

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级：**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期：**

**计算机科学与技术学院**

目 录

[1 基于顺序存储结构的线性表实现 3](#_Toc75116874)

[1.1 问题描述 3](#_Toc75116875)

[1.1.1 线性表基本运算定义 4](#_Toc75116876)

[1.2 系统设计 5](#_Toc75116877)

[1.2.1 线性表管理设计 5](#_Toc75116878)

[1.2.2 演示系统菜单设计 6](#_Toc75116879)

[1.2.3 线性表数据结构设计 6](#_Toc75116880)

[1.2.4 宏定义 7](#_Toc75116881)

[1.2.5 状态返回值类型和数据元素类型的定义 7](#_Toc75116882)

[1.2.6 文件组织形式 8](#_Toc75116883)

[1.3 系统实现 8](#_Toc75116884)

[1.3.1 程序实现的环境及程序开发工具与开发语言 8](#_Toc75116885)

[1.3.2 各函数实现 8](#_Toc75116886)

[**1.4 系统测试** 10](#_Toc75116887)

[1.4.1菜单演示 10](#_Toc75116888)

[1.4.2函数测试 11](#_Toc75116889)

[1.5 实验小结 17](#_Toc75116890)

[2 基于链式存储结构的线性表实现 18](#_Toc75116891)

[2.1 问题描述 18](#_Toc75116892)

[2.1.1线性表基本运算定义 18](#_Toc75116893)

[2.2 系统设计 20](#_Toc75116894)

[2.2.1 总体流程框架 20](#_Toc75116895)

[2.2.2 定义常量、数据类型及数据结构 20](#_Toc75116896)

[2.2.3 基本操作函数实现 21](#_Toc75116897)

[2.2.4 多线性表管理设计 25](#_Toc75116898)

[2.3 系统实现 25](#_Toc75116899)

[2.3.1 程序实现的环境及程序开发工具与开发语言 25](#_Toc75116900)

[2.3.2 文件组织形式及代码实现 25](#_Toc75116901)

[2.4 系统测试 26](#_Toc75116902)

[2.4.1菜单演示 26](#_Toc75116903)

[2.4.2各函数测试 27](#_Toc75116904)

[2.5 实验小结 35](#_Toc75116905)

[3 基于二叉链表的二叉树实现 36](#_Toc75116906)

[3.1 问题描述 36](#_Toc75116907)

[3.1.1 二叉树基本运算定义 36](#_Toc75116908)

[3.2 系统设计 38](#_Toc75116909)

[3.2.1 总体流程框架 38](#_Toc75116910)

[3.2.2 定义常量、数据类型及数据结构 39](#_Toc75116911)

[3.2.3 多二叉树管理设计 40](#_Toc75116912)

[3.3 系统实现 40](#_Toc75116913)

[3.3.1 程序实现的环境及程序开发工具与开发语言 40](#_Toc75116914)

[3.3.2 文件组织形式及代码实现 40](#_Toc75116915)

[3.4 系统测试 41](#_Toc75116916)

[3.4.1 菜单演示 41](#_Toc75116917)

[3.4.2 各函数测试 42](#_Toc75116918)

[3.5 实验小结 48](#_Toc75116919)

[4 基于邻接表的图实现 49](#_Toc75116920)

[4.1 问题描述 49](#_Toc75116921)

[4.1.1 图基本运算定义 49](#_Toc75116922)

[4.2 系统设计 51](#_Toc75116923)

[4.2.1 总体流程框架 51](#_Toc75116924)

[4.2.2 定义常量、数据类型及数据结构 51](#_Toc75116925)

[4.2.3 多图管理设计 53](#_Toc75116926)

[4.3 系统实现 53](#_Toc75116927)

[4.3.1 程序实现的环境及程序开发工具与开发语言 53](#_Toc75116928)

[4.3.2 文件组织形式及代码实现 53](#_Toc75116929)

[4.4 系统测试 54](#_Toc75116930)

[4.4.1 菜单演示 54](#_Toc75116931)

[4.4.2 各函数测试 54](#_Toc75116932)

[4.5 实验小结 60](#_Toc75116933)

[参考文献 61](#_Toc75116934)

[附录A 基于顺序存储结构线性表实现的源程序 62](#_Toc75116935)

[附录B 基于链式存储结构线性表实现的源程序 77](#_Toc75116936)

[附录C 基于二叉链表二叉树实现的源程序 92](#_Toc75116937)

[附录D 基于邻接表图实现的源程序 117](#_Toc75116938)

# 1 基于顺序存储结构的线性表实现

## 1.1 问题描述

采用顺序表作为线性表的物理结构，构造一个具有菜单的功能演示系统，并实现线性表的基本运算。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。程序还定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素、查找元素、获得前驱、获得后继、插入元素、删除元素、遍历表等基本运算对应的函数。在此基础上，可以选择以文件的形式高效保存线性表数据逻辑结构(D,{R})的完整信息，实现了对线性表的存储和加载，即既可以将生成的线性表存入到相应的文件中，也可以从文件中获取线性表进行操作。同时，演示系统可选择实现多个线性表的管理。

## 1.1.1 线性表基本运算定义

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

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

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

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

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

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

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

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

⑻**获得前驱**：函数名称是PriorElem(L,cur\_e,pre\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre\_e无定义。

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

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

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

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

## 1.2 系统设计

## 1.2.1 线性表管理设计

本次实验中采用顺式储存的结构来管理各个线性表，每个线性表有一个名字 (如图1-1所示),实现如下

typedef struct{ //线性表的集合类型定义

struct { char name[30];

SqList L;

} elem[10];

int length;

}LISTS;

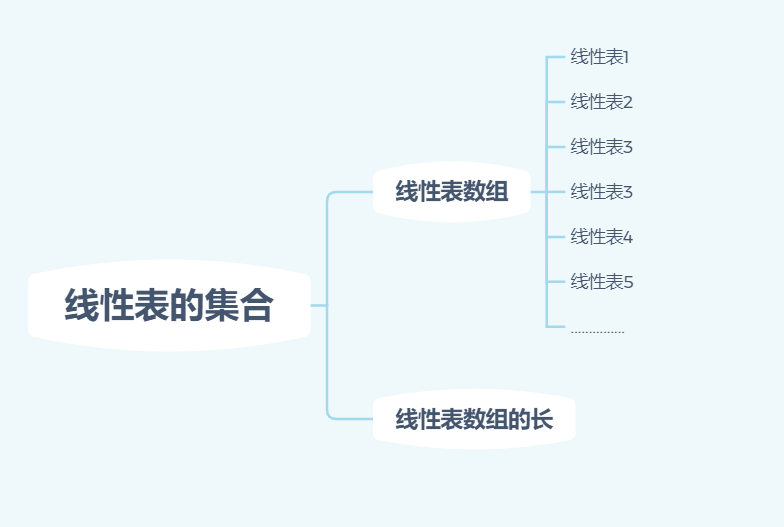


图1-1线性表管理示意图

## 1.2.2 演示系统菜单设计

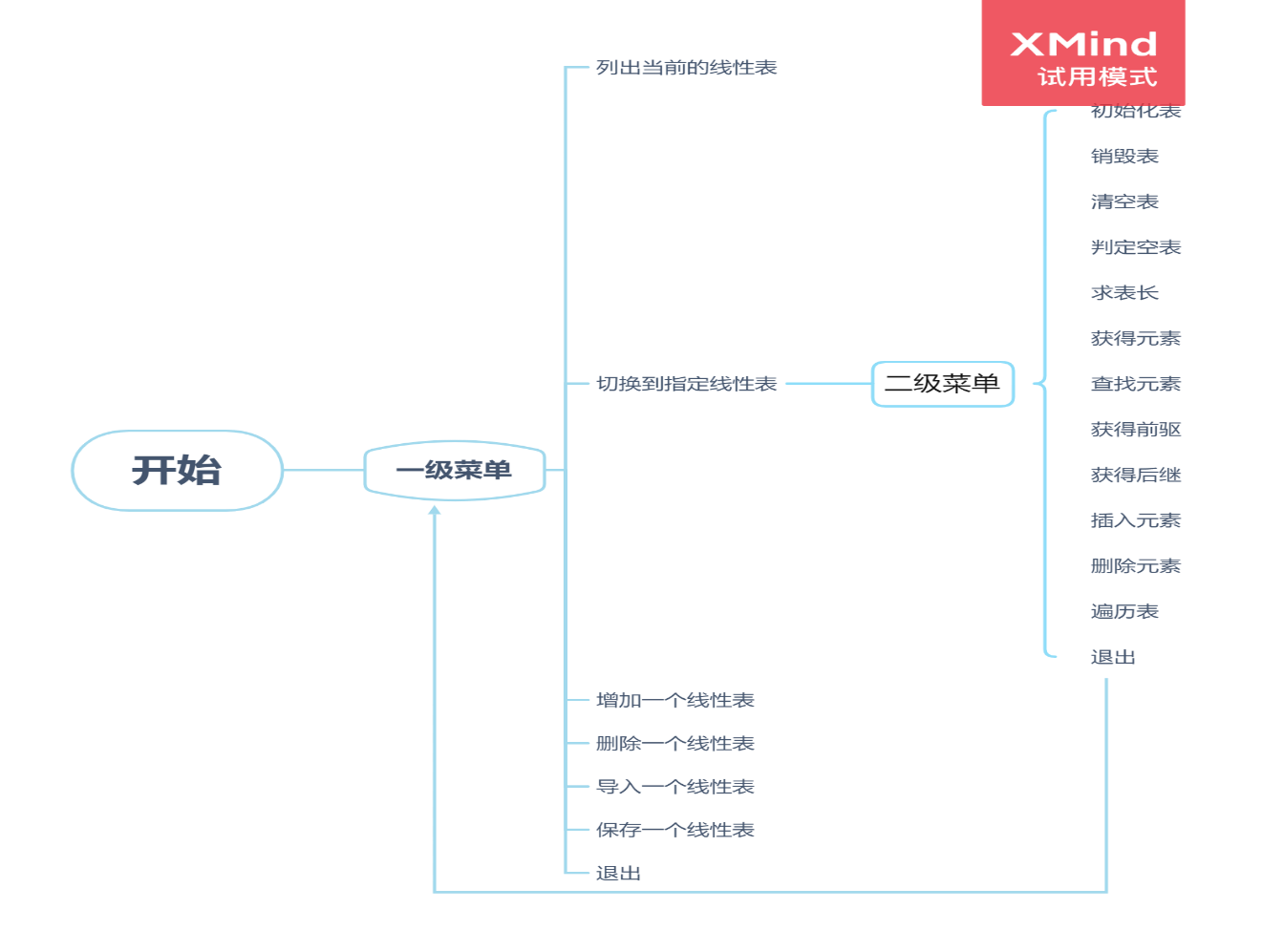
演示系统采用二级菜单，在一级菜单中可以查看当前已经创建的线性表，并通过切换到指定线性表进入二级菜单。（主要功能如图1-2所示）

图1-2演示系统菜单功能示意图

## 1.2.3 线性表数据结构设计

采用一个指向数据元素的指针来保存数据元素，在每次初始化重新分配内存，并能在空间不足时拓宽内存，用一个数length保存当前线性表的长度，用listsize保存线性表的最大长度，将这三个数组成一个结构体(如图1-3所示)

实现如下。

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

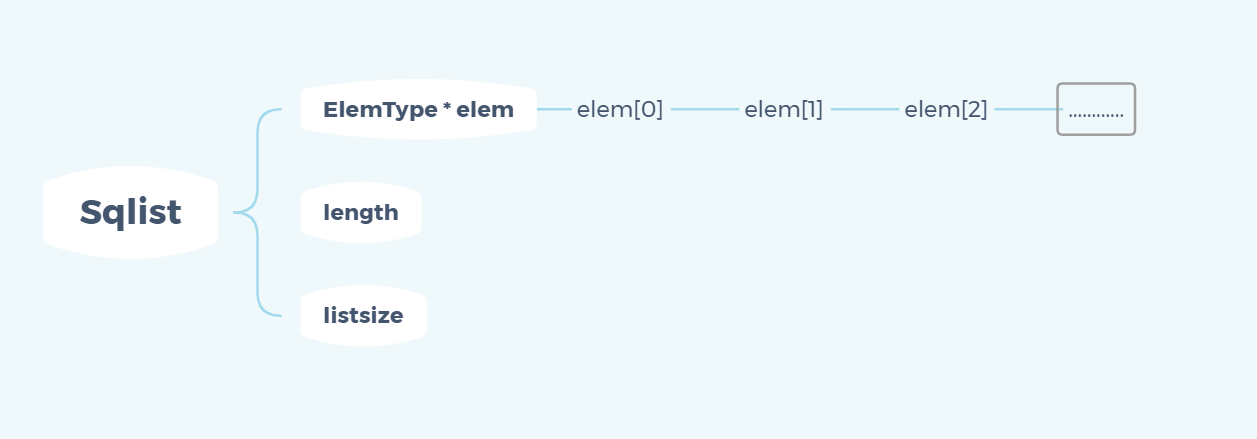


图1-3顺序表结构示意图

## 1.2.4 宏定义

此处用编译预处理（宏定义）的方法，定义了TRUE、FALSE、OK、ERROR、OVERFLOW等六个值，表示真、假、成功、失败、溢出等状态。 此外，还定义LIST\_INIT\_SIZE、LISTINCREMENT，分别用于表示线性表初始化时最大长度和再扩充时增加的长度。

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

## 1.2.5 状态返回值类型和数据元素类型的定义

typedef int status;

typedef int ElemType;

## 1.2.6 文件组织形式

参考工程标准，将一些定义以及需要使用的头文件放在了同目录的def.h文件中。并将测试文件放在同目录下（如图1-4所示）。



图1-4文件组织示意图

## 1.3 系统实现

## 1.3.1 程序实现的环境及程序开发工具与开发语言

本人未使用集成开发环境(IDE),代码由代码编辑器VScode编辑，代码使用c++编写，在cmd命令行下使用mingw编译器g++进行编译，由于g++默认的编码为GBK，故需要改变编码为UTF-8，用于显示中文，编译命令如下：

**> g++ -finput-charset=UTF-8 -fexec-charset=GBK -o demo main.cpp**

获得可执行文件（演示菜单）demo.exe。

## 1.3.2 各函数实现

(1)表初始化:初始条件为线性表不存在，传入一个线性表，为储存申请空间，若申请不成功则以OVERFLOW退出，若申请成功则将其长度设置为0，最大表长设置为LIST\_INIT\_SIZE，返回OK。该算法空间复杂度为O(1),时间复杂度为O(1)。

(2)销毁表:如果线性表L存在，销毁线性表L，释放数据元素的空间，即清空elem中的空间并将其设置为NULL，返回OK，否则返回INFEASIBLE。该算法空间复杂度为O(1),时间复杂度为O(1)。

(3)清空表:在c语言中，由于无需关心储存的具体元素，可通过直接将表长设为0来清空表。该算法空间复杂度为O(1),时间复杂度为O(1)。

(4)判断空表:通过检查表长length字段是否为零来判断是否为空表。该算法空间复杂度为O(1),时间复杂度为O(1)。

(5)求表长:返回length字段的值。该算法空间复杂度为O(1),时间复杂度为O(1)。

(6)获得元素:根据传入的逻辑序号-1后通过elem[i-1]获得元素的值并返回。该算法空间复杂度为O(1),时间复杂度为O(1)。

(7)查找元素:遍历表中的值并与传入的值对比，一旦找到就停止遍历，并返回逻辑序号。若未找到值则返回ERROR。据分析，该算法时间复杂度为O(N)，其空间复杂度为O(1)。

(8)获得前驱

遍历表中的值并与传入的值对比，一旦找到就停止遍历，并返回前一个元素的值。若为序号为0则返回ERROR。据分析，该算法时间复杂度为O(N)，其空间复杂度为O(1)。

(9) 获得后继

遍历表中的值并与传入的值对比，一旦找到就停止遍历，并返回后的值。若为序号length-1则返回ERROR。据分析，该算法时间复杂度为O(N)，其空间复杂度为O(1)。

(10) 插入元素

首先判断插入位置是否有误，若无误，找到插入位置，将该位置往后的元素向后移动（赋值拷贝）,然后将该元素直接赋值到插入位置上。据分析，该算法时间复杂度为θ(N)，其空间复杂度为O(1)

(11) 删除元素

首先判断删除位置是否有误，若无误，记录下该元素的值，将之后的元素向前移动一位，之后返回记录下的值。据分析，该算法时间复杂度为O(N)。其空间复杂度为O(1)。

(12) 遍历表

从elem[0]到elem[length-1]依次对元素执行visit()函数（目前实现了打印所有元素）。据分析，该算法时间复杂度为O(N)。其空间复杂度为O(1)。

**1.4 系统测试**

## 1.4.1菜单演示

（1）一级菜单如下：

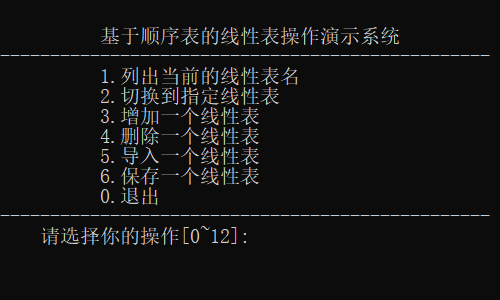


图 1-5 一级菜单

（2）二级菜单如下：

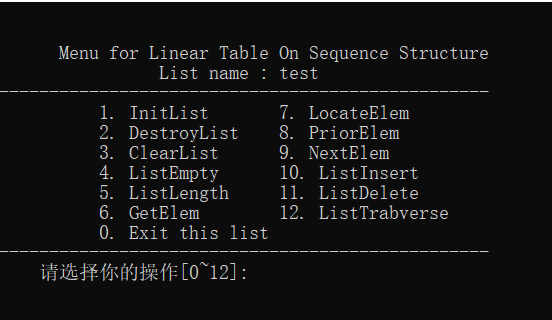


图 1-6 二级菜单

## 1.4.2函数测试

⑴**初始化表**

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 创建线性表成功 |  |
| 用例二 | 线性表存在 | 创建线性表失败 |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义创建了对应的线性表，符合实验要求。

⑵**销毁表**：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 销毁失败 |  |
| 用例二 | 线性表存在 | 销毁成功 |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义销毁了对应的线性表，符合实验要求。

⑶**清空表**：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 清空失败，给出提示 |  |
| 用例二 | 线性表存在 | 清空成功 |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义清空了对应的线性表，符合实验要求。

⑷**判定空表**：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 判定失败，给出提示 |  |
| 用例二 | 线性表存在且为空 | 判定线性表为空 |  |
| 用例三 | 线性表存在且不为空 | 判定线性表不为空 |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义判断线性表是否为空并返回，符合实验要求。

⑸**求表长**：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 求表长失败，给出提示 |  |
| 用例二 | 线性表存在，表长为0 | 给出表长 |  |
| 用例三 | 线性表存在，表长为3 | 给出表长 |  |

测试结论：综合上述测试，函数对线性表存在于不存在，表为空的情况都可以处理，按照定义给出了表长，符合实验要求。

⑹**获得元素**：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 失败，给出提示 |  |
| 用例二 | 线性表存在，  [2,2,1],1 | 给出值为2 |  |
| 用例三 | 线性表存在，[2,2,1],4 | 失败，给出提示 |  |

测试结论：综合上述测试，函数对线性表存在于不存在，输入不合法的情况都可以处理，按照定义给出了对应逻辑序号元素的值，符合实验要求。

⑺**查找元素**：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 失败，给出提示 |  |
| 用例二 | 线性表存在，  [2,3,1，4,1]  查找1 | 给出第一个1的逻辑序号3 |  |
| 用例三 | 线性表存在，[2,3,1，4,1]  查找5 | 失败，给出提示 |  |

测试结论：综合上述测试，函数对线性表存在于不存在，输入不合法的情况都可以处理，按照定义查找对应的元素的第一个逻辑序号，符合实验要求。

⑻**获得前驱**：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 失败，给出提示 |  |
| 用例二 | 线性表存在，  [2,3,1，4,1]  输入2 | 第一个2无前驱 |  |
| 用例三 | 线性表存在，  [2,3,1，4,1]  输入1 | 第一个1前驱为3 |  |
| 用例四 | 线性表存在，[2,3,1，4,1]  查找5 | 该元素无前驱 |  |

测试结论：综合上述测试，函数对线性表存在于不存在，输入不合法的情况都可以处理，按照定义返回输入元素的前驱，符合实验要求。

⑼**获得后继**：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 失败，给出提示 |  |
| 用例二 | 线性表存在，  [2,3,1，4,5]  输入5 | 5无后继 |  |
| 用例三 | 线性表存在，  [2,3,1，4,5]  输入4 | 第一个4后继为5 |  |
| 用例四 | 线性表存在，[2,3,1，4,5]  输入6 | 该元素无后继 |  |

测试结论：综合上述测试，函数对线性表存在于不存在，输入不合法的情况都可以处理，按照定义返回输入元素的后继，符合实验要求。

⑽**插入元素**：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 失败，给出提示 |  |
| 用例二 | 线性表存在，  [2,3,1,4,5]  插入6到3。 | 插入成功，线性表变为[2,3,6,1,4,5] |  |
| 用例三 | 线性表存在，  [2,3,1,4,5]  插入6到6 | 插入成功，线性表变为[2,3,1,4,5，6] |  |
| 用例四 | 线性表存在，[2,3,1,4,5]  插入6到7 | 插入失败，给出提示 |  |

测试结论：综合上述测试，函数对线性表存在于不存在，输入不合法的情况都可以处理，按照定义将元素插入要求的位置，符合实验要求。

⑾**删除元素**：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 失败，给出提示 |  |
| 用例二 | 线性表存在，  [2,3,1,4,5]  删除位置4的元素。 | 删除成功，线性表变为[2,3,1,5]，  给出删除的值4 |  |
| 用例三 | 线性表存在，  [2,3,1,4,5]  删除位置6的元素 | 删除失败，给出提示 |  |

测试结论：综合上述测试，函数对线性表存在于不存在，输入不合法的情况都可以处理，按照定义删除输入位置的元素，符合实验要求。

⑿**遍历表**：

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例 | 程序输入 | 理论结果 | 运行结果 |
| 用例一 | 线性表不存在 | 给出提示：线性表不存在 |  |
| 用例二 | 线性表存在，为空表 | 打印空表，给出提示 |  |
| 用例三 | 线性表存在，  [2,3,1,4,5] | 打印 |  |

测试结论：综合上述测试，函数对线性表存在于不存在，输入不合法的情况都可以处理，设置visit（）为打印元素，结果为打印了线性表，符合实验要求。

测试小结：按照上述测试，基于顺序存储结构的线性表的基本运算都已经实现，且对特殊输入都进行了处理，故函数实现符合要求。

## 1.5 实验小结

在本次实验中，由于本人对内存管理较为熟悉，对错误输入的处理也有一定经验，故没有碰到较大的困难，较大的收获是对函数的分块和复用有了一定的理解，也自行了解了大型项目的程序组织方式并加以模仿，采用了更为规范的程序书写形式和组织形式。

# 2 基于链式存储结构的线性表实现

## 2.1 问题描述

采用链表作为线性表的物理结构，构造一个具有菜单的功能演示系统，并实现线性表的基本运算。其中，在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。程序还定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素、查找元素、获得前驱、获得后继、插入元素、删除元素、遍历表等基本运算对应的函数。在此基础上，可以选择以文件的形式高效保存线性表数据逻辑结构(D,{R})的完整信息，实现了对线性表的存储和加载，即既可以将生成的线性表存入到相应的文件中，也可以从文件中获取线性表进行操作。同时，演示系统可选择实现多个线性表的管理。

## 2.1.1线性表基本运算定义

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

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

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

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

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

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

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

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

8)获得前驱：函数名称是PriorElem(L,e,pre)；初始条件是线性表L已存在；操作结果是若e是L的数据元素，且不是第一个，则用pre返回它的前驱，否则操作失败，pre无定义。

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

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

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

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

本系统还实现了对多个线性表操作的支持。

本系统提供对线性表的文件输入输出功能。本功能通过两个函数实现，两个 函数的基本情况如下。

1) 保存到文件：函数名称为SaveList(L,char，FileName)初始条件是线性 表 L 已存在，操作结果是将线性表 L 中的元素逐个存入文件 Filename 中。

2) 从文件载入数据到线性表：函数名称为LoadList(L, FileName)初始条件为线性表 L 已存在且不包含元素；操作结果是从文件 Filename 中逐个 读取元素并载入到线性表 L 中。

## 2.2 系统设计

由于本次实验中只是采用链式存储作为线性表的实现方式，与第一次实验只是在底层实现上有区别，而对外表现相同，体现在每个函数上就是函数的功能相同，返回值相同，但实现方式完全不同（采用链表）。故可以采用第一次实验的演示系统，只需修改函数接口就可实现本次系统。

## 2.2.1 总体流程框架

演示系统采用二级菜单，在一级菜单中可以查看当前已经创建的线性表，并通过切换到指定线性表进入二级菜单。（主要功能如图2-1所示）

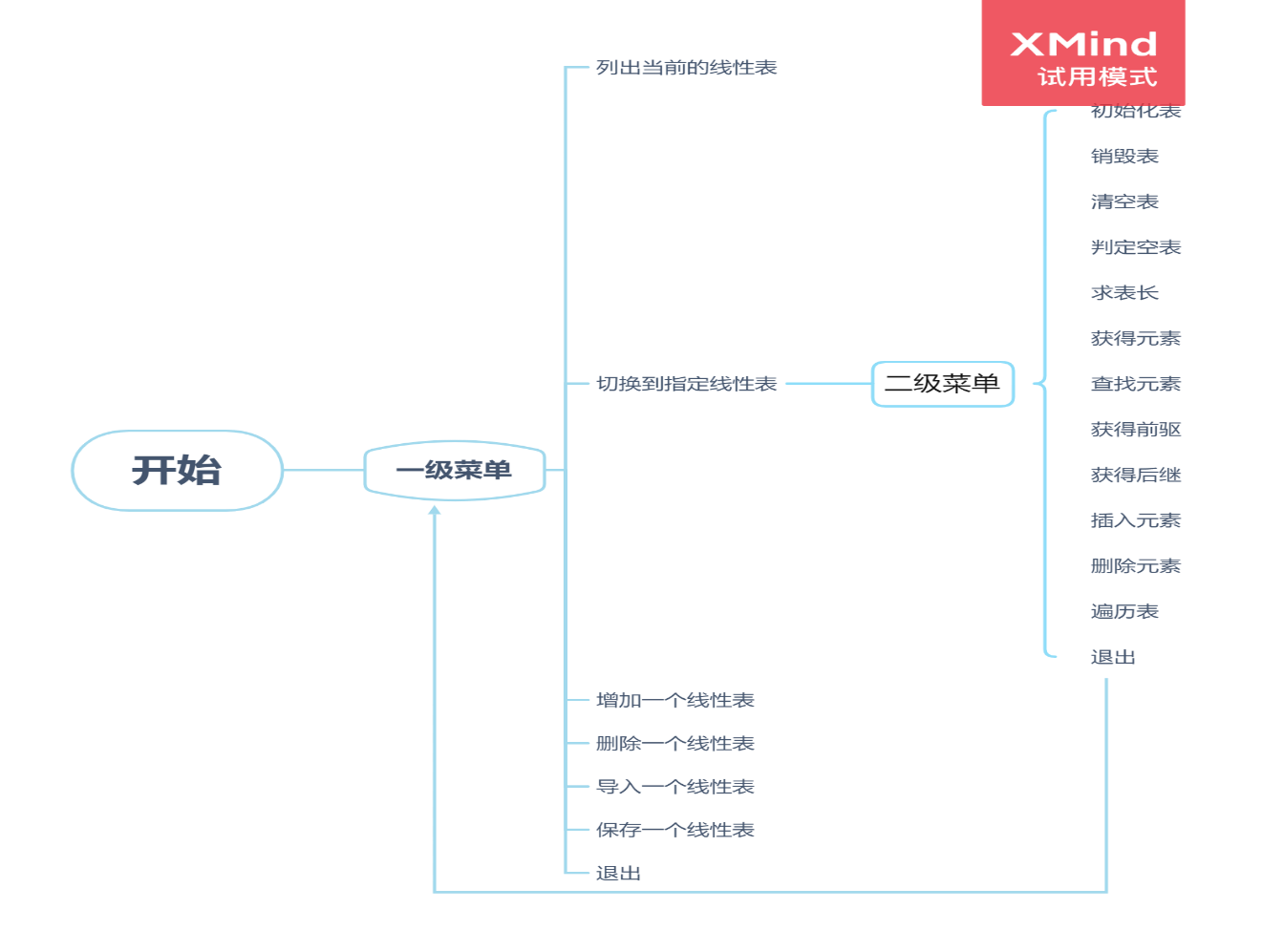


图2-1演示系统流程图

## 2.2.2 定义常量、数据类型及数据结构

1)定义常量。定义常量用于指定函数的返回值，用来判断函数执行的情况。 定义的常量有：TRUE(1), FALSE(0), OK(1), ERROR(0), INFEASTABLE(-1), OVERFLOW(-2)。表示真、假、成功、失败、溢出等状态。

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

2)定义数据元素的类型。此处定义了部分函数返回值的类型 status 以及单 链表中耽搁元素的类型 ElemType。本实现中将两者均定义为 int 类型。

typedef int status;//返回状态码类型定义

typedef int ElemType; //数据元素类型定义

3)定义单链表的数据结构。单链表结构体中包含两个元素，第一个元素为 单链表的数据域 data，为 ElemType 类型，第二个元素为单链表的指针域 next，为单链表结构体类型的指针。将该结构体类型定义为 Lnode。

单链表的数据结构在内存中的存储状态如图2-2所示，图中操作指针指向单 链表的第一个节点，单链表中的节点之间通过指针域的指针相互关联。

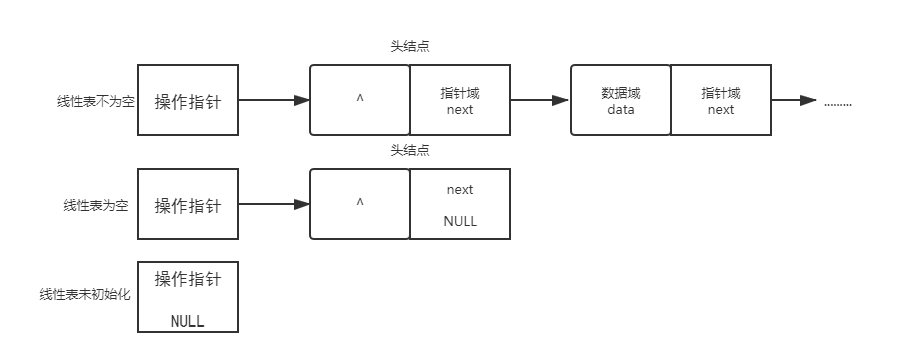


图2-2 单链表数据结构展示

## 2.2.3 基本操作函数实现

关于函数调用方法的说明：初始化表函数(InitaList)接受的单链表指针指 向的单链表应为未初始化的单链表，否则不调用函数并输出“单链表已存在”。其余11个基本操作的函数所接受的单链表指针指向的单链表应为已初始化表，否则不调用该函数并输出“线性表不存在”。

1)初始化表(InitaList)该函数接受一个Lnode类型的指针L，若L为空指针，则为L赋予一个值使其指向一个Lnode结点（头结点），将L的指针域next置为NULL，表明初始化成功，并返回OK。该函数只包含顺序结构，因此其时间复杂度为O(1)。

2) 销毁单链表(DestroyList)该函数接受一个LNode类型的指针L，

若L为空指针，表明线性表不存在，返回INFEASIBLE。

若L不为空指针，首先，若头结点的指针域为 NULL，说明单链表中没有元素，只需释放头结点的空间并把L设置为NULL，然后返回OK。若头结点的指针域不为NULL，则令now为头结点的指针域，释放头结点的空间，将L设置为NULL，然后用last记录当前now的值，将now后移，同时将last所指节点的内存释放，循环至now所指结点指针域为NULL，释放该结点,此时单链表中的全部元素均被释放，返回OK。该函数包含一个循环结构，由于函数需要逐个的将单链表中的每个元素的内存空间释放，因此需要循环n次(n为单链表的表长)，故函数的时间复杂度为O(n)。

3) 清空单链表(ClearList) 该函数接受一个 LNode 类型的指针L.

若L为空指针，表明线性表不存在，返回INFEASIBLE。

若L不为空指针，首先，若头结点的指针域为 NULL，说明单链表中没有元素，只需返回 OK。若头结点的指针域不为NULL，则令now为头结点的指针域，将头结点的指针域设为NULL表示链表为空，然后用last记录当前now的值，将now后移，同时将last所指节点的内存释放，循环至now所指结点指针域为NULL，释放该结点,此时单链表中的全部元素均被释放,即清空，返回OK。该函数包含一个循环结构，由于函数需要逐个的将单链表中的每个元素的内 存空间释放，因此需要循环n次(n为单链表的表长)，故函数的时间复杂度为O(n)。

4)判断空表(ListEmpty) 该函数接受一个 LNode 类型的指针L，若 L 指向头结点的指针域为 NULL，说明单链表为 空，返回 TRUE，若 L 指向头结点的指针域不为 NULL，说明单链表不为空，返回 FALSE。 该函数只有顺序和选择结构，因此时间复杂度为 O(1)。

5) 求单链表表长(ListLength) 该函数接受一个 LNode 类型的指针 L，令 length 为 0，将now每次置为当前节点的指 针域 next(初始置为头结点的指针域)，并将 length自增1，直到now的指针域为 NULL。此时length的值即 为单链表的长度。返回 length。 该函数包含一个循环结构，函数需要遍历单链表的每一个节点来计算元素的 数量，因此循环需要执行 n 次，故函数的时间复杂度为 O(n)。

6)获取线性表元素(GetElem)该函数接受一个LNode类型的指针L，待获取元素的位置i，以及该元素的值的存放地址e。若位置i超出了单链表的范围，则函数返回ERROR，否则，函数从单链表头指针的拷贝now开始向后移动i次，将此时L指向的节点数据域的值存储到e所指向的位置，并返回OK。该函数包含一个循环结构，当i小于0时，循环不执行，当获取第一个元素的位置时，循环执行1次，此时函数耗时最短，当获取单链表最后一个元素或i大于单链表的长度时，循环执行n次(n为单链表表长)，此时耗时最长，因此循环平均执行n/2次，故函数的时间复杂度为O(n)。

7)查找元素(LocateElem)该函数接受一个LNode类型的指针L，以及待查找的元素e。函数从L开始不断将L的拷贝now向后移动并设置计数器i统计移动次数，直到now的数据域与e的值相等，此时找到待查找元素，其位置为计数器的值i，若L的值最终变为NULL，则表明单链表中不存在待查找元素e，此时返回ERROR。该函数包含一个循环结构，若待查找元素为第一个元素，则循环执行1次，此时函数执行时间最短，若待查找元素为单链表的最后一个元素或待查找元素不在该单链表中，则循环执行n次，此时函数执行时间最长。因此循环平均执行n/2次，故函数的时间复杂度为O(n)。

8)查找前驱(PriorElem) 该函数接受一个 LNode 类型的指针L，待查找前驱的元素e，以及存储前驱值的内存地址pre。初始化now为头结点的指针域，若带查找前驱的元素e不在单链表L中，则函数返回INFEASTABLE，若单链表的第一个元素为e，则函数返回INFEASTABLE，否则，设置另一个指针last与now相同，now向后移动，直到now指向待查找 前驱的元素，此时last指向该元素的前驱，将last指向的节点的数据域的值赋值给pre指向的内存空间，并返回OK。若now最终指向NULL，即未在单链表中找到待查找前驱的元素，则返回INFEASTABLE。该函数包含一个循环结构，若单链表为空或待查找元素为单链表的第一个元素，则循环不执行，若待查找前驱的元素为单链表的第二个元素，则循环执行1次，此时函数耗时最短；若待查找前驱的元素为单链表的最后一个元素或该元素不在单链表中，则循环执行n次，此时函数耗时最长。因此循环平均执行n/2次，故函数的时间复杂度为O(n)。

9)查找后继(NextElem) 该函数接受一个LNode类型的指针 L，待查找后继的元素e，以及存储后继值的内存地址next。若单链表为空，则函数返回INFEASTABLE；将L沿着单链表向后移动，直到找到待查找后继的元素，若该元素为单链表的最后一个元素，则返回INFEASTABLE，若该元素不为单链表的最后一个元素，则将它的下一个元素赋值给next指向的内存空间，并返回OK，若未在单链表中找到该元素，则返回INFEASTABLE。该函数包含一个循环结构。若单链表为空，则循环不执行，若待查找后继的元素为单链表的第一个元素，则循环执行一次，此时函数耗时最短；若待查找元素为单链表的最后一个元素或者该元素不在单链表中，则循环执行n次，此时函数耗时最长。因此循环平均执行n/2次，故函数的时间复杂度为O(n)。

10)插入元素(ListInsert)该函数接受一个LNode类型的指针 L，待插入元素的位置i，以及待插入的元素e。首先将L逐个后移到第i-1 个元素的位置，若该位置没有元素，则表明i超出了可插入的范围，函数返回INFEASTABLE，否则函数新建一个节点存储待插入的元素e，若创建节点失败，则说明内存不足，无法分配足够的空间，此时返回OVERFLOW，若成功创建节点，则将该节点指向第i个节点并将第i-1个节点指向该节点，这样函数成功插入一个元素到单链表中，返回OK。该函数包含一个循环结构，循环用于将L移动到第i-1个元素，故循环执行i-1次(i为待插入元素的位置)，故函数的时间复杂度为O(n)。

11)删除元素(ListDelete)该函数接受一个LNode类型的指针L，待删除元素的位置i，以及保存待删除元素的位置地址e。首先将L逐个后移到第i-1个元素的位置，若第i个位置没有元素，则表明待删除元素不存在，函数返回ERROR，否则设置指针now指向第i个元素即待删除节点，将第i-1个节点指向第i+1个节点，并将now指向的内存空间释放，这样就将第i个元素从单链表中删除，函数返回OK。该函数包含一个循环结构，该循环用于将L移动到第i-1个位置，故循环执行i-1次(i为待删除元素的位置)，故函数的时间复杂度为 O(n)。

12) 遍历单链表(ListTrabverse)该函数接受一个LNode类型的指针L。函数将L逐个后移并依次打印每一个节点的数据域，直到L指向NULL，表明单链表全部读取完毕，函数返回OK。该函数包含一个循环结构，该循环将L从单链表的首部逐个的移动到尾部，故循环执行n次，因此函数的时间复杂度为O(n)。

## 2.2.4 多线性表管理设计

本次实验中采用顺式储存的结构来管理各个线性表，每个链表有一个名字，实现如下（结构如图）：

typedef struct{ //线性表的集合类型定义

struct { char name[30];

LinkList L;

} elem[10];

int length;

}LISTS;

## 2.3 系统实现

## 2.3.1 程序实现的环境及程序开发工具与开发语言

本人未使用集成开发环境(IDE),代码由代码编辑器VScode编辑，代码使用c++编写，在cmd命令行下使用mingw编译器g++进行编译，由于g++默认的编码为GBK，故需要改变编码为UTF-8，用于显示中文，编译命令如下：

**> g++ -finput-charset=UTF-8 -fexec-charset=GBK -o demo main.cpp**

获得可执行文件（演示菜单）demo.exe。

## 2.3.2 文件组织形式及代码实现

参考工程标准，将一些定义以及需要使用的头文件放在了同目录的list2.h文件中。并将测试文件放在同目录下（如图2-3所示）。

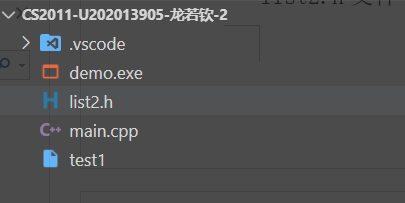


图2-3文件组织图

系统代码实现见附件（代码使用UTF-8编码）。

## 2.4 系统测试

## 2.4.1菜单演示

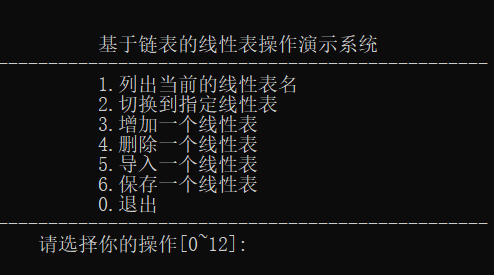


图2-4 一级菜单

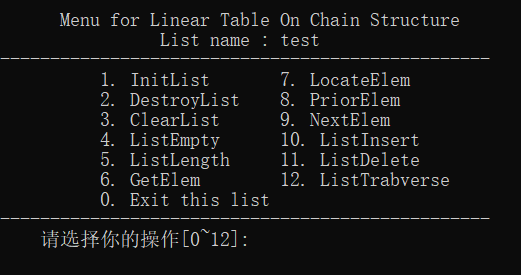


图2-5 二级菜单

## 2.4.2各函数测试

本测试中，对12个基本操作以及文件输入输出功能进行测试。测试时创建1个单链表作为待操作的单链表。

1)初始化表：函数名称是InitaList(L),初始条件是线性表L不存在或已存在,操作结果是构造一个空的线性表。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表不存在（未初始化） | 提示:  单链表创建成功 |  |
| 错误输入 | 线性表已存在。 | 提示：  单链表已存在 |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义创建了对应的线性表，符合实验要求。

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表已存在 | 提示:  线性表销毁成功 |  |
| 错误输入 | 线性表不存在。 | 提示：  线性表销毁失败。 |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义销毁了对应的线性表，符合实验要求。

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表已存在 | 提示:  线性表清空成功 |  |
| 错误输入 | 线性表不存在。 | 提示：  线性表不存在。 |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义清空了对应的线性表，符合实验要求。

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表已存在且为空 | 提示:  线性表为空！ |  |
| 正确输入 | 线性表已存在且不为空 | 提示：  线性表不为空！。 |  |
| 错误输入 | 线性表不存在 | 提示：线性表不存在！ |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义判定线性表是否为空，符合实验要求。

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表已存在且为空 | 提示:  线性表长度为0. |  |
| 正确输入 | 线性表已存在且不为空 | 给出线性表的表长。 |  |
| 错误输入 | 线性表不存在 | 提示：线性表不存在！ |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义给出了线性表的表长，符合实验要求。

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表为[3,2,4,1]  查询3号元素 | 给出对应元素的值 |  |
| 错误输入 | 线性表为[3,2,4,1]  查询5号元素或查询0号元素。 | 提示：输入不合法! |  |
| 错误输入 | 线性表不存在 | 提示：线性表不存在！ |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义给出了对应序号的元素的值，对于错误输入也能处理，符合实验要求。

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表为[3,2,4,1]  查询元素4 | 给出元素对应序号 |  |
| 错误输入 | 线性表为[3,2,4,1]  查询元素5 | 提示：该元素不存在 |  |
| 错误输入 | 线性表不存在 | 提示：线性表不存在！ |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义给出了对应值的元素的序号，对错误输入也能够处理，符合实验要求。

8)获得前驱：函数名称是PriorElem(L,e,pre)；初始条件是线性表L已存在；操作结果是若e是L的数据元素，且不是第一个，则用pre返回它的前驱，否则操作失败，pre无定义。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表为[3,2,4,1]  查询元素4的前驱 | 给出元素4的前驱元素 |  |
| 错误输入 | 线性表为[3,2,4,1]  查询元素5的前驱 | 提示：该元素无前驱 |  |
| 错误输入 | 线性表为[3,2,4,1]  查询元素3的前驱 | 提示：该元素无前驱 |  |
| 错误输入 | 线性表不存在 | 提示：线性表不存在！ |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义给出了输入元素的前驱元素，对错误输入也能够处理，符合实验要求。

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表为[3,2,4,1]  查询元素4的后继 | 给出元素4的后继元素 |  |
| 错误输入 | 线性表为[3,2,4,1]  查询元素5的后继 | 提示：该元素无后继 |  |
| 错误输入 | 线性表为[3,2,4,1]  查询元素1的后继 | 提示：该元素无后继 |  |
| 错误输入 | 线性表不存在 | 提示：线性表不存在！ |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义给出了输入元素的后继元素，对错误输入也能够处理，符合实验要求。

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表为空，插入2到位置1 | 提示：插入成功！ |  |
| 错误输入 | 线性表为[2],插入3到位置3或位置0. | 提示：插入位置不正确. |  |
| 错误输入 | 线性表不存在 | 提示：线性表不存在！ |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义插入了元素，对错误输入也能够处理，符合实验要求。

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表为[3,2,4,1]  删除位置为3的元素 | 给出删除元素的值 |  |
| 错误输入 | 线性表为[3,2,4,1]  删除位置为5的元素或位置为0的元素。 | 提示：输入位置不正确 |  |
| 错误输入 | 线性表不存在 | 提示：线性表不存在！ |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义删除了元素，对错误输入也能够处理，符合实验要求。

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 正常输入 | 线性表为[3,2,4,1] | 打印线性表 |  |
| 正确输入 | 线性表为空表 | 提示：线性表为空表 |  |
| 错误输入 | 线性表不存在 | 提示：线性表不存在！ |  |

测试结论：综合上述测试，函数对线性表存在于不存在的情况都可以处理，按照定义打印了线性表的元素，对错误输入也能够处理，符合实验要求。

测试小结：按照上述测试，基于链式存储结构的线性表的基本运算都已经实现，且对特殊输入都进行了处理，故函数实现符合要求。

## 2.5 实验小结

在本次实验中，我沿用了上一次实验的演示系统，只对相应的函数接口做了修改，并把函数实现替换为链式实现，由于函数定义基本相同，返回值已经约定，故在使用函数时不必关心内部结构，类似于“封装”。

在本次实验报告的编写上，我更换了一种形式，采用了更精简的编写方式。

# 3 基于二叉链表的二叉树实现

## 3.1 问题描述

采用二叉链表作为二叉树的物理结构，实现二叉树的基本运算。其中ElemType为数据元素的类型名，具体含义可自行定义，但要求二叉树结点类型为结构型，至少包含二个部分，一个是能唯一标识一个结点的关键字（类似于学号或职工号），另一个是其它部分。其它有关类型和常量的定义和引用详见文献[1]的p10。

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

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

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

## 3.1.1 二叉树基本运算定义

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

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

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

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

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

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

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

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

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

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

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

特殊情况，c插入作为根结点？可以考虑LR为-1时，作为根结点插入，原根结点作为c的右子树。

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

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

注：前序、中序和后序三种遍历算法，要求至少一个用非递归算法实现。

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

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

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

## 3.2 系统设计

## 3.2.1 总体流程框架

演示系统采用二级菜单，在一级菜单中可以查看当前已经创建的二叉树，并通过切换到指定二叉树进入二级菜单。（主要功能如图3-1所示）

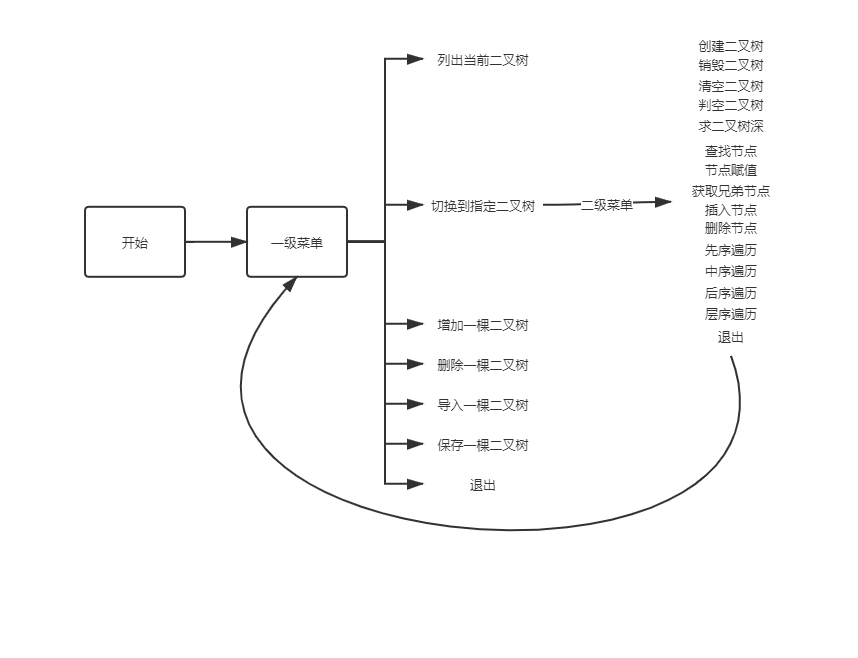


图3-1演示系统流程图

## 3.2.2 定义常量、数据类型及数据结构

1)定义常量。定义常量用于指定函数的返回值，用来判断函数执行的情况。 定义的常量有：TRUE(1), FALSE(0), OK(1), ERROR(0), INFEASTABLE(-1), OVERFLOW(-2)。表示真、假、成功、失败、溢出等状态。

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

2)定义数据元素的类型。此处定义了部分函数返回值的类型 status 以及单 链表中单个元素的类型 ElemType。本实现中将两者均定义为 int 类型。

typedef int status;//返回状态码类型定义

typedef int ElemType; //数据元素类型定义

3)定义二叉链表结点的数据结构。二叉链表结点结构体中包含三个元素，第一个元素为节点的数据域data，为TElemType 类型，第二,三个元素为二叉链表的指针域lchild,rchild，为二叉链表结构体类型的指针。将该结构体类型定义为BiTNode。

## 3.2.3 多二叉树管理设计

可采用线性表的方式管理多个二叉树，线性表中的每个数据元素为一个二叉树的基本属性，包含有二叉树的名称。

typedef struct{ //二叉树的集合类型定义

struct { char name[30];

BiTree T;

} elem[10];

int nums;

int Forestsize;

}Forest;

## 3.3 系统实现

## 3.3.1 程序实现的环境及程序开发工具与开发语言

本人未使用集成开发环境(IDE),代码由代码编辑器VScode编辑，代码使用c++编写，在cmd命令行下使用mingw编译器g++进行编译，由于g++默认的编码为GBK，故需要改变编码为UTF-8，用于显示中文，编译命令如下：

**> g++ -finput-charset=UTF-8 -fexec-charset=GBK -o demo main.cpp**

获得可执行文件（演示菜单）demo.exe。

## 3.3.2 文件组织形式及代码实现

参考工程标准，将一些定义以及需要使用的头文件放在了同目录的def.h文件中，同时把函数实现放在了def.h中，并将测试文件放在同目录下（如图3-2所示）。



图3-2文件组织图

系统代码实现见附件（代码使用UTF-8编码）。

## 3.4 系统测试

## 3.4.1 菜单演示

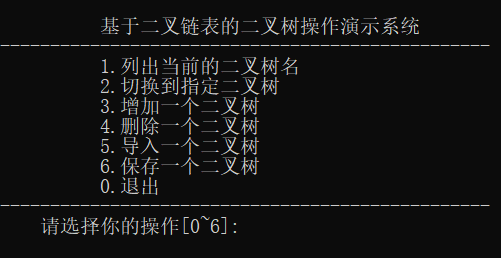


图3-3一级菜单

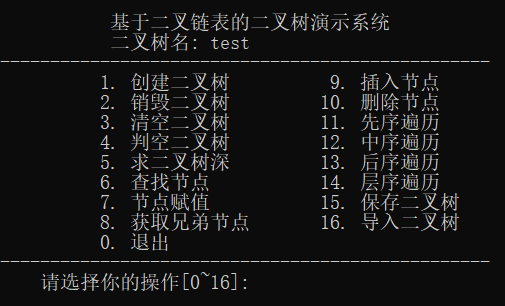


图3-4 二级菜单

## 3.4.2 各函数测试

测试用例：

树：

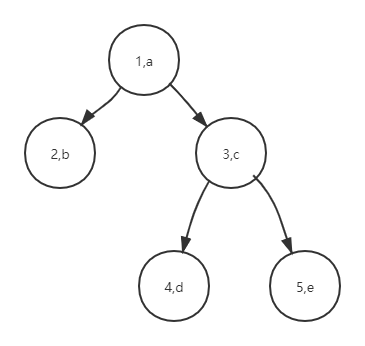


图3-5 测试用树

（1）创建二叉树： CreateBiTree(T,definition)

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 建树序列为1 a 2 b 0 null 0 null 3 c 4 d 0 null 0 null 5 e 0 null 0 null -1 null | 建树成功 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\DM5M60AU~}XXXS{S]KP~(0C.png |

⑵销毁二叉树：函数名称是DestroyBiTree(T)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1 | 销毁成功 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\(9GGJ@DXW(0){R]X3@[Z750.png |

⑶清空二叉树：函数名称是ClearBiTree (T)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1 | 清空成功 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\X{)@QX6OO9D81_L}F`4BAF3.png |
| 2 | 空树 | 清空成功 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\X{)@QX6OO9D81_L}F`4BAF3.png |

⑷判定空二叉树：函数名称是BiTreeEmpty(T)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1 | 树不为空 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\65])EUK`K[%(1ZIJ]VX}226.png |
| 2 | 空树 | 树为空 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\GFEGTI4YAX87MUE_Z9L%UW3.png |

⑸求二叉树深度：函数名称是BiTreeDepth(T)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1 | 树深为3 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\O)4X87KC9V$_MV}Y@JT3J[F.png |
| 2 | 空树 | 树深为0 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\2V%NPG2T63RS3{9LJ8D42%2.png |

⑹查找结点：函数名称是LocateNode(T,e)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1，e为3 | 给出结点数据 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\@L9U~5(T7]DK9H`UM@)2W6F.png |
| 2 | 树1，e为6 | 无该结点 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\@]4[2F1P4X~CC$X2[OV(VXP.png |

⑺结点赋值：函数名称是Assign(T,e,value)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1，将e为3结点赋值为（3,6，box） | 赋值成功 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\PAB1AEHUSY0XN6D3MW$EO}O.png |
| 2 | 树1，将e为6结点赋值为（6,6，box） | 无该结点 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\%KH[(8})[FAEXDV`FP7U@HW.png |

⑻获得兄弟结点：函数名称是GetSibling(T,e)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1，求关键字为1的兄弟节点 | 无兄弟节点 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\HYMUCL_DI8YN%VUF8VU`O0Y.png |
| 2 | 树1，求关键字为4的兄弟 | 给出其兄弟节点 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\MBQWOXYF@6NC$P(ZGYG)OUO.png |

⑼插入结点：函数名称是InsertNode(T,e,LR,c)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1，向2左插入节点（6,6，f） | 插入成功 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\IG8(E9B@``XIQ`XK_2][LPF.png  C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\6Q8F}(%V[~HQAGGP7KE40]G.png |
| 2 | 上树，插入（7，7，root）为根节点 | 插入成功 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\@JV40E%]07(KPXY{AG3IYIS.png  C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\U76795RHI8Q((G{`]KUM%]Y.png |

⑽删除结点：函数名称是DeleteNode(T,e)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1，删除结点（2,2，b） | 删除成功 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\O5F)O8ALQ0CV[S1%BU`XMQG.png |
| 2 | 树1，删除结点（1,1，a） | 删除成功 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\(XG46TDXIUC$V%8HHGX7)TO.png |

⑾前序遍历：函数名称是PreOrderTraverse(T,Visit())（非递归）

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1 | 打印先序序列 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\2`Z`(U7P4`_~@1}O}G5OV~5.png |

⑿中序遍历：函数名称是InOrderTraverse(T,Visit))

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1 | 打印中序序列 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\T%B[TW66EZZ0{{VBL@YV799.png |

⒀后序遍历：函数名称是PostOrderTraverse(T,Visit))

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1 | 打印后序序列 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\S_@(Z4T~W9AV@1(XI%6([EW.png |

⒁按层遍历：函数名称是LevelOrderTraverse(T,Visit))

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 树1 | 打印层序遍历序列 | C:\Users\12903\AppData\Roaming\Tencent\Users\1290363659\QQ\WinTemp\RichOle\CMYHX]3$2H%$8WY6FA[U}KA.png |

测试小结：按照上述测试，基于二叉链表存储的二叉树的基本运算都已经实现，且对特殊输入都进行了处理，故函数实现符合要求。

## 3.5 实验小结

本次实验较前两次实验来讲难度有了比较大的提高。主要在打印上花费了很多时间，并且最后由于时间关系也没有很好的解决。

无法以一种很好的方式把二叉树纵向打印出来，而无法把二叉树打印出来的话，在调试其他函数的时候就会很麻烦，最后由于时间关系选择了不打印二叉树，采用前序中序来观察。

# 4 基于邻接表的图实现

## 4.1 问题描述

采用邻接表作为图的物理结构，实现图的基本运算，可任选无向图、有向图、无向网和有向网这四种图中的一种实现。其中ElemType为数据元素的类型名，具体含义可自行定义，但要求顶点类型为结构型，至少包含二个部分，一个是能唯一标识一个顶点的关键字（类似于学号或职工号），另一个是其它部分。其它有关类型和常量的定义和引用详见文献[1]的p10。

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

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

演示系统可选择实现多个图管理。

## 4.1.1 图基本运算定义

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

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

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

⑵销毁图：函数名称是DestroyGraph(G)；初始条件图G已存在；操作结果是销毁图G。

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

⑷顶点赋值：函数名称是PutVex (G,u,value)；初始条件是图G存在，u是和G中顶点关键字类型相同的给定值；操作结果是对关键字为u的顶点赋值value。

⑸获得第一邻接点：函数名称是FirstAdjVex(G, u)；初始条件是图G存在，u是G中顶点的位序；操作结果是返回u对应顶点的第一个邻接顶点位序，如果u的顶点没有邻接顶点，否则返回其它表示“不存在”的信息。

⑹获得下一邻接点：函数名称是NextAdjVex(G, v, w)；初始条件是图G存在，v和w是G中两个顶点的位序，v对应G的一个顶点,w对应v的邻接顶点；操作结果是返回v的（相对于w）下一个邻接顶点的位序，如果w是最后一个邻接顶点，返回其它表示“不存在”的信息。

⑺插入顶点：函数名称是InsertVex(G,v)；初始条件是图G存在，v和G中的顶点具有相同特征；操作结果是在图G中增加新顶点v。（在这里也保持顶点关键字的唯一性）

⑻删除顶点：函数名称是DeleteVex(G,v)；初始条件是图G存在，v是和G中顶点关键字类型相同的给定值；操作结果是在图G中删除关键字v对应的顶点以及相关的弧。

⑼插入弧：函数名称是InsertArc(G,v,w)；初始条件是图G存在，v、w是和G中顶点关键字类型相同的给定值；操作结果是在图G中增加弧<v,w>，如果图G是无向图，还需要增加<w,v>。

⑽删除弧：函数名称是DeleteArc(G,v,w)；初始条件是图G存在，v、w是和G中顶点关键字类型相同的给定值；操作结果是在图G中删除弧<v,w>，如果图G是无向图，还需要删除<w,v>。

⑾深度优先搜索遍历：函数名称是DFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

⑿广深度优先搜索遍历：函数名称是BFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

## 4.2 系统设计

## 4.2.1 总体流程框架

演示系统采用二级菜单，在一级菜单中可以查看当前已经创建的图，并通过切换到指定图进入二级菜单。（主要功能如图4-1所示）

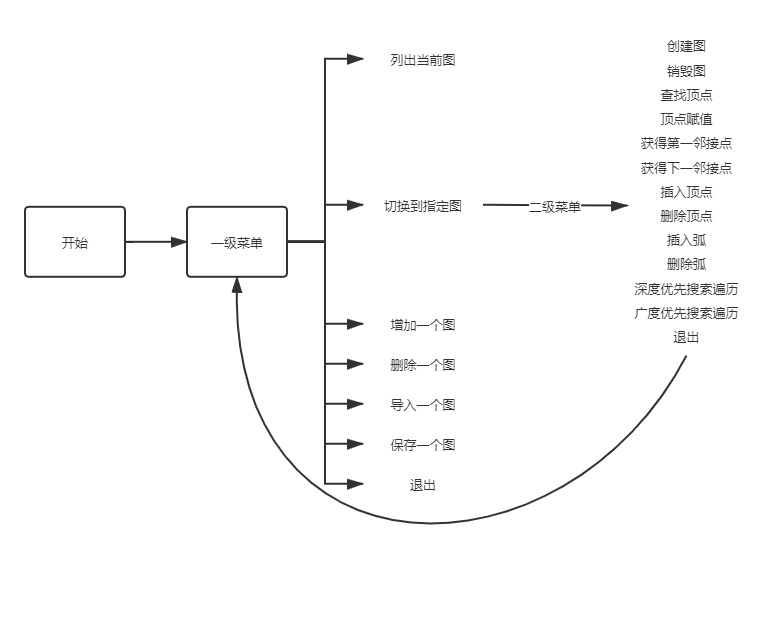


图4-1演示系统流程图

## 4.2.2 定义常量、数据类型及数据结构

1)定义常量。定义常量用于指定函数的返回值，用来判断函数执行的情况。 定义的常量有：TRUE(1), FALSE(0), OK(1), ERROR(0), INFEASTABLE(-1), OVERFLOW(-2)。表示真、假、成功、失败、溢出等状态。

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 20

2)定义数据元素的类型。此处定义了部分函数返回值的类型 status 以及图中关键字的类型 KeyType。本实现中将两者均定义为 int 类型。同时还定义了图的类型GraphKind。

typedef int status;//返回状态码类型定义

typedef int KeyType;//关键字类型

typedef enum {DG,DN,UDG,UDN} GraphKind;//图类型

3)定义图的数据结构。包括顶点结构类型，表节点类型，头结点及其数组类型定义，最后定义邻接表的类型。这样可以通过说明语句：ALGraph G; 定义结构变量G，使得G可以表示一个图。

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;

## 4.2.3 多图管理设计

可采用线性表的方式管理多个图，线性表中的每个数据元素为一个图的基本属性，包含有图的名称。

typedef struct{ //图的集合类型定义

struct { char name[30];

ALGraph G;

} elem[10];

int nums;

int Graphs\_size;

}Graphs;

## 4.3 系统实现

## 4.3.1 程序实现的环境及程序开发工具与开发语言

本人未使用集成开发环境(IDE),代码由代码编辑器VScode编辑，代码使用c++编写，在cmd命令行下使用mingw编译器g++进行编译，由于g++默认的编码为GBK，故需要改变编码为UTF-8，用于显示中文，编译命令如下：

**> g++ -finput-charset=UTF-8 -fexec-charset=GBK -o demo main.cpp**

获得可执行文件（演示菜单）demo.exe。

## 4.3.2 文件组织形式及代码实现

参考工程标准，将一些定义以及需要使用的头文件放在了同目录的def.h文件中，同时把函数实现放在了def.h中，并将测试文件放在同目录下（如图4-2所示）。



图4-2文件组织图

系统代码实现见附件（代码使用UTF-8编码）。

## 4.4 系统测试

## 4.4.1 菜单演示

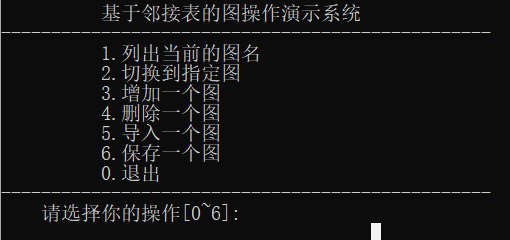


图4-3一级菜单

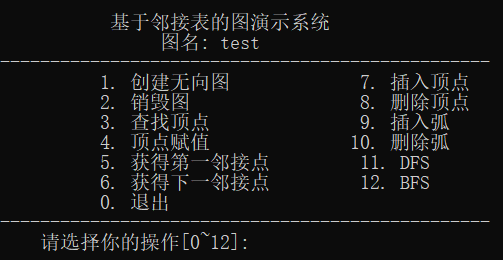


图4-4 二级菜单

## 4.4.2 各函数测试

测试用例：图（如图4-5）

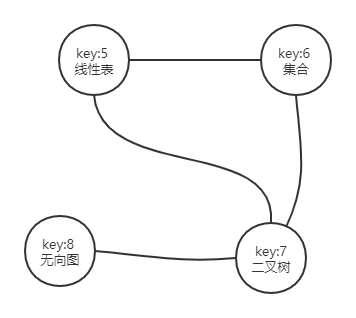


图4-5 测试用图

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 测试图建图序列 | 创建成功 |  |

⑵销毁图：初始条件图G已存在；操作结果是销毁图G。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 测试图 | 销毁成功 |  |

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 测试图，查找7 | 位序2 |  |
| 2 | 测试图，查找9 | 查找失败 |  |

⑷顶点赋值：初始条件是图G存在，u是和G中顶点关键字类型相同的给定值；操作结果是对关键字为u的顶点赋值value。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 测试图，更改6 集合，为6 链表，使用dfs查看结果 | 修改成功 |  |

⑸获得第一邻接点：初始条件是图G存在，u是G中顶点的位序；操作结果是返回u对应顶点的第一个邻接顶点位序，如果u的顶点没有邻接顶点，否则返回其它表示“不存在”的信息。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 测试图，获得5的第一邻接点 | 7 |  |
| 2 | 测试图，获得9的第一邻接点 | 查找失败 |  |

⑹获得下一邻接点：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 测试图，获得5的邻接点7的下一邻接点 | 6 |  |
| 2 | 测试图，获得8的邻接点7的下一邻接点 | 查找失败 |  |

⑺插入顶点：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 测试图，插入顶点9 红黑树 | 插入成功 |  |
| 2 | 测试图，插入顶点5 哈希表 | 插入失败 |  |

⑻删除顶点：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 上图，删除顶点9 红黑树 | 删除成功 |  |
| 2 | 测试图，删除顶点10 | 删除失败 |  |

⑼插入弧：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 测试图，插入（5,8） | 插入成功 |  |
| 2 | 测试图，插入（5,10） | 插入失败 |  |

⑽删除弧：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 上图，删除（5,8） | 删除成功 |  |
| 2 | 测试图，删除（5,10） | 删除失败 |  |

⑾深度优先搜索遍历：函数名称是DFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行深度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 测试图 | Dfs序列 |  |

⑿广深度优先搜索遍历：函数名称是BFSTraverse(G,visit())；初始条件是图G存在；操作结果是图G进行广度优先搜索遍历，依次对图中的每一个顶点使用函数visit访问一次，且仅访问一次。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入 | 理论输出 | 运行结果 |
| 1 | 测试图 | Bfs序列 |  |

测试小结：按照上述测试，基于邻接表存储的图的基本运算都已经实现，且对特殊输入都进行了处理，故函数实现符合要求。

## 4.5 实验小结

由于这次实验选择的是无向图，所以实验难度其实比二叉树还要低，实验中遇到的最大困难也就是对于各种异常输入的处理，主体部分基本上都很顺利。

# 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

/\*---------def.h ---------\*/

#ifndef DRF\_H

#define DRF\_H

#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

#define LIST\_INIT\_SIZE 3

#define LISTINCREMENT 10

typedef int status;

typedef int ElemType; //数据元素类型定义

typedef int ElemType;

typedef struct{ //顺序表（顺序结构）的定义

ElemType \* elem;

int length;

int listsize;

}SqList;

typedef struct{ //线性表的集合类型定义

struct { char name[30];

SqList L;

} elem[10];

int length;

}LISTS;

#endif

/\*---------main.cpp---------\*/

#include "list2.h"

void menu2(LinkList &L,int op, char lname[]);

int main(){

LISTS lists;

memset(&lists, 0, sizeof(LISTS));//设置为空

lists.length = 0;

int op=1;

while(op){

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

printf(" 基于链表的线性表操作演示系统 \n");

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

printf(" 1.列出当前的线性表名 \n");

printf(" 2.切换到指定线性表 \n");

printf(" 3.增加一个线性表 \n");

printf(" 4.删除一个线性表 \n");

printf(" 5.导入一个线性表 \n");

printf(" 6.保存一个线性表 \n");

printf(" 0.退出\n");

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

printf(" 请选择你的操作[0~12]:");

scanf("%d",&op);

switch (op)

{

case 1:

//列出线性表名

if(0 == LISTSTrabverse(lists))

printf("当前无线性表！");

printf("按任意键继续...\n");

getchar();getchar();

break;

case 2:

{

//切换到指定线性表

int index;

printf("请输入需要切换到的线性表序号:");

scanf("%d",&index);

if(index <= lists.length){

menu2( lists.elem[index-1].L, 1,lists.elem[index-1].name);

}

else printf("输入不正确！\n");

getchar();getchar();

break;

}

case 3:{

//插入一个线性表

char name[30];

getchar();

printf("请输入增加表的名称:");

scanf("%s",name);

int con = AddList(lists, name);

printf("增加成功！\n");

getchar();getchar();

break;

}

case 4:{

//删除一个线性表

char name[30];

getchar();

printf("请输入删除表的文件名:");

scanf("%s",name);

int con = RemoveList(lists, name);

printf("删除成功！\n");

getchar();getchar();

break;

}

case 5:{

//导入一个线性表

char name[30];

getchar();

printf("请输入导入文件的文件名:");

scanf("%s",name);getchar();

AddList(lists, name);

int con = LoadList(lists.elem[lists.length-1].L, name);

if(con == OK){

printf("导入成功！\n");

}

else{

printf("导入失败！\n");

}

getchar();getchar();

break;

}

case 6:{

//保存一个线性表

char name[30];

getchar();

printf("请输入保存的线性表名:");

scanf("%s",name);getchar();

int index = LocateList(lists, name);

int con = SaveList(lists.elem[index-1].L, name);

if(con == OK){

printf("保存成功！\n");

}

getchar();getchar();

break;

}

case 0:

break;

default:

break;

}

}

printf("欢迎下次再使用本系统！\n");

return 0;

}//end of main()

void menu2(LinkList &L,int op, char lname[]){

while(op){

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

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

printf(" List name : %s\n",lname);

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

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

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

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

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

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

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

printf(" 0. Exit this list\n");

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

printf(" 请选择你的操作[0~12]:");

scanf("%d",&op);

switch(op){

case 1:

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

else printf("单链表已存在！\n");

getchar();getchar();

break;

case 2:

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

else printf("线性表销毁失败！\n");

getchar();getchar();

break;

case 3:

if(OK == ClearList(L)) printf("线性表清空成功！\n");

else printf("线性表不存在！\n");

getchar();getchar();

break;

case 4:{

int c = ListEmpty(L);

if(c == INFEASIBLE) printf("线性表不存在！\n");

else if(c == TRUE) printf("线性表为空！\n");

else printf("线性表不为空！\n");

getchar();getchar();

break;

}

case 5:

{

int l = ListLength(L);

if(l == INFEASIBLE) printf("线性表不存在！\n");

else printf("线性表长度为%d",l);

getchar();getchar();

break;}

case 6:

{

int tof;

printf("请问您要查询哪个元素：");

scanf("%d", &tof);

ElemType e;

status con = GetElem(L, tof, e);

if(con == INFEASIBLE) printf("线性表不存在！\n");

else if(con == OK) printf("线性表L的第%d个元素是%d\n", tof, e);

else printf("输入不合法！\n");

getchar();getchar();

break;}

case 7:

{

ElemType e;

printf("请问您要查询元素的值是：");

scanf("%d", &e);

status con = LocateElem(L, e);

if(con == INFEASIBLE) printf("线性表不存在！\n");

else if(con == ERROR) printf("该元素不存在！\n");

else printf("该元素在线性表L中序号是%d\n",con);

getchar();getchar();

break;}

case 8:

{

ElemType e, pre;

printf("请问当前元素的值是：");

scanf("%d", &e);

status con = PriorElem(L, e, pre);

if(INFEASIBLE == con) printf("线性表不存在！\n");

else if(con == ERROR) printf("该元素没有前驱！\n");

else printf("线性表L中该元素的前驱是%d\n", pre);

getchar();getchar();

break;}

case 9:

{

ElemType e, next;

printf("请问当前元素的值是：");

scanf("%d", &e);

status con = NextElem(L, e, next);

if(INFEASIBLE == con) printf("线性表不存在！\n");

else if(con == ERROR) printf("该元素没有后继！\n");

else printf("线性表L中该元素的后继是%d\n", next);

getchar();getchar();

break;}

case 10:

{

ElemType e;

int pos = 0;

printf("请问要插入元素的值是(仅支持数字)：");

scanf("%d", &e);

printf("请问要插入元素的位置是：");

scanf("%d",&pos);

status con = ListInsert(L, pos, e);

if(con == INFEASIBLE) printf("线性表不存在！\n");

else if(con == ERROR) printf("插入位置不正确\n");

else printf("插入成功\n");

getchar();getchar();

break;}

case 11:

{

ElemType e;

int pos = 0;

printf("请问要删除元素的位置是：");

scanf("%d",&pos);

status con = ListDelete(L, pos, e);

if(con == INFEASIBLE) printf("线性表不存在！\n");

else if(con == ERROR) printf("输入位置不正确\n");

else printf("删除成功, 删除的值为%d\n", e);

getchar();getchar();

break;}

case 12:

{

status con =ListTraverse(L);

if(con == INFEASIBLE){

printf("线性表不存在！\n");

}

else if(con == 0)

printf("线性表是空表！\n");

getchar();getchar();

break;}

case 0:

break;

}//end of switch

}//end of while

}

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 != NULL)return INFEASIBLE;

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

if( n == NULL)return INFEASIBLE;

L= n;

n->next =NULL;

return OK;

}

status DestroyList(LinkList &L)

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

{

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

free(L);

L = NULL;

while(now != NULL){

LinkList last = now;

now = now->next;

free(last);

}

return OK;

}

status ClearList(LinkList &L)

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

{

if(L == NULL)return INFEASIBLE;

LinkList now = L->next;

L->next = NULL;

while(now!=NULL){

LinkList last = now;

now = now->next;

free(last);

}

return OK;

}

status ListEmpty(LinkList L)

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

{

if(L == NULL)return INFEASIBLE;

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

return FALSE;

}

int ListLength(LinkList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

if(L == NULL) return INFEASIBLE;

int length = 0;

LinkList now = L->next;

while(now != NULL){

now = now->next;

length++;

}

return length;

}

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(i<1)return ERROR;

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

int n = 0;

while(now != NULL){

n++;

if(n == i){

e = now ->data;

return OK;

}

now =now->next;

}

return ERROR;

}

status LocateElem(LinkList L,ElemType e)

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

{

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

int i = 0;

while(now != NULL){

i++;

if(now->data == e){

return i;

}

now =now->next;

}

return ERROR;

}

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

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

{

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

LinkList last;

int n = 0;

while(now != NULL){

n++;

if(now->data == e){

if(n == 1){

return ERROR;

}

pre = last->data;

return OK;

}

last = now;

now =now->next;

}

return ERROR;

}

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

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

{

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

int n = 0;

while(now != NULL){

n++;

if(now->data == e){

if(now->next == NULL){

return ERROR;

}

next = now->next->data;

return OK;

}

now =now->next;

}

return ERROR;

}

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

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

{

// 请在这里补充代码，完成本关任务

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

if(i<1)return ERROR;

if(L == NULL) return INFEASIBLE;

LinkList now = L;

int x;

for(x= 1 ; (x<i && now!=NULL); now=now->next,x++);

if(i==1)now = L;

if( x == i && now == NULL){

return ERROR;

}

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

if(New == NULL) return OVERFLOW;

New->data = e;

New->next = now->next;

now->next = New;

return OK;

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

}

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

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

{

int length = ListLength(L);

if( i<=0 || i>length)return ERROR;

if(L == NULL) return INFEASIBLE;

LinkList pre = L;

int x;

for(x= 1 ; (x<i && pre!=NULL); pre=pre->next,x++);

if(i==1)pre = L;

if( x == i && pre == NULL){

return ERROR;

}

LinkList now = pre ->next;

pre->next = now ->next;

e = now->data;

free(now);

return OK;

}

status ListTraverse(LinkList L)

// 如果线性表L存在，依次显示线性表中的元素，返回OK；如果线性表L不存在，返回INFEASIBLE。

{

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

int i = 0;

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

printf("%4s %16s\n","元素序号", "元素值");

while(now != NULL){

printf("%4d %16d\n",1+i, now->data);

i++;

now = now->next;

}

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

return i;

}

status SaveList(LinkList L,char FileName[])

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

{

if(L == NULL)return INFEASIBLE;

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

LinkList now = L->next;

while(now != NULL){

fprintf(fp, "%d\n",now->data);

now = now->next;

}

fclose(fp);

return OK;

}

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 != NULL)return INFEASIBLE;

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

L->next =NULL;

LinkList now = L;

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

ElemType e;

while(fscanf(fp, "%d", &e) != EOF){

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

New->data = e;

New->next =NULL;

now->next = New;

now=New;

}

fclose(fp);

return OK;

}

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

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

{

int i = Lists.length++;

int j=0;

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

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

j++;

}

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

Lists.elem[i].L = NULL;

Lists.elem[i].L = 0;

return 0;

}

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

// Lists中删除一个名称为ListName的线性表

{

int ok = 0;

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

int tag = 1;

int c=0;

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

if(Lists.elem[i].name[c] != ListName[c]){

tag = 0;

break;

}

c++;

}

if(tag){

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

Lists.length--;

if( i == Lists.length){

break;

}

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

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

}

ok =1;

break;

}

}

if(ok){

return OK;

}

return ERROR;

}

int LocateList(LISTS Lists,char ListName[])

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

{

int ok=0;

int i=0;

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

int tag = 1;

int c=0;

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

if(Lists.elem[i].name[c] != ListName[c]){

tag = 0;

break;

}

c++;

}

if(tag){

ok=1;

break;

}

}

if(ok){

return i+1;

}

return 0;

}

status LISTSTrabverse(LISTS &Lists){//遍历线性表集合，并打印

int i;

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

for(i=0;i<Lists.length;i++) printf("%4d: %s \n", i+1, Lists.elem[i].name);

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

return Lists.length;

}

# 附录B 基于链式存储结构线性表实现的源程序

/\*---------list2.h ---------\*/

#ifndef LIST2

#define LIST2

#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

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef int status;//返回状态码类型定义

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;

struct LNode \*next;

}LNode,\*LinkList;

typedef struct{ //线性表的集合类型定义

struct { char name[30];

LinkList L;

} elem[10];

int length;

}LISTS;

//以下为链表操作函数

status InitList(LinkList &L);

status DestroyList(LinkList &L);

status ClearList(LinkList &L);

status ListEmpty(LinkList L);

int ListLength(LinkList L);

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

status LocateElem(LinkList L,ElemType e);

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

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

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

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

status ListTraverse(LinkList L);

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

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

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

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

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

status LISTSTrabverse(LISTS &Lists);

#endif

/\*---------main.cpp---------\*/

#include "list2.h"

void menu2(LinkList &L,int op, char lname[]);

int main(){

LISTS lists;

memset(&lists, 0, sizeof(LISTS));//设置为空

lists.length = 0;

int op=1;

while(op){

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

printf(" 基于链表的线性表操作演示系统 \n");

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

printf(" 1.列出当前的线性表名 \n");

printf(" 2.切换到指定线性表 \n");

printf(" 3.增加一个线性表 \n");

printf(" 4.删除一个线性表 \n");

printf(" 5.导入一个线性表 \n");

printf(" 6.保存一个线性表 \n");

printf(" 0.退出\n");

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

printf(" 请选择你的操作[0~12]:");

scanf("%d",&op);

switch (op)

{

case 1:

//列出线性表名

if(0 == LISTSTrabverse(lists))

printf("当前无线性表！");

printf("按任意键继续...\n");

getchar();getchar();

break;

case 2:

{

//切换到指定线性表

int index;

printf("请输入需要切换到的线性表序号:");

scanf("%d",&index);

if(index <= lists.length){

menu2( lists.elem[index-1].L, 1,lists.elem[index-1].name);

}

else printf("输入不正确！\n");

getchar();getchar();

break;

}

case 3:{

//插入一个线性表

char name[30];

getchar();

printf("请输入增加表的名称:");

scanf("%s",name);

int con = AddList(lists, name);

printf("增加成功！\n");

getchar();getchar();

break;

}

case 4:{

//删除一个线性表

char name[30];

getchar();

printf("请输入删除表的文件名:");

scanf("%s",name);

int con = RemoveList(lists, name);

printf("删除成功！\n");

getchar();getchar();

break;

}

case 5:{

//导入一个线性表

char name[30];

getchar();

printf("请输入导入文件的文件名:");

scanf("%s",name);getchar();

AddList(lists, name);

int con = LoadList(lists.elem[lists.length-1].L, name);

if(con == OK){

printf("导入成功！\n");

}

else{

printf("导入失败！\n");

}

getchar();getchar();

break;

}

case 6:{

//保存一个线性表

char name[30];

getchar();

printf("请输入保存的线性表名:");

scanf("%s",name);getchar();

int index = LocateList(lists, name);

int con = SaveList(lists.elem[index-1].L, name);

if(con == OK){

printf("保存成功！\n");

}

getchar();getchar();

break;

}

case 0:

break;

default:

break;

}

}

printf("欢迎下次再使用本系统！\n");

return 0;

}//end of main()

void menu2(LinkList &L,int op, char lname[]){

while(op){

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

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

printf(" List name : %s\n",lname);

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

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

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

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

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

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

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

printf(" 0. Exit this list\n");

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

printf(" 请选择你的操作[0~12]:");

scanf("%d",&op);

switch(op){

case 1:

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

else printf("单链表已存在！\n");

getchar();getchar();

break;

case 2:

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

else printf("线性表销毁失败！\n");

getchar();getchar();

break;

case 3:

if(OK == ClearList(L)) printf("线性表清空成功！\n");

else printf("线性表不存在！\n");

getchar();getchar();

break;

case 4:{

int c = ListEmpty(L);

if(c == INFEASIBLE) printf("线性表不存在！\n");

else if(c == TRUE) printf("线性表为空！\n");

else printf("线性表不为空！\n");

getchar();getchar();

break;

}

case 5:

{

int l = ListLength(L);

if(l == INFEASIBLE) printf("线性表不存在！\n");

else printf("线性表长度为%d",l);

getchar();getchar();

break;}

case 6:

{

int tof;

printf("请问您要查询哪个元素：");

scanf("%d", &tof);

ElemType e;

status con = GetElem(L, tof, e);

if(con == INFEASIBLE) printf("线性表不存在！\n");

else if(con == OK) printf("线性表L的第%d个元素是%d\n", tof, e);

else printf("输入不合法！\n");

getchar();getchar();

break;}

case 7:

{

ElemType e;

printf("请问您要查询元素的值是：");

scanf("%d", &e);

status con = LocateElem(L, e);

if(con == INFEASIBLE) printf("线性表不存在！\n");

else if(con == ERROR) printf("该元素不存在！\n");

else printf("该元素在线性表L中序号是%d\n",con);

getchar();getchar();

break;}

case 8:

{

ElemType e, pre;

printf("请问当前元素的值是：");

scanf("%d", &e);

status con = PriorElem(L, e, pre);

if(INFEASIBLE == con) printf("线性表不存在！\n");

else if(con == ERROR) printf("该元素没有前驱！\n");

else printf("线性表L中该元素的前驱是%d\n", pre);

getchar();getchar();

break;}

case 9:

{

ElemType e, next;

printf("请问当前元素的值是：");

scanf("%d", &e);

status con = NextElem(L, e, next);

if(INFEASIBLE == con) printf("线性表不存在！\n");

else if(con == ERROR) printf("该元素没有后继！\n");

else printf("线性表L中该元素的后继是%d\n", next);

getchar();getchar();

break;}

case 10:

{

ElemType e;

int pos = 0;

printf("请问要插入元素的值是(仅支持数字)：");

scanf("%d", &e);

printf("请问要插入元素的位置是：");

scanf("%d",&pos);

status con = ListInsert(L, pos, e);

if(con == INFEASIBLE) printf("线性表不存在！\n");

else if(con == ERROR) printf("插入位置不正确\n");

else printf("插入成功\n");

getchar();getchar();

break;}

case 11:

{

ElemType e;

int pos = 0;

printf("请问要删除元素的位置是：");

scanf("%d",&pos);

status con = ListDelete(L, pos, e);

if(con == INFEASIBLE) printf("线性表不存在！\n");

else if(con == ERROR) printf("输入位置不正确\n");

else printf("删除成功, 删除的值为%d\n", e);

getchar();getchar();

break;}

case 12:

{

status con =ListTraverse(L);

if(con == INFEASIBLE){

printf("线性表不存在！\n");

}

else if(con == 0)

printf("线性表是空表！\n");

getchar();getchar();

break;}

case 0:

break;

}//end of switch

}//end of while

}

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 != NULL)return INFEASIBLE;

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

if( n == NULL)return INFEASIBLE;

L= n;

n->next =NULL;

return OK;

}

status DestroyList(LinkList &L)

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

{

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

free(L);

L = NULL;

while(now != NULL){

LinkList last = now;

now = now->next;

free(last);

}

return OK;

}

status ClearList(LinkList &L)

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

{

if(L == NULL)return INFEASIBLE;

LinkList now = L->next;

L->next = NULL;

while(now!=NULL){

LinkList last = now;

now = now->next;

free(last);

}

return OK;

}

status ListEmpty(LinkList L)

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

{

if(L == NULL)return INFEASIBLE;

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

return FALSE;

}

int ListLength(LinkList L)

// 如果线性表L存在，返回线性表L的长度，否则返回INFEASIBLE。

{

if(L == NULL) return INFEASIBLE;

int length = 0;

LinkList now = L->next;

while(now != NULL){

now = now->next;

length++;

}

return length;

}

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(i<1)return ERROR;

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

int n = 0;

while(now != NULL){

n++;

if(n == i){

e = now ->data;

return OK;

}

now =now->next;

}

return ERROR;

}

status LocateElem(LinkList L,ElemType e)

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

{

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

int i = 0;

while(now != NULL){

i++;

if(now->data == e){

return i;

}

now =now->next;

}

return ERROR;

}

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

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

{

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

LinkList last;

int n = 0;

while(now != NULL){

n++;

if(now->data == e){

if(n == 1){

return ERROR;

}

pre = last->data;

return OK;

}

last = now;

now =now->next;

}

return ERROR;

}

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

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

{

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

int n = 0;

while(now != NULL){

n++;

if(now->data == e){

if(now->next == NULL){

return ERROR;

}

next = now->next->data;

return OK;

}

now =now->next;

}

return ERROR;

}

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

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

{

// 请在这里补充代码，完成本关任务

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

if(i<1)return ERROR;

if(L == NULL) return INFEASIBLE;

LinkList now = L;

int x;

for(x= 1 ; (x<i && now!=NULL); now=now->next,x++);

if(i==1)now = L;

if( x == i && now == NULL){

return ERROR;

}

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

if(New == NULL) return OVERFLOW;

New->data = e;

New->next = now->next;

now->next = New;

return OK;

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

}

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

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

{

int length = ListLength(L);

if( i<=0 || i>length)return ERROR;

if(L == NULL) return INFEASIBLE;

LinkList pre = L;

int x;

for(x= 1 ; (x<i && pre!=NULL); pre=pre->next,x++);

if(i==1)pre = L;

if( x == i && pre == NULL){

return ERROR;

}

LinkList now = pre ->next;

pre->next = now ->next;

e = now->data;

free(now);

return OK;

}

status ListTraverse(LinkList L)

// 如果线性表L存在，依次显示线性表中的元素，返回OK；如果线性表L不存在，返回INFEASIBLE。

{

if(L == NULL) return INFEASIBLE;

LinkList now = L->next;

int i = 0;

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

printf("%4s %16s\n","元素序号", "元素值");

while(now != NULL){

printf("%4d %16d\n",1+i, now->data);

i++;

now = now->next;

}

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

return i;

}

status SaveList(LinkList L,char FileName[])

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

{

if(L == NULL)return INFEASIBLE;

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

LinkList now = L->next;

while(now != NULL){

fprintf(fp, "%d\n",now->data);

now = now->next;

}

fclose(fp);

return OK;

}

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 != NULL)return INFEASIBLE;

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

L->next =NULL;

LinkList now = L;

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

ElemType e;

while(fscanf(fp, "%d", &e) != EOF){

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

New->data = e;

New->next =NULL;

now->next = New;

now=New;

}

fclose(fp);

return OK;

}

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

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

{

int i = Lists.length++;

int j=0;

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

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

j++;

}

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

Lists.elem[i].L = NULL;

Lists.elem[i].L = 0;

return 0;

}

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

// Lists中删除一个名称为ListName的线性表

{

int ok = 0;

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

int tag = 1;

int c=0;

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

if(Lists.elem[i].name[c] != ListName[c]){

tag = 0;

break;

}

c++;

}

if(tag){

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

Lists.length--;

if( i == Lists.length){

break;

}

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

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

}

ok =1;

break;

}

}

if(ok){

return OK;

}

return ERROR;

}

int LocateList(LISTS Lists,char ListName[])

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

{

int ok=0;

int i=0;

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

int tag = 1;

int c=0;

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

if(Lists.elem[i].name[c] != ListName[c]){

tag = 0;

break;

}

c++;

}

if(tag){

ok=1;

break;

}

}

if(ok){

return i+1;

}

return 0;

}

status LISTSTrabverse(LISTS &Lists){//遍历线性表集合，并打印

int i;

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

for(i=0;i<Lists.length;i++) printf("%4d: %s \n", i+1, Lists.elem[i].name);

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

return Lists.length;

}

# 附录C 基于二叉链表二叉树实现的源程序

/\*---------def.h ---------\*/

#pragma once

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

typedef int status;

typedef int KeyType;

typedef struct {

KeyType key;

int ID;

char name[20];

} TElemType; //二叉树结点类型定义

typedef struct BiTNode{ //二叉链表结点的定义

TElemType data;

struct BiTNode \*lchild,\*rchild;

} BiTNode, \*BiTree;

typedef struct{ //二叉树的集合类型定义

struct { char name[30];

BiTree T;

} elem[10];

int nums;

int Forestsize;

}Forest;

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

status DestroyBiTree(BiTree &T);//销毁树

status ClearBiTree(BiTree &T);status subClearBiTree(BiTree &T);

status EmptyBiTree(BiTree &T);//判空

int BiTreeDepth(BiTree T);int subBiTreeDepth(BiTree T);//非递归

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

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

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

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

status DeleteNode(BiTree &T,KeyType e);

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

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

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

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

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

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

status AddTree(Forest &f,char TreeName[]);

status RemoveTree(Forest &f,char TreeName[]);

int LocateTree(Forest &f,char TreeName[]);

status Treerabverse(Forest &f);

//补充函数

void visitBiTree(BiTree T)//打印节点值

{

printf("%4d%4d%8s\n",T->data.key, T->data.ID, T->data.name);

return;

}

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

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

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

{

// 请在这里补充代码，完成本关任务

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

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

T =NULL;

return OK;

}

BiTree stack[100] = {NULL};

int top = 0;

KeyType buf[1000] = {0};

int i=0;

if(buf[definition[i].key] == 1) return ERROR;

else buf[definition[i].key] = 1;

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

root->lchild = root->rchild = NULL;

root->data = definition[i++];

stack[top++] = root;

BiTree now = root;

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

//向左建树

while(definition[i].key != 0 && now->lchild == NULL)

{

if(buf[definition[i].key] == 1) return ERROR;

else buf[definition[i].key] = 1;

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

now->lchild->lchild = now->lchild->rchild = NULL;

now->lchild->data = definition[i++];

stack[top++] = now;

now =now->lchild;

}

if(now->lchild == NULL) i++;

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

if(buf[definition[i].key] == 1) return ERROR;

else buf[definition[i].key] = 1;

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

now->rchild->lchild = now->rchild->rchild = NULL;

now->rchild->data = definition[i++];

now = now->rchild;

}

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

break;

}

else {

i++;

now = stack[--top];

}

}

T = root;

return OK;

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

}

status DestroyBiTree(BiTree &T)//销毁树

{

if(T == NULL) return ERROR;

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

free(T);

T = NULL;

return OK;

}

if(T->lchild)DestroyBiTree(T->lchild);

if(T->rchild)DestroyBiTree(T->rchild);

free(T);

T = NULL;

return OK;

}

status ClearBiTree(BiTree &T)//递归算法

//将二叉树设置成空，并删除所有结点，释放结点空间

{

// 请在这里补充代码，完成本关任务

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

if(T == NULL) return OK;

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

free(T);

T = NULL;

return OK;

}

if(T->lchild)ClearBiTree(T->lchild);

if(T->rchild)ClearBiTree(T->rchild);

free(T);

T = NULL;

return OK;

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

}

status subClearBiTree(BiTree &T)//非递归算法

//将二叉树设置成空，并删除所有结点，释放结点空间

{

// 请在这里补充代码，完成本关任务

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

if(T == NULL)return OK;

BiTree stack[100] = {NULL};

int top = 0;

stack[top++] = T;

while(top != 0){

BiTree now = stack[--top];

if(now -> lchild != NULL)stack[top++]=now ->lchild;

if(now -> rchild != NULL)stack[top++]=now ->rchild;

free(now);

}

T = NULL;

return OK;

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

}

status EmptyBiTree(BiTree &T){

if(T == NULL)return OK;

else return ERROR;

}

int BiTreeDepth(BiTree T){//递归

if(T==NULL) {

return 0;

}

int nLeft=BiTreeDepth(T->lchild);

int nRight=BiTreeDepth(T->rchild);

return nLeft>nRight?nLeft+1:nRight+1;

}

BiTNode\* LocateNode(BiTree T,KeyType e)

//查找结点

{

// 请在这里补充代码，完成本关任务

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

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

BiTree ret = NULL;

if(T->lchild){

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

if(ret)return ret;

}

if(T -> rchild){

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

if(ret)return ret;

}

return NULL;

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

}

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

//实现结点赋值。此题允许通过增加其它函数辅助实现本关任务

{

// 请在这里补充代码，完成本关任务

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

int buf[1000]= {0};

BiTree queue[100];

int top = 0, base = 0;

BiTree now, find=NULL;

queue[top++] = T;

while(top > base){

now = queue[base++];

if(now -> lchild)queue[top++] = now ->lchild;

if(now -> rchild)queue[top++] = now -> rchild;

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

find = now;

continue;

}

buf[now ->data.key] = 1;

}

if( buf[value.key] == 1)return ERROR;

if( !find)return ERROR;

find ->data = value;

return OK;

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

}

BiTNode\* GetSibling(BiTree T,KeyType e)

//实现获得兄弟结点

{

// 请在这里补充代码，完成本关任务

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

if(T->data.key == e)return NULL;

BiTree queue[100];

int top = 0, base = 0;

int tag =0;

BiTree now;

queue[top++] = T;

while(top > base){

now = queue[base++];

if(now -> lchild){

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

now = now ->rchild;

tag = 1;

break;

}

else

queue[top++] = now ->lchild;

}

if(now -> rchild){

if( now ->rchild->data.key == e){

now = now->lchild;

tag = 1;

break;

}

else

queue[top++] = now -> rchild;

}

}

if(tag == 1)

return now;

else return NULL;

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

}

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

//插入结点。此题允许通过增加其它函数辅助实现本关任务

{

// 请在这里补充代码，完成本关任务

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

if(LR == -1){

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

now ->lchild = NULL;

now ->rchild = T;

now -> data = c;

T = now;

return OK;

}

int buf[1000]= {0};

BiTree queue[100];

int top = 0, base = 0;

BiTree now, find=NULL;

queue[top++] = T;

while(top > base){

now = queue[base++];

if(now -> lchild)queue[top++] = now ->lchild;

if(now -> rchild)queue[top++] = now -> rchild;

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

find = now;

}

buf[now ->data.key] = 1;

}

if( buf[c.key] == 1)return ERROR;

if( !find)return ERROR;

if(LR){

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

t -> lchild = NULL;

t -> rchild = find -> rchild;

t -> data = c;

find -> rchild = t;

}

else {

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

t -> lchild = NULL;

t -> rchild = find -> lchild;

t -> data = c;

find -> lchild = t;

}

return OK;

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

}

status DeleteNode(BiTree &T,KeyType e)

//删除结点。此题允许通过增加其它函数辅助实现本关任务

{

// 请在这里补充代码，完成本关任务

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

BiTree find = NULL;

int tag =0;

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

if(T->lchild){

BiTree t = T->lchild;

while(t->rchild !=NULL)t=t->rchild;

t->rchild = T->rchild;

find =T;

T=T->lchild;

free(find);

}

else if(T->rchild){

find =T;

T=T->rchild;

free(find);

}

else{

free(T);

T= NULL;

}

return OK;

}

else{

BiTree queue[100];

int top = 0, base = 0;

BiTree now;

queue[top++] = T;

while(top > base){

now = queue[base++];

if(now -> lchild){

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

find = now;

break;

}

else

queue[top++] = now ->lchild;

}

if(now -> rchild){

if( now ->rchild->data.key == e){

find = now;

tag = 1;

break;

}

else

queue[top++] = now -> rchild;

}

}

if(!find)return ERROR;

BiTree todel = tag?find->rchild:find->lchild;

if(todel->lchild){

BiTree t = todel->lchild;

while(t->rchild != NULL)t = t->rchild;

t->rchild=todel->rchild;

if(tag) find ->rchild = todel->lchild;

else find ->lchild = todel -> lchild;

free(todel);

return OK;

}

else{

if(tag) find ->rchild = todel->rchild;

else find ->lchild = todel -> rchild;

free(todel);

return OK;

}

}

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

}

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

//先序遍历二叉树T , 非递归

{

// 请在这里补充代码，完成本关任务

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

if(T == NULL)return OK;

BiTree stack[100];

int top =0;

stack[top++] =T;

BiTree now;

while(top!= 0){

now = stack[--top];

visit(now);

if(now->rchild != NULL){

stack[top++] = now->rchild;

}

if( now ->lchild != NULL)stack[top++] = now ->lchild;

}

return OK;

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

}

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

//先序遍历二叉树T 递归

{

// 请在这里补充代码，完成本关任务

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

if(T == NULL)return OK;

(\*visit)(T);

PreOrderTraverse(T->lchild, visit);

PreOrderTraverse(T->rchild, visit);

return OK;

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

}

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

//中序遍历二叉树T

{

// 请在这里补充代码，完成本关任务

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

if(T == NULL)return OK;

InOrderTraverse(T->lchild, visit);

(\*visit)(T);

InOrderTraverse(T->rchild, visit);

return OK;

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

}

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

//后序遍历二叉树T

{

// 请在这里补充代码，完成本关任务

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

if(T == NULL)return OK;

PostOrderTraverse(T->lchild, visit);

PostOrderTraverse(T->rchild, visit);

(\*visit)(T);

return OK;

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

}

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

//按层遍历二叉树T

{

// 请在这里补充代码，完成本关任务

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

BiTree queue[100];

int top =0, base=0;

queue[top++] = T;

BiTree now;

while(top>base){

now =queue[base++];

(\*visit)(now);

if(now->lchild) queue[top++] = now->lchild;

if(now->rchild) queue[top++] = now->rchild;

}

return OK;

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

}

status SaveBiTree(BiTree T, char FileName[])

//将二叉树的结点数据写入到文件FileName中

{

// 请在这里补充代码，完成本关任务

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

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

if(T == NULL){

fprintf(fp, "\n");

}

else{

BiTree stack[100];

int top=0;

stack[top++] = T;

BiTree now ;

while(top != 0){//先序

now = stack[--top];

if(now == NULL){

fprintf(fp, "0 0 null\n");

continue;

}

while(now -> lchild != NULL){

fprintf(fp, "%d %d %s\n", now->data.key, now->data.ID, now->data.name);

if( now -> rchild ){

stack[top++] = now -> rchild;

}

else {

stack[top++] = NULL;

}

now = now -> lchild;

}

fprintf(fp, "%d %d %s\n", now->data.key, now->data.ID, now->data.name);

fprintf(fp, "0 0 null\n");

if(now->rchild){

stack[top++] = now -> rchild;

}

else {

fprintf(fp, "0 0 null\n");

}

}

}

fclose(fp);

return OK;

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

}

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

//读入文件FileName的结点数据，创建二叉树

{

// 请在这里补充代码，完成本关任务

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

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

TElemType definition[100];

int i = 0;

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

fscanf(fp, "%d", &definition[i].ID);

fscanf(fp, "%s",definition[i].name);

i++;

fgetc(fp);

while(fscanf(fp, "%d", &definition[i].key) != EOF ){

fscanf(fp, "%d", &definition[i].ID);

fscanf(fp, "%s",definition[i].name);

i++;

fgetc(fp);

}

definition[i].key = -1;

definition[i].ID = 0;

sprintf(definition[i].name,"null");

free(T);

T = NULL;

CreateBiTree(T, definition);

fclose(fp);

return OK;

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

}

status AddTree(Forest &f,char TreeName[])

// 只需要在Forest中增加一个名称为TreeName的空二叉树

{

int i = f.nums++;

int j=0;

while(TreeName[j]!='\0'){

f.elem[i].name[j]=TreeName[j];

j++;

}

f.elem[i].name[j]=TreeName[j];

f.elem[i].T ==NULL;

return 0;

}

status RemoveTree(Forest &f,char TreeName[])

// f中删除一个名称为TreeName的二叉树

{

int index = LocateTree(f, TreeName);

if(index == 0){

printf("无此二叉树！\n");

return ERROR;

}

DestroyBiTree(f.elem[index-1].T);

for(int i = index-1;i<f.nums-1;i++){

strcpy(f.elem[i].name , f.elem[i+1].name);

f.elem[i].T = f.elem[i+1].T;

}

f.nums--;

return OK;

}

int LocateTree(Forest &f,char TreeName[])

// 在f中查找一个名称为TreeName的二叉树，成功返回逻辑序号，否则返回0

{

int ok=0;

int i=0;

for(; i<f.nums; i++){

int tag = 1;

int c=0;

while(TreeName[c] != '\0'){

if(f.elem[i].name[c] != TreeName[c]){

tag = 0;

break;

}

c++;

}

if(tag){

ok=1;

break;

}

}

if(ok){

return i+1;

}

return 0;

}

status Treerabverse(Forest &f)

{//遍历二叉树集合，并打印

int i;

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

for(i=0;i<f.nums;i++) printf("%4d: %s \n", i+1, f.elem[i].name);

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

return f.nums;

}

/\*---------main.cpp---------\*/

#include "def.h"

//1 a 2 b 0 null 0 null 3 c 4 d 0 null 0 null 5 e 0 null 0 null -1 null

void menu2(BiTree &T,int op, char tname[]);

/\*TElemType definition[100]= {

{1,1,"a"},

{2,2,"b"},

{0,1, "null"},

{0,1,"null"},

{3,3,"c"},

{4,4,"d"}, {0,1,"null"},{0,1,"null"},{5,5,"e"},{0,1,"null"},{0,1,"null"},{-1,1,"null"}

};\*/

int main(){

Forest f;

memset(&f, 0, sizeof(Forest));//设置为空

f.nums = 0;

int op=1;

while(op){

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

printf(" 基于二叉链表的二叉树操作演示系统 \n");

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

printf(" 1.列出当前的二叉树名 \n");

printf(" 2.切换到指定二叉树 \n");

printf(" 3.增加一个二叉树 \n");

printf(" 4.删除一个二叉树 \n");

printf(" 5.导入一个二叉树 \n");

printf(" 6.保存一个二叉树 \n");

printf(" 0.退出\n");

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

printf(" 请选择你的操作[0~6]:");

scanf("%d",&op);

switch (op)

{

case 1:

if(0 == Treerabverse(f))

printf("当前无树！");

printf("按任意键继续...\n");

getchar();getchar();

break;

case 2:

{

//切换到指定tree

int index;

printf("请输入需要切换到的树序号:");

scanf("%d",&index);

if(index <= f.nums){

menu2( f.elem[index-1].T, 1, f.elem[index-1].name);

}

else printf("输入不正确！\n");

getchar();getchar();

break;

}

case 3:{

//插入一个tree

char name[30];

getchar();

printf("请输入增加树的名称:");

scanf("%s",name);

int con = AddTree(f, name);

printf("增加成功！\n");

getchar();getchar();

break;

}

case 4:{

//删除一个tree

char name[30];

getchar();

printf("请输入删除树名:");

scanf("%s",name);

int con = RemoveTree(f, name);

printf("删除成功！\n");

getchar();getchar();

break;

}

case 5:{

//导入一个tree

char name[30];

getchar();

printf("请输入导入文件的文件名:");

scanf("%s",name);getchar();

AddTree(f, name);

f.elem[f.nums-1].T = NULL;

int con = LoadBiTree(f.elem[f.nums-1].T, name);

if(con == OK){

printf("导入成功！\n");

}

else{

printf("导入失败！\n");

}

getchar();getchar();

break;

}

case 6:{

//保存一个tree

char name[30];

getchar();

printf("请输入保存的树名:");

scanf("%s",name);getchar();

int index = LocateTree(f, name);

int con = SaveBiTree(f.elem[index-1].T, name);

if(con == OK){

printf("保存成功！\n");

}

getchar();getchar();

break;

}

case 0:

break;

default:

break;

}

}

printf("欢迎下次再使用本系统！\n");

return 0;

}

void menu2(BiTree &T,int op, char tname[]){

while(op){

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

printf(" 基于二叉链表的二叉树演示系统 \n");

printf(" 二叉树名: %s \n",tname);

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

printf(" 1. 创建二叉树 9. 插入节点\n");

printf(" 2. 销毁二叉树 10. 删除节点\n");

printf(" 3. 清空二叉树 11. 先序遍历 \n");

printf(" 4. 判空二叉树 12. 中序遍历\n");

printf(" 5. 求二叉树深 13. 后序遍历\n");

printf(" 6. 查找节点 14. 层序遍历\n");

printf(" 7. 节点赋值 15. 保存二叉树\n");

printf(" 8. 获取兄弟节点 16. 导入二叉树\n");

printf(" 0. 退出\n");

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

printf(" 请选择你的操作[0~16]:");

scanf("%d",&op);

switch(op){

case 1:

{

printf("请选择输入建树序列：");

TElemType definition[100]={0};

int i=0;

do{

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

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

if(CreateBiTree(T, definition) == OK){

printf("建树成功！\n");

}

else

printf("建树失败！\n");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 2:

{

if( DestroyBiTree(T) == OK){

printf("二叉树销毁成功!\n");

}

else printf("二叉树已销毁！\n");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 3:

{

if( ClearBiTree(T) == OK){

printf("二叉树清空成功!\n");

}

else printf("二叉树已经为空!\n");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 4:

{

if( EmptyBiTree(T) == OK){

printf("二叉树为空！\n");

}

else printf("二叉树不为空!\n");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 5:

{

int depth = BiTreeDepth(T);

printf("二叉树深度为%d!\n", depth);

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 6:

{

KeyType e;

printf("请给出节点关键字：");

scanf("%d",&e);

BiTree now = LocateNode(T, e);

if( now != NULL){

printf("关键字为%d的节点数据域为:\n",e);

printf("id:%d, name:%s\n",now->data.ID,now->data.name);

}else{

printf("没有关键字为%d的节点!\n",e);

}

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 7:

{

KeyType e;

printf("请给出节点关键字：");

scanf("%d",&e);

getchar();

BiTree now = LocateNode(T, e);

if( now != NULL){

printf("关键字为%d的节点数据域为:\n",e);

printf("id:%d, name:%s\n",now->data.ID,now->data.name);

TElemType value={0};

printf("请给出新的key:");

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

getchar();

printf("请给出新的id:");

scanf("%d",&value.ID);

getchar();

printf("请给出新的name:");

scanf("%s",value.name);

if(Assign(T, e, value) == OK){

printf("赋值成功！\n");

}

else printf("赋值失败！\n");

}else{

printf("赋值失败，没有关键字为%d的节点!\n",e);

}

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 8:

{

KeyType e;

printf("请给出节点关键字：");

scanf("%d",&e);

BiTree now = GetSibling(T, e);

if( now != NULL){

printf("关键字为%d的节点的兄弟节点关键字为:%d,数据域为\n",e,now->data.key);

printf("id:%d, name:%s\n",now->data.ID,now->data.name);

}else{

printf("关键字为%d的节点没有兄弟节点!\n",e);

}

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 9:

{

TElemType value={0};

printf("请给出待插入节点的key:");

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

getchar();

printf("请给出待插入节点的id:");

scanf("%d",&value.ID);

getchar();

printf("请给出待插入节点的name:");

scanf("%s",value.name);

int LR;

printf("请给出插入节点为左子树或右子树：(0左 1右 -1插入为根节点)\n");

scanf("%d",&LR);getchar();

KeyType e;

if(LR != -1){

printf("请给出插入节点父节点的关键字：");

scanf("%d",&e);

getchar();

}

if(InsertNode(T,e,LR,value) == OK){

printf("插入成功！\n");

}

else printf("插入失败！\n");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 10:

{

KeyType e;

printf("请给出需删除节点关键字：");

scanf("%d",&e);

getchar();

if( DeleteNode(T, e) == OK ){

printf("删除成功！\n");

}else{

printf("删除失败，没有关键字为%d的节点!\n",e);

}

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 11:

{

if(EmptyBiTree(T) == OK){

printf("二叉树为空！\n");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

printf("二叉树先序遍历如下：\n");

printf("%4s%4s%8s\n","key","ID","name");

PreOrderTraverse(T, visitBiTree);

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 12:

{

if(EmptyBiTree(T) == OK){

printf("二叉树为空！\n");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

printf("二叉树中序遍历如下：\n");

printf("%4s%4s%8s\n","key","ID","name");

InOrderTraverse(T, visitBiTree);

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 13:

{

if(EmptyBiTree(T) == OK){

printf("二叉树为空！\n");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

printf("二叉树后序遍历如下：\n");

printf("%4s%4s%8s\n","key","ID","name");

PostOrderTraverse(T, visitBiTree);

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 14:

{

if(EmptyBiTree(T) == OK){

printf("二叉树为空！\n");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

printf("二叉树层序遍历如下：\n");

printf("%4s%4s%8s\n","key","ID","name");

LevelOrderTraverse(T, visitBiTree);

printf("请按任意键继续...");

getchar();getchar();

getchar();getchar();

continue;

}

case 15:

{

printf("请输入写入文件名：");

char filename[100];

scanf("%s",filename);

if(OK == SaveBiTree(T, filename) ){

printf("保存成功！\n");

}

else printf("保存失败！\n");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 16:

{

printf("请输入导入文件名：");

char filename[100];

scanf("%s",filename);

if(OK == LoadBiTree(T, filename) ){

printf("导入成功！\n");

}

else printf("导入失败！\n");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 0:

break;

}//end of switch

}//end of while

}

# 附录D 基于邻接表图实现的源程序

/\*---------def.h ---------\*/

#pragma once

#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

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{ //图的集合类型定义

struct { char name[30];

ALGraph G;

} elem[10];

int nums;

int Graphs\_size;

}Graphs;

//这样可以通过说明语句：ALGraph G; 定义结构变量G，使得G可以表示一个图。

//函数声明

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

status DestroyGraph(ALGraph &G);

int LocateVex(ALGraph G,KeyType u);

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

int FirstAdjVex(ALGraph G,KeyType u);

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

status InsertVex(ALGraph &G,VertexType v);

status DeleteVex(ALGraph &G,KeyType v);

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

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

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

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

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

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

//

status Grabverse(Graphs &g);

status AddG(Graphs &f,char GName[]);

status RemoveG(Graphs &f,char GName[]);

int LocateG(Graphs &f,char GName[]);

void Visit(VertexType v){

printf("key:%8d others:%8s\n",v.key, v.others);

}

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

/\*根据V和VR构造图T并返回OK，如果V和VR不正确，返回ERROR

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

{

// 请在这里补充代码，完成本关任务

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

if(V[0].key == -1)return ERROR;

KeyType buf[100] = {0};

G.vexnum = 0;

G.arcnum = 0;

G.kind = UDG;

int i =0;

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

if(i>=20)return ERROR;

if( buf[V[i].key] == 1)return ERROR;

else buf[V[i].key] = 1;

G.vexnum++;

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

G.vertices[i].firstarc = NULL;

i++;

}

int vri = 0;

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

{

if( buf[VR[vri][0]] == 0 || buf[VR[vri][1]] == 0)return ERROR;

int ii=0;

while(ii<G.vexnum&& G.vertices[ii].data.key != VR[vri][1]) ii++;

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

ti->adjvex = ii;

ti->nextarc = NULL;

int jj = 0;

while(jj<G.vexnum&& G.vertices[jj].data.key != VR[vri][0]) jj++;

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

tj->adjvex = jj;

tj->nextarc = NULL;

tj->nextarc =G.vertices[ii].firstarc;

G.vertices[ii].firstarc = tj;

ti->nextarc =G.vertices[jj].firstarc;

G.vertices[jj].firstarc = ti;

G.arcnum++;

vri++;

}

return OK;

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

}

status DestroyGraph(ALGraph &G)

/\*销毁无向图G,删除G的全部顶点和边\*/

{

// 请在这里补充代码，完成本关任务

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

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

if(G.vertices[i].firstarc == NULL)continue;

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

G.vertices[i].firstarc = NULL;

ArcNode\* c;

while(t != NULL){

c=t;

t=t->nextarc;

free(c);

}

}

G.arcnum=0;

G.vexnum=0;

return OK;

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

}

int LocateVex(ALGraph G,KeyType u)

//根据u在图G中查找顶点，查找成功返回位序，否则返回-1；

{

// 请在这里补充代码，完成本关任务

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

int i= 0;

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

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

}

return (i>= G.vexnum)?-1:i;

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

}

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

//根据u在图G中查找顶点，查找成功将该顶点值修改成value，返回OK；

//如果查找失败或关键字不唯一，返回ERROR

{

// 请在这里补充代码，完成本关任务

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

int index = LocateVex(G,u);

if(index == -1)return ERROR;

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

{

if(i==index)continue;

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

}

G.vertices[index].data = value;

return OK;

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

}

int FirstAdjVex(ALGraph G,KeyType u)

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

{

// 请在这里补充代码，完成本关任务

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

int index = LocateVex(G,u);

if(index == -1)return ERROR;

if(G.vertices[index].firstarc == NULL)return -1;

else return G.vertices[index].firstarc->adjvex;

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

}

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

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

{

// 请在这里补充代码，完成本关任务

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

int inv = LocateVex(G,v);

int inw = LocateVex(G,w);

if(inv == -1 || inw == -1)return -1;

ArcNode\* t;

if(G.vertices[inv].firstarc == NULL)return -1;

t = G.vertices[inv].firstarc;

while(t->adjvex != inw && t!= NULL)t=t->nextarc;

if(t==NULL || t->nextarc == NULL)return -1;

return t->nextarc->adjvex;

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

}

status InsertVex(ALGraph &G,VertexType v)

//在图G中插入顶点v，成功返回OK,否则返回ERROR

{

// 请在这里补充代码，完成本关任务

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

if(LocateVex(G,v.key) != -1)return ERROR;

if(G.vexnum >= MAX\_VERTEX\_NUM)return ERROR;

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

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

G.vexnum++;

return OK;

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

}

status DeleteVex(ALGraph &G,KeyType v)

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

{

// 请在这里补充代码，完成本关任务

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

if(G.vexnum == 1)return ERROR;

int index = LocateVex(G,v);

if( index == -1)return ERROR;

if(G.vertices[index].firstarc != NULL){//删除弧

ArcNode\* prv = G.vertices[index].firstarc;

while(prv != NULL){

int todel = prv->adjvex;

ArcNode\* todelpr = G.vertices[todel].firstarc;

ArcNode\* last;

if(todelpr->adjvex == index){

G.vertices[todel].firstarc = todelpr->nextarc;

free(todelpr);

}

else{

last = todelpr;

todelpr = todelpr->nextarc;

while(todelpr->adjvex != index){

last = todelpr;

todelpr = todelpr->nextarc;

}

last->nextarc = todelpr->nextarc;

free(todelpr);

}

G.arcnum--;

last = prv;

prv = prv->nextarc;

free(last);

}

G.vertices[index].firstarc = NULL;

}

//顶点删除

for(int i = index; i<G.vexnum-1;i++){

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

}

G.vexnum--;

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

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

while(pr!= NULL){

if(pr->adjvex > index)pr->adjvex-=1;

pr=pr->nextarc;

}

}

return OK;

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

}

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

//在图G中增加弧<v,w>，成功返回OK,否则返回ERROR

{

// 请在这里补充代码，完成本关任务

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

int inv = LocateVex(G, v);

int inw = LocateVex(G, w);

if( inv == -1)return ERROR;

if( inw == -1)return ERROR;

ArcNode\* pr = G.vertices[inv].firstarc;

while(pr!= NULL){

if(pr->adjvex == inw)return ERROR;

pr = pr->nextarc;

}

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

newone->adjvex = inw;

newone->nextarc = G.vertices[inv].firstarc;

G.vertices[inv].firstarc = newone;

ArcNode\* newone1 = (ArcNode\*)malloc(sizeof(ArcNode));

newone1->adjvex = inv;

newone1->nextarc = G.vertices[inw].firstarc;

G.vertices[inw].firstarc = newone1;

G.arcnum++;

return OK;

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

}

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

//在图G中删除弧<v,w>，成功返回OK,否则返回ERROR

{

// 请在这里补充代码，完成本关任务

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

int inv = LocateVex(G, v);

int inw = LocateVex(G, w);

if( inv == -1)return ERROR;

if( inw == -1)return ERROR;

int tag = 0;

ArcNode\* pr = G.vertices[inv].firstarc;

while(pr!= NULL){

if(pr->adjvex == inw)tag = 1;

pr = pr->nextarc;

}

if(!tag)return ERROR;

ArcNode\* last;

pr = G.vertices[inv].firstarc;

if(G.vertices[inv].firstarc->adjvex == inw){

G.vertices[inv].firstarc = pr->nextarc;

free(pr);

}

else{

last = pr;

pr=pr->nextarc;

while(pr->adjvex != inw){

last = pr;

pr=pr->nextarc;

}

last->nextarc = pr->nextarc;

free(pr);

}

pr = G.vertices[inw].firstarc;

if(G.vertices[inw].firstarc->adjvex == inv){

G.vertices[inw].firstarc = pr->nextarc;

free(pr);

}

else{

last = pr;

pr=pr->nextarc;

while(pr->adjvex != inv){

last = pr;

pr=pr->nextarc;

}

last->nextarc = pr->nextarc;

free(pr);

}

G.arcnum--;

return OK;

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

}

status DFS(ALGraph &G, void (\*visit)(VertexType), int buf[], int index){

buf[index] = 1;

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

ArcNode\* pr = G.vertices[index].firstarc;

while( pr!= NULL){

if(buf[pr->adjvex] == 0)DFS(G, visit, buf, pr->adjvex);

pr=pr->nextarc;

}

return OK;

}

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

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

{

// 请在这里补充代码，完成本关任务

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

int buf[MAX\_VERTEX\_NUM+1] = {0};

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

if(buf[i] == 0)DFS(G, visit, buf, i);

}

return OK;

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

}

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

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

{

// 请在这里补充代码，完成本关任务

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

int queue[MAX\_VERTEX\_NUM +2] = {0};

int top=0,rear=0;

int buf[MAX\_VERTEX\_NUM+2] = {0};

int num = 0;

while(num != G.vexnum){

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

if(buf[i]==0){

queue[top++] = i;

buf[i]=1;

num++;

break;

}

}

while(top != rear){

int index = queue[rear++];

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

ArcNode\* pr = G.vertices[index].firstarc;

while(pr!= NULL){

if(buf[pr->adjvex] == 0){

queue[top++] = pr->adjvex;

buf[pr->adjvex] = 1;

num++;

}

pr=pr->nextarc;

}

}

}

return OK;

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

}

status SaveGraph(ALGraph G, char FileName[])

//将图的数据写入到文件FileName中

{

// 请在这里补充代码，完成本关任务

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

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

fprintf(fp, "%d %d %d\n", G.kind, G.vexnum, G.arcnum);

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

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

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

while(pr != NULL){

fprintf(fp, "%d ", pr->adjvex);

pr = pr->nextarc;

}

fprintf(fp, "-1\n");

}

fclose(fp);

return OK;

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

}

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

//读入文件FileName的图数据，创建图的邻接表

{

// 请在这里补充代码，完成本关任务

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

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

fscanf(fp, "%d %d %d", &G.kind, &G.vexnum, &G.arcnum);

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

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

G.vertices[i].firstarc = NULL;

ArcNode\* now =NULL;

int readc=-1;

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

while(readc != -1){

if(G.vertices[i].firstarc == NULL){

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

newone->nextarc = NULL;

newone->adjvex = readc;

G.vertices[i].firstarc = newone;

now = newone;

}

else{

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

newone->nextarc = NULL;

newone->adjvex = readc;

now->nextarc = newone;

now = newone;

}

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

}

}

fclose(fp);

return OK;

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

}

status Grabverse(Graphs &g)

{//遍历图集合，并打印

int i;

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

for(i=0;i<g.nums;i++) printf("%4d: %s \n", i+1, g.elem[i].name);

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

return g.nums;

}

status AddG(Graphs &f,char GName[])

// 只需要在Graphs中增加一个名称为GName的空二叉树

{

int i = f.nums++;

int j=0;

while(GName[j]!='\0'){

f.elem[i].name[j]=GName[j];

j++;

}

f.elem[i].name[j]=GName[j];

f.elem[i].G.vexnum=0;

return 0;

}

status RemoveG(Graphs &f,char GName[])

// f中删除一个名称为GName的图

{

int index = LocateG(f, GName);

if(index == 0){

printf("无此图！\n");

return ERROR;

}

DestroyGraph(f.elem[index-1].G);

for(int i = index-1;i<f.nums-1;i++){

strcpy(f.elem[i].name , f.elem[i+1].name);

f.elem[i].G= f.elem[i+1].G;

}

f.nums--;

return OK;

}

int LocateG(Graphs &f,char GName[])

// 在f中查找一个名称为GName的图，成功返回逻辑序号，否则返回0

{

int ok=0;

int i=0;

for(; i<f.nums; i++){

int tag = 1;

int c=0;

while(GName[c] != '\0'){

if(f.elem[i].name[c] != GName[c]){

tag = 0;

break;

}

c++;

}

if(tag){

ok=1;

break;

}

}

if(ok){

return i+1;

}

return 0;

}

/\*---------main.cpp---------\*/

#include "def.h"

/\*

5 线性表 2 3

8 集合 2

7 二叉树 1 3 0

6 无向图 2 0

\*/

//VertexType V[] = {

// {5,"线性表"},{8, "jihe"},{7,"ercs"},{6, "wuxiangtu"},{-1,"sada"}

//};

//KeyType VR[][2]={{5,6},{5,7},{6,7},{7,8},{-1,-1}};

void menu2(ALGraph &G, int op, char tname[]);

int main(){

Graphs graphs;

memset(&graphs, 0, sizeof(Graphs));//设置为空

graphs.nums = 0;

int op=1;

while(op){

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

printf(" 基于邻接表的图操作演示系统 \n");

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

printf(" 1.列出当前的图名 \n");

printf(" 2.切换到指定图 \n");

printf(" 3.增加一个图 \n");

printf(" 4.删除一个图 \n");

printf(" 5.导入一个图 \n");

printf(" 6.保存一个图 \n");

printf(" 0.退出\n");

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

printf(" 请选择你的操作[0~6]:");

scanf("%d",&op);

switch (op)

{

case 1:

if(0 == Grabverse(graphs))

printf("当前无图！");

getchar();getchar();

break;

case 2:

{

//切换到指定图

int index;

printf("请输入需要切换到的图序号:");

scanf("%d",&index);

if(index <= graphs.nums){

menu2( graphs.elem[index-1].G, 1, graphs.elem[index-1].name);

}

else printf("输入不正确！\n");

getchar();getchar();

break;

}

case 3:{

//插入一个图

char name[30];

getchar();

printf("请输入增加图的名称:");

scanf("%s",name);

int con = AddG(graphs, name);

if(con == OK)printf("增加成功！\n");

getchar();getchar();

break;

}

case 4:{

char name[30];

getchar();

printf("请输入删除图名:");

scanf("%s",name);

int con = RemoveG(graphs, name);

if(con == OK)printf("删除成功！\n");

getchar();getchar();

break;

}

case 5:{

char name[30];

getchar();

printf("请输入导入文件的文件名:");

scanf("%s",name);getchar();

AddG(graphs, name);

int con = LoadGraph(graphs.elem[graphs.nums-1].G, name);

if(con == OK){

printf("导入成功！\n");

}

else{

printf("导入失败！\n");

}

getchar();getchar();

break;

}

case 6:{

char name[30];

getchar();

printf("请输入保存的图名:");

scanf("%s",name);getchar();

int index = LocateG(graphs, name);

int con = SaveGraph(graphs.elem[index-1].G, name);

if(con == OK){

printf("保存成功！\n");

}

getchar();getchar();

break;

}

case 0:

break;

default:

break;

}

}

printf("欢迎下次再使用本系统！\n");

return 0;

}

void menu2(ALGraph &G, int op, char tname[]){

while(op){

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

printf(" 基于邻接表的图演示系统 \n");

printf(" 图名: %s \n",tname);

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

printf(" 1. 创建无向图 7. 插入顶点\n");

printf(" 2. 销毁图 8. 删除顶点\n");

printf(" 3. 查找顶点 9. 插入弧 \n");

printf(" 4. 顶点赋值 10. 删除弧\n");

printf(" 5. 获得第一邻接点 11. DFS\n");

printf(" 6. 获得下一邻接点 12. BFS\n");

printf(" 0. 退出\n");

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

printf(" 请选择你的操作[0~12]:");

scanf("%d",&op);

switch(op){

case 1:

{

VertexType V[20];KeyType VR[20][2];

int i = 0;

printf("请输入顶点序列:\n");

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

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

{

i++;

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

}

printf("请输入边序列:\n");

int j = 0;

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

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

{

j++;

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

}

int con = CreateCraph(G, V, VR);

if(con == OK)printf("创建成功！");

else printf("创建失败");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 2:

{

int con = DestroyGraph(G);

if(con == OK)printf("销毁成功！");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 3:

{

KeyType e;

printf("请输入关键字：");

scanf("%d", &e);

int con = LocateVex(G, e);

if(con == -1){

printf("查找失败！");

}

else{

printf("该顶点位序为%d,data为:\n", con);

Visit(G.vertices[con].data);

}

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 4:

{

KeyType u;VertexType value;

printf("请输入关键字：");

scanf("%d", &u);

int con = LocateVex(G, u);

if(con == -1){

printf("查找失败！");

}

else{

printf("请输入新值(key,others):");

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

int a = PutVex(G,u,value);

if(a == ERROR)printf("关键字不唯一！");

else printf("修改成功！");

}

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 5:

{

KeyType e;

printf("请输入关键字：");

scanf("%d", &e);

int con = FirstAdjVex(G, e);

if(con == ERROR){

printf("查找失败！");

}

else{

printf("该顶点第一邻接点位序为%d,data为:\n", con);

Visit(G.vertices[con].data);

}

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 6:

{

KeyType v,w;

printf("请输入一对关键字(1:顶点 2：相对顶点)：");

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

int con = NextAdjVex(G, v, w);

if(con == -1){

printf("查找失败！");

}

else{

printf("该顶点相对%d顶点下一邻接点位序为%d,data为:\n", w, con);

Visit(G.vertices[con].data);

}

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 7:

{

VertexType value;

printf("请输入新值(key,others):");

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

int con = InsertVex(G,value);

if(con == ERROR)printf("插入失败！");

else printf("插入成功！");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 8:

{

KeyType e;

printf("请输入关键字：");

scanf("%d", &e);

int con = DeleteVex(G, e);

if(con == ERROR)printf("删除失败！");

else printf("删除成功！");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 9:

{

KeyType v,w;

printf("请输入一对关键字(两个顶点)：");

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

int con = InsertArc(G, v, w);

if(con == ERROR)printf("插入失败！");

else printf("插入成功！");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 10:

{

KeyType v,w;

printf("请输入一对关键字(两个顶点)：");

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

int con = DeleteArc(G, v, w);

if(con == ERROR)printf("删除失败！");

else printf("删除成功！");

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 11:

{

DFSTraverse(G, &Visit);

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 12:

{

BFSTraverse(G, &Visit);

printf("请按任意键继续...");

getchar();getchar();

continue;

}

case 0:

break;

}//end of switch

}//end of while

}