否小于二叉树集合空间Lists.listsize，否则返回OVERFLOW。

（2）若Lists.length<Lists.listsize，根据用户输入的二叉树名称ListName，遍历二叉树集合用strcmp函数判断是否有重名二叉树，有则用flag标记并退出循环，返回ERROR。

（3）若无重名二叉树，将二叉树名称ListName赋给集合末位二叉树，循环将二叉树位置、关键字、字符串入栈definition，直至位置为0时退出循环。

（4）使用CreateBiTree函数判断二叉树是否创建成功，成功则集合长度Lists.length++，返回OK，否则返回INFEASIBLE。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**2.删除二叉树**

函数功能概述：在Lists中删除一个名称为ListName的二叉树。

具体实现：

（1）根据用户输入ListName，使用i遍历集合，使用strcmp函数判断是否有重名二叉树，有则用flag标记并退出循环。

（2）若未被标记，说明集合中无此二叉树，返回ERROR。

（3）若被标记，调用销毁函数将当前二叉树销毁，从i起至Lists.length-1，将集合中所有二叉树元素前移一位。若Lists.elem[Lists.length-1]的二叉树未被销毁，调用销毁函数销毁之。将Lists.elem[Lists.length-1]的名称置为空，二叉树置为空，释放该数组元素空间，集合长度Lists.length--。返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度（n）。

**3.查找二叉树**

函数功能概述：在Lists中查找一个名称为ListName的二叉树，查找成功返回逻辑序号，否则返回ERROR。

具体实现：

（1）根据用户输入的二叉树名称ListName，用数组下标i遍历二叉树集合用strcmp函数判断是否有重名二叉树，有则用flag标记并退出循环。

（2）若未被标记或二叉树集合长度为0，返回ERROR；否则返回i+1（线性表在集合中的逻辑序号）。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**4.进入二级菜单**

函数功能概述：将所选的多二叉树集合中的待操作二叉树传递进入二级菜单中，实现对单二叉树的基本操作。

具体实现：

使用while函数实现循环输入，switch函数进行功能选择以实现二叉树基本操作。

### 3.3.5 演示系统实现

主菜单演示界面设计如下：

|  |
| --- |
| system("cls");  printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("--------------------------------------------------------------\n");  printf(" -1.MainTable/MultiBiTree -2.SingleBiTree\n");  printf(" -3.Exit\n");  printf("--------------------------------------------------------------\n");  printf(" 请选择所在的菜单层级：\n");  printf(" -1：主菜单实现多二叉树管理，-2：临时单二叉树基本操作。\n"); |

多二叉树管理界面设计如下：

|  |
| --- |
| system("cls");  printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("---------------------------------------------------------------------\n");  printf(" 17.AddBiTree 18.RemoveBiTree 19.LocateBiTree\n");  printf(" 20.SecondaryTable\n");  printf(" 0. Exit\n");  printf("---------------------------------------------------------------------\n");  for (i = 0; i < 5; i++)  {  if (Lists.elem[i].T == NULL && Lists.elem[i + 5].T == NULL )  printf("\t二叉树%d：未创建 \t二叉树%d：未创建", i + 1, i + 6);  else if (Lists.elem[i].T == NULL && Lists.elem[i + 5].T != NULL)  printf("\t二叉树%d：未创建 \t二叉树%d：%s", i + 1, i + 6, Lists.elem[i + 5].name);  else if (Lists.elem[i].T != NULL && Lists.elem[i + 5].T == NULL)  printf("\t二叉树%d：%s \t二叉树%d：未创建", i + 1, Lists.elem[i].name, i + 6);  else  printf("\t二叉树%d：%s \t二叉树%d：%s", i + 1, Lists.elem[i].name, i + 6, Lists.elem[i + 5].name);  printf("\n");  }  printf("\n 请选择你的操作[0或17~20]:"); |

单二叉树操作界面设计如下：

|  |
| --- |
| system("cls");  printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("---------------------------------------------------------------------------------------\n");  printf(" 1. CreateBiTree 7. Assign 13. PostOrderTraverse\n");  printf(" 2. DestroyBiTree 8. GetSibling 14. LevelOrderTraverse\n");  printf(" 3. ClearBiTree 9. InsertNode 15. SaveBiTree\n");  printf(" 4. BiTreeEmpty 10. DeleteNode 16. LoadBiTree\n");  printf(" 5. BiTreeDepth 11. PreOrderTraverse\n");  printf(" 6. LocateNode 12. InOrderTraverse\n");  printf(" 0. Exit\n");  printf("---------------------------------------------------------------------------------------\n");  printf(" 请选择你的操作[0~16]:"); |

## 3.4 系统测试

根据实验要求，系统测试主要包括以下内容。

（1）测试计划。

（2）测试结果。

（3）测试分析。

### 3.4.1 测试计划

测试计划包含3个模块：多二叉树管理菜单测试，单二叉树操作菜单测试，文件操作测试。

测试模块1：多二叉树管理菜单测试（如表3-2所示）。

表3-2 多二叉树管理菜单测试内容

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试序号 | 测试前总体情况描述 | 测试描述 | 测试操作 | 预计输出 |
| 1 | 二叉树集合有3个二叉树 | 测试添加二叉树功能 | 选择添加树，提供合法输入 | 添加二叉树成功，二叉树集合菜单尾部显示新添加的二叉树名称，提示依次输入二叉树位置、关键字、字符串 |
| 2 | 二叉树集合中有3个二叉树 | 测试添加二叉树功能 | 选择添加树，输入已存在的表名称 | 添加二叉树失败，提示二叉树已存在 |
| 3 | 二叉树集合已满 | 测试添加二叉树功能 | 选择添加树，提供合法输入 | 添加二叉树失败，提示二叉树集合已满 |
| 4 | 二叉树集合有3个二叉树 | 测试删除二叉树功能 | 选择删除树，删除全部二叉树 | 删除二叉树成功，二叉树集合菜单显示所有二叉树均未创建 |
| 5 | 二叉树集合有3个二叉树 | 测试删除二叉树功能 | 选择删除树，输入集合中不存在的二叉树名称 | 删除二叉树失败，提示集合中无此二叉树 |
| 6 | 二叉树集合有3个二叉树 | 测试查找二叉树功能 | 选择查找树，查找第3个二叉树 | 查找二叉树成功，提示二叉树在集合中位置 |
| 7 | 二叉树集合有3个二叉树 | 测试查找二叉树功能 | 选择查找树，查找不存在于集合中的树 | 查找二叉树失败，提示集合中无此二叉树 |
| 8 | 二叉树集合有3个二叉树 | 测试进入二级菜单功能 | 选择进入二级菜单，选择不存在于集合中的树名称 | 进入二级菜单失败，提示集合中无此二叉树 |

测试模块2：单二叉树操作菜单测试（如表3-3所示）。

表3-3 单二叉树操作菜单测试内容

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试序号 | 测试前总体描述 | 测试描述 | 测试操作 | 预计输出 |
| 1 | 多二叉树进入单二叉树操作 | 测试创建功能 | 选择创建，提供合法输入 | 创建二叉树失败，提示二叉树已存在 |
| 2 | 二叉树已存在 | 测试销毁功能 | 选择销毁功能 | 销毁成功，提示二叉树被销毁 |
| 3 | 二叉树已被销毁 | 测试创建功能 | 选择创建，提供合法输入 | 创建二叉树成功，提示创建成功 |
| 4 | 二叉树已存在 | 测试判空功能 | 选择判空 | 显示二叉树不为空 |
| 5 | 二叉树已存在 | 测试清空功能 | 选择清空 | 清空二叉树成功，提示二叉树被清空 |
| 6 | 二叉树已被清空 | 测试创建功能 | 选择创建，提供关键字有重复的输入 | 创建二叉树失败，提示关键字重复 |
| 7 | 二叉树不存在 | 测试销毁功能 | 选择销毁 | 销毁二叉树失败 |
| 8 | 二叉树不存在 | 测试清空功能 | 选择清空 | 清空二叉树失败 |
| 9 | 二叉树不存在 | 测试判空功能 | 选择判空 | 显示二叉树为空 |
| 10 | 二叉树不存在 | 测试求深度功能 | 选择求深度 | 显示二叉树深度为0 |
| 11 | 二叉树存在且为一般树 | 测试求深度功能 | 选择求深度 | 显示二叉树深度 |
| 12 | 二叉树存在且为含5个结点的单枝树 | 测试求深度功能 | 选择求深度 | 显示二叉树深度为5 |
| 13 | 二叉树存在且为一般树 | 测试先序遍历、中序遍历、后序遍历、层序遍历功能 | 依次选择先序遍历、中序遍历、后序遍历、层序遍历 | 显示遍历结果 |
| 14 | 二叉树存在且为满二叉树 | 测试先序遍历、中序遍历、后序遍历、层序遍历功能 | 依次选择先序遍历、中序遍历、后序遍历、层序遍历 | 显示遍历结果 |
| 15 | 二叉树存在且为单枝树 | 测试先序遍历、中序遍历、后序遍历、层序遍历功能 | 依次选择先序遍历、中序遍历、后序遍历、层序遍历 | 显示遍历结果 |
| 16 | 二叉树存在 | 测试查找功能 | 选择查找，查找根结点 | 显示结点关键字和字符串 |
| 17 | 二叉树存在 | 测试查找功能 | 选择查找，查找左叶子结点 | 显示结点关键字和字符串 |
| 18 | 二叉树存在 | 测试查找功能 | 选择查找，查找非叶子结点 | 显示结点关键字和字符串 |
| 19 | 二叉树存在 | 测试赋值功能 | 选择赋值，为某个关键字新赋值的关键字不与二叉树中任一个相同 | 赋值成功，提示赋值成功 |
| 20 | 二叉树存在 | 测试赋值功能 | 选择赋值，为某个关键字新赋值的关键字与原关键字相同 | 赋值成功，提示赋值成功 |
| 21 | 二叉树存在 | 测试赋值功能 | 选择赋值，为某个关键字新赋值的关键字与二叉树中某一个相同 | 赋值失败，提示关键字不存在或关键字重复 |
| 22 | 二叉树存在 | 测试获取兄弟结点功能 | 选择获取兄弟结点，选择无兄弟结点的结点 | 显示无兄弟结点 |
| 23 | 二叉树存在 | 测试获取兄弟结点功能 | 选择获取兄弟结点，选择有右兄弟结点的结点 | 显示兄弟结点的关键字和字符串 |
| 24 | 二叉树存在 | 测试插入功能 | 选择插入，选择根结点，提供合法输入 | 显示插入成功 |
| 25 | 二叉树存在 | 测试插入功能 | 选择插入，选择有左右子树的结点，插入为左子树，提供合法输入 | 显示插入成功 |
| 26 | 二叉树存在 | 测试插入功能 | 选择插入，选择只有左子树的结点，插入为左子树，提供合法输入 | 显示插入成功 |
| 27 | 二叉树存在 | 测试插入功能 | 选择插入，选择只有右子树的结点，插入为右子树，提供合法输入 | 显示插入成功 |
| 28 | 二叉树存在 | 测试插入功能 | 选择插入，选择无子树的结点，插入为右子树，提供合法输入 | 显示插入成功 |
| 29 | 二叉树存在 | 测试插入功能 | 选择插入，选择任一结点，待插入结点关键字与原二叉树中某关键字重复 | 显示插入失败，提示关键字重复 |
| 30 | 二叉树存在 | 测试删除功能 | 选择删除，选择根结点 | 显示删除成功 |
| 31 | 二叉树存在 | 测试删除功能 | 选择删除，选择有左右子树的结点 | 显示删除成功 |
| 32 | 二叉树存在 | 测试删除功能 | 选择删除，选择只有左子树的结点 | 显示删除成功 |
| 33 | 二叉树存在 | 测试删除功能 | 选择删除，选择只有右子树的结点 | 显示删除成功 |
| 34 | 二叉树存在 | 测试删除功能 | 选择删除，选择无子树的结点 | 显示删除成功 |

测试模块3：文件操作菜单测试（如表3-4所示）。

表3-4 文件操作菜单测试内容

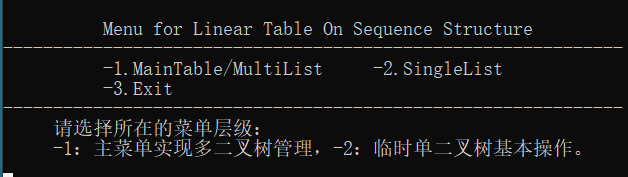
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试序号 | 测试前总体描述 | 测试描述 | 测试操作 | 预计输出 |
| 1 | 二叉树存在 | 测试存档功能 | 选择存档，提供合法输入 | 存档成功，在指定文件下可见二叉树中数据 |
| 2 | 二叉树不存在 | 测试存档功能 | 选择存档提供合法输入 | 存档失败，提示二叉树不存在 |
| 3 | 二叉树存在 | 测试读档功能 | 选择读档 | 读档失败，提示二叉树已存在，读档会失去二叉树中原有元素 |

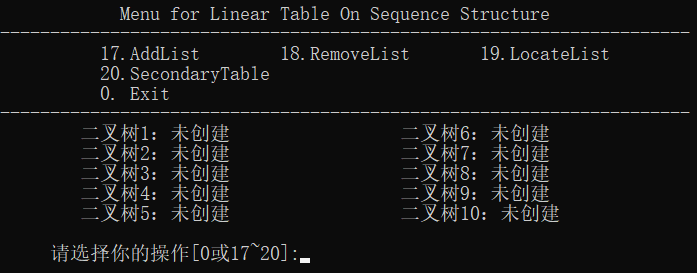
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 二叉树不存在 | 测试读档功能 | 选择读档 | 读档成功，遍历二叉树可见二叉树中元素与原文件存档顺序和内容相同 |

### 3.4.2 测试结果

根据测试计划，测试结果包含3个部分：多二叉树管理菜单测试结果，单二叉树操作菜单测试结果，文件操作测试结果。

各初始菜单选择界面如图3-8所示。





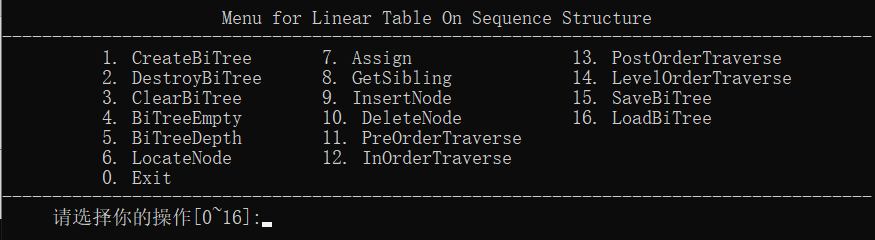
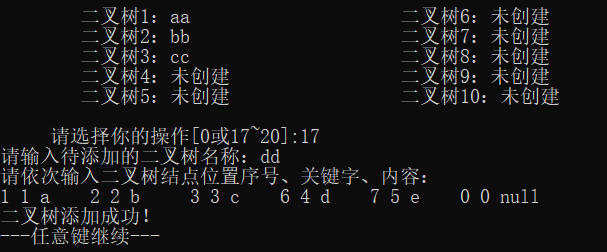
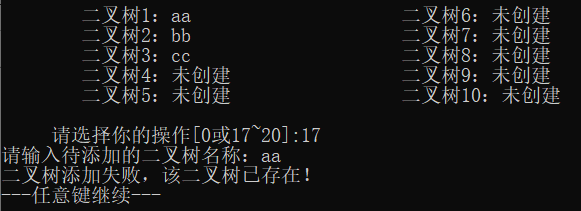


图3-8 菜单界面展示

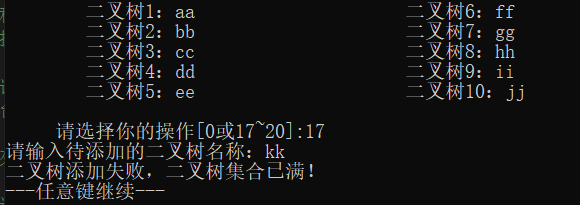
单二叉树管理菜单测试结果如图3-9所示。



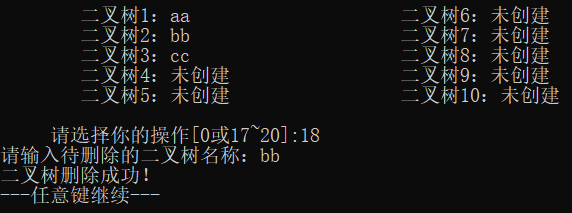
测试序号1

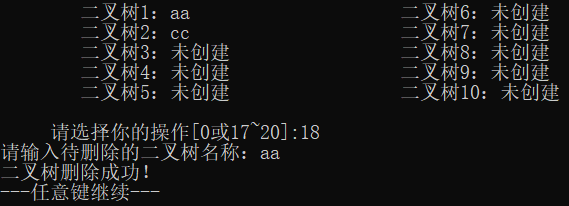


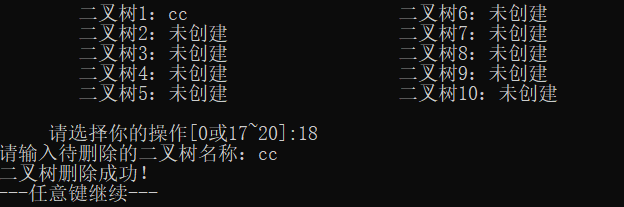
测试序号2



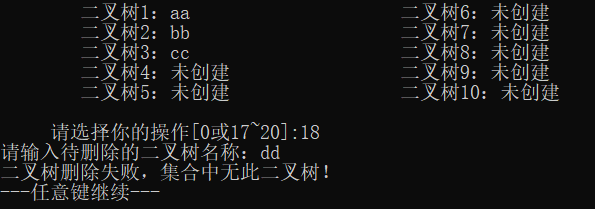
测试序号3



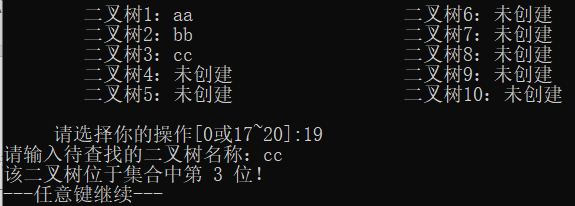




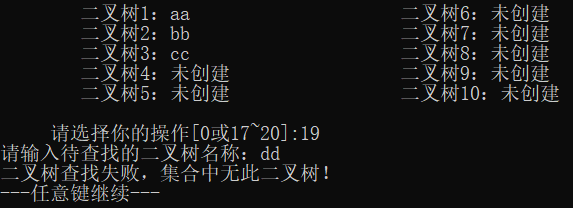
测试序号4



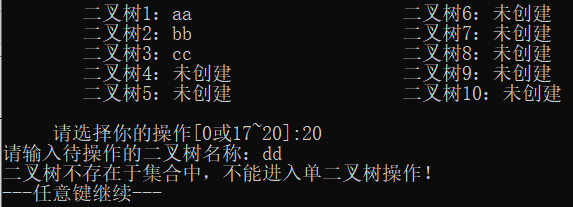
测试序号5



测试序号6



测试序号7



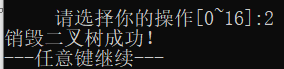
测试序号8

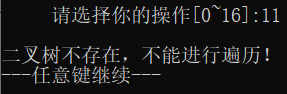
图3-8 多二叉树管理菜单测试结果

多二叉树操作菜单测试结果如图3-9所示。

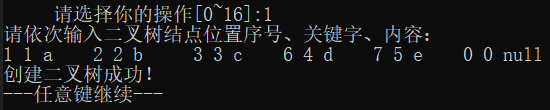


测试序号1

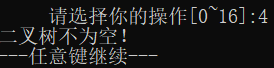




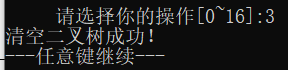
测试序号2

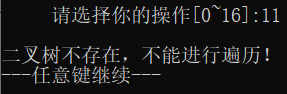


测试序号3

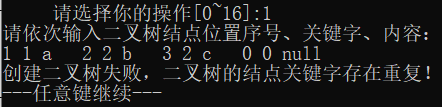


测试序号4

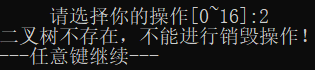




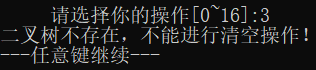
测试序号5



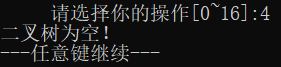
测试序号6



测试序号7



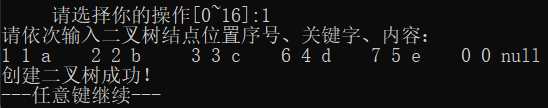
测试序号8

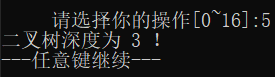


测试序号9

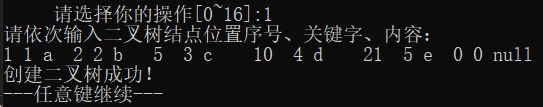


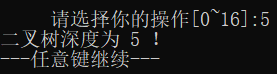
测试序号10



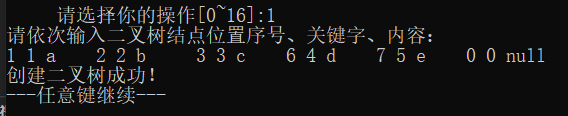


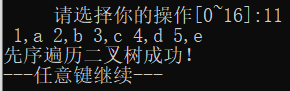
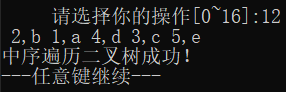
测试序号11

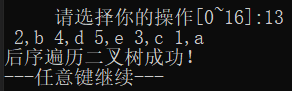




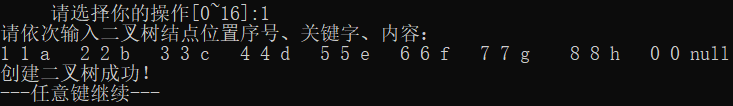
测试序号12

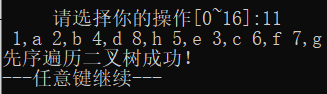
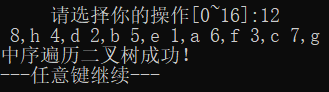


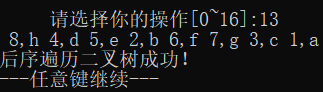
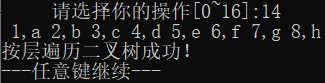
 

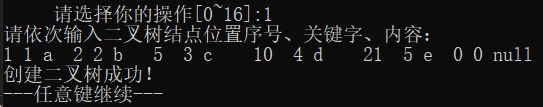
测试序号13

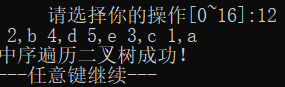
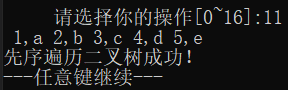


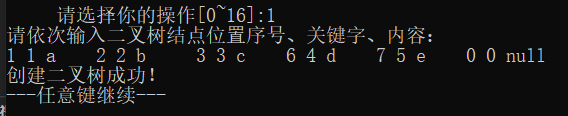
测试序号14

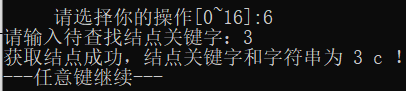
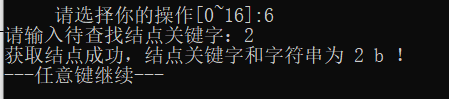
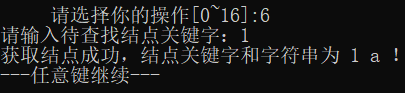




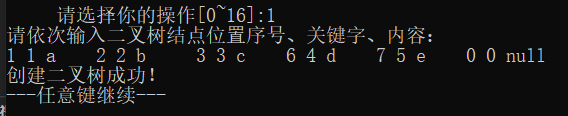


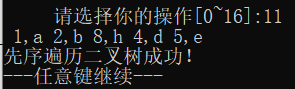
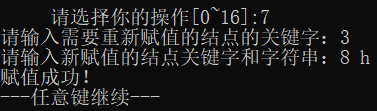
测试序号15



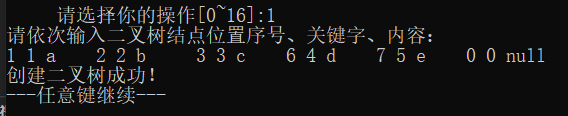


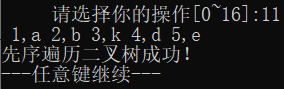
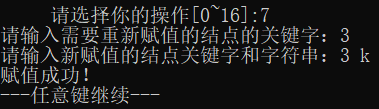
测试序号16



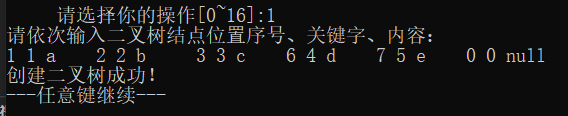


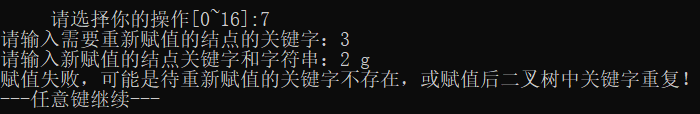
测试序号17

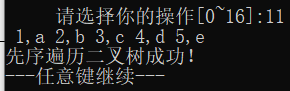




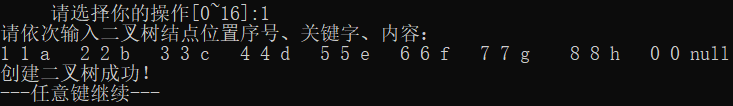
测试序号18

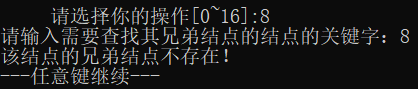




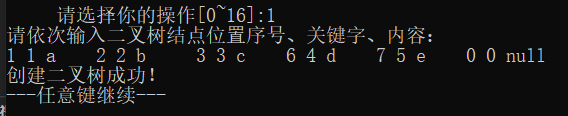


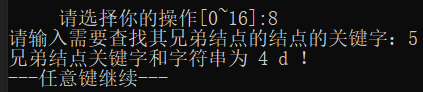
测试序号19



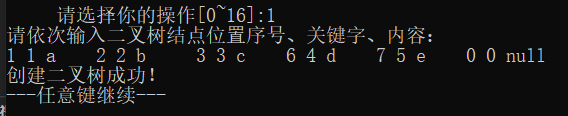


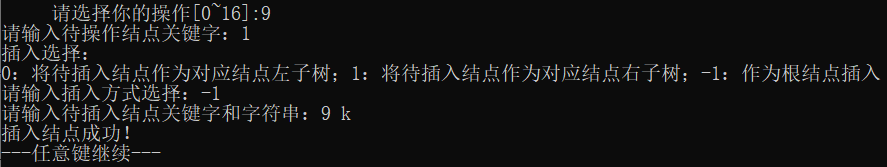
测试序号20

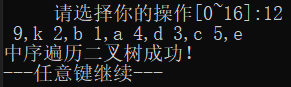
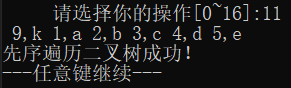




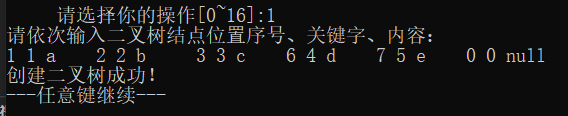
测试序号21

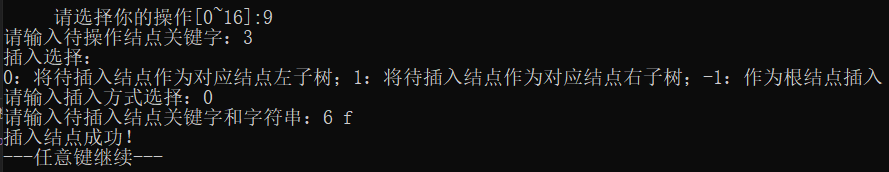


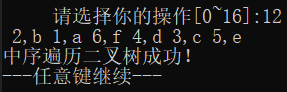
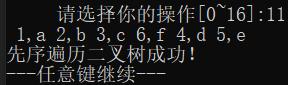




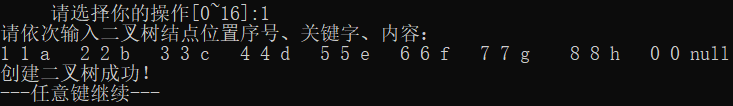
测试序号22

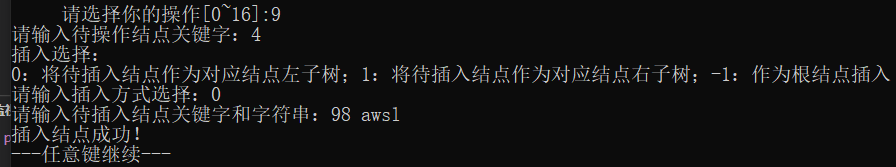


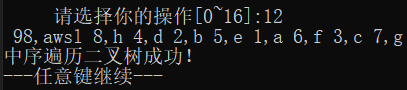
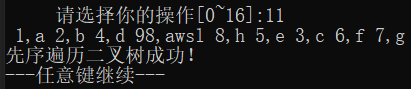




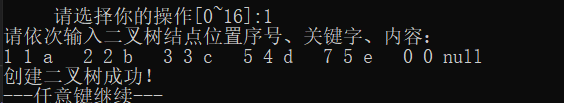
测试序号23

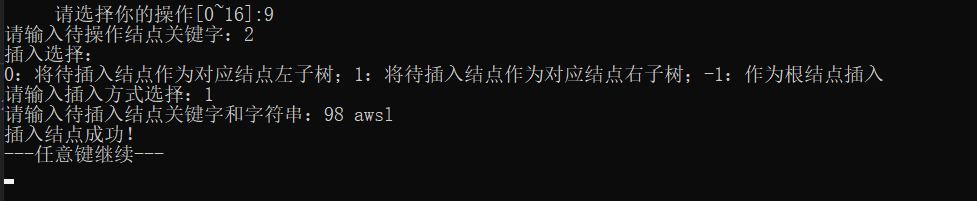


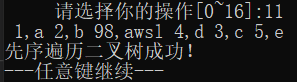
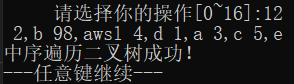




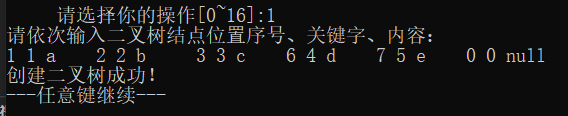
测试序号24

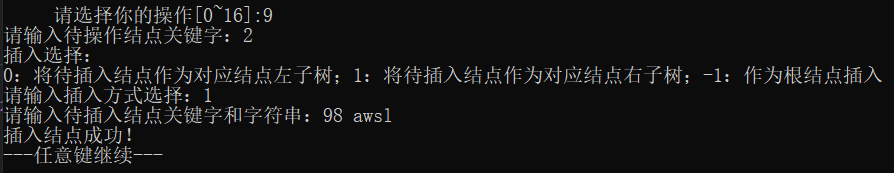


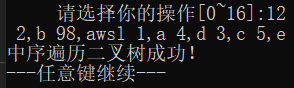
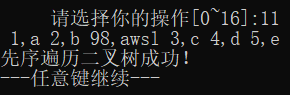


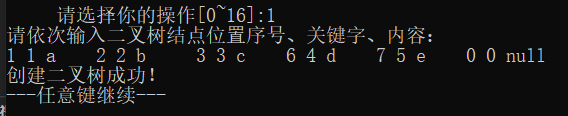
测试序号25

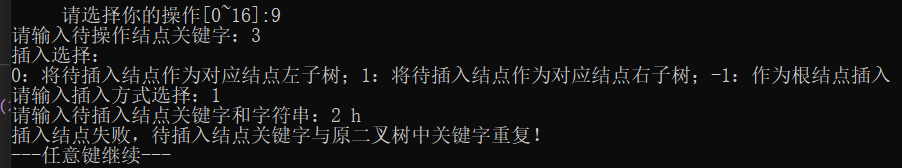




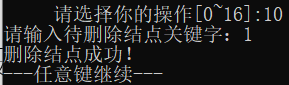
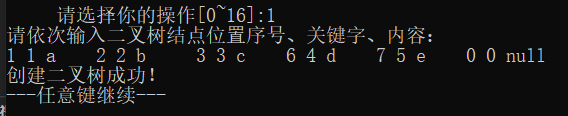


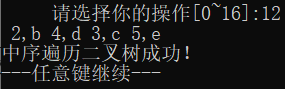
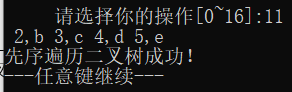
测试序号26



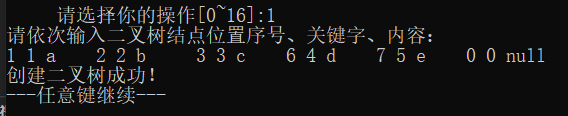


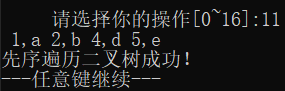
测试序号27



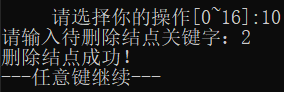
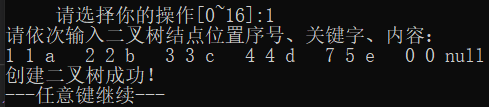


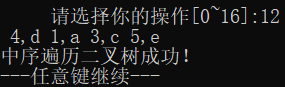
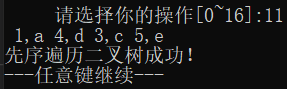
测试序号28



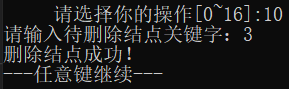
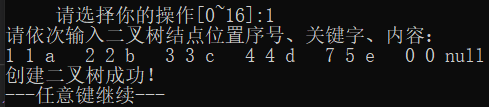


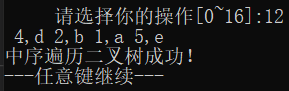
测试序号29



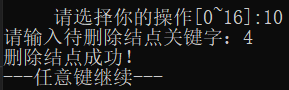
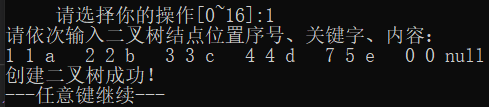


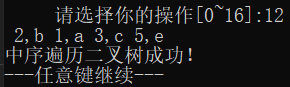
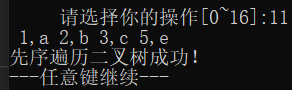
测试序号30





测试序号31

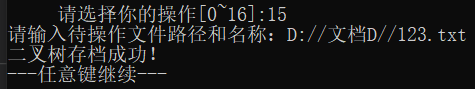
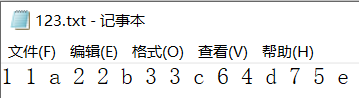




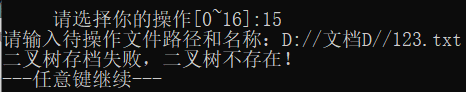
测试序号32

图3-9 单二叉树操作菜单测试结果

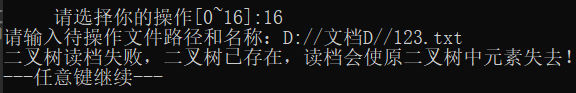
文件操作菜单测试结果如图3-10所示。

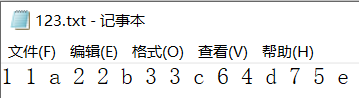
测试序号1

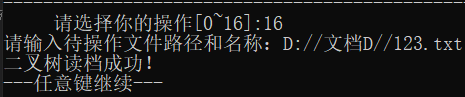


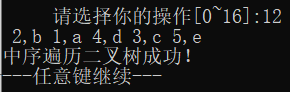
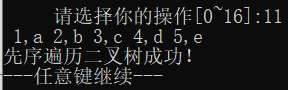
测试序号2



测试序号3







测试序号4

图3-10 文件操作菜单测试结果

### 3.4.3 测试分析

根据实验要求，以测试计划的3个模块的测试结果为根本依据，将测试结果总结如下：

多二叉树管理菜单测试：总计8项测试内容全部通过，通过率为100%。

单二叉树操作菜单测试：总计32项测试内容全部通过，通过率为100%。

文件操作菜单测试：总计4项内容全部通过，通过率为100%。

## 3.5 实验小结

本次实验是第3次数据结构实验，主要实现对二叉树的管理和功能操作。实验使用二叉链表作为物理结构，比线性表实验中使用的单链表相对更为复杂，但基本原理和性质相同。在实验过程中，比较鲜明的特点是使用了较多的递归使函数简单易写，但要深刻理解递归思想则相较递推困难。如在编写查找结点函数时，我起初遗漏了进入右子树递归时的判断条件中的!p，使得程序即使查找到待查找结点，最终返回值也会在上一层递归中改变。而最终在实际应用中，我们应该熟练掌握递推、递归两种方法，在此次实验中部分函数我还不能做到用递推实现，这是后续需要继续学习的。又如几种非递归遍历二叉树的方法都比较经典和巧妙，需要反复学习领悟其用栈思想。

总体上来说，本次实验难度较前两次有了较大的提升，让我们对二叉树的操作有了更深入的了解和认识。相信以后可以利用本次实验中学习到的知识和思想更好地处理问题。

# 4 基于邻接表的图实现

## 4.1 问题描述

本次实验的主要内容是以邻接表作为无向图的物理结构，通过实现图的基本操作，掌握图的相关知识，实现线性表抽象数据类型（Abstract Data Type，缩略名 ADT）的基本运算，构造一个具有菜单的功能演示系统，并要求实现无向图的文件形式保存和多无向图管理。本实验将以上述内容为主要目标，使用 C++语言构建问题解决方案。

### 4.1.1 问题1：无向图ADT的邻接表实现

根据实验要求，本次实验中要以邻接表作为无向图的物理结构，通过函数形式定义和实现无向图的基本功能，掌握和了解无向图的代码实现和功能操作思想。基本运算包括创建无向图、销毁无向图、查找顶点、顶点赋值、获取第一邻接点、获取下一邻接点、插入顶点、删除顶点、插入弧、删除弧、深度优先搜索遍历、广度优先搜索遍历等12种，基本可以囊括无向图的各项基本功能和管理。

### 4.1.2 问题2：实现无向图的文件形式保存和读取

根据实验要求，实验者需要实现将无向图内容保存在指定文件中、将指定文件中的数据内容读取到无向图中两种操作。其中，①需要设计文件数据记录格式，以高效保存无向图数据逻辑结构(D,{R})的完整信息；②需要设计无向图文件保存和加载操作合理模式。

### 4.1.3 问题3：实现多线性表管理

根据实验要求，实验者要实现多个无向图管理，包括无向图添加、无向图删除、无向图查找等3个功能。无向图集合中对每个无向图进行唯一命名，便于选择进入对某个无向图的操作；无向图集合应当设置上限，达到上限后不能再进行添加无向图操作；对不存在的无向图进行删除或查找操作应返回操作错误提示信息；对命名已存在的无向图实行添加操作时返回操作错误提示。

### 4.1.4 问题4：构造具有菜单的功能演示系统

根据实验要求，需要实验者自行设计和构造可以完整展现无向图ADT功能的演示系统，可以通过用户选择实现对无向图的基本功能操作。演示系统中应包括功能选项、用户需要输入内容提示、操作结果提示等，便于用户直接使用和操作系统，进行合法输入。同时，因为演示系统包括多无向图管理和单无向图功能操作，应当设计好主菜单和子菜单的层级关系，使用户操作结果可以在两层菜单中传递和保存。

## 4.2 系统设计

根据实验要求，系统设计主要包括以下几个方面。

（1）设计邻接表实现下的无向图ADT的物理结构。

（2）设计文件格式。

（3）设计多无向图管理模式。

（4）设计具有菜单的功能演示系统。

### 4.2.1 邻接表实现下无向图ADT的物理结构设计

邻接表ADT的物理结构如下：

|  |
| --- |
| 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; |

此处使用预定义的方式实现无向图ADT的物理结构，便于后续对无向图的定义和使用。

### 4.2.2 文件格式设计

实验中使用二进制文件格式进行对无向图中数据元素的存储和读取，有利于数据形式在文件中保存和读取到线性表中。在存档过程中，设计将邻接表以表格内容形式存储，每个顶点的邻接顶点由输入“-1”表示邻接顶点已表示完毕，顶点数组由输入“-1 nil”表示顶点遍历完成。在读档过程中，设计利用栈保证邻接表顶点顺序（首插法创建邻接表）与原来相同。

### 4.2.3 多无向图管理模式设计

多无向图存放管理模式定义如下：

|  |
| --- |
| typedef struct{ //无向图的管理表定义  struct { char name[30]; //无向图名称  ALGraph G;  } elem[10]; //存放无向图的集合  int length; //无向图集合的长度  int listsize; //无向图集合分配的空间  }LISTS; |

使用数组方式建立无向图集合，同时记录无向图集合长度和空间，便于对多无向图的管理和选择。

### 4.2.4 具有菜单功能的演示系统设计

在演示系统中，为实现多无向图管理和单无向图操作，设计多级菜单和多、单无向图系统。演示系统结构如图4-1所示。



图4-1 演示系统结构

演示系统包括在多无向图选择下的主菜单、二级菜单与临时单无向图选择。多无向图管理菜单包含添加、删除、查找、进入二级菜单、退出5个选项，进入二级菜单前需要输入待操作的无向图名称，如该无向图不存在则返回错误提示。二级菜单包含无向图的12个基本功能操作和无向图的文件操作。临时单无向图选项下可以创建不在无向图集合中的无命名的临时无向图，进入选项后可进行对无向图的12个基本功能操作和无向图的文件操作。

针对系统各层级的进入和退出、无向图功能的选择，通过switch语句，由用户输入数字化选择op（演示系统中有对应功能提示表），根据用户输入进入不同层级或功能选择。在主函数中针对需要获取用户输入的函数设计对应提示，以便用户知晓需要输入以实现函数正常功能的内容；针对正常情况（或合法输入）和异常情况（或非法输入）设计对应提示，以便用户知晓操作是否成功及操作结果，有利于用户下一步操作。

## 4.3 系统实现

本次实验使用的开发环境为Microsoft Visual Studio Code 1.56.2。

系统实现方面主要包括如下内容。

（1）项目工程说明。

（2）无向图ADT运算的邻接表实现。

（3）文件输入输出实现。

（4）多无向图管理实现。

（5）演示系统实现。

### 4.3.1 项目工程说明

项目工程包含3个标准库头文件：stdio.h、stdlib.h、string.h。

项目工程中预定义的常量和类型声明如下：

|  |
| --- |
| #define TRUE 1  #define FALSE 0  #define OK 1  #define ERROR 0  #define INFEASIBLE -1  #define OVERFLOW -2  #define MAX\_VERTEX\_NUM 20  #define MAX\_KEY 100  typedef int status;  typedef int KeyType;  typedef enum {DG,DN,UDG,UDN} GraphKind;  typedef int ElemType; //栈和队列的定义  typedef struct stack  {  ElemType elem[MAX\_VERTEX\_NUM];  int p;  } stack; //栈的定义  typedef struct QUEUE  {  ElemType elem[MAX\_VERTEX\_NUM];  int front, length;  } QUEUE; //循环队列 |

### 4.3.2 无向图ADT运算的邻接表实现

相关常量定义和顶点类型VertexType、邻接表结点ArcNode、头结点数组AdjList、邻接表类型ALGraph定义如下：

|  |
| --- |
| typedef int KeyType;  typedef enum {DG,DN,UDG,UDN} GraphKind;  typedef struct {  KeyType key;  char others[20];  } VertexType; //顶点类型定义  typedef struct ArcNode { //表结点类型定义  int adjvex; //顶点位置编号  struct ArcNode \*nextarc; //下一个表结点指针  } ArcNode;  typedef struct VNode{ //头结点及其数组类型定义  VertexType data; //顶点信息  ArcNode \*firstarc; //指向第一条弧  } VNode,AdjList[MAX\_VERTEX\_NUM];  typedef struct { //邻接表的类型定义  AdjList vertices; //头结点数组  int vexnum,arcnum; //顶点数、弧数  GraphKind kind; //图的类型  } ALGraph; |

使用的自定义函数如表4-1：

表4-1 自定义无向图函数表

|  |  |  |
| --- | --- | --- |
| 序号 | 函数定义 | 函数功能 |

|  |  |  |
| --- | --- | --- |
| 1 | status CreateGraph(ALGraph &G,VertexType V[],KeyType VR[][2]) | 创建无向图 |
| 2 | status DestroyGraph(ALGraph &G) | 销毁无向图 |
| 3 | int LocateVex(ALGraph G,KeyType u) | 查找顶点 |
| 4 | status PutVex(ALGraph &G,KeyType u,VertexType value) | 顶点赋值 |
| 5 | int FirstAdjVex(ALGraph G,KeyType u) | 获取第一邻接点 |
| 6 | int NextAdjVex(ALGraph G,KeyType v,KeyType w) | 获取下一邻接点 |
| 7 | status InsertVex(ALGraph &G,VertexType v) | 插入顶点 |
| 8 | status DeleteVex(ALGraph &G,KeyType v) | 删除顶点 |
| 9 | status InsertArc(ALGraph &G,KeyType v,KeyType w) | 插入弧 |
| 10 | status DeleteArc(ALGraph &G,KeyType v,KeyType w) | 删除弧 |
| 11 | status DFSTraverse(ALGraph &G,void (\*visit)(VertexType)) | 深度优先搜索遍历 |
| 12 | status BFSTraverse(ALGraph &G,void (\*visit)(VertexType)) | 广度优先搜索遍历 |
| 13 | status SaveGraph(ALGraph G, char FileName[]) | 无向图存档 |
| 14 | status LoadGraph(ALGraph &G, char FileName[]) | 无向图读档 |
| 15 | status AddGraph(LISTS &Lists,char ListName[]) | 增加无向图 |
| 16 | status RemoveGraph(LISTS &Lists,char ListName[]) | 删除无向图 |
| 17 | int LocateGraph(LISTS Lists,char ListName[]) | 查找无向图 |
| 18 | void SecondaryTable(ALGraph G,int op) | 进入二级菜单 |
| 19 | void visit(VertexType v) | 访问顶点 |
| 20 | void DFS(AdjList verx,int index,int visited[100],void (\*visit)(VertexType)) | 深搜递归 |
| 21 | void iniStack(stack &S) | 初始化栈 |
| 22 | int isEmptyStack(stack &S) | 栈判空 |
| 23 | int push(stack &S, ElemType e) | 元素入栈 |
| 24 | ElemType pop(stack &S) | 元素出栈 |
| 25 | void iniQueue(QUEUE &Q) | 初始化队列 |
| 26 | int enQueue(QUEUE &Q, ElemType e) | 元素入队 |
| 27 | int deQueue(QUEUE &Q, ElemType &e) | 元素出队 |
| 28 | int isEmptyQueue(QUEUE &Q) | 队列判空 |

**1.创建无向图**

函数功能概述：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。

具体实现：

（1）初始化无向图，将其顶点数、弧数置为0，确定图类型。

（2）根据用户输入顶点类型V数组，若V[0].key=-1，表明图中无顶点，返回ERROR。否则，根据V数组循环标记顶点数组，每次循环顶点数G.vexnum+1。用flag数组标记关键字是否出现过，出现关键字重复则返回ERROR。若顶点数G.vexnum>20（超出图顶点集上限），返回ERROR。

（3）判断输入顶点合法后，根据V数组循环构造顶点数组，为顶点赋值关键字和字符串。

（4）根据用户输入，用i遍历VR数组构造邻接表。通过flag判断，若输入的弧顶点任一个不在顶点集中，返回ERROR。通过VR[i][0]、VR[i][1]判断，若输入的弧顶点相同，不进行本次创建操作。用j遍历顶点集数组，每次用f数组标记已创建的边（即该顶点邻接表中已存在的顶点位序），若弧已存在则不进行创建操作。由于弧顶点顺序对邻接表创建无影响，且对弧的两顶点都需创建邻接表表示，因此判断顶点集中顶点关键字与VR[j][0]、VR[j][1]是否相同，相同且弧不存在，则用k遍历顶点集查找到另一顶点位序，用首插法创建该顶点下的邻接表，在上述任一情况下执行弧G.arcnum增加操作。循环结束返回OK。

首插法：定义邻接表顶点指针tmp，指向第一邻接点G.vertices[j].firstarc。在第一邻接点位置使用malloc函数开辟新空间，将新弧顶点位序赋给第一邻接点位置的adjvex，第一邻接点的下一顶点指针指向tmp。

复杂度分析：时间复杂度：O（）；空间复杂度：O（n）。



图4-2 创建无向图流程图

**2.销毁无向图**

函数功能概述：销毁无向图G,删除G的全部顶点和边。

具体实现：

（1）用i遍历顶点集数组，定义邻接表顶点指针tmp1、tmp2，tmp1指向每个顶点的第一邻接点。若tmp1为空，则该顶点无边相连，不需要进行删除弧操作；否则，tmp2指向第一邻接点的下一邻接点G.vertices[i].firstarc->nextarc。循环删除头顶点后的弧，每次将第一邻接点置为tmp2，释放tmp1空间，再次将tmp1、tmp2置为第一邻接点与后一邻接点，至tmp1为空时退出循环。

（2）将顶点数G.vexnum与弧数G.arcnum置为0，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**3.查找顶点**

函数功能概述：根据u在图G中查找顶点，查找成功返回位序，否则返回INFEASIBLE。

具体实现：

用i遍历顶点数组，查找到顶点关键字与用户输入待查找关键字u相同时，用flag标记并退出循环。被标记则返回位序i，否则返回INFEASIBLE。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**4.顶点赋值**

函数功能概述：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

具体实现：

（1）遍历顶点集数组，用v数组标记图中顶点。

（2）遍历顶点集数组，查找到顶点关键字与用户输入待查找关键字u相同时，当新关键字不与现有关键字重复，或新关键字与待修改顶点关键字相同时，进行关键字和字符串赋值操作，返回OK。

（3）若待修改顶点不在顶点集数组中，循环结束后返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**5.获取第一邻接点**

函数功能概述：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。若顶点无邻接顶点，返回OVERFLOW；若顶点不在图中，返回INFEASIBLE。

具体实现：

（1）遍历顶点数组，查找到与u关键字相同的顶点。顶点有邻接顶点时，返回第一邻接点位序。

（2）若查找到与u关键字相同的顶点时该顶点无邻接点，返回OVERFLOW。若顶点集中无该顶点，返回INFEASIBLE。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**6.获取下一邻接点**

函数功能概述：根据v在图G中查找顶点，查找成功返回顶点v的邻接顶点相对于w的下一邻接顶点的位序，查找邻接顶点失败返回ERROR，查找顶点失败返回INFEASIBLE

具体实现：

（1）遍历顶点数组，查找到与v关键字相同的顶点。遍历邻接表顶点，查找到与w关键字相同的邻接顶点，当w有下一邻接点时，返回下一邻接点位序。

（2）若查找w失败或w无下一邻接点，返回ERROR。若查找v失败，返回INFEASIBLE。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**7.插入顶点**

函数功能概述：在图G中插入顶点v，成功返回OK,否则返回ERROR。

具体实现：

（1）若顶点数G.vexnum=MAX\_VERTEX\_NUM，顶点集已满，不能增加顶点，返回OVERFLOW。

（2）遍历顶点集，用flag数组标记各顶点。

（3）若待插入顶点关键字未被标记，在顶点集数组最后增加顶点，顶点数G.vexnum+1，返回OK。若待插入顶点关键字被标记，关键字重复，返回ERROR。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**8.删除顶点**

函数功能概述：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。

具体实现：

（1）遍历顶点集，用flag数组标记各顶点。

（2）图中无顶点（空图）或无对应关键字v的顶点时，返回ERROR。

（3）否则，用i遍历顶点集，查找到关键字v对应的顶点时，将其邻接表中的弧全部删掉（操作与销毁无向图同），用a记录该顶点位序i；不是关键字v对应的顶点时，循环邻接表查找与关键字顶点相关的弧并删除（若该顶点无邻接顶点则不需执行删除操作），使用tmp1、tmp2两个顶点指针。由于无向图无图头顶点，因此待删除弧顶点位于第一邻接点与位于其他邻接点的情况要分别讨论。顶点对应于第一邻接点时，tmp2指向第一邻接点G.vertices[i].firstarc，再执行G.vertices[i].firstarc=G.vertices[i].firstarc->nextarc，将第一邻接点转为邻接表中下一邻接点，释放tmp2指向空间；关键字对应于除第一邻接点外的其他顶点时，tmp1指向第一邻接点，tmp2指向tmp1下一邻接点，执行tmp1->nextarc=tmp2->nextarc，释放tmp2指向空间。

（4）再次遍历顶点集，查找到关键字不是v对应的顶点时，遍历其邻接表，位序大于a的邻接顶点的位序均减一；查找到关键字v对应的顶点时，将其后顶点在顶点集数组中前移一位，同时对移至被删除邻接点位置的邻接点执行上述位序减一操作，顶点数G.vexnum-1。返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。



图4-3 删除顶点流程图

**9.插入弧**

函数功能概述：在图G中增加弧<v,w>，成功返回OK,否则返回ERROR。

具体实现：

（1）遍历顶点集，用flag数组标记图中各顶点。根据用户输入弧的两顶点v、w，当查找到关键字为v、w的顶点时，分别用a、b记录其位序。

（2）当a=b时，插入弧为自环，不执行操作，返回OK。当v、w对应顶点中任一个不在图中时，返回INFEASIBLE。

（3）遍历顶点集，分别查找v、w对应的顶点，用顶点指针tmp指向第一邻接点，遍历邻接表，若待插入弧已存在，返回ERROR；若不存在，在其邻接表中首插法插入弧，待插入弧的位序分别为b、a。

（4）图中弧数G.arcnum+1。返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**10.删除弧**

函数功能概述：在图G中删除弧<v,w>，成功返回OK,否则返回ERROR。

具体实现：

（1）遍历顶点集，用flag数组标记图中各顶点。根据用户输入弧的两顶点v、w，当查找到关键字为v、w的顶点时，分别用a、b记录其位序。

（2）当v、w对应顶点中任一个不在图中时，返回INFEASIBLE。

（3）遍历顶点集，分别查找v、w对应的顶点，用顶点指针tmp指向第一邻接点，遍历邻接表，若待删除弧存在，用pt标记并退出循环。若pt被标记，用顶点指针tmp1、tmp2进行删除操作。由于无向图无图头顶点，因此待删除弧顶点位于第一邻接点与位于其他邻接点的情况要分别讨论（法同删除顶点）。若pt未被标记，则待删除弧不存在，返回ERROR。

（4）v与w的情况均讨论完成后，图中弧数G.arcnum-1。返回OK。

**11.深度优先搜索遍历**

函数功能概述：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

具体实现：

（1）若图中顶点数G.vexnum=0，图为空图，返回ERROR。

（2）用visited数组标记搜索过的顶点位序i。用i遍历顶点集，若顶点位序i为被标记，使用深搜递归函数搜索。循环结束后返回OK。

（3）设计深搜递归函数便于递归查找。

无返回值的深搜递归函数：

传递参数：头顶点数组verx，搜索序号index，标记数组visited[100]，访问顶点函数visit。

a.若index被标记，返回。

b.标记index。访问顶点集序号为index的顶点。

c.定义顶点指针p指向顶点的第一邻接点。p不为空时，若p对应顶点位序未被访问，对该位序顶点递归深搜。p指向下一邻接点。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**12.广度优先搜索遍历**

函数功能概述：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

具体实现：

（1）若图中顶点数G.vexnum=0，图为空图，返回ERROR。

（2）初始化队列。用i遍历顶点集，使用visited数组标记顶点是否被访问过，若顶点未被访问，访问并标记该顶点，将顶点位序i入队。队列非空时进入循环，将队首元素出队，赋值给i。定义邻接顶点指针q，循环遍历邻接表，若邻接表顶点位序未被访问，将其进队，访问并标记该邻接顶点。

（3）遍历顶点集结束后，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**13.访问结点**

函数功能概述：对当前结点执行特定操作。

具体实现：打印当前结点的关键字和字符串。

复杂度分析：时间复杂度：O（1）；空间复杂度：O（1）。

**14.栈函数**

初始化栈：将栈顶位置S.p置为0。

栈判空：判断栈顶位置S.p是否为0，是则为空栈，返回1；否则非空栈，返回0。

元素进栈：判断栈顶位置S.p是否为MAX\_VERTEX\_NUM-1，是则说明栈满，返回OVERFLOW；否则将元素e进栈，栈顶位置上移，返回OK。

元素出栈：判断栈顶位置S.p是否为0，是则为空栈，返回ERROR；否则非空栈，返回栈顶元素，栈顶位置下移。

**15.队列函数**

初始化队列：将队头位置和队尾位置均置为0。

队列判空：判断队尾位置是否为0，是则队列为空，返回1；否则队列非空，返回0。

元素进队：判断队尾位置是否为MAX\_VERTEX\_NUM，是则队满，返回0；否则执行Q.elem[(Q.front+Q.length++)%MAX\_VERTEX\_NUM]=e，元素在队尾入队，返回1。

元素出队：判断队尾位置是否为0，是则队空，返回0；否则执行e=Q.elem[(Q.front++)%MAX\_VERTEX\_NUM]，元素在队头出队，赋值给e，返回1。

### 4.3.3 文件输入输出实现

在无向图的文件存取过程中，将邻接表中元素直接存放到文件中。由于邻接表的创建使用首插法，读取文件时利用栈的数据结构，倒序读取与顶点相连的弧，保证创建邻接表创建与原无向图相同。

**1.无向图存档**

函数功能概述：将图的数据写入到文件FileName中。

具体实现：

（1）若以二进制写入方式打开文件失败，返回ERROR。

（2）遍历顶点集，向文件中写入顶点数组中顶点的关键字和字符串，遍历邻接表，向文件中写入当前顶点的邻接点位序。遍历当前顶点邻接表结束后，写入“-1”标记。遍历顶点集结束后，写入“-1 nil”标记。

（3）关闭文件，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（1）。

**2.无向图读档**

函数功能概述：读入文件FileName的图数据，创建图的邻接表。

具体实现：

（1）若无向图中顶点数不为0，图不为空，读档会使无向图中元素失去，返回INFEASIBLE。若以二进制读取方式打开文件失败，返回ERROR。

（2）初始化无向图，初始化栈。设置临时变量key和others暂时存储文件中读取的关键字和字符串。遍历顶点集，当暂存关键字key不为-1且顶点数G.vexnum<MAX\_VERTEX\_NUM时，赋值给顶点。循环读取文件中邻接表位序，暂存在临时变量ConnectVerLocate中，当ConnectVerLocate不为-1时，将位序入栈。邻接表循环结束后，当栈非空时将栈顶元素循环出栈创建位序邻接表，每次弧数G.arcnum+1。最后顶点数G.vexnum+1，并读取下一顶点。

（3）因过程中每条边都增加了两次，最后弧数G.arcnum/2。关闭文件，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

### 4.3.4 多无向图管理实现

多无向图管理主要包括对无向图集合的无向图进行增加、删除、查找操作，同时包括实验员为从一级菜单进入二级菜单使用的进入单无向图操作函数。对多无向图的管理使用数组实现。

**1.增加无向图**

函数功能概述：在Lists中增加一个名称为ListName的无向图。

具体实现：

（1）判断无向图集合长度Lists.length是否小于无向图集合空间Lists.listsize，否则返回OVERFLOW。

（2）若Lists.length<Lists.listsize，根据用户输入的无向图名称ListName，遍历无向图集合用strcmp函数判断是否有重名无向图，有则用flag标记并退出循环，返回ERROR。

（3）若无重名无向图，将无向图名称ListName赋给集合末位无向图，循环将顶点关键字和字符串写入顶点集V数组，将邻接表位序写入邻接表VR数组。调用CreateBiTree函数创建无向图，j接收CreateBiTree返回值。若j=ERROR，返回ERROR，否则无向图集合长度Lists.length+1，返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**2.删除无向图**

函数功能概述：在Lists中删除一个名称为ListName的无向图。

具体实现：

（1）根据用户输入ListName，使用i遍历集合，使用strcmp函数判断是否有重名无向图，有则用flag标记并退出循环。

（2）若未被标记，说明集合中无此无向图，返回ERROR。

（3）若被标记，调用销毁函数销毁当前无向图，从i起至Lists.length-1，将集合中所有无向图元素前移一位。若Lists.elem[Lists.length-1]的无向图未被销毁，调用销毁函数销毁之。将Lists.elem[Lists.length-1]的名称置为空，调用DestroyGraph函数销毁集合中最后一个无向图，集合长度Lists.length--。返回OK。

复杂度分析：时间复杂度：O（n）；空间复杂度（n）。

**3.查找无向图**

函数功能概述：在Lists中查找一个名称为ListName的无向图，查找成功返回逻辑序号，否则返回ERROR。

具体实现：

（1）根据用户输入的无向图名称ListName，用数组下标i遍历二叉树集合用strcmp函数判断是否有重名无向图，有则用flag标记并退出循环。

（2）若未被标记或无向图集合长度为0，返回ERROR；否则返回i+1（线性表在集合中的逻辑序号）。

复杂度分析：时间复杂度：O（n）；空间复杂度：O（n）。

**4.进入二级菜单**

函数功能概述：将所选的多无向图集合中的待操作无向图传递进入二级菜单中，实现对单无向图的基本操作。

具体实现：

使用while函数实现循环输入，switch函数进行功能选择以实现无向图基本操作。

### 3.3.5 演示系统实现

主菜单演示界面设计如下：

|  |
| --- |
| system("cls"); printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("--------------------------------------------------------------\n");  printf(" -1.MainTable/MultiGraph -2.SingleGraph\n");  printf(" -3.Exit\n");  printf("--------------------------------------------------------------\n");  printf(" 请选择所在的菜单层级：\n");  printf(" -1：主菜单实现多无向图管理，-2：临时单无向图基本操作。\n"); |

多无向图管理界面设计如下：

|  |
| --- |
| system("cls"); printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("---------------------------------------------------------------------\n");  printf(" 15.AddGraph 16.RemoveGraph 17.LocateGraph\n");  printf(" 18.SecondaryTable\n");  printf(" 0. Exit\n");  printf("---------------------------------------------------------------------\n");  for(i=0;i<5;i++)  {  if(Lists.elem[i].G.vexnum==0&&Lists.elem[i+5].G.vexnum==0)  printf("\t无向图%d：未创建 \t无向图%d：未创建",i+1,i+6);  else if(Lists.elem[i].G.vexnum==0&&Lists.elem[i+5].G.vexnum!=0)  printf("\t无向图%d：未创建 \t无向图%d：%s",i+1,i+6,Lists.elem[i+5].name);  else if(Lists.elem[i].G.vexnum!=0&&Lists.elem[i+5].G.vexnum==0)  printf("\t无向图%d：%s \t无向图%d：未创建",i+1,Lists.elem[i].name,i+6);  else  printf("\t无向图%d：%s \t无向图%d：%s",i+1,Lists.elem[i].name,i+6,Lists.elem[i+5].name);  printf("\n");  }  printf(" 请选择你的操作[0或15~18]:"); |

单无向图操作界面设计如下：

|  |
| --- |
| system("cls");  printf("\n\n");  printf(" Menu for Linear Table On Sequence Structure \n");  printf("---------------------------------------------------------------------------------------\n");  printf(" 1. CreateGraph 7. InsertVex 13. SaveGraph\n");  printf(" 2. DestroyGraph 8. DeleteVex 14. LoadGraph\n");  printf(" 3. LocateVex 9. InsertArc\n");  printf(" 4. PutVex 10. DeleteArc\n");  printf(" 5. FirstAdjVex 11. DFSTraverse\n");  printf(" 6. NextAdjVex 12. BFSTraverse\n");  printf(" 0. Exit\n");  printf("---------------------------------------------------------------------------------------\n");  printf(" 请选择你的操作[0~14]:"); |

## 4.4 系统测试

根据实验要求，系统测试主要包括以下内容。

（1）测试计划。

（2）测试结果。

（3）测试分析。

### 4.4.1 测试计划

测试计划包含3个模块：多无向图管理菜单测试，单无向图操作菜单测试，文件操作测试。

测试模块1：多无向图管理菜单测试（如表4-2所示）。

表4-2 多无向图管理菜单测试内容

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试序号 | 测试前总体情况描述 | 测试描述 | 测试操作 | 预计输出 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 无向图集合有3个无向图 | 测试添加无向图功能 | 选择添加图，提供合法输入 | 添加无向图成功，无向图集合菜单尾部显示新添加的无向图名称，提示依次输入无向图顶点关键字和字符串、弧关系 |
| 2 | 无向图集合中有3个无向图 | 测试添加无向图功能 | 选择添加图，输入已存在的表名称 | 添加无向图失败，提示无向图已存在 |
| 3 | 无向图集合已满 | 测试添加无向图功能 | 选择添加图，提供合法输入 | 添加无向图失败，提示无向图集合已满 |
| 4 | 无向图集合有3个无向图 | 测试删除无向图功能 | 选择删除图，删除全部无向图 | 删除无向图成功，无向图集合菜单显示所有无向图均未创建 |
| 5 | 无向图集合有3个无向图 | 测试删除无向图功能 | 选择删除图，输入集合中不存在的无向图名称 | 删除无向图失败，提示集合中无此无向图 |
| 6 | 无向图集合有3个无向图 | 测试查找无向图功能 | 选择查找树，查找第3个无向图 | 查找无向图成功，提示无向图在集合中位置 |
| 7 | 无向图集合有3个无向图 | 测试查找无向图功能 | 选择查找图，查找不存在于集合中的图 | 查找无向图失败，提示集合中无此无向图 |
| 8 | 无向图集合有3个无向图 | 测试进入二级菜单功能 | 选择进入二级菜单，选择不存在于集合中的图名称 | 进入二级菜单失败，提示集合中无此无向图 |

测试模块2：单无向图管理菜单测试（如表4-3所示）。（不做特殊说明时创建输入为：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 | 测试前总体情况描述 | 测试描述 | 测试操作 | 预期输出 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 多无向图进入单无向图操作 | 测试创建功能 | 提供合法输入 | 创建无向图失败，提示无向图已存在 |
| 2 | 无向图已存在 | 测试销毁功能 | 选择销毁 | 销毁成功 |
| 3 | 无向图已被销毁 | 测试创建功能 | 选择创建，提供关键字重复的非法输入 | 创建无向图失败，提示数据输入错误 |
| 4 | 无向图已被销毁 | 测试创建功能 | 选择创建，提供弧顶点不在顶点集中的非法输入 | 创建无向图失败，提示数据输入错误 |
| 5 | 无向图已被销毁 | 测试创建功能 | 选择创建，提供合法输入 | 创建无向图成功 |
| 6 | 无向图已被销毁 | 测试销毁功能 | 选择销毁 | 销毁失败 |
| 7 | 无向图存在 | 测试查找功能 | 选择查找，提供合法输入 | 查找成功，提示待查找顶点在邻接表中位序 |
| 8 | 无向图存在 | 测试查找功能 | 选择查找，提供不在顶点集中的关键字输入 | 查找失败，提示顶点不存在 |
| 9 | 无向图存在 | 测试顶点赋值功能 | 选择顶点赋值，提供不在顶点集中的关键字输入 | 赋值失败，提示可能待赋值关键字不存在或赋值后关键字重复 |
| 10 | 无向图存在 | 测试顶点赋值功能 | 选择顶点赋值，提供与顶点集中非此顶点的关键字相同的输入 | 赋值失败，提示可能待赋值关键字不存在或赋值后关键字重复 |
| 11 | 无向图存在 | 测试顶点赋值功能 | 选择顶点赋值，提供合法输入 | 赋值成功 |
| 12 | 无向图存在 | 测试获取第一邻接点功能 | 选择获取第一邻接点，提供不在顶点集中的顶点关键字输入 | 获取失败，提示顶点集无此顶点 |
| 13 | 无向图存在 | 测试获取第一邻接点功能 | 选择获取第一邻接点，提供无邻接点的顶点关键字输入 | 获取失败，提示无邻接点 |
| 14 | 无向图存在 | 测试获取第一邻接点功能 | 选择获取第一邻接点，提供合法输入 | 获取成功，返回第一邻接点位序 |
| 15 | 无向图存在 | 测试获取下一邻接点功能 | 选择获取下一邻接点，提供不在顶点集中的顶点关键字输入 | 获取失败，提示顶点集无此顶点 |
| 16 | 无向图存在 | 测试获取下一邻接点功能 | 选择获取下一邻接点，提供不在邻接表中的w关键字输入 | 获取失败，提示w无下一邻接点 |
| 17 | 无向图存在 | 测试获取下一邻接点功能 | 选择获取下一邻接点，提供无下一邻接点的w关键字输入 | 获取失败，提示w无下一邻接点 |
| 18 | 无向图存在 | 测试获取下一邻接点功能 | 选择获取下一邻接点，提供合法输入 | 获取成功，提示下一邻接点位序 |
| 19 | 无向图存在 | 测试插入顶点功能 | 选择插入顶点，提供与已有关键字重复的输入 | 插入失败，提示关键字重复 |
| 20 | 无向图存在 | 测试插入顶点功能 | 选择插入顶点，提供与当前顶点相同的关键字输入 | 插入成功 |
| 21 | 无向图存在且顶点数已至最大上限 | 测试插入顶点功能 | 选择插入顶点，提供合法输入 | 插入失败，提示顶点集已满 |
| 22 | 无向图不存在 | 测试插入顶点功能 | 选择插入顶点，提供合法输入 | 插入成功 |
| 23 | 无向图存在 | 测试删除顶点功能 | 选择删除顶点，提供不在顶点集中的顶点关键字输入 | 删除失败，提示无向图为空或待删除顶点不在顶点集中 |
| 24 | 无向图存在 | 测试删除顶点功能 | 选择删除顶点，提供合法输入 | 删除成功 |
| 25 | 无向图不存在 | 测试插入弧功能 | 选择插入弧，提供弧顶点输入 | 插入失败，提示弧顶点不在图中 |
| 26 | 无向图存在 | 测试插入弧功能 | 选择插入弧，提供两个顶点相同的输入 | 插入成功 |
| 27 | 无向图存在 | 测试插入弧功能 | 选择插入弧，提供已存在的弧输入 | 插入失败，提示弧已存在 |
| 28 | 无向图存在 | 测试插入弧功能 | 选择插入弧，提供合法输入 | 插入成功 |
| 29 | 无向图存在 | 测试删除弧功能 | 选择删除弧，提供任一顶点不在顶点集中的弧输入 | 删除失败，提示弧顶点不在图中 |
| 30 | 无向图存在 | 测试删除弧功能 | 选择删除弧，提供不存在的弧输入（顶点存在） | 删除失败，提示弧不存在 |
| 31 | 无向图存在 | 测试删除弧功能 | 选择删除弧，提供合法输入 | 删除成功 |
| 32 | 无向图不存在 | 测试深度搜索、广度搜索功能 | 选择深度搜索、广度搜索 | 搜索成功，返回搜索顺序 |
| 33 | 无向图存在且有一个连通子图 | 测试深度搜索、广度搜索功能 | 选择深度搜索、广度搜索 | 搜索成功，返回搜索顺序 |
| 34 | 无向图存在且有两个连通子图 | 测试深度搜索、广度搜索功能 | 选择深度搜索、广度搜索 | 搜索成功，返回搜索顺序 |

测试模块3：文件操作菜单测试（如表4-4所示）。

表4-4 文件操作菜单测试内容

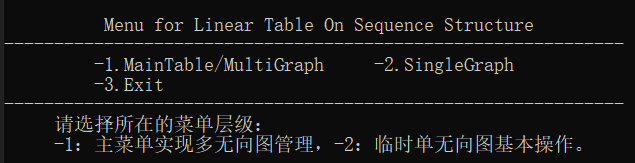
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试序号 | 测试前总体描述 | 测试描述 | 测试操作 | 预计输出 |

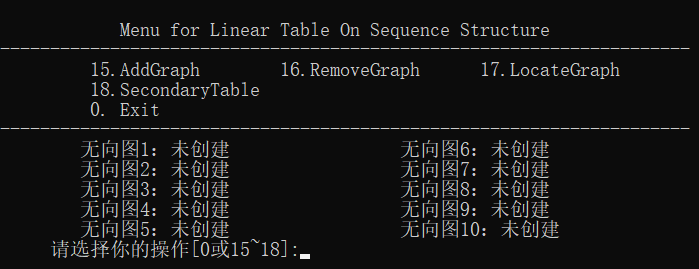
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 无向图存在 | 测试存档功能 | 选择存档，提供合法输入 | 存档成功，在指定文件下可见无向图中数据 |
| 2 | 无向图不存在 | 测试存档功能 | 选择存档提供合法输入 | 存档失败，提示无向图不存在 |
| 3 | 无向图存在 | 测试读档功能 | 选择读档 | 读档失败，提示无向图已存在，读档会失去无向图中原有元素 |
| 4 | 无向图不存在 | 测试读档功能 | 选择读档 | 读档成功，遍历无向图可见无向图中元素与原文件存档顺序和内容相同 |

### 4.4.2 测试结果

根据测试计划，测试结果包含3个部分：多无向图管理菜单测试结果，单无向图操作菜单测试结果，文件操作测试结果。

各初始菜单选择界面如图4-4所示。





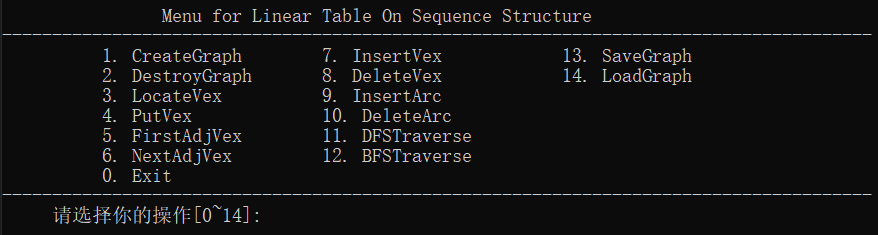
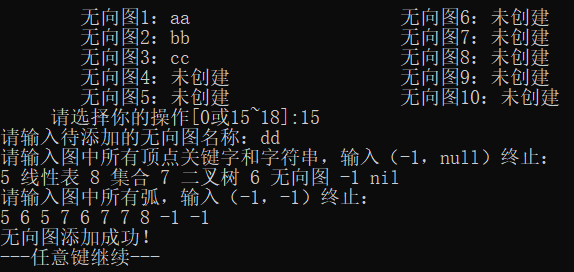
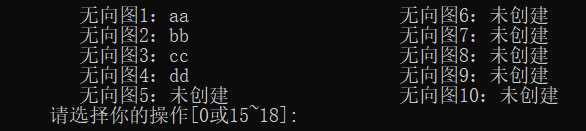


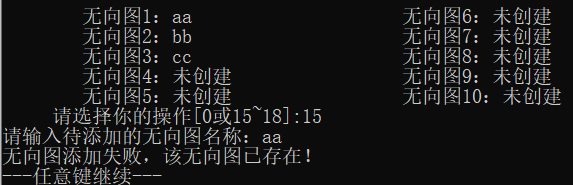
图4-4 菜单界面展示

多无向图管理菜单测试结果如图4-5所示。

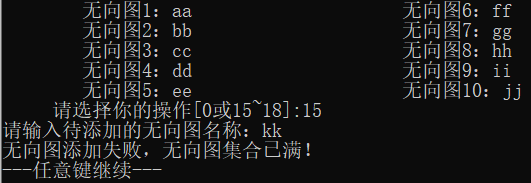




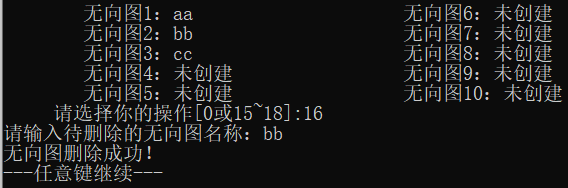
测试序号1

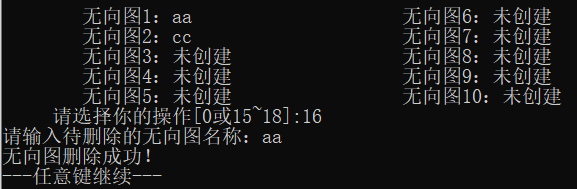


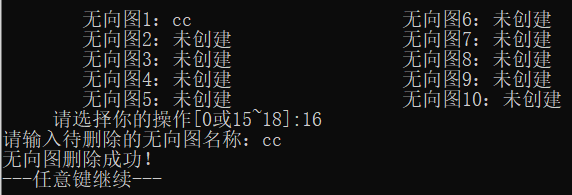
测试序号2



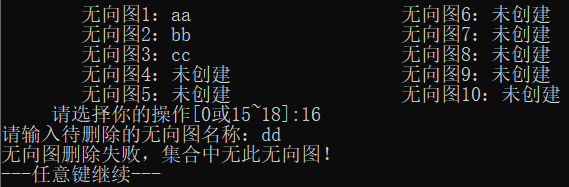
测试序号3



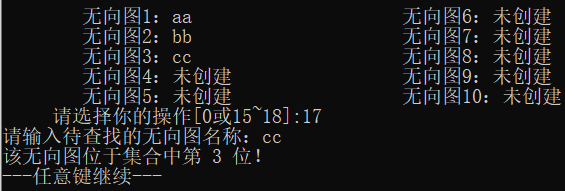




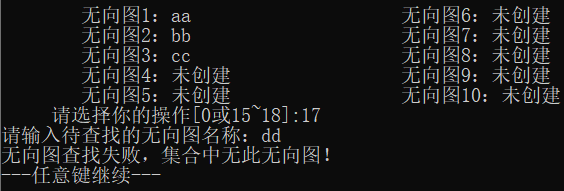
测试序号4



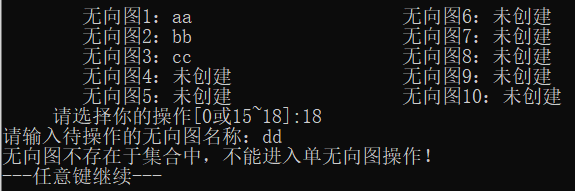
测试序号5



测试序号6



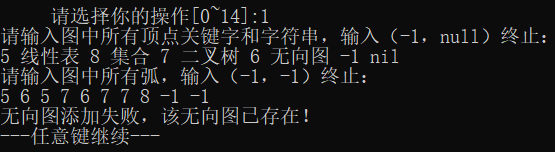
测试序号7



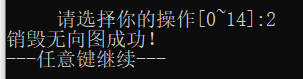
测试序号8

图4-5 多无向图管理菜单测试结果

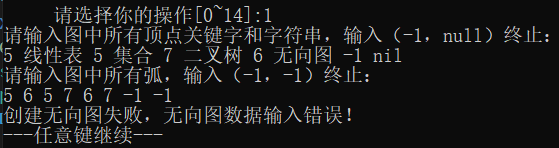
单无向图操作菜单测试结果如图4-6所示。



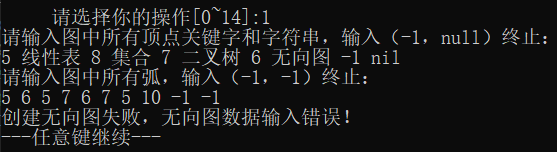
测试序号1



测试序号2



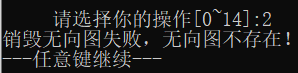
测试序号3



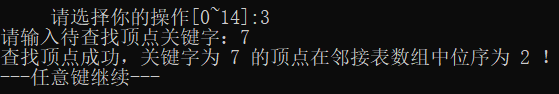
测试序号4



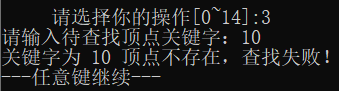
测试序号5



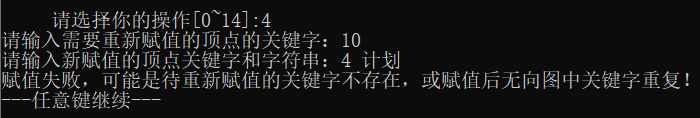
测试序号6



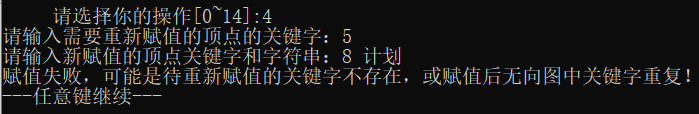
测试序号7



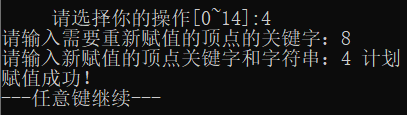
测试序号8



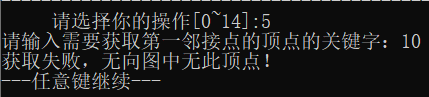
测试序号9



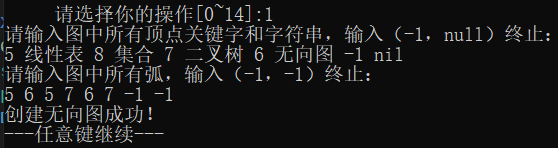
测试序号10

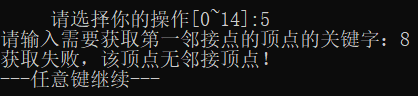


测试序号11

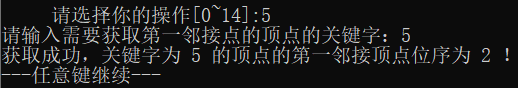


测试序号12

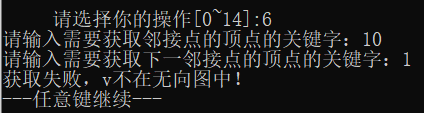




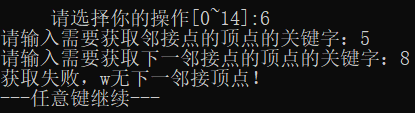
测试序号13



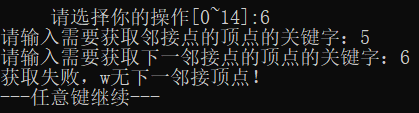
测试序号14



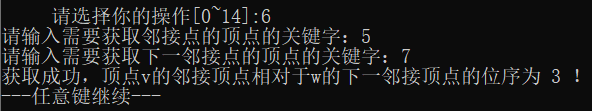
测试序号15



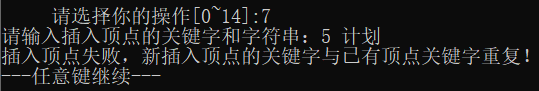
测试序号16



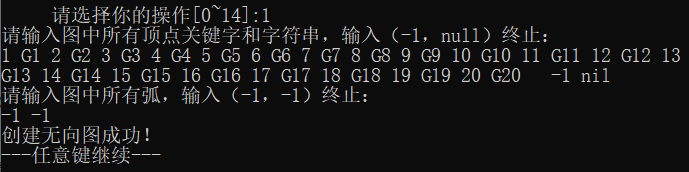
测试序号17

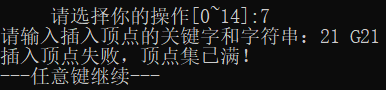


测试序号18

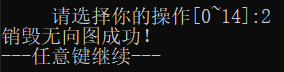


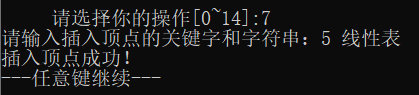
测试序号19



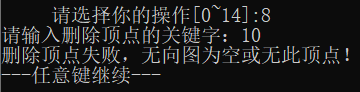


测试序号20

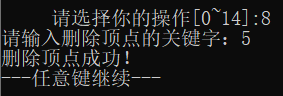




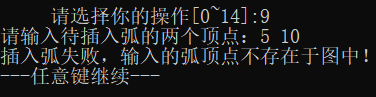
测试序号21



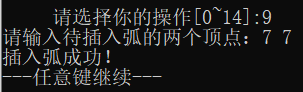
测试序号22



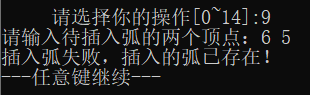
测试序号23



测试序号24



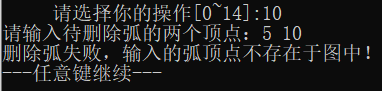
测试序号25



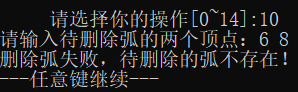
测试序号26



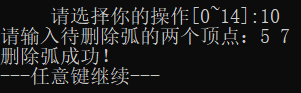
测试序号27



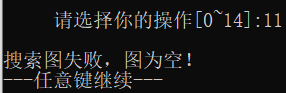
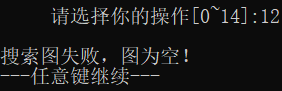
测试序号28



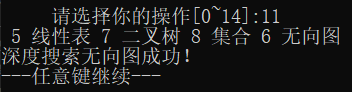
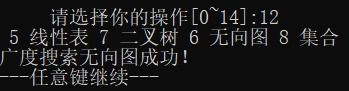
测试序号29



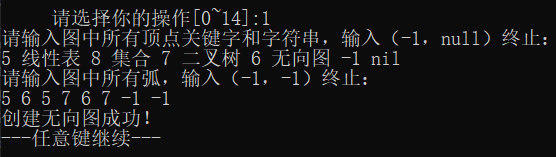
测试序号30

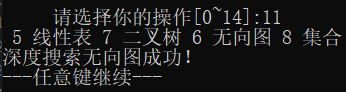
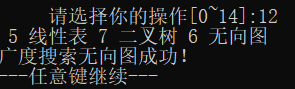
 

测试序号31

测试序号32

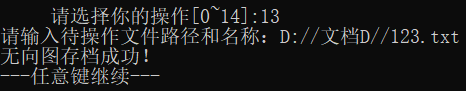


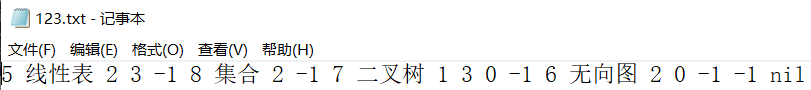
 

测试序号33

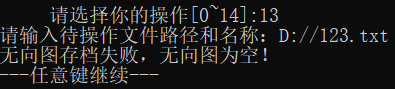
图4-6 单无向图操作菜单测试结果

文件操作菜单测试结果如图4-7所示。

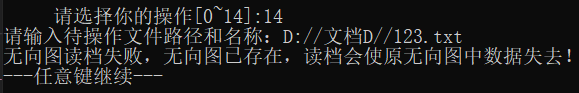




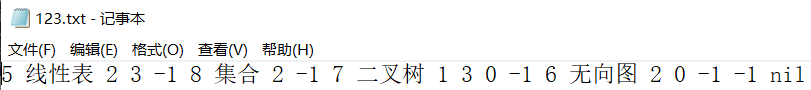
测试序号1

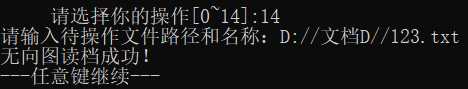


测试序号2



测试序号3





测试序号4

图4-7 文件操作菜单测试结果

### 4.4.3 测试分析

根据实验要求，以测试计划的3个模块的测试结果为根本依据，将测试结果总结如下：

多二叉树管理菜单测试：总计8项测试内容全部通过，通过率为100%。

单二叉树操作菜单测试：总计33项测试内容全部通过，通过率为100%。

文件操作菜单测试：总计4项内容全部通过，通过率为100%。

## 4.5 实验小结

本次实验是第4次数据结构实验，主要实现对无向图的管理和功能操作。实验使用邻接表作为物理结构，定义了一个顶点数组，每个顶点后链接一个单链表，实现对图中弧的记录。图的结构体变量中包括顶点数、弧数、邻接表、图类型等信息。

图是非常重要且最复杂的一种数据结构，可分为无向图、有向图、无向网、有向网四种，每一种可以有多种形式的物理结构表示。本次实验选取的无向图的邻接表物理结构，是比较基础的一种，有利于初学者掌握和熟悉图的基本知识。

在实验过程中，有较多实验操作需要的思维并不复杂，但是过程比较繁琐、细致，要完整考虑所有情况需要耗费较多耐心。如删除顶点的功能，删除顶点后后续顶点的位序也会随之改变，因此需要对整个图结构进行检查和修改才能保证弧关系正确。在本次实验中使用了深度搜索和广度搜索两种最基本的搜索方法。这两种搜索应用十分广泛，此次实验也相当于让我们打下了今后学习和运用更复杂算法的基础。

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

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

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

[4] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# 附录A 基于顺序存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2 //预定义的常量

typedef int status; //函数返回结果标志

typedef int ElemType; //数据元素类型定义

#define LIST\_INIT\_SIZE 100 //线性表存储结构分配的空间

#define LISTINCREMENT 10 //线性表空间分配增量

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem; //存放抽象数据元素的顺序表

int length; //线性表表长

int listsize; //线性表分配空间大小

}SqList;

typedef struct{ //线性表的管理表定义

struct { char name[30]; //线性表名称

SqList L;

} elem[10]; //存放线性表的集合

int length; //线性表集合的长度

int listsize; //线性表集合分配的空间

}LISTS;

void SecondaryTable(SqList &L,int op);//进入二级菜单

status InitInput(SqList &L); //线性表数据输入

status InitList(SqList &L); //线性表初始化

status DestroyList(SqList &L); //线性表销毁

status ClearList(SqList &L); //线性表清空

status ListEmpty(SqList L); //线性表判空

status ListLength(SqList L); //求线性表长度

status GetElem(SqList L,int i,ElemType &e); //获取线性表中某位置的元素

int LocateElem(SqList L,ElemType e); //查找元素在线性表中的位置

status PriorElem(SqList L,ElemType e,ElemType &pre); //获取线性表中某元素的前驱元素

status NextElem(SqList L,ElemType e,ElemType &next); //获取线性表中某元素的后继元素

status ListInsert(SqList &L,int i,ElemType e); //线性表插入元素

status ListDelete(SqList &L,int i,ElemType &e); //线性表删除元素

status ListTraverse(SqList L); //遍历输出线性表元素

status SaveList(SqList L,char FileName[]); //线性表存档

status LoadList(SqList &L,char FileName[]); //线性表读档

status AddList(LISTS &Lists,char ListName[]); //增加空线性表

status RemoveList(LISTS &Lists,char ListName[]); //移除线性表

int LocateList(LISTS Lists,char ListName[]); //查找线性表

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

int main(){

SqList L;

LISTS Lists;

int op=1;

int i,j;

ElemType e,pre,next;

char Name[30]={'\0'},FileName[100]={'\0'};

L.elem=NULL;

L.length=0;

L.listsize=LIST\_INIT\_SIZE;

Lists.length=0;

Lists.listsize=10;

for(j=0;j<10;j++)

{

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

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

}//初始化

while(op!=-3){

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

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

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

printf(" -1.MainTable/MultiList -2.SingleList\n");

printf(" -3.Exit\n");

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

printf(" 请选择所在的菜单层级：\n");

printf(" -1：主菜单实现多线性表管理，-2：临时单线性表基本操作。\n");

scanf("%d",&op);

if(op==-1)

{

while(op)

{

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

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

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

printf(" 15.AddList 16.RemoveList 17.LocateList\n");

printf(" 18.SecondaryTable\n");

printf(" 0. Exit\n");

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

for(i=0;i<5;i++)

{

if(Lists.elem[i].L.elem==NULL&&Lists.elem[i+5].L.elem==NULL)

printf("\t线性表%d：未创建 \t线性表%d：未创建",i+1,i+6);

else if(Lists.elem[i].L.elem==NULL&&Lists.elem[i+5].L.elem!=NULL)

printf("\t线性表%d：未创建 \t线性表%d：%s",i+1,i+6,Lists.elem[i+5].name);

else if(Lists.elem[i].L.elem!=NULL&&Lists.elem[i+5].L.elem==NULL)

printf("\t线性表%d：%s \t线性表%d：未创建",i+1,Lists.elem[i].name,i+6);

else

printf("\t线性表%d：%s \t线性表%d：%s",i+1,Lists.elem[i].name,i+6,Lists.elem[i+5].name);

printf("\n");

}

printf(" 请选择你的操作[0或15~18]:");

scanf("%d",&op);

switch(op){

case 15:

printf("请输入待添加的线性表名称：");

scanf("%s",Name);

j=AddList(Lists,Name);

if(j==OK)

{

printf("线性表添加成功！\n");

i=LocateList(Lists,Name);

InitInput(Lists.elem[i-1].L);

}

else if(j==ERROR)

printf("线性表添加失败，该线性表已存在！\n");

else if(j==OVERFLOW)

printf("线性表添加失败，线性表集合已满！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 16:

printf("请输入待删除的线性表名称：");

scanf("%s",Name);

j=RemoveList(Lists,Name);

if(j==OK)

printf("线性表删除成功！\n");

else if(j==ERROR)

printf("线性表删除失败，集合中无此线性表！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 17:

printf("请输入待查找的线性表名称：");

scanf("%s",Name);

j=LocateList(Lists,Name);

if(j>0)

printf("该线性表位于集合中第 %d 位！\n",j);

else

printf("线性表查找失败，集合中无此线性表！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 18:

printf("请输入待操作的线性表名称：");

scanf("%s",Name);

j=LocateList(Lists,Name);

if(j>0)

SecondaryTable(Lists.elem[j-1].L,op);

else

{

getchar();

printf("线性表不存在于集合中，不能进入单线性表操作！\n");

getchar();

break;

}

break;

case 0:

break;

}//end of switch

}//end of while

}//end of if

if(op==-2)

{

while(op)

{

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

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

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

printf(" 1. InitList 7. LocateElem 13.SaveList\n");

printf(" 2. DestroyList 8. PriorElem 14.LoadList\n");

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

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

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

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

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~14]:");

scanf("%d",&op);

switch(op){

case 1:

j=InitList(L);

if(j==OK)

{

printf("线性表创建成功！\n");

InitInput(L);

}

else if(j==ERROR)

printf("线性表创建失败！\n");

else if(j==INFEASIBLE)

printf("线性表已存在，不能进行线性表初始化操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 2:

j=DestroyList(L);

if(j==OK)

printf("线性表销毁成功！\n");

else if(j==INFEASIBLE)

printf("线性表不存在，不能进行线性表销毁操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 3:

if(ClearList(L)==OK)

printf("线性表清空成功！\n");

else

printf("线性表不存在，不能进行线性表清空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 4:

j=ListEmpty(L);

if(j==TRUE)

printf("线性表为空！\n");

else if(j==FALSE)

printf("线性表不为空！\n");

else if(j==INFEASIBLE)

printf("线性表不存在,不能进行线性表判空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 5:

j=ListLength(L);

if(j>=0)

printf("线性表长度为 %d ！\n",j);

else

printf("线性表不存在，不能求表长！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 6:

printf("请输入查找元素的位置：");

scanf("%d",&i);

j=GetElem(L,i,e);

if(j==OK)

printf("第 %d 位元素是 %d ！\n",i,e);

else if(j==ERROR)

printf("所选择的元素位置不合法，可能小于1或大于线性表长度！\n");

else if(j==INFEASIBLE)

printf("线性表不存在，不能进行元素获取！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 7:

printf("请输入待查找元素：");

scanf("%d",&e);

j=LocateElem(L,e);

if(j>0)

printf("待查找元素 %d 在线性表中的第 %d 位！\n",e,j);

else if(j==ERROR)

printf("查找元素失败，该元素不在线性表中！\n");

else if(j==INFEASIBLE)

printf("线性表不存在，不能进行元素查找！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 8:

printf("请输入待查找元素以获取其前驱元素：");

scanf("%d",&e);

j=PriorElem(L,e,pre);

if(j==OK)

printf("元素 %d 的前驱元素是 %d ！\n",e,pre);

else if(j==ERROR)

printf("元素 %d 没有前驱元素！\n",e);

else if(j==INFEASIBLE)

printf("线性表不存在，不能获取元素！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 9:

printf("请输入待查找元素以获取其后继元素：");

scanf("%d",&e);

j=NextElem(L,e,next);

if(j==OK)

printf("元素 %d 的后继元素是 %d ！\n",e,next);

else if(j==ERROR)

printf("元素 %d 没有后继元素！\n",e);

else if(j==INFEASIBLE)

printf("线性表不存在，不能获取元素！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case 10:

printf("请输入待插入元素：");

scanf("%d",&e);

printf("请输入插入位置后一个元素序号：");

scanf("%d",&i);

j=ListInsert(L,i,e);

if(j==OK)

printf("插入元素成功！\n");

else if(j==ERROR)

printf("插入位置不正确！\n");

else if(j==INFEASIBLE)

printf("线性表不存在，不能进行插入操作！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case 11:

printf("请输入需要删除的元素的位置：");

scanf("%d",&i);

j=ListDelete(L,i,e);

if(j==OK)

printf("删除元素成功，所删除元素为 %d ！\n",e);

else if(j==ERROR)

printf("删除位置不正确！\n");

else if(j==INFEASIBLE)

printf("线性表不存在，不能进行删除操作！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case 12:

j=ListTraverse(L);

if(j==OK)

printf("\n线性表遍历成功！\n");

else if(j==ERROR)

printf("\n线性表中无元素！\n");

else if(j==INFEASIBLE)

printf("\n线性表不存在！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case 13:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=SaveList(L,FileName);

if(j==OK)

printf("线性表中数据已成功写入指定文件中！\n");

else if(j==INFEASIBLE)

printf("线性表不存在！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case 14:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=LoadList(L,FileName);

if(j==OK)

printf("文件中数据已成功写入线性表中！\n");

else if(j==INFEASIBLE)

printf("线性表已存在，直接写入会导致表中原数据丢失！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case 0:

break;

}//end of switch

}//end of while

}//end of if

}//end of while

printf("欢迎下次再使用本系统！\n");

}//end of main()

//菜单功能选择 1 初始化线性表

status InitList(SqList& L)

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

{

if(L.elem==NULL)

{//使用malloc函数为顺序表开辟空间，表长为LIST\_INIT\_SIZE

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

L.length=0;

L.listsize=LIST\_INIT\_SIZE;

return OK;

}

else

return INFEASIBLE; //线性表已存在，不能构造空表

}

//菜单功能选择 2 销毁线性表

status DestroyList(SqList& L)

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

{

if(L.elem)//通过顺序表首地址的指针是否为空判断线性表是否存在

{

L.elem=NULL;

free(L.elem);

L.length=0; //将销毁后线性表表长和空间大小均置为0

L.listsize=0;

return OK;

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 3 清空线性表

status ClearList(SqList& L)

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

{

if(L.elem)

{

L.length=0; //清空线性表只清除线性表中元素，不销毁表头

return OK;

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 4 线性表判空

status ListEmpty(SqList L)

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

{

if(L.elem)

{

if(L.length==0) //通过线性表表长判断线性表是否为空

return TRUE;

else

return FALSE;

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 5 求线性表长度

status ListLength(SqList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

if(L.elem)

return L.length; //线性表长度可直接由L.length反映

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 6 获取元素

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

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

{

if(L.elem)

{

if(i<1||i>L.length) //待获取元素在线性表中位置不合法

return ERROR;

e=L.elem[i-1]; //i为逻辑顺序，i-1为物理顺序

return OK;

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 7 查找元素

int LocateElem(SqList L,ElemType e)

// 如果线性表L存在，查找元素e在线性表L中的位置序号并返回该序号；如果e不存在，返回ERROR；当线性表L不存在时，返回INFEASIBLE（即-1）。

{

int i,flag=1;

if(L.elem)

{

for(i=0;i<L.length;i++) //遍历线性表

{

if(e==L.elem[i]) //获取线性表中与待查找元素相同的结点

{

flag=0; //flag可用以标记线性表中是否有与待查找元素相同的结点（无flag也可）

return i+1; //返回值为逻辑序号

}

}

if(flag)

return ERROR; //查找与e元素相同的结点失败

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 8 获取前驱元素

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

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

{

int i;

if(L.elem)

{

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

{//遍历线性表

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

{//查找到对应元素后，返回其前驱元素

pre=L.elem[i-1];

return OK;

}

}

return ERROR; //查找与e元素相同的结点失败

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 9 获取后继元素

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

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

{

int i;

if(L.elem)

{

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

{//遍历线性表

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

{//查找到对应元素后，返回其后继元素

next=L.elem[i+1];

return OK;

}

}

return ERROR; //查找与e元素相同的结点失败

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 10 插入元素

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

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

{

int j;

if(L.elem)

{

if(i<=L.length+1&&i>0&&L.length<L.listsize) //i位置合法且线性表未满

{

for(j=L.length;j>=i-1;j--) //将第i个位置（逻辑位置）后的元素全部后移一位，将待插入元素插入到i对应位置

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

L.elem[i-1]=e;

L.length++; //线性表长度增加

return OK;

}

else if(i<=L.length+1&&i>0&&L.length>=L.listsize) //i位置合法且线性表已满

{

L.elem=(ElemType \*)realloc(L.elem,(L.listsize+LISTINCREMENT)\*sizeof(ElemType)); //为线性表增加空间以存放新增数据元素

L.listsize+=LISTINCREMENT; //线性表空间增加

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

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

L.elem[i-1]=e;

L.length++;

return OK;

}

else

return ERROR; //i位置不合法

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 11 删除元素

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

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

{

if(L.elem)

{

if(L.length>0&&i<L.length&&i>0) //i位置合法

{

e=L.elem[i-1]; //利用i定位元素，并保存在e中

for(;i<L.length;i++) //将i位置之后的元素在线性表中前移一位

L.elem[i-1]=L.elem[i];

L.length--; //表长减1

return OK;

}

else

return ERROR; //i位置不合法

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 12 遍历线性表

status ListTraverse(SqList L)

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

{

if(L.elem)

{

if(!L.length) //线性表为空

return ERROR;

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

{

if(i!=L.length-1) //遍历线性表达到末位

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

else

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

}

return OK;

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 13 线性表存档

status SaveList(SqList L,char FileName[])

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

{

FILE \*fp;

int i=0;

if(L.elem)

{

fp=fopen(FileName,"wb");

if(!fp) //文件打开失败

return ERROR;

while(i<L.length)

{

fprintf(fp,"%d ",L.elem[i]); //将线性表中数据写入文件

i++;

}

fclose(fp);

return OK;

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 14 线性表读档

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

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

{

FILE \*fp;

int i=0;

if(L.elem)

return INFEASIBLE; //线性表已存在，读档会使元素失去

else

{

L.elem=(ElemType \*)malloc(sizeof(ElemType)\*LIST\_INIT\_SIZE); //初始化线性表

L.length=0;

L.listsize=LIST\_INIT\_SIZE;

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

{

while((fscanf(fp,"%d ",&L.elem[i]))!=EOF) //文件未读取完成

{

if(L.length<L.listsize-1) //线性表未满，直接读入数据

{

L.length++;

i++;

}

else //线性表已满，重新分配空间再读入数据

{

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

L.listsize+=LISTINCREMENT;

L.length++;

i++;

}

}

}

fclose(fp);

return OK;

}

}

//菜单功能选择 15 多线性表管理之增加空线性表

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

// 在Lists中增加一个名称为ListName的空线性表，线性表数据由主函数插入。

{

if(Lists.length<Lists.listsize)

{

int i,flag=0;

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

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

{//对线性表名称进行比较查找，查找成功时标记并退出循环

flag=1;

break;

}

if(!flag) //查找失败，说明中无此线性表，可以新增

{

strcpy(Lists.elem[Lists.length].name,ListName); //调用strcpy函数复制线性表名称

Lists.elem[Lists.length].L.elem=(ElemType \*)malloc(LIST\_INIT\_SIZE\*sizeof(ElemType)); //为线性表开辟空间初始化空间大小和长度

Lists.elem[Lists.length].L.listsize=LIST\_INIT\_SIZE; //初始化空间大小和长度

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

Lists.length++; //集合长度增加

return OK;

}

else //线性表已存在，不能新增

return ERROR;

}

else //线性表集合已满，不能添加线性表

return OVERFLOW;

}

//菜单功能选择 16 多线性表管理之删除线性表

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

// 在Lists中删除一个名称为ListName的线性表。

{

int i,flag=0;

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

{

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

{//对线性表名称进行比较查找，查找成功时标记并退出循环

flag=1;

break;

}

}

if(flag) //查找成功

{

for(;i<Lists.length-1;i++) //将查找到的线性表后的线性表顺序前移一位

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

Lists.elem[Lists.length-1].name[0]='\0'; //将原本最末位的线性表名称和内容置为空

Lists.elem[Lists.length-1].L.elem=NULL;

Lists.elem[Lists.length-1].L.length=0;

Lists.elem[Lists.length-1].L.listsize=0;

Lists.length--;

return OK;

}

else

return ERROR;

}

//菜单功能选择 17 多线性表管理之查找线性表

int LocateList(LISTS Lists,char ListName[])

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

{

int i,flag=0;

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

{

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

{//对线性表名称进行比较查找，查找成功时标记并退出循环

flag=1;

break;

}

}

if(!flag||!Lists.length) //查找失败或线性表集合长度为0

return ERROR;

else //查找成功返回逻辑位置

return i+1;

}

//辅助函数：线性表数据输入

status InitInput(SqList &L)

// 选择是否向线性表中输入数据，是则由用户输入以0为结束标志的一组数据，返回OK，否则返回0。

{

int ord, temp;

printf("输入数据选择1，否则选择0：\n");

scanf("%d",&ord);

if(ord==0) //以0为结束标志

return 0;

else

{

printf("请输入一组数据，0为结束标志：\n");

scanf("%d",&temp);

while(temp != 0) //元素输入截止的标志

{

ListInsert(L,L.length+1,temp); //调用插入函数在线性表尾部增加元素

scanf("%d",&temp);

}

printf("输入成功！\n");

return OK;

}

}

//辅助函数：进入二级菜单

void SecondaryTable(SqList &L,int op)

// 将所选的多线性表集合中的待操作线性表传递进入二级菜单中，实现对单线性表的基本操作。

{

int i,j;

ElemType e,pre,next;

char FileName[30];

while(op!=-1)

{

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

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

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

printf(" 1. InitList 7. LocateElem 13.SaveList\n");

printf(" 2. DestroyList 8. PriorElem 14.LoadList\n");

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

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

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

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

printf(" -1. Exit\n");

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

printf(" 请选择你的操作[-1或1~14]:");

scanf("%d",&op);

switch(op){

case 1:

j=InitList(L);

if(j==OK)

{

printf("线性表创建成功！\n");

InitInput(L);

}

else if(j==ERROR)

printf("线性表创建失败！\n");

else if(j==INFEASIBLE)

printf("线性表已存在，不能进行线性表初始化操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 2:

j=DestroyList(L);

if(j==OK)

printf("线性表销毁成功！\n");

else if(j==ERROR)

printf("线性表销毁失败！\n");

else if(j==INFEASIBLE)

printf("线性表不存在，不能进行线性表销毁操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 3:

if(ClearList(L)==OK)

printf("线性表清空成功！\n");

else

printf("线性表不存在，不能进行线性表清空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 4:

j=ListEmpty(L);

if(j==TRUE)

printf("线性表为空！\n");

else if(j==FALSE)

printf("线性表不为空！\n");

else if(j==INFEASIBLE)

printf("线性表不存在,不能进行线性表判空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 5:

j=ListLength(L);

if(j>=0)

printf("线性表长度为 %d ！\n",j);

else

printf("线性表不存在，不能求表长！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 6:

printf("请输入查找元素的位置：");

scanf("%d",&i);

j=GetElem(L,i,e);

if(j==OK)

printf("第 %d 位元素是 %d ！\n",i,e);

else if(j==ERROR)

printf("所选择的元素位置不合法，可能小于1或大于线性表长度！\n");

else if(j==INFEASIBLE)

printf("线性表不存在，不能进行元素获取！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 7:

printf("请输入待查找元素：");

scanf("%d",&e);

j=LocateElem(L,e);

if(j>0)

printf("待查找元素 %d 在线性表中的第 %d 位！\n",e,j);

else if(j==ERROR)

printf("查找元素失败，该元素不在线性表中！\n");

else if(j==INFEASIBLE)

printf("线性表不存在，不能进行元素查找！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 8:

printf("请输入待查找元素以获取其前驱元素：");

scanf("%d",&e);

j=PriorElem(L,e,pre);

if(j==OK)

printf("元素 %d 的前驱元素是 %d ！\n",e,pre);

else if(j==ERROR)

printf("元素 %d 没有前驱元素！\n",e);

else if(j==INFEASIBLE)

printf("线性表不存在，不能获取元素！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 9:

printf("请输入待查找元素以获取其后继元素：");

scanf("%d",&e);

j=NextElem(L,e,next);

if(j==OK)

printf("元素 %d 的后继元素是 %d ！\n",e,next);

else if(j==ERROR)

printf("元素 %d 没有后继元素！\n",e);

else if(j==INFEASIBLE)

printf("线性表不存在，不能获取元素！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case 10:

printf("请输入待插入元素：");

scanf("%d",&e);

printf("请输入插入位置后一个元素序号：");

scanf("%d",&i);

j=ListInsert(L,i,e);

if(j==OK)

printf("插入元素成功！\n");

else if(j==ERROR)

printf("插入位置不正确！\n");

else if(j==INFEASIBLE)

printf("线性表不存在，不能进行插入操作！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case 11:

printf("请输入需要删除的元素的位置：");

scanf("%d",&i);

j=ListDelete(L,i,e);

if(j==OK)

printf("删除元素成功，所删除元素为 %d ！\n",e);

else if(j==ERROR)

printf("删除位置不正确！\n");

else if(j==INFEASIBLE)

printf("线性表不存在，不能进行删除操作！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case 12:

j=ListTraverse(L);

if(j==OK)

printf("\n线性表遍历成功！\n");

else if(j==ERROR)

printf("\n线性表中无元素！\n");

else if(j==INFEASIBLE)

printf("\n线性表不存在！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case 13:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=SaveList(L,FileName);

if(j==OK)

printf("线性表中数据已成功写入指定文件中！\n");

else if(j==INFEASIBLE)

printf("线性表不存在！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case 14:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=LoadList(L,FileName);

if(j==OK)

printf("文件中数据已成功写入线性表中！\n");

else if(j==INFEASIBLE)

printf("线性表已存在，直接写入会导致表中原数据丢失！\n");

getchar();

printf("---任意键继续---");

getchar();

break;

case -1:

break;

}//end of switch

}//end of while

}

# 附录B 基于链式存储结构线性表实现的源程序

/\* Linear Table On Sequence Structure \*/

#include "stdio.h"

#include "stdlib.h"

#include <string.h>

#include <malloc.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2 //预定义的常量

typedef int status;

typedef int ElemType; //数据元素类型定义

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

ElemType data; //单链表数据元素存放

struct LNode \*next; //单链表下一结点指向

}LNode,\*LinkList;

typedef struct{ //线性表的管理表定义

struct { char name[30]; //线性表名称

LinkList L; //线性表头指针

} elem[10]; //存放线性表的集合

int length; //线性表集合的长度

int listsize; //线性表集合分配的空间

}LISTS;

void SecondaryTable(LinkList &L,int op);

status InitInput(LinkList &L); //线性表数据输入

status InitList(LinkList &L); //线性表初始化

status DestroyList(LinkList &L); //线性表销毁

status ClearList(LinkList &L); //线性表清空

status ListEmpty(LinkList L); //线性表判空

int ListLength(LinkList L); //求线性表表长

status GetElem(LinkList L,int i,ElemType &e); //获取元素

status LocateElem(LinkList L,ElemType e); //查找元素

status PriorElem(LinkList L,ElemType e,ElemType &pre); //获取前驱元素

status NextElem(LinkList L,ElemType e,ElemType &next); //获取后继元素

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

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

status ListTraverse(LinkList L); //遍历线性表

status SaveList(LinkList L,char FileName[]); //线性表存档

status LoadList(LinkList &L,char FileName[]); //线性表读档

status AddList(LISTS &Lists,char ListName[]); //增加空线性表

status RemoveList(LISTS &Lists,char ListName[]); //移除线性表

int LocateList(LISTS Lists,char ListName[]); //查找线性表

int main(){

LinkList L;

LISTS Lists;

int op=1;

int i,j;

ElemType e,pre,next;

char Name[30]={'\0'},FileName[30]={'\0'};

L->data=NULL;

L->next=NULL;

Lists.length=0;

Lists.listsize=10;

for(j=0;j<10;j++)

{

Lists.elem[j].L=NULL;

}//初始化

while (op!=-3){

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

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

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

printf(" -1.MainTable/MultiList -2.SingleList\n");

printf(" -3.Exit\n");

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

printf(" 请选择所在的菜单层级：\n");

printf(" -1：主菜单实现多线性表管理，-2：临时单线性表基本操作。\n");

scanf("%d",&op);

if(op==-1){

while(op)

{

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

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

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

printf(" 15.AddList 16.RemoveList 17.LocateList\n");

printf(" 18.SecondaryTable\n");

printf(" 0. Exit\n");

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

for(i=0;i<5;i++)

{

if(Lists.elem[i].L==NULL&&Lists.elem[i+5].L==NULL)

printf("\t线性表%d：未创建 \t线性表%d：未创建",i+1,i+6);

else if(Lists.elem[i].L==NULL&&Lists.elem[i+5].L!=NULL)

printf("\t线性表%d：未创建 \t线性表%d：%s",i+1,i+6,Lists.elem[i+5].name);

else if(Lists.elem[i].L!=NULL&&Lists.elem[i+5].L==NULL)

printf("\t线性表%d：%s \t线性表%d：未创建",i+1,Lists.elem[i].name,i+6);

else

printf("\t线性表%d：%s \t线性表%d：%s",i+1,Lists.elem[i].name,i+6,Lists.elem[i+5].name);

printf("\n");

}

printf("\n 请选择你的操作[0或15~18]:");

scanf("%d",&op);

switch(op){

case 15:

printf("请输入待添加的线性表名称：");

scanf("%s",Name);

j=AddList(Lists,Name);

if(j==OK)

{

printf("线性表添加成功！\n");

i=LocateList(Lists,Name);

InitInput(Lists.elem[i-1].L);

}

else if(j==ERROR)

printf("线性表添加失败，该线性表已存在！\n");

else if(j==OVERFLOW)

printf("线性表添加失败，线性表集合已满！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 16:

printf("请输入待删除的线性表名称：");

scanf("%s",Name);

j=RemoveList(Lists,Name);

if(j==OK)

printf("线性表删除成功！\n");

else if(j==ERROR)

printf("线性表删除失败，集合中无此线性表！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 17:

printf("请输入待查找的线性表名称：");

scanf("%s",Name);

j=LocateList(Lists,Name);

if(j>0)

printf("该线性表位于集合中第 %d 位！\n",j);

else

printf("线性表查找失败，集合中无此线性表！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 18:

printf("请输入待操作的线性表名称：");

scanf("%s",Name);

j=LocateList(Lists,Name);

if(j>0)

SecondaryTable(Lists.elem[j-1].L,op);

else

{

getchar();

printf("线性表不存在于集合中，不能进入单线性表操作！\n");

printf("---任意键继续---\n");

getchar();

break;

}

break;

case 0:

break;

}//end of switch

}//end of while

}//end of if

if(op==-2){

while(op){

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

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

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

printf(" 1. InitList 7. LocateElem 13.SaveList\n");

printf(" 2. DestroyList 8. PriorElem 14.LoadList\n");

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

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

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

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

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~14]:");

scanf("%d",&op);

switch(op){

case 1:

j=InitList(L);

if(j==OK)

{

printf("创建线性表成功！\n");

InitInput(L);

}

else

printf("创建线性表失败，线性表已存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 2:

j=DestroyList(L);

if(j==OK)

printf("销毁线性表成功！\n");

else

printf("线性表不存在，不能进行销毁操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 3:

j=ClearList(L);

if(j==OK)

printf("清空线性表成功！\n");

else

printf("线性表不存在，不能进行清空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 4:

j=ListEmpty(L);

if(j==TRUE)

printf("线性表为空！\n");

else if(j==FALSE)

printf("线性表不为空！\n");

else

printf("线性表不存在，不能进行判空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 5:

j=ListLength(L);

if(j>=0)

printf("线性表长度为 %d ！\n",j);

else

printf("线性表不存在，不能进行求表长操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 6:

printf("请输入查找元素的位置：");

scanf("%d",&i);

j=GetElem(L,i,e);

if(j==OK)

printf("获取元素成功，第 %d 个元素是 %d ！\n",i,e);

else if(j==ERROR)

printf("获取元素失败，i的值不合法！\n");

else

printf("线性表不存在，不能获取元素！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 7:

printf("请输入待查找元素：");

scanf("%d",&e);

j=LocateElem(L,e);

if(j>0)

printf("查找元素成功，%d 是线性表中第 %d 个元素！\n",e,j);

else if(j==ERROR)

printf("查找元素失败，%d 不在线性表中！\n",e);

else

printf("线性表不存在，不能查找元素！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 8:

printf("请输入待查找元素以获取其前驱元素：");

scanf("%d",&e);

j=PriorElem(L,e,pre);

if(j==OK)

printf("获取元素前驱成功，%d 的前驱元素是 %d ！\n",e,pre);

else if(j==ERROR)

printf("获取元素前驱失败，%d 无前驱元素！\n",e);

else

printf("线性表不存在，不能获取前驱元素！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 9:

printf("请输入待查找元素以获取其后继元素：");

scanf("%d",&e);

j=NextElem(L,e,next);

if(j==OK)

printf("获取元素后继成功，%d 的后继元素是 %d ！\n",e,next);

else if(j==ERROR)

printf("获取元素后继失败，%d 无后继元素！\n",e);

else

printf("线性表不存在，不能获取后继元素！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 10:

printf("请输入待插入元素：");

scanf("%d",&e);

printf("请输入插入位置后一个元素序号：");

scanf("%d",&i);

j=ListInsert(L,i,e);

if(j==OK)

printf("插入元素成功！\n");

else if(j==ERROR)

printf("插入元素失败，插入位置不正确！\n");

else

printf("线性表不存在，不能进行插入操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 11:

printf("请输入需要删除的元素的位置：");

scanf("%d",&i);

j=ListDelete(L,i,e);

if(j==OK)

printf("删除元素成功，删除的元素为 %d ！\n",e);

else if(j==ERROR)

printf("删除元素失败，删除位置不正确！\n");

else

printf("线性表不存在，不能进行删除操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 12:

j=ListTraverse(L);

if(j==OK)

printf("\n遍历线性表成功！\n");

else if(j==ERROR)

printf("\n线性表中无元素！\n");

else

printf("\n线性表不存在，不能进行遍历操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 13:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=SaveList(L,FileName);

if(j==OK)

printf("线性表存档成功！\n");

else

printf("线性表不存在，不能进行存档操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 14:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=LoadList(L,FileName);

if(j==OK)

printf("线性表读档成功！\n");

else

printf("线性表已存在，读档可能失去原始数据！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

}//end of switch

}//end of while

}//end of if

}

printf("欢迎下次再使用本系统！\n");

}//end of main()

//菜单选择功能 1 创建线性表

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。

{

if(L) //表头结点已存在，说明线性表存在

return INFEASIBLE;

else

{

L=(LinkList)malloc(sizeof(LNode)); //初始化表头结点

return OK;

}

}

//菜单选择功能 2 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)

{

LNode \*p=L->next;

while(p)

{//利用指针p逐个释放单链表结点空间

L->next=p->next;

free(p);

p=NULL;

p=L->next;

}

free(L);

L=NULL; //释放表头结点空间

return OK;

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 3 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。

{

if(L)

{

struct LNode \*p=L->next;

while(p)

{//利用指针p逐个释放单链表结点空间

L->next=p->next;

free(p);

p=L->next;

}

return OK; //表头结点依然保留

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 4 线性表判空

status ListEmpty(LinkList L)

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

{

if(L)

if(L->next==NULL) //线性表存在且表头结点的下一结点不为空

return TRUE;

else

return FALSE;

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 5 求表长

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。

{

int len;

if(L)

{

LinkList t=L;

for(;t;t=t->next)

{//利用指针t遍历线性表，用len记录表长

if(t->next)

len++;

}

return len;

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 6 获取元素

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

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

{

if(L)

{

if(i<1) //i的位置不合法

return ERROR;

else

{

LinkList t=L;

for(;t;i--,t=t->next)

if(!i)

{//i递减至0时即查找到元素，保存在e中

e=t->data;

return OK;

}

return ERROR; //i的位置大于表长

}

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 7 查找元素

status LocateElem(LinkList L,ElemType e)

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

{

if(L)

{

LinkList t=L;

for(int i=0;t;t=t->next,i++) //用t遍历线性表，用i记录元素位置

if(e==t->data)

return i;

if(!t) //遍历结束后仍无法查找到e，即元素e不存在于线性表中

return ERROR;

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 8 获取前驱元素

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

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

{

if(L)

{

LinkList t=L;

for(int i=0;t->next;t=t->next,i++)

if(e==t->next->data&&i!=0) //由于要获取前驱元素，因此需要找到t的下一结点值与e相等，将此时t结点值保存在pre中

{

pre=t->data;

return OK;

}

return ERROR; //元素e不在线性表中

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 9 获取后继元素

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

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

{

if(L)

{

LinkList t=L;

for(int i=0;t->next;t=t->next,i++)

if(e==t->data) //查找到与e相等的结点后，返回后一结点值

{

next=t->next->data;

return OK;

}

return ERROR; //元素e不在线性表中

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 10 插入元素

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

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

{

if(L)

{

LinkList t=L;

for(int j=0;t;t=t->next,j++)

{

if(j+1==i) //j标记t指向i位置的前一个元素

{//利用两个结点指针完成插入操作

LinkList tag;

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

tag->data=e;

tag->next=t->next;

t->next=tag;

return OK;

}

}

return ERROR; //i位置不合法

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 11 删除元素

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

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

{

if(L)

{

LinkList t=L;

for(int j=0;t;t=t->next,j++)

if(j+1==i) //j标记t指向i位置的前一个元素

{//利用两个结点指针完成删除操作

LinkList tag=t->next;

e=t->next->data;

t->next=t->next->next;

free(tag);

return OK;

}

return ERROR; //i位置不合法

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 12 遍历线性表

status ListTraverse(LinkList L)

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

{

if(L)

{

LinkList t=L->next;

int flag=0;

for(;t;t=t->next)

{//用t指针遍历线性表

flag=1; //标记线性表是否为空（不使用flag也可有正确返回值）

if(t->next)

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

else

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

}

if(flag)

return OK;

else

return ERROR; //线性表为空

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 13 线性表存档

status SaveList(LinkList L,char FileName[])

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

{

if(L)

{

LinkList t=L->next;

FILE \*fp;

fp=fopen(FileName,"wb");

while(t)

{//用t指针遍历线性表，将线性表元素写入文件中

fprintf(fp,"%d ",t->data);

t=t->next;

}

fclose(fp);

return OK;

}

else

return INFEASIBLE; //线性表不存在

}

//菜单功能选择 14 线性表读档

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。

{

if(!L)

{

L=(LinkList)malloc(sizeof(LNode)); //初始化表头结点

L->next=NULL;

FILE \*fp;

LinkList t;

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

{

L->next=(LinkList)malloc(sizeof(LNode)); //使用先进先出的单链表，将文件中数据元素读入线性表中

t=L->next;

t->next=NULL;

while((fscanf(fp,"%d ",&t->data))!=EOF)

{

if(!feof(fp))

{

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

t=t->next;

}

}

t->next=NULL;

fclose(fp);

}

return OK;

}

else

return INFEASIBLE; //线性表已存在，读档会使其中元素失去

}

//菜单功能选择 15 多线性表管理之增加空线性表

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

// 在Lists中增加一个名称为ListName的空线性表，线性表数据由主函数插入。

{

if(Lists.length<Lists.listsize)

{

int i,flag=0;

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

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

{//对线性表名称进行比较查找，查找成功时标记并退出循环

flag=1;

break;

}

if(!flag) //查找失败，说明中无此线性表，可以新增

{

strcpy(Lists.elem[Lists.length].name,ListName); //调用strcpy函数复制线性表名称

Lists.elem[Lists.length].L=(LinkList)malloc(sizeof(LNode)); //为线性表表头结点开辟空间

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

Lists.length++; //集合长度增加

return OK;

}

else //线性表已存在，不能新增

return ERROR;

}

else //线性表集合已满，不能添加线性表

return OVERFLOW;

}

//菜单功能选择 16 多线性表管理之删除线性表

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

// 在Lists中删除一个名称为ListName的线性表。

{

int i,flag=0;

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

{

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

{//对线性表名称进行比较查找，查找成功时标记并退出循环

flag=1;

break;

}

}

if(flag)

{

DestroyList(Lists.elem[i].L); //调用销毁线性表函数销毁当前线性表

for(;i<Lists.length-1;i++) //将查找到的线性表后的线性表顺序前移一位

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

if(i!=Lists.length-1) //若最末位线性表未被销毁，调用销毁线性表函数销毁

DestroyList(Lists.elem[Lists.length-1].L);

Lists.elem[Lists.length-1].name[0]='\0'; //将原本最末位的线性表名称置为空，释放其表头结点

Lists.elem[Lists.length-1].L=NULL;

free(Lists.elem[Lists.length-1].L);

Lists.length--;

return OK;

}

else

return ERROR;

}

//菜单功能选择 17 多线性表管理之查找线性表

int LocateList(LISTS Lists,char ListName[])

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

{

int i,flag=0;

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

{

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

{//对线性表名称进行比较查找，查找成功时标记并退出循环

flag=1;

break;

}

}

if(!flag||!Lists.length) //查找失败或线性表集合长度为0

return ERROR;

else //查找成功返回逻辑位置

return i+1;

}

//辅助函数：输入线性表元素

status InitInput(LinkList &L)

// 选择是否向线性表中输入数据，是则由用户输入以0为结束标志的一组数据，返回OK，否则返回0。

{

int ord,temp;

LinkList tail=L,p=L;

printf("输入数据选择1，否则选择0：\n");

scanf("%d",&ord);

if(ord)

{

printf("请输入一组数据创建先进先出链表，0为结束标志：\n");

scanf("%d",&temp);

while(temp)

{//先进先出的单链表创建

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

tail->data=temp;

tail->next=NULL;

p->next=tail;

p=p->next;

scanf("%d",&temp);

}

printf("输入成功！\n");

return 0;

}

else

return 0;

}

//辅助函数：进入二级菜单

void SecondaryTable(LinkList &L,int op)

// 将所选的多线性表集合中的待操作线性表传递进入二级菜单中，实现对单线性表的基本操作。

{

int i,j;

ElemType e,pre,next;

char FileName[30];

while(op!=-1){

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

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

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

printf(" 1. InitList 7. LocateElem 13.SaveList\n");

printf(" 2. DestroyList 8. PriorElem 14.LoadList\n");

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

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

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

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

printf(" -1. Exit\n");

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

printf(" 请选择你的操作[-1或1~14]:");

scanf("%d",&op);

switch(op){

case 1:

j=InitList(L);

if(j==OK)

{

printf("创建线性表成功！\n");

InitInput(L);

}

else

printf("创建线性表失败，线性表已存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 2:

j=DestroyList(L);

if(j==OK)

printf("销毁线性表成功！\n");

else

printf("线性表不存在，不能进行销毁操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 3:

j=ClearList(L);

if(j==OK)

printf("清空线性表成功！\n");

else

printf("线性表不存在，不能进行清空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 4:

j=ListEmpty(L);

if(j==TRUE)

printf("线性表为空！\n");

else if(j==FALSE)

printf("线性表不为空！\n");

else

printf("线性表不存在，不能进行判空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 5:

j=ListLength(L);

if(j>=0)

printf("线性表长度为 %d ！\n",j);

else

printf("线性表不存在，不能进行求表长操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 6:

printf("请输入查找元素的位置：");

scanf("%d",&i);

j=GetElem(L,i,e);

if(j==OK)

printf("获取元素成功，第 %d 个元素是 %d ！\n",i,e);

else if(j==ERROR)

printf("获取元素失败，i的值不合法！\n");

else

printf("线性表不存在，不能获取元素！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 7:

printf("请输入待查找元素：");

scanf("%d",&e);

j=LocateElem(L,e);

if(j>0)

printf("查找元素成功，%d 是线性表中第 %d 个元素！\n",e,j);

else if(j==ERROR)

printf("查找元素失败，%d 不在线性表中！\n",e);

else

printf("线性表不存在，不能查找元素！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 8:

printf("请输入待查找元素以获取其前驱元素：");

scanf("%d",&e);

j=PriorElem(L,e,pre);

if(j==OK)

printf("获取元素前驱成功，%d 的前驱元素是 %d ！\n",e,pre);

else if(j==ERROR)

printf("获取元素前驱失败，%d 无前驱元素！\n",e);

else

printf("线性表不存在，不能获取前驱元素！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 9:

printf("请输入待查找元素以获取其后继元素：");

scanf("%d",&e);

j=NextElem(L,e,next);

if(j==OK)

printf("获取元素后继成功，%d 的后继元素是 %d ！\n",e,next);

else if(j==ERROR)

printf("获取元素后继失败，%d 无后继元素！\n",e);

else

printf("线性表不存在，不能获取后继元素！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 10:

printf("请输入待插入元素：");

scanf("%d",&e);

printf("请输入插入位置后一个元素序号：");

scanf("%d",&i);

j=ListInsert(L,i,e);

if(j==OK)

printf("插入元素成功！\n");

else if(j==ERROR)

printf("插入元素失败，插入位置不正确！\n");

else

printf("线性表不存在，不能进行插入操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 11:

printf("请输入需要删除的元素的位置：");

scanf("%d",&i);

j=ListDelete(L,i,e);

if(j==OK)

printf("删除元素成功，删除的元素为 %d ！\n",e);

else if(j==ERROR)

printf("删除元素失败，删除位置不正确！\n");

else

printf("线性表不存在，不能进行删除操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 12:

j=ListTraverse(L);

if(j==OK)

printf("\n遍历线性表成功！\n");

else if(j==ERROR)

printf("\n线性表中无元素！\n");

else

printf("\n线性表不存在，不能进行遍历操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 13:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=SaveList(L,FileName);

if(j==OK)

printf("线性表存档成功！\n");

else

printf("线性表不存在，不能进行存档操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 14:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=LoadList(L,FileName);

if(j==OK)

printf("线性表读档成功！\n");

else

printf("线性表已存在，读档可能失去原始数据！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case -1:

break;

}//end of switch

}//end of while

}

# 附录C 基于二叉链表二叉树实现的源程序

/\* Linear Table On Sequence Structure \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define Maxlength 100

typedef int status;

typedef int KeyType;

typedef struct

{

KeyType key;//二叉树结点关键字定义

char others[20];//二叉树结点字符串

} TElemType; //二叉树结点类型定义

typedef struct BiTNode

{

TElemType data;

struct BiTNode \*lchild, \*rchild;//二叉树左右子树定义

} BiTNode, \*BiTree;//二叉链表结点的定义

typedef struct

{

int pos;

TElemType data;

} DEF;//用于存放用户输入的位置、关键字、字符串信息

typedef struct

{ //二叉树的管理表定义

struct

{

char name[30]; //二叉树名称

BiTree T;//二叉树根结点指针

} elem[10]; //存放二叉树的集合

int length; //二叉树集合的长度

int listsize; //二叉树集合分配的空间

} LISTS;

BiTNode \*LocateTree(BiTree T, BiTree t);

void visit(BiTree T);

void SecondaryTable(BiTree &T,int op);

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

status DestroyBiTree(BiTree &T);

status ClearBiTree(BiTree &T);

status BiTreeEmpty(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);

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

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

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

int main()

{

BiTree T = NULL, tmp = NULL;

LISTS Lists;

TElemType value;

DEF definition[Maxlength];

KeyType e;

int op = 1;

int i, j;

char Name[30] = {'\0'}, FileName[30] = {'\0'};

value.key=0;

value.others[0]='\0';

Lists.length = 0;

Lists.listsize = 10;

for (j = 0; j < 10; j++)

{

Lists.elem[j].T = NULL;

} //初始化

while (op != -3)

{

system("cls");

printf("\n\n");

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

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

printf(" -1.MainTable/MultiBiTree -2.SingleBiTree\n");

printf(" -3.Exit\n");

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

printf(" 请选择所在的菜单层级：\n");

printf(" -1：主菜单实现多二叉树管理，-2：临时单二叉树基本操作。\n");

scanf("%d", &op);

if (op == -1)

{

while (op)

{

system("cls");

printf("\n\n");

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

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

printf(" 17.AddBiTree 18.RemoveBiTree 19.LocateBiTree\n");

printf(" 20.SecondaryTable\n");

printf(" 0. Exit\n");

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

for (i = 0; i < 5; i++)

{

if (Lists.elem[i].T == NULL && Lists.elem[i + 5].T == NULL )

printf("\t二叉树%d：未创建 \t二叉树%d：未创建", i + 1, i + 6);

else if (Lists.elem[i].T == NULL && Lists.elem[i + 5].T != NULL)

printf("\t二叉树%d：未创建 \t二叉树%d：%s", i + 1, i + 6, Lists.elem[i + 5].name);

else if (Lists.elem[i].T != NULL && Lists.elem[i + 5].T == NULL)

printf("\t二叉树%d：%s \t二叉树%d：未创建", i + 1, Lists.elem[i].name, i + 6);

else

printf("\t二叉树%d：%s \t二叉树%d：%s", i + 1, Lists.elem[i].name, i + 6, Lists.elem[i + 5].name);

printf("\n");

}

printf("\n 请选择你的操作[0或17~20]:");

scanf("%d", &op);

switch (op)

{

case 17:

printf("请输入待添加的二叉树名称：");

scanf("%s",Name);

j=AddList(Lists,Name);

if(j==OK)

printf("二叉树添加成功！\n");

else if(j==ERROR)

printf("二叉树添加失败，该二叉树已存在！\n");

else if(j==OVERFLOW)

printf("二叉树添加失败，二叉树集合已满！\n");

else if(j==INFEASIBLE)

printf("创建二叉树失败，二叉树的结点关键字存在重复！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 18:

printf("请输入待删除的二叉树名称：");

scanf("%s",Name);

j=RemoveList(Lists,Name);

if(j==OK)

printf("二叉树删除成功！\n");

else if(j==ERROR)

printf("二叉树删除失败，集合中无此二叉树！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 19:

printf("请输入待查找的二叉树名称：");

scanf("%s",Name);

j=LocateList(Lists,Name);

if(j>0)

printf("该二叉树位于集合中第 %d 位！\n",j);

else

printf("二叉树查找失败，集合中无此二叉树！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 20:

printf("请输入待操作的二叉树名称：");

scanf("%s",Name);

j=LocateList(Lists,Name);

if(j>0)

SecondaryTable(Lists.elem[j-1].T,op);

else

{

getchar();

printf("二叉树不存在于集合中，不能进入单二叉树操作！\n");

printf("---任意键继续---\n");

getchar();

break;

}

break;

case 0:

break;

} //end of switch

} //end of while

} //end of if

if(op==-2)

{

while(op){

system("cls");

printf("\n\n");

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

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

printf(" 1. CreateBiTree 7. Assign 13. PostOrderTraverse\n");

printf(" 2. DestroyBiTree 8. GetSibling 14. LevelOrderTraverse\n");

printf(" 3. ClearBiTree 9. InsertNode 15. SaveBiTree\n");

printf(" 4. BiTreeEmpty 10. DeleteNode 16. LoadBiTree\n");

printf(" 5. BiTreeDepth 11. PreOrderTraverse\n");

printf(" 6. LocateNode 12. InOrderTraverse\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~16]:");

scanf("%d",&op);

switch(op){

case 1:

i=0;

printf("请依次输入二叉树结点位置序号、关键字、内容：\n");

do {

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

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

j=CreateBiTree(T,definition);

if(j==OK)

printf("创建二叉树成功！\n");

else if(j==ERROR)

printf("创建二叉树失败，二叉树的结点关键字存在重复！\n");

else if(j==INFEASIBLE)

printf("创建二叉树失败，二叉树已存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 2:

j=DestroyBiTree(T);

if(j==OK)

printf("销毁二叉树成功！\n");

else

printf("二叉树不存在，不能进行销毁操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 3:

j=ClearBiTree(T);

if(j==OK)

printf("清空二叉树成功！\n");

else

printf("二叉树不存在，不能进行清空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 4:

j=BiTreeEmpty(T);

if(j==TRUE)

printf("二叉树为空！\n");

else if(j==FALSE)

printf("二叉树不为空！\n");

else

printf("二叉树不存在，不能进行判空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 5:

j=BiTreeDepth(T);

if(j>=0)

printf("二叉树深度为 %d ！\n",j);

else

printf("二叉树不存在，不能进行求深度操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 6:

printf("请输入待查找结点关键字：");

scanf("%d",&e);

tmp=LocateNode(T,e);

if(tmp)

printf("获取结点成功，结点关键字和字符串为 %d %s ！\n",tmp->data.key,tmp->data.others);

else

printf("关键字为 %d 结点不存在，查找失败！\n",e);

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 7:

printf("请输入需要重新赋值的结点的关键字：");

scanf("%d",&e);

printf("请输入新赋值的结点关键字和字符串：");

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

j=Assign(T,e,value);

if(j==OK)

printf("赋值成功！\n");

else if(j==ERROR)

printf("赋值失败，可能是待重新赋值的关键字不存在，或赋值后二叉树中关键字重复！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 8:

printf("请输入需要查找其兄弟结点的结点的关键字：");

scanf("%d",&e);

tmp=GetSibling(T,e);

if(tmp)

printf("兄弟结点关键字和字符串为 %d %s ！\n",tmp->data.key,tmp->data.others);

else

printf("该结点的兄弟结点不存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 9:

printf("请输入待操作结点关键字：");

scanf("%d",&e);

printf("插入选择：\n0：将待插入结点作为对应结点左子树；1：将待插入结点作为对应结点右子树；-1：作为根结点插入\n");

printf("请输入插入方式选择：");

scanf("%d",&i);

printf("请输入待插入结点关键字和字符串：");

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

j=InsertNode(T,e,i,value);

if(j==OK)

printf("插入结点成功！\n");

else if(j==INFEASIBLE)

printf("插入结点失败，该二叉树中无关键字 %d ！\n",e);

else if(j==ERROR)

printf("插入结点失败，待插入结点关键字与原二叉树中关键字重复！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 10:

printf("请输入待删除结点关键字：");

scanf("%d",&e);

j=DeleteNode(T,e);

if(j==OK)

printf("删除结点成功！\n");

else

printf("删除结点失败！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 11:

j=PreOrderTraverse(T,visit);

if(j==OK)

printf("\n先序遍历二叉树成功！\n");

else

printf("\n二叉树不存在，不能进行遍历！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 12:

j=InOrderTraverse(T,visit);

if(j==OK)

printf("\n中序遍历二叉树成功！\n");

else

printf("\n二叉树不存在，不能进行遍历！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 13:

j=PostOrderTraverse(T,visit);

if(j==OK)

printf("\n后序遍历二叉树成功！\n");

else

printf("\n二叉树不存在，不能进行遍历！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 14:

j=LevelOrderTraverse(T,visit);

if(j==OK)

printf("\n按层遍历二叉树成功！\n");

else

printf("\n二叉树不存在，不能进行遍历！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 15:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=SaveBiTree(T,FileName);

if(j==OK)

printf("二叉树存档成功！\n");

else

printf("二叉树存档失败，二叉树不存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 16:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=LoadBiTree(T,FileName);

if(j==OK)

printf("二叉树读档成功！\n");

else

printf("二叉树读档失败，二叉树已存在，读档会使原二叉树中元素失去！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 0:

break;

}//end of switch

}//end of while

}//end of if

} //end of while

printf("欢迎下次再使用本系统！\n");

} //end of main

//菜单功能选择 1 创建二叉树

status CreateBiTree(BiTree &T, DEF 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。

{

if(!T)

{

int i=0,j;

BiTNode \*p[Maxlength]={0};

int v[Maxlength]={0};

while (definition[i].data.key!=0)

{//标记二叉树结点是否出现过，有相同关键字时说明数据输入错误

if(!v[definition[i].data.key])

v[definition[i].data.key]=1;

else

return ERROR;

i++;

}

i=0;

while (j=definition[i].pos)

{//根据二叉树结点位置输入及其关系创建二叉树

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

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

p[j]->lchild=NULL;

p[j]->rchild=NULL;

if (j!=1) //左子树位置是父亲结点位置的2倍，右子树是父亲结点位置的2倍+1

if (j%2)

p[j/2]->rchild=p[j];

else

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

i++;

}

T=p[1]; //根结点赋给T

return OK;

}

else

return INFEASIBLE; //二叉树已存在，不能创建

}

//菜单功能选择 2 销毁二叉树

status DestroyBiTree(BiTree &T)

//将二叉树设置成空，并删除所有结点，释放结点空间,并释放头结点。

{

if(!T) //二叉树不存在

return INFEASIBLE;

else

{//递归调用销毁二叉树函数

if(T->lchild)

DestroyBiTree(T->lchild);

if(T->rchild)

DestroyBiTree(T->rchild);

free(T);

T = NULL;

return OK;

}

}

//菜单功能选择 3 清空二叉树

status ClearBiTree(BiTree &T)

//将二叉树设置成空，并删除所有结点，释放结点空间。

{

if (!T) //二叉树不存在

return INFEASIBLE;

else

{//递归调用清空二叉树函数

if(T->lchild)

ClearBiTree(T->lchild);

if(T->rchild)

ClearBiTree(T->rchild);

free(T);

T = NULL;

return OK;

}

}

//菜单功能选择 4 二叉树判空

status BiTreeEmpty(BiTree T)

//判断二叉树是否为空树，是则返回TRUE，否则返回FALSE。

{

if (!T)

return TRUE;

else

return FALSE;

}

//菜单功能选择 5 求二叉树深度

int BiTreeDepth(BiTree T)

//递归求二叉树T深度并返回深度值。

{

if (!T) //二叉树不存在

return 0;

int lh = BiTreeDepth(T->lchild); //递归调用

int rh = BiTreeDepth(T->rchild);

return 1 + (lh > rh ? lh : rh);

}

//菜单功能选择 6 查找结点

BiTNode \*LocateNode(BiTree T, KeyType e)

//e是和T中结点关键字类型相同的给定值；根据e查找符合条件的结点，返回该结点指针，如无关键字为e的结点，返回NULL。

{

if (!T) //二叉树根结点不存在

return NULL;

else

{

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

return T; //搜索成功，返回当前结点

BiTree p = NULL;

if (T->lchild)

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

if (T->rchild && !p) //注意!p条件的判断，即左子树搜索失败才搜索右子树

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

return p;

}

}

//菜单功能选择 7 结点赋值

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

//e是和T中结点关键字类型相同的给定值；查找结点关键字等于e的结点，将结点值修改成value，返回OK（要求结点关键字保持唯一性）。如果查找失败，返回ERROR。

{

int v[Maxlength] = {0};

if (!T) //二叉树根结点不存在

return ERROR;

else

{

BiTree data[Maxlength], p;

int top = 0;

if (T)

{//先序遍历，对二叉树结点关键字进行标记，以判断赋值时是否出现关键字重复

data[top++] = T;

while (top)

{

p = data[--top];

if(!v[p->data.key])

v[p->data.key]=1;

if (p->rchild)

data[top++] = p->rchild;

if (p->lchild)

data[top++] = p->lchild;

}

}

if (T)

{//先序遍历，对二叉树结点进行搜索以赋值

data[top++] = T;

while (top)

{

p = data[--top];

if ((p->data.key == e && !v[value.key]) || (p->data.key == e && p->data.key == value.key))

{//查找到待重新赋值结点且新关键字未被标记或与原关键字相同，执行赋值操作

p->data.key = value.key;

strcpy(p->data.others, value.others);

return OK;

}

else if (v[value.key] && p->data.key == e && p->data.key != value.key)//查找到待重新赋值结点但新关键字与二叉树中其他关键字重复

return ERROR;

if (p->rchild)

data[top++] = p->rchild;

if (p->lchild)

data[top++] = p->lchild;

}

}

}

}

//菜单功能选择 8 获取兄弟结点

BiTNode \*GetSibling(BiTree T, KeyType e)

//e是和T中结点关键字类型相同的给定值；查找结点关键字等于e的结点的兄弟结点，返回其兄弟结点指针。如果查找失败，返回NULL。

{

if (T->lchild != NULL && T->rchild != NULL)

{//左右子树皆不为空，当左子树为待查找结点时，返回右子树；当右子树为待查找结点时，返回左子树；

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

return T->rchild;

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

return T->lchild;

}

BiTree p = NULL;

if (T->lchild != NULL) //递归查找

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

if (T->rchild != NULL && !p)

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

return p;

}

//菜单功能选择 9 插入结点

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

/\*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；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，返回OK。如果插入失败，返回ERROR。

特别地，当LR为-1时，作为根结点插入，原根结点作为c的右子树。\*/

{

BiTree p, t;

BiTree data[Maxlength];

int top = 0;

p = LocateNode(T, c.key);

if (p) //调用查找函数判断待插入结点关键字是否与已有关键字重复，重复则报错

return INFEASIBLE;

p = LocateNode(T, e); //若关键字不重复，调用查找函数找到待操作结点

if (p) //待操作结点查找成功

{

if (!LR)

{//左插

t = p->lchild;

p->lchild = (BiTree)malloc(sizeof(BiTNode));

p->lchild->data.key = c.key;

strcpy(p->lchild->data.others, c.others);

p->lchild->lchild = NULL;

p->lchild->rchild = t;

}

else if (LR == 1)

{//右插

t = p->rchild;

p->rchild = (BiTree)malloc(sizeof(BiTNode));

p->rchild->data.key = c.key;

strcpy(p->rchild->data.others, c.others);

p->rchild->lchild = NULL;

p->rchild->rchild = t;

}

else if (LR == -1)

{//头插

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

t->data.key = c.key;

strcpy(t->data.others, c.others);

t->lchild = NULL;

t->rchild = T;

T = t;

}

return OK;

}

else

return INFEASIBLE;

}

//菜单功能选择 10 删除结点

status DeleteNode(BiTree &T, KeyType e)

/\*e是和T中结点关键字类型相同的给定值。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。

成功删除结点后返回OK，否则返回ERROR。\*/

{

BiTree p, pre, tmp;

p = LocateNode(T, e);

if (!p)

return ERROR;

else

{

if (p == T)

pre = NULL;

else

pre = LocateTree(T, p);

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

{

if (!pre)

;

else if (pre->lchild == p)

pre->lchild = NULL;

else if (pre->rchild == p)

pre->rchild = NULL;

}

else if (p->lchild && p->rchild)

{

tmp = p->lchild;

while (tmp->rchild)

{

tmp = tmp->rchild;

}

tmp->rchild = (BiTree)malloc(sizeof(BiTNode));

tmp->rchild = p->rchild;

if (!pre)

{

pre = p->lchild;

T = pre;

}

else if (pre->lchild == p)

pre->lchild = p->lchild;

else if (pre->rchild == p)

pre->rchild = p->lchild;

}

else if (p->lchild)

{

if (!pre)

{

pre = p->lchild;

T = pre;

}

else if (pre->lchild == p)

pre->lchild = p->lchild;

else if (pre->rchild == p)

pre->rchild = p->lchild;

}

else if (p->rchild)

{

if (!pre)

{

pre = p->rchild;

T = pre;

}

else if (pre->lchild == p)

pre->lchild = p->rchild;

else if (pre->rchild == p)

pre->rchild = p->rchild;

}

free(p);

return OK;

}

}

//菜单功能选择 11 先序遍历二叉树

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

//先序遍历二叉树T，对每个结点访问一次且仅一次。

{

BiTree data[Maxlength], p;

int top = 0;

if (T)

{//利用栈对二叉树进行非递归先序遍历

data[top++] = T;

while (top)

{

p = data[--top];

visit(p);

if (p->rchild)

data[top++] = p->rchild;

if (p->lchild)

data[top++] = p->lchild;

}

return OK;

}

else

return INFEASIBLE;

}

//菜单功能选择 12 中序遍历二叉树

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

//中序遍历二叉树T，对每个结点访问一次且仅一次。

{

BiTree data[Maxlength], p;

int top = 0;

if (T)

{//利用栈对二叉树进行非递归中序遍历

p = T;

do

{

while (p)

{

if (top == Maxlength)

exit(OVERFLOW);

data[top++] = p;

p = p->lchild;

}

if (top)

{

p = data[--top];

visit(p);

p = p->rchild;

}

} while (top || p);

return OK;

}

else

return INFEASIBLE;

}

//菜单功能选择 13 后序遍历二叉树

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

//后序遍历二叉树T，对每个结点访问一次且仅一次。

{

BiTree data[Maxlength], p, pre;

int top = 0, flag;

if (T)

{//利用栈对二叉树进行非递归后序遍历

p = T;

do

{

while (p)

{

if (top == Maxlength)

exit(OVERFLOW);

data[top++] = p;

p = p->lchild;

}

pre = NULL;

flag = 1;

while (top && flag)

{

p = data[top - 1];

if (p->rchild == pre)

{

visit(p);

top--;

pre = p;

}

else

{

p = p->rchild;

flag = 0;

}

}

} while (top);

return OK;

}

else

return INFEASIBLE;

}

//菜单功能选择 14 层序遍历二叉树

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

//按层遍历二叉树T，对每个结点访问一次且仅一次。

{

BiTree data[Maxlength];

int in = 0, out = 0;

data[in++] = T;

while (in > out)

{//利用栈对二叉树进行非递归层序遍历

if (data[out])

{

visit(data[out]);

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

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

}

out++;

}

return OK;

}

//菜单功能选择 15 二叉树存档

status SaveBiTree(BiTree T, char FileName[])

//将二叉树的结点数据写入到文件FileName中。

{

if(T)

{//文件存取时利用二叉树位置，便于标记和创建

FILE \*fp;

BiTree node[Maxlength + 1] = {0};

node[1] = T;

int i = 1;

int depth = BiTreeDepth(T);

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

{

while (i < pow(2,depth))

{

if (node[i] != NULL)

{

fprintf(fp, "%d %d %s ", i, node[i]->data.key, node[i]->data.others);

if (node[i]->lchild != NULL)

{

node[2 \* i] = node[i]->lchild;

}

if (node[i]->rchild != NULL)

{

node[2 \* i + 1] = node[i]->rchild;

}

}

i++;

}

fclose(fp);

return OK;

}

else

return ERROR;

}

else //二叉树不存在

return INFEASIBLE;

}

//菜单功能选择 16 二叉树读档

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

//读入文件FileName的结点数据，创建二叉树。

{

if(!T)

{

BiTree node[Maxlength];

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

node[i] = NULL;

FILE \*fp;

fp = fopen(FileName, "rb");

int a, b;

char string[20];

while (fscanf(fp, "%d %d %s ", &a, &b, string) != EOF)

{

node[a] = (BiTree)malloc(sizeof(BiTNode));

node[a]->data.key = b;

strcpy(node[a]->data.others, string);

node[a]->rchild = node[a]->lchild = NULL;

}

for (int i = 2; i < Maxlength; ++i)

{

if (node[i / 2] != NULL)

{

if (i % 2 == 1)

node[i / 2]->rchild = node[i];

else

node[i / 2]->lchild = node[i];

}

}

T = node[1];

fclose(fp);

return OK;

}

else //二叉树已存在，读档会使二叉树中数据失去

return INFEASIBLE;

}

//菜单功能选择 17 多二叉树管理之增加二叉树

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

// 在Lists中增加一个名称为ListName的二叉树。

{

if (Lists.length < Lists.listsize)

{

int i, j, flag = 0;

DEF definition[Maxlength];

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

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

{//利用strcmp函数比较二叉树集合中树与当前输入的名称，若相同则标记

flag = 1;

break;

}

if (!flag) //为被标记，当前树不存在于二叉树集合中，可以增加

{//由于树的根结点为空时相当于树不存在，增加树时即向树中输入元素，利用位置标记构造二叉树

i=0;

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

printf("请依次输入二叉树结点位置序号、关键字、内容：\n");

do {

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

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

j=CreateBiTree(Lists.elem[Lists.length].T,definition); //调用创建二叉树函数

if(j==OK)

{

Lists.length++;

return OK;

}

else //创建二叉树失败

return INFEASIBLE;

}

else //被标记，当前树已存在，不能增加

return ERROR;

}

else //二叉树集合已满，不能添加二叉树

return OVERFLOW;

}

//菜单功能选择 18 多二叉树管理之删除二叉树

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

// 在Lists中删除一个名称为ListName的二叉树。

{

int i, flag = 0;

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

{

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

{//对二叉树名称进行比较查找，查找成功时标记并退出循环

flag = 1;

break;

}

}

if (flag)

{

DestroyBiTree(Lists.elem[i].T); //调用销毁二叉树函数销毁当前二叉树

for (; i < Lists.length - 1; i++) //将查找到的二叉树后的二叉树顺序前移一位

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

if(i!=Lists.length-1) //若最末位二叉树未被销毁，调用销毁二叉树函数销毁

DestroyBiTree(Lists.elem[Lists.length-1].T);

Lists.elem[Lists.length - 1].name[0] = '\0'; //将原本最末位的二叉树名称置为空，释放其根结点

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

free(Lists.elem[Lists.length - 1].T);

Lists.length--;

return OK;

}

else

return ERROR;

}

//菜单功能选择 19 多二叉树管理之查找二叉树

int LocateList(LISTS Lists, char ListName[])

// 在Lists中查找一个名称为ListName的二叉树，查找成功返回逻辑序号，否则返回ERROR。

{

int i, flag = 0;

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

{

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

{//对二叉树名称进行比较查找，查找成功时标记并退出循环

flag = 1;

break;

}

}

if (!flag || !Lists.length) //查找失败或二叉树集合长度为0

return ERROR;

else //查找成功返回逻辑位置

return i + 1;

}

//辅助函数：查找父亲结点

BiTNode \*LocateTree(BiTree T, BiTree t)

//根据子树t查找其父亲结点，查找成功则返回父亲结点指针，否则返回NULL。

{

if (!T)

return NULL;

else

{

if (T->lchild == t || T->rchild == t)

return T;

BiTree p = NULL;

if (T->lchild)

p = LocateTree(T->lchild, t);

if (T->rchild && !p)

p = LocateTree(T->rchild, t);

return p;

}

}

//辅助函数：访问结点

void visit(BiTree T)

//对当前结点执行特定操作。

{

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

}

//辅助函数：进入二级菜单

void SecondaryTable(BiTree &T,int op)

// 将所选的多二叉树集合中的待操作二叉树传递进入二级菜单中，实现对单二叉树的基本操作

{

BiTree tmp;

TElemType value;

DEF definition[Maxlength];

KeyType e;

int i, j;

char FileName[30]={'\0'};

while(op){

system("cls");

printf("\n\n");

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

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

printf(" 1. CreateBiTree 7. Assign 13. PostOrderTraverse\n");

printf(" 2. DestroyBiTree 8. GetSibling 14. LevelOrderTraverse\n");

printf(" 3. ClearBiTree 9. InsertNode 15. SaveBiTree\n");

printf(" 4. BiTreeEmpty 10. DeleteNode 16. LoadBiTree\n");

printf(" 5. BiTreeDepth 11. PreOrderTraverse\n");

printf(" 6. LocateNode 12. InOrderTraverse\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~16]:");

scanf("%d",&op);

switch(op){

case 1:

i=0;

printf("请依次输入二叉树结点位置序号、关键字、内容：\n");

do {

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

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

j=CreateBiTree(T,definition);

if(j==OK)

printf("创建二叉树成功！\n");

else if(j==ERROR)

printf("创建二叉树失败，二叉树的结点关键字存在重复！\n");

else if(j==INFEASIBLE)

printf("创建二叉树失败，二叉树已存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 2:

j=DestroyBiTree(T);

if(j==OK)

printf("销毁二叉树成功！\n");

else

printf("二叉树不存在，不能进行销毁操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 3:

j=ClearBiTree(T);

if(j==OK)

printf("清空二叉树成功！\n");

else

printf("二叉树不存在，不能进行清空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 4:

j=BiTreeEmpty(T);

if(j==TRUE)

printf("二叉树为空！\n");

else if(j==FALSE)

printf("二叉树不为空！\n");

else

printf("二叉树不存在，不能进行判空操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 5:

j=BiTreeDepth(T);

if(j>=0)

printf("二叉树深度为 %d ！\n",j);

else

printf("二叉树不存在，不能进行求深度操作！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 6:

printf("请输入待查找结点关键字：");

scanf("%d",&e);

tmp=LocateNode(T,e);

if(tmp)

printf("获取结点成功，结点关键字和字符串为 %d %s ！\n",tmp->data.key,tmp->data.others);

else

printf("关键字为 %d 结点不存在，查找失败！\n",e);

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 7:

printf("请输入需要重新赋值的结点的关键字：");

scanf("%d",&e);

printf("请输入新赋值的结点关键字和字符串：");

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

j=Assign(T,e,value);

if(j==OK)

printf("赋值成功！\n");

else if(j==ERROR)

printf("赋值失败，可能是待重新赋值的关键字不存在，或赋值后二叉树中关键字重复！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 8:

printf("请输入需要查找其兄弟结点的结点的关键字：");

scanf("%d",&e);

tmp=GetSibling(T,e);

if(tmp)

printf("兄弟结点关键字和字符串为 %d %s ！\n",tmp->data.key,tmp->data.others);

else

printf("该结点的兄弟结点不存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 9:

printf("请输入待操作结点关键字：");

scanf("%d",&e);

printf("插入选择：\n0：将待插入结点作为对应结点左子树；1：将待插入结点作为对应结点右子树；-1：作为根结点插入\n");

printf("请输入插入方式选择：");

scanf("%d",&i);

printf("请输入待插入结点关键字和字符串：");

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

j=InsertNode(T,e,i,value);

if(j==OK)

printf("插入结点成功！\n");

else

printf("插入结点失败！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 10:

printf("请输入待删除结点关键字：");

scanf("%d",&e);

j=DeleteNode(T,e);

if(j==OK)

printf("删除结点成功！\n");

else

printf("删除结点失败！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 11:

j=PreOrderTraverse(T,visit);

if(j==OK)

printf("\n先序遍历二叉树成功！\n");

else

printf("\n二叉树不存在，不能进行遍历！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 12:

j=InOrderTraverse(T,visit);

if(j==OK)

printf("\n中序遍历二叉树成功！\n");

else

printf("\n二叉树不存在，不能进行遍历！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 13:

j=PostOrderTraverse(T,visit);

if(j==OK)

printf("\n后序遍历二叉树成功！\n");

else

printf("\n二叉树不存在，不能进行遍历！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 14:

j=LevelOrderTraverse(T,visit);

if(j==OK)

printf("\n按层遍历二叉树成功！\n");

else

printf("\n二叉树不存在，不能进行遍历！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 15:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=SaveBiTree(T,FileName);

if(j==OK)

printf("二叉树存档成功！\n");

else

printf("二叉树存档失败，二叉树不存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 16:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=LoadBiTree(T,FileName);

if(j==OK)

printf("二叉树读档成功！\n");

else

printf("二叉树读档失败，二叉树已存在，读档会使原二叉树中元素失去！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 0:

break;

}//end of switch

}//end of while

}

# 附录D 基于邻接表图实现的源程序

/\* Linear Table On Sequence Structure \*/

#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

#define MAX\_VERTEX\_NUM 20

#define MAX\_KEY 100

typedef int status;

typedef int KeyType;

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

typedef struct {

KeyType key;

char others[20];

} VertexType; //顶点类型定义

typedef struct ArcNode { //表结点类型定义

int adjvex; //顶点位置编号

struct ArcNode \*nextarc; //下一个表结点指针

} ArcNode;

typedef struct VNode{ //头结点及其数组类型定义

VertexType data; //顶点信息

ArcNode \*firstarc; //指向第一条弧

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

typedef struct { //邻接表的类型定义

AdjList vertices; //头结点数组

int vexnum,arcnum; //顶点数、弧数

GraphKind kind; //图的类型

} ALGraph;

typedef struct{ //无向图的管理表定义

struct { char name[30]; //无向图名称

ALGraph G;

} elem[10]; //存放无向图的集合

int length; //无向图集合的长度

int listsize; //无向图集合分配的空间

}LISTS;

typedef int ElemType; //栈和队列的定义

typedef struct stack

{

ElemType elem[MAX\_VERTEX\_NUM];

int p;

} stack; //栈的定义

typedef struct QUEUE

{

ElemType elem[MAX\_VERTEX\_NUM];

int front, length;

} QUEUE; //循环队列

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

status DestroyGraph(ALGraph &G);

int LocateVex(ALGraph G,KeyType u);

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

int FirstAdjVex(ALGraph G,KeyType u);

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

status InsertVex(ALGraph &G,VertexType v);

status DeleteVex(ALGraph &G,KeyType v);

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

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

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

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

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

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

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

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

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

void SecondaryTable(ALGraph &G,int op);

void DFS(AdjList verx,int index,int visited[100],void (\*visit)(VertexType));

void visit(VertexType v);

void iniStack(stack &S);

int isEmptyStack(stack &S);

int push(stack &S, ElemType e);

ElemType pop(stack &S);

void iniQueue(QUEUE &Q);

int enQueue(QUEUE &Q, ElemType e);

int deQueue(QUEUE &Q, ElemType &e);

int isEmptyQueue(QUEUE &Q);

int main(){

int i,j,op=1;

char Name[30]={'\0'},FileName[30]={'\0'};

KeyType u,w;

ALGraph G;

VertexType value;

LISTS Lists;

G.vexnum=G.arcnum=0;

G.kind=UDG;

value.key=0;

value.others[0]='\0';

Lists.length=0;

Lists.listsize=10;

for(j=0;j<10;j++)

{

Lists.elem[j].G.arcnum=Lists.elem[j].G.vexnum=0;

Lists.elem[j].G.kind=UDG;

}//初始化

while(op!=-3){

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

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

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

printf(" -1.MainTable/MultiGraph -2.SingleGraph\n");

printf(" -3.Exit\n");

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

printf(" 请选择所在的菜单层级：\n");

printf(" -1：主菜单实现多无向图管理，-2：临时单无向图基本操作。\n");

scanf("%d",&op);

if(op==-1)

{

while(op)

{

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

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

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

printf(" 15.AddGraph 16.RemoveGraph 17.LocateGraph\n");

printf(" 18.SecondaryTable\n");

printf(" 0. Exit\n");

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

for(i=0;i<5;i++)

{

if(Lists.elem[i].G.vexnum==0&&Lists.elem[i+5].G.vexnum==0)

printf("\t无向图%d：未创建 \t无向图%d：未创建",i+1,i+6);

else if(Lists.elem[i].G.vexnum==0&&Lists.elem[i+5].G.vexnum!=0)

printf("\t无向图%d：未创建 \t无向图%d：%s",i+1,i+6,Lists.elem[i+5].name);

else if(Lists.elem[i].G.vexnum!=0&&Lists.elem[i+5].G.vexnum==0)

printf("\t无向图%d：%s \t无向图%d：未创建",i+1,Lists.elem[i].name,i+6);

else

printf("\t无向图%d：%s \t无向图%d：%s",i+1,Lists.elem[i].name,i+6,Lists.elem[i+5].name);

printf("\n");

}

printf(" 请选择你的操作[0或15~18]:");

scanf("%d",&op);

switch(op){

case 15:

printf("请输入待添加的无向图名称：");

scanf("%s",Name);

j=AddGraph(Lists,Name);

if(j==OK)

printf("无向图添加成功！\n");

else if(j==INFEASIBLE)

printf("无向图添加失败，该无向图已存在！\n");

else if(j==OVERFLOW)

printf("无向图添加失败，无向图集合已满！\n");

else if(j==ERROR)

printf("创建无向图失败，无向图为空或数据输入错误！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 16:

printf("请输入待删除的无向图名称：");

scanf("%s",Name);

j=RemoveGraph(Lists,Name);

if(j==OK)

printf("无向图删除成功！\n");

else if(j==ERROR)

printf("无向图删除失败，集合中无此无向图！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 17:

printf("请输入待查找的无向图名称：");

scanf("%s",Name);

j=LocateGraph(Lists,Name);

if(j>0)

printf("该无向图位于集合中第 %d 位！\n",j);

else

printf("无向图查找失败，集合中无此无向图！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 18:

printf("请输入待操作的无向图名称：");

scanf("%s",Name);

j=LocateGraph(Lists,Name);

if(j>0)

SecondaryTable(Lists.elem[j-1].G,op);

else

{

getchar();

printf("无向图不存在于集合中，不能进入单无向图操作！\n");

printf("---任意键继续---\n");

getchar();

break;

}

break;

case 0:

break;

}//end of switch

}//end of while

}//end of if

if(op==-2)

{

while(op){

system("cls");

printf("\n\n");

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

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

printf(" 1. CreateGraph 7. InsertVex 13. SaveGraph\n");

printf(" 2. DestroyGraph 8. DeleteVex 14. LoadGraph\n");

printf(" 3. LocateVex 9. InsertArc 15. ShowGraph\n");

printf(" 4. PutVex 10. DeleteArc\n");

printf(" 5. FirstAdjVex 11. DFSTraverse\n");

printf(" 6. NextAdjVex 12. BFSTraverse\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~15]:");

scanf("%d",&op);

switch(op){

case 1:

i=0;

VertexType V[30];

KeyType VR[100][2];

printf("请输入图中所有顶点关键字和字符串，输入（-1，null）终止：\n");

do {

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

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

i=0;

printf("请输入图中所有弧，输入（-1，-1）终止：\n");

do {

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

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

j=CreateGraph(G,V,VR);

if(j==OK)

printf("创建无向图成功！\n");

else if(j==INFEASIBLE)

printf("无向图添加失败，该无向图已存在！\n");

else if(j==ERROR)

printf("创建无向图失败，无向图数据输入错误！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 2:

j=DestroyGraph(G);

if(j==OK)

printf("销毁无向图成功！\n");

else

printf("销毁无向图失败，无向图不存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 3:

printf("请输入待查找顶点关键字：");

scanf("%d",&u);

j=LocateVex(G,u);

if(j==INFEASIBLE)

printf("关键字为 %d 顶点不存在，查找失败！\n",u);

else

printf("查找顶点成功，关键字为 %d 的顶点在邻接表数组中关键字和字符串为 %d %s ！\n",u,G.vertices[j].data.key,G.vertices[j].data.others);

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 4:

printf("请输入需要重新赋值的顶点的关键字：");

scanf("%d",&u);

printf("请输入新赋值的顶点关键字和字符串：");

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

j=PutVex(G,u,value);

if(j==OK)

printf("赋值成功！\n");

else if(j==ERROR)

printf("赋值失败，可能是待重新赋值的关键字不存在，或赋值后无向图中关键字重复！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 5:

printf("请输入需要获取第一邻接点的顶点的关键字：");

scanf("%d",&u);

j=FirstAdjVex(G,u);

if(j==OVERFLOW)

printf("获取失败，该顶点无邻接顶点！\n");

else if(j==INFEASIBLE)

printf("获取失败，无向图中无此顶点！\n");

else

printf("获取成功，关键字为 %d 的顶点的第一邻接顶点关键字和字符串为 %d %s ！\n",u,G.vertices[j].data.key,G.vertices[j].data.others);

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 6:

printf("请输入需要获取邻接点的顶点的关键字：");

scanf("%d",&u);

printf("请输入需要获取下一邻接点的顶点的关键字：");

scanf("%d",&w);

j=NextAdjVex(G,u,w);

if(j==ERROR)

printf("获取失败，w无下一邻接顶点！\n");

else if(j==INFEASIBLE)

printf("获取失败，v不在无向图中！\n");

else

printf("获取成功，顶点v的邻接顶点相对于w的下一邻接顶点的关键字和字符串为 %d %s ！\n",G.vertices[j].data.key,G.vertices[j].data.others);

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 7:

printf("请输入插入顶点的关键字和字符串：");

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

j=InsertVex(G,value);

if(j==OK)

printf("插入顶点成功！\n");

else if(j==OVERFLOW)

printf("插入顶点失败，顶点集已满！\n");

else if(j==ERROR)

printf("插入顶点失败，新插入顶点的关键字与已有顶点关键字重复！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 8:

printf("请输入删除顶点的关键字：");

scanf("%d",&u);

j=DeleteVex(G,u);

if(j==OK)

printf("删除顶点成功！\n");

else

printf("删除顶点失败，无向图为空或无此顶点！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 9:

printf("请输入待插入弧的两个顶点：");

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

j=InsertArc(G,u,w);

if(j==OK)

printf("插入弧成功！\n");

else if(j==INFEASIBLE)

printf("插入弧失败，输入的弧顶点不存在于图中！\n");

else if(j==ERROR)

printf("插入弧失败，插入的弧已存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 10:

printf("请输入待删除弧的两个顶点：");

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

j=DeleteArc(G,u,w);

if(j==OK)

printf("删除弧成功！\n");

else if(j==INFEASIBLE)

printf("删除弧失败，输入的弧顶点不存在于图中！\n");

else if(j==ERROR)

printf("删除弧失败，待删除的弧不存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 11:

j=DFSTraverse(G,visit);

if(j==OK)

printf("\n深度搜索无向图成功！\n");

else

printf("\n搜索图失败，图为空！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 12:

j=BFSTraverse(G,visit);

if(j==OK)

printf("\n广度搜索无向图成功！\n");

else

printf("\n搜索图失败，图为空！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 13:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=SaveGraph(G,FileName);

if(j==OK)

printf("无向图存档成功！\n");

else if(j==ERROR)

printf("创建/打开文件失败！\n");

else

printf("无向图存档失败，无向图为空！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 14:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=LoadGraph(G,FileName);

if(j==OK)

printf("无向图读档成功！\n");

else if(j==ERROR)

printf("打开文件失败！\n");

else

printf("无向图读档失败，无向图已存在，读档会使原无向图中数据失去！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 15:

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

{

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

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

while (p)

{

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

p=p->nextarc;

}

printf("\n");

}

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 0:

break;

}//end of switch

}//end of while

}//end of if

}//end of while

printf("欢迎下次再使用本系统！\n");

}

//栈的函数

void iniStack(stack &S)

//该函数初始化栈S

{

S.p = 0;

}

int isEmptyStack(stack &S)

//判断是不是空栈，是则返回1，不是则返回0

{

if (S.p)

return 0;

else

return 1;

}

int push(stack &S, ElemType e)

//该函数将元素进栈，成功则返回1，失败返回0

{

if (S.p == MAX\_VERTEX\_NUM - 1)

return OVERFLOW;

else

{

S.elem[++S.p] = e;

}

return OK;

}

ElemType pop(stack &S)

//该函数将元素出栈，返回出栈的元素值

{

if (S.p == 0)

return ERROR;

else

return S.elem[S.p--];

}

//队列的函数

void iniQueue(QUEUE &Q)

//该函数实现初始化Q

{

Q.front = 0;

Q.length = 0;

}

int isEmptyQueue(QUEUE &Q)

//判断队列是否为空，若为空返回1，否则返回0

{

if (Q.length)

return 0;

else

return 1;

}

int enQueue(QUEUE &Q, ElemType e)

//将元素e入队Q。成功入栈返回1，否则返回0

{

if (Q.length == MAX\_VERTEX\_NUM)

return 0;

Q.elem[(Q.front + Q.length++) % MAX\_VERTEX\_NUM] = e;

return 1;

}

int deQueue(QUEUE &Q, ElemType &e)

//将Q队首元素出队，赋值给e。成功出队返回1，否则返回0

{

if (Q.length == 0)

return 0;

e = Q.elem[(Q.front++) % MAX\_VERTEX\_NUM];

Q.length--;

return 1;

}

//菜单功能选择 1 创建无向图

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。\*/

{

int i,j,flag[MAX\_KEY]={0};

ArcNode \*tmp;

if(G.vexnum)

return INFEASIBLE;

G.vexnum=G.arcnum=0;

G.kind=UDG; //初始化无向图

if(V[0].key==-1)

return ERROR; //无向图为空

for(i=0;V[i].key!=-1;i++)

{ //用flag数组标记存在的顶点

if(flag[V[i].key])

{

G.vexnum=G.arcnum=0;

return ERROR; //重复输入结点关键字

}

flag[V[i].key]=1;

G.vexnum++;

if(G.vexnum>20)

{//顶点数超出最大值

G.vexnum=0;

return ERROR;

}

}

for(i=0;V[i].key!=-1;i++)

{ //为顶点赋值

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

strcpy(G.vertices[i].data.others,V[i].others);

G.vertices[i].firstarc=NULL;

}

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

{ //创建邻接表

if(!flag[VR[i][0]]||!flag[VR[i][1]])

{ //输入的边的顶点不存在

G.vexnum=G.arcnum=0;

return ERROR;

}

if(VR[i][0]==VR[i][1])

continue; //输入的边为自环

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

{

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

ArcNode \*t=G.vertices[j].firstarc;

if(G.vertices[j].data.key==VR[i][0]) //输入边的顶点顺序可能有两种情况，但最终创建邻接表的结果是一样的

{

for(;t;t=t->nextarc) //标记已经创建的边

f[t->adjvex]=1;

int k=0;

while (G.vertices[k].data.key!=VR[i][1]) //找到另一顶点的序号

k++;

if(!f[k])

{//此边未创建时用首插法创建边

tmp=G.vertices[j].firstarc;

G.vertices[j].firstarc=(ArcNode \*)malloc(sizeof(ArcNode));

G.vertices[j].firstarc->adjvex=k;

G.vertices[j].firstarc->nextarc=tmp;

G.arcnum++; //边数只需要在顶点顺序的某一种情况下增加

}

}

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

{

for(;t;t=t->nextarc)

f[t->adjvex]=1;

int k=0;

while (G.vertices[k].data.key!=VR[i][0])

k++;

if(!f[VR[i][0]])

{

tmp=G.vertices[j].firstarc;

G.vertices[j].firstarc=(ArcNode \*)malloc(sizeof(ArcNode));

G.vertices[j].firstarc->adjvex=k;

G.vertices[j].firstarc->nextarc=tmp;

}

}

}

}

return OK;

}

//菜单功能选择 2 销毁无向图

status DestroyGraph(ALGraph &G)

/\*销毁无向图G,删除G的全部顶点和边\*/

{

if(!G.vexnum)

return INFEASIBLE;

ArcNode \*tmp1,\*tmp2=NULL;

int i;

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

{

tmp1=G.vertices[i].firstarc;

if(tmp1) //对任一顶点，若没有边与它相连，则不需要进行删除边操作

tmp2=G.vertices[i].firstarc->nextarc;

for(;tmp1;)

{//依次删除头顶点后的边

//通过两个顶点指针的移动实现循环删除边顶点

G.vertices[i].firstarc=tmp2;

free(tmp1);

tmp1=NULL;

tmp1=tmp2;

if(tmp2)

tmp2=tmp2->nextarc;

}

}

G.vexnum=G.arcnum=0; //将顶点数和边数置为0

return OK;

}

//菜单功能选择 3 查找顶点

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INFEASIBLE

{

int i,flag=0;

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

{//遍历顶点数组，至查找到待查找顶点时标记并退出循环

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

{

flag=1;

break;

}

}

if(flag)

return i;

else

return INFEASIBLE;

}

//菜单功能选择 4 顶点赋值

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

//根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK；如果查找失败或关键字不唯一，返回ERROR

{

int i,v[MAX\_KEY]={0};

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

{//标记已出现过的顶点关键字

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

}

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

{//遍历顶点数组

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

{//查找到待修改顶点

if(value.key==u||!v[value.key])

{//当新关键字不与现有关键字重复，或新关键字与待修改顶点关键字相同时，进行赋值操作

G.vertices[i].data.key=value.key;

strcpy(G.vertices[i].data.others,value.others);

return OK;

}

}

}

return ERROR;

}

//菜单功能选择 5 获取第一邻接点

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。若顶点无邻接顶点，返回ERROR；若顶点不在图中，返回INFEASIBLE。

{

int i;

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

{//遍历顶点数组，查找到与u关键字相同的顶点

if(u==G.vertices[i].data.key&&G.vertices[i].firstarc) //顶点有邻接顶点

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

else if(u==G.vertices[i].data.key) //顶点无邻接顶点

return OVERFLOW;

}

return INFEASIBLE;

}

//菜单功能选择 6 获取下一邻接点

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

//根据v在图G中查找顶点，查找成功返回顶点v的邻接顶点相对于w的下一邻接顶点的位序，查找邻接顶点失败返回ERROR，查找顶点失败返回INFEASIBLE

{

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

{

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

{

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

for(;tmp;tmp=tmp->nextarc)

if(w==G.vertices[tmp->adjvex].data.key&&tmp->nextarc)

return tmp->nextarc->adjvex;

return ERROR; //查找邻接顶点失败

}

}

return INFEASIBLE; //查找顶点失败

}

//菜单功能选择 7 插入顶点

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

{

int i,flag[MAX\_KEY]={0};

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

return OVERFLOW; //顶点集已满

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

{//标记出现过的关键字

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

}

if(!flag[v.key])

{//关键字不重复，则新增顶点，否则报错

G.vertices[i].data.key=v.key;

strcpy(G.vertices[i].data.others,v.others);

G.vertices[i].firstarc=NULL;

G.vexnum++;

return OK;

}

else

return ERROR;

}

//菜单功能选择 8 删除顶点

status DeleteVex(ALGraph &G,KeyType v)

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

{

int a=-1,flag[MAX\_KEY]={0};

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

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

if(!flag[v]||!G.vexnum)

return ERROR;//图中无顶点（空图）或无对应关键字的顶点时，返回ERROR

else

{

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

{

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

{//查找到关键字对应的顶点时，将其邻接表中的弧全部删掉

for(ArcNode \*tmp1=G.vertices[i].firstarc;tmp1;)

{

G.vertices[i].firstarc=tmp1->nextarc;

ArcNode \*tmp2=tmp1;

tmp1=tmp1->nextarc;

free(tmp2);

tmp2=NULL;

G.arcnum--;

}

a=i; //a用于记录待删除顶点的位序，便于后续修改邻接表

}

else

{//不是关键字对应的顶点时，查找邻接表中与关键字顶点相关的弧并删除

for(ArcNode \*tmp1=G.vertices[i].firstarc;tmp1;tmp1=tmp1->nextarc)

{//因无图头结点，关键字对应于第一邻接点或其他点时要分开讨论

//使用tmp1、tmp2两个顶点指针进行顶点删除操作

ArcNode \*tmp2;

if(G.vertices[G.vertices[i].firstarc->adjvex].data.key==v)

{//关键字对应于第一邻接点

tmp2=G.vertices[i].firstarc;

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

free(tmp2);

tmp2=NULL;

break;

}

else if(tmp1->nextarc&&G.vertices[tmp1->nextarc->adjvex].data.key==v)

{//关键字对应于除第一邻接点外的其他顶点

tmp2=tmp1->nextarc;

tmp1->nextarc=tmp2->nextarc;

free(tmp2);

tmp2=NULL;

break;

}

}

}

}

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

{//在顶点数组中删除顶点，并修改其后位序的顶点在邻接表中的位序值

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

{

for(int j=i;j<G.vexnum-1;j++)

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

for(ArcNode \*t=G.vertices[i].firstarc;t;t=t->nextarc)

{//在顶点数组中移位后要对补充到原删除顶点位置的顶点的邻接表进行修改

if(t->adjvex>a)

t->adjvex-=1; //删除顶点其后位序的顶点在邻接表中位序减一

}

G.vexnum--;

}

else

for(ArcNode \*t=G.vertices[i].firstarc;t;t=t->nextarc)

if(t->adjvex>a)

t->adjvex-=1;

}

return OK;

}

}

//菜单功能选择 9 插入弧

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

//在图G中增加弧<v,w>，成功返回OK,否则返回ERROR

{

int i,a,b,flag[MAX\_KEY]={0};

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

{//标记图中所有顶点；用a、b记录待插入弧的两个顶点

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

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

a=i;

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

b=i;

}

if(a==b) //待插入弧为自环时，不进行操作

return OK;

if(!flag[v]||!flag[w]) //待插入弧的任一顶点不在图中时，返回ERROR

return INFEASIBLE;

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

{//分别查找v、w对应的顶点，在其邻接表中首插法插入弧

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

{

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

for(;tmp;tmp=tmp->nextarc) //判断待插入的弧是否已经存在，存在则返回ERROR

if(tmp->adjvex==b)

return ERROR;

tmp=G.vertices[i].firstarc;

G.vertices[i].firstarc=(ArcNode \*)malloc(sizeof(ArcNode));

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

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

}

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

{

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

for(;tmp;tmp=tmp->nextarc)

if(tmp->adjvex==a)

return ERROR;

tmp=G.vertices[i].firstarc;

G.vertices[i].firstarc=(ArcNode \*)malloc(sizeof(ArcNode));

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

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

}

}

G.arcnum++;

return OK;

}

//菜单功能选择 10 删除弧

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

//在图G中删除弧<v,w>，成功返回OK,否则返回ERROR

{

int i,a,b,flag[MAX\_KEY]={0};

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

{//标记图中所有顶点；用a、b记录待删除弧的两个顶点

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

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

a=i;

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

b=i;

}

if(!flag[v]||!flag[w]) //待删除弧的任一顶点不在图中时，返回ERROR

return INFEASIBLE;

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

{

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

{//分别查找v、w对应的顶点，在其邻接表中进行删除弧操作

ArcNode \*tmp1=G.vertices[i].firstarc;

int pt=0; //pt用于标记待删除弧是否存在，不存在则返回ERROR

for(;tmp1;tmp1=tmp1->nextarc)

if(tmp1->adjvex==b)

{

pt=1;

break;

}

if(pt)

{//因无图头结点，关键字对应于第一邻接点或其他点时要分开讨论

//使用tmp1、tmp2两个顶点指针进行弧删除操作

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

{//关键字对应于第一邻接点

tmp1=G.vertices[i].firstarc->nextarc;

ArcNode \*tmp2=G.vertices[i].firstarc;

G.vertices[i].firstarc=tmp1;

free(tmp2);

tmp2=NULL;

}

else

{//关键字对应于除第一邻接点外的其他顶点

for(tmp1=G.vertices[i].firstarc;tmp1->nextarc->adjvex!=b;tmp1=tmp1->nextarc);

ArcNode \*tmp2=tmp1->nextarc;

tmp1->nextarc=tmp2->nextarc;

free(tmp2);

tmp2=NULL;

}

}

else

return ERROR;

}

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

{

ArcNode \*tmp1=G.vertices[i].firstarc;

int pt=0;

for(;tmp1;tmp1=tmp1->nextarc)

if(tmp1->adjvex==a)

{

pt=1;

break;

}

if(pt)

{

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

{

tmp1=G.vertices[i].firstarc->nextarc;

ArcNode \*tmp2=G.vertices[i].firstarc;

G.vertices[i].firstarc=tmp1;

free(tmp2);

tmp2=NULL;

}

else

{

for(tmp1=G.vertices[i].firstarc;tmp1->nextarc->adjvex!=a;tmp1=tmp1->nextarc);

ArcNode \*tmp2=tmp1->nextarc;

tmp1->nextarc=tmp2->nextarc;

free(tmp2);

tmp2=NULL;

}

}

else

return ERROR;

}

}

G.arcnum--;

return OK;

}

//深搜的辅助函数，用于递归查找

void DFS(AdjList verx,int index,int visited[100],void (\*visit)(VertexType))

{

ArcNode \*p=verx[index].firstarc;

if(visited[index])

return;

visited[index]=1;

visit(verx[index].data);

while(p)

{

if(!visited[p->adjvex])

DFS(verx,p->adjvex,visited,visit);

p=p->nextarc;

}

}

//菜单功能选择 11 深度优先搜索遍历

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

{

if(!G.vexnum)//空图

return ERROR;

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

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

if(!visited[i])

DFS(G.vertices,i,visited,visit);

return OK;

}

//菜单功能选择 12 广度优先搜索遍历

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

{

if (G.vexnum==0) //空图

return ERROR;

QUEUE Q;

int visited[G.vexnum]={0}; //数组标记各顶点是否被访问过

iniQueue(Q);

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

{//广度优先遍历每一个连通子图

if(!visited[i])

{//如果当前顶点不在被遍历过的连通子图中

visited[i]=1;

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

enQueue(Q,i);

while(!isEmptyQueue(Q))

{//队列非空时循环

deQueue(Q,i);

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

for(;q!=NULL;q=q->nextarc)

{//依次将当前顶点的所有未访问过的邻接点入队列，并标记为已访问

if(!visited[q->adjvex])

{

enQueue(Q,q->adjvex);

visited[q->adjvex]=1;

visit(G.vertices[q->adjvex].data);

}

}

}

}

}

return OK;

}

//菜单功能选择 13 无向图存档

status SaveGraph(ALGraph G, char FileName[])

//将图的数据写入到文件FileName中

{

FILE \*fp;

if(!G.vexnum)

return INFEASIBLE;

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

return ERROR;

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

{ //写入每一个顶点

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

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

{ //顺序写入该顶点的每一个邻接点的位置

fprintf(fp, "%d ", p->adjvex);

}

fprintf(fp, "%d ", -1); //邻接点结尾处写上-1

}

fprintf(fp, "%d %s ", -1, "nil"); //顶点结尾处写上-1

fclose(fp);

return OK;

}

//菜单功能选择 14 无向图读档

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

//读入文件FileName的图数据，创建图的邻接表

//本函数调用栈的数据结构及其操作函数

{

if (G.vexnum > 0)

return INFEASIBLE;

FILE \*fp;

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

return ERROR;

G.vexnum = G.arcnum = 0;

KeyType key; //存储从文件中读取顶点key值的临时变量

int ConnectVerLocate; //存储读取的顶点邻接点的位置的临时变量

char others[20]; //存储读取的顶点的data中others分量的临时变量

stack S; //定义栈

iniStack(S);

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

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

for (int i = 0; key != -1 && G.vexnum < MAX\_VERTEX\_NUM; i++)

{ //创建顶点

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

strcpy(G.vertices[i].data.others, others);

G.vertices[i].firstarc = NULL;

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

for (; ConnectVerLocate != -1;)

{ //建立顶点的邻接表

//通过入栈出栈把顺序倒置，确保读取后顶点邻接表顺序与原图相同

push(S, ConnectVerLocate);

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

}

for (ArcNode \*p; !isEmptyStack(S);)

{ //栈不为空时循环

ConnectVerLocate = pop(S); //出栈

//从邻接表头部插入邻接点结点

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

p->adjvex = ConnectVerLocate;

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

G.vertices[i].firstarc = p;

G.arcnum++; //图的边数自增

}

G.vexnum++; //图的顶点数自增

fscanf(fp, "%d %s", &key, others); //读取下一顶点

}

G.arcnum/=2;

fclose(fp);

return OK;

}

//菜单功能选择 15 多无向图管理之增加无向图

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

// 在Lists中增加一个名称为ListName的无向图。

{

if(Lists.length<Lists.listsize)

{

int i,flag=0;

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

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

{

flag=1;

break;

}

if(!flag)

{

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

VertexType V[30]; //由于图为空时相当于图不存在，增加图时即向图中输入元素

KeyType VR[100][2];

int i=0,j;

printf("请输入图中所有顶点关键字和字符串，输入（-1，null）终止：\n");

do {

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

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

i=0;

printf("请输入图中所有弧，输入（-1，-1）终止：\n");

do {

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

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

j=CreateGraph(Lists.elem[Lists.length].G,V,VR);

if(j==ERROR)

return ERROR;

Lists.length++;

return OK;

}

else

return INFEASIBLE;

}

else

return OVERFLOW;

}

//菜单功能选择 16 多无向图管理之删除无向图

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

// 在Lists中删除一个名称为ListName的无向图。

{

int i,flag=0;

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

{

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

{

flag=1;

break;

}

}

if(flag)

{

DestroyGraph(Lists.elem[i].G);

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

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

if(i!=Lists.length-1)

DestroyGraph(Lists.elem[Lists.length-1].G);

Lists.elem[Lists.length-1].G.vexnum=Lists.elem[Lists.length-1].G.arcnum=0;

Lists.elem[Lists.length-1].name[0]='\0';

Lists.length--;

return OK;

}

else

return ERROR;

}

//菜单功能选择 17 多无向图管理之查找无向图

int LocateGraph(LISTS Lists,char ListName[])

// 在Lists中查找一个名称为ListName的无向图，查找成功返回逻辑序号，否则返回ERROR。

{

int i,flag=0;

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

{

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

{

flag=1;

break;

}

}

if(!flag||!Lists.length)

return ERROR;

else

return i+1;

}

//辅助函数：访问图顶点

void visit(VertexType v)

{

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

}

//辅助函数：进入二级菜单

void SecondaryTable(ALGraph &G,int op)

{

int i,j;

char Name[30]={'\0'},FileName[30]={'\0'};

KeyType u,w;

VertexType value;

while(op){

system("cls");

printf("\n\n");

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

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

printf(" 1. CreateGraph 7. InsertVex 13. SaveGraph\n");

printf(" 2. DestroyGraph 8. DeleteVex 14. LoadGraph\n");

printf(" 3. LocateVex 9. InsertArc 15. ShowGraph\n");

printf(" 4. PutVex 10. DeleteArc\n");

printf(" 5. FirstAdjVex 11. DFSTraverse\n");

printf(" 6. NextAdjVex 12. BFSTraverse\n");

printf(" 0. Exit\n");

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

printf(" 请选择你的操作[0~15]:");

scanf("%d",&op);

switch(op){

case 1:

i=0;

VertexType V[30];

KeyType VR[100][2];

printf("请输入图中所有顶点关键字和字符串，输入（-1，null）终止：\n");

do {

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

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

i=0;

printf("请输入图中所有弧，输入（-1，-1）终止：\n");

do {

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

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

j=CreateGraph(G,V,VR);

if(j==OK)

printf("创建无向图成功！\n");

else if(j==INFEASIBLE)

printf("无向图添加失败，该无向图已存在！\n");

else if(j==ERROR)

printf("创建无向图失败，无向图数据输入错误！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 2:

j=DestroyGraph(G);

if(j==OK)

printf("销毁无向图成功！\n");

else

printf("销毁无向图失败，无向图不存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 3:

printf("请输入待查找顶点关键字：");

scanf("%d",&u);

j=LocateVex(G,u);

if(j==INFEASIBLE)

printf("关键字为 %d 顶点不存在，查找失败！\n",u);

else

printf("查找顶点成功，关键字为 %d 的顶点在邻接表数组中关键字和字符串为 %d %s ！\n",u,G.vertices[j].data.key,G.vertices[j].data.others);

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 4:

printf("请输入需要重新赋值的顶点的关键字：");

scanf("%d",&u);

printf("请输入新赋值的顶点关键字和字符串：");

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

j=PutVex(G,u,value);

if(j==OK)

printf("赋值成功！\n");

else if(j==ERROR)

printf("赋值失败，可能是待重新赋值的关键字不存在，或赋值后无向图中关键字重复！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 5:

printf("请输入需要获取第一邻接点的顶点的关键字：");

scanf("%d",&u);

j=FirstAdjVex(G,u);

if(j==OVERFLOW)

printf("获取失败，该顶点无邻接顶点！\n");

else if(j==INFEASIBLE)

printf("获取失败，无向图中无此顶点！\n");

else

printf("获取成功，关键字为 %d 的顶点的第一邻接顶点关键字和字符串为 %d %s ！\n",u,G.vertices[j].data.key,G.vertices[j].data.others);

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 6:

printf("请输入需要获取邻接点的顶点的关键字：");

scanf("%d",&u);

printf("请输入需要获取下一邻接点的顶点的关键字：");

scanf("%d",&w);

j=NextAdjVex(G,u,w);

if(j==ERROR)

printf("获取失败，w无下一邻接顶点！\n");

else if(j==INFEASIBLE)

printf("获取失败，v不在无向图中！\n");

else

printf("获取成功，顶点v的邻接顶点相对于w的下一邻接顶点的位序为 %d %s ！\n",G.vertices[j].data.key,G.vertices[j].data.others);

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 7:

printf("请输入插入顶点的关键字和字符串：");

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

j=InsertVex(G,value);

if(j==OK)

printf("插入顶点成功！\n");

else if(j==OVERFLOW)

printf("插入顶点失败，顶点集已满！\n");

else if(j==ERROR)

printf("插入顶点失败，新插入顶点的关键字与已有顶点关键字重复！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 8:

printf("请输入删除顶点的关键字：");

scanf("%d",&u);

j=DeleteVex(G,u);

if(j==OK)

printf("删除顶点成功！\n");

else

printf("删除顶点失败，无向图为空或无此顶点！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 9:

printf("请输入待插入弧的两个顶点：");

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

j=InsertArc(G,u,w);

if(j==OK)

printf("插入弧成功！\n");

else if(j==INFEASIBLE)

printf("插入弧失败，输入的弧顶点不存在于图中！\n");

else if(j==ERROR)

printf("插入弧失败，插入的弧已存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 10:

printf("请输入待删除弧的两个顶点：");

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

j=DeleteArc(G,u,w);

if(j==OK)

printf("删除弧成功！\n");

else if(j==INFEASIBLE)

printf("删除弧失败，输入的弧顶点不存在于图中！\n");

else if(j==ERROR)

printf("删除弧失败，待删除的弧不存在！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 11:

j=DFSTraverse(G,visit);

if(j==OK)

printf("\n深度搜索无向图成功！\n");

else

printf("\n搜索图失败，图为空！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 12:

j=BFSTraverse(G,visit);

if(j==OK)

printf("\n广度搜索无向图成功！\n");

else

printf("\n搜索图失败，图为空！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 13:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=SaveGraph(G,FileName);

if(j==OK)

printf("无向图存档成功！\n");

else if(j==ERROR)

printf("创建/打开文件失败！\n");

else

printf("无向图存档失败，无向图为空！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 14:

printf("请输入待操作文件路径和名称：");

scanf("%s",FileName);

j=LoadGraph(G,FileName);

if(j==OK)

printf("无向图读档成功！\n");

else if(j==ERROR)

printf("打开文件失败！\n");

else

printf("无向图读档失败，无向图已存在，读档会使原无向图中数据失去！\n");

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 15:

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

{

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

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

while (p)

{

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

p=p->nextarc;

}

printf("\n");

}

getchar();

printf("---任意键继续---\n");

getchar();

break;

case 0:

break;

}//end of switch

}//end of while

}